

# Node.js

U jednoj od prethodnih lekcija spomenut je pojam izomorfih, odnosno univerzalnih aplikacija. Kreiranje izomorfih JavaScript aplikacija postalo je moguće sa pojmom Node.js izvršnog okruženja. Tako su stvorenii uslovi da JavaScript izđe iz okvira web pregledača i svoje mesto pronađe i na serveru.

Ono što nas u ovom kursu posebno zanima jeste primena izvršnog okruženja Node.js kada je u pitanju frontend razvoj. Naime, iako primarno namenjen za izvršavanje JavaScript logike na serveru, Node.js je u potpunosti promenio način na koji se kreiraju moderne JavaScript aplikacije. Tako su Node.js i njegov menadžer paketa (npm) postali neizostavni alati u arsenalu frontend programera.

U lekciji pred vama prvo će biti prikazano kako se obavlja instalacija izvršnog okruženja Node.js i npm menadžera paketa. Nakon toga biće prikazane osnove korišćenja takvih alata, čime će biti postavljeni temelji za njihovo korišćenje u lekcijama koje slede.

## Šta je Node.js?

Node.js je JavaScript izvršno okruženje otvorenog koda, koje omogućava da se JavaScript izvršava izvan web pregledača. Zasniva se na V8 JavaScript izvršnom okruženju Chrome web pregledača. Node.js se primarno koristi:

- za izvršavanje JavaScript koda na serveru i
- za izvršavanje alata koji olakšavaju i ubrzavaju frontend razvoj.

Node.js predstavlja svojevrsnu realizaciju, odnosno otelotvorene *JavaScript everywhere* paradigmme, kojom se teži postizanju web razvoja koji podrazumeva korišćenje jednog programskog jezika i na klijentu, ali i na serveru.

### Pitanje

Node.js je izvršno okruženje koje omogućava izvršavanje programskog koda, jezika:

- a) **JavaScript**
- b) HTML
- c) CSS
- d) Python

### Objašnjenje:

*Node.js je JavaScript izvršno okruženje otvorenog koda, koje omogućava da se JavaScript izvršava izvan web pregledača.*

## Instalacija Node.js-a

Pre nego što budemo u mogućnosti da iskoristimo mogućnosti izvršnog okruženja Node.js, njega je neophodno instalirati. Node.js je dostupan za sve relevantne platforme, a preuzimanje potrebnog instalacionog fajla je moguće obaviti sa sledeće adrese:

<https://nodejs.org/en/download/>

Na stranici za preuzimanje Node.js-a potrebno je odabrati varijantu koja odgovara osobinama vašeg operativnog sistema (slika 3.1).

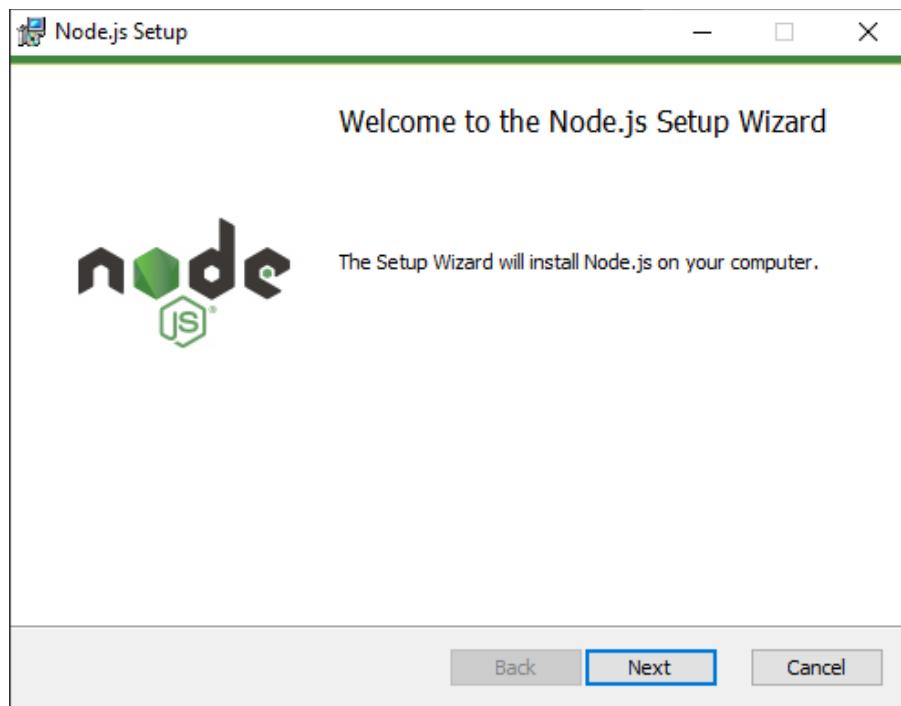
The screenshot shows the Node.js Downloads page. At the top, there's a navigation bar with links: HOME, ABOUT, DOWNLOADS (which is highlighted), DOCS, GET INVOLVED, SECURITY, and NEWS. Below the navigation bar, a large green button labeled "LTS Recommended For Most Users" is visible. To its right, a smaller green button labeled "Current Latest Features" is shown. Under the LTS section, there are icons for Windows Installer (Windows logo) and macOS Installer (apple logo). Under the Current section, there is an icon for Source Code (cube). Below these sections, there are two tables of download links:

	32-bit	64-bit
Windows Installer (.msi)	<a href="#">node-v12.17.0-x86.msi</a>	<a href="#">node-v12.17.0-x64.msi</a>
Windows Binary (.zip)	<a href="#">node-v12.17.0-win32.zip</a>	<a href="#">node-v12.17.0-win64.zip</a>
macOS Installer (.pkg)	<a href="#">node-v12.17.0-macos.pkg</a>	<a href="#">node-v12.17.0-macos64.pkg</a>
macOS Binary (.tar.gz)	<a href="#">node-v12.17.0-macos.tar.gz</a>	<a href="#">node-v12.17.0-macos64.tar.gz</a>
Linux Binaries (x64)		<a href="#">node-v12.17.0-linux-x64.tar.gz</a>
Linux Binaries (ARM)	<a href="#">node-v12.17.0-linux-armv7.tar.gz</a>	<a href="#">node-v12.17.0-linux-armv8.tar.gz</a>
Source Code		

A pink circle highlights the "Downloads" section of the page, and a pink arrow points from the text "Slika 3.1. Node.js instalaciona stranica" to this highlighted area.

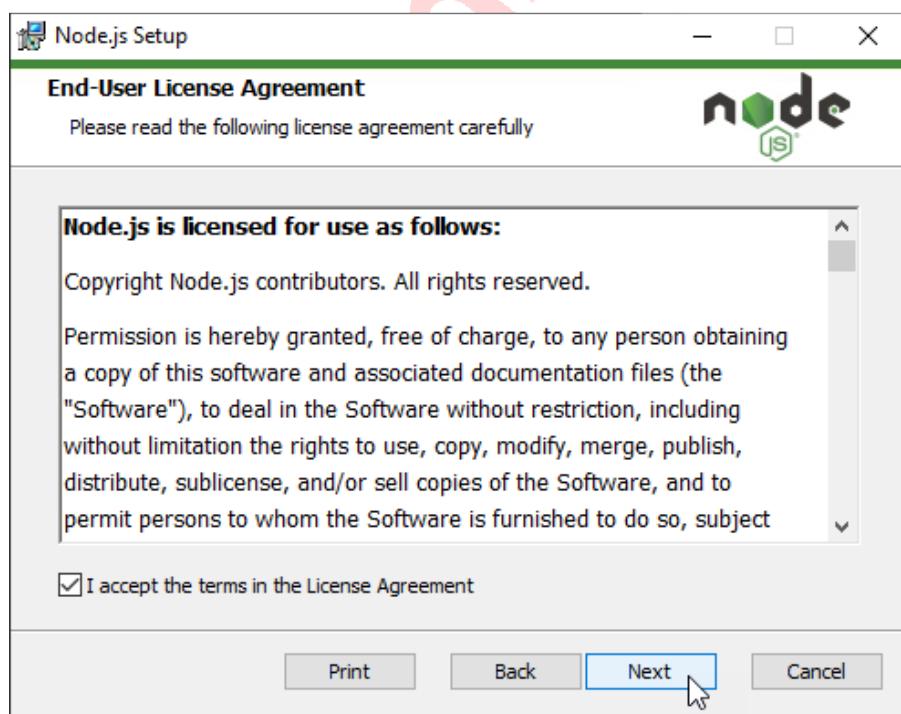
*Slika 3.1. Node.js instalaciona stranica*

Preuzeti instalacioni fajl je potrebno pokrenuti i pratiti uputstva (slika 3.2).

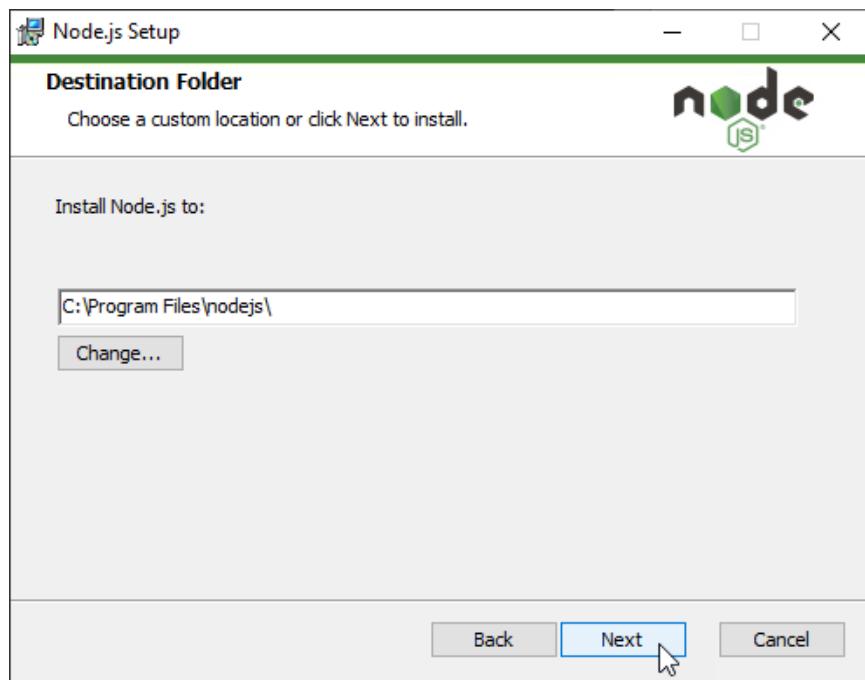


Slika 3.2. Početak instalacije Node.js-a

Prvih nekoliko koraka u konfigurisanju instalacije su tipični i uobičajeni (slike 3.3. i 3.4).

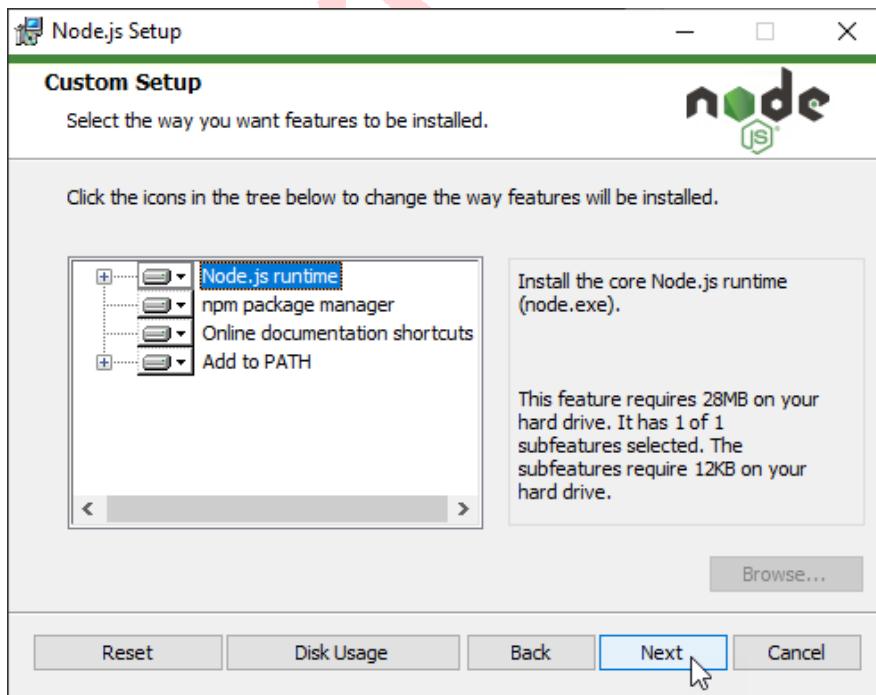


Slika 3.3. Prihvatanje uslova licenciranja



Slika 3.4. Definisanje instalacione putanje

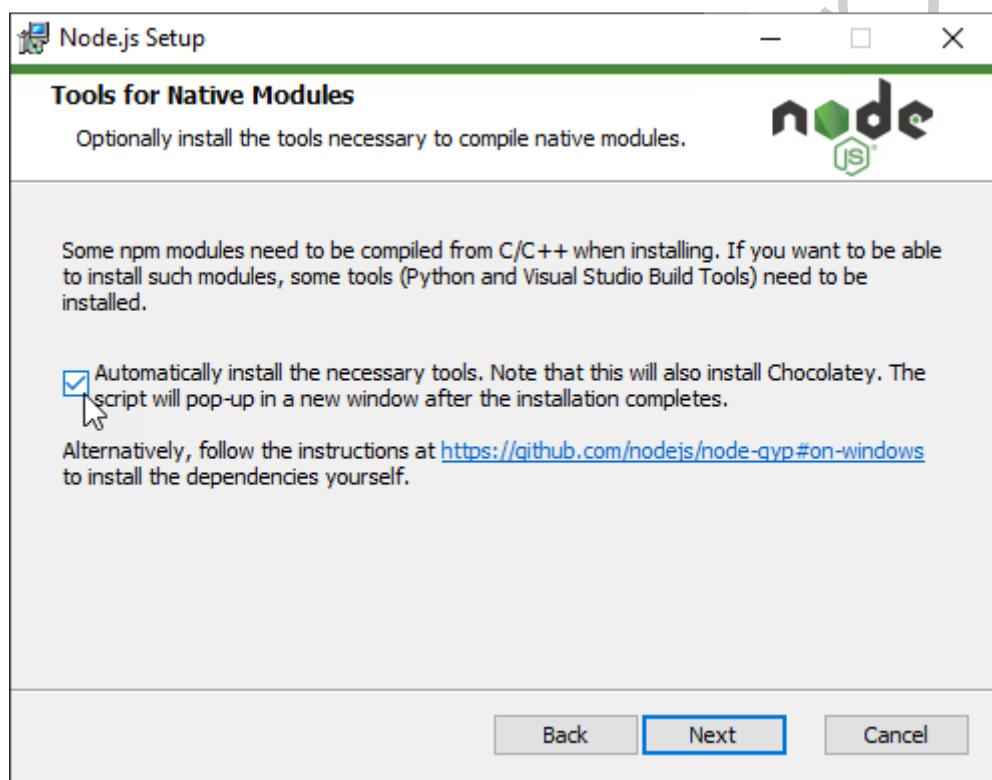
Veoma bitan segment instalacije Node.js-a odnosi se na mogućnost instalacije Node.js menadžera paketa (npm, skraćeno) i postavljanja instalacione putanje unutar sistemske Path promenljive (slika 3.5).



Slika 3.5. Odabir funkcionalnosti koje će biti instalirane

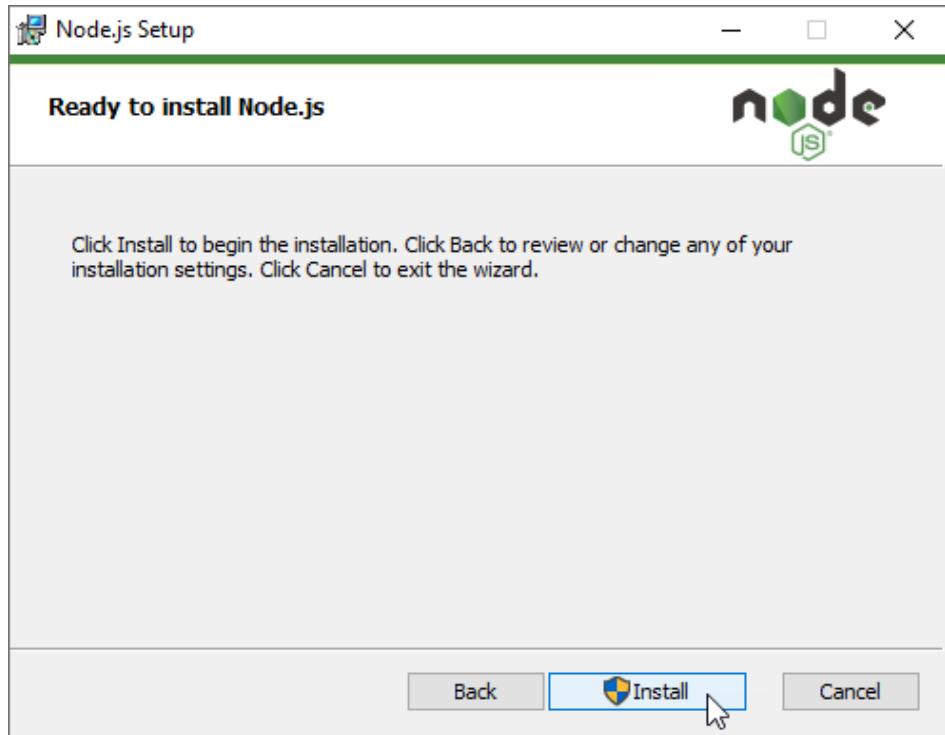
Node.js se podrazumevano instalira zajedno sa npm menadžerom paketa, a tokom instalacije se obavlja i postavljanje putanje u sistemsku Path promenljivu. To možete videti i na slici 3.5, gde je na levoj polovini prozora prikazana lista svih stavki koje će biti obavljene tokom instalacije Node.js-a. U ovom trenutku nije potrebno ni na koji način menjati podrazumevane postavke, s obzirom na to da će nam npm menadžer paketa biti potreban u nastavku.

Sledeći korak instalacije odnosi se na instalaciju određenih alata koji omogućavaju korišćenje Node.js modula koji su pisani korišćenjem C/C++ jezika. S obzirom na to da je takve module neophodno kompajlirati kako bi mogli da se koriste, za obavljanja kompajliranja je potrebno imati odgovarajuće alate, a Node.js prozor za instaliranje nam omogućava da u ovom trenutku obavimo instaliranje upravo takvih alata (slika 3.6).

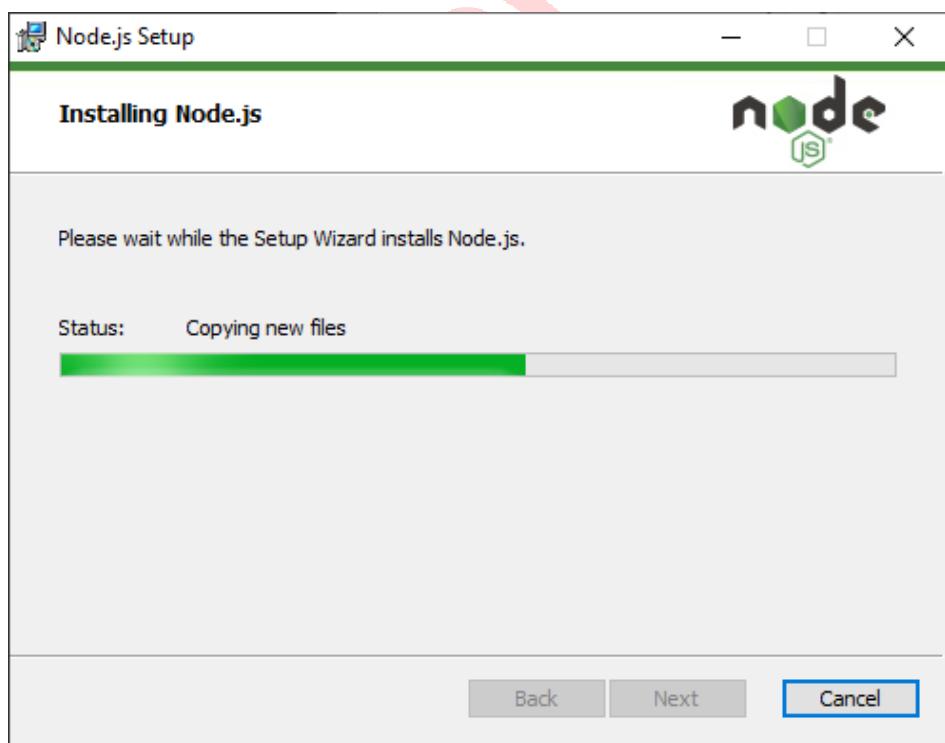


*Slika 3.6. Opcija kojom je moguće instalirati dodatne alate za kompajliranje Node.js modula napisanih C/C++ jezicima*

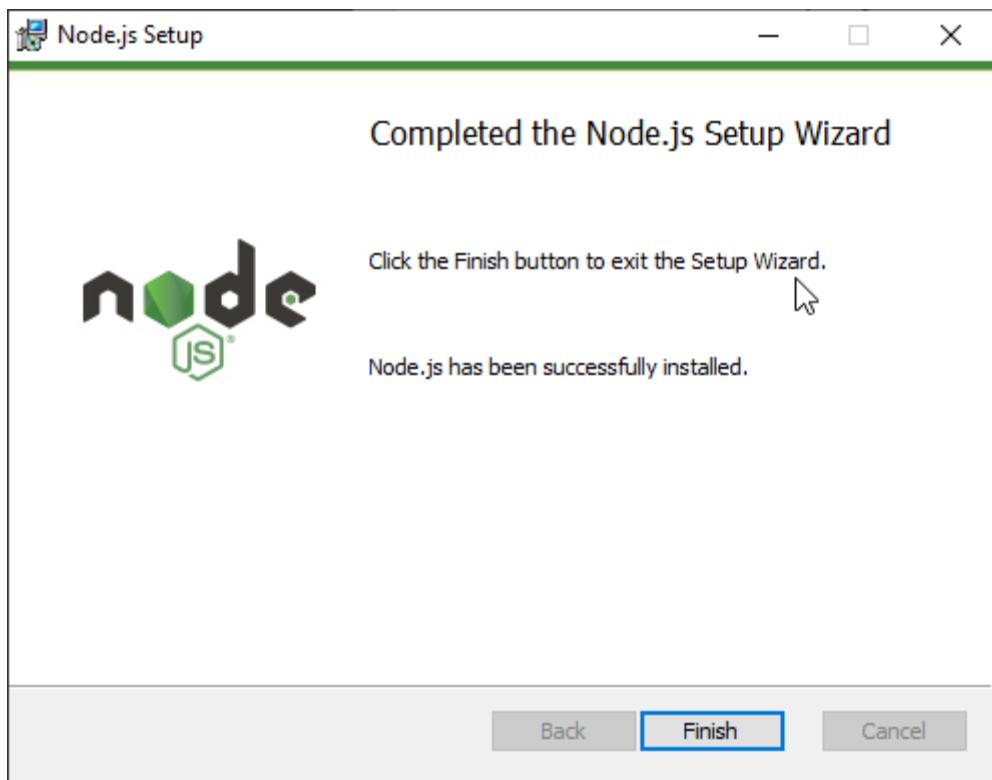
Budući da mi u ovom kursu nećemo detaljno obrađivati Node.js, pošto prevaziđa naš okvir interesovanja u ovom trenutku, vi možete, ali i ne morate da instalirate ove alate. U svakom slučaju, alate za kompajliranje uvek možete instalirati naknadno.



Slika 3.7. Pokretanje instalacije Node.js-a



Slika 3.8. Instalacija Node.js-a



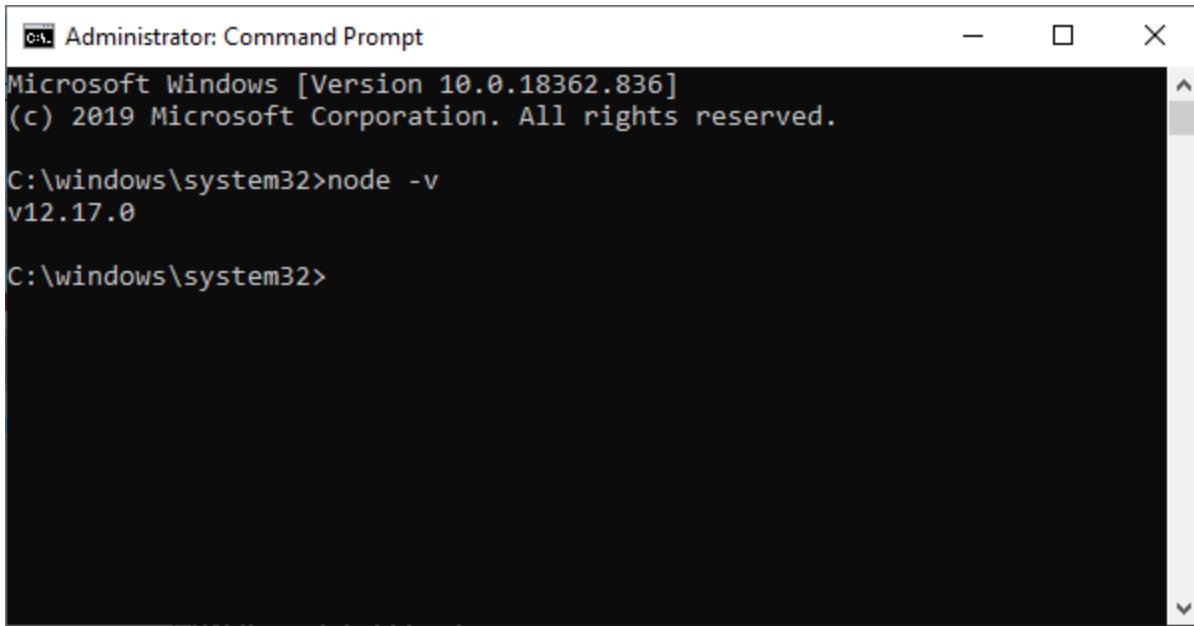
Slika 3.9. Uspešan završetak instalacije Node.js-a

## Provera instalacije

Kako biste se uverili da je instalacija Node.js-a obavljena uspešno, dovoljno je da otvorite konzolu (Command Prompt ili PowerShell, na operativnim sistemima Windows) ili Terminal na MacOS-u i da pokrene sledeću komandu:

```
node -v
```

Ovakvu komandu je moguće pokrenuti sa bilo koje putanje, zato što je nešto ranije tokom instalacije putanja na kojoj je Node.js instaliran automatski smeštena unutar sistemske Path promenljive.



```
C:\Administrator: Command Prompt
Microsoft Windows [Version 10.0.18362.836]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\windows\system32>node -v
v12.17.0

C:\windows\system32>
```

Slika 3.10. Provera raspoloživosti Node.js-a korišćenjem Command Prompta na operativnom sistemu Windows

## Razlike između Node.js-a i web pregledača

Pre nego što pređemo na pisanje prvog JavaScript koda koji će izvršiti izvršno okruženje Node.js, neophodno je razumeti osnovnu razliku između izvršavanja JavaScript koda unutar web pregledača i Node.js izvršnog okruženja.

Iako oba okruženja razumeju JavaScript jezik, pisanje frontend logike znatno je drugačije od pisanja koda koji se posredstvom Node.js-a izvršava na serveru. Za početak, neophodno je razumeti da Node.js ne poseduje različite [Web API-je](#) koje web pregledači izlažu na korišćenje JavaScript kodu koji se nalazi na frontendu. Stoga, prilikom rada sa Node.js-om nema DOM, Location, History, Cookie i ostalih aplikativnih programskih interfejsa koji su svojstveni za web pregledače. Ukoliko malo razmislite, takvo nešto je potpuno očekivano, s obzirom na to da je Node.js prevashodno kreiran kako bi omogućio izvršavanje JavaScript koda na serveru, a predmet interesovanja serverske logike nije k-kod koji barata prezentacionom logikom.

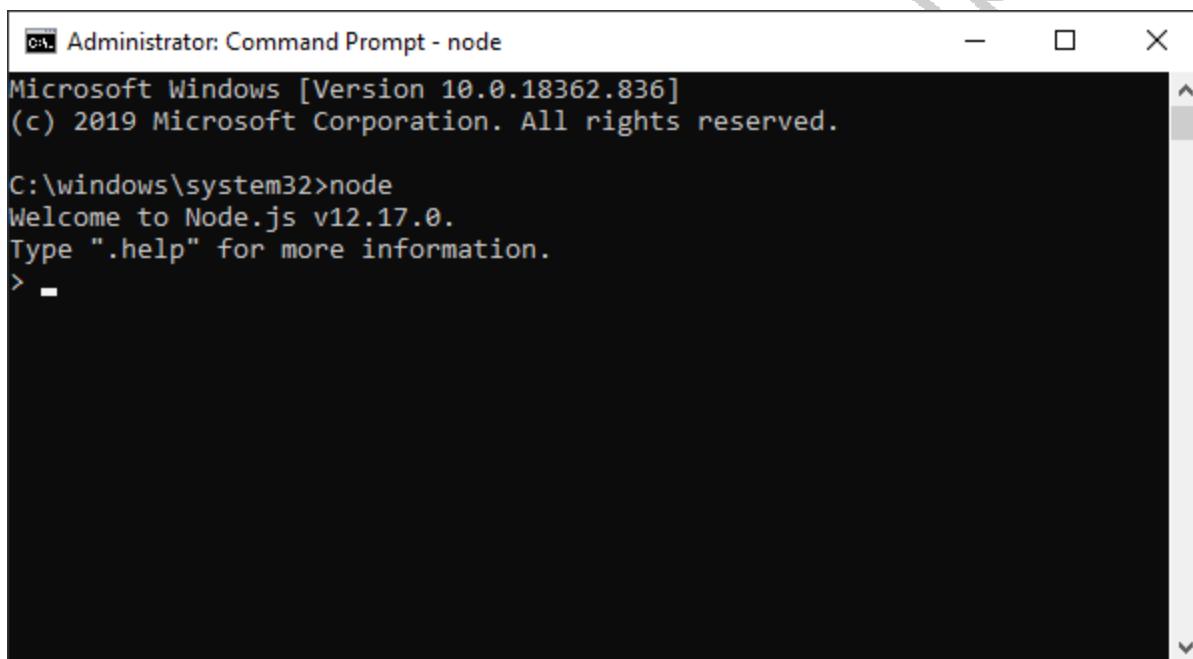
S druge strane, Node.js razume sve JavaScript jezičke elemente koje je moguće koristiti i prilikom pisanja frontend koda: izraze, tipove, promenljive, funkcije, petlje, nizove, literale, Promise, Closure itd... S obzirom na to da se zasniva na najpopularnijem izvršnom okruženju koje koristi i većina modernih web pregledača, unutar Node.js-a se redovno implementiraju nove jezičke funkcionalnosti definisane [ECMAScript](#) specifikacijom.

Još jedna značajna razlika između Node.js-a i web pregledača odnosi se na način na koji Node.js rukuje [modulima](#). Budući da JavaScript jezik dugo nije posedovao izvorni način za kreiranje i rukovanje modulima, takva funkcionalnost je unutar Node.js-a implementirana korišćenjem CommonJS biblioteke. U međuvremenu je pojam modula izvorno uvršten i u jezik JavaScript, ali Node.js i dalje koristi CommonJS modul.

## Korišćenje Node.js-a pomoću konzole

Sastavni deo Node.js-a jeste komandni alat koji omogućava rukovanje izvršnim okruženjem uz korišćenje konzole, odnosno terminala. Takav alat ogleda se u izvršnom fajlu node čija je putanja tokom instalacije smeštena unutar sistema promenljive Path, te stoga mi njega možemo da koristimo jednostavnim kucanjem reči node unutar konzole. Node alat omogućava izvršavanje koda koji je smešten unutar nekog .js fajla, ali i izvršavanje koda koji se direktno prosleđuje node komandnom alatu.

Kucanjem node komande unutar konzole ili terminala ulazi se u poseban režim korišćenja Node.js-a koji se naziva **REPL** (engl. **R**ead-**E**val-**P**rint-**L**oop). U pitanju je mod koji je moguće koristiti za pisanje jednostavnih JavaScript naredaba, te na taj način testirati izvršno okruženje (slika 3.11).

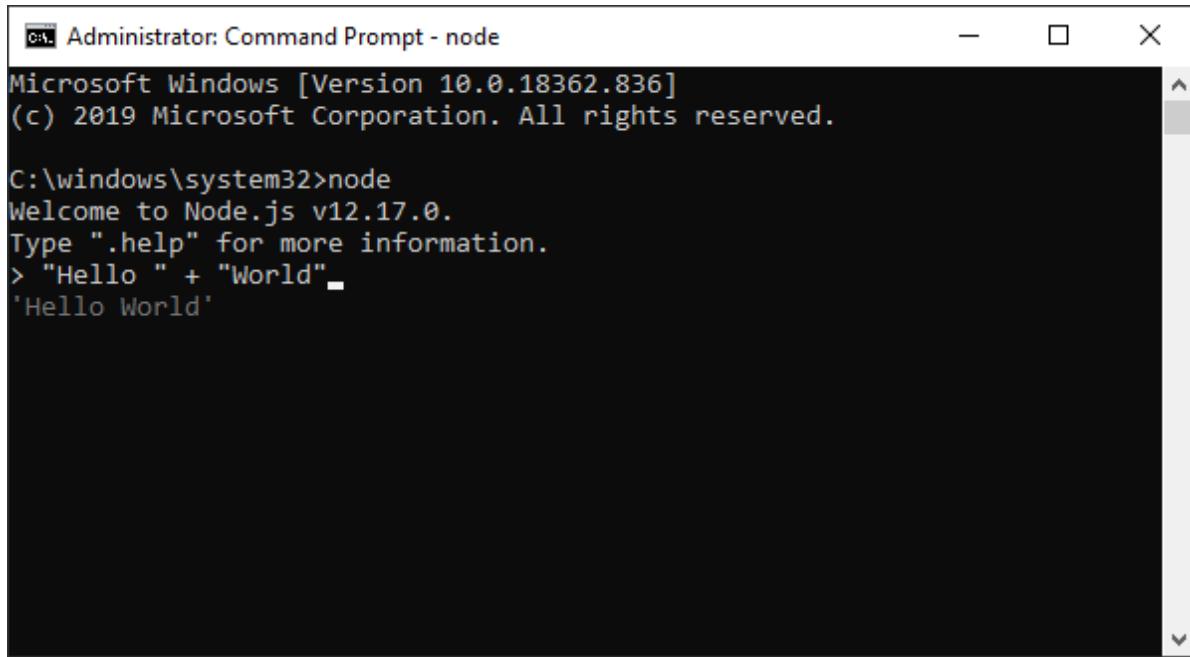


```
Administrator: Command Prompt - node
Microsoft Windows [Version 10.0.18362.836]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\windows\system32>node
Welcome to Node.js v12.17.0.
Type ".help" for more information.
>
```

Slika 3.11. Aktiviranje REPL moda

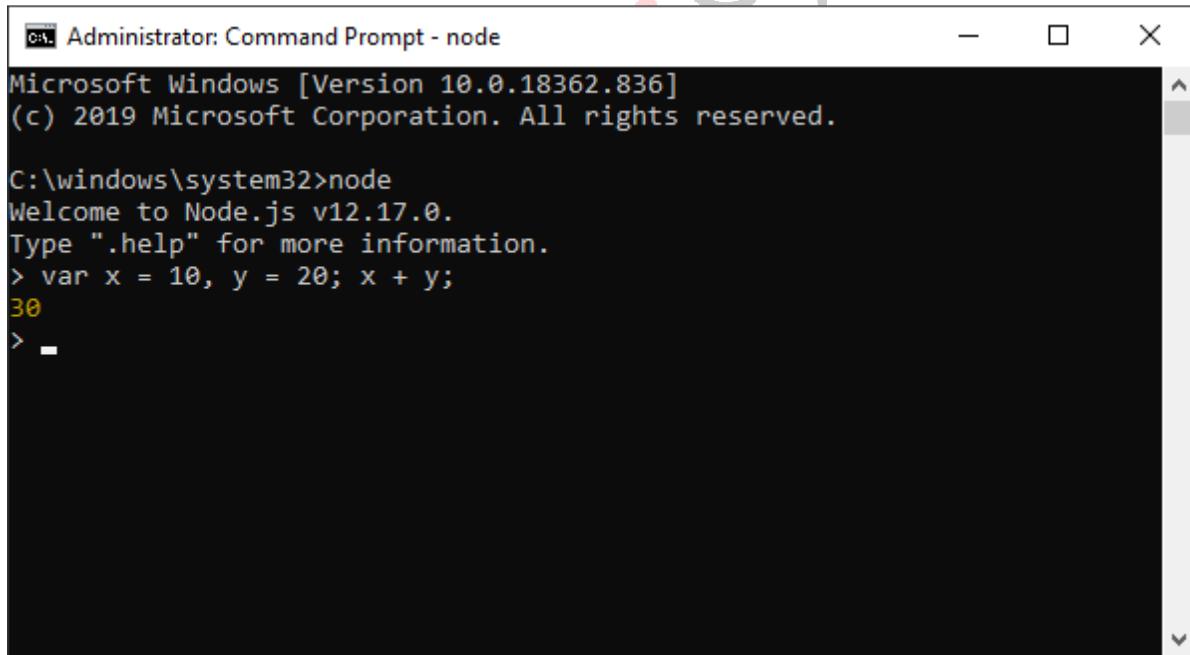
Na slici 3.11. se može videti aktivacija konzolnog moda koji omogućava direktno pisanje JavaScript koda. Stoga je sada moguće uraditi nešto kao na slikama 3.12. i 3.13.



```
Administrator: Command Prompt - node
Microsoft Windows [Version 10.0.18362.836]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\windows\system32>node
Welcome to Node.js v12.17.0.
Type ".help" for more information.
> "Hello " + "World"
'Hello World'
```

Slika 3.12. Primer direktnog izvršavanja JavaScript koda unutar konzole (1)



```
Administrator: Command Prompt - node
Microsoft Windows [Version 10.0.18362.836]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\windows\system32>node
Welcome to Node.js v12.17.0.
Type ".help" for more information.
> var x = 10, y = 20; x + y;
30
> -
```

Slika 3.13. Primer direktnog izvršavanja JavaScript koda unutar konzole (2)

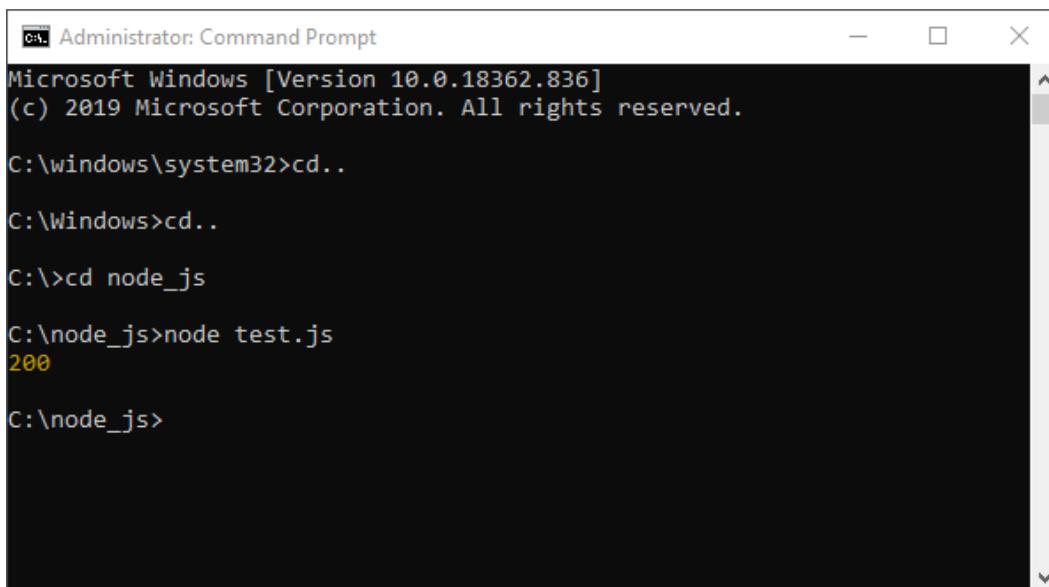
#### Napomena

Iz REPL moda može se izaći korišćenjem komande `.exit`.

Komanda `node` se može koristiti i za izvršavanje JavaScript koda koji je smešten unutar nekog `.js` fajla. Na primer, evo kako može izgledati sadržaj jednog JavaScript fajla:

```
var x = 10;
var y = 20;
console.log(x*y);
```

Ukoliko se ovakav kod nalazi unutar fajla sa nazivom `test.js` unutar foldera `C:/node_js`, on se korišćenjem izvršnog okruženja Node.js može izvršiti kao na slici 3.14.



The screenshot shows a Windows Command Prompt window titled "Administrator: Command Prompt". The window displays the following text:  
Microsoft Windows [Version 10.0.18362.836]  
(c) 2019 Microsoft Corporation. All rights reserved.  
  
C:\windows\system32>cd..  
  
C:\Windows>cd..  
  
C:\>cd node\_js  
  
C:\node\_js>node test.js  
200  
  
C:\node\_js>

Slika 3.14. Izvršavanje JavaScript koda definisanog unutar fajla

### Console objekat

Iz upravo prikazanog primera možete videti da i Node.js izvršno okruženje omogućava ispisivanje poruka unutar konzole, na identičan način kao što je to moguće postići i unutar web pregledača. Ipak, u web pregledačima `console` je svojstvo globalnog `Window` objekta, a nešto ranije je rečeno da `Window` objekat ne postoji unutar Node.js-a. Pa odakle onda mogućnost korišćenja `Console` objekta?

`Console` objekat je moguće koristiti zato što i izvršno okruženje Node.js poseduje takav objekat, sa sličnim skupom funkcionalnosti kao i istoimeni objekat unutar web pregledača. Unutar Node.js-a `Console` je globalni objekat, pa je upravo to razlog zbog koga smo mi u prethodnom primeru njega mogli iskoristiti na prikazani način. Kao što je rečeno, Node.js `Console` objekat poseduje sličan skup funkcionalnosti kao i objekat istog naziva unutar web pregledača:

- `console.log()` – funkcija za štampu poruka uopštenog tipa,
- `console.info()` – funkcija za štampu informacija,
- `console.error()` – funkcija za štampu grešaka,
- `console.warn()` – funkcija za štampu upozorenja.

## Kreiranje prvog Node.js HTTP servera

Svakako najpopularniji oblik korišćenja izvršnog okruženja Node.js jeste mogućnost pisanja serverske logike upotrebom JavaScript jezika. Osnovni oblik koda koji će korišćenjem Node.js izvršnog okruženja stvoriti HTTP server izgleda ovako:

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

const port = 3000;
const hostname = '127.0.0.1';

const server = http.createServer((req, res) => {
    //handle received request...
})

server.listen(port, hostname, () => {
    console.log(`Server running at ${hostname}:${port}`)
})
```

Kreiranje Node.js HTTP servera započinje pozivanjem jedne posebne funkcije `require()`, kojom se u tekući dokument uključuje jedan Node.js modul.

### Node.js moduli

Funkcionalnosti izvršnog okruženja Node.js podeljene su na module. Moduli omogućavaju logičko, ali i fizičko grupisanje i organizovanje funkcionalnosti. Takođe, svaki Node.js modul ujedno je i jedinstveni prostor imena, pa na kraju moduli sprečavaju pojavu konflikata usled korišćenja identičnih identifikatora.

Osnovne Node.js funkcionalnosti podeljene su u nekoliko modula:

- **http** modul sa funkcionalnostima za kreiranje HTTP servera,
- **url** – funkcionalnosti za analizu i parsiranje URL adresa,
- **querystring** – modul sa funkcionalnostima za rad sa query stringom,
- **path** – funkcionalnosti za rad sa putanjama,
- **fs** – funkcionalnosti za rad sa fajl sistemom,
- **util** – modul sa dodatnim, pomoćnim funkcionalnostima.

Nijedan od upravo navedenih modula nije podrazumevano uključen u aplikaciju koja se na izvršnom okruženju Node.js izvršava. Stoga je svaki od modula čije su nam funkcionalnosti potrebne neophodno uključiti u dokument u kojem će takve funkcionalnosti da budu korišćene. Modul se uključuje korišćenjem funkcije `require()`, koja poseduje sledeći oblik:

```
var module = require('module_name');
```

Funkciji `require()` prosleđuje se naziv modula, a ona kao svoju povratnu vrednost emituje prostu vrednost ili objekat koji je definisan kao povratna vrednost modula.

U upravo prikazanom primeru kreiranja HTTP servera korišćenjem Node.js-a obavlja se uključivanje `http` modula. Uključivanjem `http` modula dobija se objekat tipa `Server` koji se u nastavku koristi za kreiranje i pokretanje servera.

Server se prvo kreira korišćenjem metode `createServer()`. Prilikom kreiranja servera, metodi `createServer()` se prosleđuje jedna callback funkcija. Reč je o funkciji koja će biti aktivirana prilikom prijema svakog zahteva. Takva callback funkcija prihvata dva parametra, odnosno dva objekta:

- `req` – objekat kojim se predstavlja zahtev,
- `res` – objekat kojim se predstavlja odgovor koji će biti upućen klijentu.

Metoda `createServer()` ne stavlja server u pogon, već se to postiže pozivanjem metode `listen()` nad instancom servera. Metoda `listen()` prihvata tri parametra:

- port na kome će HTTP server slušati klijentske zahteve; u primeru je port definisan korišćenjem istoimene promenljive sa vrednošću 3000,
- naziv hosta preko koga će server biti dostupan; u primeru je naziv hosta definisan promenljivom `hostname` sa vrednošću 127.0.0.1; na taj način je naziv host postavljen na lokalnu IP adresu,
- callback funkcija koja će se aktivirati kada server postane spremjan.

Upravo prikazani kod za kreiranje servera ne poseduje logiku za kreiranje serverskog odgovora. Evo kako može izgledati kod za kreiranje jednostavnog odgovora:

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

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {

    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/html');
    res.end('<h1>Hello World</h1>');

})

server.listen(port, hostname, () => {
    console.log(`Server running at ${hostname}:${port}`);
})
```

Sada je unutar callback funkcije, koja se aktivira prilikom prijema HTTP zahteva, postavljena logika kojom se formira HTTP odgovor. Za obavljanje takvog posla koristi se objekat čija je referenca smeštena unutar drugog parametra koji takva callback funkcija prihvata. Definisanom logikom se formira vrlo jednostavan HTTP odgovor:

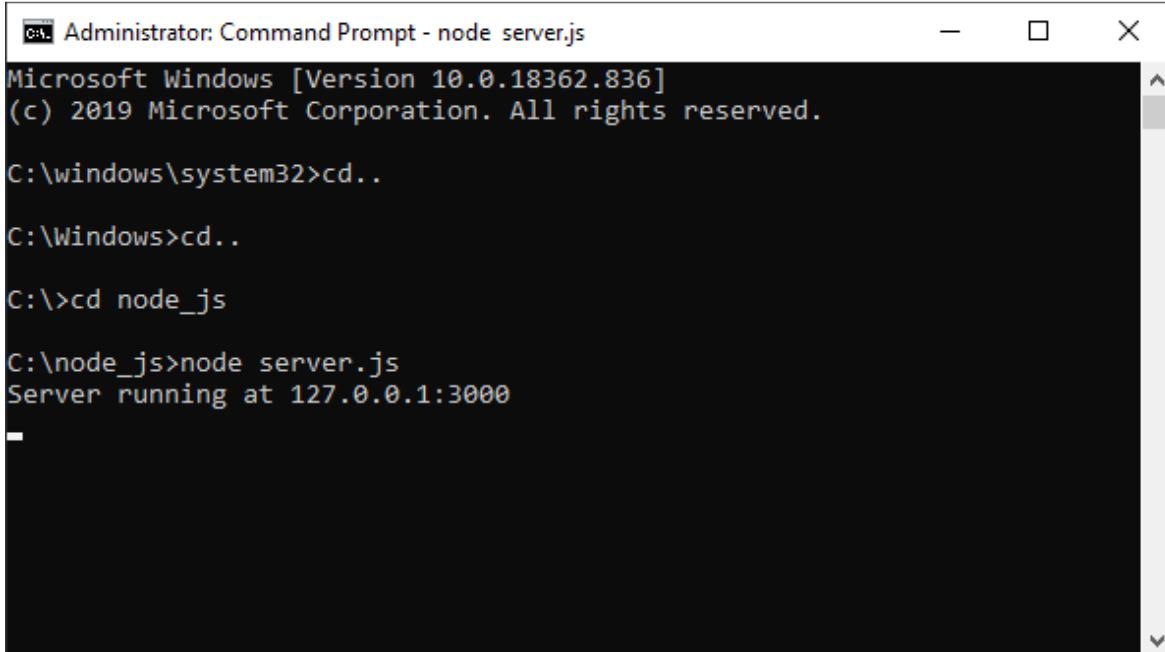
- svojstvom `statusCode` se statusni kod odgovora postavlja na 200,
- metodom `setHeader()` se definiše zaglavje `Content-Type` kojim se definiše tip sadržaja koji se isporučuju unutar tela odgovora,
- metodom `end()` se definiše sadržaj tela HTTP odgovora.

## Pokretanje Node.js servera i slanje prvog zahteva

Upravo kreirani HTTP server može se pokrenuti korišćenjem node komande unutar konzole ili terminala. Naravno, neophodno je pozicionirati se unutar foldera u kome se nalazi fajl sa upravo prikazanim JavaScript kodom, a onda pokrenuti komandu:

```
node server.js
```

Pokretanje Node.js servera ilustruje slika 3.15.



```
Administrator: Command Prompt - node server.js
Microsoft Windows [Version 10.0.18362.836]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\windows\system32>cd..

C:\Windows>cd..

C:\>cd node_js

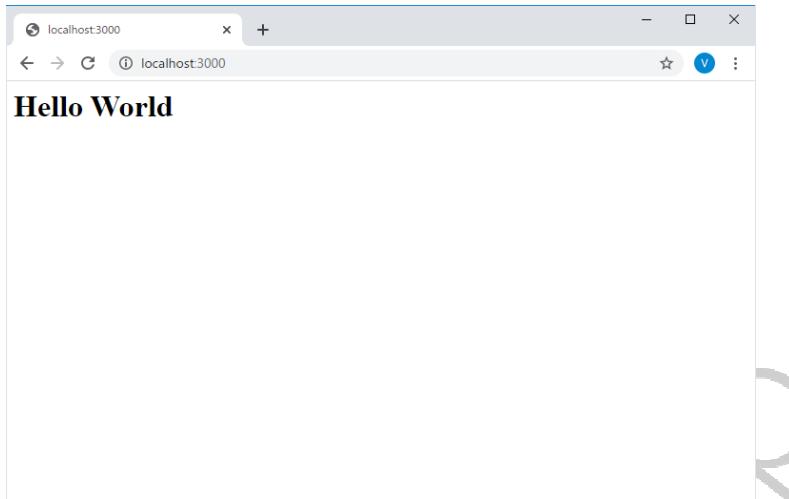
C:\node_js>node server.js
Server running at 127.0.0.1:3000
```

Slika 3.15. Pokretanje Node.js servera

Nakon pokretanja servera, moguće je otvoriti web pregledač i poslati zahtev upravo kreiranom serveru, navođenjem localhost adrese i porta 3000:

<http://localhost:3000/>

Izgled stranice dobijene od Node.js servera ilustruje slika 3.16.



Slika 3.16. Stranica u web pregledaču koja je dobijena od Node.js servera

### Napomena

Obratite pažnju na to da je Node.js server aktivan sve dok je otvoren prozor konzole ili terminala. Kada se takav prozor zatvori, uništava se i kreirani Node.js server.

## Npm

Na početku ove lekcije je rečeno da je sastavni deo Node.js ekosistema i jedan poseban menadžer paketa – npm (engl. *Node.js Package Manager*). Tokom vremena on je uspeo da se nametne kao absolutni standard kada su u pitanju repozitorijumi koji omogućavaju automatsko razrešavanje zavisnosti prilikom korišćenja JavaScript jezika.

### Šta su menadžeri paketa?

npm je jedan od specijalizovanih programa koji se nazivaju menadžeri paketa. Reč je o programima koji se u svetu programiranja koriste za lako upravljanje zavisnostima. Zavisnosti su sve one eksterne funkcionalnosti koje su jednoj aplikaciji neophodne za rad. Tako na primer, zavisnosti jedne web aplikacije mogu da budu jQuery, Vue, Modaal, Moment, React, Lodash... Pored nabrojanih zavisnosti, koje su u stvari JavaScript biblioteke, zavisnosti se prilikom razvoja web sajtova mogu odnositi i na različite biblioteke namenjene stilizovanju – Bootstrap, Font Awesome, Material UI, Normalize...

Menadžeri paketa omogućavaju automatizovano rukovanje bibliotekama i softverskim okvirima koje koristi jedna aplikacija. Tako oni efikasno oslobađaju programera od potrebe da samostalno pronalazi, preuzima i uključuje različite biblioteke i da kasnije brine o njihovom ažuriranju.

Kako bi mogli da se automatizovano brinu o različitim zavisnostima, menadžeri paketa se u pozadini oslanjaju na još jedan, veoma važan pojam u priči o automatskom rukovanju zavisnostima. Reč je o repozitorijumima. **Repozitorijum** je digitalna (online) baza u kojoj se čuvaju različite softverske biblioteke.

Svaka platforma poseduje svoje menadžere paketa i repozitorijume. *Maven* i *Gradle* se koriste u svetu Java programiranja, *Composer* prilikom rada sa PHP jezikom, a *pip* i *pypi* prilikom korišćenja jezika Python. Kada je u pitanju JavaScript jezik, absolutni standard je npm.

Npm je zapravo pojam koji podrazumeva:

- **repozitorijum** – baza u kojoj se čuvaju informacije i programski kod paketa,
- **klijentsku aplikaciju** – aplikacija koja se instalira na klijentskim uređajima, a koristi se za rukovanje paketima (reč je o izvršnom fajlu sa nazivom `npm`, koji se tokom instalacije smešta unutar sistemske Path promenljive),
- **web sajt** (`npmjs.com`) – korišćenjem ovog web sajta moguće je pretraživati pakete koji se nalaze unutar repozitorijuma.

Bitno je reći da je npm repozitorijum sa najviše biblioteka jednog jezika na svetu. Više od 400.000 biblioteka namenjenih JavaScript programiranju nalazi se unutar npm repozitorijuma.

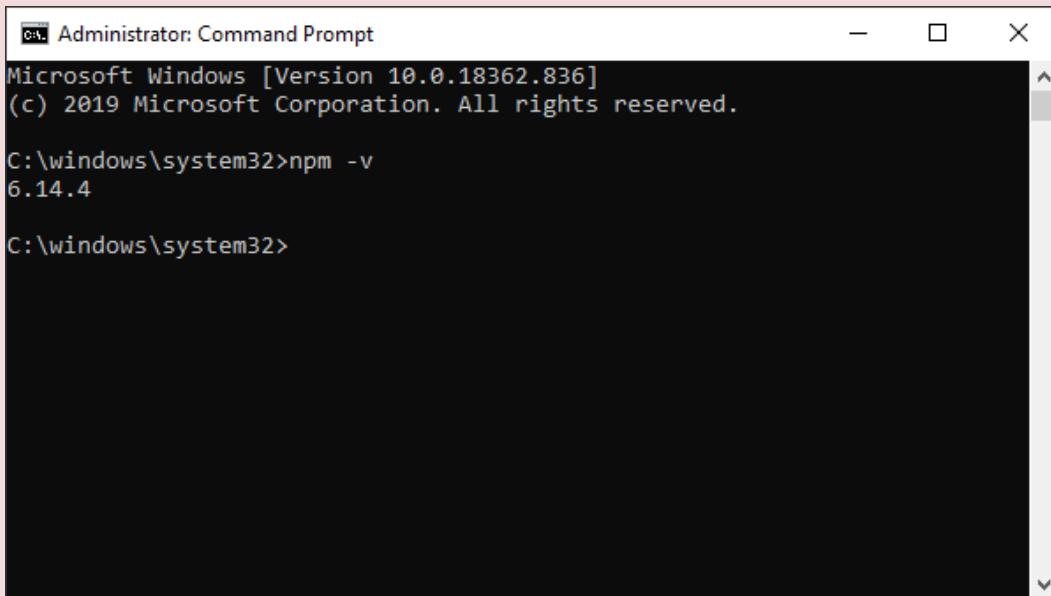
Takođe, npm je zapravo JavaScript aplikacija, koja se izvršava posredstvom izvršnog okruženja Node.js.

### Provera raspoloživosti npm-a

Kao što ste mogli videti na početku ove lekcije, npm se podrazumevano automatski instalira zajedno sa izvršnim okruženjem Node.js. Kako biste ipak proverili da li na vašem operativnom sistemu postoji instaliran npm, dovoljno je da unutar konzole ili terminala unesete sledeću komandu:

```
npm -v
```

U slučaju raspoloživosti npm menadžera paketa, dobija se ispis kao na slici 3.17.



The screenshot shows a Windows Command Prompt window titled "Administrator: Command Prompt". The window displays the following text:  
Microsoft Windows [Version 10.0.18362.836]  
(c) 2019 Microsoft Corporation. All rights reserved.  
C:\windows\system32>npm -v  
6.14.4  
C:\windows\system32>

Slika 3.17. Komanda kojom se proverava raspoloživost npm-a

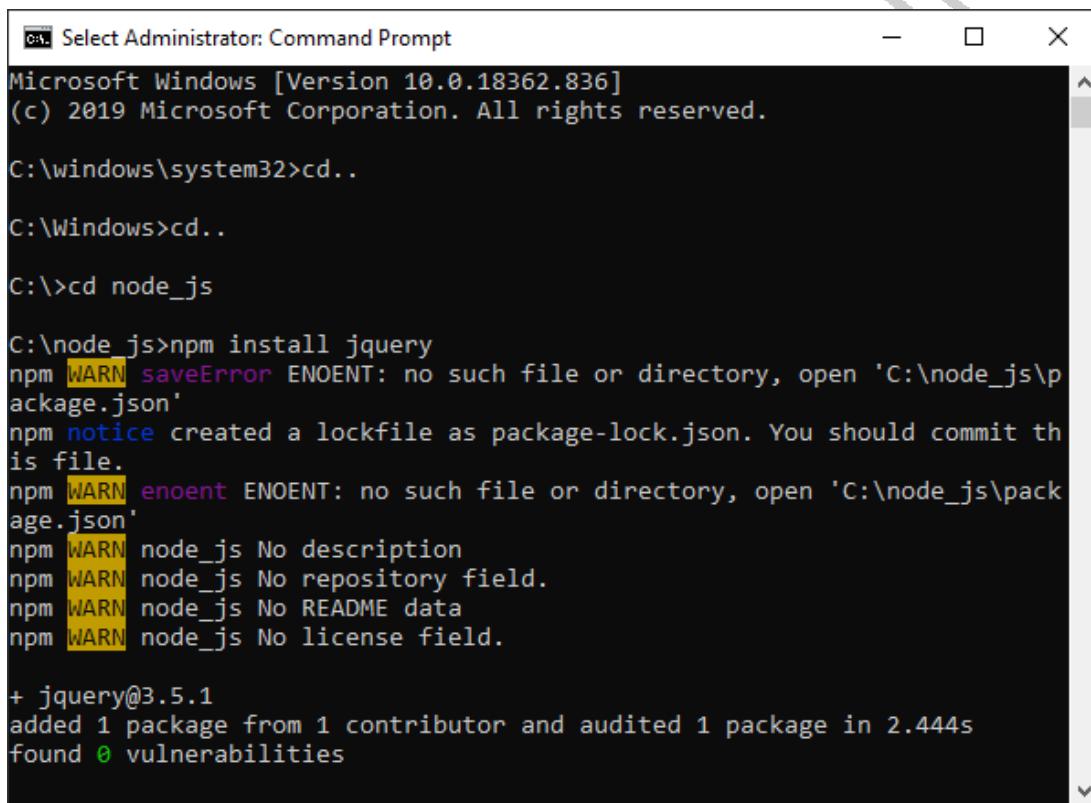
Sa slike 3.17. možemo videti da je npm instaliran i spreman za korišćenje.

## Instaliranje paketa korišćenjem npm-a

Do sada ste verovatno na sajтовима različitih JavaScript biblioteka primetili da gotovo svaka biblioteka, pored tradicionalnog načina integracije, koji se zasniva na preuzimanju fajlova i ručnom smeštanju na određenu putanju unutar projekta, omogućava i integraciju korišćenjem npm menadžera paketa. Na primer, kako bi se korišćenjem npm-a obavila integracija biblioteke jQuery, dovoljno je uputiti sledeću komandu:

```
npm install jquery
```

Pre nego što pokrenete prikazanu komandu unutar konzole ili terminala, bitno je znati da je neophodno da budete pozicionirani unutar korenog foldera projekta u koji želite da integrišete definisanu biblioteku (slika 3.18).



The screenshot shows a Windows Command Prompt window titled "Select Administrator: Command Prompt". The window displays the following output:

```
Microsoft Windows [Version 10.0.18362.836]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Windows\system32>cd..

C:\Windows>cd..

C:\>cd node_js

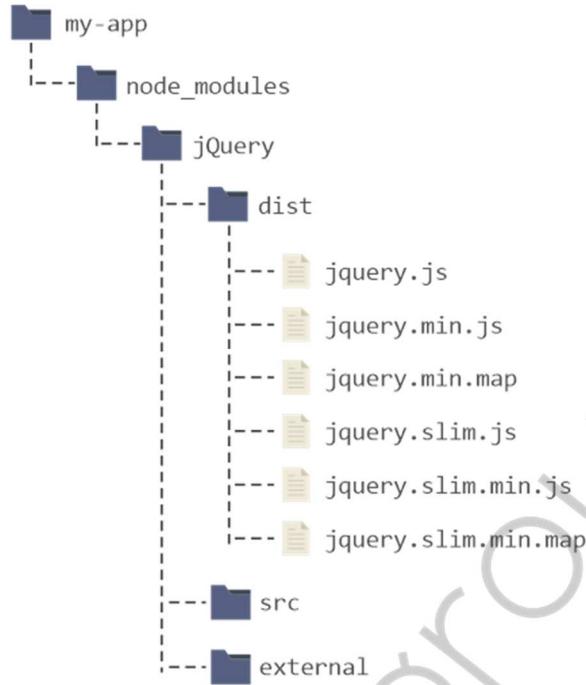
C:\node_js>npm install jquery
npm WARN saveError ENOENT: no such file or directory, open 'C:\node_js\p
ackage.json'
npm notice created a lockfile as package-lock.json. You should commit th
is file.
npm WARN enoent ENOENT: no such file or directory, open 'C:\node_js\pack
age.json'
npm WARN node_js No description
npm WARN node_js No repository field.
npm WARN node_js No README data
npm WARN node_js No license field.

+ jquery@3.5.1
added 1 package from 1 contributor and audited 1 package in 2.444s
found 0 vulnerabilities
```

Slika 3.18. Preuzimanje biblioteke jQuery korišćenjem npm-a

## Fajl struktura npm paketa

Način na koji npm obavlja integraciju paketa unutar nekog projekta razlikuje se od klasičnog pristupa koji podrazumeva preuzimanje fajlova i njihovo ručno smeštanje unutar projektne strukture. Naime, paketi koji se u neki projekat integriraju korišćenjem npm-a smeštaju se u poseban folder, koji se naziva `node_modules` (slika 3.19).



*Slika 3.19. Uprošćena struktura projekta nakon integracije jQuery biblioteke korišćenjem npm-a*

Na slici 3.19. možete da vidite da je npm, prilikom integracije prvog paketa, unutar korenog foldera projekta napravio novi folder sa nazivom `node_modules`. To je folder unutar koga se smeštaju preuzeti paketi. Pri tome se svaki paket smešta unutar zasebnog foldera, koji se imenuje po nazivu paketa.

Još jedna razlika u odnosu na integraciju tradicionalnim putem odnosi se na strukturu zavisnosti koje se integrišu. Sa slike 3.19. možete videti da je biblioteka jQuery predstavljena korišćenjem nekoliko različitih foldera: `src`, `dist`, `external`... Drugim rečima, integracijom nekog paketa korišćenjem npm-a, pored fajlova koji će biti korišćeni unutar HTML dokumenata, preuzima se i izvorni kod biblioteke, ali i sve eventualne zavisnosti i eksterne biblioteke koje mogu biti potrebne za korišćenje preuzetog paketa.

Struktura foldera i fajlova unutar paketa može se razlikovati od paketa do paketa, ipak, gotovo uvek će postojati folderi:

- `src` ili `source` – folder sa izvornim kodom paketa,
- `dist` – folder sa fajlovima koji su spremni za upotrebu, odnosno integraciju u neki drugi projekat.

## package.json fajl

Do sada ste videli samo jednu očiglednu prednost korišćenja npm-a, koja se odnosi na veoma lak način preuzimanja paketa. Još jedna veoma korisna osobina npm projekata ogleda se u mogućnosti definisanja jednog posebnog fajla – package.json.

Fajl package.json koristi se za opisivanje jednog npm paketa i po pravilu se smešta unutar korenog foldera paketa koji opisuje. Ukoliko, na primer, pogledate unutar korenog foldera nešto ranije preuzete biblioteke jQuery, moći ćete da pronađete upravo spomenuti package.json fajl.

Pored informacija o paketu, package.json fajl omogućava da se precizno definišu sve zavisnosti jednog projekta, što na kraju npm menadžeru paketa omogućava da njima automatski rukuje.

Odlična stvar je što i mi za projekte koje samostalno kreiramo možemo da iskoristimo blagodeti package.json fajla. Najlakši način da se za jedan projekat definiše package.json fajl, jeste korišćenje npm init komande:

```
npm init
```

Pokretanje ove komande inicira seriju pitanja na koje je potrebno dati odgovor. Na osnovu odgovora, npm će formirati sadržaj package.json fajla. Primer jednostavnog package.json fajla za projekat iz ove lekcije može da izgleda ovako:

```
{
  "name": "my-app",
  "version": "1.0.0",
  "description": "This is my demo app",
  "main": "server.js",
  "dependencies": {
    "jquery": "^3.5.1"
  },
  "devDependencies": {},
  "author": "Vladimir Dresevic",
  "license": "ISC"
}
```

Posebno važan odeljak unutar package.json fajla jeste onaj koji je označen ključem dependencies. Unutar takvog svojstva JSON objekta čuvaju se sve zavisnosti našeg projekta. Postojanje ovakvog fajla koji opisuje kompletan projekat, zajedno sa spiskom svih zavisnosti, omogućava da se sve zavisnosti preuzmu odjednom, definisanjem sledeće komande:

```
npm install
```

Ovakav pristup znatno olakšava deljenje koda i timski rad, zato što bilo ko poseduje package.json fajl može u potpunosti da kreira strukturu zavisnosti korišćenjem samo jedne komande. npm će prilikom izvršavanja install komande da preuzme i sve zavisnosti koje su definisane unutar dependencies sekcije package.json fajla.

## Node.js, npm i frontend razvoj

Nas u ovom trenutku posebno zanima upotrebna vrednost izvršnog okruženja Node.js i npm menadžera paketa kada je u pitanju frontend razvoj. Do sada ste mogli da vidite na koji način npm olakšava proces preuzimanja i instalacije JavaScript biblioteka unutar projekta na kome radimo. Ipak, još uvek nismo videli kakve nam to prednosti može doneti prilikom frontend razvoja, odnosno kako da preuzete pakete integrišemo unutar HTML dokumenata kojima se predstavlja prezentacija naših JavaScript aplikacija.

Ono što ste mogli da vidite u prethodnim redovima jeste to da npm preuzete pakete smešta unutar posebnog foldera `node_modules`. S pravom se možete zapitati zbog čega npm obavlja integraciju paketa na ovaj način, a ne na način koji se koristi i prilikom ručne integracije. Ne zaboravite da je npm primarno menadžer paketa za Node.js, čija je osnovna oblast interesovanja izvršavanje JavaScript koda na serveru. Stoga je npm struktura paketa prilagođena korišćenju toga unutar Node.js aplikacija. Naime, prilikom razvoja Node.js aplikacija (aplikacija koje će da izvršava izvršno okruženje Node.js), jednom instalirani paket jednostavno je moguće uključiti u projekat pozivanjem već viđene `require()` funkcije (ona je nešto ranije prikazana kao alat kojim se obavlja uključivanje osnovnih Node.js modula).

S obzirom na to da ćemo mi u nastavku ovoga kursa npm pakete koristiti za frontend razvoj (odnosno za pisanje aplikacija koje se izvršavaju unutar web pregledača), ne možemo se osloniti na upravo spomenutu funkciju `require()` kako bi se obavila integracija preuzetih fajlova. Najjednostavniji način za obavljanje integracije preuzetih fajlova jeste da ručno pronađemo potrebne fajlove unutar dobijene fajl strukture i da unutar našeg HTML dokumenta (ili više njih) postavimo kod za povezivanje. Na primer:

```
<script src="node_modules\jquery\dist\jquery.js"></script>
```

Korišćenjem upravo prikazane linije HTML koda može se obaviti integracija jQuery paketa, koji je nešto ranije preuzet korišćenjem npm-a.

S obzirom na nešto komplikovaniji način integracije fajlova koji su preuzeti korišćenjem npm menadžera paketa, tokom vremena su razvijeni brojni alati koje je, između ostalog, moguće koristiti za automatizaciju takvog procesa. Takvi alati napisani su korišćenjem JavaScript jezika, a izvršava ih izvršno okruženje Node.js. Tako su Node.js i npm efikasno postali neizostavni alati koji se koriste i prilikom frontend razvoja. Na primer, kako bi se obavila automatska integracija preuzetih npm paketa, kada se kreira aplikacija koja će se izvršavati unutar web pregledača, programerima su na raspolaganju različiti alati za izgradnju projekata. Neki od najpoznatijih takvih alata su Browserify i Webpack. Reč je o alatima koji posredstvom izvršnog okruženja Node.js mogu da transformišu kod jednog npm projekta, odnosno da automatski izgrade fajlove koji su spremni za korišćenje unutar web pregledača. Tako oni predstavljaju moćnu alternativu upravo prikazanom pristupu koji podrazumeva ručno uključivanje preuzetih biblioteka navođenjem putanja na kojima se one nalaze unutar `node_modules` foldera.

Node.js i npm osnova su funkcionisanja i različitih alata koji olakšavaju i ubrzavaju kreiranje frontend logike kada se koriste različite biblioteke ili softverski okviri koji su namenjeni frontend razvoju. Stoga će Node.js i npm biti korišćeni i u nastavku ovoga kursa, prilikom upoznavanja Vue, React i Angular softverskih okvira. Naime, svi navedeni softverski okviri namenjeni frontend programiranju poseduju i odgovarajuće komandne interfejse (engl. *CLI, Command Line Interface*) koji olakšavaju razvoj i automatizuju neke uobičajene operacije prilikom kreiranja JavaScript aplikacija.

## Rezime

- Node.js je JavaScript izvršno okruženje koje omogućava da se JavaScript izvršava izvan web pregledača.
- Node.js omogućava izvršavanje JavaScript koda na serveru.
- Node.js omogućava izvršavanje alata koji olakšavaju i ubrzavaju frontend razvoj.
- Node.js je dostupan za sve relevantne operativne sisteme: Windows, Linux, macOS.
- Web API-je koje izlažu web pregledači nije moguće koristiti prilikom izvršavanja koda na Node.js-u.
- Sastavni deo Node.js-a jeste komandni alat koji omogućava rukovanje izvršnim okruženjem korišćenjem konzole ili terminala; takav komandni alat sadržan je u izvršnom fajlu sa nazivom `node`.
- Provera raspoloživosti Node.js-a može se obaviti upućivanjem komande: `node -v`.
- REPL je mod koji omogućava pisanje jednostavnih JavaScript naredaba direktno unutar konzole; REPL mod pokreće se upućivanjem `node` komande.
- Kada se nakon `node` komande navede putanja do nekog `.js` fajla, kod takvog fajla biva izvršen od Node.js okruženja.
- npm je menadžer paketa Node.js izvršnog okruženja.
- npm je JavaScript aplikacija, koja se izvršava posredstvom izvršnog okruženja Node.js.
- Proveru raspoloživosti npm-a je moguće obaviti upućivanjem komande `npm -v`.
- Instalacija nekog paketa (modula) unutar tekućeg projekta se korišćenjem npm-a može obaviti upućivanjem komande `npm install package_name`, gde se `package_name` odnosi na naziv paketa.
- Paketi koji se u neki projekat integrišu korišćenjem npm-a smeštaju se u poseban folder koji se naziva `node_modules`.
- Fajl `package.json` se koristi za opisivanje jednog npm paketa i po pravilu se smešta unutar korenog foldera paketa koji opisuje.
- `package.json` fajl se može kreirati korišćenjem `npm init` komande.
- Instalacija svih zavisnosti iz `package.json` fajla obavlja se komandom `npm install`.
- Node.js i npm osnova su funkcionalnosti različitih alata koji olakšavaju i ubrzavaju kreiranje frontend logike.

