UNIVERZA V MARIBORU

FAKULTETA ZA ELEKTROTEHNIKO,

RAČUNALNIŠTVO IN INFORMATIKO

Demijan Lesjak

UPORABA PROGRESIVNIH SPLETNIH APLIKACIJ NA MOBILNIH NAPRAVAH

Magistrsko delo

Maribor, marec 2021

UPORABA PROGRESIVNIH SPLETNIH APLIKACIJ NA MOBILNIH NAPRAVAH

Magistrsko delo

Študent:	Demijan Lesjak
Študijski program:	Študijski program 2. stopnje,
	Informatika in tehnologije komuniciranja
Mentor:	doc. dr. Domen Verber
Lektorica:	lekt. dr. Mateja Gaber

ZAHVALA

Zahvala gre družini, ki mi je omogočila študij in me vsa ta leta zelo potrpežljivo spodbujala.

Prav tako se zahvaljujem svoji punci Evi, ki me je ves čas izdelave podpirala.

Zahvaljujem se tudi mentorju doc. dr. Domnu Verberju za vse koristne nasvete in odlično usmerjanje skozi celoten proces.

Uporaba progresivnih spletnih aplikacij na mobilnih napravah

Ključne besede: progresivne spletne aplikacije, PSA, storitveni delavec, spletni manifest, predpomnilnik

UDK: 004.777(043.2)

Povzetek

V nalogi smo predstavili posebnosti progresivnih spletnih aplikacij PSA (Progressive Web Apps – PWA). Podrobno smo raziskali podporne tehnologije, ki omogočajo njihov obstoj, kot sta »spletni manifest« in skripta »storitveni delavec«. Opisali smo življenjski cikel storitvenega delavca in predstavili več dobrih praks predpomnjenja spletnih virov.

V nadaljevanju smo s pridobljenim znanjem v celoti načrtovali in implementirali lastno PSA, ki je mobilni nadomestek že obstoječi platformi. Uporabili smo več modernih vmesnikov in podrobno opisali njihovo implementacijo. Nato smo s pomočjo analitičnih podatkov o uporabi, zbranih v realnem okolju, interpretirali uspešnost aplikacije v prvih tednih obstoja. V zaključnem delu smo s komparativno metodo primerjali še delovanje aplikacije v različnih brskalnikih.

Use of progressive web applications on mobile devices

Keywords: progressive web apps, PWA, service worker, web manifest, cache

UDC: 004.777(043.2)

Abstract

The thesis describes the Progressive Web Apps (PWA). Supportive technologies like for example the “web manifest” and the script “service worker” were researched in detail. We described the life cycle of the service worker and we presented multiple good methods of caching online sources.

In continuation we used the acquired knowledge to completely plan and implement our own PWA, which is a mobile substitute for an existing platform. We used various modern interfaces and we described their implementation in detail. Then we used analytical usage data, which was collected in a real environment so that we could interpret the successfulness of the app in its first weeks of existence. In the concluding part we used the comparative method to analyse the activity of the app in different web browsers.

KAZALO VSEBINE

KAZALO SLIK

KAZALO TABEL

KAZALO IZSEKOV KODE

UPORABLJENE KRATICE V ANGLEŠČINI:

PWA – Progressive Web App

API – Application Programming Interface

HTTPS – HyperText Transfer Protocol Secure

TLS – Transport Layer Security

URL – Uniform Resource Locator

USB – Universal Serial Bus

W3C - World Wide Web Consortium

JSON - JavaScript Object Notation

MIME - Multipurpose Internet Mail Extensions

DOM - Document Object Model

APK - Android Package Kit

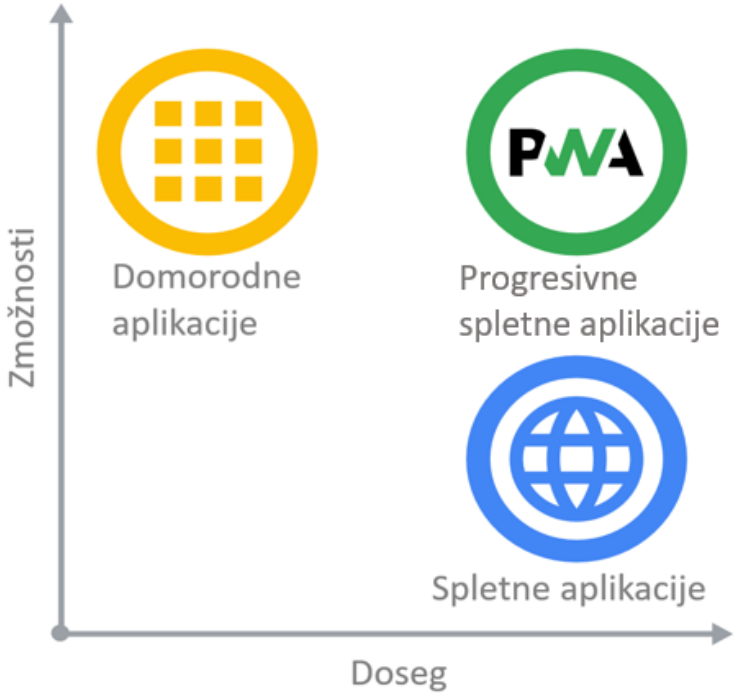
1 UVOD

Dandanes je svetovni splet na voljo na vsakem koraku. Dostop do interneta preko mobilnih telefonov je že prehitel promet z osebnih računalnikov. Za sodobno spletno stran se tako že v osnovi pričakuje, da je prilagojena velikosti zaslona mobilnih naprav. Prav tako se spreminja uporabniška izkušnja spletnih strani. Veliko strani je zgrajenih predvsem s poudarkom na mobilnih napravah in interakcija z njimi poteka na podoben način, kot so uporabniki navajeni iz ostalih, tako imenovanih »domorodnih« aplikacij.

Za domorodne aplikacije predpostavljamo, da so izjemno zanesljive in prinašajo bogato uporabniško izkušnjo. Delujejo lahko brez povezave in se vedno odprejo v samostojnem načinu. Dostopajo lahko do datotečnega sistema, upravljajo naprave preko USB (Universal Serial Bus) ali Bluetooth povezav. Prav tako lahko dostopajo do podatkov na napravi, kot so seznam stikov ali dogodki v koledarju. Zaradi neposrednega dostopa do strojne opreme lahko učinkovito poganjajo zahtevnejše računalniške algoritme. Te aplikacije dajo občutek, da so del naprave, na kateri se izvajajo.

Kljub temu pa nimajo vseprisotnosti, ki jo ponuja svetovni splet. Domorodne aplikacije so dostopne le preko trgovin aplikacij v operacijskem sistemu, za katerega so razvite, pa najsi gre pri tem za Apple, Android ali Microsoftov ekosistem. Po drugi strani pa lahko spletne strani dosežejo vsakogar ne glede na napravo ali operacijski sistem. Zelo enostavno jih je najti preko spletnih iskalnikov. Zaradi svoje povezljivosti jih je mogoče v trenutku deliti s komerkoli preko spletnega URL (Uniform Resource Locator) naslova. [1]

Tukaj vskočijo progresivne spletne aplikacije (v nadaljevanju PSA; ang. Progressive Web Apps). Kot je razvidno na Sliki 1, združujejo PSA lastnosti obeh svetov. Ne samo, da je splet postal enakovredna možnost za razvoj mobilnih aplikacij, mnogokrat so te celo boljše. Seveda bodo še nekaj časa obstajali robni primeri, kjer bodo domorodne aplikacije absolutno dominirale, ampak to število se zmanjša vsakokrat, ko brskalnik doda nov vmesnik. [2]



Slika 1 – Umestitev PSA med domorodne in spletne aplikacije *[1]*

Cilj naloge je bil raziskati ključne tehnologije PSA in jih nato uporabiti na praktičnem primeru. Z uporabo teh tehnologij smo želeli izboljšati mobilno uporabniško izkušnjo že obstoječega sistema. Prav tako nas je zanimalo, kako podprte so te tehnologije med platformami in med različnimi brskalniki in katere so pomanjkljivosti takšnih aplikacij.

V grobem je naloga razdeljena na tri dele. V prvem delu naloge predstavimo, kaj so PSA, in predstavimo njihove prednosti. Nato sledijo poglavja bolj tehnične narave, kjer preučimo ključne tehnologije, kot sta na primer spletni manifest in storitveni delavec.

Drugi del opisuje razvoj PSA. Začetna poglavja drugega dela opisujejo aplikacijo kot tako in načrtovanje izvedbe, potem pa se dotaknemo specifičnih funkcionalnosti, ki se povezujejo s temo pričujoče naloge.

V zaključnem delu pregledamo rezultate uporabe PSA po prvih dveh tednih uporabe resničnih uporabnikov. Prav tako predstavimo podprtost PSA v različnih brskalnikih in zaznane probleme ter pomanjkljivosti.

2 KAJ SO PROGRESIVNE SPLETNE APLIKACIJE?

2.1 Začetki PSA

Skovanko progresivne spletne aplikacije (Progressive Web Apps) sta sprva uporabila člana ekipe Google Chrome Alex Russel in Frances Berriman leta 2015. Russel je v svojem zapisu istega leta postavil temelje takšnih aplikacij in identificiral njihove ključne lastnosti.

Prilagodljivost: stran mora biti dovolj prilagodljiva za prikaz na kakršnemkoli zaslonu.

Neodvisnost od povezave: opremljena mora biti s skripto storitvenega delavca, ki omogoča delovanje tudi takrat, ko ni na voljo internetne povezave.

Izgled kot aplikacija: uporabniki so navajeni na izgled mobilnih aplikacij. Oblikovanje progresivne aplikacije naj sledi mobilnim vzorcem. To se odraža predvsem pri navigaciji po aplikaciji in uporabniški interakciji.

Stalna posodobljenost: transparentno posodabljanje s pomočjo storitvenega delavca. Ker aplikacija ni ločeno zapakirana kot domorodne aplikacije, uporabnik za posodobitev ne potrebuje celotnega prenašanja aplikacijskega paketa. Ko storitveni delavec zazna, da se je katera izmed datoteka spremenila, jo lahko takoj zamenja z novejšo različico. Izkazalo se je, da je najbolje, če o možnosti posodobitve opozorimo uporabnika in posodobitev izvedemo šele na njegovo željo ali ob naslednjem odpiranju aplikacije.

Varnost: aplikacija mora biti dostopna in ponujena preko varne povezave zaščitene s protokolom TLS (Transport Layer Security). Ta preprečuje kakršnekoli neavtorizirane vdore do podatkov na poti med klientom in serverjem.

Zmožnost odkritja z iskalnikom: kljub temu, da se PSA identificirajo kot aplikacije, so v osnovi to še vedno spletne strani, ki živijo na spletu. To daje možnost spletnim pajkom, da te strani indeksirajo in se kasneje lahko pojavijo kot organski zadetki v spletnih iskalnikih.

Ponovna interakcija: storitveni delavec omogoča implementacijo potisnih sporočil, ki preko uporabniškega vmesnika posameznega operacijskega sistema doprinesejo h kasnejši ponovni angažiranosti uporabnika.

Zmožnost namestitve: uporabnik ima možnost , da takšno aplikacijo doda na domač zaslon svoje naprave brez uporabe trgovine z aplikacijami. Naslednjič lahko do aplikacije dostopa neposredno preko ikone brez uporabe brskalnika.

Lasten spletni naslov (URL): pomeni, da do aplikacije zlahka dostopamo, za uporabo ni potrebna namestitev in jo tudi zlahka delimo preko hiperpovezave.

Prav tako je Russel izpostavil: »Te [progresivne] aplikacije niso zapakirane in naložene preko trgovin, to so le spletne strani, ki so vzele pravilne vitamine«. [3]

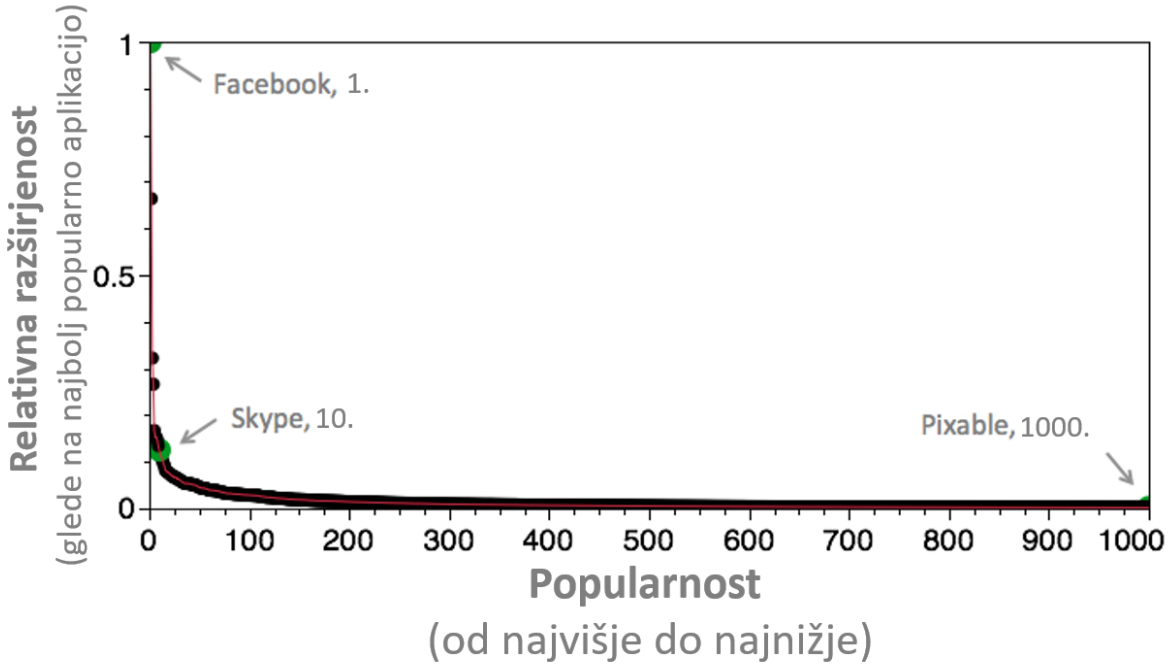
Prepad v zmožnostih med spletnimi in domorodnimi aplikacijami še vedno obstaja. Da bi to premostili, so se povezali trije giganti: Google, Microsoft in Intel. Z ambicioznim projektom z imenom Fugu želijo spletu približati vse funkcionalnosti, ki so še preostale, da bi se lahko v celoti kosale z domorodnimi aplikacijami. V ospredju razvoja je varnost in privatnost uporabnikov, saj dostop do vmesnikov naprave prinaša veliko prednost, a hkrati odpira nove možnosti za neželene spletne napade. [4]

2.2 Zakaj potrebujemo PSA?

V svetu pametnih telefonov se zdi, da za vsak uporabniški problem obstaja ustrezna aplikacija. Uporabniki so se v letih od predstavitve prvega pametnega telefona navadili na model, da za vsak problem obstaja aplikacija v trgovini aplikacij. In ko omenimo aplikacije, ponavadi pomislimo na domorodne aplikacije, posebej razvite za specifičen operacijski sistem ali celo napravo. Te aplikacije so še vedno glavni način, kako uporabniki konzumirajo vsebino. Še vedno obstaja splošno prepričanje, da so te aplikacije boljše kot spletna stran v brskalniku. Seveda takšno prepričanje izvira iz časov, ko spletni brskalniki niso bili konkurenčni zmogljivostim aplikacij. Danes to ni več tako. Velik procent aplikacij v trgovinah aplikacij bi lahko implementirali in izvajali v brskalniku. Domorodna aplikacija ostaja predvsem prestiž.

Seveda bodo še vedno obstajali tipi domorodnih aplikacij, ki jih nima smisla implementirati na spletu. Govorimo o igrah, grafičnih urejevalnikih in vseh kompleksnih programih, ki potrebujejo zmogljivost naprave za poganjanje algoritmov.

Izstopiti iz povprečja in lansirati dobičkonosno aplikacijo pa postaja vse težje, kajti trg aplikacij je postal izjemno zasičen. Že leta 2015 so v podjetju Branch ugotavljali, da je v ekosistemu aplikacij ekstremno neenakomerna porazdelitev obiskanosti in s tem posledično tudi kapitala.



Slika 2 – Neenakomerna porazdelitev prvih 1000 aplikacij

V vzorec so umestili 1000 najbolj obiskanih aplikacij, ki v trgovini aplikacij iOS ne spadajo v kategorijo iger. Slika 2 nazorno prikaže stanje trga. Skype, deseta najbolj priljubljena aplikacija v trgovini, uživa le dobro desetino prometa, ki ga ima Facebook. Tudi od desetega mesta naprej ta delež strmo pada in se ustali na zelo nizki vrednosti za večino aplikacij. Pixable na tisočem mestu, ima le še 0.2 % Facebookovega deleža prometa. Trenutno je v Applovi trgovini okrog 1.9 milijona aplikacij; v povprečju jih dodajo približno 40 tisoč novih na mesec. Verjetnost, da se bo katera izmed teh aplikacij kdaj povzpela med prvih tisoč, je zanemarljiva. [5]

Podatki o monetizaciji prav tako kažejo na monopol velikih. Pri podjetju Activate so za podobno obdobje ugotovili, da 20 največjih izdajateljev v ekosistemu iOS zasluži 48 % vsega kapitala, v ekosistemu Android pa celo 64 % zaslužka. To postavlja sto tisoče ostalih razvijalcev v nezavidljiv položaj.

Kljub skokovitemu naraščanju števila aplikacij pa statistika kaže, da se mesečno število prenesenih aplikacij povprečnega uporabnika ne povečuje. Prav tako povprečno število aplikacij, ki jih mesečno uporabljamo, ne narašča, ampak je v zadnjih letih malce padlo. Uporabniki v letu 2019 so povprečno uporabljali samo 20.1 različnih aplikacij na mesec. [6]

Razlog, da se aplikacije tako težko prebijejo na vrh seznama, je v zasnovi trgovin. Zlasti na vrhu seznamov aplikacij je zelo težko, da bi se pojavilo nekaj novega, saj seznami že sami po sebi predlagajo najbolj popularne aplikacije. To pa le še stopnjuje prepad med vrhom in vsemi ostalimi. K uporabi aplikacij pripomore tudi dobra ocena uporabnikov, vendar se je za to treba še posebej truditi in spodbujati svoje uporabnike, da oddajo pozitivno oceno. V naravi uporabnikov je, da pustijo oceno le, kadar naletijo na težave ali so nezadovoljni, medtem ko je zadovoljnega uporabnika potrebno pozivati k oddaji ocene. To pa dodatno škodi izpostavljenosti v trgovini.

Ob poplavi aplikacij po podatkih strani Statista, več kot polovica (51 %) uporabnikov mesečno ne prenese na svojo napravo nobene nove aplikacije. [7] Uporabniki so razvili določene navade in preživijo večino mobilnega časa v peščici svojih najljubših aplikacij. Skupni čas, preživet na mobilni napravi, pa se vsako leto povečuje. Po podatkih za leto 2020 preživimo na mobilni napravi malo več kot dve uri in pol na dan; od tega kar 90 % časa uporabljamo aplikacije. [8] Dejstvo je, da so uporabniki navajeni na izgled in uporabo aplikacij ter v večji meri preferirajo konzumiranje informacij na tak način kot pa uporabo spletnih strani v brskalniku.

Ne glede na to, da je le desetina mobilnega časa namenjenega brskanju po svetovnem spletu, pa uporabniki v tem kratkem času obiščejo relativno veliko število različnih spletnih strani. Navadno niso ciljno usmerjeni na določeno stran, ampak sledijo zadetkom iskalnika – odlična priložnost za progresivne aplikacije.

2.3 Prednosti PSA

Progresivne spletne aplikacije (PSA) ne predstavljajo neke specifične tehnologije. Gre bolj za koncept, ki povezuje uporabo več sodobnih spletnih tehnologij s težnjo po zagotavljanju čim

boljše uporabniške izkušnje. Inženirji pri Googlu so tako identificirali bistvene lastnosti dobre spletne aplikacije:

Hitra – hiter prikaz vsebine, gladke animacije in hiter odziv na uporabniške vnose.

Integrirana – izkušnja na spletu mora izgledati kot interakcije z vmesniki izven brskalnika.

Zanesljiva – izkušnja mora biti podobna tako na slabi povezavi kot tudi brez nje.

Vključujoča, privlačna, interaktivna – uporaba pravih vmesnikov na pravi način. Potisna sporočila, funkcionalno oblikovan uporabniški vmesnik in upravičena uporaba API-jev, ki zahtevajo dovoljenje uporabnika.

Vse te lastnosti je možno pokriti v progresivni spletni aplikaciji. Vendar zgolj uporaba tehnologij še ne prinaša rezultatov. Današnji splet premore hitre, privlačne in zelo zanesljive spletne strani, obstajajo pa tudi takšne, ki uporabnike odvračajo od uporabe le-teh. Za to pa ni kriva tehnologija, temveč neupoštevanje dobrih praks razvoja.

2.3.1 Izgled kot aplikacija

Kot smo spoznali v prejšnjem poglavju, sta izgled in poglobljena izkušnja domorodne aplikacije uporabnikom ljubša kot pa dostopanje do vsebine preko spletnega brskalnika. Kaj pa se zgodi, če spletna stran izstopi iz brskalnika?

To je pravzaprav glavna lastnost, ki loči spletno stran od progresivne spletne aplikacije. PSA lahko uporabnik doda na domač zaslon. Od takrat naprej se PSA obnaša kot prvorazredni član domorodnih aplikacij. Ikona se pojavi med ostalimi ikonami domorodnih aplikacij in pridobi njihove lastnosti. Lahko jo upravljamo, premikamo, pojavi se v iskalniku aplikacij. Ob zagonu se odpre ločeno od brskalnika čez celoten zaslon in s svojim prikaznim zaslonom. Prav tako ima svoje mesto v zaganjalniku aplikacij, kjer se lahko njeno izvajanje prekine, in v nastavitvah aplikacij, kjer imamo na voljo podatke o zasedenem prostoru, obvestilih in dovoljenjih. Ko PSA izstopi iz zavihka v brskalniku in se prelevi v polnozaslonski način aplikacije, jo uporabniki začnejo dojemati kot domorodno aplikacijo.

2.3.2 Delovanje brez povezave

Da pa bi uporabniki PSA dojemali kot enakovredno z ostalimi aplikacijami, mora delovati brez povezave. To je namreč privzeta lastnost aplikacij, ki so prenesene iz trgovin aplikacij. Za to skrbi skripta storitveni delavec. Deluje kot posrednik med klientom in serverjem, kar izjemno izboljša hitrost nalaganja. Ne deluje pa takrat, kadar je PSA v polnozaslonskem načinu. Skripta je lahko dodana tudi na navadno spletno stran in tako izboljša uporabniško izkušnjo v primerih, ko uporabnik ostane brez stabilne povezave.

2.3.3 Prisotnost v brskalnikih

V nasprotju z domorodnimi aplikacijami PSA ne potrebujejo posebne platforme, preko katere uporabniki odkrijejo in namestijo želene aplikacije. Ker so PSA v osnovi spletne strani, posedujejo vse prednosti svetovnega spleta. Spletne strani so bile vedno enostavno dostopne preko spletnih brskalnikov. Prav tako lahko domeno oglašujemo na raznovrstnih kanalih in večini ljudem je samoumevno, kako do nje dostopati. PSA so tako dostopne spletnim brskalnikom, ki so še vedno veliko bolj izpopolnjeni kot iskalniki v trgovinah aplikacij. Velika prednost je tudi možnost globokega indeksiranja vsebine v PSA. Spletni pajki indeksirajo vsebino ogromno podstrani na isti domeni. Vsaka ta stran se lahko pojavi v zadetkih iskanja in postane potencialna vstopna točka v PSA. Drugače je z ekosistemom trgovin aplikacij, kjer je edina vstopna točka v aplikacijo preko predstavitvene strani s kratkim opisom same aplikacije. [2]

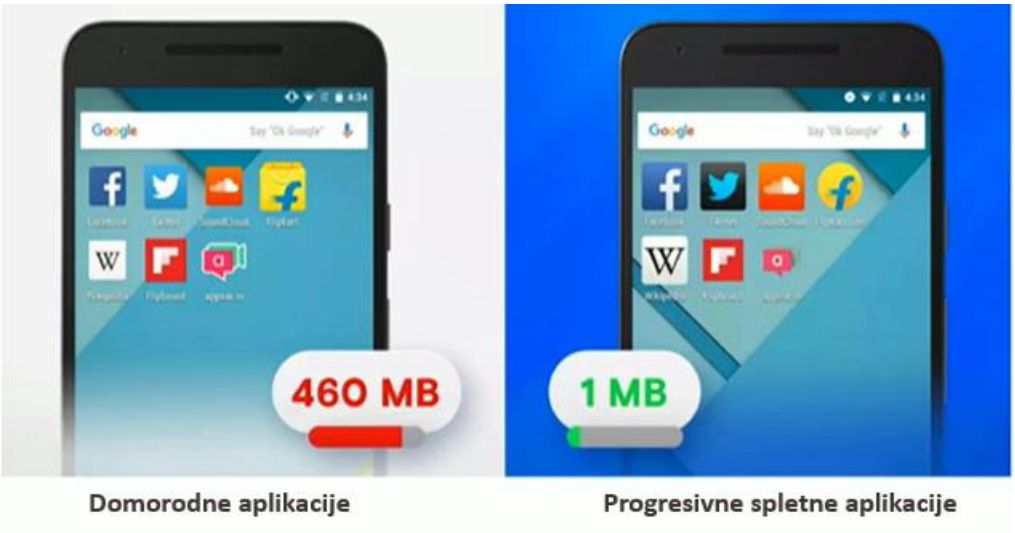
Dostopnost preko spletne povezave pa omogoča tudi enostavnost deljenja aplikacije, saj lahko uporabniki to storijo le s posredovanjem URL naslova.

2.3.4 Prihranek v porabljenih podatkih

V državah v razvoju, kjer so povezave do interneta šibkejše in je cena prenesenih podatkov precej visoka, so uporabniki veliko bolj pozorni na nepotrebno koriščenje mobilnih podatkov. Prav tako pa povprečna mobilna naprava v teh državah premore znatno manj prostora za pomnjenje podatkov. Posledično so ti uporabniki precej bolj izbirčni pri izbiri aplikacije. Za

nepreverjeno aplikacijo ne bodo žrtvovali veliko podatkov ali prostora na napravi. Prav tako raje odlašajo z večjimi posodobitvami, dokler ne pridejo na izbrano Wi-Fi točko. Takšne težave opažajo tudi pri tehnoloških gigantih, kot je na primer Facebook. Kljub že takrat uspešni mobilni aplikaciji, ki je med najbolj prenesenimi aplikacijami na svetu, so se pri Facebooku odločili ustvariti poenostavljeno verzijo, namenjeno tem trgom. Njihov cilj je bil, da namestitveni paket ne presega 1 MB. V svoji raziskavi so namreč ugotovili, da se povprečna aplikacija velikosti približno 20 MB na 2G omrežju lahko nalaga več kot 30 minut. Zaradi nestabilnih povezav pa prenos velikokrat tudi spodleti pred zaključkom. Aplikacija Facebook Lite je bila najhitrejša verzija platforme Facebook, ki je dosegla 100 milijonov uporabnikov – in to v samo devetih mesecih! Danes ima preko milijardo prenosov in še vedno zaseda le 1.5 MB. Aplikacija je najbolj popularna v državah, kot so: Brazilija, Indija, Indonezija, Mehika in Filipini, kjer je velikost prenesenega paketa še vedno zelo pomembna. [9]

Kljub temu pa so to zelo pomembni in ogromni trgi in tega se zavedajo velika podjetja. To je eden izmed razlogov, da na splet prihaja vedno več progresivnih aplikacij od podjetij, ki imajo že razvite uspešne domorodne aplikacije. Progresivna aplikacija je v osnovi le spletna stran in ob namestitvi porabi bistveno manj podatkov in prostora. Na Sliki 3 je prikazana količina potrebnega prostora za namestitev nekaj domorodnih aplikacij v primerjavi z njihovimi progresivnimi nadomestitvami.



Slika 3 – Primerjava zasedenega prostora med domorodnimi in progresivnimi aplikacijami [10]

Kot vidimo, so namestitveni paketi izbranih sedmih aplikacij skupaj veliki 460 MB. Prav tako pa je osnovni zasedeni prostor vseh sedmih progresivnih aplikacij le 1 MB. Tako znaten prihranek prostora lahko na slabših napravah naredi veliko razliko.

3 TEMELJNE TEHNOLOGIJE

3.1 Minimalne zahteve PSA

PSA je spletna stran z dodanimi modernimi tehnologijami in vmesniki, ki nadgradijo funkcionalnosti strani. Kljub temu obstajajo minimalne zahteve, ki jih mora izpolnjevati vsaka spletna stran, da jo brskalniki prepoznajo kot PSA.

Stran mora biti servirana preko varne HTTPS (HyperText Transfer Protocol Secure) povezave. Priporočilo za uporabo varne šifrirane povezave velja tudi za vse ostale spletne strani, za PSA pa je to pogoj, saj so v uporabi precej zmogljivi vmesniki in bi bilo tveganje zlorabe preveliko.

Stran mora registrirati storitvenega delavca. Ta skripta skrbi predvsem za hranjenje virov aplikacije v pomnilniku brskalnika in za nemoteno delovanje aplikacije tudi v načinu brez internetne povezave.

Stran mora vsebovati dokument spletni manifest. Je ključna datoteka za namestitev aplikacije na domač zaslon. Vsebuje vse informacije, ki jih potrebuje operacijski sistem, da pravilno prikaže PSA.

3.2 Spletni manifest

Spletni manifest je dokument, osnovan na podlagi JSON (JavaScript Object Notation) formata. Manifest omogoča razvijalcem centraliziran prostor za meta podatke spletne aplikacije. Polja manifesta so opcijska, prav tako seznam možnih atributov ni končen, ampak se ti občasno še vedno dodajajo. Najpogostejše uporabljeni atributi predstavljajo informacije o izgledu in obnašanju spletne aplikacije ob namestitvi na domač zaslon naprave.

Uradna končnica dokumenta je *.webmanifest,* vendar večina spletnih strežnikov prenaša dokument z uporabo standardnega tipa MIME (Multipurpose Internet Mail Extensions): *application/manifest+json*. Posledično to dovoljuje razvijalcem uporabo končnice *.json*. [11]

Manifest sam po sebi ne zagotavlja posebnega namestitvenega vmesnika, ampak lahko služi kot eden izmed potrebnih signalov brskalniku, da se spletna aplikacija sploh lahko namesti.

Da pa se ta signal ustrezno sproži, mora manifest vsebovati vsaj naslednje atribute: *»name«, »display«, »icons«, »start_url«* in *»background_color«.*

3.2.1 Obvezni atributi

name / short_name – To polje predstavlja ime aplikacije, ki se uporablja na različnih mestih. Ime je prikazano ob pozivu k namestitvi aplikacije, pod ikono nameščene aplikacije, v iskalniku aplikacij itd. »*name*« predstavlja polno ime aplikacije, »*short_name*« pa je skrajšana verzija za uporabo v primerih, kjer primanjkuje prostora za prikaz. Čeprav »*short_name*« ni obvezen podatek za PSA, je zelo priporočljivo, da se doda krajša verzija, zlasti kadar celotno ime obsega več besed. [12]

display – Predstavlja način, na katerega je aplikacija predstavljena znotraj konteksta operacijskega sistema. Obstajajo štiri možne vrednosti atributa »*display«*: *browser, minimal-ui, standalone* in *fullscreen.*

Fullscreen predstavlja način, kjer aplikacije prekrije ves razpoložljiv prostor na zaslonu, medtem ko pri *standalone* načinu ostane vidna le statusna vrstica operacijskega sistema. Oba načina sta si zelo podobna in prikažeta PSA v lastnem kontekstu izven brskalnika.

minimal-ui način se razlikuje po tem, da so nekateri elementi brskalnika lahko ohranjeni. Implementacija se razlikuje med različnimi brskalniki, vendar navadno ostane viden trenuten URL naslov in nekateri elementi nastavitev ali navigacije.

Če je izbran način *browser*, se aplikacija vedno odpre v kontekstu brskalnika in je ni mogoče namestiti na domač zaslon.

Če brskalnik ne podpira katere izmed vrednosti, se nastavi vrednost, ki je najnižje po hierarhiji. Če ne podpira nobene vrednosti, se nastavi *browser,* ki je vedno podprta aplikacija. [11]

icons – Vsebuje polje ikon, ki jih brskalnik uporablja na več mestih. Pojavijo se kot ikona na domačem zaslonu, v začetnem zaslonu aplikacije, v upravljalniku aplikacij. Vsaka ikona definira *src, type in* sizes. Možni so različni formati ikon, vendar je od brskalnika odvisno, ali jih podpira. Vsak vnos definira velikost, za katero je priložena velikost primerna. Operacijski sistem avtomatsko izbere ikono, ki je najprimernejša glede na definirano velikost.

Za brskalnik Chrome je potrebno pripraviti ikono velikosti vsaj 192 x 192 slikovnih pik in ikono velikosti 512 x 512 slikovnih pik. Chrome nato samodejno prilagaja priložene ikone na pravilne mere. Če želimo imeti kontrolo nad prilagajanjem velikosti, lahko priložimo vse možne velikosti v inkrementih po 48 slikovnih pik. [13]

start_url – Predstavlja lokacijo želene vstopne točke v našo aplikacijo. S tem povemo brskalniku, katero stran naj naloži ob zagonu aplikacije, in sicer ne glede na to, na kateri strani je uporabnik bil ob namestitvi aplikacije. Vrednost URL povezave mora biti definirana relativno na lokacijo samega manifesta.

background_color – Vrednost opisuje pričakovano barvo aplikacije. Brskalnik uporabi vrednost barve le za obarvanje ozadja ob zagonu, dokler se nalagajo začetne datoteke. Prav tako se uporablja kot ozadje začetnega zaslona. Atribut je predviden zgolj kot izboljšava uporabniške izkušnje, ne pa kot dejanska vrednost ozadja izbranega elementa DOM.

3.2.2 Opcijski atributi

Poleg nujno potrebnih vrednosti, ki omogočajo namestitev PSA, je v manifestu možno definirati še ostale uporabne lastnosti aplikacije.

orientation – Definira želeno orientacijo prikaza naše aplikacije. Če brskalnik upošteva ta atribut, se njegova vrednost nastavi kot osnovna smer orientacije za cel življenjski cikel aplikacije, razen če se vrednost ponastavi v času izvajanja. [14]

theme_color – Definira osnovno tematsko barvo aplikacije. To vrednost lahko operacijski sistem uporabi v asociaciji s prikazom aplikacije. Primer je obroba aplikacije in ikone v menjalniku aplikacij na platformi Android.

scope – Vsebuje URL, ki kontrolira aktiven obseg aplikacije. Če uporabnik zaide izven definiranega obsega, se pogled vrne v zavihek brskalnika. Če je definirani URL relativen, je URL osnova lokacija manifest dokumenta. [15]

related_applications – Vsebuje seznam aplikacij, ki so lahko nameščene na operacijski sistem z drugih virov in imajo povezavo s trenutno aplikacijo. Povezave, definirane v manifestu, so

enostranske. Te povezave se lahko uporabijo kot predlogi uporabniku, katere že nameščene aplikacije lahko uporabi v povezavi s spletno aplikacijo.

shortcuts – Definira polje bližnjic oziroma povezav do ključnih vsebin v aplikaciji. Te vrednosti so uporabljene v kontekstnem meniju v primeru, da ga operacijski sistem podpira.

description – Polje, ki je namenjeno opisu namena aplikacije.

3.3 Storitveni delavec

Storitveni delavec je skripta, ki jo brskalnik izvaja v ozadju in ločeno od spletne strani. Skrbi za funkcionalnosti, ki ne potrebujejo interakcije s stranjo ali uporabnikom. V bistvu predstavlja nek most med spletno aplikacijo in omrežjem. Vsebuje zmogljive vmesnike, ki omogočajo nekatere ključne funkcionalnosti PSA.

Cache API (Application Programming Interface) omogoča predpomnjenje statičnih virov strani ob prvem zagonu. To posledično pomeni učinkovito izvajanje in zmožnost delovanja aplikacije v načinu brez povezave ob naslednjih obiskih. Z uporabo *Notifications* API in *Push* API lahko dosežemo pošiljanje potisnih sporočil – to je bilo včasih mogoče le v domorodnih aplikacijah. *BackgroundSync* APi nam omogoča, da lahko preko mehanizma sinhronizacijskega dogodka oddamo zahtevek za strežnik v omrežje, četudi nimamo aktivne internetne povezave. Zahtevek se shrani in čaka v kontekstu storitvenega delavca. Kakor hitro vmesnik zazna, da se je internetna povezava ponovno vzpostavila, se zahtevek izvede.

Storitveni delavec pa ni zaključena celota. Je nek nov mehanizem, v katerega se lahko dodajajo novi vmesniki, ki izkoriščajo prednosti izvajanja v ozadju brskalnika. Zadnji tak vmesnik je *PeriodicBackgroundSync* API. Kot nam pove že ime, omogoča periodično izvajanje kode v kontekstu storitvenega delavca, tudi kadar sama stran ni v uporabi.

V prihodnosti lahko pričakujemo podporo tudi za nekatere geo-lokacijske vmesnike, kot je na primer geografsko ograjevanje. To so vmesniki, ki v ozadju periodično pridobivajo podatke o lokaciji naprave in ob prečkanju navidezne meje sprožijo določen dogodek.

3.3.1 Glavne značilnosti

Osnovna funkcionalnost, okrog katere je storitveni delavec zgrajen, je prestrezanje spletnih zahtevkov in manipuliranje z njimi. Rezultate teh zahtevkov lahko predpomnimo in naslednjič na enak zahtevek odgovorimo z rezultatom iz predpomnilnika. To pomeni, da je rezultat na voljo bistveno hitreje, kot pa če bi prišel z oddaljenega strežnika, prav tako pa to funkcionira tudi takrat, kadar internetna povezava ni na voljo. Storitveni delavec omogoča granularen nadzor nad obnašanjem mehanizmov hranjenja virov in daje razvijalcem orodje za oblikovanje uporabniške izkušnje, ki podpira način brez povezave.

Predhodnik storitvenega delavca je bil vmesnik *AppCache*. Ta vmesnik je bil veliko bolj visokonivojski in je bil široko podprt med brskalniki. Bil je zelo enostaven za uporabo, vendar je imel v temeljih več pomanjkljivosti. Dandanes je *AppCache* označen kot zastarel. Storitveni delavec na drugi strani pa je bil zasnovan na nižjem nivoju z namenom, da odstrani težave z *AppCache*-om. Seveda to pomeni tudi večjo kompleksnost in zahtevnost implementacije. [16] Storitveni delavec skripta je v bistvu oblika JavaScript spletnih delavcev. Te skripte se ne izvajajo v istem kontekstu kot glavna skripta spletne strani in njenega izvajanja ne blokirajo. Idealne so za uporabo ob izvajanju kakšnih kompleksnih procesov in tako sprostijo glavno skripto, ki lahko ostane v pripravljenosti za spremembe v grafičnem vmesniku in procesiranje dogodkov, ki jih sproži interakcija z uporabnikom. Vendar pa nimajo dostopa do prav vseh funkcionalnosti JavaScript. Tako ne morejo dostopati do DOM (Document Object Model) in do *window*, *document* ter *parent* objekta. [17]

Celotna implementacija storitvenega delavca sloni na uporabi objekta *Promise*, ki omogoča moderni način pisanja asinhrone kode v JavaScriptu. Posledica tega je, da iz skripte ni dostopa do vmesnikov, ki se izvajajo sinhrono. Tako storitveni delavec ne more dostopati do *localStorage*, *sessionStorage* in do piškotkov. Kljub temu pa ostaja dostop do podatkovne baze IndexedDb, ki je asinhrona in tako najboljša alternativa za lokalno hranjenje podatkov.

3.3.2 Registracija storitvenega delavca

Da se storitveni delavec začne izvajati, se mora v kodi spletne strani JavaScript izvesti registracija. Najbolj primeren trenutek, da se ta koda izvede, je takoj zatem, ko stran konča z

nalaganjem. Razloga, da ni smiselno registrirati storitvenega delavca, preden je stran popolnoma naložena, sta dva.

Prvi tiči v specifičnem obnašanju storitvenega delavca. Kljub temu, da se skripta uspešno namesti ob prvem ogledu strani, ta stran še ni pod njegovim nadzorom. Storitveni delavec prevzame nadzor šele ob naslednjih ogledih. To pomeni, da sploh ni pomembno, ali se registracija izvede čim hitreje, saj se to na strani ne bo poznalo.

Po drugi strani pa je za brskalnik odpiranje nove skripte ob začetku nalaganja le dodatna obremenitev. Glede na to, da težimo k optimizaciji nalaganja strani, ne bi bilo smiselno v tej fazi zaganjati še skripte, ki v ozadju prav tako izrablja dragoceno pasovno širino za predpomnjenje istih datotek. [18]

3.3.3 Obseg

Privzet obseg storitvenega delavca je relativen na lokacijo skripte. Če se registrirani storitveni delavec nahaja v korenu strukture strani, potem so v njegovem obsegu tudi vse podstrani na tej domeni. Na primer:

```
/moja-stran.si/service-worker.js
```

Če pa je skripta gnezdena globlje v strukturi, ima posledično lahko pod nadzorom le strani od tam naprej.

3.3.4 Življenjski cikel storitvenega delavca

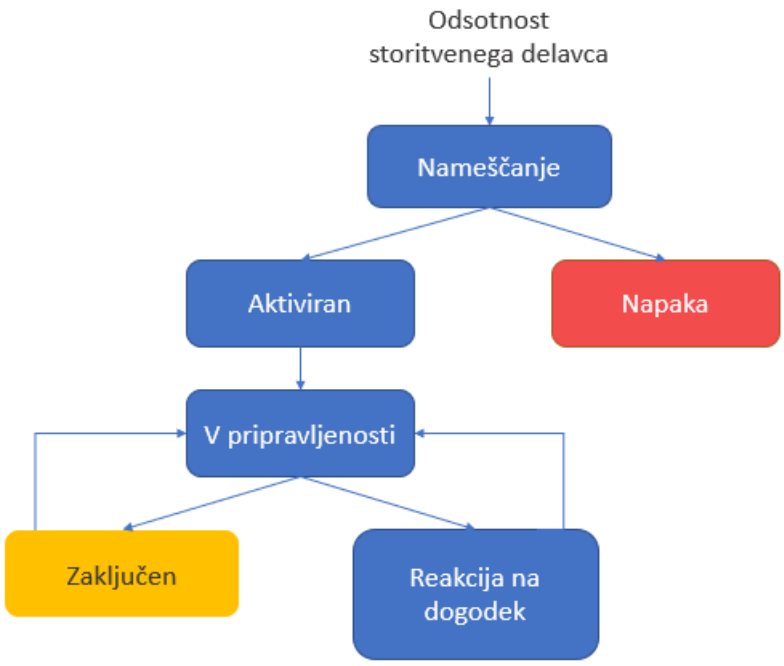
Izvajanje skripte storitvenega delavca je precej specifično. Njen življenjski cikel se ne ujema s spletno stranjo, s katere je bila sprožena. Življenjski cikel je prikazan v Sliki 4.

Za začetek namestitve storitvenega delavca je potrebna registracija v kodi spletne strani JavaScript Ob registraciji začne brskalnik neopazno v ozadju nameščati skripto. Običajno želimo v tem koraku v predpomnilnik shraniti nekatere (ali vse) statične vire.

Po uspešni namestitvi nastopi faza aktivacije. V tej fazi se izvede upravljanje z verzijami predpomnilnika. Po aktivaciji storitveni delavec pridobi nadzor nad vsemi stranmi, ki so v

njegovem obsegu. Kljub temu pa stran, ki ga je registrirala, ne bo pod kontrolo do naslednjega obiska (ali osvežitve).

Ko storitveni delavec enkrat prevzame nadzor, je ta lahko v dveh stanjih. Ali je ustavljen zaradi nedejavnosti in varčevanja s spominom ali pa je dejaven in prestreza zahtevke ter sporočila. [19]



Slika 4 – Življenjski cikel storitvenega delavca

Storitveni delavec je dogodkovno naravnan dokument. Implementira poslušalce na razne dogodke, ki jih proži ali brskalnik ali pa stran, ki ga je registrirala. Življenjski cikel pa se malenkost razlikuje glede na to, ali je to prva registrirana instanca skripte ali pa je že nameščena in se posodablja. Pri posodabljanju se pojavijo nekateri dodatni dogodki, ki preprečijo nekonsistentno upravljanje z viri v različnih zavihkih.

3.3.5 Dogodek “install”

Začetni dogodek, ki se zgodi v storitvenem delavcu, je vedno namestitveni dogodek »install«. Sproži se v trenutku po registraciji na spletni strani. Ta dogodek se zgodi samo enkrat v celem življenjskem ciklu. Ponuja idealno priložnost za predpomnjenje vseh potrebnih statičnih virov, ki jih potrebujemo za uspešen kasnejši nadzor strani.

Potem ko so vse datoteke uspešno prenesene in shranjene, storitveni delavec postane uspešno nameščen. Če se katerakoli datoteka neuspešno prenese v predpomnilnik, je namestitev neuspešna. Na izvajanje same strani pa to ne vpliva. V tem primeru se skripta poizkuša namestiti ob naslednji priložnosti. [19]

3.3.6 Dogodek »activate«

Ko je storitveni delavec enkrat uspešno nameščen, se aktivira dogodek »activate«. Vendar ta dogodek ne zagotavlja, da bo stran takoj pod okriljem skripte. Če se stran začne nalagati brez storitvenega delavca, ta ne prevzame kontrole, dokler se stran ponovno ne odpre. To obnašanje zagotavlja dosledno izvajanje strani.

Aktivacijski dogodek pa se lahko uporabi tudi za čiščenje starejših, nepotrebnih verzij predpomnilnika. To pride v poštev v primeru, če se skripta storitvenega delavca posodobi in definira drugačne verzije predpomnilnika v dogodku »install«, kot so bile doslej.

3.3.7 Posodabljanje skripte in faza čakanja

Proces posodabljanja se začne avtomatsko ob začetku nalaganja strani, ki je v obsegu, ali ob funkcijskih dogodkih, kot sta »push« ali »sync«. Brskalnik zazna, da je potrebna posodobitev, če se nova skripta vsaj za bajt razlikuje od aktivne. To preverjanje zajema tudi vse zunanje skripte, ki so vključene v storitveni delavec. Če se skripti razlikujeta, nova skripta dobi svoj dogodek »install« in prične svoj življenjski cikel vzporedno z že aktivno skripto.

Če se zgodi, da je namestitev nove skripte neuspešna, se ta skripta zavrže in trenutna ostane aktivna še naprej. Ob uspešni namestitvi pa preide nova skripta v fazo čakanja. V tej fazi ostane, vse dokler aktivna skripta nadzoruje katerokoli stran. Te strani so lahko tudi v drugih zavihkih in/ali oknih istega brskalnika. Ko so vse strani v obsegu storitvenega delavca zaključene oziroma zaprte, nova skripta zaključi fazo čakanja in prejme dogodek »activate«. Kot je bilo že omenjeno, lahko na tej točki upravljamo z verzijami predpomnilnika in pobrišemo verzije stare skripte, ki niso več aktualne. [20]

3.3.8 Strategije uporabe predpomnilnika

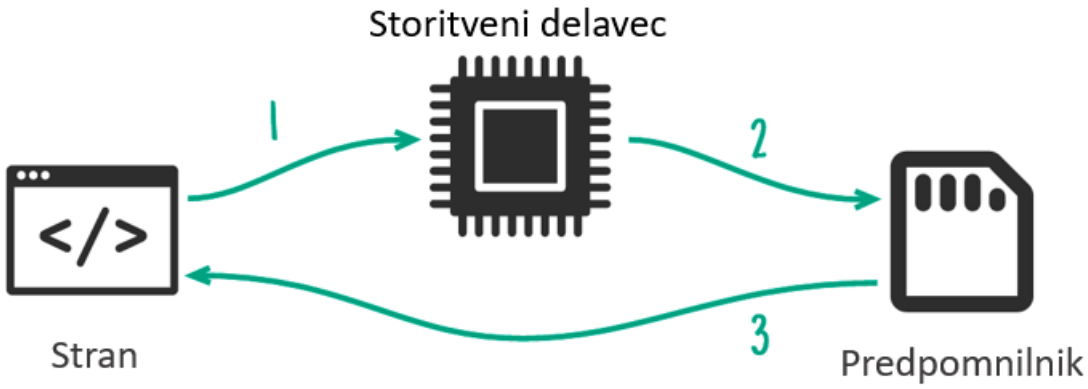
Progresivne spletne aplikacije se močno opirajo na uporabo predpomnjenja svojih virov. Čeprav je Cache API dostopen tudi izven storitvenega delavca, se večina predpomnjenja dogaja v sami skripti, saj lahko prestreza vse spletne zahtevke. Kljub temu pa se to ne izvaja generično, ampak imajo razvijalci popoln nadzor nad procesom predpomnjenja. Jake Archibald je že leta 2014 definiral osnovne vzorce, ki se pojavljajo za različne vrste virov. V praksi ti vzorci živijo v simbiozi drug z drugim ali se celo prepletajo, odvisno od problema, ki ga rešujejo.

Vire, za katere predvidevamo, da se za trenutno verzijo ne bodo spreminjali, lahko predpomnimo že v namestitvenemu dogodku »install«. Kako pa hranimo zahtevane vire, ki se skozi čas spreminjajo? Na primer vsebino strani, članke, avatarje ali časovnice socialnih omrežij? In kako jih potem serviramo?

Tukaj bomo pregledali nekaj najbolj primernih načinov pridobivanja podatkov s souporabo predpomnilnika.

Samo predpomnilnik

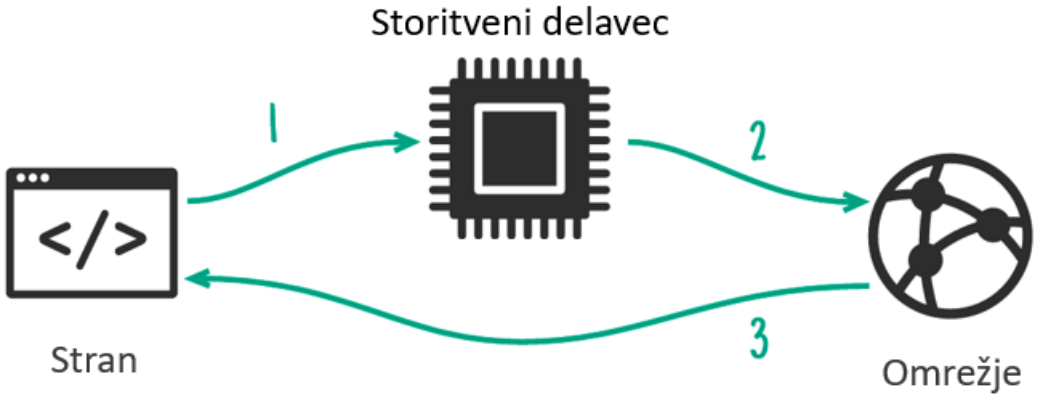
To je strategija, ki je namenjena izključno statičnim podatkom. Te podatke je priporočljivo shraniti v predpomnilnik ob namestitvi storitvenega delavca, kajti le tako se lahko zanesemo, da bodo tam, ko jih bomo potrebovali. Ta strategija je brez posebne logike. Kot je prikazano na Sliki 5, ob zahtevi s strani odpremo predpomnilnik, poiščemo zahtevano datoteko in vrnemo. Ta strategija je mišljena izključno za podatke, ki se ne bodo spreminjali za to verzijo aplikacije. Nadgradnja te strategije je vzorec »najprej predpomnilnik«.



Slika 5 – Diagram strategije "samo predpolnilnik" [21]

Samo omrežje

Ta vzorec je primeren za vse tiste zahtevke, kjer ne pričakujemo podatkov, ampak jih sami pošiljamo. To so vse zahteve, ki niso tipa GET. Tukaj se ne moremo zanašati na predpomnilnik, ki bi lahko zadeve pospešil, ampak morajo zahteve v vsakem primeru skozi omrežje, kot je razvidno na Sliki 6. To obnašanje je popolnoma enako obnašanju brskalnika brez aktivnega storitvenega delavca. Prednost njegove uporabe je, da lahko ob prečkanju zahtevka izvedemo izbran stranski učinek. Lahko na primer shranimo kakšno vrednost v IndexedDb. Ta strategija se zelo redko pojavlja, saj jo pokriva in nadgrajuje naslednja – “najprej predpomnilnik”. [21]

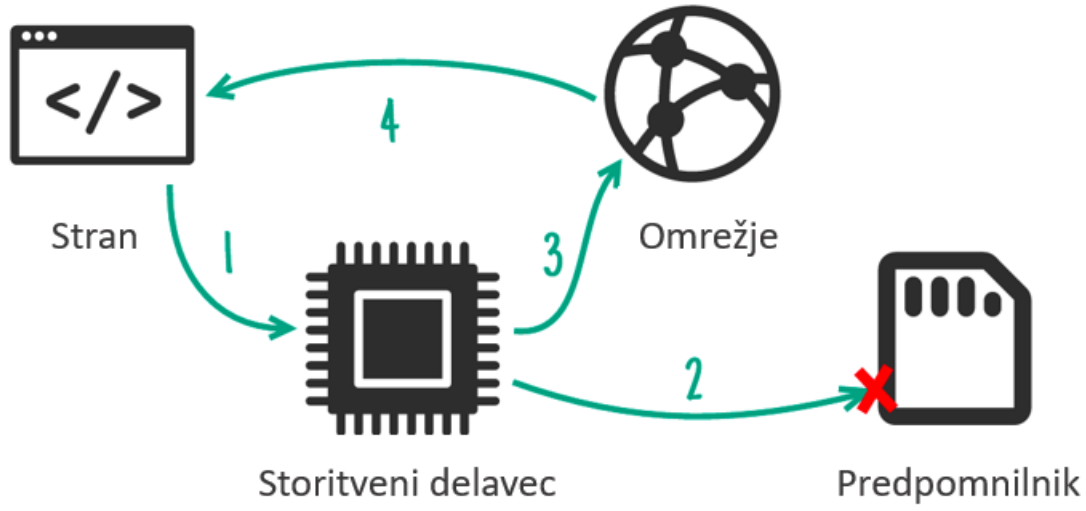


Slika 6 – Diagram strategije "samo omrežje" *[21]*

Najprej predpomnilnik

To je najbolj pogost vzorec za implementacijo aplikacij, ki delujejo brez povezave. Ob zahtevku s strani storitveni delavec najprej preveri, ali so zahtevani podatki v predpomnilniku. Šele v primeru, da v predpomnilniku ne najdemo podatkov, se obrnemo na omrežje. Ko se podatki

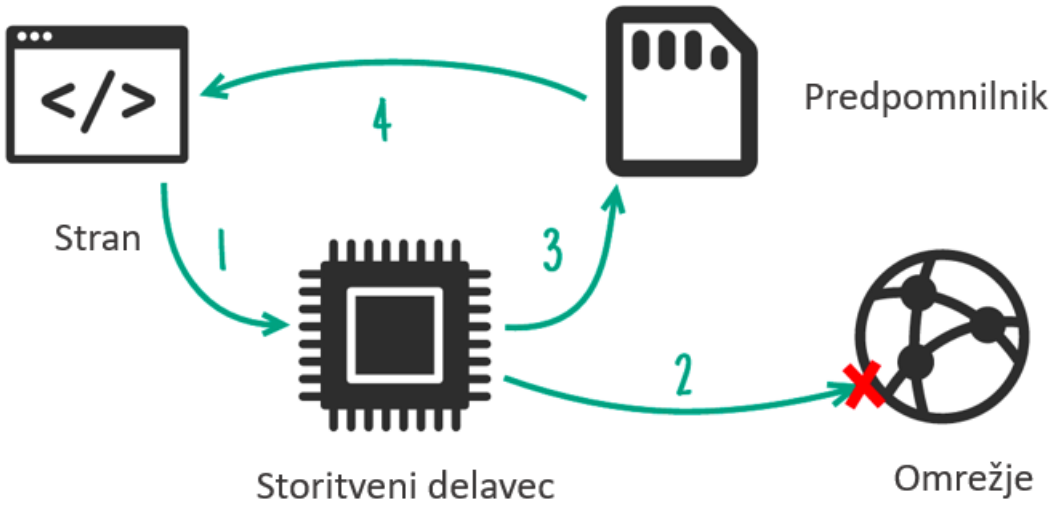
iz omrežja vrnejo, jih lahko shranimo v predpomnilnik in takoj posredujemo na stran. Tako se manjkajoči podatki v predpomnilnik postopoma dodajajo in so ob naslednjem enakem zahtevku takoj na voljo. [21]



Slika 7 – Diagram strategije "najprej predpomnilnik" *[21]*

Najprej omrežje

Ta strategija se uporabi za zahtevke virov, ki se lahko hitro spreminjajo. Za uporabnika je ključno, da pridobi čim bolj svežo verzijo teh podatkov. V tem primeru storitveni delavec najprej posreduje zahtevek naprej na omrežje in v primeru neuspeha poizkusi še v predpomnilniku. To pomeni, da so prikazani podatki vedno sveži, razen kadar je uporabnik brez internetne povezave. Takrat dobi zadnjo shranjeno verzijo podatkov. Hitrost serviranja teh podatkov je v tem primeru zelo odvisna od stabilnosti internetne povezave. Na prvi izgled se obnaša podobno kot normalno obnašanje brskalnika, le da se podatki servirajo, tudi kadar povezava izgine. Če pa je povezava precej počasna, mora uporabnik še vedno počakati, da se zahtevek uspešno vrne ali pa se sproži napaka. Šele ob ujeti napaki storitveni delavec pogleda v predpomnilnik.

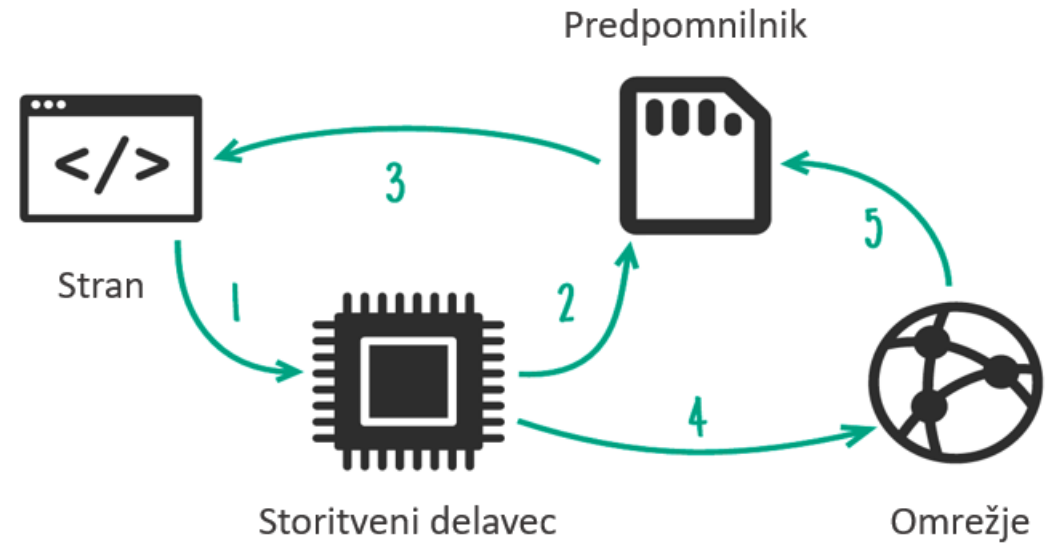


Slika 8 – Diagram strategije "najprej omrežje" [21]

Zastarel do ponovnega preverjanja

Ta vzorec se uporablja za spreminjajoče podatke, kjer ni prioriteta, da so ti podatki popolnoma sveži. Prednost je, da podatke iz predpomnilnika lahko serviramo takoj. Vendar pa ni zagotovljeno, da so ti podatki najnovejši. To se lahko uporablja denimo za avatarje ali slike artiklov.

Ko storitveni delavec prestreže zahtevo, najprej preveri predpomnilnik in v primeru zadetka zahtevo takoj vrne. Kljub temu pa posreduje zahtevek še v omrežje in z vrnjenimi podatki posodobi zapis v predpomnilniku. [21]

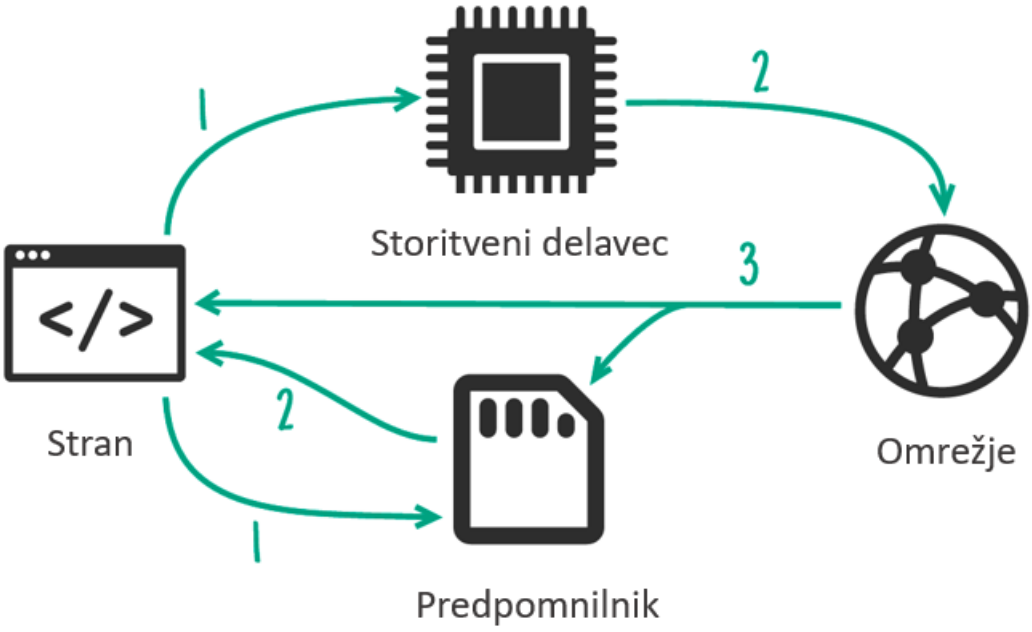


Slika 9 – Diagram strategije "zastarel do ponovnega preverjanja" *[21]*

Najprej predpomnilnik, potem omrežje

Najnaprednejša tehnika je vsekakor najprej predpomnilnik, nato pa omrežje. Odpravlja nekatere pomanjkljivosti ostalih strategij, vendar je implementacija znatno bolj zapletena. V bistvu je to nekakšna kombinacija strategij »samo predpomnilnik« in zahteve v omrežje, ki ob vračanju podatkov na stran mimogrede posodobi predpomnilnik. To rešitev pa ne moremo implementirati samo v skripti storitvenega delavca. Ideja je, da naša stran naredi dva istočasna zahtevka: enega v predpomnilnik in enega v omrežje. Pričakovano je, da bodo podatki iz predpomnilnika hitrejši. Če predpomnilnik slučajno ne uspe vrniti podatkov, se čaka na omrežje. V nasprotnem primeru se takoj prikažejo podatki iz predpomnilnika. Ko se vrnejo še podatki iz omrežja, se na strani le zamenjajo stari podatki s svežimi. [21]

Kadar pride do velikih sprememb ob zamenjavi podatkov, lahko to negativno vpliva na uporabniško izkušnjo, zlasti če se zamenjajo elementi DOM ali struktura vsebine. V takšnem primeru je potrebno ohraniti pozicijo elementov na strani ali pa prikazati obvestilo, da se bo vsebina osvežila.



Slika 10 – Diagram strategije "najprej predpomnilnik, potem omrežje" *[21]*

3.3.9 Potisna sporočila

Možnost obveščanja uporabnikov, kadar je aplikacija v ozadju ali celo čisto zaprta, je zelo dragocena. Namen takšnih sporočil je obveščanje uporabnika o kakšnem dogodku, ki zahteva njegovo pozornost ali pa pripomore k izboljšanju interakcije z aplikacijo. Sporočila morajo biti relevantna in časovno usklajena. Če uporabnik prejme veliko število sporočil, ki niso ciljno usmerjena, je velika možnost, da jih bo v celoti izklopil.

Interakcija z uporabniki preko potisnih sporočil je postal standard, ki deluje. Industrija uporablja potisna sporočila za interakcijo s svojimi uporabniki že več kot desetletje. Raziskovalci so razkrili številke, ki neposredno povezujejo to tehniko s prihodkom podjetja. Google je v svoji raziskavi prišel do naslednjih izsledkov:

- uporabniki, ki pridejo na stran preko potisnih sporočil, ostanejo na strani 72 % dlje časa
- v povprečju uporabniki, ki prispejo na stran preko potisnih sporočil, zapravijo 26 % več denarja
- možnost ponovnega obiska v naslednjih treh mesecih se poveča za 50 %.

To so le nekateri razlogi, zakaj projektni vodje obožujejo potisna sporočila. Še do pred kratkim je bil splet prikrajšan za to funkcionalnost. Veliko znamk se je odločilo za dražji in zahtevnejši razvoj lastnih domorodnih aplikacij, samo da so pridobili možnost pošiljanja potisnih sporočil. [2] Dandanes to ni več potrebno. Potisna sporočila so podprta v več brskalnikih in na različnih operacijskih sistemih. To omogočata dva komplementarna vmesnika: Notifications API in Push API. Lahko se uporabljata tudi ločeno, vendar za popolno izkušnjo potisnih sporočil potrebujemo oba. Oba vmesnika se zanašata na implementacijo skripte storitvenega delavca, ki v ozadju reagira na potisne dogodke in dogodke sporočil.

3.4 Pomoč pri razvoju

Storitveni delavec prinaša popoln nadzor nad vsemi zahtevki iz domene in omogoča razvijalcem programatično upravljanje z njimi. Vsak zahtevek ali dogodek lahko natančno prilagodimo, da ustreza našim potrebam. Čeprav se uporabljeni vzorci med različnimi primeri

začnejo ponavljati, se lahko kompleksnost kode skokovito povečuje. Obstajajo pa knjižnice, ki to kompleksnost ovijejo in poenostavijo delo s storitvenim delavcem. Najbolj razširjeno orodje za delo s storitvenim delavcem je Googlov odprtokodni projekt Workbox. Ta je na voljo za uporabo na raznih Javascript ogrodjih ali brez njih. Prav tako pa obstaja Angular storitveni delavec, ki pa je na voljo izključno razvijalcem v ogrodju Angular.

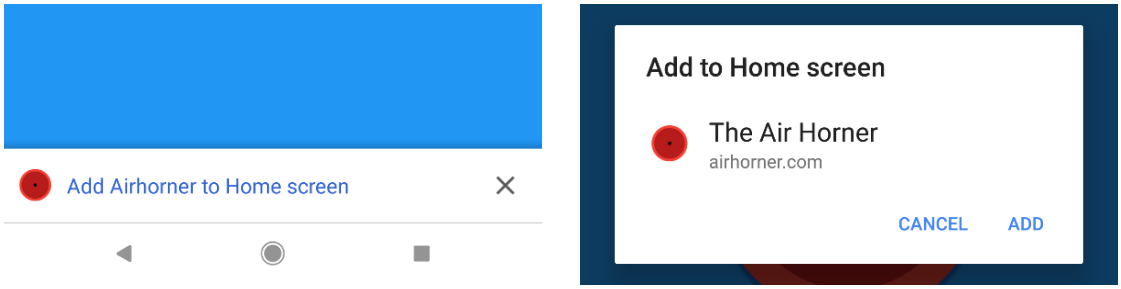
3.5 Dodajanje na domač zaslon

V prvih poglavjih je bila zmožnost namestitve opisana kot ena izmed ključnih lastnosti PSA. Na dodajanje na domač zaslon se gleda kot na neko poglobljeno vez med uporabnikom in s stranjo povezano znamko. Na koncu se uporabnik sam odloči, ali bo na na podlagi pretekle interakcije s stranjo tej strani namenil tudi prostor na svojem domačem zaslonu. Vsekakor je dodajanje na domač zaslon neka metrika uspešne interakcije. Prav tako pa se možnosti za ponovne obiske močno povečajo, ko je enkrat ikona znamke na dosegu prstov med ostalimi priljubljenimi aplikacijami na domačem zaslonu. Zato je zelo pomembno, kakšen je proces dodajanja oziroma namestitve.

Način namestitve se med posameznimi ponudniki razlikuje. Ker implementacija dodajanja na domač zaslon ni zapisana v specifikacijah W3C (World Wide Web Consortium), se je pri vsakem brskalniku lotevajo nekako po svoje.

Originalna ideja procesa namestitve je bila, da uporabnik začne uporabljati spletno mesto preko brskalnika. Če uporabnik pokaže določeno mero zanimanja za interakcijo s stranjo, mu brskalnik ponudi možnost namestitve. Seveda je kriterij zanimanja zelo težko definirati, zato se ti kriteriji občasno spreminjajo in se tudi razlikujejo med ponudniki.

V skladu s tem so leta 2018 svoj poziv uporabniku k namestitvi spremenili tudi v Chromu in ga naredili še bolj nevsiljivega. Namesto poziva, ki je zavzel dobršen del zaslona, se trenutno pojavi samo informativna pasica na dnu strani z napisom: dodaj aplikacijo na domač zaslon. Če jo uporabnik aktivno zapre, se do izteka časovnega intervala ne prikaže več na tej strani. Ta interval lahko traja tudi nekaj mesecev. Tako se prepreči nepotrebno obveščanje tistih uporabnikov, ki jih namestitev ne zanima. Trenutno se poziv prikaže približno po pol minute uporabe spletnega mesta. [22]



Slika 11 – Prikaz sistemskega poziva k namestitvi

Če uporabnik potrdi poziv, se na zaslonu pojavi enostaven dialog z imenom aplikacije in ikono. Ko uporabnik potrdi namero, se aplikacija začne nameščati. Ta proces nameščanja so do neke mere prevzeli tudi pri ostalih ponudnikih.

Dialog in proces nameščanja pa se da kadarkoli sprožiti tudi ročno. V vseh brskalnikih lahko najdemo v meniju opcijo dodajanja na domač zaslon oziroma opcijo namestitve. Uporaba te možnosti pa je za povprečne uporabnike spleta velikokrat preveč skrita.

Brskalniki, ki bazirajo na jedru Chromium, omogočajo razvijalcem, da prevzamejo nadzor nad tem, kako in kdaj bodo sprožili namestitveni dialog. V grafični uporabniški vmesnik lahko svobodno umestijo element, ki sproži poseben API. Ta vmesnik posreduje namero brskalniku, ki prikaže sistemski poziv.

Uspešna implementacija tega procesa lahko zelo pripomore k širitvi prisotnosti PSA tudi izven obsega brskalnika. Razvijalcem daje moč, da popolnoma prilagodijo uporabniško izkušnjo. Tega pa nikakor ne smemo izkoriščati in siliti uporabnike v namestitev. Aplikacija mora kljub temu biti normalno uporabna tudi preko brskalnika.

3.5.1 WebAPK

Najbolj izpopolnjeno PSA izkušnjo ponuja Google Chrome za Android, kjer so leta 2017 oznanili izboljšave v procesu namestitve. Od takrat naprej se ob dodajanju na domač zaslon v ozadju zgodi še veliko več. Google Chrome se poveže s storitvijo Google Play Store, ki služi za distribucijo aplikacij v sistemu Android. Google Play Store v oblaku generira posebno Android namestitveno datoteko APK (Android Package Kit), ki se v tem primeru poimenuje WebAPK. Google Play Store datoteko opremi še s svojim podpisom, preden jo naprava v ozadju namesti

v sistem. Takšna aplikacija je nameščena v napravo popolnoma enako kot vse ostale domorodne aplikacije in se tako tudi obnaša. Namestitev WebAPK omogoča še boljšo integracijo PSA v operacijski sistem.

Kljub temu, da je Google javno oznanil, da proces generiranja in podpisovanja APK s pomočjo njihove storitve odpira tudi za druge ponudnike, je Chrome zaenkrat edini s to funkcionalnostjo. Potrebno pa je omeniti tudi Samsung Internet, ki na podoben način namesti aplikacije s pomočjo lastne storitve preko trgovine Galaxy. To pa posledično pomeni, da se to lahko zgodi samo na Samsung napravah, kjer obstaja globlji dostop v sam operacijski sistem. [23]

4 RAZVOJ PSA

Vzporedno s to nalogo smo razvili tudi PSA, ki rešuje problem iz realnega sveta. V tem poglavju bomo na kratko predstavili glavne značilnosti aplikacije. Opisali bom uporabljene tehnologije ter nekatere ključne rešitve, ki so usmerjene v izboljšanje uporabniške izkušnje.

4.1 Opis aplikacije

V podjetju, v katerem sem zaposlen, tržimo in razvijamo lasten produkt elektronskega laboratorijskega dnevnika. Elektronski laboratorijski dnevnik predstavlja računalniški sistem, ki nadomešča papirnate laboratorijske dnevnike. Na žalost se dandanes za beleženje rezultatov in vodenje raziskav še vedno velikokrat uporabljata le papir in svinčnik. Po drugi strani pa se število podatkov, ki jih lahko pridobimo s pomočjo elektronskih merilnih instrumentov, eksponentno povečuje. S pomočjo elektronskih laboratorijskih dnevnikov lahko te podatke vodimo v digitalni obliki, kar pripomore k preglednosti in dostopnosti podatkov in posledično k izboljšani produktivnosti. Naš produkt sodi med 5 najbolj prepoznanih elektronskih laboratorijskih dnevnikov, ki so na voljo. Kljub temu pa ima eno pomanjkljivost: razvit je bil predvsem za uporabnike širokih zaslonov namiznih in prenosnih računalnikov, uporabnost na mobilnih napravah pa je zaradi kompleksnosti grafičnega vmesnika precej okrnjena. Čeprav večina uporabnikov za resno delo še vedno uporablja računalnike, pa zanje v laboratorijskem okolju marsikdaj ni prostora.

Odločili smo se, da implementiramo ločeno mobilno verzijo v obliki PSA. Ta v prvi verziji ciljno podpira osnovne funkcionalnosti, ki jih raziskovalec potrebuje pri svoji delovni postaji za izvedbo nalog.

Naš cilj je bil, da implementiramo PSA, ki bo delovala v simbiozi z že obstoječim sistemom in bo prinesla dodano vrednost za uporabnike, ki želijo imeti pregleden prikaz protokolov tudi na mobilni napravi. Pomembno je bilo tudi, da izvedba protokola nemoteno poteka v načinu brez povezave. S tem je omogočeno nemoteno delo v laboratorijih brez aktivne internetne povezave.

4.2 Načrtovanje aplikacije

V naslednjih poglavjih bo predstavljena arhitektura nekaterih ključnih sklopov aplikacije. Ti deli so bili premišljeni pred samim začetkom implementacije. V procesu je sodeloval tudi oblikovalec uporabniških vmesnikov, ki je predstavil izgled zaslonov in povezavo med njimi.

4.2.1 Uporabljene tehnologije

PSA je implementirana v obliki enostranske aplikacije (ang. Single page application). Za razvoj smo uporabili ogrodje Ionic, ki temelji na JavaScript ogrodju Angular. Ogrodje Ionic je skupek knjižnic in orodij za razvoj hibridnih aplikacij med različnimi platformami. Razvijalcem omogoča, da enako kodo uporabijo ne glede na ciljno platformo, Ionic pa ovije aplikacijo s potrebnimi specifičnimi vmesniki za samo platformo. V zadnjih letih pa ponujajo tudi odlično podporo za razvoj PSA aplikacij – torej aplikacij, kjer je ciljna platforma splet. Takšne aplikacije na koncu ne potrebujejo posebnega ovoja vmesnikov, saj je koda že v osnovi napisana z uporabo osnovnih spletnih tehnologij*, JavaScript*, HTML in CSS.

PSA za hrambo podatkov uporablja podatkovno bazo *IndexedDB*. Za dostop in upravljanje z njo pa uporabljamo knjižnico Dexie.js. Ta knjižnica predstavlja ovoj, ki spremeni asinhrone akcije baze v objekte *Promise*. Prav tako odstrani kompleksnost in poenostavi uporabo osnovnih funkcij.

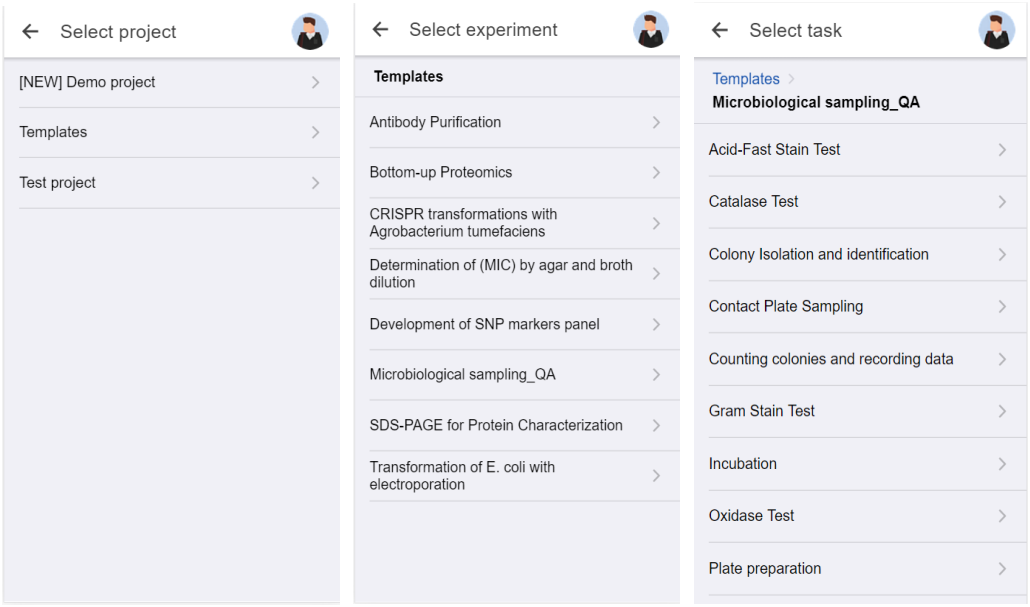
4.2.2 Struktura uporabniškega vmesnika

V platformi so podatki organizirani hierarhično. Vsak uporabnik je član ene ali več ekip. Vsaka ekipa vsebuje projekte, v katerih so eksperimenti. Vsak eksperiment vsebuje svoja opravila. Opravilo pa je glavni konceptualni objekt, ki lahko vsebuje podatke različnih oblik.

Jedro aplikacije v prvi verziji je torej pregled lastnih opravil in njihova izvedba. Vsako opravilo vsebuje protokol z naborom korakov, ki jih je potrebno dokončati med izvajanjem. Korak lahko vsebuje različne elemente, ki uporabniku pomagajo pri izvajanju protokola. To so lahko podroben opis, obogaten s slikovnim materialom, tabele z rezultati ali formulami, raznovrstne

ček liste ali priponke datotek različnih formatov. Uporabnik lahko izvaja protokol v obliki izpolnjevanja posameznih korakov in se tako pomika med njimi.

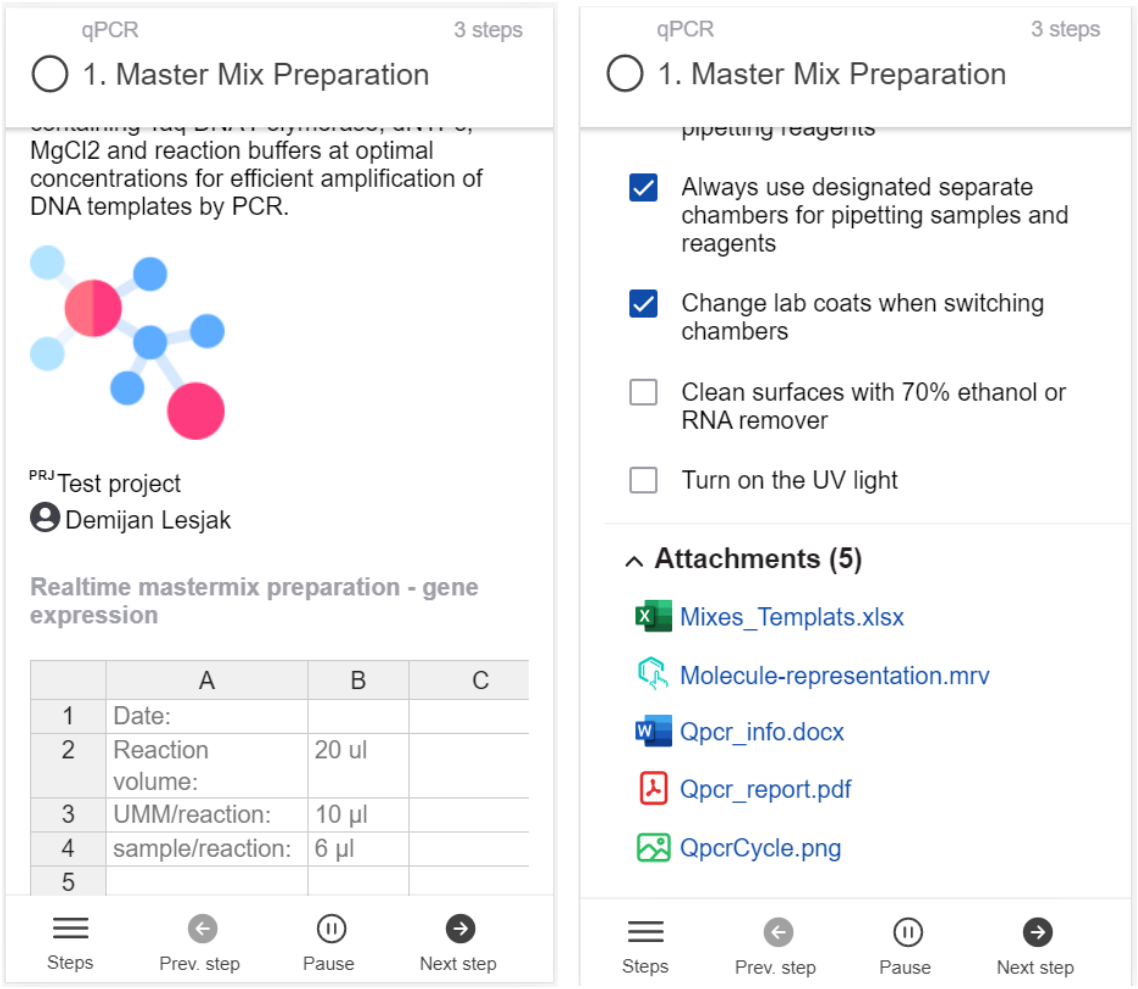
Aplikacija se nekako loči na dva dela: izbiranje opravil in njihova izvedba. Pri izboru opravil se mora uporabnik prebiti čez vse nivoje hierarhije. To se izvede z izbiranjem elementov iz enostavnih seznamov. Z vsakim izborom je en sloj globlje v strukturi. Potek izbora lahko vidimo na spodnjih posnetkih zaslona.



Slika 12 – Zaslonske slike izbirnih zaslonov

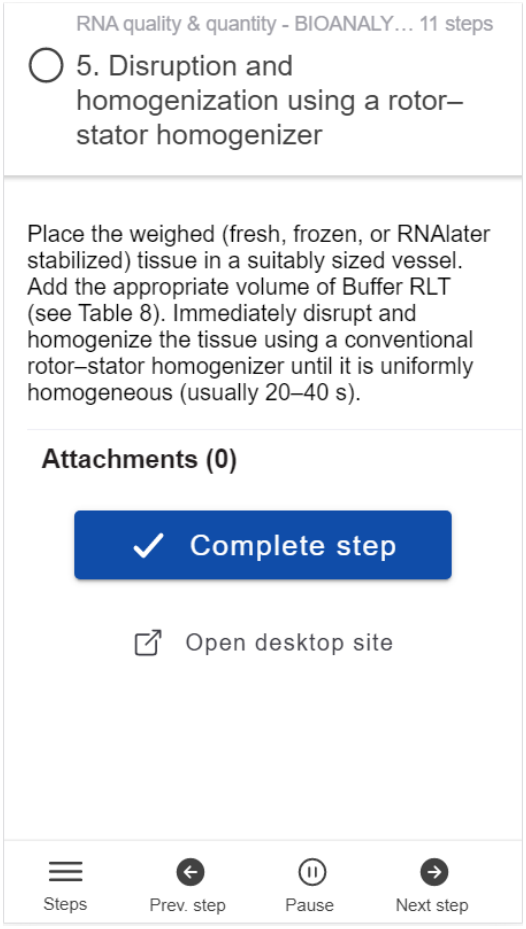
Ko uporabnik izbere opravilo, se protokol naloži in avtomatsko odpre prvi korak. Vsak korak lahko vsebuje opis, slikovno gradivo, pametne označbe (povezave z zunanjimi objekti), tabele, ček liste ali dodatne priloge. Vsi ti elementi so razvidni na spodnjih posnetkih zaslona.

Za prikaz tabele se uporablja zunanja knjižnica Handsontable in ni prilagojena širini zaslona, uporabnik jo namreč lahko premika v horizontalni smeri.



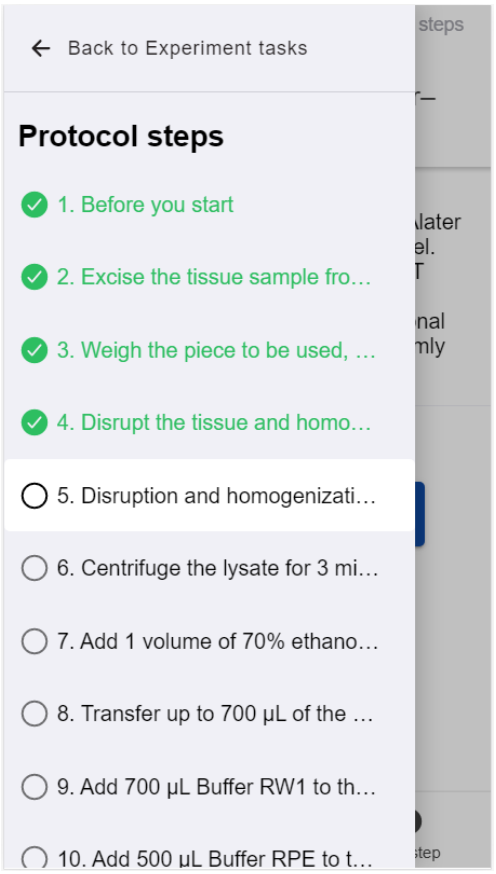
Slika 13 – Zaslonske slike korakov opravila

Na dnu vsakega koraka sta pozicionirana dva gumba: »Končaj korak« in »Odpri stran v namiznem načinu«. Prvi označi korak kot končan, posodobi vrednost na strežniku in spremeni izgled naslova v uporabniškem vmesniku. Drugi gumb zapusti kontekst PSA in v brskalniku odpre naš korak na komplementarni lokaciji v platformi. Ker PSA trenutno še ne podpira nekaterih funkcionalnosti, kot na primer komentiranje in urejanje podatkov, ima uporabnik možnost to izvesti preko brskalnika na glavni platformi. Primer gumbov lahko vidimo na spodnji sliki.



Slika 14 – Zaslonska slika opravila z gumbom za zaključek

Med izvajanjem opravila je ves čas prisotna še navigacijska opravilna vrstica na dnu zaslona. Kot vidimo na zgornjih slikah, ta vsebuje nekaj osnovnih gumbov za premikanje med koraki. Za enostavno sprehajanje med koraki lahko uporabimo gumba »naslednji korak« ali »prejšnji korak«. Gumb med njima predstavlja začasno prekinitev opravila in uporabnika pelje nazaj na izbirne zaslone. Skrajno levi gumb pa nam olajša pregled nad vsemi koraki. Ob kliku nanj se odpre stranski panel, kjer so našteti vsi koraki opravila. Takoj lahko vidimo, kateri so že opravljeni in kateri ne. Z izbiro koraka v tem seznamu pa se lahko takoj premaknemo nanj.



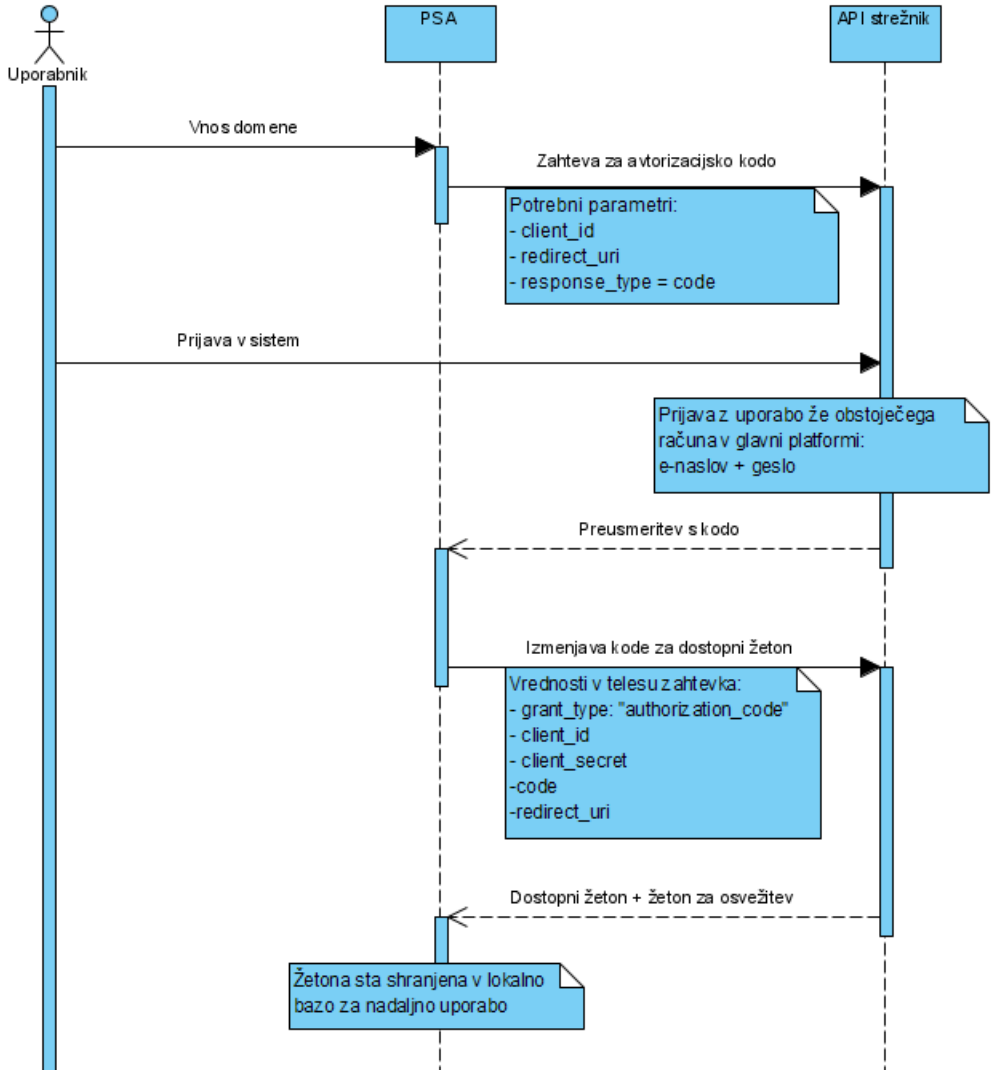
Slika 15 – Stranski meni korakov

4.2.3 Avtorizacija

Vse funkcionalnosti aplikacije so dostopne po uspešni avtorizaciji uporabnika. Avtorizacija je implementirana z uporabo standardnega protokola OAuth 2 z uporabo žetona JWT (ang. JSON Web Token). Začetna avtorizacija je prikazana na spodnjem sekvenčnem diagramu.

Cel proces se začne z vnosom domene, s katero se uporabnik želi povezati. To je potrebno zato, ker lahko raziskovalne institucije pridobijo celotno instanco sistema zase. Tako obstaja več instanc platforme na različnih domenah. Aplikacija PSA pa je postavljena centralno in se je sposobna povezati z vsako izmed teh domen. Po potrjenem vnosu domene se sproži klic na vmesnik na strežniku pravilne domene. Ta zahteva vsebuje v URL naslednje obvezne parametre: *client_id*, *redirect_uri* in pa *response_type*. Strežnik nas preusmeri na prijavno stran glavne platforme. Ko se uporabnik uspešno prijavi, ga stran vpraša, ali dovoli avtorizacijo naši aplikaciji. Po potrditvi nas strežnik ponovno preusmeri nazaj v PSA. Ob preusmeritvi se v URL naslovu nahaja parameter »code«. To kodo je potrebno prestreči, saj jo potrebujemo za

izmenjavo za dostopni žeton. Veljavnost te kode je zelo kratkotrajna. Takoj ponovno pošljemo zahtevek na strežnik. Ta zahtevek je oblike POST z vrednostjo parametra grant_type: »authorization_code«. Vsebuje tudi obvezne parametre: »client_id«, »client_secret«, »code« in »redirect_uri«. Če vsi poslani atributi ustrezajo, nam strežnik vrne dostopni žeton in žeton za osvežitev. Ta dva žetona shranimo v lokalno bazo za nadaljnjo uporabo.



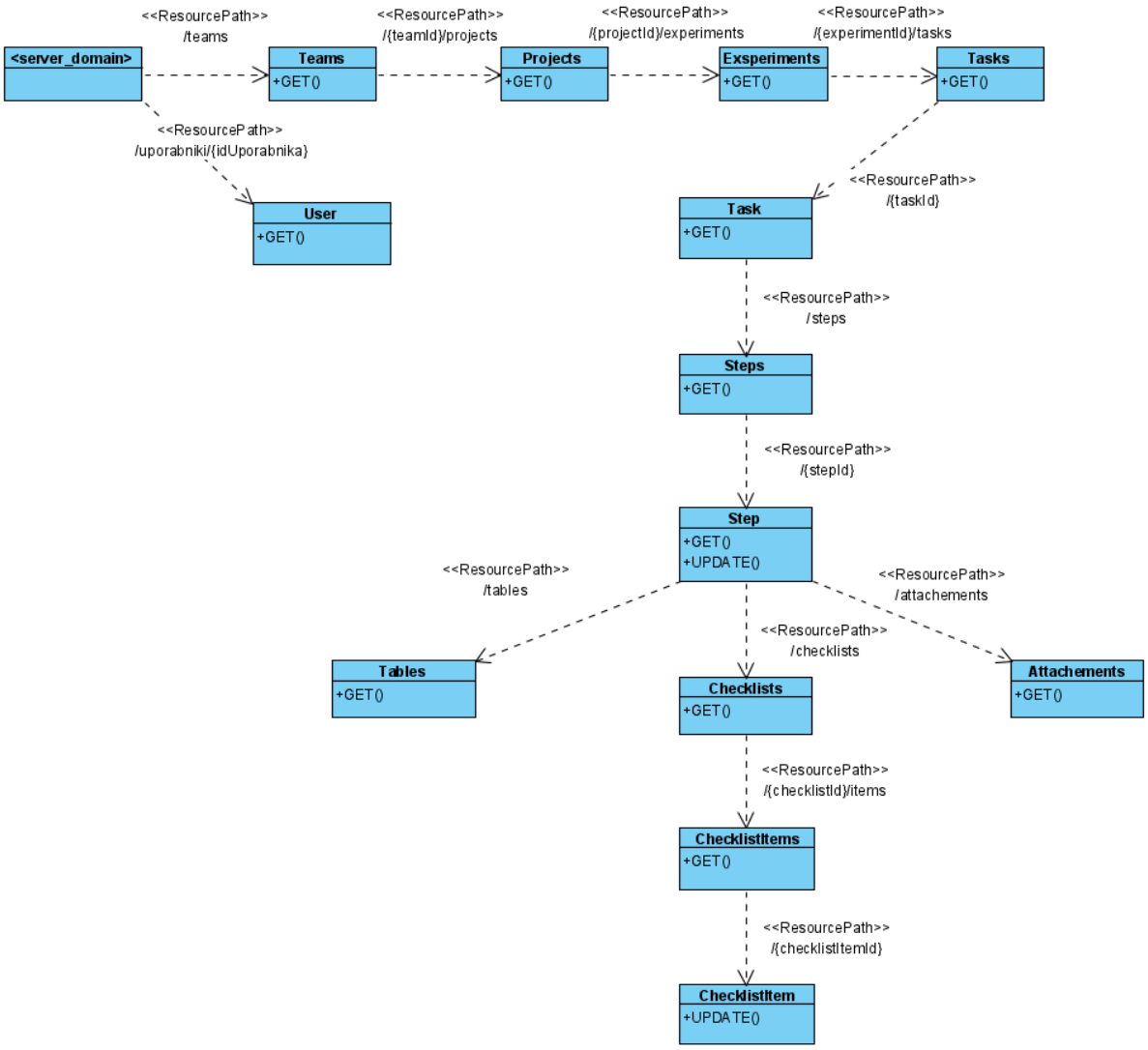
Slika 16 – Sekvenčni diagram avtorizacije

Dostopni žeton moramo pripeti vsakemu naslednjemu zahtevku, poslanemu na strežnik. Ta žeton ima omejeno življenjsko dobo. Ko poteče, uporabimo žeton za osvežitev za obnovitev avtorizacije. V tem primeru nam strežnik vrne nov dostopni žeton in nov žeton za osvežitev.

4.2.4 Pridobivanje podatkov

Vsi podatki uporabljeni v PSA se pridobijo preko aplikacijskega programskega vmesnika (API) z uporabo principa REST. V tem poglavju bomo podrobneje pregledali strukturo uporabljenih končnih točk vmesnika. Na spodnjem modelu se lahko razloči del namenjen za izbiranje po hierarhiji, in del, ki se uporablja za prikaz ter izvedbo opravila. Kot vidimo, moramo priti precej globoko v strukturo, preden se diagram razveja. Korak je prva entiteta, ki ima nase vezane različne vsebine. To se odraža tudi v sami aplikaciji, saj do odprtja opravila, kjer se prikaže prvi korak, ne vidimo veliko vsebine, ampak le potujemo skozi strukturo.

Najkompleksnejši element vsebine predstavljajo ček liste. Da jih prikažemo, moramo najprej pridobiti identifikatorje vseh ček list. Nato pa te identifikatorje uporabimo, da pridobimo še elemente vsake ček liste posebej. Ko je posamezen element obkljukan, moramo za posodobitev iti še nivo globlje in uporabiti identifikator spremenjenega elementa.



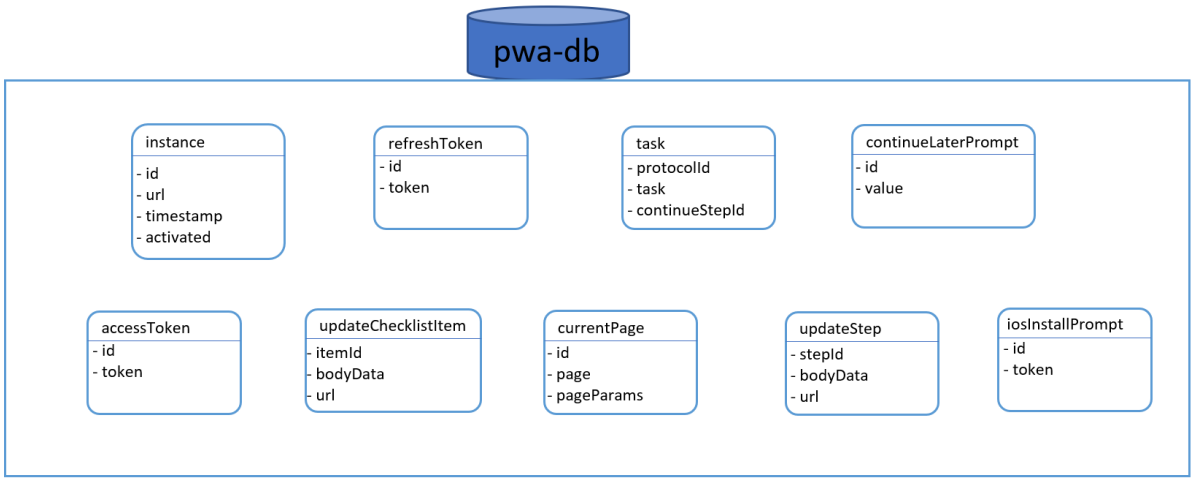
Slika 17 – Diagram API-ja

V tej verziji aplikacije smo se osredotočili na prikaz podatkov. Brisanje in kreacija zaenkrat še nista podprta, kljub temu pa podpiramo posodobitve obstoječih podatkov ob spremembi ček list elementov in ob zaključku koraka.

4.2.5 Shema lokalne podatkovne shrambe

IndexedDB omogoča hranjenje podatkov, ki so indeksirani po svojih ključih. Sama baza ne vsebuje tabel, ampak shrambe objektov. Vsak objekt mora imet svoj ključ, oblika objektov pa ni pomembna. Preden lahko bazo uporabljamo, moramo definirati shemo teh shramb. Spodaj je prikazana shema, ki smo jo uporabili v naši aplikaciji. Pomembno je, da je shema dobro

načrtovana od začetka, saj moramo ob najmanjši spremembi spremeniti verzijo. Tako brskalnik zazna, da se je struktura podatkov spremenila, in jo na novo postavi.



Slika 18 – Shema lokalne shrambe

Kot je razvidno, imamo v bazi definiranih devet različnih entitet. Najpomembnejše so *instance*, *accessToken* in *refreshToken*. Te so nujne za avtorizacijo in uporabo aplikacijskega programskega vmesnika. »instance« hrani ime domene, s katero se je uporabnik uspešno avtoriziral. Dostopni in žeton za posodobitev se prav tako shranita ob uspešni avtorizaciji. Vrednost domene in dostopnega žetona preberemo iz baze ob vsakem zahtevku na strežnik.

Naslednja zelo pomembna entiteta je *task*. Ta predstavlja celotno opravilo, ki je že bilo uspešno preneseno in odprto v aplikaciji. Shranjevanje celotnega objekta nam omogoči nemoteno nadaljevanje izvajanja v primeru, ko se izgubi povezava. Zraven tega objekta pa je shranjen še identifikator zadnjega odprtega koraka. To nam omogoča, da ob vrnitvi uporabnika na opravilo prikažemo korak, ki ga je nazadnje izvajal.

Ostale podatkovne shrambe v bazi vsebujejo enostavne vrednosti, ki nam omogočajo dolgotrajno hranjenje nekaterih nastavitev aplikacije.

4.3 Implementacija

V tem poglavju bomo natančneje pregledali nekatere bolj zanimive koščke implementacije v specifični aplikaciji. Večina kode v naslednjih primerih je napisana v jeziku TypeScript, ki se uporablja v ogrodju Angular. TypeScript je nadnabor jezika JavaScript. Ima vse funkcionalnosti JavaScript, a hkrati podpira dodatne strukture, ki omogočajo bolj objektno orientiran razvoj. Preden želimo takšno aplikacijo pognati v brskalniku, se mora TypeScript najprej prevesti v osnovni JavaScript. V primerih iz storitvenega delavca bo uporabljena le koda jezika JavaScript.

4.3.1 Vključitev spletnega manifesta

Prvi korak k implementaciji PSA je dodajanje manifesta. Za vključitev dokumenta na stran se uporabi <*link*> HTML označba, in sicer nekje proti začetku HTML dokumenta.

```
<!– Začetna konfiguracija -->
<link rel="manifest" href="manifest.webmanifest">
```

Izsek kode 1 – Vključitev spletnega manifesta na spletno stran

Spodaj so prikazane vrednosti, ki smo jih uporabili v spletnem manifestu.

```
{
    "name": "Lab app",
    "short_name": "Laboratory app",
    "theme_color": "#104da9",
    "background_color": "#104da9",
    "display": "standalone",
    "scope": "./",
    "start_url": "./?source=homescreen",
    "icons": [
      {
        "src": "assets/icons/icon-192x192.png",
        "sizes": "192x192",
        "type": "image/png",
        "purpose": "maskable any"
      },
      {
        "src": "assets/icons/icon-512x512.png",
        "sizes": "512x512",
        "type": "image/png",
        "purpose": "maskable any"
      }
    ],
    "related_applications": [{
      "platform": "webapp",
      "url": "https://labapp.net/manifest.webmanifest"
    }]
  }
```

Izsek kode 2 – Uporabljeni atributi spletnega manifesta

4.3.2 Angular storitveni delavec

Javascript ogrodje Angular že od verzije 5 ponuja implementacijo storitvenega delavca. Za vzpostavitev storitvenega delavca v projektu moramo dodati paket *@angular/pwa*. To

poskrbi za potrebne osnove, da se aplikacija prelevi v progresivno aplikacijo. Ustvari se manifest datoteka z prednastavljenimi ikonami, ki se avtomatsko vključi v *index.html*. Doda se koda za registracijo storitvenega delavca. Zraven pa se doda tudi konfiguracijska datoteka »ngsw-config.json«. Ta datoteka vsebuje nastavitve za izgradnjo storitvenega delavca. Nastavitve določajo, katere datoteke naj bodo predpomnjenje in na kakšen način. Sama skripta storitvenega delavca pa se generira ob izgradnji produkcijskega paketa aplikacije z uporabo komande »ng build --prod«. To ustvari datoteko ngsw-worker.js.

Ker pa se ta skripta generira ob izgradnji aplikacije, je ne moremo poljubno spreminjati. Nastavljati je možno le parametre v konfiguracijski datoteki. Za naše primere uporabe to ni bilo dovolj. Potrebovali smo možnost razširitve same skripte. Da smo to zaobšli, smo ustvarili novo datoteko z generičnim imenom: *service-worker.js*. V to datoteko smo nato vključili skripto, ki se kasneje zgradi. Ob zagonu pa smo zamenjali ime datoteke storitvenega delavca, ki ga stran registrira.

```
...
ServiceWorkerModule.register(
  "service-worker.js",
  { enabled: environment.production }),
...
```

Izsek kode 3 – Registracija storitvenega delavca

```
// Vključitev generirane skripte iz konfiguracijske datoteke.
importScripts('./ngsw-worker.js');

// Dodana specifična koda za naše potrebe.
...
```

Izsek kode 4 – Vključitev zgenerirane skripte

4.3.3 Povezava med platformo in PSA

Namen te aplikacije ni, da bi bila samostojen produkt, temveč da dopolni primere uporabe že obstoječega sistema. PSA omogoča veliko bolj enostavno uporabo na mobilnih napravah,

kljub temu pa še vedno ne podpira številnih funkcionalnosti osrednje platforme. Zaradi tega smo vzpostaviti enostaven preklop med kontekstom PSA in spletno platformo.

Na vseh straneh v PSA lahko najdemo gumb, ki vodi na spletno platformo. Tukaj se lokacija dinamično določi glede na stran, na kateri se nahajamo. Če izbiramo projekte, nas vodi na pregled projektov, če izbiramo opravila, nas vodi na pregled opravil. Če to izberemo na samem koraku, se odpre točno specifično opravilo, v katerem se trenutni korak nahaja.

Na spletni platformi se pojavi podoben gumb, odpri v mobilni aplikaciji, ampak le, ko odpremo določeno opravilo. Ta gumb nas vodi k prvemu koraku opravila. Ideja tega gumba je tudi obveščanje uporabnikov o obstoju mobilne verzije. Želeli smo, da se preklop za nove uporabnike zgodi čim bolj enostavno. Seveda lahko v PSA z uporabo natančnega URL naslova ciljamo točno določeno stran. Ker pa je PSA centralna za vse instance platforme, nima informacije, s katere domene želimo dostopati. Implementacija gumba na platformi ni bila v obsegu te naloge, kljub temu pa je bilo potrebno vzpostaviti nek sistem, ki bo zagotavljal zunanje proženje avtorizacije na specifično domeno.

To smo dosegli s predstavitvijo novega parametra na koncu ciljnega URL naslova. Če želimo ciljati stran na določeni domeni, na konec URL naslova dodamo »?domain=<ime_domene>«. Če ta podatek manjka in v PSA nihče ni prijavljen, se prikaže začetni zaslon, kjer se zahteva vpis domene.

Da to funkcionira, moramo ob vsakem začetnem nalaganju PSA preveriti, ali je parameter prisoten. Če parameter obstaja, ustvarimo nov objekt instance z imenom domene. Nato preverimo, ali morda že obstaja kakšen tak objekt v lokalni shrambi. Če v shrambi še ni bilo shranjene te instance, shranimo novo. Če pa v shrambi najdemo instanco, preverimo, ali se ime ujema s parametrom. Če je ime drugo, moramo prejšnjega uporabnika odjaviti in zamenjati instanco s trenutno. V vseh primerih pa se kasneje avtomatsko poizkusimo avtorizirati v strežnik na tej domeni. Spodaj je prikazan primer funkcije ob zagonu aplikacije.

```
// Angular način pridobivanja parametra iz URL.
const domainParam = event.snapshot.queryParams["domain"];

// Preverjanje ali je domena prisotna v parametru.
if (domainParam !== undefined) {
    // Ustvarimo nov objekt instance.
    const newServerInstance = new ServerInstance(domainParam);
    // Pridobimo shranjeno instanco v IndexedDB.
    const savedDomain = await this.storage.getServerInstance();

    // Dodamo označbo za avtomatsko prijavo
    // ob vzpostavitvi aplikacije.
    this.autoLogIn = true;
    // Preverjanje ali že obstaja kakšna instance.
    if (savedDomain === null) {
        // Dodamo novo instanco v bazo.
        await this.storage.setServerInstance(newServerInstance);
    } else {
        // Preverimo ali se že obstoječa instanca ujema z novo.
        if (savedDomain.url !== domainParam) {
            // Če je bila prej shranjena druga domena,
            // je potrebno podatke prejšnjega uporabnika izbrisati
            // in ga odjaviti iz Sistema.
            this.auth.logout();
            // Nato prepišemo vrednost instance z novo.
            await this.storage.setServerInstance(newServerInstance);
        }
    }
}
```

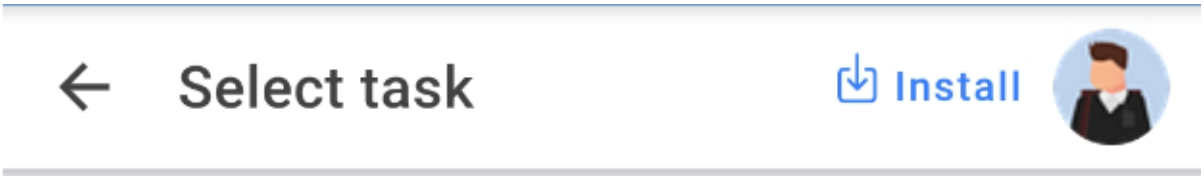
Izsek kode 5 – Preverjanje domene v URL parametru

Prehod iz platforme v brskalniku se najboljše obnaša, kadar smo PSA namestili z brskalnikom Chrome. Ker Chrome uporablja WebAPK, sistem zazna, da je ciljni URL v obsegu aplikacije. PSA se tako avtomatsko odpre v samostojnem načinu.

4.3.4 Namestitveni gumb

V prejšnjih poglavjih smo že omenili, da je za PSA zelo pomembna metrika število namestitev na domač zaslon. Ta proces smo želeli čim bolj poenostaviti, a hkrati uporabnikom omogočiti

povsem normalno uporabo znotraj brskalnika. Tako smo umestili v opravilno vrstico vsakega zaslona enostaven gumb »Install«, kot je prikazano na spodnji sliki.



Slika 19 – Namestitveni gumb v naslovni vrstici

Ta gumb seveda ni statičen. Prikaže se le, kadar brskalnik sproži »beforeInstallPrompt« dogodek. To se ne zgodi, kadar je aplikacija že nameščena. Prav tako ta dogodek podpirajo le brskalniki Chromium. Poslušalec dogodka je prikazan v naslednji kodi.

```
// Poslušalec dogodka za namestitev.
window.addEventListener('beforeinstallprompt', (e) => {
  // Preprečimo, da se pojavi standarden poziv brskalnika.
  e.preventDefault();
  // Shranimo poziv za kasnejšo uporabo.
  this.deferredPrompt = e;
  // Posodobimo uporabiški vmesnik z gumbom za namestitev.
  this.enableInstallButton();
});
```

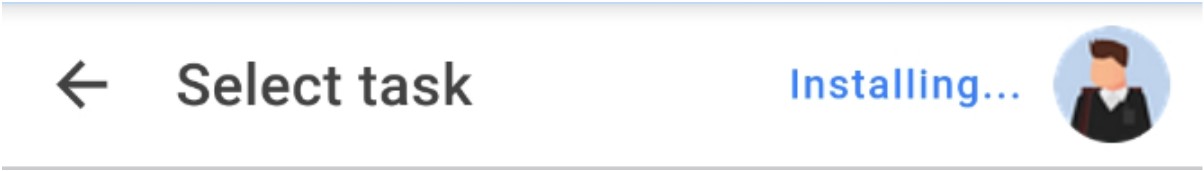
Izsek kode 6 – Poslušalec dogodka za namestitev

Ko uporabnik pritisne na gumb, se ta skrije in sproži se sistemski poziv na namestitev. Če uporabnik prekliče namestitev, se gumb ponovno prikaže, v nasprotnem primeru pa se prične namestitev. V brskalnikih, ki za namestitev ne uporabljajo WebAPK, se ikona doda na domač zaslon skoraj v istem trenutku. Namestitev v Google Chrome pa poteka v ozadju in lahko traja tudi do pol minute. Navadno se pojavi majhno sistemsko obvestilo v statusni vrstici Android, vendar je proces za uporabnika precej nejasen.

Tako smo šli še korak dlje in poizkusili čim bolj jasno prikazati trenutno stanje. K sreči je ravno v času razvoja ta funkcionalnost postala mogoča v brskalniku Chrome, vendar le za Android. Od verzije 84 naprej lahko s pomočjo atributa »related_applications« v spletnem manifestu identificiramo lastno, že nameščeno v sistemu PSA.

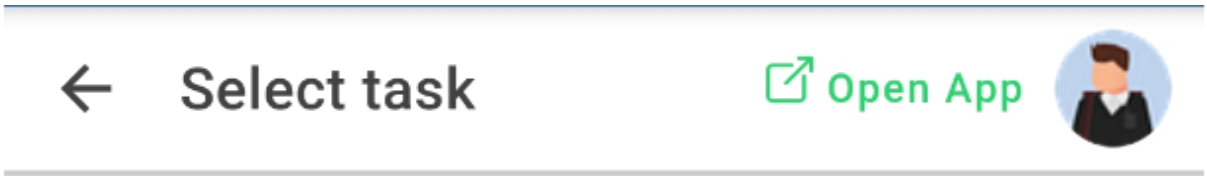
To nam je omogočilo dve stvari: po kliku na »Install« lahko prikažemo stanje nameščanja (razvidno na Sliki 20), dokler aplikacije ne zaznamo v seznamu nameščenih aplikacij, in prikaz

gumba »Open app«, ki se prikaže, kadar se uporabnik nahaja v brskalniku, a ima PSA že nameščeno. Ta gumb odpre PSA iz brskalnika na enakem mestu, kot smo zapustili brskalnik.



Slika 20 – Napis med potekom namestitve

To nam omogoča, da uporabnika vodimo skozi proces. V nasprotnem primeru se lahko zgodi, da uporabnik sploh ne zazna uspešne namestitve, saj ne najde ikone takoj po potrditvi poziva za nameščanje. Takoj po namestitvi pa ima uporabnik možnost odpreti vsebino neposredno v PSA.



Slika 21 – Gumb za odpiranje že nameščene aplikacije iz brskalnika

»Open App« gumb je, podobno kot namestitveni gumb, na voljo konsistentno skozi celotno aplikacijo. Primer takšnega gumba je na Sliki 21.

Aktivacija teh gumbov poteka preko globalnega servisa. Ta servis je instanca edinca in je na voljo vsem komponentam v aplikaciji. Ko uporabnik uporabi gumb »Install«, sproži metodo »addToHomeScreen«. Ta metoda sproži sistemski poziv in počaka na odgovor uporabnika. Glede na rezultat nastavi pravilna stanja ali aktivira nameščanje ali pa ponovno prikaže gumb. To funkcijo iz globalnega servisa lahko vidimo v spodnjem delčku kode.

```
// Ta koda se nahaja v globalnem servisu “appService”.
addToHomeScreen() {
    // Skrijemo "install" gumb.
    this.disableInstallButton();
    // Prikažemo zakasnjeni poziv, ki smo ga shranili ob poslušanju
    // na "beforeinstallprompt" dogodek.
    this.deferredPrompt.prompt();
    // Počakamo na odziv uporabnika.
    this.deferredPrompt.userChoice.then((choiceResult) => {
        // Preverimo rezultat odziva.
        if (choiceResult.outcome === 'accepted') {
            // Uporabnik je potrdil namestitev.
            // Aktiviraj "installing" stanje
            this.installing.next(true);
        } else {
            // Uporabnik je  preklical namestitev.
            // Ponovno prikažemo "Install" gumb.
            this.enableInstallButton();
        }
    });
}
```

Izsek kode 7 – Funkcija, ki sproži poziv za dodajanje na domač zaslon

V komponenti se naročimo na obvestila o spremembi vrednosti stanja. Ko zaznamo, da je stanje nameščanja aktivno, prikažemo tekst »Installing« in ustvarimo interval, ki vsako sekundo preverja, ali je naša PSA nameščena.

```
// poslušamo na spremembo installing stanja
this.appService.installing.subscribe((value: boolean) => {
    // shranimo vrednost v globalno spremenljivko,
    // ta vrednost skrbi za pravilen prikaz “installing” teksta
    this.isInstalling = value;
    if (this.isInstalling) {
        //nastavimo interval, ki preverja seznam nameščenih aplikacij
        this.checkingInterval =
            setInterval(() => this.checkForInstalledApp(), 1000);
    }
});
```

Izsek kode 8 – Nameščanje in zaznavanje nameščenih aplikacij

Funkcija, ki preverja nameščene aplikacije, najprej preveri, ali je vmesnik »getInstalledRelatedApps« sploh na voljo. Ker ta vmesnik najde PSA samo na verziji brskalnika

Android , moramo preveriti tudi to. Vmesnik vrne rezultat v obliki seznama aplikacij. V tem seznamu so lahko vnaprej definirane aplikacije, ki so v povezavi z našo stranjo. V našem primeru pričakujemo le en rezultat. Url aplikacije v rezultatu se mora ujemati z domeno, na kateri se nahaja naš spletni manifest. Ko najdemo zadetek, zamenjamo napis »Installing« z gumbom »Open App«. Prav tako zaključimo interval preverjanja.

```
checkForInstalledApp() {
    const localNavigator: any = window.navigator;
    // Preverimo ali je vmesnik na voljo in ali je sistem android
    if ('getInstalledRelatedApps' in localNavigator &&
        /android/i.test(localNavigator.userAgent.toLowerCase())) {
        // Uporabimo navigator, da pridobi seznam nameščenih aplikacij
        localNavigator.getInstalledRelatedApps().then(relatedApps => {
            // Rezultat nameščenih aplikacij je seznam,
            // čeprav v našem primeru lahko v njem pričakujemo
            // le en element.
            // Za vsak element v seznamu preverimo podatke
            relatedApps.forEach((app) => {
                // Vsak element lahko vsebuje id, platform in url.
                console.log(app.id, app.platform, app.url);
                // Preverimo ali element iz seznama vsebuje enako domeno,
                // kot jo ima naša aplikacija
                if (app.url ===
                      `${environment.redirectUri}manifest.webmanifest`) {
                    // Nastavimo, da je PWA že nameščena
                    this.appAlreadyInstalled = true;
                    // Končamo interval preverjanja
                    this.endInstallCheck();
                }
                });
        });
    } else {
        console.log("getInstalledRelatedApps doesnt exists in navigator");
    }
}
```

Izsek kode 9 – Zaznavanje, ali je PSA že nameščena

Vse funkcije komponente, ki upravlja gumbe nameščanja, so implementirane s strategijo progresivnega izboljšanja. Brskalniki, ki teh funkcionalnosti ne podpirajo, to kodo preskočijo.

4.3.5 Prilagoditve za sistem iOS

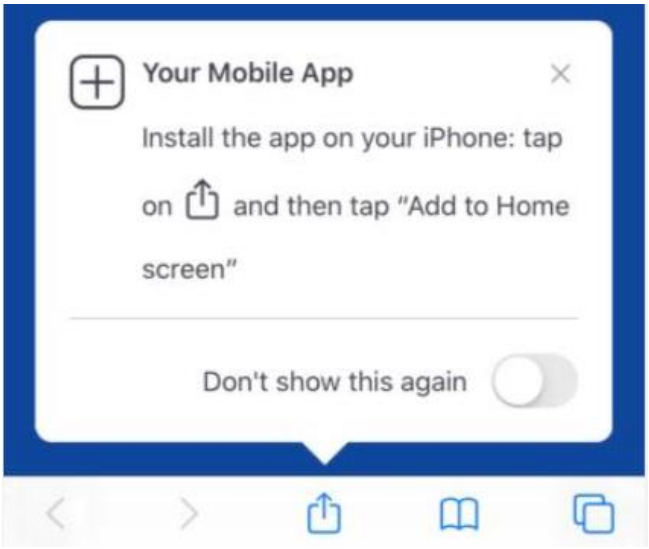
Applov brskalnik Safari je eden izmed brskalnikov, ki zgornjega načina namestitve ne podpirajo, čeprav je fraza »Dodaj na domač zaslon« pravzaprav njihova domislica. Že pred nekaj leti so predstavili serijo meta označb, ki se dodajo v glavo dokumenta HTML. Te označbe zna prebrati njihov brskalnik in prilagoditi izgled aplikacije, če jo uporabnik doda na domač zaslon. Te funkcionalnosti niso nikoli odstranili, a je bila vedno relativno skrita.

Kljub temu, da Apple javno ne podpira izraza PSA oz. PWA, so v zadnjih letih postopoma implementirali standardne tehnologije in so PSA postale funkcionalne tudi na iOS platformi. Toda še vedno je treba uporabljati nekatere njihove označbe, saj so le nekatere vrednosti v spletnem manifestu dejansko upoštevane. Tako smo v indeks aplikacije dodali naslednje označbe:

```
<meta name="apple-mobile-web-app-capable" content="yes"/>
<meta name="apple-mobile-web-app-title" content="Lab app">
<meta name="apple-mobile-web-app-status-bar-style" content="default"/>
<!-- iOS označbe za dodajanje na domač zaslon -->
<link rel="apple-touch-icon" href="/assets/icons/touch-icon-iphone.png">
<link rel="apple-touch-startup-image"
  href="/assets/icons/apple-touch-startup-image-640x920.png">
<link href="/assets/splashscreens/iphone5_splash.png"
  media="(device-width: 320px) and (device-height: 568px)
        and (-webkit-device-pixel-ratio: 2)"
  rel="apple-touch-startup-image" />
```

Izsek kode 10 – Meta označbe za naprave iOS

Nameščanje PSA na naprave Apple je na voljo samo preko izbire v meniju. Ker sistem uporabnika sam ne opozori, da je spletna aplikacija primerna za namestitev, je verjetnost za namestitev precej manjša. Naprave Apple pa so zelo razširjene na našem največjem trgu, v ZDA. Tako smo se odločili za implementacijo posebnega obvestila z enostavnim navodilom namestitve izključno za te uporabnike. Na spodnji sliki (Slika 22) lahko vidimo takšno obvestilo, ki se prikaže le, kadar uporabnik uporablja brskalnik Safari.



Slika 22 – Obvestilo za namestitev na sistemu iOS

Obvestilo se ne prikaže v primeru, da je uporabnik že znotraj PSA. Ker Safari ne podpira eksplicitnega preverjanja, ali je aplikacija že nameščena, tukaj preverjamo, ali se nahajamo v samostojnem (ang. standalone) načinu. To lahko preverimo preko navigatorja.

```
if ('standalone' in navigator && navigator.standalone) {
    // Aplikacija je v samostojnem načinu.
}
```

Izsek kode 11 – Safari metoda za zaznavanje samostojnega načina

Ta metoda pa ni standardizirana in je na voljo le v brskalniku Safari. Če potrebujemo bolj univerzalno rešitev, lahko preverimo, ali je media vrednost pravilno nastavljena. To lahko vidimo na spodnjem primeru.

```
if (window.matchMedia('(display-mode: standalone)').matches) {
    // Aplikacija je v samostojnem načinu.
}
```

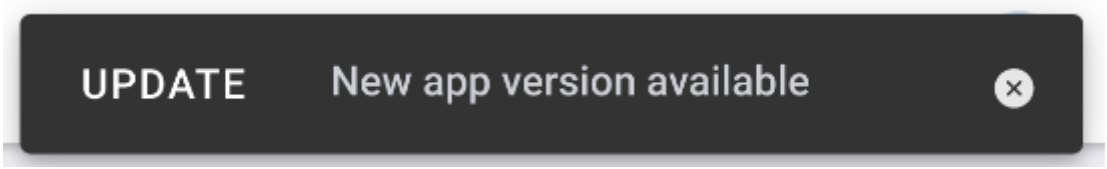
Izsek kode 12 – Alternativna metoda za zaznavanje samostojnega načina

4.3.6 Posodobitve z uporabo Angular storitvenega delavca

Prednost aplikacij PSA je tudi v enostavnosti dostavljanja novih verzij do uporabnika. Čeprav imajo uporabniki PSA nameščeno na sistemu, za njeno posodobitev ne potrebujejo posebnega procesa. Za to v ozadju skrbi storitveni delavec.

Angular storitveni delavec olajša upravljanje s posodobitvami. Vse datoteke aplikacije Angular, ki so bile zgrajene z zgornjo komando, so označene z unikatnim identifikatorjem »hash«. Večino teh datotek predstavlja ogrodje aplikacije, ki ga želimo v celoti shraniti v predpomnilnik pred naslednjo uporabo. Ko so datoteke enkrat v predpomnilniku, storitveni delavec ob vsakem naslednjem zagonu preveri, ali se je identifikator katerekoli datoteke na strežniku spremenil. To bi pomenilo, da se je tudi datoteka spremenila. V tem primeru storitveni delavec obravnava vse datoteke ogrodja kot celoto in poizkusi datoteke ponovno prenesti in posodobiti v predpomnilniku. Ko so vse datoteke uspešno prenesene, je storitveni delavec pripravljen na aktivacijo nove verzije aplikacije. To pa se samodejno zgodi šele ob naslednjem zagonu. [24]

Da pa bi uporabnikom ponudili svežo verzijo kodo takoj, ko je ta na voljo, smo v naši aplikaciji implementirali enostavno sporočilo, ki je prikazano v spodnji sliki (Slika 23). To sporočilo ima dve vlogi: sporoči uporabniku, da obstaja nova verzija PSA, in ponudi gumb za takojšnjo posodobitev.



Slika 23 – Obvestilo o možni posodobitvi aplikacije

Angular storitveni delavec vsebuje priročno storitev »swUpdate«, ki rešuje prav ta primer. S »swUpdate« je naša aplikacija lahko v pripravljenosti, kadar pride do sprememb datotek, in posluša na trenutek, ko so pripravljene na aktivacijo. Nato imamo možnost takojšnje aktivacije teh datotek. S tem se izognemo uporabi zastarele različice spletne aplikacije. Podobno kot upravljanje stanja gumba za namestitev tudi to obvestilo upravljamo v globalnem servisu aplikacije.

```
// Najprej preverimo, ali so posodobitve storitvenega delavca
// sploh podprte.
if (updates.isEnabled) {
    // Naročimo se na spremembe posodobitev.
    updates.available.subscribe(version => {
        // V prejetem objektu lahko preverimo trenutno verzijo
        // in verzijo, ki je na voljo.
        console.log('Trenutna verzija je: ', version.current);
        console.log('Nova verzija je: ', version.available);
        // Prikažemo sporočilo.
        this.presentAppUpdateToast();
    });
}
```

Izsek kode 13 – Zaznavanje posodobitev aplikacije

»Update« gumb, ki se nahaja v sporočilu, nato sproži aktivacijo nove verzije. Ko se aktivacija uspešno izvede, ne naredi nič drugega, kot le ponovno naloži trenuten URL naslov. S tem je takoj prikazana posodobljena verzija aplikacije.

```
// Izvedba ob kliku gumba "Update".
updateButtonClicked() {
    // Zahtevana aktivacija nove verzije.
    this.updates.activateUpdate().then(
        // Ponoven zagon na trenutni lokaciji.
        () => document.location.reload());
}
```

Izsek kode 14 – Akcija ki se izvede ob kliku na "Posodobi gumb"

4.3.7 Način brez povezave

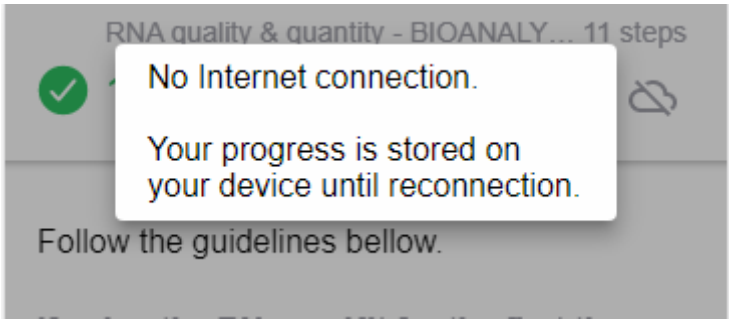
V tem poglavju bomo pregledali, kaj zagotavlja kriterij delovanja brez povezave. Da smo zadostili tem kriterijem, smo uporabili kombinacijo predpomnilnika, podatkovne baze IndexedDB in sinhronizacije v ozadju.

Kot je standardna dobra praksa aplikacij PSA, smo uporabili strategijo aplikacijske lupine. Naš Angular storitveni delavec ob prvem zagonu naloži in shrani v predpomnilnik vse datoteke, ki so potrebne za samostojen zagon. Te datoteke se ne spreminjajo, in ko so enkrat v

predpomnilniku, za dostop do njih ne potrebujemo več spletne povezave. To zagotavlja konsistenten zagon aplikacije v vseh pogojih.

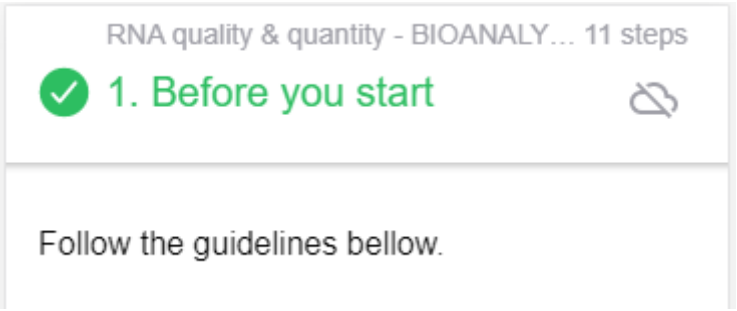
Vendar pa za prikaz vsebine še vedno potrebujemo povezavo. Ker je glavni element aplikacije sprehod in izpolnjevanje protokolov, smo temu posvetili več pozornosti. Predvidevamo, da je za uporabnika zelo dobrodošlo, da lahko konča protokol, potem ko ga je enkrat že začel, četudi je vmes izgubil povezavo. Tako smo se odločili, da se ob odpiranju prvega koraka naloži vsa vsebina protokola. Medtem ko se vsebina nalaga, se sproti gradi objekt opravila, ki vsebuje vse podatke o protokolu in vse korake vključno z vsemi potrebnimi viri. Ta objekt se nato shrani v IndexedDb. Ko je protokol enkrat v bazi, za samo izvedbo ne potrebujemo več povezave. Tako smo na račun daljšega časa odpiranja protokola pridobili možnost normalnega delovanja brez povezave. Da zagotavljamo čim bolj posodobljeno vsebino, se ob vzpostavljeni povezavi naslednji korak normalno prenese in ob spremembi posodobi korak v bazi. Dokler pa povezave ni, se zanašamo samo na podatke iz baze.

Aplikacija v primeru, ko ostane brez povezave, o tem obvesti uporabnika. To omogoča globalen servis, ki ves čas preverja povezavo na objektu »window«. Takoj ko se povezava izgubi, o tem obvesti vse svoje poslušalce. Podobno se zgodi tudi v nasprotnem primeru, ko se povezava ponovno vzpostavi. V uporabniškem vmesniku prikažemo na vrhu zaslona manjše prikazno sporočilo, kot je prikazano na spodnji sliki. To se lahko enostavno skrije in več ne moti uporabnika.



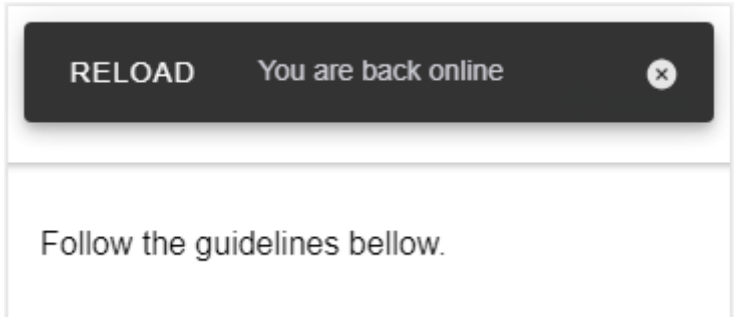
Slika 24 – Obvestilo o izgubljeni internetni povezavi

Zraven sporočila prikažemo v orodni vrstici tudi ikono, ki nakazuje odsotnost povezave. Ta ikona se ne umakne in ves čas subtilno opozarja uporabnika o trenutnem stanju. Ta ikona je razvidna v desnem kotu spodnje slike.



Slika 25 – Ikona v naslovni vrstici ob odsotnosti internetne povezave

Uporabnik je prav tako obveščen, ko se povezava vrne. To pa se v aplikaciji lahko zgodi na dva načina. Na statičnih zaslonih in zaslonih z izbirnimi seznami elementov se prikaže samo začasno sporočilo, ki po nekaj trenutkih izgine. Istočasno se v ozadju sproži osvežitev podatkov, ki prenese nov seznam elementov. Seveda se lahko med odsotnostjo povezave podatki na strežniku spremenijo in mi prikazujemo še vedno starejšo verzijo. Ker bi ti novi podatki lahko povsem spremenili postavitev vsebine v trenutnem koraku, tega ne naredimo avtomatsko. Nepričakovano spreminjanje vsebine med izvajanjem določenega opravila bi lahko negativno vplivalo na uporabniško izkušnjo. Zato uporabnika obvestimo, da se je povezava vrnila, in mu damo izbiro, da sam izvede posodobitev vsebine, ko bo pripravljen. Prikazano na spodnji sliki.



Slika 26 – Obvestilo o ponovni vzpostavitvi povezave

4.3.8 Sinhronizacija podatkov v ozadju

Četudi vnaprej shranimo vse potrebne podatke za delovanje brez povezave, nam to koristi samo pri prikazu podatkov. Še vedno ne moremo komunicirati s strežnikom. Torej ne moremo izvajati nobenih interakcij, kjer bi spreminjali podatke na strežniku. V našem primeru sta takšni akciji izpolnjevanja ček liste in pa zaključevanje koraka. V obeh primerih posodobimo

stanje elementa na strežniku. Te težave smo do neke mere rešili s kombinacijo vmesnika BackgroundSync in pa strategije optimistične posodobitve.

BackgroundSync je del specifikacij storitvenega delavca. Omogoča nam registracijo sinhronizacijskih dogodkov. Kadar naprava nima vzpostavljene povezave, storitveni delavec te dogodke shrani in jih sproži takoj, ko se povezava vzpostavi. To se izvede v ozadju, tudi če je uporabnik že zaprl zavihek ali celo brskalnik.

Registracija sinhronizacijskega dogodka pa ne podpira posredovanja nobenih dodatnih atributov. To izmenjavo podatkov smo implementirali s pomočjo indexedDB.

V metodi za posodabljanje koraka in ček liste, najprej preverimo ali je povezava vzpostavljena. Uporabimo enak servis kot za prikaz obvestil v prejšnjem poglavju. Če zaznamo način brez povezave, shranimo potrebne podatke v lokalno shrambo. Potrebna podatka sta telo in URL naslov. Identifikator koraka uporabimo za ključ, nato pa z registracijskim objektom storitvenega delavca registriramo dogodek. Z označbo bomo kasneje identificirali dogodek in izvedli pravilne akcije. V spodnjem primeru je prikaz registracije sinhronizacijskega dogodka za posodobitev koraka.

```
// Preverjanje ali smo brez povezave.
if (!this.network.isOnline$.value) {
    // Testiranje ali je vmesnik na voljo.
    if ('serviceWorker' in navigator && 'SyncManager' in window) {
        // Shranjevanje podatkov zahtevka v bazo.
        this.storage.updateStepEvent(step.id, body, url);
        // Registracija sync dogodka.
        navigator.serviceWorker.ready.then(
            (swRegistration) => {
                swRegistration.sync.register("update-step");
            }).catch(
                (e) => console.log(e));
    } else {
        // Sinhronizacija v ozadju ni podprta.
        console.log("Background sync not supported");
    }
}
```

Izsek kode 15 – Uporaba sinhronizacije v ozadju ob odsotnosti povezave

V storitvenem delavcu je implementiran poslušalec na vse sinhronizacijske dogodke. Te lahko ločimo na podlagi označb, s katerimi so bili registrirani. Tako jih lahko tudi obravnavamo na različen način.

```
//Poslušalec sinhronizacijskih dogodkov.
self.addEventListener("sync", async (event) => {
    // Ker potrebujemo podatke iz indexedDB,
    // mora biti ta odprta.
    if (!db || !db.isOpen()) {
        await openDB();
    }
    // Na podlagi označbe se odločimo,
    // kako dogodek obravnavamo.
    switch (event.tag) {
        case "update-step":
            event.waitUntil(updateStep());
            break;
        case "update-checklist-item":
            event.waitUntil(updateChecklistItem());
            break;
    }
});
```

Izsek kode 16 – Poslušalec sinhronizacijskih dogodkov v storitvenemu delavcu

Ker pa vmesnik BackgroundSync trenutno ni na voljo v vseh brskalnikih, je pred uporabo v klientu potrebno preveriti, ali je podprt. Tako ne pride do nepotrebnih napak ali izgube podatkov. V brskalnikih, ki sinhronizacije v ozadju ne podpirajo, uporabniku ob kliku prikažemo enostavno sporočilo. S tem ga opozorimo, da nekaterih interakcij ne more izvesti v načinu brez povezave.

4.3.9 Optimistična posodobitev

Vključno z zgornjo kodo je potrebno ustrezno posodobiti tudi uporabniški vmesnik. Za to uporabljamo strategijo optimistične posodobitve. Ta strategija se uporabi tudi takrat, kadar je povezava vzpostavljena. Zaradi boljše zaznave učinkovitosti uporabniški vmesnik posodobimo takoj, ko uporabnik izvede interakcijo, pa čeprav se dejanska vrednost še ni uspešno poslala. Tako takoj ob kliku na gumb za zaključek koraka spremenimo izgled uporabniškega vmesnika tako, kakor da je korak zaključen. Ker posodabljamo relativno enostavno vrednost, kjer ne more priti do konfliktov s podatki na strežniku, vsakokrat

predvidevamo, da se bo posodobitev uspešno izvedla. To se zgodi neodvisno od tega, ali ob kliku pošljemo dejanski zahtevek na strežnik ali pa le registriramo dogodek za sinhronizacijo.

Seveda je potrebno zaznati tiste redke primere, kjer gre nekaj narobe in je zahtevek neuspešen. To upravljanje z napakami je odvisno od kompleksnosti podatkov, v našem primeru le enostavno vrnemo uporabniški vmesnik v pravilno stanje.

Podobno strategijo lahko zaznamo v primeru socialnih omrežij. Velikana, kot sta na primer Twitter in Facebook, jo uporabljata pri všečkanju objav. Četudi se uporabnik nahaja na počasni povezavi, gumb v trenutku nakazuje uspešno akcijo, v ozadju pa se akcija uspešno izvede šele naknadno.

4.3.10 Zbiranje analitičnih podatkov

Za vpogled v uporabo aplikacije smo uporabili Google Analytics. Poleg klasičnih podatkov, kot je denimo ogledi strani, spremljamo tudi nekatere dogodke. En sklop dogodkov je vezan na izvajanje opravil, drugi, bolj zanimiv del, pa beleži interakcije nameščanja PSA. To so naslednji dogodki:

- klik na gumb namesti
- odpri aplikacijo
- odpri v namiznem načinu
- sprejetje poziva za namestitev
- preklic poziva za namestitev
- zagon aplikacije z domačega zaslona.

Umestitev kode za pošiljanje teh dogodkov ni bila preveč zapletena. Kadar gre za nek klik na gumb, se doda košček kode k funkciji, ki se ob tem kliku izvede. To se zgodi pri prvih treh dogodkih. Pri beleženju odziva na poziv se koda nastavi v poslušalec »userChoice«. Nato se preveri vrednost izbire in pošlje pravilno vrednost. Najbolj problematično je sprožiti dogodek, kadar je aplikacija zagnana z domačega zaslona. Obstajajo nekatere metode, kako programatično zaznati aplikacije v samostojnem načinu, vendar so v brskalnikih različno podprte. Tako smo si pomagali s spletnim manifestom. Vrednosti začetnega URL naslova smo dodali parameter »homescreen«. Ob vsakem zagonu aplikacije tako preverjamo, ali obstaja

ta parameter v začetnem naslovu. Če je parameter zaznan, pošljemo dogodek. Ob tem pa tudi shranimo to vrednost za kasneje. Zraven vseh dogodkov namreč pošiljamo tudi dve posebni dimenziji. Prva je ravno način uporabe, ali se aplikacija uporablja v brskalniku ali kot samostojna aplikacija. Druga dimenzija pa sporoča, ali je aplikacija v načinu brez povezave.

Ker pa se dogodki neuspešno pošljejo v okolju brez povezave, smo zato razširili naš storitveni delavec s skripto »sw-offline-google-analytics.js«. To je manjša skripta, ki so jo razvili Googlovi inženirji. Skripta zazna vse zahtevke, ki ciljajo končne točke Google Analytics in jih shrani v IndexedDb. Ko se pojavi povezava, nastavi pravilen čas dogodkov in pošlje vse zahtevke v bazi.

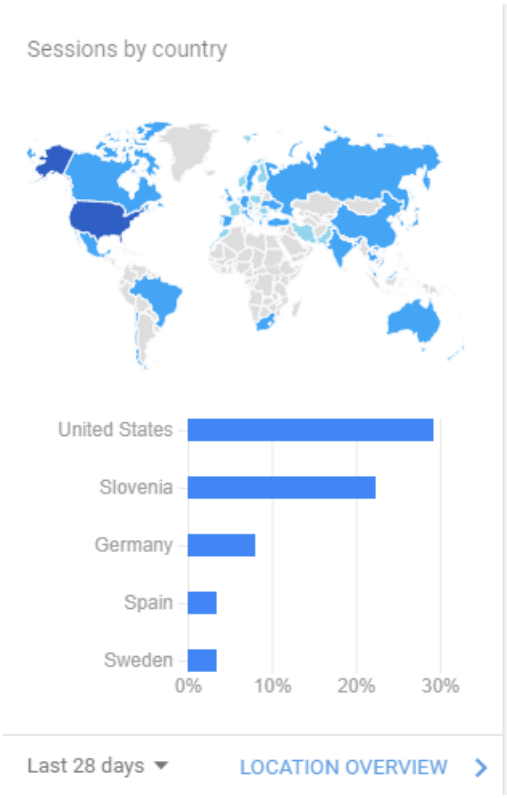
5 PREDSTAVITEV REZULTATOV

V tem poglavju bomo pregledali rezultate, ki so do sedaj na voljo. Po dveh tednih od izdaje že imamo nekaj odzivov uporabnikov. Odzivi, ki smo jih sprejeli, so bili zelo pozitivni, na splošno so uporabniki zadovoljni z delovanjem. Veliko jim pomeni, da lahko svoja opravila odprejo kjerkoli na svoji mobilni napravi. Prav tako so zapisali še kar nekaj funkcionalnosti, ki jih trenutno pogrešajo.

5.1 Analiza uporabe

V tem poglavju bomo pregledali nekatere podatke o uporabi v realnem svetu. Na razpolago imamo podatke o prvih dveh tednih po izdaji. Zavedati se moramo, da je PSA zaprtega tipa, torej so potencialni uporabniki že obstoječi uporabniki naše spletne platforme. Uporabniki so za PSA izvedeli preko email obvestila po izdaji, preko naše spletne strani ali pa so jo odkrili preko novega gumba v platformi.

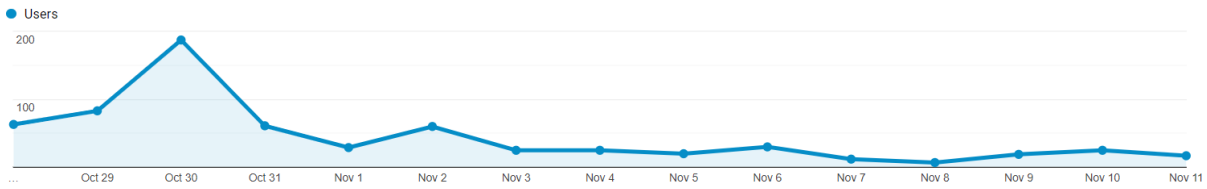
Po dveh tednih rezultatov lahko rečemo, da je PSA resnično globalna. Uporabniki so dostopali prav z vseh celin (razen Antarktike). Google Analytics je zabeležil podatke iz 50 različnih držav, čeprav je treba navesti, da je iz kar nekaj držav dostopal samo po en uporabnik. Na Sliki 27 je prikazan obarvan zemljevid držav, iz katerih so uporabniki dostopali, in pa relativno število uporabnikov najbolje zastopanih držav.



Slika 27 – Porazdeljenost uporabnikov po državah

Kot smo pričakovali, je največ uporabnikov dostopalo iz ZDA, saj je tam tudi največja prisotnost naše platforme. Prav tako smo pričakovali, da bo kar nekaj uporabnikov dostopalo iz Slovenije, kajti tudi tukaj smo že prisotni v nekaterih institucijah. Nekaj odstotkov uporabnikov prav gotovo predstavljajo naši zaposleni, ki so PSA preizkusili na svojih napravah. Nato si sledijo tri večje evropske države: Nemčija, Španija in Švedska. Takoj za njimi pa je prva azijska država, Tajvan.

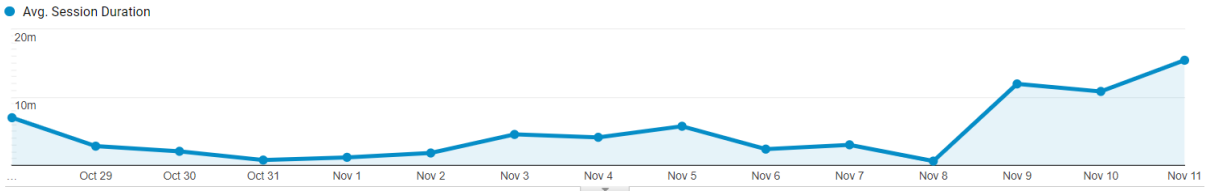
Če pogledamo strukturo dnevnih uporabnikov (Slika 28), vidimo začetni vrh, potem pa drastičen upad. To je razumljivo, saj to nekako sovpada z e-poštnim obveščanjem o aplikaciji. Kasneje se je število dnevnih uporabnikov nekako ustalilo.



Slika 28 – Število dnevnih uporabnikov skozi čas

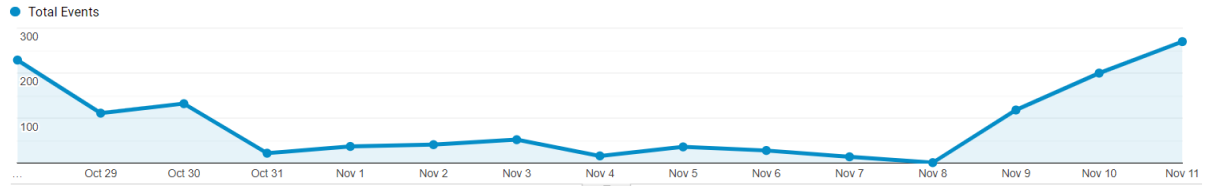
Zelo zanimiv graf predstavlja podatke o dolžini povprečne seje uporabnika. Kot vidimo, je po prvem dnevu bil čas seje nekoliko daljši, potem pa je hitro padel zelo nizko. Verjetno je razlog

v tem, da so se ob prvem obisku uporabniki le malo razgledali po aplikaciji brez namena resne uporabe. Na Sliki 29 lahko vidimo, da je v zadnjih dneh povprečen čas skokovito narasel. Iz tega lahko sklepamo, da so nekateri uporabniki začeli PSA uporabljati za resne naloge in so posledično dlje v sami aplikaciji .



Slika 29 – Povprečen čas obiska po dnevih

Našo ugotovitev potrjujeta tudi graf ogledov strani in graf dogodkov. Izstopajo predvsem dogodki, povezani z izvedbo opravil. V spodnji sliki (Slika 30) lahko vidimo, da je število dogodkov preraslo prve dni, ko je bilo dnevno veliko več uporabnikov.

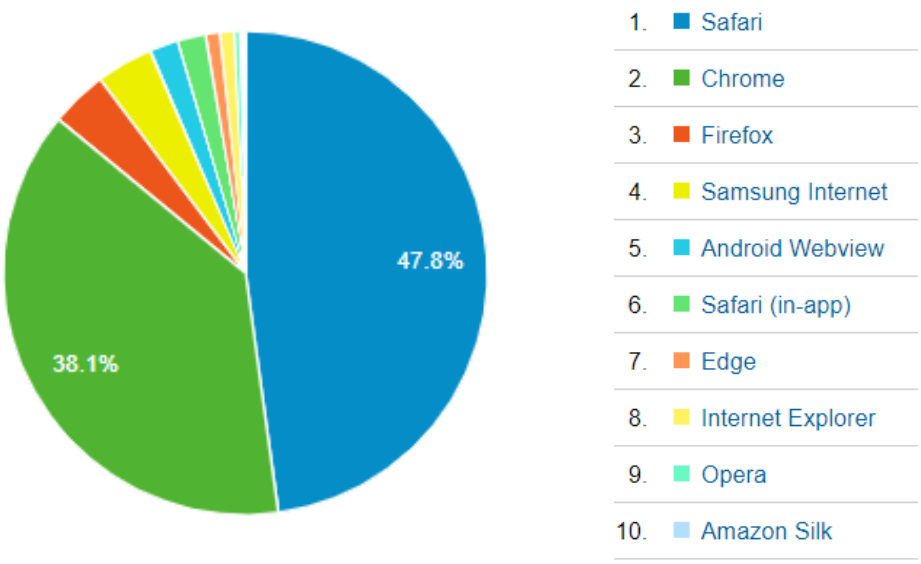


Slika 30 – Število izvedenih dogodkov po dnevih

Če pogledamo razdeljenost uporabnikov po kategoriji naprav, pričakovano prevladujejo uporabniki mobilnih telefonov. Kljub temu, da je aplikacija PSA načrtovana za mobilne naprave, pa je presenetljivo veliko uporabnikov prisotnih na osebnih računalnikih – skoraj četrtina. Le dobrih 6 % uporabnikov je obiskalo PSA preko tablice.

Zanimivo je tudi, da je najbolj razširjen operacijski sistem med uporabniki Applov iOS. S 45 odstotki krepko vodi pred sistemom Android (33 %), ki je v svetovnem merilu veliko bolj razširjen. Na tretjem mestu se nahaja Windows (11 %), ki je malenkost bolj razširjen med uporabniki osebnih računalnikov kot pa Macintosh (9 %). Uporabniki so za dostop do PSA uporabili tudi Linux in Chrome OS, vendar je njun delež zanemarljiv.

Med brskalniki tako prevladuje Safari, ki ima večinski delež med uporabniki iOS. Skoraj polovica uporabnikov (48 %) je dostopala preko brskalnika Safari. Na drugem mestu je Chrome (38 %). Ta dominira na platformi Android, čeprav imata dobršen delež tudi Samsung Internet in Mozilla Firefox.



Slika 31 – Porazdeljenost uporabljenih brskalnikov

Iz dogodkov, ki jih spremljamo, lahko dobimo tudi približno sliko o tem, koliko uporabnikov si je aplikacijo namestilo na domač zaslon. V tem obdobju smo zaznali 227 primerov, kjer se je seja začela z zagonom z domačega zaslona. To predstavlja 28 % vseh zagonov aplikacije. Če izvzamemo prvi teden, kjer je bilo največ obiska, lahko vidimo, da je ta številka precej višja, dobrih 40 %. Razlog je v tem, da je prvih nekaj dni bilo veliko uporabnikov, ki so aplikacijo samo na hitro odprli in odšli po ogledu prve strani. Odstotek takšnih pa vztrajno pada.

1.	App opened from home screen	Chrome	**143** (47.04%)
2.	App opened from home screen	Safari	**112** (36.84%)
3.	App opened from home screen	Samsung Internet	**39** (12.83%)
4.	App opened from home screen	Firefox	**8** (2.63%)
5.	App opened from home screen	Edge	**2** (0.66%)

Slika 32 – Dostopanje do aplikacije z domačega zaslona glede na brskalnik

Dogodki so med brskalniki nekoliko drugače razporejeni kot pa sami ogledi. Chrome in Safari sta zamenjala mesti. Lahko predvidevamo, da k temu prispeva enostavna namestitev s klikom

na gumb v brskalnikuChrome. Kljub temu pa je odstotek v Safari brskalniku zelo zadovoljiv, saj pomeni, da implementirano obvestilo o možnosti namestitve deluje.

Iz podatkov lahko razberemo uporabo gumba »Install«. Ta dogodek nam sicer ne pove točnega števila namestitev, saj gumb ni podprt v vseh brskalnikih. Kljub temu je bil uporabljen v 110 primerih. Od tega uporabnik v 10 % ni potrdil poziva za namestitev na domač zaslon.

5.2 Izboljšave trenutnih funkcionalnosti

V trenutni verziji aplikacije je še kar nekaj možnosti za izboljšave. Ena najpomembnejših je nalaganje opravila. Trenutno mora uporabnik prenesti opravilo z vsemi koraki in njihovo vsebino. Iz tega se zgradi objekt in shrani v bazo. To nalaganje bi se lahko prestavilo v ozadje, uporabniku pa bi prikazali podatke, takoj ko je prvi korak na voljo. Čeprav to nalaganje na stabilnih povezavah ni moteče, pa je lahko kar moteče na počasnih mobilnih omrežjih.

Za delovanje brez povezave bi bilo idealno, če bi dodali predpomnjenje slik. Trenutno so slike v opisu koraka v obliki URL povezave. Tako se tudi zapišejo v objekt, ki se kasneje uporablja za uporabo brez povezave. Takrat pa do teh URL povezav več ne moremo dostopati. Prikaže se simbolična slika.

Smiselno bi bilo tudi implementirati strategijo *najprej predpomnilnik, potem omrežje* za izbirne sezname. Tako bi se elementi seznamov, ki smo jih že obiskali, dodajali v predpomnilnik. Ko bi naslednjič odprli enak seznam, bi se elementi seznama v trenutku prikazali, a bi se v ozadju vseeno preverili podatki na omrežju in v skladu s tem bi se seznam posodobil.

5.3 Podpora v brskalnikih

Po končani implementaciji smo naredili tudi popis funkcionalnosti v različnih brskalnikih. Osredotočili smo se na najpopularnejše na svoji platformi. Potrebno je omeniti, da na sistemu iOS ostali proizvajalci ne morejo ponuditi svojega jedra brskalnika. Omejeni so na pogon

WebView. Podpora PSA se tudi razlikuje med zadnjimi verzijami sistema, saj se stalno dopolnjuje. V našem popisu smo uporabili zadnjo verzijo sistema (iOS 14).

	Android			iOS 14		Windows 10		
Funkcionalnost	Chrome	Firefox	Samsung I.	Safari	WebView	Chrome	Edge	Firefox
Zaznavanje omrežne povezave	DA	DA	DA	DA	DA	DA	DA	DA
Delovanje brez povezave	DA	DA	DA	DA	DA	DA	DA	DA
Sinhronizacija v ozadju	DA	NE	DA	NE	NE	DA	DA	NE
Zmožnost namestitve	DA	DA	DA	DA	NE	DA	DA	NE
Prikaz poziva k namestitvi	DA	DA	DA	NE	NE	DA	DA	NE
Podpora "Install" gumba	DA	NE	DA	NE	NE	DA	DA	NE
Zaznavanje nameščene PWA	DA	NE	NE	NE	NE	DA	NE	NE
WebAPK	DA	NE	DA	NE	NE	NE	NE	NE
Splash screen	DA	DA	DA	DA	NE	NE	NE	NE
Odpiranje povezave iz brskalnika v PWA	DA	NE	DA	NE	NE	NE	NE	NE
Deljenje avtorizacijskih podatkov z brskalnikom	DA	DA	DA	NE	NE	DA	DA	NE

Tabela 1 – Podprtost uporabljenih funkcionalnosti v različnih sistemih

Kot vidimo, Chrome podpira popolnoma vse implementirane funkcionalnosti. Presenetljivo dobro se je izkazal tudi Samsung Internet. Firefox dovolj dobro podpira osnovno obnašanje PSA, čeprav ima nekaj pomanjkljivosti pri nameščanju. Safari denimo ne podpira bolj naprednih opcij, brskalniki, ki so bazirani na pogonu WebView, pa na žalost še niso zmožni namestiti PSA na domač zaslon. So pa z zadnjo posodobitvijo iOS sistema pridobili podporo za storitvenega delavca.

Čeprav nekatere funkcionalnosti ne delujejo v vseh brskalnikih, pa to na končnega uporabnika ne bo preveč vplivalo. Vse je namreč implementirano z uporabo progresivnega izboljšanja.

6 SKLEP

Pričujoče naloge smo se lotili s specifičnim problemom v mislih. Sistem, ki je bil načrtovan za uporabo na zaslonu namiznega računalnika, smo hoteli približati uporabnikom mobilnih naprav z implementacijo PSA. Ker smo želeli rešitev za vse platforme, je bila odločitev za razvoj PSA najbolj smiselna, tudi s finančnega vidika. Ker so PSA tehnologije podprte že skoraj v vseh brskalnikih, lahko enako kodo uporabimo v vseh sistemih.

Aplikacija je bila uspešno implementirana. Podpira ključne funkcionalnosti platforme, ki so bile najbolj pomembne za prvo verzijo. V več tednih razvoja PSA smo se srečali z marsikaterim izzivom in večino izzivov smo tudi uspešno obšli. Posledično smo morali dodobra preučiti delovanje vseh podpornih tehnologij. To znanje in ugotovitve so zapisane v teoretičnem delu naloge.

Za nadaljnji razvoj PSA bi bilo potrebno zbrati čim več odzivov trenutnih uporabnikov in identificirati njihove potrebe ter želje. Skladno s tem bi lahko nekoliko prilagajali prioritete razvoja. Neodvisno od tega so nekatere funkcionalnosti za nadaljnji razvoj že načrtovane.

V naslednji iteraciji aplikacije je smiselna nadgradnja funkcionalnosti izvajanja opravila. V trenutni verziji smo se osredotočili predvsem na prikaz vseh vsebin. Velik doprinos uporabnikom pa bi bilo komentiranje in dodajanje rezultatov takoj po izvedbi posameznega koraka. Poleg tekstovnih rezultatov bi bilo smiselno dodati možnost zajema slik ali avdio posnetka. Te želje smo identificirali tudi v prvih odzivih uporabnikov.

Za uporabnike z velikim številom eksperimentov in opravil bi bila dobrodošla implementacija nekakšnega sistema iskanja ali poljubnega sortiranja seznamov.

Dolgoročno bi bila zanimiva funkcionalnost za laboratorijska opravila, denimo nek vnaprej pravilno nastavljen odštevalnik časa za opravila, kjer je pomembna dolžina izvajanja. Vključno s tem bi se lahko implementiralo sporočilo, ki bi uporabnika obvestilo o poteku časa, potem ko je že zapustil aplikacijo.

Tudi sicer bi bila potisna sporočila dobrodošla ob izbranih kolaboracijskih dogodkih znotraj ekipe. Kadar nekdo delegira opravilo drugemu uporabniku, zaključi opravilo od nekoga drugega ali pa je uporabnik označen v komentarju.

Med razvojem smo prepoznali tudi nekatere slabosti PSA. Čeprav so PSA podprte že v vseh modernih brskalnikih, obstaja velika razlika pri podpori novejših naprednih vmesnikov. To je bilo treba upoštevati med načrtovanjem nekaterih funkcionalnosti. Predvideti je potrebno obnašanje, kadar je potreben vmesnik na voljo, in obenem zagotoviti dovolj dobro alternativo za ostale brskalnike.

V nekaj mesecih razvoja smo spoznali precejšnjo neuveljavljenost PSA v širši javnosti. Povprečni uporabniki se sploh ne zavedajo njihovega obstoja in uporabnosti. Morda se z njimi še niso niti srečali ali pa jih celo uporabljajo, vendar niso seznanjeni z izrazom PSA oz. »PWA«. Ta izvorno angleška skovanka je že sama po sebi precej neposrečena, saj si je težko predstavljati, kaj točno predstavlja. Po drugi strani pa je uporabnikom le pomembno, da aplikacija deluje in izpolni cilj, ne glede na strokovno poimenovanje ali tehnologijo v ozadju.

Naslednji izziv predstavlja distribucija PSA. Več kot desetletje, od začetka pametnih telefonov, so se uporabniki učili, da aplikacijo najdemo v trgovini aplikacij posameznega sistema. Ta filozofija se bo počasi morala spreobrniti proti spletu, preden bodo PSA dosegle večji tržni delež. Morda pa se bodo trgovine prilagodile in omogočale tudi objavo spletnih aplikacij kot enakovrednega člana v ekosistemu. Nekateri koraki v tej smeri so že bili narejeni. Zagotovo pa bi ta preskok bil še hitrejši, če bi gigantom, ki obvladujejo obstoječi trg aplikacij, bilo to v interesu. Trenutno so precej zadovoljni s svojimi poslovnimi modeli, kjer za vsak nakup skozi domorodne aplikacije pobirajo provizijo.

Seveda pa k prepoznavnosti največ pripomore vseprisotnost. Če bodo uporabniki pogosto prihajali v stik s kvalitetnimi progresivnimi spletnimi aplikacijami, bodo postali pozorni na nov način interakcije. Tak moment pa bo sprožil tudi večje zanimanje naročnikov za investiranje v razvoj novih PSA.

7 LITERATURA IN VIRI

[1] S. Richard in P. LePage, „Web.dev,“ 6 Januar 2020. [Elektronski]. Available: https://web.dev/what-are-pwas/. [Poskus dostopa Maj 2020].

[2] C. Love, Progressive Web Application Development by Example, Birmingham - Mumbai: Packt Publishing, 2018, p. 19.

[3] A. Russel, „Medium.com,“ 10 Avgust 2015. [Elektronski]. Available: https://medium.com/@slightlylate/progressive-apps-escaping-tabs-without-losing-our-soul-3b93a8561955. [Poskus dostopa Maj 2020].

[4] T. Steiner, Interviewee, *Project Fugu interview: Bridging the app gap.* [Intervju]. 28 Februar 2020.

[5] A. Austin, „Medium,“ 4 November 2015. [Elektronski]. Available: https://medium.com/swlh/mobile-app-developers-are-suffering-a5636c57d576. [Poskus dostopa September 2020].

[6] I. Mansoor, „Business of Apps,“ 30 Julij 2020. [Elektronski]. Available: https://www.businessofapps.com/data/app-statistics/. [Poskus dostopa September 2020].

[7] Statista, „Statista,“ Avgust 2017. [Elektronski]. Available: https://www.statista.com/statistics/325926/monthly-app-downloads-of-us-smartphone-users/. [Poskus dostopa Oktober 2020].

[8] P. Saccomani, „MobiLoud,“ [Elektronski]. Available: https://www.mobiloud.com/blog/mobile-apps-vs-the-mobile-web/. [Poskus dostopa September 2020].

[9] R. Gautam, „Facebook Engineering,“ 9 Marec 2016. [Elektronski]. Available: https://engineering.fb.com/android/how-we-built-facebook-lite-for-every-android-phone-and-network/. [Poskus dostopa September 2020].

[10] B. Lawson, „WWW: World Wide Web, not Wealthy Westerners' Web,“ 30 September 2016. [Elektronski]. Available: https://speakerdeck.com/brucel/parisweb-paris-30-september-2016. [Poskus dostopa Avgust 2020].

[11] W3C, „w3.org,“ 27 Julij 2020. [Elektronski]. Available: https://www.w3.org/TR/appmanifest/. [Poskus dostopa September 2020].

[12] Mozilla, „developer.mozilla.org,“ 25 Junij 2020. [Elektronski]. Available: https://developer.mozilla.org/en-

US/docs/Web/Progressive_web_apps/Add_to_home_screen#Manifest. [Poskus dostopa Avgust 2020].

[13] web.dev, „web.dev,“ 28 Maj 2020. [Elektronski]. Available: https://web.dev/add-manifest/. [Poskus dostopa September 2020].

[14] Mozilla, „MDN web docs,“ 2020. [Elektronski]. Available: https://developer.mozilla.org/en-US/docs/Web/Manifest/orientation. [Poskus dostopa Oktober 2020].

[15] Mozilla, „MDN web docs,“ 2020. [Elektronski]. Available: https://developer.mozilla.org/en-US/docs/Web/Manifest/scope. [Poskus dostopa September 2020].

[16] M. Firtman, „Medium,“ 11 April 2016. [Elektronski]. Available: https://medium.com/@firt/service-workers-replacing-appcache-a-sledgehammer-to-crack-a-nut-5db6f473cc9b. [Poskus dostopa Avgust 2020].

[17] E. Bidelman, „html5rocks,“ 26 Julij 2010. [Elektronski]. Available: https://www.html5rocks.com/en/tutorials/workers/basics/. [Poskus dostopa Avgust 2020].

[18] J. Posnick, „Web Fundamentals,“ 2019. [Elektronski]. Available: https://developers.google.com/web/fundamentals/primers/service-workers/registration. [Poskus dostopa Oktober 2020].

[19] M. Gaunt, „Web Fundamentals,“ [Elektronski]. Available: https://developers.google.com/web/fundamentals/primers/service-workers. [Poskus dostopa September 2020].

[20] J. Archibald, „Web Fundamentals,“ 2019. [Elektronski]. Available: https://developers.google.com/web/fundamentals/primers/service-workers/lifecycle. [Poskus dostopa September 2020].

[21] J. Archibald, „The Offline Cookbook,“ 9 December 2014. [Elektronski]. Available: https://web.dev/offline-cookbook/. [Poskus dostopa September 2020].

[22] P. LePage, „Web - developers.google,“ September 2020. [Elektronski]. Available: https://developers.google.com/web/updates/2018/06/a2hs-updates. [Poskus dostopa September 2020].

[23] M. Firtman, „Medium,“ 2 Januar 2020. [Elektronski]. Available: https://medium.com/@firt/progressive-web-apps-in-2020-c15018c9931c. [Poskus dostopa September 2020].

[24] Angular, „Angular,“ 2020. [Elektronski]. Available: https://angular.io/guide/service-worker-communications. [Poskus dostopa September 2020].

[25] 9to5Mac, „9to5Mac,“ 21 Oktober 2011. [Elektronski]. Available: https://9to5mac.com/2011/10/21/jobs-original-vision-for-the-iphone-no-third-party-native-apps/. [Poskus dostopa Maj 2020].

[26] R. Ritchie, „iMore,“ 5 Marec 2018. [Elektronski]. Available: https://www.imore.com/history-app-store-year-zero. [Poskus dostopa Maj 2020].

[27] E. Marcotte, „A list apart,“ 25 Maj 2010. [Elektronski]. Available: https://alistapart.com/article/responsive-web-design/. [Poskus dostopa Maj 2020].

[28] C. Liebel, „International JavaScript Conference,“ 31 Marec 2020. [Elektronski]. Available: https://javascript-conference.com/blog/making-the-web-more-powerful-with-project-fugu/. [Poskus dostopa Maj 2020].

[29] Mozilla, „MDN Web Docs,“ [Elektronski]. Available: https://developer.mozilla.org/en-US/docs/Web/API/ExtendableEvent/waitUntil. [Poskus dostopa Oktober 2020].

[30] Google, „Web Fundamentals,“ 2019. [Elektronski]. Available: https://developers.google.com/web/ilt/pwa/introduction-to-push-notifications#what_are_push_notifications. [Poskus dostopa 2020].

[31] J. Posnick, „web.dev,“ 20 Junij 2019. [Elektronski]. Available: https://web.dev/google-search-sw/. [Poskus dostopa September 2020].

[32] W3C Community Group Draft Report, „W3C Community Group Draft Report,“ 28 Maj 2020. [Elektronski]. Available: https://wicg.github.io/background-sync/spec/. [Poskus dostopa September 2020].

[33] Mozilla, „MDN web docs,“ [Elektronski]. Available: https://developer.mozilla.org/en-US/docs/Glossary/Progressive_Enhancement. [Poskus dostopa Oktober 2020].

[34] Mozilla, „MDN web docs,“ 5 Maj 2020. [Elektronski]. Available: https://developer.mozilla.org/en-US/docs/Learn/Performance/perceived_performance. [Poskus dostopa September 2020].

[35] Akamai, „Akamai,“ 14 September 2009. [Elektronski]. Available: https://www.akamai.com/uk/en/about/news/press/2009-press/akamai-reveals-2-seconds-as-the-new-threshold-of-acceptability-for-ecommerce-web-page-response-times.jsp. [Poskus dostopa September 2020].

[36] UXPin, „uxpin.com,“ Oktober 2020. [Elektronski]. Available: https://www.uxpin.com/studio/blog/a-hands-on-guide-to-mobile-first-design/. [Poskus dostopa Avgust 2020].

[37] „web.dev,“ 10 Junij 2020. [Elektronski]. Available: https://web.dev/rail/. [Poskus dostopa September 2020].

[38] „WorkBox,“ 28 Januar 2020. [Elektronski]. Available: https://developers.google.com/web/tools/workbox/modules/workbox-cli#options_used_by_generatesw. [Poskus dostopa Oktober 2020].

[39] A. Bar, „What web can do today?,“ Maj 2020. [Elektronski]. Available: https://whatwebcando.today/. [Poskus dostopa Avgust 2020].

[40] A. Bovens, „Dev.Opera,“ 31 Marec 2016. [Elektronski]. Available: https://dev.opera.com/blog/web-app-install-banners/. [Poskus dostopa September 2020].

[41] G. Cselle, „Tales of Creation by Gabor Cselle,“ 23 Oktober 2012. [Elektronski]. Available: https://blog.gaborcselle.com/2012/10/every-step-costs-you-20-of-users.html. [Poskus dostopa Avgust 2020].

[42] N. Hoizey, „Nicolas Hoizey,“ 12 Januar 2017. [Elektronski]. Available: https://nicolas-hoizey.com/articles/2017/01/12/how-much-data-should-my-service-worker-put-upfront-in-the-offline-cache/. [Poskus dostopa September 2020].

[43] P. LePage, „web.dev,“ 6 Avgust 2020. [Elektronski]. Available: https://web.dev/get-installed-related-apps/. [Poskus dostopa Oktober 2020].

[44] P. LePage, „web.dev,“ 12 Maj 2020. [Elektronski]. Available: https://web.dev/persistent-storage/. [Poskus dostopa September 2020].

[45] P. LePage, „web.dev,“ 7 Maj 2020. [Elektronski]. Available: https://web.dev/storage-for-the-web/. [Poskus dostopa Oktober 2020].

[46] C. Love, „Love2Dev,“ 8 Oktober 2019. [Elektronski]. Available: https://love2dev.com/pwa/add-to-homescreen/. [Poskus dostopa Avgust 2020].

[47] A. Osmani, „Medium,“ 15 Avgust 2016. [Elektronski]. Available: https://medium.com/dev-channel/offline-storage-for-progressive-web-apps-70d52695513c. [Poskus dostopa Avgust 2020].

[48] J. Posnick, „web.dev,“ 5 November 2018. [Elektronski]. Available: https://web.dev/workbox/. [Poskus dostopa September 2020].

[49] T. Steiner, „web.dev,“ 29 Junij 2020. [Elektronski]. Available: https://web.dev/progressively-enhance-your-pwa/. [Poskus dostopa September 2020].

[50] J. Wilander, „WebKit,“ 24 Marec 2020. [Elektronski]. Available: https://webkit.org/blog/10218/full-third-party-cookie-blocking-and-more/. [Poskus dostopa September 2020].