JUNIORDev DIGITALNA DALMACIJA

Uvod u React > Uvod

Uvod

U prethodnom poglavlju smo napravili i proučili početnu strukturu naše Vite React aplikacije. Sada ćemo se malo detaljnije osvrnuti na datoteke unutar *src* direktorija i način na koji ćemo izrađivati aplikaciju.

Početni dio aplikacije

Početna točka aplikacije je datoteka **main.jsx** koja izgleda ovako:

main.jsx

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App'
import './index.css'

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
)
```

Iz ove datoteke možete uočiti da pozivom *ReactDOM.createRoot* metode stvaramo početnu točku (*root*) React aplikacije i to na HTML elementu sa ID atributom "*root*" (to je definirano unutar *index.html* datoteke). Zatim se nad tom početnom točkom poziva metoda *render* koja služi za iscrtavanje (renderiranje) React komponenti. U ovom konkretnom primjeru iscrtavamo komponentu pod nazivom "**App**" koju smo na početku učitali sa naredbom *import*. Osim te komponente, na početku datoteke smo također učitali React i ReactDOM biblioteke te CSS datoteku kako bi primijenili oblikovanje na razini cijele aplikacije.

App komponenta

Sada ćemo pogledati i sadržaj datoteke "**App.jsx**" tj. naše "glavne" React komponente koja predstavlja strukturu cijele aplikacije:

App.jsx

```jsx
import { useState } from 'react'
import reactLogo from './assets/react.svg'
import './App.css'

function App() {
  const [count, setCount] = useState(0)

  return (
    <div className="App">
      <div>
        <a href="https://vitejs.dev" target="_blank">
          <img src="/vite.svg" className="logo" alt="Vite logo" />
        </a>
        <a href="https://reactjs.org" target="_blank">
          <img src={reactLogo} className="logo react" alt="React logo" />
        </a>
      </div>
      <h1>Vite + React</h1>
      <div className="card">
        <button onClick={() => setCount((count) => count + 1)}>
          count is {count}
        </button>
        <p>
          Edit <code>src/App.jsx</code> and save to test HMR
        </p>
      </div>
      <p className="read-the-docs">
        Click on the Vite and React logos to learn more
      </p>
    </div>
  )
}

export default App
```

Ako usporedite sadržaj ove datoteke i prikaz naše aplikacije koju smo pokrenuli u pregledniku, možete otprilike dobiti dojam kako je definirana struktura aplikacije. Ukoliko napravite neku promjenu u datoteci (npr. neki dio teksta), možete vidjeti kako razvojni poslužitelj odmah prevodi novi verziju naše aplikacije i automatski osvježava prikaz u prozoru preglednika.

Za početak ćemo malo pojednostaviti ovu aplikaciju, te krenuti sa osnovnim konceptima i polako je nadograđivati. Zamijenite postojeći kôd aplikacije sa primjerom ispod

App.jsx

```
import './App.css'

const App = () => (
  <div>
    <p>Dobar dan React!</p>
  </div>
)

export default App
```

Zapravo se radi o skraćenom zapisu JS funkcije koja kao povratnu vrijednost vraća izraz u zagradama. Osnovna zadaća React komponente je definirati HTML strukturu koja predstavlja prikaz te komponente. Puni oblik gornje komponente bi mogli napisati i ovako:

App.jsx

```
function App() {
  return (
    <div>
      <p>Dobar dan React!</p>
    </div>
  )
}
```

⚠ Komponente možete imenovati po želji ali pazite da ime uvijek započinje velikim slovom.

S obzirom da je React komponenta zapravo JavaScript funkcija, unutar nje možemo pisati bilo koji ispravni JS kôd. Sve što je napisano prije "return" izraza će se izvršiti prilikom renderiranja (prikazivanja) komponente na zaslonu. Za početak možemo dodati jednostavnu naredbu za ispis poruke na konzoli preglednika.

App.jsx

```
const App = () => {
  console.log("Pozdrav iz komponente")
  return (
    <div>
      <p>Dobar dan React!</p>
    </div>
  )
}
```

Spremite promjene unutar aplikacije, te u pregledniku provjerite ispisuje li se poruka svaki put kada osvježite (F5) stranicu. Ispis iz konzole možemo vidjeti na "DevTools" sučelju (pod pretpostavkom da koristimo Chrome preglednik, ali svi moderni preglednici imaju svoju verziju tog alata).

Dinamički sadržaj

Komponente mogu sadržavati dinamički sadržaj čija se vrijednost računa u trenutku prikaza komponente. Za razliku od "čistog" HTML-a koji je statički, unutar React komponente možemo prikazati vrijednost neke varijable. Na idućem primjeru možemo vidjeti kako unutar komponente prikazujemo vrijednosti dvije varijable i njihov zbroj (koji se dinamički računa). Uz vrijednost varijabli prikazujemo i trenutno vrijeme koje nije unaprijed zadano već će se prilikom svakog prikazivanja komponente stvoriti novi "Date" objekt i prikazati njegova vrijednost.

Dinamički sadržaj komponente se navodi unutar vitičastih zagrada - bilo koji JS kôd unutar tih zagrada će se evaluirati (izvršiti) i na tom mjestu će biti prikaz rezultat operacije ili vrijednost navedene varijable.

App.jsx

```
function App(){
  const datum = new Date()
  const a = 10
  const b = 20

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

Svaki put kada ručno osvježite stranicu, na zaslonu će se prikazati trenutno vrijeme.

Osnovno oblikovanje

Osvrnimo se kratko i na oblikovanje (dizajn) komponente. Pri vrhu same komponente imamo naredbu za uključivanje CSS datoteke koja se nalazi na istoj lokaciji kao i naša "*App.jsx*" datoteka.

```
import './App.css'
```

Ako pogledamo sadržaj te datoteke, vidjeti ćemo da se radi o standardnoj CSS datoteci sa poznatom sintaksom (selektori i pravila). Za početak možemo izbrisati većinu pravila (jer se odnose na strukturu iz predloška koju smo izbrisali), ostavite samo prvo pravilo za *root* element.

App.css

```
#root {
  max-width: 1280px;
```

```
  margin: 0 auto;
  padding: 2rem;
  text-align: center;
}
```

Napisati ćemo jedan *class* selektor u CSS-u i pravilo koje mijenja boju i veličinu teksta, te ćemo to pokušati primijeniti na našu komponentu. CSS dio je jednostavan, dodajemo novi selektor:

App.css

```
.izracun {
  font-size: 24px;
  color: #0887ee;
}
```

Sada moramo željenom HTML elementu dodati *class* atribut kako bi se mogao primijeniti definirani stil. Tu dolazimo do prve specifičnosti React sintakse. Za razliku od HTML-a, ključna riječ "**class**" je rezervirana u JavaScriptu (sjetite se, React komponenta je JS funkcija) pa moramo koristiti atribut "**classname**" koji se prilikom prevođenja u HTML strukturu pretvara u standardni oblik.

Naša (osvježena) komponenta izgleda ovako:

App.jsx

```
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>
    </div>
  )
}
```

Rezultat bi trebao biti uvećani prikaz teksta sa operacijom zbrajanja i to u plavoj boji. Prije nego nastavimo sa detaljima vezanima uz React komponente, kratko ćemo se osvrnuti i na samu sintaksu koja, kako smo vidjeli u prethodnom primjeru, sadrži neke specifičnosti koje bi bilo dobro upoznati na početku rada sa React-om.

‹ Instalacija JSX sintaksa ›