Primjena JavaScript biblioteka i okvira za izradu dinamičkih web aplikacija s grafičkim prikazima

Anđel, Mihael

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Mihael Anđel

Primjena JavaScript biblioteka i okvira za izradu dinamičkih web aplikacija s grafičkim prikazima

ZAVRŠNI RAD

Varaždin, 2019.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Mihael Anđel

Matični broj: 44864/16–R

Studij: Informacijski sustavi

Primjena JavaScript biblioteka i okvira za izradu dinamičkih web aplikacija s grafičkim prikazima

ZAVRŠNI RAD

Mentor:

Prof. dr. sc. Dragutin Kermek

Varaždin, rujan 2019.

Mihael Anđel

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Skriptni jezici nalik su standardnim programskim jezicima, no posjeduju neke prednosti, ali i nedostatke. Povijest skriptnih jezika dosta je mlada što se tiče računalnih tehnologija. Usprkos tome postoji mnogo skriptnih jezika, neki su odumrli, a neki se i danas koriste. Popularnost skriptnih jezika raste s vremenom, a razlike između standardnih programskih jezika i skriptnih jezika postaju manje izražene. Web aplikacija sve je popularniji i učestaliji tip aplikacije. Razvojem tehnologije rastu i mogućnosti razvojnih programera da stvore web aplikacije nalik standardnim, izvornim aplikacijama. Glavna tehnologija kod razvoja modernih web aplikacija je skriptni jezik JavaScript. Nastankom ECMAScript specifikacije JavaScript postaje standardiziran, što omogućuje lakši razvoj web aplikacija na bilo kojem web pregledniku. Posljedica standardizacije je ubrzan razvoj različitih web tehnologija. Jedna od najznačajnijih web tehnologija vezanih uz JavaScript je Node.js. Node.js omogućio je razvoj čitavog sustava web aplikacije istim jezikom. To znači da je postalo moguće napisati standardnu web aplikaciju na klijentskoj strani i poslužitelj s visokim performansama koristeći isti jezik - JavaScript. Činjenica da je Node.js izvorni kod javno dostupan samo je pomogla u razvoju raznih web tehnologija koje se danas svakodnevno koriste. Tržište zahtijeva sve brži razvoj što boljih web aplikacija. Iz tog razloga nastaju razne JavaScript biblioteke i okviri. Dok se biblioteke mogu shvatiti samo kao dodaci za JavaScript, okviri mijenjaju čitav tok i način razvoja web aplikacije. Vrhunac samog rada je u samostalnoj izradi moderne web aplikacije korištenjem JavaScript biblioteka i okvira.

Ključne riječi: skripta; skriptni jezik; JavaScript; web aplikacija; JavaScript biblioteka; JavaScript okvir

Sadržaj

1. Uvod

Od svojih početaka u drugoj polovici prošlog stoljeća, Internet je doživio mnoge promjene. U ranim danima postojanja, Internet se sastojao gotovo isključivo od web stranica koje su, nakon učitavanja, prikazale svoj sadržaj i dalje se nisu mijenjale. Vrhunac tadašnjih web tehnologija bili su HTML i CSS. HTML, strogo temeljen na „Standard Generalized Markup Language“ (SGML), je 1990. godine postao fundamentalna tehnologija za web razvoj [1]. Iako su HTML i CSS bile i ostale korisne tehnologije za izradu statičkih web stranica, one omogućuju upravo samo to – izradu statičkih web stranica. Javnost tada još nije u potpunosti prihvatila Internet kao ozbiljnu tehnologiju – zahtijevala je više. Emery [1] navodi kako je generalnoj populaciji bilo potrebno grafičko sučelje, pa tako i web preglednik kako bi shvatila potencijal Interneta. Internet je danas puno drugačije mjesto. Kompleksniji je, zahtjevniji i moćniji, a među glavnim krivcima su skriptni jezici poput JavaScript-a i PHP-a. Današnje web stranice više nisu samo „stranice“, već legitimne aplikacije poput onih koje bi našli na osobnim računalima. Jedan od najboljih primjera evolucije Interneta jesu progresivne web aplikacije. „Progresivne web aplikacije su nova vrsta web aplikacija koje kombiniraju koristi izvornih aplikacija s niskim trenjem web-a.“[2, str. 2].

U današnjem, modernom, svijetu sve veći udio uređaja ovisi o Internetu. Gotovo svaka osoba posjeduje pametni telefon, pa je iz tog razloga pametni telefon postao jedna od najpopularnijih vrsta uređaja za pretraživanje Interneta. Rastom popularnosti različitih uređaja mijenjaju se i potrebna znanja web programera. Prema istraživanju provedenog od strane Pew razvojnog centra (*Pew Research Center - PRC*) [3] od svibnja 2011. do veljače 2019., udio punoljetnih stanovnika SAD-a koji posjeduju pametni telefon porastao je od 35% do 81%. Kombinacijom ove informacije i prijašnjeg odlomka moguće je donijeti zaključak kako se cijeli svijet orijentira prema web-u, pa su stoga web tehnologije poput JavaScript-a u samom centru pažnje.

U ovom radu obradit će se razvoj modernih web aplikacija uz pomoć JavaScript skriptnog jezika i njegovih proširenja, odnosno biblioteka i okvira. Istražit će se razne biblioteke i okviri i pogledati njihove jake i slabe strane. U samom vrhuncu rada izradit će se moderna web aplikacija korištenjem nekih od obrađenih biblioteka i okvira JavaScript skriptnog jezika. Motivacija iza ovog rada je interes i želja za unaprjeđenjem znanja i vještina. Web aplikacije su budućnost pa se stoga isplati učiti kako ih razviti.

2. Skriptni jezici

U ovom poglavlju napravit će se općeniti pregled povijesti skriptnih jezika i njihovih svojstava, pravila i mogućnosti kako bi se u sljedećim poglavljima lakše mogle definirati neki pojmovi. Također će se napraviti pregled nekih od najuspješnijih i najpopularnijih skriptnih jezika koji se koriste, ili su bili korišteni, za web razvoj.

2.1. Povijest skriptnih jezika

Iako ne postoji točan datum kada su programeri počeli koristiti skriptne jezike, riječ „skripta“ se u računalstvu počela koristiti u drugoj polovici prošlog stoljeća.

Sedamdesetih godina prošlog stoljeća tvorci UNIX operacijskog sustava koristili su termin „skripta“ za blok naredbi koji bi se spremio u datoteku. Računalo bi zatim izvodilo svaku liniju skripte kao da je linija napisana tipkovnicom [4]. Iz toga se može zaključiti da su skriptni jezici nastali kao pomoćni alat za pisanje nekog niza naredbi koji bi se često izvodio na računalu. Bili su jako moćan alat, pogotovo u vrijeme kada operacijski sustavi nisu imali grafička sučelja. Iako se i danas koriste, moglo bi se reći da takvi skriptni jezici nisu prvo što čovjeku padne na pamet kad čuje izraz „skriptni jezik“.

S vremenom, skriptni jezici počeli su biti sve kompleksniji i sve sličniji standardnim programskim jezicima poput C i C++. Uvedeni su programski koncepti, što je omogućilo skriptama da budu puno korisnije nego što su bile do tad. Danas postoji više tipova skriptnih jezika, no najpoznatiji su oni koji se koriste za web razvoj. Osim njih, među najpoznatijim skriptnim jezicima nalazi se i Python, koji se može opisati kao programski jezik za sve namjene (*all purpose programming language*).

Što se tiče skriptnih jezika za web razvoj, najpoznatiji i najkorišteniji danas su definitivno JavaScript i PHP. Osim njih, kroz povijest se pojavilo još nekoliko jezika koji nisu ispunili očekivanja. Jedan od takvih bio je jezik *„tcl“*. Prema Bloom [5], bilo je mnogo pokušaja da se jezik tcl integrira u web preglednike, poput *„ActiveX“* priključka i službene zakrpe za Firefox web preglednik. Osim toga, tcl se nalazi i u HTML 4.01 referenci kao vrsta skriptnog jezika koji se može koristiti unutar <*script*> elementa [6].

2.2. Svojstva skriptnih jezika

Skriptni jezici razlikuju se od ostalih jezika po jednoj velikoj stvari: ne koriste prevoditelj kako bi pretvorili izvorni kod u konkretne procesorske instrukcije. Umjesto prevoditelja, skriptni jezici koriste interpreter. Interpreter, za razliku od prevoditelja, pretvara izvorni kod u

procesorske instrukcije liniju po liniju. Ta razlika odbija mnoge programere od ideje da krenu učiti neki skriptni jezik. U početku se čini da je s takvim jezikom jako naporno i teško raditi, što baš i nije istina. Najveći problem programerima koji su naviknuti na prevoditelje je obično to što interpreter ne izbaci poruke o greškama i upozorenjima tako dugo dok ne dođe do linije koja stvara problem. Nakon pronalaska problematične linije koda, interpreter prestaje s izvađanjem programa. Ova činjenica zapravo ide u korist argumentu da su interpreteri bolji i lakši za shvatiti od prevoditelja, pogotovo kada je riječ o novim programerima. Prema Škotskom tijelu za kvalifikacije (*Scottish Qualifications Authority - SQA*) [7], interpreteri su laganiji za korištenje jer se greške prikazuju odmah, tako dugo dok ih korisnik ne ispravi. Unatoč tomu, danas postoji mnogo alata koji omogućuju što lakše otklanjanje grešaka u kodu. Isto tako, svi poznatiji web preglednici imaju ugrađeni skup programa za web programere kako bi se što više olakšao web razvoj.

Još jedna stvar koja pada na um kada se spomenu skriptni jezici jest prenosivost. Kako se ovdje ne radi o prevoditeljima, s lakoćom je moguće uzeti postojeći program, odnosno skriptu i prenijeti ju na novo računalo bez ijedne promjene u programskom kodu. Jedini preduvjet je taj da računalo na kojem želimo pokrenuti skriptu mora imati instaliran korektni interpreter. Vezano uz prenosivost je i prilagodljivost. Programer može uzeti nečiju skriptu i s lakoćom ju prilagoditi za svoje računalo jer preuzimanjem skripte ne preuzima izvršnu datoteku, već izvorni kod.

Lakoća korištenja definitivno je jedna od najvećih pozitivnih strana skriptnih jezika. Dok standardni programski jezici koriste kompleksne strukture podataka kako bi postigli željeni cilj, skriptni jezici imaju unaprijed ugrađene strukture koje se često koriste. Te strukture su obično poprilično jednostavne. Barron [4] navodi kako je u skriptnim jezicima broj tipova podataka i struktura podataka ograničen – obično se za većinu toga koristi tip *string*, a polje (*array*) je često jedina struktura podataka koja se koristi.

S druge strane, poznata je činjenica da je skriptnim jezicima efikasnost daleko od prioriteta. Skripte ne zahtijevaju visoku efikasnost, bitnije stvari su brzina razvoja i mogućnost promjene koda da se riješe novi, nepredvidivi zahtjevi [4]. Nerazumno je napisati program u standardnom programskom jeziku pa zatim napisati ekvivalentan program u skriptnom jeziku i očekivati da će se program brže izvađati u skriptnom jeziku. Ovo je nažalost jedna od najvećih negativnih strana skriptnih jezika; kompromis – dobiva se lakoća učenja i korištenja, prenosivost i prilagodljivost, a gubi efikasnost.

2.3. PHP

PHP: Hypertext Preprocessor ili skraćeno samo PHP, jedan je od najkorištenijih skriptnih jezika danas. Kao što i ime govori PHP je preprocesor za HTML jezik, što znači da se on primarno koristi, između ostalog, za generiranje HTML sadržaja web stranice. Nastao je 1994. godine, a stvorio ga je Rasmus Lerdorf. Tatroe, MacIntyre, Lerdorf [8] navode kako je PHP nastao kao odgovor na zahtjeve šire javnosti: zaštita web stranica lozinkom, jednostavno kreiranje obrasca i pristup unesenim podacima na sljedećoj stranici.

PHP za kakvog se danas zna zapravo je osmišljen kao nasljednik drugog alata zvanog PHP/FI. Prva verzija PHP jezika zapravo niti nije bila svoj jezik – napisana je kao skup skripata u programskom jeziku C. Potrebom za većom funkcionalnošću, PHP je bio ponovo napisan od početka kao samostalan jezik s puno većom funkcionalnošću [9].

```
Primjer programskog koda za PHP/FI [9]:
<!—include /text/header.html→

<!—sql database select * from table where user='$username'→
<!—ifless $numentries 1→
  Sorry, that record does not exist<p>
<!—endif exit→
  Welcome <!--$user→!<p>
  You have <!--$index:0→ credits left in your account.<p>

<!—include /text/footer.html→
```

Iz primjera se može vidjeti kako je PHP/FI, iako je bio rana verzija PHP-a, već imao mogućnosti rada s bazom podataka. PHP/FI instrukcije su ovdje pisane unutar HTML komentara, no one bi se inače nalazile unutar standardnog PHP bloka – između „<?php“ i „?>“. Moglo bi se reći da je moderan PHP, sličan onome kakav se danas koristi, nastao 1998. godine nastankom PHP verzije 3.0. Prema [9] Andi Gutmans i Zeev Suraski su, u suradnji s Rasmusom Lerdorfom, počeli s kompletnom preradom PHP-a 1997. godine. Izlaskom PHP 3.0 šestog lipnja 1998., jezik je doživio neviđen rast u popularnosti, a posljedica toga bio je ubrzan razvoj jezika što se može vidjeti iz datuma izlaska sljedećih verzija.

PHP izvorna datoteka obično u sebi sadrži HTML sadržaj koji tvori web stranicu. Naravno, PHP ne bi bio svoj jezik da je to sve što radi. HTML sadržaj je sam po sebi statičan; ne mijenja se. PHP omogućuje HTML sadržaju da se generira dinamički uz pomoć programskih koncepata.

Primjerice, korisnik se prijavi na neki web servis putem obrasca. Uneseni podaci se dalje šalju drugoj PHP skripti koja provjerava unesene podatke spajanjem na bazu podataka. Stvara se sesija ukoliko su podaci ispravni. Nakon uspješne prijave otvara se naslovna stranica web servisa na kojoj korisnik može vidjeti podatke usklađene prema njegovim željama. Kako

niti jedan korisnik nije jednak, svaki korisnik otvorit će istu stranicu, no sadržaj stranice biti će različit za svakog korisnika.

Iz primjera mogu se izvući dvije najčešće korištene funkcionalnosti PHP jezika: rad s bazom podataka i korištenje obrasca kao posrednika između dvije PHP skripte.

Svaka PHP skripta sadrži PHP blok naredbi koji je omeđen nizovima znakova: „<?php“ i „?>“. Niz znakova za završetak bloka može se izbaciti ukoliko se nakon bloka više ne pojavljuje nikakav drugačiji sadržaj. Unutar PHP bloka pišu se linije PHP koda. Svaka linija mora, kao i u nekim drugim jezicima, završiti s točkom i zarezom. Unutar svake linije mogu se nalaziti ključne riječi, varijable, funkcije, sistemske klase i klase izrađene od strane korisnika. Bitno je napomenuti kako PHP razlikuje velika i mala slova unutar linija. Ako se radi o varijabli, PHP raspoznaje velika i mala slova. U slučaju da se radi o nazivima imenskih prostora, funkcija, klasa, sučelja ili metoda PHP ne raspoznaje velika i mala slova.

Primjer velikih i malih slova u PHP-u:

```
<?php
echo "Test";
Echo "Test";
ECHO "Test";
?>
```

Sve tri linije unutar primjera ekvivalentne su, odnosno PHP ne raspoznaje nikakve razlike između njih.

Nadalje, sve varijable unutar PHP jezika pišu se u sljedećem obliku: $*naziv_varijable*. Ispred naziva svake varijable nalazi se poseban znak „$“. Kao što se može vidjeti, PHP nema deklaracije tipa varijable, tako da se bilo kojoj varijabli u bilo kojem trenutku može pridružiti bilo koja vrijednost. Nepisano pravilo je da se prilikom deklaracije varijable varijabli dodijeli vrijednost, a dobra praksa je ne mijenjati tip podatka pojedine varijable. Primjerice, deklarira li se varijabla *$ime*, primjereno je odmah joj dodijeliti vrijednost i ne mijenjati dodijeljen tip vrijednost:

```
<?php
$ime = "Mihael";
$ime = 123 //ovo u pravilu nije dobro raditi
$ime = "Marko";
$ime = "123"; //iako nema smisla u kontekstu, ovo bi bilo u redu
?>
```

PHP pruža osam tipova podatka: cijeli broj (*integer*), broj s pomičnom decimalnom točkom (*float*), tekst (*string*), istina/laž (*boolean*), polje (*array*), objekt (*object*), resurs (*resource*) i praznina (*NULL)* [8]. Najčešće korišteni tipovi podataka unutar web aplikacija su cijeli brojevi i tekst, odnosno *string*.

Cijeli brojevi mogu poprimiti vrijednost bilo kojeg cijelog broja, a minimum i maksimum određeni su ovisno o postavkama poslužitelja. Ako se radi o negativnom broju, potrebno je

prije broja staviti znak minus (-). Zanimljivo je to što se cijeli broj može zapisati u različitim oblicima, ovisno o tome koja se baza koristi. Broj se zapisuje u oktalni sustav tako da mu se kao početna znamenka stavi nula (0), u heksadecimalni tako da započinje s „0x“ i u binarni tako da započinje s „0b“.

Tekst se u PHP-u može navesti na dva načina: korištenjem dvostrukih ili jednostrukih navodnika. Bitna razlika između jednostrukih i dvostrukih navodnika je ta da se unutar jednostrukih navodnika sav tekst zapisuje doslovno kako je zapisan unutar linije. Tekst unutar dvostrukih navodnika podržava interpolaciju varijabli, što znači da će se umjesto naziva varijable prikazati njezin sadržaj.

Primjer jednostrukih i dvostrukih navodnika:

```
<?php
$ime = "Mihael";
echo '$ime';
echo "$ime";
?>
```

Prva *echo* naredba unutar primjera će ispisati „$ime“, dok će druga *echo* naredba ispisati „Mihael“.

2.4. Perl

Među svim skriptnim jezicima Perl se definitivno ističe po svojstvima. Perl je programski jezik, no po njegovim leksičkim svojstvima može se vidjeti da je dizajniran s prirodnim jezikom na umu. Upravo to je i glavna prednost (mana) jezika: Perl je toliko prilagođen ljudskom govoru i toliko je jednostavan za shvatiti da klasični programeri naviknuti na standardne programske jezike ponekad imaju problema s prilagođavanjem.

Perl je osmislio Larry Wall, a prvi je put bio objavljen 1987. godine. Wall je zapravo jezikoslovac i svoje znanje jezika prenio je na dizajn Perl-a. Prva namjena bila je da Perl bude jezik za obradu teksta; da popuni praznine između UNIX skriptnih jezika i programa pisanih u C programskom jeziku. Popularnost Perl-a rasla je eksponencijalno do 1994. godine, kada je verzija 4 postala standardni UNIX programski jezik. Unatoč tomu, Perl je bio dosta ograničen, a posljedica toga bio je gotovo nikakav rast u njegovu korištenju. To je potaklo Walla da unaprijedi jezik. 1994. godine službeno je izdana Perl verzija 5, s kojom je Perl postao moćan programski jezik opće namjene. [10]

Po mnogim svojstvima ne razlikuje se od ostalih skriptnih jezika (PHP), ali posjeduje neka „neobična“ svojstva. Primjerice, kao i ostali skriptni jezici, nema deklaracije tipa podatka, ali ipak koristi različite simbole za razlikovanje običnih varijabli (skalara) od drugačijih struktura. Za skalare Perl, kao i PHP, koristi znak „$“. Osim tog znaka Perl koristi još neke: „@“ za polja, „&“ za funkcije i „%“ za hash-tablice.

Skalari mogu poprimiti uobičajene vrijednosti poput cijelog broja ili teksta. Što se tiče teksta, ovdje se opet nailazi na velike sličnost s ostalim jezicima, a pogotovo s PHP-om. Tekst između dvostrukih navodnika radi interpolaciju varijabli i obrnute kose crte, a tekst unutar jednostrukih navodnika shvaća se doslovno kako je napisan.

Osim skalara, Perl posjeduje još dva tipa podatka: polje (*array*) i hash-tablica (*hash table*). Ovi tipovi, odnosno strukture, podataka standardni su u programerskom svijetu. Polje je indeksirani niz (prvi indeks je 0). Hash-tablica može se usporediti s poljem, samo što ima imenovane indekse. Sastoji se od parova koji formiraju ključ i vrijednost. Ključ mora biti jedinstven na razini cijele hash-tablice.

Dio gdje Perl pokazuje još svojih neobičnih konvencija je dio o objektima i klasama. Wall [11] piše kako tehnički, objekt nije sama referenca, već referent na kojeg referenca pokazuje. Svaki objekt pripada nekoj klasi, a ako klasa definira neke metode tada se te metode mogu pozvati nad bilo kojim objektom klase. Perl definira dva načina na koji se to može napraviti. Jedan je klasičan način poznat programerima jer se koristi i u ostalim jezicima – korištenje operatora strelice (*arrow operator*). Drugi način poseban je za sam Perl, a iz njega se može vidjeti utjecaj prirodnog jezika. Nema nekakav poseban naziv, pa ga je najlakše prikazati u konkretnom obliku [11]:

```
METODA POZIVATELJ (LISTA)
METODA POZIVATELJ LISTA
METODA POZIVATELJ
```

Primjer pozivanja metoda ovim načinom [11]:

```
$mage = summon Wizard "Gandalf";
speak $mage "You cannot pass";
```

U ovom primjeru *Wizard* je naziv klase, a *summon* je metoda te klase koja vraća objekt, točnije čarobnjaka pod imenom „*Gandalf“*. *Speak* je metoda koju može pozvati objekt tipa *Wizard*, a „*You cannot pass!*“ je argument te metode.

2.5. JavaScript

Današnji web bio bi nezamisliv da ne postoji JavaScript. JavaScript je izrazito popularan skriptni jezik primarno korišten u svrhe web razvoja. Njegova popularnost može se direktno povezati s omjerom jednostavnosti korištenja i općenitom snagom. „JavaScript je bitan jezik jer je on jezik web preglednika.“ [12, str. 2] Usprkos tomu, percepcija JavaScripta-a od strane javnosti je izrazito negativna. Pretraživanjem bilo kojeg programerskog foruma jako brzo se nađe objava o tomu kako je JavaScript loš jezik. Ponekad je to samo u šali, ali velika količina programera jednostavno ne voli raditi u JavaScriptu. Možda je to slučaj jer su programeri „prisiljeni“ koristiti JavaScript ukoliko žele programirati bilo što u vezi weba. Učiti novi jezik nije uvijek praktično, pogotovo ako je potrebno napisati program manjeg opsega, a web stranice obično nemaju velik opseg. Bitno je napomenuti da „web stranica“ i „web aplikacija“ nisu iste stvari – opseg web aplikacija dakako može biti ogroman. JavaScript omogućuje da se uz minimalnu količinu programskog koda postignu dosta korisne funkcionalnosti. Pomoću *Document Object Model* (DOM), koji je nemoguće izbjeći kod korištenja JavaScript-a, može se pristupiti bilo kojem objektu, odnosno elementu, web stranice. Pristupanjem elementa omogućuje se i manipulacija svih svojstava tog elementa. Postoji mnogo ovakvih pomoćnih alata za razvoj koji su direktno ugrađeni u JavaScript. Ovo je samo jedan primjer, no ide u korist argumentu kako je JavaScript dobro podržan, lagan za korištenje, no najbitnije, dobar jezik. Crockford [12] piše kako je pomoću JavaScripta moguće napraviti puno toga bez da se dobro poznaje sam jezik, a jezik postaje još bolji kada se zna što se radi.

JavaScript nastao je 1995. godine. Originalna namjena bila je da se u *Netscape Navigator* web preglednik dodaju korisni programi kako bi se nadogradila funkcionalnost. Rastom popularnosti JavaScripta-a sve više web preglednika počelo ga je podržavati. Svaki web preglednik mogao je napraviti svoju interpretaciju jezika; nije postojao nikakav standard. To se promijenilo nastankom dokumenta zvanog „ECMAScript“. ECMAScript opisuje kako se neki standardni web programski jezik (JavaScript) treba ponašati. Osim toga, određuje koje nove funkcionalnost jezika ulaze u službeno izdanje. Nažalost, JavaScript se još uvijek ne ponaša identično na svim web preglednicima. Istina je da su to minimalne razlike, ali nije prikladno da postoji standard i da se jezik još uvijek ne ponaša jednako na svim preglednicima.

Kao i kod bilo kojeg drugog jezika, postoji više verzija JavaScripta. Između 2000. i 2010. godine najaktualnija verzija bila je ECMAScript 3, a u razvoju je bila verzija 4 s mnogim poboljšanjima i nadogradnjama jezika. Nažalost, razvoj verzije 4 bio je prekinut, a umjesto nje 2009. godine izašla je malo manje ambiciozna verzija 5. 2015. godine nastao je ECMAScript 6. Verzija 6 implementirala je mnoge ideje koje su prvotno bile namijenjene za verziju 4. [13]

2.5.1. Svojstva

Budući da je JavaScript skriptni jezik, on dijeli mnoga svojstva s ostalim skriptnim jezicima. Od svih prije navedenih jezika, najsličniji je standardnom programskom jeziku. Naime, mnogo svojih ponašanja i svojstava „posudio“ je od jezika poput C i C++. Iako se u imenu nalazi „Java“, JavaScript nema nikakve veze s jezikom Java. Glavna stvar do koje se opet dolazi je to da JavaScript, kao i ostali skriptni jezici, nema deklaracije tipa podatka. JavaScript omogućuje rad sa standardnim tipovima podatka poput brojeva, teksta, polja i objekta. Po čemu se razliku od ostalih, ne samo skriptnih, jezika je po svojim „posebnim“ tipovima podatka.

Brojevi mogu biti cijeli ili racionalni, no neovisno o tome JavaScript zauzima 64 bita u memoriji za jedan broj. Što se tiče prikaza cijelih brojeva, 64 bita trebala bi biti dovoljna za bilo koji problem. Ipak, 64 bita znači da se sveukupno može prikazati 2^{64} brojeva, što je otprilike jednako $1.845 * 10^{19}$ – definitivno dovoljno brojeva. Usprkos tomu, može se pojaviti problem kod prikaza beskonačnih brojeva poput π (pi) ili φ (fi). Kao pomoć kod prikaza izrazito malih ili izrazito velikih brojeva, JavaScript omogućuje znanstvenu notaciju za brojeve. Primjerice, prije navedeni broj 2^{64} mogao bi se prikazati kao 1.845e19. Postoji jedna posebna vrijednost koja spada pod brojčane vrijednosti, no označuje upravo suprotno od bilo kojeg broja. Vrijednost se prikazuje kao „NaN“, što se skraćeno za „*Not a Number*“, odnosno nije broj. Ovu vrijednost moguće je dobiti nelegitimnim matematičkim operacijama, poput dijeljenja nulom.

Tekst može biti bilo što pod uvjetom da se nalazi unutar jednostrukih ili dvostrukih navodnika. Tekst u JavaScriptu kodiran je u „Unicode“ standardu. Problem je to što JavaScript koristi 16 bita za prikaz jednog tekstualnog znaka, što nije dovoljno za prikaz svih Unicode znakova – njih je dvostruko više; neki posebni znakovi su zato prikazani koristeći dva „mjesta“ [13]. Za razliku od PHP-a i Perl-a, JavaScript interpretira tekst jednako neovisno o tome nalazi li se unutar jednostrukih ili dvostrukih navodnika što znači da JavaScript ne podržava interpolaciju varijabli, barem ne bez posebnih znakova. Postoji treći način zapisivanja teksta, a to je pomoću otvorenih jednostrukih navodnika (` `). Tekst zapisan pomoću otvorenih jednostrukih navodnika podržava interpolaciju varijabli i obrnute kose crte, a sve ostalo shvaća se doslovno kako je napisano što znači da se prelazak u novi red može napraviti jednostavnim prekidom reda. Interpolacija varijabli radi se pisanjem „$“ znaka prije naziva varijable i pisanjem dobivenog izraza između vitičastih zagrada.

Primjer različitih prikaza teksta:

```
var tekst1 = "Ovaj tekst nalazi se u jednom retku.";
var tekst2 = 'Ovaj tekst nalazi \n se u dva retka';
var tekst3 = `Ovaj tekst
           nalazi se
           u tri retka.`;
```

U primjeru nalaze se tri različita teksta. Prvi tekst pisan je unutar dvostrukih navodnika i nalazi se u jednom retku. Drugi tekst pisan je unutar jednostrukih navodnika i korištena je interpolacija obrnute kose crte kako bi se tekst razbio i prikazao u dva retka. Bitno je napomenuti da bi ovo funkcioniralo i s dvostrukim navodnicima. Zadnji tekst napisan je unutar otvorenih jednostrukih navodnika i pomoću tipke „enter“ tekst je razbijen u tri retka.

Primjer interpolacije varijabli:

```
var ime = "Mihael";
var godina = 1997;
var tekst1 = "Ime autora ovog rada je " + ime + ", a rođen je " + godina
            + ". Godine.";
var tekst2 = `Ime autora ovog rada je ${ime}, a rođen je
            ${godina}.Godine`;
```

U ovom primjeru prikazano je kako standardan tekst pisan dvostrukim navodnicima ne podržava interpolaciju varijabli i potrebno je operatorom nadovezivanja, odnosno konkatenacije (concatenation), spojiti tekst i varijable. Isto tako bi funkcioniralo da su se koristili jednostruki navodnici. Drugi tekst pisan je između otvorenih jednostrukih navodnika i nije potrebno operatorom spajati tekst i varijable – varijable se nalaze direktno u tekstu, između navodnika.

U JavaScript-u postoje dva tipa podatka koji reprezentiraju prazninu, odnosno manjak konkretne vrijednosti. Ta dva tipa zovu se *null* i *undefined*. Oni sami po sebi nisu jednaki, no u većini slučajeva može se pretpostaviti da predstavljaju istu stvar. Zapravo, oni bi trebali biti jednaki, no zbog greške u dizajnu JavaScript-a nisu. [13]

Već je prije bilo navedeno kako JavaScript ne podržava deklariranje tipa podatka varijable. Postoje tri ključne riječi koje se koriste kako bi se definirala varijabla, a to su: *var*, *let* i *const*. Kao prvo, *const* se, kako mu i naziv govori, ne koristi za definiranje varijable, već konstante, no ta dva koncepta su gotovo identična u programskom svijetu. Jedina razlika je to što se konstanta mora inicijalizirati i dalje joj se vrijednost ne smije mijenjati. Ključnim riječima *var* i *let* definiraju se varijable, no očito se po nečemu razlikuju. Glavna riječ ovdje je djelokrug (*scope*). Varijable definirane pomoću *var* ključne riječi imaju globalan djelokrug, što znači da im se može pristupiti iz bilo kojeg dijela skripte ili klase. S druge strane, varijable definirane ključnom riječju *let* imaju ograničen djelokrug, odnosno može im se pristupiti samo unutar istog bloka koda.

2.5.2. Document Object Model

Jedan od najčešće korištenih tehnologija uz JavaScript je *Document Object Model* (DOM). DOM je zapravo sučelje za programiranje koje se koristi za pristup i manipulaciju

elemenata HTML i XML dokumenata. Dio je W3C specifikacije, tako da osigurava standard za web programiranje. Budući da je DOM sučelje za programiranje, moguće ga je koristiti s bilo kojim programskim jezikom. [14]

HTML i XML dokumenti podijeljeni su u pojedine elemente. Ti elementi organizirani su u stablastu strukturu gdje je korijenski element sam HTML ili XML dokument. Svaki HTML dokument, primjerice, mora u sebi sadržavati elemente pod nazivom „html“, „head“ i „body“ kako bi se korektno mogao prikazati na bilo kojem web pregledniku. Svaki element može, ali i ne mora imati jedno ili više djece. Primjerice, svaki odjeljak HTML dokumenta jedan je element, a pomoću DOM sučelja moguće mu je s lakoćom pristupiti i promijeniti bilo koje svojstvo, pa čak i sam sadržaj. Ako odjeljak u sebi ima element poveznice, tom elementu također je moguće pristupiti i mijenjati ga istim postupkom.

Jedan od najčešćih načina pristupa nekom elementu HTML web stranice jest putem njegovog jedinstvenog identifikatora. Da to funkcionira korektno, svaki identifikator mora biti jedinstven – inače DOM ne zna na koji element se misli. Da bi se na ovaj način pristupilo elementu, postoji metoda objekta „Document“ pod nazivom „getElementById“. Metoda prima jedan argument (identifikator elementa), a vraća referencu na taj element u obliku objekta. Još neke od korisnih metoda istog objekta su „getElementsByClassName“ i „getElementsByTagName“. Iz naziva se vidi da metode obično vraćaju više elementa u obliku liste, ovisno o nazivu njihove klase ili o konkretnoj vrsti elementa. Kombinacijom ovih metoda i mogućnošću da HTML elementi posjeduju više klasa mogu se postići dosta jake i kompleksne operacije u nekoliko linija koda.

Primjer pristupu i manipulaciji elemenata pomoću DOM sučelja:

```
var sviOdjeljci = Document.getElementsByTagName('p');
var sviBlokovi = Document.getElementsByClassName('blok');

for(var i = 0; i < sviOdjeljci.length; i++){
      let redniBroj = i + 1;
      sviOdjeljci[i].textContent += redniBroj;
}

for(var j = 0; i < sviBlokovi.length; j++){
      sviBlokovi[j].style.backgroundColor = „blue“;
}
```

U primjeru prvo se dohvaćaju svi odjeljci i svi elementi s klasom „blok“ HTML dokumenta. Zatim, pomoću dvije petlje iterira se kroz obje liste elemenata. Prva petlja odjeljcima na kraj dodaje redni broj tog odjeljka. Druga petlja svim elementima s klasom „blok“ mijenja pozadinsku boju u plavu.

3. Web aplikacije

U ovom poglavlju napravit će se pregled JavaScript web aplikacija. Od njihovih svojstava, najčešće korištenih tehnologija pa do modernih trendova u razvoju web aplikacija.

Moglo bi se reći da je razvoj JavaScript web aplikacija započeo implementacijom JavaScript-a u Netspace web pregledniku. Doduše, tadašnje aplikacije ne bi se baš mogle nazvati aplikacijama, barem ne u rangu današnjih. Prije standardizacije JavaScript-a, razvoj web aplikacija jednostavno nije imao prostora za napredak i poboljšanje. Iako su to bili manji programi, svatko je JavaScript koristio na svoj način, a web preglednici također nisu imali jedinstvenu implementaciju. Standardizacijom jezika ECMAScript-om jezik je konačno mogao konkretno napredovati. Danas se koriste različiti JavaScript interpreteri, neki od kojih su: „Chrome V8“ kojeg koristi Google Chrome, „SpiderMonkey“ kojeg koristi Firefox i „Chakra“ kojeg koristi Microsoft Egde. Ti interpreteri omogućili su JavaScript-u da se odvija brže i efikasnije, a razvojnim programerima da implementiraju najnovije programske koncepte i funkcionalnosti. MacCaw [15] piše kako su mogućnosti danas bile nezamislive ranim razvojnim programerima.

Moderne web aplikacije dosegle su stupanj takav da ih je ponekad teško raspoznati od klasičnih, izvornih aplikacija. Sa servisima poput Google-ovog Gmail-a i YouTube-a granica između izvorne i web aplikacije praktički i ne postoji. MacCaw [15] navodi da su web preglednici postali toliko snažni da je razvoj velikih JavaScript web aplikacija postao izvodljiv, ali i sve popularniji.

Jedna od negativnih točaka web aplikacija je potreba za konstantnom internetskom vezom. Ponekad se čovjek nađe u situaciji kada jednostavno nema pristup internetskoj vezi, a potrebne su mu određene informacije. S današnjom infrastrukturom i mobilnim tehnologijama ovaj problem polako prestaje postojati. Također, danas su sve popularnije progresivne web aplikacije koje mogu, uz određena ograničenja, funkcionirati i bez internetske veze.

3.1. Svojstva

Da bi aplikacija bila web aplikacija mora zadovoljiti jedan očiti uvjet – mora biti moguće pokrenuti ju, ali i izvoditi, putem web preglednika. Usprkos tomu današnje web aplikacije sadrže svojstva koja bi se prije nalazila isključivo u velikim izvornim aplikacijama. Dalje navedena svojstva nisu nužno isključivo vezana uz moderne web aplikacije, ona se mogu nalaziti i u standardnim aplikacijama – to je samo dokaz kako razlike između standardnih i web aplikacija prestaju postojati.

Čovjek veliku većinu informacija prima putem vida. Zato nije ni čudno da se moderne web aplikacija toliko fokusiraju na grafički dizajn. Ljudi obično na prvi pogled ne vide sve funkcionalnosti web aplikacije ili koliko je ona efikasna što se tiče upravljanja dodijeljenom memorijom, već koliko su njene boje usklađene ili koliko su korištene animacije ugodne za gledati. Dobrim i ukusnim grafičkim dizajnom moguće je prosječnu web aplikaciju podići iznad ostalih po popularnosti. Dalia [16] navodi kako dobar grafički dizajn ostavlja trajni dojam, pomaže kod izgradnje identiteta brenda i kod prenošenja informacija korisnicima.

Današnji web izvodi se na mnogo različitih tipova uređaja. Rastom popularnosti mobilnih uređaja poput pametnih telefona, tableta i prijenosnih računala sve manjih dimenzija, potrebno je da web aplikacije budu prilagođene svakom od njih. Rješenje ovog problema nalazi se u odzivu (*responsiveness*) web aplikacija. Ovo svojstvo omogućuje da se ista web aplikacija na korektan način prikazuje na uređajima različitih dimenzija.

Iako to nisu stvari koje u početku zvuče bitne, a pogotovo ne primamljive, održivost i skalabilnost su jedne od najbitnijih svojstava modernih web aplikacija. Činjenica je da se moderan web iz godine u godinu sve više mijenja. Kako bi web aplikacija mogle pratiti tok promjena moraju biti održive, odnosno potrebno je moći s lakoćom promijeniti neki dio aplikacije kako bi zadovoljio nove zahtjeve od korisnika ili kako bi implementirao najnovije programske koncepte i trendove u svoj sustav. Također, svaki web razvojni programer želi da njegova aplikacija uspije u tržištu. Rastom uspješnosti web aplikacije raste i njezina popularnost, odnosno raste broj jedinstvenih korisnika koji koristi aplikaciju. Zato je bitno da aplikacija bude skalabilna; mora moći podržati sve više rastući broj njenih korisnika.

Struktura je definitivno jedno od definirajućih svojstava modernih web aplikacija. Kako bi web aplikacija bila skalabilna i održiva potrebno je na korektan način podijeliti aplikaciju na određene sustave. Skalabilnost i održivost obično nisu na umu web razvojnim programerima kada prvi puta razvijaju web aplikaciju, no ignoriranjem tih bitnih aspekata web aplikacije jedino što kasnije može nastati je frustracija i u konačnici prepravljanje velikog dijela, ako ne i cijele, web aplikacije. MacCaw [15] piše kako je kod izrade velike JavaScript web aplikacije potrebno u početku raspodijeliti aplikaciju na što više međusobno neovisnih dijelova, odnosno modula.

3.2. Popularne web tehnologije

Sve web aplikacije temelje se na tri tehnologije. Mogli bi ih nazvati temeljem cijelog današnjeg weba; one čine tri različita sloja modernih web aplikacija: struktura, dizajn, ponašanje. Struktura web aplikacije određena je HTML jezikom. Dizajn, odnosno prezentacija, web aplikacije određena je CSS-om, kaskadnim stilskim uputama. Ponašanje web aplikacije određeno je već poznatim JavaScript jezikom. Moglo bi se reći da su sve ove tehnologije dio takozvanog „front-end“ web razvoja, odnosno programiranja na strani korisnika.

Naravno, za ozbiljnu web aplikaciju nije dovoljno napraviti samo sučelje koje korisnik vidi – potrebne su dodatne tehnologije koje omogućuju komunikaciju aplikacije s bazom podataka i sličnim sustavima. Tehnologije koje se bave time dio su takozvanog back-end razvoja, odnosno programiranja na strani poslužitelja. Spajanjem ta dva kraja dobiva se moderna web aplikacija.

Obično se programiranje na strani poslužitelja radi s nekim jezikom koji nije JavaScript, primjerice PHP-om ili C#-om. JavaScript je obično programski jezik koji se izvodi isključivo u web pregledniku. U zadnje vrijeme to se sve više mijenja, a glavni krivac tomu je Node.js. Node.js program za izvađanje JavaScript programskog koda izvan web preglednika, a temelji se na asinkronu izvađanju koda i događajima (*event driven*). Pomoću Node.js moguće je JavaScript pretvoriti u nešto što više sliči standardnom programskom jeziku. Glavna stvar oko Node.js su moduli. Moduli su nešto poput biblioteka ili klasa koje se ubacuju u postojeći JavaScript kod kako bi povećali funkcionalnost, slično kao što se radi i u klasičnim programskim jezicima. Prema [17] nemoguće je za Node.js da uđe u stanje deadlock-a zato što gotovo niti jedna funkcija direktno ne obrađuje ulazno-izlazne operacije, a to čini Node.js vrlo pogodnim alatom za izradu skalabilnih aplikacija.

Sadržaj web stranice obično je teško promijeniti bez da se putem web preglednika osvježi sama web stranica. Kao i kod ostalih stvari, rješenje ovog problema nalazi se u JavaScript-u. JavaScript može koristiti skup metoda, odnosno tehnika, koje se skupno nazivaju „AJAX“ (asynchronous JavaScript and XML). Iz naziva se mogu vidjeti dvije tehnologije koje se ovdje koriste: JavaScript programski jezik i XML tip datoteke. Umjesto XML datoteke moguće je koristiti i JSON (*JavaScript Object Notation*) datoteke, što je nekim programerima pogodnije. AJAX omogućuje razvojnom programeru da putem nekolicine metoda promijeni sadržaj web stranice bez da osvježi samu stranicu, što uvelike povećava korisničko iskustvo same web aplikacije. Putem JavaScript-a moguće je na poslužitelj poslati zahtjev na koji onda poslužitelj odgovara i šalje datoteku s potrebnim podacima. Nakon toga, sadržaj dobivene datoteke može se formatirati i prikazati korisniku na web stranici - bez da se na trenutak zamrzne sučelje ili osvježi web stranica.

4. JavaScript biblioteke i okviri

Iako je JavaScript sam po sebi savršeno dovoljan za izradu jednostavnih i složenih web aplikacija, današnji standardi tržišta web aplikacija zahtijevaju što veću brzinu razvoja uz odgovarajuću razinu kvalitete samog proizvoda. Osim toga, mnogo web aplikacija dijeli veliku količinu zajedničkih svojstava. Primjerice, gotovo svaka aplikacija u nekom obliku koristi bazu podataka. Činjenica je da se programski kod izrazito često ponavlja, čak uspoređujući dvije potpuno različite web aplikacije. Iz tih razloga nastali su razni okviri i biblioteke koji na više načina olakšavaju i ubrzavaju razvoj web aplikacija. Neki okviri i biblioteke specijalizirani su za određene svrhe, dok se ostali mogu primijeniti gotovo u svakom slučaju. Primjer JavaScript biblioteke koja se i danas koristi u mnogim web aplikacijama je jQuery. jQuery je definitivno jedna od, ako ne i najpopularnija JavaScript biblioteka ikad. Ljudi obično ne znaju razlike između biblioteke i okvira pa često koriste te riječi kao da su sinonimi. Stoga bi bilo bitno napomenuti razlike između biblioteke i okvira. Morris [18] navodi dobru analogiju za razliku između JavaScript biblioteka i okvira: biblioteke su poput namještaja – on dodaje funkciju, ali i stil u već postojeću kuću, dok su okviri poput nacrta i alata koji služe za izgradnju same kuće. Dakle, JavaScript biblioteka zapravo je dodatak, odnosno proširenje na već postojeće funkcije i načine razvoja web aplikacija JavaScript-om. Stvar je u tome da se bilo koja biblioteka može dodati na bilo koji, novi ili već postojeći, JavaScript kod; uvijek će na isti način unaprijediti JavaScript aplikaciju istim funkcijama. Naravno, moguće je da biblioteka ovisi o nekim drugim bibliotekama kako bi korektno funkcionirala. S druge strane, JavaScript okvir ne može se, barem ne na jednostavan način, dodati u već postojeću JavaScript aplikaciju. U pravilu je potrebno da se razvoj čitave aplikacije započne korištenjem odabranog okvira. JavaScript okvir na fundamentalan način mijenja način razvoja web aplikacije - obično pomoću integracije novih i, za taj okvir, posebnih koncepata i pravila. Osnovni cilj svakog JavaScript okvira je što brži i efikasniji razvoj JavaScript aplikacija. Novi okvir potrebno je naučiti koristiti nalik novom programskom jeziku. Jednostavno se toliko puno razlikuje od standardnog JavaScript-a. JavaScript okviri mogli bi se podijeliti na više načina, no jedna od glavnih podjela je ta na koju „stranu“ web programiranja se odnosi određen okvir. Pod „stranu“ misli se na podjelu web programiranja na strani poslužitelja i web programiranja na strani klijenta. Neki okviri, poput „Vue“ i „Angular“, namijenjeni su za izradu dinamičkog grafičkog sučelja web aplikacije koje korisnik direktno vidi i s kojim je u neposrednoj interakciji. Neki drugi okviri, poput express-a, namijenjeni su, između ostalog, za razvoj sučelja za programiranje aplikacija (*API*) na strani poslužitelja i za rukovanje HTTP zahtjevima. Što se tiče fleksibilnosti, mogli bi se reći da su biblioteke fleksibilnije iz razloga što se mogu integrirati u bilo koju aplikaciju i što se mogu koristiti u skladu s bilo kojim drugim bibliotekama. Osim toga, popularnije biblioteke, poput

jQuery, imaju i dodatke koji proširuju funkcionalnosti same biblioteke. S druge strane, okviri su obično zatvoreni za istovremeni rad s drugim okvirima. Osim toga, neke biblioteke se ne mogu uključiti u aplikacije razvijene određenim okvirima. Iznimke postoje u slučaju da je određena biblioteka razvijena s određenim okvirom na umu.

Tablica 1: Usporedba JavaScript biblioteka i okvira

Kategorija	Biblioteka	Okvir
Cijena integracije	Minimalna, ili nepostojeća. Biblioteke se, u pravilu, mogu umetnuti u bilo koju, novu ili već postojeću, JavaScript aplikaciju. Iako biblioteka možda sadrži mnogo funkcija, nije nužno koristiti svaku od njih kako bi se aplikacija nadogradila željenim funkcionalnostima.	Okviri obično imaju puno veću cijenu integracije. Obično je potrebno čitavu aplikaciju prilagoditi odabranom okviru. U nekim slučajevima potrebno je aplikaciju iz početka razviti u odabranom okviru. Potrebno je koristiti većinu ponuđenih klasa i funkcija za korektnu integraciju.
Kontrola	Kod korištenja biblioteke, korisnik, odnosno razvojni programer, je pod kontrolom. Korisnik poziva ponuđene funkcije biblioteke i na taj način popunjava svoju JavaScript aplikaciju dodatnim funkcionalnostima.	Korištenjem okvira razvojni programer daje kontrolu odabranom okviru. Okvir razvojnom programeru nudi šablonu za rad. Od razvojnog programera zahtijeva se da popuni tu šablonu, dok okvir sam odrađuje ostatak posla.
Lakoća korištenja	Biblioteke su, u pravilu, lakše za koristiti. Iako postoje izuzetci, biblioteka se koristi na način da razvojni programer koristi unaprijed definirane funkcije i klase. Obično nema potrebe za učenjem novih koncepata.	U prosjeku, okvir je teže za koristiti. Okviri, kako bi se korektno koristili, zahtijevaju učenje njihove arhitekture i novih koncepata. Potrebno je da razvojni programer nauči na efikasan način koristiti alate koji donosi odabrani okvir.

Utjecaj na razvoj aplikacije	Kako se kod biblioteka obično radi o nadopunjavanju aplikacije novim funkcionalnostima, biblioteka, sama po sebi, nema velik utjecaj na način razvoja aplikacije. Neke biblioteke zahtijevaju nekakvo početno postavljanje prije samog korištenja, no obično u malim količinama.	Utjecaj okvira na razvoj aplikacije je maksimalan. Naravno, količina utjecaja ovisi o odabranom okviru, no kod korištenja bilo kojeg okvira potrebno je poštivati postavljena pravila korištenja okvira, poput strukture direktorija. Uz to potrebno je naučiti i korektno koristiti arhitekturu i koncepte koje postavlja sam okvir.
Fleksibilnost	Veći broj biblioteka se obično može koristiti u tandemu. Isto tako, biblioteka se može priključiti na neke okvire. Postoje i biblioteke koje su napisane isključivo za korištenje sa specifičnim okvirom. Iako postoje izuzetci, fleksibilnost biblioteka je izrazito velika.	Okviri su po svojoj prirodi zatvoreni, odnosno nisu pogodni za korištenje s drugim okvirima ili bibliotekama. Postoje biblioteke koje nadopunjuju određene okvire dodatnim, često potrebnim i korištenim, funkcionalnostima. Fleksibilnost okvira dosta je mala.

4.1. React

React, ili ReactJS, jedna je od najpopularnijih JavaScript biblioteka za izgradnju grafičkih sučelja. Nastao je 2011. godine, a tvorac mu je Jordan Walke – programski inženjer kod Facebook-a. React je nastao pod utjecajem XHP-a, okvira za PHP koji se temelji na HTML komponentama, i da je 2013. godine otvoren za javnost [19]. Iako se to možda na prvi pogled čini, React nije okvir. Shvatljivo je zašto se događa zabuna – React uz sebe obično vuče koncepte i načine razvoja aplikacije. Usprkos tome, web programer kod korištenja React biblioteke slobodan je u izboru načina razvoja aplikacije.

React se temelji na nekoliko glavnih koncepata kako bi efikasno mogao obaviti izgradnju korisničkog sučelja. Najbitniji od njih jesu komponente. Komponente su u biti HTML elementi, no React omogućuje neka dodatna svojstva za bolju manipulaciju tih elemenata.

Komponenta može biti napisana kao JavaScript funkcija ili kao JavaScript klasa. Jöch [20] piše kako u trenutnoj inačici React biblioteke nema bitnijih razlika između ta dva načina pisanja komponenti, no razvojni programeri React-a navode kako bi u idućim inačicama React-a komponente pisane kao funkcije mogle imati sveukupno bolje performanse. Osim toga, svaka React komponenta u sebi sadrži objekt pod nazivom „*props*“ koji posjeduje sva svojstva (*property*), odnosno atribute te komponente.

Primjer jednostavne React komponente u obliku funkcije:

```
function Pozdrav(props){
      return <div className='pozdrav'>Pozdrav {props.ime}</div>;
}
//-------------------------------------------------------------
//Korištenje komponente:
<div>
      <Pozdrav ime='Mihael' />
</div>
```

Komponenta „Pozdrav“ ispisuje HTML element s tekstom pozdrava nekoj osobi. Iz primjera se može vidjeti na koji način se koristi objekt „*props*“. Unutar funkcije komponente svojstva se dohvaćaju kao i kod bilo kojeg drugog JavaScript objekta. Svojstva se dodaju komponenti na identičan način kao što se dodaju atribute standardnim HTML elemenitma. Jedina razlika ovdje je što se koristi drugačija notacija za nazive svojstava. Sve što je potrebno napraviti kako bi se komponenta koristila u aplikaciji je to da se napiše element čiji naziv je naziv funkcije koja opisuje komponentu. U ovom primjeru to je komponenta „Pozdrav“, kojoj se, između ostalog, dodao i atribut „ime“ s vrijednošću „Mihael“. Komponente se naravno ne moraju sastojati od samo jednog HTML elementa, no nužan uvjet je da svaka komponenta ima jedan korijenski HTML element koji onda sadrži sve ostale elemente komponente. Osim toga, komponenta kao funkcija može primjerice obraditi dobivene podatke i vratiti drugačiju komponentu ovisno o tim podacima.

Primjer malo složenije React komponente:

```
function Kalkulator(props){
      if(props.operacija === '+'){
            return (
                  <div>
                  {parseInt(props.prviBroj)                    +
                  parseInt(props.drugiBroj)}
                  </div>
            )
      }
      else if(props.operacija === '-'){
            return (
                  <div>
                  {parseInt(props.prviBroj)                    -
                  parseInt(props.drugiBroj)}
                  </div>
            )

      }
}
```

```
//-------------------------------------------------------
//Korištenje komponente:
<div>
      <Kalkulator prviBroj=2 drugiBroj=4 operacija='+' />
</div>
```

Komponenta „Kalkulator“ prima tri svojstva: „prviBroj“, „drugiBroj“ i „operacija“ i vraća HTML element u kojem se nalazi zbroj brojeva ako je operacija jednaka „+“, odnosno razliku brojeva ako je operacija jednaka „-“. Konkretno u primjeru, komponenta ispisuje broj 6. Dakle, funkcija komponente opisuje kako se komponenta treba prikazati na aplikaciji.

Dok su komponente pisane kao funkcije dovoljne za neke jednostavnije komponente aplikacije, ponekad je poželjno nad komponentom imati više kontrole. Komponente pisane kao JavaScript klase omogućuju neke dodatne React koncepte koji upravo donose dodatnu kontrolu. Jedan od bitnijih koncepata je praćenje stanja komponente. Praćenje stanja komponente omogućuje to da komponenta ne ovisi o drugoj komponenti i da komponenta sama sebe mijenja. Svaka komponenta pisana kao klasa, osim objekta „*props*“, u sebi sadrži i objekt „*state*“ koji opisuje njeno stanje. Osim toga, svaka takva komponenta mora zadovoljavati neke uvjete. Mora imati metodu pod nazivom „*render*“ koja, poput funkcije komponente, opisuje kako se komponenta treba prikazati unutar aplikacije. Osim toga, komponenta, kako bi bila valjana komponenta, mora naslijediti od klase „*React.Component*“. Jako bitno je znati na koji način React koristi metodu „*render*“ svake klase. Metoda „*render*“ poziva se kod samog stvaranja komponente, ali i kod svake promjene svojstava ili stanja same komponente. To znači da je dovoljno da se, primjerice, promijeni stanje komponente kako bi se sve promjene mogle vidjeti u samoj aplikaciji. Kao i kod drugih objektno orijentiranih programskim jezika, svaka klasa u React biblioteci ima konstruktor koji se poziva kod stvaranja komponente u aplikaciji. Konstruktor je mjesto gdje se inicijalizira objekt stanja, odnosno „*state*“.

Primjer jednostavne React komponente u obliku klase:

```
class Poruka extends React.Component {
      constructor(props){
            super(props);

            this.state = {
                  vidljivo: true
            }

            this.Sakrij();
      }

      render(){
            if(this.state.vidljivo){
                  return <div>{this.props.tekst}</div>;
            }
            else return null;
      }
```

```
	Sakrij(){
		setTimeout(() => {
			this.setState({vidljivo: false});
		}, 2000);
	}
}
//-----------------------------------------------
//Korištenje komponente
<div><Poruka tekst='Pozdrav!' /></div>
```

Klasa „Poruka“ iz primjera sadrži tri metode, jedna od kojih je konstruktor. Svaki konstruktor, ukoliko je definiran, mora kao argument primati „*props*“ i u sebi sadržavati „*super(props)*“ kako bi se proslijeđena svojstva mogla korektno koristiti. Konstruktor u primjeru postavlja objekt stanja i dodaje mu svojstvo „vidljivo“, te postavlja vrijednost tog svojstva na istina, odnosno „*true*“. Nakon toga, konstruktor poziva metodu „Sakrij“, koja nakon dvije sekunde poziva metodu „*setState*“ kako bi postavila svojstvo „vidljivo“ objekta stanja na neistina, odnosno „*false*“. Promjena stanja okida ponovno pozivanje metode „*render*“ što znači da će metoda vratiti „*null*“ i komponenta se više neće prikazati.

React komponente, bile one funkcije ili klase, mogu u sebi sadržavati ili pozivati, svoje ili tuđe, rukovatelje događaja. Komponente funkcije ne mogu imati svoje rukovatelje događaja, no mogu, preko svojstava pozivati rukovatelje svojih roditelja. Stoga vrijedi da samo komponente klase mogu imati svoje rukovatelje događaja. Prijašnji primjer mogao bi se promijeniti tako da se vidljivost poruke promijeni svaki puta kad korisnik klikne na gumb.

```
class Poruka extends React.Component {
	constructor(props){
		super(props);

		this.state = {
			vidljivo: true
		}

		This.PodesiVidljivost = this.podesiVidljivost.bind(this);
	}

	render(){
		if(this.state.vidljivo){
			return(
			<div>
				{this.props.tekst}
				<button onClick={this.PodesiVidljivost}>
					Klikni me!
				</button>
			</div>);
		}
		else return(
		<div>
			<button onClick={this.PodesiVidljivost}>
			Klikni me!
			</button>
		</div>);
	}

	PodesiVidljivost(){
```

```
            this.setState({vidljivo: !this.state.vidljivo});
        }
}
//---------------------------------------------
//Korištenje komponente
<div><Poruka tekst='Pozdrav!' /></div>
```

Dakle, u ovom slučaju vidljivost poruke podešava se klikom na gumb, koji okida metodu „PodesiVidljivost“ koja mijenja stanje komponente. Bitno je primijetiti jednu liniju koda u konstruktoru, a to je poziv metode „*bind“* nad metodom klase „PodesiVidljivost“. Taj poziv potrebno je napraviti kada god se neka metoda okida na neki događaj, ili kad se metoda želi proslijediti drugoj komponenti.

Jedna zanimljiva sintaksa koja se javlja, no nije obavezna, kod korištenja React biblioteke je „*JSX“*. Razvojni programeri React biblioteke osmislili su tu sintaksu kako bi što lakše bilo konstantno pisati HTML sintaksu unutar JavaScript koda [21]. Primjer te sintakse može se vidjeti u svakom primjeru unutar svake „*render*“ metode. Ta sintaksa ne predstavlja niti tekst niti konkretan HTML, već ga React biblioteka na odgovarajući način pretvara u HTML elemente kod korištenja aplikacije. Korisna stvar kod te sintakse je što se bilo kakav JavaScript kod može integrirati unutar HTML-a. U primjerima se obično dohvaća neko svojstvo komponente kako bi se ono prikazalo unutar same komponente, no moguće je i napraviti pozive metoda – React biblioteka prikazat će vraćenu vrijednost.

Jedan od najpopularnijih dodataka za React je React Router. React Router zapravo je skup navigacijskih komponenata koje se mogu koristiti u React aplikaciji. Te komponente omogućuju takozvano dinamičko usmjeravanje. To znači da kad korisnik klikne, primjerice, neku poveznicu na web aplikaciji, aplikacija osvježi samo predodređen dio svojeg prikaza s novim sadržajem. To sprječava nepotrebno gubljenje vremena čekajući učitavanja resursa ili osvježavanje učitane stranice, za čije vrijeme korisničko sučelje ne reagira. Glavna korist uporabe ovog dodatka je olakšana mogućnost izrade jednostraničnih web aplikacija – web aplikacija koje ne zahtijevaju osvježavanje prikaza web preglednika, odnosno odvijaju se u potpunosti na jednoj stranici. Maki [22] navodi kako je dinamičko usmjeravanje korisničke strane web aplikacije sve popularniji način razvoja besprijekornog korisničkog iskustva. Dvije glavne komponente uključene u React Router su: „*BrowserRouter*“, „*Link*“ i „*Route*“. Komponentu BrowserRouter je potrebno koristiti kao korijenski element dijela aplikacije koji želi koristiti funkcionalnosti koje nudi React Router. Komponenta Link ima sličnu funkciju kao i element poveznice u HTML-u *(<a>*), no ona ne vodi na eksterna odredišta, već mijenja trenutno aktivnu putanju. Komponenta Route prikazuje određenu komponentu ukoliko je njezina putanja „pogođena“, odnosno web preglednik se nalazi na njenoj putanji.

Primjer jednostavne React aplikacije koja koristi React Router:

```
<BrowserRouter>
<div>
```

```
            <ul>
                  <li>
                        <Link to='/profesori/'>Profesori</Link>
                  </li>
                  <li>
                        <Link to='/studenti/'>Studenti</Link>
                  </li>
            </ul>

            <Route path='/profesori/' component={ListaProfesora} />
            <Route path='/studenti/' component={ListaStudenata} />
      </div>
      </BrowserRouter>
```

Dakle, u primjeru se nalaze dvije Route komponente. Jedna se aktivira za putanju „/profesori/“, a prikazuje komponentu koja predstavlja listu profesora. Druga se aktivira za putanju „/studenti/“, a prikazuje komponentu koja predstavlja listu studenata. Svi elementi iznad Route komponenata uvijek su prisutni na stranici i ne mijenja im se sadržaj bez osvježavanja stranice.

4.2. Express

Express, ili ExpressJS, je okvir za NodeJS namijenjen za razvoj sučelja za programiranje aplikacija na strani poslužitelja. Hahn [23] piše kako je Express relativno malen okvir koji za cilj ima učiniti da razvoj Node web poslužitelja bude što jednostavniji, a njegove funkcionalnosti što moćnije. Budući da je riječ o okviru, Express donosi neke programske koncepte koji, iako su mali teret performansama, omogućuju jednostavan i lagan izgradnju Node HTTP web poslužitelja. Naravno, Node HTTP web poslužitelj moguće je razviti i bez ikakvih dodataka na Node, no u tom slučaju nastaju neke nepotrebne komplikacije koje Express rješava. U srži, Express radi nekoliko stvari:

- dodaje funkcionalnosti rukovateljima Node HTTP zahtjeva na način da dodaje svojstva objektu zahtjeva i objektu odgovora,
- stvara takozvani međusoftver, ili posrednički softver, (*middleware*) – funkcije koje se izvode na određene primljene HTTP zahtjeve,
- omogućuje jednostavan sustav usmjeravanja koji se temelji na HTTP zahtjevima.

Primjer jednostavnog Node HTTP poslužitelja izgrađenog Express okvirom:

```
const express = require('express');
const posluzitelj = express();
const port = 1000;

posluzitelj.get('*', (zahtjev, odgovor) => {
      odgovor.send('Pozdrav, ja sam Express poslužitelj!');
});

app.listen(port);
```

Dakle, funkcija „express“ temelj je čitavog okvira. Ona vraća objekt koji predstavlja poslužitelj na koji se dalje može dodati različiti posrednički softver. U primjeru se nalazi samo jedan takav posrednički softver, koji odgovara na HTTP GET zahtjev na bilo koju putanju poslužitelja (prvi argument „posluzitelj.get“ metode), a kao odgovor vraća navedeni tekst. Na kraju se nalazi naredba poslužitelju da sluša određeni mrežni priključak koji je definiran iznad. Obično web poslužitelji nisu ovako jednostavni – imaju više krajnjih putanja i tipova zahtjeva na koje odgovaraju. Osim toga, često se koristi više posredničkog softvera kako bi se povećala funkcionalnost poslužitelja.

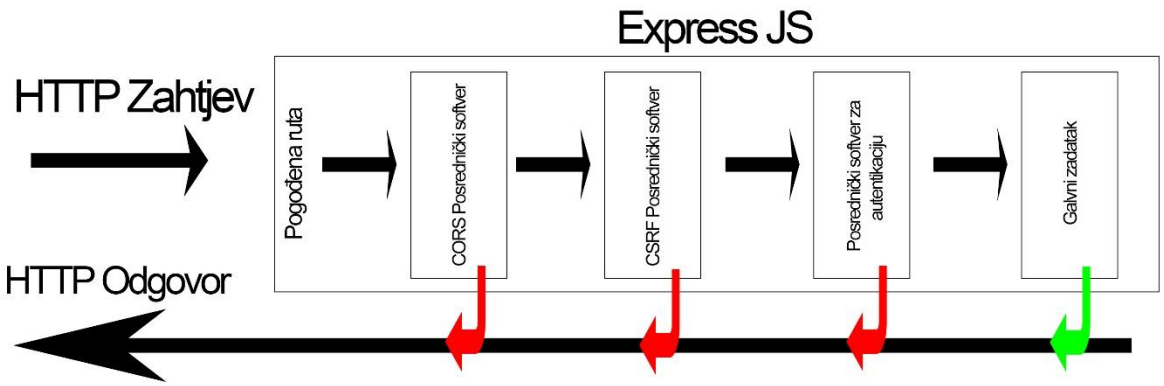
Primjer Express poslužitelja koji koristi više mogućnosti okvira:

```
const express = require('express');
const posluzitelj = express();
const port = 1000;

posluzitelj.use(function(zahtjev, odgovor, sljedeci){
	var trenutnoVrijeme = Date.now();
	zahtjev.vrijeme = trenutnoVrijeme;
	console.log(zahtjev);
	odgovor.json({
		vrijeme: trenutnoVrijeme
	});
	sljedeci();
});

posluzitelj.get('/dohvati/studenti', (zahtjev, odgovor) => {
	response.json({
		studenti: [
			{ime: 'Marko', prezime: 'Marković'},
			{ime: 'Ivan', prezime: 'Ivanović'}
			]
	});
});

posluzitelj.post('/posaljiPodatke', (zahtjev, odgovor) => {
	var imeKorisnika = zahtjev.body.ime;
	var prezimeKorisnika = zahtjev.body.prezime;
	odgovor.json({
		ime: imeKorisnika,
		prezime: prezimeKorisnika
	});
});

posluzitelj.listen(port);
```

Ovaj, malo složeniji, poslužitelj odgovara na nekoliko različitih zahtjeva. Uz to ima i jednu funkciju koja se izvrši za svaki zahtjev, neovisno o vrsti ili putanji. Ta funkcija bilježi podatke o zahtjevu u konzolu, dodaje vrijeme primitka zahtjeva kao svojstvo samog zahtjeva i vraća JavaScript objekt s vremenom primitka zahtjeva. Zabilježeno vrijeme primitka zahtjeva sada je dostupno i drugim funkcijama unutar poslužitelja. Poslužitelj također odgovara na HTTP GET zahtjev za putanju „/dohvati/studenti“ i vraća JavaScript objekt u kojem se nalazi polje s imenima i prezimenima dva studenta. Poslužitelj odgovara i na HTTP POST zahtjev za

putanju „/posaljiPodatke“ koja odgovara s JavaScript objektom koji sadrži poslane podatke o korisnikovom imenu i prezimenu. Bitno je napomenuti kako poslužitelj ne odgovara samo na navedene putanje, već i na sve putanje koji započinju s navedenim putanjama. Primjerice, poslužitelj bi za HTTP GET zahtjev za putanju „/dohvati/studenti/nesto“ odgovorio na isti način kao i za putanju „/dohvati/studenti“. Ista stvar vrijedi i za HTTP POST zahtjev definiran ispod. Na kraju, bitno je primijetiti kako se koristi funkcija „sljedeci“ unutar posredničkog softvera. Da bi se to lakše moglo shvatiti, potrebno je objasniti kojim redoslijedom Express izvršava određene funkcije. Dakle, Express, nakon primitka zahtjeva, traži prvi posrednički softver koji se podudara s primljenim zahtjevom. U slučaju da se koristi metoda „posluzitelj.use“, svaki zahtjev podudarat će se s tim posredničkim softverom. Korištenjem funkcije „sljedeci“, Express nastavlja s traženjem sljedećeg posredničkog softvera koji se podudara sa zahtjevom, ako takav postoji. To se nastavlja toliko dugo dok više nema poziva funkcija „sljedeci“, ili dok se ne dođe do zadnjeg posredničkog softvera koji se podudara sa zahtjevom.

Express, iako je okvir, nema neku veliku arhitekturu. Budući da se radi o okviru za izradu HTTP poslužitelja, glavna mu je funkcionalnost rukovanje primljenim zahtjevima. S obzirom na to, arhitektura Express okvira mogla bi se svesti na prije naveden posrednički softver. Na slici 1 nalazi se uobičajena moguća arhitektura Express aplikacije. Prava Express aplikacija sastojala bi se od mnogo ovakvih dijelova. Primitkom HTTP zahtjeva (*HTTP Request*), aplikacija radi pregled svih svojih ruta i, ukoliko je neka ruta pogođena, odvija se određen posrednički softver. HTTP odgovor (*HTTP response*) šalje se tek nakon što se odvije zadnji posrednički softver. Svi prethodni posrednički softveri tu su da dopune funkcionalnost glavnog zadatka (*main taks*) rukovanja zahtjeva.



Slika 1: Arhitektura Express JavaScript okvira [24]

4.3. jQuery

Kao jedna od najkorištenijih biblioteka, jQuery biblioteka poznata je gotovo svakom JavaScript razvojnom programeru. Funkcionalnosti koje donosi jQuery pomažu gotovo u svakom području razvoja web aplikacije. Usprkos pojednostavljenju JavaScript jezika, jQuery je nekima još uvijek prvi izbor. Stvari poput pretraživanje i prolazak kroz HTML dokument, slanje AJAX zahtjeva i rukovanje događajima puno su jednostavnije za implementirati koristeći jQuery biblioteku [25].

Svaka funkcija jQuery biblioteke započinje znakom „$“. Funkcije za dohvaćanje objekata HTML dokumenta jedne su od najčešće korištenih jQuery funkcija iz razloga što je sintaksa puno kraća nego što je to kod standardnog JavaScript-a.

Primjer usporedbe dohvaćanja i manipulacije elemenata standardnog JavaScript-a i jQuery biblioteke:

```
//standardan JavaScript
var sviOdjeljci = document.getElementsByTagName('p');

//jQuery
var sviOdjeljciJQuery = $('p');

//standardan JavaScript
var zaglavlje = document.getElementById('zaglavlje');

//jQuery
var zaglavljeJQuery = $('#zaglavlje');

sviOdjeljci.foreach(odjeljak => {
      //standardan JavaScript
      odjeljak.style.color = 'white';
      odjeljak.style.backgroundColor = 'blue';

      //jQuery
      $(odjeljak).css({
            'color': 'white',
            'background-color': 'blue'
      });
});
```

Iako obje varijante rade istu stvar, može se vidjeti da je varijanta gdje se koristi jQuery dosta jednostavnija, pogotovo kod dohvaćanja elemenata. Dakle, u primjeru se prvo dohvaćaju svi odjeljci HTML dokumenta, a zatim se dohvaća element s identifikatorom „zaglavlje“. Nakon toga, svakom odjeljku dodaje se plava boja pozadine i bijela boja teksta. Može se primijetiti razlika kod postavljanja CSS svojstava elementa – korištenjem jQuery biblioteke moguće je više svojstava postaviti ili promijeniti pozivom samo jedne funkcije. Osim toga, moglo bi se reći da je u ovom slučaju „prirodnije“ koristiti jQuery iz razloga što su nazivi CSS svojstava identični onima u konkretnoj CSS datoteci.

Još jedno područje gdje se često koristi jQuery je slanje AJAX zahtjeva za dohvaćanje podataka s poslužitelja bez osvježavanja web stranice. Naravno, sve to je moguće napraviti koristeći standardni JavaScript, no, kao i u prijašnjem primjeru, jednostavnije je postići isti rezultat koristeći jQuery biblioteku.

Primjer usporedbe standardnog JavaScript-a i jQuery biblioteke kod slanja jednostavnog AJAX zahtjeva:

```
function Standardno(){
	const zahtjev = new XMLHttpRequest();

	zahtjev.onreadystatechange = () => {
		if(this.readyState === 4 && this.status === 200){
		  document.getElementById('odjeljak').innerHTML =
		  this.responseText;
		}
	}

	zahtjev.open('GET', 'podaci.html', true);
	zahtjev.send();
}

function jQueryZahtjev(){
	$.ajax({
		type: 'GET',
		url: 'podaci.html',
		success: (podaci) => {
			$('#odjeljak').html(podaci);
		}
	});
}
```

Kao i prijašnji primjer, obje funkcije rade istu stvar – dohvaćaju podatke iz vanjskog dokumenta i postavljaju dobivene podatke kao HTML vrijednost jednog elementa. Dok je, kod korištenja standardnog JavaScript-a, potrebno stvoriti objekt koji predstavlja HTTP zahtjev i konstanto pratiti njegovo stanje, jQuery to sve radi pozivom samo jedne funkcije. Uz to, jQuery varijanta čitljivija je, dijelom zato što se sve nalazi unutar poziva iste funkcije, a dijelom zato što se koristi notacija JavaScript objekta (JSON).

Kao i kod ostalih popularnih biblioteka, razvojni programeri imaju na izbor velik broj dodataka za jQuery. Jedan od popularnijih je „Muuri“. Muuri omogućuje izradu izrazito fleksibilnih rasporeda objekata unutar web stranice. Osim toga, čitav raspored elemenata postaje odzivan i podesiv, a s lakoćom ga se može i filtrirati. Još jedan zanimljiv dodatak je „Tooltipster“. Iz naziva može se vidjeti da se radi o opisima ili uputama koje se pojavljuju kod prelaska mišem preko određenog elementa. Dodatak omogućuje jednostavno stvaranje i podešavanje tih opisa. [26]

4.4. Angular

Angular je jedan od najpopularnijih okvira zasnovanih na JavaScript jeziku, a u vlasništvu je Google-a. Zapravo postoje dvije „verzije“ Angular okvira: Angular i AngularJS. AngularJS starija je verzija okvira koja u potpunosti koristi JavaScript kao standardan jezik okvira. Angular je novija verzija koja se ponekad naziva i „Angular 2“, odnosno „Angular 4“. Iako je u potpunosti moguće napisati Angular web aplikaciju koristeći isključivo JavaScript, razvojni programeri Angular-a preporučuju korištenje TypeScript-a – Microsoft-ovog jezika koji se temelji na JavaScript-u, no koji također posjeduje dodatna svojstva i mogućnosti. Iako je TypeScript drugi jezik, on se kod prevođenja svodi na čisti JavaScript iz razloga što web preglednici ne podržavaju TypeScript, već samo JavaScript. [27]

U ovom poglavlju napravit će se pregled Angular okvira (novije verzije).

Angular, slično kao i React, temelji se na nekoliko koncepata, od kojih su glavne komponente. Angular komponenta zapravo je TypeScript klasa s nekim dodatnim svojstvima. Moglo bi se reći da se svaka Angular komponenta sastoji od četiri dijela: naziva selektora, HTML predloška (šablone), CSS datoteke za stilske upute i TypeScript datoteke u kojoj se nalazi sav programski kod.

Primjer jednostavne Angular komponente:

```
import { Component, OnInit } from '@angular/core';
import { listaStudenata } from './listaStudenata.ts';

@Component({
      selector: 'popis-studenata',
      templateUrl: './popis-studenata.component.html',
      styleUrls: ['./popis-studenata.component.css']
})
export class PopisStudenata implements OnInit{
      studenti = listaStudenata;


}
```

U samoj deklaraciji klase ne događa se ništa posebno. Najbitniji dio za primijetiti ovdje je „@Component“ prefiks koji se nalazi prije same deklaracije klase. U njemu se definiraju glavna svojstva komponente. U ovom slučaju selektor komponente je „popis-studenata“, šablona za komponentu je „popis-studenata.component.html“, a stilske upute za komponentu nalaze se u „popis-studenata.component.css“. Osim toga, bitno je primijetiti uvoz datoteke „listaStudenata.ts“, u kojoj se nalazi standardno JavaScript polje u kojem se nalazi više JavaScript objekata od kojih svaki sadrži podatke o nekom studentu. Naravno, klasa ne govori kako će komponenta zapravo izgledati na stranici, niti od kojih elemenata će se sastojati. Te stvari definirane su u HTML datoteci koja predstavlja šablonu za ovu komponentu.

Primjer HTML šablone za komponentu „popis-studenata“:

```
<h2>
      Studenti
</h2>

<div *ngFor='let student of studenti'>
      <ul [id]='student.maticniBroj'>
            <li>Ime: {{student.ime}}</li>
            <li>Prezime: {{student.prezime}}</li>
            <li>Studij: {{student.studij}}</li>
      </ul>
</div>
```

Iz primjera se vidi na koji način Angular integrira HTML i JavaScript, odnosno TypeScript. Angular definira posebne atribute za HTML elemente pomoću kojih se izbjegava nepotrebno pisanje koda. Šablona može koristiti bilo koje atribute i svojstva koji su definirani unutar klase. U ovom slučaju to je objekt „listaStudenata“ koji je uvezen iz odgovarajuće TypeScript datoteke. Najbitnija stvar iz primjera je Angular smjernica „*ngFor“ koja funkcionira kao petlja, a zapravo prikazuje onoliko „ “ HTML elemenata koliko ima studenata u polju. Također, za svaki prikaz studenta Angular stvara i listu u kojoj se prikazuju ime, prezime i smjer određenog studenta. Unutar liste može se vidjeti način na koji Angular obavlja interpolaciju varijabli – naziv varijable nalazi se unutar dva para vitičastih zagrada. Što se tiče vezanja atributa s vrijednošću neke varijable, to se obavlja na drugi način. Potrebno je naziv atributa staviti unutar uglatih zagrada, a naziv varijable unutar navodnika. Primjer toga je u dodjeljivanju matičnog broja studenta identifikatoru liste u kojoj se nalaze podaci o studentu.

Prijašnji primjer dosta je jednostavan. Angular, naravno, omogućuje puno više funkcionalnosti. Primjer će se unaprijediti na sljedeći način. Ukoliko student ima upisani studij koji polaže, isti će se prikazati. Ukoliko student nema upisani studij, umjesto praznog polja ispisat će se poruka o tome da student još nije u potpunosti obavio upis u neki studij. U ovom slučaju neće se mijenjati TypeScript datoteka s klasom, već samo HTML datoteka šablone.

Primjer unaprijeđene komponente:

```
<h2>
      Studenti
</h2>

<div *ngFor='let student of studenti'>
      <ul [id]='student.maticniBroj'>
            <li>Ime: {{student.ime}}</li>
            <li>Prezime: {{student.prezime}}</li>
            <li *ngIf='student.studij; else nijeUpisan'>
                  Studij:{{student.studij}}
            </li>
            <ng-template #nijeUpisan>
                  <li>Student nije obavio upis u studij</li>
            </ng-template>
      </ul>
</div>
```

Dakle, u prijašnji primjer dodalo se samo par linija koda. Dodana je Angular smjernica „*ngIf“. Kao vrijednost smjernice upisan je uvjet, te što da Angular učini ukoliko uvjet nije zadovoljen. Također je dodan element „<ng-template>“ i vrijednost elementu koja se podudara s vrijednošću nakon „else“ naredbe u prijašnjem elementu. To znači da će se taj element prikazati samo kada ne postoji zabilježen podatak o studiju studenta. Važno je da se takve slučajeve isključivo koristi element „<ng-element>“ – taj element ne postoji u HTML-u, no Angular ga na korektan način interpretira i prikazuje na web stranici [28].

Još jedna nadogradnja koja bi se mogla napraviti na postojećem primjeru je to da se doda mogućnost klika na gumb nakon kojeg se ispisuje poruka o dostupnim studijima, u slučaju da student nije upisan u niti jedan studij. To se može ostvariti kreiranjem funkcije unutar klase i povezivanjem te funkcije na rukovatelj podataka unutar HTML šablone.

Nadogradnja klase prijašnjeg primjera:

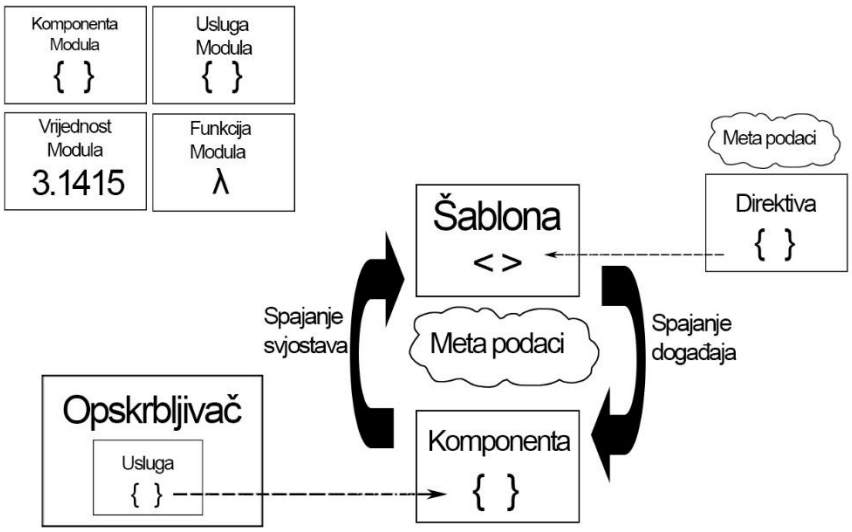
```
import { Component, OnInit } from '@angular/core';
import { listaStudenata } from './listaStudenata.ts';

@Component({
	selector: 'popis-studenata',
	templateUrl: './popis-studenata.component.html',
	styleUrls: ['./popis-studenata.component.css']
})
export class PopisStudenata implements OnInit{
	studenti = listaStudenata;

	PrikaziInformacije(){
		alert('Dostupni studiji: Informacijski i poslovni
			sustavi, Ekonomika poduzetništva');
	}
}
```

Nadogradnja HTML šablone prijašnjeg primjera:

```
<h2>
	Studenti
</h2>

<div *ngFor='let student of studenti'>
	<ul [id]='student.maticniBroj'>
		<li>Ime: {{student.ime}}</li>
		<li>Prezime: {{student.prezime}}</li>
		<li *ngIf='student.studij; else nijeUpisan'>
			Studij:{{student.studij}}
		</li>
		<ng-template #nijeUpisan>
			<button (click)='PrikaziInformacije()'>
				Informacije
			</button>
		</ng-template>
	</ul>
</div>
```

Na gumb dodan je rukovatelj događaja koji reagira na klik, a vrijednost rukovatelja je naziv odgovarajuće funkcije koja se treba izvršiti. Povezivanje rukovatelja događaja obavlja se pisanjem događaja unutar oblih zagrada.

Na slici 2 nalazi se prikaz osnovne arhitekture Angular okvira. Najbitniji dio čitave arhitekture nalazi se u sredini slike – šablone (*template*) su u suradnji s komponentama (*component*) na način da komponente na šablone vežu svoja svojstva (*property binding*), a šablone na komponente vežu događaje (*event binding*). Pod „service“ spadaju svakakve funkcije i vrijednosti koje Angular aplikacija koristi [29].



Slika 2: Arhitektura Angular JavaScript okvira [29]

4.5. Grafički prikazi

Mnogo modernih web aplikacija temelji se na nekom obliku prikaza podataka i informacija. Jedna stvar je na korektan način dohvatiti podatke iz određenog izvora, bila to baza podataka ili neki drugi vanjski izvor, a druga stvar je na dobar način prikazati iste podatke. Osoba obično koristi web aplikaciju kako bi dobila najnovije informacije o, njoj bitnim, stvarima. Budući da čovjek veliku većinu informacija prima vizualnim putem, ima smisla što više se potruditi da se podaci prikažu na čitljiv i pristupačan način, sa što manjom vjerojatnošću postojanja zabune. Drugim riječima, poželjno je podatke prikazati na sažet i razumljiv način.

Jedan od najpopularnijih načina prikaza podataka i informacija je stvaranje grafikona, odnosno grafova. Oni imaju moć prikazati veliku količinu podataka i pritom ostaviti jako mali trag, što ih, između ostalog, čini savršenim kandidatom za korištenje na web aplikacijama. Još

jedan razlog zašto su grafikoni dobri za prikaz podataka na web aplikacijama je to da mogu pridonijeti sveukupnom grafičkom stilu i dizajnu web aplikacije.

Postoji mnogo vrsta i stilova grafikona, a oni se također mogu urediti raznim svojstvima, poput boje, veličine i pomoćnih linija. Kod korištenja grafikona potrebno je pripaziti na to da vrsta grafikona bude usklađena s podacima koji se prikazuju. Primjerice, kod prikaza udjela sastojaka nekog kemijskog spoja pametno bi bilo koristiti kružni grafikon (*pie chart*), dok bi kod prikaza rasta ili pada vrijednosti dionica bilo pametnije koristiti linijski grafikon (*line chart*). Uz to, potrebno je pripaziti na količinu dostupnih podataka za prikaz. Ukoliko se radi o izrazito velikoj količini podataka, tada ne bi bilo loše grupirati podatke u logičke cjeline.

Rastom popularnosti JavaScript-a i razvojem web tehnologija nastale su mnoge JavaScript biblioteke za stvaranje grafičkih prikaza, odnosno grafikona. Dio njih svodi se na uvoz biblioteke u skriptu i prosljeđivanje JavaScript objekta nekoj funkciji koja kao rezultat vraća grafikon u nekom obliku. Često se radi o SVG elementima iz razloga što je takve elemente lakše prikazati na svim mogućim uređajima s različitim dimenzijama zaslona. Budući da postoji mnogo takvih biblioteka, niti jedna nije savršena za bilo koju problemsku domenu. Obično se radi o specijaliziranim bibliotekama koje pokrivaju određene tipove grafikona koji su savršeni za korištenje samo kod nekih vrsta podataka. Uz to, neke biblioteke omogućuju dodatno uređivanje generiranih grafikona poput promijene boje linija, stupaca i sličnih elemenata, dodavanje pomoćnih linija za lakše čitanje grafikona, mijenjanje skale grafikona i mijenjanje količine oznaka na osima grafikona.

U nastavku će se napraviti pregled nekih popularnijih JavaScript biblioteka za izradu grafičkih prikaza.

4.5.1. Chart.js

Chart.js jedna je od najpopularnijih JavaScript biblioteka za generiranje grafičkih prikaza. Glavni razlog zašto je ova biblioteka toliko popularna je kontinuirano unaprjeđenje od strane Chart.js razvojnih programera. Osim toga, biblioteka je, za razliku od nekih drugih, u potpunosti besplatna za korištenje.

Temelj korištenja ove biblioteke je „canvas“ HTML element. Taj element se obično i koristi za stvari poput generiranja grafičkih prikaza, pa se stoga često javlja u ovakvim bibliotekama [30].

Dakle, kako bi se ova biblioteka mogla koristiti, na stranici mora postojati „canvas“ HTML element s nekim identifikatorom. Nakon toga, u JavaScript skripti, potrebno je dohvatiti taj element, odnosno njegov grafički kontekst. Sa svim potrebnim podacima, moguće je kreirati novi objekt tipa „Chart“, te u konstruktor proslijediti sve potrebne podatke za prikaz grafikona.

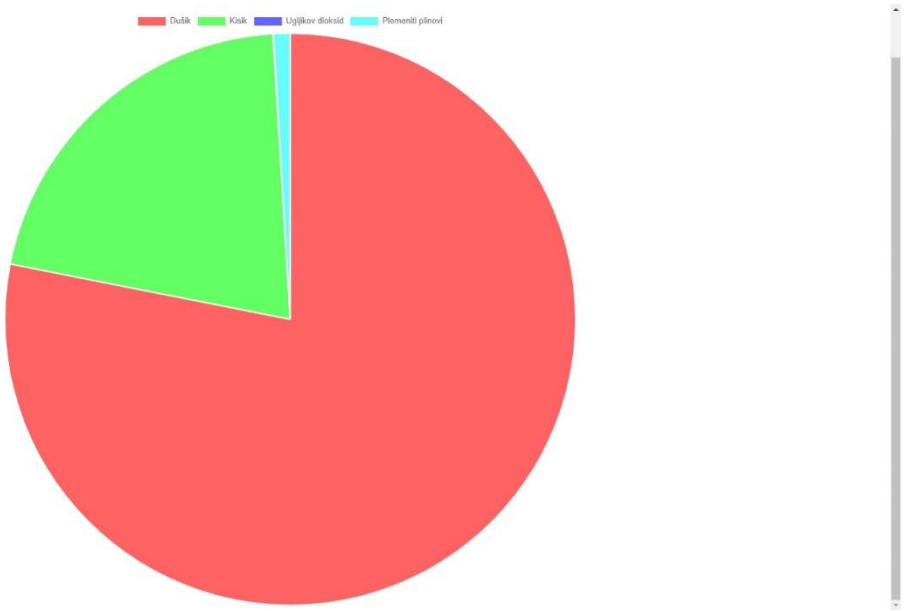
Jednostavan primjer grafikona korištenjem Chart.js biblioteke – HTML stranica:

```
<body>

	<canvas id="grafikon"></canvas>

	<script src=
		"https://cdn.jsdelivr.net/npm/chart.js@2.8.0">
	</script>

	<script src="app.js"></script>

</body>
```

JavaScript skripta u kojoj se generira grafikon:

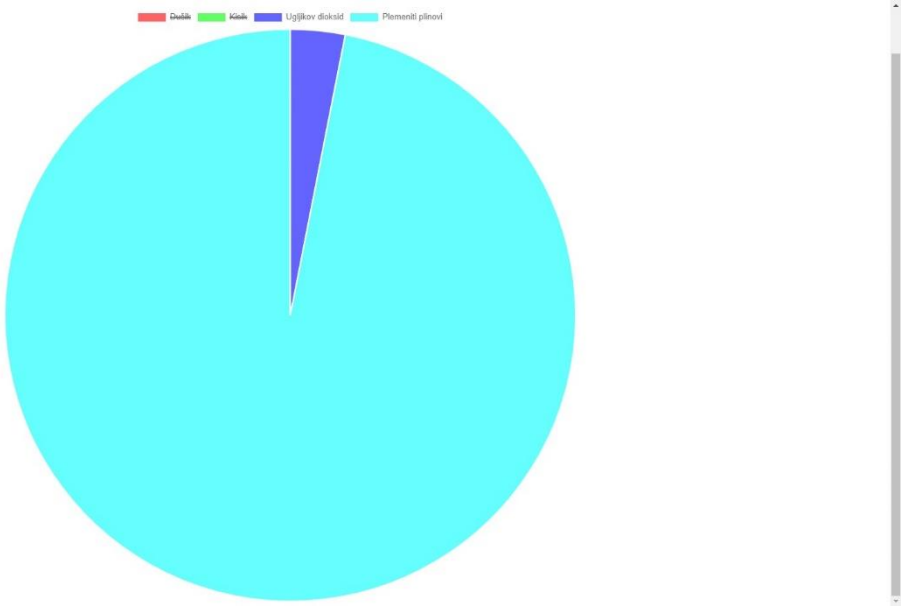
```
var kontekst = document.getElementById('grafikon').getContext('2d');

var grafikon = new Chart(kontekst, {
	type: 'pie',

	data: {
		labels:
		['Dušik', 'Kisik', 'Ugljikov dioksid',
		'Plemeniti plinovi'],

		datasets: [
			{
				label: 'Udio plinova u zraku',
				backgroundColor: [
					'rgb(255, 100, 100)',
					'rgb(100, 255, 100)',
					'rgb(100, 100, 255)',
					'rgb(100, 255, 255)'
				],
				data: [78.1, 20.93, 0.03, 0.94]
			}
		]
	}
});
```

Prvi korak generiranja ovog grafikona je dohvaćanje grafičkog konteksta „canvas“ HTML elementa. U ovom primjeru radi se od dvodimenzionalnom (2d) grafikonu, pa se zato i dohvaća dvodimenzionalni grafički kontekst. Slijedi stvaranje objekta tipa „Chart“. Prvi argument konstruktora je prije dohvaćeni kontekst. Drugi argument je JSON objekt u kojem se nalaze svi potrebni podaci za prikaz grafikona. Tip grafikona je kružni grafikon (*pie*), a sam graf prikazuje udio plinova u zraku. Zanimljivo je za primijetiti kako je svojstvo „datasets“ zapravo polje, što znači se na istom grafikonu može prikazati nekoliko, međusobno ovisnih ili neovisnih, skupova podataka. Osim toga, svojstvo „backgroundColor“ je u ovom primjeru također polje, iz razloga što je svaki podatak u kružnom grafu potrebno prikazati različitom bojom. Kod drugih tipova grafikona, moguće je ovo svojstvo postaviti na samo jednu boju kako bi čitav grafikon bio iste boje.



Slika 3: Grafikon generiran Chart.js bibliotekom [snimka zaslona]

Iznad samog grafikona, biblioteka je generirala legendu radi čitljivosti. Prelaskom preko bilo kojeg elementa skupa podataka pojavljuje se animirani okvir s podacima koji odgovaraju tom elementu. Nadalje, klikom na neki element u legendi iznad grafikona, moguće je podesiti vidljivost bilo kojeg elementa skupa, što je također animirano. Budući da je udio ugljikovog dioksida i plemenitih plinova u zraku vrlo malen, oni su jedva vidljivi na grafikonu. Isključivanjem dušika i kisika iz grafikona dobije se bolji pregled ostalih plinova.



Slika 4: Grafikon generiran Chart.js bibliotekom - eliminiranje pojedinih elemenata [snimka zaslona]

4.5.2. Anychart

Anychart je JavaScript biblioteka koja služi za generiranje raznim grafikona. U vlasništvu je istoimene kompanije. Za razliku od Chart.js biblioteke, Anychart nije u potpunosti besplatna biblioteka. Postoji besplatna verzija, no korištenjem besplatne verzije na dnu svakog grafikona javlja se žig s naznakom da se koristi besplatna verzija biblioteke.

Unatoč svojoj cijeni, Anychart biblioteka nudi ogroman broj različitih vrsta grafikona – od običnih stupčastih grafikona pa do raznim grafikona rasipanja, trodimenzionalnih grafikona površine i gantograma, odnosno Ganttovog dijagrama. Isto tako, Anychart biblioteka nudi veliku slobodu što se tiče uređivanja generiranih grafikona bojama, natpisima i simbolima.

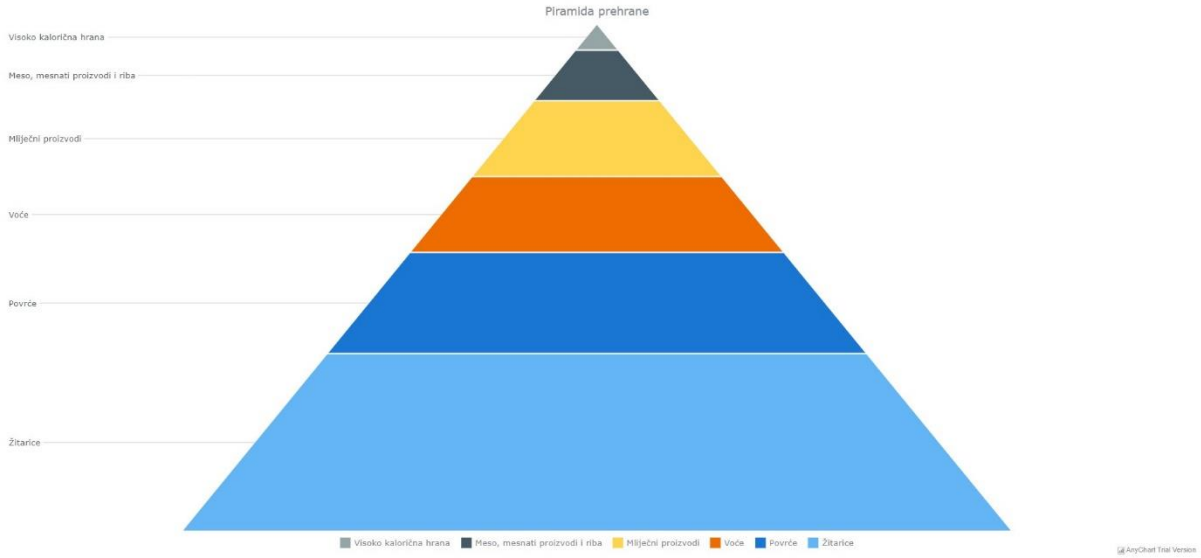
Dok je chart.js biblioteka zahtijevala korištenje „canvas“ HTML elementa za generiranje grafikona, Anychart biblioteka funkcionira s bilo kojim HTML blok elementom, toliko dugo dok taj element ima jedinstveni identifikator za dohvaćanje reference. Obično se u svrhe ove biblioteke koristi „div“ HTML element. Osim toga, za generiranje grafikona moguće je koristiti nekoliko načina, odnosno formata. Uobičajeno je da se koristi standardno JavaScript sučelje. Ukoliko se podaci nalaze u nekom drugom formatu poput JSON, XML i CSV, moguće je koristiti vanjske datoteke kao izvor podataka. [31]

Primjer piramidalnog grafikona generiranog pomoću Anychart biblioteke [32]:

```
<!-- HTML stranica -->
<head>
      <script
      src="https://cdn.anychart.com/releases/8.7.0/js/
      anychart-core.min.js">
      </script>
      <script
      src="https://cdn.anychart.com/releases/8.7.0/js/
      anychart-pyramid-funnel.min.js">
      </script>
</head>

<body>

<div id='graf'> </div>

</body>

//JavaScript kod
anychart.onDocumentLoad(function() {
      var graf = anychart.pyramid();
      graf.data([
            ['Žitarice', 35],
            ['Povrće', 20],
            ['Voće', 15],
            ['Mliječni proizvodi', 15],
            ['Meso, mesnati proizvodi i riba', 10],
            ['Visoko kalorična hrana', 5]
      ]);

      graf.title('Piramida prehrane');
      graf.container('graf');
```

```
        graf.pointsPadding(2);

        graf.draw();
});
```

Na HTML stranici uvoze se potrebne skripte, a na samoj stranici nalazi se jedan „div“ HTML element s identifikatorom u kojem se generira grafikon. Nadalje, u JavaScript kodu čeka se da se HTML dokument do kraja učita, pa se zatim poziva funkcija „anychart.pyramid“. Ona funkcionira poput konstruktora klase, odnosno vraća objekt tipa piramidalnog grafikona. Zatim se svojstvo „data“ stvorenog objekta puni podacima za prikaz. U ovom primjeru radi se o piramidi prehrane. Tekstualni dio svakog elementa za prikaz je naziv, dok brojčani dio predstavlja postotak piramide kojeg taj element treba zauzeti. Nakon toga se na objekt dodaju neka dodatna svojstva, od kojih je najbitnije svojstvo „container“. Ono određuje u koji HTML element na stranici će se generirani grafikon smjestiti. Osim toga, postavljeno je svojstvo naslova grafikona i svojstvo „padding“, odnosno razmak između svakog pojedinog elementa grafikona. Na kraju je potrebno pozvati metodu „draw“ kako bi se grafikon zapravo prikazao unutar odabranog elementa.



Slika 5: Grafikon generiran Anychart bibliotekom - piramida prehrane [snimka zaslona]

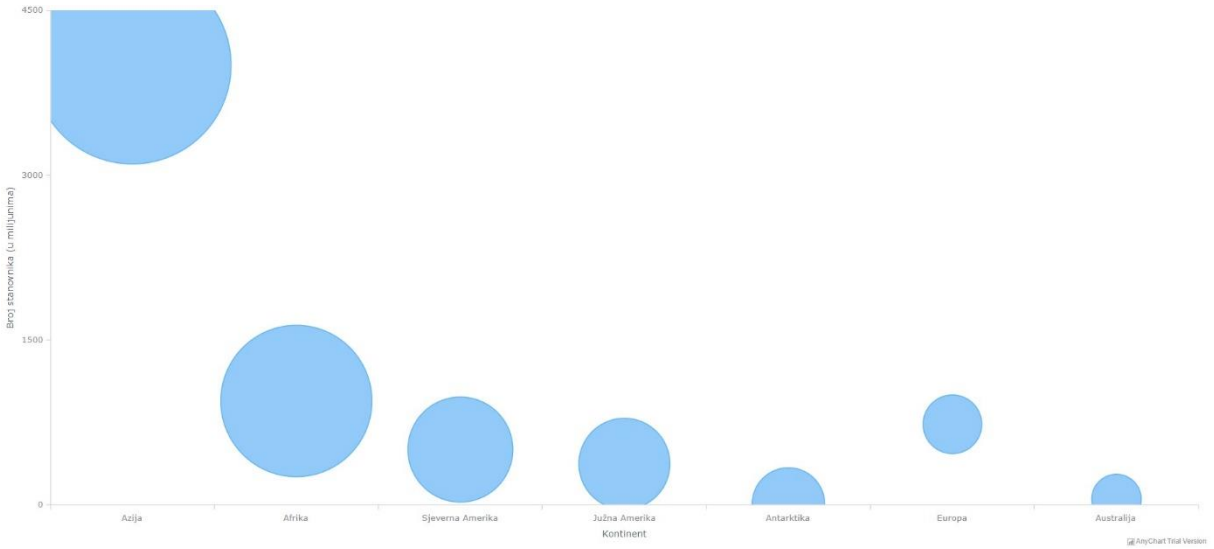
Kao što se može vidjeti, u donjem desnom kutu javlja se naznaka o korištenju besplatne verzije biblioteke. Također se može vidjeti mali razmak između svakog elementa piramide, kao što je i bilo postavljeno prije navedenim svojstvom.

Još jedan od zanimljivih tipova grafikona koje nudi ova biblioteka je mjehurićasti grafikon (*bubble chart*). Sastoji se od dvije osi od kojih jedva, obično vertikalna os, prikazuje nekakvu vrijednost, a druga prikazuje kategoriju ili grupaciju podataka. Sam postupak generiranja sličan je i prijašnjem primjeru, uz male razlike.

Primjer mjehurićastog grafikona generiranog bibliotekom Anychart [33]:

```
<!-- HTML stranica -->
            <head>
                  <script
                  src="https://cdn.anychart.com/releases/8.7.0/js/
                  anychart-core.min.js">
                  </script>
                  <script
                  src="https://cdn.anychart.com/releases/8.7.0/js/
                  anychart-pyramid-funnel.min.js">
                  </script>
            </head>

            //JavaScript kod
            anychart.onDocumentLoad(function() {
                  graf = anychart.cartesian();

                  podaci = [
                        ['Azija', 4000, 40],
                        ['Afrika', 942, 30],
                        ['Sjeverna Amerika', 500, 20],
                        ['Južna Amerika', 370, 17],
                        ['Antarktika', 4, 13],
                        ['Europa', 730, 10],
                        ['Australija', 50, 8]
                  ];

                        graf.bubble(podaci);

                        graf.title('Kontinenti');

                        graf.xAxis().title('Kontinent');

                        graf.yAxis().title
                              ('Broj stanovnika (u milijunima)');

                        graf.container('graf');

                        graf.draw();
                  });
```

Budući da se ovdje radi o Kartezijevom koordinatnom sustavu, za stvaranje objekta grafikona koristi se funkcija „anychart.cartesian“. Nakon definiranja podataka, potrebno je specificirati konkretan tip grafikona iz razloga što pod kategoriju „cartesian“ spada velik broj grafikona. Kao argument ove metode navode se definirani podaci za prikaz. Nakon toga se, kao i u prošlom primjeru, definiraju neka svojstva objekta. Zanimljivo je primijetiti da se ovdje koriste metode „xAxis“ i „yAxis“ i prikladna svojstva za uređivanje izgleda koordinatnih osi grafikona. Na kraju se ponovno koriste metode „container“ i „draw“ za određivanje HTML elementa za prikaz, odnosno prikazivanje samog grafikona.



Slika 6: Grafikon generiran Anychart bibliotekom - površina i stanovništvo kontinenata [snimka zaslona]

U ovom primjeru prikazuju se kontinenti uz njihove površine i broj stanovnika. Na horizontalnoj osi nalaze se nazivi kontinenata, a na vertikalnoj osi se nalazi količina koja obilježava broj stanovnika kontinenta. Površina kontinenta određena je površinom mjehurića, odnosno kruga. Dakle, što je krug na višoj poziciji, time kontinent ima više stanovnika. S druge strane, što je krug veći, time je kontinent površinski veći.

4.5.3.React-vis

Kao što se i iz imena ove biblioteke vidi, React-vis je JavaScript biblioteka namijenjena za korištenje uz React biblioteku. U stvari, nemoguće je koristiti ovu biblioteku bez React biblioteke. Moglo bi se reći da je ovo biblioteka zapravo dodatak na React biblioteku.

React-vis dizajniran je poštivanjem nekoliko principa. Prvi je to da se React-vis biblioteka mora savršeno pasati uz React biblioteku. Drugim riječima, React-vis sadrži mnoge komponente koje funkcioniraju poput onih u React-u. Drugi princip je velika razina prilagodbe. Moguće je napraviti svoje React komponente iz dostupnih komponenata React-vis biblioteke. Osim toga, prilagodljivost svake od ponuđenih komponenata je velika. Treći princip dosta je jednostavan – biblioteka mora podržavati velik broj različitih vrsta grafikona. [34]

Budući da ova biblioteka ne funkcionira bez React biblioteke, generiranje grafikona biti će malo drugačije nego što je to bilo u prijašnjim primjerima. Ovdje nije potrebno koristiti standardan HTML iz razloga što se React biblioteka brine za taj dio posla. Naravno, preduvjet za korištenje ove biblioteke je da se u projektu koristi React biblioteka.

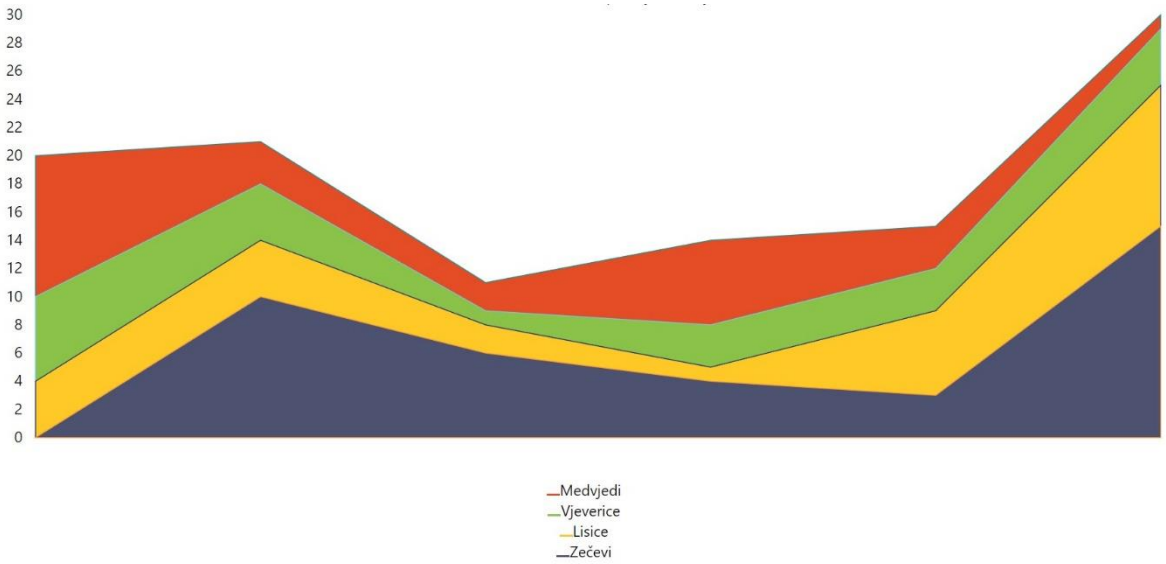
Najbitnija React-vis komponenta je „XYPlot“. Ona se zapravo koristi kao korijenska komponenta kod stvaranja bilo kojeg React-vis grafikona. Osim nje, često se koriste

komponente „XAxis“ i „YAxis“ radi prikaza i prilagođavanja koordinatnih osi. Kako se ovdje radi o React komponentama, jako je jednostavno nekoliko istih, ili različitih, vrsta grafikona prikazati na istom koordinatnom sustavu. Potrebno je samo ugnijezditi nekoliko komponenata unutar „XYPlot“ komponente.

Primjer jednostavnog grafikona izrađenog korištenjem React-vis biblioteke:

```
function Grafikon() {
      return (
            <div>
            <XYPlot height={500} width={1250}>

                  <XAxis hideTicks></XAxis>
                  <YAxis></YAxis>

                  <AreaSeries
                  fill="#E44D26" data={medvjedi}>
                  </AreaSeries>

                  <AreaSeries
                  fill="#8BC34A" data={vjeverice}>
                  </AreaSeries>

                  <AreaSeries
                  fill="#FFCA28" data={lisice}>
                  </AreaSeries>

                  <AreaSeries
                  fill="#4D5271" data={zecevi}>
                  </AreaSeries>

                  <DiscreteColorLegend items={zivotinje}>
                  </DiscreteColorLegend>
            </XYPlot>
            </div>
      );
}
```

U ovom primjeru se stvara nekoliko različitih „AreaSeries“ komponenata, od kojih svaka predstavlja jedan grafikon. Svakoj komponenti se, poput atributa, prosljeđuje polje u kojem se nalazi nekoliko objekata s podacima za prikaz. Uz to, stvara se i legenda radi lakšeg čitanja grafikona. Komponenti legende također se prosljeđuje polje s potrebnim podacima za prikaz same komponente.



Slika 7: Grafikon generiran korištenjem React-vis biblioteke - populacija životinja [snimka zaslona]

Dio podataka koji su korišteni za prikaz grafikona:

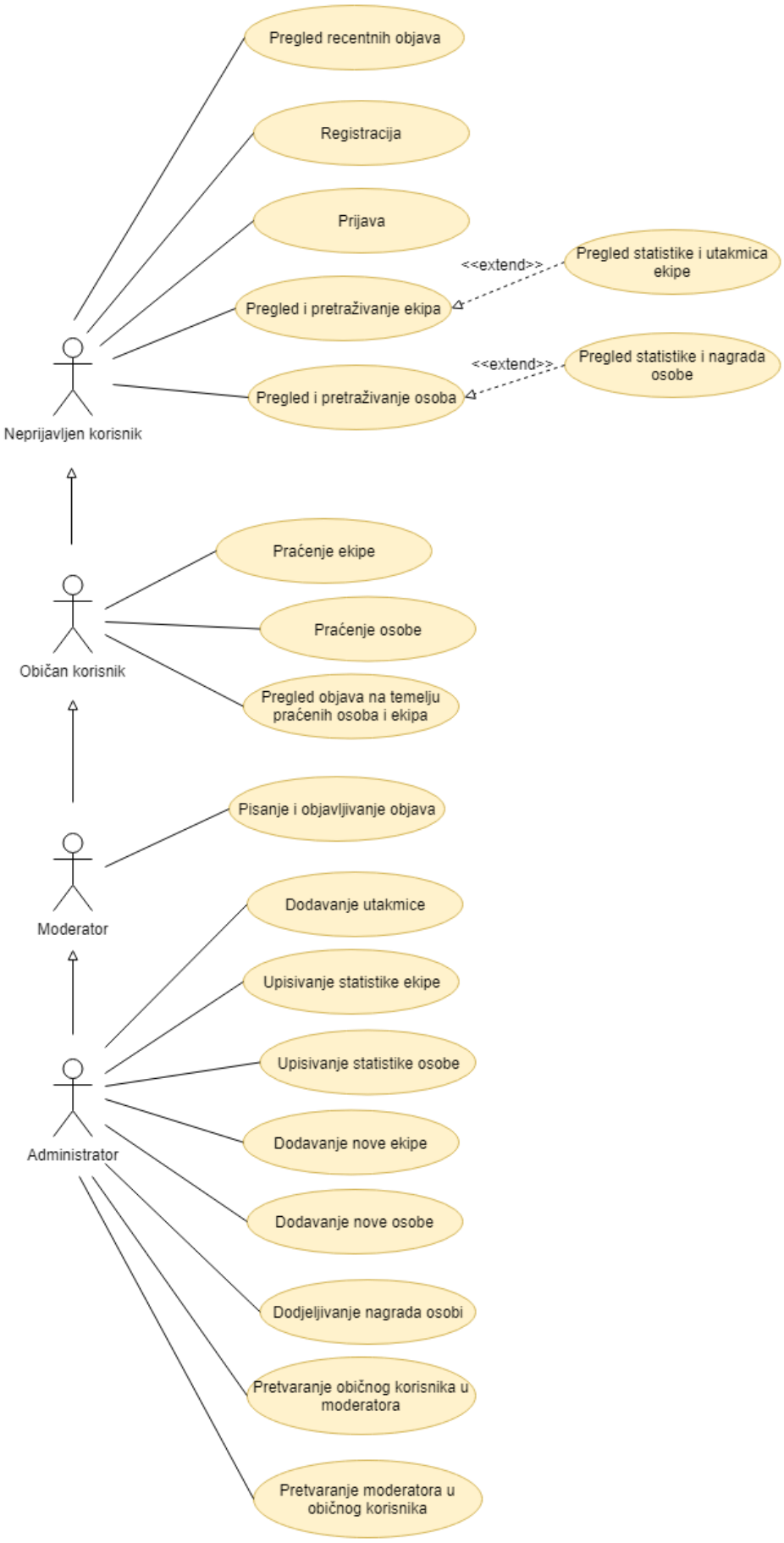
```
var zivotinje = [
      { title: 'Medvjedi', color: '#E44D26'},
      { title: 'Vjeverice', color: '#8BC34A'},
      { title: 'Lisice', color: '#FFCA28'},
      { title: 'Zečevi', color: '#4D5271'}
];

var zecevi = [
      { x: 0, y: 0 },
      { x: 1, y: 10 },
      { x: 2, y: 6 },
      { x: 3, y: 4 },
      { x: 4, y: 3 },
      { x: 5, y: 15 }
];
```

Polje „zivotinje“ prosljeđuje se komponenti legende i sadrži četiri objekata sa svojstvima „title“ i „color“ koji određuju tekst i boju uz koju je vezan tekst. Polje „zecevi“ prosljeđuje se jednoj od „AreaSeries“ komponenata i sadrži jednostavne koordinate koje predstavljaju točke u koordinatnom sustavu. Ista takva polja, no s drugim podacima, prosljeđuju se i drugim „AreaSeries“ komponentama.

5. Praktični dio

Praktični dio ovog rada u potpunosti je izrađen korištenjem JavaScript jezika. Dio aplikacije na strani poslužitelja izrađen je korištenjem Node.js-a, Express JavaScript okvira i MySQL Node.js modula za spajanje na bazu. Taj dio aplikacije odgovoran je za komunikaciju s bazom podataka i posluživanje korisnika s odgovarajućim datotekama i podacima. Osim toga, korišten je „path“, „md5“ i „nodemailer“ Node.js moduli za ostvarivanje ostalih funkcionalnosti poslužiteljskog dijela aplikacije. Dio aplikacije na strani korisnika izrađen je korištenjem nekoliko biblioteka. Radi se o aplikaciji koja se u potpunosti odvija na jednoj web stranici, tako da, nakon što se aplikacija jednom učita, nema dodatnog učitavanja novih stranica. Kako bi se to postiglo korištena je React JavaScript biblioteka i njen dodatak „React Router“. Za realizaciju AJAX-a korištena je „axios“ JavaScript biblioteka za slanje HTTP zahtjeva na poslužiteljsku stranu aplikacije. Kako bi aplikacija mogla koristiti kolačiće, korištena je i mala „js-cookie“ biblioteka. Svi navedeni okviri, biblioteke i dodaci dohvaćeni su korištenjem „npm“ menadžera paketa za Node.js. Korištena je i usluga besplatnog web hostinga MySQL baze podataka „RemoteMySQL“ [35]. Kako bi se aplikacija mogla pokretati bilo gdje, korištena je besplatna usluga web hostinga od strane Heroku organizacije [36].

Aplikacija služi kao izvor sportskih novosti za obožavatelje određenog sporta. Kao tema ovog rada odabrana je NBA košarkaška liga. Glavna funkcionalnost aplikacije je mogućnost praćenja novosti za odabrane osobe i ekipe unutar sportske lige. Osim toga, moguće je pratiti statistička i ostala postignuća igrača i ekipa, te rezultate utakmica. Postoji nekoliko uloga korisnika ove aplikacije. Najosnovnija uloga je neprijavljen korisnik. Neprijavljen korisnik može vidjeti zadnjih 50 napisanih objava za sve ekipe i osobe. Osim toga, može napraviti pregled svih osoba i ekipa unutar sportske lige, te vidjeti njihove rezultate po sezonama.



Slika 8: Dijagram slučajeva korištenja [autorski rad]

Neprijavljen korisnik, prilikom otvaranja aplikacije, dolazi do pregleda zadnjih 50 objava.

Slika 9: Naslovna strana - neprijavljen korisnik [snimka zaslona]

Dio programskog koda koji je odgovoran za dohvaćanje i prikazivanje objava za neprijavljenog korisnika:

```
var korisnik = Cookies.get('id');
if(!korisnik){
	axios.get(`/api/objave/dohvati`).then(rezultat => {
		this.setState({
			objave: rezultat.data
		});
		this.SrediDatume();
	});
}

//metoda SrediDatume
var objave = this.state.objave;

for (var i = 0; i < objave.length; i++) {
	var datum = objave[i].datum;
	datum = datum.slice(0, -5);
	datum = datum.split('T');
		objave[i].datum = datum[0];
		objave[i].vrijeme = datum[1];
	}
this.setState({
	objave: objave
});
```

Dakle, prvi korak je provjera postojanja kolačića kako bi se provjerilo stanje trenutnog korisnika. Nakon toga, ovisno o određenom stanju, šalje se HTTP GET zahtjev na poslužitelj

koji vraća polje popunjeno objektima objava. Nakon postavljanja stanja komponente poziva se metoda „SrediDatume“ kako bi se dohvaćeni datumi uredili za prikaz koji je lakši za čitanje. Prilikom postavljanja stanja komponente ona vrši osvježavanje prikaza i prikazuju se dohvaćene objave.

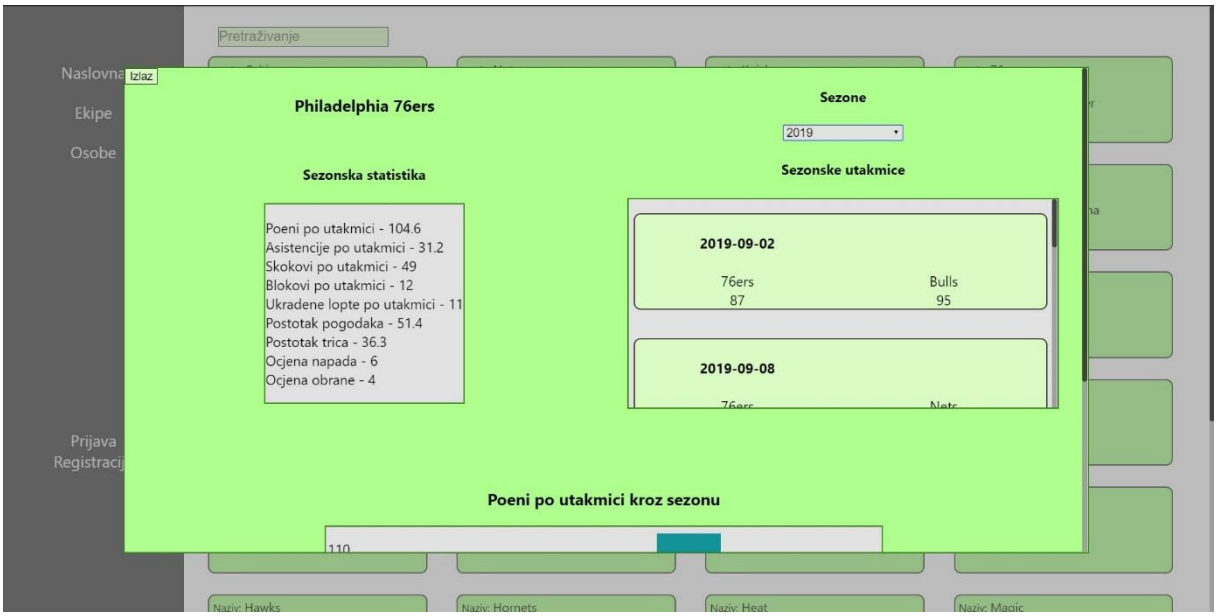
Metoda za prikaz naslovne komponente:

```
render() {
      if (this.state.objave.length === 0) {
            return <Ucitavanje></Ucitavanje>;
      } else {
            return (
            <div className="grid">
                  {this.state.objave.map(objava => (
                  <div
                  key={objava.id}
                  id={objava.id}
                  >
            {objava.naslov}
            ({objava.prezime},{objava.ime})
            <br />
            Napisano: {objava.datum} - {objava.vrijeme}
            <p>{objava.tekst}</p>
                  </div>
                  ))}
            </div>
            );
      }
}
```

Ukoliko još nisu dohvaćene objave, prikazuje se poruka učitavanja podataka. Za prikaz svake pojedine objave u polju koristi se metoda „map“. Ona prolazi kroz svaki element polja i vraća određen HTML element kao rezultat. Neprijavljen korisnik još može vidjeti sve ekipe i osobe u sportskoj ligi.

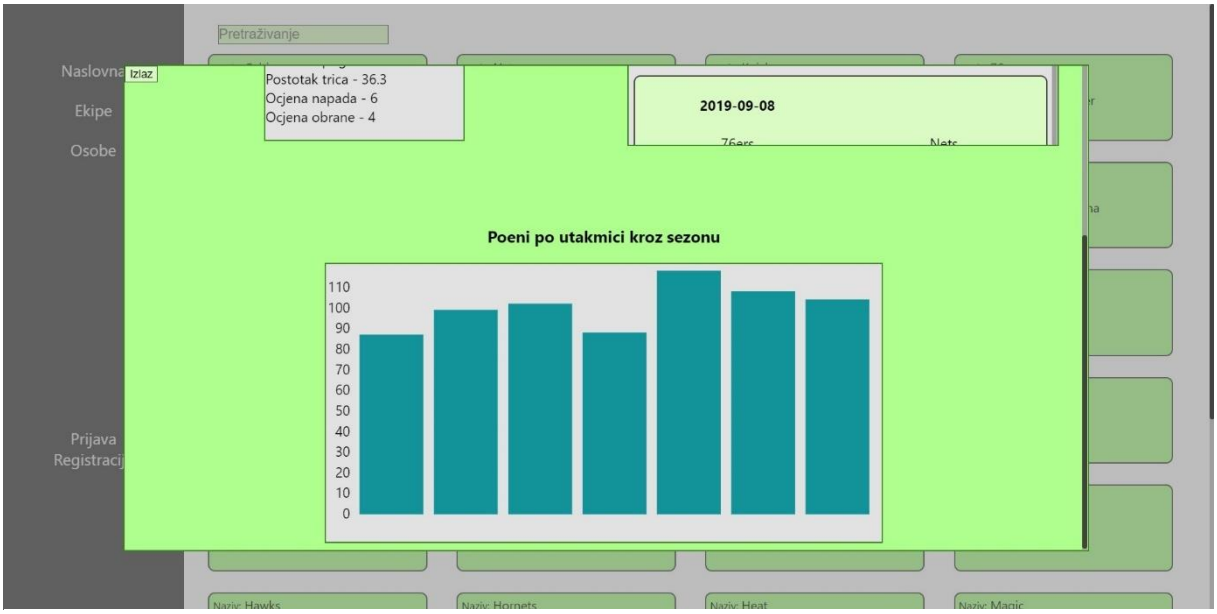
Slika 10: Pregled svih ekipa - neprijavljen korisnik [snimka zaslona]

Na vrhu ovog pregleda nalazi se polje za unos pomoću kojeg se mogu pretraživati ekipe. Klikom na određenu ekipu otvara se detaljni prikaz te ekipe gdje se može odabrati sezona. Nakon odabira sezone prikazuju se podaci te ekipe vezani za tu sezonu.



Slika 11: Detaljni prikaz ekipe - statistika [snimka zaslona]

Na dnu detaljnog prikaza nalazi se grafički prikaz koji pokazuje broj postignutih poena u svakoj utakmici odabrane sezone. Klikom na gumb „Izlaz“ ili van detaljnog prikaza korisnik se vraća na prikaz svih ekipa.



Slika 12: Detaljni prikaz ekipe - grafički prikaz [snimka zaslona]

Dio koda koji prikazuje grafički prikaz na dnu detaljnog prikaza ekipe:

```
<div className="graf scrollable-x">
      <h3 className="tekst-center">Poeni po utakmici kroz sezonu</h3>
```

```
        <XYPlot className="obrub pozadina-neutral-svjetlo padding-small"
              width={800}
              height={400}>
              <YAxis tickPadding={0} />
              <VerticalBarSeries data={this.state.statistikaZaGraf} />
        </XYPlot>
  </div>
```

Grafički prikaz sastoji se od glavne komponente „XYPlot“ u kojoj se nalaze komponente „YAxis“ i „VerticalBarSeries“. Komponenta „YAxis“ određuje hoće li se prikazivati y-os u grafičkom prikazu, dok komponenta „VerticalBarSeries“ određuje stil grafičkog prikaza. U ovom slučaju to je vertikalni stupčasti graf.

Dio koda koji dohvaća podatke za prikaz:

```
var statistika = e.target.value.split('-')[0];
var sezona = e.target.value.split('-')[1];

if (statistika !== '0') {
      axios.get(`/api/statistika/dohvati?id=${statistika}`)
      .then(rezultat => {
            this.setState({
                  statistika: rezultat.data,
                  odabranaSezona: statistika
            });
      });

      axios
            .get(`/api/utakmice/dohvati?sezona=${sezona}&ekipa=${this
            .state.id}`)
      .then(rezultat => {
            if (rezultat.data === 'error') {
                  this.setState({ utakmice: 'error' });
            } else {
                  var statistikaGraf = this.state.statistikaZaGraf;
                  for (var i = 0; i < rezultat.data.length; i++) {
                        statistikaGraf.push({
                              x: i,
                              y:
                  rezultat.data[i].naziv1   ===   this.state.naziv   ?
                  rezultat.data[i].poeniDomacin
                  : rezultat.data[i].poeniGost
                        });
                  }
                  this.setState({ utakmice: rezultat.data });
            }
      });
} else {
      this.setState({
            statistika: [],
            odabranaSezona: ''
      });
}
```

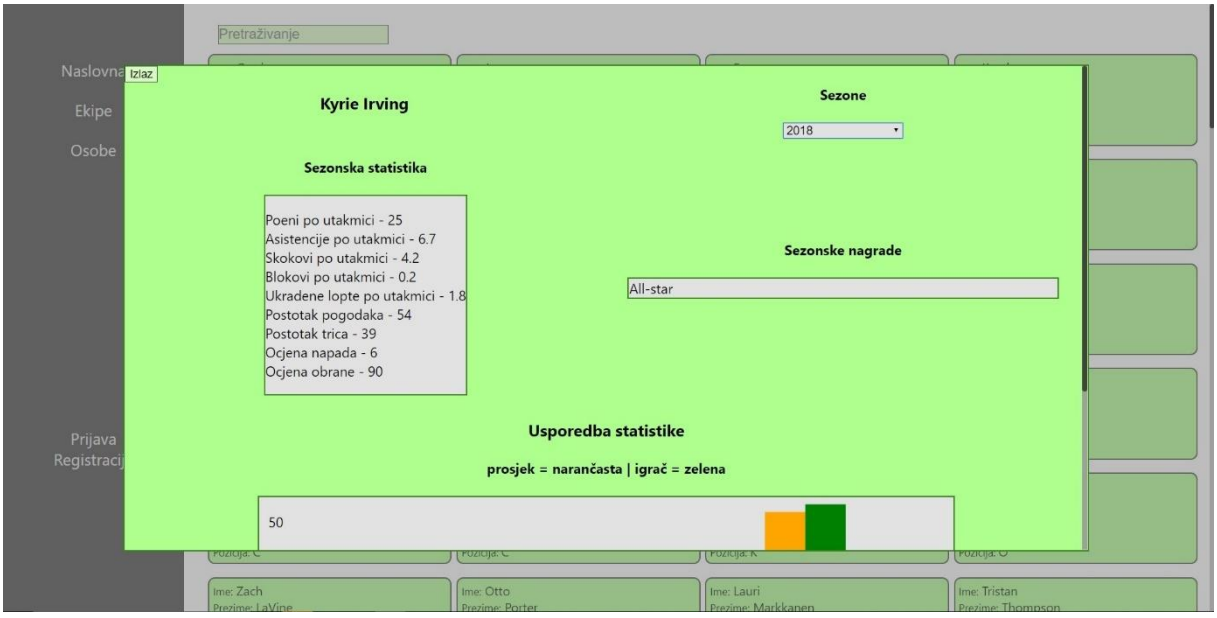
Prvi korak je dohvaćanje sezonske statistike slanjem HTTP GET zahtjeva na poslužitelj i postavljanjem stanja komponente. Nakon toga se dohvaćaju sve utakmice HTTP GET zahtjevom na poslužitelj, nakon čega se puni polje koje se koristi za grafički prikaz na dnu.

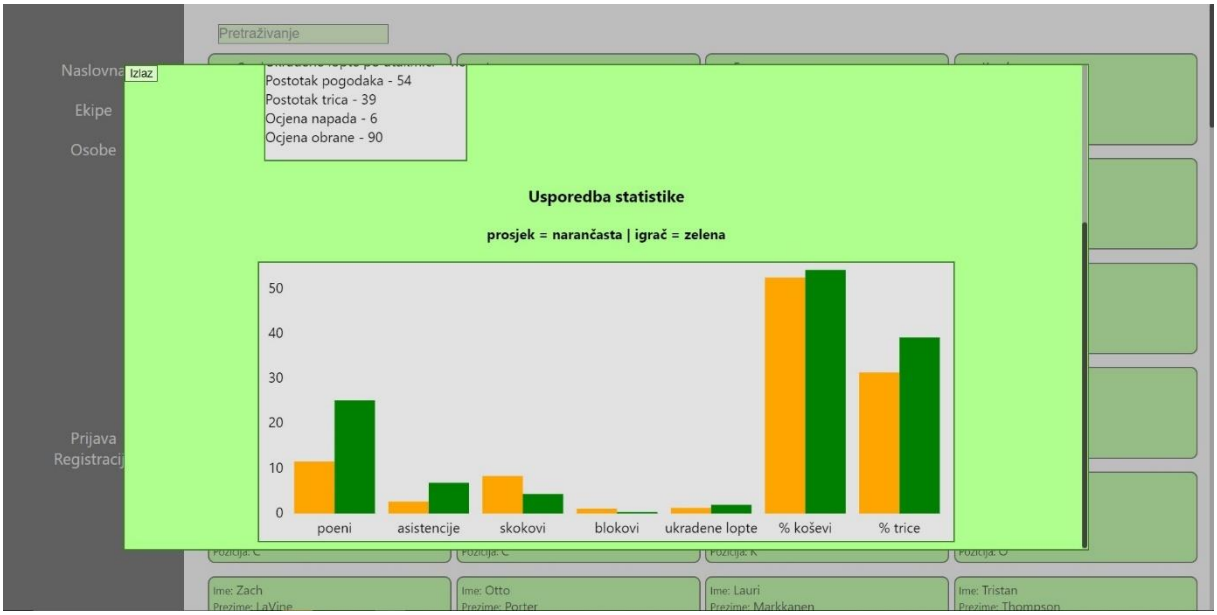
Budući da svaka utakmica ima domaću i gostujuću ekipu, potrebno je da metoda odredi je li odabrana ekipa bila domaćin ili gost utakmice kako bi korektno prikazala poene ekipe.

Prikaz osoba sportske lige vrlo je sličan prikazu ekipa. Također je moguće pretraživati osobe po imenu, prezimenu ili ekipi kojoj pripadaju. Ukoliko se radi o igraču, uz njegove podatke stoji i pozicija koju obično igra.

Slika 13: Prikaz svih osoba [snimka zaslona]

Klikom na određenu osobu otvara se detaljni prikaz koji funkcionira na sličan način kao i detaljan prikaz ekipa. Odabirom sezone prikazuje se sezonska statistika igrača, te sve nagrade koje je te sezone igrač osvojio. Na dnu detaljnog prikaza prikazuje se grafički prikaz s usporedbom igračeve statistike i prosječne statistike.





Slika 15: Detaljni prikaz osobe - usporedba statistike [snimka zaslona]

Budući da se ovdje radi o usporedbi statistika, potrebno je bilo napraviti dva elementa za graf. Dio koda koji prikazuje grafički prikaz na dnu detaljnog prikaza osobe:

```
<XYPlot
      xType="ordinal"
      className="obrub pozadina-neutral-svjetlo padding-small"
      width={1000}
      height={400}>

      <YAxis tickTotal={5} tickPadding={0} />
      <XAxis />

      <VerticalBarSeries
            data={this.state.statistikaProsjek}
            color="orange"
      />

      <VerticalBarSeries
            data={this.state.statistikaZaGraf}
            color="green"
      />
</XYPlot>
```

Dakle, u ovom grafičkom prikazu također se koristi komponenta „XYPlot“, no unutar nje nalaze se dvije komponente „VerticalBarSeries“ – jedna za prikaz statistike igrača, a druga za prikaz prosječne statistike. Bitno je napomenuti kako se ovdje koristi atribut „xType“ komponente „XYPlot“. Njemu je dodijeljena vrijednost „ordinal“. To je napravljeno zato da bi se podaci, odnosno stupci, mogli grupirati po nazivu.

Dio koda koji dohvaća podatke za prikaz statistike:

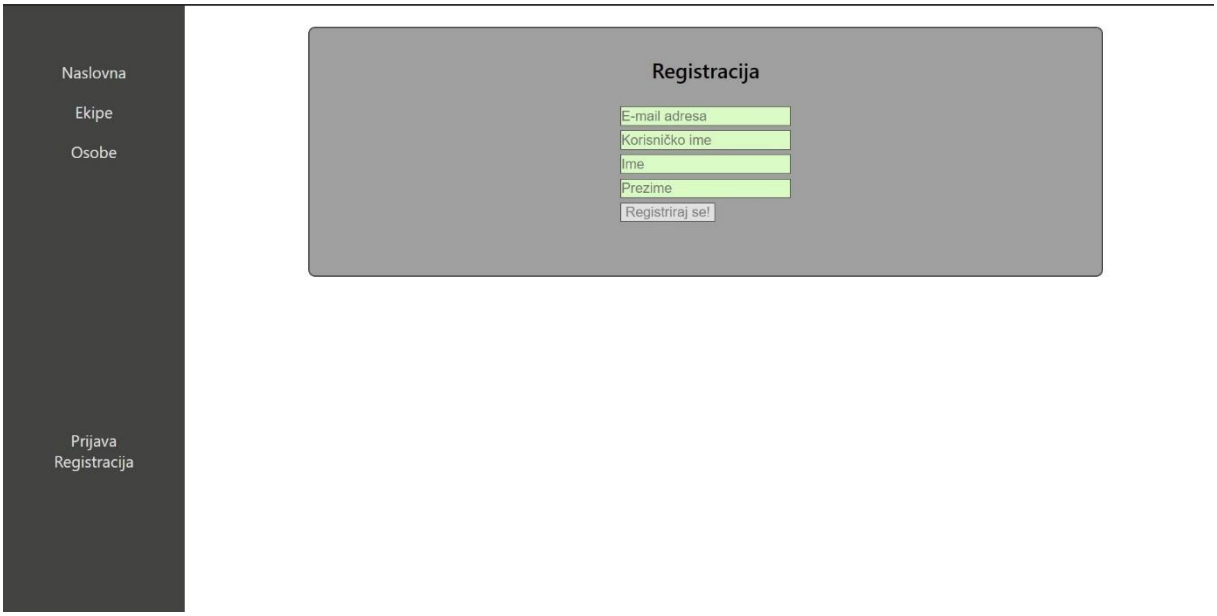
```
var statistika = e.target.value.split('-')[0];
var sezona = e.target.value.split('-')[1];
```

```
this.setState({ odabranaSezona: '' });

if (statistika !== '0') {
      axios.get(`/api/statistika/dohvati?id=${statistika}`)
      .then(rezultat => {
            var statistikaGraf = [];
            statistikaGraf.push
            ({ x: 'poeni', y: rezultat.data.poeni });

            statistikaGraf.push
            ({ x: 'asistencije', y: rezultat.data.asistencije });

            statistikaGraf.push
            ({ x: 'skokovi', y: rezultat.data.skokovi });

            statistikaGraf.push
            ({ x: 'blokovi', y: rezultat.data.blokovi });

            statistikaGraf.push
      ({ x: 'ukradene lopte', y: rezultat.data.ukradeneLopte });

            statistikaGraf.push
      ({ x: '% koševi', y: rezultat.data.postotakPogodaka });

            statistikaGraf.push
      ({ x: '% trice', y: rezultat.data.postotakTrica });

      this.setState({
            statistika: rezultat.data,
            odabranaSezona: statistika,
            statistikaZaGraf: statistikaGraf
      });
      });
      axios
      .get(`/api/nagrade/dohvati?sve=sve&igrac=${this.state.id}
&sezona=${sezona}`)
      .then(rezultat => {
            if (rezultat.data === 'error') {
            } else {
                  this.setState({ nagrade: rezultat.data });
            }
      });
} else {
      this.setState({
            statistika: [],
            odabranaSezona: ''
      });
}
```

Metoda se sastoji od dva HTTP GET zahtjeva. Prvim zahtjevom dohvaća se statistika. Nakon što se dohvati statistika igrača za odabranu sezonu, puni se polje koje se kasnije koristi za grafički prikaz na dnu. Drugi zahtjev dohvaća sve nagrade koje je te sezone igrač osvojio.

Ovo su sve funkcionalnosti kojima može pristupiti običan, neprijavljen, korisnik. Bilo tko može izraditi račun za ovu aplikaciju. Klikom na poveznicu za registraciju otvara se obrazac s nekoliko polja za unos. Prilikom unosa podataka konstantno se provjeravaju razne stvari. RegExp izrazom provjerava se valjanost unesene e-mail adrese. Osim toga, provjerava se

postojanje upisane e-mail adrese i korisničkog imena u sustavu. Ukoliko su sva polja ispunjena, korisnik se može registrirati. Nakon registracije korisnik prima e-mail poruku u kojoj se nalazi lozinka za njegov novi račun.



Slika 16: Obrazac registracije [snimka zaslona]

Dio koda koji obavlja registraciju nakon unosa svih podataka:

```
if (!this.state.greskaEmail && !this.state.greskaKorisnickoIme)
{
      axios
      .post('/api/registriraj', {
            email: this.state.email,
            korisnickoIme: this.state.korisnickoIme,
            ime: this.state.ime,
            prezime: this.state.prezime
      })
      .then(poruka => {
            if (poruka === 'error') {
            this.setState({ poruka: 'Greška kod registracije!' },
                  function() {
                        setTimeout(() => {
                        this.setState({ poruka: '' });
                              }, 2000);
                        });
            } else {
                  this.setState({
                        email: '',
                        korisnickoIme: '',
                        ime: '',
                        prezime: '',
                        poruka:
                        'Poslana je poruka na vašu e-mail adresu
                        s lozinkom za vaš novi račun!'
                  },
                  () => {
                        this.render();
```

```
                        setTimeout(() => {
                              this.setState({ redirect: true });
                        }, 3000);
                  });
            }
      });
}
```

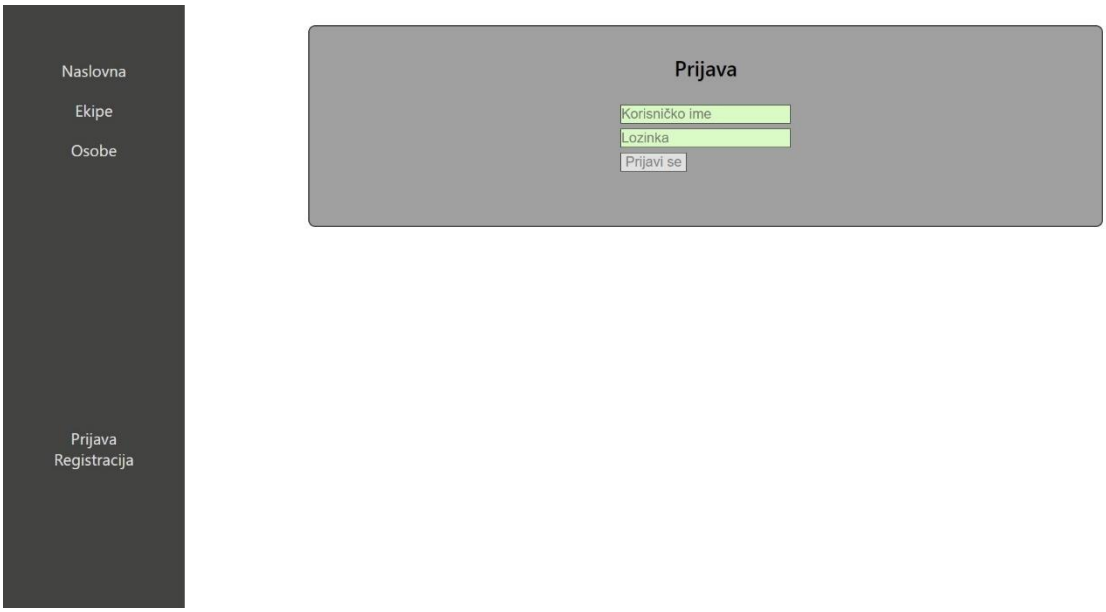
Prvo se provjerava ima li greške u e-mail adresi ili korisničkom imenu. Ukoliko nema grešaka, šalje se HTTP POST zahtjev na poslužitelj sa svim potrebnim podacima. Primljena lozinka prolazi kroz „md5 hash“ kako se bi se što sigurnije zapisala u bazu podataka. Nakon što poslužitelj odradi svoj dio posla, vraća se poruka o uspješnosti operacije.

Dio koda za registraciju na poslužitelju:

```
//rukovatelj putanje

var korisnik = {
      korisnickoIme: request.body.korisnickoIme,
      email: request.body.email,
      ime: request.body.ime,
      prezime: request.body.prezime
};

registracija.RegistrirajKorisnika(korisnik, poruka => {
      response.json(poruka);
});


//funkcija RegistrirajKorisnika iz modula registracija

var lozinka = generator.Generiraj();
var kriptiranaLozinka = md5(lozinka);

var sqlUnos =
      `INSERT INTO Korisnik (tipKorisnika, korisnickoIme, email,
      ime, prezime, lozinka) ` +
      `VALUES(1, '${korisnik.korisnickoIme}',
      '${korisnik.email}','${korisnik.ime}',
      '${korisnik.prezime}','${kriptiranaLozinka}')`;

baza.Upit(sqlUnos, (rezultat, error) => {
      var poruka = error ? 'error' : 'ok';

      if (!error) {
            var naslov = 'Vaša nova lozinka';
            var poruka =
             `Poštovani   ${korisnik.ime}   ${korisnik.prezime},
            lozinka za Vaš novi račun je: ${lozinka}.`;
            mail.PosaljiEmail(korisnik.email, naslov, poruka);
      }

      callback(poruka);
});
```

Nakon uspješne registracije, korisnik se, s dobivenim podacima, može prijaviti u aplikaciju. Prilikom prijave obavlja se provjera postojanja upisane kombinacije korisničkog

imena i lozinke. Nakon uspješne prijave stvara se nekoliko kolačića. Oni sadrže podatke o korisničkom imenu, tipu korisnika i njegovom identifikacijskom broju.



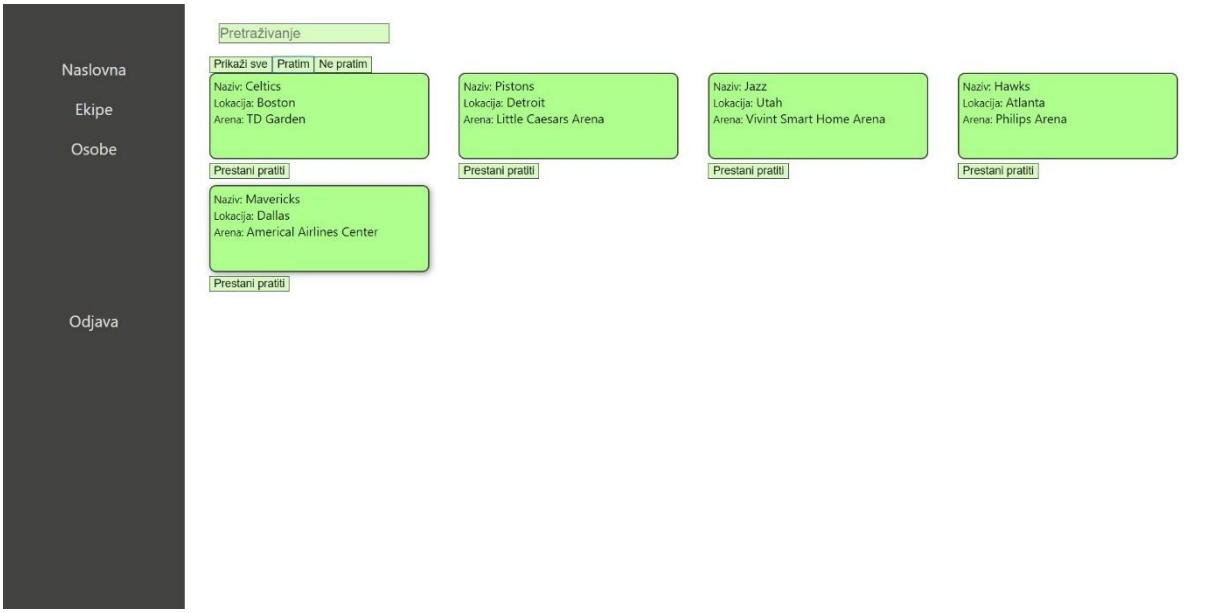
Slika 17: Obrazac prijave [snimka zaslona]

Dio koda na korisničkoj strani koji obavlja prijavu korisnika u aplikaciju:

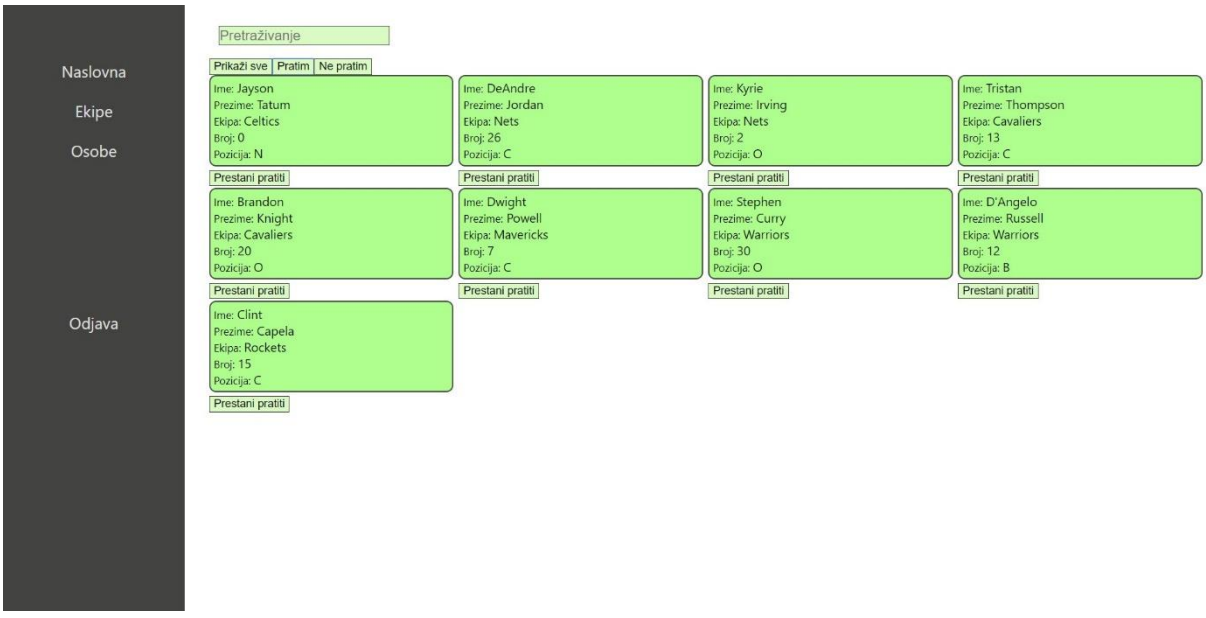
```
axios
.get(
`/api/prijavi?korisnickoIme=${this.state.korisnickoIme}&
lozinka=${this.state.lozinka}`)

.then(response => {
      if (response.data.length !== 0) {
            Cookies.set('korisnik', response.data[0].kIme);
            Cookies.set('id', response.data[0].id);
            Cookies.set('tip', response.data[0].tid);

            this.props.funkcija();
            this.setState({ redirect: true });
      } else {
            this.setState({   poruka:   'Neispravan   unos!'   },
            function() {
                  setTimeout(() => {
                        this.setState({ poruka: '' });
                  }, 2000);
            });
      }
});
```

Metoda prvo šalje HTTP GET zahtjev na poslužitelj kako bi dohvatila korisnički račun s upisanom kombinacijom korisničkog imena i lozinke. Ukoliko postoji takav račun, stvaraju se tri prije navedena kolačića i generira se odgovarajući izbornik za tip korisnika koji se upravo prijavio, te se korisnika preusmjeruje na naslovnu stranu aplikacije. U suprotnom, prikazuje se poruka o nevažećem unosu.

Nakon što se korisnik prijavi, pregledi ekipa i osoba malo se promijene. Prijavljen korisnik može početi pratiti bilo koju ekipu ili osobu za koji je zainteresiran i za koji želi primati objave. Dakle, osim samog prikaza osoba i ekipa, ispod svake osobe i ekipe nalazi se gumb. Klikom na gumb, ovisno o stanju praćenja, prestaje se i počinje pratiti određena osoba ili ekipa. Osim toga, prijavljen korisnik može filtrirati osobe i ekipe na temelju stanja praćenja.



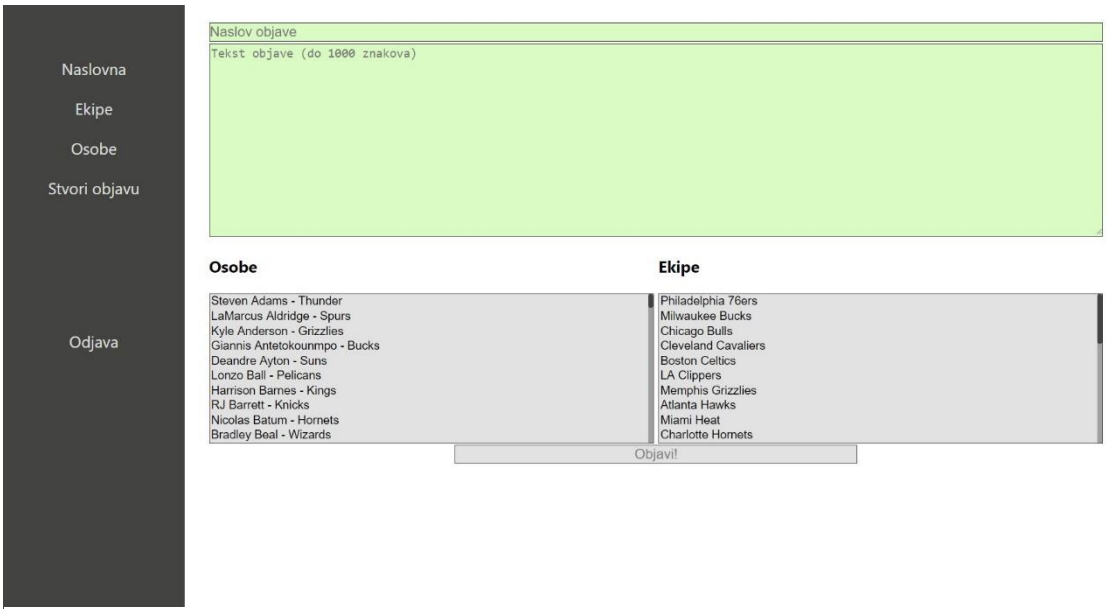
Slika 18: Filtriranje ekipa za prijavljenog korisnika [snimka zaslona]



Slika 19: Filtriranje osoba za prijavljenog korisnika [snimka zaslona]

Korisnik koji ima ulogu moderatora ima na izbor još jednu funkcionalnost. Može dodati objave. Stvaranje objave sastoji se od ispunjenja obrasca. Potrebno je upisati naslov i tekst

objave i označiti barem jednu osobu ili ekipu. Moguće je označiti neograničen broj osoba i ekipa za bilo koju objavu.



Slika 20: Obrazac za stvaranje nove objave [snimka zaslona]

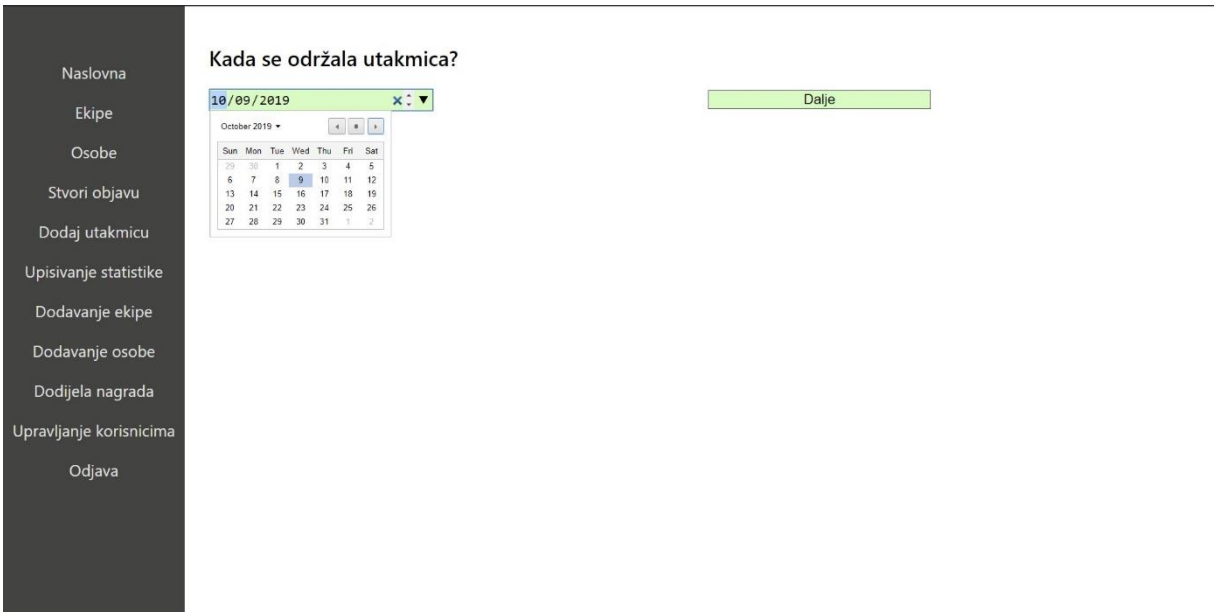
Klikom na gumb „Objavi!“ korisniku se pojavljuje poruka o uspjehu ili grešci, te ga se, ukoliko se radi o uspjehu, preusmjeruje na naslovnu stranu aplikacije.

Metoda koja rukuje s događajem odabira osoba i ekipa:

```
OznaciOdabrano(e) {
	var opcije = e.target.options;
	var ekipe = e.target.id === 'ekipe' ? true : false;
	var oznaceneOpcije = [];
	for (var i = 0; i < opcije.length; i++) {
		if (opcije[i].selected) {
			oznaceneOpcije.push(opcije[i].value);
		}
	}
	if (ekipe) {
		this.setState({
			odabraneEkipe: oznaceneOpcije
		});
	} else {
		this.setState({
			odabraneOsobe: oznaceneOpcije
		});
	}
}
```

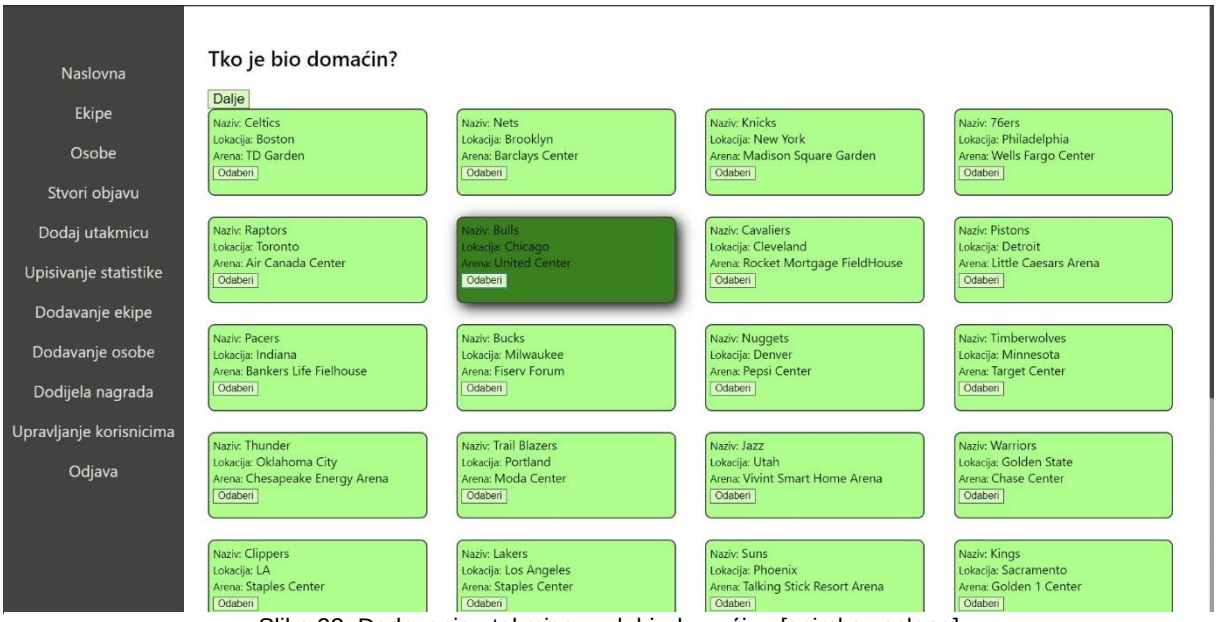
Metoda prolazi kroz sve moguće opcije elementa za izbor i u polje umeće one koje su označene. Na kraju povjerava radi li se o odabiru ekipa ili osoba, te postavlja odgovarajuće stanje komponente.

Administrator sustava ima na izbor sve funkcionalnosti aplikacije. Jedna od funkcionalnost je dodavanje utakmica. Dodavanje utakmica odrađeno je na način da postoji nekoliko koraka u procesu dodavanja. Prvi korak je odabir datuma utakmice.



Slika 21: Dodavanje utakmice - odabir datuma [snimka zaslona]

Nakon odabira datuma, administrator mora odabrati ekipu koja je domaćin utakmice. Bitno je napomenuti da se u ovom koraku ne mogu pojaviti ekipe koje na taj datum već imaju upisanu utakmicu.

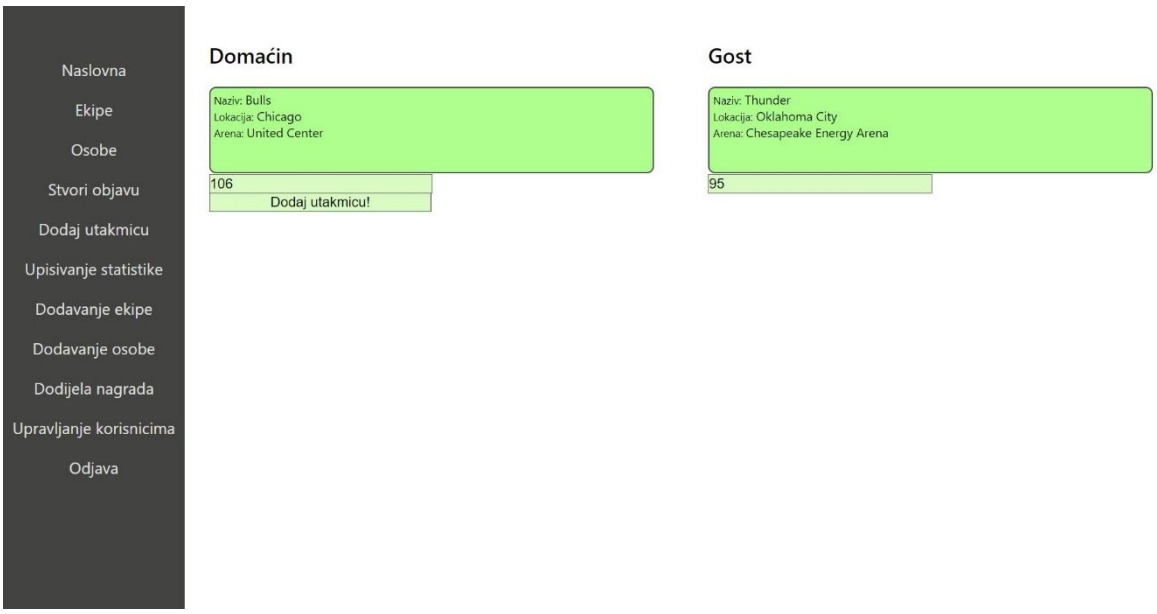


Slika 22: Dodavanje utakmice - odabir domaćina [snimka zaslona]

Nakon odabira domaćina slijedi odabir gostujuće ekipe. Kao i kod prijašnjeg koraka, ovdje se ne pojavljuju ekipe koje na odabrani datum imaju utakmicu. Osim toga, nemoguće je da se u ovom koraku pojavljuje domaćin, iz očitih razloga.

Slika 23: Dodavanje utakmice - odabir gostujuće ekipe [snimka zaslona]

Nakon odabira gostujuće ekipe, dolazi se do završnog koraka gdje se mogu vidjeti odabrane ekipe. Ispod svake ekipe potrebno je upisati ostvarene poene te ekipe u utakmici. Klikom na gumb „Dodaj utakmicu!“ korisnika se preusmjeruje na naslovnu stranu ako nema greške kod unosa u bazu podataka.



Slika 24: Dodavanje utakmice - završni korak [snimka zaslona]

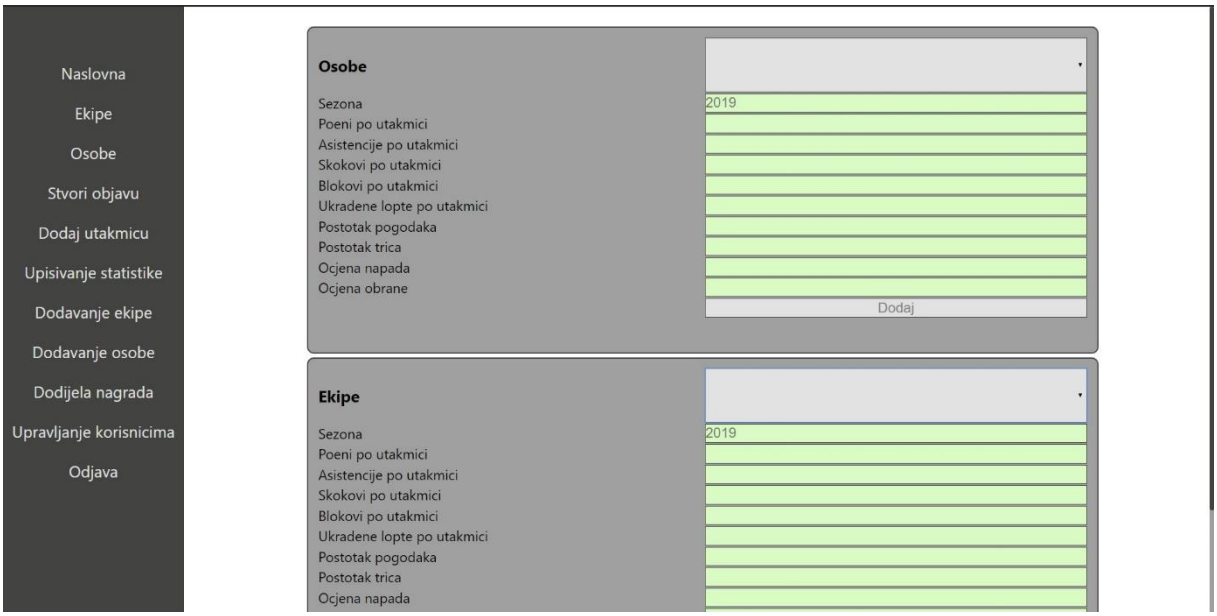
Pobjednika utakmice nije potrebno dodatno označavati. Aplikacija sama određuje pobjednika na temelju unesenih podataka.

Ovakav oblik obrasca moguće je napraviti korištenjem React Router-a. Potrebno je napraviti glavnu komponentu koja sadrži rute do svih koraka obrasca. Dio koda glavne komponente obrasca:

```
<div>
      <Poruka poruka={this.state.poruka} />

      <Route
      exact path="/dodavanje-utakmice"
      component={this.UnosDatuma}
      />

      <Route
      exact path="/dodavanje-utakmice/odabir-domacina"
      component={this.OdabirDomacina}
      />

      <Route
      exact path="/dodavanje-utakmice/odabir-gosta"
      component={this.OdabirGosta}
      />

      <Route
      exact path="/dodavanje-utakmice/zavrsetak"
      component={this.Zavrsetak}
      />
</div>
```

Glavna komponenta u stanju pamti sve unesene podatke – nema potrebe za prosljeđivanjem podataka od jednog do drugog koraka u obrascu. Isto tako, puno je lakše kontrolirati čitav obrazac jer je svako polje za unos pod direktnom kontrolom glavne komponente.

Administrator može dodavati zapise o statistici za igrače i ekipe. Potrebno je ispuniti svako polje nekom vrijednošću. Osim toga, igrač ili ekipa ne mogu imati dva zapisa o statistici u jednoj sezoni. Iz toga razloga aplikacija provjerava postojanje zapisa kod odabira igrača ili ekipe i sezone.



Slika 25: Obrazac za dodavanje statistike [snimka zaslona]

Dio koda koji provjerava postojanje zapisa statistike:

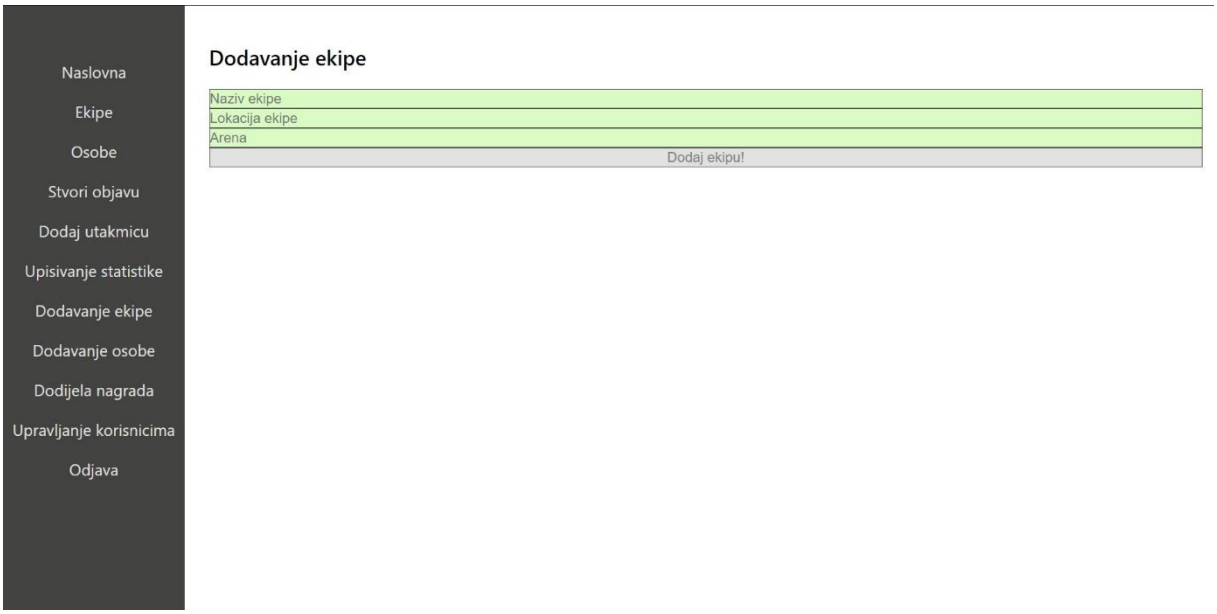
```
//provjera za igrače

axios
.get(
`/api/statistika/osobe/postojanje?osoba=
${this.state.osobaID}&sezona=${this.state.osobaSezona}`
)
.then(rezultat => {
      if (rezultat.data === 'ok') {
            this.setState({ osobaPostoji: false });
      } else if (rezultat.data === 'no') {
            this.setState({ osobaPostoji: true });
      }
});

//provjera za ekipe

axios
.get(
`/api/statistika/ekipe/postojanje?ekipa=
${this.state.ekipaID}&sezona=${this.state.ekipaSezona}`
)
.then(rezultat => {
      if (rezultat.data === 'ok') {
            this.setState({ ekipaPostoji: false });
      } else if (rezultat.data === 'no') {
            this.setState({ ekipaPostoji: true });
      }
});
```

Uz to, administrator može u sustav dodati nove osobe i ekipe. I jedna i druga operacija poprilično su slične, no postoji razlika. Kod dodavanja osobe potrebno je dodati tip osobe. Ukoliko se odabere tip osobe igrač, tada je također potrebno odabrati poziciju koju igrač igra.

Odabirom bilo kojeg drugog tipa osobe potrebno je upisati samo ime, prezime, kojoj ekipi pripada i njen broj. Ovdje se nalaze standardne provjere ispunjenosti svih polja.

Slika 26: Obrazac za dodavanje osobe [snimka zaslona]

Unos nove ekipe malo je jednostavniji. Potrebno je unijeti naziv, lokaciju i naziv arene ekipe. Isto kao i kod dodavanja osoba, ovdje se nalaze standardne provjere ispunjenosti polja.



Slika 27: Obrazac za dodavanje ekipe [snimka zaslona]

Posljednja funkcionalnost aplikacija je upravljanje korisnicima. Točnije, administrator je u mogućnosti promijeniti tip korisnika. Običnog korisnika moguće je pretvoriti u moderatora i s

time mu dati pristup pisanju novih objava. Moderatora je moguće pretvoriti u običnog korisnika i s time mu onemogućiti pisanje novih objava.

Slika 28: Obrazac za upravljanje korisnicima [snimka zaslona]

Ovo je najvjerojatnije najjednostavnija komponenta čitave aplikacije. Radi se o jednostavnim upitima u bazu kako bi se promijenili tipovi korisnika.

Glavni izbornik, odnosno navigacija, aplikacije nalazi se s lijeve strane i konstantno je prisutan. On je svoja posebna komponenta, a također se generira putem HTTP GET zahtjeva na poslužitelj. Na poslužitelju se nalazi datoteka s jednim JavaScript poljem u kojem je svaki element jedan JavaScript objekt. Svaki od tih objekata sadrži sve potrebne informacije za generiranje ovog izbornika. Ovakvom implementacijom moguće je, prilikom prijave i odjave, promijeniti čitav izbornik. Ukoliko korisnik nije prijavljen, u izborniku se nalaze poveznice na obrasce za prijavu i registraciju. U suprotnom se u izborniku nalazi poveznica na odjavu.

Dio koda koji dohvaća poveznice s poslužitelja:

```
var korisnik = Cookies.get('tip') ? Cookies.get('tip') : 4;
korisnik = korisnik.toString();
axios.get(`/api/navigacija/dohvati`).then(response => {
      for (var i = 0; i < response.data.length; i++) {
            if (response.data[i].
            vrsteKorisnika.indexOf(korisnik) !== -1) {
                  response.data[i].prikazi = true;
            } else {
                  response.data[i].prikazi = false;
            }
      }
      this.setState({ poveznice: response.data });
});
```

Prvi korak je određivanje tipa korisnika. Ako korisnik nije prijavljen, odnosno ne postoji kolačić, tada se uzima tip korisnika koji ne postoji u bazi podataka. U ovom slučaju to je tip „4“. Zatim se HTTP GET zahtjevom dohvaća sadržaj prije navedenog JavaScript polja. Nakon toga, kako bi se izbjeglo konstantno dohvaćanje podataka, svakom elementu polja dodaje se svojstvo „prikazi“. Ovakvom implementacijom je, kod mijenjanja izbornika, potrebno samo ponovo proći kroz polje i promijeniti navedeno svojstvo.

Dio sadržaja datoteke s JavaScript poljem elemenata poveznice:

```
var poljeNavigacije = [
      {
            id: 1,
            vrsteKorisnika: '1234',
            naziv: 'Naslovna',
            putanja: '/'
      },
      {
            id: 2,
            vrsteKorisnika: '1234',
            naziv: 'Ekipe',
            putanja: '/ekipe/'
      },
      {
            id: 3,
            vrsteKorisnika: '1234',
            naziv: 'Osobe',
            putanja: '/osobe/'
      }
//ovaj oblik se ponavlja do kraja datoteke
```

Bitno je napomenuti kako se u svojstvu „vrsteKorisnika“ nalaze identifikacijski brojevi tipova korisnika iz baze podataka.

6. Zaključak

Razvoj moderne web aplikacije danas više nije stvar izbora između nekolicine ponuđenih jezika. Iako se JavaScript danas nameće kao očit izbor za razvoj web aplikacije, postoji mnogo različitih biblioteka i okvira koji na fundamentalan način mijenjaju i olakšavaju način, ali i mogućnosti, razvoja web aplikacije. Nastankom Node.js okruženja za izvođenje drastično su povećane mogućnosti JavaScript-a. JavaScript okviri ubrzavaju i olakšavaju razvoj velikih web aplikacija, dok biblioteke nadopunjuju aplikacije nekim funkcionalnostima. Osim toga, Node.js omogućuje i korištenje JavaScript jezika za izradu poslužiteljske strane web aplikacije koja upravlja rutama i HTTP zahtjevima.

JavaScript mogao bi se nazvati jezikom web preglednika. Svaki moderni web preglednik kao primarni jezik koristi JavaScript. Stoga je potrebno, ukoliko se želi biti web razvojni programer, naučiti svaki aspekt JavaScript-a i kako ga najbolje koristiti. Također postoji i mogućnost korištenja JavaScript-a na potpuno drugoj strani arhitekture web aplikacije. Zašto se onda ne bi iskoristila prilika i, umjesto da se uči i koristi neki drugi jezik, iskoristi JavaScript kao jezik poslužitelja web aplikacije? Node.js okruženje pretvara JavaScript u jezik sličan, primjerice, PHP-u i C#-u. Postoje moduli za gotovo svaku primjenu – od spajanja na udaljenu bazu podataka do slanja e-mail poruka.

S druge strane, JavaScript je, iz očitih razloga, užasno popularan jezik. Zbog tolike popularnosti javlja se problem konkurencije. Lakše je pronaći posao ako se zna neki manje popularan jezik, poput Ruby on Rails. Uz to, iz sličnih razloga, postoji ogroman izbor JavaScript biblioteka i okvira, a nastajanje novih čini se kao da je svakodnevna pojava. Netko tko je nov u svijetu web razvoja lako se izgubi u samoj širini ponuđenih izbora i izgubi svu volju učenja jezika. Osim toga, mnogo razvojnih programera jednostavno ne voli JavaScript. Većina programera započinje svoje učenje u nekom drugom programskom jeziku, poput C ili C++. Iz tog razloga teško se naviknuti na neke koncepte koje nude skriptni jezici, točnije JavaScript.

Izradom praktičnog dijela ovog rada naučio sam mnogo novih stvari. Između ostalog, poznavanje samog JavaScript-a puno je veće. Isprobavanjem i korištenjem raznih okvira upoznao sam drugu stranu razvoja web aplikacija. Svidjelo mi se korištenje Node.js JavaScript okruženja kod izrade poslužiteljske strane web aplikacije. Express okvir definitivno donosi jednostavnost, ali i moć, kod izrade JavaScript poslužitelja sa svojim konceptom posredničkog softvera.

React, iako nije okvir, omogućuje dodavanje velikog broja funkcionalnosti u web aplikaciju, a sam razvoj web aplikacije njegovim korištenjem postaje puno jednostavniji. Iz razloga što nije okvir lakše ga se naučiti koristiti – ne postoji stroga struktura koje se mora

pridržavati kod izrade aplikacije, a većim dijelom se koriste standardne, ugrađene, JavaScript funkcije i klase.

Sve u svemu, JavaScript je jezik koji se kod razvoja web aplikacije, neovisno o njenoj veličini i opsegu, jednostavno mora znati koristiti. Neke stvari koje sam po sebi JavaScript ne može učiniti mogu se nadopuniti drugim jezicima, no definitivno se isplati barem isprobati neke od pregršt JavaScript biblioteka i okvira.

Popis literature

[1] C. Emery, „A Brief History of Web Development“, 08.07.2016. [Na internetu]. Dostupno: https://www.techopedia.com/2/31579/networks/a-brief-history-of-web-development [pristupano 10.07.2019.]

[2] T. Ater, *Building Progressive Web Apps*. Sebastopol, CA, USA: O'Reilly Media. 2017.

[3] Pew Research Center, „Mobile Fact Sheet“, 12.06.2019. [Na internetu]. Dostupno: https://www.pewinternet.org/fact-sheet/mobile/ [pristupano 11.07.2019.]

[4] D. W. Barron, *The World of Scripting Languages*. University of Southampton, UK: John Wiley & Sons, 2000.

[5] Z. Bloom, „A Brief History of Weird Scripting Languages on the Web“ [Blog post]. Dostupno: https://eager.io/blog/a-brief-history-of-weird-scripting-languages/ [pristupano 13.07.2019.]

[6] World Wide Web Consortium, „Scripts in HTML documents“ [Na internetu]. Dostupno: https://www.w3.org/TR/html4/interact/scripts.html [pristupano 15.07.2019.]

[7] Scottish Qualifications Authority, „Main Differences between Interpreters and Compilers“ [Na internetu]. Dostupno: https://www.sqa.org.uk/e-learning/ClientSide01CD/page_17.htm [pristupano: 16.07.2019.]

[8] K. Tatroe, P. MacIntyre, R. Lerdorf, *Programming PHP*. Sebastopol, CA, USA: O'Reilly Media. 2013.

[9] PHP Manual, „History of PHP“ [Na internetu]. Dostupno: https://www.php.net/manual/en/history.php.php [pristupano 16.07.2019]

[10] B. M. Kuhn, „Picking Up Perl – Background of Perl“, 2001. [Na internetu]. Dostupno: http://www.ebb.org/PickingUpPerl/pickingUpPerl_9.html [pristupano 19.07.2019.]

[11] L. Wall, T. Christiansen, J. Orwant, *Programming Perl.* Sebastopol, CA, USA: O'Reilly Media. 2000.

[12] D. Crockford, *JavaScript: The Good Parts.* Sebastopol, CA, USA: O'Reilly Media. 2008.

[13] M. Haverbeke, *Eloquent JavaScript: A Modern Introduction to Programming.* [Besplatna knjiga na internetu]. Dostupno: https://eloquentjavascript.net/

[14] J. Robbie, „What is the Document Object Model?“ [Na internetu]. Dostupno: https://www.w3.org/TR/WD-DOM/introduction.html [pristupano 19.07.2019.]

[15] A. MacCaw, *JavaScript Web Applications.* Sebastopol, CA, USA: O'Reilly Media. 2011.

[16] A. Dalia, „The Importance of Graphic design for Business“ 20.06.2017. [Na internetu]. Dostupno: https://www.ikf.co.in/importance-of-graphic-design-for-business-in-2019/ [pristupano 21.07.2019.]

[17] Node.js Foundation, „About Node.js“ [Na internetu]. Dostupno: https://nodejs.org/en/about/ [pristupano 21.07.2019.]

[18] S. Morris, „Tech 101: JavaScript Frameworks vs Libraries – What's the Difference?“ 04.06.2019. [Na internetu]. Dostupno: https://skillcrush.com/2019/05/22/javascript-frameworks-vs-libraries/#library [Pristupano 22.08.2019.]

[19] Educational Ecosystem, „React.js History“ [Na internetu]. Dostupno: https://www.education-ecosystem.com/guides/programming/react-js/history [Pristupano 23.08.2019.]

[20] D. Jöch, „Functional vs Class-Components in React“ 11.07.2019. [Na internetu]. Dostupno: https://medium.com/@Zwenza/functional-vs-class-components-in-react-231e3fbd7108 [Pristupano 24.08.2019.]

[21] Facebook, „Introducing JSX“ [Na internetu]. Dostupno: https://reactjs.org/docs/introducing-jsx.html [Pristupano 24.08.2019.]

[22] M. Maki, „A Brief Overview of React Router and Client-Side Routing“ 04.12.2017. [Na internetu]. Dostupno: https://medium.com/@marcellamaki/a-brief-overview-of-react-router-and-client-side-routing-70eb420e8cde [Pristupano 24.08.2019.]

[23] E. M. Hahn, *Express In Action*, Shelter Island, NY, USA: Manning Publications. 2016.

[24] A Curious Animal, „Using async/await in ExpressJS middlewares“ [Na internetu]. Dostupno: http://www.acuriousanimal.com/2018/02/15/express-async-middleware.html [Pristupano 14.09.2019.]

[25] jQuery Foundation, „jQuery API Documentation“ [Na internetu]. Dostupno: https://api.jquery.com/ [Pristupano 25.08.2019.]

[26] P. Grosvenor, „20 brilliant jQuery plugins“ 11.10.2018. [Na internetu]. Dostupno: https://www.creativeblog.com/jquery/top-jquery-plugins-6133175 [Pristupano 25.08.2019.]

[27] P. Kumar, „AngularJS Vs. Angular 2 Vs. Angular 4: Understanding the Differences“ 11.07.2019. [Na internetu]. Dostupno: https://www.simplilearn.com/angularjs-vs-angular-2-vs-angular-4-differences-article [Pristupano 25.08.2019.]

[28] L. Marx, „How to use *ngif else in Angular“ 11.05.2019. [Na internetu]. Dostupno: https://malcoded.com/posts/angular-ngif-else/ [Pristupano 25.08.2019.]

[29] Google, „Angular Architecture Overview“ [Na internetu]. Dostupno: https://v2.angular.io/docs/ts/latest/guide/architecture.html [Pristupano 14.09.2019.]

[30] E. Timberg i sur., „Chart.js documentation“ [Na internetu]. Dostupno: https://www.chartjs.org/docs/latest/ [Pristupano 13.09.2018.]

[31] Anychart, „Anychart Documentation – Quick Start“ [Na internetu]. Dostupno: https://docs.anychart.com/Quick_Start/Quick_Start [Pristupano 14.09.2019.]

[32] Anychart, „Anychart Documentation – Pyramid Chart“ [Na internetu]. Dostupno: https://docs.anychart.com/Basic_Charts/Pyramid_Chart [Pristupano 14.09.2019.]

[33] Anychart, „Anychart Documentation – Bubble Chart“ [Na internetu]. Dostupno: https://docs.anychart.com/Basic_Charts/Bubble_Chart [Pristupano 14.09.2019.]

[34] A. Bulyonov i sur. „React-vis Documentation“ [Na internetu]. Dostupno: https://uber.github.io/react-vis/documentation/welcome-to-react-vis [Pristupano 14.09.2019.]

[35] RemoteMySQL. *RemoteMySQL* [Usluga web hostinga]. Dostupno: https://remotemysql.com [pristupano 15.09.2019.]

[36] Heroku. *Heroku* [Usluga web hostinga]. Dostupno: https://heroku.com [Pristupano 15.09.2019.]

Popis slika

Popis tablica