JUNIORDev DIGITALNA DALMACIJA

Komunikacija > Slanje zahtjeva

# Slanje zahtjeva

Slanje zahtjeva sa klijenta na poslužitelja možemo implementirati na tri načina:

- korištenjem ***XMLHttpRequest*** (XHR) objekta
- korištenjem ***Fetch API*** sučelja
- korištenjem neke od vanjskih biblioteka za klijent-poslužitelj komunikaciju (***Axios***)

Slanje zahjteva putem XHR-a je pomalo zastarjelo pa ćemo se osvrnuti na preostala dva načina komunikacije, primarno na *fetch API*.

## API

Za početak ćemo krenuti samo sa dohvatom podataka sa udaljenog poslužitelja. Kako ne bi gubili vrijeme na izradu vlastitog poslužitelja, iskoristiti ćemo neki od postojećih API-ja. API je kratica od *Application Programming Interface* i predstavlja skup definicija i protokola koji omogućavaju komunikaciju između dvije aplikacije. U ovom slučaju želimo uspostaviti komunikaciju sa nekim već postojećim poslužiteljem i poslati mu zahtjev za podacima.

Iako postoje različite vrste API-ja, te oni mogu slati različite tipove podataka, za sada nećemo ulaziti detaljnije u to već ćemo koristiti vrstu API.ja koja se naziva "REST API" i podaci koje će nam slati će biti u JSON formatu. JSON je kratica od "*JavaScript Object Notation*" i kao što mu ime govori radi se o zapisu koji izgleda kao zapis JS objekta (uz poneke razlike koje nam trenutno nisu važne.)

## Dohvat podataka

Prvi korak pri uspostavi komunikacije je saznati adresu (URL) na koju ćemo slati zahtjev. U početnom primjeru ćemo iskoristiti besplatni REST API pod nazivom JSON Placeholder koji pruža nekoliko kolekcija (lažnih) podataka.

Započnimo sa početnom React aplikacijom kojoj ćemo dodati jedan `<button>` element za početak slanja zahtjeva.

App.jsx

```jsx
function App() {
  function dohvatiPodatke() {
    // Slanje zahtjeva
  }

  return (
    <div className='App'>
      <h1>Dohvat podataka</h1>
      <button onClick={dohvatiPodatke}>Dohvati podatke</button>
    </div>
  );
}
export default App;
```

Sada se možemo fokusirati na funkciju za dohvat podataka u kojoj ćemo iskoristi znanje asinkronog izvršavanja:

```jsx
function dohvatiPodatke() {
  const url = "https://jsonplaceholder.typicode.com/posts/1";

  fetch(url)
    .then(res => res.json())
    .then(data => console.log(data));
}
```

Iako primjer izgleda jednostavno, ovdje imamo nekoliko detalja na koje moramo obratiti pozornost. Za početak smo (zbog preglednosti) u konstantu spremili URL adresu na koju šaljemo zahtjev. Nakon toga smo pozivom ugrađene `fetch()` funkcije poslali zahtjev na tu adresu. Ukoliko nije drukčije navedeno, *fetch* šalje zahtjev tipa ***GET*** tj. zahtjev za dohvat

podataka, kao što je slučaj u ovom primjeru. Ostale tipove zahtjeva ćemo obraditi u idućem poglavlju.

Kako je već spomenuto, dohvat zadataka je asinkrona operacija pa koristimo `then()` metodu kako bi definirali *callback* funkciju koja će se izvršiti kada se askinrona operacija izvrši (tj. kada Promise bude u stanju *fulfilled*). *Callback* funkcija kao argument prima rezultat operacije - u ovom slučaju podataka sa poslužitelja. S obzirom da poslužitelj šalje odgovor u JSON formatu nad njim pozivamo metodu `.json()` kako bi taj odgovor parsirali i preveli u JS objekt.

Metoda `.json()` je također asinkrona operacija pa i na nju moramo vezati `then()` metodu. U drugoj *callback* funkciji napokon dobiveni rezultat ispisujemo na konzolu.

💡 Možete pokušati ispisati odgovor odmah u prvom koraku, bez parsiranja, pa ćete vidjeti kako izgleda potpuni odgovor od poslužitelja.

```
fetch(url).then(res => console.log(res))
```

# Prikaz podataka

Pogledamo li prethodni ispis u konzoli vidimo da podaci izgledaju ovako:

```
{
  body: "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit m
  id: 1,
  title: "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
  userId:1
}
```

Imamo JS objekt sa četiri svojstva i pripadajućim vrijednostima. Idući korak nam je pokušati prikazati te podatke u našoj komponenti. Iskoristiti ćemo već poznati *useState* kako bi spremili dobiveni objekt u stanje komponente. Vidjeli smo da u stanje možemo spremiti cijeli objekt pa nam je dovoljna samo jedna varijabla stanja.

App.jsx

```jsx
import { useState } from "react";

function App() {
  const [podatak, postaviPodatak] = useState({
    body: "",
    id: null,
    title: "",
    userId: null
  })

  function dohvatiPodatke(){
    fetch('https://jsonplaceholder.typicode.com/posts/1')
      .then(res => res.json())
      .then(data => postaviPodatak(data))
  }

  return (
    <div className='App'>
      <h1>Dohvat podataka</h1>
      <button onClick={dohvatiPodatke}>Dohvati podatke</button>
      <div>
        <h3>{podatak.title}</h3>
        <p>{podatak.body}</p>
      </div>
    </div>
  );
}
```

Kao početno stanje smo definirali objekt sa istim svojstvima kao i podatak koji dohvaćamo. Također smo umjesto ispisa u konzolu, dohvaćeni podatak spremili kao novu vrijednost stanja. Ostatak je određen po već poznatom scenariju - promjena stanja uzrokuje ponovno renderiranje komponente te se podaci ispisuju na aplikaciji.

# Automatsko slanje

Nakon što smo naučili osnovnu komunikaciju sa poslužiteljem, možemo se kratko vratiti na ***useEffect*** *hook* i iskoristiti ga kako bi pri renderiranju komponente automatski poslali zahtjev

za dohvat podataka. S obzirom da već imamo napisanu funkciju za dohvat podataka, možemo je jednostavno pozvati unutar *useEffect callback* funkcije (mogli smo samo poslati referencu na funkciju *dohvatiPodatke* ali ovako ostavljamo mogućnost dodatnih naredbi u efektu):

```
useEffect(() => {
  dohvatiPodatke();
}, []);
```

Napraviti ćemo još jednu nadogradnju aplikacije. Zadnji dio adrese ( `"/1"` ) se odnosi na *id* podatka kojeg dohvaćamo. Umjesto da uvijek dohvaćamo isti podatak, dodati ćemo mogućnost korisničkog unosa. Ovo nam je dobra vježba za ponoviti koncepte kao šta su upravljane komponente i *dependency* vrijednosti *useEffect hook*-a. Dokumentacija API-ja navodi da postoje podaci sa *id* vrijednostima od 1-100 pa dodajte i mogućnost validacije unosa.

Jedno od mogućih rješenja izgleda kao u primjeru ispod.

App.jsx

```
import { useState, useEffect } from "react";

function App() {
  const [id, postaviId] = useState(1);
  const [podatak, postaviPodatak] = useState({
    body: "",
    id: null,
    title: "",
    userId: null,
  });

  useEffect(() => {
    fetch(`https://jsonplaceholder.typicode.com/posts/${id}`)
      .then(res => res.json())
      .then(data => postaviPodatak(data))
  }, [id]);

  function idPromjena(e){
    if (0 < e.target.value && e.target.value < 101 )
      postaviId(e.target.value)
  }

  return (
```

```
      <div className='App'>
        <h1>Dohvat podataka</h1>
        <label htmlFor="broj">Unesi ID poruke: </label>
        <input
          onChange={idPromjena}
          type='number'
          min={1}
          max={100}
          value={id}
          id="broj"
        />
        <div>
          <h3>{podatak.title}</h3>
          <p>{podatak.body}</p>
        </div>
      </div>
    );
  }
  export default App;
```

Last updated on March 28, 2023

‹ Klijent-poslužitelj Axios ›