

# Web aplikacije (WA)

---

**Nositelj:** doc. dr. sc. Nikola Tanković

**Asistent:** Luka Blašković, mag. inf.

**Ustanova:** Sveučilište Jurja Dobrile u Puli, Fakultet informatike u Puli



Fakultet informatike u Puli

## (1) Uvod u HTTP, Node i Express

---

#1

WA

Web aplikacije su sofisticirana programska rješenja koja se pokreću na web poslužitelju, a korisnici im pristupaju putem web preglednika. Njihova najveća prednost je široka dostupnost na gotovo svim platformama i uređajima, bez potrebe za lokalnom instalacijom. Ovaj kolegij usmjeren je na dizajn i razvoj web aplikacija korištenjem modernih tehnologija i alata. Za razliku od kolegija Programsko inženjerstvo, ovdje ćete naučiti kako implementirati poslužiteljski sloj web aplikacije – ključni dio koji možemo zamisliti kao "mozak" aplikacije, zadužen za logiku i obradu podataka.

 Posljednje ažurirano: 21.10.2025.

## Sadržaj

---

- [Web aplikacije \(WA\)](#)
- [\(1\) Uvod u HTTP, Node i Express](#)
  - [Sadržaj](#)
- [1. Uvod](#)
  - [1.1 Kratak povijesni pregled](#)
- [2. Instalacija potrebnih alata](#)
  - [2.1 Node.js](#)
  - [2.2 VS Code](#)
  - [2.3 Git](#)
- [3. Kako započeti novi projekt?](#)
  - [3.1 Inicijalizacija novog repozitorija](#)
  - [3.2 Izrada Node projekta](#)
- [4. Postavljanje osnovnog Express poslužitelja](#)
  - [4.1 Instalacija Express.js](#)

- [4.2 Osnovni Express.js poslužitelj](#)
- [4.3 Kako definirati osnovni endpoint?](#)
- [4.4 Nodemon](#)
- [4.5 Git commit](#)
- [5. HTTP protokol](#)
  - [5.1 HTTP zahtjev \(eng. HTTP request\)](#)
    - [5.1.1 Obavezni dijelovi HTTP zahtjeva](#)
      - [Vježba 1 - HTTP zahtjev prema našem Expressu](#)
    - [5.1.2 Opcionalni dijelovi HTTP zahtjeva](#)
  - [5.2 HTTP odgovor \(eng. HTTP response\)](#)
    - [5.2.1 Obavezni dijelovi HTTP odgovora](#)
    - [5.2.2 Opcionalni dijelovi HTTP odgovora](#)
      - [Vježba 2: Kako vidjeti cijeli HTTP odgovor?](#)
- [Samostalni zadatak za Vježbu 1](#)

# 1. Uvod

---

Web aplikacije su softverski programi koji se koriste putem internetskog preglednika, bez potrebe za instalacijom na korisničkom računalu ili uređaju. One se izvršavaju na web poslužitelju, a korisnici im pristupaju putem internetskog preglednika. Web aplikacije su dostupne na gotovo svim platformama i uređajima, što ih čini vrlo popularnima među korisnicima.

Primjeri web aplikacija:

- Gmail: web aplikacija za slanje i primanje e-pošte
- Google Docs: omogućuje stvaranje i uređivanje dokumenata u stvarnom vremenu
- Facebook: društvena mreža za povezivanje s prijateljima i obitelji
- Online trgovine: web shopovi poput Amazona ili Ebaya za kupnju proizvoda

Svaka web aplikacija sastoji se od minimalno dva dijela:

1. **Klijentski dio** (eng. *client side*): izvršava se na korisničkom uređaju (npr. računalo, pametni telefon) i koristi se za prikaz korisničkog sučelja. Napisan je u jezicima poput HTML-a, CSS-a i JavaScripta, odnosno razvojnim okvirima poput Reacta, Angulara ili Vue.js-a.
2. **Poslužiteljski dio** (eng. *server side*): izvršava se na web poslužitelju i koristi se za obradu zahtjeva korisnika, komunikaciji s bazom podataka i definiranje poslovne logike aplikacije. Napisan je u jezicima poput JavaScripta (Node.js), Pythona (Django, Flask), Rubyja (Ruby on Rails) ili Jave (Spring).

## 1.1 Kratak povijesni pregled

---

Premda nije predmet ovog kolegija, **PHP** je također popularan jezik za izradu poslužiteljskog dijela web aplikacija. Glavna prednost PHP-a je generiranje dinamičkih HTML stranica na poslužiteljskoj strani, što ga čini idealnim za izradu web stranica i aplikacija. Iako je prisutan već dugi niz godina, PHP i dalje ima veliku bazu korisnika i popularan je izbor za izradu web aplikacija (posebice moderni PHP okviri poput Laravela i Symfonya).

**JavaScript** je nešto mlađi programski jezik od PHP-a (svega nekoliko mjeseci), a prvi put je implementiran u Netscape Navigatoru (najpopularniji web preglednik u to vrijeme) 1995. godine. JavaScript je postao popularan zbog svoje sposobnosti stvaranja interaktivnih korisničkih sučelja na klijentskoj strani web aplikacija, što je dovelo do razvoja modernih web aplikacija poput Gmaila i Google Mapsa. Te aplikacije su imale interaktivno korisničko sučelje napisano u JavaScriptu, dok se se za poslužiteljski sloj koristili PHP i C++ jezici.

Danas gotovo 99% web stranica koristi JavaScript na klijentskoj strani za implementaciju interaktivnog ponašanja, a svaki moderni web preglednik ima ugrađen svoj JavaScript engine koji izvršava JavaScript kod.

Međutim, 2009. godine na tržište izlazi novi revolucionarni alat koji je promijenio način na koji se danas razvijaju moderne web aplikacije - **Node.js**. Node.js je JavaScript okruženje (*eng. runtime environment*) koje dozvoljava izvođenje JavaScript koda na poslužiteljskoj strani. Drugim riječima, Node.js omogućava izvršavanje JavaScript koda izvan web preglednika.

Mi ćemo se na ovom kolegiju fokusirati upravo na taj poslužiteljski sloj web aplikacija, koristeći Node.js, odnosno biblioteku **Express.js** za izradu poslužiteljskog dijela web aplikacija.



A za one koji žele više, proučite [Deno](#) - novi JavaScript *runtime environment* koji brzo dobiva na popularnosti, a razvija ga isti programer koji je razvio Node.js!

## 2. Instalacija potrebnih alata

### 2.1 Node.js

**Node.js** možete preuzeti sa [službene stranice](#). Preuzmite LTS verziju (Long Term Support) koja je stabilna i sigurna za produkciju. Nakon preuzimanja, pokrenite instalacijski program i slijedite upute za instalaciju.

Preporuka je preuzeti verziju LTS 20+.

Nakon što ste uspješno instalirali Node.js, možete provjeriti je li instalacija uspješna tako da otvorite terminal i upišete sljedeću naredbu:

```
node -v
```

Ako je instalacija uspješna, trebali biste vidjeti verziju Node.js-a koju ste instalirali. Na primjer:

```
v22.9.0
```

Instalacijom Node-a dobivate i `npm` (*Node Package Manager*) koji koristimo za instalaciju paketa i modula potrebnih za razvoj web aplikacija.

```
npm -v
```

## 2.2 VS Code

**Visual Studio Code** je besplatni uređivač koda koji je dostupan za Windows, macOS i Linux. Možete preuzeti Visual Studio Code sa [službene stranice](#). Nakon preuzimanja, pokrenite instalacijski program i slijedite upute za instalaciju.

Nakon što ste uspješno instalirali Visual Studio Code, možete provjeriti je li instalacija uspješna tako da otvorite terminal i upišete sljedeću naredbu:

```
code --version
```

Možete pokrenuti Visual Studio Code tako da upišete sljedeću naredbu:

```
code
```

ili jednostavno pokrenite kroz grafičko sučelje.

## 2.3 Git

**Git** je besplatni sustav za upravljanje izvornim kodom koji je dostupan za Windows, macOS i Linux. Možete preuzeti Git sa [službene stranice](#). Nakon preuzimanja, pokrenite instalacijski program i slijedite upute za instalaciju.

Iako nije nužan za sam razvoj web aplikacija, Git je koristan alat koji ćemo često koristiti za verzioniranje izvornog koda.

Nakon što ste uspješno instalirali Git, možete provjeriti je li instalacija uspješna tako da otvorite terminal i upišete sljedeću naredbu:

```
git --version
```

Ako je instalacija uspješna, trebali biste vidjeti verziju Git-a koju ste instalirali. Na primjer:

```
git version 2.47.0
```

Ako još uvijek nemate, svakako morate izraditi i [Github](#) račun. GitHub je vrlo popularna platforma gdje developeri mogu pohranjivati, dijeliti te surađivati na kodu i projektima, a bazira se na Gitu, kao pozadinskom sustavu za verzioniranje koda.

## 3. Kako započeti novi projekt?

---

Nakon što ste uspješno instalirali Node.js, Visual Studio Code i Git, možete započeti raditi na novom projektu.

## 3.1 Inicijalizacija novog repozitorija

Prvi korak je definiranje strukture projekta, budući da smo odlučili verzionirati izvorni kod, koristit ćemo Git za inicijalizaciju novog repozitorija. Međutim krenut ćemo od GitHuba: idemo na [Github izraditi novi repozitorij](#), a zatim ćemo ga klonirati na naše računalo. Klonirati (eng. *clone*) znači preuzeti udaljeni repozitorij na naše računalo (lokalno).

Nazovite repozitorij "**wa\_vjezbe\_01**" i dodajte opis po želji. Možete ga postaviti kao privatni ili javni, a svakako odaberite opciju "Add a README file" kako ne bi inicijalno bio prazan.

### Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).


Required fields are marked with an asterisk (\*).

#### Repository template

No template ▾

Start your repository with a template repository's contents.

#### Owner \*

 lukablaskovic ▾

#### Repository name \*

wa\_vjezbe\_01

✔ wa\_vjezbe\_01 is available.

Great repository names are short and memorable. Need inspiration? How about [stunning-enigma](#) ?

#### Description (optional)

Repozitorij za prve vježbe iz kolegija Web aplikacije (FIPU)



#### Public

Anyone on the internet can see this repository. You choose who can commit.



#### Private

You choose who can see and commit to this repository.

#### Initialize this repository with:



#### Add a README file

This is where you can write a long description for your project. [Learn more about READMEs](#).

Nakon što ste izradili repozitorij, kopirajte URL repozitorija, npr. <https://github.com/lukablaskovic/wa-vjezbe-01.git>

### 1. Način (terminal)

Otvorite terminal i navigirajte do direktorija u kojem želite spremiti projekt. Zatim upišite sljedeću naredbu:

```
cd putanja/do/direktorija
```

```
git clone <URL>
```

Na primjer, ako se direktorij nalazi na radnoj površini, naredba bi mogla izgledati ovako:

Windows:

```
cd C:\Users\<VAŠ USERNAME>\Desktop
```

macOS/linux:

```
cd Desktop
```

Zamijenite `<URL>` s URL-om repozitorija koji ste prethodno kopirali. Na primjer:

```
git clone https://github.com/lukablaskovic/wa-vjezbe-01.git
```

Kako biste se uvjerali da ste u pravom direktoriju, upišite naredbu:

Windows:

```
ls
```

ili

```
dir
```

macOS/linux:

```
ls
```

Ove naredbe će vam ispisati popis datoteka i direktorija u trenutnom direktoriju.

## 2. Način (VS Code)

Drugi način je kloniranje repozitorija direktno iz Visual Studio Codea. Otvorite Visual Studio Code i pritisnite `Ctrl + Shift + P` (Windows) ili `Cmd + Shift + P` (macOS) kako biste otvorili *Command Palette*. Upišite `Git: Clone` i pritisnite `Enter`. Zatim unesite `URL` repozitorija i pritisnite `Enter`.

Ako vam ne radi, uvjerite se da imate instaliran Git i da je dostupan u PATH-u. Dodatno, u VS Codeu morate biti prijavljeni na GitHub račun.

Možete se uvjeriti da je Git dostupan u PATH-u tako da otvorite terminal i upišete:

```
git --version
```

Ako nije, dobit ćete grešku neovisno o okruženju u kojem otvarate terminal. U tom slučaju, potrebno je reinstalirati Git kroz instalacijski program i odabrati opciju koja dodaje Git u PATH.

## 3. Način (Github Desktop)

Treći način je kloniranje repozitorija direktno iz Github Desktop aplikacije. Otvorite [Github Desktop](#) aplikaciju i pritisnite `Ctrl + Shift + O` (Windows) ili `Cmd + Shift + O` (macOS) kako biste otvorili Clone Repository prozor. Upišite URL repozitorija i pritisnite Clone.

GitHub desktop je odlična aplikacija za početnike jer nudi jednostavan način za upravljanje repozitorijima, ali nije nužna za rad na projektu. Sve što možete napraviti u GitHub Desktopu možete napraviti i u terminalu ili Visual Studio Codeu, ali Desktop nudi vizualni prikaz promjena i jednostavno upravljanje repozitorijima, promjenama, *branchovima*, što može biti vrlo korisno.

## 3.2 Izrada Node projekta

Jednom kad ste uspješno klonirali repozitorij, možete započeti s izradom Node projekta.

Otvorite terminal i navigirajte do direktorija projekta. Zatim upišite sljedeću naredbu:

```
code .
```

ili otvorite Visual Studio Code i navigirajte do direktorija projekta.

Možete i kroz GitHub Desktop i to tako da otvorite repozitorij u aplikaciji i pritisnete `Ctrl + Shift + A` (Windows) ili `Cmd + Shift + A` (macOS) kako biste otvorili repozitorij u Visual Studio Codeu.

Kada ste otvorili projekt u Visual Studio Codeu, otvorite novi terminal: `Terminal` -> `New Terminal`.

Zatim upišite sljedeću naredbu:

```
npm init
```

Ova naredba pokreće inicijalizaciju novog Node projekta. Slijedite upute i unesite podatke o projektu. Ako želite preskočiti neko polje, jednostavno pritisnite `Enter`.

Ako želite preskočiti cijeli upitnik i koristiti zadane postavke, dodajte `-y` opciju:

```
npm init -y
```

Wohoo! Uspješno ste inicijalizirali novi Node projekt! 🎉

Primjetit ćete da se u direktoriju projekta pojavila datoteka `package.json`. Ova datoteka sadrži informacije o projektu, poput naziva, verzije, autora, skripti i ovisnosti. Kroz kolegij ćemo detaljno vidjeti što znači svaki dio `package.json` datoteke i kako ju možemo koristiti za upravljanje projektom.

Struktura direktorija projekta trebala bi izgledati ovako (samo 1 datoteka):

```
.
└─ package.json

1 directory, 1 file
```

Idemo još malo ponoviti terminal: Konzola mi je nakrcana nakon ove inicijalizacije, kako da očistim? `clear` ili `cls` (Windows) | `clear` (macOS/linux) i sve će biti čisto. 🧹

Kako se mogu kretati kroz direktorije? `cd ime_direktorija` za ulazak u direktorij, `cd ..` za izlazak iz direktorija, `cd` za povratak u korijenski direktorij. 🚶

Ok sad opet ne znam di sam? `pwd` (macOS/linux) ili `cd` (Windows) će vam reći trenutnu lokaciju. 📍

# 4. Postavljanje osnovnog Express poslužitelja

## 4.1 Instalacija Express.js

Express.js je popularni web okvir za Node.js koji omogućava brzu izradu web aplikacija na poslužiteljskom sloju. Express.js je jedan od najpopularnijih web okvira za Node.js, a koristi se za izradu **poslužiteljskog dijela** web aplikacija.

express

Kako instalirati Express.js? U terminalu upišite sljedeću naredbu:

```
npm install express
```

Naredba `npm install` koristi se za instalaciju paketa i modula iz Node paketnog registra. U ovom slučaju, instalirali smo [Express.js](#) paket.

[Node paketni registar](#) je online baza podataka koja sadrži tisuće paketa i modula koje možemo koristiti u našim Node projektima.

Nakon što je instalacija završena, u direktoriju projekta trebali biste vidjeti nekoliko dodanih stavki:

- direktorij `node_modules` koji sadrži sve instalirane pakete i module odnosno njihov izvorni kod
- datoteku `package-lock.json` koja sadrži informacije o verzijama paketa i modula

`package-lock.json` datoteka je važna jer osigurava da svi članovi tima koriste iste verzije paketa i modula. Ova datoteka se automatski generira prilikom instalacije paketa i modula i također ju ne smijete mijenjati ručno.

Struktura direktorija projekta trebala bi izgledati ovako:

```
.
├── node_modules
├── package-lock.json
└── package.json

2 directories, 2 files
```

U sljedećem poglavlju ćemo izraditi naš prvi Express.js poslužitelj.

## 4.2 Osnovni Express.js poslužitelj

Krenimo napokon s implementacijom Express.js-a! 🚀 Dodat ćemo novu JavaScript datoteku proizvoljnog naziva, uobičajeno je koristiti `app.js`, `index.js` ili `server.js`.

Mi ćemo koristiti `index.js`.



Dodajte datoteku ručno, desni klik na direktorij projekta -> `New File` -> `index.js`. Ili ako želite biti terminal ninja 🥷, upišite:

```
touch index.js
```

Osnovni Express.js poslužitelj možemo definirati u svega nekoliko linija koda:

Prvo ćemo uključiti Express.js modul u našu datoteku:

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

Zatim ćemo stvoriti novu Express aplikaciju:

```
const app = express(); // u varijablu app pohranjujemo objekt koji predstavlja Express aplikaciju
```

OK, kako pokrećem Express.js poslužitelj? Koristimo `listen` metodu:

```
const PORT = 3000; // port na kojem će poslužitelj slušati zahtjeve  
  
app.listen(PORT); // Express aplikacija "sluša" na portu 3000
```

Cijeli kod izgleda ovako:

```
const express = require('express');  
const app = express();  
  
const PORT = 3000;  
app.listen(PORT);
```

Spremite datoteku i pokrenite Express.js poslužitelj tako da u terminalu upišete:

```
node index.js
```

Što se dogodilo? 🤔

Čini se kao da nije ništa, međutim možemo u terminalu vidjeti da je proces pokrenut budući da ne možemo više upisivati nove naredbe. To znači da je Express.js poslužitelj uspješno pokrenut i da sluša zahtjeve na portu 3000.

Gdje ovo mogu vidjeti? Aplikaciju pokrećemo na vlastitom računalu, tako da se ona izvodi na adresi

`127.0.0.1`, odnosno `localhost`. Dodatno, sluša na portu `3000`, tako da je puna adresa `http://localhost:3000`. Otvorite internetski preglednik i upišite ovu adresu, trebali biste vidjeti poruku "Cannot GET /" ili slično. To je u redu, jer nismo definirali nikakve rute ili putanje.

Kako zatvoriti Express.js poslužitelj? U terminalu pritisnite `Ctrl + C` (Windows) ili `Cmd + C` (macOS) kako biste prekinuli izvođenje programa. Ovo će zaustaviti Express.js poslužitelj i vratiti vam kontrolu nad terminalom.

Da bi bili sigurni da se naša aplikacija "vrti" možemo dodati `callback` funkciju našoj `listen` metodi:

```
app.listen(PORT, function () {  
  console.log(`Server je pokrenut na http://localhost:${PORT}`);  
});
```

Ova funkcija prima i `error` argument, tako da možemo uhvatiti potencijalne greške prilikom pokretanja poslužitelja:

Možemo ju zapisati i kao `arrow callback`:

```
app.listen(PORT, error => {  
  if (error) {  
    console.error(`Greška prilikom pokretanja poslužitelja: ${error.message}`);  
  } else {  
    console.log(`Server je pokrenut na http://localhost:${PORT}`);  
  }  
});
```

Spremite datoteku i ponovno pokrenite Express.js poslužitelj. Ovaj put ćete vidjeti poruku: `"Server je pokrenut na http://localhost:3000"` u terminalu.

I to je to! Uspješno ste izradili prvi Express.js poslužitelj! 🎉

## 4.3 Kako definirati osnovni endpoint?

Rute (eng. *routes*) su putanje koje korisnici mogu posjetiti u internetskom pregledniku. Na primjer, korisnik može posjetiti putanju `/` kako bi vidio početnu stranicu aplikacije, ili putanju `/about` kako bi vidio stranicu s informacijama o aplikaciji.

Nazivamo ih još i **endpoints** ili **API endpoints**.

**API** (eng. *Application Programming Interface*) je skup pravila i definicija koje omogućuju različitim softverskim aplikacijama da komuniciraju jedna s drugom. Express, iako se najčešće i koristi za izgradnju API servisa, može se koristiti i za izgradnju drugih vrsta softvera.

Mi ćemo u sklopu ovog kolegija koristiti Express.js za izgradnju ukupnog poslužiteljskog sloja naše web aplikacije, uključujući i API servis za komunikaciju s klijentskim dijelom aplikacije (Vue.js). Sigurno ste čuli i za **REST API** (eng. *Representational State Transfer*), međutim o tome ćemo uskoro!

Vratimo se na rute. Rekli smo da su to putanje koje korisnici mogu posjetiti u internetskom pregledniku i koje odgovaraju na određene zahtjeve korisnika.

Sigurno ste dosad imali priliku vidjeti rute u internetskim preglednicima, npr.

`https://moodle.srce.hr/2024-2025/` gdje je `2024-2025` ruta koja odgovara akademskoj godini, odnosno pretpostavljamo da će nas odvesti na stranicu s informacijama o akademskoj godini 2024/2025 (u pozadini: korisnik je zatražio informacije o akademskoj godini 2024/2025, a poslužitelj pokušava vratiti taj resurs).

Definirat ćemo osnovnu rutu `/` koja će korisnicima prikazati poruku "Hello, world!". Koristit ćemo `get` metodu koja obrađuje **HTTP GET** zahtjev.

```
app.get('/'); // definiramo rutu/endpoint
```

Zatim ćemo dodati `callback` funkciju koja će se izvršiti kada korisnik pošalje zahtjev na tu rutu.

Ova *callback* funkcija najčešće prima dva argumenta: `req` (request) i `res` (response).

`req` objekt sadrži informacije o zahtjevu korisnika, dok `res` objekt koristimo za slanje odgovora korisniku (možemo ih nazvati bilo kako ali ovo je konvencija i dobro je se držati).

Postoji i treći argument - `next` koji koristimo za preusmjeravanje zahtjeva na sljedeću funkciju u lancu middlewarea, ali o tome ćemo kasnije.

Osnovna metoda `res` objekta je `send` koja služi za slanje jednostavnog odgovora korisniku. Osim nje, postoji još mnogo metoda `response` objekta: poput `json` koja šalje podatke u obliku JSON-a ili `sendFile` koja šalje datoteku.

```
app.get('/', function (req, res) {  
  res.send('Hello, world!');  
});
```

ili `arrow callback`:

```
app.get('/', (req, res) => {  
  res.send('Hello, world!'); // šaljemo odgovor korisniku  
});
```

To je to!

Cijeli kod izgleda ovako:

```
const express = require('express');  
const app = express();  
  
const PORT = 3000;  
  
app.get('/', (req, res) => {  
  res.send('Hello, world!');  
});  
  
app.listen(PORT, error => {  
  if (error) {  
    console.error(`Greška prilikom pokretanja poslužitelja: ${error.message}`);  
  } else {  
    console.log(`Server je pokrenut na http://localhost:${PORT}`);  
  }  
});
```

Obavezno spremite datoteku i ponovo pokrenite Express.js poslužitelj. Otvorite internetski preglednik i posjetite adresu `http://localhost:3000`. Trebali biste vidjeti poruku "Hello, world!".

Međutim, što smo ustvari dobili nazad? Otvorimo konzolu u pregledniku (F12) i vidjet ćemo da smo dobili HTML stranicu s porukom `"Hello, world!"`.

```
<html>
  <head></head>
  <body>
    <text>Hello, world!</text>
  </body>
</html>
```

Kada budemo učili o zaglavljima HTTP zahtjeva, vidjet ćemo zašto je ovo tako. Za sad, zapamtite da će `send` metoda poslati odgovor korisniku u obliku HTML stranice.

## 4.4 Nodemon

Primijetite da je potrebno svaki put ručno zaustaviti i ponovno pokrenuti Express.js aplikaciju kada napravimo promjene u kodu. Ono što smo prilikom razvoja Vue.js aplikacija uzimali zdravo za gotovo, ovdje si moramo ručno podesiti. Iz tog razloga koristimo `nodemon` - paket koji automatski prati promjene u kodu i ponovno pokreće Express.js aplikaciju.

Kako instalirati `nodemon` kroz `npm`?

```
npm install -g nodemon
```

Opcija `-g` označava globalnu instalaciju, što znači da će `nodemon` biti dostupan u cijelom sustavu (našem računalu). Ovo je korisno jer možemo koristiti `nodemon` za pokretanje bilo koje Node.js aplikacije, a ne samo Express.js aplikacija. Ako vam `nodemon` ne radi globalno nakon instalacije, pokušajte restartirati računalu.

Rekli smo da u `package.json` datoteci definiramo aplikacije koje naš paket koristi. Kako naša aplikacija nema direktne koristi od `nodemon` paketa, već samo mi kao developeri, možemo koristiti `--save-dev` opciju prilikom instalacije koja će dodati `nodemon` paket u `devDependencies` dio `package.json` datoteke (odnosno pakete koji su potrebni samo prilikom razvoja aplikacije).

```
npm install --save-dev nodemon
```

Vaša `package.json` datoteka sada bi trebala izgledati ovako:

```
{
  "name": "wa_vjezbe_01",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
```

```
"express": "^4.21.1"
},
"devDependencies": {
  "nodemon": "^3.1.7"
}
}
```

Kako koristiti `nodemon`? Umjesto `node` naredbe, koristimo `nodemon` naredbu. Simple as that! 🚀

```
nodemon index.js
```

Sada kada napravimo promjene u kodu, `nodemon` će automatski prepoznati promjene i ponovno pokrenuti Express.js aplikaciju. To nam štedi vrijeme i olakšava razvoj aplikacije.

## 4.5 Git commit

Napravili smo dosta promjena u kodu, vrijeme je za prvi `commit`! 🎉

Primjećujemo da su se u lijevom izborniku VS Codea evidentirale promjene u našem projektu (vjerojatno njih 500+). Zašto se ovo dešava ako smo samo inicijalizirali node projekt i dodali jednu datoteku i napisali nekoliko linija koda? 🤔

Odgovor je jednostavan: `node_modules` direktorij. Ovaj direktorij sadrži sve instalirane pakete i module potrebne za uspješno izvođenje naše aplikacije. Ovaj direktorij je velik i sadrži tisuće datoteka, što znači da će se pojaviti puno promjena u našem projektu. Međutim, `node_modules` direktorij nije potreban za izvođenje naše aplikacije jer možemo ponovno instalirati sve pakete i module koristeći `npm install` naredbu.

`npm install` metoda čita `package.json` datoteku i instalira sve pakete i module navedene u `dependencies` i `devDependencies` dijelovima datoteke. Ovo je vrlo korisno jer omogućava drugim developerima da lako instaliraju sve potrebne pakete i module za izvođenje naše aplikacije.

Kako bismo izbjegli dodavanje `node_modules` direktorija u repozitorij, dodajemo ga u `.gitignore` datoteku. Ova datoteka sadrži popis datoteka i direktorija koje ne želimo dodati u repozitorij. Dodajte `node_modules` direktorij u `.gitignore` datoteku:

Datoteka: `.gitignore`

```
node_modules
```

Struktura našeg projekta sada izgleda ovako:

```
.
├── index.js
├── node_modules
├── package-lock.json
└── package.json

2 directories, 3 files
```

Primjetit ćete da se promjene u `node_modules` direktoriju više ne pojavljuju u lijevom izborniku VS Codea i da se broj smanjio na nekoliko promjena.

Sada smo spremni napraviti naš prvi `commit`!

## 1. Način (kroz terminal):

Prvo provjerimo stanje indeksa:

```
git status
```

Ova naredba će ispisati sve promjene u projektu. Za sada nismo definirali što dodajemo u indeks, pa će nas tražiti da dodajemo datoteke s naredbom `git add`.

Možemo dodati sve datoteke u indeks tako da kao argument navedemo `.`:

```
git add .
```

Pozvat ćemo opet `git status` kako bismo provjerili jesu li sve datoteke dodane u indeks:

```
git status
```

Uvjerite se da nema datoteke `node_modules` u popisu datoteka koje će se dodati u indeks.

```
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   .gitignore
    new file:   index.js
    new file:   package-lock.json
    new file:   package.json
```

Sada možemo pohraniti promjene kroz `commit` naredbu:

Dodajemo obaveznu poruku uz `-m` opciju:

```
git commit -m "Inicijalni commit"
```

Ako je sve prošlo u redu, dobit ćete poruku s popisom promjena:

```
[main 465f003] Inicijalni commit
4 files changed, 1184 insertions(+)
create mode 100644 1. Uvod u Node i Express/wa_vjezbe_01/.gitignore
create mode 100644 1. Uvod u Node i Express/wa_vjezbe_01/index.js
create mode 100644 1. Uvod u Node i Express/wa_vjezbe_01/package-lock.json
create mode 100644 1. Uvod u Node i Express/wa_vjezbe_01/package.json
```

Ono što još trebamo napraviti je pohraniti promjene na udaljeni repozitorij. Ovo radimo kroz `push` naredbu:

```
git push
```

## 2. Način (kroz VS Code):

Otvorite Source Control tab u lijevom izborniku VS Codea. Prikazat će se sve promjene u projektu. Unesite poruku i jednostavno pritisnite `✓ Commit` ikonu kako biste pohranili promjene (ovo je ekvivalentno `git add` i `git commit` naredbama). Zatim odaberite `Sync Changes` kako biste pohranili promjene na udaljeni repozitorij (ovo je ekvivalentno `git push` naredbi).

## 3. Način (kroz GitHub Desktop):

Otvorite GitHub Desktop aplikaciju i pronađite vaš repozitorij. Vidjet ćete vizualni prikaz promjena u projektu na tabu `Changes`.

Možete dodati opis promjena i pritisnuti `Commit to main` kako biste pohranili promjene (ovoje ekvivalentno `git commit` naredbi). Zatim pritisnite `Push origin` kako biste pohranili promjene na udaljeni repozitorij (ovo je ekvivalentno `git push` naredbi).

# 5. HTTP protokol

HTTP (eng. *Hypertext Transfer Protocol*) je protokol koji se koristi za **prijenos podataka na webu**. HTTP definira skup pravila i definicija koje omogućuju web preglednicima i poslužiteljima da komuniciraju jedni s drugima. HTTP protokol uključuje **zahtjeve** (eng. *requests*) koje klijenti šalju poslužiteljima, te **odgovore** (eng. *responses*) koje poslužitelji šalju klijentima.

HTTP koristi različite **metode** (eng. *HTTP method*) za različite vrste zahtjeva. Najčešće korištene HTTP metode su:

- **GET** - koristi se za dohvaćanje podataka
- **POST** - koristi se za slanje podataka
- **PUT** - koristi se za ažuriranje podataka
- **DELETE** - koristi se za brisanje podataka
- **PATCH** - koristi se za djelomično ažuriranje podataka

Ove metode koriste se za različite vrste zahtjeva. Na primjer, korisnik može poslati `GET` zahtjev kako bi dohvatio podatke s poslužitelja, ili `POST` zahtjev kako bi poslao podatke poslužitelju. Sve ove metode koriste se u web razvoju za komunikaciju između klijenta i poslužitelja. U nastavku ćemo obraditi svaku metodu posebno i pokazati kako ih implementirati u Express.js aplikaciji.

Međutim, prije nego što krenemo, važno je naučiti od čega se sastoje HTTP zahtjevi i odgovori.

HTTP prati klasičnu **klijent-poslužitelj** arhitekturu (eng. *client-server architecture*). Ukratko, to znači da klijent šalje zahtjev poslužitelju, a poslužitelj šalje odgovor klijentu. Preciznije, klijent otvara **TCP/IP** vezu s poslužiteljem, šalje HTTP zahtjev i onda čeka sve dok poslužitelj ne pošalje odgovor.

HTTP je **stateless** protokol, što znači da svaki zahtjev poslužitelju ne zna ništa o prethodnim zahtjevima. Na primjer, kada korisnik posjeti stranicu, poslužitelj ne zna ništa o prethodnim posjetama korisnika. Ovo je korisno jer omogućava poslužitelju da bude brži i efikasniji, međutim postoje tehnike kojima možemo na klijentskoj strani zapamtiti prethodne interakcije, npr. kroz kolačiće (eng. *cookies*) ili lokalno pohranjivanje (eng. *local storage*) te na taj način imati neki oblik stanja koji šaljemo s klijenta na poslužitelj.

Dakle, za sad je važno zapamtiti da klijent šalje HTTP zahtjeve poslužitelju, čeka odgovor i zatim prikazuje odgovor krajnjem korisniku. Naravno, to ne mora biti i vrlo često i nije (1 <-> 1) komunikacija, već klijent može slati različite zahtjeve na različite poslužitelje. No mi ćemo u sklopu ovog kolegija raditi samo s jednim poslužiteljem i jednim klijentom.

## 5.1 HTTP zahtjev (eng. HTTP request)

HTTP zahtjev predstavlja zahtjev klijenta poslužitelju, npr. klijent (web preglednik) zahtjeva određeni web resurs (npr. HTML stranicu) od poslužitelja.

HTTP zahtjev sastoji se od nekoliko dijelova od kojih su neki **obavezni**, a neki **opcionalni**:



### 5.1.1 Obavezni dijelovi HTTP zahtjeva

Kako bi klijent poslao najjednostavniji mogući HTTP zahtjev, potrebno je navesti kome šaljemo zahtjev (eng. *Host Header*) te što želimo (eng. *Request Line*).

Obavezni dijelovi HTTP zahtjeva	Opis	Primjer
Request Line	Sastoji se od HTTP <b>metode</b> , traženog <b>URI</b> i HTTP <b>verzije</b>	<code>GET /index.html</code> <code>HTTP/1.1</code>
Host Header	Navodi se naziv domene ili IP adresa poslužitelja	<code>Host:</code> <code>www.example.com</code>

Međutim, **Host Header** je ustvari jedini obavezni dio zahtjeva, ali to u pravilu ne želimo raditi. Idemo demonstrirati programom `curl` kako izgleda najjednostavniji HTTP zahtjev. Ovaj program je u pravilu dostupan na svakom OS-u, a koristi se za slanje HTTP zahtjeva iz terminala. Možete provjeriti imate li ga instaliranog s naredbom `curl --version`.

Idemo poslati najjednostavniji mogući HTTP zahtjev prema `http://www.google.com`:

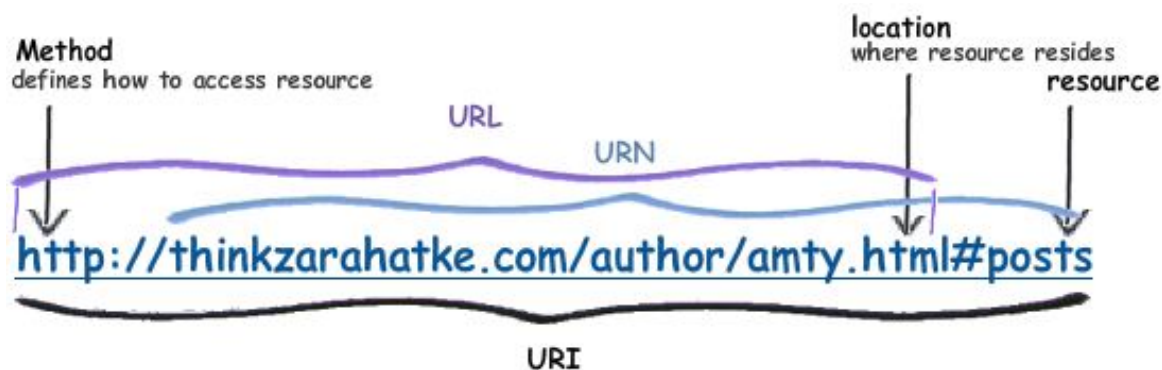
```
curl http://www.google.com
```



Uočite što smo dobili - HTML stranicu koja definira Googleovu početnu stranicu. `curl` je automatski odabrao `GET` metodu, ali metodu možemo navesti i eksplicitno opcijom `-X`:

```
curl -X GET http://www.google.com
```

Koji smo **URI** (eng. *Uniform Resource Identifier*) dohvatili u ovom slučaju? URI predstavlja jedinstveni identifikator elektroničkog resursa. URI se često koristi kao sinonim za URL (eng. *Uniform Resource Locator*), međutim URI je općenitiji pojam koji uključuje i URL i URN (eng. *Uniform Resource Name*). Točnije, URL i URN su podskup URI-a.



U ovoj skripti će se često koristiti termin URI.

Dakle što je ovdje URI? `http://www.google.com`

Sve navedeno, ali što onda dohvaćamo? Odgovor je osnovni endpoint definiran putanjom `/`. Vidimo da je Google definirao osnovni endpoint kao početnu stranicu, to je jasno, ali sad već možemo i pretpostaviti kako se zove datoteka koju dohvaćamo - `index.html`. Endpoint ili ruta `/` je u pravilu početna stranica web stranice, a datoteka `index.html` je osnovna HTML stranica koja se prikazuje korisniku.

```
curl -X GET http://www.google.com/index.html
```

Radi! 🎉

Što ako probamo dohvatiti nešto što ne postoji? Na primjer, `http://www.google.com/about_me.html`

```
curl -X GET http://www.google.com/about_me.html
```

Vidimo da kao odgovor dobivamo HTML stranicu s porukom "404. That's an error. The requested URL was not found on this server. That's all we know.". Ako otvorimo u web pregledniku, ona izgleda ovako:



404. That's an error.

The requested URL /about\_me was not found on this server. That's all we know.



Dakle, **Request Line** se sastoji od HTTP **metode**, traženog **URI** i HTTP **verzije**.

Dijelovi Request Line komponente	Opis	Primjer
HTTP metoda	Akcija koju klijent želi izvršiti (npr. GET, POST, PUT)	GET
URI zahtjeva	Specifični resurs na poslužitelju koji klijent traži	/over/here? name=something
HTTP verzija	Verzija HTTP-a koja se koristi u zahtjevu	HTTP/1.1

Verziju HTTP-a možemo navesti i eksplicitno, međutim u pravilu se automatski koristi `HTTP/1.1`.

## Vježba 1 - HTTP zahtjev prema našem Expressu

Pokrenite Express poslužitelj koji smo izradili i pošaljite HTTP zahtjev prema njemu koristeći `curl` program. Koji odgovor očekujete?

### 5.1.2 Opcionalni dijelovi HTTP zahtjeva

Osim obaveznih dijelova HTTP zahtjeva, postoje i opcionalni dijelovi koji se koriste za slanje dodatnih informacija poslužitelju. Konkretno, možemo poslati **HTTP zaglavlja** (eng. *HTTP headers*) i **HTTP tijelo** (eng. *HTTP body*).

Opcionalni dijelovi HTTP zahtjeva	Opis	Primjer
Opcionalna zaglavlja zahtjeva (eng. <i>Request Headers</i> )	Ključ-vrijednost parovi koji pružaju dodatne informacije o zahtjevu (zamislimo ih kao metapodatke)	Content-Type: application/json Authorization: Bearer <token> Accept: text/html
Tijelo zahtjeva (eng. <i>Request Body</i> )	Stvarni podaci koje šaljemo, često u JSON formatu, a tipično se koristi u metodama poput POST, PUT, itd.	{ "username": "Pero", "password": "password123" }

Zaglavlja ćemo raditi detaljnije na nekim drugim vježbama, za sada morate znati samo da postoje i da se koriste za slanje dodatnih informacija poslužitelju.

Tijelo se koristi u metodama poput POST, PUT, DELETE, PATCH, itd. gdje šaljemo podatke poslužitelju. Na primjer, kada se korisnik registrira na web stranici, šaljemo podatke kao što su **korisničko ime**, **lozinka**, **e-mail**, itd. u tijelu zahtjeva. Tijelo se može poslati u različitim formatima podataka, za sada nas zanima **JSON format**.

Kako ćemo definirati tijelo zahtjeva iznad kao JSON?

```
{
  "korisniko_ime": "pero_peric",
  "lozinka": "password123",
  "email": "pperic@gmail.com"
}
```

Recimo da naš poslužitelj ima definirani endpoint `/registracija` koji očekuje ove podatke. Dakle, korisnik ne traži nikakav resurs od poslužitelja pa niti HTML stranicu, već isključivo šalje podatke poslužitelju i očekuje nekakav odgovor (ne resurs). Ovakav endpoint definiramo `POST` metodom koja nam dozvoljava slanje podataka kroz tijelo zahtjeva, za razliku od `GET` metode.

Koristeći `curl` program, možemo poslati tijelo zahtjeva kroz opciju `-d`:

```
curl -X POST http://www.nas-super-server.com/registracija -d '{"korisniko_ime":
"pero_peric", "lozinka": "password123", "email": "pperic@gmail.com"}'
```

Naravno, ovo neće raditi.

Više o HTTP zahtjevima možete pročitati na [MDN web dokumentaciji](#).

## 5.2 HTTP odgovor (eng. HTTP response)

HTTP odgovor predstavlja odgovor poslužitelja klijentu, npr. poslužitelj šalje HTML stranicu ili JSON podatke klijentu. HTTP odgovor sastoji se od nekoliko dijelova od kojih su, kao i kod zahtjeva, neki **obavezni**, a neki **opcionalni**:



### RESPONSE



### 5.2.1 Obavezni dijelovi HTTP odgovora

Kako bi poslužitelj poslao najjednostavniji mogući HTTP odgovor, potrebno je navesti **HTTP verziju**, **statusni kod** i **statusni tekst** kao i obavezna zaglavlja odgovora (eng. *Response headers*).

Obavezni dijelovi HTTP odgovora	Opis	Primjer
Status Line	Sadrži HTTP <b>verziju</b> , <b>statusni kod</b> (eng. <i>status code</i> ) i <b>reason phrase</b>	HTTP/1.1 200 OK HTTP/1.1 404 Not Found
Obavezna zaglavlja odgovora (eng. <i>Response headers</i> )	Pruža obavezne metapodatke o odgovoru (npr. Content-Type)	Content-Type: application/json Date: Mon, 21 Oct 2024 10:32:45 GMT

Kod **Status Line** komponente, najzanimljiviji nam je **statusni kod**. Vjerojatno smo se svi do sad susreli sa statusnim kodovima koje nam je vratio neki poslužitelj.

Primjerice, poznati statusni kod `404` označava da traženi resurs nije pronađen, odnosno da je korisnik poslao zahtjev za resurs koji ne postoji.

Statusni kod `503` označava grešku na poslužitelju, odnosno da poslužitelj trenutno nije dostupan.

U grubo, brojevi ovih kodova označavaju različite situacije koje se mogu dogoditi prilikom slanja zahtjeva poslužitelju:

- `1xx` (100 - 199) - Informacijski odgovori (eng. *Informational responses*): Poslužitelj je primio zahtjev te ga i dalje obrađuje
- `2xx` (200 - 299) - Odgovori uspjeha (eng. *Successful responses*): Zahtjev klijenta uspješno primljen i obrađen
- `3xx` (300 - 399) - Odgovori preusmjerenja (eng. *Redirection messages*): Ova skupina kodova govori klijentu da mora poduzeti dodatne radnje kako bi dovršio zahtjev
- `4xx` (400 - 499) - Greške na strani klijenta (eng. *Client error responses*): Sadrži statusne kodove koji se odnose na greške nastale na klijentskoj strani
- `5xx` (500 - 599) - Greške na strani poslužitelja (eng. *Server error responses*): Sadrži statusne kodove koji se odnose na greške nastale na poslužiteljskoj strani

Više o statusnim kodovima uskoro, a možete ih pročitati i sami na [MDN web dokumentaciji](#).

*reason phrase* odnosi se na kratki opis statusnog koda, npr. `OK` za statusni kod `200` ili `Not Found` za statusni kod `404`. Ovaj dio u pravilu nikad ne želimo mijenjati.

## 5.2.2 Opcionalni dijelovi HTTP odgovora

Osim obaveznih dijelova HTTP odgovora, postoje i opcionalni dijelovi koji se koriste za slanje dodatnih informacija klijentu. Konkretno, možemo poslati **HTTP zaglavlja** (eng. *HTTP headers*) i **HTTP tijelo** (eng. *HTTP body*).

Component	Description	Example
<b>Tijelo odgovora (eng. Response body)</b>	Stvarni podaci koji se vraćaju korisniku, npr. u JSON ili XML formatima	<pre>{ "message": "Success",   "data": { "id": 1, "name":             "John" } }</pre>
<b>Opcionalna zaglavlja odgovora (eng. Response headers)</b>	Pružaju opcionalne metapodatke o odgovoru (npr. Set-Cookie)	<pre>Set-Cookie: sessionId=abc123 Cache-Control: no-cache</pre>

## Vježba 2: Kako vidjeti cijeli HTTP odgovor?

Pošaljite ponovo zahtjev programom `curl` na Express poslužitelj koji smo definirali.

```
curl http://localhost:3000
```

Kao odgovor dobili smo tijelo s porukom `"Hello, world!"`. Koji statusni kod očekujete? 🤔

Provjerit ćemo obavezna zaglavlja i statusni kod koji smo dobili kao odgovor kako bi vidjeli što Express radi u pozadini. Možemo koristiti opciju `-i` kako bismo vidjeli ukupan HTTP odgovor:

```
curl -i http://localhost:3000
```

Možemo vidjeti da cijeli HTTP odgovor ustvari izgleda ovako:

```
HTTP/1.1 200 OK  
  
X-Powered-By: Express  
Content-Type: text/html; charset=utf-8  
Content-Length: 13  
ETag: W/"d-1DpwLQbzRZmu4fjaJvn3KWax1pk"  
Date: Mon, 21 Oct 2024 13:37:34 GMT  
Connection: keep-alive  
Keep-Alive: timeout=5  
  
Hello, world!
```

- Prvi dio je **Status Line** koji sadrži HTTP **verziju**, **statusni kod** i **reason phrase**.
- Drugi dio sadrži **zaglavlja odgovora** koja pružaju metapodatke o odgovoru.
- Treći dio je **tijelo odgovora** koje sadrži stvarni sadržaj koji se šalje klijentu.

## Samostalni zadatak za Vježbu 1

Izmijenite vaš Express poslužitelj tako da:

1. Nadogradite postojeću GET rutu `/` koja sad mora vratiti HTML stranicu s `<h1>` naslovom "Hello, Express!".

2. Dodate još jednu GET rutu `/about` koja će vratiti HTML stranicu s `<h1>` naslovom "Ovo je stranica o nama!".

Objekte HTML stranice pohranite u direktorij `/public`.

Kako biste vratili podatke u obliku HTML stranice, koristite `res.sendFile()` metodu.

Sintaksa:

```
res.sendFile(__dirname + 'putanja_do_datoteke');
```

3. Dodajte i posljednju GET rutu `/users` koja će vratiti korisnike u JSON formatu. Korisnike pohranite u polju kao objekte s atributima `id`, `ime` i `prezime`. Dodajte barem 3 korisnika. Kako biste vratili korisnike u JSON formatu, koristite `res.json()` metodu.

Testirajte u web pregledniku i s programom `curl` sve tri rute.

Testirajte poslužitelj i u izrađenom Postman okruženju.

Kada završite, pohranite promjene na GitHub repozitorij s komentarom "Samostalni zadatak za vježbu 1".