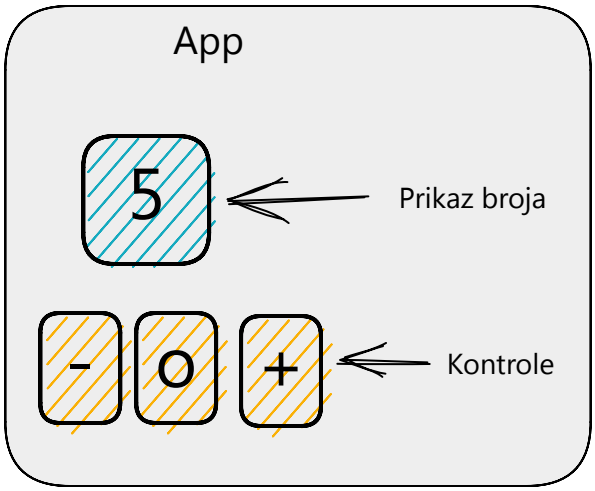
JUNIORDev DIGITALNA DALMACIJA

Prosljeđivanje stanja

React komponente se često nalaze ugniježđene jedne unutar drugih tj. imamo komponente u *parent-child* relaciji. To nije ništa neuobičajeno jer je i cijela HTML struktura zamišljena kao stablo sa roditeljskim i elementima djecom.

Uvođenjem koncepta unutarnjeg stanja komponente dolazimo do pitanja što se događa ako komponenta to stanje mora podijeliti sa nekom drugom komponentom - bilo roditeljem ili djecom. Kako bi demonstrirali ovaj scenarij, napraviti ćemo dvije nove komponente ćemo zatim iskoristiti unutar glavne "*App.jsx*" komponente. Idejno bi aplikacija trebala izgledati ovako:



Aplikaciju logički možemo podijeliti na dva dijela - dio za prikaz broja i dio za promjenu vrijednosti (umanji, povećaj, reset). Naravno da bi ovo sve mogli napraviti unutar jedne komponente ali zbog vježbe ćemo je razdvojiti u manje cjeline. Dio za prikaz broja će biti jedna komponenta (nazvati ćemo je "*Prikaz*"), dok ćemo za kontrole također napraviti jednu komponentu (pod nazivom "*Tipka*") koju možemo iskoristiti više puta.

Podizanje stanja

Prva odluka koju moramo napraviti je odlučiti gdje ćemo definirati varijablu stanja u koju će biti spremljena vrijednost brojača. Na prvi pogled se čini da bi to trebalo biti u komponenti *Prikaz* jer ona unutar sebe koristi tu vrijednost. Problem sa tim pristupom je što su nam za promjenu vrijednosti zadužene druge komponente - i to komponente koje su u hijerarhiji na istoj razini kao *Prikaz* komponenta (tzv. *sibling* relacija).

Kod određivanja stanja u Reactu dobra praksa je koristiti tzv. *lifting the state* princip. Ideja je jednostavna - u slučajevima kada više komponenti mora imati pristup istoj varijabli stanja, to stanje se podiže na najbliži roditeljski element od svih tih komponenti.

Konkretno, to u našem slučaju znači da ćemo varijablu stanja definirati na razini glavne *App* komponente, a zatim ćemo *child* komponentama slati vrijednost varijable stanja ili funkcije za promjenu stanja.

Izrada komponenti

Za početak ćemo samo napraviti prazne komponente (CSS pravila možemo pisati u "*App.css*" zbog jednostavnosti, samo pazite na jedinstvenost klasa i pravila).

Za komponentu "Prikaz" očekujemo da će primiti vrijednost broja kao *props* i tu primljenu vrijednost prikazati na zaslonu. Početni izgled komponente bi bio otprilike ovakav:

Prikaz CSS

/components/Prikaz.jsx

```
function Prikaz(props) {
  return (
    <div className="prikazBroja">
      <p>{props.broj}</p>
    </div>
  );
}

export default Prikaz;
```

Na sličan način radimo i komponentu "*Tipka*". Napraviti ćemo je na način da se može prilagoditi za različite svrhe pa ćemo natpis na *button* elementu primati kroz *props* objekt. Glavni zadatak koji ova komponenta mora raditi je omogućiti da se pritiskom na tipku promijeni stanje brojača. Kao što smo već naveli, sama varijabla stanja se nalazi u roditeljskoj komponenti, i jedini način na koji možemo mijenjati varijablu stanja roditelja je ako nam to roditeljska komponenta "omogući" slanjem odgovarajuće funkcije. Zbog toga uz *onClick* atribut povezujemo funkciju koju ćemo također dobiti kroz *props* objekt. Za početak ćemo radi lakšeg praćenja iskoristiti pomoćnu funkciju, a kasnije ćemo vidjeti kako to i kraće zapisati.

Tipka CSS

/components/Tipka.jsx

```
function Tipka(props){

  function handleClick(){
    // Pozivamo iz props-a
    props.akcija()
  }

  return(
    <div className="tipkaOkvir">
      <button onClick={handleClick}>{props.natpis}</button>
    </div>
  )
}
export default Tipka
```

Sada možemo iskoristiti naše komponente u glavnoj *App* komponenti. U skladu sa prethodnim planom, ovdje ćemo definirati varijablu stanja i postaviti je na početnu vrijednost. Uključiti ćemo dvije nove komponente te ih iskoristiti kao *child* elemente glavne komponente. Također moramo napisati pomoćnu funkciju za promjenu stanja. Na kraju *child* komponentama moramo poslati odgovarajuće dijelova stanja (vrijednost ili funkciju za promjenu).

Započnimo samo sa komponentom za prikaz i jednom kontrolom (za umanjivanje):

App.jsx

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

import Prikaz from "./components/Prikaz";
import Tipka from "./components/Tipka";

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

  const umanjiBroj = () => {
    postaviBroj(broj - 1);
  }
  return (
    <div>
      <Prikaz broj={broj} />
      <div>
        <Tipka natpis='-' akcija={umanjiBroj} />
      </div>
    </div>
  );
}

export default App;
```

Ovdje je najvažnije naglasiti dio sa slanjem funkcije kao parametra. S obzirom da komponenta *Tipka* nema direktnu mogućnost mijenjati varijablu stanja svog roditelja, tu mogućnost moramo implementirati na ovaj način. Definiramo pomoćnu funkciju za promjenu varijable stanja (kao i do sada) ali umjesto da je pozivamo negdje unutar glavne komponente, jednostavno smo je proslijedili kao *props* nekoj od *child* komponenti koja će je pozvati kada to bude potrebno (ovisno o njenoj internoj logici).

Sada možemo dodati i drugu komponentu - tipku za vraćanje broja na 0, pri čemu ćemo koristiti kraći zapis:

```jsx
<Tipka natpis="0" akcija={() => postaviBroj(0)} />
```

Ne zaboravite da kao *props* moramo poslati **referencu** na funkciju, a ne poziv funkcije. Jedna

od čestih grešaka koja se može dogoditi početnicima je da pošalju funkciju u obliku:

```
akcija={postaviBroj(0)} ❌
```

Ovaj način je naravno pogrešan jer ne šaljemo funkciju kao parametar već odmah radimo poziv te funkcije - rezultat bi naravno bila promjena stanja koja bi uzrokovala ponovno renderiranje koje bi zatim ponovno pokrenulo poziv funkcije... i tako bi dobili beskonačnu petlju renderiranja i samim time neispravnu aplikaciju. Ukoliko niste potpuno ovladali JavaScript sintaksom, za početak je sigurnije pisati pomoćne funkcije kao u prvom primjeru.

Zadnji korak je još napraviti i treću instancu komponente *Tipka*, onu koja će biti zadužena za uvećanje brojača, a to već možete i samostalno odraditi. Prije prelaska na malo složeniji primjer za vježbu, možete također i doraditi izgled samih komponenti.

Last updated on March 13, 2023

‹ Čuvanje stanja

Praktični dio ›