VISOKA TEHNIČKA ŠKOLA
STRUKOVNIH STUDIJA NIŠ

PRAKTIKUM

KLIJENT SERVER SISTEMI

MILOŠ KOSANOVIĆ, MIRKO KOSANOVIĆ

Sadržaj:

Laboratorijska vežba broj 1: Uvod u JavaScript

Ova laboratorijska vežba podrazumeva da je student već upoznat sa osnovnim konceptima JavaScript jezika. Sadržaj vežbe obajšnjava neke osnovne koncepte jezika koji su neophodni da bi se radilo sa NodeJs tehnologijom.

Primer 1: Logovanje. Konzola je alat koju programeri koriste kako bi snimili i prikazali rezultate rada JavaScript programa. Moguće je logovati string, promenljivu, niz ili bilo koji drugi tip podatka.

```
var a = 3;
console.log("Hello");
console.log(a);
```

Tipovi podataka

Tipovi podataka u JavaScript jeziku se mogu podeliti na primitivne i složene. Pristupi primitivnom tipu, zapravo pristupa direktno vrednosti koju ta promenljiva sadrži. Pristup kompleksnom tipu zapravp pristupa referenci koja se odnosi na vrednost te promenljie.

Primitivni tipovi su Number, Boolean , String , null (odsustvo vrednosti), and undefined (vrednost neinicijalizovane promenljive) . Vrednost primitivnih tipova je immutable, što znači da se sama vrednost ne može promeniti. Na primer, ako je x=3.14, moguće je dodeliti novu vrednost promenljivoj x, ali je vrednost 3.14 nemoguće promeniti.

- Složeni (referntni) tipovi su Array , Function , i Object .

Ilustrujmo to primerom:

```
// primitives
var a = 5;
var b = a;
b = 6;
a; // will be 5      b; // will be 6
// complex
var a = [‘hello’, ‘world’];
var b = a;
b[0] = ‘bye’;        a[0]; // will be ’bye’          b[0]; // will be ‘bye’
```

U drugom primeru b će sadržati istu reference kao i a. Kada menjamo prvi čan niza b, istovremeno menjamo i član niza a.

Dobra praksa je da primitivne tipove uvek deklarišete kao brojve, stringove ili Boolean. Deklaracija ovih tipova podataka kao objekata usporava izvršavanje koda i može dovesti do čudnih grešaka u kodu (*side effects*).

Problemi sa tipovima podataka

Određivanje tačnog tipa podataka ume da bude problematiično u JavaScript jeziku. Sam jezik sadrži objekte omotače za svaki od primitivnih tipova, kao i većina OO jezika. Na primer String se može kreirati na sledeća dva načina.

```
var a = ‘woot’;
var b = new String(‘woot’);
a + b; // ‘woot woot’
```

Ukoliko se iskoriste operatori typeod i instanceof na ove dve promenljive dobijaju se internesantni rezultati.

```
typeof a; // ‘string’
typeof b; // ‘object’
a instanceof String; // false
b instanceof String; // true
```

Ukoliko se iskoriste operatori ekvivalencije == i === dobijaju se različiti rezultati.

```
a == b; // true
a === b; // false
```

Svojstva (*Properties*)
Svaki podatak u JavaScript jeziku se pamti kao instanca nekog tipa podatka što drugačije nazivamo i objekat. Svaki objekat ima određeni skup predefinsianih svojstava i metoda. Na primer, reč "*Hello*" će biti zapamćena kao instance string objekta. Svi string objekti imaju svojstvo **length** u kome se pamti dužina tog stringa.

```
Console.log('Hello'.length);
```

Različiti objekti imaju predefinisan (built in) različiti skup svojstava. String objekti, na primer, imaju definisana svojstva:

- toUpperCase() koji povećava sva slova
- startsWith(str) koji proverava da li tenutni string počinje stringom str i vraća logičku vrednost.

Pitanje 1: Koje još metode (svojstva) objekata string poznajete?-

Korišćenje bibliotke
Korišćenje metoda objekata po definiciji zahtevaju da taj objekat prvo kreirate pre nego što ga koristite. Za pozivanje metoda bez kreiranja instance objekta u JavaScriptu se koriste biblioteke ili paketi. Primer takvog paketa je Math koji sadrži različite matematičke funkcije. Primer poziva je:

```
Math.random();
```

Deklaracija promenljivih
Svaka promenljiva mora da se deklariše pre nego što je koristite u programu. Deklarisanje promenljivih se može izvršiti pomoću ključnih reči:

- **var** - definiše promenljivu čija vrednost može da se menja, a čija je oblast važenja vezana za **funkciju u kojoj je deklarisana.**
- **let** – definiše promenljivu čija vrednost može da se menja, a čija je oblast važenja vezana za **trenutni blok naredbi.**
- **const** – je skraćenica od constant. Program će kreirati promenljivu čije ime nije moguće menjati. Ukoliko probate da promenite vrednost ovoj promenljivoj, program će baciti grešku.

```
function scopeDiff(){
   for (let i=0; i<5; i++)
      console.log(i);
   console.log(i); //i is not visable
}
```

```
function scopeDiff(){
   for (var i=0; i<5; i++)
console.log(i);
   console.log(i);  //i is visable
}
```

Globalne and lokalne promenljive
Na liniji 3 ukloni var i rezultat će u oba slučaja biti 14.

```
var my_number = 7; //this has global scope
var timesTwo = function(number) {
    var my_number = number * 2;
    console.log("Inside the function my_number is: ");
    console.log(my_number);
};

timesTwo(7);

console.log("Outside the function my_number is: ")
console.log(my_number);
```

Interpolacija Stringa u ES6
Kako bi korišćenje operatora + za konkatenaciju stringova bilo izbegnuto, moguće je koristiti znak *backtick* umesto znakova jednostrukog i dvostrukog navoda. Promenljive se tada diretkno ubacuju u string obavijene sa ${}. Priemr:

```
let myPet = 'aligator';
console.log(`I own a pet ${myPet}.`);
```

Nizovi
Definicija niza:

```
[element0, element1, ..., elementN]
new Array(element0, element1[, ...[, elementN]])
new Array(arrayLength)
```

```
myArray = [10,true,"Test"];
var fruits = ['Apple', 'Banana'];
console.log(fruits.length);     // 2
```

Pristup elementima niza:

```
var first = fruits[0];          // Apple
var last = fruits[fruits.length - 1];     // Banana
```

Dodavanje na kraj niza:

```
var newLength = fruits.push('Orange');
// ["Apple", "Banana", "Orange"]
```

Prolazak kroz niz:

```
var languages = ["HTML", "CSS", "JavaScript", "Python", "Ruby"];
var i;
for (i = 0; i<languages.length; i++)
    console.log(languages[i]);
```

Abstrakcija Iteracije niza:
U javascriptu je prethodni primer sklon greškama, može da se desi da ponovo koristite promenljivu ili da pogrešno napišete length itd…. Zbog toga programeri često abstrakuju obilazak niza pozivom funkcije foreach na sledeći način:

```
[1, 2, 3].forEach(function (v) {
console.log(v);
});

fruits.forEach(function(item, index, array) {
  console.log(item, index);            // Apple 0 // Banana 1
});
```

Promena vrednosti svakog člana niza

```
[5, 10, 15].map(function (v) {
   return v * 2;
}); // will return [10, 20, 30]
```

Heterogeni nizovi

```
myArray = [10,true,"Test"];
```

Multidimenzionalni nizovi

```
var jagged = [ [1,true], [1,2,3,4], [true,1]];
```

FUNKCIJE (functions)

Kod koji deklariše funkciju ima sledeću sintaksu:

```
function name(param1, param2, param3){
   //code
}
```

Funkcije imaju jako bitnu ulogu u JavaScript jeziku. Mogu se zapamtiti u promenljivoj ili preneti kao parametar funkcije.

```
var a = function () {}
console.log(a); // passing the function as a parameter
```

Može da se dodeli i promenljivoj ili da poziva samou sebe.

```
var factorial = function fac(n) { return n < 2 ? 1 : n * fac(n - 1); };
console.log(factorial(3));
```

Funkcija će se izvršiti ili biti pozvana u sledećim slučajevima:

- Kada se dogodi neki događaj (pristisak na dugme);
- Kada se eksplicitno pozove iz koda
- Automatski (self ivoked)

Drugi način deklarisanja funkcije je korišćenjem objekta Function i njegovog konstruktora.

```
var myfunction = new Function("a", "b", "return a*b");
var x = myFunction(3,4);
```

Prethodni kod je ekvivalentan sledećem kodu:

```
Var myFunction = function(a,b) { return a*b };
```

Function Hoisting

Hoisting je osobina JavaScript jezika koja sve deklaracije funkcija pomera na početak fajla. Kao posledica ove osobine, funckije u JavaScriptu mogu biti pozvane i pre nego što se deklarišu.

Function arity

Interesantna osobina funckije je njena *arnost*, koja predstavlja broj parametara sa koji su navedeni prilikom njene deklaracije. Pošto se funkcije tretiraju kao objekti u JavaScript jeziku one imaju svoja svojstva i svoje metode. Na primer, svojstvo length će vratiti broj parametara koje funkcija sadrži, dok će metoda toString vtratiti kod funckije.

```
var a = function (a, b, c);
a.length == 3; // true
cosole.log(a.toString);
```

Iako se retko koristi na klijentskoj strani, ova osobina se koristi u nekim Node.js frejmvorkcima u slučajevima kada funckija treba da ima drugačiju logiku u zavisnosti od broja prosleđenih parametara.

Arrow functions

Drugačiji način zapisa funkcije je uveden u ES2015 standardu. Sintaksa ovakvog načina poziva funkije je:

```
(param1, param2, …, paramN) => { statements }
(param1, param2, …, paramN) => expression
// equivalent to: => { return expression; }

// Parentheses are optional when there's only one parameter name:
(singleParam) => { statements }
singleParam => { statements }

// The parameter list for a function with no parameters should be written with a pair of parentheses.
() => { statements }
```

Razlika u odnosu na korišćenje normalne notacije za deklaraciju funckije je u načinu na koji se koristi ključna reč this. One ne mogu biti korišćene kao konstruktori i nemaju lokalnu promenljivu arguments, kao ostale funckije.

P1: Kreirati funkciju izracunaj(), koja prihvata 3 parametra: pod1, pod2, tip, gde je tip u formatu sum, sub, div, mul. Funkcija vraca rezultat operacije tip, nad operandima pod1 i pod2. Koristeći kreiranu funkciju, izračunati vrednost izraza y = (2X + 3)/4. Za vrednost X uzeti proizvoljnu vrednost.

```
function izracunaj(pod1, pod2, tip)
{









}

var x = 3;
var y = izracunaj(izracunaj(izracunaj(2, x, 'mul'), 3, 'sum'), 4, 'div');
console.log(y);
```

P2: Funkciju iz prethodnog zadatka napisati u ES6 notaciji (arrow funkcija) i izračunati isti izraz. "Za vrednost X uzeti proizvoljnu vrednost.

```

```

P3: Formirati niz koji se sastoji od brojeva dana u svakom od 12 meseci u godini. Koristiti switch-case petlju.

```
var months = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11];
    switch(months) {
        case 0:
            console.log('Januar'); break;
        case 1:
            console.log('Februar'); break;
        case 2:
            console.log('Mart'); break;
        case 3:
            console.log('April'); break;
        case 4:
            console.log('Maj'); break;
        case 5:
            console.log('Jun'); break;
        case 6:
            console.log('Jul'); break;
        case 7:
            console.log('Avgust'); break;
        case 8:
            console.log('Septembar'); break;
        case 9:
            console.log('Oktobar');break;
        case 10:
            console.log('Novembar'); break;
       default:
            console.log('Decembar'); break;
    }
```

P4: Formirati funkciju maxElementNiza() koja vraća maksimalni element niza. Štampati indeks maksimalnog elementa niza Niz = [5,12,3,55,123] i njegovu vrednost u formatu: Maksimalni element je: 55 i nalazi se na 4. mestu.

```
function maxElementNiza(arr)
{
    var x = arr[0];
    var ind = 0;

    for(var i = 1; i<arr.length; i++) {
        if(arr[i] > x) {
            x=arr[i];
            ind=i;
        }
    }
    console.log('Maks je: ' + ind + 1 + '-i element' + x );
}

var Niz = [5,12,3,55,123];

maxElementNiza(Niz);
```

P5: Iz zadatog niza niz = [1, 2, 12, true, s, 17, false] u niz1 izdvojiti sve elemente čiji je tip number. Iskoristiti funkciju maxElementNiza() iz prethodnog zadataka i štampati maksimalni element niza niz1.

```
function maxElementNiza(arr)
{
    var x = arr[0];
    var ind = 0;

    for(var i = 1; i<arr.length; i++) {
        if(arr[i] > x) {
            x=arr[i];
            ind=i;
        }
    }
    console.log('Maks je: ' + ind + '-i element' + x );
}

var niz = [1,2,12,true, "s", 17, false];
var niz2 = [];

niz.forEach(function(n){
    if(typeof n == 'number') {
        niz2.push(n);
    }
});

maxElementNiza(niz2);
Pitanja:
```

1. Sta su primitivni a sta slozeni tipovi podataka? Navesti predstavnike oba tipa podataka u JavaScript-u?

2. Koje su kljucne reci za deklarisanje promenljivih u JavaScript-u? Objasniti razlike izmedju navedenih.

3. Objasniti termin interpolacija? Dati primer interpolacije u JavaScript jeziku.

4. Objasniti Hoisting osobinu JavaScript jezika?

5. Objasniti Arity osobinu JavaScript jezika?

Laboratorijska vežba broj 2: Objektno orjentisani JavaScript

Objekti - Kreiranje Objekata

Javascript objekti su kontejneri koji mogu da sadrže podatke i funkcije. U javaScriptu skoro sve, osim primitivnih tipova podataka, može da se posmatra kao objekat. Postoje tri načina za kreiranje novog objekta:

1. Pomoću literalne notacije
2. Kreiranje jedinstvenog objekta pomoću ključne reči new
3. Kreiranjem konstruktora, i zatim kreiranjem objekata pomoću konstruktora

P1: Kreiranje objekata pomoću literalne notacije i new operatora

```
let o1 = new Object();       o1.name = "mmm";   o1.age = 21;
Let o2 = {                   name: "mmm",       age: 21                };

let person = {               // kreira promenljivu person
  name: ['Bob', 'Smith'],          // {} kreira objekat
  age: 32,              // name, age… su kljucevi kojima se pristupa vrednostima
  gender: 'male',            // kljucevi se raydvajaju od vrednosti pomocu (:)
  interests: ['music', 'skiing']        // svaki par je raydvojen zarezom (,)
  }
};
```

Objekti o1 i o2 su potpuno isti. Nema razlike u kreiranju jednog i drugog, već se samo radi o drugačijoj notaciji.

P2: Pomoću new operatora

```
var myObj = new Object();
var me ={};
myObj.name = "Misa";
myObj.age = 23;
```

P3: Pristup članovima objekta pomoću DOT notacije i pomoću zagrada

```
let person = {
  name: 'Milos',
  age: 40
};

console.log(person.name)
console.log(person['age'])
person.age = 24;
```

Prednost pristupa članovima objekata pomoću zagrada je što se onda umesto ključa može koristiti i promenljiva. U slučaju dot notacije to nije moguće.

```
let person = {
  name: 'Tyron',
  age: 40,
  weekendAlarm: 'No alarms needed',
  weekAlarm: 'Alarm set to 7AM'
};

let day = 'Tuesday';    let alarm;
if (day === 'Saturday' || day === 'Sunday' ) {
  alarm = 'weekendAlarm';
} else
  alarm = 'weekAlarm';
console.log(person[alarm]);
```

P4: Definisanje ili dodavanje metoda objektu

```
let person = {
  name: 'Tyron',
  age: 40,
  weekendAlarm: 'No alarms needed',
  weekAlarm: 'Alarm set to 7AM',
  sayHello : function(){  return 'Hello, there!'; },
  sayGoodbye(){         return 'Goodbye!'    }    //ES2015 standard.
};
```

P5: Pozivanje metoda objekta

```
console.log(bob.getYearOfBirth());
```

P6: Ključna reč this

U javascriptu this predstavlja objekat koji je „vlasnik" trenutnog koda. Kada se koristi u funkciji this je objekat koji je vlasnik te funkcije. Napomenimo da this nije promenljiva, već ključna reč, što znači da se njena vrednost ne može promeniti.

Ova ključna reč nam omogućava da pristupimo članovima objekta iz samog tog objekta. U prethodnom primeru možemo dodati sledeću metodu:

```
sayHello : function(){
    return `Hello, my name is ${this.name}`;
  },
```

P7. Korišćenje reči this u funkcji iako objekat ne postoji.

```
// here we define our method using "this", before we even introduce bob
var setAge = function (newAge) {
  this.age = newAge;       //ako napisem samo age vratice gresku!!!!
};
// now we make bob
var bob = new Object();      bob.age = 30;      bob.setAge = setAge;
bob.setAge(50);         // change bob's age to 50 here
```

Ovakav način korišćenja this omogućava nam da objekti i metodi budu fleksibilniji. Istu funckiju setAge možemo dodeliti različtiim objektima i dokle god postoji član objekta age ona će mu dodeliti neku vrednost.

Geteri i seteri

Objektno orjentisana paradigma programiranja podrazumeva da članovima objekata treba pristupati pomoću getera i setera, odnosno odgovarajućih metoda. Postoji nekoliko prednosti kada se koriste ove metode za pristup. Moguće je izvršiti validaciu podataka pre upisa ili jemoguće kontrolisati pristup osetljivim podacima. JavaScript konvencija je da članove objekata kojima ne treba direktno pristupati, već preko get i set metoda, na početku imena doda znak (_).

P8: Primer get i set metoda

```
let person = {
  _name: 'Lu Xun',
  _age: 137,
  set age(ageIn) {            // seter metoda
    if (typeof ageIn === 'number') {
      this._age = ageIn;
    }
    else {
      console.log('Invalid input');
      return 'Invalid input';
    }
  },
  get age(){               // geter metoda
    console.log(`${this._name} is ${this._age} years old.`);
    return this._age;
  }
```

```
};

person.age = 'Thirty-nine'        // poziv seter metode
person.age = 39;
console.log(person.age);          // poziv getter metode
```

Nepromenljivost (Immutability) Objekata

Numbers, Strings and Booleans are all imutable. To znači da se njihova vrednost ne može promeniti. Sa druge strane za objekte kažemo da su mutable, odnosno da se vrednost njihovih članova može promeniti.

```
Var object1 = {value:10};
Var object2 = object1;
Var object3 = {value:10};

// object1 == object2   true
//object1 === object2   false
object1.value = 15
//object2.value    15
//object3.value    10
```

object1 i object 2 se referenciraju na isti objekat, dok object3 predstavlja poseban objekat.

JavaScript for...in Loop

Naredba for..in može da prođe kroz sva svojstva nekog objekta.Sintakse takve naredbe je:

```
                var person = {fname:"John", lname:"Doe", age:25};
for (variable in object) {      for (x in person) {
    code to be executed         txt += person[x];
}               }
```

Klase konstruktori i nasleđivanje

Moderni JavaScript je objektno orjentisani programski jezik kojim možemo da modeliramo stvari iz realnog sveta. Kao i u drugim objektno orjentisanim jezicima, moguće je kreirati objekte koji se ponašaju kao klase pomoćuz ključne reči class. Ovi objekti-klase se koriste za kreiranje drugih sličnih objekata.
Umesto da kreiramo objekat za svaku osobu, mi kreiramo klasu sa konstruktorom koji će kreirati objekat xxxx. Na taj način eliminišemo dupli kod. Glavna razlika između objekta i klase je što klasa sadrži constructor metodu. Svaki put kada klasa kreira (instancira) novi objekat ona poziva metodu constructor. Napomenimo još da se po konvenciji sva imena klasa pišu početnim velikim slovom.

P1: Kreiranje klase sa konstruktorom

```
class Surgeon {
  constructor(name, department) {
    this.name = name;
    this.department = department;
  }
}
```

P2: Kreiranje instance objekta

```
const surgeonCurry = new Surgeon('Curry', 'Cardiovascular');
const surgeonDurant = new Surgeon('Durant', 'Orthopedics');
```

P3: Kreiranje konstruktora sa metodama

Kreiranje metoda je isto kao i kod objekata, sa jednom biotnom razlikom. Izme]u metoda se ne stavlja znak (,).

```
class Surgeon {
  constructor(name, department) {
    this._name = name;
    this._department = department;
    this._remainingVacationDays = 20;
  }

  get name(){   return this._name;}
  get department(){  return this._department;   }
  get remainingVacationDays(){    return this._remainingVacationDays;  }
  takeVacationDays(daysOff){      this._remainingVacationDays  =   this._remainingVacationDays  -
daysOff;}
}

console.log(surgeonCurry.name);
surgeonCurry.takeVacationDays(3);
console.log(surgeonCurry.remainingVacationDays);
```

P4: Definisanje niza objekta 7.26

```
// Now we can make an array of people
var team = new Array();
team [0] = new Surgeon ("alice", 'Orthopedics');
team [1] = new Surgeon ("bob", 'Orthopedics');
team [2] = new Surgeon ("michelle", 'Orthopedics');
```

Prototipovi i nasleđivanje

```
Var empty = {};
console.log(empty.toString);       //function toString(){…}
console.log(empty.toString());     //[object Object]
```

Svi objekti imaju svoj prototip, odnosno su izvedeni iz klase **Object**. Za razliku od drugih progrmaskih jezika, u JavaScriptu objekat može imati samo jednog roditelja i svaki objekat čuva referencu na svog roditelja (i njegove metode i članove).

Kada probate da pristupite metodi ili članu nekog objekta, on prvo traži u tom objektu, pa zatim u svom prototipu, pa u prtotipu prototipa i tako redom.

P5: Primer nasleđivanja

```
class Doctor extends HospitalEmployee { }
class Animal {
  constructor(name) {
    this._name = name;
    this._behavior = 0;
  }
  get name() {     return this._name; }
  get behavior() {    return this._behavior; }
  incrementBehavior() {    this._behavior++; }
}

class Cat extends Animal {
  constructor(name, usesLitter) {
    super(name);
    this._usesLitter = usesLitter;
  }
}
```

Statički metodi klase

P6: Dodati statičku metodu u prethodnu klasu

```
static generateName() {
    const names = ['Angel', 'Spike', 'Buffy', 'Willow', 'Tara'];
    const randomNumber = Math.floor(Math.random()*5);
    return names[randomNumber];
  }

Animal.generateName();
o1 = new Animal("Dog");
o1. generateName();     // javice gresku!!!!! Staticka metoda ne moze se pozvati preko objekta
```

P7: Formirati klasu Server koja od atributa ima naziv, adresu(ip), i niz Umrezeni uredjaji.Od metoda ova klasa poseduje konstruktor i metode: Rename() koja menja naziv serveru, Reset() koja raskida vezu sa svim uparenim uređajima, Pair() koja uparuje server sa jednim uređajem, Unpair() koja raskida vezu sa željenim uredjajem. U glavnom programu kreirati Server, i prikazati izvšenje svih metoda.

```
class Server {

    constructor(name, address, pairedDevices)
    {
        this.name = name;
        this.address = address;
        this.pairedDevices = pairedDevices;
    }

    rename(anotherName){ this.name = anotherName; }

    reset(){ this.pairedDevices.length = 0; }
    pair(newDevice){ this.pairedDevices.push(newDevice); }
    unpair(device){ this.pairedDevices.splice(device); }
}

module.exports = Server;
var s = new server("VTS", "192.168.13.11", ["gearS3", "Host1", "Racunar13"]);

s.pair("Samsung Galaxy Watch");
s.rename("VTS1");
s.unpair(2);
console.log(s);
```

P8: Kreirati klasu Kvadrat. Od atributa klasa ima stranicu kvadrata, a od metoda konstruktor, duzinaStranice() koja vraca stranicu kvadrata, povrsina() koja racuna povrsinu kvadrata i metod obim() koja racuna obim kvadrata. U glavnom programu kreirati objekat klase kvadrat, i pozvati sve kreirane metode.

P9: Formirati klasu kvadrat kao iz prethodnog zadatka. U glavnom programu kreirati niz objekata klase Kvadrat i u konzoli odštampati onaj kvadrat koji ima najvecu povrsinu.

P10: Formirati klasu figura koja ima svoj podrazumevani konstruktor kao i metode obim i povrsina. Formirati i klase krug i trougao koje su nasleđene iz klase figura. Predefinisati konstruktor i metode iz nadklase respektivno. U glavnom programu napraviti po jedan objekat obe nasleđene klase i prikazati rezultat pozvanih metoda.

```
class Figura {
    constructor() { }
    obim() { }
    povrsina() { }
}

-----------------------------------------------------------------------------------------------
const Figura = require('./figura');
class Krug extends Figura {

    constructor(poluprecnik) {
        super();
        this.poluprecnik = poluprecnik;
    }

    povrsina() {
        return Math.pow(2, this.poluprecnik) * Math.PI;
    }

    obim() {
        return 2 * poluprecnik * Math.PI;
    }
}
```

```
-----------------------------------------------------------------------------------------------
module.exports = Krug;
const Figura = require('./figura');
class Trougao extends Figura {
    constructor(stranica_a, stranica_b, stranica_c) {
        super();

        this.stranica_a = stranica_a;
        this.stranica_b = stranica_b;
        this.stranica_c = stranica_c;
    }
    povrsina() {
        return (this.stranica_a * this.stranica_b) / 2; //P = bh/2
    }

    obim() {
        return this.stranica_a + this.stranica_b + this.stranica_c;
    }
}
module.exports = Trougao;

-----------------------------------------------------------------------------------------------
const krug = require("./krug");
const trougao = require("./trougao");

var k = new krug(4);
var t = new trougao(2, 3, 4);

console.log(t.povrsina());
```

P11: Koristeći klase iz prethodnog zadatka, u glavnom programu kreirati niz figura. U konzoli štampati figuru sa najmanjim obimom.

Kratak podsetnik

- Klase predstavljaju templejte za objekte
- JS poziva konstruktor metod prilikom kreiranja novih instanci, odnosno objekata
- Nasleđivanje podrazumeva kreiranje roditeljske klase sa metodama I članovima koje želimo da nasledi klasa dete.
- Ključna reč extends se koristi za kreiranje podklase
- Ključna reč super se koristi za pozivanje konstruktora roditeljske klase
- Statički metodi se mogu pozivati za klasu, ali ne i za njenu instance (objekat)

Moduli u JavaScript jeziku

Kreiranje modula u JavaScript jeziku omogućava kreiranje nezavisnih grupa klasa, objekata i funkcija u obliku biblioteka, koje se po potrebi uključuju i koriste u različitim projektima. Modul se kreira u zasebnom fajlu.

P1: Kreiranje modula

```
let Menu = {};
Menu.specialty = "Roasted Beet Burger with Mint Sauce";
module.exports = Menu;
```

Instrukcija module.export = Menu će izvesti objekat Menu kao modul. Taj modul će biti dostupan kao objekat za korišćenje iz drugih fajlova. U tom smislu da bi smo kreirali modul neophodno je:

1. Definisati objekat koji predstavlja modul
2. Dodati članove i metode u taj objekat
3. Izvesti (Export) modul

Da bismo uspešno koristili modul neophodno je uključiti ga u naš projekat ili fajl. Instrukcija koja uključuje modul u trenutni fajl je require(). Kao parameter se prosleđuje putanja do fajla. Objekat koji ova instrukcija vrati dodeljujemo promenljivoj.

P2: Korišćenje modula

```
const Menu = require('./menu.js');
function placeOrder() {  console.log('My order is: ' + Menu.specialty);  }
placeOrder();
```

U ES6 uvedena je pojednostavljena sintaksa za uvoženje modula. Postoji podela na dve vrste:

- Default export – koristi se export default umesto module.export
- Named export – kada želimo da uvezemo samo neki objekat iz fajla

Primetimo da import instrukcija traži putanju do modula, ali bez ekstenzije fajla.

P3: Primer default exporta

```
let Airplane = {
  availableAirplanes: [
    { name: 'AeroJet', fuelCapacity: 800},
    {name: 'SkyJet', fuelCapacity: 500}]
};
export default Airplane;
```

```
import Airplane from './airplane';
  console.log(Airplane.availableAirplanes);
```

P4: Primer named export-a

```
let person = "Milos";
function test(){  console.log("test");
function test1(){  console.log("test1");
export {person, test};
```

```
import { person, test } from './person';
  console.log(person);
    test();
```

P5: export on declaration

```
export let specialty = '';
export function isVegetarian() {  };
function isLowSodium() {     };   //ovo se ne izvozi
```

P6: Promena imena uvezenih objekata

```
export {person as p, test as t};
import {person as pp, test as tt};
```

Podsetnik:
Moduli u JavaScript jeziku predstavljaju delove koda koji se mogu više puta upotrebiti. Oni se mogu izvesti iz jednog programa u vrlo lako uključiti u drugi program.

- Module.exports naredba će izvesti modul tako da neki drugi program može da ga koristi
- Naredba require će uključiti modul u trenutni program.

Standatd ES6 je uveo nove načine zapisivanja uključivanja modula.

- Imenovani (*named export*) koristi ključnu reč export kako bi označio koji objekat fajla se izvozi
- Imenovani exporti mogu da budu preimenovani pomoću ključne reči as
- Klučna reč import uključuje bilo koji objekat ili modul u trenutni fajl

Pitanja

P1: Na koji nacin se mogu kreirati objekti u JavaScript-u? Dati primer.

P2: Objasniti znacenje i upotrebu kljucne reci this u Javascript jeziku.

P3: Za sta koristimo Get i Set metode? Dati primer gettera i settera u JavaScript-u.

P4: Objasniti 'Immutability' osobinu objekata.

P5: Sta je nasledjivanje, koji su benefiti koriscenja koncepta nasledjivanja?

Laboratorijska vežba broj 3: Upoznavanje sa NodeJS i Npm tehnologijama

Uvod u Node.JS

Node.js je programski jezik zasnovan na JavaScript jeziku. On je ne-blokirajući, *event driven, lightweight,* efikasan jezik čija je glavna namena da se koristi kod distribuiranih aplikacija koje rade na različitim platformama i koje imaju potrebu da rade sa velikim količinama zahteva ili podataka u realnom vremenu. Jezik ima sledeće karakteristike:

- Koristi se iz komandne linije i može se koristiti kao regularni veb-server
- Veoma je brz - Koristi Googlov V8 JavaScript engine
- Radi u jednoj niti (single thread) ali je skalabilan i dobar kada treba da radite više stvari u isto vreme. To se postiže pomoću event loop-a. Ovaj mehanizam omogućava da server odgovara asinhrono, bez blokiranja, za razliku od standardnih servera, kao što je Appache HTTP server, koji su ograničeni brojem niti koje mogu da kreiraju.
- Asinhron je i baziran je na događajima što je princip programiranja koji je poznat većini programera. Gotovo sve biblioteke NodeJs-a su asinhrone, odnosno ne blokirauće. To u suštini znači da node koji se iyvršavs na serveru nikada ne čeka da mu API vrati podatke. On nastavlja da izvršava kod, a kada podaci stignu oni kreiraju događaj (event) koji zatim pozivom odgovarajuće funkcije omogućava serveru da dobije odgovor od prethodno pozvane API funckije.
- Nema baferovanja – NodeJS aplikacije nikada ne baferuju podatke, već ih jednostavno šalju u paketima (chunks).
- Može komunicirati sa bazom podataka kao što je MySQL ili MongoDB
- Može koristiti isti kod i na serveru i na klijentu (JavScript u oba slučaja)
- Licenciran je sa MIT licencom

NodeJS je naročito pogodan za aplikacije koje moraju da održavaju perzistentnu konekciju sa serverom, najčešće korišćenjem veb soketa (primer takve aplikacije bi bio chat program). Mrežene aplikacije koje zahtevaju brzinu, skalabilnost i podržavaju veliki broj istovremenih konekcija se razvijaju u ovom programskom jeziku.

NPM Package Manager

Node Package Manager (NPM) ima dve glavne funkcionalnosti. Omogućava nam da;

- Pristupimo i pretražimo bazu postojećih modula na search.nodejs.org
- Pomoću programa komandne linije instaliramo module, upravljamo verzijama i upravljamo zavisnostima (dependancy management)

Neki od najpopularnijih NPM modula su:

- express – Express.js, web framework poput Sinatre. Standard je za većinu Node.js aplikacija danas.
- connect – Connect je HTTP server framework za Node.js, koji omogućava kolekciju pluginova širokog spektra performansi, poznatijih kao middleware.
- socket.io – Websocket komponente.
- Jade – Jedan od najpopularnijih templating engine-a, inspirisan HAML-om.
- mongo i mongojs – MongoDB omotač. Omogućava API za MongoDB objekte za bazu u Node.js
- redis – Redis klijentska biblioteka.
- coffee-script – CoffeeScript kompajler koji omogućava developerima da pišu svoje Node.js aplikacije u coffeescript-u.
- underscore (loadsh, lazy) – Jedan od popularnijih alatskih biblioteka u JavaScript-u.
- forever – Alat koji omogućava da vaša node skripta radi neprekidno i obezbeđuje da Node.js u proizvodnji ne dođe do nekih neočekivanih grešaka

Za insatalciju modula se koristi sledća naredba:

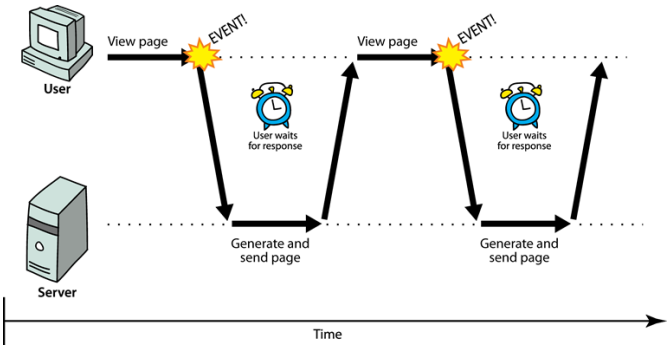
```
$ npm install learnyounode  -g
```

P0: U čemu je razlika između globalne (opcija –g) i lokalne instalacije modula? Navedite još par npm naredbi i šta one rade.

```
Opcija -g ća modul instalirati globalno, u suprtonom će  modul biti instaliran lokalno.
```

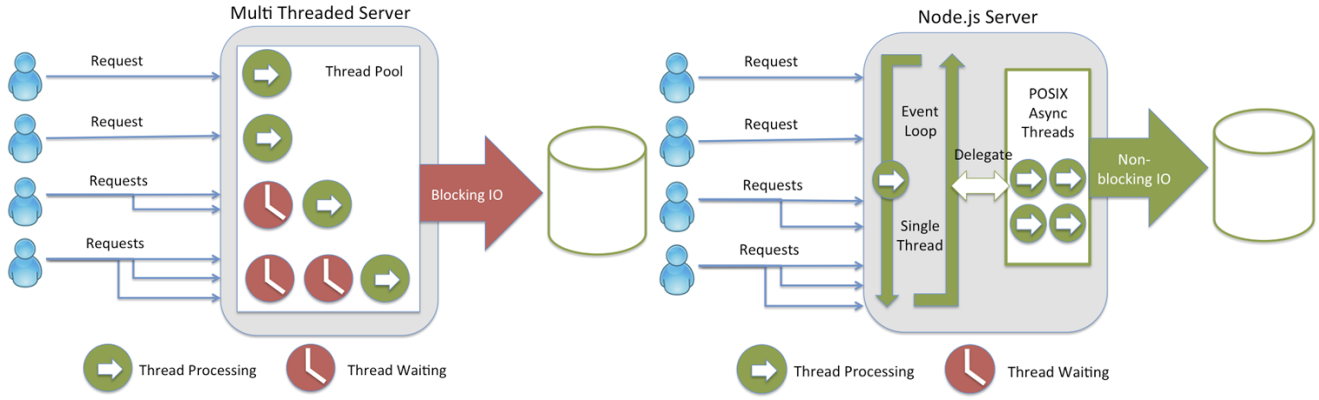
Asinhrono Programiranje

Tradicionalno, programiranje se obavlja sinhrono: Linija koda se izvršava, sistem čeka rezultat, rezultat se procesuira i zatim se izvršavanje programa nastavlja. Ponekad taj sistem izvršavanja zahteva dugo čekanje; na primer kada sa obavlja čitanje iz baze podataka.



Slika 1 - Primer izvršenja tradicinoalnog programa na serverskoj strani

U jezicima poput Jave i C#, rešenje za ovaj problem je uvođenje nove niti (*Thread*) i njeno izvršavanje. Višenitno programiranje može biti problematično kada više izvršavajuićih niti pokušava da pristupi istovremeno deljivim resursima. Zamislite situaciju u kojoj jedna nit povećava brojač, a druga ga u isto vreme smanjuje.
JavaScript ima sasvim drugačiji pristup takvom problemu. Uvek se izvršava samo jedna Nit. Kada se izvršavaju neke spore I/O operacije, kao što je čitanje podataka iz baze, program ne čeka, nego ide na izvršavanje sledeće linije koda. Kada se I/O operacija vrati, pokreće se callback funkcija i rezultat se procesuira. Node.js nudi jednostavan, brz, asinhroni event-driven model model programiranja za izradu modernih web aplikacija.



Slika 2 - Prikaz rada sinhorong i asinhronog programa

Zadatak 1. Instalirati Node JS na vašem računaru i podesiti path promenljivu.

Zadatak 2. Napisati program koji na standardni izlaz štampa „HELLO WORLD“. Pokrenuti program (na serveru) sa konzolne linije:

```
console.log("HELLO WORLD");
console.log(process.argv);
```

Kako bi pokrenuli program neophodno je prvo instalirati nodeJS, proveriti da li su path promenljive podešene i zatim u *comand prompt*-u otkucati:

node ime_programa [param1] [param2] …

P1:Šta je process.argv?

Process je globlni objekat koji pruža informacije o trenutnom procesu (progrmu) koji se izvršava.
https://nodejs.org/docs/latest/api/process.html#process_process
argv je svojstvo ovog objekta koji vraća vrednosti parametara koje su prosleđene programu prilikom poziva funkcije.

Zadatak 3. Napisati program koji sabira sve brojeve koje prosledimo kao argumente pri pozivu funkcije:

```
var result = 0
    for (var i = 2; i < process.argv.length; i++)
      result += Number(process.argv[i])

    console.log(result)
```

P2: Zašto petlja počinje od vrednosti 2? U slučaju da pokrenem program naredbom *node program Klijent Server,* koje vrednosti će imati promenljiva process.argv?

Prvi parametar je node. Drugi parametar je naziv ili putanja programa koji pokrećemo.

Zadatak 4. Napisati program koji računa N!. Argument n proslediti preko konzolne linije.

Rad sa fajlovima

Zadatak 5. Primer blokirajućeg i ne blokirajućeg čitanja iz fajla

```
var fs = require('fs');
var contents = fs.readFileSync('index.html');
console.log(contents);
console.log('KRAJ');
```

```
var fs = require('fs');
fs.readFile('index.html', function(err, contents){
   console.log(contents.toString());
});
console.log('KRAJ');
```

P3: Koji od ovih poziva je blokirajući? Zašto? Koje parametre prihvata funkcija readFile, a koje readFileSync?

P4: Šta se dobija kao rezultat prvog primera, a šta kao rezultat drugog. Objasniti?

P5: Objasnite zašto je jedan od primera neblokirajući? Kako se zove programerski princip koji ovo omogućava? Objasnite parametre err i contents?

P6: Koje naredbe možete iskoristiti za upis u fajl?

Rad sa strimom podataka

Zadatak 6. Čitanje fajla:

```
var fs = require('fs');
var file = fs.createReadStream('fruits.txt');
file.on('readable', function(){
  var chunk;
  while(null !== (chunk = file.read())){
    console.log(chunk.toString());
  }
});
```

P7: U čemu je razlika pri čitanju fajla kada koristimo stream i običnog čitanja u prethodnom primeru? Kada je zgodno koristiti jedan a kada drugi način čitanja?

Zadatak 7. Korišćenje pipe funkcije:

```
var fs = require('fs');

var file = fs.createReadStream("fruit.txt");
//TO DO
```

P8: Šta radi naredba require?

Zadatak 8. Napisati isti program tako da funkcija pipe ne zatvori strim podataka nakon završetka funkcije, već fajl zatvorite na drugi način.

```
var fs = require('fs');
var file = fs.createReadStream('origin.txt');
var destFile = fs.createWriteStream('destination.txt');
// TO DO

file.on('end', function(){
  destFile.end('Finished!');
});
```

Zadatak 9. Napisati program koji kopira celokupni sadržaj fajla index.html u fajl back_up.html.

Zadatak 10.Kreirajte fajl input.txt koji u sebi sadrži tekst "Klijent server sistemi. Lab. Vežba br. 3!". Zatim, napisati program zadatak9.js koji prikazuje podatke o fajlu input.txt.

```
var fs = require("fs");
console.log("Going to get file info!");
fs.stat('input.txt', function (err, stats) {
   if (err) {   return console.error(err);   }
   console.log(stats);
});
```

P9: Objasnite značenje svojstava objekta stats?

Zadatak 11.Izmenite prethodni program tako da se izvrši provera da li je fajl input.txt fajl, direktorijum ili soket.

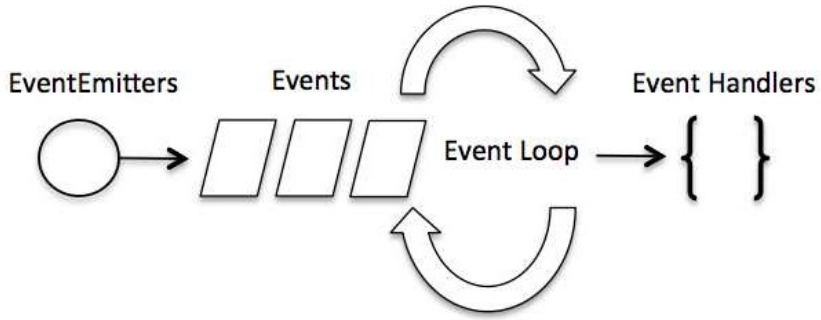
Preporučena literatura:

1. https://nodejs.org/
2. https://www.tutorialspoint.com/nodejs/index.htm
3. http://www.nodebeginner.org/
4. http://code.tutsplus.com/tutorials/node-js-for-beginners--net-26314
5. http://nodeschool.io/#workshoppers
6. https://www.airpair.com/javascript/node-js-tutorial
7. http://stackoverflow.com/questions/2353818/how-do-i-get-started-with-node-js
8. https://github.com/maxogden/art-of-node/#the-art-of-node
9. http://blog.modulus.io/absolute-beginners-guide-to-nodejs

Laboratorijska vežba broj 4: Upoznavanje sa mrežnim programiranjem

Programiranje vođeno događajima (Event Driven Programming)

Node.js je aplikacija koje se izvršava u jednoj niti, ali podržava paralelno izvršavanje programa pomoću događaja i callback funkcija. Svaki API u NodeJS tehnologiji je asinhron i na taj način podržava konkurentnost. NodeJS održava petlju sa događajima i svaki put kada se neki zadatak završi on okine odgovarajući događaj koji zatim signalizira odgovarajućoj hendlerskoj funkciji da se izvrši. U suštini svaki NoideJS program kada se pokrene prvo inicijalizuje promenljive i funkcije, a zatim čeka da se neki događaj desi. Glavna petlja programa (main loop) osluškuje da li desio neki događaj, a kada se on desi okine odgovarajuću hendlersku funkciju.



Slika 3 - Prikaz rada programa vođenog događajima

Iako događaji liče na callback funkcije, oni su u suštini različiti. Funkcija koja osluškuje događaje se ponaša po Observer obrascu i svaki put kada se događaj desi, odgovarajuća hendlerska funkcija se izvrši. Sa druge strane callback funkcije se pozivaju tek kada se neki asinhroni task ili funkcija završe i vrate rezultat.
NodeJS ima veliki broj predefinisanih događaja koji su dostupni preko EventEmitter klase.

Zadatak 1. KreiranjeEventEmitter objekta i primer emitovanja događaja:

```
var events = require('events');          // import events module
var eventEmitter = new events.EventEmitter()    // create EventEmitter object

eventEmitter.on("message", function(message){   //eventEmitter.on('eventName', eventHandler);
   console.log("TEST");
   console.log(message);
});

eventEmitter.emit('message');                 // to emit an event
eventEmitter.emit('message', 'pozdrav');        // to emit with args
```

Zadatak 2. Primer višestrukih hendlera za isti događaj:

```
var events = require('events');          // Import events module
var eventEmitter = new events.EventEmitter();    // Create an eventEmitter object
var firstHandler = function connected() {        // Create an event handler as follows
   console.log('connection succesful.');
   eventEmitter.emit('data_received');        // Fire the data_received event
}

eventEmitter.on('connection', firstHandler);     // Bind the connection event with the handler
eventEmitter.on('connection', function(){        // Second hendler
   console.log("Second Hendler!!!");
});

eventEmitter.on('data_received', function(){
   console.log(Data Event.');           //Bind the data_received event with the anonymous f.
});

eventEmitter.emit('connection');           // Fire the connection event
console.log("Program Ended.");
```

Rezultat poziva će biti redom First Hendler, Data Event, Second Hendler. Razlog za to je što kada eventEmitter emituje događaj, sve hendlerske funkcije koje su povezane sa tim događajem se pozivaju **sinkrono** (funkcija koja emituje događaj se blokira) **i to u redosledu kojim su dodavane.** Često programeri pogrešno pretpostave de je sam eventEmitter asinhron po prirodi jer se često koristi da signalizira kraj nekog asinhronog zadatka, što nije tačno. Prethodni primer to ilustruje. U primeru se emituje događaj connect. Usled toga se prvo poziva prva prijavljena funkcija firstHendler, koja štampa First Hendler, a zatim asinhrono emituje novi događaj. Ovaj događaj se obrađuje u event loopu i poziva se njegova hendler funkcija koja štampa Data Event. Tek nakon završetka obrade ovog događaja će roditeljska funkcija nastaviti izvršavanje i izvršiti naredbu nekon poziva metode eventEmitter.emit.

P0: Nakon zadnje naredbe u primeru 2 dodajte kod koji će da ukloni sve hendler funkcije koje osluškuju događaj data_received. Emitujte događaj kako bi demonstrirali da ga nijedna funkcija ne hvata.

P1:. Kako možete proveriti koliko funkcija osluškuje neki događaj?

P2: Kako možete da podesite da se neka hendlerska funkcija izvrši samo jednom?

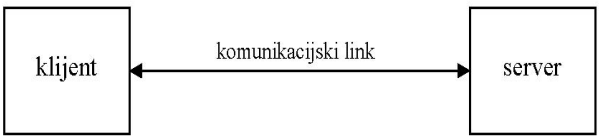
Zadatak 3. U NodeJS tehnologiji bilo koja asinhrona funkcija prihvata callback kao zadnji parametar. Svaka callback funkcija prihvata grešku (error) kao prvi parametar. Ilustrujmo to sledećim primerom: Kreirajte tekstualni fajl input.txt sa sledećim sadržajem: „ Klijent server sistemi nisu tako teški ☺“

```
var fs = require("fs");
fs.readFile('input.txt', function (err, data) {
   if (err){
      console.log(err.stack); return;
   }
   console.log(data.toString());
});
console.log("Program Ended");
```

P3: Kojim redosledom će biti ispisane poruke i zašto? Navedite još neku metodu klase EventEmitter:

Mrežno programiranje

Mrežno programiranje obuhvata pisanje programa koji komuniciraju sa drugim programima preko računarske mreže. Većina mrežnih aplikacija se može podeliti na dve vrste aplikacija i to jedan od programa uobičajeno se naziva ***klijent***, a drugi ***server***. Jedan klasičan primer klijent-server komunikacije je veb čitač kao klijent a veb server kao server (slika br.1).



Slika 4 - Mrežna aplikacija: klijent i server

Klijenti uobičajeno komuniciraju sa jednim serverom u jednom trenutku ali to nije pravilo (primer veb čitača koji može da komunicira sa više različitih servera). Sa serverske strane, sasvim je uobičajeno da server u

jednom trenutku ima konekciju na više klijenata istovremeno. Svi klijenti i serveri mogu se nalaziti u istoj lokalnoj mreži (LAN), ali je takođe moguće da oni budu u različitim mrežama, što je u većini situacija slučaj.

Transmission Control Protocol – TCP – je *connection –oriented* protokol koji obavlja pouzdanu i uređenu isporuku podataka od jednog do drugog računara. On se koristi kada želimo da dostavimo sve podatke u tačno određenom redosledu. Zbog ovog i drugih razloga većina protokola, koji se danas koriste su implementirani preko ovog transportnog protokola. Na primer veb čitači koriste HTTP, emajl klijenti SMTP, IMAP, POP, programi za četovanje IRC, XMPP, programi za udaljeni pristup SSH i drugi.

Uspostavljanje TCP veze

1. Server mora biti spreman da prihvati dolazeću vezu i to se uobičajeno ostvaruje pozivanjem ***socket, bind i listen*** funkcija. To stanje poznato je kao **passive open** stanje.
2. Klijent započinje aktivnim otvaranjem – stanje **active open** pozivanjem ***connect*** funkcije. To uzrokuje da TCP klijent šalje SYN segment (služi za sinhronizaciju) da se kaže serveru da je klijent inicijaliziovao broj sekvence za podatke koji će se slati preko uspostavljene veze. Uobičajeno je da se podaci ne šalju sa SYN, pa on samo sadrži IP header,TCP header i neke TCP opcije.
3. server mora da potvrdi klijentov SYN – šalje ACK, i server mora takođe da pošalje svoj SYN koji sadrži inicijalizovani broj sekvence za podatke koje će server slati na vezu. Oba podatka se šalju u istom segmentu.
4. klijent mora da potvrdi serverov SYN.

Kao što vidimo broj potvrdnih paketa je tri pa se zato uspostavljanje TCP veze naziva ***three-way handshake.***

Raskidanje veze TCP veze

Dok je kod uspostavljanja veze bilo potrebno razmeniti ukupno tri segmenta, za raskid veze potrebna su četiri segmenta i to:

1. Jedna od aplikacija prva zahteva raskidanje veze: poziv funkcije ***close***, stanje poznato kao **active close**. Ta strana šalje FIN segment što označava da slanje podataka završeno.
2. Druga strana koja primi FIN segment prihvata zatvaranje veze i izvodi **passive close**. Primljeni FIN je potvrđen od TCP-a i isti podatak se prosleđuje aplikaciji kao end of file (EOF).
3. Malo kasnije aplikacija koja primi EOF zatvara soket (funkcija ***close***) što uzrokuje da TCP pošalje FIN.
4. TCP na sistemu koji je primio taj zadnji FIN (kraj sa **active close**) potvrđuje FIN.

Soket

Da bi se uspešno uspostavila komunikacija dva procesa iz različitih aplikacija, bilo na istom ili posebnim računarima, potreban je par soketa. Prema tome soket predstavlja jedan kraj komunikacionog kanala i to za svaki proces po jedan. On je **određen IP adresom i brojem porta** preko koga se kanal za komunikaciju uspostavlja. Uglavnom se soketi koriste u klijent-server arhitekturi gde predstavljaju dominantan oblik uspostavljanja komunikacije. Server čeka zahteve klijenata tako što „sluša“ na određenom portu soketa. Oni obično implementiraju specifične servise na određenom portu (Telent-23, FTP-21, HTTP-80). Nepisano pravilo je da kad god se klijent spoji na soket, da mu se dodeljuje broj porta koji je veći od 1024, jer su svi brojevi portova manji od 1024 već rezervisani za standardne aplikacije. Za svaki par klijent server uvek se dodeljuje različiti broj porta, tako da imamo jedinstveni par soketa za svaki par. Postoji veliki broj različitih soketa ali se u praksi najčešće koriste dva tipa i to: **stream soket** i **datagram soket**. Stream soketi su pouzdani dvosmerni komunikacioni kanali kod kojih sve što se upiše na jednom soketu izlazi na drugom u istom obliku. Ovde su i sve moguće greške prilikom transporta ispravljene tako da se ima utisak da je soket bezgrešan – *error free*. Stream sokete koriste aplikacije kao što su Telnet i WEB čitač. Datagram soketi se još nazivaju *connectionless* tj. soketi bez stalne veze. Oni nisu toliko pouzdani kao prethodni soketi jer datagrami koji se šalju mogu ali i ne moraju da stignu do drugog soketa.

Koncept soketa

Kod kreiranja soketa moramo definisati tri osnovna parametra i to:

1. protokol – specificira kako se prenose podaci.
2. IP adresa
3. Broj porta

Osnovne funkcije za rad sa soketima

Postoje dva modula koja se koriste za rad sa soketima. Modul **net** se koristi za rad sa TCP soketima, a modul **datagram** za rad sa UDP soketima. Osnovne metode za rad sa modulom net su:

Klasa Net:

- Server net.createServer([options][, connectionListener])
- net.connect(options[, connectionListener])
- net.createConnection(options[, connectionListener])

Klasa Server:

- Server.listen(port);
- server.close([callback])

Klasa Socket:

- socket.connect(port[, host][, connectListener])
- socket.address()
- socket.write(data[, encoding][, callback])
- socket.on(event, callback); event može biti dana, close, open, end
- socket.end();

Zadatak 4. Kreiraćemo mali TCP server na koji može da se konektuje bilo ko. Kada se konektujete na server on će Vas pozdraviti i pitati za ime. Reći će Vam koliko je drugih korisnika konektovano. Nakon ukucavanja Vašeg imena smatra se da ste uspostavili konekciju. Kada ste konektovani možete da šaljete i primate ostalim klijentima. Otkucajte za početak sledeći kod:

```
var net = require("net");
var server = net.createServer(function(conn){
   console.log("'\033[90m Nova Konekcija!\033[39m");

}).listen(3000);
```

P4: Koji tip objekta vraća funkcija net.createServer()? Šta omogućava funkcija server.listen(port)? Da li je nakon poziva te funkcije TCP konekcija uspostavljena i da li je soket kreiran?

(Pogledaj https://nodejs.org/api/net.html#net_class_net_server)

Pokrenite vašu node aplikaciju. Testirajte je pomoću powershell aplikacije odnosno protokola. Otvoriti powershell klijent i ukucajte *Test-NetConnection -Port [vrednost] -ComputerName [vrednost]*. Za mac sisteme možete koristit *nc [adresa] [port]*

P5: Koju naredbu treba otkucati kako bi se testirao napisani program?

Zadatak 5. Računanje broja konektovanih klijenata: Proširite prethodnu aplikaciju tako da ona pamti broj konektovanih klijenta u promenljivoj count. Aplikacija treba i da vrati odgovor, broj konektovanih klijenata, onome koje uspostavio konekciju.

```
var net = require("net");
var count = 0;

var server = net.createServer(function(conn){
   var nickname;
```

```
   conn.write(
   '\r\n> welcome to \033[92mnode-chat\033[39m!'
   + '\r\n> ' + count + ' other people are connected at this time.'
   + '\r\n> please write your name and press enter: '
   );
   count++;
   console.log("`\033[90m new connection!\033[39m");

   conn.on(‘close’, function () {
      count--;
    });

}).listen(3000);
```

P6: Kada će soket conn da okine događaj *close* a kada događaj *end*?

(Pogledaj https://nodejs.org/api/net.html#net_class_net_socket**)**

Zadatak 6. **Primanje poruka** – Omogućimo sada našoj aplikaciji da primi poruke koje nam klijent šalje. Poruku ćemo logovati na serveru. Dodajte u prethodni primer sledeći kod:

```
conn.on('data', function (data) {
     console.log(data);
   });
```

P7: Da li ste uspeli da pošaljete i primate poruku? Šta je problem i zašto se javlja? Šta treba izmeniti da bi program ispravno štampao podatke?
(Pomoć: http://www.w3schools.com/jsref/jsref_tostring_number.asp)

Zadatak 7. **Emitovanje poruka**. Omogućiti aplikaicji da poslatu poruku prosledi svim klijentima zajedno sa imenom klijent. Klijent nakon uspostavljanja konekcije prvo šalje svoje ime (nickname) a tek onda može da šalje poruke. Dva korisnika ne smeju imati isto ime.
Tcp konekcija se ostvaruje između dva entiteta, klijenta i server. Da bi server poslao poruku svim klijentima on mora u nekoj programerskoj strukturi da čuva sve sokete koji se kreiraju svaki put kada se klijent prvi put konektuje na našu aplikaciju, odnosno pošalje svoj nadimak. U našoj aplikaciji ćemo kreirati asocijativni niz **users** koji će nadimak (**nickname**) koristiti kao indeks, a vrednost će biti soket objekat (**conn)**.

P8: Gde treba deklarisati promenljivu users, a gde promenljivu nickname?

Primljena poruka se prosleđuje svim aktivnim klijentima iz niza *users*. Za to koristimo pomoćnu funkciju broadcast koju smo sami napisali i kojoj kao parametre prosleđujemo poruku koju treba poslati svim klijentima.

```
function broadcast (msg) {
   for (var i in users) {
      users[i].write(msg);
   }
}
```

Napomena: Ukoliko koristite windows telnet program za testiranje serverske aplikacije, neophodno je napraviti malu modifikaciju. Widows Telnet program šalje podatke serveru nakon svakog otkucanog karaktera, i zato je neophodno da mi na serverskoj strani baferujemo te podatke u promenljivu str i tek kada registrujemo znak za novi red (enter) obradimo te podatke kao nadimak ili poruku. U ovom primeru mi ćemo koristiti samo jedan bafer što znači da naša aplikacija neće podržavati istovremeni unos poruka od strane više korisnika. Za to je potrebno koristiti posebni bafer za svakog korisnika.

```
if (data.toString('utf8') != "\r\n"){
        str = str + data;
        return;
     }
```

U slučaju da koristite neki od naprednijih programa kao što je putty, ova modifikacija nije neophodna. Dovoljno je u podešavanjima za putty program označiti stavku raw i uneti odgovarajući port i ip adresu.
Svaki put kada neki klijent pošalje podatke mi ćemo prvo proveriti da li je klijent poslao svoj nadimak, ako nije proveravamo da li klijent sa datim nadimkom postoji, i ako ne postoji dodajemo ga u niz aktivnih korisnika. Ukoliko je nadimak već poslat podatke od klijenta tretiramo kao poruku.

```
// dodati u delu gde se kreira server
conn.on('data', function (data) {
     str = data.toString();
     if (!nickname) {     //proveravamo da li je poslao nadimak ili poruku
        str = str.trim();
        if (users[str]) {
           conn.write('\033[93m> nickname already in use. try again:\033[39m ');
           return;
        } else {
           nickname = str;
           users[nickname] = conn;
           broadcast(nickname + ' joined the room\r\n');
        }
     }
     else{   // klijent je poslao poruku
        console.log(nickname + ": " + str);
        broadcast('nickname + ": " + str + '\r\n');
     }
     str = "";
  });
```

Zadatak 8. Izmenite kod u prethodnoj aplikaciji tako da se svim korisnicima pošalje poruka svaki put kada korisnik napusti chat aplikaciju. Napišite izmene:

Zadatak 9. Izmenite kod u prethodnoj aplikaciji tako da se poruka ne šalje osobi koja je poslala poruku (kako ne bi dolazilo do dupliranja poruka). Napišite izmene:

Zadatak 10.Kreirati program koji će otvoriti socket na portu 3490 i čekati da se neko spoji na njega (server program). Nakon što se neki klijent spoji na soket potrebno je ispisati "Konekcija je uspostavljena !" i regularno zatvoriti soket.

Zadatak 11.Napisati aplikaciju node JS koja omogućuje komunikaciju korišćenjem Datagram (UDP) soketa.

- Studenti sa parnim brojem indeksa prave serversku aplikaciju, a oni sa neparnim – klijentsku.
- Server zahteva unos porta na kome radi, a klijent adresu servera i porta sa kojim komunicira prilikom startovanja
- Nakon startovanja moguće je međusobno razmeniti poruke koje se unose sa tastature

Da biste uneli podatke sa tastature koristite modul readline. Njega uključujete u projekat naredbom i kreirate interfejs sledećim kodom:
(Pomoć https://nodejs.org/api/dgram.html)

```
var readline = require('readline');
var rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});
```

Nakon toga svaki put kada se na standardnom ulazu unese neki tekst i pritisne enter emitovaće se događaj 'line' koji kao parametar prenosi otkucani tekst i koji možete da uhvatite sledećim kodom:

```
rl.on('line', function (cmd) {
   console.log('You just typed: '+cmd);
});
```

Za korišćenje UDP soketa neophodno je koristiti modul dgram. Nakon uključivanja modula soket se kreira komandom:

```
var client = dgram.createSocket("udp4");
```

Ukoliko želite da primite pakete potrebno je da soket povežete sa nekim portom što se radi naredbom socket.bind(ime_porta). Adresi i portu na koji je soket povezan možete pristupiti sa socket.address() i socket.address().port

```
client.bind(3001, function() {
 // client.addMembership('127.0.0.1');
});
```

U trenutku kada nova poruka stigne na soketu emituje se događaj "message". Za toj događaj treba napisati handler funkciju koja hvata događaj i štampa ga na standardni izlaz. Msg je objekat tipa Buffer, a rinfo je objekat koji sadrži informacije o pošiljaocu poruke.

```
client.on('message', function(msg, rinfo) {
   console.log('>>%s:  Primio %d bajta od %s:%d\n', msg, msg.length, rinfo.address, rinfo.port);
});
```

Ukoliko želite da pošaljete poruku koristeći UDP soket, to možete uraditi naredbom socket.send, gde kao parametre prosleđujete poruku koju šaljete kao Buffer objekat ili string, pomeraj u baferu od koga počinje poruka, dužinu poruke, broj porta, ip adresa odredišta ili grupe odredišta, callback funkcija. Ukoliko želite da pošaljete broadcast poruku grupi odredišta neophodno je prvo postaviti fleg socket.setBroadcast(true), a zatim proslediti broadcast adresu u funkciji send.

```
client.send(cmd, 0, cmd.length, 3001, "localhost", function(err) {
   //client.close();
   });
```

Ukoliko prilikom testiranja programa naiđete na grešku „bind EADDRINUSE“ pokušajte da promenite port.

Napisati i testirati ceo program:

```
var dgram = require('dgram');                    var readline = require('readline');
var client = dgram.createSocket("udp4");
var rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});
client.bind(3001, function() {
  // client.addMembership('127.0.0.1');
});

rl.on('line', function (cmd) {       // console.log('You just typed: '+cmd);
  client.send(cmd, 0, cmd.length, 3001, "localhost", function(err) {
   //client.close();
   });
});

client.on('message', function(msg, rinfo) {
   console.log('>>>%s:    Received  %d  bytes  from  %s:%d\n',  msg,  msg.length,  rinfo.address,
rinfo.port);
});
```

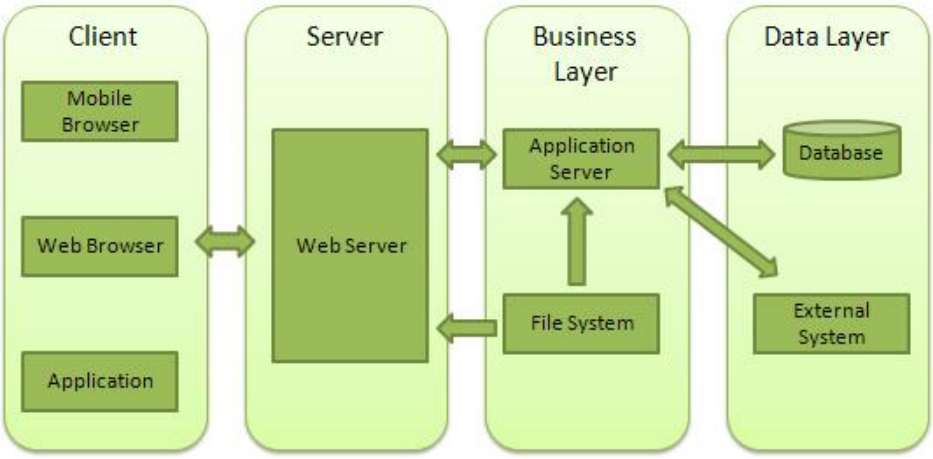
Zadatak 12.Izmenite prethodnu aplikaciju tako da se poruka šalje svima računarima na mreži i to na port 4444. Podesite da i vaša aplikacija osluškuje na tom portu. Testirajte aplikaciju tako što ćete poslati vaše ime ili neku drugu poruku i sačekati da Vam par kolega odgovori.

P9: Šta se dešava ako je i klijent i server registrovan na istom portu, ali se nalaze na istom računaru? Šta se dešava ako se i server i klijent aplikacija nalaze na istom portu ali na različitim računarima?

P10: Da li u slučaju primera 6 postoji razlika između klijentske i serverske aplikacije?

Zadatak 13.Izmenite kod u prethodnom primeru tako da aplikacija pamti sve poruke koje su poslate odnosno primljene u fajl koji se zove poruke.txt.

Laboratorijska vežba broj 5: Mrežno programiranje – koncept HTTP servera



Slika 5 - Slojevi aplikaicje

Primer 1: Kreiranje HTTP servera:
Ukoliko uporedite implementaciju TCP servera i HTTP servera, uočićete da su jako slične. U oba slučaja se poziva metoda *createServer* i zatim, kada se klijent konektuje, poziva se callback funkcija. Fundamentalna razlika je u tipu objekta koji dobijate u callback funkciji. U slučaju TCP servera dobija se *connection* odnosno soket objekat, a u slučaju HTTP servera dobijaju se *request* i *response* objekti.
Postoje dva razloga za to, prvo HTTP je API višeg nivoa koji Vam omogućava da upravljate specifičnim skupom funkcionalnosti vezanim za HTTP protokol. Drugo, moderni veb čitači su sposobni da pošalju nekoliko paralelnih konekcija kako bi optimizovali učitavanje veb strane. Node js Vam omogućava da ne brinete o samoj konekciji između klijenta i servera, mada TCP konekciji možete pristupiti preko parametra *request.connection*, jer vrši apstrakciju komunikacije na nivo zahteva i odgovora.

```
var http = require('http');

http.createServer(function(request, response) {
  response.writeHead(200);
  response.write("<h2>Hello, this is Server</h2>");
  response.end();
}).listen(8080);
```

Kako pokrenuti i testirati aplikaciju
Pokrenite serverski deo aplikacije tako što ćete na serveru otkucati **node name_of_the_program.**

Pokrenite klijentski deo aplikacije tako što ćete na lokalnom računaru iz veb čitača otvoriti odgovarajuću adresu i port servera. U slučaju lokalne mašine to će najverovatnije biti adresa http://127.0.0.1:8080/ ili iskoristite telnet ili **curl 127.0.0.1:8080** u konzoli.

P1: Testirajte kreirani http server pomoću Telnet ili curl protokola, a zatim i pomoću veb čitača. Otkucajte u telnet klijntu GET / HTTP/1.1 i pritisnite dva puta enter. Objasnite šta se desilo. Zbog čega veb čitač nije parsirao html dokument? Šta treba promeniti da bi se to ispravilo?

P2: Naš primer može da bude skraćen jer response.end() metoda kao parametar prihvata podatke koje treba vratiti kao parametar. Uklonite response.Write naredbu i umesto nje podatke koje treba vratiti prosledite pomoću response.end() funkcije.
(Pomoć https://nodejs.org/api/http.html#http_response_end_data_encoding_callback)

```

```

Primer 2: Vraćanje fajla putem HTTP protokola klijentu

```
var http = require('http');
var fs = require('fs');
http.createServer(function(request, response) {
  response.writeHead(200);
  fs.readFile( "index.html", function(err, data){
   response.write(data);
   response.end();
  });
}).listen(8080);
```

P3: U dokumentaciji na internet pronađite i na osnovu toga izmenite prethodni kod tako da tip fajla koji vraća naš http server bude text/html. (pomoć https://nodejs.org/api/http.html).

```
var http = require('http');
var fs = require('fs');

http.createServer(function(request, response) {
  //TO DO


  fs.readFile('index.html', function(err, contents) {
    response.write(contents);
    response.end();
  });

}).listen(8080);
```

P4: Prepravite prethodni kod tako da Vam server vrati neku html stranu. Pokušajte da rešenje implementirate korišćenjem naredbe pipe. Testirajte pokretanjem iz veb čitača.

```
var fs = require('fs');
var http = require('http');

http.createServer(function(request, response) {
  response.writeHead(200, {'Content-Type': 'text/html'});

//TO DO




}).listen(8080);
```

Primer 3: Prosleđivanje parametra serveru - Napišimo sada primer programa kome se u http zahtevu prosleđuju ime, prezime i broj indeksa. Aplikacija na serveru čita ove podatke, upisuje ih u tekstualni fajl i vraća ceo taj fajl nazad klijentu. Ukoliko aplikaciju pozovete bez parametara ili sa nepravilnim parametrima ona će samo izlistati studente koji su se upisali. Ukoliko joj pravilno prosledite parametre ona će vas upisati u listu studenata koji su uspešno uradili ovaj zadatak.

```
var fs=require('fs');var http = require("http");     var url = require("url");
var ind = 0;       var d = new Date();
fs.writeFile("Vezba8.txt", "\n\n\n Application started  " + d + "\n", { flag: "a" }, function(err){
  if (err) console.log(err);
});

var server = http.createServer(function (req, res){
  var o = url.parse(req.url, true);
  res.writeHead(200);
  var ime = o.query.ime;     var prezime = o.query.prezime;  var indeks = o.query.indeks;
  var str = ind + " " + o.query.ime + "  " + o.query.prezime + "  " + o.query.indeks + d + "\n";
  console.log(str);

  if (ime == undefined || prezime == undefined || indeks == undefined){
    res.write("Niste uspesno upisani \n");
    fs.readFile("Vezba8.txt", function(err, data){
      if (err) console.log(err);
      res.end(data);
    });
  }else {
    fs.writeFile("Vezba8.txt", str, { flag: "a" }, function(err){
      if (err) console.log(err);
    });
  ind++;
  res.write("Uspesno ste upisani \n");
  fs.readFile("Vezba8.txt", function(err, data){
    if (err) console.log(err);
    res.end(data);
    });
  }
}).listen(8080);
```

P5: Pozovite aplikaciju iz primera 3 koja se nalazi na serveru (pitati asistenta za ip adresu i port).

- **Proverite da li se nalazite na listi upisanih studenata. Ako Vas nema dodajte se u listu. Tako što ćete proslediti odgovarajući URL veb čitaču (ipadresa:port/?ime=Milos&….)**
- **Ispravite kod u prethodnoj aplikaciji tako da se fajl briše svaki put kada se izvrši upis u fajl.**
- **Isparavate kod tako da se fajl briše svaki put kada se pokrene aplikacija.**

Primer 4: Dobavljanje JSON fajla

Podatke iz prethodnog primera vratiti klijentu u obliku json fajla:

```
var fs=require('fs');        var http = require("http");
var url = require("url");       var stArr = [];
var server = http.createServer(function (req, res){
   var o = url.parse(req.url, true);
   res.writeHead(200);
   var student = {};                        student.ime = o.query.ime;
   student.prezime = o.query.prezime;       student.indeks = o.query.indeks;
   var op = o.query.op;

   if (student.ime== undefined || student.prezime == undefined || student.indeks == undefined){
        console.log( "Pogresan zahtev: " + stArr.length);
        var str = JSON.stringify(stArr);
   }
```

```
   else { //upisi studenta
      console.log( "Upisujem studenta: " + stArr.length);
      stArr.push(student);
      var str = JSON.stringify(stArr);
   }
   res.end(str);
}).listen(8080);
```

P6: Testirajte aplikaciju. Primetićete da svaki put kada pomoću veb čitača pošaljete HTTP zahtev server primi ustvari 2 HTTP zahteva. Zašto i koje zahteve primi vaš server?
(Pomoć: http://en.wikipedia.org/wiki/Favicon)

P7: Ispraviti kod iz prethodnog primera tako da se pri slanju odgovarajućeg zahteva obriše određeni student sa spiska, kao i da se spreči upisivanje studenata koji imaju identične podatke. Na koji način možete ovo realizovati?

P8: Kreirati html stranicu koja će vršiti prikaz, upis, proveru i brisanje studenata sa spiska. Napisati liniju koda iz ove stranice kojom se definiše ili poziva HTTP zahtev:

P9: Unaprediti aplikaciju iz prethodnog primera tako da se pri startovanju aplikacije izvrši čitanje, a pri bilo kakvoj promeni i upis u fajl, kako bi sprečili gubitak podataka u slučaju prestanka rada servera.

```
fs.readFile("Vezba8.txt", {flag:"a+"}, function(err, data){
   //To do



});

var saveFile = function(){
   // To do



};
//Poziv funkcije svaki put kada se niz stArr menja!
```

Primer 5: Dobavljanje slike

```
require("http").createServer(function (req, res) {
   var fs = require("fs");
   res.writeHead(200, { "Content-Type": "image/jpeg"});
   stream = fs.createReadStream("smile4.jpg");

   stream.on("data", function (data) {
      res.write(data);
   });

   stream.on("end", function () {
      res.end();
   });
}).listen(3000)
```

Primer 6: Primer poziva neke veb stranice i objekata request i response

Kreirajmo aplikaciju koja će pozvati neku drugu veb stranicu i prikazati njen odgovor. Prvo ćemo konstruisati odgovarajući URL korišćenjem modula request. Prvi parametar prilikom poziva metode request može biti URL ili objekat options. Članovi objekta options na koje treba obratiti pažnju su:

- url: URL adresa destinacije kojoj šaljemo zahtev
- method: HTTP metoda koju želimo da koristimo (GET, POST, DELETE, etc)
- headers: heder našeg zahteva koji se sastoji iz (key-value) parova
- form: objekat koji sadrži podatke forme u key-value obliku

```
var url = require('url');
var request = require("request");

// parameter je url                          //parameter je objekat options
var options = {                              //var options = {
  protocol: "https:",                        //url: 'https://www.reddit.com/r/funny.json',
  host: "twitter.com",                          //method: 'GET',
  // pathname: '1.1/search/tweets.json',        //headers: { 'Accept': 'application/json',
  // query: { q: "codeschool"}                  // 'Accept-Charset': 'utf-8', }
};                                           //};
var tURL = url.format(options);
});
```

Zatim koristimo request modul da pošaljemo jednostavan http zahtev i logujemo odgovor u konzoli.

```
request(tURL, function(err, res, body){           // request(options, function(err, res, body) {
  console.log("SERVER: " + res.headers.server);   // let json = JSON.parse(body);
  console.log("Last-Mod: "+res.headers['last-modified']);
});                                               //});
```

P10: Koje još članove ima objekat res, a koje objekat res.headers?

Primer 7: Primer veb servera sa obradom grešaka. Server se može testirati pozivom sledećih naredbi iz PowerShell-a:

- curl -method POST http://127.0.0.1:8081/2.1.html (HTTP status 405)
- curl http://127.0.0.1:8081/2.1.h (HTTP status 404)
- curl http://127.0.0.1:8081/2.1.html (HTTP status 200)

```
var http = require('http');     var fs = require('fs'); var url = require('url');
http.createServer( function (request, response) {           // Create a server
   var pathname = url.parse(request.url).pathname;          // Parse the req containing file name
   console.log("Request for " + pathname + " received.");   // Print who made the request
   if (request.method != 'GET'){          // HTTP Status: 405 : Bad REquest method
        response.writeHead(405, {'Content-Type': 'text/html'});
     response.end();
   }else{  // Read the requested file content from file system
      fs.readFile(pathname.substr(1), function (err, data) {
       if (err) {      // HTTP Status: 404 : NOT FOUND, Content Type: text/plain
        console.log(err);
        response.writeHead(404, {'Content-Type': 'text/html'});
       }else {         //Page found HTTP Status: 200 : OK Content Type: text/plain
        response.writeHead(200, {'Content-Type': 'text/html'});
        console.log("Page found HTTP Status: 200 " + pathname);
        response.write(data.toString());  // Write the content of the file to res body
       }
       response.end();            // Send the response body
      });
   }
}).listen(8081);
console.log('Server running at http://127.0.0.1:8081/');  //Console will print the message
```

P11: Šta bi se desilo kada bi metod response.end obrisali sa oba mesta gde se on pojavljuje i umesto toga ga napisali iznad metoda listen(8081)?

P12: Vaš broj indeksa podeliti sa 10. Ostatak koji dobijete je broj zadatka koji treba da uradite. Svaki student je dužan da demonstrira dobavljanje fajlova različitih formata, odnosno da prikaže nastajanje svake od grešaka navedenih u zadatku:

1. Napisati HTTP server koji je u stanju klijentima da vrati slike u JPG, GIF ili PNG formatu, ukoliko je zahtev korektan, a u suprotnom poruku sa odgovarajućim kodom (400, 404 ili 505).
2. Napisati HTTP server koji je u stanju klijentima da vrati tekstualne datoteke, ukoliko je zahtev korektan i postoji tražena datoteka na zadatoj lokaciji, a u suprotnom poruku sa odgovarajućim kodom (400, 404 ili 505).
3. Napisati HTTP server koji je u stanju klijentima da vrati html stranicu bez slika i drugih ugrađenih objekata (koju čita iz odgovarajuće datoteke), ukoliko je zahtev korektan i postoji tražena datoteka na zadatoj lokaciji, a u suprotnom poruku sa odgovarajućim kodom (400, 404 ili 505).
4. Napisati HTTP server koji je u stanju klijentima da vrati veličinu i datum kreiranja tekstualnog fajla ukoliko je zahtev korektan i postoji tražena datoteka na zadatoj lokaciji, a u suprotnom poruku sa odgovarajućim kodom (400, 404 ili 505).
5. Napisati HTTP server koji je u stanju klijentima da vrati datum zadnjeg pristupa i datum zadnjeg modifikovanja tekstualnog fajla ukoliko je zahtev korektan i postoji tražena datoteka na zadatoj lokaciji, a u suprotnom poruku sa odgovarajućim kodom (400, 404 ili 505).
6. Napisati HTTP klijenta koji utvrđuje i štampa na ekranu koji HTTP server pokreće sajt www.vtsnis.edu.rs. i još jedan sajt po vašem izboru, a u suprotnom poruku sa odgovarajućim kodom (400, 404 ili 505).
7. Napisati HTTP klijenta koji utvrđuje i štampa na ekranu datum kada je zadnji put modifikovana prva stranica na adresi: www.vtsnis.edu.rs., a u suprotnom poruku sa odgovarajućim kodom (400, 404 ili 505).
8. Napisati HTTP klijenta koji utvrđuje i štampa na ekranu veličinu stranice na adresi: www.vtsnis.edu.rs, a u suprotnom poruku sa odgovarajućim kodom (400, 404 ili 505).
9. Napisati HTTP klijenta koji utvrđuje i štampa na ekranu čitavo HTTP zaglavlje odgovora prilikom zahtevanja stranice na adresi: www.vtsnis.edu.rs, a u suprotnom poruku sa odgovarajućim kodom (400, 404 ili 505).
10. Napisati HTTP klijenta koji utvrđuje i štampa na ekranu sadržaj stranice na adresi: www.vtsnis.edu.rs, , a u suprotnom poruku sa odgovarajućim kodom (400, 404 ili 505).

Implementaciju HTTP servera proveriti prosleđivanjem zahteva preko nekog standardnog veb klijenta (npr. Internet Explorer ili Google Chrome) i pozivom curl naredbe iz powershell-a.

Laboratorijska vežba broj 6: Express Web Framework

Express frejmvork predstavlja najpoznatiji frejmvork za razvoj veb aplikacija zasnovanih na node.js programskom jeziku. U sebi sadrži podršku za rutiranje, konfiguraciju, templejt engine, POST parsiranje zahteva, i pristup različitim bazama podataka i druge funkcionalnosti. Iako predstavlja solidno rešenje, za sada se još uvek po funkcionalnostima ne može uporediti sa drugim *fullstack* frejmvorcima kao što su Rails, Larawell, Django i drugi…

Primer 1: Instalacija i podešavanje veb aplikacionog frejmvorka Express

1. Instalacija express frejmvorka:

```
$ npm init (kreira package.json)
$ npm install express –save (ne koristi se od npm 5, pamti modul u dependency listi package.json)
$ npm install express   (instaliranje modula, i od npm verzije 5 pamti u dependency listi)
```

Neki od modula koji se često koriste zajedno sa Express frejmvorkom su body-parser, cookie-parser, multer i drugi. Za pomoć pri korišćenju npm menadžera pogledajte *npm help install* i *man npm*.

2. Express nudi generator koji kreira skeleton strukturu vaše aplikacije. Da biste kreirali strukturu neophodno je otvoriti powerShell kao administrator i u komandnoj liniji otkucati:

```
$ npm install -g express-generator, a zatim otkucati komandu:
$ express --view=ejs myapp
```

Ovo će kreirati projekat sa imenom myapp u istoimenom direktorijumu, a templejt engine ejs će biti podržan. Moguće je podesiti i dodatne opcije express aplikacije kao što su CSS pretprocesor ili podrška za sesije. Ukoliko prethodna naredba nije prepoznata od strane command prompta verovatno je potrebno dodati u path promenljivu mesto gde je express generator instaliran (često je to C:\Users\username\AppData\Roaming\npm).

3. Otvoriti direktorijum gde je kreirana aplikacija i instalirati ostale module koji su zapamćeni u package.json fajlu. U protivnom NodeJS će vratiti grešku, obično će javiti da nedostaje neki modul ili favicon. Moduli za neku aplikaciju su definisani u packet.json fajlu i instaliraju se pomoću npm paket menadžera i naredbe:

```
$npm install
```

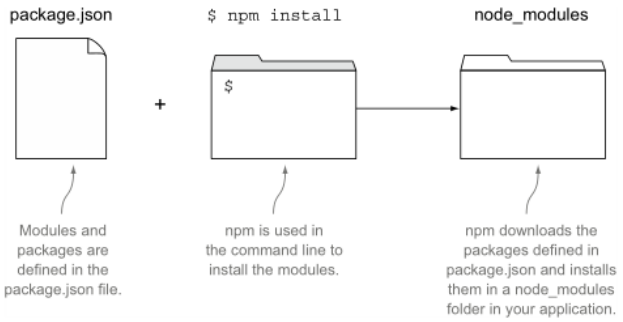
4. Aplikaciju je moguće pokrenuti pozicioniranjem u direktorijum aplikacije myapp i komandom

```
npm start (widows) ili
DEBUG=myapp:* npm start (Mac i Linux)
```

Ova komanda ustvari pokreće node bin/www skriptu što je definisano u fajlu package.json.

5. Da biste pristupili aplikaciji neophodno je u veb čitaču posetiti link http://localhost:3000/. Na slici ispod je prikazana fajl struktura aplikacije myApp koju smo kreirali u koraku 2. Proverite da li na je ona kreirana u vašem fajl sistemu.

```
.
├── app.js
├── bin
│   └── www
├── package.json
├── public
│   ├── images
│   ├── javascripts
│   └── stylesheets
│       └── style.css
├── routes
│   ├── index.js
│   └── users.js
└── views
    ├── error.ejs
    ├── index.ejs
    └── layout.ejs
```



Slika 6 - Instalacija modula pomoću npm menadžera

Prosleđivanje parametra sa klijenta

Razlika između param i query parametara:

- `req.params` – sadrži parametre rute (deo putanje URL stringa).
- `req.query` – sadrži URL parametar upita koji nisu deo putanje, odnosno navedeni su nakon znaka?
- `req.param(name)` – će tražiti promenljivu name na oba mesta, kao i u req.body

Primer 2: Slanje parametara u http zahtevu pomoću URL upita:

```
var express = require('express');
var app = express();
var quotes = {
  'einstein': 'Life is like riding a bicycle. To keep your balance you must keep moving',
  'berners-lee': 'The Web does not just connect machines, it connects people',
  'crockford': 'The good thing about reinventing the wheel is that you can get a round one',
  'hofstadter': 'Which statement seems more true: (1) I have a brain. (2) I am a brain.'
};

app.get('/quotes', function(req, res) {
   var name = req.query.name;
   var quote = quotes[name];
   console.log(req.query);
    res.end(quotes[name]);
   console.log(name, quotes[name]);
});

app.listen(3000);
//poziva se sa http://localhost:3000/quotes/crockford/
```

Primer 3: Slanje parametara u http zahtevu pomoću rute:

```
var express = require('express');
var app = express();

var quotes = {
  'einstein': 'Life is like riding a bicycle. To keep your balance you must keep moving',
  'berners-lee': 'The Web does not just connect machines, it connects people'
};

app.get('/quotes/:name', function(req, res){
   var name = req.params.name;
   res.end("Quote is " + quotes[name]);

});

app.listen(3000);
// poziva se sa http://localhost:3000/quotes/einstein
```

P1: Ukoliko u aplikaciji postoji ruta app.get('/hi/:param1', function(req,res){}) i u veb čitaču je otkucan URL http://www.google.com/hi/there?qs1=you&qs2=tube. Kolika će biti vrednost parametara req.query, a kolika req.param?

P2: Dodati kod u primeru 1 tako da se parametar šalje i kao deo URL upita i kao deo URL putanje. Korisniku vratiti oba citata, a ukoliko su citati isti obavestiti korisnika. Dati primer URLa koji će pozvati traženu stranu.

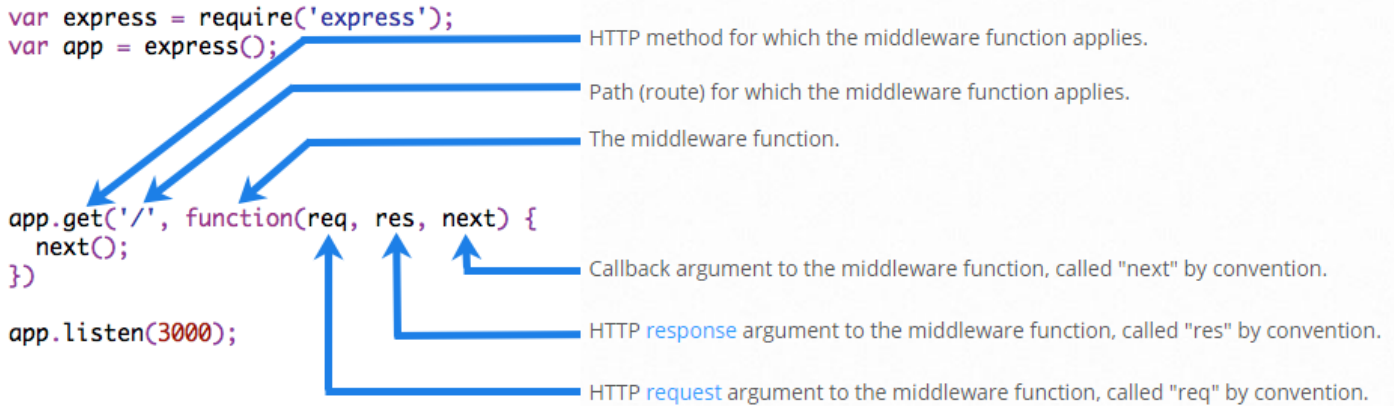
Korišćenje rutiranja – implementacija middleware sloja

Rutiranje predstavlja odgovor aplikacije na http zahtev klijenta. Zahtev klijenta, koji još nazivamo i endpoint sarži vrstu http zahteva (GET, POST, itd.) i putnju (URI). Svaka ruta ima jednu ili više hendlerskih funkcija koje će se izvršiti ako se rute poklapaju. Definicija rute ima sledeću strukturu:

```
app.METHOD(PATH, HANDLER)
```

gde je:

- App je instanca express objekta
- Method je vrsta HTTP zahteva zapisana malim slovima
- Path je putanja do servera
- Handler je funkcija koja će se izvršiti ako se rute poklope



Slika 7 - Primer kreiranja rute

Application-level middleware - Obično rutiranje

- App.method (path, callback [, callback ...]), method može da bude get, post, delete, put i drugi http metod
- App.all
- app.use([path,] function [, function...])
- next()
- next('route')

Primer 4: Primer kreiranja različitih ruta

```
var express = require('express');
var app = express(); // respond with "hello world" when a GET request is made to the homepage
app.get('/', function(req, res) {
  res.send('hello world');
});

app.post('/', function (req, res) {             // POST method route
  res.send('POST request to the homepage');
});

app.all('/secret', function (req, res, next) {
  console.log('Accessing the secret section ...');
  next();                     // pass control to the next handler
});

app.get('/secret', function (req, res, next) {
  res.send('Secrete area accessed!');
});

//This example shows a middleware sub-stack that handles GET requests to the/user/:id path.
```

```
app.get('/user/:id', function (req, res, next) {
  if (req.params.id == 0) next('route');  //if the user ID is 0, skip to the next route
  else next();          // or pass control to the next middleware f in this stack
}, function (req, res, next) {
  // returns a regular page
  res.send('regular id ' + req.params.id);
  next();
});

// handler for the /user/:id path, which returns a special page
app.get('/user/:id', function (req, res, next) {
  res.send('special');
});
app.listen(3000);
```

P3: Šta je postman (https://www.getpostman.com/). Korišćenjem postman aplikacije testirati svaku od ruta i zapisati URL koji pozivate i rezultat koji ste dobili. Šta bi se desilo ukoliko bi u ruti '/user/:id' obrisao next() metodu?

```
1. http://localhost:3000,
2. http://localhost:3000, metoda post
3. http://localhost:3000/secret, metoda get
4. http://localhost:3000/secret, metoda post
5. http://localhost:3000/user/55,
6. http://localhost:3000/user/0
7. Ukoliko obrišemo next metodu:
```

Primer 5: Primer korišćenja app.use metode:

```
var express = require('express');          var app = express();
//A middleware f-on with no mount path. The f-on is executed every time the app receives a req.
app.use(function (req, res, next) {
  console.log('Time:', Date.now());
  next();
});

app.use('/', function (req, res, next) {
  console.log('Request Type:', req.method);
  next();
});

app.use('/', function (req, res, next) {
  console.log('Treci poziv');
  res.end("KRAJ");
});
app.listen(3000);
```

P4: U koju svrhu biste vi iskoristili metodu app.use? Šta se dešava ako se next() ukolni iz neke od ruta?

```
TO DO
```

Third-party middleware

Express često koristi module koje je napravio neko drugi. Potrebno je samo instalirati traženi modul koristeći npm paket menadžera, uključiti ga u vašu aplikaciju koristeći require na nivou aplikacije ili rutera. Za detalje pogledajte dokumentaciju: **THIRD-PARTY MIDDLEWARE**.

Pribavljanje statičkog sadržaja

Počevši od verzije 4.0 Express više ne zavisi od biblioteke Connect. Osim Express.static modula, svi ostali midleware moduli su sada zasebni moduli. Listu dostupnih module možete pogledati ovde: (**LISTA MODULA**). Jedini middleware ugrađen u Express biblioteku je Express.static i odgovoran je za pristup ili pribavljanje

statičkih resursa kao što su HTML fajlovi, slike i drugo. Primer poziva funkcije ovog modula je: `express.static(root, [options])`
Parametar ***root*** predstavlja root direktorijum fajlova kojima treba pristupiti. Parametar ***options*** specificira neke detalje i uslove pod kojima se može pristupiti ovom fajlu. Moguće je dodati i više ruta, i u tom slučaju nodeJS će traženi fajl pretraživati po redosledu dodavanja statičkih ruta. Ilustrujmo primerom korišćenje ovog modula:

Primer 6: Primer podešavanja direktorijuma sa sadržajem kome se može pristupiti statički

```
var express = require('express');           var app = express();

var options = {
  dotfiles: 'ignore',
  etag: false,
  extensions: ['htm', 'html'],
  index: false,
  maxAge: '1d',
  redirect: false,
  setHeaders: function (res, path, stat) {
    res.set('x-timestamp', Date.now());
  }
}
//Može da postoji više direktorijuma za statičke fajlove:
app.use(express.static('Files'));
app.use(express.static('Images', options));

app.listen(3000);
```

Za dodatne informacije o `serve-static` funkcionalnostima i opcijama pogledajte dokumentaciju.

P5: Dodati još dve proizvoljne rute za rutiranje.
Rute treba da imaju različiti http metod i različite putanje. Navedite još par Third-party middlewara modula. Instalirati modul morgan i dodati ga u vašu aplikaciju. Ovaj modul vam omogućava automatsko logovanje svih zahteva koje vaš server prmi. Dodajte ga u putanju za rutiranje i testirajte pomoću aplikacije postman za bar 3 različita tipa HTTP zahteva. Napišite kod koji ste dodali u aplikaciju:

U express frejmvorku je moguće dodati i virtuelne rute. To su rute koje u fajl sistemu ne postoje, ali iz određenih razloga mi želimo da ih kreiramo. Rute se dodaju pozivom metode app.use('/static', express.static('public')), gde je prvi parametar virtuelna putanja, a drugi stvarna putanja u fajl sistemu.

P6:Dodajte još dve virtuelne rute u prethodni primer i napisati URL kojim se one mogu testirati:

Rutiranje vezano za instancu klase – Router-level middleware

Router-level middleware funkcioniše na isti način kao i rutiranje na nivou aplikacije samo što je vezan za instancu klase `express.Router()`. Koristi iste funkcije, kao i aplikaciono rutiranje a to su router.use() i router.METHOD(). Koristi se najčešće kada želimo da u okviru jedne aplikacije napravim hijerarhijsko rutiranje. Ilustrujmo to primerom.

Primer 7: Rutiranje pomoću ruter klase

```
var express = require('express');
var router = express.Router();

// middleware that is specific to this router
router.use(function timeLog(req, res, next) {
  console.log('Time: ', Date.now());
  next();
});

// define the home page route
router.get('/', function(req, res) {
  res.send('Birds home page');
});

// define the about route
router.get('/about', function(req, res) {
  res.send('About birds');
});

module.exports = router;
_____________________________________________________________________________
//Then, load the router module in the app.js file:
var birds = require('./birds');
app.use('/birds', birds);
```

P7: Primer 5 proširite ali tako da se skup ruta iz primera 7 uključi kao moduli u primer 5. Zapisati izmene u primeru 6 i testirati.

Ne zaboravite da u primeru 5 obrišete res.end kako ne bi vratio http odgovor, kao i da prosledite HTTP zahev sledećoj funkciji na dalju obradu.

```
TO DO
```

Sloj za upravljanje greškama - Error-handling *middleware*

Middleware koji se bavi obradom grešaka uvek sadrži 4 parametra koja je neophodno proslediti funkciji. Čak i kada ne želimo da koristimo next objekat, neophodno ga je proslediti u potpisu funkcije, u suprotnom next će biti protumačen kao deo normalnog middlewae sloja i neće obraditi nastale greške. Parametri koje treba proslediti su err, req, res i next. Pokažimo to primerom:

Primer 8: Primer sloja za hvatanje i prikaz greške

```
app.use(function(err, req, res, next) {
   console.log("ERROR COUGHT!!!!!!!!!!!!!!!!!!");
   console.error(err.stack);
   res.status(500).send('Something broke!');
});

//Primer Izazivanje greške
fs.readFile("Vezba8.txt", {flag:'a+'}, function(err, data){
        if (err) {
           console.log("ERROR Thrown!");
           return next(err);
        }
```

Za detalje pogledajte dokumentaciju na sledećem linku: **Error handling.**

P8: Namerno izazvati grešku i uhvatiti je pomoću sloja za upravljanje greškama.

```
TO DO
```

Primer 9: Vraćanje slike ili fajla koristeći Express.static

```
//index.html
<a href="http://localhost:3000/smile1.jpg" value="slika1" > slika 1</a>
<a href="http://localhost:3000/smile2.png" value="slika2" > slika 2</a>
<a href="http://localhost:3000/smile3.gif" value="slika3" > slika 3</a>

//primer10.js
var express = require('express');           var app = express();
app.use(express.static('Views'));           app.use(express.static('Images'));
app.listen(3000);
```

P9: Dodati još jedan direktorijum i još dve proizvoljne slike i izmeniti kod u primeru 10 tako da je moguće pristupiti tim slikama koristeći Express.static.

```
TO DO
```

Za dalji razvoj preporučuje se korišćenje nekog IDE alata. Za izradu primera koji prate ovu vežbu je korišćen IDEA alat kompanije JetBrains sa instaliranim modulom koji podržava razvoj nodeJS aplikacija..

Primer 10: Vraćanje slike ili fajla, ali bez korišćenja Express.static

- Res.sendFile(path, options, callback);- Path mora da bude apsolutna putanja.

```
//Fajl zad.js
var express = require('express');
var app = express();
var path = require('path');
app.get('/', function(req, res, next) {   res.send('Choose the zad number!!!!');    });

app.get('/zad1', function (req, res){
  var s = req.query.s;
  if (s == undefined){
     var p = path.join(__dirname, 'Views/', '9.10.html');
     res.sendFile(p);
  }
  else if (s == "slika1") {
     var p = path.join(__dirname, 'Images/', 'smile1.jpg');
     res.sendFile(p);
  }
  else if (s == "slika2") {
      var p = path.join(__dirname, 'Images/', 'smile2.png');
      res.sendFile(p);
  }
  else if (s == "slika3") {
     var p = path.join(__dirname, 'Images/', 'smile3.gif');
   res.sendFile(p);
  }
});
app.listen(3000);

//zad.html
<form action="/zad/zad1" method="get">
    <select name="s">
        <option value="slika1"> slika 1</option>
        <option value="slika2"> slika 2</option>
        <option value="slika3"> slika 3</option>
    </select>
    <input type="submit"></input>
</form>
```

P10: Izmeniti strukturu vaših zadataka tako da se svi oni pozivaju okviru jedne nodeJS aplikacije i to pozivom odgovarajuće rute. Na primer zadatak 6 bi se pozivao pozivom rute ipaddress:port/zadatak6.

```
TO DO
```

P11: Podeliti vaš broj indeksa po modulu 10. Dobijeni broj predstavlja broj zadataka koji treba da uradite:

1. Napraviti HTML formu za izbor jedne od tri ponuđene slike i NODE aplikaciju koja obrađuje taj zahtev i vraća tražene slike ili poruku da slika nije nađena. Koristiti HTTP GET metodu i SELECT tag za izbor jedne od tri ponuđene slike.
2. Napraviti HTML formu za izbor jedne od tri ponuđene slike i NODE aplikaciju koja obrađuje taj zahtev i vraća tražene slike ili poruku da slika nije nađena. Koristiti HTTP POST metodu i SELECT tag za izbor jedne od tri ponuđene slike.
3. Napraviti HTML formu za izbor jedne od tri ponuđene slike i NODE aplikaciju koja obrađuje taj zahtev i vraća tražene slike ili poruku da slika nije nađena. Koristiti HTTP GET metodu i RADIO dugmadi za izbor jedne od tri ponuđene slike.
4. Napraviti HTML formu za izbor jedne od tri ponuđene slike i NODE aplikaciju koja obrađuje taj zahtev i vraća tražene slike ili poruku da slika nije nađena. Koristiti HTTP POST metodu i RADIO dugmadi za izbor jedne od tri ponuđene slike.
5. Napraviti HTML formu za unos srednjih ocena jednog studenta po godinama studija (po jedno polje za jednu godinu/broj) i NODE aplikaciju koja računa ukupnu srednju ocenu i vraća je klijentu u obliku HTML stranice. Koristiti HTTP GET metodu za prosleđivanje parametara.
6. Napraviti HTML formu za unos srednjih ocena jednog studenta po godinama studija (po jedno polje za jednu godinu/broj) i NODE aplikaciju koja računa ukupnu srednju ocenu i vraća je klijentu u obliku HTML stranice. Koristiti HTTP POST metodu za prosleđivanje parametara.
7. Napraviti HTML formu za sastavljanje računarske konfiguracije (zbog jednostavnosti podrazumevati da se mogu izabrati samo 2 komponente, npr. procesor i ploča, svaka sa po 3 različita modela) i NODE aplikaciju koja računa ukupnu cenu konfiguracije i vraća je klijentu u obliku HTML stranice. Koristiti HTTP GET metodu za prosleđivanje parametara.
8. Napraviti HTML formu za sastavljanje računarske konfiguracije (zbog jednostavnosti podrazumevati da se mogu izabrati samo 2 komponente, npr. procesor i ploča, svaka sa po 3 različita modela) i NODE aplikaciju koja računa ukupnu cenu konfiguracije i vraća je klijentu u obliku HTML stranice. Koristiti HTTP POST metodu za prosleđivanje parametara.
9. Napraviti HTML formu za sastavljanje računarske konfiguracije. Osnovna konfiguracija je 500E, a korisnik može dodati još DVD rezač (30), štampač (50) i TV tuner (40). NODE aplikacija računa ukupnu cenu konfiguracije i vraća je klijentu u obliku HTML stranice. Koristiti HTTP GET metodu za prosleđivanje parametara.
10. Napraviti HTML formu za sastavljanje računarske konfiguracije. Osnovna konfiguracija je 500E, a korisnik može dodati još DVD rezač (30), štampač (50) i TV tuner (40). NODE aplikacija računa ukupnu cenu konfiguracije i vraća je klijentu u obliku HTML stranice. Koristiti HTTP POST metodu za prosleđivanje parametara.

Literatura:

- https://github.com/maxogden/art-of-node/#the-art-of-node
- https://www.safaribooksonline.com/blog/2014/03/10/express-js-middleware-demystified/

Laboratorijska vežba broj 7: Ejs i jade templetski jezici

Izgled Express aplikacije

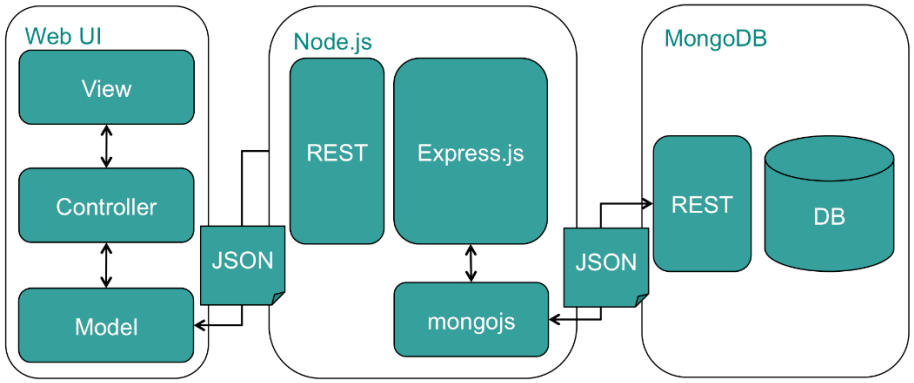
U ovoj vežbi će biti predstavljeni neki napredni koncepti i tehnike programiranja veb aplikacija. Pre nego što počnemo da se bavimo ovim tehnikama neophodno je još jednom kratko sumirati tipičnu strukturu jedne veb aplikacije pisane u Express frejmvorku. Glavni fajl aplikacije se najčešće naziva app.js ili server.js i sastoji se iz:

1. Zavisnosti (Dependencies)– skup naredbi koje u naš fajl uključuju različite module i biblioteke.
2. Instanci (Instantiations)– skup naredbi koje kreiraju objekte
3. Konfiguracije (Configurations)– skup naredbi koje podešavaju sistem
4. Srednji sloj (Middleware)– skup naredbi koje se izvršavaju za svaki dolazeći zahtev
5. Rute (Routes)– skup naredbi koje definišu serverske rute, krajnje tačke (endpoints) i strane
6. Startovanje (BootUp) – skup naredbi koji pokreće server i počinje da osluškuje na odgovarajućem portu dolazeće zahteve.

Full Stack Development

Obuhvata razvoj aplikacije koja se izvršava na serveru (back-end) i klijentu (front-end) i najčešće koristi bazu podataka za smeštanje podataka.

Troslojna (Three tier) arhitektura



Slika 8 - Izgled troslojne arhiekture

Napomena: Za primere i zadatke u nastavku vežbe se preporučuje korišćenje razvojnog okruženja IDEA kompanije jetbreins.

Template Engines

Kada se koristi veb frejmvork Express postoje 4 template engina koja se sa njim mogu koristiti. To su Jade, EJS, JsHtml i Hogan. Svi templejti funkcionišu na isti način. U html kodu postoje određena mesta – placeholderi u koje treba umetnuti neku vrednost. Pri pozivu nekog templejta enginu treba proslediti odgovarajuće vrednosti koje će on zatim umetnuti na odgovarajuće mesto. Ovaj proces se često naziva renderovanje strane.

Korišćenje ejs templejta i funkcija render

Primer 1: Vraćanje nekog teksta

```
var express = require('express');
var app = express();
app.set('views', path.join(__dirname, '/Views'));

var quotes = {
  'einstein': 'Life is like riding a bicycle. To keep your balance you must keep moving',
```

```
  'berners-lee': 'The Web does not just connect machines, it connects people',
  'crockford': 'The good thing about reinventing the wheel is that you can get a round one',
  'hofstadter': 'Which statement seems more true: (1) I have a brain. (2) I am a brain.'
};

app.get('/quotes', function(req, res) {
  var quote = quotes[req.query.name];
  res.locals = {
     name: req.params.name,
       quote: quote
};
  res.render("quote.ejs");
});
app.listen(3000);

//sadržaj fajla Views/quote.ejs
<h2>Quote by <%= name %></h2>
<blockquote>
  <%=  quote %>
</blockquote>
```

poziva se sa http://localhost:3000/quotes?name=einstein

P1: Dodati još jednu stranicu koja će se zvati citat.ejs i koja će ispisati sve moguće citate, a zatim i citat koji je odabran.

Odabrani citat treba da bude istaknut (veći font i podebljan). Izmeniti kod tako da se prikaže novo-kreirana stranica.

```
TO DO
```

P2: U kom direktorijumu će Express tražiti ejs fajlove. Da li je moguće promeniti ovaj direktorijum i kako?

Primer 2: Prijem parametra, prosleđivanje promenljive templejtu i vraćanje teksta, fajla i promenljive.

Primer je urađen za projekat kreiran u IDEA razvojnom okruženju.

- res.sendFile(path, options, callback);- Path mora da bude apsolutna putanja.

```
//Fajl zad.js
var express = require('express');    var router = express.Router();  var path = require('path');

router.get('/', function(req, res, next) {
  res.send('Choose the zad number!!!!');
});

router.get('/zad1', function (req, res){
  var s = req.query.s;
  if (s == undefined)
    res.render('zad', { title: "Zadatak 1"});
  else if (s == "slika1") {
      res.send("slika1");
  }
  else if (s == "slika2") {
      var p = path.join(__dirname, '../public/images/', 'slika2.jpg');
      res.sendFile(p);
  }
  else if (s == "slika3") {
      res.send res.send(msg:'slika3.jpg');
);
  }
});

//Fajl app.js
var zad = require('./routes/zad');
app.use('/zad', zad);
```

//Fajl zad1.ejs

```
<form action " method="get">
    <select name="s">
        <option value="slika1"> slika 1</option>
        <option value="slika2"> slika 2</option>
        <option value="slika3"> slika 3</option>
    </select>
    <input type="submit"></input>
</form>
```

P3: Izmeniti kod tako da ispisuje naziv slike koja je odabrana zajedno sa njenom veličinom, i ip adresom kompjutera koji je zahtevao sliku, a zatim prikazati i samo sliku ispod.

```
TO DO
```

Jade

Jade template engine je na prvi pogled neobičan jer ne sadrži HTML tagove. Umesto toga on koristi minimalistički pristup koji koristi imena tagova, razmak, i CSS-like metodu za referenciranje za definisanje HTML koda. Jedini izuzetak je div tag, koji se toliko često koristi da jade, u slučaju da je tag u templejtu preskočen, ga sam podrazumeva. U nastavku će biti prikazan primer jade koda i kompajliranog ili renderovanog izlaznog html fajla:

```
#banner.page-header
h1 My page
p.lead Welcome to my page
```

```
<div id="banner" class="page-header">
   <h1>My page</h1>
   <p class="lead">Welcome to my page</p>
</div>
```

Iz navedenog primera možemo zaključiti da kada tag nije specificiran jade doda div tag kao u prvoj liniji koda. Kod #banner postaje id=banner u HTML-u. Kod .page-header postaje class=“page-header“ u HTML-u.

Primer 4: Kompletan index.jade fajl

```
extends layout
block content
   h1= title
   p Welcome to #{title.}
```

Podrazumevani izgled (*layout*) jade fajla

```
doctype html
html
   head
     title= title
     link(rel='stylesheet', href='/stylesheets/style.css')
   body
     block content
```

```
   script(src='/bootstrap/js/bootstrap.min.js')
```

P4: Napisati primer 2, ali pomoću jade templejta.

```
TO DO
```

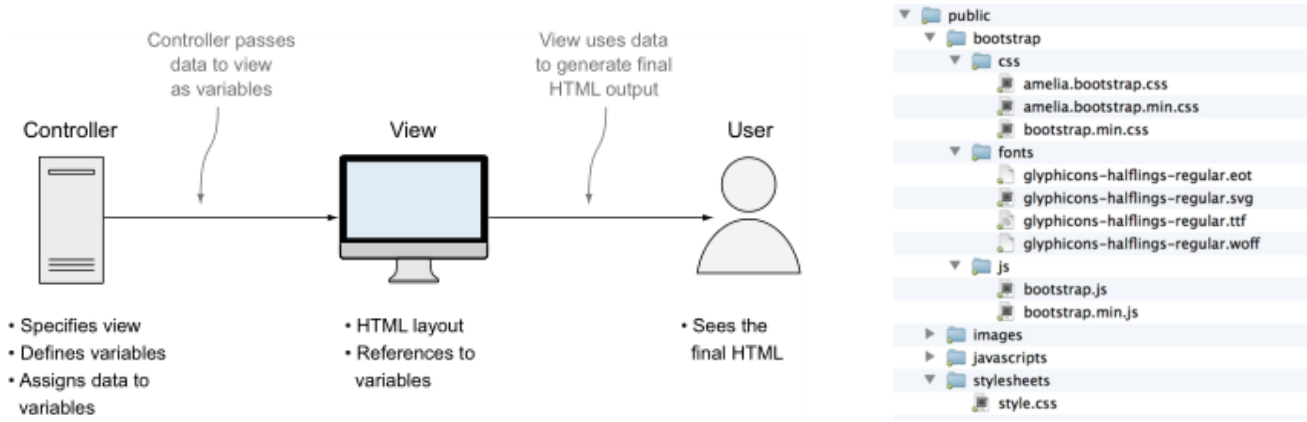
Referenciranje promenljivih u Jade templejtima

Postoje dva načina da referencirate promenljivu iz jade koda. Prvi način se naziva interpolacija i izgleda ovako:

```
h1 Welcome to #{pageHeader.title}
h1 Welcome to !{pageHeader.title}
```

Prvi red vrši uklanjanje HTML koda (escaping) pre svega iz sigurnosnih razloga. Ukoliko iz nekog razloga želite da se izvrši renderovanje HTML koda koji prosleđujete templejtu možete koristiti notaciju iz drugog reda, ali u treba napomenuti da to nije preporučljivo i da predstavlja sigurnosni propust.
Drugi način za referenciranje promenljivih je kreiranje JavaScript stringa. Primer je dat u nastavku i prvi red ponovo vrši uklanjanje HTML koda, dok ga drugi primer renderuje.

```
h1= "Welcome to " + pageHeader.title
h1!= "Welcome to " + pageHeader.title
```



Slika 9 Veza kontrolera i prezentacije i struktura aplikacije koja uključuje bootstrap

Using Bootstrap

Primer 5: Dodavanje Bootstrapa u jade fajl. Izmeniti layout.jade:

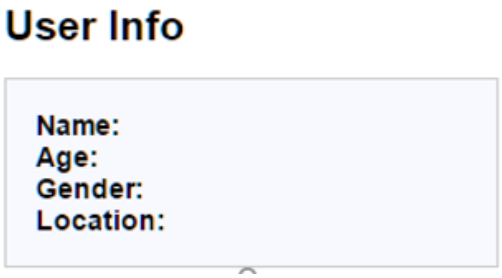
```
doctype html
html
head
     meta(name='viewport', content='width=device-width, initial-scale=1.0')
     title= title
     link(rel='stylesheet', href='https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css')
     link(rel='stylesheet', href='/stylesheets/style.css')
body
   block content
      script(src='https://code.jquery.com/jquery-3.1.1.slim.min.js')
      script(src='https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js')
```

P5: Kreirati jade ili ejs stranicu koja koristi neku Bootstrap komponentu. Potrebno je uključiti bootstrap na sličan način kao u prethodnom primeru za jade.

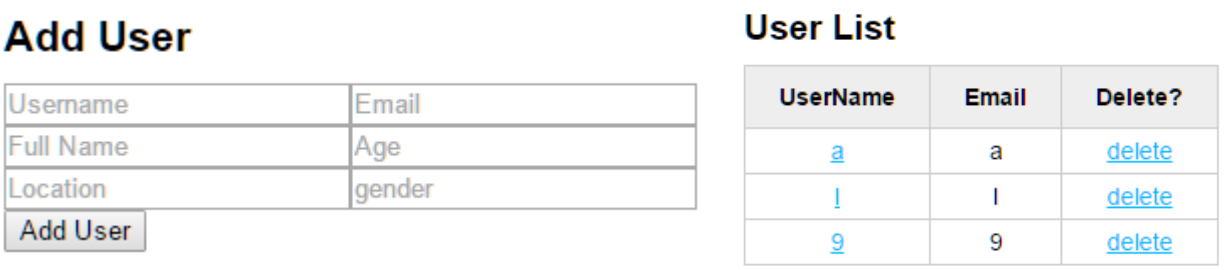
```
doctype html
html
  head
    meta(name='viewport', content='width=device-width, initial-scale=1.0')
    title= title
    link(rel='stylesheet', href='/bootstrap/css/amelia.bootstrap.css')
```

```
  link(rel='stylesheet', href='/stylesheets/style.css')
body
 .navbar.navbar-default.navbar-fixed-top
    .container
      .navbar-header
        a.navbar-brand(href='/') Loc8r
        button.navbar-toggle(type='button', data-toggle='collapse', data-target='#navbar-main')
          span.icon-bar
          span.icon-bar
          span.icon-bar
      #navbar-main.navbar-collapse.collapse
        ul.nav.navbar-nav
          li
            a(href='/about/') About

  .container
    block content

    footer
      .row
        .col-xs-12
          small &copy; Simon Holmes 2014

  script(src='/javascripts/jquery-1.11.1.min.js')
  script(src='/bootstrap/js/bootstrap.min.js')
```

P6: Napisati jade templejt index.ejs koji treba da prikaže informacije o studentu kao što je prikazano na slici ispod.



P7: Napisati templejt koji treba da prikaže formu za unos podataka o studentu kao na slici ispod.



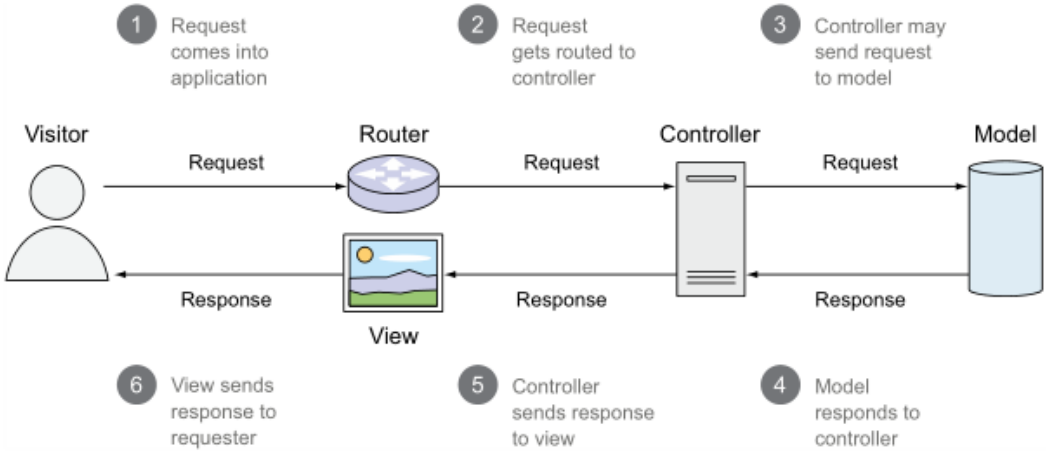
P8: Napisati ejs templejt koji treba da prikaže podatke o studentima u obliku tabele. U prvoj koloni se prikazuje username, u drugoj email a u trećoj koloni dugme delete. Podaci o studentima se učitavaju iz json fajla koji ima sledeći format:

[{"username":"a","email":"a","fullname":"a","age":"a","location":"a","gender":"a","_id":0},{"username":"l","email":"l","fullname":"l","age":"l","location":"l","gender":"l","_id":1},{"username":"9","email":"9","fullname":"9","age":"9","location":"9","gender":"9","_id":3}]

MVC – Model – View - Controler

MVC je skraćenica od Model – View – Controler i predstavlja arhitekturu koja treba da razdvoji podatke (data) od prikaza (view) i logike aplikacije (controler). Razdvajanje treba da raskine čvrste veze između ovih komponenti, što u teoriji kod treba da učini lakšim za održavanje, i ponovno korišćenje. Dodatna prednost je modularnost, i to što omogućava da različiti programeri gotovo nezavisno razvijaju određene delove sistema (jedan radi view, a drugi data).
Većina veb aplikacija se zasniva na tome da se primi zahtev, zahtev se obradi i vrati se odgovor. U MVC arhitekturi obrada jednog zahteva uključuje sledeće korake:

1. Aplikacija primi zahtev
2. Zahtev se rutira od strane kontrolera
3. Kontroler, ako je potrebno, poziva model
4. Model odgovori kontroleru
5. Kontroler prosleđuje odgovor prikazu (view)
6. Prikaz šalje odgovor onome ko je poslao prvobitni zahtev



Slika 10 - Komunikacija slojeva kod MVC aplikacije

U primerima koje smo do sada radili naša logika za rutiranje i logika kontrolera je bila pomešana što se smatra lošim stilom programiranja. Preporučuje se da se sva logika kontrolera izvuče u posebne fajlove i smesti u poseban direktorijum koji se najčešće zove controllers. Pogledajmo razliku između ova dva stila programiranja.

Primer 6: Razdvajanje logike za rutiranje i kontrolera

```
/* Rutiranje i kontroler razdvojeni. */
var homepageController = function (req, res) {
   res.render('index', { title: 'Express' });
};
router.get('/', homepageController);
```

```
/* rutiranje i kontroler zajedno*/
router.get('/', function(req, res) {
   res.render('index', { title: 'Express' });
});
```

Primer 7: Razdvajanje logike rutiranja i kontrolera
Logika za rutiranje i logika kontrolera su u različitim fajlovima. U ovom primeru mapiramo rute, logiku kontrolera učitavamo iz fajlova ../controllers/locations i ../controllers/others i zatim koristimo metode iz ovih fajlova za mapiranje ruta.

```
var express = require('express');
var router = express.Router();
var ctrlLocations = require('../controllers/locations');
var ctrlOthers = require('../controllers/others');

/* Locations pages */
router.get('/', ctrlLocations.homelist);
router.get('/location', ctrlLocations.locationInfo);
```

```
router.get('/location/review/new', ctrlLocations.addReview);
/* Other pages */
router.get('/about', ctrlOthers.about);
module.exports = router
```

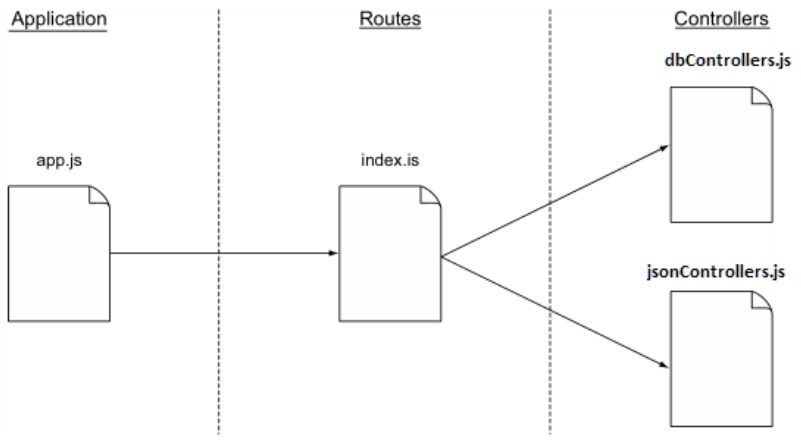
P9: Kreirati aplikaciju koja sadrži sledeće rute. Svaka kolekcija ruta bi trebalo da se nalazi u posebnom fajlu.

Collection	Screen	URL path
test	test	test/bootstrap
userJson	userlist	userJson/userlist (get metoda)
userJson	adduser	userJson/adduser (post metoda)
userJson	deleteuser/:id	userJson/deleteuser/:id (delete metoda)
userDb	userlist	userDb (get metoda)
userDb	adduser	userDb (post metoda)
userDb	deleteuser/:id	userDb (delete metoda)

Primer 8: Primer greške koju možete da dobijete prilikom menjanja arhitekture.
Prvi podebljani red nam govori kojije tip greške. U ovom slučaju očigledno da u fajl, odnosno traženi modul ili ima pogrešno ime ili je putanja do njega pogrešna. U drugom podebljanom redu vidimo i mesto gde je greška nastala.

```
module.js:340
throw err;

Error: Cannot find module '../controllers/other'
at Function.Module._resolveFilename (module.js:338:15)
at Function.Module._load (module.js:280:25)
at Module.require (module.js:364:17)
at require (module.js:380:17)
at module.exports (/Dropbox/Manning/Getting-MEAN/Code/Loc8r/BookCode/routes/index.js:2:3)
at Object.<anonymous> (/Dropbox/Manning/Getting-MEAN/Code/Loc8r/BookCode/app.js:26:20)
at Module._compile (module.js:456:26)
at Object.Module._extensions..js (module.js:474:10)
at Module.load (module.js:356:32)
at Function.Module._load (module.js:312:12)
```

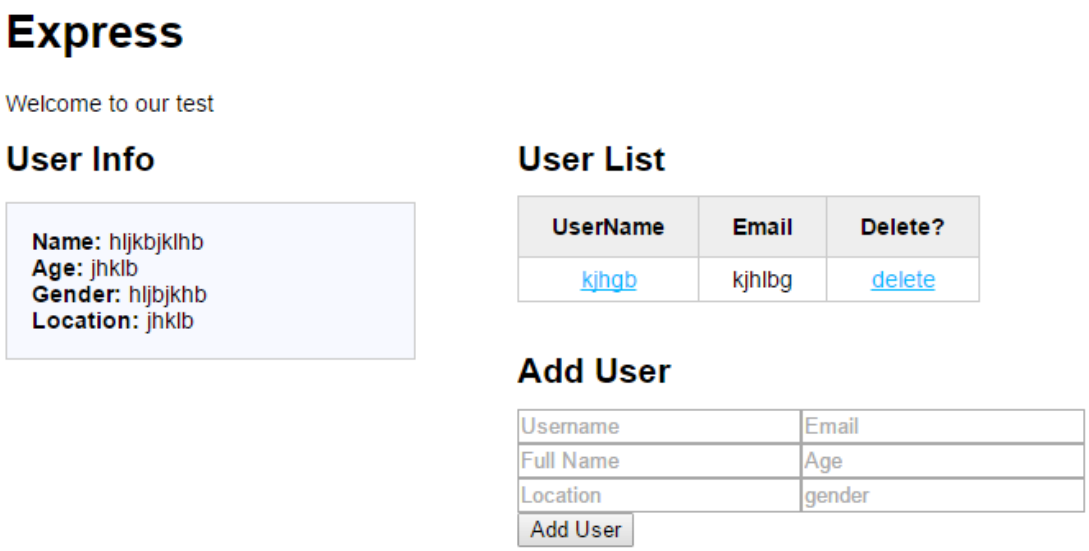
P10:Razdvojiti logiku za rutiranje i logiku kontrolera. Arhitektura aplikacije treba da izgleda kao na slici:



Slika 11 - Izgled arhitekture aplikacije za prijavu studenata

P11: U direktorijumu gde se nalazi tekst laboratorijske vežbe možete naći i primer aplikacije koja vrši upis i brisanje studenata iz nekog spiska studenata. Aplikacija je do pola završena i od vas se očekuje da je završite. Potrebno je uraditi sledeće:

1. Implementirati da ukoliko fajl data.txt ne postoji, da se kreira i da se u njega upiše prazan javascript objekat u json formatu.
2. Implementirati funkciju populateTable koja treba da popuni tabelu sa studentima podacima dobijenim sa servera. Podaci se dobijaju pozivom rute userJson/userlist (get metoda) i vraćaju u json obliku. Primer json podataka je prikazan u primeru broj x.
3. Implementirati da se klikom na tekst u koloni username prikažu i ostali podaci o studentu.
4. Implementirati poziv za dodavanje studenta u listu klikom na dugme addUser. Potrebno je pozvati odgovarajuću rutu. Serverska strana poziva je već implementirana.
5. Implementirati veb servis deleteUser u fajlu jsonControler.js koji treba da obriše studenta iz liste.



Slika 12 – Korisnički interfejs aplikacije u zadatku P11

Laboratorijska vežba broj 8: NodeJS i baze podataka

Baze podataka - šta su zašto nam trebaju?

Baze podataka su nezaobilazni elementi svih modernih aplikacija. Danas je nemoguće napraviti funkcionalnu aplikaciju a da ona ne sadrži bilo kakvu bazu. Baze podataka su nam potrebne zato što nam pružaju mogućnost trajnog skladištenja podataka. Pored toga, baze će na, uslovno rečeno, optimizovan način čuvati te podatke i omogućiti nam pristup i manipulisanje tim podacima.

SQL (Structured Query Language) baze

SQL je jezik koji se koristi za komunikaciju između ljudi i baza, odnosno za zadavanje upita bazi. Pomoću tih upita programer može definisati šemu baze (kolone, ključeve itd.) ili manipulisati podacima. U zavisnosti od DBMS-a (*Database Management System*) koji se koristi, SQL upiti mogu imati različitu sintaksu. Neke od najpoznatijih SQL baza su MySQL, Oracle SQL, Microsoft SQL, itd.

No - SQL baze

NoSQL baze ili one koje ne podržavaju SQL jezik i nisu relacione. One imaju neki drugi interfejs za komunikaciju između baze i programera koji je različit od baze do baze, odnosno od DBMS-a do DBMS-a. U kombinaciji sa NodeJS-om se najčešće koristi MongoDB, koji je ujedno i deo MEAN steka tehnologija. MongoDB podatke čuva svrstane u kolekcije, a svaka kolekcija sadrži objekte. Objekti nemaju nikakvu predefinisanu strukturu i mogu im se dodeliti bilo koji atributi.

Veb servisi

Veb servisi su servisi koje jedan računar (server) nudi drugom računaru (klijentu), a komunikacija između njih se vrši pomoću interneta, odnosno World Wide Weba. Komunikacija se najčešće vrši pomoću HTTP protokola. U aplikaciji koja utilitarizuje bazu podataka veb servis može klijentu pružiti interfejs do baze podataka, odnosno zadatak veb servisa bi bio da po nalogu klijenta izvrši neku interakciju sa bazom, bilo to čitanje podataka ili pisanje/brisanje.

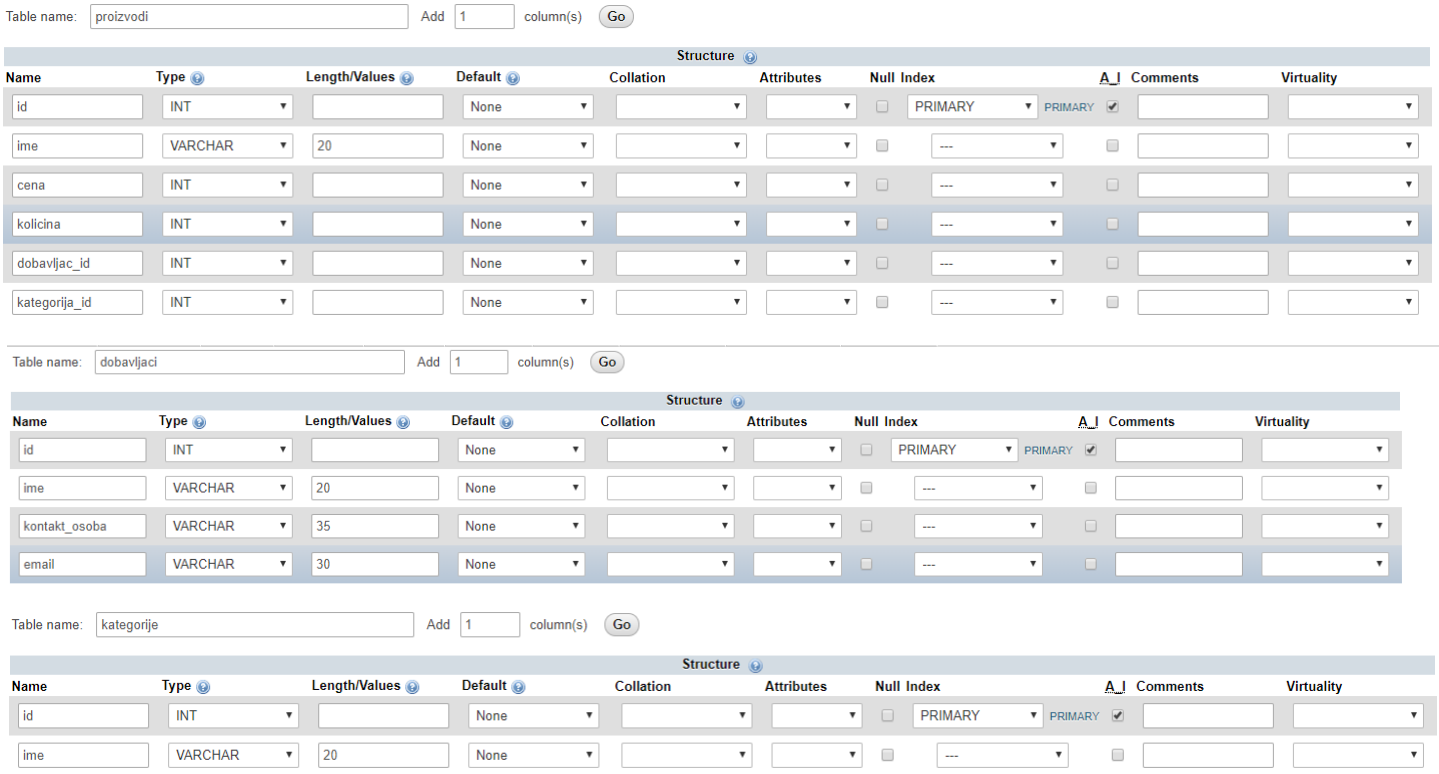
Primer 1: Pravljenje MySql baze podataka

Za korišćenje baze podataka neophodno je da je odgovarajuća baza podataka:

1. Instalirana
2. Pokrenuta – server ili odgovrajaući servis koji nam omoguća da koristimo bazu podataka
3. Da je instaliran NodeJs modul koji omogućava rad sa bazom

Za potrebe laboratorijskih vežbi preporučuje se korišćenje Wamp servera koji u sebi sadrži MySql bazu podataka, jer je on već korišćen na vežbama iz drugih predmeta. Student može koristiti i neki drugi server ili MySql bazu podataka.

Pomoću *phpMyAdmin* alata napraviti novu bazu podataka i nazvati je „prodavnica“. Unutar nje kreirati tri tabele, kao na slici ispod.



Slika 13 - Kreiranje baze podataka u alatu phpMyAdmin

Primer 2: Povezivanje NodeJS-a sa MySQL-om

NodeJS sa MySQL-om komunicira preko biblioteke *mysql*. Kreirati novi projektni direktorijum i u njemu napraviti dokument *server.js*. Pokrenuti *npm init* unutar projektnog foldera radi inicijalizacije projekta i kreiranja *package.json* dokumenta. Potom, iz komandne linije, a unutar projektnog direktorijuma, pokrenuti *npm install --save mysql,* što će instalirati *mysql* biblioteku.

```
npm install --save mysql
```

Nakon instalacije, potrebno je uvesti biblioteku u svoj projekat, odnosno u server.js dokument. To se postiže komandom require. Potom, uneti podatke za pristup bazi (adresu, korisnički nalog i lozinku) i napraviti objekat konekcije.

```
var mysql = require('mysql');
const con = mysql.createConnection({
   host: 'localhost',   user: 'root',database: 'prodavnica'
});

con.connect( function(err) {
   if (err) throw err;
   console.log('Konekcija sa bazom uspostavljena');
});
```

Metoda za povezivanje je asinhrona i prihvata *callback* funkciju kao svoj jedini parametar. *Callback* treba da prihvati samo jedan parametar koji će predstavljati objekat greške, ukoliko do nje dođe. **Bilo koja metoda koja radi sa bazom kao prvi parametar *Callback funkcije* prosleđuje objekat greške.**

Primer 3: Rad sa bazom podataka

Pozivanjem metode *connect* je izvršeno povezivanje sa bazom i sada je moguć rad sa podacima. Nad bazama podataka, odnosno podacima u njima, je moguće vršiti 4 osnovne akcije, tj. moguće je vršiti CRUD (Create, Read, Update, Delete). Pomoću SQL upita definišemo koju akciju hoćemo da izvršimo, a upit putem *query* metode objekta konekcije (u našem slučaju to je promenljiva *con*) prosleđujemo bazi.

Napomena: ovo je neophodno uraditi unutar *callback* metode kako bismo bili sigurni da je konekcija uspostavljena

Upite prosleđujemo bazi podataka pomoću metode *query* objekta konekcije. Potrebno je da to uradimo unutar *callback* metode koju smo prosledili kao parametar metode *connect* kako bismo bili sigurni da je veza uspostavljena. SQL upit možemo formirati kao običan *string*, ali je neophodno da se pridržavamo pravila SQL-a koja propisuje DBMS sa kojim radimo, što je u ovom slučaju MySQL.

```
con.connect( function(err) {
   if (err) throw err;
   let sql = "INSERT INTO dobavljaci (ime, kontakt_osoba, email)
   VALUES ('Maxi', 'Miroslav', miroslav@maxi.com'), ('Tempo', 'Milan', 'milan@tempo.com');";

   con.query( sql, function(err, result) {
      if (err) throw err;
      console.log('Kreiran je novi dobavljac sa ID-em ' + result.insertId);
   });
});
```

U prethodnom primeru je bazi poslat upit za kreiranje novog reda u tabeli *dobavljaci*. Prvi parametar koji smo prosledili metodi *query* je SQL upit koji smo hteli da izvršimo, a drugi je *callback* funkcija koja se izvršava po završetku komunikacije sa bazom.
Callback funkcija prima dva parametra, prvi predstavlja potencijalnu grešku pri izvršavanju upita, a biće null ukoliko do greške nije došlo, a drugi parametar je objekat koji sadrži rezultat upita. U slučaju *INSERT* upita nećemo dobiti nikakve podatke iz baze, a objekat *result* će sadržati sledeće vrednosti:

```
  fieldCount: broj dobijenih redova (0 u ovom slučaju)
  affectedRows: broj izmenjenih redova (2 u ovom slučaju)
  insertId: ID prvog unetog reda (1 u ovom slučaju)
```

Pored *INSERT* upita, moguće je bazi poslati bilo koji upit. Metoda *query* objekta *con* je potpuno nezavisna od upita, samo će parametar *result* koji se prosleđuje *callback* funkciji biti drugačiji.

Primer 3.1: Read (SELECT) upit

```
con.connect( function(err) {
   if (err) throw err;
   let sql = "SELECT * FROM dobavljaci;";
   con.query( sql, function(err, result) {
      if (err) throw err;
      result.forEach( function(row) {
         console.log(row);
      });
   });
});
```

Poziv prethodnog isečka koda će logovati sledeći sadržaj u konzoli:

```
C:\Users\Misa\prodavnica>node server.js
RowDataPacket { id: 1, ime: 'Maxi', kontakt_osoba: '', email: '' }
RowDataPacket { id: 2, ime: 'Tempo', kontakt_osoba: '', email: '' }
```

Primer 3.2: Update upit

```
con.connect( function(err) {
   if (err) throw err;
   let sql = "UPDATE dobavljaci SET ime = 'Beer commerce' WHERE id = 1;";
   con.query( sql, function(err, result) {
      if (err) throw err;
      console.log(result);
   });
});
```

Sadržaj konzole nakon prethodnog poziva je:

```
C:\Users\Misa\prodavnica>node server.js
OkPacket {
  fieldCount: 0,
  affectedRows: 1,
  insertId: 0
}
```

Primer 3.3: Delete upit

```
con.connect( function(err) {
   if (err) throw err;
   let sql = "DELETE FROM dobavljaci WHERE id = 1;";
   con.query( sql, function(err, result) {
      if (err) throw err;

      console.log(result);
   });
});
```

Prethodni kod će u konzolu ispisati:

```
C:\Users\Misa\prodavnica>node server.js
OkPacket { fieldCount: 0,  affectedRows: 1,  insertId: 0  }
```

Primer 4: Integracija baze podataka u ExpressJS aplikaciju

Jedna od bazičnih funkcija serverskog dela aplikacija jeste da komunicira sa bazom podataka i klijentskom delu služi kao interfejs do baze podataka. Može se reći da je backend (serverska strana) posrednik između frontenda (klijentske strane) i baze podataka.

Integracijom baze podataka u ExpressJS bismo dobili mogućnost trajnog čuvanja podataka, što je danas neophodno maltene svakom sistemu. Bazu u kombinaciji sa ExpressJS-om možemo koristiti tako što ćemo pri programiranju veb servisa uključiti i komunikaciju sa bazom. Naime, obrada klijentovih HTTP zahteva i priprema odgovora će podrazumevati neki rad sa bazom, odnosno čitanje ili pisanje nekih podataka.

Primer 4.1: GET zahtev za čitanje podataka

```
const app = require('express');
const mysql = require('mysql');
const con = mysql.createConnection({ host: 'localhost', user: 'root', database: 'prodavnica'});
app = express();
app.get('/proizvodi', function (req, res) {
   con.connect( function(err) {
      if (err) {
         res.status(500);
         return res.end(err.message);
      }
      let sql = 'SELECT id, ime FROM proizvodi';
      con.query(sql, function(err, result) {
         if (err) {
            res.status(500);
            return res.end(err.message);
         }
         res.status(200);
         result.forEach( function(row) {
            res.write(row.name + '<br>');
         })
         return res.end();
      });
   });
});
app.listen(3000);
```

U prethodnom primeru smo postavili ExpressJS aplikaciju sa jednom rutom, koja kada je pozvana učitava sve proizvode iz baze podataka i šalje ih nazad klijentu. U ovom slučaju nam nisu potrebni nikakvi podaci od klijenta, već imamo sve što nam je potrebno na serveru.

Primer 4.2: POST zahtev za upis podataka u bazu

```
const app = require('express');           const mysql = require('mysql');
const parser = require('body-parser');
const con = mysql.createConnection({
   host: 'localhost',   user: 'root',      database: 'prodavnica'
});

app = express();
app.use(parser.urlencoded);
app.post('/dodaj/proizvod', function (req, res) {
   con.connect( function(err) {
      if (err) { res.status(500);          return res.end(err.message); }

      let sql = 'INSERT INTO PROIZVODI (ime, cena, kolicina, dobavljac_id, kolicina_id)
   VALUES (';
      sql += "'" + req.body.ime + "', ";   sql += "'" + req.body.cena + "', ";
      sql += "'" + req.body.kolicina + "', ";   sql += req.body.dobavljac_id + ", ";
      sql += req.body.kolicina_id + ", ";     sql += ');';
      con.query(sql, function(err, result) {
         if (err) { res.status(500);        return res.end(err.message); }
         res.status(200);
         return res.end('Proizvod je uspešno dodat pod brojem ' + result.insertId);
      });
   });
});
app.listen(3000);
```

Veb servis iz primera iznad omogućava klijentskoj strani da pošalje POST zahtev za kreiranje novog proizvoda. Middleware koji je postavljen (body-parser) parsira POST zahtev i dodaje prosleđene vrednosti u body atribut request objekta. Nakon toga se povezujemo sa bazom, formiramo SQL upit i prosleđujemo ga bazi. Klijentu šaljemo povratnu informaciju o uspešnosti zahteva, odnosno upita.
Zahtevi za izmenu podataka koji već postoje u bazi se takođe realizuju POST metodom jer je potrebno da klijent serveru prosledi podatke koje korisnik želi da izmeni. Zahtev za brisanje redova iz baze, kao i za čitanje pojedinačnih redova, je moguće realizovati putem GET-a, gde bi se ključevi (ID) traženih redova prosledili kao parametri *query stringa*.

P1: Koja se greška javlja ukoliko pokušamo da pristupimo bazi na pogrešnoj adresi?

P2: Koja se greška javlja, odnosno koji je sadržaj uhvaćenog objekta *err*, ukoliko pokušamo da radimo sa bazom koja ne postoji?

P3: Koji je sadržaj objekta *result* koji prosleđujemo *callback* funkciji pri izvršavanju *SELECT* upita?

P4: Na koji način čovek interaguje sa No - SQL bazom podataka?

P5: Šta može biti zatadak veb servisa na serveru baze podataka?

Primer 5: Zadatak

Napraviti aplikaciju pomoću ExpressJS-a i MySQL-a koja će omogućiti manipulaciju podacima u bazi podataka i tabelama kreiranim u primeru P1. Veb servisi koje aplikacija treba da podržava su navedeni u tabeli ispod. Testirati aplikaciju uz pomoć Postman aplikacije i napraviti po nekoliko redova u svim tabelama.

Tabela 1 - Lista veb servisa za primer P5

Metoda	Ruta	Parametri	Opis
POST	dodaj/proizvod	ime, cena, kolicina, dobavljac_id, kategorija_id	Dodaje novi proizvod u bazu. Potrebno je da dobavljač i kategorija već postoje u bazi. Vraća povratnu informaciju o uspešnosti.
GET	proizvodi	/	Vraća listu proizvoda koja sadrži ID i ime svakog proizvoda.
GET	proizvod/:id	ID proizvoda	Vraća sve podatke o određenom proizvodu.
GET	proizvod/izbrisi/:id	ID proizvoda	Briše određeni proizvod i vraća povratnu informaciju o uspešnosti
POST	dodaj/kategorija	ime	Dodaje novu kategoriju. Vraća povratnu informaciju o uspešnosti.
GET	kategorije	/	Vraća listu kategorija koja sadrži ID i ime svake kategorije.
GET	kategorija/:id	ID kategorije	Vraća sve podatke o određenoj kategoriji.
GET	kategorija/izbrisi/:id	ID kategorije	Briše određenu kategoriju i vraća povratnu informaciju o uspešnosti.
POST	dodaj/dobavljac	ime, kontakt_osoba i email	Dodaje novog dobavljača. Vraća povratnu informaciju o uspešnosti.
GET	dobavljaci	/	Vraća listu dobavljača koja sadrži ID i ime svakog dobavljača.
GET	dobavljac/:id	ID dobavljača	Vraća sve podatke o određenom dobavljaču.
GET	dobavljac/izbrisi/:id	ID dobavljača	Briše određenog dobavljača i vraća povratnu informaciju o uspešnosti.

Laboratorijska vežba broj 9: Upoznavanje sa različitim protokolima

IP adresiranje i komande koje se koriste kod podešavanja mreže

Pojam IP adresa
Predstavlja 32-bitna adresu, koja je odvojena tačkama na četiri osmobitne, odnosno četiri decimalna broja. Postoje tri osnovne klase IP adresa A, B i C.

A klasa - Kod adresa A klase vrednost prvog broja može biti 1-126 i on služi za adresiranje mreže, a ostala tri broja služe za adresiranje host-ova. Postoji 126 mreža A klase od kojih u svakoj može maksimalno da se adresira 16777214 računara. 2^{24}- 2=16777214.
Na primer, ako je vrednost prvog broja 126 (time je označena mreža), slobodna su još tri broja (24 bita) za adresiranje računara, a to je 2^{24} što iznosi 16777216. Od tog broja treba još odbiti dve adrese: 126.0.0.0 (adresa same mreže) i 126.255.255.255 (adresa koja služi za emitovanje).

Napomena: 16777214 je teoretski najveći mogući broj hostova na mreži A klase. Ppretpostavka je da tako velika mreža ima podmreže, a u tom slučaju se na svakoj podmreži „gube“ po dve adrese.

Mrežna maska (subnet mask) za mrežu A klase je 255.0.0.0. Ako se mrežna maska napiše ispod IP adrese, onaj deo IP adrese koji se nalazi iznad 255-ice je fiksiran i služi za adresiranje mreže, a onaj deo koji je iznad nula služi za adresiranje računara.

Napomena: Ovo je uprošćeno objašnjenje, moguće je tzv. podmrežavanje, pri kom se 32-bitna adresa deli proizvoljno na određen broj bita, koji adresira mrežu (ne mora biti deljiv sa 8) i ostatak, koji adresira hostove. Mrežna maska u tom slučaju sadrži i brojeve koji nisu jednaki 0 ili 255.

B klasa - Prvi broj ima vrednosti između 128 i 191. Pored ovog, kod adresa B klase i drugi broj služi za adresiranje mreže, pa imamo ukupno (191-127)*256=16384 mreža na kojima se maksimalno može adresirati 2^{16} - 2=65534 hosta. Mrežna maska je 255.255.0.0.
C klasa - Prvi broj ima vrednosti 192-223. Prva tri broja služe za adresiranje mreža, a poslednji za adresiranje hostova. Imamo (223-191) * 2^{16}=2097152 mreža sa po 2^{8} - 2=254 hosta. Mrežna maska je 255.255.255.0.

Napomena: Kod prvog broja izostavljeni su broj 127 - koristi se za povratnu petlju (loopback) i brojevi 224-255, koji se koriste za multicasting (224-239) ili eksperimentalno (240-255).

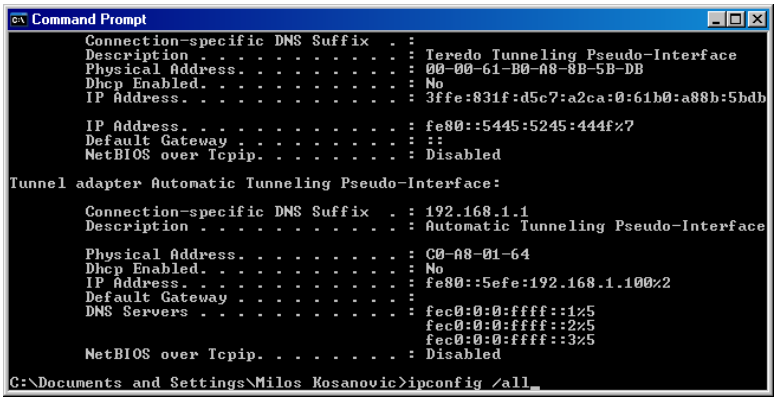
Privatne IP adrese
Računari koji se nalaze u LAN mreži moraju takođe imati IP adrese da bi međusobno komunicirali. IP adrese koje se **ne pojavljuju na Internetu** (privatne adrese), već su rezervisane za lokalne mreže i koriste se u velikom broju mreža su:

- 10.0.0.0 - 10.255.255.255
- 172.16.0.0 - 172.31.255.255
- 192.168.0.0 - 192.168.255.255

Računari u lokalnoj mreži su sakriveni iza rutera koji ima adresu dobijenu od provajdera internet usluga (može biti statička – uvek ista ili dinamička – dodeljena na neki vremenski period ili do prekida veze). Hostovi u lokalnoj mreži se ne „vide“ na internetu. Ruteri koriste NAT (network address translation) filtritanje tako da svaki računar u lokalnoj mreži dobija odgovarajuće (tražene) pakete podataka sa Interneta ili neke druge mreže sa kojom ih ruter povezuje.
Navešćemo nekoliko osnovnih komandi, koje se koriste prilikom podešavanja rada lokalne mreže. Sve komande se obično nalaze u direktorijumu c:\windows\system32, pa se moraju pokrenuti iz tog direktorijuma ili taj direktorijum mora biti dodat u path promenljiu sistema.

- **ping 127.0.0.1** – radi povratnu petlju, odnosno proverava ispravnost mrežnog adaptera.

- **ping 192.168.0.x** – proverava kakva je veza do određenog hosta
- **net view** – daje spisak imena i opisa hostova
- **nslookup** imedomena – daje adrese DNS servera, npr: nslookup www.teamnet.co.yu daje adresu njihovog DNS servera
- **ipconfig /all** – daje IP adresu, podrazumevani prolaz, adrese DNS i DHCP servera, kao i fizičku (MAC) adresu mrežnog adaptera. MAC adresa je hardverska adresa (ne možete je promeniti i jedinstvena je za svaki adapter) i može se koristiti u sigurnosne svrhe, tako što ćete na ruteru zadati spisak MAC adresa koje mogu da pristupe mreži.
- **arp** (address resolution protocol) – izlistava tabelu IP adresa i odgovarajućih MAC adresa (arp -a), dodaje statičke adrese adapterima (arp –s, dodela je „trajna"-važi do restarta) ili ih briše iz tabele (arp -d)
- **tracert** (trace route) – prati rutu do odgovarajućeg servera. Može da posluži za otkrivanje uskih grla, jer daje vreme koje je potrebno za „prolaz" kroz pojedinačne servere.
- **net use x: \\imeracunara\imedirektorijuma$** - Taj direktorijum će biti mapiran kao x: . U Computer Managementu može se videti spisak deljenih direktorijuma. Oni koji imaju dolarski znak nisu vidljivi, ali im administratori mogu pristupiti sa mreže pomoću ove komande (da bi se ovako mapirao drajv nekog hosta u mreži morate znati korisničko ime i lozinku za taj računar).
- **msg [username]** ***tekst poruke*** – šalje tekstualnu poruku svim računarima u radnoj grupi (ako se umesto * otkuca ime nekog računara, onda se samo njemu šalje poruka). Da bi ova komanda funkcionisala potrebno je da bude pokrenut Messenger servis. Komanda ***net start*** daje spisak svih pokrenutih servisa. Ako među njima nije Messenger, možete ga startovati pomoću net start messenger u komandnoj liniji ili preko *Administrative Tools > Services*. Ukoliko želite da pošaljete poruku kolegi iz grupe potrebno je da poruku pošaljete računaru (serveru) na koji je on logovan komandom msg * /SERVER:[ipadresa]



Slika 14 – Prikaz podataka koje prikazuje naredba ipconfig /all

1. Da bi mogli da se upoznamo sa nekim standardnim komandama TCP/IP protokola potrebno je da izaberite MS-DOS operativni sistem na sledeći način: **Start – All Programs – Acessoriess - Command Promt .** Zadajte komandu ***ipconfig /all.*** Dobićete ekran sa slike broj 1.

P1. Koje informacije nam daje ova komanda ?

P2. Koja je vaša MAC adresa, IP adresa, a koja je IP adresa mreže kojoj pripada vaš računar ?

P3. Kojom naredbom možemo da oslobodimo sve IP konekcije na našem računaru ?

P4. Kojom naredbom možemo da obnovimo sve IP konekcije na našem računaru ?

2. Detaljno se upoznati sa svim opcijama koje na raspolaganju ima komanda ***ping*.** Nakon toga zadati komandu **ping 127.0.0.1 , ping xxx.xxx.xxx (**xxx označava adresu vaše stanice) i na kraju **ping yyy.yyy.yyy.yyy** (yyy označava adresu neke od susednih stanica koje se nalaze na lokalnoj mreži).

P5. Objasniti zašto služi komanda ping ?

P6. Opisati šta ste dobili u sva tri slučaja i objasniti zašto je to tako ?

3. Zadajte komande ***arp, netstat,* tracert** i upoznati se sa funkcijama svake od ovih naredbi**.**

P7. Opisati pojedinačno šta svaki od ovih programa radi i koje nam informacije daje ?

4. Pokrenite komandu ***tracert*** i upoznati se sa mogućnostima kojima raspolaže ova naredba.

P8. Navesti sve IP adrese uređaja preko kojih ide veza do lokacije školskog DNS servera?

5. Pokrenite ponovo komandu ***tracert*** ali sada zadajte adresu **www.vtsnis.edu.rs.**

P9. Navesti šta se sada dogodilo ?

6. Proverite naredbom ***ping*** da li postoji veza do sajta **www.vtsnis.edu.rs.**

P10. Navesti IP adresu sajta www.vtsnis.edu.rs i kojoj klasi adresa on pripada.

P11. Testirati mogućnosti naredbe msg tako što ćete poslati poruku samom sebi i jednom kolegi iz grupe. Da li se razlikuje server koji šalje poruke u prvom i drugom slučaju?

Upoznavanje sa TCP/IP servisima: TELNET, FTP i drugi

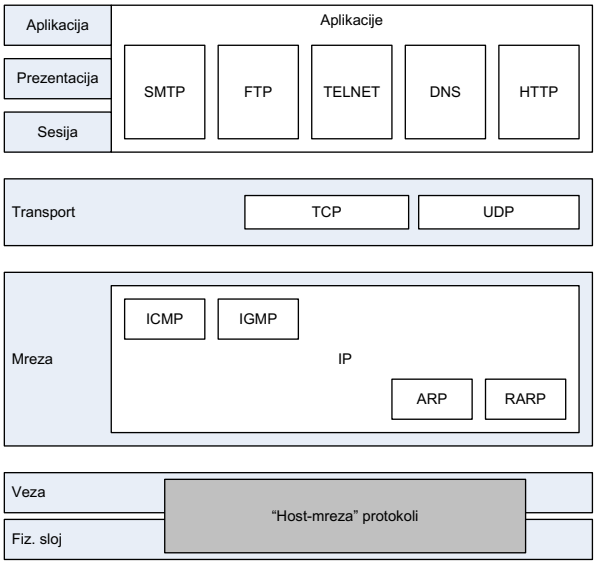
Postoji nekoliko standardnih servisa na aplikacionom nivou koji se koriste za uspostavljanje klijent-server veze preko TCP/IP protokola. To su:

- TELNET - obezbedjuje udaljeni pristup nekom računaru (*remote login)* koristeći TCP/IP
- FTP (*File Transfer Protocol*) - vrši prenos fajlova izmedju dva računara
- FINGER – prikazuje informacije o lokalnim ili mrežnim korisnicima
- MIME (*Multipurpose Internet Mail Extension*) - poboljšane mogućnosti elektronske pošte uz pomoć TCP/IP
- SMTP (*Simple Mail Management Protocol*) - pruža jednostavne usluge kod elektronske pošte
- DNS (*Domain Name Service)* - preslikava imena hostova u mrežne adrese
- DHCP (*Dynamic Host Configuration Protocol*) - je protokol za dinamičko konfigurisanje računara
- HTTP (*Hypertext Transfer Protocol*) - protokol koji se koristi kod WWW (*World Wide Web*) a može se koristiti kod *client-server* aplikacija za prenos hipertekstova.

Važan koncept TCP/IP komunikacija predstavlja korišćenje *port*-ova i *socket*-a. *Port* identifikuje tip procesa (kakvi su FTP (21), TELNET(23), FINGER(79), SMTP(25) i dr), dok *socket* identifikuje dodelu jedinstvenog broja veze (*connection number*). Na ovaj način TCP/IP može istovremeno da podržava rad većeg broja veza koje se izvršavaju preko mreže a pripadaju različitim aplikacijama. TCP/IP model ne predviđa prezentacioni i sloj sesije, već su funkcije ovih slojeva pripojene aplikacionom sloju. To znači da aplikacije moraju samostalno da realizuju funkcije koje se odnose na sesiju i prezentaciju podataka, ako su im takve funkcije uopšte potrebne. Aplikacioni sloj sadrži veći broj protokola visokog nivoa. Prvobitno su razvijeni protokoli: TELNET (vituelni terminal), FTP (*File Transfer Protocol*) - protokol za prenos fajlova i SMTP (*Simple Mail Transfer Protocol*) - protokol za prenos elektronske pošte. Vremenom, aplikcioni sloj je proširen brojnim protokolima, od kojih su najznačajniji: DNS (*Domain Name System*) - za preslikavanje imena hostova u njihove mrežne adrese i HTTP - za pribavljanje strana na Web-u. Na sl.broj 1 prikazan uporedni prikaz oba modela model

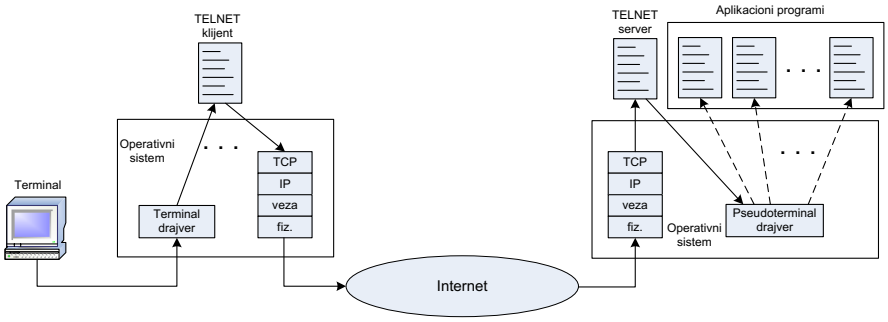
TELNET

Glavni zadatak Interneta i TCP/IP protokol steka je da obezbede mrežne servise korisnicima. Na primer, korisnici bi želeli da mogu da izvršavaju različite aplikacione programe na udaljenim računarima, a da kreirane rezulte potom prenesu na svoj lokalni računar. Jedan način kako se može ostvariti ovaj zahtev jeste da se kreira poseban klijent-server aplikacioni program za svaki od potrebnih servisa. Programi kao što su FTP, WEB, E-mail i td. su već široko dostupni, primeri su ovakvog pristupa. Međutim, bilo bi nemoguće napisati aplikacioni program za svaku iskazanu potrebu. Bolje rešenje je klijent-server program opšte namene, koji će omogućiti korisniku da pristupi bilo kom aplikacionom programu na udaljenom računaru. Drugim rečima, da omogući korisniku da se prijavi za rad (*login*-uje) na udaljenom računaru. Nakon prijavljivanja, korisnik može da koristi servise dostupne na udaljenom računaru i prenese rezultate nazad na svoj računar. TELNET je upravo takav klijent-server program. TELENT omogućava uspostavljanje konekcije sa udaljenim sistemom na način da se lokalni terminal ponaša kao da je terminal tog udaljenog sistema. (Terminal je: monitor + tastatura).



Slika 15 – Prikaz različitih protokola na OSI nivoima

Lokalni login: Kada se korisnik prijavi za rad na lokalnom više-korisničkom sistemu (putem korisničkog imena i lozinke) kaže se da je izvršio **lokalni login.** Kako korisnik kuca na tastaturi terminala, svaki pritisak na dirku se prenosi *drajveru terminala.* Drajver terminala prenosi karaktere (znaci koji odgovaraju dirkama) operativnom sistemu. Operativni sistem, interpretira kombinacju karaktera i poziva odgovarajući aplikacioni program (slika br.2-a). Takođe, operativni sistem prepoznaje kombanicije karaketera koje imaju specijalno značenje i preduzima odgovarajuće akcije, tj. komande (npr. Ctrl+Z za suspenziju ili Ctrl+C za trenutni prekid rada aplikacije).



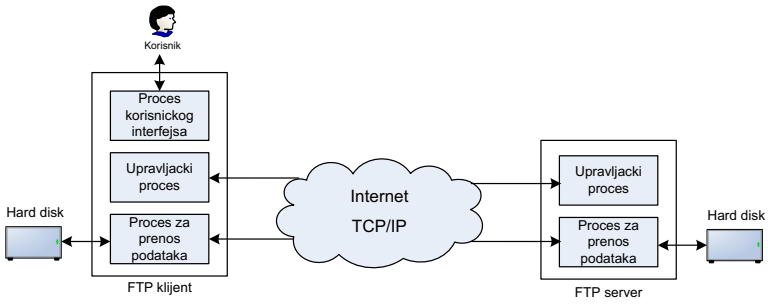
Slika 16 – Udaljen i lokalni login

Udaljeni login: Kada korisnik pristupa aplikacionom programu lociranom na udaljenom računaru, on obavlja **udaljeni login (***remote login*). TELNET je posrednik u ovoj interakcji. Klijentska strana TELNET aplikacije izvršava se na strani korisnika, a serverska na strani udaljenenog računara. Pritisak na dirku tastature lokalnog terminala prenosi se terminal drajveru, od koga operativni sistem preuzima karaktere, ali ih ne interpretira, već ih šalje TELNET klijentu.

TELNET klijent prevodi karaktere (koji u zavisnosti od tipa loaklnog operativnog sistema mogu biti kodirani ASCII, EBCDIC ili nekim trećim kodom) u univerzalni karakter kôd (NVT - *Network Virtual Terminal chracters*) i isporučuje ih lokalnom TCP/IP steku. Komande ili tekst, u NVT obliku, prenosi se kroz Internet i stiže do TCP/IP steka udaljenog računara, koji ih isporučuje operativnom sistemu, a on TELNET serveru, gde se karakteri konvertuju iz NVT formata u oblik razumljiv udaljenom računaru. Sada bi smo očekivali da TELNET server vrati konvertovane karktere operativnom sistemu koji bi onda obavio interakciju sa odgovarajućom aplikacijom. Međutim, to se ne dešava jer operativni sistem nije projektovan da karkatere dobija od TELNT servera već od lokalnog terminala (posredstvom terminal drajvera). Rešenje je u ugradnji specifičnog programa, *pseudoterminal drajver*, koji se prema operativnom sistemu ponaša kao terminal drajver, ali zato sa druge strane, karaktere ne očekuje od lokalnog terminala već od TELNET servera. Konačno, operativni sistem prenosi karaktere do odgovarajućeg aplikacionog programa.

FTP

FTP (*File Transfer Protocol – Protokol za prenos fajlova*) je aplikacioni protokol za kopiranje fajlova od jednog na neki drugi host. Prenos fajlova između hostova je jedan od najčešćih zadataka koji se očekuje od bilo kog mrežnog okruženja. Iako se čini da se fajlovi mogu jednostavno preneti sa jednog na drugi sistem,postoje problemi koji se moraju rešiti kako bi prenos uopšte bio moguć. Na primer, dva sistema mogu koristiti različite konvencije za imenovanje fajlova, načine za predstavljanje teksta i podataka, ili podržavati raziičite strukture direktorijuma. Na FTP-u je da reši ove probleme.



Slika 17 – FTP protokol

FTP se razlikuje od drugih klijent-server aplikacija po tome što uspostavlja dve konekcije između hostova. Jedna konekcija se koristi za prenos podataka, a druga za prenos upravljačkih informacija (komande i odzivi). Osnovni model FTP-a prikazan je na slici br.3. FTP klijent sadrži tri komponente: korisnički interfejs, proces za upravljanje klijentom i proces prenos podataka. Server ima dve komponente: proces za upravljanje

serverom i proces za prenos podataka. Upravljačka konekcija se uspostavlja između upravljačkih, a konekcija za prenos podataka između procesa zaduženih za prenos podataka na stranama klijenta i servera. Upravljačka konekcija ostaje aktivna za sve vreme trajanja jedne FTP sesije. Konekcija za prenos podataka se otvara, a onda i zatvara za prenos svakog pojedinačnog fajla. FTP koristi TCP upravljačku konekciju koja se ostvaruje preko porta 20, a konekcija za prenos podataka preko porta 21.

Kod većine operativnih sistema, FTP klijent je dostupan kao program koji se poziva iz komandne linije, npr: `C:\WINDOWS>ftp.` Po pokretanju programa, startuje se korisnički interfejs koji koristi komandnu liniju: `ftp>` .U nastavku gornje linije program očekuje da korisnik unese komandu. FTP podržava više komandi, od kojih su najznačajnije *open, bye, get, put* i td. Komanda *open*, koja se obično prva koristi, služi za otvaranje sesije sa udaljenim računarom. Npr. `ftp>open ftp.elfak.ni.ac.rs` otvara sesiju sa FTP serverom Elektronskog fakulteta. Uočimo da je navedno ime domena, a ne IP adresa servera. To znači FTP klijent koristi usluge DNS-a za pronalaženje IP adrese. Ukoliko je kontakt ostvaren, biće zatraženo korisničko ime i lozinka:

```
ftp> open ftp.elfak.ni.ac.yu
Connected to black.elfak.ni.ac.yu.
220 black Microsoft FTP Service (Version 5.0).
User (black.elfak.ni.ac.yu☹none)): anonymous
331 Anonymous access allowed, send identity (e-mail name) as password.
Password:
230-Elektronski fakultet
230-Univerzitet u Nisu
230-Racunski centar
230 Anonymous user logged in.
ftp>
```

U ovom trenutku sesija sa FTP serverom je otvorena. To znači da je upostavljena upravljačka konekcija (preko TPC porta 20), koja će trajati za sve vreme trajanja sesije i koristiće se za prenos komandi i odziva. Na primer, korisnik može da pregleda sadržaj direktorijum na strani FTP servera komandom *dir*, da izabere fajl koji želi da uzme i komandom `get ime_fajla` ostvari prenos izabranog fajla na svoj računar. Za prenos fajla otvara se konekcija na portu 20 i zatvara kada je prenos fajla završen. Postoji i mogućnost da korisnik prenese fajl sa svog računara na FTP server, za šta koristi komanda *put*. U toku iste sesije, korisnik može da izvrši više komandi (kao što su *get*, *dir* i *put*). Nakon što je preneo sve fajlove koji su mu potrebni, korisnik zatvara sesiju komandom *bye*.

Za rad sa FTP-om danas postoje brojni programi sa grafičkim korisničkim interfejsom (nalik Windows Explorer-u) koji značajno olakšavaju rad korisniku (ne mora da zna FTP komande).
Napomena: Ukoliko vaš DOS prompt ne prepoznaje naredbe Telnet i FTP moguće je da su u novijim verzijama Windowsa one isključene. Potrebno je otići u *Control Panel*, zatim u *Programs and Features*, zatim kliknuti na *Windows features on or off*. Iz liste izabrati odgovarajuće komponente windowsa koje treba instalirati kao što je na primer *Telnet Client*.

ZADATAK:
1. Izabrati DOS promt i upoznati se sa servisom **TELNET.** Kada se startuje **Telnet** naredbom **help** ili **/?** možete videti njene mogućnosti. Iskoristite ovaj servis za konektovanje na računar vašeg kolege.

P11. Objasniti šta nam omogućava ovaj servis i navesti njegove komande?

3. Skinuti program putty sa interneta i konektovati se na kompjuter vašeg kolege koristeći ovaj program.

P12. Telnet servis se danas retko koristi. Zašto? Koji servis se koristi umesto njega? Navedite neku grafičku aplikaciju koju možete iskoristiti za konektovanje na udaljeni sistem, a koja koristi Telnet ili neki drugi protokol.

2. Izabrati DOS promt i pokrenuti servis **FTP.** Kada se startuje **FTP** naredbom **help** ili **?** pregledati mogućnosti koje vam stoje na raspolaganju.

P13. Navesti kojim komandama raspolaže FTP servis u windows klijentu i objasniti ih.

3. Izabrati neki FTP server i konektovati se njega (pitati asistenta za primer servera). Pregledati raspoložive direktorijume i fajlove na izabranom serveru i pokušajte da skinite neke fajlove. Kreirajte txt fajl indeks_ime_prezime.txt i stavite ga na server.

P14. Kako možemo da skinemo neke fajlove sa ili da pošaljemo naše fajlove na izabrani FTP server?

P15. U kom direktorijumu će se smestiti fajlovi koje ste preuzeli sa FTP servera. Šta radi komanda lcd?

P16. Kako se prekida konekcija? Kako se izlazi iz FTP mode u command promptu?

4. Upoznajte se sa servisom **FINGER** i pokrenite isti pozivajući neki od računara u vašoj lokalnoj mreži.

P17. Navesti koje parametre ste dobili nakon što ste pokrenuli servis FINGER.

P18. Šta je XMPP, ko ga koristi i gde?

Laboratorijska vežba broj 10: Upoznavanje sa Npm paket menadzerom i testiranje aplikacije

- Tutorial: npm install -g how-to-npm
- Prirpema aplikacije za instaliranje
- Instaliacija modula
- Postavljanje aplikacije na server
- Testiranje aplikacije

Testiranje aplikacije

Vrste testiranja

1. Unit **Testing**,
2. Integration **Testing**,
3. Regression **Testing**,
4. Testiranje ruta – **API test**

Kreiranje strukture aplikacije:

Kreirati prazan direktorijum na željenoj lokaciji.

Pozicionirati se u novo-kreirani direktorijum u terminalu.

Startovati komandu **npm init.** Ova komanda će pokrenuti wizard za podešavanje konfiguracionog fajla (package.json) i samog projekta. Ukoliko ne želite da unosite podatke u konfiguracioni fajl ručno, prethodnoj komandi dodajte **--yes** i konfiguracioni fajl će se sam kreirati sa podrazumevanim vrednostima.

Nakon što se komanda izvrši, otvorite direktorijum u željenom editoru.

Instalacija paketa za testiranje aplikacije:

Sledeći u nizu je framework za unit testiranje. U ovoj aplikaciji korišćen je framework **Mocha** i kao dodatak **Chai.** U terminalu izvršiti komande:

```
npm install mocha --save
npm install chai --save.
```

Opcija **--save** se koristi kako bi se automatski sačuvale ove biblioteke u našem **package.json** fajlu.
Na kraju instalirati **Express** framework po uzoru na uputstvo iz vežbe broj 6.

Ovim smo završili sa struktuiranjem aplikacije, međutim ostaje da podesimo **test** komandu u **package.json** fajlu, kako bi lakše pokrenuli sami testove, korisćenjem komande **npm test**.

Podešavanje **test** skripte u fajlu package.json:

```
"scripts": {
    "test": "./node_modules/.bin/mocha --reporter spec"
}
```

Opis aplikacije:

Kreirati direktorijum **app** i u njemu fajl **converter.js.** U njemu kreiramo funkciju rgbToHex koja prihvata 3 argumenta i vraća string koji predstavlja boju u heksadekadnoj notaciji. Pomoćna funkcija **pad**, nam omogucava da uvek imamo "dvocifren" rezultat, npr. A -> 0A.

Kreiranje funkcije rgbToHex

```
exports.rgbToHex = function (red, green, blue) {
    var redHex = red.toString(16);
    var greenHex = green.toString(16);
    var blueHex = blue.toString(16);
    return pad(redHex) + pad(greenHex) + pad(blueHex);

};

function pad(hex) {
    return (hex.length === 1 ? "0" + hex : hex);
}
```

Funkcija koja boju konvertuje iz heksadecimlane notacije u RGB format

```
exports.hexToRgb = function (hex) {
    var red = parseInt(hex.substring(0, 2), 16);
    var green = parseInt(hex.substring(2, 4), 16);
    var blue = parseInt(hex.substring(4, 6), 16);

    return [red, green, blue];
};
```

Kreiranje ruta (veb API)
Za kreiranje dve rute "/rgbToHex" i "/hexToRgb" u fajlu server.js u app direktorijumu koristićemo Express framework:

```
var express = require("express");
var app = express();
var converter = require("./converter");

app.get("/rgbToHex", function (req, res) {
    var red = parseInt(req.query.red, 10);
    var green = parseInt(req.query.green, 10);
    var blue = parseInt(req.query.blue, 10);

    var hex = converter.rgbToHex(red, green, blue);
    res.send(hex);
});

app.get("/hexToRgb", function (req, res) {
    var hex = req.query.hex;
    var rgb = converter.hexToRgb(hex);
    res.send(JSON.stringify(rgb));
});

app.listen(3000);
```

Kreiranje Unit testova

Mocha nam daje mogućnost da **opišemo** šta želimo da implementiramo. Funkcija se simbolično zove **describe.** Kako joj i samo ime kaže, prvi parameter funkcije je string koji **opisuje** test, dok je drugi parametar funkcija koje predstavlja telo opisa. Ilustrujmo to primerom:

```
describe("Color Code Converter", function () {

});
```

Ukoliko je potrebno, svaka pojedinačna funkcija **describe** može sadržati više drugih za što detaljnije opisivanje. Nakon opisivanja u širem smislu, implementiraćemo **it** funkciju, gde ćemo dati konkretan opis testa.
It funkcija je vrlo slična **describe** funkciji, sa tom razlikom što se u telo **it** funkcije može staviti *samo* **očekivanje testa (*eng Assertion*),** odnosno ono što očekujemo da bude rezultat izvršenja funkcije koju testiramo.

```
describe("Color Code Converter", function () {
   describe("RGB to Hex conversion", function () {
         it("converts the basic colors", function () {

         });
    });
});
```

Sada možemo uvesti i prvo ***očekivanje.*** Frejmvork Mocha podržava različite biblioteke za pisanje testova kao što su should.js, expect.js itd. Mi ćemo koristiti **Chai** biblioteku i njen metod **expect** da uporedimo rezultat našeg modula i rezultata onoga sto ***očekujemo*** da dobijemo. U prvoj **it** funkciji, testiraćemo rgbToHex metodu. Kako joj i samo ime kaže, ona konvertuje boju iz RGB formata u heksadecimalni zapis. Kreirati poddirektorijum **test** i u njemu kreirati fajl **converter.js** koji će sadržati sledeće:

Primer 1: Unit test za funkciju rgbToHex

```
var expect = require("chai").expect;
var converter = require("../app/converter");

describe("Color Code Converter", function () {
   describe("RGB to Hex conversion", function () {
         it("converts the basic colors", function () {
             var redHex = converter.rgbToHex(255, 0, 0);
             var greenHex = converter.rgbToHex(0, 255, 0);
             var blueHex = converter.rgbToHex(0, 0, 255);

             expect(redHex).to.equal("ff0000");
             expect(greenHex).to.equal("00ff00");
             expect(blueHex).to.equal("0000ff");
         });
    });
});
```

Primetite **.to.(deep).equal** deo koda. Ovo se zove **matcher** ili izjednačavač i on spaja rezultate funkcije sa **očekivanim** rezultatima.

P1: Napisati unit test za funkciju hexToRgb

Sada možemo testirati aplikaciju. U terminalu pokrenite komandu: **npm test** i trebalo bi da dobijete sledeću poruku:

```
Color Code Converter
  RGB to Hex conversion
    ✓ converts the basic colors
  Hex to RGB conversion
    ✓ converts the basic colors

2 passing (8ms)
```

Testiranje ruta – API test

Sada kad su testovi uspešni, napravićemo pristup našem konverteru preko **HTTP API-a**, i pokazati pisanje testova za asinhroni kod koristeći frejmvork **Mocha**. U direktorijum **test** dodajte fajl **server.js**. U promenljivu **url** smestićemo celu putanju do resursa koji želimo da testiramo. U ovom slučaju to je **GET request** sa parametrima red, green i blue 255, 255, 255 respektivno.

Primer 2: API test za rutu /rgbToHex

```
describe("Color Code Converter API", function () {

    describe("RGB to Hex conversion", function () {

        var url = "http://localhost:3000/rgbToHex?red=255&green=255&blue=255";

        it("returns status 200", function (done) {
            request(url, function (error, response, body) {
                expect(response.statusCode).to.equal(200);
                done();
            });
        });

        it("returns the color in hex", function (done) {
            request(url, function (error, response, body) {
                expect(body).to.equal("ffffff");
                done();
            });
        });

    });
```

P2: Napisati API test za rutu hexToRgb:

```
    describe("Hex to RGB conversion", function () {
        var url = "http://localhost:3000/hexToRgb?hex=00ff00";

        // TO DO









});
```

Na kraju pokrećemo **server** komandom **node app/server.js**. **Testiranje** se pokreće komandom **npm test** u zasebnom terminalu ili shell sesiji. Ukoliko ste zadatak uspešno uradili trebalo bi da dobijete sledeće rezultate:

```
Color Code Converter
  RGB to Hex conversion
    ✓ converts the basic colors
  Hex to RGB conversion
    ✓ converts the basic colors

Color Code Converter API
  RGB to Hex conversion
    ✓ returns status 200
    ✓ returns the color in hex
  Hex to RGB conversion
    ✓ returns status 200
    ✓ returns the color in RGB

6 passing (50ms)
```

Primer 3: TODO Merenje trajanja izvršavanja metoda

```
describe('something slow', function() {
  this.slow(300000); // five minutes

  it('should take long enough for me to go make a sandwich', function() {


    // ...
  });
});
```

Primer 4: TODO Postavljanje timeouta na test:

```
describe('a suite of tests', function() {
  this.timeout(500);

  it('should take less than 500ms', function(done) {
    // TO DO
  });

  it('should take less than 500ms as well', function(done) {
    // TO DO
  });
});
```

P3:

P4:

P5:

Laboratorijska vežba broj 11: Pakovanje i postavljanje aplikacije

Pakovanje aplikacije

PKG interface komandna linija omogućava Vam da svoj Node.js projekat upakujete u izvršnu datoteku koja se može pokrenuti čak i na uređajima bez instaliranog Node.js –a.

Potrebno je pokrenuti komandu:

```
npm install -g pkg
```

PKG može generisati izvršne fajlove za nekoliko ciljnih mašina istovremeno. Možete odrediti listu ciljeva odvojenih zarezom pomoću opcije **`–targets.`**

Tokom procesa pakovanja PKG analizira izvorni kod, detektuje pozive koji su vam potrebni, prelazi zavisnost vašeg projekta i uključuje ih u izvršnu datoteku. U većini slučajeva ne morate ručno da navodite nista. Međutim vaš kod možda zahteva (variable) pozive (takozvane non-literal argumente) ili korišćenje non-javascript fajlove (views, css, images...).

```
require('./build/' + cmd + '.js')
path.join(__dirname, 'views/' + viewName)
```

Ovakvi slučajevi se ne rešavaju sa pkg. Dakle morate specificirati datoteke ručno u svojstvu pkg datoteke package.json fajlu.

```
"pkg": {
    "scripts": "build/**/*.js",
    "assets": "views/**/*"
}
```

Opcije
Node.js aplikacija se može pozvati sa runtime opcijama. Da biste ih popisali, otkucajte **`--help`**

```
-h, --help       output usage information
  -v, --version    output pkg version
  -t, --targets    comma-separated list of targets (see examples)
  -c, --config     package.json or any json file with top-level config
  --options        bake v8 options into executable to run with them on
  -o, --output     output file name or template for several files
  --out-path       path to save output one or more executables
  -d, --debug      show more information during packaging process [off]
  -b, --build      don't download prebuilt base binaries, build them
  --public         speed up and disclose the sources of top-level project

  Examples:
  - Makes executables for Linux, macOS and Windows
    $ pkg index.js
  - Takes package.json from cwd and follows 'bin' entry
    $ pkg .
  - Makes executable for particular target machine
    $ pkg -t node6-alpine-x64 index.js
  - Makes executables for target machines of your choice
    $ pkg -t node4-linux,node6-linux,node6-win index.js
  - Bakes '--expose-gc' into executable
    $ pkg --options expose-gc index.js
```

PKG ima takozvane „binarne datoteke“ one su zapravo isti izvršni čvorovi ali sa nekim zakrpama. Koriste se kao osnova za svaki izvršni pkg. Pkg preuzima prekompajlirane osnovne binarne datoteke pre pakovanja vaše aplikacije. Ako želite da kompajlirate osnovne binarne datoteke iz izvora umesto da ih preuzimate možete da prosledite `--build` opciju. Prvo se uverite da vaš računar zadovoljava zahteve kompajliranja originalnog Node.jsČ BUILDING.md .

Postavljanje aplikacije

Za postavljanje aplikacije na server, neophodan nam je pristup samom serveru. Ukoliko to nije slučaj imamo nekoliko opcija. Iznajmiti hosting na platformama koje su kompatibilne sa Node.js-om i NPM-om kao što su : appfog, Azure, Clever Cloud, AWS i drugi.
Druga opcija je FTP pristup putem nekog od softvera za udaljeni pristup serveru kao što je FileZilla, gde je neophodno uneti parametre kao što su **host, username, password** i **port.**
Potrebno je na server poslati fajlove naše aplikacije, instalirirati biblioteke i package.json fajl i pokrenuti aplikaciju na već poznati način.

Instaliranje aplikacije

Primer aplikacije opisane u ovoj vežbi možete naći na adresi **https://github.com/tonicfilip/KSS.git.**

Instalacija aplikacije se može izvesti na **dva načina.**
Klikom na dugme “clone or download” dobijaao opcije da downloadujemo ZIP arhivu li korišćenjem gita skinemo repositorijum.

1. Neophodno je **zip** fajl raspakovati na željenu lokaciju na disku, otvoriti u željenom editor, I u terminal izvršiti komandu **npm install** . Pokrenuće se instalacija svih biblioteka iz **pacage.json** fajla. Nakon što se instalacija završšila, možemo pokrenuti aplikaciju komandom **npm start.**

2. U terminalu uneti komandu **git clone** a zatim iskopirati link repositorijuma. Pokrenuće se download aplikacije I formirace se folder KSS na lokaciji sa koje je pokrenut terminal. Nakon što se ovaj proces zvaršio, izvršiti u terminal komandu **npm install** a zatim **npm start.**

P6: Kreirati modul niz koji sadrži dve funkcije: min i max. Obe funkcije prihvataju niz kao parametar i vraćaju najveći tj najmanji element niza, respektivno. Koristeći biblioteke Moca i Chai, napisati unit testove za obe ove aplikacije, takođe predvideti unošenje praznog niza kao parametra testiranih funkcija.

```
exports.max = (niz) => {
    if (niz.length === 0) {
        throw ('Prazan niz');
    }

    var _max = niz[0];
    for (var i = 1; i < niz.length; i++) {
        if (niz[i] > _max) {
            _max = niz[i];
        }
    }
    return _max;
}

exports.min = (niz) => {
    if (niz.length === 0) {
        throw ('Prazan niz');
    }

    var _min = niz[0];
    for (var i = 1; i < niz.length; i++) {
        if (niz[i] > _min) {
```

```
            _min = niz[i];
        }
    }
    return _min;
}

describe('Klasa za rad sa nizovima', function () {
    describe('Funkcija max trazi maksimalni element niza', function () {
        it('treba da vrati maksimalni element niza', function () {
            var _niz = [1, 11, 0, 9, 14];
          //TO DO

        });
    });
});
```

P7: Kreirati modul koji sadrži funkciju counter . Funkcija prihvata dva parametra,rečenicu i karakter. Funkcija kao rezultat treba da vrati da li rečenica koja je uneta sadrži zadati karakter kao i koliko puta se zadati karakter ponavlja. Napisati testove koji proveravaju da li u rečenici postoji karakter „a“ , takože treba proveriti da li je unešena poruka prazna.

```
exports.counter = (text, letter) => {









};

-------------------------------------------------------------------------------------------------

describe('Klasa za rad sa stringovnim podacima', function () {
    describe('Funkcija counter vraca broj ponavljanja zadatog karaktera u stringu', function () {
        it('Treba da vrati broj ponavljanja zadatog karaktera', function () {
            //TO DO


        });
    });
});
```

P8: Kreirati modul koji sadrži funkciju izračunaj . Funkcija prihvata tri parametra, operacija, op1 i op2. Funkcija treba da vrati rezultat operacije koja je unešena zajedno sa operandima. Napisati testove za sve operacije.

P9: Kreirati klasu osoba koja ima parametarski konstruktor i metodu fullName() koja vraća puno ime i prezime osobe. Napisati unit testove koji će ispitati povratnu vrednost funckije i tip instanciranog objekta.

```
class Person {
    constructor(name, lastName, age) {
        this.name = name;
        this.lastName = lastName;
        this.age = age;
    }

    fullName() {
        return `${this.name + ` ` + this.lastName}`;
    }
}

module.exports.Person = Person;

-----------------------------------------------------------------------------------------------
// TO DO
```

P10: Kreirati klasu auto, koja ima tri parametra, marku, model i godinu proizvodnje i metodu show(), koja prikazuje podatke jednog automobila. Testirati rezultat ove metode putem statusa koji vraća web server, kada se pozove url u obliku:
http://localhost:3000/show?make=vasaMarka&model=vasModel&year=vasaGodinaProizvodnje


VISOKA TEHNIČKA ŠKOLA
STRUKOVNIH STUDIJA NIŠ