Dizajn i razvoj osobnog web sjedišta primjenom modernih web tehnologija i alata

Brakus, Bruna

Undergraduate thesis / Završni rad

2018

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University North / Sveučilište Sjever**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:122:396866

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-12-20**

Repository / Repozitorij:

University North Digital Repository

zir.nsk.hr

DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

Sveučilište
Sjever

Završni rad br. 602/MM/2018

Dizajn i razvoj osobnog web sjedišta primjenom modernih web tehnologija i alata

Bruna Brakus, 0321/336

Varaždin, listopad 2018. godine

Sveučilište
Sjever

Odjel za Multimediju, oblikovanje i primjenu

Završni rad br. 602/MM/2018

Dizajn i razvoj osobnog web sjedišta primjenom modernih web tehnologija i alata

Student

Bruna Brakus, 0321/336

Mentor

Dr.sc. Mario Tomiša, izvanredni profesor

Sažetak

Moderna programska rješenja sve češće se razvijaju u obliku *web* aplikacija, a rjeđe kao tzv. nativne aplikacije (engl. native apps) namijenjene određenom operacijskom sustavu. Razlog tome ogleda se u dostupnosti *web* aplikacija izravno kroz *web* preglednik, pri čemu su jedini ograničavajući čimbenici brzina internet veze, *hardverske* performanse uređaja i mogućnosti *web* preglednika. Kako bi se eliminirala navedena ograničenja razvijena je specifikacija progresivnih *web* aplikacija (PWA), te sukladno s njom i tehnologije koje omogućuju realizaciju. Najvažnije karakteristike progresivnih *web* aplikacija ogledaju se u responzivnosti korisničkog sučelja i raznim tehnikama realizacije korisničkog iskustva u *web* okruženju kako bi bilo što sličnije onome pri korištenju nativne aplikacije, s velikim naglaskom na performansama.

Završni rad objašnjava metodologiju responzivnog *web* dizajna i koncept progresivnih *web* aplikacija te kritički razmatra razloge njihove primjene u modernom *web* dizajnu. U kontekstu promatranih tema, opisane su tehnologije i alati koji omogućuju njihovu praktičnu realizaciju. Preact je JavaScript radni okvir koji se koristi za realizaciju dijelova korisničkog sučelja konceptom komponenti. SCSS *preprocessor* omogućava modularizaciju CSS-a kakvu inače CSS ne podržava. Webpack *bundler* se koristi za automatizaciju povezivanja izvornog koda, prevođenje izvornog koda koji koristi Preact u nativni JavaScript kod podržan od strane pretraživača, te optimizaciju veličine sadržaja.

U praktičnom dijelu završnog rada prikazan je proces dizajniranja te praktične realizacije osobnog *web* sjedišta koje zadovoljava svojstva responzivnosti, brzog učitavanja sadržaja i brzog prividnog učitavanja medijskih datoteka, koje su navedene PWA specifikacije. Koristeći Figma *web* aplikaciju izrađeno je korisničko sučelje, a opisanim tehnologijama i alatima realizirano je responzivno *web* sjedište visokih performansi u smislu brzog učitavanja klijentskog sadržaja, te brzog odgovora na interakciju korisnika sa sučeljem.

Ključne riječi: *web*, PWA, HTML, SCSS, JavaScript, Preact, Webpack, responzivnost, osobno *web* sjedište

Abstract

Modern software solutions are more often being developed in the form of web applications, and less frequently as native applications intended for a particular operating system. The reason for this is reflected in the accessibility of web applications directly through the web browser, with only limits being internet connection speed, hardware performance of devices, and browser capabilities. In order to eliminate these limitations, the specification of Progressive Web Applications (PWA) has been developed, as well as the technologies that enable it's implementation.

The most important features of progressive web applications are reflected in the responsiveness of the user interface and the various web experience realization techniques to be more similar to those used in native application, with a great focus on performance.

This work explains the methodology of responsive web design and the concept of progressive web applications. It critically considers the reasons for their application in modern web design. It also describes tools and technologies (SCSS, Preact, Webpack) which enable the practical implementation of these concepts.

Practical part of the work shows the process of designing and implementing a personal web site which satisfies the characteristics of the PWA specification - responsiveness, fast content loading and fast perceived loading of media files. User interface was realised using the Figma web application. Described technologies and tools have been used to implement the responsive and high performance web site.

Keywords: web, PWA, HTML, CSS, JavaScript, Preact, Webpack, responsiveness, web portfolio

Popis korištenih kratica

HTML	HyperText Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
AJAX	Asynchronous JavaScript and XML
XHR	XMLHttpRequest
PWA	Progressive web application
SASS	Syntactically awesome style sheets
3G	Mreža treće generacije mobilne telefonije
NPM	Node package manager
JSON	JavaScript Object Notation

Sadržaj

1. Uvod 1
2. Web aplikacije 2
   - 2.1. Pretraživač kao platforma izvršavanja 2
   - 2.2. Single-Page aplikacije 3
   - 2.3. Progresivne web aplikacije 5
   - 2.4. Prednosti PWA u odnosu na nativne aplikacije 7
3. Razlozi za realizaciju osobnog *web* sjedišta kao PWA 8
   - 3.1. Dokaz koncepta 8
   - 3.2. Obavezni zahtjevi osobnog web sjedišta 8
   - 3.3. Neobavezni zahtjevi osobnog web sjedišta 9
4. Korištene tehnologije 10
   - 4.1. Preact radni okvir 10
   - 4.2. Webpack JavaScript modul 11
   - 4.3. Babel prevoditelj 12
   - 4.4. Syntactically awesome style sheets preprocessor 13
   - 4.5. Git distibutivni sustav 13
5. Postavljanje aplikacije 14
6. Arhitektura 17
   - 6.1. Arhitektura agnostična prema platformi (Platform agnostic) 17
   - 6.2. Jedinstvena točka ulaza 18
   - 6.3. Upravljanje stanjem web sjedišta 18
7. Struktura 20
8. Izrada web sjedišta 21
   - 8.1. Izrada skica (Wireframes) 21
   - 8.2. Dizajn specifikacije 25
   - 8.3. Karta web sjedišta 28
9. Realizacija dizajna pomoću komponenti 29
   - 9.1. „App“ komponenta 29
   - 9.2. „Contact“ komponenta 29

9.3. „ProgressiveImage“ komponenta 30
9.4. Ostale komponente i realizacija dizajna 31
10. Optimizacije veličine, brzine učitavanja i prikaza klijentskog sadržaja 32
10.1. Kritični CSS kod 32
10.2. Minifikacija koda 32
10.3. Učitavanje slika 32
11. Zaključak 34
12. Literatura 35
13. Popis slika ..... Error! Bookmark not defined.

1. Uvod

U početku razvoja računala i programa za računala rješenja su bila namijenjena specifičnim problemima. Zbog toga nije postojala potreba da pokretanje i izvršavanje programa bude podržano na više različitih uređaja. S pojavom računala opće namjene i rastućom distribucijom istih ta potreba se promijenila, no probleme različitog *hardwarea* rješavali su operativni sustavi koji su generalizirali set naredbi koji se može izvršavati na različitim uređajima. Tako su se proizvodile aplikacije za razne operativne sustave poput Mac OS-a, Windows-a ili raznih distribucija Linux-a. Svakako, napisane aplikacije bile su namijenjene isključivom pokretanju na specifičnom operativnom sustavu. Kako bi se podržalo više operativnih sustava pisane su aplikacije za svaki operativni sustav posebno. Takve aplikacije nazivaju se nativnim aplikacijama.

Kod pametnih telefona problem podrške pokretanja i izvršavanja aplikacija na većem broju uređaja bio je još veći jer je različitost *hardwarea* bila veća, kao i broj operativnih sustava. Razni pristupi su vodili nastanku višeplatformskih rješenja u obliku hibridnih aplikacija. Nastali su radni okviri i platforme koji ujedinjuju set naredbi u jednu naredbu, te ga potom prevode u naredbe specifične za pojedine sustave, poput Ionic-a, PhoneGap-a i React Native-a.

Često je set naredbi koji pruža radni okvir definiran programskim jezicima nastalim u svrhu programiranja za *web* - poput JavaScripta, Cascading Style Sheetsa (CSS) i HyperText Markup Languagea (HTML). Razlog tome je rastuća popularnost programiranja za *web*, te pristup realizaciji korisničkih sučelja. Među ostalim, kao rješenje prepoznate su i same *web* aplikacije koje su već odgovarale potrebama podrške različitih okruženja.

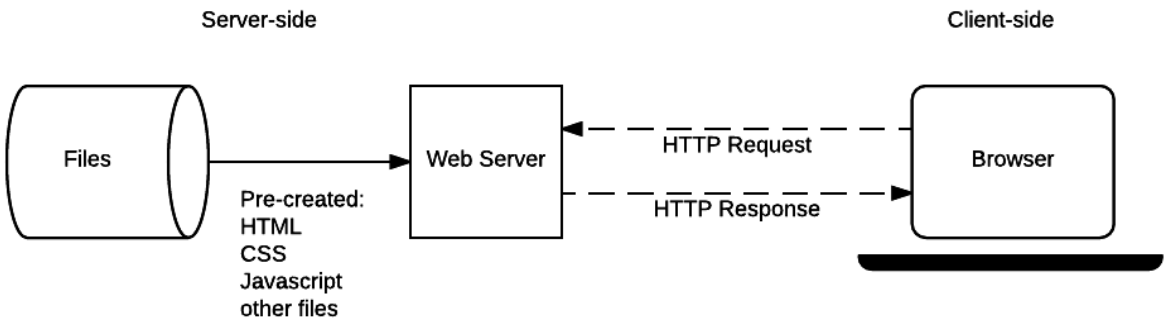
Web aplikacije često slijede upravo suprotnu logiku od nativnih pa su rađene s ciljem podrške za više platformi izvršavanja. Platforma izvršavanja je u ovom slučaju *web* pretraživač koji se može pronaći na većini modernih uređaja.

2. Web aplikacije

2.1. Pretraživač kao platforma izvršavanja

Klijentski dio *web* aplikacije obično se realizira u obliku *web* dokumenata pisanog pomoću HTML, CSS i JavaScript programskih jezika. Navedeni jezici definirani su specifikacijom koja govori o načinu njihove primjene, ali ne i o načinu izvršavanja. Tako različiti pretraživači realiziraju specifikacije na vlastiti način, što programeru omogućuje da razvija *web* aplikaciju u standardnim jezicima. Rezultat je podrška izvršavanju jedinstvene *web* aplikacije na različitim pretraživačima. [1]

Uz *web* sjedišta statičkog i dinamičkog sadržaja, danas imamo sve veću zastupljenost *web* aplikacija. *Web* sjedišta, kao i *web* aplikacije, mogu se realizirati klijent - server arhitekturom. Na slici ispod prikazana je klijent - server arhitektura koja dijeli zadaće u aplikaciji na poslužitelja usluga - server, te korisnika usluga - klijenta. [2]



Slika 2.1. Klijent-server arhitektura [3]

Korisnik koristeći pretraživač zatražuje resurse s određene *web* adrese ili Uniform Resource Locator-a (URL-a). Pristupanjem adresi korisnik preuzima klijentski sadržaj pomoću The Hypertext Transfer Protocol-a (HTTP-a), koji se potom interpretira i prikazuje unutar pretraživača. [4]

U početku raširenog korištenja, *web* sjedišta su bila znatno primitivnija od nativnih aplikacija. Za razliku od dinamičkih nativnih aplikacija, *web* sjedišta su zahtijevala preuzimanje i izvršavanje potpuno novog sadržaja na svaku bitnu promjenu ili interakciju korisnika.

Do prve bitne promjene došlo je s pojavom JS-a koji je omogućio dinamičke promjene korisničkog sučelja bez da se preuzima novi sadržaj sa servera, poput skrivanja i prikazivanja elemenata, te validacije unesenih podataka od strane korisnika.

Revoluciju je izazvao nastanak Asynchronous JavaScript and XML (AJAX) modela i korištenje XMLHttpRequest (XHR) objekta. XHR objekt omogućuje preuzimanje željenih sadržaja s *web* poslužitelja, bez potrebe za osvježavanjem prikaza sučelja *web* stranice u *web* pregledniku korisnika, pri izvršavanju zahtjeva. Koristeći XHR objekt, AJAX model definira način dohvaćanja sadržaja sa servera, te selektivnog osvježavanja samo određenih dijelova korisničkog sučelja. Sadržaj *web* sjedišta postaje dinamičan. [5]

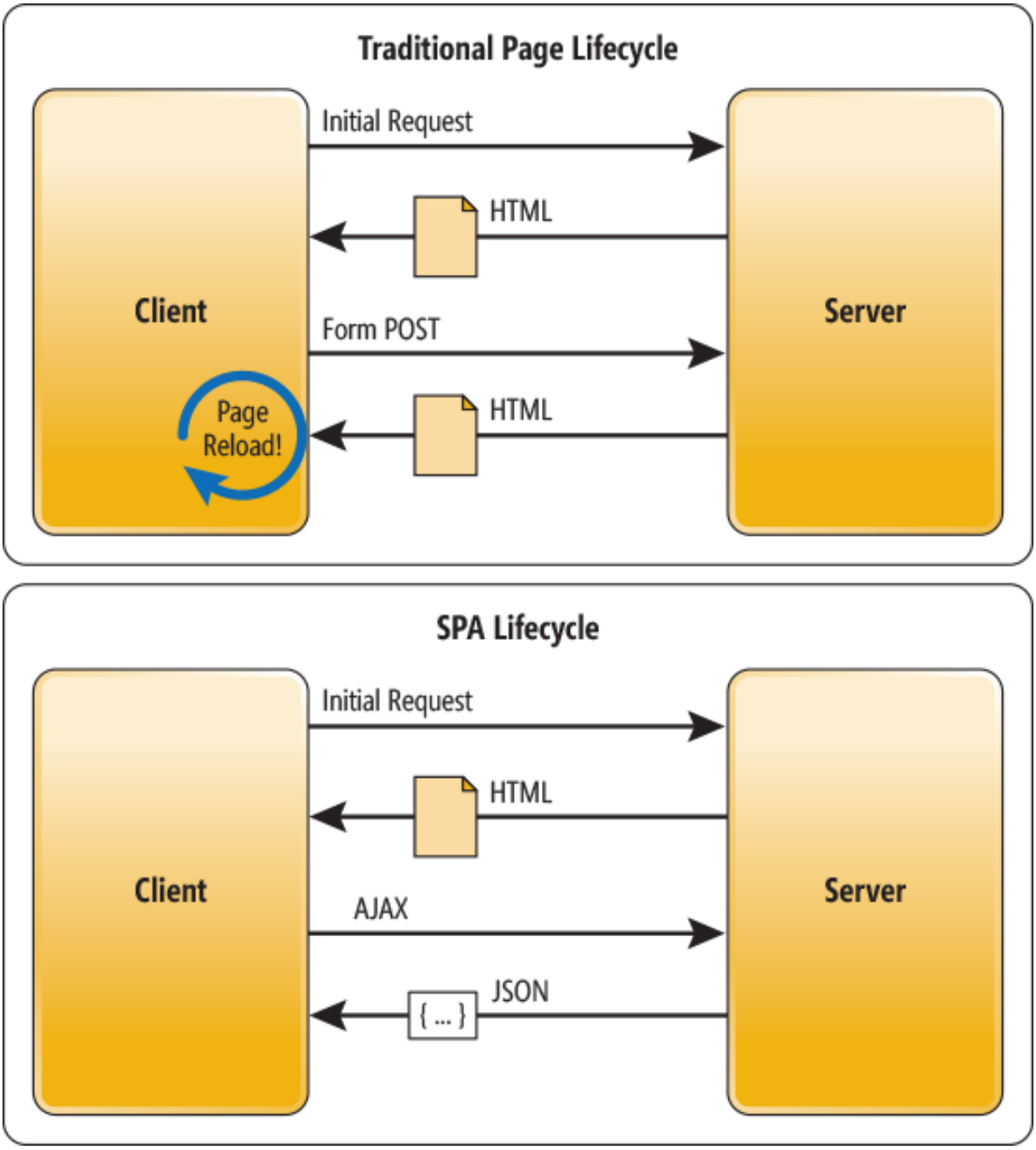
2.2. Single-Page aplikacije

Unatoč dinamičkim *web* aplikacijama, korisnički doživljaj i dalje nije bio brz i nesmetan kao kod nativnih aplikacija.

Organizacija sadržaja bila je takva da je dio sadržaja bio dinamičan, ali kod većih promjena bi se i dalje radilo potpuno osvježavanje korisničkog sučelja.

Nastaje koncept *single-page* aplikacija. Odbacuje se ideja učitavanja potpuno novog sadržaja i cjelovitog osvježavanja korisničkog sučelja. Jednom kada se preuzme klijentski sadržaj aplikacija odgovara na interakciju korisnika dinamički te mijenja samo što je potrebno. [6]

Podaci se dohvaćaju dinamički sa servera pomoću AJAX-a te se na osnovu njih izmjenjuje samo potrebni dio sadržaja. Tako su izbjegnuti prekidi interakcije korisnika sa *web* sučeljem i približava se korisnički doživljaj nativnih aplikacija.



Slika 2.2: Ciklus klasične stranice i SPA [7]

Rezultat nastanka *single-page* aplikacija su trenutno najpopularniji radni okviri za razvoj korisničkih sučelja poput React-a, Vue-a i AngularJS-a.

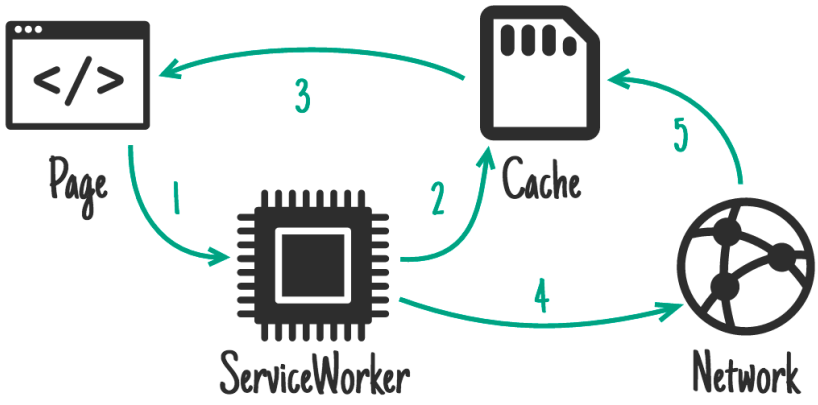
2.3. Progresivne web aplikacije

S razvojem *single-page* aplikacija najveća preostala razlika između nativnih i *web* aplikacija bila je ovisnost rada o Internet vezi. Većina *single-page* aplikacija pružala je loše korisničko iskustvo zbog sporog učitavanja sadržaja pri slaboj internet vezi. Kada korisnikov uređaj nije spojen na internet učitavanje sadržaja postizalo se učitavanjem zadnje dohvaćene verzije iz priručne memorije[1] ili se sadržaj ne bi učitavao.

U cilju razvoja *web* aplikacija koje su pouzdane, brze i zanimljive za korištenje, Google je razvio specifikaciju pod nazivom „Progresivna *web* aplikacija“ ili skraćeno PWA. Zadovoljavanjem niza zahtjeva postiže se aplikacija koja se učitava trenutačno na zahtjev korisnika unatoč internetskoj vezi. Na korisničku interakciju odgovara brzo i time pruža tečno korisničko iskustvo.

Početak razvoja specifikacije *service worker-a* 2014. i 2015. omogućilo je bolje upravljanje dohvaćanja sadržaja ovisno o stanju internet veze. *Service worker* je program koji djeluje između *web* aplikacije, pretraživača i veze na internet, upravljajući navigacijom te dohvaćanjem i *cache-iranjem* sadržaja kako je prikazano na slici ispod. [8]

Service worker se preuzima kada korisnik prvi put pristupi *web* sjedištu kontroliranom od strane *service worker-a*, te sljedeća svaka dvadeset četiri sata. Potom se instalira ako pretraživač *Byte-Wise*[2] usporedbom ne ustanovi da je identičan *service worker* već instaliran. Ako već ne djeluje aktivni stari *service worker*, aktivira se novi. U suprotnom se aktivira tek kada u potpunosti prestane djelovati stari *service worker*. [9]



[1] Memorija koja služi za pohranu često korištenih podataka (Cache).
[2] Usporedba po bitu

Slika 2.3. Cache first način rada service workera [10]

Service worker je ključan kod rada PWA aplikacije u slučaju slabe veze, te bez veze na internet. Zahtjeva se postavljanje *cache first* načina rada *service worker-a* kako bi se najbrže dohvatio sadržaj za što bolje iskustvo korištenja. U slučaju rada bez veze na internet, sadržaj iz *cache-a* je i dalje moguće dohvatiti, pa se sadržaj razvija na način kako bi se moglo i dalje koristi aplikaciju na osnovu *cache-iranog* sadržaja. Uz to se zahtjeva i prikazivanje obavijesti korisniku da je u izvanmrežnom načinu rada, ali i da svi URL-ovi prikazuju sadržaj. [4]

Među zahtjevima, bitno istaknuti i podršku za sve često korištene pretraživače poput Firefox-a, Chrome-a i Safari-a, podršku rada za različite veličine ekrana za uređaje poput mobitela i tableta kroz svojstvo responzivnosti, te posluživanje sadržaja sigurnim HTTPS protokolom. Sadržaj se mora učitavati brzo i kod veze na mrežu treće generacije (3G), gdje se mjeri vrijeme do prve moguće interakcije korisnika sa sadržajem. Aplikacija može obavještavati korisnika putem nativnih *push* obavijesti.

S obzirom na to da SPA dinamički mijenjaju sadržaj, najčešće pomoću internog *routera* realiziranog unutar same aplikacije, javlja se problem praćenja povijesti i navigacije pomoću pretraživača.

Kako bi se taj problem izbjegao PWA zahtijevaju korištenje novog *history* API-a. Moderne realizacije *router-a* u JS-u najčešće i koriste navedeni API koji osvježava povijest u pretraživaču.

U odnosu na *web* sjedišta statičkog sadržaja, veliki nedostatak SPA i PWA je nemogućnost indeksiranja sadržaja za tražilice. Za razliku od statičkog sadržaja, *web crawleri*[3] ne vide dinamički dohvaćen sadržaj SPA i PWA.

Iako *web crawleri* napreduju iz dana u dan u skladu s arhitekturama *web* aplikacija, ovom problemu se već sada može doskočiti pomoću tehnike zvane *prerendering*. *Prerendering* omogućava iscrtavanje dinamičkog sadržaja unaprijed, prije samog pristupanja, upravo kako bi bio vidljiv *web crawlerima* i kako bi ga mogli indeksirati.

Konačno, PWA omogućuju instalaciju, tj. dodavanje kratice za pokretanje aplikacije na početnom zaslonu. Mogućnost je postignuta posebnim formatima meta podataka o aplikaciji,

[3] *Web crawler* – program koji pretražuje *web* sjedišta čitajući njihov sadržaj kako bi ga indeksirao za tražilice

prvenstveno datotekom zvanom „*manifest*“. Tako je korisnicima uvelike olakšano pristupanje aplikaciji, te je korisničko iskustvo bliže onom nativnih aplikacija. [11]

2.4. Prednosti PWA u odnosu na nativne aplikacije

PWA je koncept koji je još u razvoju. Unatoč tome prednosti PWA u odnosu na nativne aplikacije su već sada očite. S gledišta osobe koja razvija aplikaciju, PWA olakšava razvoj i održavanje. PWA je jedinstveno rješenje za mnoštvo platformi i uređaja. Moguće ih je pokrenuti na pametnim mobitelima, tabletima, računalima i pametnim satovima, zbog čega nema potrebe za razvojem više aplikacija za različite platforme izvršavanja.

PWA kao *web* aplikacija objedinjuje tehnologije potrebne za razvoj u smislu standardnih *web* tehnologija poput HTML5, CSS3 i JS-a. Standardizirani alati i tehnologije omogućuju programeru da se fokusira na apstraktne probleme iz poslovne domene i arhitekture aplikacije.

Responzivni dizajn pojednostavljuje izvorni kôd aplikacije, jer se istim kôdom postiže ispravno prikazivanje na različitim veličinama ekrana i uređajima. Korisnik istoj aplikaciji može pristupiti s mobilnog uređaja i računala što mu omogućuje da na jednostavniji način koristi aplikaciju u različitim prilikama i okolnostima. [4]

Za razliku od nativnih aplikacija koje je potrebno optimizirati za predstavljanje u trgovinama aplikacija, PWA je potrebno samo predstaviti na webu pa ih je stoga i jeftinije plasirati na tržište.

Korisnicima je lakše pronaći same aplikacije i pristupiti im. PWA dio su *web* svijeta, pa se kratkom pretragom može doći do njih, posredstvom *web* tražilica. Korisnik ne mora proći kroz proces instalacije već pristupom *web* sjedištu može početi koristiti aplikaciju. [4]

3. Razlozi za realizaciju osobnog *web* sjedišta kao PWA

3.1. Dokaz koncepta

Današnji svijet *softwarea* iznimno je dinamičan. Dolaze nove tehnologije, stare se u potpunosti mijenjaju. Neke zažive i pokažu se velikim uspjehom, a neke se odbace. Promjene u tehnologiji vođene su napretkom s ciljem kraćeg razvoja sustava, razvoja kvalitetnijeg sustava, te lakšeg i uspješnijeg održavanja i nadograđivanja. Kada se tehnologije ne pokažu korisnim, budu odbačene.

Takve tehnologije mogu izazvati velike probleme onima koji ih koriste za razvoj aplikacija koje skaliraju, stvarajući tehnološki dug koji može biti čak toliko velik da se odbaci cijeli sustav ili dio sustava, te započne izrada od početka. Stoga kod izrade sustava treba biti jako oprezan pri odabiru tehnologija provjerom stabilnosti, održavanja, prihvaćenosti te drugim faktorima, kao i primjeni unutar sustava.

O uspjehu tehnologije najčešće govori prihvaćenost od strane zajednice kojoj je tehnologija namijenjena, uzimajući u obzir starost tehnologije, kao i značajnost promjena koje unosi. Pored toga, korisno je isprobati tehnologiju kako bi se upoznali s prednostima i manama, te u konačnici saznali rješava li naš problem i pod koju cijenu.

PWA je nov koncept koji se realizira pomoću novih tehnologija. Unatoč tome, zajednica je iznimno prihvatila ovaj koncept i trenutno svi popularni pretraživači rade na podršci tehnologija kojima se realizira PWA.

Rad realizira koncepte i zahtjeve PWA specifikacija poput responzivnog dizajna, brzog učitavanja i generalno visokih performansi korisničkog sučelja, fluidnog korisničkog iskustva, posluživanja putem HTTPS protokola, te pristupa sadržaju neovisno o stanju veze na internet.

3.2. Obavezni zahtjevi osobnog web sjedišta

S obzirom na svrhu osobnog *web* sjedišta, neke zahtjeve možemo opisati kao obavezne jer su kritični za njegov uspjeh. Takvi zahtjevi direktno utječu na promet korisnika i njihovo ponašanje.

Responzivnost sadržaja zahtjev je koji se odnosi na prilagodbu prikaza sadržaja različitim veličinama ekrana. Responzivnost je postignuta korištenjem responzivnog dizajna i *mobile-first* pristupa. Prikaz sadržaja dizajniran je primarno za mobilne uređaje s malim ekranima, te

zatim prilagođen većim ekranima. Kako bi razvoj i održavanje bilo što lakše koriste se najnovija svojstvima CSS3 poput CSS *grid-a* i *flexbox-a*.

Podrška za popularne pretraživače postignuta je korištenjem osobina HTML5, CSS3 i JS-a koje su podržane u navedenim pretraživačima, ili uz korištenje *prefix-a* i *polyfill-a* kojima autori pretraživača omogućuju korištenje osobina koje još nisu u potpunosti podržane. U praksi se najčešće koriste razvojni alati koji osiguravaju podršku kroz automatizaciju.

Responzivnost sadržaja kao i podrška za popularne pretraživače bitno utječu na dostizanje do korisnika. Za primjer, kroz kolovoz 2018. godine postotak korisnika koji koristi Internet Explorer je 6.97% ili Microsoft Edge 4.24%, što se može smatrati značajnim gubitkom korisnika ako *web* sjedište ne podržava navedene pretraživače. [12]

3.3. Neobavezni zahtjevi osobnog web sjedišta

Neki zahtjevi PWA nisu nužni za uspjeh osobnog *web* sjedišta, ali su svakako dobri za demonstrirati potencijalnom klijentu. Primjer takvog zahtjeva je podrška rada kada uređaj nije spojen na internet. Posljedica toga je brzo učitavanje i fluidno korisničko iskustvo u slučaju slabe veze na internet.

Mogućnost dodavanja aplikacije na *home screen* uređaja omogućava brzo pokretanje što doprinosi kvalitetnom korisničkom iskustvu. Sadržaj je danas poželjno posluživati sigurnim HTTPS protokolom.

4. Korištene tehnologije

4.1. Preact radni okvir

Dinamičke promjene sadržaja omogućene su od strane JS programskog jezika. Jezik omogućava korištenje koncepata imperativnog i funkcionalnog programiranja. Od svoje pojave najčešće se u njemu pisao kôd u imperativnom stilu, dok su svojstva jezika iz funkcionalne paradigme - funkcije kao „*first class citizen*“ - većinom korištena za kontrolu toka izvršavanja asinkronog kôda, poput AJAX zahtjeva.

Imperativnim stilom, naredbom po naredbom, mijenjamo stanje aplikacije brinući o tome gdje je stanje spremljeno te u što će se promijeniti. Također naredbama definiramo pojedine promjene u sučelju. Za takav pristup programiranju kažemo da je niske razine apstrakcije, upravo jer je manji dio izvršavanja u odnosu na naredbe apstrahiran u usporedbi s drugim pristupima. Takav je kôd težak za održavati i nadograđivati jer izlaže programera većoj vjerojatnosti da napravi pogrešku. Pogreške se najčešće javljaju zbog nuspojava promjene stanja u kombinaciji s kontrolom toka programa. Kako bi se doskočilo tome, Facebook je razvio React radni okvir za izradu korisničkih sučelja.

React pristupa razvoju sučelja na deklarativan način. Deklarativnim programiranjem definira se logika kompjutacije, ali bez da se definira kontrola toka. Drugim riječima, naredbe govore što se dogodi, ali ne i kako. O kontroli toka u pozadini brine React, te se tako umanjuju nuspojave mijenjanja stanja. Uspjeh ovog pristupa posebno je naglašen kod asinkronog izvršavanja naredbi. [13]

Deklarativni pristup postignut je konceptom komponenti. React komponenta koristi se kao i HTML element, a definira se JS klasom. Komponenta iscrtava povratnu vrijednost „render“ funkcije koju je obavezno definirati. Vrijednost se sastoji od drugih React komponenti ili HTML elemenata, čime se definira stablasta struktura Document Object Modela (DOM-a). [14]

Pomoću „setState“ metode definira se način na koji se mijenja stanje komponente, čime se postižu promjene u sučelju. Komponente vežu podatke za sučelje preko *propsa,* koji podatke vežu jednosmjerno na sučelje. [15] Izmjene podataka zadaju se reaktivnim pristupom programiranju. React komponente preplaćene su na događaje u DOM-u kroz „Event“ sučelje DOM-a. Funkcije pokrenute određenim događajima obično mijenjaju podatke, koji zatim

dolaze kroz *props-e* nazad do DOM-a kako bi se osvježilo sučelje. Najčešći su događaji interakcije korisnika sa sučeljem.

React radi efikasno osvježavanje sučelja pomoću takozvanog virtualnog DOM-a. Virtualni DOM je DOM koji se nalazi u memoriji i pretraživač ga ne iscrtava na ekran. Koristi ga samo React, te prvenstveno osvježi njega. Potom algoritmom pronalazi razlike između virtualnog DOM-a i stvarnog DOM-a koji iscrtava pretraživač kako bi osvježio samo onaj dio stvarnog DOM-a koji se zapravo promijenio.

React komponente pružaju dodatne metode kojima se kontrolira ponašanje komponente u određenim trenucima njenog života u DOM-u. Prema tome, te metode se nazivaju „lifecycle“ metodama.

Primjeri „lifecycle“ metoda su: [14]

1. „componentDidMount“ koja se izvršava jednom kada komponenta postane dio DOM-a,
2. „componentWillReceiveProps“ koja se izvršava kada komponenta primi nove „propse“ od komponente roditelja,
3. „componentDidUpdate“ koja se izvršava kada se sadržaj komponente osvježi u DOM-u kao rezultat promjene podataka

Sadržaj se eventualno iz React komponenti pretvori u HTML elemente koje pretraživač prepoznaje.

React je napisan u JS-u, a pri korištenju se osim sintakse JS-a koristi i JavaScript XML (JSX) sintaksa. JSX je proširenje JS sintakse, jako sličan HTML-u. Služi definiranju sadržaja korisničkog sučelja unutar povratne vrijednosti „render“ funkcije. [16]

U ovom radu korišten je radni okvir koji je iznimno sličan i napravljen po uzoru na React - Preact. Preact je radni okvir koji koristi iste koncepte korištenja kao i React, ali je lakši pa samim tim omogućava korisniku brže preuzimanje pri posjeti *web* sjedištu.

4.2. Webpack JavaScript modul

JS možemo pokrenuti u pregledniku na dva načina. Prvi način odnosi se na uključivanje skripte pomoću HTML „script“ oznake. Tako se to radilo dugo vremena, ali se pokazao veliki problem u održavanju i nadogradnji kôda. Uključivanje mnoštva skripti unutar preglednika rezultira sporom učitavanju *web* aplikacije. Proces razvoja nalazi novi niz problema jer programer mora ručno dodavati skripte. Komunikacija između skripti je loše riješena

globalnim varijablama, što rezultira opasnošću da prebrišemo stanje koje ne želimo. S vremenom i sam kôd postaje težak za čitati i racionalizirati. Druga opcija je učitati JS kroz posebnu JS datoteku u kojoj je pohranjen sav kôd, ali tu opet nailazimo na probleme čitljivosti kôda, veličine datoteke koja se u učitava korisniku, te ograničavanja vidljivosti varijabli.

Pristup rješavanju modela nalazi se u takozvanim modulima. Moduli definiraju svoj sadržaj te mu odrede imenski prostor (*namespace*) tako da ga ne može pregaziti sadržaj drugog modula. Modul se definira na razini datoteke, a može sadržavati i druge module. Tako korisnikov pretraživač može odvojeno dohvaćati module. Za programera moduli predstavljaju i moćan način određivanja granica enkapsulacije te apstrahiranja. Uz to olakšavaju proces *debugiranja* i testiranja jer nameću pisanje labavo povezanog kôda - različiti dijelovi kôda minimalno utječu na druge pa je vjerojatnost javljanja grešaka manja. [17]

S obzirom na to da je podrška za module u pretraživačima još u razvoju, postoje alati koji omogućuju njihovo korištenje – „bundleri“. „Bundler“ koji se koristi u praktičnom dijelu rada naziva se Webpack. [18]

Osim mogućnosti korištenja koncepta modula, Webpack omogućava razne transformacije kôda kroz proširenja (engl. plugins). Tako omogućava uključivanje transformatora kako bi se podržali različiti pretraživači ili nova svojstva programskih jezika, optimizirao i minificirao kôd, pisao Domain Specific Language (DLS) ili slično. Također omogućuje pakiranje različitih dobara (npr. slike i fontovi) kako bi se mogli koristiti u kôdu, te i sam optimizira kôd pomoću „tree-shaking“ tehnike. Tako eliminira kôd koji se ne koristi. [19]

4.3. Babel prevoditelj

Jedan od transformatora koje ovaj rad koristi pomoću Webpacka je Babel. Babel je prevoditelj programskog jezika koji prevodi JS kôd u stariju i podržaniju verziju JS koda. Glavni razlog za to je upravo kako bi se osigurala podrška JS kôdu za razne pretraživače. Osim toga tako se može omogućiti i korištenje svojstava JS-a koja još nisu dio službene specifikacije pa samim tim nisu još dobro podržana u pretraživačima. [20]

Babel se može konfigurirati da automatski podržava određene okoline u kojima se JS pokreće, najpopularnije okoline, ili da podržava pojedina svojstva. Babel se može koristiti i za proširenje JS jezika proširenjima poput već spomenutog JSX-a, Flow sustava tipova i drugih. [20]

4.4. Syntactically awesome style sheets preprocessor

Kako bi se olakšalo održavanje i nadogradnja JS kôda koriste se razni, već navedeni alati. Kako bi i CSS kôd bio održiv koriste se proširenja CSS jezika kroz takozvane *preprocessor* alate.

U radu se koristi SASS jezik za stiliziranje sadržaja. SASS preprocessor je konfiguriran i radi kao Webpack proširenje. Korištenje SASS-a omogućuje nam pisanje modularnog CSS-a kroz korištenje svojstava koja inače u CSS-u ne postoje. Primjer svojstava su varijable, ugnježđivanje, parcijalne klase, *import-i, mixin-ovi* i nasljeđivanje. [21]

Pomoću njih možemo izdvojiti dijelove kôda te ih koristiti na drugim mjestima. Ukoliko je potrebna nadogradnja ili izmjena, na taj način treba mijenjati kôd samo na jednom mjestu, umjesto na svakom zasebno. SASS pruža dodatne operatore i funkcije za definiranje vrijednosti poput zbrajanja, oduzimanja, množenja, dijeljenja, povećanja prozirnosti, posvjetljivanja i potamnjivanja boje. [21]

4.5. Git distibutivni sustav

Kako bi omogućili verzioniranje kôda s ciljem jednostavnijeg i sigurnijeg procesa razvoja, koristi se Git sustavom verzioniranja. Git omogućuje upravljanje izvornim kôdom i koordinacijom rada na datotekama među više ljudi, praćenjem povijesti promjena u datotekama. Pruža podršku nelinearnog razvoja kroz grananje i spajanje. Za timski rad koriste se Git *web* servisi poput GitHub-a, kako bi cijeli izvorni kôd bio u oblaku, pristupačan članovima tima. Izvorni kôd se nalazi u takozvanim repozitorijima, a osim repozitorija u oblaku – „remote“ repozitorija. Svaki član tima ima instancu repozitorija i na vlastitom računalu - lokalni repozitorij.

Git se koristi tako da se prije početka rada na novom svojstvu aplikacije stvori nova grana gdje će se navedeno svojstvo razvijati. Prilikom razvoja svojstva, logičke cjeline pohranjuju se u lokalnom repozitoriju, na stvorenoj grani, pomoću „commit“ akcije. „Commit“ je definiran *hash* kôdom te sadrži informacije o promjenama. Kako bi se osvježile informacije o promjenama unutar grane na „remote“ repozitoriju izvršava se „push“ akcija. Kada se dovrši razvoj cjeline, kako bi se spojile promjene u glavnu granu sustava, najčešće zvanu „master“, otvara se takozvani „pull request“ - zahtjev za povlačenje promjena. Odobravanjem takvog zahtjeva te spajanjem, u glavnu granu povlačimo promjene grane svojstva. U ovom radu korišten je GitHub *web* servis. [22]

5. Postavljanje aplikacije

Prije početka razvoja aplikacije, prvo se postavlja razvojna okolina, kako bi se djelomično automatizirao razvoj i objava sadržaja. Tako se pojednostavljuje i ubrzava sam razvoj.

Započinje se s izradom novog Git repozitorija pomoću Github-ovog sučelja. Kako bi se mogla razvijati aplikacija u novo napravljenom repozitoriju mora imati lokalnu instancu istog repozitorija. Klonira se repozitorij na vlastito računalo izvršavanjem naredbe „git clone“ preko *terminala* operativnog sustava. Zatim se inicijalizira repozitorij datotekom koja sadrži konfiguraciju projekta, „package.json“. Umjesto da se ručno izradi datoteku, to se radi pomoću Node package manager-a (NPM), izvršavanjem naredbe `npm init -y`. Tako se dobije konfiguracijska datoteka koja definira konfiguraciju u JavaScript Object Notation (JSON) formatu. Popune se poznate vrijednosti, poput imena i autora projekta.

Sljedeće što treba je HTML datoteka koja će se poslužiti korisniku kada pristupi adresi *web* aplikacije - nazivana je „200.html“ prema HTTP odgovoru koji kôdom 200 ukazuje da je HTTP zahtjev bio uspješan. [23]

S obzirom na to da će izvorni kôd biti transformiran zbog podrške za pretraživače, optimizacije, ali i olakšanja razvoja kroz module, potrebna je mapa u koju će se spremati statički sadržaji koji će se zapravo posluživati korisniku. Mapa se naziva „dist“, skraćeno od „distribution“ (raspodjela). „200.html“ je dio navedenog sadržaja pa se prebacuje u „dist“ mapu.

Budući da se ova *Web* aplikacija razvija se u Preact radnom okviru potrebna je datoteka u kojoj se pokreće Preact aplikacija. Nazvana je `app.js`. Datoteku se sprema u mapu „src“, skraćeno od „source“ (izvor), gdje će se stavljati i ostali izvorni kôd.

U „app.js“ datoteci radi se prva Preact komponenta nazvana „App“ koja ispisuje „hello world“ tekst. Zatim se iz Preacta uključuje i poziva funkcija „render“, koja navedenu komponentu prikaže unutar HTML elementa dohvaćenog metodom „document“ objekta – „getElementById“.

Uz to za prevođenje JSX-a u HTML i JS iz Preact paketa treba uključiti i „h“ funkciju. U „200.html“ dodaju se osnovni HTML elementi od kojih se sastoji svaka HTML datoteka – „html“ i „body“. Unutar „body“ elementa dodaje se element unutar kojeg će se prikazivati Preact komponenta koja crta ostatak aplikacije, s „Id“ atributom postavljenim na vrijednost koja se proslijedila „getElementById“ metodi – „app“.

Slika 5.1. Testna komponenta koja u pregledniku ispisuje „hello world“

Izvorni kôd bi pokretanjem u browseru trebao prikazati „hello world“ poruku. Kako bi se iz izvornog kôda zapravo dobio kôd za poslužiti pretraživaču, koristi se Webpack. Instalira se NPM-om, te konfigurira pomoću „webpack.config.js“. U njoj se dodaju podaci Webpack-u o ulaznoj datoteci u aplikaciju - „app.js“, mapu za datoteke koje će se posluživati korisniku - „dist“, te ime JS datoteke koja će sadržavati izvorni kôd Preact aplikacije – „bundle.js“. U „package.json“ pod „scripts“ vrijednost dodaju se naredbe koje će se moći pokrenuti imenom pomoću NPM-a.

Imenovane naredbe:

1. Naredba „start“ - pokreće Webpack server za razvoj, koji poslužuje na vlastitom računalu ne transformiran izvorni kôd kako bi posluženi sadržaj odmah reflektirao promjene u kôdu (naredba je `webpack-dev-server --open`)
2. Naredba „build“ - transformira izvorni kôd u produkcijski kôd koji se poslužuje korisniku, pritom koristeći konfiguraciju proslijeđenu kroz „—config“ parametar (naredba je `NODE_PATH=./src/ webpack --config webpack.config.js`)

Kako bi se zaokružio određeni ciklus razvoja (pisanje kôda, prevođenje, te posluživanje) preostaje još postaviti jednostavan način za objavu sadržaja. Za to se koristi „Surge.sh“ servis koji omogućuje objavu sadržaja kroz jednostavnu naredbu. Surge biblioteka instalira se pomoću NPM-a. Dodaje se „deploy“ naredba u „package.json“ koja obavi transformiranje kôda te ga objavi na domenu „bruna.surge.sh“ pomoću naredbe `npm run build && surge ./dist --domain bruna.surge.sh`.

Izvršavanjem naredbe `npm run deploy` objavljuje se sadržaj na navedenu domenu. Posjetom domene testira se da sve radi kako je očekivano. Na ekranu se dobije ispisana „hello world“ poruka koja to potvrđuje.

Kako bi osigurali podršku za određene pretraživače koristi se Babel. Instalira se pomoću NPM-a te se zatim konfiguracija upiše u „.babelrc“ datoteku. Koristi se „env" *preset,* skraćeno od „environment“ (eng. okruženje), koji omogućuje definiranje okruženja koja transformirani kôd treba podržavati. Definira se podrška za zadnje tri verzije popularnih pretraživača, Chrome pedesetdva, te Safari sedam pretraživača. Uz to se koristi Preact *preset* kako bi „Babel“ znao prevesti JSX, te „stage-1“ [24] za podršku osobnosti JS-a koje su još u razvoju. Priključak se instalira preko NPM-a, te se dodaje u Babel konfiguraciju „transform-class-properties“ osobnost, kako bi je mogli koristiti za lakši razvoj.

Za pisanje modularnog CSS-a koristi se SASS. Instalira se ga pomoću NPM-a zajedno sa potrebnim Webpack priključkom. U Webpack konfiguraciju dodaje se SASS priključak koji datoteke sa „*.scss“ nastavkom u imenu transformira u odgovarajući CSS.

Konačno, kako bi se osiguralo da kôd prati dobre prakse, instaliraju se ESLint i Stylelint[4] alati pomoću NPM-a, te njihovi priključci dodaju u Webpack konfiguraciju. Prilikom transformiranja kôda od strane Webpack-a, ovi alati detektiraju loše prakse u kôdu te ih prikazuju kao greške. Kako bi se spriječilo da se takav kôd pohrani, pomoću „husky“ alata definiram „git hook-ove“ - izvršavanje naredbe prije ili nakon određene Git naredbe. Ovdje koristim „precommit hook“ koji izvršava analizu kôda pomoću ESLint i Stylelint alata prije nego se izvrši „commit“, te ga spriječi ako pronađe greške.

[4] Linteri su alati koji pomažu izbjeći pogreške u pisanju kôda.

6. Arhitektura

6.1. Arhitektura agnostična prema platformi (Platform agnostic)

Za aplikaciju se kaže da je „platform agnostic“ kada radi jednako dobro na više od jedne platforme. Cilj ovog osobnog *web* sjedišta je da se može pokretati i kvalitetno koristiti na osobnim računalima, mobitelima i tabletima, neovisno o pretraživaču. O podršci za pretraživače, kao što je već navedeno, brinu *transpileri* poput Babel-a i PostCSS-a.

Kako bi sadržaj bio pregledan na svim uređajima, dizajn mora biti responzivan. Raspored, veličina i način prikaza elemenata prilagođavat će se s veličinom prozora pretraživača ili ekrana. U svrhu olakšanja razvoja, problemu valja pristupiti "mobile-frist" metodologijom. [25]

Prvo se razvija jednostavan sadržaj za mobilne uređaje s malim ekranima jer oni pružaju najmanje prostora za smjestiti sadržaj. Zatim se sadržaj prilagođava osobnim računalima koja u pravilu pružaju više prostora, a potom se poboljšava sadržaj za male i velike ekrane. Tako se izbjegava stiskanje sadržaja koji nije bio predviđen za male ekrane.

Za postizanje responzivnog sadržaja koristi se CSS svojstvo „flexbox“ koje automatski prilagođava raspored i veličinu djece elemenata unutar „flexbox“ elementa ovisno o njegovoj veličini. Kako bi drugačija svojstva bila primjenjena na elemente ovisno o veličini prozora pretraživača koriste se „media query-i“, kojima se definiraju koja svojstva se primjenjuju na koje veličine prozora pretraživača. Gdje se ne mogu primijeniti navedene tehnike sadržaj se prikazuje u drugačijem obliku za različite veličine prozora pretraživača. To se postiže pomoću JS-a, te se pritom konceptom „resize observera“ zadržava svojstvo responzivnosti.

„Resize observer“ je modul aplikacije koji iščekuje promjenu veličine prozora pretraživača i izvršava niz naredbi kada se promjena dogodi. Ova tehnika koristi se samo kada je nužno jer otežava održavanje i nadogradnju aplikacije. [26]

S obzirom na to da mobilni uređaji često pristupaju sadržaju *web* sjedišta preko sporije 3G veze, potrebno je brinuti i o performansama. Prije svega, kôd valja optimizirati kako bi se reducirala njegova količina, u svrhu postizanja bržeg učitavanja web aplikacije. O tome brine Webpack koji optimizira kôd pomoću „tree-shaking“ algoritma te drugih priključaka. Također, treba brinuti da sadržaj poput slika i videa bude optimiziran. Zbog toga se koriste formati koji nude najbolji omjer kvalitete i veličine medijske datoteke. Pomoću alatia za

optimizacije „ImageOptim“ i „SVGOMG“ uklanjaju se nepotrebni meta podaci kako bi veličina datoteka bila manja.

Osim veličine, treba obratiti pažnju i kada se što učitava kako bi se smanjio „above the fold“ sadržaj - sadržaj kojem korisnik može pristupiti bez skrolanja. „Above the fold“ sadržaj se učitava prvi, kao resurs najvišeg prioriteta, kako bi se korisniku što prije prikazao umjesto praznog sadržaja. Potom se asinkrono učitavaju ostali resursi, poput ostatka sadržaja i fontova. Slike i video su resursi niskog prioriteta te se učitavaju među zadnjima upravo zbog njihove veličine.

6.2. Jedinstvena točka ulaza

Aplikaciji se pristupa posjetom jedinstvenoj *web* adresi, preuzima se sadržaj, te o ostatku interakcije i sadržaju dinamički brine Preact. Kako bi se osiguralo da korisnik završi na pravom mjestu pri pokušaju pristupanja aplikaciji, sve rute se preusmjeravaju na naslovnu stranicu.

Kako se radi o SPA, ne poslužuje se u potpunosti novi sadržaj na posjet drugoj ruti. Sadržaj se dinamički osvježava pomoću „preact-router“ paketa. „Preact-router“ brine o navigaciji po stranici, te održavanju stanja pretraživača u smislu adrese i povijesti, svježim. Također definira koji se sadržaj, i kako, osvježava ovisno o ruti.

6.3. Upravljanje stanjem web sjedišta

S obzirom na to da je u pitanju jednostavno i malo *web* sjedište u smislu sadržanog izvornog kôda, jedinstveno globalno stanje *web* sjedište nije potrebno, pa tim ni alat za upravljanje istim. Umjesto toga, sadržaj kojeg koriste komponente je hardkodiran[5], a stanje služi kako bi se definiralo ponašanje komponenti. Tako komponente same sadrže potrebno stanje, a ako je potrebno, prosljeđuju ga kroz „propse“ djeci komponentama. Za slučaj potrebe prosljeđivanja stanja komponenti roditelju, koristi se koncept „render propsa“. Komponenta roditelj komponenti dijete prosljeđuje funkciju „render“ kroz „propse“. „Render“ funkcija kao

[5] Hardkodiranje način izrade programske potpore koja se zasniva na smještanju podataka izravno u kôd programa. [30]

parametar može primiti stanje komponente djeteta, pa komponenta dijete može kroz poziv proslijediti stanje komponenti roditelju.

7. Struktura

Na prvoj razini strukture direktorija i datoteka izvornog kôda nalaze se konfiguracijske datoteke *web* sjedišta i razvojnih alata.

Direktoriji su podijeljeni na:

1. „assets“ - u njemu su smještene medijske datoteke poput slika, videa i sličnih resurse koje želimo koristiti u aplikaciji
2. „dist“ - distribucijski kôd koji je rezultat transformacija pomoću Webpack-a; to je kôd koji se poslužuje korisniku
3. „src“ - izvorni kôd koji se prepravlja i razvija
4. „node_modules“ - alati razvijeni od strane drugih koji se koriste u razvoju web sjedišta
5. „theme“ - nepromjenjive vrijednosti koje se koriste za stilizirati sučelje; pružaju jednostavan i jedinstven način unošenja promjena u izgledu *web* sjedišta

Unutar „src“ direktorija nalaze se:

1. Datoteka „app.js“ - početna komponenta i naredba za crtanje aplikacije unutar HTML elementa
2. Datoteka „globals.scss“ - CSS stilovi koji se primjenjuju na globalnoj razini unutar *web* sjedišta, te postavljaju početna pravila za daljnji razvoj
3. Direktorij „components“ - sadrži Preact komponente koje se sastoje od JSX dijela („index.js“), SCSS dijela za stiliziranje komponente („styles.scss“), te podataka u JSON formatu

Komponente se realiziraju tako da ih se može koristiti na različitim mjestima i proširiti njihov izgled i ponašanje.

8. Izrada web sjedišta

Držeći na umu primarni cilj ovog *web* sjedišta, informiranje ciljane skupine o osobnim radovima i zanimanjima, započela se izrada aplikacije. Vizualnim jezikom želi se primarno odvući pažnja na istaknute radove s ciljem poticanja zanimanja korisnika. Osim radova, izgled *web* sjedišta ističe osobnost i stil, kao i umijeća i vještine izrade *web* sjedišta. Cilj je potaknuti interes potencijalnih klijenata kao i drugih programera i dizajnera zainteresiranih za suradnju.

Kako je ranije objašnjeno, ovo osobno *web* sjedište radi kao SPA. Vodeći se time organizirana je naslovna stanica. Ovo *web* sjedište se primarno dijeli na sekcije, kojima se razdvaja sadržaj i određuju prioriteti elemenata na *web* sjedištu.

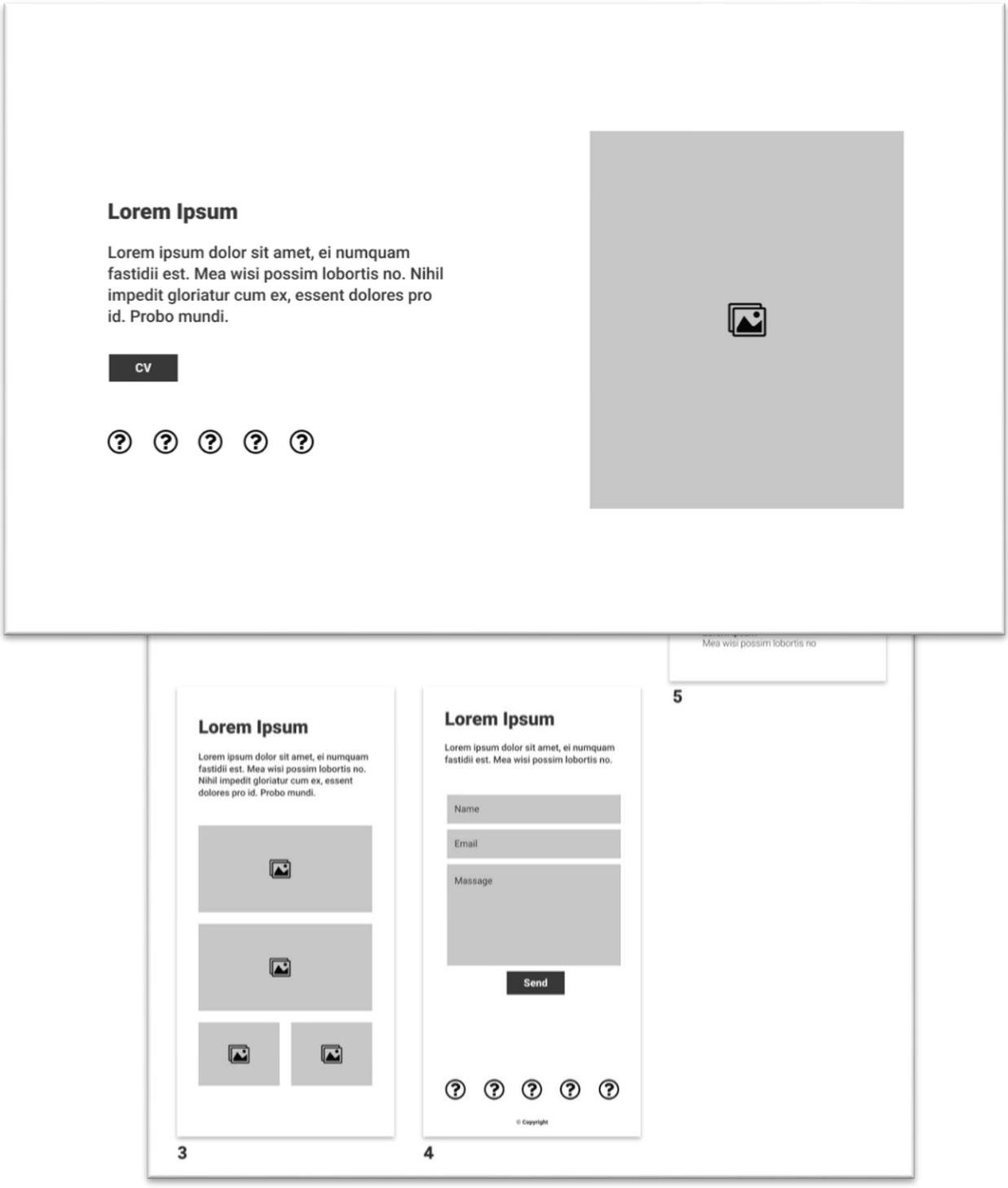
Osnovne sekcije koje se prikazuju su:

1. „O meni" koji predstavlja uvodni sadržaj *web* sjedišta
2. „Projekt" sekcije u kojima su prikazane putem fotografija i video animacija vještine i umijeća u područjima struke kao i najistaknutiji projekt
3. „Kontakt" koji je na dnu *web* sjedišta te omogućava slanje elektronske pošte putem forme
4. „Životopis" je posebna ruta na kojoj je prikazan životopis, a može se prikazati klikom na gumb smješten u uvodnoj „O meni" sekciji

8.1. Izrada skica (Wireframes)

Nakon istraživanja i postavljanja primarne svrhe i ciljeva dolazi se do jednog od najbitnijih dijelova dizajn procesa - izrade skica (engl. Wireframing). Skica *web* sjedišta je vizualni reprezent strukture samog *web* sjedišta. Većinom pomoću *placeholdera* i generičkog teksta prikazuje se što će se događati na ekranu, ali ne i kako će to izgledati. Drugim riječima, izbjegava se korištenje boja, grafika, različitih tipografija dok se glavni fokus pridodaje funkcionalnosti, ponašanju elemenata i postavljanju arhitekture sadržaja. Time se prikazuje „user flow", zamišljeni tok ponašanja korisnika na *web* sjedištu. [27]

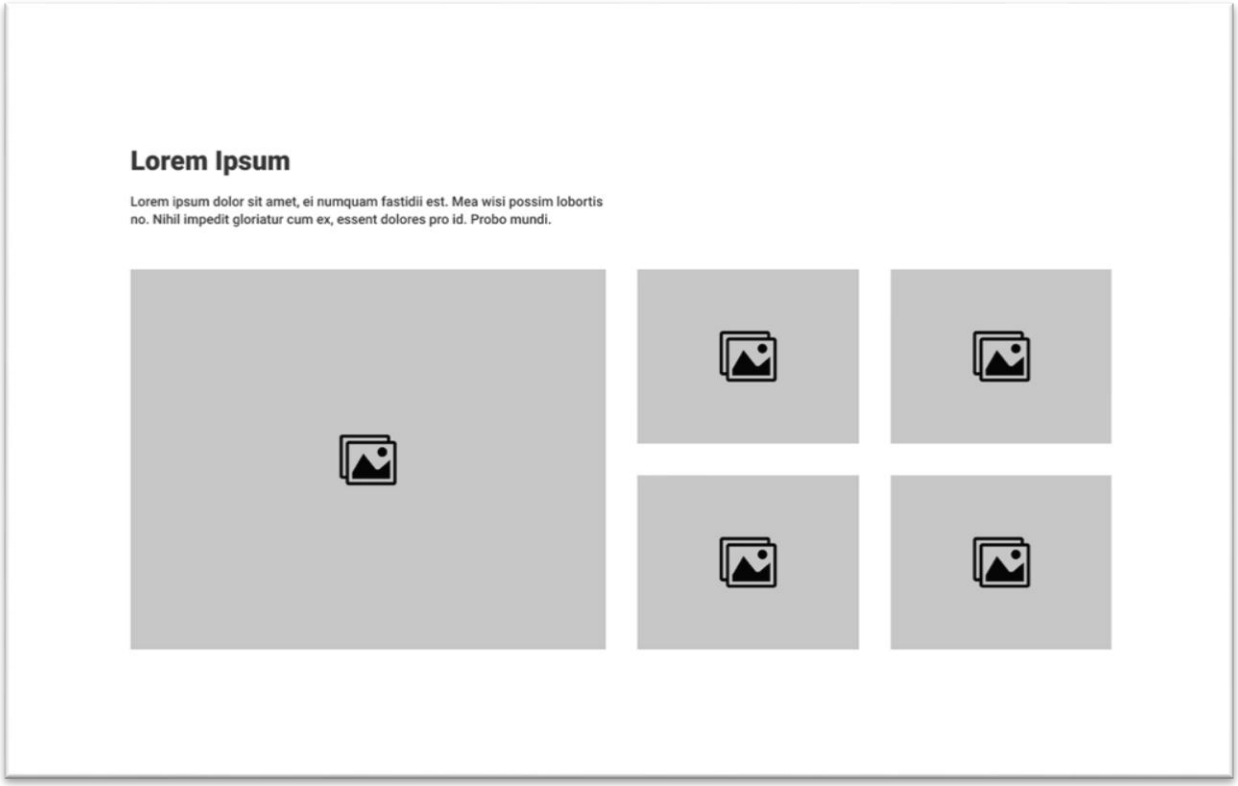
Skica *web* sjedišta može se realizirati na papiru ili pomoću različitih digitalnih alata. U radu je korištena Figma, besplatna online aplikacija za dizajn korisničkih sučelja. Za vizualizaciju naslova, paragrafa i ostalih tekstualnih elemenata korišten je alternativni latinički tekst „Lorem Ipsum". Kvadrati s ikonom fotografije kao medijske datoteke prikazuju

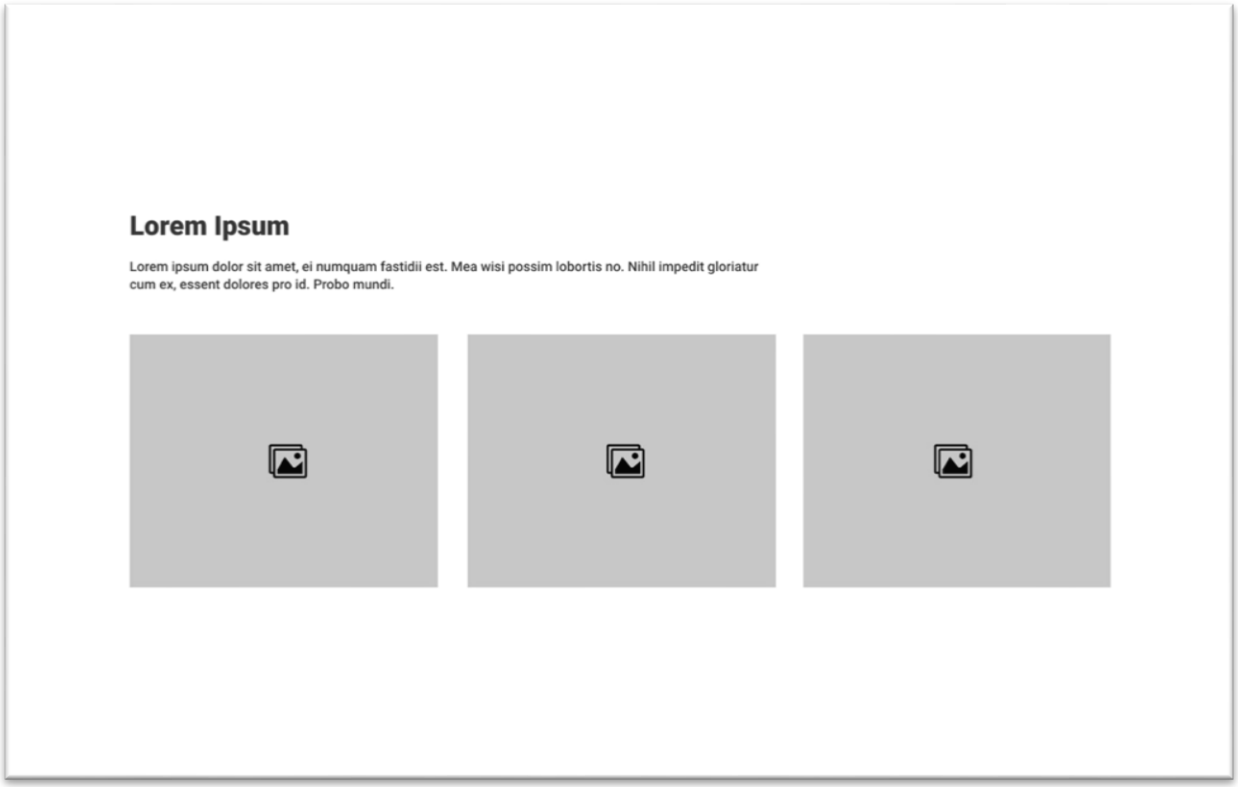


prostor koji će se zamijeniti fotografijama, a ikone kruga s upitnikom linkove na kanale društvenih mreža. Skice su realizirane za dvije širine zaslona, za mobilne i desktop uređaje „mobile-first“ principom.

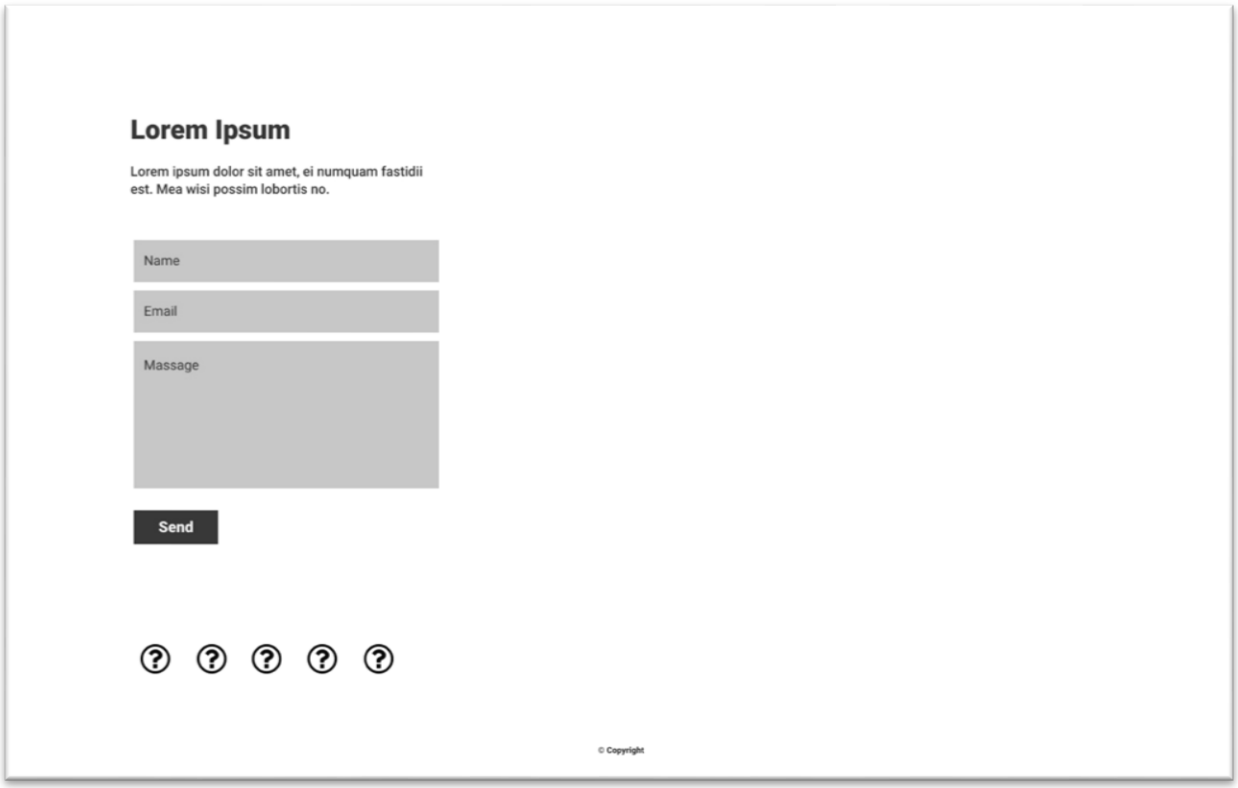
Slika 8.1. Skice sekcija naslovne stranice za manje ekrane (O meni (1), Projekt (2), Prijekt2 (3), Kontakt (4), Životopis (5))

Slika 8.2. Skica "Projekt" sekcije desktop veličine ekrana


Lorem Ipsum
Lorem ipsum dolor sit amet, ei numquam fastidii est. Mea wisi possim lobortis no. Nihil impedit gloriatur cum ex, essent dolores pro id. Probo mundi.



Slika 8.4. Slika 6.4. Slika 6.4. Skica „Projekt2" sekcije za verzije za desktop ekrane



Slika 8.5. Skica „Kontakt" sekcije za verzije za desktop ekrane

8.2. Dizajn specifikacije

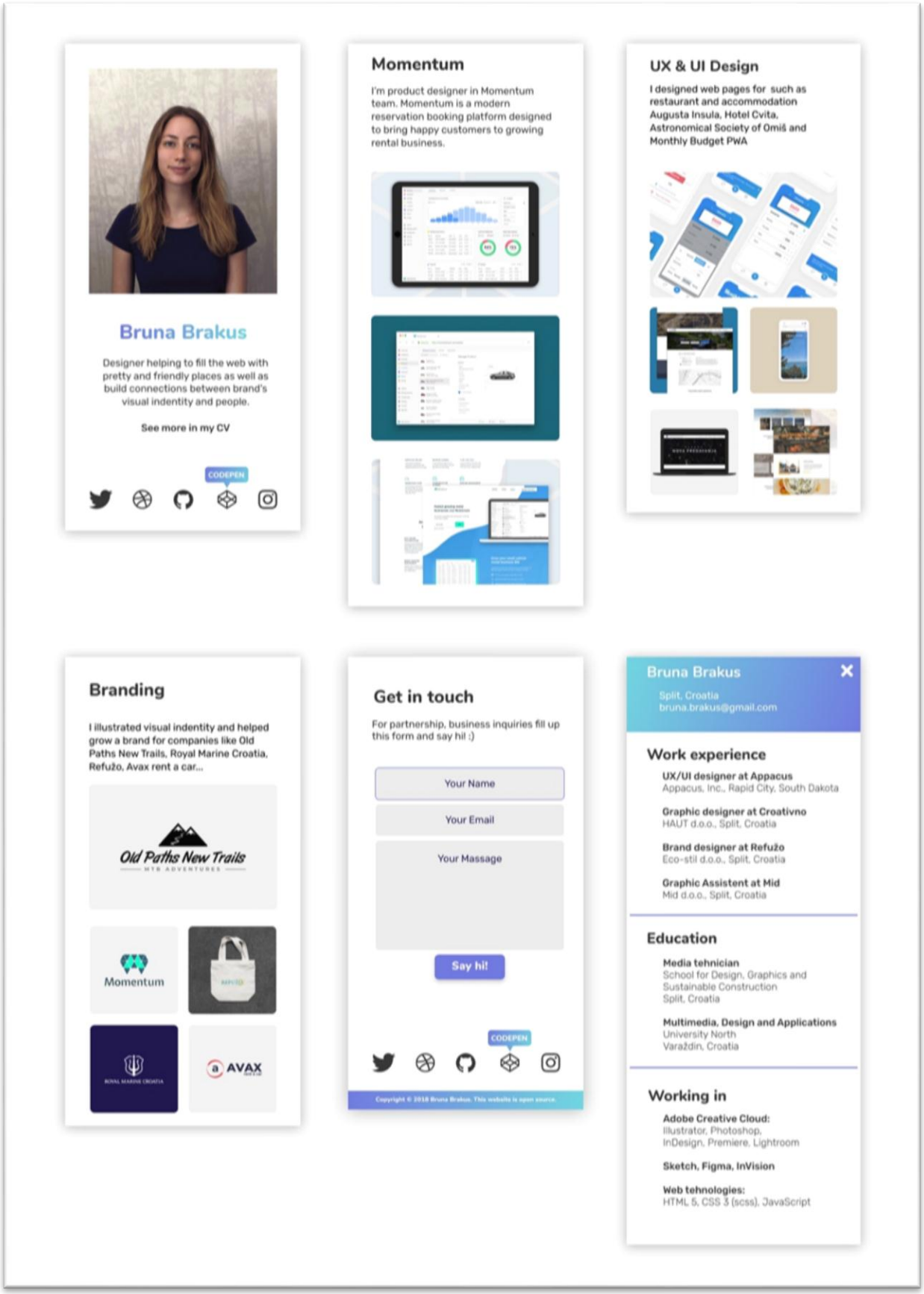
Koristeći Figma *web* aplikaciju nastavljen je proces izrade dizajna *web* sjedišta. Dodani su naslovi cjelina i tekstovi paragrafa istaknuti i zamijenila generičke tekstove unutar paragrafa. Web sjedište je pisano na engleskom jeziku i ima 7 sekcija: 6 na naslovnoj stranici i jednu na „CV“ ruti.

Pomoću programa Adobe Illustrator i Photoshop, Sketch-a i iMovie-a pripremljeni su multimedijski sadržaji. *Placeholder-i* namijenjeni slikovnim sadržajima iz skica zamjenjeni su fotografijama, *mockup-ima* i video materijalima. Dodane su i ikone društvenih mreža. Prikazuju na dva mjesta na naslovnoj stanici – u prvoj sekciji i na dnu.

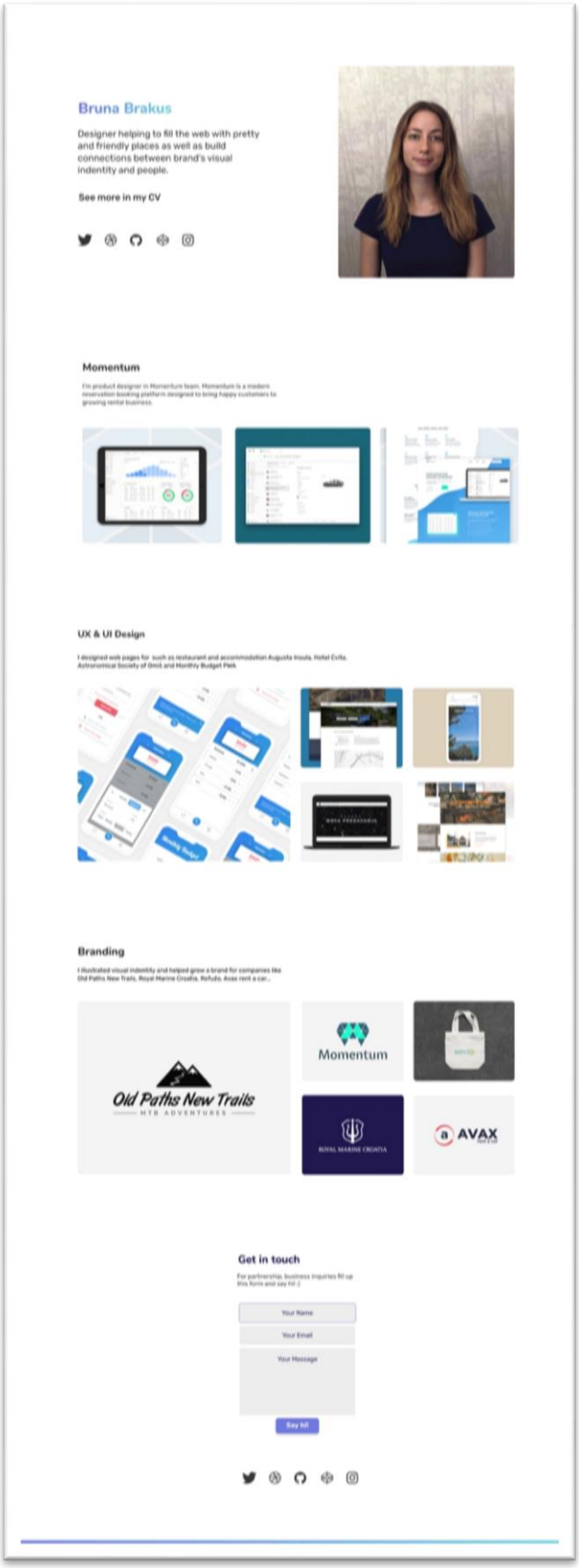
Odabrana je kombinacija jednostavne bijele pozadine kako bi sadržaj došao do izražaja. Da bi se izrazio kontrast s pozadinom i time nesmetano čitao sadržaj na *web* sjedištu tipografija je prikazana u tamno sivoj boji. Kako dizajn ne bi bio toliko minimalističan, dodaje se ljubičasto – tirkizni linearni gradijent kao osnovni stilski detalj. U gradijentu je prvi naslov i nekoliko detalja kako bi se postigla dinamika.

Za odabir fontova korišten je „Google Fonts“ direktorij besplatnih obitelji fontova koji se na jednostavan način ugrade u dizajn koristeći CSS *property*. Za naslove je odabran sans serifni font „Nunito“ koji je namijenjen prikazivanju na ekranima. Font dolazi u 7 težina, a u ovom projektu korištena je Extra-Bold varijanta. Za prikaz paragrafa odabran je također sans serifni font „Rubik“, koji omogućuje konzistentno i nesmetano čitanje rečenica.

Dizajn je prvotno pripremljen za manje ekrane mobilnih uređaja koristeći „mobile-first“ princip te zatim korigiran po ranije pripremljenim skicama za veće ekrane. Verzije dizajna za male i velike ekrane prikazane su na slikama 8.6 i 8.7.



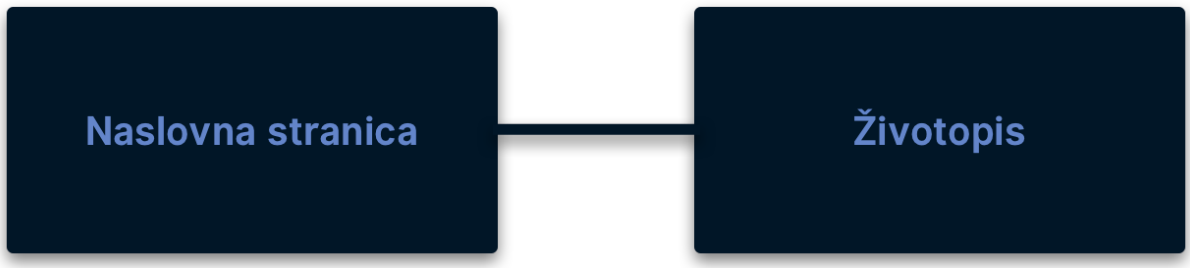
Slika 8.6. dizajn osobnog web sjedišta za male ekrane



Slika 8.7. Slika 8.6. dizajn osobnog web sjedišta za velike ekrane

8.3. Karta web sjedišta

„Site map“ ilustrira hijerarhiju prikazivanja sadržaja. U ovom slučaju postoji jedno *web* sjedište, ali s dvije rute. Dodatna ruta je „Životopis“ koja se otvara preko naslovne stranice te zatvara klikom na „X“ ikonu.



Slika 8.8. Site map osobnog web sjedišta

9. Realizacija dizajna pomoću komponenti

9.1. „App“ komponenta

Kao što je već navedeno, izvorna komponenta crta se unutar HTML elementa proslijeđenog kao drugi parametar „render“ funkciji uključenoj iz Preact-a. Budući da se radi o izvornoj komponenti, „App“ komponenta nema komponentu roditelja, pa kroz *propse* ne prima ništa. U „render“ funkciji definiran je klijentski dio aplikacije odnosno korisničkog sučelja.

Pomoću „preact-routera“ definirane su rute i komponente koje se crtaju za pojedine rute.

Sadržane su dvije rute:

1. '/cv' - prikazuje „Cv“ komponentu koja apstrahira sadržaj koji se prikazuje preko cijelog prozora pretraživača - cijelu stranicu
2. '/' - početna ruta prikazuje „Home“ komponentu koja apstrahira početnu stranicu, te sve podrute koje ne postoje preusmjerene su na ovu rutu

„Home“ komponenta crta komponente koje predstavljaju sekcije stranice – „Welcome“, „Work“, „Contact“, „Footer“.

Navedene komponente, osim „Contact“ komponente koja sadrži formu, isključivo su apstrakcija nad postojećim HTML elementima te nemaju definirano ponašanje.

9.2. „Contact“ komponenta

Ova komponenta sadrži formu koja pruža korisnicima aplikacije način kontaktiranja putem elektroničke pošte. Forma koju realizira komponenta je klasična HTML forma, koja kao akciju ima definiranu adresu servisa elektroničke pošte „Formspree“.

Na „submit“ akciju korisnika klikom na „Say hi!“ gumb, forma šalje podatke kroz HTTP POST zahtjev servisu elektroničke pošte, koji ih zatim prosljeđuje elektroničkoj pošti koja je definirana kao ruta adrese servisa elektroničke pošte. Kako bi se zaštitilo od *spam* robota forma sadrži skriveni „input“ element imena „_gotcha“ pomoću kojeg servis elektroničke pošte zna radi li se o *spam-u* (roboti će za razliku od korisnika vidjeti ovo polje, te ga stoga ispuniti).

9.3. „ProgressiveImage“ komponenta

„Progressive Image“ komponenta definira ponašanje koje nije kontrolirano od strane korisnika. Služi poboljšanju prividnog vremena učitavanja sadržaja koristeći Low Quality Image Placeholders (LQIP) tehniku. Komponenta unutar svog stanja pohranjuje adresu slike za prikazati.

Prije nego se komponenta veže na DOM, započne s asinkronim učitavanjem *placeholder* slike koja je slabije kvalitete i manje veličine, te se stoga brže učitava. Jednom kada se učita, osvježi se stanje komponente adresom *placeholder* slike, što pokrene novo crtanje komponente. Dok se korisniku prikazuje *placeholder* slika slabije kvalitete, komponenta već započne s učitavanjem originalne slike. Jednom kada se učita opet se osvježava stanje komponente, ovoga puta s adresom originalne slike, te se korisniku novim crtanjem prikazuje originalna slika.

```
import { h, Component } from 'preact';

export default class ProgressiveImage extends Component {
  state = { src: '' };

  componentWillMount() {
    const { src, placeholder } = this.props;
    this.fetchImage(placeholder)
      .then(this.loadImage)
      .then(() => this.fetchImage(src))
      .then(this.loadImage);
  }

  loadImage = src => {
    this.setState({ src });
  };

  fetchImage = src =>
    new Promise(resolve => {
      const image = new Image();
      image.src = src;
      image.addEventListener('load', () => resolve(src), false);
    });

  render({ className, alt }) {
    return <img className={className} alt={alt} src={this.state.src} />;
  }
}
```

Slika 9.1. kod „ProgressiveImage“ komponente

9.4. Ostale komponente i realizacija dizajna

Ostale komponente isključivo su apstrakcija nad postojećim HTML elementima. Osim apstrahiranja HTML elemenata, apstrahiraju se stilske promjene u prikazu koji se na njih primjenjuje. Kako bi se lakše postigla konzistencija, ali i olakšalo održavanje koda, zajedničke vrijednosti razmaka među elementima, boja, tipografije, širina prozora, sjena te rubova, sadržane su u SASS konstantama poput:

1. „typography.scss“ gdje su sadržane vrijednosti obitelji fontova koje se koriste na web sjedištu, kerninga i veličine fonta.
2. „spacings.scss“ gdje su sadržane vrijednosti koje se koriste za postizanje razmaka između dvije komponente, te unutar njih. Rezultat je lakše postizanje konzistencije u razmacima među elementima, kao i njihova promjena, jer je potrebna izmjena samo jedne vrijednosti.
3. „screens.scss“ gdje su sadržane širine prozora na kojima se mijenja način prikaza elemenata te njihov izgled. Na taj način postignuta je responzivnost dizajna i osigurano da se sučelje transformira konzistentno kroz promjene veličine ili orijentacije ekrana, što smanjuje vjerojatnost grešaka u sučelju za određene veličine.

Kao što je ranije navedeno, kako bi razvoj i održavanje bilo što lakše koriste se najnovija svojstva CSS3 kao što su CSS *grid* i *flexbox*.

CSS *grid* pruža jednostavan i prirodan način rasporedbe elemenata unutar korisničkog sučelja definirajući ponašanje ovisno o veličini i promjenama veličine ekrana. *Flexbox* se koristi kako bi se riješili isti problemi, ali za jednostavne slučajeve. On pruža podskup funkcionalnosti CSS *grida*, ali je jednostavniji za koristiti.

10. Optimizacije veličine, brzine učitavanja i prikaza klijentskog sadržaja

10.1. Kritični CSS kod

„Above the fold“ je sadržaj kojeg korisnik vidi u trenutku kada pristupi *web* sjedištu, bez listanja sadržaja. Stoga navedeni sadržaj ima najveći prioritet učitavanja, te je cilj da bude što manji.

Kako bi smanjili „above the fold“ sadržaj, razdvaja se kritični CSS. CSS koji je definiran za „above the fold“ sadržaj stavlja se u početnu HTML datoteku *inline* u „style“ atribut HTML elemenata. Tako će se učitati zajedno s DOM-om, dok ostatak koji nije jednako visokog prioriteta učitavamo naknadno.

Rješenje je realizirano priključkom za Webpack naziva „webpack-critical“. Nakon instaliranja navedenog pomoću NPM-a, dodajese u listu priključaka u Webpack konfiguraciji, te prosljeđuje put do mape s distribucijskim datotekama. Priključak prilikom generiranja distribucijskih datoteka crta sadržaj i zaključuje koji je CSS kritičan za „above the fold“ sadržaj. Potom ga izdvaja i ubacuje pomoću „style“ HTML atributa u „200.html“ datoteku.

10.2. Minifikacija koda

Imena u izvornom kodu definirana su kako bi olakšala čitanje koda, te ne utječu na korisničko iskustvo korištenja aplikacije. Kako bi smanjili sadržaj koji se učitava korisniku te samim tim ubrzali učitavanje, koristi se Webpack priključak koji minificira kod, tj. skraćuje korištena imena. Priključak koji se koristi u radu naziva se „UglifyJS“, te osim minifikacije može raditi i druge transformacije nad kodom. Prije svega male optimizacije koje vode do bržeg izvršavanja koda.

Kako bi se koristio ovaj priključak, potrebno je instalirati ga pomoću NPM-a te dodati u listu priključaka u Webpack konfiguraciji.

10.3. Učitavanje slika

Slike su učitane Low Quality Image Placeholders (LQIP) tehnikom, tako da se prvo učita verzija slike kojoj je smanjena kvaliteta i veličina. Potom se učitava originalna slika. Iako je trajanje dva HTTP zahtjeva za slikama duže od jednog, sadržaj prije dolazi do korisnika, pa je

prividno vrijeme učitavanja kraće. Iako i za ovaj problem postoji priključak za Webpack koji ga rješava, u ovom radu napravljena je posebna Preact komponenta koja brine o učitavanju slike. *Placeholder* slike generirane su pomoću Adobe Photoshop-a, te su zapravo smanjena verzija originala. Kada se prikazuju korisniku rastegnu se i budu loše kvalitete, međutim ispunjavaju svrhu predstavljanja sadržaja korisniku u što kraćem roku.

Slika 10.1. placeholder slika manje kvalitete (lijevo) i slika u originalnoj kvaliteti (desno)

11. Zaključak

Ovaj rad prikazuje postupak realizacije osobnog *web* sjedišta pomoću modernih tehnologija s ciljem postizanja korisničkog iskustva usporedivog s iskustvom korištenja nativne aplikacije. Realizacija rješava problem responzivnosti sučelja i dostupnosti sadržaja, bez prekida u radu do kojih inače dolazi zbog request/response prirode HTTP protokola. Pritom su korišteni Preact radni okvir koji omogućava apstrakciju dijelova sučelja pomoću koncepta komponente, SASS koji omogućava modularizaciju CSS-a, Webpack *bundler* i Babel prevoditelj koji zajedno omogućavaju automatizaciju optimiziranja veličine klijentskog sadržaja, podrške za različite verzije pretraživača i povezivanja izvornog koda.

S ciljem postizanja brzog učitavanja i prikaza sadržaja korisniku koriste se tehnike asinkronog učitavanja sadržaja i minifikacije koda. Za podršku različitih verzija pretraživača koriste se *polyfilli* uključeni pomoću Babel prevoditelja. Tehnikom responzivnog dizajna podržavaju se različite veličine ekrana, dok se generalno visokim performansama umanjuje efekt performansi *hardwarea* i internetske veze na korisničko iskustvo.

Prednost ovakve realizacije je što omogućava većem broju korisnika korištenje aplikacije. Rezultat je povećana mogućnost uspjeha aplikacije na tržištu. Suvremeni alati i tehnologije pojednostavnjuju ovakvu realizaciju tako što rješavaju tehničke probleme, ili pružaju jednostavan način rješavanja tehničkih problema. To omogućava programeru da se fokusira na probleme koje rješava *web* sjedište umjesto tehničkih detalja realizacije.

Nedostatak ovakve realizacije je nemogućnost indeksiranja sadržaja od strane tražilica. Navedeni problem rješava se tehnikom *prerenderinga* kojom je omogućeno iscrtavanje dinamičkog sadržaja u DOM prije pristupanja sadržaju od strane *web crawler-a*. Time je postignuta vidljivost sadržaja *web crawler-ima* tako da bi ga mogli indeksirati.

12. Literatura

[1] W. t. f. developers, »MDN web docs,« 13 September 2018. [Mrežno]. Available: https://developer.mozilla.org/en-US/docs/Web. [Pokušaj pristupa rujan 2018.].

[2] PWA, »Progressive Web Apps,« 24 srpanj 2018. [Mrežno]. Available: https://developers.google.com/web/progressive-web-apps/. [Pokušaj pristupa kolovoz 2018].

[3] C.-S. Overview, »MDN Web Docs,« 25 rujan 2018. [Mrežno]. Available: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Client-Server_overview. [Pokušaj pristupa rujan 2018].

[4] P. Checklist, »Progressive Web App Checklist,« 24. srpanj 2018.. [Mrežno]. Available: https://developers.google.com/web/progressive-web-apps/checklist#site-uses-cache-first-networking. [Pokušaj pristupa kolovoz 2018.].

[5] XHR, »MDN developer docs,« 29 siječanj 2018. [Mrežno]. Available: https://developer.mozilla.org/en-US/docs/Glossary/XHR_(XMLHttpRequest). [Pokušaj pristupa kolovoz 2018.].

[6] SPA, »Single Page Application Tracking,« 30 lipanj 2018. [Mrežno]. Available: https://developers.google.com/analytics/devguides/collection/analyticsjs/single-page-applications. [Pokušaj pristupa kolovoz 2018].

[7] M. Wasson, »Microsoft Magazine,« studeni 2013. [Mrežno]. Available: https://msdn.microsoft.com/en-us/magazine/dn463786.aspx. [Pokušaj pristupa rujan 2018].

[8] S. Workers, »w3,« 2 studeni 2018. [Mrežno]. Available: https://www.w3.org/TR/service-workers-1/. [Pokušaj pristupa rujan 2018].

[9] S. W. API, »MDN developer docs,« 30 kolovoz 2018. [Mrežno]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API. [Pokušaj pristupa rujan 2018.].

[10] J. Archibald, »Web Fundamentals,« 21 rujan 2018. [Mrežno]. Available: https://developers.google.com/web/fundamentals/instant-and-offline/offline-cookbook/. [Pokušaj pristupa rujan 2018].

[11] P. K. Matt Gaunt, »The Web App Manifest,« 24 srpanj 2018. [Mrežno]. Available: https://developers.google.com/web/fundamentals/web-app-manifest/. [Pokušaj pristupa kolovoz 2018].

[12] StatCounter, »Browser Market Share,« kolovoz 2018. [Mrežno]. Available: http://gs.statcounter.com/browser-market-share/desktop/worldwide/#monthly-200901-201807. [Pokušaj pristupa rujan 2018].

[13] Lifecycle, »React,« 2018. [Mrežno]. Available: https://reactjs.org/docs/state-and-lifecycle.html. [Pokušaj pristupa rujan 2018].

[14] Component, »React,« 2018. [Mrežno]. Available: https://reactjs.org/docs/react-component.html. [Pokušaj pristupa rujan 2018].

[15] State, »React,« 2018. [Mrežno]. Available: https://reactjs.org/docs/faq-state.html. [Pokušaj pristupa 23 rujan 2018].

[16] I. JSX, »Introducing JSX,« 2018. [Mrežno]. Available: https://reactjs.org/docs/introducing-jsx.html. [Pokušaj pristupa rujan 2018].

[17] S. Larkin, »webpack,« 2018. [Mrežno]. Available: https://webpack.js.org/concepts/modules/. [Pokušaj pristupa 2018].

[18] W. webpack, »Webpack,« [Mrežno]. Available: https://webpack.js.org/concepts/why-webpack/.

[19] T. Shaking, »Webpack,« 2018. [Mrežno]. Available: https://webpack.js.org/guides/tree-shaking/. [Pokušaj pristupa rujan 2018].

[20] W. i. Babel, »Babel,« 2018. [Mrežno]. Available: https://babeljs.io/docs/en/. [Pokušaj pristupa rujan 2018].

[21] SASS, »SASS Reference,« 19 rujan 2018. [Mrežno]. Available: https://sass-lang.com/documentation/file.SASS_REFERENCE.html. [Pokušaj pristupa rujan 2018].

[22] G. guides, »Github,« 30 studeni 2017. [Mrežno]. Available: https://guides.github.com/introduction/flow/. [Pokušaj pristupa rujan 2018].

[23] Success, »httpstatuses,« 2018. [Mrežno]. Available: https://httpstatuses.com/200. [Pokušaj pristupa rujan 2018].

[24] D. A. Rauschmayer, Exploring ES2016 and ES2017, 2018.

[25] W. Design, »Mobile websites,« 31 listopad 2016. [Mrežno]. Available: https://allthewayupmedia.com/mobile-website/. [Pokušaj pristupa rujan 2018].

[26] Surma, » Resize Observer,« 2 lipanj 2018. [Mrežno]. Available: https://developers.google.com/web/updates/2016/10/resizeobserver. [Pokušaj pristupa rujan 2018].

[27] D. M. Brown, Communicating Design: Developing Web Site Documentation for Design and Planning (2nd ed.), New Riders Press, 2011.

[28] m. Pale, »Pregled pojmova,« 10 svibnja 2016. [Mrežno]. Available: https://www.cis.hr/www.lexis/pojmovi_view.php?l=H. [Pokušaj pristupa rujan 2018].

[29] I. Naylor, »Progressive Web Apps,« 3 studeni 2017. [Mrežno]. Available: https://appinstitute.com/pwa-vs-native-apps/. [Pokušaj pristupa rujan 2018].

[30] C. Dictionaries, Collins English Dictionary, London: HarperCollins Publishers, 2014.

13. Popis slika

Slika 2.1. Klijent-server arhitektura [3] 2
Slika 2.2: Ciklus klasične stranice i SPA [7] 4
Slika 2.3. Cache first način rada service workera [10] 5
Slika 5.1. Testna komponenta koja u pregledniku ispisuje „hello world“ 15
Slika 8.1. Skice sekcija naslovne stranice za manje ekrane (O meni (1), Projekt (2), Projekt 2 (3), Kontakt (4), Životopis (5)) 21
Slika 8.4. Skica „Projekt2" sekcije za verzije za desktop ekrane 23
Slika 8.5. Skica „Kontakt" sekcije za verzije za desktop ekrane 23
Slika 8.6. Dizajn osobnog web sjedišta za male ekrane 25
Slika 8.7. Dizajn osobnog web sjedišta za velike ekrane 26
Slika 8.8. Site map osobnog web sjedišta 27
Slika 9.1. Kod „ProgressiveImage“ komponente 29
Slika 10.1. Slika manje kvalitete (lijevo) i slika u originalnoj kvaliteti (desno) 32

U Varaždinu, 20.11.2018. B. Brekalo

Sveučilište Sjever
Sveučilišni centar Varaždin
104. brigade 3, HR–42000 Varaždin

UNIVERSITY NORTH

Prijava završnog rada

Definiranje teme završnog rada i povjerenstva

ODJEL: Odjel za multimediju, oblikovanje i primjenu

PRISTUPNIK: Bruna Brakus | MATIČNI BROJ: 0321/336

DATUM: 20.09.2018 | KOLEGIJ: Web dizajn

NASLOV RADA: Dizajn i razvoj osobnog web sjedišta primjenom modernih web tehnologija i alata

NASLOV RADA NA ENGL. JEZIKU: Design and development of personal website using modern web technologies and tools

MENTOR: dr.sc. Mario Tomiša | ZVANJE: Izvanredni profesor

ČLANOVI POVJERENSTVA:

1. doc.dr.sc. Dean Valdec - predsjednik
2. doc.art. Robert Geček - član
3. izv.prof.dr.sc. Mario Tomiša - mentor
4. mr.sc. Vladimir Stanisavljević, v.pred. - zamjenski član
5.

VŽ KC

MMI

Zadatak završnog rada

BROJ: 602/MM/2018

OPIS

Zadatak završnog rada je prikazati razvoj osobnog web sjedišta primjenom metodologije koja podrazumijeva implementaciju responzivnih i progresivnih funkcionalnosti pomoću modernih web tehnologija i alata. Moderna programska rješenja sve se više razvijaju u obliku web aplikacija, a sve manje kao tzv. nativne aplikacije (engl. native apps) namijenjene određenom operacijskom sustavu. Razlog tome ogleda se u dostupnosti web aplikacija izravno kroz web preglednik, pri čemu su jedini ograničavajući čimbenici brzina internet veze, hardverske performanse uređaja i mogućnosti web preglednika. Ključne karakteristike progresivnih web aplikacija ogledaju se u responzivnosti korisničkog sučelja i raznim tehnikama realizacije korisničkog iskustva u web okruženju koje je vrlo slično onome pri korištenju nativne aplikacije. U završnom radu potrebno je prikazati izradu osobnog web sjedišta koje prema tehničkim karakteristikama zadovoljava preporuke za razvoj progresivnih web aplikacija, navedene u okviru "Google Developers – Progressive Web Apps" web sjedišta (https://developers.google.com/web/progressive-web-apps). U procesu realizacije praktičnog zadatka potrebno je koristiti tehnologije otvorenog koda, odnosno HTML i CSS opisne jezike, JavaScript programski jezik te pomoćne alate poput "Preact" radnog okvira, "Webpack" "bundlera" JavaScript modula, "Babel" prevoditelja JavaScript koda, "PostCSS" prevoditelja CSS koda i SASS CSS "preprocesora".

U radu je potrebno:
- definirati pojam responzivnog web dizajna te progresivne web aplikacije u modernom web okruženju
- odrediti i objasniti svrhu i cilj razvoja osobnog web sjedišta u obliku progresivne web aplikacije, prema načelima responzivnog web dizajna
- odrediti i opisati relevantne web tehnologije i alate otvorenog koda, potrebne za realizaciju planiranog web sjedišta
- prikazati postupak praktične realizacije planiranog web sjedišta
- izvesti zaključak rada s naglaskom na prednostima i nedostacima prikazane metodologije te mogućnostima daljnjeg unaprjeđenja funkcionalnosti realiziranog web sjedišta

ZADATAK URUČEN: 12. 10. 2018. | POTPIS MENTORA:

REPUBLIKA HRVATSKA KOMORA SVEUČILIŠTE SJEVER

SVEUČILIŠTE SJEVER

Sveučilište
Sjever

SVEUČILIŠTE
SJEVER

IZJAVA O AUTORSTVU
I
SUGLASNOST ZA JAVNU OBJAVU

Završni/diplomski rad isključivo je autorsko djelo studenta koji je isti izradio te student odgovara za istinitost, izvornost i ispravnost teksta rada. U radu se ne smiju koristiti dijelovi tuđih radova (knjiga, članaka, doktorskih disertacija, magistarskih radova, izvora s interneta, i drugih izvora) bez navođenja izvora i autora navedenih radova. Svi dijelovi tudih radova moraju biti pravilno navedeni i citirani. Dijelovi tuđih radova koji nisu pravilno citirani, smatraju se plagijatom, odnosno nezakonitim prisvajanjem tuđeg znanstvenog ili stručnoga rada. Sukladno navedenom studenti su dužni potpisati izjavu o autorstvu rada.

Ja, BRUNA BRAKUS *(ime i prezime)* pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor/ica završnog/diplomskog *(obrisati nepotrebno)* rada pod naslovom DIZAJN I RAZVOJ OSOBNOG WEB SJEDIŠTA PRIMJENOM MODERNIH WEB TEHNOLOGIJA I ALATA *(upisati naslov)* te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student/ica:
(upisati ime i prezime)

B. Brakus
(vlastoručni potpis)

Sukladno Zakonu o znanstvenoj djelatnost i visokom obrazovanju završne/diplomske radove sveučilišta su dužna trajno objaviti na javnoj internetskoj bazi sveučilišne knjižnice u sastavu sveučilišta te kopirati u javnu internetsku bazu završnih/diplomskih radova Nacionalne i sveučilišne knjižnice. Završni radovi istovrsnih umjetničkih studija koji se realiziraju kroz umjetnička ostvarenja objavljuju se na odgovarajući način.

Ja, BRUNA BRAKUS *(ime i prezime)* neopozivo izjavljujem da sam suglasan/na s javnom objavom završnog/diplomskog *(obrisati nepotrebno)* rada pod naslovom DIZAJN I RAZVOJ OSOBNOG WEB SJEDIŠTA PRIMJENOM MODERNIH WEB TEHNOLOGIJA I ALATA *(upisati naslov)* čiji sam autor/ica.

Student/ica:
(upisati ime i prezime)

B. Brakus
(vlastoručni potpis)