JUNIORDev DIGITALNA DALMACIJA

Komunikacija > Oblikovanje

Izgled React komponenti

U svim dosadašnjim primjerima se nismo posebno osvrtali na oblikovanje samih komponenti. Vratimo se kratko na početni predložak aplikacije, napraviti ćemo početni predložak i jednostavnu strukturu aplikacije (postojeće CSS datoteke ne morate modificirati, samo "App.jsx" komponentu)

App.jsx

```
import './App.css'

function App() {
  return (
    <div className="App">
      <h1>Pozdrav React!</h1>
      <p>Tekst u glavnoj komponenti!</p>
    </div>
  )
}
export default App
```

Ovdje vidimo primjer učitavanja CSS datoteke `App.css` koja sadrži standardne selektore i pravila. Iako je ovaj način prilično jednostavan, postoji jedan problem kojeg ćemo demonstrirati na primjeru. Napravite novu komponentu "*Kartica*" sa sljedećim sadržajem:

Kartica.jsx

```
function Kartica(){
    return(
        <div>
            <h1>Naslov kartice</h1>
            <p>Tekst kartice</p>
        </div>
    )
}
```

```
export default Kartica
```

Zatim je možemo odmah i uključiti u glavni dio aplikacije (ne zaboravite na `import`)

App.jsx

```
function App() {
  return (
    <div className="App">
      <h1>Pozdrav React!</h1>
      <p>Tekst u glavnoj komponenti!</p>
      <Kartica />
    </div>
  )
}
```

Sada nam je želja oblikovati samo komponentu "Kartica" tj. njene elemente. Možemo napraviti datoteku "Kartica.css", dodati po jedno pravilo za `<h1>` i `<p>` elemente i naravno uključiti CSS datoteku u samu komponentu.

Kartica.jsx Kartica.css

```
import './Kartica.css'

function Kartica() {
  return (
    <div>
      <h1>Naslov kartice</h1>
      <p>Tekst kartice</p>
    </div>
  );
}
export default Kartica;
```

Rezultat ovoga je možda neočekivan. Stilovi koje smo napisali u datoteci `Kartica.css` su se primijenili na sve elemente aplikacije iako smo tu datoteku uključili (*import*) samo unutar komponente `Kartica.jsx`. Razlog leži u već spomenutom ***bundler*** alatu i načinu na koji se React aplikacije "spajaju" u jednu cjelinu. Bez obzira u koju komponentu uključite CSS datoteku

(na ovaj način!) ona će se dodati u aplikaciju kao **globalni stil** i vrijediti će za sve elemente koju odgovaraju napisanim CSS selektorima.

Jedno od potencijalnih rješenja je naravno korištenje klasa (atribut `cLassName`), gdje bi svaka komponenta sadržavala imena klasa jedinstvenih za tu komponentu. Na taj način bi postigli željeni učinak ali ovo nije idealno rješenje iz dva razloga. Prvi je što sve stilove učitavamo globalno i to može utjecati na performanse aplikacije. Drugi problem je što je kod složenijih aplikacija teško pamtiti sva imena klasa i može vam se dogoditi da ponovite neko ime klase - kasnije je takve probleme teško uočiti i ispraviti. Zbog toga ćemo pogledati još nekoliko pristupa oblikovanja React komponenti.

Inline stilovi

Najjednostavniji način za postići jedinstveno oblikovanje je pisanje *inline* stilova koristeći `style` atribut kojemu kao vrijednost dodijelimo objekt sa pravilima oblikovanja. Prethodni primjer bi u tom slučaju izgledao ovako:

```
function Kartica() {
  return (
    <div>
      <h1 style={{color: "red"}}>Naslov kartice</h1>
      <p style={{textDecoration: "underline"}}>Tekst kartice</p>
    </div>
  );
}
export default Kartica;
```

Sada smo sigurni da će se ovi stilovi odnositi samo na pripadajuće elemente. Uočite dva detalja u navedenom primjeru. Atribut `style={}` prima **objekt** (preciznije *Style objekt*) kao vrijednost pa zbog toga imamo dvije vitičaste zagrade na početku i kraju izraza. Drugi važan detalj je pisanje samih CSS pravila tj. njihova sintaksa. Kod *inline* pravila koristimo *JavaScript* sintaksu koja se malo razlikuje od klasične CSS sintakse. Glavna razlika je da se svojstva koja u CSS-u pišemo sa crticom u imenu (npr. "*text-size*", "*background-color*" ili "*border-radius*"), u JS-u pišu koristeći već spomenuti *camelCase* - dakle "*textSize*", "*backgroundColor*" i "*borderRadius*".

Kod pisanja složenijih pravila postaje prilično nepregledno sve pisati uz samu oznaku pa možete *Style* objekt izdvojiti izvan komponente, na vrh (ili dno) datoteke.

```
function Kartica() {

  return (
    <div>
      <h1 style={h1Stil}>Naslov kartice</h1>
      <p style={pStil}>Tekst kartice</p>
    </div>
  );
}

 const h1Stil = {
  color: "red",
  fontSize: 22,
  border: "1px solid black"
 }

 const pStil = {
  textDecoration: "underline",
  padding: "20px 15px",
  backgroundColor: "#ccc"
 }

export default Kartica;
```

Mogu se svi stilovi staviti u isti objekt (pa tako čak i odvojiti u posebnu datoteku)

```
function Kartica() {
  return (
    <div>
      <h1 style={karticaStilovi.h1Stil}>Naslov kartice</h1>
      <p style={karticaStilovi.pStil}>Tekst kartice</p>
    </div>
  );
}

const karticaStilovi = {
  h1Stil: {
    color: "red",
    fontSize: 22,
```

```
    border: "1px solid black",
  },
  pStil: {
    textDecoration: "underline",
    padding: "20px 15px",
    backgroundColor: "#ccc",
  },
};

export default Kartica;
```

Iako ovaj pristup izgleda jednostavan, u službenoj React dokumentaciji se preporuča korištenje `className` atributa zbog boljih performansi aplikacije. Prema istoj bi korištenje ***style*** svojstva trebalo biti ograničeno za potrebe dinamičkih stilova koji se računaju prilikom renderiranja pa ćemo pogledati još jedan način kako povezati CSS datoteku sa komponentom.

CSS moduli

Alternativni način je *import* CSS datoteka kao **modula**. Princip je zapravo vrlo jednostavan. Za početak možemo napisati klasičnu CSS datoteku sa standardnom sintaksom ali **ime datoteke** mora završavati sa ekstenzijom ***.module.css***

Kartica.module.css

```
/* Kartica.module.css */
.naslov{
  color: red;
  font-size: 22px;
  border: 1px solid black;
}
.odlomak{
  text-decoration: underline;
  padding: 20px 10px;
  background-color: #ccc;
}
```

Idući korak je u React komponenti uključiti željeni modul. Razlika je što ovaj put CSS datoteku uključujemo kao objekt. Na taj način su nam sve klase koje smo definirali u CSS modulu

dostupne kao svojstva uključenog objekta (možemo koristiti standardnu "`.`" ili "`[]`" notaciju za pristupanje svojstvima objekta). Pripazite na korištenje atributa `className`.

Kartica.jsx

```jsx
import stil from './Kartica.module.css'

function Kartica() {
  return (
    <div>
      <h1 className={stil.naslov}>Naslov kartice</h1>
      <p className={stil["odlomak"]}>Tekst kartice</p>
    </div>
  );
}

export default Kartica;
```

Ako pogledamo u *Developer Tools* prozoru konačni rezultat stranice, možete uočiti nešto slično ovome:

```
<p class="_odlomak_xd5yk_15">Tekst kartice</p>
```

Kao što vidite, React je automatski napravio ono što je bilo predstavljeno kao potencijalno rješenje - elementu u komponenti je generirao jedinstveno ime klase (kombinacija imena kojeg smo naveli u CSS datoteci i nasumičnog teksta). Na ovaj način možemo pisati jednostavna imena klasa bez straha da će nam se neko ime ponoviti unutar projekta.

> Pravila u CSS modulima bi trebala biti navedena unutar selektora klase. Ukoliko napišete "obični" selektor elementa (npr `h1{...}`) pravila navedena unutar njega će se također primijeniti kao **globalna** pravila ali zato možete koristiti kombinirane selektore, pseudoklase i *media* upite kao i uobičajeno.

Osim ovih "ugrađenih" metoda, React komponente možemo oblikovati korištenjem nekih gotovih biblioteka pa ćemo se kratko osvrnuti i na taj pristup.

Last updated on March 28, 2023

Stilizirane komponente >