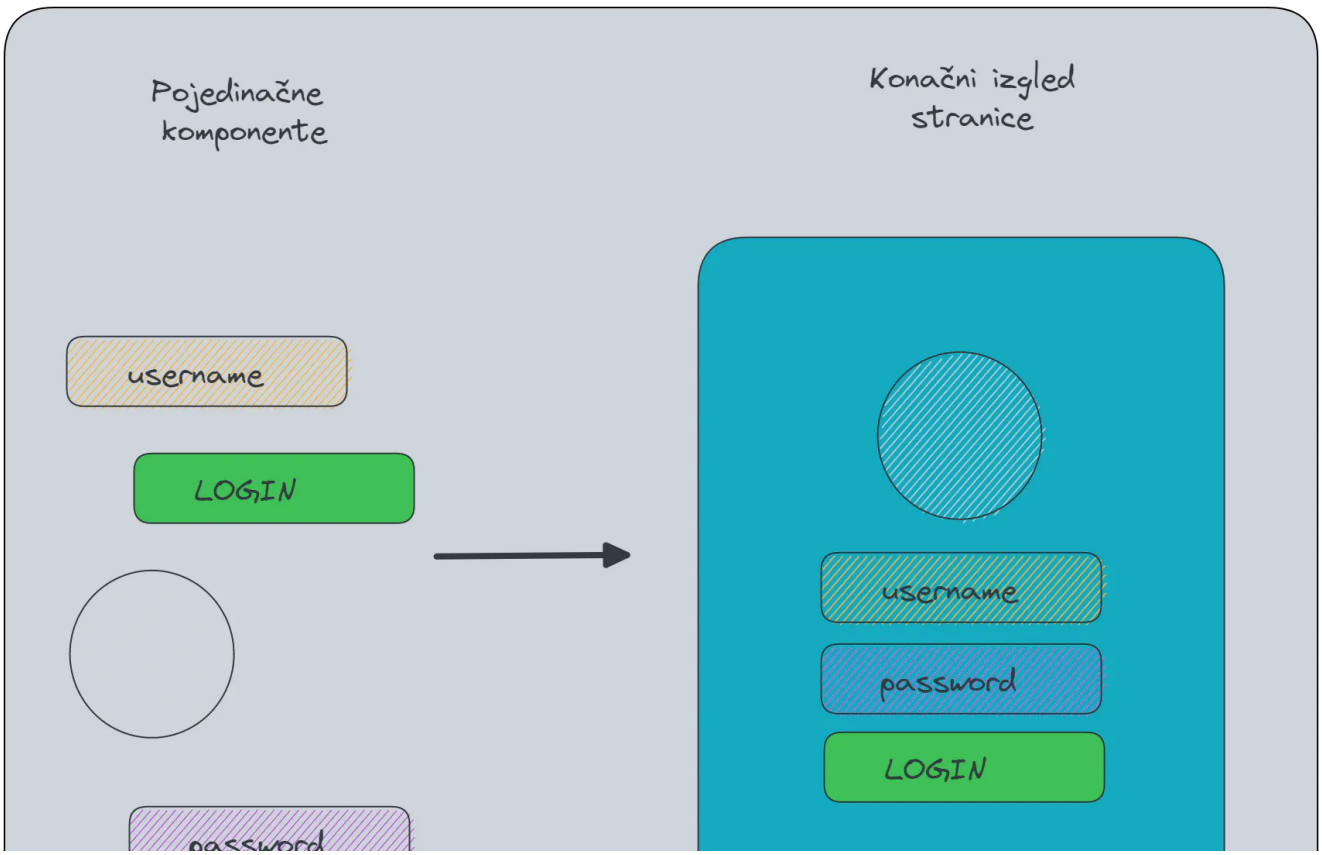
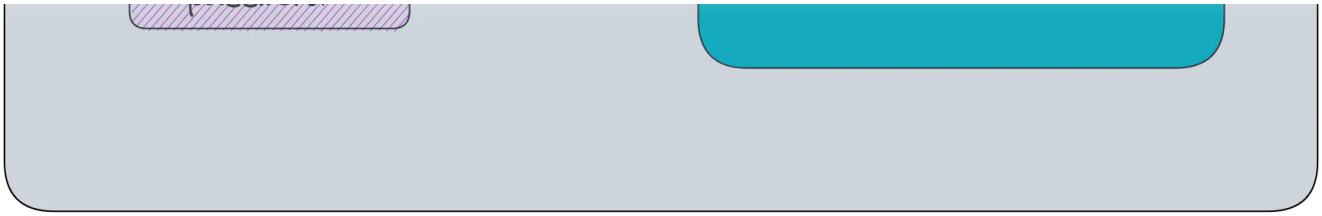
JUNIORDev
DIGITALNA DALMACIJA

React komponente

Komponente su jedan od osnovnih koncepata Reacta i često se nazivaju "građevnim blokovima" React aplikacije. Najjednostavnije rečeno, komponente su pojedinačni elementi stranice. Kod "klasičnih" web stranica, izgled i struktura stranice bi se definirali kroz HTML strukturu, za svaku pojedinačnu stranicu - unutar `<body>` elementa bi definirali strukturu cijele aplikacije uz pomoć HTML-a te bi dodali oblikovanje (CSS) i eventualno mogućnost interakcije (JS). Često bi se u takvim situacijama javila potreba za ponovnim korištenjem dijela kôda što bi najčešće rezultiralo njegovim kopiranjem sa jedne stranice na drugu. Naravno, problem sa takvim pristupom je održavanje i izmjena dijelova stranice, koje sa porastom stranice postaje sve složenije. Ideja React komponenti je kombinirati HTML, CSS i JavaScript u jednu logičku cjelinu (komponentu) koja se može koristiti više puta na različitim dijelovima stranice. Time se postiže jedna od glavnih principa React-a, a to je **ponovno korištenje** (eng. **Reusability**)





Korištenjem komponenti možemo korisničko sučelje (UI) "razbiti" na više neovisnih elemenata i svaki od tih elemenata promatrati kao izoliranu cjelinu.

Pisanje komponenti

Komponente u React-u možemo pisati na dva načina - u obliku **funkcije** ("*Function Components*" - funkcijske komponente) ili u obliku **klase** ("*Class Components*" - klasne komponente)

U primjeru ispod možete vidjeti jednostavnu React komponentu napisanu na oba načina:

Funkcija Klasa

Pozdrav.jsx

```
function Pozdrav() {
  return <h3>Pozdrav od React funkcijske komponente</h3>
}
```

Iz perspektive samog React-a, ove dvije komponente su identične, tj. rezultiraju istim prikazom. U starijim verzijama React-a postojala je razlika između ove dvije vrste komponenti - funkcijske komponente su se koristile samo za prikaz statičkog sadržaja i u slučajevima kada komponenta nije trebala imati dodatnu unutarnju logiku. Zbog toga su se funkcijske komponente često nazivale *prezentacijske* ili čak "glupe" (eng. *dumb*) komponente. S druge strane, klasne komponente su se koristile za zahtjevnije zadatke jer su imale mogućnost pamćenja unutarnjih stanja i izvršavanja složenijih logičkih operacija (nazivale su se *smart* ili *container* komponentama).

Pojavom posebnih funkcija koje se nazivaju **React Hooks**, funkcijske komponente su dobile mogućnost pamćenja stanja i upravljanja životnim ciklusom komponente, te se izgubila podjela

na "pametne/glupe" komponente. Dapače, zbog nešto jednostavnije sintakse, funkcijske komponente su postale standard te se u praksi koriste češće od klasnih komponenti. Zbog toga će i u ostatku tečaja svi primjeri biti navedeni u obliku funkcijskih komponenti, te ćemo se također posebno osvrnuti na React Hooks i njihovo korištenje.

Korištenje komponenti

Kroz jednostavni primjer ćemo demonstrirati pisanje nove komponente i njeno korištenje. U prethodno napravljenom početnom projektu, napravite novi direktorij "**components**", te u njemu datoteku "**Pozdrav.jsx**. Za početak možemo kopirati gornji primjer, samo nam još na kraju nedostaje naredba za izvoz (export) cijele funkcije kako bi je mogli uključiti u drugu datoteku.

components/Pozdrav.jsx

```
function Pozdrav() {
 return <h3>Pozdrav od React funkcijske komponente</h3>
}
export default Pozdrav
```

Nakon što smo definirali našu komponentu, možemo se vratiti u glavni dio aplikacije ("*App.js*") te unutar njega učitati našu funkcijsku komponentu. Ako smo ispravno naveli putanju do datoteke, možemo unutar App funkcije stvoriti jednu instancu naše komponente "Pozdrav".

App.js

```
import Pozdrav from "./components/Pozdrav";
import './App.css'
function App(){
    const datum = new Date()
    const a = 10
    const b = 20

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

```
            </p>
            <Pozdrav />
        </div>
    )
}
 export default App
```

Višestruko korištenje komponenti

Kao što je već prije spomenuto, glavna svrha komponenti je mogućnost ponovne upotrebe. Jednom kada smo definirali komponentu "Pozdrav", možemo je više puta iskoristiti, čak i na istoj stranici. Sve što je potrebno je napraviti novu instancu te 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 />
        <Pozdrav />
        <Pozdrav />
    </div>
)
```

Odmah možemo uočiti prednost ovog pristupa sa komponentama. Na primjer, uočili smo neku pogrešku ili jednostavno želimo promijeniti izgled naše komponente za pozdrav. U tom slučaju je dovoljno otvoriti datoteku u kojoj smo definirali komponentu (njen funkcijski oblik) te napraviti željenu promjenu. Ta promjena će se odraziti na cijelu aplikaciju - gdje god smo koristili našu komponentu. Ovim pristupom smo izbjegli situaciju u kojoj moramo pretraživati cijelu aplikaciju kako bi pronašli dijelove kôda koje moramo ažurirati.

Postoji i jedan potencijalni problem - što ako želimo napraviti komponentu koja će većim dijelom biti identična ali će se razlikovati po sadržaju. Na primjer, bilo bi dobro kada bi naša komponenta mogla uz pozdrav ispisivati ime korisnika - ali to ime mora biti promjenjivo inače bi morali raditi komponentu za svako moguće ime, čime bi izgubili smisao ponovnog

korištenja.

Srećom, React komponente imaju mogućnost slanja i primanja parametara čime postaju još korisnije.

Last updated on March 7, 2023

‹ JSX sintaksa

Slanje podataka ›