Primjena React.JS i web tehnologija u izradi mobilne aplikacije

Prožek, Daria

SVEUČILIŠTE U ZAGREBU
GRAFIČKI FAKULTET

DARIA PROŽEK

PRIMJENA REACT.JS I WEB TEHNOLOGIJA U IZRADI MOBILNE APLIKACIJE

DIPLOMSKI RAD

Zagreb, 2018.

Sveučilište u Zagrebu
Grafički fakultet

DARIA PROŽEK

PRIMJENA REACT.JS I WEB TEHNOLOGIJA U IZRADI MOBILNE APLIKACIJE

DIPLOMSKI RAD

Mentor:
doc. dr. sc. Tajana Koren Ivančević

Student:
Daria Prožek

Zagreb, 2018.

SAŽETAK

U radu se predstavljaju osnove sintakse ReactJS-a, njegovo korištenje te alati i praksa potrebni za izradu mobilne aplikacije i dizajn korisničkog sučelja.

ReactJS je JavaScript biblioteka koja se koristi za izgradnju interaktivnih korisničkih sučelja web i mobilnih aplikacija pomoću ponovno upotrebljivih cjelina. Zbog prednosti korištenja ReactJS tehnologije, ona danas zauzima mjesto jedne od najpopularnijih JavaScript biblioteka. Dijeljenje aplikacije na manje cjeline, koje se nazivaju komponente, omogućava brz i jednostavan odaziv aplikacije. Pri svakoj promjeni podataka ReactJS omogućava automatsko renderiranje. Na taj način se korisniku u svakom trenutku pruža uvid u stvarno stanje aplikacije.

Krajnji produkt rada je primjena teorijskog znanja i web tehnologija u izradi funkcionalne mobilne aplikacije koja prezentira kandidata poslodavcu, odnosno ima ulogu životopisa i mape radova. Ovim pristupom se kandidat predstavlja na jedinstven način, a kao i autor aplikacije, ujedno predstavlja i svoje vještine u izradi iste. Svrha aplikacije je, osim na zanimljiv i interaktivan način predstaviti kandidata, istražiti novitete u području web tehnologija i njihovu primjenu u višeplatformnom razvoju.

KLJUČNE RIJEČI: ReactJS, JS, web tehnologije, mobilna aplikacija

ABSTRACT

This master's thesis covers the fundamentals of React.JS sintax it's purpose as well as the needed tools and skills for mobile application development and building user interfaces.

React.JS is JavaScript library used for building interactive user interfaces of web applications with reusable components. React.JS has many advantages, therefore today it is one of the most popular JavaScript libraries. Applications build with reusable components offer fast response to a user. React.JS automatically renders it's content on every change that user makes inside of the user interface. This way, the user has the insight of the application in a real time.

Final product of this paper is building a functional mobile application using acquired knowledge and needed web technologies. The application has a role of Curriculum Vitae. It presents a candidate to employer, showing candidate's professional and social skills as well as the candidate's work in a portfolio. In such a way, candidate is being presented in a unique way, and also as the author of the application, candidate shows skills in application development. Except mentioned, the purpose of this thesis is seeing into new web technologies and their use in a multiplatform development.

KEYWORDS: ReactJS, JS, web technologies, mobile application

SADRŽAJ

1. UVOD

Razvojem tehnologije i pojavom pametnih telefona javljaju se i mobilne aplikacije koje su danas najzastupljenija vrsta softvera. Mobilne aplikacije su primjenjiva programska podrška, odnosno softveri, razvijeni za prijenosne uređaje kao što su pametni telefoni i tableti. Osmišljene su uzimajući u obzir sve zahtjeve i ograničenja uređaja te njihove specijalizirane značajke koje je moguće iskoristiti u svrhu obavljanja funkcije aplikacije. Aplikacije mogu biti informativnog, zabavnog ili praktičnog karaktera, a njihova glavna zadaća je izvršiti svoju funkciju na način da zadovolji korisnika.

Danas je jedna od glavnih karakteristika mobilnih aplikacija njihova interaktivnost. Uključivanjem korisnika u obavljanje njihovih funkcija postiže se bolje korisničko iskustvo. Korisničko iskustvo je dojam koji se ostavlja na korisnika izazivajući emocije pri korištenju proizvoda, odnosno mobilne aplikacije. U 2018. godini se na tržištu nalazi oko četiri milijuna mobilnih aplikacija. Zbog tolike konkurencije, korisničko iskustvo je vrlo važna stavka za uspjeh na tržištu. Također, kako bi korisničko iskustvo bilo što bolje, posebna se pažnja pridodaje dizajnu korisničkog sučelja, odnosno samom izgledu aplikacije.

Kod samog razvoja aplikacije potrebno je odabrati odgovarajuće alate i tehnologije, odnosno programske ili stilske jezike, njihova razvojna okruženja (eng. *framework*) te biblioteke. Pri takvom odabiru, važno je poznavati mogućnosti i primjenu jezika te odrediti svojstva aplikacije u razvoju. JavaScript danas zauzima mjesto najzastupljenijeg programskog jezika. Njegovim stalnim unaprjeđenjem razvijaju se i JavaScript biblioteke s ciljem olakšanog korištenja tog jezika u određene svrhe. React.JS je vrlo zastupljena JavaScript biblioteka koja se koristi za izgradnju interaktivnih korisničkih sučelja. Kako bi izrada aplikacije bila moguća, osim programskog jezika, potrebno je poznavati i druge web tehnologije.

2. TEORIJSKI DIO

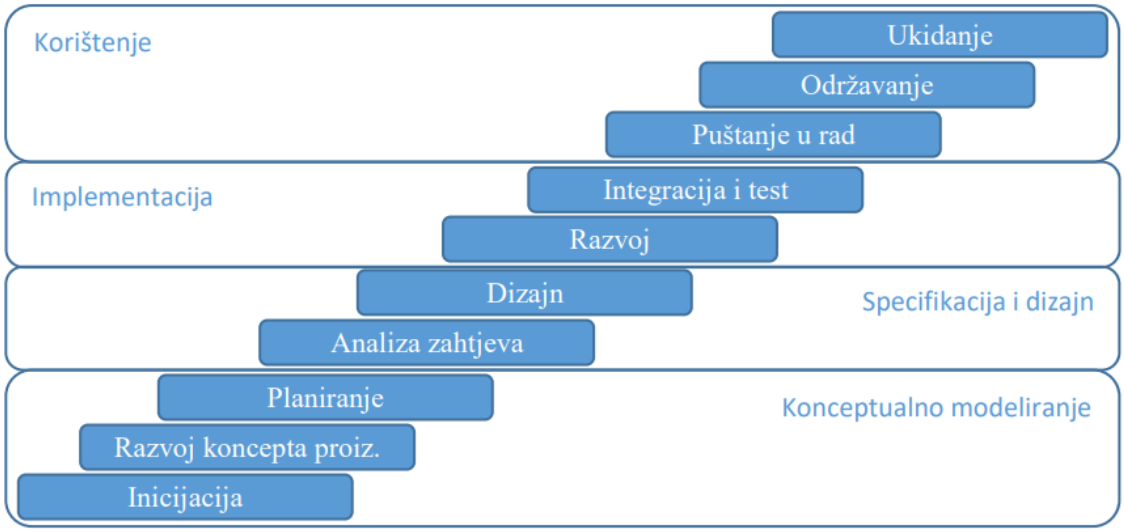
2.1 Mobilne aplikacije

Mobilne aplikacije su programske podrške za prijenosne uređaje kao što su pametni telefoni ili tableti. [1] Prve ideje o mobilnim aplikacijama iznio je Steve Jobs 1983. gdje predstavlja zamisao o kupovini *softvera* putem telefonskih linija. Prvi pametni telefon, IBM Simon, 1992. sadržavao je nekoliko jednostavnih aplikacija kao što su bilješke i aplikacija za pregledavanje elektroničke pošte. Veliki zaokret u razvoju mobilnih aplikacija donosi Apple 2007. godine kada je izdao svoj prvi *smartphone*, odnosno pametni telefon iPhone. 2008. se ostvaruje zamisao Stevea Jobsa te se izdaje App Store, odnosno platforma za trgovinu aplikacija. U početku je App Store sadržavao 552 aplikacije od kojih je 135 bilo potpuno besplatno. O vrlo uspješnom razvoju mobilnih aplikacija koje su danas neizostavan alat u svakodnevnom životu govori trenutni broj aplikacija na App Store-u koji iznosi dva milijuna. Iste je godine izašla i Google-ova platforma za prodaju aplikacija pod nazivom Android Market koji je 2012. preimenovan u Google Play Store. Isti danas sadrži gotovo četiri milijuna aplikacija. 2010. Microsoft izdaje Windows Phone Store. Danas su mobilne aplikacije jedna od vodećih usluga za korištenje interneta. Ukupan broj preuzimanja mobilnih aplikacija u 2017. iznosio je 197 milijardi. Najpopularnija aplikacija je Facebook koju koristi 81% ukupnog broja korisnika pametnih telefona. [2]

Zadaća mobilne aplikacije je izvršavanje njenih funkcija s ciljem zadovoljavanja korisnika. Funkcije mobilnih aplikacija ovise o tome za što su iste namijenjene. Mobilne aplikacije mogu biti informativnog, praktičnog ili zabavnog karaktera. Osmišljene su uzimajući u obzir sve zahtjeve i ograničenja uređaja te njihove specijalizirane mogućnosti koje je moguće iskoristiti u svrhu obavljanja funkcije aplikacije. Ono što razvoj mobilnih aplikacija čini drugačijim od aplikacija koje se koriste na računalu je raznolikost veličine zaslona mobilnih uređaja, potrošnja baterije, ograničenje memorije i mogućnosti procesora te zaslon osjetljiv na dodir. Većina mobilnih uređaja dolazi s unaprijed instaliranim mobilnim aplikacijama kako bi se korisniku od samog početka pružila mogućnost praktične uporabe uređaja. [3]

2.1.1 Razvoj mobilnih aplikacija

Razvoj mobilnih aplikacija čini složen skup aktivnosti koje se provode od strane članova projektnog tima u svrhu osmišljavanja, izrade, puštanja u rad te održavanja mobilnog *softvera*. Proces razvoja mobilnih aplikacija započinje razradom projektne ideje i definiranjem funkcionalnosti same aplikacije. Nakon idejnog razvoja i razrade slijedi dizajn korisničkog iskustva te korisničkog sučelja, odnosno njenog izgleda i načina korištenja. Programiranje je sljedeći i ključan korak u razvoju aplikacija. Razvoj aplikacija završava njihovim testiranjem i puštanjem u rad, ali ovdje započinje nova faza, odnosno korištenje aplikacije koje podrazumijeva održavanje sve do njenog ukidanja. Slika 1. prikazuje redoslijed razvoja mobilne aplikacije.



Slika 1. Shematski prikaz razvoja mobilne aplikacije

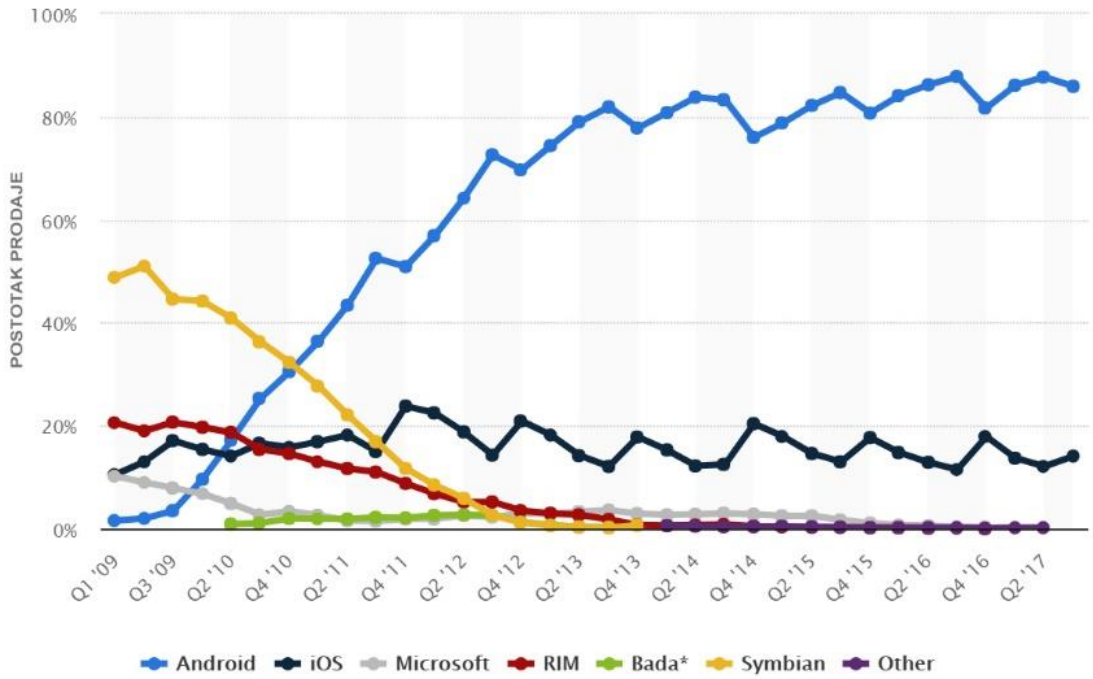
Izvor: http://www.ss-ivanec.hr/images/8_Prirucnik_Razvoj_mobilnih_aplikacija.pdf

S obzirom na velik broj aplikacija koje se nude korisnicima, njihovo tržište je veoma dinamično. Izrada kvalitetne projektne ideje kao i njena kvalitetna razrada ključne su za uspjeh mobilne aplikacije. Postoje dva osnovna principa razvoja projektne ideje, a to su nesistematski i sistematski pristup. Nesistematski pristup se temelji na brzom definiranju projektnih zadataka bez prethodnog istraživanja i segmentacije tržišta ili pozicioniranja na tržištu s obzirom na konkurenciju. Ovakav pristup se koristi pri razvijanju inovativne ideje u želji za što bržim spremanjem mobilne aplikacije za tržište. Nesistematski pristup razvoju projektne ideje se ne preporuča u razradi ideje u svrhu ostvarivanja profita.

Sistematski pristup podrazumijeva provođenje aktivnosti segmentacije tržišta i ciljanih korisnika u svrhu pozicioniranja u onom dijelu tržišta koje nije pokriveno ili zasićeno konkurentskim proizvodima, odnosno mobilnim aplikacijama iste domene. Neke od domena aplikacija koje nisu zasićene na tržištu su ekologija, zdravlje, energetska učinkovitost, poljoprivreda i slično. Nakon odabrane domene aplikacije, bitno je analizirati i istražiti navike korisnika. Proizvodi koji mijenjaju navike korisnika mogu naići na otpor njihovom prihvaćanju. Također, upravo takvi proizvodi koji nude nove mogućnosti postaju privlačni mlađim korisnicima koji lakše prihvaćaju promjene. Prepoznavanje novog trenda u navikama korisnika može donijeti značajnu konkurentsku prednost. Kod velikih projekata se za ovu aktivnost uključuju i vanjske stručne osobe iz područja marketinga ili psihologije s ciljem kreiranja inovativnih ideja za postojeće navike ili kreiranja novih trendova u korištenju mobilnih aplikacija. Tehnološki ciklus, odnosno vrijeme od pojave, preko prihvaćanja i konačnog zasićenja pojedinom tehnologijom postaje sve kraći. Pojavom nove tehnologije dolazi do otvaranja novih mogućnosti na već zasićenom tržištu. Analiza tehnoloških trendova i inovacija daje uvid u nove tehnologije te mogućnost za širenjem projektne ideje za mobilnu aplikaciju. Uzimajući u obzir utvrđene trendove, tehnologije, domenu razvoja i ciljane korisnike provodi se aktivnost osmišljavanja potencijalnih ideja, odnosno *brain storming*. Svaka ideja se temelji na ključnoj funkcionalnosti oko koje se definiraju pomoćne i dodatne stavke mobilne aplikacije u razvoju. Za svaku ideju koja se uzima kao moguće rješenja mobilne aplikacije potrebno je napraviti analizu konkurentskih proizvoda na tržištu te utvrditi njihove nedostatke, kritike korisnika i mogućnosti kreiranja boljeg i kvalitetnijeg proizvoda. U proces daljnjeg razvoja se kreće s onom idejom koja je izvediva u planiranom vremenu i s planiranim resursima te interesantna i korisna ciljanoj skupini. Analizom korisničkih zahtjeva izrađuje se arhitekturni i strukturni dizajn mobilne aplikacije, način njenog korištenja te njen izgled. U fazi implementacije odvija se izrada i testiranje programskog proizvoda, odnosno programiranje same mobilne aplikacije čime ona postaje funkcionalna. Testiranje je korak provjere njenih funkcionalnosti koje moraju zadovoljavati potrebe korisnika. U konačnici se gotov proizvod pušta u rad i održava objavama novih verzija. Kada izdavač odluči da aplikacija više nije zadovoljavajuća za njega i krajnje korisnike, ona se uklanja s tržišta. [4]

2.1.2 Mobilni operativni sustavi

Mobilni operativni sustav je softverska platforma koja je zadužena za izvršavanje osnovnih i dodatnih mogućnosti te za pokretanje aplikacija na mobilnim uređajima. Osnovne mogućnosti mobilnih operativnih sustava su mogućnost ostvarivanja glasovnih poziva, slanje tekstualnih poruka i mogućnost povezivanja na internet. Dodatne mogućnosti su one koje korisniku omogućavaju izbor načina na koji će izvršiti neku radnju, a neke od njih su ograničenje brzine ili potrošnje mobilnih podataka, opcije spremanja datoteka i opcije primanja obavijesti. Ranija izdanja mobilnih operativnih sustava bila su jednostavna. Razlog tome su bile ograničene mogućnosti mobilnih uređaja. Napredak mobilnih operativnih sustava je odgovor na tehnološki razvoj hardvera, softvera i interneta, odnosno web tehnologije. Današnji mobilni uređaji nude i više prostora za daljnji razvoj operativnih sustava zbog jakih glavnih i grafičkih procesora, velike memorije za pohranu podataka, mogućnosti obavljanja više radnji u isto vrijeme (eng. *multitasking*) te zaslona i mobilnih kamera visokih rezolucija. [5]



Slika 2. Statistički prikaz prodaje mobilnih operativnih sustava

Izvor: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/

Slika 2. prikazuje tržišnu zastupljenost vodećih mobilnih operativnih sustava kroz kvartale godina. Danas je najzastupljeniji operativni sustav pametnih telefona Android, s 85% ukupne prodaje u 2017. godini. Slijedi ga iOS sa 20% ukupne prodaje. Microsoft se nalazi na trećem mjestu sa svojim Windows phone operativnim sustavom čija je popularnost zbog nedostatka mobilnih aplikacija u stalnom opadanju. RIM (BlackBerry), Symbian (Nokia) i Bada (Samsung) su mobilni operativni sustavi koji nisu našli svoju primjenu u tehnološkom napretku te izumiru. [6]

Android mobilni operativni sustav je razvijen od strane Open Handset Alliance udruženja na čelu s Google-om. Temelji se na dorađenoj verziji Linux *kernel*-a te je otvorenog koda (eng. *open source*), a prvenstveno je zamišljen za uporabu na uređajima sa zaslonom osjetljivim na dodir. Probna inačica Android-a izlazi krajem 2007., a prva službena verzija, Android 1.0, izlazi 23. rujna 2008. godine. *Android* 2011. godine postaje najzastupljeniji mobilni operativni sustav. Do 2017. *Android* koristi više od dvije milijarde korisnika, a u 2018. Google Play store broji više od 3,3 milijarde mobilnih aplikacija. Posljednja verzija Android-a, izlazi krajem 2017. godine pod nazivom Android 8.1 „Oreo“. Sam operativni sustav napisan je u C programskom jeziku. Upravljački programi (eng. *drivers*), gotovo svih komponenti napisani su u C i C++ programskim jezicima. Mobilne aplikacije Android-a izrađuju se prema procesu koji se naziva Android Software development. Napisane su u Kotlin, Java i C++ programskim jezicima pomoću Android skupa razvojnih alata (eng. *Android software development kit*). Android *SDK* sadržava niz alata potrebnih za razvoj aplikacija za Android mobilni operativni sustav, a to su: program za ispravljanje pogrešaka (eng. *debugger*), knjižnice (eng. *libraries*), emulator temeljen na QUEMU, relevantna dokumentacija, uzorak koda i vodič kroz Android OS. Za razvoj Android mobilnih aplikacija pomoću Android *SDK* potrebno je bilo koje novije izdanje Linux desktop operativnog sustava, *Mac OS X 10.5.8*, *Windows 7* ili kasnija izdanja tih računalnih operativnih sustava. Do 2015. godine službeno prihvaćeno integrirano razvojno okruženje je bilo Eclipse sa Android Development Tools *(ADT)* dodatkom. Google 2015. prestaje razvijati *ADT* i izdaje novo službeno razvojno okruženje pod nazivom Android Studio. Programerima je dopušteno koristiti bilo koje programe za uređivanje teksta za pisanje kodova te bilo koje druge alate za izgradnju mobilnih aplikacija za Android mobilni operativni sustav. Android mobilne

aplikacije su spremljene u obliku paketa *.apk* formata, a nalaze se u mapi */data/app* operativnog sustava. [7]

iOS je mobilni operativni sustav koji je razvio Apple Inc. samo za svoje uređaje. Nakon Android-a drugi je najzastupljeniji mobilni operativni sustav, a koriste ga iPhone, iPad i iPod Touch uređaji. Izvorni kod iOS-a je zatvorenog tipa, a temelji se na *XNU kernelu* Darwin operativnog sustava otvorenog koda. Prvo izdanje iOS-a izlazi 2007. godine na iPhone uređaju. Iste godine postaje dostupan za iPod, a 2010. izlazi na iPad uređajima. Nove verzije iOS-a izlaze svake godine, a posljednja verzija, iOS 11, je izdana 2017. godine. App Store sadrži više od dva milijuna mobilnih aplikacija za iOS mobilni operativni sustav koje su preuzete 130 milijardi puta. Ovaj operativni sustav napisan je C i C++ programskim jezicima. Mobilne aplikacije za iOS razvijaju se pomoću iOS skupa razvojnih alata (iOS Software Development kit) u Objective C i Swift programskim jezicima. iOS *SDK* je dostupan i besplatan za korištenje na Mac računalima te nije predviđen za uporabu na Windows operativnom sustavu. Razvojno okruženje za izgradnju iOS aplikacija je Xcode. [8]

2.1.3 Vrste mobilnih aplikacija

Odabir pristupa razvoju mobilnih aplikacija direktno utječe na rezultat projekta. Prema vrsti mobilnih aplikacija, razlikujemo nativne (eng. *native*), web i hibridne (eng. *hybrid*) mobilne aplikacije. Postoji niz faktora koji uvjetuju odabir vrste mobilne aplikacije, a to su: ukupni budžet pripremljen za financiranje projekta, zadani rokovi u kojima projekt mora biti završen, ciljani korisnici i funkcija mobilne aplikacije u izradi. Svaki od pristupa donosi određene prednosti, ali i ograničenja. Ne postoji najbolji pristup, već je potrebno naći onaj koji zadovoljava većinu faktora. [9]

U prethodnom poglavlju opisani su uvjeti za izradu nativnih mobilnih aplikacija za Android i iOS mobilne operativne sustave. Ova vrsta aplikacija je najzastupljenija. Nativne mobilne aplikacije su izgrađene isključivo za određeni operativni sustav pa tako i programski jezik kojim su napisane mora odgovarati operativnom sustavu. Za Android aplikacije koristi se Java, a za iOS Swift i Objective-C programski jezici. Za izgradnju ovakvih aplikacija koristi se određeno razvojno okruženje i razvojni alati te određena

pravila kod izgradnje korisničkog sučelja prema zahtjevima operativnih sustava. Nativne aplikacije koriste izvršne datoteke koje su preuzete i pohranjene na uređaju. Kôd aplikacija se izvršava na mobilnom operativnom sustavu. Najčešći i najjednostavniji način za preuzimanje nativnih aplikacija je putem platforma za njihovo preuzimanje, a najpoznatije su App Store i Google Play Store. Instalacijom nativne aplikacije, ona dobiva pristup aplikacijskom programskom sučelju (eng. *application programming interface, API*) mobilnog operativnog sustava. *API* je skup određenih metoda i funkcija te podatkovnih struktura, njihovih pravila i specifikacija. Pristupom *API-u,* mobilna aplikacija dobiva mogućnosti koje su karakteristične za sam mobilni operativni sustav. Postoje dvije razine aplikacijskih programskih sučelja. Niža razina (eng. *low-level API*) aplikaciji omogućava korištenje osnovnih mogućnosti kao što su: korištenje tipkovnice, povezivanje na internet, korištenje kamere, mikrofona ili zvučnika i slično. Viša razina (eng. *high-level API*) služi za prilagodbu uređaja korisniku pristupom kalendaru, povijesti pretraživanja, kontaktima, fotografijama i drugim mogućnostima. Osim aplikacijskog programskog sučelja, kod izgradnje nativne mobilne aplikacije koristi se i niz pravila grafičkog korisničkog sučelja (eng. *graphical user interface, GUI*) mobilnog operativnog sustava. Nativne aplikacije imaju brz odaziv, responzivne su i optimizirane za korištenje na određenom mobilnom operativnom sustavu. Za korištenje ove vrste aplikacija nije potrebna povezanost na internet ukoliko to ne umanjuje njenu funkcionalnost. Nativne aplikacije pružaju najbolje korisničko iskustvo, vrlo su interaktivne i jednostavne za korištenje. Zbog složenosti procesa izgradnje ove vrste aplikacija, njihova uporaba se izbjegava kod jednostavnih mobilnih aplikacija. Razvoj nativnih mobilnih aplikacija je skup, ali privlači više korisnika, a time i poveća mogućnost financijske dobiti. [10]

Internet preglednici pametnih telefona podržavaju mnoge mogućnosti HTML 5 prezentacijskog, CSS 3 opisnog i JavaScript programskog jezika. Web aplikacije su aplikacije izgrađene pomoću navedenih web tehnologija, a pokreću se unutar internet preglednika. Kôd web mobilnih aplikacija se izvršava na serveru, za razliku od nativnih kod kojih se kôd izvršava na operativnom sustavu. Web aplikacije ne moraju biti preuzete i instalirane na uređaju, a od web stranica se razlikuju po tome što im je sadržaj sažet te je naglašena njihova funkcionalnost. Također, moguće ih je pokrenuti stvaranjem prečaca (eng. *shortcut*) na početnom zaslonu što ih čini jednako jednostavnima za korištenje kao i nativne aplikacije. Kako bi se olakšala izgradnja web aplikacija, postoji niz alata i

knjižnica za Javascript jezik, a najpoznatije su: jQuery Mobile, Sencha Touch i dojox.mobile. Većina internet preglednika koristi WebKit alat za iscrtavanje (eng. *rendering engine*). S obzirom na to da su web mobilne aplikacije napisane standardnim web jezicima kompatibilnim s WebKit-om, one pružaju mogućnost uporabe s jednakim korisničkim iskustvom na svim mobilnim operativnim sustavima. Sam razvoj ovakve vrste mobilnih aplikacija je jednostavan i nije skup. Njihovo održavanje je lako i ni na koji način ne uključuje korisnika te time ne ometa njegovo korištenje mobilne aplikacije. Internet preglednici koji pokreću web mobilne aplikacije su nativne aplikacije koje imaju pristup *API*-u mobilnog operativnog sustava. Same web aplikacije imaju pristup vrlo malom dijelu *API*-a što ih čini manje interaktivnim te zbog toga nemaju pristup mogućnostima uređaja. Također, web aplikacije su sporije od nativnih. Progresivne web aplikacije (eng, *progressive* web *apps*) su napredne web aplikacije koje, poput nativnih aplikacija, koriste neke od mogućnosti mobilnog uređaja kao što je primanje obavijesti (eng. *push messages*) vibracijom. Mana ove vrste web aplikacija je mogućnost korištenja samo u Google Chrome internet pregledniku što znači da ih korisnici iOS mobilnog operativnog sustava ne mogu koristiti.

Hibridna vrsta mobilnih aplikacija (eng. *Hybrid apps*) je pristup razvoju kojim se kombiniraju značajke nativnih i web aplikacija. Veliki dio aplikacije je napisan višeplatformskom (eng. *cross-over*) web tehnologijom, dok je nativan tek onaj dio aplikacije zadužen za pristup *API*-u mobilnog operativnog sustava. Na ovaj način aplikacije postaju višeplatformske, a moguće ih je koristiti kao nativne. Hibridne aplikacije je potrebno preuzeti instalirati na mobilni uređaj. Web tehnologija koja se koristi za izradu *cross-over* dijela aplikacije uključuje HTML, CSS i Javascript. Kôdovi hibridne mobilne aplikacije se mogu nalaziti na serveru ili mogu biti pohranjeni na uređaju unutar aplikacije. Ukoliko se kôd nalazi na serveru, proces održavanja i ažuriranja (eng. *update*) aplikacije je jednostavniji, ali aplikaciju u tom slučaju nije moguće koristiti bez povezivanja na internet. Spremanjem kôda u datoteku aplikacije se postiže brži odaziv kod korištenja, ali održavanje je otežano. Nativni dio hibridne mobilne aplikacije služi za povezivanje (eng. *wrapper*) preglednika i aplikacijskog programskog sučelja uređaja. Na taj način se omogućava korištenje svih mogućnosti mobilnog uređaja kao kod nativnih mobilnih aplikacija. Za prikaz hibridnih aplikacija koristi se web prikaz (eng. *WebView*) koji je jednostavniji od web preglednika te ne sadrži opcije koje sadrži

preglednik. Kako bi jedna aplikacija bila podržana na više mobilnih operativnih sustava, koristi se Cordova biblioteka (eng. *library*) otvorenog koda. Razvoj hibridnih mobilnih aplikacija je jednostavniji i ekonomski isplativiji od nativnih aplikacija. Sporije su od nativnih aplikacija. Na Slici 3. su prikazane vrste mobilnih aplikacija te njihov princip rada.

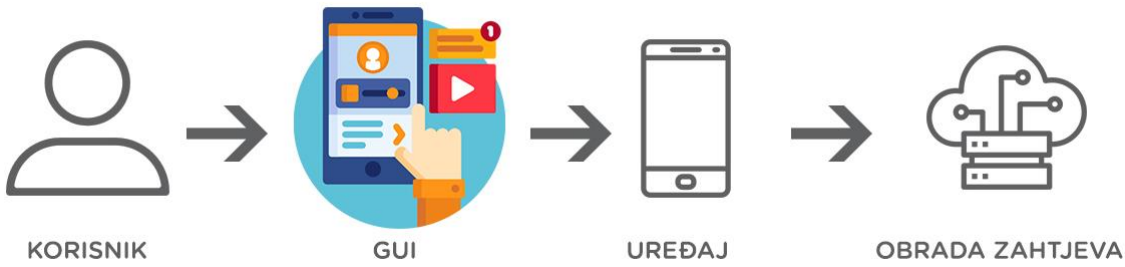
Slika 3. Vrste mobilnih aplikacija

Izvor: https://developer.telerik.com/featured/what-is-a-webview/

2.1.4 Grafičko korisničko sučelje mobilnih aplikacija

Grafičko korisničko sučelje (eng. *graphical user interface, GUI*) je način interakcije korisnika s uređajima kroz manipulaciju grafičkim elementima. Uz pomoć tekstualnih poruka i obavijesti, grafičko korisničko sučelje je posrednik u komunikaciji korisnika i uređaja, s ciljem ispunjenja zahtjeva korisnika (Slika 4.). Ovakvom obliku interakcije prethodilo je tekstualno sučelje (eng. *command-line interface, CLI*) preko kojeg je korisnik upravljao sučeljem pomoću naredbenog retka. Prva izdanja grafičkog korisničkog sučelja su zamišljena za upravljanje putem miša i tipkovnice. Izumom mobilnih uređaja, ono je prilagođeno za upravljanje putem zaslona osjetljivih na dodir ili drugih tehnologija zaduženih za interakciju.

Prednost korištenja grafičkog korisničkog sučelja je jednostavnija uporaba uređaja. Grafički elementi su razumljivi većini korisnika, neovisno o njihovoj razini tehnološkog znanja. Također, grafičko korisničko sučelje daje trenutačan odaziv na naredbe korisnika te time korisniku daje uvid u upravljanje uređajem u stvarnom vremenu.

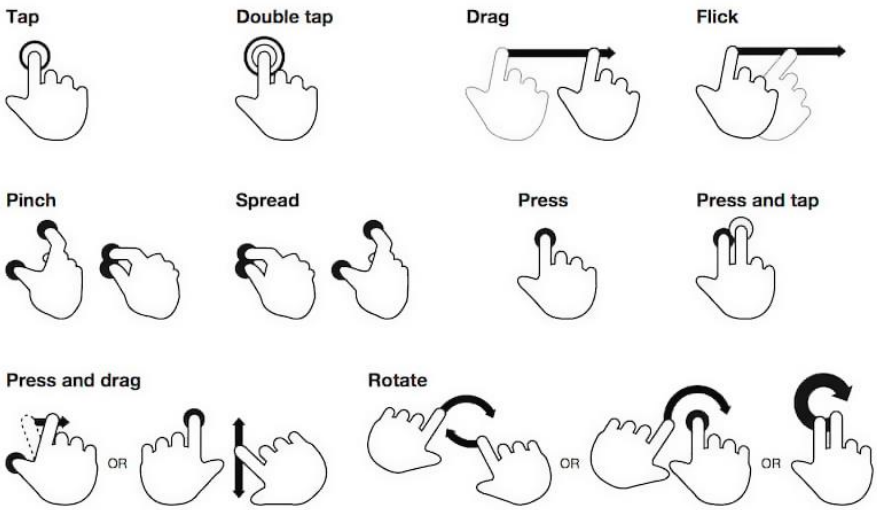


Slika 4. *GUI* kao posrednik komunikacije između korisnika i uređaja

Elementi grafičkog korisničkog sučelja tvore konzistentan vizualni jezik kojim prenose informacije do korisnika. [11] Razlikuju se dvije vrste grafičkih korisničkih sučelja. Strukturni, koji grade grafičko korisničko sučelje i interaktivni, koji korisniku daju povratnu informaciju o njegovoj interakciji s uređajem, sustavom ili aplikacijom. Prozori su glavni strukturni grafički elementi korisničkog sučelja. To su površine zaslona koje prikazuju informacije neovisno o ostatku zaslona. Za svaku novu otvorenu mapu, datoteku ili aplikaciju, otvara se novi prozor. Zbog mogućnosti uređaja otvaranja više različitih prozora odjednom, korisnik može obavljati više radnji u isto vrijeme. Izbornici (eng. *menu*) korisniku pružaju odabir ponuđenih naredbi s ciljem proširenog i olakšanog upravljanja sustavom. Ikone su mali vizualni elementi koji predstavljaju zasebne jedinice grafičkog korisničkog sučelja kao što su datoteke, aplikacije, web stranice ili određene naredbe. Služe za brzo izvršavanje naredbi, odnosno za otvaranje aplikacija, datoteka ili mapa. Kontrolni grafički elementi korisničkog sučelja omogućuju direktnu manipulaciju i upravljanje sučeljem, odnosno aplikacijom. Svaki od kontrolnih grafičkih elemenata služi za određenu vrstu interakcije. Neki od ove vrste elemenata grafičkog korisničkog sučelja su padajuće liste i kvadratići za odabir (eng. *check box*) koji nude jedan ili više odabira, kružići za odabir (eng. *radio button*) koji nude jedan odabir i klizači (eng. slider) koji omogućavaju stupanj odabira. Grafički elementi korisničkog sučelja koji se koriste

za navigaciju, odnosno orijentaciju su poveznice (eng. *link*), listanje (eng. *scroll*) i kartice (eng. *tab*). Kartice omogućavaju korištenje jednog prozora kao grupu manjih zasebnih cjelina. Internet preglednici koriste kartice za olakšano pregledavanje više web stranica istovremeno. U grafičke elemente korisničkog sučelja spadaju i interaktivni kalendari, satovi, kalkulatori i bilješke. Grafički elementi interakcije opisuju upravljanje korisnika uređajem. Neki od ove vrste elemenata su pokazivači (eng. *cursor, pointer*), isticanje označenog teksta (eng. *selection*) i mogućnost ubacivanja teksta (eng. *insert*).

Korisnik koristi gestikularnu komunikaciju za upravljanje korisničkim sučeljima uređaja sa zaslonima osjetljivim na dodir. Gestikularna komunikacija je set pokreta prstiju s kojima se izvršavaju željene radnje u sustavu ili aplikaciji. Pokreti gestikularne komunikacije najčešće ne ovise o uređaju, mobilnom operativnom sustavu ili aplikaciji te su sveopće prihvaćeni od strane korisnika. Najčešći pokreti prikazani su slikom 5.



Slika 5. Gestikularna komunikacija

Izvor: https://www.smashingmagazine.com/2017/02/touch-gesture-controls-mobile-interfaces/

Grafička korisnička sučelja mobilnih aplikacija uvelike utječu na korisničko iskustvo, a time i na prihvaćenost neke aplikacije. Zbog toga je kod izrade aplikacije bitno prilagoditi koncept sučelja na način da korisnik ne nailazi na prepreke u njegovu korištenju.

2.1.5 UX / UI

Kako bi neka aplikacija bila prihvaćena od strane korisnika, njeno grafičko korisničko sučelje mora biti praktično za uporabu i vizualno privlačno. Prema tome se razvoj korisničkog sučelja dijeli na korisničko iskustvo (eng. *user experience, UX*) i izgled korisničkog sučelja (eng. *user interface, UI*). Razlika korisničkog iskustva i izgleda korisničkog sučelja je prikazana na slici 6.

Slika 6. Razlika između pojmova *UX* i *UI*

Izvor: https://youteam.io/blog/ui-vs-ux-the-vital-guide-to-ui-design/

Dizajn korisničkog iskustva je proces kojim se pozitivno utječe na doživljaj korisnika pri interakciji s aplikacijom čineći ju jednostavnom za korištenje, pristupačnom i praktičnom te bolje pozicioniranom na tržištu. Kako bi se stvorilo pozitivno korisničko iskustvo, odnosno pozitivna percepcija aplikacije od strane korisnika, postoji niz kriterija koji moraju biti zadovoljeni. Prvi od koraka je istraživanje ciljanih korisnika. Postavljaju se pitanja o njihovim potrebama, željama i očekivanjima s ciljem pronalaženja boljih rješenja. Također, važno je sagledati pozitivne i negativne značajke sličnih proizvoda na tržištu. Idući korak u izradi dizajna korisničkog iskustva sučelja je izrada skice (eng. *wireframe*). Cilj izrade skice je odrediti raspored grafičkih elemenata prije dodavanja

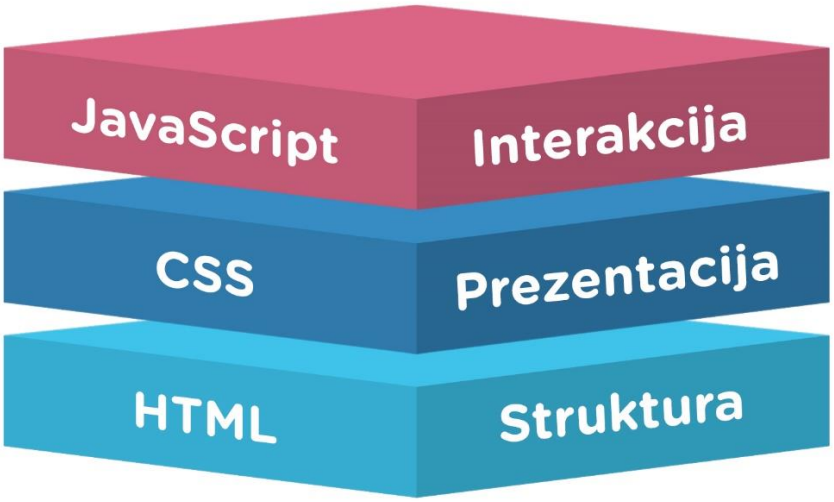
vizualnih rješenja. Nakon izrade dizajna korisničkog sučelja, proizvod se testira kao prototip, usporedbom više mogućih rješenja ili uz pomoć praćenja korisnika koji koristi aplikaciju kako bi se uvidjele moguće prepreke u ostvarivanju željenih ciljeva. Nakon lansiranja web stranice ili aplikacije, bitno je pratiti reakcije korisnika kako bi se naknadno ispravile značajke koje umanjuju zadovoljstvo korisnika.

Dizajn korisničkog sučelja bazira se na izradi samog izgleda aplikacije ili web stranice i njihove prilagodbe za sve uređaje. Cilj dizajna korisničkog iskustva je stvaranje vizualno prihvatljive interaktivne okoline. Kod dizajna korisničkog sučelja, bitno je primijeniti značajke određene dizajnom korisničkom iskustva, odnosno, izgled proizvoda ne smije negativno utjecati na njegovu funkcionalnosti. [12] Kao alati za izradu dizajna korisničkog sučelja, najzastupljeniji su: Sketch za korisnike Mac OS operativnog sustava i Adobe XD koji je kao alat namijenjen isključivo za dizajn korisničkog sučelja zamijenio Adobe Photoshop. U fazi testiranja, odnosno prezentacije dizajna korisničkog sučelja koriste se prototipi i animacije kao simulacije korištenja budućih web stranica ili aplikacija. Alati koji se koriste za njihovu izradu su: Principle i Flinto za Mac OS korisnike te ProtoPie i Adobe After Effects. Ovi alati se koriste i za takozvane mikrointerakcije čija je zadaća odgovor na interakciju s aplikacijom. One korisniku daju do znanja da je njihova akcija izvršena.

Kako bi korisnici odlučili koristiti neku aplikaciju, važno je da njihov prvi dojam bude pozitivan. Dizajn korisničkog sučelja je prvo što korisnici opažaju pri ostvarivanju interakcije s aplikacijom. Prema istraživanju Google-a, 26% instaliranih aplikacija koristi se samo jednom, a razlog tome je nezadovoljstvo korisnika izgledom aplikacije. Praćenjem trendova u izradi dizajna korisničkog sučelja nastoji se korisniku pružiti vizualno prihvatljiv i funkcionalan proizvod. Iako se trendovi mijenjaju, njihov osnovni princip ostaje isti, a to je pružiti korisniku proizvod koji je u skladu s tehnološkim napretkom te koji ne mijenja navike korisnika. Također, kako bi se ostvarilo određeno raspoloženje korisnika, bitan je odabir palete boja koja se koristi u dizajnu. Ukoliko sučelje izgleda zadovoljavajuće, korisnici dobivaju povjerenje pri korištenju usluga ili proizvoda prezentiranih aplikacijom ili web stranicom. U izradi dizajna mobilnih aplikacija, naglasak se stavlja na korisničko iskustvo, odnosno na funkcionalan proizvod. Teži se izgradnji jednostavnih i interaktivnih korisničkih sučelja. [13]

2.2 Web tehnologije

Web tehnologija je pojam koji se odnosi na skup jezika i multimedijskih paketa koji se koriste s ciljem stvaranja dinamičkog web sadržaja (*DHTML*). Svaka pojedina tehnologija ima svoju zadaću, ali i ograničenja te se one uvijek nadopunjuju, odnosno koriste se zajedno. Upotrebljavaju se one tehnologije s kojima je moguće ostvariti krajnji cilj. Tehnologije su međusobno zavisne jedna o drugoj. Najzastupljenije tehnologije za izgradnju sučelja (eng. *frontend*) dinamičkog *web* sadržaja su: HTML, CSS i JavaScript. Uloga svake od navedenih tehnologija prikazana je na slici 7. Kako bi se olakšala izgradnja web stranica i aplikacija, postoji niz knjižnica (eng. *library*) i razvojnih okruženja, odnosno kostura (eng. *framework*). Njihovom upotrebom je moguće postići željeni cilj u kraće vrijeme, kraćim kodovima, a osim toga, smanjene su i mogućnosti pogrešaka. *Library* je set unaprijed definiranih klasa i metoda koje se koriste, odnosno pozivaju po potrebi čime ne mijenja samu strukturu razvoja aplikacije ili web stranice. Na taj način se koristi već napisan kôd te je vrijeme razvoja kraće i jednostavnije. *Framework* je radno okruženje, odnosno kostur njenog razvoja. [14]

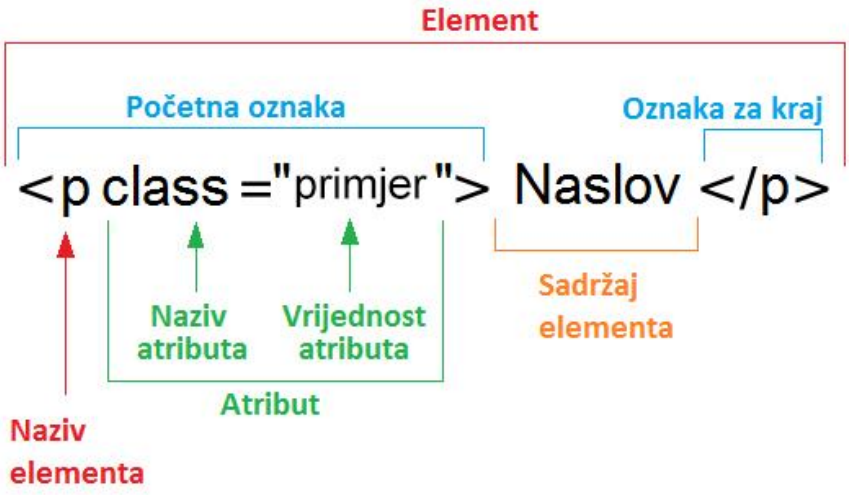


Slika 7. Uloga web tehnologija

Izvor: http://www.cagrimmett.com/development/2017/04/24/tools-for-learning-css.html

2.2.1 HTML

HyperText Markup Language, odnosno HTML, je prezentacijski jezik za izradu strukture internet stranica. Pomoću HTML jezika oblikuje se njihov sadržaj te se stvaraju hipertekstualni dokumenti i hiperveze takvih dokumenata. Internet preglednici preko HTML-a dobivaju informacije o sadržaju i strukturi internet stranice te zatim oblikuju stranicu u onakvu kakvu ju korisnik vidi. Hipertekst je naziv za tekstualnu strukturu koja se sastoji od međusobno povezanih informacija koje se prikazuju na nekom elektroničkom uređaju. HTML se ne smatra programskim jezikom jer se njime ne može izvršiti nikakva funkcija, već je kategoriziran kao označni jezik. HTML kôd se može pisati u jednostavnim programima za uređivanje teksta, a HTML datoteka je obična tekstualna datoteka s ekstenzijom .html ili .htm. Osnovni dio HTML-a su njegovi elementi koji se nalaze unutar svojih oznaka (eng. *tag*). Sadržaju HTML elementa moguće je dodijeliti dodatne informacije. Dodatne informacije HTML elementa nazivaju se atributi. Svaki element može imati atribute, ali svaki element ne može imati bilo koji atribut. Oni se pišu u otvarajućoj oznaci te se sastoje od dva dijela - naziva i vrijednosti. Opći oblik pisanja atributa je naziv=”vrijednost”, (eng. *name=”value”*). Naziv atributa upućuje na vrstu informacije koja se pridodaje HTML elementu, a vrijednost je informacija dana atributu. Slika 8. prikazuje strukture HTML elementa.



Slika 8. HTML element

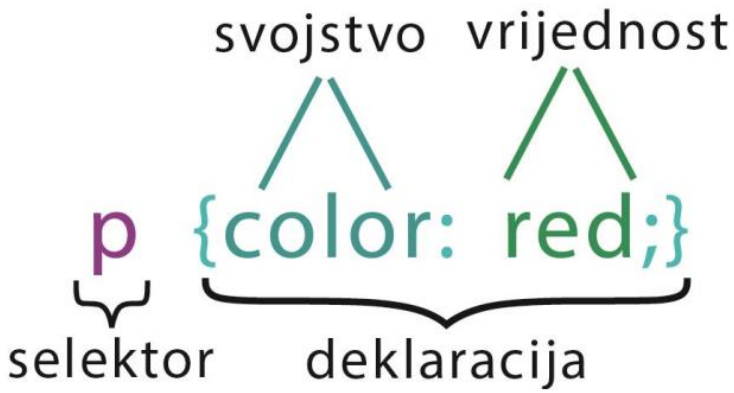
Izvor: http://www.webtech.com.hr/html.php

HTML je jednostavan jezik koji razumije svaki internet preglednik. Besplatan je za korištenje zbog čega je postao neizostavan dio u izgradnji web stranica. Posljednja izdana inačica HTML-a je HTML 5 kojom su stvorene nove mogućnosti u oblikovanju web sadržaja. Kako bi se proširilo područje prikaza grafičkog i multimedijskog sadržaja, uvedeni su elementi *<video>, <audio> i <canvas>* te elementi kojim se definira struktura buduće stranice, a neki od njih su: *<main>, <section>, <article>, <header> i <footer>, <nav> i <figure>*.

2.2.2 CSS

Cascading Style Sheets, odnosno CSS je stilski jezik koji služi za opis prezentacije dokumenata napisanih pomoću označnog jezika. CSS definira kako vizualno prikazati HTML elemente. Iako je HTML najrašireniji označni jezik, CSS se može koristiti i kod opisa prezentacije drugih označnih jezika, npr: XHTML, XML ili SVG. Uz HTML i JavaScript, CSS je jedna od temeljnih web tehnologija. CSS je osmišljen kako bi se odvojio pojam strukture dokumenta od njegova izgleda, odnosno prezentacije. Ovakvim odvajanjem pojednostavljeno je upravljanje izgradnje web stranica, smanjena je kompleksnost i ponavljanje s ciljem ostvarenja zamišljenog izgleda stranice. Osim toga, ovim jezikom je vraćena i prvobitna ideja HTML-a, koji je bio osmišljen da definira strukturu stranice, a ne i njen izgled pa je tako smanjena potreba osmišljanja neslužbenih elemenata. Također, velika prednost je uvođenje odvojenog .css dokumenta koji definira izgled svih stranica jedne internet stranice čime se ubrzao proces oblikovanja jer više nije bilo potrebno raditi na svakoj zasebnoj stranici. CSS se koristi i za prilagodbu prikaza stranice ovisno o veličini zaslona na kojem se prikazuje, odnosno za izradu responzivnih *web* stranica. Sintaksa CSS-a je jednostavna, a sastavljena je od engleskih riječi kojima su određena imena različitih stilskih svojstava. Sadrži skup pravila koja se odnose na elemente. Pravila određuju kako će oni biti prikazani. Sastoje od dva dijela, odnosno od selektora i deklaracije. Selektor upućuje na koji element se pravilo odnosi. Postoji više vrsta selektora, a ukoliko se neko pravilo odnosi na više elemenata, onda se nazivi elemenata odvajaju zarezom. Drugi dio pravila, koji se nalazi u vitičastoj zagradi, naziva

se deklaracija. Deklaracija govori kako će se element određen selektorom prikazati. Deklaracije se sastoje od dva dijela, a to su svojstva i vrijednosti koji su međusobno odvojeni dvotočkom, kao što prikazuje slika 9. Svojstvo deklaracije određuje koji se aspekt nekog elementa definira, na primjer: boja, font, visina ili širina. Vrijednost definira svojstvo uz koje stoji. Ukoliko se u jednoj deklaraciji definira više svojstava, njih je potrebno odvojiti znakom "točka-zarez". Selektori *id* i *class* opisuju elemente s istoimenim *id* ili *class* atributima. *Id* opisuje jedan element, dok *class* opisuje sve elemente pripadajuće klase.

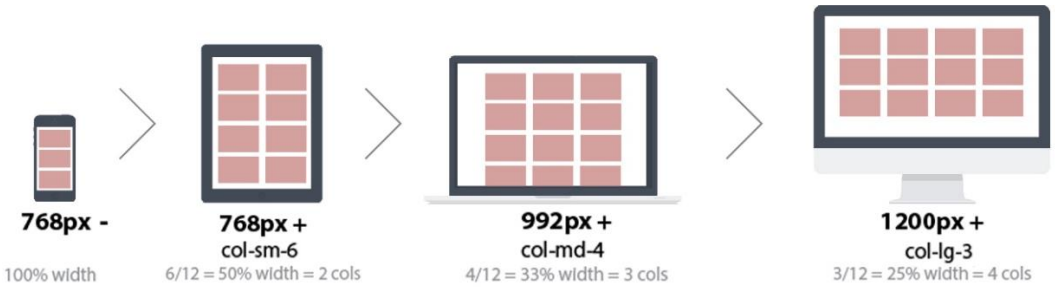


Slika 9. CSS pravilo

CSS-om je moguće na više načina odrediti stil jednog elementa. Zbog toga su određena pravila kojima je ustanovljena težina pojedinog načina određivanja stila. Ukoliko je stil istog elementa određen više puta, a način određivanja stila ima istu težinu svaki put, koristi se onaj stil koji je određen tako što je u kôdu napisan zadnji. Svaki element koji je smješten u neki element, odnosno *child* element preuzima stil elementa u kojem se nalazi, odnosno *parent* element, ukoliko nije posebno definiran stil *child* elementa. CSS se unutar HTML dokumenta može pisati na dva načina. Prvi način je pisanje stilova u zaglavlju HTML dokumenta, odnosno unutar elementa <*head*> u koji se ubacuje element <*style*>. Drugi način je unutar HTML oznaka, odnosno *inline* kao atribut *style*. Također, CSS može biti zaseban dokument u kojem su definirani stilovi elemenata, a on se povezuje s HTML dokumentom uz pomoć elementa <*link*>. [15]

2.2.3 Bootstrap

Bootstrap je HTML, CSS i JavaScript besplatan *framework* koji se koristi za izgradnju responzivnih web stranica ili web aplikacija. Razvijen je od strane Twitter-a s ciljem stvaranja alata kojim će se pojednostaviti upravljanje i prilagodba sadržaja web stranica ili web aplikacija. Prvo izdanje je postalo dostupno javnosti od 2011. godine, a trenutna verzija, Bootstrap 4, je izdana 2018. CSS ovog *framewrok-a* je izgrađen pomoću Less sustava. Bootstrap je podržan od strane svih internet preglednika. Sadržaj oblikovan pomoću Bootstrap-a se u potpunosti prilagođava prijenosnim uređajima (eng. *mobile first*). Bootstrap sadrži responzivnu mrežu prilagođenu prijenosnim uređajima koja se sastoji od redaka podijeljenih u dvanaest stupaca ovisno o veličini zaslona uređaja, odnosno veličini prikaza sadržaja (eng. *viewport*). Prikazom se upravlja određivanjem klasa elemenata koje definiraju njihovu veličinu u stupcima, ovisno vrsti uređaja na kojem se ti elementi prikazuju. Za uređaje širine prikaza manje od 768*px* koristi se klasa *.col-xs-*, za uređaje jednake ili veće širine prikaza koristi se klasa *.col-sm-*. Za uređaje čija je širina prikaza jednaka ili veća od 992*px,* koristi se klasa *.col-md-* te za širine prikaza veće ili jednake 1200*px,* koristi se klasa *.col-lg-*. Primjer mreže i uporabe na zaslonima različite širine se nalazi na *slici* 10.



Slika 10. Bootstrap mreža

Izvor: https://webscope.co.nz/optimising-your-front-end-development

Osim unaprijed definiranih klasa zaduženih za izradu responzivnog rasporeda sadržaja, Bootstrap uključuje i klase za olakšanu izradu responzivnih formi i tablica, za gumbe (eng. *buttons*) različitih veličina te za dodavanje responzivnih slika. React-

Bootstrap je prilagođena inačica Bootstrap *framework-a* kako bi se mogla koristiti za izradu aplikacija izrađenih pomoću React.js knjižnice. [16]

2.2.4 JavaScript

JavaScript je skriptni programski jezik više razine koji služi za izgradnju dinamičkih web stranica i aplikacija. Dinamičke web stranice omogućavaju međudjelovanje s korisnikom. JavaScipt je zadužen za stvaranje interaktivnog sadržaja. Ovaj programski jezik je pogodan za objektno orijentirano i funkcijsko programiranje (eng. *multi-paradigm*). Prvo izdanje JavaScripta izlazi 1995. godine, a izdaje ga NetScape. Dvije godine kasnije, JavaScript je standardiziran prema ECMAScript specifikacijama. Zadnje izdanje ECMAScript standarda izlazi 2015. godine pod nazivom ECMAScript 6. Gotovo sve web stranice su izgrađene pomoću JavaScripta, a svi suvremeni internet preglednici, na računalima i mobilnim uređajima, koriste JavaScript interpretator (eng. *interpreter, scripting engine*) što JavaScript čini jednim od najzastupljenijih programskih jezika. *Interpreter* koristi JavaScript kôd i prevodi ga u set uputa prema kojima internet preglednik izvršava tražene naredbe. Kao i HTML i CSS, kôd JavaScripta može biti napisan u bilo kojem programu za uređivanje teksta, a sprema se kao datoteka s ekstenzijom *.js.* Kako bi pisanje kôda bilo olakšano, postoji dokumentacija u kojoj su opisani korištenje i mogućnosti JavaScripta.

JavaScript razlikuje velika i mala slova (eng. *case*-sensitive). Kôd JavaScripta, odnosno skrpita, je niz uputa koje internet preglednik slijedi jednu po jednu. Svaka zasebna uputa ili korak naziva se izjava (eng. *statement*). Izjave započinju u zasebnim redovima, a kako bi interpretator prepoznao kraj neke izjave, potrebno ga je označiti znakom točka-zarez. Niz izjava koje se nalaze unutar vitičastih zagrada čini blok kôda. Izjava uvjeta služi za određivanje dijela kôda koji će se izvršiti. Pored ključne riječi se u zagradi piše uvjet koji mora biti ispunjen kako bi se izvršio blok kôda u nastavku. Ukoliko uvjet nije ispunjen, izvršava se onaj blok kôda koji se nalazi nakon ključne riječi *else.*

Kako bi bilo moguće upravljati informacijama i koristiti ih u kôdu, one se pohranjuju kao varijable. Varijable su promjenjive, odnosno njihova vrijednost ne mora biti stalna. U kôdu se varijable označavaju sa ključnom riječi *var* iza koje se piše njeno ime. Ključne

riječi omogućavaju interpretatoru daljnje upravljanje s kôdom. Označavanje varijable se naziva deklaracija. Nakon imena varijable, dodjeljuje joj se vrijednost, odnosno vrsta informacije. Osim ključne riječi *var,* varijable se mogu deklarirati i s ključnom riječi *let.* Takve varijable su vidljive samo unutar određenog bloka kôda, odnosno imaju lokalni domet (eng. *scope*). Ključna riječ *const* deklarira varijable konstantnih vrijednosti, odnosno vrijednosti koje se nikada ne mijenjaju. JavaScript razlikuje nekoliko različitih vrsta informacija, a to su: brojevi (eng. *number*), tekst (eng. *string*), logičke vrijednosti (eng. *boolean*), objekte, funkcije i polja. Kako bi interpretator razaznao brojeve od teksta, *string* se kao vrijednost piše između navodnika. Logičke vrijednosti varijable mogu biti istina (eng. *true*) ili laž (eng. *false*). Slika 11. prikazuje izjave koje su napisane zelenom bojom i izjave uvjeta koje su napisane ljubičastom bojom. Izjave koje se nalaze unutar rozih vitičastih zagrada tvore blok kôda.

```
var danas = new Date();
var vrijeme = danas.getHours();
var pozdrav;

if (vrijeme > 18) {
pozdrav = "Dobra večer!"
} else if  (vrijeme > 12) {
pozdrav = "Dobar dan!"
} else if  (vrijeme > 0) {
pozdrav = "Dobro jutro!"
}

document.write(pozdrav);
```

Slika 11. JavaScript kôd

Funkcije omogućavaju grupiranje izjava s ciljem izvršavanja određenog zadatka onda kada je to potrebno, odnosno do trenutka pozivanja funkcije. Pozivanjem funkcije se izbjegava stalno ponavljanje kôda čime i kôd izgleda urednije, a samim time je i lakši za održavanje. Izjave koje čine neku funkciju se nalaze unutar njenog bloka kôda. Parametri funkcije su dodatne informacije koje su potrebne za njeno izvršavanje. Ukoliko je zadaća funkcije vratiti neku dobivenu vrijednost, koristi se ključna riječ *return.* Funkcija se deklarira pomoću ključne riječi *function* i zadanog imena funkcije.

Objekti su skup varijabli i funkcija pomoću kojih se stvara model za daljnje korištenje. Varijable koje su dio objekta se nazivaju svojstva (eng. *properties*). Svaki objekt može imati različite vrijednosti nekog svojstva. Funkcija unutar objekta se naziva metoda (eng. *method*). Metode i svojstva se jednako kao i varijable i funkcije sastoje od svojeg naziva i vrijednosti. Naziv metoda i svojstva se naziva ključ (eng. *key*). Isti objekt ne može imati dva ključa istog naziva. Vrijednost svojstava može biti broj, tekst, logička vrijednost, polje ili drugi objekt. Vrijednost metode je neka funkcija. Postoji nekoliko načina izgradnje objekta. Objekt može biti zapisan kao varijabla čija je vrijednost niz svojstava zapisanih u vitičastim zagradama. Također, vrijednost varijable mogu biti i ključna riječ *new* te unaprijed zadana funkcija *Object()*. Svojstva ovako izgrađenog objekta se pridodaju na način da se pišu iza imena varijable, odijeljeni točkom. Ukoliko je potrebno stvoriti nekoliko objekata s istim svojstvima, tada se oni stvaraju uz pomoć funkcije koja služi kao predložak. Svaki novi objekt stvoren uz pomoć funkcije kao predloška stvara se kao varijabla čija je vrijednost ključna riječ *new* te naziv funkcije čiji su parametri vrijednosti svojstava zadanih predloškom. Slika 12. prikazuje ovakav način stvaranja objekata. Funkcija naziva Hotel prima tri parametra koji definiraju svojstva budućih objekata. Također, ova funkcija sadrži metodu koja računa slobodna mjesta u pojedinom objektu koji predstavlja hotel. [17]

```
function Hotel(name, rooms, booked) {
  this.name = name;
  this.rooms = rooms;
  this.booked = booked;
  this.checkAvailability = function() {
    return this.rooms - this.booked;
  };
}

var Hotel1 = new Hotel('Hotel1', 40, 25);
var Hotel2 = new Hotel('Hotel2', 120, 77);
```

Slika 12. Stvaranje objekata

2.3 React.JS

React.JS je JavaScript knjižnica (eng. *library*) koja se koristi za izgradnju interaktivnih korisničkih sučelja web stranica i mobilnih aplikacija. Osmislio ga je Facebook kao odgovor na sve veće probleme s održavanjem postojećeg kôda, nastale stalnim proširivanjem već tada najpoznatije društvene mreže. Prototip React-a je napravljen 2011. pod nazivom FaxJS te je primijenjen kao element za pretraživanje Facebook-a. Godinu kasnije je razvijen ReactJS koji 2013. postaje dostupan javnosti, *open source*. Daljnjim usavršavanjem i stvaranjem brojnih alata s kojima se olakšava korištenje React.JS-a, on postaje jedna od najpoznatijih JavaScript knjižnica. Zbog vrlo dobrih rezultata u smislu korisničkog iskustva, prihvaćen je od strane mnogih popularnih web servisa. Facebook je 2015. izdao *framework* za izgradnju nativnih aplikacija pomoću React-a, pod nazivom React Native.

2.3.1 Virtualni *DOM*

DOM (eng. *Document Object Model*) predstavlja razgranatu strukturu HTML elemenata koja se stvara prilikom učitavanja stranice. Kod uobičajenih stranica, korisniku se prikazuje upravo ta struktura. *Single-page* web aplikacije ili web stranice kod interakcije s korisnikom ne učitavaju nove stranice sa servera, već mijenjaju trenutni kôd stranice, odnosno kôd onog dijela koji se mijenja. Umjesto učitavanja cijele stranice, mijenja se prikaz samo onog dijela na kojem je napravljena izmjena. Kako bi to bilo moguće, moraju se stvoriti novi elementi koji će se prikazati ovisno o korisnikovoj interakciji. To je potrebno napraviti manipulacijom *DOM-a* koja se može izvesti pomoću JavaScripta. Internet preglednici nude JavaScript *DOM API* kojim je tu manipulaciju moguće izvesti. Manipulacija, odnosno mutacija *DOM-a* putem JavaScripta ima dva nedostatka. Prvi nedostatak je kôd koji je nepraktičan za održavanje, a drugi nedostatak je to što se takve manipulacije sporo izvršavaju. Manipulacija *DOM-a* je potrebna zato što web aplikacije nisu statične. One se tokom korištenja nalaze u određenim stanjima koja se predstavljaju korisničkim sučeljem koje preglednik iscrtava (eng. *render*). Stanja

mogu biti promijenjena događajima (eng. *event*). Postoje dvije vrste događaja koji utječu na stanje aplikacije, a to su događaji sa strane korisnika i događaji sa strane servera. Događaji sa strane korisnika podrazumijevaju interakciju korisnika s aplikacijom, a događaji na strani servera podrazumijevaju primanje informacija aplikacije sa servera, na primjer, primanje informacije o greški. Prilikom događaja se ažuriraju stanja podataka s kojima aplikacija raspolaže. Promjenom stanja podataka, potrebno je promijeniti i stanje korisničkog sučelja. Potrebno je postići sinkronizaciju između promjena stanja. React.JS koristi virtualni *DOM*. Virtualni *DOM* je brzi prikaz stvarnog *DOM-a* pohranjen u memoriji. Prilikom svake promjene React uspoređuje dva virtualna *DOM-a*, *DOM* prije promjene i nakon promjene. Usporedbom nalazi njihovu razliku te ju primjenjuje na pravi *DOM*, mijenjajući pritom samo onaj dio na kojem dolazi do promjene. Ovakvim pristupom se smanjuje ometanje u korištenju aplikacije što pozitivno utječe na korisničko iskustvo.

Kako bi se olakšalo pisanje kôda React.JS-a, koristi se JSX sintaksa za JavaScript. Sintaksa JSX-a izgleda kao HTML te je pomoću nje lakše vizualizirati strukturu web aplikacije, odnosno njen *DOM*.

2.3.2 Komponente

Korisnička sučelja izgrađena pomoću React.JS-a su najčešće jednostrana (eng. *single-page*), a izgrađena su od ponovno upotrebljivih cjelina koje se nazivaju komponente (eng. *components*). Komponente korisničkog sučelja predstavljaju neovisne promjenjive dijelove stranice ili aplikacije, one su promjenjivi elementi virtualnog *DOM-a*. One, kao i JavaScript funkcije prihvaćaju proizvoljne unose (eng. *properties, props*).

Postoje dva načina definiranja komponente prikazana na slici 13. Prvi način je definiranje komponente kroz JavaScript funkciju. Funkcija postaje React komponenta jer prihvaća svojstvo *'props'* objekta i vraća (eng. *return*) React element. Ovakve komponente nazivaju se funkcijskim jer su građene pomoću JavaScript funkcije. Drugi način definiranja je pomoću ES6 klasa. Iako komponente definirane kao klase imaju dodatne značajke, oba načina daju jednako važeće React komponente. Unutar svake komponente napisane kao klasa nalazi se funkcija, odnosno metoda *render()*. Ova

funkcija preispituje sva zadana stanja i svojstva, a vraća ono što se prikazuje na zaslonu korisnika, najčešće React elemente. Postoje dva načina za proslijediti podatke ovoj funkciji, a to je pomoću svojstava (*this.props*) i stanja (*this.state*).

Slika 13. React komponenta

Izvor: https://reactjs.org/docs/components-and-props.html

2.3.3 Stanja i ciklusi

Kako bi neki React element korisničkog sučelja mogao mijenjati svoj izgled ovisno o interakciji korisnika i zadanim promjenama, potrebno mu je zadati stanje. React elementi ne mogu imati zadano stanje (eng. *stateless*). Njihova zadaća je izgraditi strukturu virtualnog *DOM-a*. Zato ih je potrebno umetnuti unutar React komponenata. Komponente mogu imati zadana stanja (eng. *state*). Svaki događaj, odnosno korisnikova akcija može mijenjati ta stanja, a svako stanje može biti prezentirano određenim React elementima.

React komponente opisuju što će biti prikazano pomoću funkcije *render()*. Funkcija *render()* ne može utjecati sama na sebe, odnosno na to hoće li se prikaz izvršiti. Svaka komponenta prolazi nekoliko faza, a to su faza prije prikazivanja, prikazivanje i nakon prikazivanja. Ove faze se zovu ciklusi komponente (eng. *lifecycle*), a oni su funkcije kao i funkcija *render()*. Ciklusi omogućavaju veću kontrolu nad prikazom komponenti, a mogu biti svrstani u tri faze. Prva faza je *mounting*, a događa se za vrijeme kada se komponenta postavlja u *DOM*. Druga faza je *updating* i ona nastupa kada se

komponenta ponovno prikazuje u virtualnom *DOM-u* nakon čega dolazi do traženja razlike i promjene stvarnog *DOM-a*. *Unmounting* je posljednja faza, a nastupa kada se komponenta uklanja iz *DOM-a*. Ciklusi mogu biti određeni funkcijama: *componentWillMount(), componentDidMount(), componentWillUnmount()* i *componentDidUnmount()*. Također, faza *updating* može biti određena funkcijama kao što su: *componentWillUpdate()* i *componentDidUpdate()*.

2.3.4 Uvjetno prikazivanje

Uvjetno prikazivanje u React.JS-u funkcionira na isti način kao i u JavaScriptu. Na ovaj način moguće je prikazati samo neke komponente, ovisno o stanju aplikacije, odnosno o ispunjenom uvjetu. Za pisanje kôda koristi se *if* operater. Za druge uvjete, koristi se *else*, odnosno *else if*. Moguće je koristiti i operatere uvjeta „:“ i „?“ kao skraćeni oblik u formi *condition ? true : false*. Ovdje „?“ predstavlja if, a „:“ predstavlja else uvjete. Uvjetnim se prikazivanjem može spriječiti neku komponentu od prikazivanja. Primjer uvjetnog prikazivanja nalazi se na slici 14. [18]

```
render() {
  const isLoggedIn = this.state.isLoggedIn;
  return (
    <div>
      The user is <b>{isLoggedIn ? 'currently' : 'not'}</b> logged in.
    </div>
  );
}
```

Slika 14. Uvjetno prikazivanje

Izvor: https://reactjs.org/docs/conditional-rendering.html

3. EKSPERIMENTALNI DIO

3.1 Razvoj ideje

Ideja aplikacije koja se razvija u nastavku ovog rada jest predstavljanje kandidata, odnosno autora aplikacije, poslodavcu. Aplikacija je interaktivan životopis koji, osim informacija o samom kandidatu, na zanimljiv način daje uvid i u kompetencije na području *frontend* razvoja. Cilj aplikacije je isticanje kandidata nad konkurencijom na tržištu rada jedinstvenim pristupom prezentaciji sebe i svojih vještina.

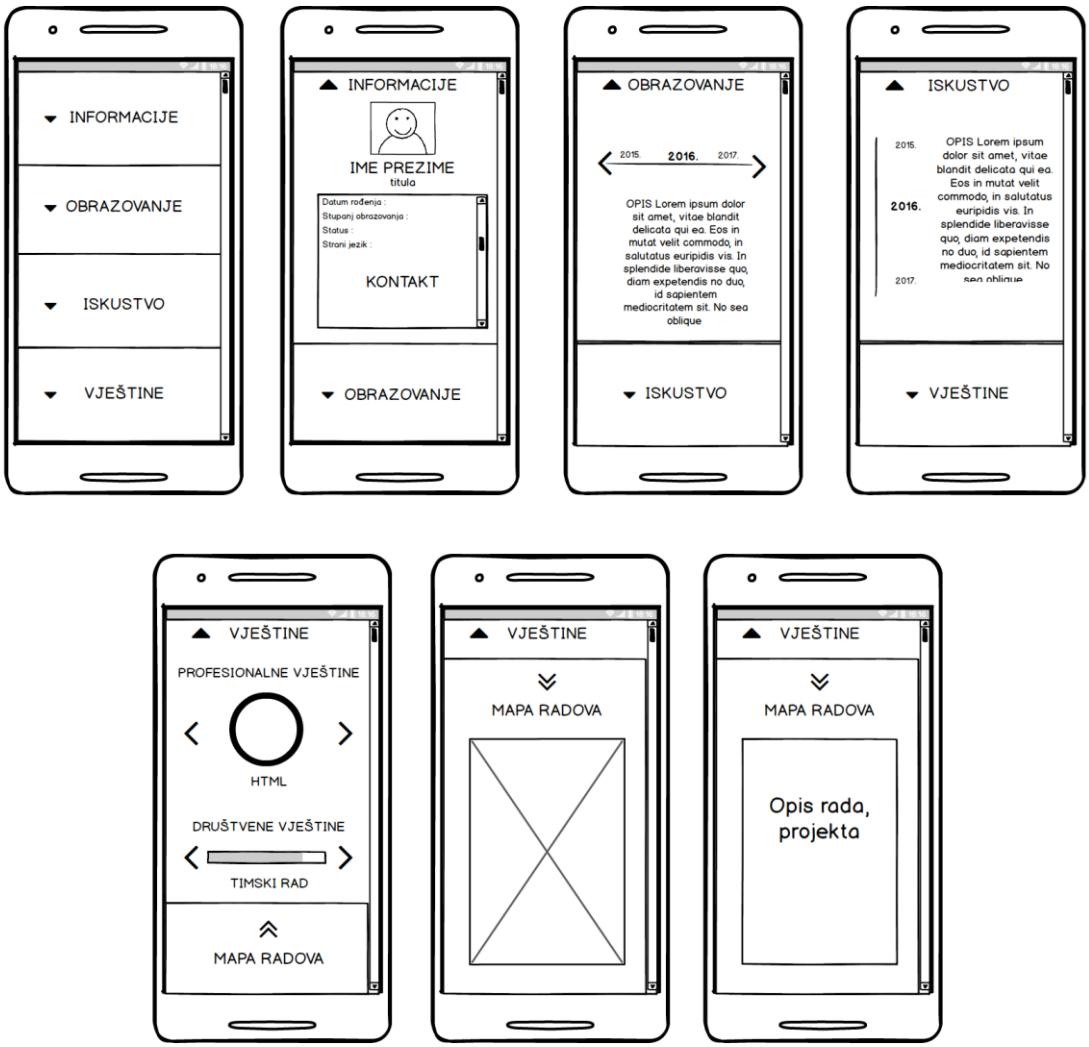
S obzirom na to da je aplikacija napravljena u svrhu istraživanja mogućnosti React.JS JavaScript knjižnice, ciljana publika ove aplikacije je manji broj ljudi koji se može podijeliti u dva dijela. Prvi dio čini kandidat, koji je u ovom slučaju ujedno i autor aplikacije. Drugi dio je skupina ljudi su potencijalni poslodavci kojima se kandidat predstavlja pomoću aplikacije. Budući da je u oba slučaja riječ o ljudima u mlađoj ili starijoj odrasloj dobi, aplikacija je namijenjena dobnoj skupini ljudi od dvadeset do šezdeset godina.

Istraživanjem tržišta nije pronađena slična ideja web aplikacije koja na interaktivni način predstavlja životopis kandidata. Time se otvaraju mogućnosti daljnjeg unaprjeđenja aplikacije dodavanjem opcija kreiranja zasebnog profila kandidata te proširivanjem prve skupine ciljane publike.

3.2 Dizajn aplikacije

3.2.1 Izrada modela aplikacije

Prvi korak u izradi aplikacije nakon same ideje je izrada žičanog modela (eng. *wireframe*). *Wireframe* prikazuje raspored elemenata aplikacije i njenih mogućnosti. Prve skice *wireframe-a* su napravljene na papiru. Tek nakon razvoja ideja, napravljen je *wireframe* pomoću Balsamiq Mockups 3 softvera. Ovaj softver je odabran zbog jednostavnog korisničkog sučelja i opcija koje nude praktičnu i jednostavnu izradu *wireframe-a*. Konačni izgled *wireframe-a* prikazan je na slici 15.



Slika 15. *Wireframe* aplikacije

Cijela aplikacija se sastoji od ukupno sedam različitih zaslona prikazanih *wireframe-om.* Broj zaslona zadovoljava optimalnu raspodjelu informacija, a ujedno je i dovoljno praktičan s obzirom na svrhu aplikacije.

Aplikacija je podijeljena u četiri osnovne kategorije u obliku kartica koje se prve prikazuju na zaslonu u njegovoj punoj veličini. Klikom, odnosno dodirom na kategoriju, ona se otvara te su u njoj prikazani podaci vezani uz tu kategoriju.

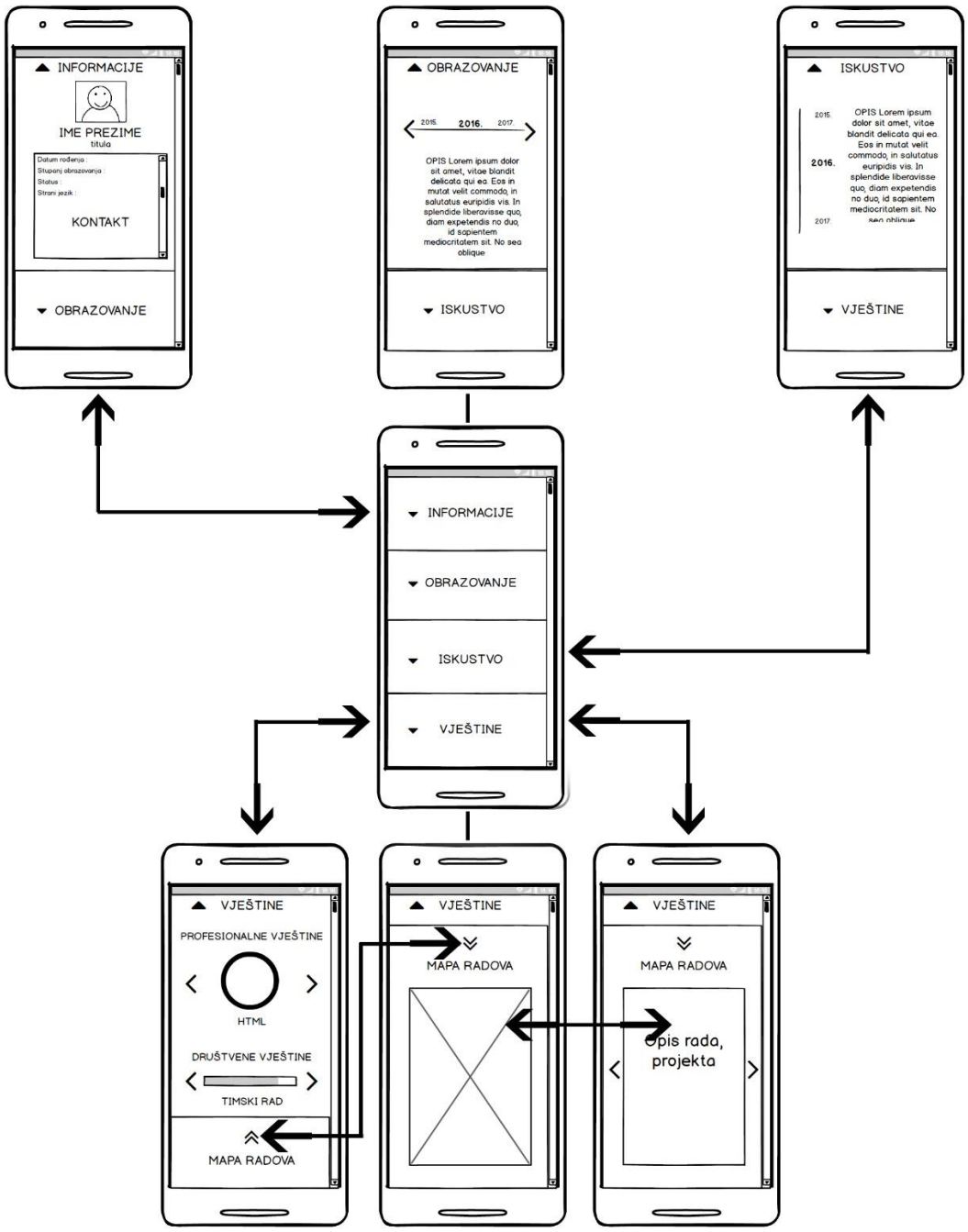
Dodirom na prvu karticu „informacije“ otvara se prva kategorija. U ovoj kategoriji nalazi se slika kandidata, njegovo ime, titula te njegovi osnovni podaci: datum rođenja, adresa, jezici koje govori i kontakt. Kontakt kandidata nalazi se ispod osnovnih podataka, a prikazuje se klizanjem kroz informacije.

Druga kategorija, „obrazovanje“ sadrži informacije o obrazovanju kandidata. Informacije su prikazane u obliku vremenske crte. Mijenjajući godine klikom, odnosno dodirom na strelice „lijevo“ i „desno“, prikazuju se ključne godine u obrazovanju kandidata kao i kratki tekst koji ih opisuje.

Treća kategorija „iskustvo“ na isti način kao i prethodna prikazuje informacije o radnom iskustvu kandidata. Ova vremenska crta je okrenuta vertikalno, a s desna se nalaze godine te tekst koji opisuje ključnu godinu.

Posljednja, odnosno četvrta kategorija „vještine“ prikazuje informacije o vještinama kandidata. Vještine su podijeljene u dvije skupine. Prva skupina sadrži profesionalne vještine u području rada i alate kojima se kandidat koristi. Razina vještina prikazana je postotkom u obliku kružnog dijagrama. Dijagram će stupnjevima prikazati postotak usvojenosti pojedinih vještina. Dodirom ili klikom na strelice „lijevo“ i „desno“ se mijenjaju prikazane vještine, a s njima i postotak usvojenosti prikazan dijagramom. Druga skupina vještina su društvene vještine. One su prikazane pomoću trake za napredak (eng. *progress bar*) koja se puni ovisno o postotku usvojenosti vještine na isti način kao i kružni dijagram. Pomoću strelica „lijevo“ i „desno“ mijenjaju se društvene vještine, a s njima i postotak usvojenosti prikazan trakom. Na samom kraju kategorije koja prikazuje vještine, nalazi se nova kartica „Mapa radova“. Klikom na nju, mapa radova se otvara preko kategorije vještina. Unutar mape radova nalaze se slike radova i projekata na kojima je kandidat imao prilike surađivati. Dodirom na sliku, ona se zatamnjuje, a na njoj se prikazuje kratki tekst koji sadrži opis rada, odnosno projekta.

Ponovnim dodirom na svaku od otvorenih kategorija, one se smanjuju u prvobitan oblik kartica. Slika 16. prikazuje slijed korištenja aplikacije opisan u prethodnom paragrafu.



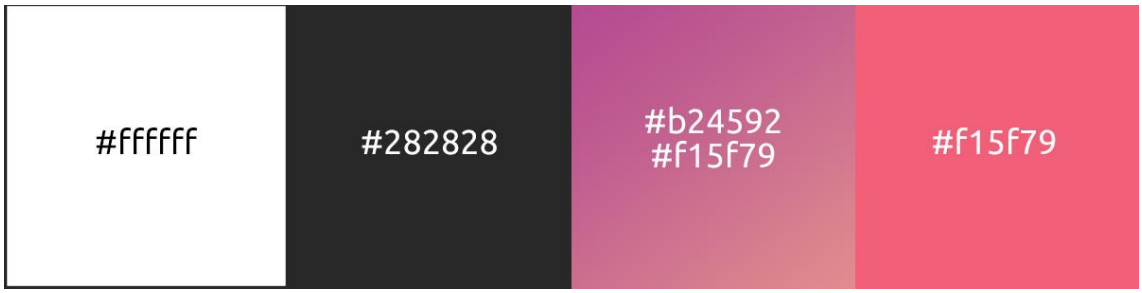
Slika 16. Hodogram aplikacije

3.2.2 Dizajn korisničkog sučelja

Kao alat za izradu dizajna korisničkog sučelja koristi se program Adobe Photoshop, verzije CC 2015. Ovaj program je odabran zbog dobrog poznavanja alata unutar njega, praktičnosti i velikog broja mogućnosti koje nudi.

Izgled korisničkog sučelja zamišljen je kao vrlo jednostavan i minimalistički prikaz informacija s naglaskom na interaktivnost same aplikacije. Odabrane boje su veoma neutralne jer se izgledom aplikacije želi postići dojam profesionalnosti te izgled stvarnog životopisa. Kao osnovne boje korištene su: bijela (#ffffff) i tamnosiva (#282828). Ostale korištene boje u izradi korisničkog sučelja aplikacije su: gradijent (#b24592, #f15f79) i ružičasta koja se pojavljuje u gradijentu kao zasebna boja (#f15f79). Na nekim elementima je korištena transparentnost kako bi se ublažio kontrast. Korištena paleta sa pripadajućim heksadecimalnim vrijednostima je prikazana na slici 17.

Bijela boja je odabrana kao pozadina svih kartica kako bi korisnika asocirala na bijeli papir, odnosno klasični prikaz životopisa. Za tekst je uglavnom korištena tamnosiva i tek ponegdje bijela. Osim kod tipografije, tamnosiva se koristi i kao boja ikona i interaktivnih elemenata. Zbog vrlo jednostavnog odabira boja pozadine i tipografije, za interaktivni prikaz informacija odabran je gradijent kojim se nastoji istaknuti informacije i dodati veću vizualnu vrijednost samoj aplikaciji. Također, gradijentom se nastoji pratiti postavljen trend u dizajnu korisničkog sučelja koji je započeo Instagram 2016. godine predstavljanjem novog loga društvene mreže.



Slika 17. Korištena paleta boja

S obzirom na postavljene ciljeve cjelokupnog izgleda aplikacije koji teže jednostavnosti, odabran je Ubuntu set fontova preuzet sa Google fonts biblioteke. Ubuntu je font bez serifa (fr. *sans-serif*) modernog izgleda. U izradi korisničkog sučelja aplikacije korištena su tri pismovna reza ovog fonta. Prvi pismovni rez je svijetli (eng. *light*) kojoj je pridodana vrijednost 300, drugi je normalni (eng. *regular*) s vrijednosti 400 te posljednji srednji (eng. *medium*) s vrijednosti 600. Korišteni font i njegovi rezovi prikazani su slikom 18.

Slika 18. Korišteni set fontova „Ubuntu“

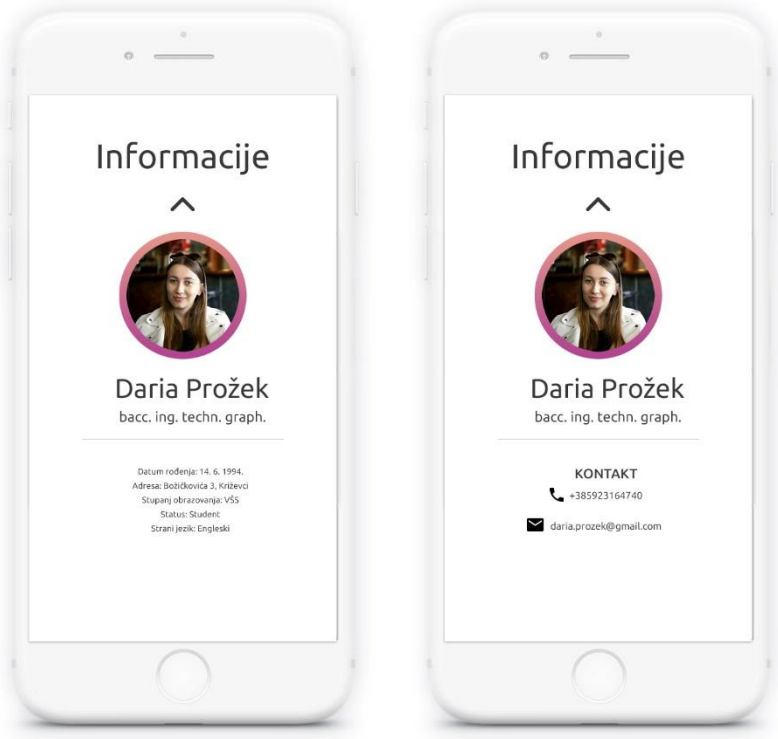
U aplikaciji se koristi nekoliko ikona za vizualno naglašavanje tekstualnih informacija. Ikone koje se koriste preuzete su iz biblioteke Ionicons. Osim vizualnog naglašavanja, neke ikone se koriste kao elementi interakcije (strelice). Korištene ikone prikazane su slikom 19.

Slika 19. Korištene „Ionicons“ ikone

Početni zaslon aplikacije koji prikazuje kartice kategorija je bijele boje. Svaka kartica na svojem donjem rubu ima tamnosivu sjenu kojom se odvaja od prethodne. Naslovi kategorija i ikone koje navode na otvaranje kartica su tamnosivi. Debljina pisma koja se koristi za naslov kartica je *light*. Za prikaz zaslona, odnosno dizajna korisničkog sučelja koristi se predložak (eng. *mockup*) preuzet sa The Mockup Club web stranice (https://themockup.club/mockups/iphone-clay-frontal-mockup/). Slika 20. prikazuje početni zaslon aplikacije, odnosno izbornik sastavljen od kartica kategorija životopisa.

Slika 20. Početni zaslon aplikacije

Otvaranjem kartica zadržava se postojeći izgled naslova svake od kategorija. Prva kategorija, 'Informacije', sadrži tipografiju u dva pismovna reza. Srednjim rezom je napisano ime kandidata i naslov 'kontakt'. Titula i informacije o kandidatu napisane su svijetlim pismovnim rezom. Slika kandidata u kružnom obliku smještena je u kružni okvir s obrubom ispunjenim gradijentom. Za označavanje kontakta korištene su već opisane ikone. Izgled prve kategorije sa svim informacijama nalazi se na slici 21.

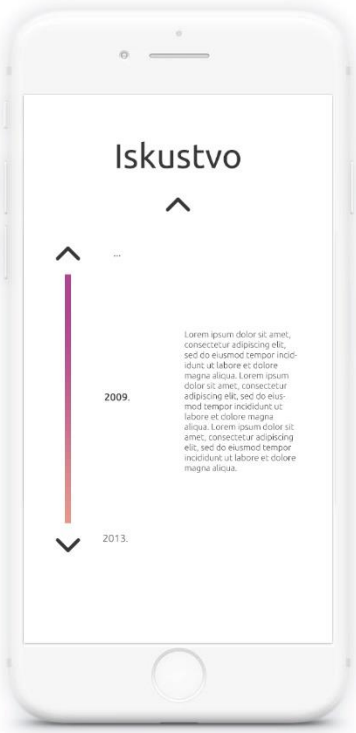


Slika 21. Izgled kategorije „Informacije“

Sljedeća kategorija, „Obrazovanje“ je oblikovana na isti način čime se postiže kontinuirani stil kroz cijelu aplikaciju. U prvom planu nalazi se vremenska crta (eng. *timeline*) na kojoj se nalaze godine. U sredini se nalazi godina koja se pregledava, a s lijeve i desne strane se nalaze godine prije i poslije. Odabrana godina istaknuta je srednjim pismovnim rezom dok su druge godine manje istaknute. Tekst ispod odabrane godine je poravnat u sredinu, logički prateći pripadajuću godinu. Sama vremenska crta ispunjena je gradijentom. Na dnu kategorije se nalazi popis napisanih radova kandidata, vizualno odvojen od vremenske crte i pripadajućeg teksta. Popis radova je naznačen odgovarajućom ikonom. Izgled kategorije 'Obrazovanje' nalazi se na slici 22.

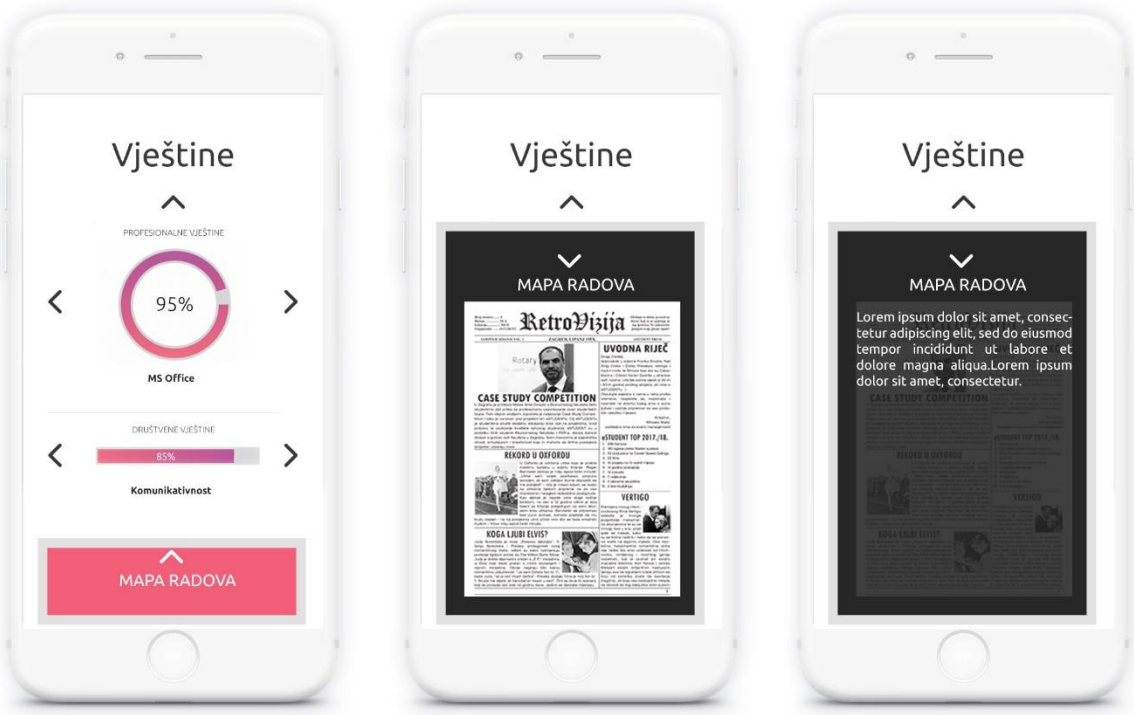
Kategorija „Iskustvo“ je uređena na isti način kao i prethodna kategorija. Tekst kategorije je poravnat ulijevo. Izgled kategorije je prikazan na slici 23.

Slika 22. Izgled kategorije „Obrazovanje“



Slika 23. Izgled kategorije „Iskustvo“

Posljednja kategorija, „Vještine“ je najkompleksnija. Sastoji se od tri dijela koja su vizualno ujednačena. Kružni dijagram na vrhu kategorije je sastavljen od dvije kružnice. Prva kružnica je transparentna, tamnosive boje. Ona označava punu vrijednost kružnog dijagrama, odnosno 100%. Kružnica koja se nalazi na njoj označava razinu svake od profesionalnih vještina. Ispunjena je gradijentom. Osim stupnjevima druge kružnice, razina vještine je označena i postotkom koji se nalazi unutar kružnica te je crne boje. Naslovi vještina su napisani svijetlim pismovnim rezom, a same vještine srednjim. *Progress bar* je od kružnog dijagrama vizualno odvojen linijom. Sastoji se od dvije linije, na isti način kao i kružni dijagram. Postotak unutar linije koja predstavlja razinu vještine je bijele boje. Ispod dijagrama nalazi se nova kartica „Mapa radova“ koja je ispunjena ružičastom, a tekst je ispisan bijelom bojom. Povlačenjem kartice prema gore, ona zauzima veći dio zaslona, postaje crna, a u njenoj sredini se prikazuje kandidatov rad. Klikom ili dodirom na rad preko njega se pojavljuje tamnosiva transparentna površina sa bijelim ispisanim tekstom koji opisuje rad. Slika 24. prikazuje izgled kategorije „Vještine“.

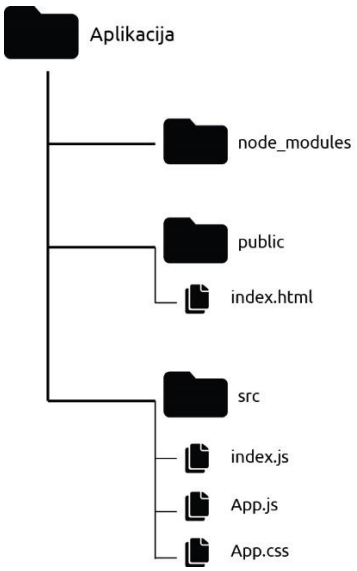


Slika 24. Izgled kategorije „Vještine“

3.3 Razvoj aplikacije

3.3.1 Postavljanje radne okoline

Kako bi bilo moguće započeti programiranje aplikacije, potrebno je pripremiti radnu okolinu i alate potrebne za njen razvoj. Prvi korak je instaliranje Node.js platforme i npm JavaScript upravitelja paketa unutar mape „Aplikacija“ u kojoj će se nalaziti datoteke aplikacije. Pomoću Node.js-a, naredbom *npm i create-react-app* kreiran je unaprijed pripremljen React.JS projekt. Struktura projekta prikazana je slikom 25.



Slika 25. Struktura mape „Aplikacija“

Unutar mape „Aplikacija“ nalaze se tri mape: src, public i node_modules. Node_modules je mapa koja sadrži sve potrebne datoteke korištenih knjižnica. Mapa src sadrži radne datoteke: App.js, indeks.js i App.css. App.js je datoteka u kojoj se razvija aplikacija, njene komponente i React elementi. App.css je datoteka koja oblikuje React elemente. U datoteci index.js nalazi se tek nekoliko linija kôda kojima se izvršava prikaz aplikacije unutar HTML *div* elementa '*root*'. Pomoću ključne riječi import uvodi se react knjižnica, react-dom i sadržaj aplikacije iz „App.js“ datoteke. Sadržaj datoteke index.js prikazan je na slici 26.

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

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

Slika 26. Sadržaj datoteke „index.js“

Datoteka index.html je predložak unutar kojeg se prikazuje sadržaj aplikacije, odnosno React elementi. Indeks.html u *<head>* oznaci sadrži *<meta name="viewport">* oznaku koja daje upute internet pregledniku o upravljanju sa veličinom sadržaja u odnosu na dimenzije prikaza. U ovoj datoteci je uveden font Ubuntu *<link>* oznakom. Na isti način su uvedene i ikone. Unutar *<body>* oznake nalazi se samo *div* element identifikatora *„root“*. Sadržaj datoteke index.html nalazi se na slici 27.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,
    initial-scale=1, shrink-to-fit=no">

    <link rel="stylesheet" href="//fonts.googleapis.com
    /css?family=Ubuntu:300,400,600,700&amp;lang=en" />

    <link href="https://unpkg.com/ionicons@4.3.0/dist/
    css/ionicons.min.css" rel="stylesheet">

    <title>Daria Prožek</title>
  </head>
  <body>
    <noscript>
    </noscript>

    <div id="root"></div>

  </body>
</html>
```

Slika 27. Sadržaj datoteke „index.html“

3.3.2 React komponente aplikacije

Daljnji razvoj aplikacije odvija se unutar „App.js“ datoteke. Kao alat za pisanje kôda koristi se Sublime Text. Zbog lakše orijentacije unutar kôda, oznake sintakse su postavljene na JavaScript (Babel). Tijekom programiranja aplikacije, ona će biti vrlo jednostavno oblikovana unutar pripadajuće CSS datoteke kako bi se napravio raspored elemenata. Tek nakon izrade svih funkcionalnosti, aplikacija će biti oblikovana prema napravljenom dizajnu korisničkog sučelja.

Na samom početku „App.js“ datoteke ubacuju se alati potrebni za izradu aplikacije: React, ReactDOM, React-Bootstrap i njegovi korišteni elementi te stil zadan u datoteci „App.css“. Sav sadržaj aplikacije koji se iscrtava unutar HTML *div* elementa dolazi iz izvezene (eng. *export*) komponente „*App“* koja je konstanta. Unutar ove konstante postavljeni su React-Bootstrap elementi čime se dobiva responzivnost cijele aplikacije te veličina prikaza u ovisnosti o veličini uređaja. Kako bi sadržaj bio prikazan od ruba do ruba zaslona, *fluid* parametru *Grid* elementa je postavljena vrijednost *true*. Osim, React-Bootstrap elemenata, u konstanti se nalazi i komponenta „*CardStack“*. Slika 28. prikazuje početni dio kôda i *App* komponentu.

```
1  import React, { Component } from 'react';
2  import ReactDOM from 'react-dom'
3  import 'bootstrap/dist/css/bootstrap.css';
4  import 'bootstrap/dist/css/bootstrap-theme.css';
5
6  import { Grid, Row, Col} from 'react-bootstrap';
7  import './App.css';
8
9
10 const App = (props) => (
11
12 <Grid fluid={true}>
13   <Row>
14     <Col xs={12} sm={12}  md={10} lg={6} xl={12} mdOffset={1} lgOffset={3}>
15
16 <CardStack/>
17
18     </Col>
19   </Row>
20 </Grid>
```

Slika 28. Komponenta *App*

Komponenta „*CardStack*“ je napisana u obliku klase. Ova komponenta služi za prikazivanje svih komponenata kategorija unutar neuređene liste s ciljem kreiranja početnog zaslona aplikacije. Ova nema zadane funkcije, već njena funkcionalnost dolazi sa ubačenim komponentama. Stil liste je definiran na dva načina, *inline* uvođenjem konstante *stackStyles* i njenih svojstava, odnosno vrijednosti te kao CSS klasa „lista“. Komponenta *CardStack* je napisana zasebno od komponente *App* zbog lakšeg održavanja i uvođenja izmjena. Na slici 29. prikazan je kôd komponente *CardStack.*

```
25  class CardStack extends Component {
26
27    render() {
28      return (
29        <ul style={stackStyles} className="lista">
30        <li> <Informacije/> </li>
31        <li> <Obrazovanje/> </li>
32        <li> <Iskustvo/> </li>
33        <li> <Vještine/> </li>
34        </ul>
35          );}
36  }
37    const stackStyles = {
38    display: 'flex',
39    flexDirection: 'column',
40    position: 'relative',
41    overflow: 'hidden',
42    padding: 0,
43    margin: 0,
44  };
```

Slika 29. Komponenta *CardStack*

Nakon što je napisan kôd komponente koja će prikazivati kategorije, potrebno je izgraditi komponente za svaku od njih. Prva komponenta je komponenta koja gradi kategoriju „Informacije“. Ona se od prethodne razlikuje po tome što ima zadano stanje (eng. *state*). Zadano stanje komponente nalazi se u metodi *constructor*. Postavljeno stanje nosi naziv *active,* a njegova vrijednost je *boolean* vrijednost *true.* Promjenom stanja će se mijenjati CSS klasa *div* elementa kartice kategorije te ovisno o promjeni, ona će se otvarati, odnosno zatvarati. Kako bi to bilo moguće, osim postavljenih CSS klasa za zatvorenu (.*Card*) i otvorenu (.*CardClicked*) karticu, potrebna je funkcija koja će mijenjati stanje, a time i klasu *div* elementa pomoću operatera uvjeta. Funkcija koja obavlja otvaranje i zatvaranje kartica je *toggleClas().* Ovom funkcijom se dohvaća trenutno stanje te se njegova vrijednost mijenja u suprotnu vrijednost, odnosno *false.* Kako bi funkcija bila pozvana, potreban je događaj (eng. *event*) koji će potaknuti njezino

izvršavanje. *Event onClick* koji poziva funkciju pridodan je *div* elementu čija je klasa pozvana iz Ionicons biblioteke ikona. Ovaj *div element* ima izgled strelice prema dolje. Dodirom ili klikom na strelicu, izvršava se funkcija te se kartica kategorije „Informacija“ otvara. *Div* element koji je imao izgled strelice prema dolje klikom na nju mijenja svoju klasu na isti način kao i kartica. Novom pridodanom klasom *div* element ima izgled strelice prema gore na čiji se klik, odnosno dodir stanje *div* elemenata vraća u početno. Na ovaj način je postignuto otvaranje svih kartica kategorija te se kôd ponavlja u svakoj od komponenata, a kôd otvaranja se nalazi na slici 30.

```
49 class Informacije extends Component {
50       constructor (props) {
51     super(props);
52     this.state = {active: true};
53   }
54 toggleClass() {
55         const currentState = this.state.active;
56         this.setState({ active: !currentState });
57     };
58
59   render() {
60     return (
61         <div className= {this.state.active ? 'Card': 'Cardclicked'}>
62             <h1 className="Card-title">Informacije</h1>
63             <div className= {this.state.active ? 'icon ion-ios-arrow-down':
64                                                  'icon ion-ios-arrow-up'}
65                 onClick={this.toggleClass.bind(this)}> </div>
```

Slika 30. Komponenta Informacije

Svi React elementi koji se pojavljuju unutar otvorene kartice mijenjaju svoje klase ovisno o zadanom stanju. Prva CSS klasa svih elemenata je klasa *hidden* u kojoj je pomoću svojstva *visibility* i vrijednosti *hidden* svaki element skriven. Izvršavanjem funkcije te otvaranjem kartica, elementi mijenjaju svoje klase te se pojavljuju unutar kartice. Prvi React element je slika kandidata koja je izvedena kao *div* element sa pozadinskom slikom. Taj *div* element se nalazi unutar *div* elementa okvira kako bi se dobio gradijentalni rub. Nakon toga slijede još tri *div* elementa: prvi sadrži naslov, odnosno ime kandidata, drugi sadrži podnaslov, odnosno titulu kandidata, a treći je okvir u kojem se nalaze informacije i kontakt kandidata u odvojenim paragrafima. U stil trećeg *div* elementa dodano je svojstvo *overflow* sa vrijednosti *auto* kako bi se kroz informacije moglo klizati umjesto da se nižu jedna ispod druge. Na slici 31. je prikazan sadržaj komponente „Informacije“.

```
66                  <div className={this.state.active ? 'hidden': 'slikaokvir'}>
67                  <div className={this.state.active ? 'hidden': 'slika'}> </div>
68                  </div>
69                  <h2 className={this.state.active ? 'hidden': 'ime'}>
70                      Daria Prožek </h2>
71                  <h3 className={this.state.active ? 'hidden': 'titula'}>
72                      bacc. ing. techn. graph. </h3>
73                  <div className={this.state.active ? 'hidden': 'info'}>
74                      <hr/>
75                      <p> Datum rođenja: 14. 6. 1994.</p>
76                      <p> Adresa: Božičkovića 3, Križevci </p>
77                      <p> Stupanj obrazovanja: VŠS</p>
78                      <p> Status: Student</p>
79                      <p> Strani jezik: Engleski </p>
80                      <hr/>
81                      <h4 className='kontakt'> KONTAKT </h4>
82                      <p className='kontakticon kontakt_icon ion-ios-call'>
83                      <p className="kontaktinfo"> +385923164740</p> </p>
84                      <p className='kontakticon kontakt_icon ion-ios-mail'>
85                      <p className="kontaktinfo"> daria.prozek@gmail.com </p></p>
86                  </div>
87          </div>
88      ); }
89  }
```

Slika 31. Komponenta Informacije

Slijedeće komponente su komponente za kategorije „Obrazovanje“ i „Iskustvo“. Obje su istog sastava, a unutar sadržaja, osim elemenata prikazanih na slici 29., sadrže po jednu komponentu, horizontalnu vremensku crtu koja se nalazi unutar kategorije „Obrazovanje“ i vertikalnu unutar kategorije „Iskustvo“. Komponenta unutar komponente „Obrazovanje“ nosi ime „Timeline“. Njeno smještanje unutar komponente obrazovanje nalazi se na slici 32. Istim načinom je smještena kategorija „TimelineVertical“ unutar komponente „Iskustvo“.

```
92  class Obrazovanje extends Component {
93    constructor (props) {
94      super(props);
95      this.state = {active: true};
96    }
97  toggleClass() {
98          const currentState = this.state.active;
99          this.setState({ active: !currentState });
100     };
101   render() {
102     return (
103         <div className= {this.state.active ? 'Card': 'Cardclicked'}      >
104               <h1 className="Card-title">Obrazovanje</h1>
105               <div className= {this.state.active ? 'icon ion-ios-arrow-down':
106                                                    'icon ion-ios-arrow-up'}
107                 onClick={this.toggleClass.bind(this)}> </div>
108               <div className={this.state.active ? 'hidden': 'timelineshow'}>
109                 <Timeline/>
110                 </div>
111             </div>
112       ); }
113 }
```

Slika 32. Komponenta Obrazovanje

Komponenta „Timeline“ ima definirana tri stanja, a to su: *activeIndex, pre* i *next.* Ova stanja definiraju koje će se godine i tekst prikazivati na vremenskoj crti. Stanje *activeIndex* definira prikaz odabrane godine koja se nalazi na sredini crte i pripadajućeg opisa. Druga stanje, *pre* i *next* definiraju prikaz godine prije i poslije, odnosno one godine na koje vodi klik ili dodir na strelice uz vremensku crtu. Godine i pripadajući tekstovi su definirani unutar *carouselSlidesData* polja. U polju se ukupno nalazi osam objekata sa svojstvima „godina“ i „tekst“. Vrijednosti svojstava su godine i kratki tekstovi koji ih opisuju. Početno stanje *activeIndex* ima vrijednost 1 jer prikazuje prvu godinu koja se u polju nalazi na drugom mjestu. Stanje *pre* ima početnu vrijednost 0 koja odgovara prvom mjestu u polju na kojem se ne nalazi godina, već tri točke koje označavaju kraj vremenske crte u jednom smjeru. Početnom stanju *next* dodijeljena je vrijednost 2 jer prikazuje godinu nakon početne, a ona se nalazi na drugom mjestu u polju. Stanja se mijenjaju klikom ili dodirom na strelice te su svakoj od strelice pridodane odgovarajuće funkcije. Lijeva strelica vodi na godine prije prikazane. Funkcija *„prev“* koja se izvršava klikom na strelicu „prije“ oduzima vrijednost trenutnim stanjima za jedno mjesto u polju. Desna strelica vodi na godinu nakon prikazane te njena pripadajuća funkcija „*next“* stanjima dodaje vrijednost za jedno mjesto u polju. Komponenta *Timeline* sa svojim zadanim stanjima i funkcijama nalazi se na slici 33.

```
195 class Timeline extends Component {
196       constructor (props) {
197     super(props);
198     this.state = {activeIndex: 1, pre:0, next:2};
199   }
200
201 prev() {
202  this.setState({ activeIndex: this.state.activeIndex - 1,
203                  pre: this.state.pre -1,
204                  next: this.state.next -1 });
205     };
206
207 next() {
208  this.setState({ activeIndex: this.state.activeIndex + 1,
209                  pre: this.state.pre +1,
210                  next: this.state.next +1});
211 };
```

Slika 33. Komponenta *Timeline*

Kako bi se godine mijenjale interakcijom korisnika s vremenskom crtom, potrebno je definirati funkciju koja će vraćati vrijednosti iz polja prema zadanim uvjetima stanja. Definirane su ukupno četiri takve funkcije, za svaku od godina te za opisni tekst. Primjer funkcije za odabranu godinu nalazi se na slici 34. S obzirom na to da se želi ograničiti listanje po vremenskoj crti od najmanje do najveće godine, stanje ne smije imati vrijednost veću od 6. Sedma vrijednost u polju su tri točke koje korisniku daju do znanja da ne postoji slijedeća godina. Ukoliko bi stanje bilo veće od 7, aplikacija ne bi funkcionirala. Funkcijom *getGodina(index)* stanje *activeIndex* može imati najveću vrijednost 6. Za sve veće vrijednosti, stanju se vraća vrijednost 6 te se uzima šesta vrijednost iz polja. Najmanja moguća vrijednost stanja *activeIndex* iznosi 1 te za sve manje vrijednosti, stanje ponovno dobiva vrijednost 1 i uzima prvu vrijednost iz polja. Za vrijednosti između njih, funkcija uzima onu vrijednost iz polja koja odgovara vrijednosti trenutnog stanja. Funkcija se poziva na mjestu sadržaja, a parametar kod njenog pozivanja je trenutno stanje sadržaja.

```
getGodina(index) {
 if (this.state.activeIndex > 6) {
  this.setState({ activeIndex: 6});
  return carouselSlidesData[6].godina;

 }
 else if (this.state.activeIndex < 1) {
  this.setState({ activeIndex: 1});
  return carouselSlidesData[1].godina;
 }

 return carouselSlidesData[index].godina;

  }
```

Slika 34. Funkcija *getGodina(index)*

Sadržaj komponente „*Timeline*“ sastoji se od paragrafa unutar kojih su napisane godine i tekst koji opisuje odabranu godinu, *div* elemenata koji predstavljaju strelice na čiji se klik ili dodir pozivaju funkcije, *div* elementa koji predstavlja vremensku crtu te od liste u kojoj su navedeni napisani radovi. Kôd sadržaja prikazan je slikom 35.

```
  render() {
    return (
    <div>
      <p className='god'> {this.getPre(this.state.pre)} </p>
      <p className='godaktivna'> {this.getGodina(this.state.activeIndex)} </p>
      <p className='god'> {this.getNext(this.state.next)} </p>
      <br/>
      <div className='ion-ios-arrow-back timelinearrow'
           onClick={this.prev.bind(this)}> </div>
      <div className='timelineline'>  </div>
      <div className='ion-ios-arrow-forward timelinearrow'
           onClick={this.next.bind(this)}> </div>
      <p className='sadrzaj'> {this.getSadrzaj(this.state.activeIndex)} </p>
      <br/>
      <hr/>
      <div className="obrazovanjeinfo">
<div className="icon ion-ios-copy"/>
<ul>
<li> Psihološki utjecaj boja na djecu- izrada slikovnice </li>
<li> 3D simulacija grafičkog proizvoda kroz HTML i CSS </li>
<li> Prilagođavanje web sadržaja različitim rezolucijama preglednika </li>
</ul>
</div>
    </div>
      ); }
}
```

Slika 35. Sadržaj komponente *Timeline*

Komponenta *TimelineVertical* koja se nalazi unutar komponente iskustvo je izvedena na isti način. Njen sadržaj je smješten unutar tablice zbog lakšeg pozicioniranja elemenata.

Posljednja komponenta u listi unutar *CardStack* komponente je komponenta Vještine. Ona je postavljena na isti način kao i prethodne tri komponente. Ova komponenta je najsloženija jer se unutar nje nalaze tri komponente koje tvore njen sadržaj, a to su: *Chart, ProgressBar* i Mapa. Komponenta *Chart* prikazuje kružni dijagram i pripadajući sadržaj koji opisuje profesionalne vještine kandidata. *ProgressBar* komponentom su prikazane društvene vještine u obliku linije napretka. Komponenta mapa prikazuje radove kandidata. Komponenta Vještine se nalazi na slici 36.

```
class Vještine extends Component {
      constructor (props) {
    super(props);
    this.state = {active: true};
  }
toggleClass() {
        const currentState = this.state.active;
        this.setState({ active: !currentState });
    };
  render() {
    return (
      <div className= {this.state.active ? 'Card': 'Cardclicked'}>
          <h1 className="Card-title">Vještine</h1>
          <div  className= {this.state.active ? 'icon ion-ios-arrow-down':
                                                'icon ion-ios-arrow-up'}
                onClick={this.toggleClass.bind(this)}> </div>
        <div className={this.state.active ? 'hidden': 'timelineshow3'} >
                  <Chart/>
                  <ProgressBar/>
                  <Mapa/>
            </div> </div>
    ); }
}
```

Slika 36. Sadržaj komponente Vještine

Chart komponenta ima zadano jedno stanje, *vjestineIndex* kojem je početna vrijednost 0. Promjenom ovog stanja, mijenjat će se vještina, postotak njene usvojenosti upisan unutar kružnice te ispunjenost dijagrama kao grafičkog prikaza postotka. Ove vrijednosti se, kao i u prethodnom primjeru uzimaju iz polja koje sadrži objekte sa svojstvima *vjestina, postotak* i *text*. Strelicama pored kružnog dijagrama se upravlja sa prikazanom vještinom. Svakoj strelici je pridodana funkcija koja povećava, odnosno smanjuje vrijednost trenutnog stanja. Funkcije za svako svojstvo objekta vraćaju stanju pripadajuću vrijednost svojstva objekta u polju. Na slici 37. se nalazi postavljena komponenta *Chart* te opisane funkcije.

```
469 class Chart extends React.Component {
470    constructor (props) {
471     super(props);
472     this.state = {vjestineIndex: 0};
473   }
474 next() {
475  this.setState({ vjestineIndex: this.state.vjestineIndex + 1});
476 };
477 prev() {
478  this.setState({ vjestineIndex: this.state.vjestineIndex - 1});
479     };
480
481 getPostotak(index) {
482  if (this.state.vjestineIndex > 7) {
483   this.setState({ vjestineIndex: 0});
484   return Vjestine[0].postotak;
485  }
486  else if (this.state.vjestineIndex < 0) {
487   this.setState({ vjestineIndex: 7});
488   return Vjestine[7].postotak;
489  }
490   return Vjestine[index].postotak;
491 }
```

Slika 37. Komponenta *Chart*

Chart komponenta se sastoji od naslova te *div* elemenata unutar kojih se nalaze kružnice izvedene kao *svg* elementi. Neka od svojstava *svg* elemenata su definirana varijablama, odnosno konstantama koje se zadaju prije samog iscrtavanja. Koriste se *circle* i *text svg* elementi te gradijent, odnosno *linearGradient* element. Gradijent se koristi za oblikovanje kružnice čiji se luk mijenja ovisno o zadanom stanju, odnosno postotku. Toj kružnici je pridodana css klasa „*animated*“ u kojoj je definirano vrijeme tranzicije iz jednog stanja u drugo. Kružnice nemaju ispunu, već samo obrub. Slika 38. prikazuje animiranu kružnicu čiji se luk mijenja prema vrijednostima iz polja te definiranim varijablama.

```
525    render() {
526      var progress = this.getPostotak(this.state.vjestineIndex);
527      const pi = 3.14159265359;
528      const r = (400 / 2);
529      const c = (2 * pi) * r;
530      const realProgress = c * progress;
531
575
576              <circle
577                className="animated"
578                cx="250"
579                cy="250"
580                r="200"
581                stroke="url(#gradient)"
582                fillOpacity="0"
583                strokeWidth={40}
584                strokeDasharray={[realProgress,  c]}
585                strokeDashoffset={c * progress}
586                strokeOpacity='0.9'
587              />
```

Slika 38. *Circle* element

ProgressBar je slijedeća komponenta koja gradi komponentu „Vještine“. Ima zadano početno stanje *progress* s vrijednosti 0. Kao i u prošlim primjerima, promjenom stanja uzima se druga vrijednost iz polja, a funkcije se izvršavaju klikom ili dodirom na strelice za upravljanje. Širina linije za napredak se smanjuje, odnosno povećava promjenom vještina. Širina je unutar varijable *progress* definirana kao funkcija čiji je parametar trenutno stanje kojoj se pridodaje znak postotka kao *string* vrijednost. Osim postotaka te širine linije, klikom na strelice mijenjaju se i vještine. Sadržaj ove komponente je naslov izveden kao paragraf, *div* elementi koji imaju ulogu strelica, *div* element klase *shell* koji je okvir same linije napretka, *div* element *bar* koji predstavlja liniju napretka te paragraf unutar kojeg se prikazuju vještine. Sadržaj komponente *ProgressBar* te varijabla kojom se definira širina se nalazi na slici 39.

Posljednja komponenta koja gradi komponentu „Vještine“ je komponenta „Mapa“. Ima zadana dva početna stanja, a to su „*active“* i *„activerad“* s početnim *boolean* vrijednostima *true*. Promjenom stanja „*active“* operaterima uvjeta se mijenja CSS klasa *div* elementa koji predstavlja mapu na dnu komponente „Vještine“. Stanje se mijenja klikom ili na strelicu prema gore kojoj je pridodana funkcija *toggleClass()*. Promjenom klase *div* element prekriva komponentu „Vještine“ te se unutar njega pojavljuje novi *div* element čija je pozadina rad kandidata. Ovaj *div* element je do trena pozivanja funkcije skriven pomoću već opisane klase *hidden*. Klikom ili dodirom na rad kandidata,

pojavljuje se tamnosivi *div* element unutar kojeg se nalazi opis rada. Slika 40. predstavlja sadržaj komponente „Mapa“.

```
673   render() {
674     var progress = {
675       width: this.getProgress(this.state.progress) + "%"
676     }
677       return (
678       <div>
679       <hr/>
680       <p className='vjestineNaslov'> DRUŠTVENE VJEŠTINE </p>
681       <div className='ion-ios-arrow-back timelinearrow3'
682             onClick={ this.prev.bind(this)} />
683       <div className='box2'>
684         <div className="shell">
685           <div className="bar" style={ progress }>
686           { this.getProgress(this.state.progress) + "%" }
687           </div>
688         </div>
689       </div>
690         <div className='ion-ios-arrow-forward timelinearrow3'
691               onClick={ this.next.bind(this) }/>
692       <p className="vjestine"> {this.getVjestina(this.state.progress)} </p>
693       </div>
694     ) }
695 }
```

Slika 39. Sadržaj komponente *ProgressBar*

```
712   render() {
713     return (
714
715     <div className= {this.state.active ? 'Map': 'Mapclicked'}>
716           <div className= {this.state.active ? 'icon ion-ios-arrow-up':
717                                                'icon ion-ios-arrow-down'}
718                onClick={this.toggleClass.bind(this)}> </div>
719           <p className="Mapa-title">MAPA RADOVA </p>
720           <div className={this.state.active ? 'hidden': 'timelineshow'}>
721
722           <div className='rad' onClick={this.toggleClassrad.bind(this)}>
723           <div className={this.state.activerad ? 'hidden': 'radclicked'}>
724           <p className="radtekst"> Izrada novina u ulozi brošure za event
725                                    otvorenja izložbe "Šezdesete"  </p>
726           </div>
727           </div>
728           </div>
729     </div>
730       ); }
```

Slika 40. Sadržaj komponente „Mapa“

3.3.3 Oblikovanje aplikacije

Tijekom procesa programiranja aplikacije, ona je oblikovana pomoću CSS-a vrlo jednostavno s ciljem postizanja funkcionalnosti. Napravljen je raspored elemenata te su izrađene klase potrebne za izvršavanje otvaranja kartica. Prvobitan izgled aplikacije pokrenute na pametnom telefonu prikazan je slikom 41.

Slika 41. Izgled aplikacije prije primjene dizajna

Sve kartice sadrže dvije klase, *„Card“* kao prvobitno stanje na početnom zaslonu aplikacije, i *„Cardclicked“* klasu kojom se kartice otvaraju. S obzirom na to da se na početnom zaslonu nalaze četiri kartice koje ukupnom visinom moraju zauzeti cijeli zaslon, visina pojedine kartice definirana unutar klase *„Card“* iznosi 25*vh*, odnosno 25 posto visine *viewporta*. Klasa *„Cardclicked“* prikazuje otvorene kartice čija visina tada iznosi 100*vh*, odnosno zauzima cijelu visinu prikaza na zaslonu. Obje klase imaju zadano svojstvo *transition* s vrijednosti od 0.5 sekunde kako bi prijelaz između klasa bio postepen. Sav tekst je poravnat u sredinu svojstvom *text-align* te mu je zadana tamnosiva boja svojstvom *color*, vrijednosti heksadecimalnog zapisa #282828. U cijeloj aplikaciji koristi se Ubuntu set fontova definiran svojstvom *font-family*, a debljina pismovnog reza, *font-weight* je podešena kao *light* s vrijednosti 300. *Box-shadow* svojstvo definira sjenu kartica. Pozadina klase *„Card“* je bijela dok je na otvorenim karticama klasom *„Cardclicked“* boja definirana kroz gradijentalni prijelaz iz bijele u svijetlosivu. Opisane klase nalaze se na slici 42.

```
.Card {
  text-align: center;
  background: white;
  padding: 20px;
  color: #383838;
  height: 25vh;
  transition: 0.5s;
  font-family: ubuntu;
  font-weight: 300;
  box-shadow: 0px 0px 5px #d7cbd8;
}
.Cardclicked {
  justify-content: center;
  text-align: center;
  background: linear-gradient(to top, #fcfcfc, #ffffff);
  padding: 20px;
  color: #383838;
  height: 100vh;
  transition: 0.5s;
  font-family: ubuntu;
  font-weight: 300;
  box-shadow: 0px 0px 5px #d7cbd8;
}
```

Slika 42. Klase *„Card“* i *„Cardclicked“*

Komponente „Obrazovanje“, „Iskustvo“ i „Vještine“ sadrže *div* elemente unutar kojih se nalazi sadržaj. Svakom od tih *div* elemenata je dodijeljena klasa *„hidden“* koja otvaranjem kartica mijenja u pripadajuće klase koje opisuju sadržaj. Primjer takve klase je klasa *„timelineshow“*. Unutar nje je definirano pozicioniranje teksta u sredinu, relativno pozicioniranje *div* elementa u odnosu na prethodne elemente s udaljenosti gornjeg ruba od 5% te je definirano svojstvo *margin* s vrijednosti *auto* čime je *div* element sa svojim sadržajem pozicioniran na sredinu *parent* elementa. Budući da se u svakoj od te tri kategorije nalaze *div* elementi u ulozi strelica, njima su pridodane dodatne klase kojima je definirana njihova pozicija, boja i način prikaza. Svojstvom *display* čija je vrijednost *inline-block* strelice su poredane u istom redu sa interaktivnim elementom kojeg opisuju. Slika 43. prikazuje klase *„timelineshow“* i *„timelinearrow“* pomoću kojih je oblikovan osnovni sadržaj komponente „Obrazovanje“. Svaka komponenta ima takve pripadajuće klase, ali se one razliku zbog različite vrste sadržaja unutar njih.

```
26  .timelineshow {
27    justify-content: center;
28    text-align: center;
29    position: relative;
30    vertical-align: middle;
31    top:5%;
32    margin:auto;
33    color: white;
34    height: 70%;
35    width: 100%;
36    }
```

```
37
38    .timelinearrow {
39     display: inline-block;
40     vertical-align: middle;
41     font-size:3em;
42     margin-left:5%;
43     margin-right:5%;
44     width:5%;
45     height: 3%;
46     color:#282828;
47  }
```

Slika 43. Klase *„timelineshow“* i *„timelinearrow“*

Vremenske crte su *div* elementi oblikovani na način da im je pridodana vrlo mala visina, odnosno širina, ovisno o njihovom položaju. Obje vremenske crte su ispunjene gradijentom koji je usmjeren ulijevo za horizontalnu vremensku crtu ili prema gore za vertikalnu. Pozicionirane su relativno, na sredini *parent* elementa. Širina horizontalne vremenske crte iznosi 70% širine elementa u kojem se nalazi. Visina vertikalne horizontalne crte je izvedena tako da zauzima 40% visine *viewporta.* Razlog drugačijem pristupu definiranja dimenzija je drugačija pozicija u odnosu na druge elemente. Vertikalna vremenska linija nalazi se unutar ćelije tablice kako bi se ispravno pozicionirala te kada bi njena visina bila izražena postotkom, zauzela bi 40% ćelije,

odnosno, bila bi premala. Definirane klase kojim su oblikovane vremenske crte nalaze se na slici 44.

```
.timelineline {
 background: linear-gradient(to left, #b24592, #f15f79);
 position: relative;
 margin:auto;
 height: 5px;
 width: 70%;
 transition: 0.5s;
 display: inline-block;
 vertical-align: middle;
}

.timelineline2 {
 background: linear-gradient(to top, #b24592, #f15f79);
 position: relative;
 height: 40vh;
 width: 5px;
 transition: 0.5s;
 vertical-align: middle;
 margin:auto;
 margin-top: 1%;
}
```

Slika 44. Klase *„timelineline“* i *„timelineline2“*

Progress bar koji se nalazi unutar kategorije „Vještine“ izveden je od dva *div* elementa. Prvom *div* elementu pridodana je klasa *shell.* On služi kao okvir interaktivnom dijelu *progress bar-a.* Boja ruba i ispune je definirana pomoću *rgba()* vrijednosti kako bi se transparentnost mogla prilagoditi neovisno jedna o drugoj te kako ne bi utjecala na *child* elemente. Tekst je poravnat u sredinu, a boja mu je definirana kao bijela. Udaljenost sadržaja ovog *div* elementa od njegovih rubova definirana je svojstvom *padding* kojem je pridodana vrijednost od 2px. Interaktivan dio linije napretka je opisan klasom *„bar“.* Pozadina ovog *div* elementa je gradijent čiji je smjer prijelaza boje ulijevo. Prozirnost je smanjena za 0.05 kako bi kontrast bio manji. Tranzicije prijelaza iz jednog stanja u drugo imaju vrijednost od jedne sekunde. Slika 45. sadrži klase koje grade liniju napretka.

```
.shell {
  float: left;
  height: 70%;
  width: 100%;
  border: 1px solid rgba(255,255,255,0.2);
  background-color: rgba(0,0,0,0.15);
  padding: 2px;
  margin:auto;
  display: inline-block;
  text-align: center;
  position: relative;
  color: white;
  }
.bar {
  background: linear-gradient(to left, #b24592, #f15f79);
  opacity: 0.95;
  transition: 1s;
}
```

Slika 45. Klase *„shell“* i *„bar“*

Mapa radova sastoji se od nekoliko *div* elemenata. Prije nego li se koristi, mapa radova opisana je klasom „Mapa“. Pozicionirana je apsolutno kako bi se smjestila na dno zaslona. Visina ovog *div* elementa iznosi 15% visine prikaza na zaslonu. Pozadinska boja je definirana kao prva boja gradijenta heksadecimalnom vrijednosti #f15f79, a boja teksta je bijela. Zadan je tamnosivi transparentni obrub debljine 4 px. U trenutku otvaranja mape, njeno stanje se mijenja i ona se opisujem novo klasom „Mapaclicked“. Pozadinska boja se mijenja u tamnosivu, a visina *div* elementa mape zauzima 100% *viewporta*. Kako bi se mapa otvorila prema gore, zadana joj je nova vrijednost *margin-top* svojstva koja je negativna. Opisane klase se nalaze na slici 46.

```
.Mapa {
  text-align: center;
  background: #f15f79;
  color: white;
  margin-top:5vw;
  height: 15vh;
  width:100%;
  transition: 0.5s;
  position: absolute;
  top:92%;
  outline: 4px solid rgba(0, 0, 0, .2);
  opacity: 0.9;
}
```

```
.Mapaclicked {
  text-align: center;
  background: #282828;
  padding: 10px;
  color: white;
  height: 100vh;
  width: 100%;
  top: 100%;
  transition: 1s;
  margin-top: -68vh;
  position: absolute;
}
```

Slika 46. Klase „Mapa“ i „Mapaclicked“

Unutar mape radova nalaze se dva *div* elementa. Prvi *div* element je element koji prikazuje rad kandidata. On je opisan klasom „rad“. Pozicioniran je na sredinu elementa u kojem se nalazi te je skriven do trenutka otvaranja mape. Pozadina *div* elementa je slika

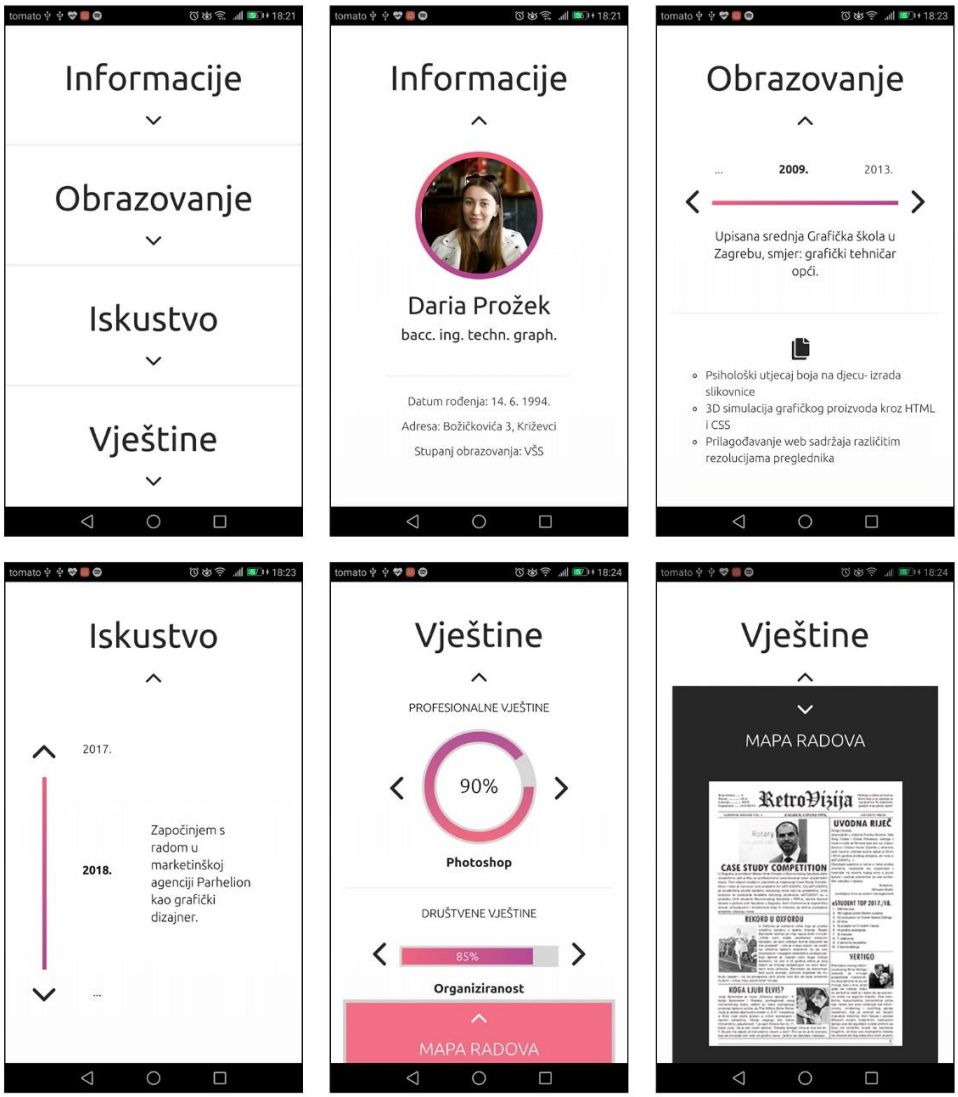
rada kandidata. Svojstvo *background-repeat* i njegova vrijednost *no-repeat* sprječavaju ponavljanje pozadine unutar *div* elementa ukoliko su dimenzije pozadinske slike manje. Pomoću svojstva *background-size* i vrijednosti *auto* definirane su dimenzije pozadinske slike tako da je cijela smještena unutar *div* elementa. Zadnje svojstvo za oblikovanje pozadinske slike je *background-position* kojem je dodijeljena vrijednost *center* kako bi se pozadinska slika smjestila na sredinu *div* elementa. Klikom ili dodirom na sliku rada kandidata prikazuje se još jedan *div* element. On se prikazuje preko rada te je definiran klasom „radclicked“. Tamnosivi je i transparentan, a unutar njega se nalazi opis rada. Boja pozadine je definirana *rgba()* vrijednostima kako bi mogla biti prozirna. Svojstvo *opacity* nije korišteno jer bi se tako transparentnost primijenila i na sve elemente unutar tog *div* elementa. Na slici 47. se nalaze klase „rad“ i „radclicked“.

```
436  .rad {
437    position: relative;
438    top:-15%;
439    margin: auto;
440    height:110%;
441    width: 100%;
442    transition: 1s;
443    background-image: url("rad1.jpg");
444    background-repeat: no-repeat;
445    background-size:auto 55vh;
446    background-position: center;
447  }
448
449  .radclicked {
450    position: relative;
451    margin: auto;
452    height:100%;
453    top:5%;
454    width: 100%;
455    background-color: rgba(0,0,0, 0.8);
456    transition: 0.5s;
457  }
```

Slika 47. Klase „rad“ i „radclicked“

4. REZULTATI I RASPRAVA

Primjenom već stečenih znanja o HTML opisnom i CSS stilskom jeziku te istraživanjem React.JS JavaScript knjižnice uspješno je razvijena aplikacija s ulogom životopisa. Kroz cijeli proces izrade koristilo se nekoliko različitih alata: Node.js, Balsamiq Mockups 3 za izradu *wireframe-a*, Adobe Photoshop za izradu dizajna aplikacije, Adobe Illustrator kao pomoćni alat i Sublime Text kao uređivač teksta za pisanje kôda aplikacije. Aplikacija je u procesu izrade i kao završni produkt rada testirana preko *localhost-a* na Huawei P9 Lite pametnom telefonu. Rezultati rada, odnosno sama aplikacija je prikazana na slici 48.



Slika 48. Izgled aplikacije na pametnom telefonu

Aplikacija može biti vrlo praktičan i jednostavan način predstavljanja kandidata poslodavcu. S obzirom na to da je riječ o web aplikaciji, ona može biti otvorena na svim uređajima bez obzira na njihov operativni sustav. Kako je riječ o specifičnoj aplikaciji koja predstavlja jednu osobu, instalacija ovakve aplikacije nije poželjna. Ovakva web aplikacija se jednostavno može podijeliti putem web adrese. Informacije mogu sadržavati i dodatne web adrese na kojima se mogu vidjeti radovi kandidata.

Prilikom istraživanja tržišta nije pronađena slična aplikacija u komercijalnom izdanju što ostavlja prostor za nadogradnju i daljnji razvoj trenutne aplikacije. Aplikacija bi se mogla proširiti tako da kandidati kreiraju svoj profil i izrade svoj životopis u obliku web aplikacije na način koji to žele. Dodavanjem informacija, vremenskih linija ili novih kategorija unutar aplikacije, ona može na vrlo zanimljiv i interaktivan način detaljno prezentirati kandidata. Također, mogla bi se dodati opcija izvoza informacija kao .pdf datoteke čime bi se, osim web aplikacije, životopis mogao kreirati i u klasičnom obliku.

5. ZAKLJUČAK

Izrađenom funkcionalnom aplikacijom odgovara se na prvu hipotezu ovog rada, odnosno na mogućnost stvaranja web aplikacije pomoću React.JS tehnologije. React.JS je kao JavaScript knjižnica uvelike olakšao izradu aplikacije. Korištenjem JSX sintakse bilo je moguće vizualizirati React elemente tokom njihove izgradnje. Odvajanjem funkcionalnosti aplikacije u vidu React komponenti pojednostavilo je uporabu, pregledavanje i ispravljanje kôda. Uporabom React.JS knjižnice je stvorena mogućnost izgradnje životopisa kao *single page* aplikacije čime je postignut brži rad i neometano korištenje iste.

Stvaranjem novih knjižnica koje olakšavaju ili obogaćuju React.JS, ova tehnologija će imati sve veću primjenu u izradi web sadržaja. Zbog svoje jednostavnosti korištenja i izgradnje interaktivnih korisničkih sučelja, sve više klijenata svoje web stranice, odnosno aplikacije prezentira napisane pomoću React.JS-a. Osim što smanjuje vrijeme izrade web sadržaja, ova JavaScript knjižnica smanjuje i trošak izrade i održavanja web stranica ili aplikacija.

Druga hipoteza postavljena u radu je omogućavanje dinamičnijeg prikaza sadržaja interaktivnošću aplikacije. Na hipotezu je odgovoreno izgradnjom interaktivnih elemenata aplikacije koji pregled informacija o kandidatu čini zanimljivijim, a samim time ističu kandidata.

Web stranice i aplikacije su postale nezamislive bez interaktivnog sadržaja. Uključivanjem korisnika u upravljanje sadržajem njegova je pažnja bolje usmjerena na informacije koje prima, ali i na samu stranicu ili aplikaciju. Što je neka stranica ili aplikacija korisnicima zanimljivija, bit će i bolje prihvaćena na tržištu.

6. POPIS SLIKA

7. LITERATURA

1. https://whatis.techtarget.com/definition/mobile-app - Mobile App, 1. 6. 2018.
2. https://www.theguardian.com/media-network/2015/feb/13/history-mobile-apps-future-interactive-timeline - 1983 to today: a history of mobile apps, 15. 6. 2018.
3. https://www.lifewire.com/what-is-a-mobile-application-2373354 - What is a Mobile Application?, 15. 6. 2018.
4. Stapić Z., Švogor I., Fodrek D. (2016.) Razvoj mobilnih aplikacija, Sveučilište u Zagrebu, Fakultet organizacije i informatike, Varaždin
5. G. R. Adesina (2014). *Mobile Operating Systems and Application Development Platforms: A Survey*, dostupno na: https://www.researchgate.net/profile/Ganiyu_Adesina, 11. 7. 2018.
6. https://study.com/academy/lesson/what-is-a-mobile-operating-system-features-types.html - What Is A Mobile Operating System, 14. 7. 2018.
7. Tutorialspoint (2014). *Android Application development*, dostupno na: https://www.tutorialspoint.com/android/android_tutorial.pdf, 16. 7. 2018.
8. T. Beaton (2018). *Introduction to iOS Development*, dostupno na: https://cdn-learn.adafruit.com/downloads/pdf/introduction-to-ios-development.pdf, 16. 7. 2018.
9. IBM Corporation (2012.) *Native, web or hybrid mobile-app development,* IBM Corporation, New York
10. https://clearbridgemobile.com/mobile-app-development-native-vs-web-vs-hybrid/ - A Guide to Mobile App Development: Web vs. Native vs. Hybrid, 18. 7. 2018.
11. http://www.itpro.co.uk/operating-systems/30248/what-is-a-graphical-user-interface - What is a graphical user interface? 23. 7. 2018.
12. https://uxdesign.cc/9-essential-ui-design-trends-you-should-stick-to-in-2018-26f33d1fe980 - The Difference Between UX & UI Design, 25. 7. 2018.
13. M. Treder (2013). UX Design For Startups, UXPin, 1. 8. 2018.
14. http://www.alphadevx.com/a/7-The-Basics-of-Web-Technologies - The Basics of Web Technologies, 11. 8. 2018.
15. Duckett J. (2011.) *HTML & CSS design and build websites,* Indianapolis: John Willey & Sons, Inc
16. Cimo F. (2015). *Bootstrap Programming Cookbook*; Web Code Geeks
17. Duckett J. (2014.) *JavaScript & jQuery,* Indianapolis: John Willey & Sons, Inc
18. Fedosejev A. (2015). *React.JS Essentials*; Birmingham, UK; Packt Publishing Ltd.