Naša prva komponenta

JSX

JSX (JavaScript XML) je sintaksno proširenje JavaScript-a koje po izgledu podseća na HTML ili XML i ustaljena je praksa u React aplikacijama. Korišćenjem JSX-a se opisuje kako će korisnički interfejs izgledati. Iako izgleda slično kao HTML/XML, kod pisan koristeći JSX se pretvara u JavaScript.

Primer JSX koda: ***const element = <h1>Hello, JSX!</h1>;***

JavaScript izrazi u JSX-u

Za razliku od HTML-a, JSX dozvoljava da se u same elemente uključi JavaScript kod koristeći velike zagrade (***{}***).

Primer:
const name = 'John';
const element = Hello, {name}! ;

HTML atributi u JSX-u

Uz JSX, moguće je koristiti atribute na način na koji se koriste u HTML-u. Vrednost ovih atributa može biti JavaScript kod korišćenjem ***{},*** ili **string**.

Upozorenje: zbog toga što je JSX suštinski JavaScript, kao naziv atributa se ne mogu koristiti neki HTML atributi, kao recimo, ***class***, zbog toga što je ***class*** rezervisana reč u JavaScript programskom jeziku. Umesto toga, koriste se alternativni atributi. U slučaju HTML ***class*** atributa, odgovarajuća JSX alternativa je ***className.***

Primer:
const element = <a href="https://www.krojacevaskola.com"
className="btn btn-primary"> Poseti kurs!;

JSX i React.js

Kada za potrebe pravljenja React komponenti koristimo JSX, ta sintaksa se transformiše u ekvivalentan JavaScript kod koji kreira drvo React elemenata (za potrebe konstruisanja Virtual DOM-a).

Proces je sledeći:

1. **Pisanje JSX koda**
    - Primer: ***const element = <h1>Hello, JSX!</h1>;***
2. **Transformacija**
    - JSX sintaksu je neophodno transformisati u kod koji React.js "razume".
    - Babel je jedan od najčešćih alata koji se koriste u ove svrhe.
    - JSX kod iz prethodnog primera bi bio tranformisan u sledeći kod: **const element = React.createElement('h1', null, 'Hello, JSX!');**
3. ***React.createElement()***
    - Ova funkcija služi za kreiranje React elemenata na osnovu prosleđenih argumenata.
    - Prvi parametar predstavlja tip elementa (HTML tag ili React komponenta).
    - Drugi parametar je objekat koji sadrži atribute elementa (ili ***null*** ukoliko nema argumenata).
    - Treći parametar su deca ovog elementa.

4. **Kreiranje virtuelnog DOM-a**
    - Rezultat ove funkcije je JavaScript objekat koji predstavlja simulaciju HTML elementa u virtual DOM-u.

React.js komponente

React.js komponente su, kako smo ranije pomenuli, osnovni gradivni elementi svake React.js aplikacije. Komponente su samostalni delovi koda koji imaju sopstvenu definiciju strukture, stila i logike. Svaka komponenta predstavlja određeni element korisničkog interfeja, i mogu biti sve od jednog dugmeta do kompetne veb stranice.

U React biblioteci, komponente se mogu implementirati kao **funckije**, i kao **klase**. S obzirom na to da u svetu React biblioteke postoji višegodišnji trend dominantnog korišćenja komponenti u vidu funckija umesto komponenti u vidu klasa, u nastavku dokumenta ćemo se baviti funkcionalnim komponentama, dok će materijal o klasnim komponentama biti dostupan na kraju kursa za one koji žele da znaju više.

Funkcionalne komponente

Predstavljaju se kao JavaScript funkcije. Ove funkcije mogu i ne moraju da uzimaju podatke putem parametara. React komponente se imenuju velikim početnim slovom. Zaglavlje može biti prazno, a može sadržati i poseban objekat props, o kome će biti više reči kasnije. Kod funkcionalnih komponenti, povratna vrednost jeste JSX kod koji će se renderovati u DOM drvo.

Primer funkcionalne komponente:

```
function MyComponent() {
    return (
        <div>
            <h1>Nasa prva komponenta</h1>
            <p> Vojin, jos jedna osoba koja ce upoznati svet React-a </p>
        </div>
    )
}
```

Kako je u React-u najčešća praksa da se svaka komponenta pravi u zasebnom .js (odnosno .jsx) fajlu, ako želimo da ovako definisanu komponentu koristimo u drugim fajlovima, važno je izvesti je koristeći sledeću ES6 sintaksu:

```
export default MyComponent;
```

Pozivanje funkcionalne komponente

Funkcionalne komponente se mogu pozvati na dva načina:

1. **Koristeći JSX**

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

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(
    <MyComponent />
)
```

2. **Koristeći ime funkcije sa zagradama iza njega**

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

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(
  MyComponent()
)
```

Prvi način je ustaljeniji.

Props

Pri pozivu komponenti, u React-u im možemo proslediti podatke kroz posebne atribute koji se nazivaju props, skraćeno od properties (na srpskom svojstva). Možemo da odaberemo bilo koji naziv za ove atribute, sve dok on nije isti kao naziv neke od postojećih generalnih atributa (***className***, ***style***, ***onClick***, ***href*** i drugi). Svi ovi atributi se zatim smeštaju u objekat props, koji kao ključ ima naziv atributa koji smo definisali, a kao vrednost ima vrednost tog atributa koju smo postavili.

Podsetnik: vrednost ovih atributa može biti bilo kakva JavaScript vrednost ili JavaScript izraz (ukoliko se obavije velikim zagradama *{}*), ali može biti i string.

Primer prosleđivanja props atributa komponenti:

```
const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(
  <React.StrictMode>
    <MyComponent imeUcenika='Vojin' nazivTehnologije='React' />
  </React.StrictMode>
)
```

U ovom primeru, komponenti MyComponent prosleđujemo dva ***props*** atributa, ***imeUcenika*** i ***nazivTehnologije.***

Primer učitavanja vrednosti props objekta:

```
function MyComponent(props) {
    return (
        <div>
            <h1>Nasa prva komponenta</h1>
            <p> {props.imeUcenika}, jos jedna osoba koja ce upoznati svet
                  {props.nazivTehnologije}-a
              </p>
        </div>
    )
}
```

Pošto je props samo objekat, česta praksa je korsistiti destruktuiranje vrednosti kako bi kod bio jednostavniji i pregledniji.

Primer istog koda koristeći destruktuiranje:

```
function MyComponent({ imeUcenika, nazivTehnologije}) {
    return (
        <div>
            <h1>Nasa prva komponenta</h1>
            <p> { imeUcenika }, jos jedna osoba koja ce upoznati svet
                { nazivTehnologije }-a
            </p>
        </div>
    )
}
```

Rezultat ovog koda izgleda ovako: