JUNIORDev DIGITALNA DALMACIJA

Komunikacija > Niz podataka

Niz podataka

Često ćemo kao odgovor od poslužitelja, umjesto samo jednog podataka, dobiti više podataka odjednom. Ti podaci će gotovo uvijek biti poslani u obliku niza (*array*) pa je ovo pravi trenutak da se osvrnemo i na to kako React upravlja sa nizovima podataka.

map metoda

Jedan od ključnih preduvjeta za rad sa nizovima u Reactu je poznavanje ***map*** metode u JavaScriptu koja je ugrađena u *Array* objekt. To znači da svaki podatak koji je tipa *Array* (tj. niz) sadrži u sebi ovu metodu i može je pozvati.

Rezultat `map()` metode je **novi niz** koji sadrži rezultate poziva zadane funkcije nad svakim elementom niza nad kojim je metoda pozvana. Jednostavnije rečeno, kada `map()` metodu pozovemo nad nekim nizom, kao ulazni argument joj moramo poslati neku funkciju. Ta funkcija će se zatim pozvati nad svakim elementom niza (trenutni element će biti ulazni parametar) i rezultati svih tih poziva će biti spremljeni u novi niz i poslani kao rezultat `map()` metode.

Pogledajmo jednostavan primjer - imamo niz brojeva i želimo sve elemente uvećati za 3.

```
const niz = [3, 7, 22, 12];

const novi = niz.map(broj => broj + 3)

console.log(novi) /// Array [6, 10, 25, 15]
```

Gornji primjer je naravno napisan u skraćenom obliku, isti rezultat bi bio i kada bi funkciju za uvećanje imali napisano odvojeno:

```
const niz = [3, 7, 22, 12];

function uvecajZaTri(broj){
  return broj + 3
}

const novi = niz.map(uvecajZaTri)

console.log(novi) /// Array [6, 10, 25, 15]
```

Ovisno o složenosti funkcije koju želimo izvršiti nad svakim elementom niza i preglednosti kôda nekada ćemo koristiti kraći, a nekada duži oblik.

Dohvat podataka

Vratimo se sada Reactu. Ovaj put ćemo za primjer uzeti stvarne podatke tj. iskoristit ćemo još jedan besplatni API ali ovaj put sa stvarnim podacima. Radi se o API-ju pomoću kojeg možemo saznati koliko se trenutno ljudi nalazi u svemiru. S obzirom da već znamo slati zahtjev, poslati ćemo ga koristeći *axios* i pogledati rezultat na konzoli (možete modificirati prethodni primjer ili napraviti novi projekt):

App.jsx

```
import axios from "axios";

function App() {

  function dohvatiPodatke() {
    axios
      .get("http://api.open-notify.org/astros.json")
      .then(res => console.log(res.data))
      .catch(err => alert(err));
  }

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

```
      <div>

      </div>
    </div>
  );
}

export default App;
```

Odgovor od poslužitelja (pod pretpostavkom da nije došlo do pogreške) se sastoji od tri podatka tj. objekt odgovora ima tri svojstva:

- "***message***" koje ima vrijednost "success"
- "***number***" koje sadrži broj ljudi u svemiru (14 u trenutku pisanja)
- "***people***" - niz sa imenima ljudi i letjelicama/postajama na kojima se nalaze Nas naravno zanima ovaj zadnji podatak pa ćemo malo modificirati gornji primjer i spremiti popis ljudi u stanje komponente.

App.jsx

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

function App() {
  const [ljudi, postaviLjude] = useState([]);

  function dohvatiPodatke() {
    axios
      .get("http://api.open-notify.org/astros.json")
      .then(res => postaviLjude(res.data.people))
      .catch(err => alert(err));
  }

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

      </div>
    </div>
  );
}
```

```
export default App;
```

Prikaz nizova

Podaci su sada spremljeni ali nam preostaje problem kako ih prikazati unutar komponente. I strukturi komponente smo ostavili prazni *div* ali ne možemo unutar njega samo napisati `{ljudi}` jer imamo cijeli niz podataka i React ne zna kako ih točno prikazati, pogotovo jer se radi o nizu objekata (čak i da nije niz objekata, gotovo nikada vam neće odgovarati zadani način prikaza osnovnih tipova podataka).

Teoretski bi mogli "izvlačiti" podatke za jednu po jednu osobu i prikazivati te podatke na komponenti (npr. `{ljudi[2].name}`) ali to bi bilo izrazito nepraktično iz više razloga i teško da se može smatrati rješenjem.

Kao očito rješenje se nameće `map()` metoda pomoću koje možemo proći kroz cijeli niz i kojoj ćemo poslati funkciju koja određuje na koji način želimo elemente prikazati unutar komponente. Jednostavni primjer oblikovanja bi izgledao ovako (pišemo samo *return* izraz komponente, ostatak je ostao isti):

App.jsx

```
1    return (
2      <div className='App'>
3        <h1>Dohvat podataka</h1>
4        <button onClick={dohvatiPodatke}>Dohvati podatke</button>
5        <div>
6          {nizB}
7        </div>
8        <div>
9          {ljudi.map(objekt => {
10           return (
11             <div>
12               <p>Osoba: {objekt.name}</p>
13               <p>Lokacija: {objekt.craft}</p>
14             </div>
15           );
16         })}
```

```
17        </div>
18      </div>
19    );
```

Logika je zapravo vrlo slična kao da imamo u stanju jedan objekt pa definiramo kako želimo prikazati podatke iz tog objekta (linije 11-14). U ovom slučaju imamo niz takvih objekata pa je zato ta logika prikaza ubačena u funkciju koju šaljemo `map()` metodi kako bi se to primijenilo na svim elementima niza.

Pomoćne komponente

Na prvi pogled gornji primjer izgleda pomalo složeno. Čak i ukoliko u potpunosti razumijemo način na koji radi, očito je da bi kod složenijih podataka ovo postalo nepregledno. Zbog toga ćemo pogledati kako malo pojednostavniti ovaj primjer.

Rješenje je naravno - u komponentama. Za početak ćemo napraviti novu komponentu ("Astronaut") unutar koje ćemo definirati željeni izgled za **jedan** podatak iz niza.

Astronaut.jsx

```jsx
function Astronaut(props) {
  return (
    <div>
      <p>Osoba: {props.name}</p>
      <p>Lokacija: {props.craft}</p>
    </div>
  )
}
export default Astronaut
```

Nakon toga možemo jednostavno tu komponentu uključiti u glavni dio aplikacije te je iskoristiti u `map()` metodi kako bi zadali prikaz svim elementima niza.

App.jsx copy

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

```
import Astronaut from "./Astronaut";

function App() {
  const [ljudi, postaviLjude] = useState([]);

  function dohvatiPodatke() {
    axios
      .get("http://api.open-notify.org/astros.json")
      .then(res => postaviLjude(res.data.people))
      .catch(err => alert(err));
  }

  return (
    <div className='App'>
      <h1>Dohvat podataka</h1>
      <button onClick={dohvatiPodatke}>Dohvati podatke</button>
      <div>
        {ljudi.map(el => (
          <Astronaut name={el.name} craft={el.craft} />
        ))}
      </div>
    </div>
  );
}

export default App;
```

key svojstvo

Na kraju nam ostaje još jedan važan detalj pri radu sa nizovima i njihovim prikazom, a to je tzv. ***key prop*** ili *key* svojstvo. Svrha *key* svojstva u Reactu je identificirati pojedinačne elemente u situacijama kada imamo puno elemenata istog tipa (kao u slučaju niza podataka). Ovo je ponovno povezano za načinom na koji React renderira i osvježava komponente. Ispravno *key* svojstvo pomaže Reactu prepoznati koji dio aplikacije je promijenio, a koji nije.

Pogledajmo gornji primjer sa astronautima (malo ćemo ga pojednostaviti). Zamislite da imamo 3 astronauta i ispišemo njihova imena u listu:

```
<li>Deng</li>
<li>Frank</li>
```

```
<li>Zhang</li>
```

Zatim u jednom trenutku na listu želimo dodati još jedan podatak - `<li>Andrey</li>`. React tada uspoređuje staro i novo stanje DOM-a i računa *diff*. To izgleda otprilike ovako:

```
<!-- Novo stanje !-->        <!-- staro stanje !-->
<li>Deng</li>   <!-- jednako !-->  <li>Deng</li>
<li>Frank</li>  <!-- jednako !-->  <li>Frank</li>
<li>Zhang</li>  <!-- jednako !-->  <li>Zhang</li>
<li>Andrey</li> <!-- PROMJENA !-->
```

Prema izračunatoj razlici React zna da mora samo dodati zadnji element na listi. Ostali elementi su ostali nepromijenjeni te njih nema potrebe ponovno renderirati. U ovom slučaju nema nikakvih problema, ali pogledajmo što bi bilo kada bi htjeli dodati taj isti podatak ali želimo da imena budu abecednim redom pa novu osobu dodajemo na početak liste:

```
<!-- Novo stanje !-->        <!-- staro stanje !-->
<li>Andrey</li> <!-- PROMJENA !-->  <li>Deng</li>
<li>Deng</li>   <!-- PROMJENA !-->  <li>Frank</li>
<li>Frank</li>  <!-- PROMJENA !-->  <li>Zhang</li>
<li>Zhang</li>  <!-- PROMJENA !-->
```

Sada React vidi čak 4 promjene i renderirati će cijelu listu iz početka, što naravno može utjecati na performanse aplikacije a u najgorem slučaju čak i uzrokovati pogrešku u vašoj aplikaciji. Upravo zbog toga React koristi *key* svojstvo kako bi mogao identificirati pojedinačne elemente. Ovaj prethodni primjer smo mogli vrlo jednostavno riješiti uvođenjem *key* svojstva. Kada elementi imaju *key* svojstvo, React će njega koristiti kao referencu za usporedbu dva elementa.

```
<li key={3}>Andrey</li>  // <-- NOVI ELEMENT
<li key={0}>Deng</li>
<li key={1}>Frank</li>
<li key={2}>Zhang</li>
```

Vrijednost key svojstva

Pripazite na koji način dodjeljujete vrijednosti *key* svojstvu. U gornjem primjeru su to bili samo redni brojevi, ali pripazite kako ne bi elementima niza kao *key* vrijednost dodijelili njihov indeks niza jer ćete ponovno izgubiti mogućnost jedinstvenog identificiranja - elemente niza možemo dodavati ili brisati te tako promijeniti indekse pojedinim elementima. Vrijednost *key* svojstva bi trebala biti **jedinstvena** za svaki element kojeg želimo prikazati. Naravno postoje situacije kada to neće uvijek biti moguće, ali to je "problem" podataka. Većina podataka sa kojima ćete susretati u praksi posjeduje neki jedinstveni identifikator kojeg možete koristiti kao vrijednost *key* svojstva. U slučaju našeg primjera sa astronautima takvog podatka baš i nema, ali zato možemo iskoristiti ime kao *key* svojstvo jer je vjerojatnost dva astronauta sa istim imenom zaista minimalna pa možemo to smatrati jedinstvenim podatkom.

App.jsx copy

```jsx
import { useState } from "react";
import axios from "axios";
import Astronaut from "./Astronaut";

function App() {
  const [ljudi, postaviLjude] = useState([]);

  function dohvatiPodatke() {
    axios
      .get("http://api.open-notify.org/astros.json")
      .then(res => postaviLjude(res.data.people))
      .catch(err => alert(err));
  }

  return (
    <div className='App'>
      <h1>Dohvat podataka</h1>
      <button onClick={dohvatiPodatke}>Dohvati podatke</button>
      <div>
        {ljudi.map(el => (
          <Astronaut key={el.name} name={el.name} craft={el.craft} />
        ))}
      </div>
    </div>
  );
}

export default App;
```

Sada nam više React ne izbacuje upozorenje u konzoli jer svaki element ima svoje *key* svojstvo. Unatoč tome, u ovom slučaju nismo potpuno riješili problem renderiranja. Potreban nam je još jedan "trik", a to je napraviti "***memoiziranu*** verziju komponente "Astronaut". Ovdje ponovno dolazimo do malo naprednijih koncepata, ali trenutno nas samo zanima krajnji rezultat - "memo" verzija komponente dozvoljava Reactu da tu komponentu ne renderira ponovno ukoliko joj se svojstva nisu promijenila - dakle dok nam komponenta ima isti ključ i podatke, neće se ponovno renderirati, čak ni ako se roditeljska komponenta ponovno renderira.

Implementacija je prilično jednostavna, dovoljno nam je napraviti sitnu izmjenu u samoj datoteci komponente:

Astronaut.jsx

```
import {memo} from 'react'

const AstronautMemo = memo(Astronaut)

function Astronaut(props) {
  console.log("Render")
  return (
    <div>
      <p>Osoba: {props.name}</p>
      <p>Lokacija: {props.craft}</p>
    </div>
  )
}

export default AstronautMemo
```

Sve što je bilo potrebno napraviti je uključiti metodu `memo()` iz React biblioteke te pomoću nje napraviti *memoiziranu* verziju naše komponente. Naravno ne smijemo zaboraviti ni zadnji korak, a to je da umjesto originalne verzije izvozimo *memo* verziju komponente. U ovom primjeru je dodana i `console.log()` metoda kako bi mogli testirati rad aplikacije. Ako u glavnom dijelu aplikacije pritisnemo tipku "Dohvati podatke", novi podaci će se dohvatiti sa poslužitelja ali neće se dogoditi novo renderiranje.

Last updated on March 28, 2023

Axios

Praktični dio