Izrada interaktivnog korisničkog sučelja primjenom Javascript biblioteke React

Kantolić, Marija

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

MARIJA KANTOLIĆ

IZRADA KORISNIČKOG SUČELJA U OKRUŽENJU REACT.JS

Završni rad

Pula, lipanj 2019.

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

MARIJA KANTOLIĆ

IZRADA KORISNIČKOG SUČELJA U OKRUŽENJU REACT.JS

Završni rad

JMBAG: 0303063270, redovita studentica
Studijski smjer: Sveučilišni preddiplomski studij Informatika

Predmet: Programiranje
Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijsko-komunikacijske znanosti
Znanstvena grana: Informacijski sustavi I informatologija
Mentor: doc. dr.sc. Tihomir Orehovački

Pula, lipanj 2019.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisana Marija Kantolić, kandidat za prvostupnika informatike ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima, te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, lipanj, 2019. godine

IZJAVA

o korištenju autorskog djela

Ja, Marija Kantolić dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom „Izrada korisničkog sučelja u okruženju React.js“ koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli,_____(datum)

Potpis

_____

1. Uvod

Od davnina, čovječanstvo je bez prestanka razvijalo i unaprjeđivalo svoje iskustvo življenja. Prvi početak je počeo s izradom alata te se od običnog rasporeda kuća razvilo do gradova. Daljnjim razvojem znanja, nastala je i potreba za boljom kvalitetom života. Ljudski narod od samih početaka izrađuje proizvode, te je bilo mnogo neuspjelih pokušaja kako bi napravili uspješne proizvode koje i danas koristimo. Rad na strojevima je doprinio razvoju tehnologije. Izumom i napretkom tehnologije, ljudski život je postao jednostavniji. Jednostavniji život više nije tražio toliko fizičkog rada u tvornicama jer se razvila automatizacija i s tim su se razvila i nova područja, te najpoznatije je informatičko područje. Kako se ljudi najviše oslanjaju na svoj vid, novi električni izumi kojima je čovjek upravljao, trebali su biti vizualno ugodni i primamljivi, te s tom potrebom, razvio se dizajn korisničkog sučelja. Raznovrsnim testiranjem i unaprjeđivanjem, dizajneri su razvili niz određenih pravila kako bi svaki budući proizvod bio namijenjen za ugodno rukovanje istim. Ljudi koji su razvijali prvobitne informatičke proizvode morali su imati određeno znanje o svom području rada, te se tada počelo razvijati programersko i računalno znanje, koje se niti danas ne prestaje razvijati.

Cilj ovog završnog rada je pobliže objasniti izradu korisničkog sučelja na primjeru internet stranice pomoću JavaScript biblioteke React.js. Ovaj rad je podijeljen u više manjih dijelova. Prvi dio se odnosi na teorijski dio gdje je objašnjen što je točno korisničko sučelje i kako nastaje izrada istoga. U drugom dijelu, koji se sastoji od više kategorija, detaljnije su objašnjeni standardi i biblioteke koje su potrebne za pravilno korištenje React.js. Biblioteka React.js nije zasebna biblioteka i ona se sastoji od određenih ECMAScript standarda koje omogućuju učinkovitije programiranje. Svaki kod, funkcija, klasa u React.js je ili zasebna komponenta ili dio određene komponente. Također je potrebno koristiti JSX sintakse i pretvarač Babel koji omogućuju jednostavniju izradu korisničkog sučelja, te pravilno prikazivanje dizajna na svim preglednicima. Firebase-ova baza podataka je Google-ovo besplatno pohranjivanje podataka na internetu. U nastavku završnog rada objašnjeno koje Firebase opcije nudi, te je u praktičnom dijelu rada

prikazano i objašnjeno kako se služi s instancama koje nam je Firebase omogućio za lakše povezivanje s istim.

Treći dio je praktičan dio, te se odnosi na izradu internet stranice i koje su komponente korištene pri izradi. Detaljnije su prikazani kompleksniji dijelovi koda za svaku datoteku koja se koristila pri izradi stranice. Također je prikazana kako je napravljeno povezivanje na Firebase bazu podataka, te implementacija s React – Redux, predviđeni spremnik koji pohranjuje određene podatke s aplikacije. Redux pomaže pri kodiranju aplikacije koje se dosljedno ponašaju, mogu se pokrenuti u različitim okruženjima (klijent, poslužitelj i nativnim okruženjima) i aplikacije koje koriste Redux je lako testirati.

Internet stranica je napravljena za ljubitelje fotografije. Ljudi koji se profesionalno bave fotografijom i zarađuju od nje, te imaju određeno znanje koje bi svaki početnik htio imati. Ideja za ovu stranicu je nikla iz takve želje za znanjem. Većina današnjih stranica koje su namijenjene za pomoć mladim fotografima je napisana na engleskom jeziku, te je ponekad teže razumjeti kako pravilno baratati s digitalnim fotoaparatom. Navedena internet stranica pod nazivom “Kroz objektiv” ima određene komponente koje bi pomogle svakom korisniku lako snalaženje po stranici.

2. Korisničko sučelje i njegova izrada

U ovom dijelu završnog rada, detaljnije je objašnjeno što je korisničko sučelje, i zašto je izrada dobrog dizajna bitna za uspjeh svakog proizvoda

2.1. Korisničko sučelje

Izgradnja korisničkog sučelja u današnje vrijeme je postala bitna stavka svakog softverskog proizvoda jer ona ključna za njegov uspjeh.

Korisničko sučelje je premosnica između računalnog sustava ili programa i korisnika. Pomoću korisničkog sučelja upravljamo računalom, koristeći se pritom ulaznim uređajima poput miša, tipkovnice i touch-screena (zaslona osjetljivog na dodir). Izlazni uređaj na kojem se vizualno manifestiraju brojne naredbe, kao i naše akcije u današnjem korisničkom sučelju je monitor [1]. Iz navedene definicije, korisničko sučelje možemo definirati kao sve grafičke, tekstualne i audiovizualne informacije koje informatički sustav predstavlja korisniku. Korisnik tim sustavom upravlja putem raznih ulaznih jedinica kao što su miš, tipkovnica ili mikrofon i pri tome obavlja određenu interakciju i zahtijeva određeni rezultat od navedenog sustava. Kvaliteta korisničkog sustava je od velike važnosti za kvalitetu cijelog sustava, jer korisnik kvalitetu sustava češće sudi prema kvaliteti, to jest dizajnu korisničkog sučelja nego koje funkcionalnosti taj sustav nudi. Danas, svaki sustav u kojem se koristi informatička tehnologija je korišteno korisničko sučelje. Važno je zato dobro i precizno analizirati, planirati, dizajnirati i izraditi korisničko sučelje.

Pojavom novih tehnologija i uređaja, pametni telefoni, tableti, pametni automobili i sl., te promjenom ljudskog ponašanja i prilagođavanjem tim tehnologijama, oblikovanje korisničkog sučelja poprima značajnu važnost u skladu sa očekivanjima korisnika. Mobilni telefoni postaju središte svakodnevice mnogih ljudi, a rastom tehnologije i mogućnostima

koje ona pruža sve više vremena trošimo držeći mobilni uređaj u rukama i gledajući u njegov sadržaj. Razvojem interneta i ostvarivanjem komunikacije i mnogih sličnih iskustava koje korisnik može doživjeti putem svog mobilnog uređaja i internetske veze, korisnici postaju zahtjevniji i njihova očekivanja svakim danom rastu. Dakle, kako se razvija tehnologija tako se razvija i društvo. Vrlo je važno softverski produkt predstaviti korisniku na jedinstven i originalan način, a pružiti mu kvalitetu i pozitivno iskustvo jer se tako postiže povjerenje. Pri kreiranju bilo kakvog korisničkog sučelja važno je pratiti, planirati i prilagođavati korisnikovo kretanje kroz sustav, te takvo kretanje učiniti što kraćim i jednostavnijim. Ukoliko ignoriramo zahtjeve korisnika, dolazi do zbunjivanja i frustriranje istih, pa ni vizualno najljepše sučelje neće pomoći pri tom kretanju. Estetski bogato sučelje bogato je jedino u onom trenutku kada ispunjava potrebu korisnika.

2.2. Izrada korisničkog sučelja

Izrada korisničkog sučelja se može promatrati kao iterativni proces podijeljen na nekoliko osnovnih aktivnosti te ju čine: analiza korisnika, izrada prototipa i evaluacija rješenja. Svako korisničko sučelje je namijenjeno korisniku i i izrađuje se prema njegovim željama ili karakteristikama. Pri izradi je potrebno analizirati korisnikove zahtjeve i prilagoditi sučelje budućim korisnicima i njihovim potrebama i sposobnostima. Izrada prototipa je bitna komponenta svakog planiranja jer se u ovoj fazi odlučuje o mogućem razmještaju elemenata u grafičkom sučelju. Nakon dobivenog dizajna, takozvanog prototipa, prema njemu se počinje razvijati određeni sustav koji se testira i dodatno unapređuje. Krajni rezultat izrade prototipa i evaluacije je implementacija korisničkog sučelja koje će pratiti zahtjeve korisnika, služit će korisniku u potpunosti i omogućiti mu laganu interakciju sa sustavom i nesmetano obavljanje akcija. Svaki dizajn ima unaprijed određena pravila, standarde i principe koje se trebaju slijediti. Prvo i osnovno je da sučelje treba biti jednostavno i lako razumljivo, to jest, da se korisnik vrlo lako može snalaziti po danom mu sučelju. Dizajneri za ostvarivanje tog cilja slijede pravilo “tri klika”. Navedeno pravilo postoji jer se želi postići da korisnik u najviše tri klika mišem ili na tipkovnici ili po ekranu dođe do glavnog izbornika. Korisničko sučelje mora biti dizajnirano na način da

bude što prihvatljivije moguće korisnicima koji ga koriste. Sučelje mora biti pouzdano kako bi bilo što manje grešaka ili “iznenađenja” pri korištenju istoga.

Pregledno, jednostavno organizirano korisničko sučelje s lako uočljivim funkcijama i cjelovitom strukturom je ključno za kvalitetu i ono se postiže Gestalt psihologijom. Geštaltizam drži da je cjelina važnija od dijelova, doživljajne cjeline, primjerice perceptivni doživljaji izazvani strukturama podražaja okoline, nisu običan zbroj pojedinih elemenata, već predočuju suprasumativnu kvalitativno novu cjelinu, koja se ne može prepoznati u njezinim sastavnim dijelovima [2]. Ljudski mozak ima sposobnost da sagleda cjelinu prije nego što opazimo detalje. Navedena psihologija se može primijeniti pri izradi internet stranica u slučaju da imaju nedostatak pri dizajniranju. Bilo da je u pitanju siromašna grafika, pogrešan izbor tipografije, nepravilno korištenje razmaka, ili pak nedovoljno funkcionalan JavaScript kod, može srušiti cjelinu.

Navedena su tri načela za dobar grafički dizajn internet stranice. Prvo načelo je neposredna blizina, to jest grupiranje. Kada imamo grupu predmeta, smatramo ih kao formiranu grupu. Pri dizajniranju internet stranica, grupiranje možemo primjeniti kod navigacijskih poveznica. Kod grupiranja poveznica možemo ih grupirati interno, to jest, mogu se slične poveznice rasporediti na jednu stranu, raditi kategorije s njima, stavljati ih u podkategorije i slično. Drugo načelo je načelo sličnosti. Dijelovi u cjelini trebaju biti sukladni kako bi se postigao određeni cilj. Opažanjem grupiramo stvari koje su slične jedna drugoj. To je ujedno i razlog zašto su poveznice plave boje ili zašto se ti isti linkovi označavaju s istom bojom. Sličan izgled uključuje i sličnu funkciju. Treće načelo je percepcija slike. Naša percepcija slike nam omogućuje da organiziramo ono što vidimo na osnovu odnosa objekata jednih prema drugima. Predmeti u prvom planu bi trebali biti uočljiviji od njihovog okruženja ili pozadine. Pogrešna primjena navedenih osnovnih načela može izazvati neželjene efekte [3].

U današnje vrijeme imamo veliku raznolikost među korisnicima, te je bitno obratiti pažnju za koju skupinu korisnika radimo dizajn i istovremeno da bude pristupačno i lako razumljivo što većem broju korisnika. Neovisno o podjeli, korisnici žele povratnu informaciju, to jest rezultat od svojih akcija, te im je stoga to potrebno i pružiti. Kako bi cijeli sustav bio smislena cjelina, treba brinuti o ustrajanome sustavu jer to korisnici očekuju [4].

Korisničko sučelje je mjesto gdje se korisnik prvi put susreće s softverskim sustavom. Dobro je poznata stvar da knjigu ne smijemo suditi po koricama, te tako niti sustav ne bi smjeli suditi po lošem dizajnu sučelja. Činjenica je da većina korisnika kvalitetu softverskog sustava odabire ili osuđuje prema lijepim bojama u korisničkom sučelju, a ne prema funkcionalnostima koje isti nudi. Korisničko sučelje ne služi samo za vizualni identitet, već je važan element za komunikaciju između korisnika i sustava. Pri dizajniranju korisničkog sučelja važno je prikupiti što više informacija od budućih korisnika kako bi im omogućili potpunu i jednostavnu interakciju sa sustavom [5].

3. JavaScript i React.js

Današnje JavaScript aplikacije imaju iste komponente. Prva je struktura aplikacije koja je modularna i njeni moduli se pišu u posebnim .js datotekama. Kako bi ti modeli komunicirali jedni s drugima, oni se uvoze u druge module i na taj način se stvara stablo zavisnosti. Druga komponenta je korištenje biblioteka koje rješavaju specifičan problem. Biblioteke su nastale kako bi nam olakšale samo programiranje i reducirali kod, jer mnogo problema s kojima se susrećemo pri programiranju su već riješene pomoću određenih biblioteka. Prijevod (eng. "Transpiling") je proces koji je omogućen od strane JS kreatora, da svaki JS kod bude podržan i preveden na svakom pregledniku. Objedinjavanje (eng. "Bundling") ili grupiranje sličnih .js datoteka u jednu mapu, te najčešće uključuje i razne optimizacije koda, kao što je minifikacija koda [12].

3.1. Razlika između frameworka i biblioteke

Okvir (eng. "Framework") je arhitekturalni obrazac pisan u JavaScript programskom jeziku koji definira dizajn aplikacije. Što znači da pri izradi aplikacije koja koristi određeni framework se nužno mora pratiti struktura koju on specificira. Na taj način se uštedi na vremenu i razvoju jedne aplikacije, jer nije nužno pisati vlastiti JavaScript kod, nego se koriste već postojeći predlošci po vlastitom nahođenju. Angular.js, Knockout.js, Ember.js, Vue.js su neki od poznatijih JavaScript framework-ova.

JavaScript biblioteka je skup funkcija pisanih u JS programskom jeziku čije korištenje olakšava programiranje i izradu aplikacija. Svatko može napisati biblioteku, povezati ih u jedan paket i podijeliti s drugima na korištenje. Poznatije biblioteke su jQuery, D3.js, Moment.js.

Bitna razlika između okvira i biblioteke je ta da je framework nastao kako bi se izradila aplikacija, biblioteka se koristi za kod s kojim će se moći izraditi aplikacija.

3.2. React.js

JavaScriptova biblioteka React.js je uvelike unaprijedila izradu određenih aplikacija. Inženjeri Facebook-a su 2013. godine kreirali biblioteku kako bi riješili postojeći problem - kako efikasno i jednostavno upravljati prikazom tradicionalnih Model-Pogled-Kontroler (eng. Model-View-Controller, MVC) aplikacija.

Sam React.js nije dovoljan kako bismo izradili kompletnu i funkcionalnu aplikaciju, on samo osigurava fleksibilan način izrade korisničkog sučelja i upravlja prikazom iste. Veliki izazov kod složenog korisničkog sučelja je ažuriranje stanja elemenata stranice kada se model ili element promijeni. React.js osigurava prikaz prezentacijskog jezika za izradu web stranica (eng. Hypertext Markup Language, HTML) i deklarativno aplikacijsko programsko sučelje (eng. Application programming interface, API), što znači da mi kao programeri, ne moramo brinuti što će se točno promijeniti tijekom svakog ažuriranja, jer React.js ažurira samo elemente koji su se promijenili. React.js ima dva bitna svojstva. Prvo je skalabilnost koja se odnosi na sposobnost sustava da nastavi funkcionirati neovisno o promjeni njegove veličine, i druga je održivost, sposobnost sustava da se nakon prekida vrlo lako i brzo vrati u operativno stanje [6].

3.3. Virtualni DOM

Objektni model dokumenta (eng. Document Object Model, DOM), je sučelje koji omogućava skripti da dinamički pristupi i izmijeni sadržaj, strukturu ili stil HTML dokumenata. Konstantno mijenjanje DOM-a nije brza operacija i kod složenih aplikacija njegovo ažuriranje može biti sporo i naporno. jQuery i druge biblioteke postoje kako bi se manipulirale DOM-om, ali struktura od DOM-a je i dalje, sama po sebi poprilično ograničena.

Kako bi se izbjegao spomenuti nedostatak, React.js koristi virtual DOM, koji predstavlja jednostavnu apstrakciju HTML DOM-a. Virtualni DOM možemo zamisliti kao kopija lokalne memorije DOM-a, i React koristi tu kopiju kako bi odradio sva izračunavanja koja su potrebna za prikaz stanja elemenata na korisničkom sučelju. Koristi i diferencijalni

algoritam koji omogućuje da će ažuriranje pojedinih elemenata biti predviđeno, a u isto vrijeme i dovoljno brzo za složene aplikacije koje zahtijevaju visoke performanse rada [9].

3.4. Props, State objekti

Prilikom korištenja React.js biblioteke, susrećemo se s dvije najčešće riječi koje su nam bitne za programiranje, a to su svojstva (eng. props) i stanje (eng. State) objekti. Props, što je skraćenica od properties označavaju kako komponente komuniciraju jedna s drugom. Props se koristi kako bi se proslijedili podaci na način da se na dijete (eng. Child) komponentu, koja se nalazi unutar metode koja izvršava naredbe (eng. Render) roditelj (eng. Parent) komponente, doda atribut i pridružena mu vrijednost pri čemu će atribut označavati ime prop parametra. Vrijednost im se pri tome ne mijenja. Protok podataka je jednosmjeran, to jest, podaci koje komponente primaju je od strane parent-a [15].

U slučaju da imamo podatke koji će se konstantno ažurirati, tada koristimo state objekt i pripadajuću mu funkciju pohrani stanje (eng. setState). State objekt sadrži privatne informacije o komponenti, jer je on kreiran unutar komponente i moguće ga je promijeniti sa funkcijom setState (). Iz navedenog razloga, komponenta se automatski renderira, po potrebi se može koristiti funkcija prethodno stanje (eng. prevState) koja služi za povratak na prijašnje stanje.

Razlika između props i state objekata, iako oni slično djeluju, je taj da se state objekti kreiraju i nalaze u jednoj komponenti te se može promijeniti i ažurirati kada želimo, dok props objekti možemo samo čitati (eng. read-only).

4. ECMAScript

JavaScript je službeno objavljen 1996. godine zajedno s Netscape-om2. Originalan naziv je bio LiveScript, te je ubrzo preimenovan u JavaScript radi stjecanja popularnosti kroz tadašnji popularan jezik Java i treba napomenuti da ova dva programska jezika imaju malo toga zajedničkoga.

Pri izradi internet stranica i aplikacija najčešće se koristi JavaScript uz HTML i stilski jezik (eng. Cascading Style Sheets, CSS) koje su jezgrene tehnologije. JavaScript se izvršava na strani klijenta u svrhu definiranja kako će se pojedini element ponašati prilikom određene akcije. Širenjem popularnosti i većim korištenjem u izradi aplikacija dovelo je do formacije specifikacija. Danas je za standardizaciju skriptnih jezika, pa tako i JavaScripta zadužena organizacija ECMA i njeni standardi se objavljuju pod nazivom ECMAScript.

U nastavku ovog rada će biti objašnjeni osnovni elementi ECMAScrpt koji su korišteni pri izradi internet stranice.

4.1. Class

U programskom svijetu ima dosta polemike oko JavaScripta i zašto ga mnogi ne vole koristiti jer se razlikuje od ostalih na način kako koja komponenta funkcionira. Prvi primjer su klase, koje nisu definirane kao u ostalim objektno-orijentiranim jezicima. Objekt nasljeđuje objekt korištenjem sintakse prototip (eng. Prototype). Korištenjem ECMAScripti uvela se riječ klasa (eng. Class) koja radi isto kao i prototype samo se omogućila jednostavnija i intuitivnija sintaksa za kreiranje objekata i proces postojećeg nasljeđivanja. Klase se kreiraju pomoću riječi class, i svaka klasa treba imati i metodu konstruktor (eng. Constructor) koja se poziva u trenutku kada se instancira novi objekt klase koristeći sintaksu novo ime klase (eng. new className) [8].

4.2. Const & Let

Konstanta (eng. Const) i engleska riječ let služe za deklariranje varijabli. Const služi za deklariranje konstanti što znači da se njima kasnije ne može dodijeliti nova vrijednosti. U slučaju da se želi staviti varijabla kojoj se kasnije može mijenjati vrijednost, tada se služimo s riječju let. Standardna riječ var ima slična svojstva, jedina je razlika kada se koristi riječ let, tada područje postojanja varijable ograničavamo samo na blok u kojem je deklarirana te se s time mogu izbjeći neželjene greške.

4.3. Arrow funkcija

ECMAScript6(ES6) uvodi jednostavniju sintaksu za pisanje anonimnih funkcija korištenjem strelica (eng. Arrow) funkcija koja implicitno vraća vrijednost ukoliko je u definiciji funkcije jedan izraz. Najbitnija karakteristika arrow funkcije je kraća sintaksa koja implicira bolju preglednost. Njihova prednost u React.js, osim sažetosti, je u načinu povezivanja this objekta unutar funkcije. Arrow funkcije nemaju this objekt i za to koriste ovo (eng. This) objekt od roditelja objekta. Korištenjem arrow funkcije, unutar funkcija koje ne dolaze iz React biblioteke, automatski će se vezati this objekt na komponentu unutar koje je funkcija definirana. Sintaksa kod definiranja arrow funkcija je da se prije znaka => postavljaju parametri funkcije, te iza njega tijelo funkcije [10].

Naveden je primjer funkcije “pomnozi” kako se prije koristila, točnije to je primjer u ES5 sintaksi, a uvođenjem arrow funkcije se izbacuje riječ function i vrati (eng. return) ukoliko funkcija ima samo jedan izraz za vratiti.

```
var pomnozi = function (x,y) {
    return x*y;
}

var pomnozi = (x,y) => {x*y};
```

Slika 1. Prikaz korištenja ECMAScript funkcija

5. Komponente

Aplikacija koja koristi React.js biblioteku je zasnovana na komponentama. Komponente enkapsuliraju jednu funkcionalnost jer su one samostalne i upotrebljive cjeline koda i sadrže oznake (eng. markup), logiku i dizajn, sve na jednom mjestu. One imaju svoje stanje (eng. state) i jasno sučelje preko kojega komuniciraju s drugim komponentama i ostalim dijelovima funkcije i aplikacije. Prihvaćaju ulazne parametre i kao izlaz vraćaju element koji sadrži što se sve treba prikazati na korisničkom sučelju. Kada dođe do trenutka da se ulazni parametar u komponenti promijeni, React ažurira (eng. render) samo tu komponentu te se osigurava da izgled na sučelju bude isti kao što je i bio, to jest, da za dani ulaz uvijek bude isti izlaz. Odgovoran način za izrađivanje komponenti je da je svaka od njih je zaslužna za jedan dio funkcionalnosti kako bi se smanjila mogućnost pogrešaka, te se omogućava ponovna upotreba komponenti na drugom mjestu.

U React.js, komponente se mogu definirati na dva načina:

- Komponenta kao funkcija

  Funkcija postaje React komponenta ukoliko ista vraća JSX format, format koji je sličan HTML-u.

```
function Hello() {
    return <h1>Hello World!</h1>
}
```

Slika 2. Prikaz korištenje funkcije

Unutar Reacta, komponentu “Hello” možemo koristiti kao tag, na sljedeći način:

```
<div>
    <Hello />
</div>
```

Slika 3. Prikaz korištenje funkcije Hello u komponenti

Potrebno je pripaziti da naziv komponente počinje velikim slovom. React tretira tagove koji počinju malim slovom kao standardne HTML tagove, a očekuje da prilagođene komponente počinju velikim slovom.

- Komponenta kao klasa

  Drugi način za definiranje komponenti je da koristimo klase pomoću riječi class. Ona mora nasljeđivati React.Component(ili samo Component) i mora imati render metodu koja vraća JSX.

```
class Hello extends Component{
    render(){
        return(
            <h1>Hello World!</h1>
        )
    }
}
```

Slika 4. Prikaz korištenja klasne komponente

Metoda render () je jedina nužna metoda za kreiranje React komponente. React koristi vraćenu vrijednost te metode kako bi zaključio što treba učiniti na sučelju aplikacije.

6. JSX i Babel.js

Pri izradi aplikacije koristeći React.js komponente možemo kreirati na dva načina. Koristeći standardne React.js objekte i metode ili koristeći JSX sintaksu. JSX je zasebna tehnologija koju nije nužno koristiti, no u suprotnom olakšava izradu aplikacije. Tko razumije JSX sintaksu? Iz tog razloga potreban nam je prevoditelj koji razumije JSX i pretvori je u format koji je razumljiv na svim preglednicima. Najpoznatiji alat koji se koristi za prevoditelja je Babel.

6.1. JSX

Kako bi postigli privlačan dizajn aplikacija, koristimo sintaksu JSX (eng. JavaScript Syntax Extension). JSX je JavaScript ekstenzija koju su napravili stručnjaci iz Facebook-a kako bi pisanje markup komponenti bilo omogućeno u HTML sintaksi. JSX nam omogućuje kreirati elemente u React-u koji se izražavaju pomoću render () metodu na DOM-u [13]. Pri radu s React-om, nužno je koristiti JavaScript, dok korištenje JSX se preporučuje koristiti kada sintaksa nije više efikasna niti čitljiva, također se preporučuje kao vizualna pomoć kada se radi na korisničkim sučeljima. Korištenje JSX-a uvelike pomaže pri kodiranju jer nam omogućuje:

- brzinu zbog korištenja optimizacije kod kompajliranja JavaScript koda
- jednostavnije uočavanje grešaka
- brže i jednostavnije pisanje predložaka uz prethodno znanje HTML-a koje je bitno za izradu privlačnog korisničkog sučelja

Naveden je primjer kako se može koristiti čisti JavaScript kod korištenjem HTML tagova i kako koristiti isti taj kod s primjenom JSX sintakse. Za korištenje JSX sintakse potrebno je koristiti React.createElement() funkciju koja prima tri argumenta: tip elementa, svojstva elementa i djecu elementa.

```
<div className="red"> <h1>Primjer</h1>  </div>

React.createElement('div', {className: "red"},
   React.createElement('h1', null, "Primjer"))
```

Slika 5. Korištenje JSX sintakse

React koristi className umjesto tradicionalne DOM riječi class. Budući da JSX je JavaScript, identifikatori kao što su klasa (eng. Class) i za (eng. For) “obeshrabreni” su kao XML atributna imena. Umjesto toga, React DOM komponente očekuju DOM svojstva kao što su ime klase (eng. className), odnosno za HTML (eng. htmlFor).

Preglednici poput Google Chrome, Mozille Firefox i sličnih ne znaju pročitati JSX sintaksu i njegova sintaksa se mora pretvoriti pomoću alata za pretvorbu – Babel.js.

6.2. Babel

Babel je besplatni i otvoreni (eng. open-source) JavaScript pretvarač koji pretvara JSX sintaksu u Vanilla ES5 koji svaki preglednik može razumjeti i ispisati [11]. Vrlo često se ovaj alat koristi kod pretvaranja u ECMAScript 2015. kod kako bi JavaScript kod bio čitljiv na svim web preglednicima. Koristi se iz razloga je većina web preglednika imaju probleme sa čitanjem određenim standardima ES6 te se upotrebljava u svrhu bolje funkcionalnosti. Takva pretvorba je poželjna i konstantno se koristi jer JavaScript kod se razvija brže nego što se razvijaju web preglednici. Zbog navedenih razloga, programerima je i dalje poželjno korištenje Babel alata jer u bilo kojem trenutku mogu iskoristiti sve prikladnosti koje su im omogućene.

7. Firebase baza podataka

Firebase Database je NoSQL baza podataka koja predstavlja platformu za razvoj mobilnih i web aplikacija te je razvijena od strane Firebase Inc 2011. godine, te ih je 2014. godine, otkupio Google. Usluge koje Firebase pruža su Firebase Realtime Database, Firebase Cloud Firestore, Firebase Storage i još mnogo drugih. Firebase je besplatan za korištenje u njihovom takozvanom „Spark“ planu koji omogućuje spremanje podataka u stvarnom vremenu na njihov posužitelj. Zahvaljući tome, aplikacije koje koriste Firebase bazu podataka vrlo brzo omogućavaju sinkronizaciju podataka jer se podaci preuzimaju sa servera paralelno tijekom korištenja aplikacije. Funkcionira na način da kada korisnik učitava određene podatke, oni se učitavaju direktno iz lokalne memorije uređaja te se s time troši manje vremena na učitavanje. Firebase omogućava čitanje podataka i kada korisnik izgubi mrežu s internet vezom, jer se podaci spremaju lokalno i su dostupni cijelo vrijeme. U trenutku ponovnog spajanja na internet, podaci koji su se u međuvremenu promijenili sinkroniziraju se sa poslužiteljem. Svi podaci se spremaju lokalno u SQLite bazi podataka koja se stvorila prilikom prve instalacije od strane Firebase-a [14].

Kako se radi o NoSQL bazi podataka, prisutne su drugačije mjere vezane uz optimizaciju i funkcionalnosti. Firebase-ov API(aplikacijsko programsko sučelje) dozvoljava samo operacije koje su lako izvedive kako bi se mogla izraditi aplikacija koju će koristiti veći broj ljudi.

7.1. Firebase: način spremanja podataka

Firebase koristi JSON(eng. JavaScript Object Notation) za spremanje podataka. JSON je otvoren tekstualni standard dizajniran za čitljivu razmjenu podataka. Firebase nudi dva rješenja spremanja podataka, oba temeljena na oblaku (eng. cloud), na klijentima i podržavaju sinkronizaciju podataka u stvarnom vremenu:

- Realtime Database – Firebase-ova orginalna baza podataka. Omogućuje učinkovito rješenje za manje mobilne aplikacije koje zahtijevaju sinkronizaciju podataka u stvarnom vremenu
- Cloud Firestore – novija baza podataka za razvoj mobilnih aplikacija. Novija verzija inačice Realtime Database donosi preglednost podataka. Također, ima bogatije i brže upite nad bazom, kao i bolje skaliranje.

Od navedenih Firebase-ovih baze podataka treba pomno izabrati koju koristiti. Stručnjaci koji su odgovorni za razvijanje Firebase alata preporučuju Cloud Firestore koja je i korištena pri izradi stranice „Kroz objektiv“.

Cloud Firestore nudi:

- bolje strukturirane upite i podatke: dok je Realtime Database jedno ogromno JSON stablo, Cloud Firestore je više struktiriran. Podaci se sastoje od dokumenata (eng. documents) i zbirki (eng. collections). Dokumenti mogu ukazivati na druge podzbirke koje sadrže druge dokumente i tako dalje.
- Ovako strukturiran način podataka pomaže kod jednostavnijeg upita jer su svi podaci pohranjeni na hijerarhijski način.
- Dizajniran za skaliranje: Cloud Firestore bolje skalira bazu podataka u stvarnom vremenu. Upiti se kreću prema veličini skupa rezultata, a ne po skupu podataka. Na taj način se osigurava brzo pretraživanje bez obzira kolika je veličina skupa podataka.
- Lakše dohvaćanje podataka: poput baze podataka u Realtime Database-u, mogu se postaviti „Listeners“ koji omogućuju prijenos podataka u stvarnom

vremenu. Ako se ne želi takvo ponašanje, Cloud Firestore ima ugrađen sustav za samo „dohvati moje podatke“.

- Podrška u više regija – što znači veću pouzdanost budući da se podaci dijele na više podatkovnih centara odjednom. Omogućena je i snažna dosljednost kako bi se u bilo kojem trenutku, kada postavimo upit, dobila najnovija verzija podataka.

7.2. React-redux-firestore

Prilikom stvaranja veće aplikacije, upravljanje podacima unutar takve aplikacije pomoću alata za pronalazak određenih podataka može biti kompliciranije za rukovanje jer se tada moraju nasljeđivati podaci iz jedne komponente u drugu ili se podaci moraju „mutirati“ kako bi se promijenila vrijednost podataka. Navedene poteškoće se mogu otkloniti uključivanjem Redux-a kako bi se olakšalo upravljanje podacima. Ova JavaScript-ova biblioteka služi kao središnja pohrana u aplikaciji u kojoj sve komponente imaju lakši pristup i osigurava se da čim se podatak promijeni šalje se određena akcija (eng. dispatch) [22]. Službena internet stranica Redux-a objašnjava da je to predvidivi spremnik stanja za JavaScript aplikacije [23].

React-redux-firestore je integracija Firestora koja se temelji na redux-firestore-u i omogućava interakciju autetifikacije, čuvanje i spremanje podataka u stvarnom vremenu dok se istovremeno odvija sinkronizacija podataka sa Firestore-om.

8. Izrada stranice „Kroz objektiv“

U ovom dijelu rada će biti detaljnije opisana sama izrada internet stranice. Opisat će se od kojih se sve elemenata sastoji navedena stranica, koji alati su se koristili pri izradi iste i na koji oni način su elementi povezani jedni s drugima. Rad na programskom kodu se započeo korištenjem create-react-app koji omogućuje brzo i jednostavno kreiranje React.js aplikacije u novo kreiranoj mapi. Za postizanje određenog dizajna je korišten framework Material-UI koji je React-ov framework za izradu korisničkog sučelja. Za svaku datoteku koja je korištena pri izradi određenih stranica se koristio prethodno navedeni Redux koji je povezan sa Firebase-om radi lakšeg dohvaćanja i spremanja podataka.

Programski kod se može pronaći na sljedećoj poveznici: https://github.com/mkantolic/kroz-objektiv/tree/master/redux-firebase/krozObjektiv

8.1. O internet stranici

Internet stranica je bazirana na čitanju članaka o fotografiji te ima tri glavna dijela, pregledavanje i čitanje članaka, pisanje članaka te kviz za odabir koja vrsta fotoaparata je za korisnika. Stranica se također sastoji i od stranice za prijavu, te samo prijavljeni korisnici mogu odrađivati prethodno navedene funkcionalnosti stranice.

Stranica za prijavu

- Stranica koja služi za prijavu korisnika, te bez navedene prijave, korisnici ne mogu unositi nove članke
- Stranica za prijavu, koja se odnosi za već prijavljene korisnike
- Stranica za registraciju, koja se odnosi za nove korisnike koji žele postati dio zajednice

Početna stranica

- Stranica na kojoj se nalaze svi članci te se svaki članak može pregledati
- Pri čitanju određenih članaka, korisnik koji je napisao svoj članak ga može i obrisati
- Članci se mogu pretraživati po određenim kategorijama: „Fotoaparati“, „Recenzije fotoaparata“, „Tips'n'tricks“ i „Oprema“

Stranica za kreiranje članaka

- Klikom na dugme „Novi članak“, može se napisati članak za bilo koju od prethodno navedenih kategorija.

Stranica za kviz

- Stranica se sastoji od 10 pitanja i sa 4 ponuđena odgovora, te se na svakom pitanju može izabrati samo jedan odgovor
- Na temelju izabranih odgovora se prikaže rezultat koji najviše odgovara prema korisničkom odabiru.
- Rezultat se odnosi na koju vrstu fotoaparata su najviše temeljeni odgovori i dodaje se sugestija koji fotoaparati bi bili odgovarajući za korisnika

8.2. Izrada stranice za prijavu

U ovom poglavlju će biti opisano kako se izradila stranica za prijavu i za registraciju. Stranica za registraciju je klasna komponenta koja sadrži određene state-ove: email, lozinka (eng. Password), ime (eng. firstName), prezime (eng. lastName). Korisnik pri prvoj registraciji ispunjava određena polja sa svojim korisničkim računom, lozinkom, imenom i prezimenom, te se ti podaci pohranjuju u state. Podaci u input poljima se pohranjuju setState(), klikom na dugme „Registraj se“, ako je korisnik sve uredno

popunio okida se metoda handleSubmit sa funkcijom signUp() kako bi se registriralo korisnički podaci . U slučaju da korisnik ne popuni dobro ime svog korisničkog računa ili ne napiše pravilnu lozinku, od strane Firebase dolazi odgovarajuća poruka.

Navedeni programski kod nam prikazuje funkciju signUp koja nam služi za registraciju novog korisnika. Funkcija se spaja s Firebase-om koji ima već definiranu instancu za određenu registraciju.

Instanci firebase.auth().createUserWithEmailAndPassword je potreban samo korisnikov e-mail i lozinka kako bi se spremilo u Firebase-ovu bazu podataka pod zbirku(collection) “users”, te se tako omogućilo korištenje stranice. Dodana je mogućnost uzimanja korisnikovog imena i prezimena kako bi se kreirali inicijali, a oni se prikazuju u određenom dugmetu pri vrhu stranice. “Dispatch” ili akcija označava Firebase-u kada je registracija uspješno prošla ili nije. U slučaju da registracija nije prošla kako treba, korisniku se javlja koja greška nastala, ili nije dobro ispunio sva potrebna polja, ili nije pravilno korisničko ime, ili nije dovoljno jaka lozinka ili je jednostavno već registriran.

```
export const signUp = newUser => {
  return (dispatch, getState, { getFirebase, getFirestore }) => {
    const firebase = getFirebase();
    const firestore = getFirestore();

    firebase
      .auth()
      .createUserWithEmailAndPassword(newUser.email, newUser.password)
      .then(resp => {
        return firestore
          .collection("users")
          .doc(resp.user.uid)
          .set({
            firstName: newUser.firstName,
            lastName: newUser.lastName,
            initials: newUser.firstName[0] + newUser.lastName[0]
          });
      })
      .then(() => {
        dispatch({ type: "SIGNUP_SUCCESS" });
      })
```

```
      .catch(err => {
        const errorCode = err.code;
        const errorMessage = err.message;
        if (errorCode === "auth/email-already-in-use") {
          console.log("it exists");
          dispatch({ type: "USER_EXIST", err });
        } else if (errorCode === "auth/weak-password") {
          console.log("The password is too weak");
          dispatch({ type: "SIGNUP_ERROR", err });
        } else {
          console.log(errorMessage);
        }
        console.log(err);
      });
  };
};
```

Programski kod 1. – Prikaz registracije novog korisnika

Prijava već prethodno registranog korisnika funkcionira na sličan način kao i registracija. U slučaju da je korisnik krivo upisao svoje podatke za prijavu ili nije već registriran u bazu podataka, Firebase šalje odgovarajuću poruku. Jedina razlika je što se ovdje pri formi handleSubmit() okida funkcija signIn() sa svojom Firebase instancom firebase.auth().signInWithEmailAndPassword(). „Dispatch“ ili akcija šalje određenu poruku Firebase-u kada je korisnik uspješno upisao svoje podatke te ga se preusmjerava na početnu stranicu aplikacije. U slučaju da korisnik nije ispunio potrebna polja kako odgovara, aplikacija javlja korisniku grešku ili nije pravilan email upisao ili je upisao krivu lozinku ili jednostavno nije registrirani korisnik.

```
export const signIn = credentials => {
  return (dispatch, getState, { getFirebase }) => {
    const firebase = getFirebase();

    firebase
      .auth()
      .signInWithEmailAndPassword(credentials.email, credentials.password)
      .then(() => {
        dispatch({ type: "LOGIN_SUCCESS" });
      })
```

```
      .catch(err => {
        const errorCode = err.code;
        const errorMessage = err.message;
        if (errorCode === "auth/user-not-found") {
          console.log("user not found");
          dispatch({ type: "LOGIN_NOT_EXIST", err });
        } else if (errorCode === "auth/wrong-password") {
          console.log("auth/wrong-password");
          dispatch({ type: "LOGIN_ERROR", err });
        } else if (errorCode === "auth/invalid-email") {
          console.log("auth/invalid-email");
          dispatch({ type: "LOGIN_INVALID", err });
        } else {
          console.log(errorMessage);
        }
        console.log(err);
      });
  };
};
```

Programski kod 2. – Prikaz prijave korisnika

U oba slučaja, kada je korisnik prijavljen, on ostaje prijavljen jedan određen dio vremena. Za ovaj slučaj, naveden programski kod, svaki put kada korisnik dođe na stranicu, provjeri je li je korisnik prijavljen ili ne.

```
if (auth.uid) return <Redirect to="/" />;
```

Programski kod 3. – Prikaz je li korisnik prijavljen

U slučaju da se korisnik želi odjaviti, na stranici će potražiti dugme „Izađi“ i klikom na to dugme će se izvršiti određena akcija. Akcija signOut() od firebase poprima instancu firebase.auth().signOut() koja šalje odgovarajuću poruku Firebase-u da je korisnik odjavljen.

```
export const signOut = () => {
```

```
  return (dispatch, {getFirebase}) => {
    const firebase = getFirebase();

    firebase.auth().signOut().then(() => {
      dispatch({ type: 'SIGNOUT_SUCCESS' })
    });
  }
}
```

Programski kod 4. – Prikaz odjave korisnika

Slika 6. Izgled stranice za prijavu.

8.3. Izrada početne stranice

Početna stranica se sastoji od dva dijela, od navigacijskog dijela i dijela gdje se prikazuju članci. Navigacijski dio se sastoji od glavnih stranica na koje korisnik može napraviti novi članak, preispitati svoje želje koji fotoaparat želi uzeti ili jednostavno se odjaviti. Također u navigacijskom dijelu može pretraživati članke po kategorijama.

Drugi dio se sastoji od pregledavanja i čitanja članaka. U navedenom programskom kodu je korišteno bez stanja (eng. Stateless) komponenta, te se umjesti class koristi riječ const. Takva komponenta se koristi u slučajevima kada želimo da funkcija samo prima props kao argument i vraća React element. Vidljivo je da komponenta ProjectList uzima props projects i koristilo se u mapiranju (eng. Map). Funcija map() kreira novi niz s rezultatima pozivanja navedene funkcije na svaki element u nizu koji je pozvan. Funkcija map() prolazi nizom elemenata nad kojim je pozvana, element koji je pozvan prima argument i tako funkcija prolazi nizom sve dok se ne završi s posljednjim elementom niza. Funkcija vraća novi niz koji sadrži vraćene vrijednosti. Svako dijete (eng. Child) u nizu mora imati jedinstveni ključ (eng. Key) ili id prop.

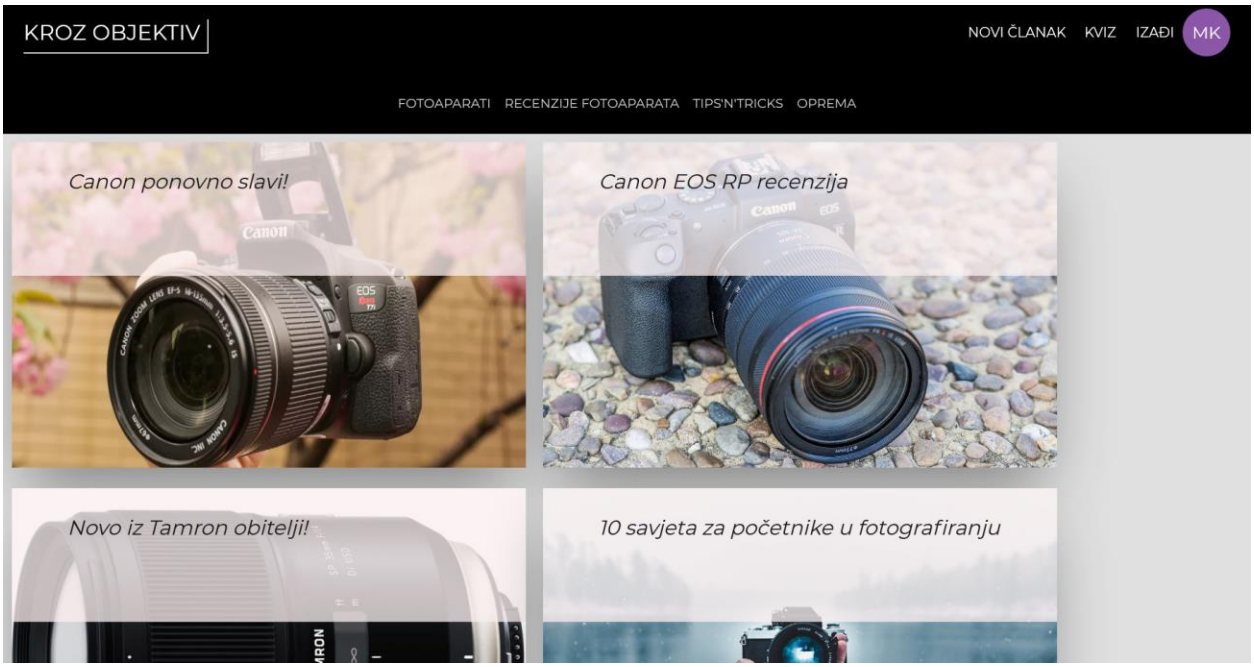
```
const ProjectList = ({ projects }) => {

  return (
    <div className="project-list section">
      {projects &&
        projects.map(project => {
          return (
            <Link to={"/project/" + project.id} key={project.id}>
              <ProjectSummary project={project} />
            </Link>
          );
        })}
      <Footer />
    </div>
  );
              };
```

Programski kod 5. – Prikaz korištenje map() funkcije

Komponenta ProjectList je pomoću props projekt (eng. „project“) povezana s još jednom stateless komponentom ProjectSummary koja navodi svaki članak posebno. U komponenti ProjectList je dodan još jedan React element „Link“ kako bi se moglo kliknuti na detaljnije čitanje članka. ProjectSummary prima props „projects“, te iz Firebase baze podataka dovaća određene podatke. Navedeno je da se za Material-UI element CardMedia, koja služi za karticu sa slikom i tekstom, dohvati slika koja je naslovljena s „project.naslovna“, naslov članka „project.title“, i ime autora koji su spremljeni prilikom prve registracije na stranicu. Linija dangerouslySetInnerHTML={{ __html: project.naslovna }} je korištena kako bi sliku koja je na Firebase spremljena kao običan HTML tag ponovno pretvorila u .jpg sliku.

```
const ProjectSummary = ({ project, classes }) => {
  return (
    <div className="tile">
      <CardMedia className={classes.card}>
        <div dangerouslySetInnerHTML={{ __html: project.naslovna }} />
        <div className="text"><h2>{project.title}</h2>
            <p className="animate-text"> <span className="sazetak">
            {project.sazetak}</span>
            </p> <p className="animate-text">
             <span>
              Autor:{project.authorFirstName}
             {project.authorLastName}{" "}
             </span>{" "}
             </p>
       </div>
       </CardMedia>
      </div>
      );
      };
```

Programski kod 6. – Prikazivanje podataka iz članka



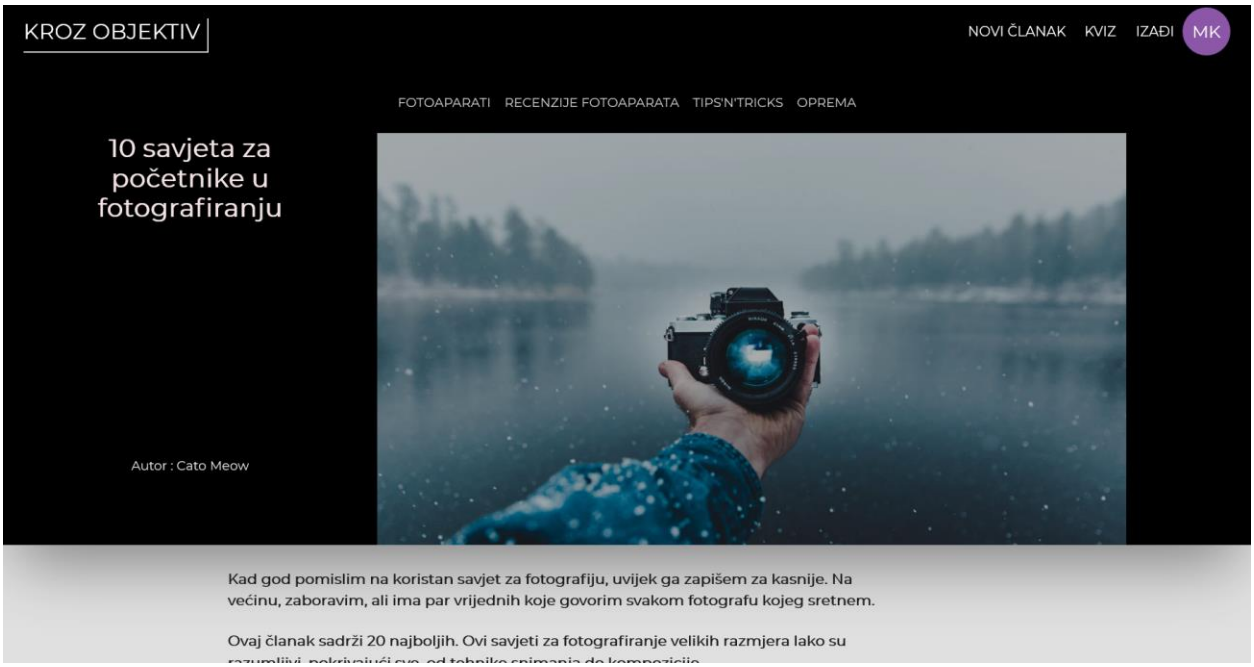
Slika 7. Pikaz početne stranice

Pri pregledavanju određenog članka korištena je ista logika kao i kod komponente ProjectSummary. Proslijeđen je props “project” s kojim smo dohvatili potrebne elemente, to jest podatke s Firebase-a. Svaki korisnik koji je napisao svoj članak, pri pregledavanju, klikom na dugme “Izbriši članak” se okida akcija handleClick() koji briše određeni članak. Komponenta mapStateToProps se koristi svaki put kada se stranica povezuje na Firebase, jer se svi podaci spremaju lokalno preko Redux-a. Službena stranica React-Redux navodi: “kao prvi argument koji se prenosi za povezivanje, mapStateToProps se koristi za odabir dijela podataka iz pohrane u kojem je potrebna povezana komeponenta [21]. Ona se zove svaki put kada se promijeni podatak u pohrani, te prima cjelokupno stanje pohrane i vraća object podataka kada je određena komponenta pozvana. U komponentu mapStateToProps je povezan članak s svojim id-om kako bi korisnik mogao obrisati svoj članak, povezani su putem autorizacijskog id-a koji se nalazi u svakom članku (eng. doc). Prilikom svakog spremanja članka, dodan je datum kada je članak kreiran. U Firebase-u, datum je spremljen u HTML format, i kako bi ga preoblikovali u hrvatski standard prikazivanja vremena, korišten je Moment.js s instancom:

```
{moment(project.createdAt.toDate()).format(" LLL ")}
```

```
  handleClick = () => {
    const { firestore, id } = this.props;
    firestore.delete({ collection: "projects", doc: id });
    this.props.history.push("/");
  };

const mapStateToProps = (state, ownProps) => {
  const id = ownProps.match.params.id;
  const projects = state.firestore.data.projects;
  const project = projects ? projects[id] : null;
  return {
    project: project,
    auth: state.firebase.auth,
    id: ownProps.match.params.id,
    doc: ownProps.match.params.id
  };
};
```

Programski kod 7. – Prikaz korištenja funkcije za brisanje članka



Slika 8. Prikaz stranice za čitanje pojedinog članka.

8.4. Izrada novog članka

Klikom na gumb „Novi članak“ otvara se nova stranica koja omogućuje se ispunjavanje forme gdje korisnik može stavljati željene slike i dijeliti svoja iskustva iz svijeta fotografije. Za kreiranje članaka u kojem korisnik može koristiti različite oblike ukrašavanje teksta, korištena je React-ova komponenta react-quill koji omogućuje i učitavanje slika sa računala ili kopiranje slika sa određenih stranica. Za korištenje react-quill potrebno je bilo dodati određene module koji omogućavaju korištenje svih funkcionalnosti quill-a. Korištena je klasna komponenta CreateProject koja ima određene state-ove:naslov (eng. Title), sadržaj (eng. content), naslovna, kategorija (eng.category), sazetak. Ispunjavanjem svakog potrebnog inputa odvija se određena akcija. Kod ispunjavanja naslova (eng. title) i sazetak se odvijala akcija handleChange() koja je ispunjena polja pohranjivala u setState(). Polja naslovna i content se odvojila jer oni su vezani za react-quill i moraju se spremati posebno kako bi se u bazu ispravno spremili. Zadnje polje je category gdje se nudi mogućnost odabira u koju kategoriju napisani članak pripada. Odabravši pravilnu kategoriju, poziva se akcija handleChangeOption() koja u setState() u polje category sprema izabranu kategoriju. Klikom na „Kreiraj“ poziva se akcija handleSubmit() koja poziva funkciju createProject().

U navedenom programskom kodu opisana je funkcija createProject() koja nasljeđuje prop project iz komponente za kreiranje novog članka. Funkcija se spaja na Firebase-ovu bazu podataka Cloud Firestore i sa instancom firestore.collection().add() se sprema novokreirani članak. Instanca getState().firebase.profile kreira polja za spremanje podataka koji su proslijeđeni putem prop project. Instanca getState().firebase.auth.uid dohvaća podatke o ulogiranom korisniku. Zbirka (eng. collection) sprema na Firebase pod nazivom „projects“, te osim teksta koji su napisani za članak, spremaju se i podaci o korisniku, autorovo ime i prezime pomoću profile.firstName i profile.lastName , njegov ID, te i vrijeme kada je članak napisan. Izvršava se akcija spremanja i Firebase-u se šalje odgovarajuća poruka ako je članak uspješno spremljen, ili ako je došlo do pogreške pri spremanju izbaci se odgovarajuća poruka greške.

```
  export const createProject = project => {
  return (dispatch, getState, { getFirestore }) => {
    const firestore = getFirestore();
    const profile = getState().firebase.profile;
    const authorId = getState().firebase.auth.uid;
    firestore
      .collection("projects")
      .add({
        ...project,
        authorFirstName: profile.firstName,
        authorLastName: profile.lastName,
        authorId: authorId,
        createdAt: new Date()
      })
      .then(() => {
        dispatch({ type: "CREATE_PROJECT_SUCCESS" });
      })
      .catch(err => {
        dispatch({ type: "CREATE_PROJECT_ERROR" }, err);
      });
  };
};
```

Programski kod 8. – Prikaz kreiranja novog članka u Firebase-u

8.5. Kreiranje komponente za kviz

U React.js postoji popularan obrazac koji dijeli komponente u dvije kategorije: prezentacijske i kontejnerske(container). Kontejnerske komponente su zadužene za kako stvari funkcioniraju, a prezentacijske se brinu kako određene komponente izgledaju.

Kviz se sastoji od 4 dijela: Question, Question Count, Answer Option, Result. Komponenta Question je stateless komponenta jer ne koristimo state za navedenu komponentu. Stručnjaci preporučuju korištenje stateless komponenti zbog elimininiranja dupliciranje koda (eng. “boilerplate code”), to je kod koji se konstantno ponavlja kako bi se dobio određeni rezultat ne razmišljajući da postoji drugi način rješenja tog koda. Question komponenta je jednostavna komponenta koja samo prikazuje koje se pitanje prikazuje na sučelju. Sadržaj pitanja se prosljeđuje putem props.

Komponenta QuestionCount sadržava na kojem broju pitanja se korisnik trenutačno nalazi i koliko mu je još ostalo za odgovoriti. Taj postupak je omogućen brojačem (eng. counter) od ukupnog zbroja pitanja (eng. total).

Sljedeća komponenta prikazuje opcije odgovora. Komponenta AnswerOption sadrži novi concept usporedbe: checked. Vrijednost izraza je Boolean (true ili false) na temelju je li je odgovor koji je odabran jednak ponuđenim odgovorima. Korištena je i funkcija renderAnswerOption koja je zadužena za prikaz odgovora za pojedino pitanje.

```
<input
        type="radio"
        className="radioCustomButton"
        name="radioGroup"
        checked={props.answerType === props.answer}
        id={props.answerType}
        value={props.answerType}
        disabled={props.answer}
        onChange={props.onAnswerSelected}
      />
      <label htmlFor={props.answerType}>
        <p className="radioCustomLabel"> {props.answerContent} </p>
      </label>
```

Programski kod 9. – Prikaz funkcije AnswerOption()

Komponenta Quiz služi kako bi se sve komponente spojile. Kviz gradimo na temelju prethodno stvorenih komponenti i proslijeđujemo im potrebne props-e. Quiz.js je prezentacijska komponenta jer želimo da sav kod koji je bitan za prikaz, odvojimo od koda koji su bitni za fukcionalnosti kviza.

```
class QuizHome extends Component {
  constructor(props) {
    super(props);

    this.state = {
      counter: 0,
      questionId: 1,
      question: "",
      answerOptions: [],
      answer: "",
      answersCount: {
        Profesionalac: 0,
        Amater: 0,
        Početnik: 0,
        Kompaktni: 0
      },
      result: ""
    };
this.handleAnswerSelected = this.handleAnswerSelected.bind(this);
}
```

Programski kod 10. – Prikaz klasne komponente QuizHome

State bi trebao sadržavati sve podatke koji se koriste prilikom kreiranja određenih događaja u komponenti. U navedenom kodu su navedena sva stanja(state) koji su potrebni za kviz. Potrebno je preuzeti određene podatke kako bi ih pohranili u state. Podaci koji su potrebni su pitanja, odgovori i tip odgovora(answerCount) koje preuzimamo iz druge datoteke quizQuestion.jsx. Akciju this.handleAnswerSelected je potrebno čvrsto povezati (eng. “hard bind”) prije render funkcije. Zbog optimizacije, bind je najbolje stavljati u constructor. Metoda bind() stvara novu funkciju koja, kada se pozove, ima ključnu riječ this postavljenu na navedenu vrijednost, s danim slijedom argumenata koji prethodi bilo kojoj opciji kada se poziva nova funkcija [17].

```
componentWillMount() {
    const shuffledAnswerOptions = quizQuestions.map(question =>
      this.shuffleArray(question.answers)
    );
    this.setState({
      question: quizQuestions[0].question,
      answerOptions: shuffledAnswerOptions[0]
    });
              }
```

Programski kod 11. – Prikaz componentWillMount()

Pomoću React ciklusa componentWillMount() pohranjujemo podatke u state. Kad god React renderira komponentu prvo će pozvati componentWillMount(). Metoda se poziva samo jednom u "životu" komponente, i to se događa prije nego se prvi put inicijalizira [16]. Zbog toga, nema pristupa DOM-u. Budući da se metoda poziva prije metode render(), to je jedina metoda koja se poziva na strani poslužitelja (kada se koristi rendering na strani poslužitelja). Funkcija ShuffleArray() služi za nasumičan redoslijed odgovora.

```
handleAnswerSelected(event) {
    this.setUserAnswer(event.currentTarget.value);

    if (this.state.questionId < quizQuestions.length) {
      setTimeout(() => this.setNextQuestion(), 300);
    } else {
      setTimeout(() => this.setResults(this.getResults()), 300);
    }
  }
```

Programski kod 12. – Prikaz akcije handleAnswerSelected()

Navedena funkcija obavlja dva zadatka: postavlja odgovore i sljedeća pitanja. Svaki zadatak je izdvojen u svoju funkciju kako bi kod bio čitljiviji.

```
  setUserAnswer(answer) {
    this.setState((state, props) => ({
      answersCount: {
        ...state.answersCount,
        [answer]: state.answersCount[answer] + 1
      },      answer: answer    }));   }
```

Programski kod 13. – Prikaz funkcije setUserAnswer()

Funkcija setUserAnswer() prima prop “answer” koji nam služi da pohranjujemo korisnikov odgovor. Što znači da odgovor koji je odabran, se pohranjuje na temelju njegove vrijednosti, to jest tipa. U ovom slučaju odgovor se pohranjuje u vrijednosti između “Profesionalac”, “Amater”, “Početnik” ili “Kompaktni”.

```
setNextQuestion() {
  const counter = this.state.counter + 1;
  const questionId = this.state.questionId + 1;

  this.setState({
    counter: counter,
    questionId: questionId,
    question: quizQuestions[counter].question,
    answerOptions: quizQuestions[counter].answers,
    answer: ""
  });
}
```

Programski kod 14. – Prikaz funkcije setNextQuestion()

Funkcija setNextQuestion() nam omogućava da povećavamo brojač(counter) i jedinstvenu oznaku pitanja(questionId), kojima prvo kreiramo odgovarajuće varijable za povećavanje te ih spremamo u setState(). Ažuriranjem stanja pitanja i ponuđenih odgovora omogućava se process prelaska na sljedeće pitanje sa odgovarajućim odgovorima.

```
getResults() {

  const answersCount = this.state.answersCount;
  const answersCountKeys = Object.keys(answersCount);
  const answersCountValues = answersCountKeys.map(key => answersCount[key]);
  const maxAnswerCount = Math.max.apply(null, answersCountValues);

  return answersCountKeys.filter(key => answersCount[key] === maxAnswerCount);
}
```

Programski kod 15. – Prikaz funkcije getResult()

Funkcija getResult() izračunava kojeg tipa odgovora ima najviše i prema tome se formira rezultat kviza. Korištena je funkcija Object.keys() koja služi za vraćanje polje stringova koji predstavljaju sva svojstva objekta. Korištena je map() funkcija koja vraća niz vrijednosti, zatim se izračunava koji tip rezultata ima vrijednost jednaku maxAnswerCount pomoću filter metode te se vraća odgovarajući rezultat.

```
setResults(result) {
  if (result.length === 1) {
    console.log(this.state);
    this.setState({ result: result[0] });
  } else {
    this.setState({ result: "Tebi odgovara bridge fotoaparat!" });
  }
}
```

Programski kod 16. – Prikaz funkcije setResults()

Funkcija setResult() prima rezultat od getResult() funkcije i pohranjuje ju u polje te pri tome provjerava ima li to polje vrijednost. Vrijednost dodijeljujemo pomoću setState(). Ako vrijednost u tom polju pripada odgovarajućem tipu odgovora, vratit će se određeni rezultat. U slučaju da zbroj ukupnih odgovora ne odgovara određenom tipu, korisniku se izbacuje rezultat “Tebi odgovara bridge fotoaparat!” jer ima više tipova koji su se podudarali jedni s drugima.

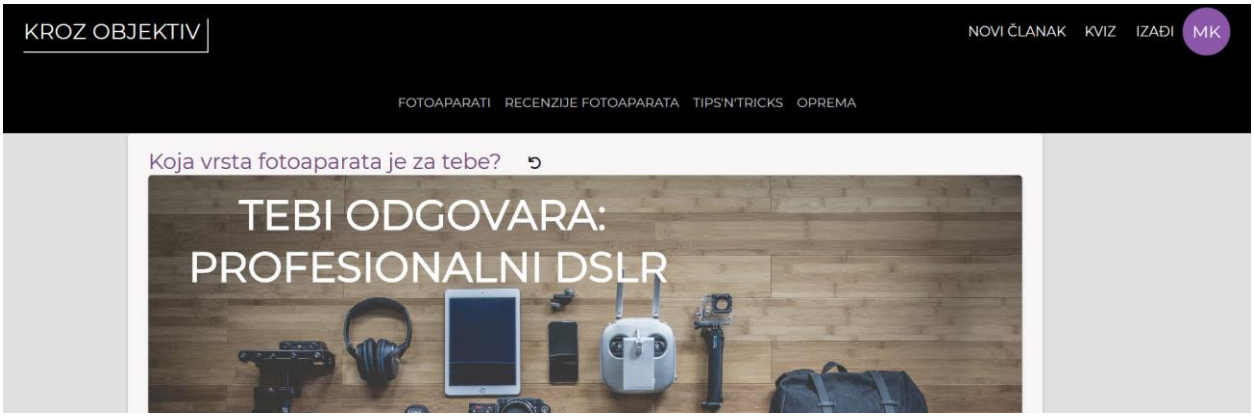
```
renderResult() {
  if (this.state.result === "Profesionalac") {
    console.log("true profesionalac");
    return (
      <div>
        {" "}
        <Profesionalac />
      </div>
    );
  } else if (this.state.result === "Amater") {
    console.log("true amater");
    return (
      <div>
        <Amater />
      </div>     );
```

```
    } else if (this.state.result === "Početnik") {
      console.log("true početnik");
      return (
        <div>
          <Pocetnik />
        </div>       );
    } else if (this.state.result === "Kompaktni") {
      console.log("true kompaktni");
      return (
        <div>
          <Kompaktni />
        </div>       );
    } else {
      console.log("bridge fotoaparat");
    }
    return <Result quizResult={this.state.result} />;
  }
  reloadPage() {
    window.location.reload();
  }
```

Programski kod 17. – Prikaz renderResult()

Korisnikov rezultat u metodi renderResult() prolazi kroz običnu for petlju provjeravajući koji tip rezultata odgovara određenom uvjetu. Kada se ispuni točan uvjet, korisniku se prikaže rezultat s danim sugestijama oko izbora fotoaparata.

Metoda reloadPage() je povezana na dugme koje služi za osvježavanje stranice, jer ako korisnik nije zadovoljan sa svojim rezultatom ili samo želi znati koji bi rezultat dobio sa drugačijim odgovorima, window.location.reload() ga vraća na prvo pitanje.



Slika 9. Prikaz rezultata iz riješenog kviza

9. Zaključak

U ovom radu je opisano što je korisničko sučelje i koliko je bitno izraditi dobar i funkcionalan dizajn za korisničko sučelje. Upravo odličan dizajn nekog proizvoda u kojem je prisutno korisničko sučelje može biti bitan za popularnost u današnjem suvremenom svijetu.

React.js biblioteka koja je kreirana 2013. godine, nije izgubila prednost pri odabiru kreiranja aplikacija, internet stranica ili mobilnih aplikacija (React Native). Danas je jedna od najćešće korištenih biblioteka za izradu raznih korisničkih sučelja. Popularnije aplikacije kao što su BBC, Facebook, Imgur, Instagram, Netflix, PayPal, Reddit, Uber, WhatsApp su sve napravljene pomoću React.js biblioteke. React.js u sebi ima ugrađeno više od 2000 komponenti koje se mogu iskoristiti za raznovrsne funkcije. U slučaju da se želi iskoristiti druga funkcija vrlo lako se implementira. Instagram koristi React.js zbog responzivnosti, Reddit zbog brzog učitavanja navigacijskih poveznica i općenito, boljeg korisničkog iskustva. React.js je postigao svoju popularnost na temelju svoje jednostavnosti i fleksibilnosti.

Zbog navedenih razloga, kreirana je stranica “Kroz objektiv” koja prikazuje mali dio što se sve može izraditi pomoću React.js biblioteke. Dizajn stranice je kreiran pomoću Material-UI, posebno dizajnirane komponenate za React.js. Upravo s tim komponentama se olakšao proces dizajniranja i kreiranja korisničkog sučelja, te se dobila tražena responzivnost stranice kao i brzo učitavanje komponenata. Sigurnost da će se dohvatiti traženi članci u bilo kojem trenutku za registriranog korisnika je omogućeno preko Google-ove baze podataka Firebase. Redux je omogućio lokalno pohranjivanje podataka kako bi dohvaćanje određenih članaka bilo brže i jednostavnije. Navigiranje na stranici je omogućeno preko BrowserRouter koji je sadržan u react-router-dom. Korištenjem React.js biblioteke, uz pomoć nekoliko dodatnih paketa zaključilo se da je React.js impresivna biblioteka koja omogućuje efektivnu izgradnju aplikacija.

Kao i kod korištenja ostalih biblioteka i programskih jezika, najveći nedostatak je što za savladati React.js treba vremena. React.js je biblioteka koja se često koristi pri izradi raznih aplikacija te postoji više načina rješavanja jednog problema. Također, mogu

se koristiti dodatni paketi koje treba proučavati i isprobavati koji način je najpristupačniji određenom problem. Prednost internet stranice je što baza podataka nije ovisna o lokalnom server i koristeći Firebase bazu podataka omogućen je pristup podacima preko interneta.

Korištenje internet stranice je zamišljeno kako bi se okupila zajednica zaljubljenika u fotografiju, prvenstveno na hrvatskom području kako bi ljudi s našeg područja dijelili svoja iskustva i znanja. Trenutno ograničenje aplikacije je što nedostaje mogućnost za uređivanje profila. Korisnici se mogu identificirati pri registraciji sa svojom e-mail adresom, no bilo bi poželjno da korisnici mogu dodati svoje osobne podatke i vlastitu sliku, gdje bi na svom profilu mogli pronaći sve članke i komentare koje su oni kreirali. Za daljnji razvoj aplikacije bi se razvila funkcionalnost komentiranja i ocjenjivanja članaka kako bi se upotpunilo korisničko sučelje, te s time i samo zadovoljstvo stranice.

Aplikacija je postavljena na Netlify hosting, te je dostupna na adresi: https://kroz-objektiv.netlify.com . Registracija i prijava na aplikaciju se može registrirati sa željenom e-mail adresom i lozinkom i aplikacija sadrži testne korisnike i članke.

10. Literatura

[1.] Korisničko sučelje, Vidipedija, dostupno na: http://www.vidipedija.com/index.php?title=Korisni%C4%8Dko_su%C4%8Delje (24.05.2019.)

[2.] "Geštaltizam", dostupno na: http://www.enciklopedija.hr/natuknica.aspx?id=21852 (24.05.2019.)

[3.] "Evo kako da primenite gestalt princip u web dizajnu", Telegraf, dostupno na https://www.telegraf.rs/hi-tech/2587812-evo-kako-da-primenite-gestalt-princip-u-web-dizajnu (24.05.2019.)

[4.] "A Good User Interface Is One That's Backed By Reproductive Evidence(A/B Tests), dostupno na https://goodui.org/ (25.05.2019.)

[5.] "Software User Interface Design" dostupno na: http://www.tutorialspoint.com/software_engineering/software_user_interface_design.htm (25.05.2019.)

[6.] "Uvod u React", dostupno na: https://skolakoda.org/uvod-u-react/ (26.05.2019.)

[7.] Robin Wieruch, The Road to learn React, Independently published (September 14, 2018).

[8.] "React.js", dostupno na : https://reactjs.org/ (04.02.2019.)

[9.] "React: The Virtual DOM", dostupno na: https://www.codecademy.com/articles/react-virtual-dom (26.05.2019.)

[10.] "ES6 Arrow functions & this", dotupno na: https://codepen.io/somethingkindawierd/post/es6-arrow-functions-this (26.05.2019.)

[11.] "What is Babel", dostupno na: https://babeljs.io/docs/en (27.05.2019.)

[12.] "Uvod u react ekosistem", dostupno na: https://medium.com/@ITesic/uvod-u-react-ekosistem-8ccfad0a1030 (27.05.2019.)

[13.] "Tutorial: JSX", dostupno na: http://buildwithreact.com/tutorial/jsx (27.05.2019.)

[14.] "Firebase", dostupno na: https://firebase.google.com/ (13.03.2019.)

[15.] “Understanding state and props in React” , dostupno na https://hackernoon.com/understanding-state-and-props-in-react-94bc09232b9c (28.05.2019.)

[16.] “How to use React Lifecycle Methods”, dostupno na: https://codeburst.io/how-to-use-react-lifecycle-methods-ddc79699b34e (28.05.2019.)

[17.] “Function.prototype.bind()”, dostupno na https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_objects/Function/bind (29.05.2019.)

[18.] “Material-UI”, dostupno na: https://material-ui.com/ (07.02.2019.)

[19.] “ npm ”, dostupno na: https://www.npmjs.com/ (04.02.2019.)

[20.] “create-react-app”, dostupno na: https://github.com/facebook/create-react-app (04.02.2019.)

[21.] “Connect: Extracting Data with mapStateToProps”, dotupno na: https://react-redux.js.org/using-react-redux/connect-mapstate (29.05.2019.)

[22.] “Understanding Redux + Reac in Easiest Way Part 1” dostupno na: https://medium.com/@tkssharma/understanding-redux-react-in-easiest-way-part-1-81f3209fc0e5 (30.05.2019.)

[23.] “Getting started with Redux”, dostupno na: https://redux.js.org/introduction/getting-started (30.05.2019.)

11. Popis slika

12. Popis programskih kodova

13. Sažetak

React.js je JavaScript biblioteka koja omogućuje programiranje i razvoj aplikacija te je važan dio za front-end development. Cilj ovog rada je pokazati kako izraditi internet stranicu pomoću React.js. Stranica „Kroz objektiv“ koristi paket za dizajn komponenti Material – UI, React-Redux za lokalno pohranjivanje i bazu podataka Firebase. Prvi dio ovog rada se sastoji od teorijskog dijela u kojem je objašnjeno što je korisničko sučelje, ukratko je objašnjeno kako napraviti dobar dizajn korisničkog sučelja, koje su sve bitne metode i dodatni paketi za pravilno funkcioniranje React.js. Drugi dio ovog rada se sastoji od praktičnog dijela u kojem je opisano kako komponente međusobno komuniciraju i kako se implementiraju. Internet stranica služi kao primjer razvojnih mogućnosti biblioteke React.js.

Ključne riječi: React.js, Material-UI, Baza podataka, Korisničko sučelje, Izrada korisničkog sučelja, front-end development

14. Abstract

React.js is a JavaScript library that allows programming and application development it is an important part of front-end development. The aim of this paper is to show how to create a website using React.js. The "Kroz objektiv" page uses a design package Material-UI, React-Redux for local storage and Firebase database. The first part of this paper consists of a theoretical part explaining what the user interface is explained, explaining in short how to make a good user interface design, all important methods and additional packages for proper functioning of React.js. The second part of the work consists of a practical part describing how the components interact and how they are implemented. The website serves as an example of React.js library development capabilities.

Keywords: React.js, Material-UI, Database, User Interface, Creating User Interface, Front-End Development