# React integracija i osnove

Upoznavanje različitih pristupa za izgradnju JavaScript aplikacija nastavljamo pričom o još jednoj, veoma popularnoj biblioteci namenjenoj frontend programiranju. Reč je o biblioteci koja se zove React.

U ovoj lekciji ćete, kroz priču o osnovnom načinu za integraciju React biblioteke, imati prilike da se upoznate sa nekim osnovnim mogućnostima takve biblioteke. U narednim lekcijama će priča o mogućnostima React biblioteke biti proširena, što će nam na kraju omogućiti da korišćenjem takve biblioteke razvijemo jednu realnu JavaScript aplikaciju.

## Šta je React?

React je JavaScript biblioteka koja se koristi za kreiranje prezentacionog sloja web aplikacija. Reč je o biblioteci otvorenog koda, koju je kreirala kompanija Facebook. React je prvi put objavljen u martu 2013. godine.

*Slika 8.1. React – logo*

React je biblioteka koju aktivno koristi kompanija Facebook za razvoj društvenih mreža Facebook i Instagram. Mogućnosti biblioteke React veoma su slične mogućnostima već ilustrovane Vue biblioteke. Centralne figure su komponente, koje omogućavaju modularnu izgradnju korisničkog okruženja. Pored toga, React omogućava efikasno kreiranje šablona korišćenjem jednog posebnog jezika koji se zove JSX. Takav jezik omogućava kombinovanje HTML i JavaScript jezika prilikom formulisanja šablona. Pored toga, korišćenjem biblioteke React moguće je veoma lako obaviti i sve uobičajene operacije prilikom izgradnje korisničkog okruženja jedne web aplikacije: generisanje prezentacije, povezivanje podataka, rukovanje događajima…

**Pitanje**

React je kreirala kompanija:

**a) Facebook**
b) Twitter
c) Google
d) Microsoft

**Objašnjenje:**

*React je JavaScript biblioteka koju je kreirala kompanija Facebook, 2013. godine.*

## Kako se React integriše u projekat?

React je biblioteka koja programeru ostavlja veliku slobodu kada je u pitanju način njenog korišćenja u nekom projektu. Tako je React moguće uključiti u postojeći projekat samo parcijalno, i to na onim mestima (stranicama) unutar kojih se takva potreba javi. Sa druge strane, jednu JavaScript aplikaciju je i u potpunosti moguće kreirati korišćenjem React biblioteke. Projektni zahtevi delimično diktiraju i način na koji se React integriše u neki projekat. Integraciju je moguće obaviti na dva osnovna načina:

- ručnim uključivanjem JavaScript fajlova koji su neophodni za funkcionisanje Reacta ili
- korišćenjem *Create React App* alata za integraciju.

I ovde je situacija identična kao i prilikom integracije Vue biblioteke. React je moguće integrisati u neki HTML dokument ručnim uključivanjem nekoliko JavaScript biblioteka ili korišćenjem alata koji je specijalno namenjen za kreiranje i rukovanje React projektima. U ovoj lekciji će biti ilustrovan postupak ručne integracije, dok će korišćenje *Create React App* alata biti predmet naredne lekcije.

## Ručna integracija React biblioteke

Ručna integracija React biblioteke podrazumeva uključivanje nekoliko JavaScript fajlova korišćenjem `script` elemenata:

```
      <script src="https://unpkg.com/react@16/umd/react.development.js"
crossorigin></script>
      <script src="https://unpkg.com/react-dom@16/umd/react-
dom.development.js" crossorigin></script>
```

Na ovaj način obavlja se uključivanje dva specijalna JavaScript fajla unutar kojih se nalaze osnovne React funkcionalnosti:

- **React** – biblioteka sa osnovnim funkcionalnostima koje su, nakon integracije, u dokumentu dostupne uz korišćenje globalnog objekta sa nazivom `React i`
- **ReactDOM** – skup React funkcionalnosti za rukovanje objektnim modelom dokumenta (DOM-om); ovakav skup funkcionalnosti je, nakon integracije, dostupan preko globalnog objekta sa nazivom `ReactDOM`.

**Razvojne i produkcione biblioteke**

Možete videti da `.js` fajlovi koji se uključuju na upravo prikazan način u svom nazivu imaju sufiks *development*. Reč je o fajlovima koje je potrebno koristiti tokom razvoja. Pre nego što se projekat objavi, potrebno je preći na produkcione fajlove, postavljanjem reči *production* umesto *development*.

Mogućnost ručne integracije Reacta iskoristićemo kako bismo u narednim redovima kreirali prve primere. Biće to vrlo jednostavni primeri, koji će vam pokazati na koji način je React moguće parcijalno integrisati u projekat koji već postoji.

## Prvi React primer

Prvi primer podrazumevaće ispis jednostavne poruke unutar HTML stranice korišćenjem React biblioteke. Kompletan kod primera izgleda ovako:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>React - Simple integration</title>
</head>

<body>
    <div id="container"></div>
</body>

<script src="https://unpkg.com/react@16/umd/react.development.js"
crossorigin></script>
<script src="https://unpkg.com/react-dom@16/umd/react-
dom.development.js" crossorigin></script>

<script>

    class App extends React.Component {
        render() {
            return React.createElement('div', null, 'Hello World');
        }
    }

    ReactDOM.render(React.createElement(App),
document.getElementById('container'))
</script>

</html>
```

U prikazanom primeru potrebno je da primetite nekoliko važnih činjenica:

- unutar `body` elementa je postavljen jedan `div` element sa `id`-jem `container`; to je element unutar koga će React da upiše dinamički generisan sadržaj,
- korišćenjem dva nešto ranije prikazana `script` elementa obavljeno je uključivanje dva osnovna React `.js` fajla – *React* i *ReactDOM,*
- unutar `script` elementa se obavlja dinamičko generisanje jednog `div` elementa sa tekstom *Hello World* i takav element se umeće unutar `div` elementa sa `id`-jem `container`.

U prikazanom primeru je prvi put kreirana jedna React komponenta. To je učinjeno kreiranjem klase (`App`) koja nasleđuje klasu `React.Component`. `React.Component` je klasa koja je sastavni deo React biblioteke koja je korišćenjem prvog `script` elementa uključena u HTML dokument.

Naša prva React komponenta unutar sebe poseduje samo jednu metodu – `render()`. Unutar nje je definisan kod za kreiranje prezentacije komponente.

**React.Component klasa**

`React.Component` je osnovna (bazna) klasa za izgradnju React komponenata. Unutar klase koja nasleđuje ovakvu klasu neophodno je definisati jednu metodu – `render()`. Metoda `render()` koristi se za definisanje prezentacione strukture jedne React komponente.

Unutar metode `render()` obavlja se dinamičko kreiranje DOM elementa, kojim će komponenta da bude predstavljena unutar HTML stranice. To se postiže pozivanjem još jedne specifične metode globalnog `React` objekta. Reč je o metodi `createElement()`.

**Metoda React.createElement()**

Metoda za kreiranje React elemenata. Poseduje sledeću sintaksu:

```
React.createElement(
  type,
  [props],
  [...children]
)
```

Metoda `createElement()` može da prihvati tri parametra, pri čemu je samo prvi obavezan:

- `type` – tip elementa koji se kreira,
- `props` – svojstva elementa koji se kreira (svojstva DOM objekata, na primer `className` ili `style`),
- `children` – sadržaj elementa, bilo da je reč o tekstu ili elementima potomcima.

U primeru je metoda `createElement()` iskorišćena za izgradnju jednog `div` elementa sa tekstom *Hello World*.

Na kraju, kreirana React komponenta dodata je unutar DOM-a, korišćenjem metode `ReactDOM.render()`.

**Metoda ReactDOM.render()**

Za dodavanje nekog elementa ili komponente unutar objektne strukture dokumenta koristi se metoda `render()` globalnog objekta `ReactDOM`. Metoda `ReactDOM.render()` poseduje sledeću sintaksu:

```
ReactDOM.render(element, container[, callback])
```

Parametri metode `ReactDOM.render()` su sledeći:

- `element` – element koji je potrebno dodati u DOM strukturu,
- `container` – DOM element unutar koga će novi element da bude dodat,
- `callback` – opcioni parametar kojim je moguće definisati callback funkciju, koja će se aktivirati kada se element doda unutar DOM-a ili kada se obavi njegovo ažuriranje.

U prikazanom primeru je DOM element konstruisan na osnovu React komponente, pozivanjem metode `React.createElement()`. Koreni element, unutar koga će biti postavljen kreirani element, dobijen je korišćenjem klasičnog JavaScript pristupa koji podrazumeva korišćenje metode `document.getElementById()`.

Na kraju, osnovna struktura prezentacionih elemenata unutar našeg HTML dokumenta, nakon izvršavanja JavaScript koda, izgledaće ovako:

```
<div id="container">
      <div>Hello World</div>
</div>
```

## Korišćenje Reacta u kombinaciji sa JSX-om

Prethodni primer ilustrovao je osnovni pristup za korišćenje Reacta. Mi ćemo u nastavku u priču o biblioteci React postepeno da uvodimo nove pojmove. Prvi takav pojam odnosi se na mogućnost korišćenja jednog posebnog jezika koji se zove JSX.

**JSX**

JSX je skraćenica za JavaScript XML, što je zapravo jezik koji proširuje sintaksu JavaScript jezika, uvođenjem mogućnosti definisanja XML, odnosno HTML koda, direktno unutar JavaScript izraza. Evo kako može izgledati jedan izraz formulisan korišćenjem JSX sintakse:

```
const element = <h1>Hello World!</h1>;
```

Na prvi pogled, prikazani kod izgleda kao klasična deklaracija i inicijalizacija jedne JavaScript konstante. Ipak, ukoliko bolje pogledate, moći ćete da vidite da je vrednost konstante `element` definisana bez upotrebe bilo kakvih navodnika. Ovo je klasičan primer jedne od osnovnih mogućnosti JSX jezika.

U realnim uslovima, JSX je nezaobilazni sastojak Reacta, koji se najčešće koristi prilikom definisanja šablona komponenata. Ipak, React ni na koji način nas ne obavezuje da koristimo JSX. Tako je korišćenje JSX-a potpuno neobavezno. Ipak, kao što je rečeno, u praksi se JSX koristi gotovo uvek, pa ćemo ga i mi koristiti u nastavku priče o Reactu.

Potpuno je jasno da web pregledači ne mogu da razumeju kod koji se napiše uz poštovanje pravila JSX jezika, s obzirom na to da je reč o jeziku koji nije ni JavaScript, a ni HTML, već njihova kombinacija. Stoga je preduslov za uspešno izvršavanje JSX koda – korišćenje jedne JavaScript biblioteke o kojoj je već bilo reči u dosadašnjem toku ovoga kursa. Reč je o biblioteci Babel. Biblioteka Babel poseduje dve osnovne namene:

- pretvara ES6+ kod u kod koji zadovoljava starije specifikacije jezika (ES5 i niže) i
- pretvara JSX kod u čist JavaScript.

Kada se integracija biblioteke React obavlja ručno, kao što je to učinjeno nešto ranije u ovoj lekciji, isto je potrebno uraditi i sa Babel bibliotekom. Njena integracija izgleda ovako:

```
      <script src="https://unpkg.com/babel-
standalone@6/babel.min.js"></script>
```

Nakon integracije, neophodno je napraviti izmenu i na `script` elementu unutar koga se piše JavaScript kod. Naime, kako bi Babel mogao da zna da je potrebno da obavi transformaciju nekog koda, `script` element unutar koga se takav kod nalazi potrebno je dekorisati `type` atributom sa vrednošću `text/babel`:

```
      <script type="text/babel">

            ...

      </script>
```

Sada je unutar ovakvog `script` elementa moguće pisati kod koji će iskoristiti blagodeti JSX sintakse. Već viđena logika za kreiranje React komponente, sada će biti transformisana na sledeći način:

```
<script type="text/babel">

    class App extends React.Component {
        render() {
            return <div>Hello World</div>
        }
    }

    ReactDOM.render(<App />, document.getElementById('container'));

</script>
```

Logika za formiranje prvog React primera sada je uprošćena korišćenjem JSX koda. JSX je iskorišćen na dva mesta:

- prilikom definisanja povratne vrednosti `render()` metode unutar klase `App i`
- prilikom prosleđivanja prvog parametra metodi `ReactDOM.render()`.

Na osnovu prikazanog koda, možete da vidite koliko JSX olakšava korišćenje Reacta. Naime, u primeru su korišćenjem JSX-a efikasno zamenjena dva poziva `React.createElement()` metode.

**Napomena**

Babel prevodi JSX kod direktno u pozive `React.createElement()` metode.

Ukoliko nakon obavljenih izmena na kodu naše prve React aplikacije, takvu aplikaciju otvorite unutar web pregledača, moći ćete da vidite identičan efekat kao i u prethodnom primeru, koji je realizovan bez JSX jezika. Ipak, potrebno je da imate na umu da je Babel transformacija u prikazanom primeru obavljena direktno unutar web pregledača. Takvo nešto se u produkciji nikada ne praktikuje, već se proces prevođenja JSX koda u JavaScript obavlja pre nego što se projekat objavi. Takođe, i ručna integracija Reacta, koja je ilustrovana u ovoj lekciji, može se koristiti samo ukoliko se još upoznajete sa Reactom. Razvoj realnih React aplikacija, baš kao što je to bio slučaju i sa Vue bibliotekom, podrazumeva korišćenje jednog posebnog alata, koji posredstvom izvršnog okruženja Node.js i npm menadžera paketa, znatno pojednostavljuje i ubrzava razvoj. Takav alat biće predmet naredne lekcije.

## Rezime

- React je JavaScript biblioteka koja se koristi za kreiranje prezentacionog sloja web aplikacija.
- React je biblioteka koju aktivno koristi kompanija Facebook za razvoj društvenih mreža Facebook i Instagram.
- React je moguće integrisati u projekat na dva načina: ručnim uključivanjem nekoliko JavaScript fajlova ili korišćenjem *Create React App* alata.
- Dva osnovna React globalna objekta su `React` i `ReactDOM`.
- Centralne figure Reacta su komponente.
- React komponentu je moguće kreirati kao klasu koja nasleđuje klasu `React.Component`.
- Klasa koja predstavlja komponentu unutar sebe mora imati jednu metodu – `render()`.
- Metoda `render()` koristi se za definisanje prezentacione strukture jedne React komponente.
- Metoda `React.createElement()` koristi se za kreiranje React elemenata.
- Dodavanje nekog elementa ili komponente unutar objektne strukture dokumenta postiže se korišćenjem metode `ReactDOM.render()`.
- Prilikom korišćenja Reacta, moguće je koristiti jedan poseban jezik – JSX.
- JSX je jezik koji omogućava kombinovanje jezika JavaScript i HTML.
- Web pregledači ne mogu da razumeju JSX kod, te je tako prilikom njegovog korišćenja neophodno obaviti konverziju JSX koda u JavaScript pre nego što se takav kod isporuči web pregledaču; takav posao obavlja biblioteka Babel.
- Babel prevodi JSX kod direktno u pozive `React.createElement()` metode.