JUNIORDev DIGITALNA DALMACIJA

Unos korisničkih podataka

Svi dosadašnji primjeri su koristili isključivo *button* element za interakciju korisnika sa aplikacijom. U praksi je često potrebno koristiti i druge kontrole za unos podataka kao što su polje za unos, *radio* ili *checkbox*. Zbog toga ćemo se kratko osvrnuti kako te standardne HTML elemente povezati sa React-om. Postoje dva načina za implementaciju elemenata za unos - možemo ih implementirati kao kontrolirane/upravljane (*controlled*) komponente ili kao samostalne/nekontrolirane (*uncontrolled*) komponente.

Kontrolirane komponente

Kontrolirane komponente u Reactu su one kod kojih se vrijednostima korisničkog unosa upravlja direktno iz same komponente - korištenjem stanja (*state*). Svaka promjena vrijednosti u polju za unos automatski osvježava stanje pripadajuće komponente. Pogledajmo primjer sa klasičnim *input* elementom. Za početak ćemo napraviti početni izgled sa jednim input elementom:

App.jsx

```
function App() {
  return (
    <div>
      <div>
        <label htmlFor="ime">Unesite ime:</label>
        <input type="text" id="ime" name="ime" />
      </div>
      <div>
        <p>Dobar dan ...</p>
      </div>
    </div>
  );
}
```

```
export default App;
```

Atribut *for* koji se koristi kod `<label>` elementa je još jedan primjer specifične JSX sintakse. Kao što možete uočiti, u JSX-u se piše u obliku **`htmlFor`** kako bi se razlikovao od ključne riječi za petlju.

Kod klasičnog JavaScripta bi vjerojatno dohvatili *input* element po njegovom *id* atributu te očitali *value* svojstvo elementa koje bi onda prikazali na sučelju (najčešće u `<span>` elementu). Kontrolirane React komponente imaju drugi pristup. Prvo ćemo u komponentu dodati varijablu stanja u kojoj želimo spremati vrijednost našeg *input* elementa:

App.jsx

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

function App() {
  const [ime, postaviIme] = useState("");
  return (
    <div>
      <div>
        <label htmlFor="ime">Unesite ime:</label>
        <input type="text" id="ime" />
      </div>
      <div>
        <p>Dobar dan {ime}</p>
      </div>
    </div>
  );
}

export default App;
```

Idući korak je povezati *input* element sa stanjem komponente. To radimo koristeći ***value*** atribut kojem ćemo kao vrijednost dodijeliti pripadajuću varijablu stanja:

App.jsx

```
function App() {
  const [ime, postaviIme] = useState("");
```

```
  return (
    <div>
      <div>
        <label htmlFor="ime">Unesite ime:</label>
        <input type="text" id="ime" value={ime} />
      </div>
      <div>
        <p>Dobar dan {ime}</p>
      </div>
    </div>
  );
}
```

Ukoliko u ovom trenutku pokušate nešto upisati u *input* element, uočiti ćete da on više ne reagira na unos. Razlog tome je što je React preuzeo kontrolu nad tim elementom i unutar njega prikazuje vrijednost varijable stanja (koja nam je postavljena na prazni string `""`). Možemo pokušati promijeniti početnu vrijednost varijable stanja kako bi lakše uočili ponašanje komponente. Očito je da nam nedostaje logika za promjenu vrijednosti stanja. Nju možemo implementirati koristeći **`onChange`** atribut *input* elementa koje ćemo naravno povezati sa pomoćnom funkcijom za promjenu stanja. *Callback* funkcija za *onChange* kao argument prima instancu ***Event*** objekta (kao i u standardnom JS-u).

App.jsx

```
function App() {
  const [ime, postaviIme] = useState("Ana");

  function noviInput(e) {
    postaviIme(e.target.value);
  }
  return (
    <div>
      <div>
        <label htmlFor="ime">Unesite ime:</label>
        <input type="text" id="ime" value={ime} onChange={noviInput} />
      </div>
      <div>
        <p>Dobar dan {ime}</p>
      </div>
    </div>
  );
}
```

Sada je *input* element u potpunosti upravljan od strane Reacta - prikaz i promjena vrijednosti su vezene uz pripadajuću React komponentu. Također, u ovom slučaju je logika za promjenu vrijednosti jednostavna pa je možemo pisati i bez pomoćne funkcije:

```
<div>
  <label htmlFor="ime">Unesite ime:</label>
  <input
    type="text"
    id="ime"
    value={ime}
    onChange={(e) => postaviIme(e.target.value)}
  />
</div>
```

Neupravljane komponente

Neupravljane (ili "nekontrolirane") komponente su one u kojima upravljanje ulaznim podacima obavlja DOM tj. nisu pod kontrolom Reacta. Princip je identičan kao i u slučaju kada ne bi koristili React. Vrijednosti ulaza tj. nekog *input* elementa se spremaju u samom HTML elementu tj. njegovoj instanci u DOM-u. Za dohvat vrijednosti nam je potrebna referenca na sami element i jednostavno pristupamo njegovom *value* svojstvu kako bi dobili željenu vrijednost. U prethodnim primjerima smo se upoznali sa *useRef hook*-om i njegovim korištenjem za referenciranje DOM elemenata pa ga možemo iskoristiti za tu svrhu, kao na primjeru ispod:

App.jsx

```
import { useRef } from "react";

function App() {
  const imeRef = useRef();

  function prikazi(){
    alert("Dobar dan " + imeRef.current.value)
  }

  return (
    <div>
```

```
        <div>
          <label htmlFor="ime">Unesite ime:</label>
          <input type="text" id="ime" ref={imeRef} />
        </div>
        <div>
          <button onClick={prikazi}>Prikaži ime</button>
        </div>
      </div>
    );
  }
  export default App;
```

Iz navedenog primjera možemo uočiti da vrijednost polja za unos nije nigdje sačuvana unutar komponente već joj direktno pristupamo preko reference na *input* element. Također nemamo direktan ispis vrijednosti u nekom drugom elementu kao u primjeru sa upravljanim komponentama. Razlog je što promjena vrijednosti na DOM elementu ne uzrokuje ponovno renderiranje komponente već nam je za to potrebna promjena stanja. Rješenje bi bilo da dohvaćenu vrijednost spremimo u varijablu stanja komponente ali u tom slučaju je logičnije koristiti pristup sa upravljanim elementima, kada već moramo koristiti stanje komponente.

Odabir pristupa

Na kraju se postavlja pitanje koji od ova dva pristupa odabrati? Točan odgovor ne postoji već odabir ovisi o osobnim preferencijama i onome što je prikladnije za konkretni slučaj. Upravljane komponente pružaju bolju kontrolu nad podacima ali ne moramo nužno imati potrebu za tim. Ukratko možemo sumirati:

- Kontrolirane komponente su pouzdanije u smislu da podacima upravlja komponenta kroz svoja stanja. U tom slučaju React je zadužen za sinkronizaciju, osvježavanje i prikaz stanja i sigurni smo da uvije imamo točne vrijednosti podataka.
- Kontrolirane komponente pružaju jednostavniji način validacije podataka, ponovno iz razloga jer su vezani uz stanje komponente gdje možemo kontrolirati kako i kada će se promijeniti stanje komponente.
- Vrijednosti neupravljanih komponenti nisu uvijek pouzdane jer elementi za unos mogu izgubiti svoje reference i sami podaci mogu biti promijenjeni od strane nekih drugih izvora - promjene vrijednosti nisu usklađene sa stanjima komponente niti sa ciklusima renderiranja.

Sve navedeno navodi na zaključak da su upravljani elementi sigurniji odabir i to je često i slučaj ali kao što je već spomenuto, postoje i situacije u kojima su neupravljani elementi sasvim prikladan odabir. Također, na raspolaganju imate i gotove biblioteke koje olakšavaju rad sa formama za unos podataka od kojih su možda najpoznatije Formik i React Hook Form.

useContext

Praktični dio