# React.js okvir

**Stojaković, Bruno**

SVEUČILIŠTE U ZAGREBU
FILOZOFSKI FAKULTET
ODSJEK ZA INFORMACIJSKE I KOMUNIKACIJSKE ZNANOSTI
Ak. god. 2022./ 2023.

Bruno Stojaković

# React.js okvir

Završni rad

Mentor: dr.sc. Kristina Kocijan, izv. prof.

Zagreb 2023.

# Izjava o akademskoj čestitosti

Izjavljujem i svojim potpisom potvrđujem da je ovaj rad rezultat mog vlastitog rada koji se temelji na istraživanjima te objavljenoj i citiranoj literaturi. Izjavljujem da nijedan dio rada nije napisan na nedozvoljen način, odnosno da je prepisan iz necitiranog rada, te da nijedan dio rada ne krši bilo čija autorska prava. Također izjavljujem da nijedan dio rada nije korišten za bilo koji drugi rad u bilo kojoj drugoj visokoškolskoj, znanstvenoj ili obrazovnoj ustanovi.

_____

(potpis)

# Sadržaj

# 1. Uvod

Ovaj završni rad analizira prednosti React.js okvira (u daljnjem tekstu samo React) naspram drugih tehnologija za razvoj internetskih stranica - Vue.js i Ember.js. Razlog odabira Reacta za temu ovog rada temelji se na tome da je on trenutno među najzastupljenijim alatima za izradu internetskih stranica i mobilnih aplikacija (Most used web frameworks among developers worldwide, as of 2022, 2022). Konstantno se radi na poboljšanju i pojednostavljivanju njegovog rada dok se pritom trudi zadržati sve one značajke po kojima se od samog početka razlikuje od svih drugih sličnih alata.

Za bolje razumijevanje nastanka Reacta potrebno je poznavati i tehnologije koje su mu prethodile. Težnja pružanju što boljeg korisničkog iskustva i usluge krajnjem korisniku povukla je sa sobom razvoj novih i boljih tehnologija i moderniziranju starih. Od 1994. godine i početka komercijalizacije interneta pa do 2004. godine te širenja interneta kao masovnog medija, osmišljene su mnoge tehnologije i programski jezici koji su većinom u uporabi i danas u nekom obliku poput HTML-a (1993.), PHP-a (od 1994.), JavaScripta (od 1995.) i CSS-a (od 1996.) (Gilyadov, 2018) odnosno tehnologije koje su danas neizostavan dio interneta.

2002. godine osmišljen je **Ajax** (Asynchronous JavaScript and XML), set tehnika koji je omogućavao dinamičko učitavanje promjena na stranici. Time je postavljen temelj daljnjem razvoju internetskih stranica kakve poznajemo danas te je omogućena izrada jednostraničnih mrežnih aplikacija (engl. *Single Page Web Applications)* (Gilyadov, 2018). Od alata koji su vodili razvoju Reacta potrebno je spomenuti i **JQuery** (J. Resig – 2006.) koji je u potpunosti promijenio način na koji se dotad koristilo JavaScriptom s pristupom „piši manje, napravi više[1]“ te **Node.js** (R. Dahl – 2009.), programsko okruženje koje JavaScript izvodi na serveru te omogućava programiranje prednje strane (engl. *frontend programming)* i stražnje strane (engl. *backend programming)* istim jezikom odnosno JavaScriptom. 2013. godine, razvojni programeri Facebooka naišli su na problem koji se javljao zbog prevelike kompleksnosti rukovatelja događajem (engl. *event handler*[2]). Naime, budući da je tada bilo moguće imati otvoreno više od jednog prozora za čavrljanje, bilo je teško pratiti protok podataka te je taj specifičan problem uzrokovao beskonačnu petlju. Rješenje tog problema Facebook je pronašao

[1] Piši manje, napravi više (engl. *write less do more)* je pristup promicanja efikasnosti i optimizacije kôda na što manje potrebnih linija za izvršenje nekog zadatka.
[2] *Rukovatelj događajima (engl. event handler) je funkcija ili metoda koja se izvršava prilikom određenog događaja (klik mišem, korisnički unos, učitavanje stranice), koja definira ponašanje aplikacije.*

u **Reactu** (Hámori, 2022), odnosno ranom prototipu Reacta kojeg je izdao razvojni programer Jordan Walke koji je u relativno kratkom vremenu postao jedan od vodećih alata za izradu internetskih stranica.

U nastavku rada pojasnit će se neki od osnovnih koncepata JavaScripta koji su neophodni za razumijevanje Reacta. Ukratko će se proći kroz jezike za označavanje; HTML i CSS te njihovu ulogu pri razvoju i stvaranju mrežnih stranica. Nadalje, pobliže će se opisati sastavnice Reacta te kroz kratke primjere ilustrirati njihova konkretna uporaba. React će se zatim usporediti s drugim alatima za izradu internetskih stranica, posebice Vue.js i Ember.js, te će biti navedene njihove prednosti i mane. Naposljetku će se na primjeru elemenata koje možemo naći na stranici internet trgovine prikazati sve sastavnice koje su spomenute u radu.

# 2. JavaScript

JavaScript je jednostavan, interpretativan programski jezik koji se primarno koristi za razvoj interaktivnih mrežnih stranica te ga danas podržava i koristi većina mrežnih preglednika. Za razliku od HTML-a koji je statičan, JavaScript je dinamični jezik te omogućava interaktivnost stranice te promjenu i stvaranje HTML sadržaja u stvarnom vremenu. Osmišljen je 1995. godine te je početna verzija napisana u samo 10 dana. Za njegov razvoj i standardizaciju zadužena je organizacija ECMA[3]. U vrijeme pisanja ovog rada, React koristi ES2015 verziju JavaScripta[4] (Rauschmayer, 2022). U nastavku ovog poglavlja, detaljnije će se opisati koncepte koje smatram važnima za lakše korištenje i razumijevanje Reacta.

## 2.1. JavaScript varijable

Varijabla[5] u JavaScriptu može sadržavati bilo koju vrijednost te omogućava pohranu i manipulaciju podatcima. Kad varijablu deklariramo ona je „prazna“ te joj možemo ili dodijeliti vrijednost, ili je kasnije dodati u nekoj funkciji ili putem korisničkog unosa. Vrijednost može biti bilo koji skup podataka stavljen u kontekst. Primjerice, broj „3“ sam po sebi JavaScriptu ne znači ništa, međutim deklariramo li `var = 3` dobijemo varijablu čija je vrijednost broj 3. Da ne bi došlo do zabune prilikom pisanja kôda bitno je zapamtiti da JavaScript razlikuje velika i mala slova tako da recimo varijabla „X“ i „x“ nisu iste varijable. Razlikujemo 3 ključne riječi pomoću kojih možemo deklarirati veličine u JavaScriptu a to su:

- `var`
- `let`
- `const.`

`Var` je bila prva ključna riječ kojom se deklarirala varijabla u JavaScriptu te se koristi u starijim preglednicima koji još nemaju podršku za `let` i `const`. ES6 preporuka je da se varijable deklariraju ključnom riječi `let`, iako se ključna riječ `var` nikad neće izbaciti iz uporabe. Na slici 1 prikazan je primjer varijable definirane ključnom riječi `var`. Varijable `x` i `y`

[3] Dodatne informacije o samoj organizaciji možete pronaći na mrežnoj stranici: http://www.ecma.ch/).
[4] ES6(ECMAScript 2015) šesta je verzija standarda za JavaScript
[5] Varijabla: *inform.* u programiranju, mjesto pohrane sposobno da primi podatak koji se tijekom izvođenja programa može mijenjati (hjp.znanje.hr, 2023)

(slika 1; kodni redak 139-140) definirane su direktnim unosom, dok je varijabla **z** (slika 1; kodni redak 141) rezultat operacije zbrajanja varijabli **x** i **y**.

```
138     // Deklariramo varijable ključnom riječi Var
139     var x = 1;
140     var y = 2;
141     var z = x + y;
142
143     //ispisujemo vrijednost varijable z
144     document.getElementById('root').innerHTML =
145         "vrijednost varijable z je 1 +" + z;
```

*Slika 1: Primjer deklariranja varijable ključnom riječi* ***var***

**Let** koristimo kada očekujemo da će se vrijednost varijable promijeniti odnosno ona dopušta da se vrijednost varijable mijenja bez izazivanja greške u dosegu funkcije[6]. Ova je mogućnost dodana u JavaScript tek u ES6 (2015) inačici te je s njom riješen sustav dosega funkcije koji je bio nejasan te izazivao većinu pogreški.

```
124     var x = 5;
125     // varijabla x = 5
126     {
127         let x = 3;
128         // varijabla x = 3
129     }
130
131     // varijabla x = 5
```

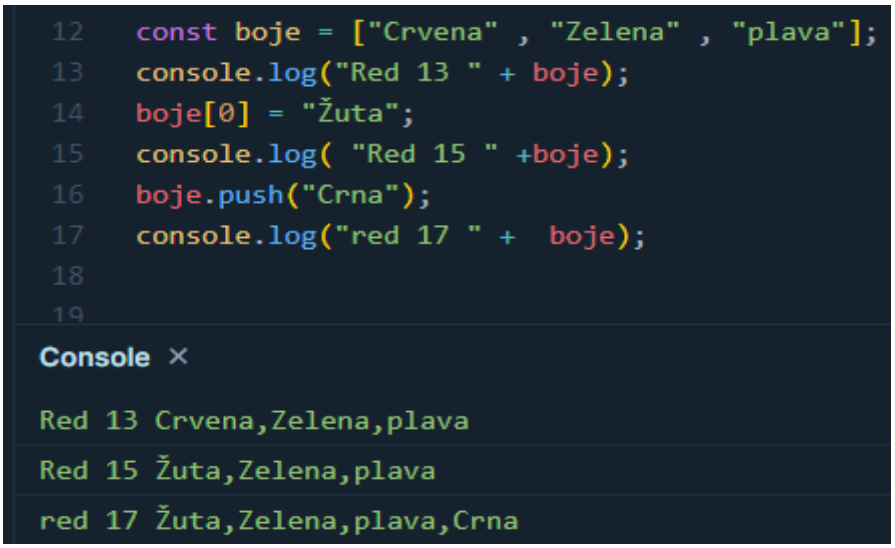
*Slika 2: Primjer ispravne uporabe ključne riječi* ***let***

Iz slike 2 vidljivo je da uporabom ključne riječi **let** možemo promijeniti vrijednost varijable unutar **dosega funkcije** (sadržaj unutar vitičastih zagrada **{ }**). Time se smanjuje potreba za ponovnim deklariranjem varijable ključnom riječi **var** te je kôd jednostavniji i izgleda preglednije.

**Const**[7]je ključna riječ koja je uvedena paralelno s **let** od koje se razlikuje u sljedećem – veličine definirane ključnom riječi **const** moraju biti deklarirane prije uporabe i ne mogu biti ponovno deklarirane te im se vrijednost ne može promijeniti. Preporuka je da se umjesto

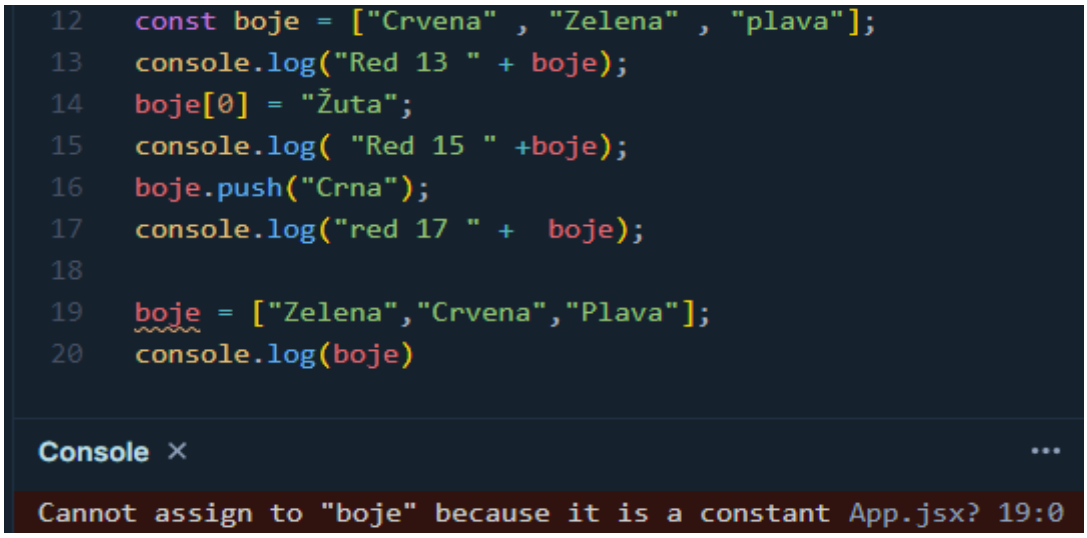
[6] **Doseg** definiramo kao granice unutar kojih možemo pozvati određenu funkciju ili varijablu. Detaljnije objašnjeno u poglavlju 2.3
[7] Konstanta (engl. *Constant*) - funkcija koja poprima konstantnu vrijednost za sve elemente domene (hjp.znanje.hr, 2023)

ključnom riječi **var**, varijable deklariraju ključnom riječi **const**, osim kada znamo da će se vrijednost varijable promijeniti (Rauschmayer, 2022).




*Slika 3: Prikaz pravilne uporabe ključne riječi **const***

Na Slici 3 u kodnom retku 12 prikazano je deklariranje konstante „**boje**“ kao niz (engl. *array)* s 3 vrijednosti: **Crvena**, **Zelena**, **plava**, što je vidljivo u konzoli kod ispisa te konstante funkcijom **console.log**. Vidljivo je dodavanje elementa na nultu poziciju na dva načina. U kodnom retku 14 mijenja se nulti element niza pristupajući mu njegovim indeksom („**boje[0]**“) što se može uočiti u drugom retku ispisa u konzoli. Koristeći metodu „**push**“ u kodnom retku 16 umeće se element „**crna**“ na kraj niza što je vidljivo u zadnjem retku ispisa u konzoli. Na slici 4 vidljivo je da prilikom pokušaja preraspodijele vrijednosti u kodnom retku 19 konzola ispisuje grešku.




*Slika 4: Prikaz greške u korištenju ključne riječi **const***

## 2.2. JavaScript funkcije

Funkciju se može objasniti kao potprogram ili blok grupiranih naredbi u JavaScriptu (slika 4; kodni redak 21-24) koji se izvršava pozivanjem imena funkcije (slika 4; kodni redak 26). Funkcije u Reactu su u suštini komponente, odnosno modularni i višestruko iskoristivi blokovi koda koji samostalno funkcioniraju. Cilj funkcije je skraćivanje kôda i pojednostavljivanje programa pozivanjem iste funkcije više puta. Tako se eliminira pisanje istog kôda više puta i smanjujemo mogućnost greške sintakse. Funkciju definiramo pomoću ključne riječi **`function`** (Functions, n.d.).

```
20    // imenujemo funkciju
21    function pozdrav() {
22        document.getElementById('root').innerHTML =
23            "Pozdrav!";
24    }
25    // pozivamo funkciju
26    pozdrav()
```

*Slika 5: Prikaz funkcije*

Funkcije se sastoje od tri dijela:

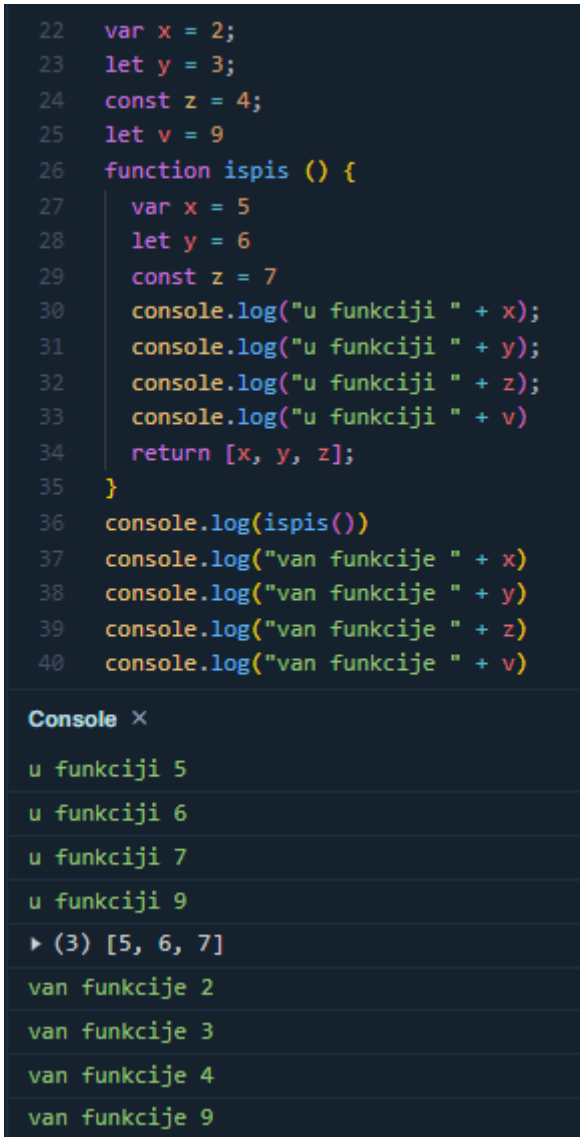
- ime funkcije
- argumenti funkcije
- izjave.

Ime funkcije je u primjeru na slici 4 **`function pozdrav().`** Argument, da postoji u ovoj funkciji, nalazio bi se u zagradi. Argument može biti bilo kakva dodatna vrijednost koju funkcija uzima u obzir. U primjeru **`function greeting(x, y)`** argumenti su „**`(x, y)`“.** Od ta tri dijela funkcije, samo ime je opcionalno dok su ostali dijelovi obavezni. Ako funkciji ne navedemo ime ona postaje "anonimna funkcija".

## 2.3. JavaScript doseg

**Doseg** se defnira kao granice unutar kojih se može pozvati određena funkcija ili varijabla. **Dosezi** također mogu biti hijerarhijski povezani tako da nadređena komponenta može pristupiti i davati upute podređenoj komponenti, ali ne i obratno. **Doseg** je najlakše opisati na primjeru funkcije. Definira li se funkcija, a unutar nje i varijabla, ta varijabla će biti dostupna samo unutar te funkcije što optimizira performanse aplikacije.

Postoje tri vrste dosega: globalan doseg, doseg funkcije te od ES6 (2015) doseg bloka. Doseg bloka koristi se isključivo s ključnim riječima **let** i **const**, odnosno varijable deklarirane unutar jednog bloka omeđenog vitičastim zagradama „{}“ mogu biti pozvane samo unutar tog dosega (Scope, n.d.). Lokalne varijable su one koje se nalaze u dosegu funkcije (Slika 6; kodni redak 27-29). Takve varijable su prepoznate isključivo unutar svoje funkcije te se one stvaraju pri pokretanju funkcije i brišu kada funkcija završi. Time se omogućava korištenje istog imena varijable izvan te funkcije (js_scope.asp, n.d.). Varijable deklarirane u kodnom retku 22-24 imaju globalan doseg. Na slici 6 u kodnom retku 25 vidljiv je primjer varijable „let v = 9“ koja je dostupna unutar i izvan funkcije te joj je vrijednost ista u oba slučaja.




*Slika 6: Primjer dosega ključne riječi **var***

## 2.4.JavaScript uzdizanje (engl. *hoisting*)

Uzdizanje je proces izvršavanja funkcija i varijabli prije ostalih dijelova kôda. JavaScript nam omogućava pozivanje funkcije prije nego što su napisane u kôdu. Točnije, funkcije i varijable uzdignute su na vrh svog dosega prije nego se kôd pokrene.

Iako se upotreba uzdizanja ne preporučuje, najveća prednost je što omogućava referenciranje funkcija, klasa i varijabli prije njihove deklaracije. Po pitanju uzdizanja možemo dobiti dva različita problema: „**`undefined`**“ i „**`ReferenceError: variable is not defined`**“. Problem „**`undefined`**“ (slika 7; kodni redak 40) javlja se kada varijabli nije dodijeljena vrijednost, dok se „**`ReferenceError`**“ (slika 7; kodni redak 51) javlja kada smo pozvali varijablu koja ne postoji ili do nje nije moguće doći.

```
37  // varijabla x definirana ključnom riječi let
38  let x;
39  //provjera varijable funkcijom console.log()
40  console.log(x);
41  // undefined
42  // definiramo funkciju
43  function broj() {
44      //ključnom riječi const definiramo varijable
45      const br1 = 2,
46          br2 = 3;
47      return br1 + br2;
48      // rezultat funkcije je 5
49  }
50  console.log(br1);
51  //ReferenceError br1 is not defined.
```

*Slika 7: Primjer pogreški u JavaScriptu*

## 2.5. Klase i Objekti

Klase su u JavaScript uvedene standardom ES6 (2015). One su predlošci za izradu objekata te ih se obavezno mora koristiti u paru s metodom **`constructor`**. Metoda konstruktor način je izrade objekata u JavaScriptu. Na slici 8 možemo vidjeti primjer konstruktor metode. Prvo u kodnom retku 55 definiramo klasu kôdom **`class car {}`**, zatim definiramo konstruktor te mu kao argumente zadajemo **`(name, brand).`** Naposljetku kao vrijednosti zadajemo prethodno zadane varijable. Ako konstruktor nije definiran, JavaScript će sam dodati praznu konstruktor metodu (Au-Yeung, n.d.).

```
54    // definiramo objekt
55    class car {
56        //definiramo konstruktor s
57        //argumentima "name" i "brand"
58        constructor(name, brand) {
59            this.name = name;
60            this.brand = brand;
61        }
62    }
```

*Slika 8: Primjer klase s dvije metode*

Klase ne mogu biti korištene prije nego što ih deklariramo, odnosno nisu uzdignute na vrh poput funkcija. Također ih se, poput funkcija, može dodijeliti varijablama. Klase se još može i definirati rezerviranom riječi **`class`**, a njihovo imenovanje je opcionalno. U slučaju da ih se ne imenuje, ime klase može se saznati koristeći „**`console.log()`**“ funkciju što se može uočiti na slici 9 u kodnom retku 76.

```
66    // čija je vrijednost objekt bez imena
67    let torba = class {
68        // definiramo konstruktor s
69        //argumentima boja i marka
70        constructor(boja, marka) {
71            this.boja = boja;
72            this.marka = marka;
73        }
74    }
75    // koristeći console.log() saznajemo ime objekta
76    console.log(torba.name)
```

*Slika 9: Primjer neimenovane klase*

Konstruktor je metoda koja se koristi za stvaranje objekta u klasi te se u klasi može nalaziti samo jedan konstruktor. Ako je definirano više od jednog konstruktora javit će se greška sintakse. Ključne riječi „**`super`**“ i „**`extends`**“ koriste se uz konstruktor kako bi se dobila svojstva i metode: **`super`** koristimo da bismo dobili svojstva i metode roditelja, a **`extends`** pri izradi potomka druge klase.

```
class computer {
    constructor (type) {
        this.computerName = type;
    } pc() {
        return 'Imam ' + this.computerName;
    }
}
class model extends computer {
    constructor(type, mod) {
        super(type);
        this.model = mod;
    }
    show(){
        return this.ps() + ' marke ' + this.model;
    }
}
mypc = new model('laptop' , 'lenovo');
document.getElementById('root').innerHTML = mypc.show();
```

*Slika 10: Primjer uporabe ključnih riječi "**extends**" i "**super**"*

Na slici 10 prikazano je deklariranje klase „**computer**“ kojoj je zadano svojstvo „**type**“ te izrada prazne varijable za povratnu vrijednost tipa varijable. Zatim deklaracija klase „**model**“ za koju se ključnom riječi „**extends**“ naznačuje da je potomak klase „**computer**“. Potom joj se zadaje svojstvo „**mod**“. Nadalje, koristeći ključnu riječ „**super**“ preuzima se svojstvo „**type**“ od roditelja. Deklarira se nova varijablu s argumentima potrebnima za stvaranje objekta tese ispisuje rezultat kôda na ekranu.

```
const car =  {
    marka: 'Volkswagen',
    model: 'Golf 2',
    boja: 'crna',
    tezina: ' 800 kg',
    brojVrata:'5',
};
```

*Slika 11: Primjer objekta*

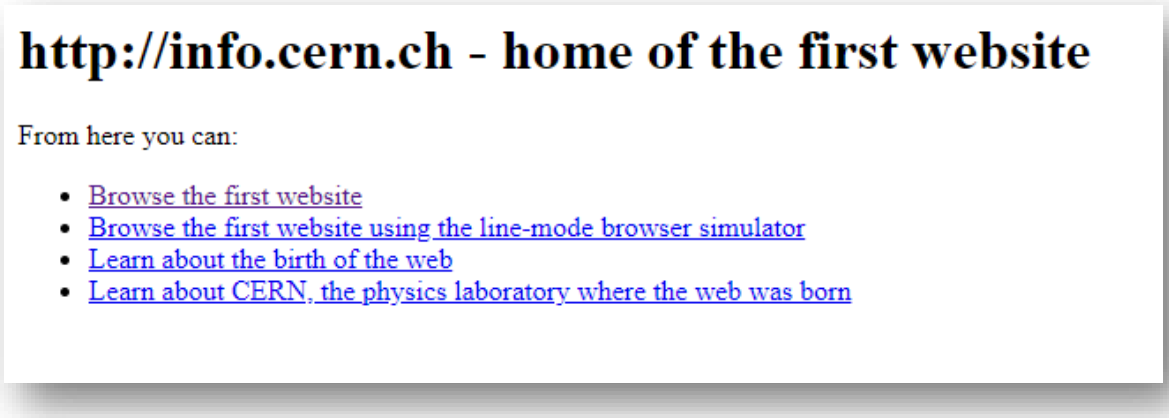
Objekti su spremnici za više svojstava. Svojstva se mogu definirati kao veza između imena (ključa) i vrijednosti. U slučaju da je vrijednost svojstva funkcija, to svojstvo je znano kao metoda. Objekte je najlakše usporedi ih li se sa stvarima u stvarnom svijetu. Uzme li se za primjer auto, njegova svojstva mogu biti marka, model, boja, broj vrata i slično, ta svojstva definiraju objekt. Na slici 11 može se vidjeti kako su ta svojstva povezana odnosno kako svaka sadrži ime ili ključ (npr. „**marka**“) i vrijednost (npr. „**Volkswagen**“). Također je bitno

napomenuti da objekt opisujemo unutar vitičastih zagrada te da svojstva odvajamo zarezom (Working with objects, n.d.).

# 3. HTML i CSS

Da bi se mogla razumjeti uloga elemenata i komponenti u Reactu, bitno je prvo razumijeti osnovne gradivne elemente interneta - HTML (*HyperText Markup Language*). HTML nema dinamičnih elemenata te mu trebaju dodatni jezici poput CSS-a i JavaScripta za vizualno oblikovanje i funkcionalnost. HTML se sastoji od elemenata koje navodimo unutar oznaka „**<**“ i „**>**“. Za razliku od JavaScripta ne razlikuje velika i mala slova tako da npr. možemo napisati **<head>**, **<Head>** ili **<HEAD>** i sva tri načina su ispravna. Preporuka w3c-a[8] je da se pri imenovanju koriste mala slova.

Najraniji primjeri internetskih stranica sačinjavali su se od čistog HTML-a. Korisnici bi se kretali po stranicama klikom na HTML veze odnosno hiperveze. Primjer kako jedna takva stranica izgleda može se vidjeti na slici 12. Dodatkom CSS-a (*Cascading Style Sheets*) omogućeno je stiliziranje elemenata time odvajajući sadržaj stranice od načina prikaza podataka.




*Slika 12: Primjer stranice s čistim HTML-om*

Da bi se napravila promjena bez ponovnog učitavanja datoteka, uz HTML i CSS, koristi se JavaScript koji omogućava promjene u stvarnom vremenu koristeći DOM[9] manipulaciju, obradu događaja u stvarnom vremenu i asinkrono programiranje[10] (JavaScript_basics, 2023).

[8] WWW Consortium, više o samoj organizaciji možete pronaći na linku: https://www.w3.org
[9] DOM(engl. Document Object Model) je programsko sučelje za mrežne dokumente odnosno prikaz strukture, sadržaja i elemenata (Document_Object_Model/Introduction , 2023)
[10] Asinkrono programiranje omogućava izvršavanje potencijalno dugoročnih zadataka bez blokiranja izvršavanja drugog kôda ( Introducing asynchronous JavaScript, n.d.)

CSS se koristi za stiliziranje i prezentaciju dokumenata napisanih HTML-om te se njime opisuje kako će elementi biti prikazani u pregledniku. Trenutna verzija je CSS3. Postoje 3 načina na koji možemo koristiti CSS:

- unutrašnji
- vanjski
- umetnuti.

```
2   <html>
3       <head>
4           <link rel = 'stylesheet' href='stilovi.css'>
5           <style>
6               p2 {
7                   color: blue;
8               }
9
10          </style>
11      </head>
12      <body>
13
14          <p style='color: red'> Prvi primjer</p>
15
16              <p2> Drugi primjer</p2>
17
18                  <p3> Treći primjer</p3>
19      </body>
20  </html>
```

*Slika 13: Primjeri uporabe CSS-a*

Na slici 13 mogu se vidjeti načini uporabe CSS-a. U 4. kodnom retku ovaj se dokument povezuje s vanjskim CSS dokumentom u kojemu se nalazi kôd za promjenu elementa **<p3>**. U 5. kodnom retku u odjeljku **<head>** možemo vidjeti element **<style>** u kojemu pišemo kôd za dodavanje stila elementu **<p2>** koji zovemo unutrašnji CSS. U kodnom retku 14. vidljiv je primjer umetnutog CSS-a dodavanjem atributa „**<style ='color: red'>**“. Umetnuti CSS koristi se da za stiliziranje jednog HTML elementa. Unutrašnji se koristi za stiliziranje jedne HTML stranice, a vanjski za stiliziranje nekoliko HTML stranica (html_css, n.d.). Valjalo bi naglasiti da „Cascading“ u imenu CSS znači to da se stil primijenjen roditeljskom elementu također primjenjuje na sve potomke tog elementa, osim ako nije drugačije navedeno.

# 4. React

React je pristupačna i besplatna JavaScript knjižnica za izradu korisničkih sučelja i jednostraničnih aplikacija. Piše se JavaScriptom te se prvenstveno sastoji od elemenata i komponenti. **Element** je osnovna gradivna jedinica React aplikacija. Koristeći React elemente gradimo komponente koje omogućavaju dijeljenje sučelja u zasebne dijelove. **Komponente** se često poistovjećuje s funkcijama zbog mogućnosti ponovne iskoristivosti. Motiv iza stvaranja Reacta bio je pojednostavljenje izrade korisničkih sučelja korištenjem jednostavnih komponenata (Buna, 2019). Jedna od prednosti Reacta je promoviranje ponovne iskoristivosti kôda. Koristeći komponente koje samostalno funkcioniraju smanjuje se potreba za ponovnim tipkanjem istog kôda te je kôd jednostavniji i pregledniji. React manipulaciju podataka provodi u virtualnom DOM-u te ažurira stvarni prikaz procesom koji se zove usklađenje (engl. *reconciliation*) (faq-internals, n.d.). Iako se React često naziva okvirom za programiranje, on je zapravo knjižnica koja se koristi za izradu različitih sučelja. Koristi se u internetskim i u mobilnim aplikacijama te se vrlo dobro kombinira s drugim knjižnicama. U praksi se preporučuje korištenje Reacta za izradu cijele stranice, iako se nerijetko može naći samo nekoliko njegovih funkcionalnosti.

## 4.1. JSX

React nudi mogućnost korištenja JSX-a (JavaScript XML). JSX je neobavezno sintaksno proširenje koje izgledom podsjeća na HTML te se koristi pri izradi React elemenata. Iako se koristi za izradu vizualnih elemenata, JSX ima sve funkcionalnosti JavaScript kôda (introducing-jsx, n.d.).

## 4.2. Komponente i svojstva (engl. *props)*

Komponente u Reactu su zasebni višekratni dijelovi kôda čiji je rezultat izvođenja HTML sadržaj. Tvore se od elemenata, te im možemo zadati stanje (engl. *state*), vrijednosti i argumente. Stanje je objekt sličan **`props`** objektu u smislu da oboje sadržavaju informacije. Ključna razlika između **`state`** i **`props`** objekta je da **`props`** prenosi informacije između komponenti dok **`state`** prenosi informacije unutar komponente.

U praksi postoje dvije vrste komponenti – komponente klase i komponente funkcije. Prilikom tvorbe komponente od nje očekujemo da će ona upravljati korisničkim sučeljem ili manipulirati podatcima.

Da bi kôd bio jednostavniji preporuka je da manjina komponenti bude zadužena za logiku, upravljanje stanjem i prenošenje instrukcija ostalim komponentama. Instrukcije se prenose putem **`props`** objekta. (Fedosejev, 2015). Ovaj je objekt argument koji roditelj šalje potomku odnosno **`props`** je način prijenosa podataka između komponenti. Svi podatci u **`props`** objektu mogu biti dosegnuti putem **`render`** metode. **`Render`** metoda koristi se za prikazivanje elemenata u korisničkom sučelju. Bitno pravilo Reacta je da sve komponente moraju djelovati kao čiste funkcije s obzirom na svoje značajke, odnosno ako je isti unos, uvijek mora biti isti i rezultat funkcije.

Na slici 14 vidljiva je jednostavna komponenta funkcije koja će na ekranu ispisati „Pozdrav Bruno“ svaki put kada se stranica ponovno učita. Takve komponente zovemo „*stateless*“ komponente odnosno statičke komponente. Statičke komponente se renderiraju svaki put dok ne dođe do promjene u kôdu. Komponente čine sve ono što vidimo na stranici, kao što su navigacijska traka, pretraživač, gumb i slično. Osim komponenti funkcije, postoje i komponente klase kod kojih se mora uključiti **`render`** metoda u suprotnom se podatci neće obraditi, a komponenta se neće prikazati na pregledniku.

```
13  function Greetings(props) {
14      return <h2> Pozdrav {props.name} ! </h2>;
15  }
16
17  const root =
18      ReactDOM.createRoot(document.getElementById('root'));
19  root.render(<Greetings name="Bruno" />);
```

*Slika 14: Primjer komponente s jednim svojstvom*

Na slici 15 možemo vidljive su neke razlike između komponenata klase i komponenata funkcije. Komponente klase obavezno se moraju upariti sa **`React.Component`** klasom i **`render`** metodom. **`React.Component`** sadrži metode životnog ciklusa koje su potrebne za prihvaćanje **`props`** argumenata od roditelja. **`Render`** metodu uključuje se da bi se komponenta klase ispisala. Svi ostali dijelovi komponenti funkcije i klase su neobavezni (react-component,

n.d.). Sve komponente u Reactu imaju životni ciklus koji se sastoji od 3 glavne faze i jedne dodatne koja se i ne mora dogoditi:

- postavljanje
- ažuriranje
- skidanje
- upravljanje pogreškama.

```
3   class Greetings extends React.Component {
4       render() {
5           return <h1> hello {this.props.name}</h1>;
6       }
7   }
8    const root =
9        ReactDom.CreateRoot(document.getElementById('root'));
10   root.render(<Greetings name="Bruno"/>)
```

*Slika 15: Primjer komponente klase*

Životni ciklus najlakše je objasniti po fazama. Najprije se stvori komponenta. Metodom `componentWillMount` komponentu se **postavlja** na virtualni DOM te se po potrebi prilažu dodatni argumenti kao što je upravitelj događajem (engl. *event handler*) uz komponentu. Komponenta se **ažurira** pri svakoj promjeni stanja koja je definirana u prethodnoj fazi. Kada komponenta završi svoju svrhu slijedi **skidanje** te ako je komponenta ispravno napisana time završava njezin životni ciklus. Ako komponenta nije ispravno napisana te sadrži grešku, javlja se **upravljanje pogreškama**, koje pravilnim korištenjem može sadržavati korisne informacije o kôdu te i upute kako najlakše otkloniti grešku.

## 4.3. Stanje

U prethodnom poglavlju su među vrstama komponente spomenute statičke komponente koje nemaju stanje. U ovom poglavlju razmotrit će se dinamičke komponente i kako promjena stanja utječe na njih. Stanje je promjenjiv objekt ugrađen u Reactu koji sadrži podatke o komponenti. Kako se stanje mijenja tako se mijenja i ponašanje komponente odnosno njeno izvođenje. Stanje komponente može biti ažurirano rezultatom nekog događaja, serverskog odgovora ili promjenom svojstva te za promjenu stanja koristimo `setState` metodu. Ona govori Reactu da se desila promjena u stanju komponente te ih potom React ponovno izvodi ažurirajući komponentu i njene potomke.

Iako stanje i svojstva zvuče slično jer se oboje koristi za upravljanje podatcima u komponentama, zapravo su poprilično različiti. Naime, **stanja se** koriste za pohranu podataka u komponentama dok **svojstva** prenose podatke do komponenti potomaka. Stanja su promjenjiva te ih se može koristiti samo u komponentama klase, dok se svojstva kad ih se jednom uspostavi ne mogu promijeniti ali ih se može koristiti i u komponentama klase i u komponentama funkcije. Naposljetku dok se stanje ažurira upraviteljem događaja, komponenta roditelja ažurira stanje za sve potomke (reactjs-tutorial, n.d.).

Povezivanje podataka je način komunikacije između komponenti i pregleda DOM-a ili obrnuto, postoje dva načina povezivanja podataka, jednosmjerno i dvosmjerno. React za razliku od većine okvira za programiranje koristi jednosmjerno povezivanje podataka. Prednosti su mu jednostavniji način rješavanja problema koji se mogu ukazati u kôdu, bolje raspolaganje resursima te brži i efikasniji prijenos podataka. Jednosmjerno povezivanje podataka može se opisati kao metoda ažuriranja DOM-a svaki put kada se promjene podatci.

## 4.4. Virtualni DOM

Manipulacija DOM-om (engl. *Document Object Model*) je vrlo spor i težak posao jer većina JavaScript okvira ažurira DOM i više nego što bi trebalo, odnosno bespotrebno ponovno izvode operacije i učitavaju komponente koje se nisu mijenjale. Reactov virtualni DOM po izlasku je bio hvaljen kao vrlo inovativna ideja. To je koncept u kojemu odvojeno od „stvarnog“ DOM-a postoji virtualna prezentacija sadržaja.

React, procesom koji se naziva usklađenje uspoređuje i ažurira samo potrebne informacije umjesto cijele stranice. Njegove prednosti su te što se ažuriranjem samo potrebnih komponenti na stranici skraćuje vrijeme učitavanja te ga je puno lakše stvarati i mijenjati. Omogućava testiranje u produkcijskom okruženju što smanjuje mogućnost stvaranja problema nakon puštanja aplikacije u javnost. Važno je napomenuti da virtualni DOM ima sve atribute kao stvarni DOM, bez mogućnosti promjene komponenti na ekranu.

## 4.5. React i knjižnice

Kao što je ranije spomenuto, React je sam po sebi knjižnica. Kao takav iznimno je prilagodljiv po pitanju kompatibilnosti s drugim knjižnicama poput React Bootstrap, CoreUI, ili Redux. Dalje u tekstu ukratko će biti opisane knjižnice korištene u praktičnoj izvedbi ovog rada.

### *4.5.1. JQuery*

JQuery je mala JavaScript knjižnica koja se koristi u Reactu. Svrha joj je olakšavanje rukovanja događajem, pojednostavljenje HTML/DOM manipulacije te općenito lakša implementacija JavaScripta unutar aplikacije „piši manje, napravi više“ pristupom. Vrlo je fleksibilna i jednostavna za korištenje te ima podršku za sve preglednike (jquery_intro_asp, 2023).

### *4.5.2. Popper.JS*

Popper.JS je JavaScript knjižnica čija je svrha lakše stiliziranje elemenata unutar stranice. Jednostavniji je od čistog CSS-a. Temelji se na „*popper positioning engine*“ algoritmu koji pri izračunu optimalne pozicije elementa koristi informacije o veličini, položaju, prostor između elemenata i slične parametre. Popper je jako dobra i prilagodljiva knjižnica koja je odličan temelj za druge knjižnice poput Bootstrap.css-a (why-popper, 2023).

### *4.5.3. Font-awesome.CSS*

Font-awesome.css je visoko kompatibilna knjižnica s velikim izborom ikona koja uvelike olakšava njihovu uporabu u aplikaciji. Omogućava skalabilnost aplikacije koristeći vektorske ikone čija kvaliteta ostaje ista neovisno o povećanju rezolucije (docs, 2023).

## 4.6. React i okviri

Kao i s knjižnicama, React je osmišljen da bude kompatibilan i sa drugim okvirima za razvoj korisničkog sučelja. Tako se rad s Reactom može pojednostaviti koristeći okvir koji sadrži već

gotove komponente za izradu korisničkih sučelja primjerice Bootstrap.CSS. No React isto tako uz sitne preinake može potpuno normalno funkcionirati i sa drugim alatima za izradu mrežnih aplikacija kao što je Vue. U ovom radu korišten je Bootstrap.CSS okvir. Uz pomoć njega u kombinaciji s Reactom moguće je u kratkom vremenu napraviti funkcionalno i dinamično sučelje. Prvenstveno je namijenjen izradi mobilnih sučelja, međutim podržava skalabilnost što omogućava da stranica izgleda isto na svim platformama i preglednicima (css, 2023).

# 5. Usporedba React-a s Vue i Ember okvirima za programiranje internetskih aplikacija

Vue se u posljednjih nekoliko godina prometnuo u jedan od najpopularnijih okvira za izradu jednostraničnih mrežnih aplikacija. Kao i React, ima arhitekturu baziranu na komponentama te se izvrsno prilagođava izradi kompleksnih korisničkih sučelja.

U usporedbi s prethodno spomenutim okvirima Ember je manje popularan s manjom bazom korisnika. Neovisno o tome ima veliku zajednicu razvojnih programera, redovno je ažuriran te ima iznimno veliku bazu znanja i dokumentacije.

## 5.1. Vue.js

Vue.js osmislio je Evan You 2014. godine. Potakla ga je ideja za jednostavnim okvirom za izradu mrežnih aplikacija (what-is-vue, 2023). Za razliku od drugih sličnih okvira, nije potrebna nikakva dodatna priprema za početak kodiranja. Vue ima integrirano dvostrano povezivanje podataka, što znači da će se promjena vrijednosti u jednom dijelu aplikacije prikazati i u drugom dijelu što uvelike pojednostavljuje način pisanja koda.

Kao i React koristi Virtualni DOM prilikom učitavanja sadržaja te je baziran na komponentama što olakšava organizaciju i ponovno korištenje koda. Osim što je odličan alat za izradu kompletne aplikacije, Vue se također može koristiti za poboljšanje samo određenih dijelova projekta.

## 5.2. Ember.js

Ember je JavaScript okvir za izradu mrežnih stranica koji se razlikuje od prethodno dva spomenuta alata time što ne koristi komponentnu strukturu nego MVC[11] arhitekturu. Kako bi mogli upravljati DOM-om, razvojni programeri Embera kreirali su Glimmer – virtualni stroj za obradu virtualnog DOM-a koji sinkronizira stvarni DOM s virtualnim te tako rješava nedostatak ugrađenog virtualnog DOM-a.

[11] *Model-View-Controller* arhitektura koristi se za organizaciju kôda te ga dijeli na podatke (*model)*, prikaz (*view)* i posrednika (*controller)*

Osim što je izuzetno svestran alat za izradu korisničkog sučelja, Ember također koristi i biblioteku Ember Data koja omogućuje komunikaciju i ažuriranje podataka sa serverom.

## 5.3. Usporedna analiza

U tablici 1 nalazi se usporedba nekih od svojstava okvira za izradu internetskih aplikacija. Kriteriji kojim su odabrani Vue.js i Ember.js za usporedbu s Reactom bila je jednostavnost, originalnost kao i činjenica da su svi navedeni okviri bazirani na komponentama te promoviraju ponovnu iskoristivost kôda koja u principu podržava jedan od temeljnih principa u programiranju općenito a to je DRY (engl. *Don't Repeat Yourself*), odnosno „ne ponavljaj se“.

*Tablica 1: Prikaz svojstava okvira React.js, Vue.js i Ember.js:*

| | **React.js** | **Vue.js** | **Ember.js** |
|---|---|---|---|
| **Vrsta alata za izradu mrežnih stranica** | jednostavna knjižnica s puno značajki | fleksibilni okvir pogodan za male i velike projekte | jednostavan okvir s pomno razrađenim dodatcima |
| **Vrsta memorije i** | virtualni DOM | virtualni DOM | stvarni DOM |
| **način upravljanja podatcima** | jednostrano povezivanje podataka | reaktivno dvostrano povezivanje podataka | dvostrano povezivanje podataka |
| **Dostupnost dokumentacije za učenje** | loša dokumentacija; | najbolje razrađena dokumentacija od svih okvira | dobro razrađena dokumentacija; |
| **zajednica korisnika** | velika zajednica | brzo rastuća zajednica | nešto manja naspram drugih alata |
| **Jednostavnost instalacije i** | jednostavna instalacija; | jednostavna instalacija; | jednostavna instalacija uz minimalno korisničkog unosa; |
| **kompatibilnost s drugim alatima** | vrlo kompatibilan s drugim alatima | uz dodatna podešavanja ovisno o alatu | teža integracija s drugim bibliotekama |
| **Upravljanje podatcima ovisno o veličini projekta** | dobro upravljanje stanjem uz flux i redux proširenja | bolje upravljanje stanjem koristeći vuex | odlično upravljanje stanjem na većim projektima |

Temeljnu razliku možemo uočiti odmah u prvom retku tablice odnosno da je React JavaScript knjižnica, iako je zbog širokog spektra mogućnosti koje pruža programerima, često tretiran kao okvir za programiranje. Iz tablice također možemo zaključiti da su React i Vue.js

dosta bliski po pitanju temeljnih principa, arhitekture koja se bazira na komponentama te korištenja virtualnog DOM-a koji je zajedno s načinom prijenosa podataka zaslužan za iznimne brzine.

Ember.js se također po mnogočemu može povezati s Reactom. Između ostalog podržava pojednostavljeno testiranje i razvoj internetskih aplikacija kao i mnogo dodataka. Za upravljanje stanjem podataka React prvenstveno koristi jednostranično povezivanje podataka dok Vue.js i Ember.js koriste dvostranično povezivanje podataka. Ember.js je u smislu instalacije po mnogima bolji od Reacta jer dolazi s već ugrađenim funkcionalnostima koje u slučaju Reacta programer mora sam instalirati.

S obzirom na to da React i Vue.js imaju poprilično veliku bazu korisnika, React zaostaje u smislu dokumentacijske potpore. Npr. renderiranje na strani servera nije definirano u službenoj dokumentaciji, dok je dokumentacija ostala dva okvira vrlo dobro razrađena. U isto vrijeme prednost i izvor problema kod Reacta je podrška i preveliko oslanjanje na strane tehnologije. Primjerice za efikasan rad u Reactu treba proučiti biblioteku *React Router*, čija je svrha organizacija i upravljanje navigacijom, a za veće projekte Redux ili Flux biblioteke koje su zadužene za upravljanje stanjem. Vue.js ima ugrađen sustav upravljanja stanjem podataka Vuex, a u slučaju Embera prijenos stanja podataka se vrši preko ruta i modela što je u potpunosti drugačiji koncept.

Od ova 3 navedena okvira, svi su hvaljeni zbog svoje fleksibilnosti. Međutim u suštini je jedino React uistinu fleksibilan okvir. Razlog tome je što ostali okviri imaju svoj vlastiti način obavljanja stvari te navode korisnika da ne odstupa od takvog načina (Buna, 2019).

# 6. Internet trgovina

U ovom poglavlju oprimjerit će se sastavnice Reacta koje su teorijski obrađene u prethodnim poglavljima. Cilj je prikazati osnovne koncepte Reacta poput komponentne arhitekture, korištenja JSX-a i korištenja **`state`** i **`props`** objekta. Bit će opisane komponente poput Košarice, navigacijske trake, početne stranice te stranice koja sadrži podatke o proizvodu. Za jednostavniju izradu ove internetske trgovine uz React korištene su dodatne knjižnice poput JQuerya za lakše rukovanje događajima i Popper.js-a za pojednostavljeno pozicioniranje i stiliziranje HTML elemenata te iz razloga što s njima Bootstrap.css najbolje funkcionira. Bootstrap.css i font-awesome.css korišteni su za stiliziranje aplikacije i prvenstveno za izradu navigacijske trake.

```
1   import React, { Component } from "react";
2   import NavBar from "./NavBar.jsx";
3   import CustomersList from "./CustomersList.jsx";
4   import Cart from "./Cart.jsx";
5   import Login from "./login.jsx";
6   import NotFound from "./NotFound.jsx";
7   import Product from "./Product.jsx";
8   import Kosarica from "./kosarica.jsx";
9
10  import { Routes, Route } from "react-router-dom";
11  import { BrowserRouter } from "react-router-dom";
```

*Slika 16: Kôd Index.js datoteke*

Na slici 16 može se vidjeti kako je u aplikaciji svaka komponenta poput navigacijske trake, korisnika, stranice za prijavu, stranice proizvoda i slično napravljena zasebno te je zajedno implementirana u **`App.jsx`** datoteci. Taj sustav omogućava jednostavno kretanje kroz kôd te mogućnost da korisnik u svakom trenutku zna u kojoj se komponenti što nalazi. Također osobito olakšava pronalaženje problema u kôdu. Manji problem koji se javio u izradi ove stranice je nedostatak serverske komponente u Reactu. Za izradu baze podataka i pozadinski dio stranice korišten je Json-server [12]prvenstveno zbog jednostavnosti korištenja naspram drugih simulatora pozadine. Izrada stranice započela je od **`Cart`** komponente. Svrha te komponente je prikaz proizvoda i informacija o njima. Komponenta nudi mogućnost dodavanja

[12] Json-server je alat za React koji omogućuje brzu i jednostavnu izradu simulirane pozadine i servera. Prvenstveno se koristi za testiranje i razvoj funkcionalnosti (github.com, 2023) .

proizvoda u košaricu, brisanja proizvoda sa stranice te navigaciju po drugim komponentama putem navigacijske trake koja je fiksni dio svake komponente na stranici.

U **`Product`** komponenti na slici 17 definiran je objekt koji sadrži informacije o proizvodu. Pomoću prenošenja stanja sa jedne roditeljske komponente („**`product`**“), informacije o različitim proizvodima u bazi podataka prenose se na svaki proizvod zasebno putem „**`id`**“ svojstva kao ključa.

```
return (
  <div className="col-lg-6">
    <div className="card m-2">
      <div className="card-body">
        <div className="text-muted">
          # {this.state.product.id}
          <span
            className="pull-right hand-icon"
            onClick={() => {
              this.props.onDelete(this.state.product);
            }} >
            <i className="fa solid fa-trash" />
          </span>
        </div>
        <h5 className="pt-2 border-top">
          {this.state.product.productName}
        </h5>
        <div>${this.state.product.cijena}</div>
      </div>
      <div className="card-footer">
        <div className="float-left">
                    <span className="badge">
                      {this.state.product.stanje}</span>
          <div className="btn-group">
            <button
              className="btn btn-outline-success"
              onClick={() => {
                this.props.onIncrement(this.state.product, 10)
              }}
            >
              +
            </button>
            <button
              className="btn btn-outline-success"
              onClick={() => {
                this.props.onDecrement(this.state.product, 0);
              }}
            >
              {" "}
              -
            </button>
          </div>
        </div>
```

*Slika 17: Kôd unutar Product komponente*

Također može se vidjeti kako se pozivanjem stanja (slika 17; kodni redak 32-34 i 39) dobivaju različite informacije o svakom proizvodu te kako se umjesto definiranja funkcije negdje drugdje u kôdu, koriste funkcije strelica koje čine kôd preglednijim i kraćim (primjerice slika 17; kodni redak 51). Ako su funkcije definirane u istoj komponenti poziva ih se u vitičastim zagradama te se koristi ključnu riječ „**this**“ kojom specificiramo što želimo pozvati.

```
17    render() {
18      //console.log("render - cart");
19      return (
20        <div>
21          <div className="row">
22            {this.state.products.map(prod => {
23              return (
24                <Product
25                  key={prod.id}
26                  product={prod}
27                  onIncrement={this.handleIncrement}
28                  onDecrement={this.handleDecrement}
29                  onDelete={this.handleDelete}
30                >
31                  <button id="btn" className="btn btn-primary"
32                    onClick={() => this.handleClick(prod)} >
33                    Dodaj
34                  </button>
35                </Product>
36              );
37            })}
38          </div>
39        </div>
40      );
```

*Slika 18: Render metoda u Cart.jsx komponenti*

Na slici 18 prikazana je **render** metoda u glavnoj komponenti. U ovoj metodi se odvija najveći dio funkcionalnosti na cijeloj stranici. Unutar **<Product>** elementa može se vidjeti pozivanje više funkcija koje su zaslužene za povlačenje ključa i informacija o proizvodu sa **Product** komponente koja je prikazana na slici 17. Također vidljiva je opcija povećanja ili smanjivanja stanja proizvoda u košarici te dodavanje istog u košaricu. Nakon **render** metode definirane su funkcije koje se koriste u **Cart.jsx** komponenti. . U slučaju **render** metode prikazane na slici 18 ne mora se koristiti **export** da bi se mogli pozivati funkcije iz razloga što ih se poziva u istom dokumentu, u suprotnom, žele li koristiti funkcije ili klase u drugim komponentama mora ih se definirati sa operacijom „**export**“ ispred imena.

Na slici 19 može se vidjeti **`componentDidMount`** funkcija koja koristeći Json-server i „**`GET`**“ metodu dohvaća podatke iz baze podataka pri svakom ponovnom pokretanju stranice te funkcije za povećavanje i smanjivanje stanja proizvoda koje se žele dodati u košaricu.

```
componentDidMount = async () => {
    var response = await fetch("http://localhost:5000/products", {
        method: "GET"
    });

    var prods = await response.json();

    this.setState({ products: prods });

};

handleIncrement = (product, maxValue) => {
    let allProducts = [...this.state.products];
    let index = allProducts.indexOf(product);

    if (allProducts[index].stanje < maxValue) {
        allProducts[index].stanje++;
    }

    this.setState({ products: allProducts });
};

handleDecrement = (product, minValue) => {
    let allProducts = [...this.state.products];
    let index = allProducts.indexOf(product);

    if (allProducts[index].stanje > minValue) {
        allProducts[index].stanje--;
    }

    this.setState({ products: allProducts });
};
```

*Slika 19: Prvi dio funkcija u Cart.jsx komponenti*

U drugom dijelu funkcija u ovoj komponenti prikazanoj na slici 20 u kodnom retku 78-89 vidljive su funkcije zadužene za brisanje proizvoda **`handleDelete`**. Funkcija radi tako da pronađe proizvod te ga na temelju indeksa briše iz stanja komponente. Funkcija **`handleClick`** (slika 20; kodni redak 91-103), koja prilikom klika na gumb „**`dodaj`**“ koji je definiran u **`Cart`** komponenti (slika 18; kodni redak 31-34), taj proizvod dodaje u košaricu korištenjem **`async`** metode. Koristeći „**`POST`**“ metodu omogućuje se dinamičko dodavanje proizvoda u košaricu.

```
78      handleDelete = product => {
79          let allProducts = [...this.state.products];
80          let index = allProducts.indexOf(product);
81
82          allProducts.splice(index, 1);
83          if (window.confirm("Are you sure you won't take another look")) {
84              //brišemo kod na temelju indexa
85              allProducts.splice(index, 1);
86              //ažuriramno stanje trenutne komponente(roditelja)
87              this.setState({ products: allProducts });
88          }
89      };
90
91      handleClick = async (prod) => {
92          const rawResponse = await fetch("http://localhost:5000/cart", {
93              method: "POST",
94              headers: {
95                  "Accept": "application/json",
96                  "Content-Type": "application/json"
97              },
98              body: JSON.stringify({ ...this.state.items, prod })
99
100         });
101         const content = await rawResponse.json();
102         console.log(content);
103     };
104
105 }
```

*Slika 20: Drugi dio funkcija u Cart.jsx komponenti*

Navigacijska traka prikazana na slici 21 napravljena je pomoću besplatnih predložaka na getbootstrap.com stranici te je fiksna i uvezena u svaku komponentu koja se prikazuje na ekranu. Komponenta navigacijske trake ugniježđena je u React Elementu kojeg se definira s **„<>“** i **„</>“** oznakama (slika 21; kodni réci 7 i 44). Koristeći **<Link>** element (slika 21; kodni redak 22-24) definira se navigacijska ruta za svaku komponentu stranice.

```
7   <>
8     <nav className="navbar navbar-expand-lg navbar-dark bg-dark">
9       <a className="navbar-brand">Internet Trgovina</a>
10            <button className="navbar-toggler"
11                type="button" data-toggle="collapse"
12          data-target="#navbarSupportedContent"
13          aria-controls="navbarSupportedContent"
14          aria-expanded="false"
15          aria-label="Toggle navigation" >
16          <span className="navbar-toggler-icon" />
17        </button>
18            <div className="collapse navbar-collapse"
19                id="navbarSupportedContent">
20          <ul className="navbar-nav mr-auto">
21            <li className="nav-item active">
22              <Link to="/login" className="nav-link">
23                Prijava
24              </Link>
25            </li>
26            <li className="nav-item active">
27              <Link to="/cart" className="nav-link">
28                Proizvodi<span className="sr-only">(current)</span>
29              </Link>
30            </li>
31            <li className="nav-item active">
32              <Link to="/customers" className="nav-link">
33                Kupci <span className="sr-only">(current)</span>
34              </Link>
35            </li>
36            <li className="nav-item active">
37              <Link to="/kosarica" className="nav-link">
38                Košarica <span className="sr-only">(current)</span>
39              </Link>
40            </li>
41          </ul>
42        </div>
43      </nav>
44    </>
```

*Slika 21: Kôd navigacijske trake*

```
export default class Kosarica extends Component {
  constructor(props) {
    super(props);

    this.state = {
      items: [],
      DataisLoaded: false
    };
  }

  componentDidMount() {
    fetch("http://localhost:5000/cart")
      .then(res => res.json())
      .then(json => {
        this.setState({
          items: json,
          DataisLoaded: true
        });
      });
  }
  render() {
    const { DataisLoaded, items } = this.state;
    if (!DataisLoaded) {
      return (
        <div>
          <h1>Ažuriranje...</h1>
        </div>
      );
    } else if (items.length <= 0) {
      return <h3> Košarica je prazna</h3>;
    }

    return (
      <div className="Kosarica">
        <div className="col-lg-6">
          {this.state.items.map(
              item => (console.log(item),
                  <Cart key={item} item={item} />)
          )}
        </div>
      </div>
    );
  }
}
```

*Slika 22: Kôd Kosarica komponente*

U komponenti **`Kosarica`** prikazanoj na slici 22 prikazuju se proizvodi, prethodno dodani u košaricu klikom na „dodaj“ gumb . Prikaz podataka u košarici vrši se tako da se pozivaju stanja iz baze podataka, te uvozi objekt „**`Product`**“ iz „**`Product.jsx`**“ datoteke za prikaz podataka. Stvara se niz u konstruktoru u kojemu se zatim koristeći „**`fetch`**“ metodu dohvaćaju podaci iz baze podataka. Za prikaz proizvoda po karticama koristi se „**`map`**“ metoda te unutar **„`<Cart key={item} item={item} />`“** poziva se roditeljsku „**`Cart`**“ komponenta.

# 7. Zaključak

U radu su prikazane prednosti i mane Reacta te usporedba s dva druga alata koji se koriste za izradu Internet aplikacija. Navedeno je koji se programski jezici mogu pronaći u Reactu te su pojedinačno obrađeni. Objašnjeno je kako se React razlikuje od ostalih alata time što on nije okvir za programiranje već se radi o JavaScript knjižnici. Stoga je njegova glavna značajka, koja mu je istovremeno i prednost i mana, da se bez upotrebe drugih alata i proširenja ne može puno napraviti samo s React knjižnicom. Pozitivna stvar je što ta činjenica korisniku ostavlja na izbor što želi ili ne želi koristiti u svojoj aplikaciji te pruža mogućnost smanjivanja kompleksnosti i veličine aplikacije.

Na temelju primjera Internet trgovine prikazane su značajke Reacta poput ugniježđenih komponenti, korištenja JSX-a za lakše razumijevanje sintakse, razlike između stanja i svojstava, upravljanja događajem na primjeru formulara. Zaokruživši sve to prikazano je da iako React ima veliku krivulju učenja, koristi vrlo jednostavne načine izrade stranice. Prilikom usporedbe različitih alata za izradu internetskih stranica mogu se uočiti i neki nedostaci Reacta poput nedovoljno razvijene dokumentacije. Zbog velike brzine razvoja koja čini učenje Reacta kompliciranim jer tek što korisnik nauči sintaksu, ona se u nekom smislu promjeni ili ažurira te da bi funkcionalno koristili React korisnik mora morate ponovno učiti drugačiji način dobivanja istog rezultata. Također je obrađena činjenicu da je React sam po sebi knjižnica te ga se ne može samostalno koristiti za razvitak neke stranice nego uz njega moraju koristiti drugi alati. Jedna od najvećih značajki Reacta je JSX jer omogućava korištenje HTML-a zajedno s JavaScriptom. Iako je prilično zahtjevan za ovladati novim korisnicima, olakotna je okolnost što JSX nije obavezan dio Reacta. S obzirom da React ima tako slobodnu formu, na većim projektima gdje radi više ljudi dolazi do komplikacija jer svatko ima svoj način rada. Potrebno je definirati ograničenja i smjernice da se projekt ne bi razgranao u krivom smjeru.

Na temelju analiziranih značajki može se zaključiti da je React izuzetno fleksibilan i skalabilan alat za izradu mrežnih aplikacija. Širok opseg knjižnica kojima ima pristup omogućuje iskoristivost na mrežnim aplikacijama svih formi i veličina. Njegova osnovna prednost nad drugim alatima je slobodna forma i mogućnost izbora do koje ga mjere želimo implementirati. Upravo zbog tih funkcionalnosti postao je jedan od najzastupljenijih alata s preko 16 miliona korisnika[13].

[13] https://github.com/facebook/react

# 8. Literatura

1. *Introducing asynchronous JavaScript*. (n.d.). Dohvaćeno iz developer.mozzila.org: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Introducing
2. (10. 7 2023). Dohvaćeno iz hjp.znanje.hr: https://hjp.znanje.hr/index.php?show=search
3. (10. 7 2023). Dohvaćeno iz hjp.znanje.hr: https://hjp.znanje.hr/index.php?show=search_by_id&id=elpiWhU%3D
4. Atuonwu, S. C. (4. 5 2020). *Var, Let, and Const – What's the Difference?* Dohvaćeno iz freecodecamp.org : https://www.freecodecamp.org/news/var-let-and-const-whats-the-difference/
5. Au-Yeung, J. (n.d.). *using classes in javascript*. Dohvaćeno iz thewebdev.info : https://thewebdev.info/2020/08/05/using-classes-in-javascript/
6. Buna, S. (2019). React Succintly. Morrisville: Syncfusion, inc.
7. *css*. (18. 7 2023). Dohvaćeno iz getbootstrap.com: https://getbootstrap.com/docs/3.4/css/
8. *docs*. (18. 7 2023). Dohvaćeno iz fontawesome.com: https://fontawesome.com/docs/
9. *Document_Object_Model/Introduction* . (10. 7 2023). Dohvaćeno iz developer.mozzilla.org: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction
10. *faq-internals*. (n.d.). Dohvaćeno iz reactjs.org : https://reactjs.org/docs/faq-internals.html
11. Fedosejev, A. (2015). *React.js Essentials.* Birmingham: Packt Publishing Ltd.
12. *Functions*. (n.d.). Dohvaćeno iz developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions
13. Gilyadov, L. (2018). *history.html*. Dohvaćeno iz developer-cheatsheets.com : http://www.developer-cheatsheets.com/history.html
14. *github.com*. (18. 7 2023). Dohvaćeno iz json-server: https://github.com/typicode/json-server
15. Hámori, F. (31. 5 2022). *The history of React.js on a timeline*. Dohvaćeno iz blog.risingstack.com : https://blog.risingstack.com/the-history-of-react-js-on-a-timeline/
16. *html_css*. (n.d.). Dohvaćeno iz w3schools.com : https://www.w3schools.com/htmL/html_css.asp

17. *introducing-jsx*. (n.d.). Dohvaćeno iz reactjs.org : https://reactjs.org/docs/introducing-jsx.html
18. *JavaScript_basics*. (10. 7 2023). Dohvaćeno iz developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/JavaScript_basics
19. *jquery_intro_asp*. (18. 7 2023). Dohvaćeno iz www.w3schools.com: https://www.w3schools.com/jquery/jquery_intro.asp
20. *js_scope.asp*. (n.d.). Dohvaćeno iz w3schools.com : https://www.w3schools.com/js/js_scope.asp
21. *Most used web frameworks among developers worldwide, as of 2022*. (8 2022). Dohvaćeno iz statista.com: https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/
22. Rauschmayer, A. (2022). *JavaScript For Impatient Programmers (ES2022 edition)*.
23. *react-component*. (n.d.). Dohvaćeno iz reactjs.org : https://reactjs.org/docs/react-component.html
24. *reactjs-tutorial*. (n.d.). Dohvaćeno iz simlilearn.com : https://www.simplilearn.com/tutorials/reactjs-tutorial/reactjs-state#the_setstate_method
25. *Scope*. (n.d.). Dohvaćeno iz developer.mozilla.org : https://developer.mozilla.org/en-US/docs/Glossary/Scope
26. *what-is-vue*. (2. 8 2023). Dohvaćeno iz vuejs.org: https://vuejs.org/guide/introduction.html#what-is-vue
27. *why-popper*. (18. 7 2023). Dohvaćeno iz popper.js.org: https://popper.js.org/docs/v2/#why-popper
28. *Working with objects*. (n.d.). Dohvaćeno iz developer.mozilla.org : https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects

# 9. Tablica sadržaja

# Sažetak

Tema ovog rada je okvir za programiranje React.js te njegov način rada i primjena u izradi modernih internetskih stranica. Cilj rada je prikazati prednosti i mane Reacta ne samo na teoretskoj razini već i praktičnoj kroz konkretni primjer stranice s internetskom trgovinom. S ciljem jasnog razumijevanja Reacta, u radu je kratko opisana povijest tehnologije koja se koristi pri izradi internetskih stranica te problemi koji su doveli do razvoja samog Reacta. Opisane su tehnologije iz mrežnih standarda poput JavaScript-e, HTML-a i CSS-a koje se koriste u Reactu kao i tehnologije po kojima se on razlikuje od ostalih alata za izradu mrežnih aplikacija. Dodatno je napravljena usporedba između okvira Reacta, Vue.js-a i Ember.js-a uz prikaz njihovih općih prednosti i mana. Na konkretnom primjeru Internet trgovine pojašnjene su komponente Reacta poput ugniježđenih komponenti, JSX-a, razlike između stanja i značajki te prednosti virtualnog DOM-a naspram stvarnog. Ukratko su prikazani i načini upotrebe različitih alata potrebnih za izradu uporabljive internetske stranice koja u svojoj podlozi ima Internet trgovinu.

**Ključne riječi**: *React.JS. JSX, JavaScript, virtualni DOM, Internet trgovina*.

# React.js framework

## Abstract

The topic of this thesis is the programming framework React.js, its functionality and application in developing modern web pages. The thesis anaylzes advantages and disadvantages of React both on the theoretical and practical levels via an example webpage with a web store. In order to better understand React, the thesis briefly describes the history of the technologies used in creating network standards such as JavaScript, HTML and CSS which are also utilised in React, as well as its technologies which make Reac different from other tools for creation of network applications. Additionally, a comparison is made between React.js, Vue.js and Ember.js framework explaining their general features. The web store example presents React components such as nested components, JSX, differences between state and properties and advantages of virtual DOM over the real one. It briefly summarizes methods of using different tools necessary for making a functional web page that includes a web store.

**Key words**: React.js, JSX, JavaScript, virtual DOM, web store