JUNIORDev DIGITALNA DALMACIJA

Slanje podataka

U prethodnom primjeru smo vidjeli kako jednu komponentu možemo iskoristiti više puta. Kako bi se povećala mogućnost ponovnog korištenja komponenti, potrebno je omogućiti mogućnost njihove prilagodbe. Konkretno, naša prethodno napravljena komponenta za pozdrav bi bila puno korisnija kada bi mogli uz pozdrav ispisati ime korisnika kojeg želimo pozdraviti - naravno pri tome ime mora biti prilagodljivo.

React svojstva (*props*)

To se u Reactu postiže korištenjem tzv. "***props***" (skraćeno od *properties*). Glavni zadatak *props*-a je slanje podataka iz jedne komponente u drugu i možemo reći da služe kao glavni komunikacijski kanal između komponenti. Glavne karakteristike *props*-a su:

- mogu se koristiti u funkcijskim i klasnim komponentama
- svojstva (*props*) se šalju od vrha prema dolje tj. roditeljske (*parent*) komponente šalju podatke komponentama potomcima (*child*)
- *props* se mogu samo čitati (*read-only*). Jednom kada komponenta primi podatke, ne može ih modificirati već samo koristiti ili proslijediti dalje (svojim *child* komponentama).

Korištenje *props*-a

Pogledajmo na konkretnom primjeru kako koristimo React *props*. Iskoristiti ćemo prethodno napravljenu komponentu Pozdrav i prilagoditi je tako da može primati *props*.

Pozdrav.jsx

```
function Pozdrav(props) {
```

```
    return (
        <>
            <h3>Pozdrav od React funkcijske komponente</h3>
            <p>Dobar dan, {props.ime}</p>
        </>
    )
}
```

Obratite pozornost na dvije stvari u prethodnom primjeru. U potpis funkcije smo dodali ulazni argument *props* i tako naznačili da naša komponenta prima neke podatke. Zatim smo u tijelu komponente iskoristili svojstvo *props.ime* tj. pristupili smo podacima koje smo dobili.

Sada nam još preostaje vratiti se na "glavni" dio aplikacije, tamo gdje koristimo ovu komponentu, i poslati odgovarajuće podatke koji su potrebni za ispravan prikaz komponente:

App.jsx

```
return (
        <div>
            <p>Dobar dan React, danas je {datum.toString()}</p>
            <p className='izracun'>
                {a} plus {b} je {a + b}
            </p>
            <Pozdrav ime="Matea" />
        </div>
)
```

Spremite promjene i pogledajte rezultat u pregledniku. Komponenta bi trebala ispravno prikazati primljeni parametar - ime osobe koju želimo pozdraviti. Sada možemo ovu komponentu koristiti svaki put kada želimo ispisati pozdrav, a pri tome imamo mogućnost promjene imena.

App.jsx

```
<Pozdrav ime="Matea" />
<Pozdrav ime="Ivan" />
<Pozdrav ime="Maja" />
```

Kao što je vidljivo iz primjera, *props* se koriste na sličan način kao i atributi unutar HTML

oznaka. Unutar oznake za komponentu navodimo ime svojstva i pripadajuću vrijednost. Naravno, nismo ograničeni na jedno svojstvo, pa komponenti možemo poslati više podataka.

App.jsx

```
return (
        <div>
            <p>Dobar dan React, danas je {datum.toString()}</p>
            <p className='izracun'>
                {a} plus {b} je {a + b}
            </p>
            <Pozdrav ime="Matea" god={27} />
        </div>
)
```

Ne zaboravite prilagoditi komponentu i prikazati nove podatke:

Pozdrav.jsx

```
function Pozdrav(props) {
    return (
        <>
            <h3>Pozdrav od React funkcijske komponente</h3>
            <p>Dobar dan, {props.ime}</p>
            <p>Imaš {props.god} godina.</p>
        </>
    )
}
```

Uočite razliku u sintaksi kada je vrijednost svojstva u tekstualnom obliku (*string*) i kada je vrijednost svojstva broj. Zapravo bi u oba slučaja trebali koristiti `{ }` zagrade ali kada je u pitanju *string* React nam dozvoljava skraćeni zapis. Komponentu smo mogli instancirati i na ovaj način sa identičnim rezultatom:

```
<Pozdrav ime={"Matea"} god={27} />
```

Možemo pogledati još jedan primjer slanja podataka kroz *props*. U ovom primjeru kao dvije vrijednosti koristimo internu varijablu i aritmetičku operaciju:

App.jsx

```
function App(){
    const osoba = "Ivan";
    return (
        <div>
            <Pozdrav ime={"Matea"} god={27} />
            <Pozdrav ime={osoba} god={10 + 18} />
        </div>
    )
}
```

children svojstvo

Svi dosadašnji primjeri su sadržavali komponente samozatvarajućeg oblika - npr. `<Pozdrav />`. Iako ta komponenta unutar sebe može sadržavati druge (ugniježđene) komponente, u samoj komponenti je točno definirano što će se u njoj nalaziti. Ponekad ćemo imati potrebu napraviti komponentu unutar koje želimo ugnijezditi različite elemente ili nam neće biti unaprijed poznato što će se nalaziti unutar nje. Jedan od primjera može biti komponenta koja sadrži `<div>` element sa nekim zadanim oblikovanjem - npr. zaobljeni okvir fiksnih dimenzija unutar kojeg želimo prikazati različiti sadržaj.

U takvim slučajevima unutar komponente koristimo ugrađeno svojstvo ***props.children*** koje predstavlja ugniježđene elemente naše komponente bez da unaprijed znamo koji su. Pogledajmo primjer - komponentu možemo definirati ovako:

Pozdrav.jsx

```
function Pozdrav(props) {
    return (
        <>
            <h3>Pozdrav od React funkcijske komponente</h3>
            <p>Ispod se prikazuje ugniježđeni sadržaj</p>
            {props.children}
        </>
    )
}
```

I sada možemo koristiti tu komponentu, ali na način da ima otvarajuću i zatvarajuću oznaku te da se između njih mogu nalaziti ***child*** elementi.

App.jsx

```
function App(){
    return (
        <div>
            <Pozdrav>
                <p>Ovo je child element komponente</p>
            </Pozdrav>
        </div>
    )
}
```

Na ovaj način možemo unutar komponente "ubaciti" koji god sadržaj želimo i on će se prikazati umjesto `{props.children}` izraza. Kao što vidite na primjeru, nije potrebno ugniježđeni sadržaj posebno označavati kao *children* svojstvo, to React radi automatski.

Pomoćne funkcije

React komponente također mogu sadržavati pomoćne funkcije. Nadograditi ćemo prethodni primjer te u komponentu dodati funkciju za računanje godine rođenja.

Pozdrav.jsx

```
function Pozdrav(props) {
    const godRod = () => {
        let rez = new Date().getFullYear() - props.god
        return rez
    }
    return (
        <>
            <h3>Pozdrav od React funkcijske komponente</h3>
            <p>Dobar dan, {props.ime}</p>
            <p>Imaš {props.god} godina.</p>
            <p>Vjerojatno si rođen(a) {godRod()}. godine</p>
        </>
    )
```

```
}

export default Pozdrav
```

Logiku izračuna godine rođenja smo izdvojili u posebnu funkciju koja se nalazi unutar same komponente - kao i njen poziv. Uočite da pomoćnoj funkciji nije potrebno slati godinu kao argument jer se ona nalazi unutar komponente i ima pristup *props* objektu.

Destrukturiranje

U slučajevima kada više puta unutar komponente pristupamo nekoj vrijednosti iz *props* objekta (ili kad imamo više podataka), stalno pisanje `props.*` može postati naporno i umanjuje čitljivost kôda. Taj problem se može riješiti korištenjem mogućnosti koju nam pruža JS sintaksa - a to je **destrukturiranje**.

Destrukturiranje nam omogućava "izvlačenje" vrijednosti iz *props* objekta i njihovo korištenje unutar komponente u skraćenom zapisu. Pogledajmo prethodni primjer sa destrukturiranim *props* objektom:

Pozdrav.jsx

```
function Pozdrav({ime, god}) {
    const godRod = () => {
        let rez = new Date().getFullYear() - god
        return rez
    }
    return (
        <>
            <h3>Pozdrav od React funkcijske komponente</h3>
            <p>Dobar dan, {ime}</p>
            <p>Imaš {god} godina.</p>
            <p>Vjerojatno si rođen(a) {godRod()}. godine</p>
        </>
    )
}

export default Pozdrav
```

Nakon što smo destrukturirali *props* objekt i iz njega "izvukli" argumente `ime` i `god`, možemo ih koristiti kao lokalne varijable unutar komponente. Ukratko, sa destrukturiranjem smo postigli isti učinak kao da smo sami na početku komponente deklarirali dvije lokalne varijable i u njih spremili vrijednosti iz *props* objekta. Možemo vidjeti i takav primjer, samo da bi dobili dojam postignutog učinka:

Pozdrav.jsx

```
function Pozdrav(props) {
    const ime = props.ime
    const god = props.god
    const godRod = () => {
        let rez = new Date().getFullYear() - god
        return rez
    }
    return (
        <>
            <h3>Pozdrav od React funkcijske komponente</h3>
            <p>Dobar dan, {ime}</p>
            <p>Imaš {god} godina.</p>
            <p>Vjerojatno si rođen(a) {godRod()}. godine</p>
        </>
    )
}

export default Pozdrav
```

Last updated on March 7, 2023

Komponente

Praktični dio