JUNIORDev DIGITALNA DALMACIJA

Komunikacija > Stilizirane komponente

Stilizirane komponente

Jedan od alternativnih načina oblikovanja React komponenti je korištenje ***styled components*** biblioteke koja pruža mogućnost izrade tzv. stiliziranih komponenti. Ovaj pristup ima nekoliko prednosti kao djelomično učitavanje pravila (samo onih koja su potrebna), automatsko generiranje imena klasa, mogućnost dinamičkog oblikovanja i prilagodbu za različite vrste preglednika.

S obzirom da se radi o vanjskoj biblioteci koja nije ugrađena u React, prvi korak je instalacija paketa u naš projekt. Instalaciju ćemo napraviti putem Terminala koji je ugrađen u VS Code (i sve moderne uređivače kôda). Otvorite novi prozor terminala (ili zaustavite razvojni poslužitelj React aplikacije sa `CTRL+C`), provjerite da ste pozicionirani u glavnom direktoriju aplikacije te upišite naredbu:

```
npm install styled-components
```

Nakon kratke instalacije možete ponovno pokrenuti aplikaciju sa "`npm run dev`" ili "`npm start`" (ovisno koji predložak koristite) i početi koristiti stilizirane komponente.

Sintaksa

Započnimo sa sintaksom *styled* komponenti. Prvi korak je naravno uključivanje ***styled*** objekta iz instalirane biblioteke. Iskoristiti ćemo postojeću komponentu "Kartica" (i ukloniti prethodna oblikovanja):

```
import styled from 'styled-components'

function Kartica() {
```

```
  return (
    <div>
      <h1>Naslov kartice</h1>
      <p>Tekst kartice</p>
    </div>
  );
}

export default Kartica;
```

Sada možemo unutar komponente definirati stilizirane elemente. Biblioteka *styled components* nam omogućava izradu stiliziranih verzija postojećih HTML elemenata u obliku novih komponenti. Osnovna sintaksa izgleda ovako:

```
const Ime = styled.oznakaDOM`
  cssSvojstvo: cssVrijednost
`
```

Pri čemu je:

- `Ime` - ime naše stilizirane komponente (po želji)
- `oznakaDOM` - HTML/JSX element kojeg želimo oblikovati
- `cssSvojstvo` - bilo koje važeće CSS svojstvo
- `cssVrijednost` - ispravna vrijednost za pripadajuće svojstvo

Još jedan ključni detalj u sintaksi je da se CSS pravila pišu unutar ***template literals*** znakova - tj. literala odvojenih ***backtick*** znakovima (`` ` ``) - `AltGr+7`. Na prvi pogled ovo izgleda čudno ali najjednostavnije je to shvatiti kao poziv funkcije kojoj šaljemo argumente samo umjesto klasičnih zagrada koristimo *backtick* oznake.

Primjer korištenja

Za početak ćemo u našoj komponenti napraviti stiliziranu verziju `<h1>` elementa:

Kartica.jsx

```jsx
import styled from 'styled-components'

const Naslov = styled.h1`
  color: red;
  font-size: 22px;
  border: 1px solid black
`

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

Prva razlika koja se ističe je da sada koristimo CSS sintaksu sa imenima koja su odvojena crticom (`font-size`), a ne JS *camelCase* verziju (`fontSize`). Za one koji su navikli na CSS ovo je prilično olakšanje.

Nakon što smo imenovali našu komponentu, odabrali pripadajući HTML element i napisali CSS pravila, možemo u `return` izrazu zamijeniti "obični" HTML element sa našom stiliziranom verzijom. Pokušajte samostalno na isti način napraviti stiliziranu verziju `<p>` elementa.

Dinamički stil

Još jedna prednost stiliziranih komponenti je mogućnost prilagodbe stila ovisno o svojstvima (***props***) koje šaljemo komponenti. Na primjer želimo napraviti stiliziranu verziju `<button>` elementa ali takvu da joj možemo promijeniti boju ovisno o atributu kojeg pošaljemo. Ovdje do izražaja dolazi korištenje *template* literala jer možemo umjesto *string*-a interpolirati funkciju. Najbolje je pogledati primjer (izostavljeni su prethodno napravljeni stilovi):

Kartica.jsx

```
const Button = styled.button`
  font-size: 1em;
  margin: 1em;
  padding: 0.25em 1em;
  border: 2px solid tomato;
  border-radius: 3px;
  background: ${props => props.rozi ? "salmon" : "white"};
`

function Kartica() {
  return (
    <div>
      <Naslov>Naslov kartice</Naslov>
      <Odlomak>Tekst kartice</Odlomak>
      <Button>Klik</Button>
      <Button rozi>Klik</Button>
    </div>
  );
}
```

Iskoristili smo mogućnost dinamičkog oblikovanja na način da se komponenti mijenja boja pozadine ovisno jesmo li joj naveli atribut "*rozi*" ili ne. Na sličan način smo mogli i napisati pravilo za odlomak teksta tako da možemo sami definirati veličinu fonta:

```
const Odlomak = styled.p`
  text-decoration: underline;
  padding: 20px 15px;
  background-color: #ccc;
  font-size: ${props => props.velicina}px;
`
```

Sada pri korištenju komponente moramo navesti i atribtut "*velicina*"

```
<Odlomak velicina={17}>Tekst kartice</Odlomak>
```

Atributi elemenata

Prilikom definiranja stiliziranih komponenti važno je odrediti koja HTML komponenta se koristi kao predložak jer će nova komponenta automatski naslijediti atribute koja ima i originalna verzija. Na primjer, kada napravimo stiliziranu verziju `<button>` elementa, možemo joj dodijeliti sve atribute kao i izvornoj verziji (npr. *onClick* ili *disabled*):

```
function Kartica() {

  const [broj, postaviBroj] = useState(0)

  return (
    <div>
      <Naslov>Naslov kartice</Naslov>
      <Odlomak velicina={17}>Broj kartice: {broj}</Odlomak>
      <Button rozi onClick={() => postaviBroj(broj + 1)}>Uvećaj</Button>
    </div>
  );
}
```

Kao što se vidi na primjeru, izradom stilizirane verzije nismo izgubili nijednu funkcionalnost izvornog `<button>` elementa.

Stilizirane komponente imaju još mnoštvo mogućnosti kao što su nasljeđivanje (između stiliziranih komponenti), promjena izvornog elementa, korištenje CSS predprocesora i još puno toga. S obzirom da fokus ovog tečaja nije isključivo na dizajnu i oblikovanju React komponenti za sada nećemo ići dublje u detalje, a ukoliko vam je ovaj pristup oblikovanju zanimljiv i želite dodatno naučiti o njemu, svakako posjetite službenu dokumentaciju koja ima mnoštvo primjera sa detaljnim opisima svih mogućnosti.

Alternative

Postoji još mnoštvo alternativnih načina na koje možete pristupiti problemu oblikovanja. Svaki

od njih ima svojih prednosti i nedostataka i na kraju se izbor uglavnom svede na pristup koji vam je osobno najdraži, sa kojim ste upoznati ili koji vam je zadan (od strane poslodavca ili klijenta). Neke od alternativa su:

- Tailwind - CSS *framework* koji se temelji na korištenju tzv. *utility* klasa umjesto pisanja vlastitog CSS kôda
- React Bootstrap - React verzija popularnog CSS okvira za oblikovanje web aplikacija
- Storybook - Web alat za izradu UI komponenti. Možete stvoriti svoju vlastitu kolekciju komponenti te također podržava mogućnosti testiranja i izrade dokumentacije.

Last updated on March 28, 2023

Oblikovanje

Klijent-poslužitelj