

# Web aplikacije (WA)

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

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

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



Fakultet informatike u Puli

## (2) Usmjeravanje na Express poslužitelju

#2

WA

Usmjeravanje (eng. routing) se odnosi na određivanje kako će krajne rute koje definiramo na našoj poslužiteljskoj strani odgovarati na dolazne zahtjeve klijenata. U prošloj skripti smo već definirali osnovni primjer usmjeravanja za nekoliko GET ruta i posluživali smo statične datoteke i jednostavne JSON objekte. Danas ćete naučiti kako definirati složenije usmjeravanje kroz sve HTTP metode, koja su pravila usmjeravanja i dodatni parametri koje koristimo. Također, implementirat ćemo neke osnovne validacije podataka na našem poslužitelju.

Posljednje ažurirano: 4.11.2025.

## Sadržaj

- [Web aplikacije \(WA\)](#)
- [\(2\) Usmjeravanje na Express poslužitelju](#)
  - [Sadržaj](#)
- [1. Ponavljanje](#)
- [2. Osnovno usmjeravanje](#)
  - [2.1 `GET` metoda i parametri](#)
  - [2.2 `POST` metoda i slanje podataka](#)
    - [2.2.1 Kako slati `POST` zahtjeve jednostavnije?](#)
  - [Vježba 1 - Naručivanje više pizze](#)
  - [Vježba 2 - Zanima nas i adresa dostave](#)
  - [2.3 `PUT` i `PATCH` metode](#)
    - [2.3.1 `PUT` metoda](#)
    - [2.3.2 `PATCH` metoda](#)
  - [2.4 `DELETE` metoda](#)

- [2.5 Kada koristiti koju `HTTP` metodu?](#)
- [3. Router objekt i organizacija ruta](#)
  - [3.1 Kako koristiti `Router` objekt?](#)
  - [3.2 Bolje strukturiranje aplikacije](#)
  - [Vježba 3 - Strukturiranje narudžbi](#) ➔
- [4. Statusni kodovi u HTTP odgovorima](#)
  - [4.1 Kako koristiti statusne kodove u Expressu?](#)
  - [Vježba 4 - Korištenje statusnih kodova u pizzeriji](#) 4 0 4
- [Samostalni zadatak za Vježbu 2](#)

# 1. Ponavljanje

Nastavljamo s radom na Express poslužitelju, na ovim ćemo vježbama detaljnije proučiti **usmjerenje i obradu zahtjeva** u Express aplikacijama.

**Usmjerenje** (*eng. routing*) se odnosi na određivanje kako će krajnje rute koje definiramo na našoj poslužiteljskoj strani odgovarati na dolazne zahtjeve klijenata. U prošloj skripti smo već definirali osnovni primjer usmjerenja za nekoliko ruta i posluživali smo statične datoteke i JSON objekte.

**Osnovna sintaksa** za definiranje ruta u Express aplikacijama je sljedeća:

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

gdje je:

- `app` je instanca Express aplikacije
- `METHOD` je HTTP metoda (npr. GET, POST, PUT, DELETE, itd.) koju želimo posluživati
- `PATH` je putanja na koju želimo reagirati (npr. `/`, `/about`, `/contact`, itd.)
- `HANDLER` je callback funkcija koja se izvršava kada se zahtjev podudara s definiranom rutom

Tako smo definirali rutu za početnu stranicu:

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

// odnosno

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

`PATH` koji smo ovdje koristili je `/`, što znači da će se ova ruta pokrenuti kada korisnik posjeti početnu stranicu našeg web sjedišta.

U ovom primjeru koristili smo `GET` metodu, za koju smo općenito rekli da se koristi kada korisnik želi dohvatiti neki resurs s poslužitelja, bio on HTML dokument, slika, CSS datoteka, JavaScript datoteka, JSON objekt, itd.

## 2. Osnovno usmjeravanje

### 2.1 `GET` metoda i parametri

U prošloj smo skripti već naučili kako koristiti `GET` metodu za dohvat resursa s poslužitelja. U ovom ćemo primjeru proširiti našu aplikaciju tako da možemo dohvatiti resurs s poslužitelja na temelju **parametara** koje korisnik prenosi u URL-u.

Osnovna sintaksa za definiranje GET rute je sljedeća:

```
app.get(PATH, (req, res) => {
  // Ovdje pišemo kod koji će se izvršiti kada korisnik posjeti PATH
});
```

Primjerice, zamislimo da radimo **aplikaciju za naručivanje pizze** 🍕. Recimo da korisnik odluči pogledati koje su pizze dostupne, želimo da dohvati sve dostupne pizze definirane na našem poslužitelju. U tom slučaju, korisnik bi mogao posjetiti URL `/pizze`.

```
app.get('/pizze', (req, res) => {
  res.send('Ovdje su sve dostupne pizze!');
});
```

Rekli smo da možemo koristiti metodu `res.json` kako bismo poslali JSON objekt korisniku. U ovom slučaju, možemo poslati listu dostupnih pizza kao JSON objekt:

No prvo moramo definirati listu dostupnih pizza:

```
const pizze = [
  { id: 1, naziv: 'Margherita', cijena: 6.5 },
  { id: 2, naziv: 'Capricciosa', cijena: 8.0 },
  { id: 3, naziv: 'Quattro formaggi', cijena: 10.0 },
  { id: 4, naziv: 'Šunka sir', cijena: 7.0 },
  { id: 5, naziv: 'Vegetariana', cijena: 9.0 }
];

app.get('/pizze', (req, res) => {
  res.json(pizze);
});
```

Kada korisnik posjeti URL `/pizze`, dobit će JSON objekt s listom dostupnih pizza. Ako nemate instaliranu jednu od ekstenzija za web preglednik koje omogućuju pregled JSON objekata u pregledniku, JSON će vam se prikazivati kao običan tekst (*eng. raw*) bez formatiranja, što može biti nepregledno. Preporuka je preuzeti jednu od JSON Formatter ekstenzija za preglednik, npr. [JSON Formatter](#) za Chromium preglednike.

Što ako korisnik želi dohvatiti **samo jednu pizzu**, a ne sve? Kako ćemo definirati rutu za dohvat jedne pizze?

Možemo definirati posebnu rutu za svaku pizzu, npr. `/margherita`, `/capricciosa`, `/quattro-formaggi`, itd. Međutim, koliko je to rješenje pametno?

Možda bi mogli proći s ovim ako restoran ima 4-5 pizza, ili 15. Što ako restoran ima 50 pizza? ili 100?

Navedeno je primjer lošeg dizajna i nepotrebno ponavljanje koda. Umjesto toga, možemo koristiti **parametre** u URL-u kako bismo dohvatali jednu pizzu.

**URL parametar** je dio URL-a koji se koristi za prenošenje informacija između klijenta i poslužitelja. URL parametri se definiraju u URL-u s prefiksom `:`.

Primjerice, ako možemo definirati rutu `/pizze/:id` koja će dohvatiti pizzu s određenim `id` parametrom:

```
app.get('/pizze/:id', (req, res) => {
  res.json(pizze);
});
```

Kako bi sad dohvatali određenu pizzu, moramo poslati zahtjev u obliku `/pizze/1`, `/pizze/2`, `/pizze/3`, itd. Nećemo navoditi eksplicitno `"id"` u URL-U, već nam služi kao svojevrsni **placeholder**.

Pošaljite zahtjev na `/pizze/1` i provjerite rezultat.

Zašto nismo dobili dobili podatke samo za jednu pizzu iako smo poslali `id` parametar?

► Spoiler alert! Odgovor na pitanje

Idemo sada definirati logiku koja će dohvatiti samo jednu pizzu na temelju `id` parametra. Za početak, stvari možemo odraditi na "ručni" način, tj. prolaskom kroz sve dostupne pizze i pronašljujući one koja ima traženi `id`.

`id` iz URL-a je tipa string i možemo ga jednostavno dohvatiti pomoću `req.params` objekta.

```
app.get('/pizze/:id', (req, res) => {
  const id_pizza = req.params.id; // dohvaćamo id parametar iz URL-a

  for (pizza of pizze) {
    if (pizza.id == id_pizza) {
      // ako smo pronašli podudaranje u id-u
      res.json(pizza); // vrati objekt pizze kao rezultat
    }
  }
});
```

Sada kada pošaljemo zahtjev na `/pizze/1`, dobit ćemo JSON objekt s podacima o pizzi s `id`-om 1, odnosno Margheriti.

```
curl -X GET http://localhost:3000/pizze/1
```

Rezultat:

```
{  
  "id": 1,  
  "naziv": "Margherita",  
  "cijena": 6.5  
}
```

Naš endpoint `/pizze` funkcionira i dalje i možemo ga pozvati bez parametara:

```
curl -X GET http://localhost:3000/pizze
```

Rezultat:

```
[  
  {  
    "id": 1,  
    "naziv": "Margherita",  
    "cijena": 6.5  
  },  
  {  
    "id": 2,  
    "naziv": "Capricciosa",  
    "cijena": 8  
  },  
  {  
    "id": 3,  
    "naziv": "Quattro formaggi",  
    "cijena": 10  
  },  
  {  
    "id": 4,  
    "naziv": "Šunka sir",  
    "cijena": 7  
  },  
  {  
    "id": 5,  
    "naziv": "Vegetariana",  
    "cijena": 9  
  }  
]
```

Kod možemo pojednostaviti korištenjem metode `find` koja će nam vratiti prvi element koji zadovoljava uvjet:

```

app.get('/pizze/:id', (req, res) => {
  const id_pizza = req.params.id; // dohvaćamo id parametar iz URL-a

  const pizza = pizze.find(pizza => pizza.id == id_pizza); // pronađimo pizzu s traženim
  id-em

  res.json(pizza);
});

```

Što ako korisnik pošalje zahtjev za pizzu koja ne postoji? Kako ćemo riješiti taj slučaj? 🤔

► Spoiler alert! Odgovor na pitanje

```

app.get('/pizze/:id', (req, res) => {
  const id_pizza = req.params.id; // dohvaćamo id parametar iz URL-a

  const pizza = pizze.find(pizza => pizza.id == id_pizza);

  if (pizza) {
    // ako je pronađeno podudaranje, vratimo pizza objekt
    res.json(pizza);
  } else {
    // ako je rezultat undefined, vratimo poruku da pizza ne postoji
    res.json({ message: 'Pizza s traženim ID-em ne postoji.' });
  }
});

```

Sada kada pošaljemo zahtjev na `/pizze/6`, dobit ćemo poruku da pizza s traženim ID-em ne postoji.

```
curl -X GET http://localhost:3000/pizze/6
```

Rezultat:

```
{
  "message": "Pizza s traženim ID-em ne postoji."
}
```

Što ako korisnik pošalje zahtjev na `/pizze/vegetariana`? Kako ćemo riješiti taj slučaj? 🤔

► Spoiler alert! Odgovor na pitanje

Možemo koristiti metodu `isNaN` (is Not a Number) kako bismo provjerili je li `id` parametar broj:

```

app.get('/pizze/:id', (req, res) => {
  const id_pizza = req.params.id;

  if (isNaN(id_pizza)) {
    // provjeravamo je li id_pizza "Not a Number"
    res.json({ message: 'Proslijedili ste parametar id koji nije broj!' });
}

```

```

        return;
    }

    const pizza = pizze.find(pizza => pizza.id == id_pizza);

    if (pizza) {
        res.json(pizza);
    } else {
        res.json({ message: 'Pizza s traženim ID-em ne postoji.' });
    }
);

```

## 2.2 POST metoda i slanje podataka

Do sada smo koristili `GET` metodu za dohvrat resursa s poslužitelja. Sada ćemo naučiti kako koristiti `POST` metodu za slanje podataka na poslužitelj.

`POST` metoda se koristi kada korisnik želi poslati podatke na poslužitelj, npr. kada korisnik želi **izraditi novi resurs na poslužitelju**, a podaci se šalju u **tijelu zahtjeva** (*eng. request body*).

Osnovna sintaksa za definiranje POST rute je sljedeća:

```

app.post(PATH, (req, res) => {
    // Ovdje pišemo kod koji će se izvršiti kada korisnik pošalje POST zahtjev na PATH
});

```

Vratimo se na primjer aplikacije za naručivanje pizze. Zamislimo da korisnik želi **naručiti pizzu**. Kako bismo omogućili korisniku da naruči pizzu, moramo definirati POST rutu koja će omogućiti korisniku da nekako pošalje podatke o narudžbi na poslužitelj.

Idemo napisati kostur POST rute za naručivanje pizze:

```

app.post('/naruci', (req, res) => {
    // Ovdje ćemo napisati logiku za naručivanje pizze
});

```

Ako otvorite ovu rutu u pregledniku, dobit ćete poruku `"Cannot GET /naruci"`. To je zato što smo definirali POST rutu, a pokušavamo je otvoriti u pregledniku, što će automatski poslati GET zahtjev!

Možemo dodati jednostavnu poruku koja će korisniku reći da je narudžba uspješno zaprimljena:

```

app.post('/naruci', (req, res) => {
    res.send('Vaša narudžba je uspješno zaprimljena!');
});

```

Zahtjev možemo poslati kroz terminal aplikaciju `curl` koju smo koristili u prethodnim primjerima:

```
curl -X POST http://localhost:3000/naruci
```

Kako možemo poslati podatke o narudžbi kroz POST HTTP zahtjev? 🤔

Hoćemo li to raditi kroz parametre u URL-u?

```
//?  
app.post('/naruci/:id', (req, res) => {  
  res.send(`Zaprimio sam narudžbu za pizzu ${req.params.id}`);  
});
```

► Spoiler alert! Odgovor na pitanje

Kako bismo poslali veličinu pizze koju želimo naručiti?

```
// ?  
app.post('/naruci/:id/:velicina', (req, res) => {  
  res.send(`Zaprimio sam narudžbu za ${req.params.velicina} pizza ${req.params.id}`);  
});
```

Dva isječka koda iznad primjeri su jako loše prakse. Zašto?

- **URL parametri su javno vidljivi** i mogu sadržavati osjetljive informacije (kako ćemo poslati podatke o plaćanju?)
- **Kod postaje nečitljiv** i teško održiv
- **Nije skalabilno** (što ako želimo poslati još više podataka? Ili više pizza?! 🍕🍕🍕)
- **Nije standardizirano** (kako ćemo znati koji parametar odgovara kojem podatku?)

Dakle, rekli smo da podatke šaljemo u **tijelu zahtjeva** (eng. *request body*). Kako ćemo to napraviti?

U prvoj skripti smo već naučili da podaci koji se šalju u tijelu zahtjeva mogu biti u različitim formatima, npr. JSON, XML, HTML, itd. Mi ćemo u pravilu slati podatke u **JSON** formatu.

Međutim, u našem web pregledniku nemamo mogućnost slanja POST zahtjeva s tijelom zahtjeva kada direktno pristupamo URL-u neke rute poslužitelja. Možemo poslati kroz naš `curl` alat s opcijom `-d`:

```
curl -X POST http://localhost:3000/naruci -d '{"pizza": "Margherita", "velicina": "srednja"}'
```

Kako ćemo sada u našoj Express aplikaciji dohvatiti podatke koje je korisnik poslao u tijelu zahtjeva?

Podaci koje korisnik šalje u tijelu zahtjeva se nalaze u `req.body` objektu.

*Primjer:*

```
app.post('/naruci', (req, res) => {  
  const narudzba = req.body;  
  console.log('Primljeni podaci:', narudzba);  
  res.send('Vaša narudžba je uspješno zaprimljena!');  
});
```

Primjetit ćete da će se u konzoli ispisati poruka "Primljeni podaci: undefined". Razlog zašto se ne ispisuju podaci je taj što Express ne zna kako parsirati podatke u tijelu zahtjeva. Da bismo to omogućili, moramo koristiti **Middleware** koji će parsirati podatke u tijelu zahtjeva.

O middleware funkcijama više u sljedećim lekcijama (WA6 - Middleware funkcije), međutim za sada ćemo koristiti ugrađeni middleware `express.json()` koji će parsirati podatke u JSON formatu.

Jednostavno dodajemo na početku naše aplikacije, nakon definiranja instance aplikacije:

```
app.use(express.json());
```

Pokušajte ponovo. Vidjet ćete da podaci i dalje ne dolaze kada šaljemo kroz `curl`. Razlog je taj što `curl` ne šalje podatke u JSON formatu po *defaultu*, već to moramo specificirati u **zaglavljiju** našeg HTTP zahtjeva.

Zaglavla možemo specificirati pomoću opcije `-H`, a dodat ćemo zaglavljje `Content-Type: application/json`:

```
curl -X POST http://localhost:3000/naruci -H "Content-Type: application/json" -d
'{"pizza": "Margherita", "velicina": "srednja"}'
```

Ako ste upisali točno naredbu, trebali biste vidjeti ispis u konzoli:

```
Primljeni podaci: { pizza: 'Margherita', velicina: 'srednja' }
```

Sada kada imamo podatke o narudžbi, možemo ih koristiti u našoj aplikaciji. Na primjer, možemo poslati korisniku poruku s informacijama o narudžbi:

```
app.post('/naruci', (req, res) => {
  const narudzba = req.body;
  console.log('Primljeni podaci:', narudzba);
  res.send(`Vaša narudžba za ${narudzba.pizza} (${narudzba.velicina}) je uspješno
  zaprimljena!`);
});
```

Što ako korisnik ne pošalje podatke o pizzi ili veličini pizze? Kako ćemo riješiti taj slučaj? 😊

Možemo izvući ključeve JavaScript objekta kroz metodu `Object.keys` i provjeriti jesu li svi ključevi prisutni:

```

app.post('/naruci', (req, res) => {
  const narudzba = req.body;
  const kljucevi = Object.keys(narudzba);

  if (!(kljucevi.includes('pizza') && kljucevi.includes('velicina'))) {
    res.send('Niste poslali sve potrebne podatke za narudžbu!');
    return;
  }

  res.send(`Vaša narudžba za ${narudzba.pizza} (${narudzba.velicina}) je uspješno
  zaprimljena!`);
});

```

Sada kada pošaljemo zahtjev bez podataka:

```
curl -X POST http://localhost:3000/naruci -H "Content-Type: application/json" -d '{}'
```

Ili s pogrešnim podacima:

```
curl -X POST http://localhost:3000/naruci -H "Content-Type: application/json" -d
'{"pizza": "Margherita", "cijena": 6.5}'
```

## 2.2.1 Kako slati **POST** zahtjeve jednostavnije?

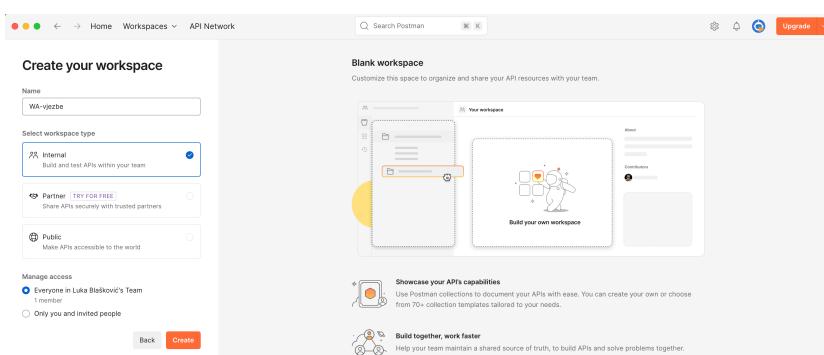
Kako ne bismo morali svaki put pisati `curl` naredbe za slanje POST zahtjeva, možemo koristiti alate koji nam omogućuje puno jednostavnije slanje HTTP zahtjeva s tijelom zahtjeva, zaglavljima i drugim opcijama.

Jedan od takvih alata je [Postman](#), koji je dostupan za sve platforme i omogućuje nam jednostavno slanje HTTP zahtjeva, testiranje API-ja, automatsko generiranje dokumentacije, itd.



Preuzmite Postman s [ovog linka](#). Potrebno je izraditi račun, ali je besplatan za korištenje.

Jednom kada se prijavite, morate napraviti novi radni prostor (*workspace*). Kliknite na `New Workspace` i unesite naziv radnog prostora. Možete ga nazvati `Web aplikacije - Vježbe`.



Odaberite **+** i dodajte novu kolekciju (`create collection`) koju možete nazvati **WA2** te dodajte novi zahtjev u kolekciju odabirom **"Add a request"**. Nazovite zahtjev **Jelovnik** i odaberite GET zahtjev (po defaultu je GET).

The screenshot shows the Postman interface. In the top left, there's a sidebar with 'Collections', 'Environments', 'Flows', and 'History'. A red arrow points from the 'New' button at the top to the 'Create Collection' button. Another red arrow points from the 'HTTP' icon in the main workspace to the 'HTTP' section in the bottom right, which contains icons for Variable, Value, What's change, Local by defa, and various API types like GraphQL, AI, MCP, gRPC, WebSocket, Socket.IO, MQTT, Collection, Environment, Flow, Workspace, and Insights.

Vidjet ćete razno-razne opcije koje možete koristiti za slanje zahtjeva, kao što su **URL, HTTP metoda, zaglavljia, tijelo zahtjeva, autorizacija** itd.

Uočite da se unutar zaglavja već nalazi postavljeno čak 7 različitih zaglavja, dakle Postman automatski postavlja neka zaglavja za nas kako bi nam olakšao posao.

Pošaljite zahtjev na endpoint `/pizze` i vidjet ćete rezultat u obliku JSON objekta s dostupnim pizzama. Morate unijeti puni URL u formatu:

```
http://localhost:3000/pizze
```

Te odaberite metodu `GET` s padajućeg izbornika.

Ako je sve OK, ispod će vam se prikazati JSON objekt unutar **Body** sekcije, ali možete vidjeti i **zaglavljia koja su došla s odgovorom**.

The screenshot shows the Postman request details and response preview. At the top, it says 'WA2 / Jelovnik' with a red arrow pointing to the 'GET' method dropdown. Another red arrow points to the URL field containing 'http://localhost:3000/pizze'. Below this are tabs for 'Params', 'Authorization', 'Headers (7)', 'Body', 'Scripts', and 'Settings'. Under 'Headers (7)', there is a table for 'Query Params' with columns 'Key', 'Value', 'Description', and 'Bulk Edit'. The 'Body' tab shows a JSON response with 20 items, each containing an ID, name, and price. The 'Headers' tab shows the response headers: '200 OK', '22 ms', '453 B', and 'Save Response'. The 'Test Results' tab is also visible.

Key	Value	Description	Bulk Edit
Key	Value	Description	

```

1 [
2   {
3     "id": 1,
4     "naziv": "Margherita",
5     "cijena": 6.5
6   },
7   {
8     "id": 2,
9     "naziv": "Capricciosa",
10    "cijena": 8
11  },
12  {
13    "id": 3,
14    "naziv": "Quattro formaggi",
15    "cijena": 10
16  },
17  {
18    "id": 4,
19    "naziv": "Šunka sir",
20    "cijena": 7
21  }
22 ]
  
```

Dodajte preostale zahtjeve u svoju kolekciju:

- Zahtjev `Dohvati pizzu` s metodom `GET` i URL-om `http://localhost:3000/pizze/1`

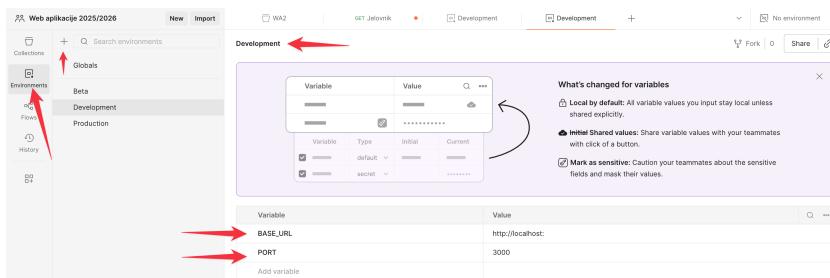
- Zahtjev `Naruči pizzu` s metodom `POST` i URL-om `http://localhost:3000/naruci`. U tijelo zahtjeva odaberite opciju `raw` i format `JSON`, te unesite sljedeći JSON objekt:

```
{
  "pizza": "Capricciosa",
  "velicina": "jumbo"
}
```

2

Ono što je još praktično postaviti je **Environment** varijabla za `base_url` i/ili `port` ako želite odvojiti, kako ne biste trebali svaki put unositi puni URL. Postman omogućava izradu raznih okruženja (*environments*) s različitim varijablama koje možete koristiti u svojim zahtjevima: npr. `Development`, `Staging`, `Production`, itd., ovisno za koje okruženje radite testove.

Odaberite `Environments` u izborniku lijevo te izradite novo okruženje `Development`. Dodajte novu varijablu `BASE_URL` s vrijednošću `http://localhost:`, i varijablu `PORT` s vrijednošću `3000`. Environment varijable je uobičajeno pisati velikim slovima - no nije nužno.



Varijable možete referencirati duplim vitičastim zagradama `{ {VAR_NAME} }`. Sada možete izmijeniti URL-ove svojih zahtjeva na sljedeći način:

- Zahtjev `Jelovnik: {{BASE_URL}}{{PORT}}/pizze`
- Zahtjev `Dohvati pizzu: {{BASE_URL}}{{PORT}}/pizze/1`
- Zahtjev `Naruči pizzu: {{BASE_URL}}{{PORT}}/naruci`

Postoji puno alternative Postmanu, npr. [Insomnia](#), [Paw](#), [Thunder Client](#), [HTTPie](#), od kojih se neki izvode na webu, a neki lokalno na računalu.

Ako hoćete, možete preuzeti i [Thunder Client](#) koji je dostupan kao ekstenzija za Visual Studio Code.



Otvorite Thunder Client ekstenziju i odaberite `New Request`. Unesite URL `http://localhost:3000/pizze` i odaberite metodu `GET`. Kliknite na `Send Request` i vidjet ćete isti rezultat kao i u Postmanu.

`POST` zahtjev možete poslati na isti način, samo odaberite metodu `POST` i unesite URL `http://localhost:3000/naruci`. U tijelo zahtjeva unesite JSON objekt s podacima o narudžbi:

```
{  
    "pizza": "Capricciosa",  
    "velicina": "jumbo"  
}
```

Trebali biste dobiti poruku: Vaša narudžba za Capricciosa (jumbo) je uspješno zaprimljena!.

## Vježba 1 - Naručivanje više pizze

Nadogradite Express poslužitelj na način da pohranjujete podatke o narudžbama "in-memory", odnosno u varijablu koja će se resetirati svaki put kada se poslužitelj ponovno pokrene.

Nadogradite POST rutu `/naruci` tako da očekuje od korisnika **polje objekata** s podacima o narudžbi. Svaki objekt mora sadržavati ključeve `pizza`, `velicina` i `kolicina`.

```
[  
  {  
    "pizza": "Capricciosa",  
    "velicina": "jumbo",  
    "kolicina": 1  
  },  
  {  
    "pizza": "Vegetariana",  
    "velicina": "srednja",  
    "kolicina": 2  
  }  
]
```

Ako neki od ključeva nedostaje, vratite korisniku poruku da nije poslao sve potrebne podatke.

Provjerite je li korisnik naručio pizzu koja postoji u vašem jelovniku. Ako korisnik naruči pizzu koja ne postoji, vratite korisniku poruku da jedna ili više pizza koju je naručio ne postoji

Ako korisnik pošalje podatke u ispravnom formatu, dodajte narudžbu u listu narudžbi i vratite korisniku poruku da je narudžba za pizze (izlistajte naručene nazive pizza) uspješno zaprimljena.

## Vježba 2 - Zanima nas i adresa dostave

Nadogradite POST rutu `/naruci` tako da očekuje od korisnika dodatne podatke o narudžbi, kao što su `prezime`, `adresa` i `broj_telefona`.

Na jednak način kao u vježbi 1, provjerite jesu li svi potrebni podaci poslani i jesu li sve pizze koje je korisnik naručio prisutne u vašem jelovniku.

*Primjer:* JSON objekt koji se šalje:

```
{  
  "narudzba": [  
    {  
      "pizza": "Capricciosa",  
      "velicina": "jumbo",  
      "prezime": "Kraljević",  
      "adresa": "Ulica 123",  
      "broj_telefona": "0987654321"  
    }  
  ]  
}
```

```

    "kolicina": 1
  },
  {
    "pizza": "Vegetariana",
    "velicina": "srednja",
    "kolicina": 2
  }
],
"klijent": {
  "prezime": "Perić",
  "adresa": "Alda Negrija 6",
  "broj_telefona": "0912345678"
}
}

```

Ako korisnik pošalje podatke u ispravnom formatu, dodajte narudžbu u listu narudžbi i vratite korisniku JSON poruku sa sljedećim podacima:

```

message: "Vaša narudžba za pizza_1_naziv (pizza_1_velicina) i pizza_2_naziv
(pizza_2_naziv) je uspješno zaprimljena!",
prezime: "Perić",
adresa: "Alda Negrija 6",
ukupna_cijena: izračunajte ukupnu cijenu narudžbe

```

## 2.3 PUT i PATCH metode

Sljedeće metode koje ćemo naučiti su **PUT** i **PATCH** metode. Obe metode se koriste za **ažuriranje resursa** na poslužitelju. Međutim, razlika između njih je u tome što **PUT** metoda **zamjenjuje cijeli resurs** novim podacima, dok **PATCH** metoda **ažurira samo određene dijelove resursa**.

### 2.3.1 PUT metoda

Krenimo s metodom **PUT**. Zahtjev s ovom HTTP metodom se koristi za ažuriranje cijelog resursa na poslužitelju. Kada klijent pošalje ovakav zahtjev, želi zamijeniti cijeli resurs novim podacima koje šalje u **tijelu zahtjeva**.

Ključni elementi:

- **zamjenjuje cijeli resurs:** Kada šaljete **PUT** zahtjev, poslužitelj očekuje da ćete uključiti **sve informacije** za taj resurs, čak i onda kada želite zamijeniti samo manji dio resursa (npr. nekoliko polja u objektu).
- **može se koristiti za stvaranje novog resursa:** Ako šaljete **PUT** zahtjev s podacima o resursu koji ne postoji, ovaj zahtjev se može koristiti za stvaranje novog resursa. Zašto? Zato što se u **URI** navodi identifikator resursa.

*Primjer:* Recimo da želite ažurirati podatke o pizzi s **id**-om 1. Slanjem **PUT** zahtjeva na **/pizze/1** poslužitelj očekuje da ćete poslati **sve podatke** o pizzi, uključujući **id**, **naziv**, **cijena**, itd.

```
const pizze = [
  { id: 1, naziv: 'Margherita', cijena: 6.5 },
  { id: 2, naziv: 'Capricciosa', cijena: 8.0 },
  { id: 3, naziv: 'Quattro formaggi', cijena: 10.0 },
  { id: 4, naziv: 'Šunka sir', cijena: 7.0 },
  { id: 5, naziv: 'Vegetariana', cijena: 9.0 }
];
```

Zahtjev bi dakle izgledao ovako:

```
curl -X PUT http://localhost:3000/pizze/1 -H "Content-Type: application/json" -d '{"id": 1, "naziv": "Margherita", "cijena": 7.0}'
```

Primijetite da smo ažurirali samo cijenu Margherite, ali smo morali poslati sve podatke o pizzi.

Rekli smo da možemo koristiti `PUT` metodu i za stvaranje novog resursa, s obzirom da se u `URI` navodi identifikator resursa, a u tijelu zahtjeva šaljemo sve podatke o resursu.

Primjer:

```
curl -X PUT http://localhost:3000/pizze/6 -H "Content-Type: application/json" -d '{"id": 6, "naziv": "Quattro stagioni", "cijena": 8.0}'
```

Ako možemo koristiti `PUT` metodu za stvaranje novog resursa, zašto onda koristimo `POST` metodu? 😊

U pravilu želimo koristiti `POST` metodu za stvaranje novog resursa. Zašto? Iako je moguće koristiti `PUT` metodu, primijetite da smo morali poslati sve podatke o resursu, uključujući `id`. Ako korisnik šalje podatke o resursu, ne bi trebao znati `id` resursa, **već bi ga trebao generirati poslužitelj**.

Kako bi izgledao `POST` zahtjev za dodavanje nove pizze u naš jelovnik? Uočite da ne šaljemo `id` pizze, već samo `naziv` i `cijenu`. Također pogledajte `URI` zahtjeva.

```
curl -X POST http://localhost:3000/pizze -H "Content-Type: application/json" -d '{"naziv": "Quattro stagioni", "cijena": 8.0}'
```

U Expressu možemo jednostavno definirati `PUT` rutu sljedećom sintaksom:

```
app.put(PATH, (req, res) => {
  // Ovdje pišemo kod koji će se izvršiti kada korisnik pošalje PUT zahtjev na PATH
});
```

Dakle sintaksa je ista kao i za `GET` i `POST` rute, samo što koristimo `app.put` umjesto `app.get` ili `app.post`.

Primjer metode `PUT` za ažuriranje podataka o pizzi:

```
app.put('/pizze/:id', (req, res) => {
  const id_pizza = req.params.id;
  const nova_pizza = req.body;
```

```

nova_pizza.id = id_pizza; // dodajemo id pizze u objekt, u slučaju da ga klijent nije
poslao u tijelu zahtjeva

const index = pizze.findIndex(pizza => pizza.id == id_pizza);

if (index !== -1) {
  pizze[index] = nova_pizza;
  res.json(pizze[index]);
} else {
  res.json({ message: 'Pizza s traženim ID-em ne postoji.' });
}
});

```

## 2.3.2 PATCH metoda

PATCH metoda se koristi za **ažuriranje dijelova resursa** na poslužitelju. Za razliku od PUT metode koja zamjenjuje cijeli resurs, PATCH metoda se koristi kada želimo ažurirati samo **određene dijelove resursa**.

Primjer: Ako želimo ažurirati samo cijenu pizze s id-om 1, koristit ćemo PATCH metodu:

```

curl -X PATCH http://localhost:3000/pizze/1 -H "Content-Type: application/json" -d
'{"cijena": 7.0}'

```

Metodu PATCH ne želimo koristiti za stvaranje novog resursa, jer ne želimo stvoriti resurs s nepotpunim podacima. Primjerice, ako korisnik pošalje PATCH zahtjev na /pizze/6, a zaboravi poslati naziv pizze, ne želimo stvoriti novu pizzu s nepotpunim podacima.

U Expressu možemo definirati PATCH rutu na sljedeći način:

```

app.patch(PATH, (req, res) => {
  // Ovdje pišemo kod koji će se izvršiti kada korisnik pošalje PATCH zahtjev na PATH
});

```

Primjer metode PATCH za ažuriranje podataka o pizzi:

```

app.patch('/pizze/:id', (req, res) => {
  const id_pizza = req.params.id;
  const nova_pizza = req.body;

  const index = pizze.findIndex(pizza => pizza.id == id_pizza);

  if (index !== -1) {
    for (const key in nova_pizza) {
      pizze[index][key] = nova_pizza[key];
    }

    // ili
    // pizze[index] = { ...pizze[index], ...nova_pizza }; // spread operator

    res.json(pizze[index]);
  }
});

```

```
    } else {
      res.json({ message: 'Pizza s traženim ID-em ne postoji.' });
    }
});
```

## 2.4 DELETE metoda

Metoda `DELETE` se koristi za **brisanje resursa** na poslužitelju. Kada klijent pošalje ovakav zahtjev, poslužitelj briše resurs s identifikatorom koji je naveden u `URI` zahtjeva.

*Primjer:* Ako želimo obrisati pizzu s `id`-om 1, koristit ćemo `DELETE` metodu:

```
curl -X DELETE http://localhost:3000/pizze/1
```

U Expressu možemo definirati `DELETE` rutu na sljedeći način:

```
app.delete(PATH, (req, res) => {
  // Ovdje pišemo kod koji će se izvršiti kada korisnik pošalje DELETE zahtjev na PATH
});
```

*Primjer:* Metoda `DELETE` za brisanje podataka o pizzi:

```
app.delete('/pizze/:id', (req, res) => {
  const id_pizza = req.params.id;

  const index = pizze.findIndex(pizza => pizza.id == id_pizza);

  if (index !== -1) {
    pizze.splice(index, 1);
    res.json({ message: 'Pizza uspješno obrisana.' });
  } else {
    res.json({ message: 'Pizza s traženim ID-em ne postoji.' });
  }
});
```

## 2.5 Kada koristiti koju HTTP metodu?

Naučili smo kako koristiti sljedeće `HTTP` metode:

- `GET` - dohvati resurs (npr. `GET /pizze` ili `GET /pizze/1` ili `GET /narudzbe`)
- `POST` - stvori novi resurs (npr. `POST /pizze` ili `POST /narudzbe` ili `POST /login` s podacima za autentifikaciju)
- `PUT` - zamijeni cijeli resurs (npr. `PUT /pizze/1` ili `PUT /korisnici/1` s cijelim podacima o resursu)
- `PATCH` - ažuriraj dio resursa (npr. `PATCH /pizze/1` ili `PATCH /korisnici/1` s parcijalnim podacima o resursu)
- `DELETE` - obriši resurs (npr. `DELETE /pizze/1` ili `DELETE /korisnici/1` bez tijela zahtjeva)

Postoje još metode koje nismo spomenuli, kao što su `HEAD`, `OPTIONS`, `TRACE`, `CONNECT` itd. Međutim, ove metode su manje uobičajene i koriste se u specifičnim situacijama. Vi ih ne morate znati koristiti.

Iako je moguće koristiti bilo koju metodu za gotovo bilo koju akciju, ipak postoje pravila i dobre prakse koje se koriste u razvoju web aplikacija. Evo nekoliko smjernica:

- `GET` metodu koristimo za dohvrat resursa s poslužitelja. Ova metoda ne bi trebala imati nikakve druge efekte osim dohvata podataka. Primjerice, ako korisnik posjeti URL u pregledniku, očekujemo da će dobiti odgovor s podacima, ali ne očekujemo da će se nešto promjeniti na poslužitelju (npr. ažurirati podaci u bazi podataka).
- `POST` metodu koristimo za stvaranje novog resursa na poslužitelju. Ova metoda se koristi kada korisnik želi poslati podatke na poslužitelj, npr. kada korisnik želi stvoriti novu pizzu u našem jelovniku. Međutim, metodu koristimo i za druge akcije, poput autentifikacije korisnika kada korisnik želi u tijelu zahtjeva poslati korisničko ime i lozinku (prisjetimo se da je kod GET zahtjeva sve vidljivo u URL-u).
- `PUT` metodu koristimo za zamjenu cijelog resursa novim podacima. Ova metoda se koristi kada korisnik želi zamijeniti cijeli resurs novim podacima. Primjerice, kada korisnik želi ažurirati podatke o pizzi, ali mora poslati sve podatke o pizzi, uključujući `:id` pa i one podatke koji se ne mijenjaju.
- `PATCH` metodu koristimo za ažuriranje dijelova resursa. Ova metoda se koristi kada korisnik želi ažurirati samo određene dijelove resursa. Primjerice, kada korisnik želi ažurirati samo cijenu pizze, a ne i naziv pizze.
- `DELETE` metodu koristimo za brisanje resursa. Ova metoda se koristi kada korisnik želi obrisati resurs s poslužitelja. Primjerice, kada korisnik želi obrisati pizzu iz našeg jelovnika.

### 3. Router objekt i organizacija ruta

Prilikom razvoja ozbiljnijeg poslužitelja, vjerojatno ćemo morati definirati mnoštvo različitih ruta. Možemo vidjeti da naša `index.js` datoteka postaje sve veća i veća kako dodajemo nove rute.

Na primjer, za jednostavno dohvaćanje pizze i pizze po ID-u, potrebne su nam dvije rute:

```
app.get('/pizze', (req, res) => {
  // implementacija
});

app.get('/pizze/:id', (req, res) => {
  // implementacija
});
```

Što ako imamo još više ruta? Na primjer, rute za naručivanje pizze, ažuriranje podataka o pizzi, brisanje pizze, itd. Naša datoteka `index.js` postaje sve veća i teže ju je održavati.

Kako bismo olakšali organizaciju koda, poželjno je koristiti `Router` objekt koji nam omogućuje grupiranje ruta i definiranje ruta u zasebnim datotekama.

`Router` objekt jedna je od ključnih komponenti Expressa koja nam omogućuje grupiranje srodnih ruta. Na primjer, sve rute vezane uz pizze možemo grupirati u jedan `Router` objekt, ili sve rute vezane uz korisnike u drugi `Router` objekt.

## 3.1 Kako koristiti Router objekt?

Naš trenutni poslužitelj sastoji se od sljedećih datoteka:

```
.
```

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

Praktično je organizirati naše rute u zasebne datoteke. Na primjer, možemo imati datoteku `pizze.js` u kojoj ćemo definirati sve rute vezane uz pizze, ili datoteku `narudzbe.js` gdje ćemo definirati sve rute vezane uz narudžbe.

Dodatno, te datoteke možemo pohraniti u zajednički direktorij `routes` ili `router`.

Dodajmo direktorij `routes` u naš projekt i datoteku `pizze.js` unutar tog direktorija:

```
.
```

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

Unutar `pizze.js` datoteke moramo uključiti ponovo Express modul, ali i definirati `Router` objekt:

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

Kako bi naš `Router` objekt bio dostupan u `index.js` datoteci, moramo ga izvesti:

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

module.exports = router;
```

Međutim, dok nismo napisali puno koda, nije loše da se napokon prebacimo na novu ES6 sintaksu koju ste vjerojatno već pisali u VUE.js aplikacijama. **ECMAScript** (JavaScript ES) je standardizacija JavaScripta, a **ES6** je šesta verzija standarda koja je donijela puno novih značajki, uključujući i modernu sintaksu za organizaciju i strukturiranje modula.

U ES6 sintaksi, umjesto `module.exports` koristimo `export default`:

`export default` sintaksa omogućava nam izvoz jednog objekta, funkcije ili varijable iz modula. Kada koristimo `export default`, možemo uvesti taj objekt, funkciju ili varijablu u drugom modulu koristeći `import` sintaksu bez vitičastih zagrada (odnosno bez navođenja imena objekta kojeg uvozimo).

Prije svega, moramo ažurirati našu `package.json` datoteku kako bismo koristili ES6 sintaksu. Dodajte sljedeći redak u `package.json` datoteku:

```
"type": "module",
```

Sada možemo koristiti ES6 sintaksu u našem projektu. Idemo prvo ispraviti `index.js` datoteku.

U `index.js` datoteci, umjesto `require` koristimo `import` sintaksu:

```
import object from 'module'; // umjesto const object = require('module');

// odnosno

import express from 'express'; // umjesto const express = require('express');
```

Ako imamo više izvoza iz jednog modula, možemo ih uvesti koristeći vitičaste zagrade:

```
import { object1, object2 } from 'module'; // umjesto const { object1, object2 } =
require('module');
```

Vratimo se na `pizze.js`, gdje ćemo također koristiti ES6 sintaksu:

```
import express from 'express';
const router = express.Router();

export default router;
```

Kako ovaj `Router` objekt možemo zamisliti kao malu aplikaciju unutar naše glavne aplikacije, možemo dodati rute na isti način kao što smo to radili u `index.js` datoteci, ali ćemo umjesto `app` koristiti `router`:

Dodat ćemo rutu za dohvata svih pizza:

```
import express from 'express';
const router = express.Router();

const pizze = [
  { id: 1, naziv: 'Margerita', cijena: 7.0 },
  { id: 2, naziv: 'Capricciosa', cijena: 9.0 },
  { id: 3, naziv: 'Šunka sir', cijena: 8.0 },
  { id: 4, naziv: 'Vegetariana', cijena: 12.0 },
  { id: 5, naziv: 'Quattro formaggi', cijena: 15.0 }
];

router.get('/', (req, res) => {
  // ruta za dohvata svih pizza, pišemo router.get umjesto app.get
  res.json(pizze);
});

export default router;
```

Na isti način kopirajte i rutu za dohvrat pizze po ID-u.

Jednom kad to imamo, možemo uvesti `Router` objekt u našu `index.js` datoteku s proizvoljnim imenom:

```
import pizzaRouter from './routes/pizza.js';
```

Zatim samo moramo reći našoj aplikaciji da koristi taj `Router` objekt:

```
app.use(pizzaRouter);
```

To je to! Testirajte dohvaćanje svih pizza i pizze po ID-u koristeći Postman ili Thunder Client.

## 3.2 Bolje strukturiranje aplikacije

Kako bismo još bolje strukturirali naš poslužitelj, možemo napraviti još nekoliko stvari.

Prvo, kopirajmo preostale `/pizza` rute iz `index.js` datoteke u `pizza.js` datoteku.

Dakle, u `pizza.js` datoteci imamo sljedeće rute:

```
router.get('/pizza');
router.get('/pizza/:id');
router.put('/pizza/:id');
router.patch('/pizza/:id');
app.delete('/pizza/:id');
```

Što je redundantno u ovim rutama? 😊

► Spoiler alert! Odgovor na pitanje

To ćemo definirati u `index.js` datoteci kada koristimo `pizzaRouter`:

```
app.use('/pizza', pizzaRouter);
```

Sada naše rute u `pizza.js` datoteci mijenjamo na sljedeći način:

```
router.get('/');
router.get('/:id');
router.put('/:id');
router.patch('/:id');
router.delete('/:id');
```

## Vježba 3 - Strukturiranje narudžbi ➡️🍕

Strukturirajte narudžbe na jednak način kao što smo to napravili za pizze. Definirajte `narudzbe.js` datoteku unutar `routes` direktorija i prebacite polje narudžbi i sve rute vezane uz narudžbe u tu datoteku.

Kako se radi o resursu `narudzbe`, prefiks `/narudzbe` dodajte samo jednom, na početku svih ruta. Dakle `URI` za dodavanje narudžbe trebao bi izgledati ovako: `/narudzbe`, a ne `/narudzbe/naruci` ili samo `/naruci`.

Kada završite, uvezite `narudzbeRouter` u `index.js` datoteku i koristite ga u aplikaciji. Vaša `index.js` datoteka trebala bi izgledati ovako:

```
import express from 'express';
import pizzaRouter from './routes/pizza.js';
import narudzbeRouter from './routes/narudzbe.js';

const app = express();

const PORT = 3000;

app.use(express.json());

app.use('/pizza', pizzaRouter);
app.use('/narudzbe', narudzbeRouter);

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

## 4. Statusni kodovi u HTTP odgovorima

**Statusni kodovi** (eng. *HTTP status codes*) su brojevi koji se koriste u **HTTP odgovorima** kako bi klijentu dali informaciju u kojem je stanju zahtjev koji je poslao. Drugim riječima, ako klijent pošalje zahtjev koji rezultira greškom, poslužitelj uz odgovarajuću poruku vraća i statusni kod koji označava vrstu greške.

Ako se podsjetimo statusnih kodova iz prve skripte, rekli smo da ih možemo podijeliti u sljedeće kategorije:

- `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

Statusni kodovi neizbjegjan su dio svakog HTTP standarda, a njihovom primjenom standardiziramo komunikaciju između klijenta i poslužitelja. Na taj način, klijent može interpretirati odgovor poslužitelja i ovisno o statusnom kodu poduzeti odgovarajuće radnje.

Na primjer, ako pošaljemo klijentu JSON poruku `message: "Pizza nije pronađena"` ili `message: "Greška prilikom obrade narudžbe"`, potrebno je posebno tumačiti te poruke na klijentskoj strani. Međutim, to ne želimo raditi, jer bi svaki programer mogao interpretirati poruke na svoj način.

Statusni kodovi su standardizirani i svaki statusni kod ima svoje značenje. Na primjer, statusni kod `404` označava da resurs nije pronađen (prvi slučaj), dok statusni kod `500` označava općenitu grešku na poslužitelju (drugi slučaj).

## 4.1 Kako koristiti statusne kodove u Expressu?

U Expressu možemo slati statusne kodove u odgovorima koristeći `res.status()` metodu. Ova metoda postavlja statusni kod odgovora na poslužitelju.

Primjer postavljanja statusnog koda `200` (*OK*) u odgovoru:

```
app.get('/pizze', (req, res) => {
  res.status(200); // postavljanje statusnog koda 200 koji označava uspješan odgovor (OK)
});
```

Na metodu `res.status()` možemo nadovezati metodu `res.send()` ili `res.json()` kako bismo poslali podatkovni odgovor klijentu:

```
app.get('/pizze', (req, res) => {
  res.status(200).json(pizze); // poslati sve pizze kao JSON odgovor s statusnim kodom 200
});
```

Što ako **poslužitelj** ne može pronaći resurs **koji je korisnik zatražio**? U tom slučaju, možemo poslati statusni kod `404` (*Not Found*):

```
app.get('/pizze/:id', (req, res) => {
  const id_pizza = req.params.id;
  const pizza = pizze.find(pizza => pizza.id == id_pizza);

  if (pizza) {
    res.status(200).json(pizza);
  } else {
    res.status(404).json({ message: 'Pizza nije pronađena.' });
  }
});
```

Koji ćemo statusni kod poslati klijentu ako korisnik pošalje zahtjev s neispravnim podacima? Na primjer, ako korisnik pošalje kao parametar `id` slovo umjesto broja? U tom slučaju, možemo poslati statusni kod `400` (*Bad Request*):

```
router.get('/:id', (req, res) => {
```

```

const id_pizza = req.params.id;

if (isNaN(id_pizza)) {
    return res.status(400).json({ message: 'ID pizze mora biti broj.' }); // poslati
statusni kod 400 ako ID pizze nije broj
}

const pizza = pizze.find(pizza => pizza.id == id_pizza);

if (pizza) {
    return res.status(200).json(pizza); // poslati statusni kod 200 ako je pizza pronađena
} else {
    return res.status(404).json({ message: 'Pizza nije pronađena.' }); // poslati statusni
kod 404 ako pizza nije pronađena
}
);

```

Statusnih kodova ima mnogo, a svaki od njih ima svoje značenje. Možete pronaći **popis i definicije svih statusnih kodova** na [ovoj poveznici](#).

Međutim, u praksi se ne najčešće ne koriste svi statusni kodovi, već nekolicina njih. Evo nekoliko najčešće korištenih statusnih kodova:

- 200 - OK: Zahtjev je uspješno primljen i obrađen (npr. GET zahtjev za dohvata svih pizza)
- 201 - Created: Resurs je uspješno stvoren (npr. nakon slanja POST zahtjeva)
- 400 - Bad Request: Zahtjev nije moguće obraditi zbog neispravnih podataka (npr. korisnik je poslao neispravan ID pizze prilikom narudžbe)
- 404 - Not Found: Resurs nije pronađen (npr. korisnik je poslao ID pizze koja ne postoji)
- 500 - Internal Server Error: Opća greška na poslužitelju (npr. greška prilikom obrade narudžbe, najvjerojatnije zbog greške u kodu na poslužitelju)

Postoji puno varijacija 4xx, 5xx i 2xx statusnih kodova, pa tako imamo:

- 401 - Unauthorized: Korisnik nije autoriziran za pristup resursu (npr. korisnik nema prava pristupa resursu jer nije prijavljen)
- 204 - No Content: Zahtjev je uspješno primljen i obrađen, ali nema sadržaja za prikazati (npr. nakon brisanja resursa)
- 403 - Forbidden: Korisnik nema prava pristupa resursu (npr. korisnik nema prava pristupa resursu jer nije administrator)
- 301 - Moved Permanently: Resurs je trajno premješten na novu lokaciju (npr. kada se mijenja URL resursa)
- 503 - Service Unavailable: Poslužitelj nije dostupan (npr. poslužitelj je preopterećen)
- 409 - Conflict: Zahtjev nije moguće obraditi zbog konflikta (npr. korisnik pokušava ažurirati resurs koji je već ažuriran, npr. kod PUT/PATCH zahtjeva)

## Vježba 4 - Korištenje statusnih kodova u pizzeriji 4 0 4

Dodajte statusne kodove u odgovore vašeg poslužitelja za sve rute vezane uz pizze i narudžbe.

Pokušajte koristiti što semantički ispravnije statusne kodove. Na primjer, ako korisnik pokuša dohvatiti pizzu koja ne postoji, pošaljite statusni kod 404 (Not Found), ali ako korisnik pošalje neispravan ID pizze, pošaljite statusni kod 400 (Bad Request).

Dodatno, dodajte 3 nove rute u vašu pizzeriju:

- dohvaćanje svih narudžbi
- dohvaćanje narudžbe po ID-u
- brisanje narudžbe po ID-u

Za testiranje koristite Postman. Testirajte statusne kodove u odgovorima koje ste dobili.

## Samostalni zadatak za Vježbu 2

---

Definirajte novi Express projekt u kojem ćete implementirati jednostavni poslužitelj za agenciju za nekretnine.

**Osmislite dizajn poslužitelja**, a podatke spremajte u polje objekata, odnosno *in-memory*. **Podaci o nekretninama** trebaju sadržavati sljedeće informacije:

- ID nekretnine
- Naziv nekretnine
- Opis nekretnine
- Cijena nekretnine
- Lokacija nekretnine
- Broj soba
- Površina nekretnine
- Cijena nekretnine

Implementirajte sljedeće rute:

- dohvati sve nekretnine
- dohvati nekretninu po ID-u
- dodaj novu nekretninu
- ažuriraj nekretninu potpuno
- ažuriraj nekretninu djelomično
- obriši nekretninu
- pošalji novu ponudu

Ponude spremajte na jednak način u polje objekata, a **svaka ponuda mora sadržavati**:

- ID ponude
- ID nekretnine
- Ime kupca
- Prezime kupca

- Ponuđena cijena
- Broj telefona kupca

Dodajte slične **provjere** kao u pizzeriji, primjerice:

- provjerite jesu li ID-evi brojevi, ako ne vratite odgovarajući statusni kod i poruku
- provjerite jesu li svi podaci poslani u tijelu zahtjeva, ako nisu vratite odgovarajući statusni kod i poruku
- provjerite jesu li svi podaci ispravni, npr. cijena nekretnine ne može biti negativna, broj soba ne može biti negativan, itd.
- prilikom izrade ponude, provjerite postoji li uopće nekretnina s navedenim ID-em

Rute za **nekretnine i ponude grupirajte u zasebne Router objekte** i organizirajte ih u zasebnim datotekama unutar `routes` direktorija. Koristite statusne kodove u odgovorima.

Za testiranje koristite Postman.