

2024/2025

SVEUČILIŠTE VERN'

# Programiranje za internet 2

Radni materijali

Dean Nižetić

SVEUČILIŠTE VERN' – INTERNET OF THINGS

**Sadržaj:**

Instalacija NODE-a (i NPM) na računalo (radi čak i u učionicama na vernu .....	2
Kreiranje novog NODE/Javascript projekta u VScode-u .....	6
Node HTTP modul – hello world .....	8
Hendlanje različitih metoda i URLova .....	10
Dotjerajmo kod za bolju preglednost i lakše održavanje .....	12
Posluživanje statičkih datoteka.....	14
Baza podataka (sqlite) .....	16
CRUD model na tablici clanova.....	17
Read svih članova u bazi (GET metoda).....	17
Read jednog člana u bazi (opet GET, no sa parametrom) .....	19
Create člana (POST).....	22
Update člana (PUT) .....	26
DELETE člana .....	27
Napomena .....	28
Kompletan kod za CRUD nad članovima .....	28

# Instalacija NODE-a (i NPM) na računalo (radi čak i u učionicama na vernu)

1. Otvoriti cmd.exe (command prompt)

```
Command Prompt
Microsoft Windows [Version 10.0.19045.3208]
(c) Microsoft Corporation. All rights reserved.

C:\Users\prof>node --version
'node' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\prof>npm --version
'npm' is not recognized as an internal or external command,
operable program or batch file.


C:\Users\prof>
```


(slika gore nam kaže da node (i npm) nisu instalirani na našem računalu)


2. Preuzimanje node (i npm) sa <https://nodejs.org/en/download>  
**CRVENO** – ako ste admin na računalu (preuzmite i pokrenite instalaciju)  
**PLAVO** – ako niste admin na računalu na kojem želite instalirati NODE

LTS  
Recommended For Most Users

Current  
Latest Features

  
Windows Installer  
node-v20.9.0-x64.msi

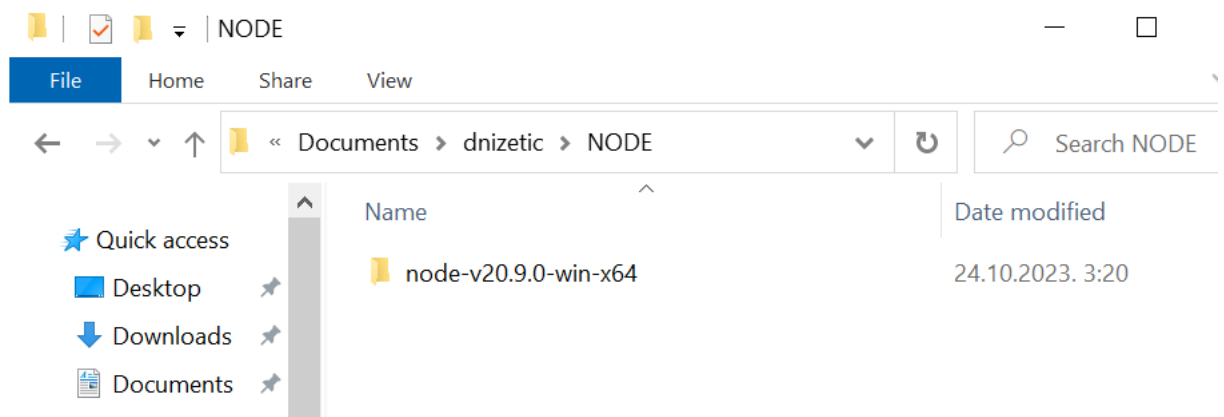
  
macOS Installer  
node-v20.9.0.pkg

  
Source Code  
node-v20.9.0.tar.gz

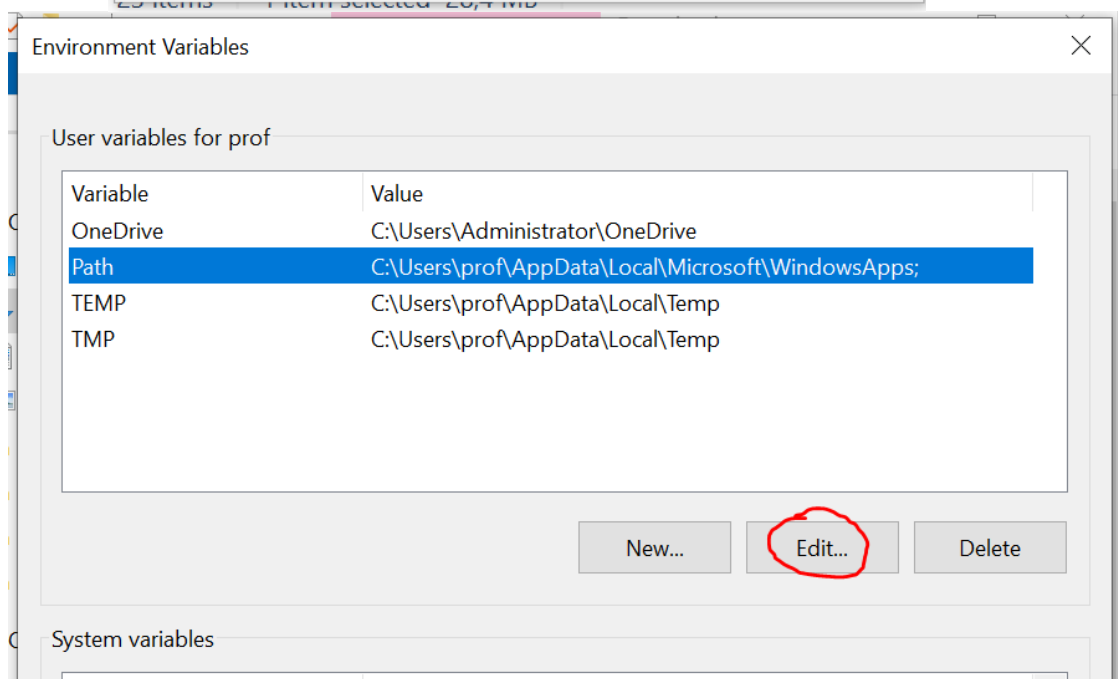
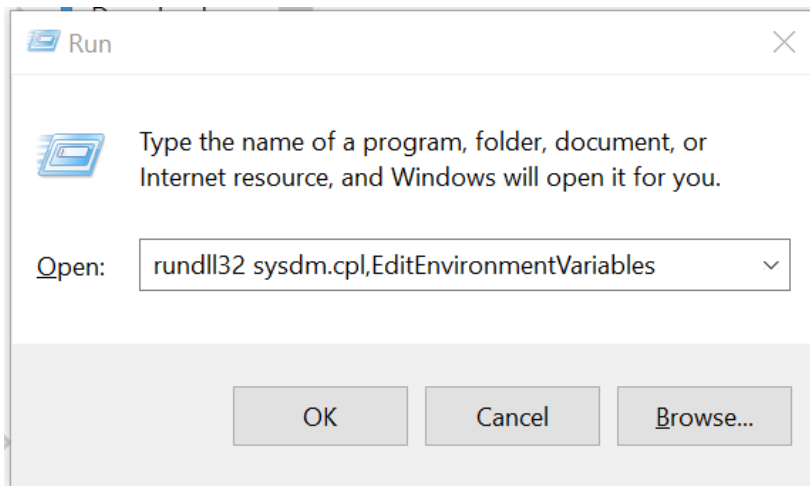
Windows Installer (.msi)	32-bit	64-bit	ARM64
Windows Binary (.zip)	32-bit	64-bit	ARM64
macOS Installer (.pkg)	64-bit / ARM64		
macOS Binary (.tar.gz)	64-bit	ARM64	
Linux Binaries (x64)	64-bit		
Linux Binaries (ARM)	ARMv7	ARMv8	
Source Code	node-v20.9.0.tar.gz		

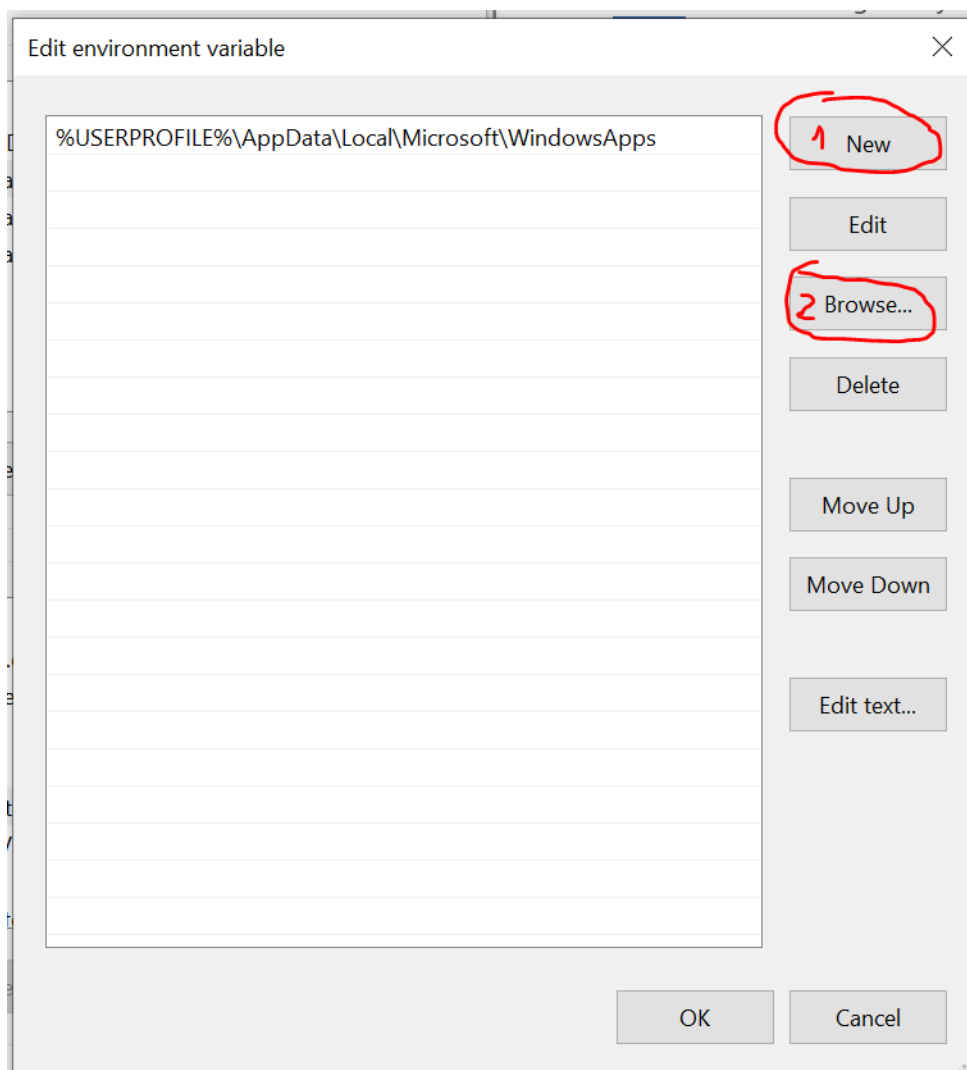
Nakon što preuzmete ZIP datoteku raspakirajte je iz DOWNLOADS u svoj NODE folder u

## DOCUMENTS

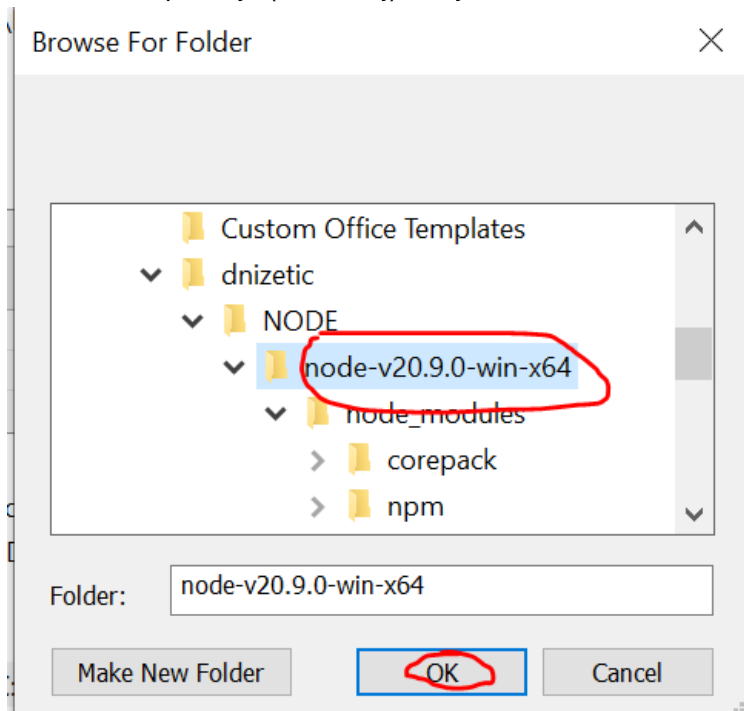


3. Trebamo taj direktorij (u kojem se nalazi NODE.EXE) dodati u PATH putanju windows okoline  
pa pokrećemo komandu  
***rundll32 sysdm.cpl,EditEnvironmentVariables***  
(WIN tipka, kucamo run, a u prozor koji nam se pokaže copy/paste komandu od maloprije)

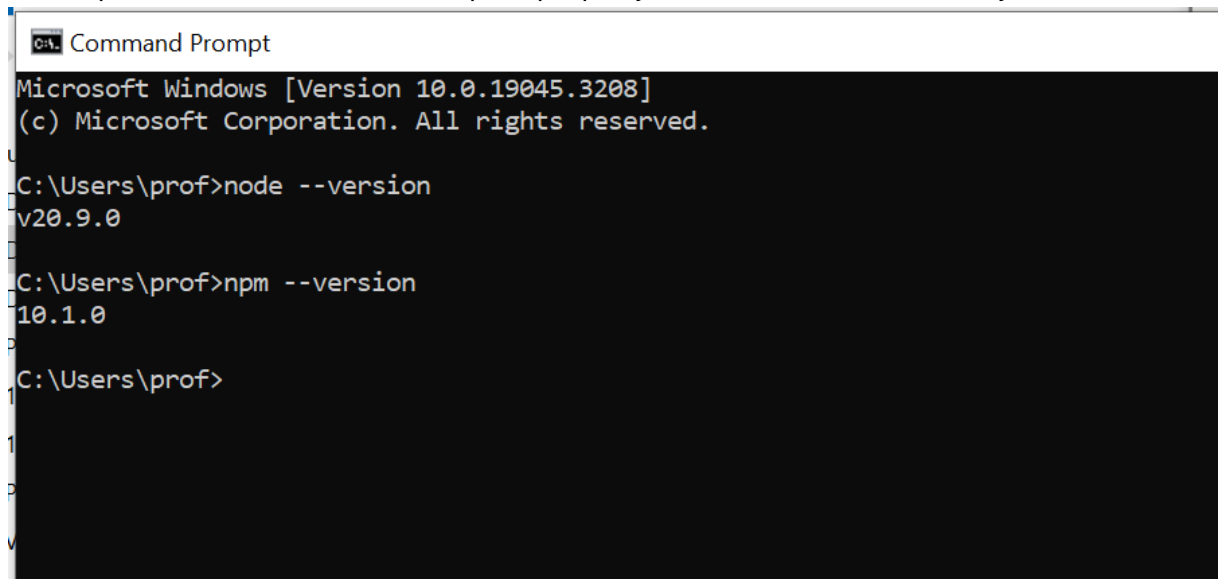




Odaberemo putanju (direktorij) u kojem se nalazi NODE.EXE



4. Potom ponovno otvorimo command prompt i provjerimo da li se node sada vidljiv



```
Command Prompt
Microsoft Windows [Version 10.0.19045.3208]
(c) Microsoft Corporation. All rights reserved.

C:\Users\prof>node --version
v20.9.0

C:\Users\prof>npm --version
10.1.0

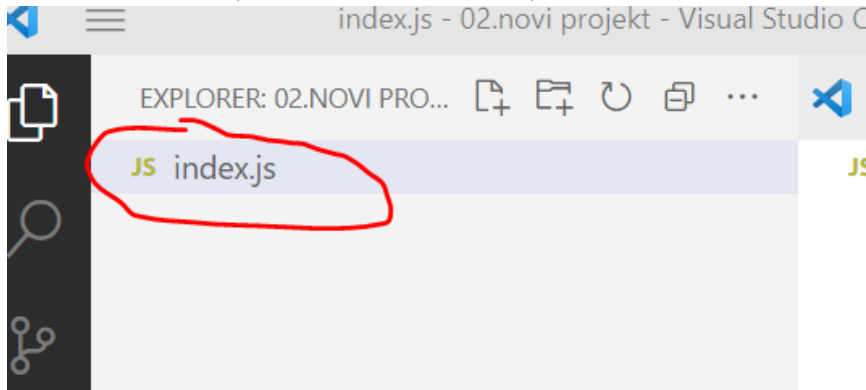
C:\Users\prof>
```

Brojevi verzija se mogu razlikovati s obzirom na to da je uputa napisana prije nekog vremena i verzije su se vjerojatno od tada promijenile

# Kreiranje novog NODE/Javascript projekta u VScode-u

Kako se kreira novi projekt (prazan)

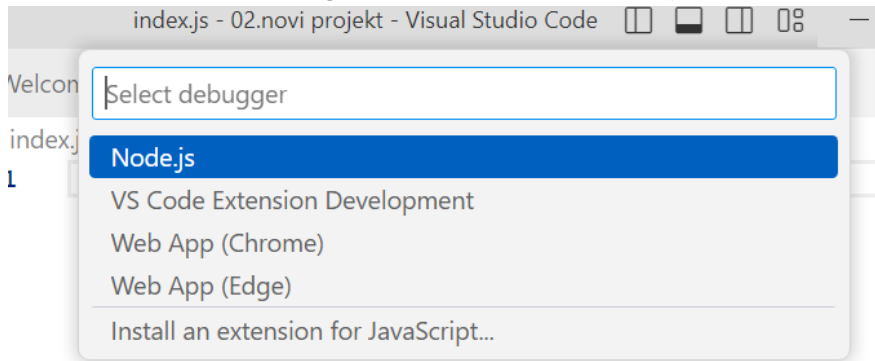
1. Kreiramo folder (negdje na disku) i otvorimo ga u VScode (File->Open Folder)
2. Kreiram datoteku (unutar foldera projekta) i nazovem je index.js



3. Otvorimo terminal (Terminal->new Terminal)
4. U terminal ukucamo: npm init (na sva pitanja odgovorimo sa ENTER) (to nam je proizvelo package.json)
5. U terminal ukucamo: npm install prompt-sync (dobili smo podfolder >node\_modules)
6. Otvorimo package.json i dopišemo start skriptu



7. Odemo u RUN>add configuration i odaberemo node.js

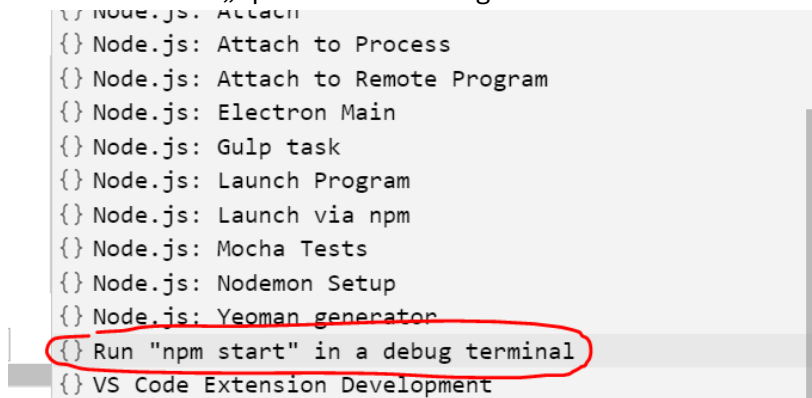


8. To nam je otvorilo launch.json datoteku, u njoj kliknemo na Add configuration (plavi

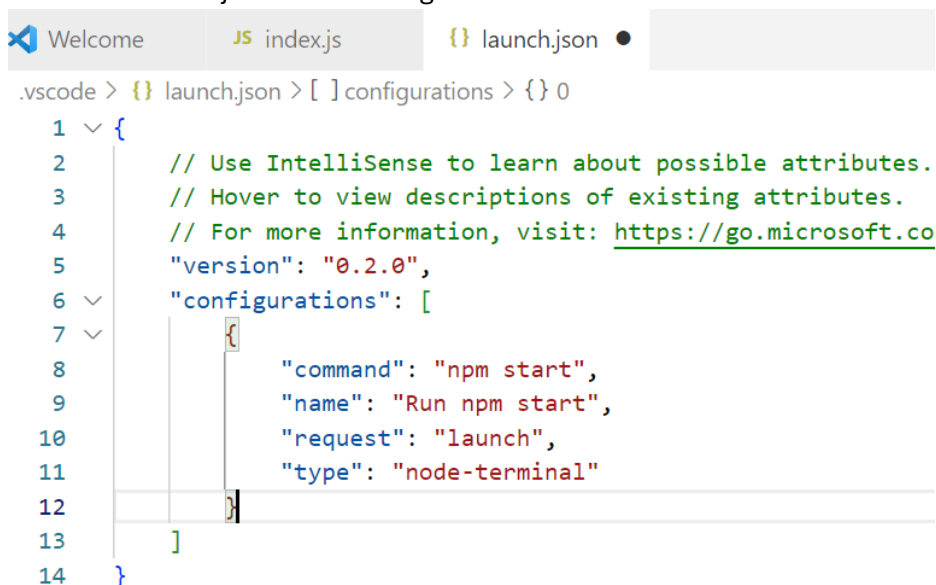
**Add Configuration...**

gumb)

9. Odaberemo Run „npm start“ in a deug terminal



10. To će nam kreirati potrebnu skriptu pa možemo obrisati staru skriptu koja nam je višak.  
Konačna launch.json datoteka izgleda ovako:





# Node HTTP modul – hello world

Ako napravimo novi projekt (datoteka od koje počinje izvršavanje našeg programa neka bude index.js)

U index.js nakucamo sljedeći kod

```
const http=require("http");
http.createServer((req, res) => {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write('<h1>Hello World!</h1>');
    res.end();
}).listen(8080);
```

Prva linija predstavlja „učitavanje“ HTTP modula (koji je dio node-a a koji nam omogućava da kreiramo HTTP server i kroz njega osluškujemo klijentske zahtjeve (browser requests))

Dio koda

```
.listen(8080);
```

Daje naredbu serveru da osluškuje na portu 8080.

Standardni port za HTTP protokol je 80, no na mnogim sustavima programima koji nisu pokrenuti sa admin/root pravima korištenje standardnih portova (sa brojemima ispod 1024) nije dozvoljeno.

Također, treba pripaziti na to je li port koji želimo koristiti slobodan (ukoliko je neka druga aplikacija koja koristi naš željeni port već pokrenuta, time je ona „zauzela“ aplikacijski port pa mi nećemo moći pokrenuti našu aplikaciju na tom istom portu.

Isti kod ću napisati drukčije, kako bih ga učinio razumljivijim

```
const http=require("http");
function serverFunkcija(req, res) {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write('<h1>Hello World!</h1>');
    res.end();
}
const server=http.createServer(serverFunkcija);
server.listen(8080);
```

funkcija „serverFunkcija“ je zadužena za obradu (svih) zahtjeva upućenih serveru, i server će po primanju zahtjeva isti proslijediti upravo na nju

```
const server=http.createServer(serverFunkcija);
```

tom linijom kreiramo server (i kažemo mu koju funkciju će pozivati kada se dogodi zahtjev.

```
server.listen(8080);
```

doista pokreće server na zadanom (8080) portu

Naša funkcija je pomalo glupava, ona na sve zahtjeve odgovara na identičan način

Pa ako zahtjev uputimo na

<http://localhost:8080/>

ili na

<http://localhost:8080/supertrooper>

odgovor će biti isti

i to

linija koja serveru govori da je status odgovora 200 (dakle zahtjev je bio uspješan)

i da te sadržaj (content) odgovora tipa text/html (umjesto npr text/plain ili application/json)

```
res.writeHead(200, {'Content-Type': 'text/html'});
```

više o Content-Type dijelu odgovora možete naći na adresi

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Type>

Naredna linija šalje sadržaj odgovora (response body)

```
res.write('<h1>Hello World!</h1>');
```

Odgovor je ustvari nepotpuni html kod, koji je poslan odjednom, no mogli smo biti nešto korektniji pa umjesto te jedne linije napisati:

```
res.write('<html>');  
res.write('<head><title></title></head>');  
res.write('<body>');  
res.write('<h1>Hello World!</h1>');  
res.write('</body>');  
res.write('</html>');
```

i dati potpuniji odgovor i to kroz nekoliko uzastopnig poziva write metode na response objektu

Na kraju imamo liniju koja označava da smo napisali sve što smo željeli i za je odgovor spreman za vratiti klijentu

```
res.end();
```

# Hendlanje različitih metoda i URLova

Kako bismo mogli razlikovati jedan request od drugog?

Primjerice

<http://localhost:8080/>

od

<http://localhost:8080/supertrooper>

Nakon što zaprimimo request možemo iskoristiti request object (u našem kodu on je req atribut koji smo primili u našoj funkciji za obradu zahtjeva.

Možemo dereferenciranjem dobiti neke nama važne podatke

```
const { headers, method, url } = req;
```

a ako nam je dereferenciranje strano, evo istog koda u razumljivijoj varijanti

```
const headers=req.headers;  
const method=req.method;  
const url = req.url;
```

headers objekt sadrži sve key-value parove iz requesta koji nam je klijent uputio, npr:

```
{  
  host: 'zoomer.vernnet.hr:8080',  
  connection: 'keep-alive',  
  pragma: 'no-cache',  
  'cache-control': 'no-cache',  
  'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36  
(KHTML, like Gecko) Chrome/130.0.0.0 Safari/537.36 Edg/130.0.0.0',  
  accept: 'image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8',  
  referer: 'http://zoomer.vernnet.hr:8080/',  
  'accept-encoding': 'gzip, deflate',  
  'accept-language': 'en-US,en;q=0.9',  
}
```

Method sadrži metodu kojom je serveru pristupljeno (GET, POST, PUT, DELETE,...)

A url sadrži adresu (putanju do „datoteke“ koja je zatražena. Pa tako u slučaju

<http://localhost:8080/>

vrijednost url-a je „/“ (samo kosa crta – adresa homepage-a)

a za slučaj

<http://localhost:8080/supertrooper>

url=“/supertrooper“

na taj način možemo, ovisno o vrsti zahtjeva, odlučiti želimo li odgovoriti na jedan ili drugi način, odnosno koji ćemo response poslati nazad klijentu

Evo jednog (malo) potpunijeg primjera: koji zna kako odgovoriti na dva različita zahtjeva i čak odgovoriti sa 404 ukoliko zahtjev nije bio niti jedan od dva očekivana

```
const http=require("http");

function serverFunkcija(req, res) {
  //const { headers, method, url } = req;
  const headers=req.headers;
  const method=req.method;
  const url = req.url;

  if (method=='GET' && url=='/'){
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write('<html>');
    res.write('<head><title></title></head>');
    res.write('<body>');
    res.write('<a href="/supertrooper">go to supertrooper</a>');
    res.write('</body>');
    res.write('</html>');
    res.end();
  }else if (method=='GET' && url=='/supertrooper'){
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write('<html>');
    res.write('<head><title></title></head>');
    res.write('<body>');
    res.write('<a href="/">back to home</a>');
    res.write('</body>');
    res.write('</html>');
    res.end();
  }else{
    res.statusCode = 404;
    res.end();
  }
}

const server=http.createServer(serverFunkcija);
server.listen(8080);
```

## Dotjerajmo kod za bolju preglednost i lakše održavanje

Naravno, ovako napisan kod nije baš pregledan, pa bismo mogli u našoj osnovnoj serverFunkciji zadržati funkcionalnost routinga, a dio koji se bavi time kako izgleda odgovarajući odgovor izdvojiti van te funkcije kako bismo dobili na preglednosti i čitljivosti koda

```
const http=require("http");

function serverFunkcija(req, res) {
  //const { headers, method, url } = req;
  const headers=req.headers;
  const method=req.method;
  const url = req.url;
  if (method=='GET' && url=='/'){
    get_Home(req,res)
  }else if (method=='GET' && url=='/supertrooper'){
    get_Supertrooper(req,res)
  }else{
    res.statusCode = 404;
    res.end();
  }
}

function get_Home(req,res){
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write('<html>');
  res.write('<head><title></title></head>');
  res.write('<body>');
  res.write('<a href="/supertrooper">go to supertrooper</a>');
  res.write('</body>');
  res.write('</html>');
  res.end();
}

function get_Supertrooper(req,res){
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write('<html>');
  res.write('<head><title></title></head>');
  res.write('<body>');
  res.write('<a href="/">back to home</a>');
  res.write('</body>');
  res.write('</html>');
  res.end();
}

const server=http.createServer(serverFunkcija);
server.listen(8080);
```

Možemo otići još i korak dalje i učiniti naš kod još preglednijim i lakšim za dodatno mijenjanje i nadogradnju – korištenjem javascript Map objekta da bismo čuvali mapiranje URL-a zahtjeva na funkciju kojom taj zahtjev obrađujemo

```
const http=require("http");

const urlmap=new Map();

function _Home(req,res){
  if (req.method=='GET'){
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write('<html>');
    res.write('<head><title></title></head>');
    res.write('<body>');
    res.write('<a href="/supertrooper">go to supertrooper</a>');
    res.write('</body>');
    res.write('</html>');
    res.end();
    return true;
  }
  return false;
}
urlmap.set("/",_Home);

function _Supertrooper(req,res){
  if (req.method=='GET'){
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write('<html>');
    res.write('<head><title></title></head>');
    res.write('<body>');
    res.write('<a href="/">back to home</a>');
    res.write('</body>');
    res.write('</html>');
    res.end();
    return true;
  }else{
    return false;
  }
}
urlmap.set("/supertrooper",_Supertrooper);

function serverFunkcija(req, res) {
  let handler=urlmap.get(req.url);
  let handled=false;
  if (handler!=undefined){
    handled=handler(req,res);
  }
  if(!handled){
    res.statusCode = 404;
    res.end();
  }
}

const server=http.createServer(serverFunkcija);
server.listen(8080);
```

---

*Za vježbu, napravite nešto slično, napravite vlastiti mali server koji će znati prepoznati nekoliko različitih zahtjeva (za sada se držimo GET metode) i odgovoriti na njih!*

---

## Posluživanje statičkih datoteka

Da bismo kao odgovor klijentu na njegov zahtjev vratili statičku datoteku (datoteka pohranjena u svom konačnom obliku negdje na disku) trebamo koristiti ugrađenu biblioteku (library) za čitanje datoteka („fs“)

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

našu funkciju za prihvatanje zahtjeva ćemo mrvicu izmijeniti tako da joj dodamo `staticHandler` funkciju, koja će ukoliko se u mapi sa funkcijama koje hendlaju dinamičke zahtjeve ne nalazi odgovarajuća, pokušati ćemo zahtijevani URL protumačiti kao zahtjev za statičkom datotekom (pod uvjetom da se radi o GET metodi – jer se statičke datoteke uvijek traže isključivo putem GET metode)

```
function serverFunkcija(req, res) {
  let handler=urlmap.get(req.url);
  let handled=false;
  if (handler!==undefined){
    handled=handler(req,res);
  }else{
    handled=staticHandler(req,res);
  }
  if(!handled){
    res.statusCode = 404;
    res.end();
  }
}
```

Čitanje sadržaja datoteke i vraćanje njenog sadržaja klijentu unutar odgovora moglo bi izgledati ovako:

```
function staticHandler(req,res){

  if (req.method!='GET') return false;

  const staticPath='./static_files'
  let doc=req.url;
  if (doc.charAt(doc.length-1)=='/'){
    doc='/index.html'
  }
  let docPath=staticPath+doc;
  try{
    const data=fs.readFileSync(docPath)
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    res.end();
    return true;
  }catch(err){
    return false;
  }
}
```

Try / catch dio bismo mogli čitati ovako: pokušaj izvršiti dio napisan u TRY bloku, a ako ne uspiješ, jer se dogodila greška onda se izvršava CATCH dio (koji je „uhvatio“ grešku i omogućuje nam da je obradimo na nama ispravan način)

Trebali bismo primijetiti i liniju koja postavlja Content type odgovora klijentu... odgovor je hardcoded na text/html a to nije istina, jer web server bi trebao moći vraćati razne tipove datoteka

```
res.writeHead(200, {'Content-Type': 'text/html'});
```

Taj problem možemo riješiti korištenjem MIME biblioteka (nije dio node.js core paketa pa ga je potrebno instalirati)

U terminalskom prozoru našeg VSCODE-alata dajemo naredbu za instalaciju biblioteke

```
npm install mime-types
```

nakon što je biblioteka uspješno instalirana možemo je koristiti u kodu

Dodajemo odgovarajući require statement koji će biblioteku učitati i omogućiti nam korištenje

```
const mime = require("mime-types");
```

Sada možemo na jednostavan način doznati koji nam je mime type (content type) za našu zatraženu datoteku (mime biblioteka će to „pogoditi“ na osnovu ekstenzije datoteke)

```
const mime_type = mime.lookup(docPath)
res.writeHead(200, {'Content-Type': mime_type});
```

U konačnici, evo cijele funkcije koja će nam hendlati vraćanje statičkih datoteka

```
function staticHandler(req,res){
  if (req.method!='GET') return false;
  const staticPath='./static_files'
  let doc=req.url;
  if (doc.charAt(doc.length-1)=='/'){
    doc='/index.html'
  }
  let docPath=staticPath+doc;
  try{
    const data=fs.readFileSync(docPath)
    const mime_type = mime.lookup(docPath)
    res.writeHead(200, {'Content-Type': mime_type});
    res.write(data);
    res.end();
    return true;
  }catch(err){
    return false;
  }
}
```



## Baza podataka (sqlite)

Za početak, trebamo neku bazu sa kojom se možemo „igrati“, za kreiranje sqlite baze možemo koristiti dBeaver alat (također, za potrebe primjera u ovim materijalima koristiti ćemo unaprijed pripremljenu bazu knjiznica.db)

Dokumentacija za korištenje sqlite biblioteke nalazi se na linku: [SQLite | Node.js v23.1.0 Documentation](#) (no i u ovim materijalima ćemo proći osnovne scenarije korištenja)

Da bismo koristili sqlite bazu podataka koja je ugrađena u node.js trebamo uključiti korištenje još uvijek eksperimentalne sqlite biblioteke.

Otvorimo package.json datoteku sa postavkama našeg projekta i izmijenimo komandu za pokretanje programa (start skripta) i za pokretanja dodati flag argument **--experimental-sqlite**

Cijela package.json datoteka treba izgledati ovako

```
{
  "name": "pzi2",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1",
    "start": "node --experimental-sqlite index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "mime-types": "^2.1.35"
  }
}
```

Potom možemo na početak datoteke u kojoj ćemo koristiti bazu, dodati require koji će učitati sqlite biblioteku

```
const sqlite = require('node:sqlite');
```

U Mapu handlera ćemo dodati funkciju za hendlanje korisnicima (\_Users)

```
function _Users(req, res, q){
  ...
}
...
urlmap.set("/users", _Users);
```

Funkcija je doduše još uvijek prazna i ništa ne radi ali polako dolazimo i do toga...

# CRUD model na tablici članova

## Read svih članova u bazi (GET metoda)

Funkcija nam za početak treba iz baze dohvatiti, i klijentu odgovoriti sa popisom svih članova upisanih u bazu (tablica članovi)

```
function _Users(req,res) {  
  if (req.method=='GET'){  
    const db=new sqlite.DatabaseSync('./knjiznica.db',{open:false});  
    db.open();  
    const sql="SELECT id,ime,prezime,datumRodjenja FROM clanovi";  
    const statement=db.prepare(sql);  
    const results=statement.all();  
    console.log(statement.sourceSQL,"\n",results);  
    res.writeHead(200, {'Content-Type': 'application/json'});  
    res.write(JSON.stringify(results));  
    res.end();  
    db.close();  
    return true;  
  }  
  return false;  
}
```

Objašnjenje koda:

Funkcija je mapirana da se pozove kada klijent zatraži /users resurs od servera.

```
urlmap.set("/users",_Users);
```

S obzirom da želimo klijentu vratiti popis svih članova (READ) onda želimo reagirati na GET metodu requesta (za sada, ukoliko se ne radi o GET requestu, metoda odgovara sa FALSE – nismo obradili zahtjev i prepuštamo to osnovnoj funkciji – možda postoji statička datoteka koju o ovom trenutku treba vratiti – no naša funkcija se time ne zamara, nije to njen posao)

Spajamo se na bazu i otvaramo konekciju

```
const db=new sqlite.DatabaseSync('./knjiznica.db',{open:false});  
db.open();
```

Pripremamo i izvršavamo SQL komandu za dohvaćanje svih zapisa iz tablice članova

```
const sql="SELECT id,ime,prezime,datumRodjenja FROM clanovi";  
const statement=db.prepare(sql);  
const results=statement.all();
```

A varijabla result nam sadrži popis (JSON ARRAY objekt) sa svim članovima (i njihovim pripadajućim atributima)

```
[
  {
    "id": 9,
    "ime": "Pero",
    "prezime": "Perić",
    "datumRodjenja": "2000-08-01"
  },
  {
    "id": 12,
    "ime": "Jure",
    "prezime": "Jurić",
    "datumRodjenja": "2003-11-04"
  }
]
```

Kao odgovor vraćamo JSON pa postavljamo Content-type na application/json

```
res.writeHead(200, {'Content-Type': 'application/json'});
```

objekt (array) sa rezultatima pretvaramo u string (tekst) koji možemo zapisati u odgovor i vratiti klijentu

```
res.write(JSON.stringify(results));
res.end();
```

I nikako ne smijemo zaboraviti zatvoriti konekciju na bazu

```
db.close();
```

## Read jednog člana u bazi (opet GET, no sa parametrom)

Slično, mogli bismo imati potrebu dohvatiti jednog od članova, proslijeđujući u GET metodi na isti URL kao query parametar ID sa vrijednošću koja sadrži identifikator traženog člana

```
GET /users?id=12
```

Za to će biti potrebno malo modificirati metodu, prvo trebamo naučiti kako pročitati query parametre (prosljeđene kroz url iza znaka ?)

Za to koristimo u node.js ugrađenu biblioteku (node:url)

```
const url = require('node:url');
```

a sadržaj query stringa možemo potom pročitati kao

```
let q=new url.URL(req.url,"http://localhost/");
```

To ćemo ugraditi u našu osnovnu serverFunkciju

```
function serverFunkcija(req, res) {  
  let q=new url.URL(req.url,"http://localhost/");  
  let pathname=q.pathname;  
  let handler=urlmap.get(pathname);  
  let handled=false;  
  if (handler!==undefined){  
    handled=handler(req,res,q);  
  }else{  
    handled=staticHandler(req,res);  
  }  
  if(!handled){  
    res.statusCode = 404;  
    res.end();  
  }  
}
```

Primjetimo kako taj query objekt (q) proslijeđujemo našim handler funkcijama kao treći argument To znači da će one handler funkcije, koje žele koristiti query parametre trebati dodati i treći argument u svoj potpis

```
function _Users(req,res, q){  
  ...
```

Time smo pripremili teren za zaprimanje query parametra u našoj funkciji koja želi zaprimiti ID i u tom slučaju vratiti jednog člana (ne više popis svih)

Cijela (za sada) handler funkcija će sada izgledati ovako:

```
function _Users(req,res, q){
  const sp=q.searchParams
  const id=sp.get("id");
  if (req.method=='GET' && id==null){
    const db=new sqlite.DatabaseSync('./knjiznica.db',{open:false});
    db.open();
    const sql="SELECT id,ime,prezime,datumRodjenja FROM clanovi";
    const statement=db.prepare(sql);
    const results=statement.all();
    console.log(statement.sourceSQL,"\n",results);
    res.writeHead(200, {'Content-Type': 'application/json'});
    res.write(JSON.stringify(results));
    res.end();
    db.close();
    return true;
  }else if (req.method=='GET' && id!=null){
    const db=new sqlite.DatabaseSync('./knjiznica.db',{open:false});
    db.open();
    const sql="SELECT id,ime,prezime,datumRodjenja FROM clanovi WHERE id=:id";
    const statement=db.prepare(sql);
    const namedParams={"id":id};
    const result=statement.get(namedParams);
    if (result==undefined){
      res.statusCode = 404; // not found
      res.statusMessage = "not found - Missing record with given ID";
      res.end();
      return true;
    }
    res.writeHead(200, {'Content-Type': 'application/json'});
    res.write(JSON.stringify(result));
    res.end();
    db.close();
    return true;
  }
  return false;
}
```

Žutom bojom su označeni novododani dijelovi

Prvo, uzimamo **searchParams** iz prosljeđenog nam query stringa, a potom iz njega uzimamo vrijednost parametra sa nazivom id... on nam predstavlja identifikator zapisa u članovima kojeg želimo dohvatiti i vratiti klijentu

```
const sp=q.searchParams
const id=sp.get("id");
```

Primjetimo da je i if dio za popis svih korisnika malo modificiran pa sada glasi

```
if (req.method=='GET' && id==null){
```

time smo ustvari rekli da cijeli popis vraćamo ukoliko je metoda bila GET, ali i id nam nije prosljeđen od strane klijenta

Ostatak koda, drugi IF se bavi onim što se događa kada metoda jest GET i kada smo dobili neki identifikator člana (id)

```
else if (req.method=='GET' && id!=null){
  const db=new sqlite.DatabaseSync('./knjiznica.db',{open:false});
  db.open();
  const sql="SELECT id,ime,prezime,datumRodjenja FROM clanovi WHERE id=:id";
  const statement=db.prepare(sql);
  const namedParams={"id":id};
  const result=statement.get(namedParams);
  if (result==undefined){
    res.statusCode = 404; // not found
    res.statusMessage = "not found - Missing record with given ID";
    res.end();
    return true;
  }
  res.writeHead(200, {'Content-Type': 'application/json'});
  res.write(JSON.stringify(result));
  res.end();
  db.close();
  return true;
}
```

Ovom prilikom ne dohvaćamo listu članova, dohvaćamo samo jednog pa će metoda koju pozivamo na statementu biti GET (umjesto ALL koju smo koristili u prethodnom slučaju)

```
const result=statement.get(namedParams);
```

Ona (.get) će vratiti prvi rezultat koji će biti vraćen od strani sql-a koji smo složili kao upit u bazu.

Očito je potrebno prilagoditi i sam upit kako bi dohvatio samo taj jedan redak, jednog člana koji nam je potreban. Stoga dodajemo WHERE uvjet u naš SQL

```
const sql="SELECT id,ime,prezime,datumRodjenja FROM clanovi WHERE id=:id";
```

Sintaksa id=:id predstavlja imenovani parametar, a statement će na njegovo mjesto postaviti vrijednost parametra sa imenom „id“

Taj parametar smo postavili u objekt (u kojeg možemo postaviti i više od jednog parametra, iako nam je u ovom slučaju bio dovoljan samo 1)... a taj parametar sadrži upravo id člana kojeg želimo dohvatiti

```
const namedParams={"id":id};
const result=statement.get(namedParams);
```

Imamo još i jedan if u kojem se pitamo jesmo li pronašli člana sa traženim IDjem, jer ako nismo onda ćemo to klijentu prijaviti kao 404 grešku (sa opisom da traženi zapis ne postoji)

```
if (result==undefined){
  res.statusCode = 404; // not found
  res.statusMessage = "not found - Missing record with given ID";
  res.end();
  return true;
}
```

Ostatak koda bi nam trebao biti jasan jer je poznat od prije

## Create člana (POST)

POST request koji nam šalje klijent će nam sadržavati JSON objekt koji je poslan kroz tijelo requesta pa ćemo trebati naučiti kako pročitati tijelo requesta (body) i kako ga pretvoriti u JSON objekt s kojim možemo lako raditi

Čitanje body-ja radimo u dva dijela, korištenjem evenata (data i end) koje nam daje request objekt.

**data event** – će okinuti svaki puta kada request pročita dio tijela zahtjeva (nažalost on nam dolazi u dijelovima, pa ga moramo slagati kroz više događaja

```
let data='';
req.on('data',function(part){
  data=data+part;
});
```

end event – će nam biti signaliziran samo jednom, nakon što je kompletan body pročitan (i proslijeđen kroz data evente

```
req.on('end',function(){
  ...
})
```

U trenutku kada je pozvana funkcija (okinut je END event, varijabla data će već sadržavati kompletan sadržaj body-ja

Sada možemo koristiti

```
params=JSON.parse(data);
```

da bismo string/text iz dana pretvorili u JSON objekt s kojim možemo lakše raditi i pristupati njegovim atributima.

Nažalost, moguće je da su podaci koje smo pročitali neispravni i da se ne mogu pretvoriti u JSON pa moramo koristiti try/catch block kako bismo uhvatili eventualnu grešku u konverziji

```
let params='';
try{
  params=JSON.parse(data);
}catch(err){
  res.statusCode = 406; // not acceptable – body nije JSON
  res.statusMessage = "not acceptable - "+err;
  res.end();
  return;
}
```

Ukoliko se dogodila greška u konverziji, klijentu ćemo vratiti 406 – poruku da nam body podaci koje nam je poslao nisu prihvatljivi – jer očekujemo JSON formatirane podatke

Kod koji se bavi čitanjem body-ja i konverzijom pročitanih podataka u JSON objekt možemo smjestiti u našu glavnu serverFunkciju i ona će sada izgledati ovako:

```
function serverFunkcija(req, res) {
  let q=new url.URL(req.url,"http://localhost/");

  let data='';
  req.on('data',function(part){
    data=data+part;
  });
  req.on('end',function(){
    if (req.headers['content-type']=='application/json'){
      try{ // konvertiramo podatke u JSON objekt
        data=JSON.parse(data);
      }catch(err){
        res.statusCode = 406; // not acceptable - trebamo JSON objekt
        res.statusMessage = "not acceptable - "+err;
        res.end();
        return true;
      }
    }
    let pathname=q.pathname;
    let handler=urlmap.get(pathname);
    let handled=false;
    if (handler!=undefined){
      handled=handler(req,res,q,data);
    }else{
      handled=staticHandler(req,res);
    }
    if(!handled){
      res.statusCode = 404;
      res.end();
    }
  });
}
```



Sada smo spremni za insert novog člana u tablicu članova (

```
const db=new sqlite.DatabaseSync('./knjiznica.db',{open:false});
db.open();
const sql="INSERT INTO clanovi(ime,prezime,datumRodjenja) VALUES
(:ime,:prezime,:datumRodjenja)";
const statement=db.prepare(sql);
const result=statement.run(params);
if (result.changes==0){
    res.statusCode = 410; // gone - zapis ne postoji
    res.statusMessage = "gone - Zero records inserted";
    res.end();
    return;
}
params['id']=result.lastInsertRowid;
res.writeHead(200, {'Content-Type': 'application/json'});
res.write(JSON.stringify(params));
res.end();
db.close();
```

Gdje su razlike?

Sql je insert sintakse i sadrži parametre (:ime, :prezime, :datumRodjenja) koje želimo dodati u tablicu

```
const sql="INSERT INTO clanovi(ime,prezime,datumRodjenja) VALUES
(:ime,:prezime,:datumRodjenja)";
```

Koristimo metodu run na statementu

```
const result=statement.run(params);
```

a ona će nam vratiti objekt sa dva atributa

```
{
  changes:1,
  lastInsertRowid:13
}
```

Changes predstavlja broj zapisa na koje smo utjecali (1 ukoliko je zapis uspješno dodan, 0 ukoliko to nije uspjelo)

lastInsertRowid će sadržavati identifikator (id) novo dodanog zapisa

Parametri koje smo koristili za dodavanje zapisa su sadržavali sve osim IDja, pa ako taj id dodamo parametrima, imati ćemo kompletan zapis tog novog člana

```
params['id']=result.lastInsertRowid;
```

Koji potom, kao JSON možemo vratiti klijentu kao informaciju o novo dodanom zapisu (i njegovom novom IDju)

```
res.writeHead(200, {'Content-Type': 'application/json'});
res.write(JSON.stringify(params));
```

A evo i cijele if grane koja se bavi dodavanjem novog člana (i koju dodajemo u handler funkciju) – dodana je i provjera kojom utvrđujemo da smo doista dobili JSON objekt koji nam je potreban

```
else if(req.method=='POST'){
  if (data==' ' || req.headers['content-type']!='application/json') {
    res.statusCode = 406; // not acceptable - trebamo JSON objekt
    res.statusMessage = "not acceptable - missing JSON body";
    res.end();
    return true;
  }
  let params=data;
  const db=new sqlite.DatabaseSync('./knjiznica.db',{open:false});
  db.open();
  const sql="INSERT INTO clanovi(ime,prezime,datumRodjenja) VALUES
(:ime,:prezime,:datumRodjenja)";
  const statement=db.prepare(sql);
  const result=statement.run(params);
  if (result.changes==0){
    res.statusCode = 410; // gone - zapis ne postoji
    res.statusMessage = "gone - Zero records inserted";
    res.end();
    return true;
  }
  params['id']=result.lastInsertRowid;
  res.writeHead(200, {'Content-Type': 'application/json'});
  res.write(JSON.stringify(params));
  res.end();
  db.close();
  return true;
}
```

## Update člana (PUT)

PUT/Update nam je umnogome sličan kao POST, i kod njega moramo pročitati body da bismo došli do JSON objekta sa vrijednostima na koje želimo promijeniti zapis u članovima, ID, kao jedna od tih vrijednosti nam predstavlja identifikator zapisa u bazi članova koji želimo mijenjati

If grana za PUT sa označenim razlikama (u odnosu na POST) nam izgleda ovako

```
else if(req.method=='PUT'){
  if (data==' ' || req.headers['content-type']!='application/json') {
    res.statusCode = 406; // not acceptable - trebamo JSON objekt
    res.statusMessage = "not acceptable - missing JSON body";
    res.end();
    return true;
  }
  let params=data;
  const db=new sqlite.DatabaseSync('./knjiznica.db',{open:false});
  db.open();
  const sql="UPDATE clanovi SET ime=:ime, prezime=:prezime,
datumRodjenja=:datumRodjenja where id=:id";
  const statement=db.prepare(sql);
  const result=statement.run(params);
  if (result.changes==0){
    res.statusCode = 410; // gone - zapis ne postoji
    res.statusMessage = "gone - Zero records updated";
    res.end();
    return true;
  }
  res.writeHead(200, {'Content-Type': 'application/json'});
  res.end();
  db.close();
  return true;
}
```

U biti, jedina razlika je u samom SQL upitu koji ne dodaje, već mijenja atribut zapisa definiranog sa proslijeđenim IDjem

## DELETE člana

Zahtjev za brisanjem člana bi trebali od klijenta dobiti kao

```
DELETE /users?id=14
```

Gdje se id mijenja s obzirom na to koji se zapis želi obrisati

Krenimo odmah sa kompletnim primjerom IF grane za brosanje zapisa

```
else if(req.method=='DELETE'){
  if (id==null){
    res.statusCode = 406; // not acceptable
    res.statusMessage = "not acceptable - missing ID parameter";
    res.end();
    return true;
  }
  const db=new sqlite.DatabaseSync('./knjiznica.db',{open:false});
  db.open();
  const sql="DELETE FROM clanovi WHERE id=:id";
  const statement=db.prepare(sql);
  const namedParams={"id":id};
  const result=statement.run(namedParams);
  if (result.changes==0){
    res.statusCode = 410; // gone -
    res.statusMessage = "gone - zapis ne postoji";
    res.end();
    return true;
  }
  res.statusCode = 200
  res.end();
  db.close();
  return true;
}
```

U prvom dijelu provjeramo jesmo li doili ID parametar (jer bez njega ne znamo koji zapis treba obrisati, ako nismo, klijentu prijavljujemo grešku

```
if (id==null){
  res.statusCode = 406; // not acceptable
  res.statusMessage = "not acceptable - missing ID parameter";
  res.end();
  return true;
}
```

Kao parametar u statement proslijeđujemo samo taj ID jer nam je on jedini potreban za identifikaciju zapisa koji treba obrisati

```
const sql="DELETE FROM clanovi WHERE id=:id";
...
const namedParams={"id":id};
const result=statement.run(namedParams);
```

Već od prije poznati result sadrži result.changes koji će nam reći jesmo li obrisali zapis (0 znači da nismo obrisali ništa, no moguće je i da smo obrisali više od jednog zapisa ukoliko smo zabrljali sql upit)

## Napomena

Napomena: u svim slučajevima (GET, POST, PUT, DELETE) bi se mogli detaljnije pozabaviti validacijom podataka koje nam je klijent proslijedio. Primjerice mogli bismo provjeriti kod PUT request je li među parametrima proslijeđen ID koji nam je nužan za identificiranje zapisa i bez njega update nema smisla. Validaciju radimo zato da bismo klijentu mogli vratiti odgovor sa što je moguće smislenijom porukom kako bismo olakšali razvoj na klijentskoj strani.

## Kompletan kod za CRUD nad članovima

```
function _Users(req,res, q, data){
  const sp=q.searchParams
  const id=sp.get("id");
  if (req.method=='GET' && id==null){
    const db=new sqlite.DatabaseSync('./knjiznica.db',{open:false});
    db.open();
    const sql="SELECT id,ime,prezime,datumRodjenja FROM clanovi";
    const statement=db.prepare(sql);
    const results=statement.all();
    console.log(statement.sourceSQL,"\n",results);
    res.writeHead(200, {'Content-Type': 'application/json'});
    res.write(JSON.stringify(results));
    res.end();
    db.close();
    return true;
  }else if (req.method=='GET' && id!=null){
    const db=new sqlite.DatabaseSync('./knjiznica.db',{open:false});
    db.open();
    const sql="SELECT id,ime,prezime,datumRodjenja FROM clanovi WHERE id=:id";
    const statement=db.prepare(sql);
    const namedParams={"id":id};
    const result=statement.get(namedParams);
    if (result==undefined){
      res.statusCode = 404; // not found
      res.statusMessage = "not found - Missing record with given ID";
      res.end();
      return true;
    }
    res.writeHead(200, {'Content-Type': 'application/json'});
    res.write(JSON.stringify(result));
    res.end();
    db.close();
    return true;
  }else if (req.method=='POST'){
    if (data==' ' || req.headers['content-type']!='application/json') {
      res.statusCode = 406; // not acceptable - trebamo JSON objekt
      res.statusMessage = "not acceptable - missing JSON body";
      res.end();
      return true;
    }
    let params=data;
    const db=new sqlite.DatabaseSync('./knjiznica.db',{open:false});
    db.open();
    const sql="INSERT INTO clanovi(ime,prezime,datumRodjenja) VALUES (:ime,:prezime,:datumRodjenja)";
    const statement=db.prepare(sql);
    const result=statement.run(params);
    if (result.changes==0){
      res.statusCode = 410; // gone - zapis ne postoji
      res.statusMessage = "gone - Zero records inserted";
      res.end();
      return true;
    }
  }
```

```

    }
    params['id']=result.lastInsertRowid;
    res.writeHead(200, {'Content-Type': 'application/json'});
    res.write(JSON.stringify(params));
    res.end();
    db.close();
    return true;
  }else if(req.method=='PUT'){
    if (data==' ' || req.headers['content-type']!='application/json') {
      res.statusCode = 406; // not acceptable - trebamo JSON objekt
      res.statusMessage = "not acceptable - missing JSON body";
      res.end();
      return true;
    }
    let params=data;
    const db=new sqlite.DatabaseSync('./knjiznica.db',{open:false});
    db.open();
    const sql="UPDATE clanovi SET ime=:ime, prezime=:prezime,
datumRodjenja=:datumRodjenja where id=:id";
    const statement=db.prepare(sql);
    const result=statement.run(params);
    if (result.changes==0){
      res.statusCode = 410; // gone - zapis ne postoji
      res.statusMessage = "gone - Zero records updated";
      res.end();
      return true;
    }
    res.writeHead(200, {'Content-Type': 'application/json'});
    res.end();
    db.close();
    return true;
  }else if(req.method=='DELETE'){
    if (id==null){
      res.statusCode = 406; // not acceptable
      res.statusMessage = "not acceptable - missing ID parameter";
      res.end();
      return true;
    }
    const db=new sqlite.DatabaseSync('./knjiznica.db',{open:false});
    db.open();
    const sql="DELETE FROM clanovi WHERE id=:id";
    const statement=db.prepare(sql);
    const namedParams={"id":id};
    const result=statement.run(namedParams);
    if (result.changes==0){
      res.statusCode = 410; // gone - zapis ne postoji
      res.statusMessage = "gone - zapis ne postoji u bazi sa navedenim
IDjem";
      res.end();
      return true;
    }
    res.statusCode = 200
    res.end();
    db.close();
    return true;
  }
  return false;
}

```