JUNIORDev DIGITALNA DALMACIJA

Koponente sa stanjem > Komponente sa stanjem

Komponente sa stanjem

Vratimo se na naš početni primjer u kojem smo pokušali mijenjati vrijednost broja kojeg ispisujemo unutar komponenti. Za početak ćemo vratiti "*main.jsx*" u početno stanje:

main.jsx

```
const root = ReactDOM.createRoot(document.getElementById("root"));

root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

Sada se možemo fokusirati samo na komponentu "*App.jsx*". Umjesto kroz *props* objekt definirati ćemo brojač unutar nje:

App.jsx

```
import "./App.css";
function App() {
  let broj = 5;
  return (
    <div>
      <p>Vrijednost broja: {broj}</p>
    </div>
  );
}
export default App;
```

Ovaj primjer je dobar sve dok ne želimo promijeniti vrijednost broja - tada nailazimo na već spomenute probleme. U ovom slučaju je *broj* varijabla koja je vezana uz samu komponentu i

njena vrijednost je važna za ispravan prikaz same komponente - želimo da je na ekranu uvijek prikazana ažurirana vrijednost varijable.

Postoje dva problema sa ovim pristupom:

1. promjena vrijednosti varijable *broj* neće uzrokovati novo renderiranje komponente
2. sjetite se da su React komponente *immutable* tj. nepromjenjive. Ponovno renderiranje komponente znači da će se kôd koji pripada njenom funkcijskom obliku ponovno izvršiti, što bi u ovom slučaju značilo da će se varijabla *broj* ponovno inicijalizirati na početnu vrijednost (5).

Ono što mi želimo postići je upravo suprotno - želimo komponenti omogućiti da pamti vrijednost koja će ostati sačuvana između uzastopnih renderiranja te čija će promjena automatski pokrenuti proces renderiranja kako bi se mogla prikazati nova vrijednost. To u Reactu možemo postići korištenjem metode *useEffect* koja nam pruža dva elementa:

- varijablu stanja (*state variable*) unutar koje možemo spremiti vrijednost koju želimo očuvati
- funkciju za postavljanje stanja (*state setter*) pomoću koje možemo mijenjati vrijednost varijable stanja, ali na kontrolirani način tako da React može zakazati promjene na DOM-u

useState

Modificirati ćemo prethodni primjer tako da našoj komponenti dodamo jednu varijablu stanja (broj):

App.jsx

```jsx
import { useState } from "react";
import "./App.css";

function App() {
  const [broj, postaviBroj] = useState(5);

  return (
    <div>
      <p>Vrijednost broja: {broj}</p>
    </div>
```

```
  );
}
export default App;
```

Pogledajmo detaljnije što smo napravili:

1. Prvi korak je iz React biblioteke uključiti *hook* funkciju koju želimo koristiti. To smo ovdje napravili odmah na početku i uključili *useState*
2. Zatim smo unutar tijela komponente pozvali *useState* i kao argument joj poslali inicijalnu vrijednost varijable stanja (u ovom slučaju broj 5)
3. Kao što smo već spomenuli, *useState* funkcija kao rezultat vraća niz sa dva elementa - prvi element je sama varijabla stanja koja sadrži inicijalnu vrijednost, a drugi element je referenca na funkciju pomoću koje možemo mijenjati tu inicijalnu vrijednost tj. našu varijablu stanja. Naravno da su nam potrebni oboje, pa spremamo i varijablu i funkciju u lokalne konstante unutar komponente koje imenujemo po želji (u ovom slučaju nam se varijabla stanja zove *broj* a funkcija za promjenu *postaviBroj*)

U našem primjeru već koristimo varijablu stanja u prikazu komponente. Nedostaje nam još logika promjene te vrijednosti. Dodati ćemo unutar komponente novi "*button*" element i napisati praznu pomoćnu funkciju koju ćemo vezati uz ***onClick*** atribut (sjetite se pravila za pisanje atributa u JSX-u):

App.jsx

```
function App() {
  const [broj, postaviBroj] = useState(5);
  function uvecajBrojac() {
    // Uvecaj brojac
  }
  return (
    <div>
      <p>Vrijednost broja: {broj}</p>
      <button onClick={uvecajBrojac}>Uvećaj</button>
    </div>
  );
}
```

Promjena stanja

Naravno, nedostaje nam još samo logika uvećavanja brojača - za početak ćemo krenuti sa najjednostavnijom verzijom - dovoljno je pozvati funkciju za promjenu stanja varijable i kao argument joj poslati novu vrijednost koju želimo spremiti u varijablu stanja:

App.jsx

```
function uvecajBrojac() {
  postaviBroj(6);
}
```

Očito je da na ovaj način broj možemo uvećati samo jednom, bolja bi opcija bila da brojač možemo uvećavati više puta. Funkciju smo mogli napisati i ovako:

App.jsx

```
function uvecajBrojac() {
  postaviBroj(broj + 1);
}
```

Vodite računa da smo kao argument poslali izračun u kojem zbrajamo trenutnu vrijednost varijable stanja i konstantu `1`. Pri tome nismo **direktno** mijenjali samu vrijednost varijable stanja - nismo pisali `"broj += 1"` ili `"broj++"`.

Ovaj način slanja novog stanja u obliku konstante ili izračuna nije uvijek najbolja opcija ali na to ćemo se osvrnuti kasnije.

U nijednom slučaju ne bi trebali direktno mijenjati vrijednost varijable stanja, već uvijek za to koristiti pripadajuću funkciju. Zbog toga je dobra praksa varijablu stanja spremati u *const* deklaraciju kao u ovom primjeru jer ćete tada dobiti upozorenje da pokušavate mijenjati konstantu, što nije moguće.

Osvježavanje prikaza

Kao što možete primijetiti na aplikaciji, svaki put kada pozovemo funkciju za promjenu stanja, aplikacija se automatski osvježi i prikaže novu vrijednost varijable stanja na zaslonu. Već smo prethodno spomenuli da React to samostalno odrađuje u pozadini, naš zadatak kao programera je samo postaviti novo stanje korištenjem pripadajuće funkcije.

Osvrnimo se na još jedan zanimljiv primjer. Unutar komponente ćemo iskoristiti JS funkciju ***setTimeout*** kako bi zakazali promjenu stanja nakon određenog broja sekundi. Sada naša "*App*" komponenta izgleda ovako:

App.jsx

```
function App() {
  const [broj, postaviBroj] = useState(5);

  function uvecajBrojac() {
    postaviBroj(broj + 1);
  }
  setTimeout(uvecajBrojac, 1500);

  return (
    <div>
      <p>Vrijednost broja: {broj}</p>
      <button onClick={uvecajBrojac}>Uvećaj</button>
    </div>
  );
}
```

Možete li zaključiti zašto je rezultat brojač koji se stalno uvećava, iako smo koristili `setTimeout`, a ne `setInterval` od koje bi se očekivao takav efekt.

Odgovor leži u načinu na koji React osvježava komponente. Krenimo od početka u koracima:

- na početku imamo prazni prozor preglednika. React zatim izvršava prvo iscrtavanje (renderiranje) naše *App* komponente. Prilikom tog prvog renderiranja izvršava se sav kôd unutar komponente, uključujući i poziv `setTimeout` funkcije koja zakazuje pokretanje

funkcije za uvećanje brojača nakon 1500 ms (1.5 sekundi).

- nakon što istekne *setTimeout* vrijeme, poziva se pomoćna funkcija `uvecajBrojac` koja mijenja vrijednost varijable stanja
- React detektira promjenu unutarnjeg stanja komponente i zaključuje da je istu potrebno osvježiti tj. ponovno je renderirati kako bi se na sučelju prikazalo aktualno stanje komponente - pokreće se proces ponovnog renderiranja
- ponovno renderiranje uzrokuje

Uvjetno renderiranje

Nakon što smo se upoznali sa konceptom stanja možemo spomenuti i jedan "trik" koji će često koristi pri izradi React aplikacija. Radi se o pojmu **uvjetnog renderiranja** koji podrazumijeva da ovisno o nekoj vrijednosti (stanju) mijenjamo izgled komponente ili čak prikazujemo različite komponente.

Krenimo sa jednostavnim primjerom i već poznatim brojačem. Želimo korisnicima ispisati upute da pritiskom na tipku mogu uvećati vrijednost brojača. Jednom kada su korisnici uvećali brojač, više nema potrebe da se poruka ispisuje. Jedan od načina kako postići tu funkcionalnost je ovakav:

App.jsx

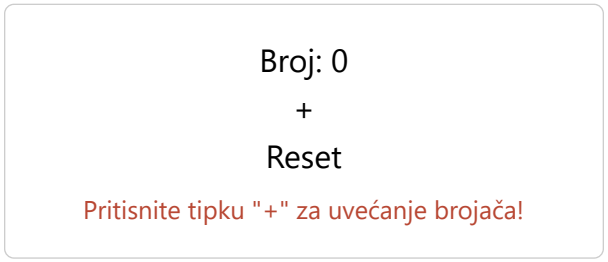
```jsx
import { useState } from "react";
import "./App.css";

function App() {
  const [broj, postaviBroj] = useState(0);

  return (
    <div className='main'>
      <p>Broj: {broj}</p>
      <button onClick={() => postaviBroj(broj + 1)}>+</button>
      <button onClick={() => postaviBroj(0)}>Reset</button>
      {broj == 0 && <small>Pritisnite tipku "+" za uvećanje brojača!</small>}
    </div>
  );
}
```

```
export default App;
```

U ovom slučaju koristimo logički operator `&&` kako bi postigli učinak uvjetnog renderiranja. Unutar `{}` zagrada pišemo logički izraz u kojem se sa lijeve strane nalazi uvjet kojeg provjeravamo, a sa desne JSX kôd kojeg želimo uvjetno renderirati. Ako je lijeva strana izraza istinita (***true***) onda će se desni dio prikazati u sklopu komponente (on je uvijek *true*). Ako je lijevi dio izraza ***false*** onda se cijeli izraz evaluira u *false* (jer koristimo operator AND) i React će to zanemariti prilikom renderiranja.



Za drugi primjer ćemo napraviti dodatnu komponentu **"Zadatak** i ovisno o *props* vrijednosti vraćati različiti izgled komponente:

Zadatak.jsx

```
function Zadatak(props) {
  if (props.gotov) {
    return <p>{props.natpis} ✅</p>;
  }
  return <p>{props.natpis} 🕓🔄</p>;
}
export default Zadatak;
```

App.jsx

```
import Zadatak from "./Zadatak";

function App() {
  return (
    <div>
      <h2>Popis zadataka:</h2>
      <Zadatak natpis="Zaliti cvijeće" gotov={true} />
      <Zadatak natpis="Kupiti kruh" gotov={false} />
      <Zadatak natpis="Upisati React tečaj" gotov={true} />
    </div>
```

```
    );
}
export default App;
```

Popis zadataka:

Zaliti cvijeće ✅
Kupiti kruh 🕒🔄
Upisati React tečaj ✅

U ovom konkretnom primjeru nismo koristili varijablu stanja nego *props* vrijednost za uvjetno renderiranje ali na isti način je uvjet mogao biti vezan uz neko stanje komponente.

Svi prethodni primjeri su sadržavali komponente sa samo jednom varijablom stanja. U praksi ćemo rijetko kada imati tako "čistu" situaciju, uobičajeno je da komponente imaju više *state* varijabli pa ćemo se detaljnije osvrnuti i na takve primjere.

Last updated on March 7, 2023

Složena stanja >