 **UNIVERZITET SINGIDUNUM U BEOGRADU**

**FAKULTET ZA INFORMATIKU I RAČUNARSTVO**

**Uvod u Node.JS i programiranje vođeno događajima**

**-Seminarski Rad-**

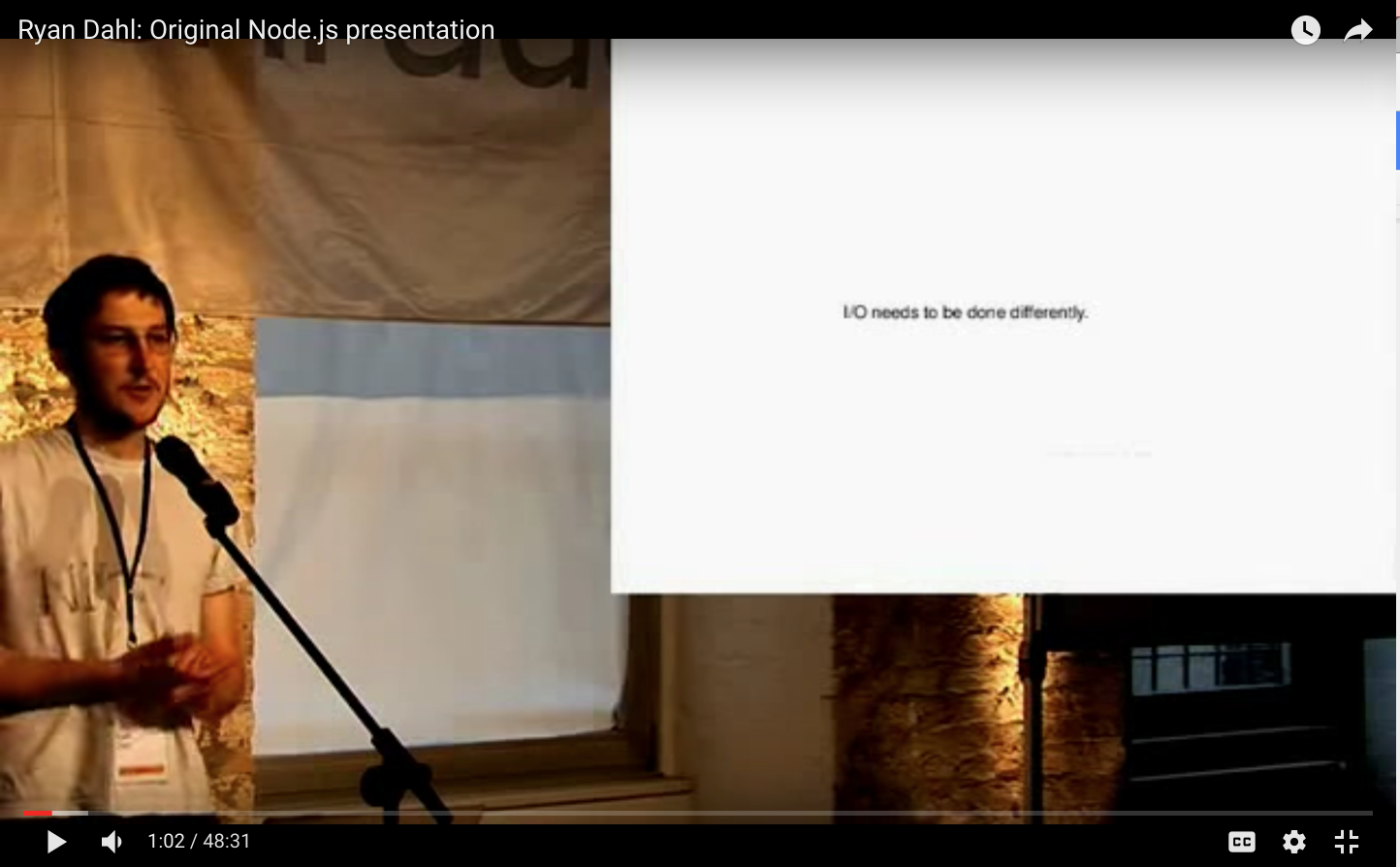
**iz predmeta Interakcija čovek računar**

|  |  |
| --- | --- |
| Mentor:  Doc. dr Milan Čabarkapa | Student:  Mihailo Joksimović  2015/202251 |

Beograd, 2017

# UVOD

*“I/O mora biti urađen drugačije. Način na koji ga koristimo je potpuno pogrešan. Ovo se mora promeniti”*. Ovo su reči kojima je Ryan Dahl, tvorac Nodejsa započeo prezentaciju na JSConf konferenciji 2009. godine.



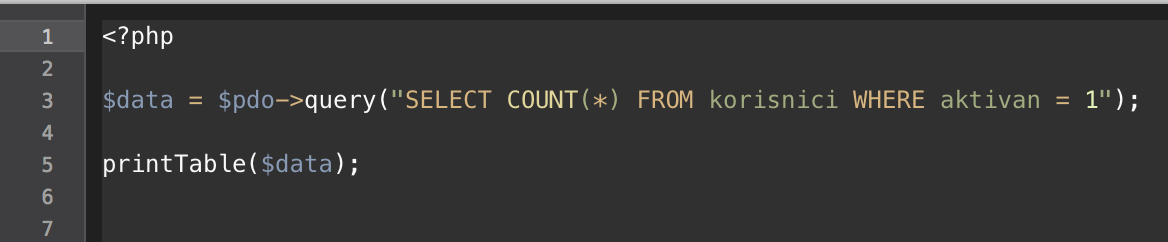
Slika 1. Ryan Dahl, JSConf 2009. godine

Nodejs je platforma otvorenog koda (eng. open source) na kojoj možemo pisati i pokretati serverske aplikacije korišćenjem dobro poznate JavaScript sintakse. Sama platforma kao i prevodilac (eng. interpreter) koda napisani su u C-u i koriste Googleovu V8 platformu za prevođenje i interpretiranje koda.

// Pawke : ovde treba da se zamene mesta paragrafima

Nodejs je nastao usled autorove frustracije načinom na koji funkcioniše ulaz/izlaz podataka i time što svaka ulazno/izlazna operacija (u daljem tekstu - I/O) blokira glavnu nit programa. Kako Ryan navodi, ovo je gubljenje vremena jer umesto čekanja da se završi blokirajuća I/O operacija, aplikacija može obrađivati druge zahteve i iskoristiti procesorsko vreme na mnogo bolje načine.

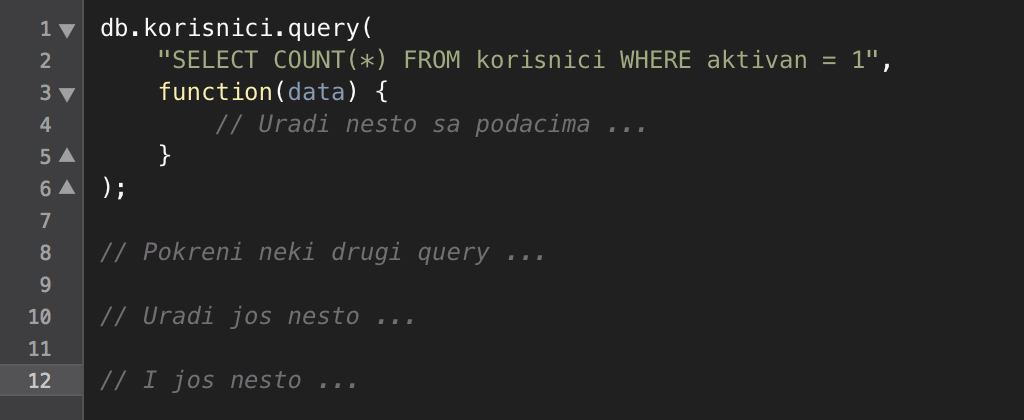
Kako bi smo bolje dočarali gore pomenuti problem, pogledajmo sledeći kod:



Slika 2. Sinhroni blokirajući kod

Problem sa ovim kodom je što aplikacija mora da sačeka da se upit završi i za to vreme nije u mogućnosti da uradi bilo šta drugo. Ovo je primer blokade usled čekanja na I/O.

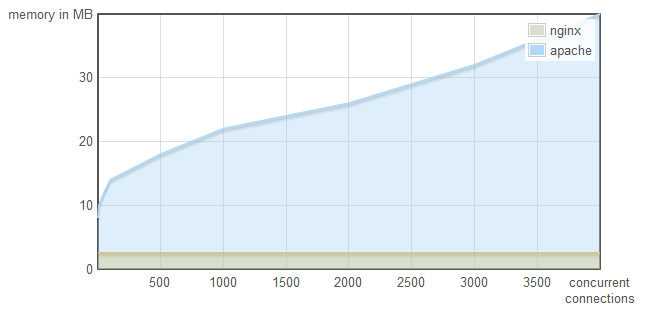
Sa druge strane, pogledajmo sledeći kod:



Slika 3. Asinhroni neblokirajući kod

Šta se promenilo? Na slici 3 možemo videti da će aplikacija pokrenuti upit i umesto da sačeka da se isti završi, ona će odraditi još dve ili tri dodatne operacije. Istovremeno, kada se I/O operacija završi, pokrenuće se povratna funkcija (eng. callback function) i procesiraće se rezultati.

Pogledajmo još jedan primer sa stvarnim rezultatima. U pitanju je poređenje Apache i nginx servera i dijagram opterećenosti u zavisnosti od broja paralelnih konekcija.



Slika 4. Dijagram opterećenosti kod Apache i nginx servera

Na horizontalnoj osi se nalazi broj klijenata povezanih na server, dok se na vertikalnoj nalazi opterećenost serverske memorije. Kao što možemo videti, kod Apachea se zahtevana memorija izuzetno povećava sa povećanjem broja konekcija što direktno utiče na opterećenje servera. Sa druge strane, možemo primetiti da se kod nginxa opterećenje uopšte ne menja sa povećanjem broja konekcija.

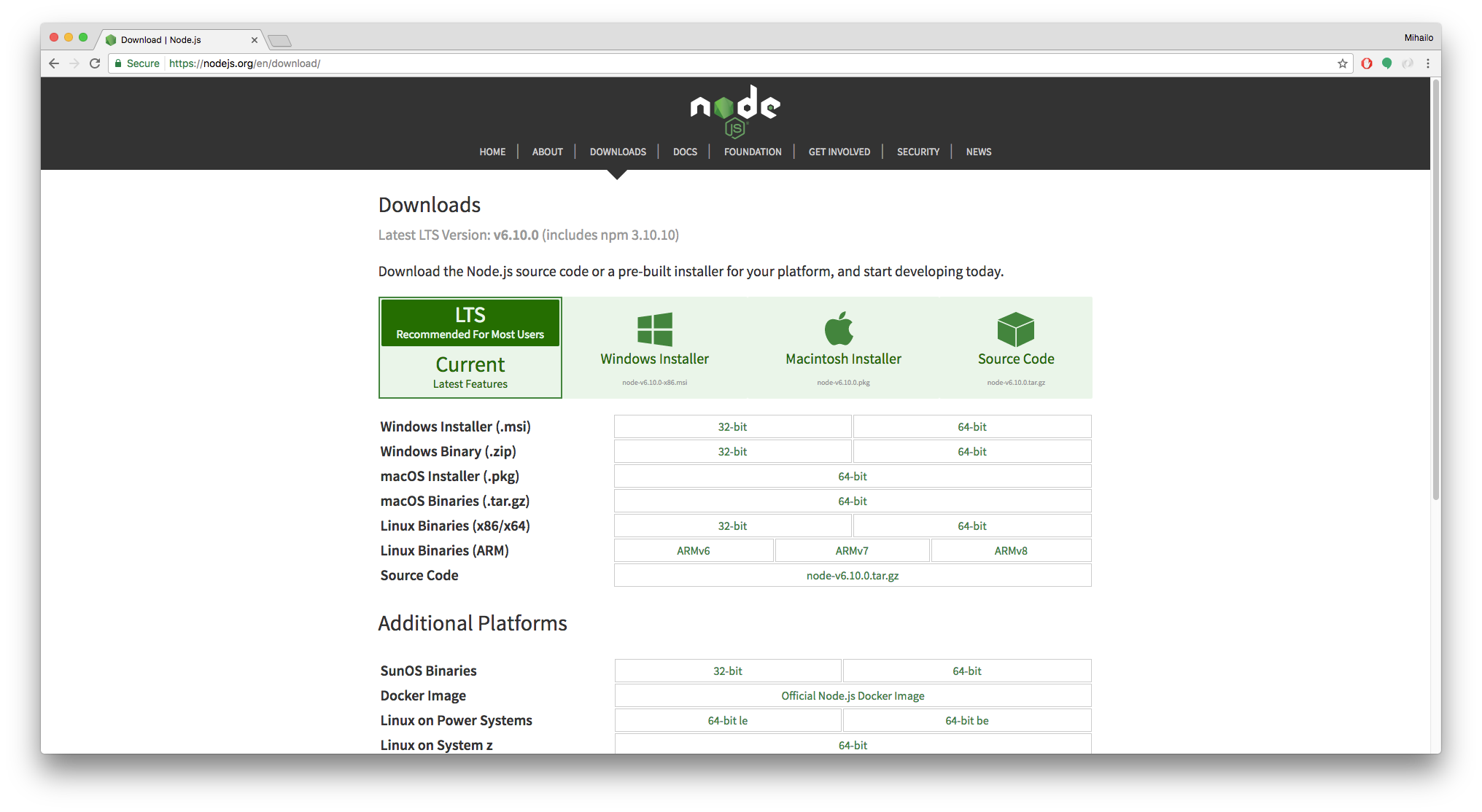
U čemu je razlika? Razlika je u pristupu i opsluživanju klijenata. Naime, Apache koristi po jednu nit (eng. Thread) za svakog klijenta, što nam govori da sa povećanjem broja klijenata direktno raste i opterećenje servera. Sa druge strane, nginx koristi arhitekturu vođenu događajima i umesto čekanja na I/O, on opslužuje druge klijente. Ovo je upravo način na koji Nodejs radi i upravo to će biti tema ovog rada.

Na kraju, korisno je pomenuti da je u trenutku pisanja ovog rada, aktuelna i stabilna verzija Nodejsa verzija 6.0. Za ovu verziju se trenutno garantuje durogorčna podrška (eng. LTS - Long Term Support) i ona je skoro u potpunosti kompatibilna sa najnovijom ECMA Script specifikacijom (ES6). Više informacija o samom Nodejsu i kompatibilnosti sa ES6 možete naći na sledećim adresama: <http://nodejs.org> i <http://node.green/>.

# Node.JS

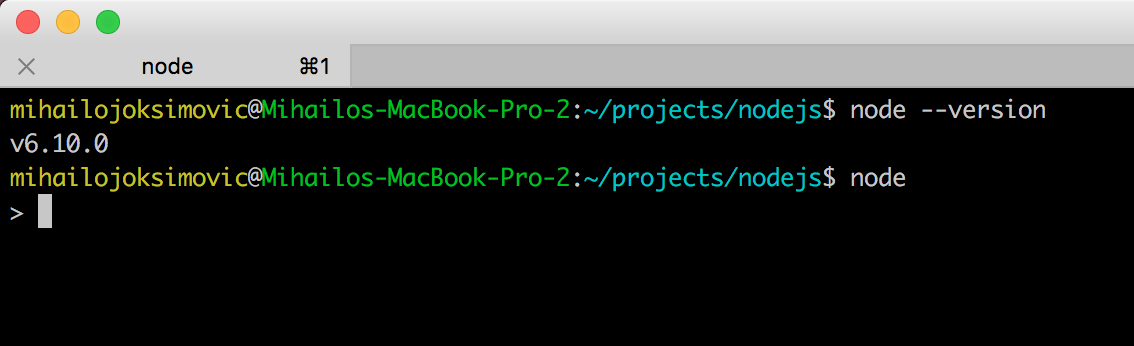
## Prvi koraci

Pre nego što počnemo sa korišćenjem Nodea, potrebno je instalirati prevodilac koda, kao i menadžer paketa - *npm* (eng. Node Package Manager). Kao i sve u Nodeu, i instalacija je prilično jednostavna. Sve što treba da uradimo je da posetimo sledeću stranicu <https://nodejs.org/en/download/>, izaberemo odgovarajući paket. downloadujemo ga i pokrenemo.



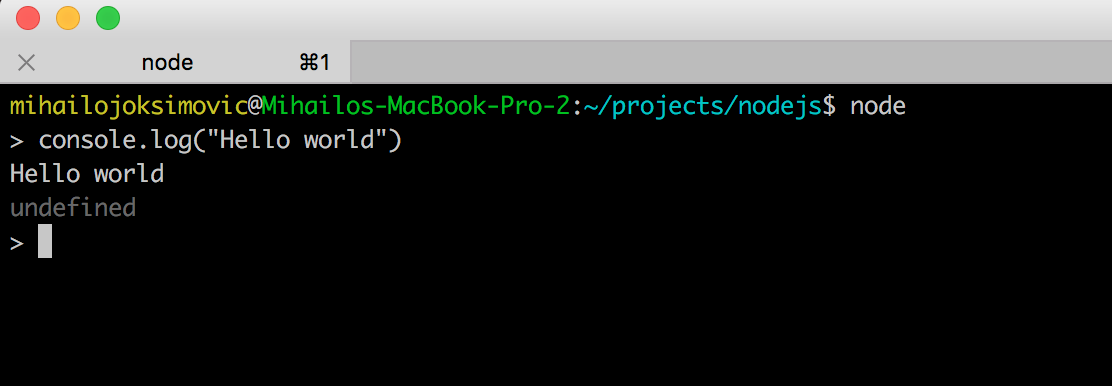
Slika 5. NodeJS download strana

Nakon downloada i uspešne instalacije, pokretanje naredbe node --version treba da nam prikaže verziju Nodea koju smo upravo instalirali. Takođe, pokretanjem naredbe node pojaviće nam se Nodeov interaktivni ekran u kome možemo izvršavati naredbe.



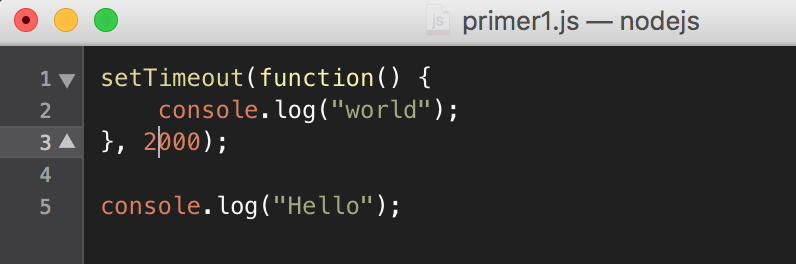
Slika 6. Node u konzoli

Kako tradicija nalaže, prvi korak pri učenju novog programskog jezika jeste ispisivanje teksta “Hello world” na ekranu. U Nodeu to izgleda ovako:



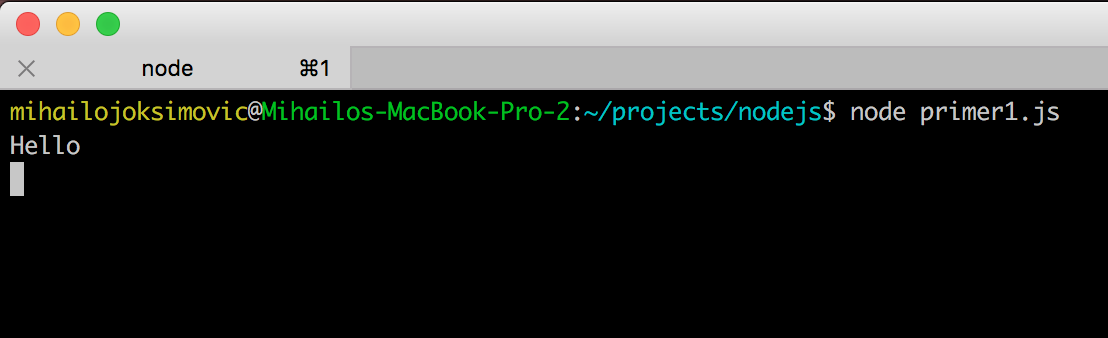
Prilično jednostavno, zar ne? Ono što možemo primetiti je da smo upravo iskoristili klasičnu JavaScript sintaksu, kao i dobro poznatu console komponentu kako bi smo na ekranu ispisali “Hello world” tekst. To je upravo ono što nam Node nudi - dobro poznata JavaScript sintaksa, kombinovana sa mnoštvom korisnih paketa i asinhronom arhitekturom vođenom događajima (ovime ćemo se detaljnije pozabaviti u kasnijim poglavljima).

Sada ćemo prikazati jedan bolji primer, koji koristi gore pomenutu arhitekturu vođenu događajima (eng. Event-driven architecture). Prvo ćemo otkucati kod koristeći dobro poznati editor teksta - TextMate:



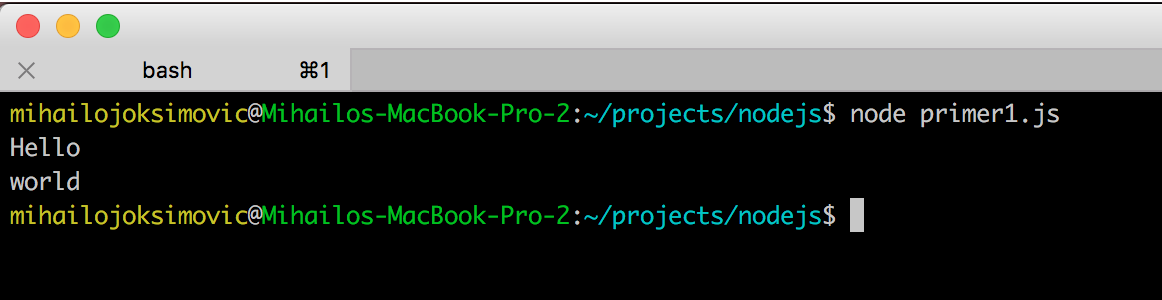
Slika 7. Primer jednostavnog, asinhronog koda vođenog događajima

Nakon izvršavanja skripta sa gore pomenutim kodom, na ekranu će se pojaviti sledeći sadržaj:



Slika 8. Izvršavanje primer1.js skripta

Dve sekunde kasnije, na ekranu će se prikazati sledeći sadržaj:

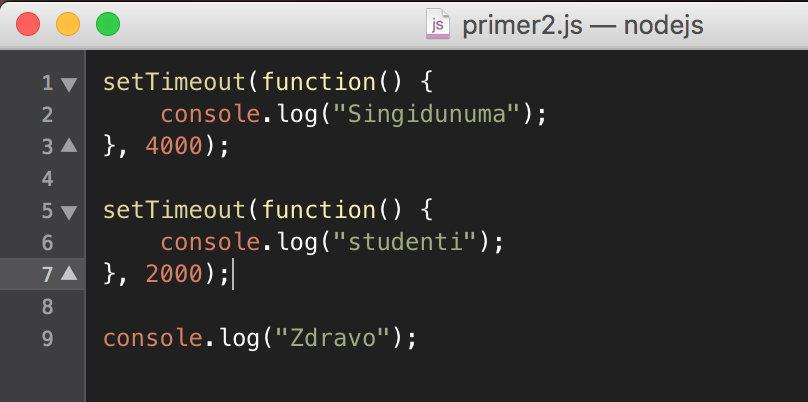


Slika 9. Kraj izvršavanja skripta

Iz ovog primera možemo primetiti nekoliko stvari:

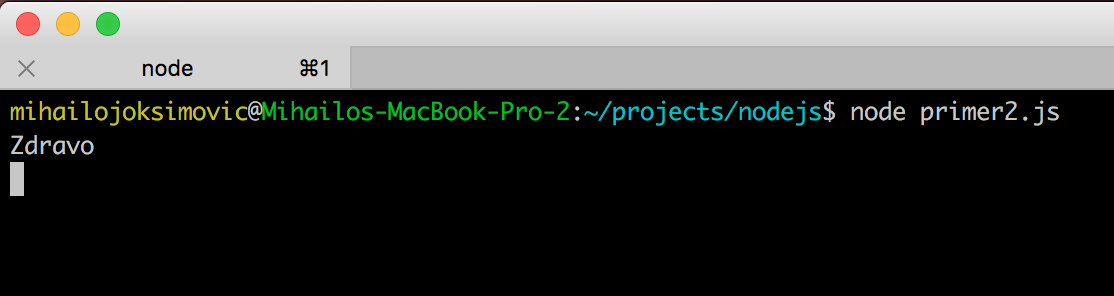
* Node koristi staru, poznatu JavaScript sintaksu
* Linija 5 se izvršila iako se kod sa linija 1-3 još uvek nije izvršio
* Dve sekunde kasnije, kada je “okinut” događaj, tekst sa linije 2 je ispisan na ekranu
* Nakon što su sve povratne (eng. callback) funkcije pozvane, izvršavanje programa je završeno

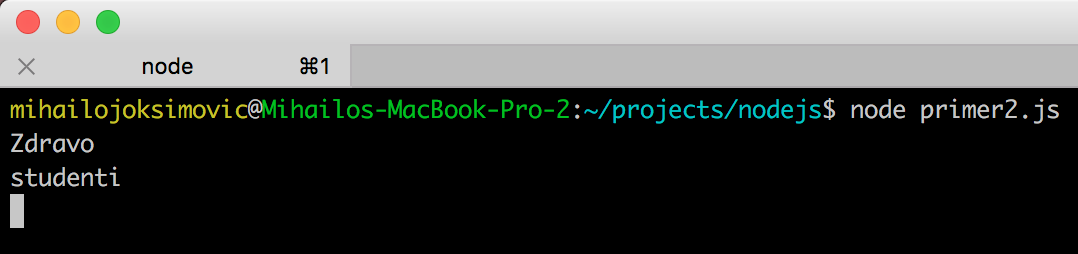
Pogledajmo još jedan primer:

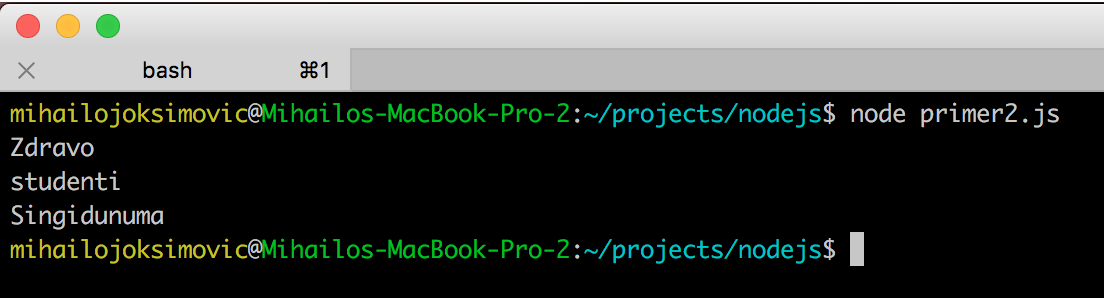


Slika 10. Kod skripta primer2.js

Da li možete da pretpostavite šta će biti izlaz ovog programa?







Slika 11. Izlaz programa nakon pokretanja, zatim 2s kasnije i 4s kasnije

Ako ste pretpostavili da će na ekranu biti ispisan tekst “Zdravo”, zatim dve sekunde kasnije “studenti” i na kraju četiri sekunde kasnije “Singidunuma” - čestitamo, bili ste u pravu!

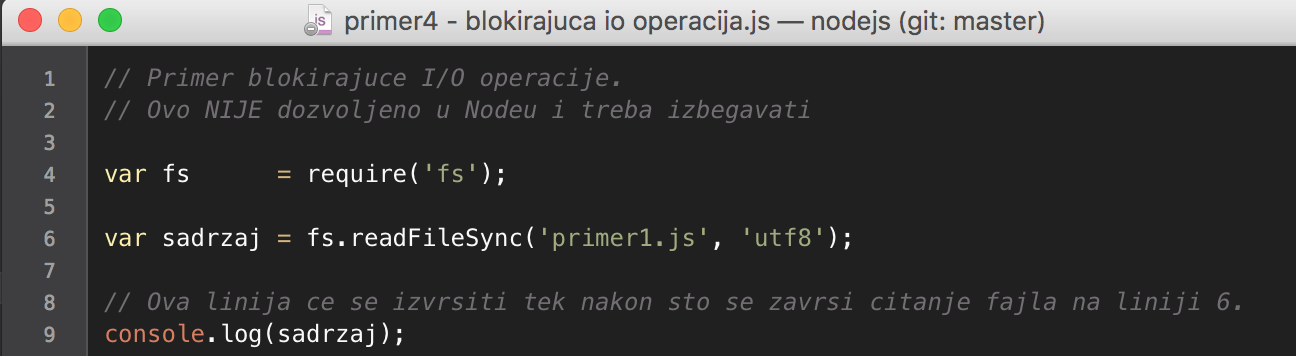
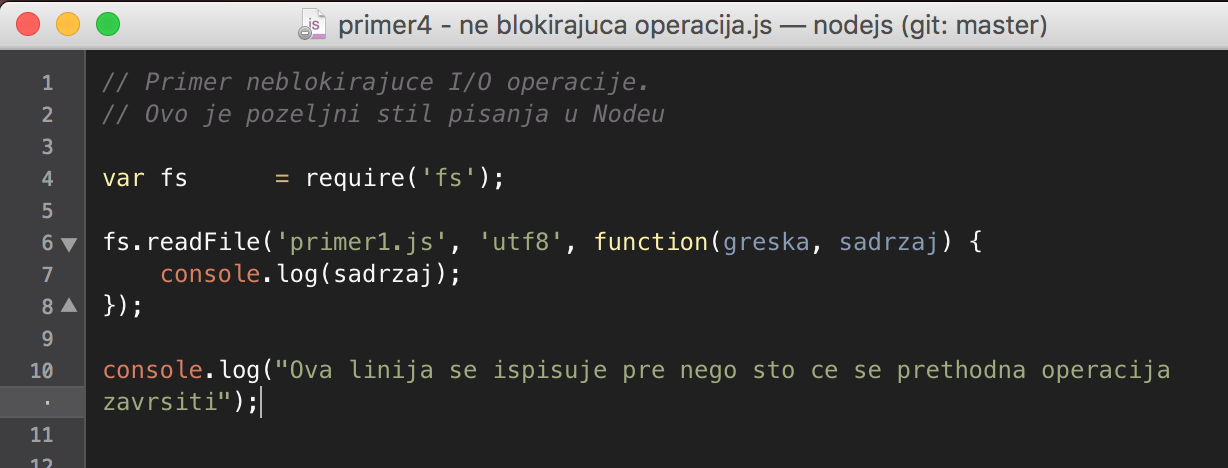
Možemo zaključiti da je za razliku od dobro poznatih jezika - C++, C#, Java, Python i drugih - Nodejs potpuno asinhron i vođen događajima. U daljem tekstu fokusiraćemo se na osnovne dizajn principe jezika, ukratko ćemo objasniti pojam arhitekture vođene događajima i daćemo primere nekoliko korisnih komponenti.

Sve primere koji su navedeni ovde, možete preuzeti na sledećoj strani:

<https://github.com/MihailoJoksimovic/introduction-to-nodejs-code-camples>

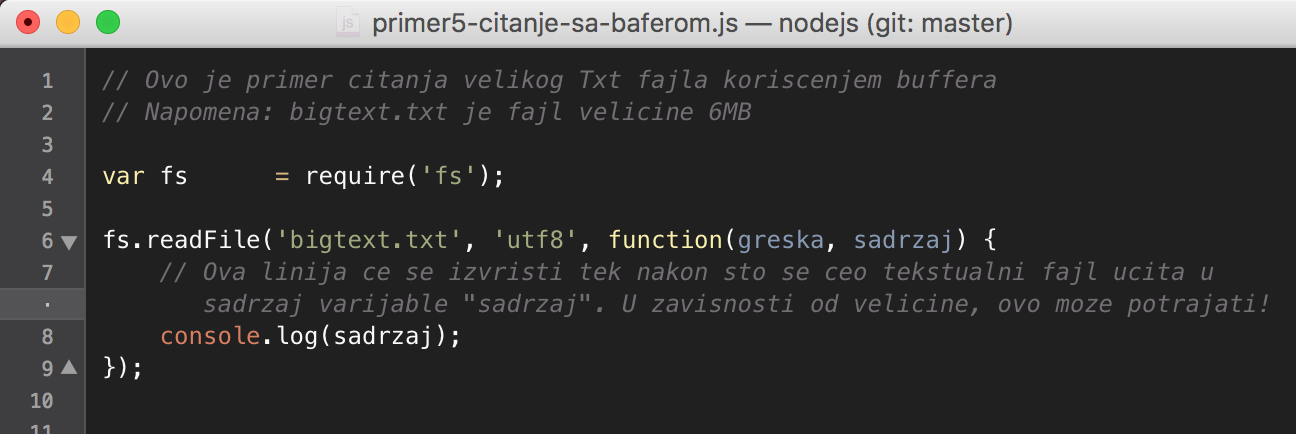
## Dizajn ciljevi

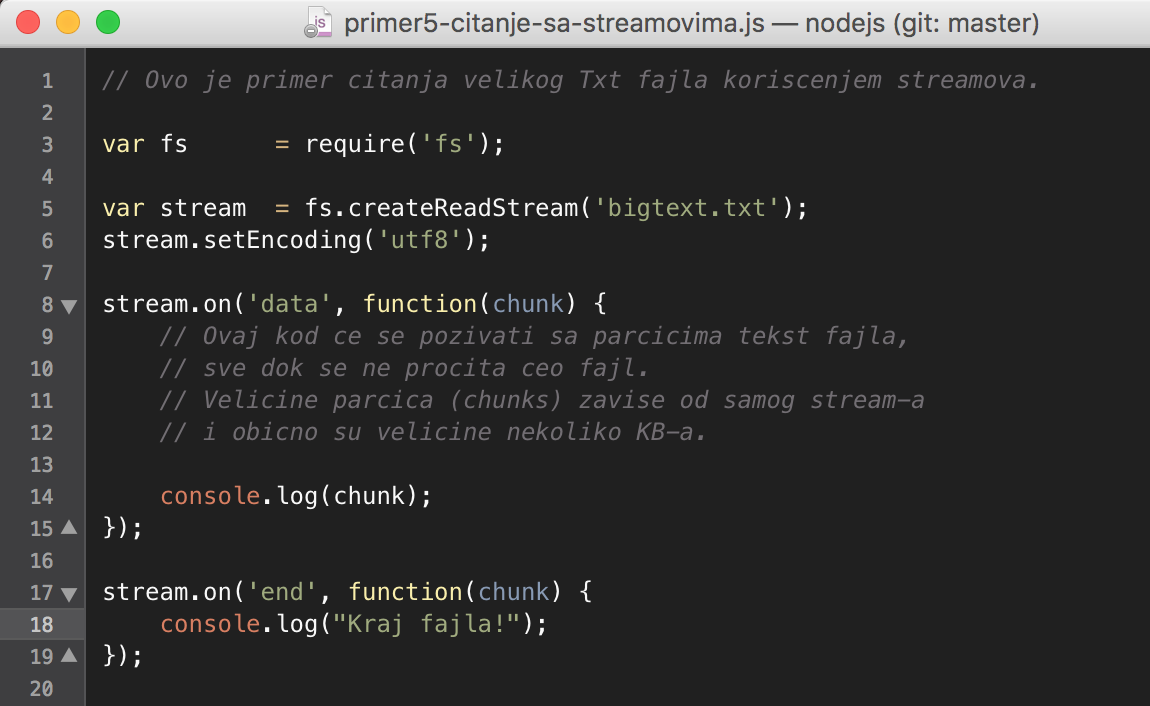
Kako bi smo bolje razumeli principe funkcionisanja, bitno je napomenuti glavne dizajn ciljeve kojima se Ryan Dahl vodio kada je napravio prvu verziju Node-a:

1. Blokirajuće I/O operacije se **moraju izbegavati** - umesto toga, **moramo koristiti povratne funkcije** kako bi smo pročitali podatke sa diska, mreže ili nekog drugog procesa.  
  
  


Slika 12. Primer blokirajuće i neblokirajuće I/O operacije

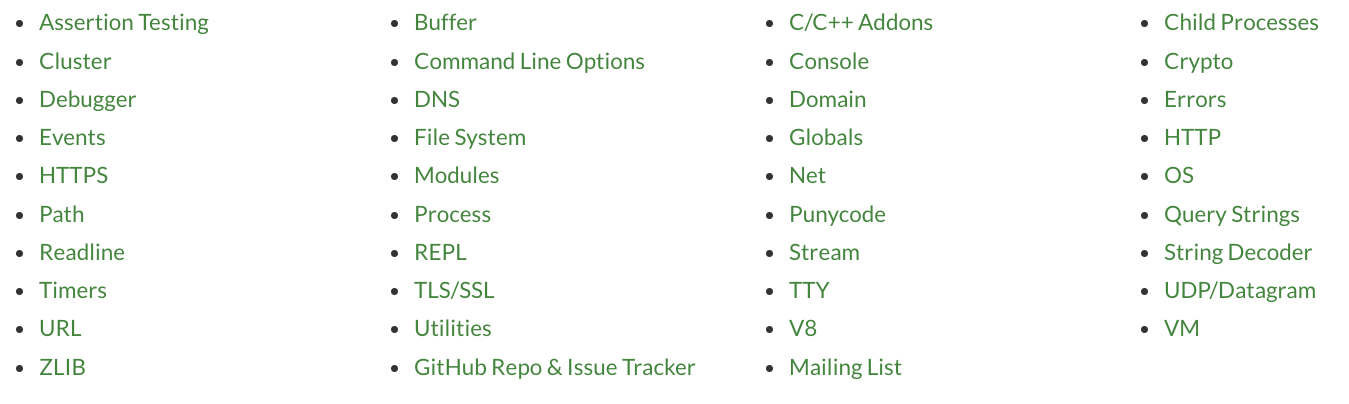
2. **Strimovi** **(eng. streams) umesto bafera (eng. buffer)** - uvek koristite strimove umesto bafera. Na ovaj način štedimo resurse i dobijamo najbolje performanse iz Nodea.





Slika 13. Citanje koriscenjem bafera i strima

3. Ugrađena **podrška za najkorišćenije servise** - jedna od osnovnih filozofija Nodejsa jeste da podrška za najkorišćenije servise (TCP, HTTP, DNS) mora biti ugrađena u jezgro jezika.



Slika 14. Lista svih ugrađenih modula

4. Jezički interfejs (eng. API - Application Programming Interface) **mora biti dobro poznat** - sintaksu moraju razumeti i frontend developeri i hakeri!

5. Jezik **mora biti nezavistan od platforme** (eng. platform independant) - Node se može pokretati na svim popularnim platformama (Windows, Linux, FreeBSD, ...

## 