LATVIJAS UNIVERSITĀTE

DATORIKAS FAKULTĀTE

PROGRESĪVO TĪMEKĻA LIETOTŅU IZSTRĀDE ANGULAR IETVARĀ

BAKALAURA DARBS

Autors: Edgars Joja

Studenta apliecības Nr.: ej14012

Darba vadītājs: prof., Dr. dat. Uldis Straujums

RĪGA 2018

ANOTĀCIJA

Tradicionālo mobilo tīmekļa lietotņu skaits turpina pieaugt, samazinot iespēju, ka jaunas lietotnes tiks pamanītas, instalētas un lietotas. Progresīvo tīmekļa lietotņu mērķis ir apvienot labākās tradicionālo mobilo lietotņu un tīmekļa vietņu īpašības, lai uzlabotu lietotāja pieredzi tīmekļa vietnē.

Pētījuma mērķis ir noskaidrot, kādas ir izstrādes iespējas progresīvo tīmekļa lietotņu realizēšanai Angular ietvarā un tieši kādi ir ieguvumi un atšķirības pār tradicionālo mobilo lietotni vai tīmekļa vietni.

Pētījuma rezultātā ir izstrādāta progresīva tīmekļa lietone Angular ietvarā trīs versijās, kur divas no tām ir ar progresīvo tīmekļa lietotņu iezīmēm un viena ir tipiska Angular ietvara lietotne, bez īpašiem uzlabojumiem, kā arī ir veikts šo versiju salīdzinājums.

Atslēgvārdi: progresīva tīmekļa lietotne, Angular ietvars, tīmeklis

ABSTRACT

DEVELOPMENT OF PROGRESSIVE WEB APPLICATIONS IN THE ANGULAR FRAMEWORK

Number of native mobile applications continues to increase, making it harder for new applications to be discovered, installed and used. Progressive web applications aim to combine best characteristics of native mobile applications and websites to improve user experience in a website.

Aim of the research is to find out what are the capabilities of progressive web applications development in Angular framework and exactly what are the advantages and differences over native mobile application or traditional website.

The outcome of the research is progressive web application, developed in Angular framework, in three versions, where two of them are with progressive web applications characteristics and one is typical Angular framework application without special improvements, as well as done comparision of these versions.

Keywords: progressive web application, Angular framework, web

SATURS

APZĪMĒJUMU SARAKSTS

3G	Trešās paaudzes bezvadu mobilo telekomunikāciju tehnoloģija un standartu kopa, kas spēj nodrošināt datu pārraides ātrumu ar vismaz 200 kbit/s.
JavaScript	Augsta līmeņa programmēšanas valoda, kas tiek plaši izmantota tīmekļa vietņu izstrādē, lai nodrošinātu klienta puses interaktivitāti (interactivity).
HTTP/2	HTTP protokola uzlabota versija, kas tiek lietota tīmeklī datu pārsūtīšanai un formatēšanai. Izveidota, lai risinātu daudzus no HTTP/1.1 protokola ierobežojumiem un problēmām.
HTTP/1.1	HTTP protokola uzlabota versija, kas tiek lietota tīmeklī datu pārsūtīšanai un formatēšanai. Izveidota, lai risinātu HTTP/1 protokola ierobežojumus un problēmas.
HTML	Valoda, kas, izmantojot speciālus kodus, nosaka hiperteksta dokumenta atveidojumu displeja ekrānā gadījumā, ja tiek lietotas interneta globālā tīmekļa lappuses. (HyperText Markup Language)
CSS	Kaskādisku stilu saraksts, kas tiek izmantots, lai attēlotu HTML izveidotu tīmekļa lapas struktūru un izskatu. (Cascading Style Sheets)
SVG	XML bāzēts divdimensiju vektorgrafikas formāts, kas atbalsta interaktivitāti un animācijas. (Scalable Vector Graphics)
JPEG	Plaši izmantota metode digitālu attēlu saspiešanai. Saspiešanas daudzums var tikt mainīts, atkarībā no nepieciešamā attēla atmiņas apjoma vai attēla kvalitātes.
PNG	Bitkartētu grafikas datņu formāts, kas izstrādāts kā formāta GIF aizstājējs un uz ko neattiecas ar formātu GIF saistītie juridiskie ierobežojumi. (Portable Network Graphics)
GIF	Bitkartētas grafikas datnes formāts, ko parasti izmanto operatīvās informācijas sistēmās, jo tas nodrošina tādas augstas izšķirtspējas grafikas kā skenētu attēlu efektīvu saspiešanu. (Graphics Interface Format)
URI	Kods dokumenta atsevišķas kopijas adreses vai cita resursa identificēšanai internetā. (universal resource identifier)

URL	Kods atsevišķa dokumenta kopijas vai jebkura cita resursa vai pakalpojuma identificēšanai internetā. (universal resource locator)
UI	Lietotāja interfeiss. Daļa no programmas, ar ko mijiedarbojas lietotājs. (user interface)
Serveris	Funkcionāls datoru tīkla bloks (dators, stacija), kas nodrošina citām tā stacijām koplietošanas pakalpojumus (piem., datņu serveris, drukas serveris, pasta serveris).
Skripts	Instrukciju virkne, kas nosaka, kā programmai jāveic kāda specifiska procedūra, piem., ieiešana elektroniskā pasta sistēmā. Dažādās programmās skripta iespējas ir jau iebūvētas, dažus skriptus veido automātiski, pierakstot, kādus taustiņus un komandu izvēlnes lietotājs izmanto procedūru laikā. Globālā tīmekļa kontekstā skripts ir programma, kas atrodas kādā tīmekļa serverī un apstrādā no pārlūkprogrammas saņemtos pieprasījumus. (script)
API	Lietojumprocesos izmantojama pilna programmatūras funkciju specifikācija, kā arī šo funkciju izmantošanas procedūru apraksts. (application programming interface)
HTTPS	Datu apmaiņas protokols, kas tiek lietots tīmeklī, lai apstrādātu vietņu lapu pieprasījumus. Satur papildus drošības slāni, lietojot drošligzdu slāņa (SSL) tehnoloģiju.
JSON	Teksta bāzēts datu apmaiņas formāts, kas ir lietots, lai pārsūtītu strukturētus datus.
MITM	Datordrošības uzbrukuma tips, kur uzbrucējs noklausās un iespējams modificē datu apmaiņu starp diviem komunicētājiem, kuri domā, ka veic tiešu datu apmaiņu viens ar otru. (man in the middle attack)
TypeScript	Atvērtā pirmkoda programmēšanas valoda, ko izstrādāja un uztur "Microsoft". Tā ir strikta sintakses virskopa JavaScript programmēšanas valodai.
ISBN	Unikāls ciparu identifikators komerciālajām grāmatām. (International Standard Book Identifier)
Android	Mobilo ierīču operētājsistēma, ko izstrādājusi "Google", pamatā ņemot modificētu Linux operētājsistēmas kodolu. Paradzēta skārienekrānu ierīcēm.

Servisskripts Programmas kods, kas reģistrēts interneta pārlūkā un spēj pārtvert un atgriezt datus uz tīmekļa pieprasījumiem atbilstošajā programmā, tādējādi spējot nodrošināt tīmekļa vietnes darbību bezsaistē vai pašpiegādes ziņojumu pārsūtīšanu. (Service Worker)

Bestsellers Grāmata, kas iekļauta visvairāk pārdoto grāmatu sarakstā, parasti balstoties uz publikāciju industrijas, grāmatu apmaiņu datiem un bibliotēku statistikas.

IEVADS

Progresīvās tīmekļa lietotnes izmanto jaunākās tīmekļa tehnoloģijas, lai uzlabotu lietotāja pieredzi, apvienojot labāko no tīmekļa vietņu un mobilo lietotņu īpašībām. Mūsdienās, tīmekļa un mobilo ierīču lietotāju skaitam palielinoties, ir svarīgi nodrošināt uzticamu tīmekļa vietnes darbību jebkādos apstākļos, ņemot vērā, ka vidējais interneta savienojuma ātrums un mobilās ierīces veiktspēja pasaulē ir zemāka, nekā attīstītajās valstīs ir pierasts.

Darbā aprakstītas progresīvo tīmekļa lietotņu vispārīgās īpašības un to ieguvumi, salīdzinot ar parastām tīmekļa vietnēm vai mobilajām lietotnēm, kā arī Angular ietvara, kas izmantots pētījuma veikšanā, vispārējā arhitektūra un darbības principi.

Pētījuma mērķis ir izpētīt Angular ietvara piedāvātās iespējas progresīvo tīmekļa lietotņu realizācijai, iztrādājot progresīvu tīmekļa lietotni, kā arī veikt salīdzinājumu starp progresīvu tīmekļa lietotni un parastu tīmekļa lietotni dažādās tīmekļa vietnei svarīgās metrikās.

Darba uzdevumi ir veikt teorētisku izpēti, iegūstot zināšanas par progresīvo tīmekļa lietotņu uzbūves un darbības principiem, lai varētu tās pielietot praktiski, izstrādājot progresīvu tīmekļa lietotni Angular ietvarā un veicot izstrādātās lietotnes metriku salīdzināšanu.

Darbs sastāv no sešām nodaļām. Pirmajā nodaļā ir aprakstīta pašreizējā tirgus situācija un veikts problēmas pamatojums. Otrajā nodaļā ir aprakstītas progresīvo tīmekļa lietotņu īpašības un biežāk lietotie risinājumi. Trešajā nodaļā ir īsi aprakstīti Angular ietvara darbības principi. Ceturtajā nodaļā ir tehnisks apraksts progresīvās tīmekļa lietotnes realizācijas iespējām Angular ietvarā. Piektajā un sestajā nodaļā ir veikti dažādu metriku mērījumi autora izstrādātajai lietotnei.

1. PAŠREIZĒJĀS SITUĀCIJAS RAKSTUROJUMS

Lai gan aizvien vairāk laika tiek pavadīts lietojot mobilās ierīces un mobilās lietotnes, šis laiks tiek pavadīts aizvien mazākā skaitā lietotņu. Lietotāji instalē mazāk lietotņu un izmanto tikai dažas no tām, kas ir instalētas. Mēģinājums ielauzties tirgū ar jaunu lietotni ir gandrīz neiespējams, nemaz neminot, ka dārgs.

Saskaņā ar 2016. gada ComScore atskaiti [32], vidējā persona iztērē 84% laika mobilajās ierīcēs, lietojot tikai piecas populārākās lietotnes. Planšetdatoros šis skaitlis ir vēl lielāks, 95% laika iztērētam piecās populārākajās lietotnēs.

ComScore atskaite arī prezentē datus, kas parāda, ka lielas audiences daudz vieglāk ir iespējams sasniegt mobilajā tīmekļa vietnē, nekā tradicionālajā lietotnē. Eksistē gandrīz 600 miljonu tīmekļa vietņu, kas sasniedz audiences, kas ir lielākas nekā pieci miljoni apmeklētāju – gandrīz četras ar pusi reizes vairāk nekā tradicionālās lietotnes ar līdzīgām audiencēm. Populārākajām 1000 mobilajām tīmekļa lapām ir gandrīz trīs reizes lielāka audience, nekā 1000 populārākajām lietotnēm un to audiences pieaug divreiz atrāk nekā tradicionālo lietotņu audiences.

Panākt, lai lietotājs instalē un lieto lietotni nav viegli. Lietotājam ir jāuzzina par vietni (tradicionāli reklamējot tīmeklī vai kādā vietnē). Viņiem ir jāapmeklē lapa lietoņu veikalā. Tad viņiem jānospiež instalācijas poga. Viņiem ir jāpiešķir lietotnei dažādas tiesības. Tad jāgaida, kamēr lietotne tiks lejupielādēta un instalēta. Visbeidzot tā ir jāatver vismaz vienu reizi un varbūt pat jālieto. Vairāki pētījumi ir parādījuši, ka vidēji 20% potenciālie lietotāji tiek zaudēti katrā no aprakstītajiem soļiem [1].

2. PROGRESĪVĀS TĪMEKĻA LIETOTNES

Progresīvās tīmekļa lietotnes ir jauna moderno tīmekļa lietotņu forma. Šīs lietotnes izmanto jaunākās tīmekļa iespējas, lai apvienotu unikālās tradicionālo mobilo lietotņu iespējas ar tīmekļa priekšrocībām.

Progresīvās tīmekļa lietotnes sāk kā parastas mājaslapas, bet, lietotājam ar to darbojoties, tās progresīvi iegūst jaunas spējas. Tās pārveidojas no tīmekļa lapas uz kaut ko, kas ir līdzīgāks tradicionālajai lietotnei [1].

Progresīvās tīmekļa lietotnes ir tīmekļa lietotnes, kurām piemīt sekojošas iezīmes:

- Augsta veiktspēja (high performance),
- Progresivitāte (progressivity),
- Atpazīstamība (discoverability),
- Sasaiste (linkability),
- Reaģētspēja (responsivity),
- Līdzība tradicionālajām lietotnēm (native app-like),
- Savienojamības neatkarība (connection independant),
- Atkal-iesaistāmība (re-engageability),
- Instalējamība (installability),
- Drošība (safety) [1, 2, 3].

Lai gan visi augstākminētie nosacījumi ir ieteicami, ir citas progresīvo tīmekļa lietotņu definīcijas, kas pieļauj tīmekļa vietni uzskatīt par progresīvo tīmekļa lietotni, ja interneta pārlūks piedāvā to instalēt mobilajā ierīcē, kas nozīmē, ka minimālie nosacījumi, lai vietni uzskatītu par progresīvo tīmekļa lietotni ir:

- Vietne tiek servēta, izmantojot HTTPS
- Spēj ielādēties bezsaistē
- Reģistrē manifesta failu, kas satur vismaz sekojošos datus:
  - "name" – lietotnes nosaukums,
  - "short_name" – lietotnes saīsināts nosaukums,
  - "start_url" – lapa, kurai jāielādējas, startējot lietotni,
  - PNG formāta ikonu – lietotnes ikona [33].

2.1. Lietotnes veiktspēja

Veiktspējai tīmekļa vietnē ir ievērojama loma veiksmīga rezultāta sasniegšanā, jo vietnes ar augstāku veiktspēju piesaista un notur lietotājus labāk, nekā vietnes ar sliktu veiktspēju [4].

Balstoties uz vairāk nekā 10000 mobilo tīmekļa domēnu (domain) analīzi, tika atklāts, ka vidējais mobilās vietnes ielādes laiks ir 19 sekundes, lietojot 3G savienojumu, un 53% mobilo vietņu tiek pamestas, ja lapas ielāde aizņem vairāk nekā trīs sekundes [5].

2.1.1. Kādus datus sūtīt

Visātrākais un vislabāk optimizētais resurss ir resurss, kurš netiek sūtīts. Nevajadzīgi resursi ir lietotnē jālikvidē.

Vietnē lietotos, ieskaitot trešo pušu, resursus jāglabā uz vietnes servera. Tas nodrošina to, ka šie resursi būs vienmēr pieejami. Katra resursa veikstspēja ir jāizmēra un jānovērtē attiecīgi pret tā vērtību. Bieži vietnes satur resursus, kas nav vajadzīgi un samazina vietnes veiktspēju, nedodot lielu labumu lietotājam. Respektīvi, ir jānoskaidro vai resurss dod gana lielu labumu, lai resursa iekļaušana vietnē, tam potenciāli samazinot kopējo veiktspēju, ir izdevīga [6].

Ir jāizvērtē vai konkrētus resursus nav iespējams aizvietot ar līdzīgiem, veiktspēju mazāk ietekmējošiem resursiem. Viena no populārākajām JavaScript bibliotēkām – "jQuery"– ievērojami atvieglo darbu programmētājiem, taču ir pieejamas veiktspēju mazāk ietekmējošas alternatīvas, piemēram, "Zepto" bibliotēka [4].

2.1.2. Kā sūtīt datus

Ja ir zināms, kādus resursus sūtīt, lai lietotne būtu vizuāli skaista un funkcionāla, tad nākošais solis ir tos pārsūtīt efektīvi.

Migrācija uz HTTP/2 protokolu var nodrošināt labāku datu sūtīšanas vaiktspēju, adresējot daudzas HTTP/1.1 protokola problēmas, kā piemēram vienlaicīgu pieprasījumu limiti un galveņu (headers) nesaspiešana [4, 7].

Modernās tīmekļa vietnes vidēji sūta salīdzinoši daudz JavaScript un CSS datu. HTTP/1 protokola vidēs bija ļoti populāri šos datus sakopot relatīvi lielos failos, lai veidotos pēc iespējas mazāks skaits pieprasījumu pēc šiem resursiem, dēļ vairāku pieprasījumu neefektivitātes. HTTP/2 protokolā tā vairs nav – daudz efektīvāk ir veikt vairākus vienlaicīgus pieprasījumus saprātīgās robežās. Tādējādi, veiktspējas uzlabošanas nolūkos, kodu var sadalīt mazākās daļās un veikt pieprasījumus konkrētās vietnes lapās tikai tam kodam, kas tiks lietots konkrētajā lapā, samazinot kopējo datu sūtīšanas daudzumu [4].

2.1.3. Cik daudz datus sūtīt

Sūtāmos datus – tekstu, bildes, u.c. - ir vēlams saspiest, lai minimizētu kopējo sūtāmo datu daudzumu.

Jebkuram teksta resursam var noņemt nevajadzīgas atstarpes, komentārus un jebko, kas neietekmē funkcionalitāti. Ja vietnē tiek izmantoti SVG elementi, tad arī tie var tikt optimizēti, jo ir uz teksta bāzēti resursi.

Servera konfigurēšana, lai saspiestu datus ar dažādiem algoritmiem, piemēram GZIP vai Brotli, var ievērojami samazināt pārsūtāmo datu daudzumu, it īpaši teksta resursiem.

Attēlu saspiešana ir svarīga, jo bieži tieši attēli ir resurss, kas aizņem lielu daļu no kopējā sūtāmā datu daudzuma tīmeklī. Saskaņā ar HTTP Archive vietni (https://httparchive.org/), 60% datu, kas tiek pārsūtīti, lai attēlotu tīmekļa vietni ir JPEG, PNG vai GIF faili [4, 8].

2.2. Progresivitāte

Progresīvām tīmekļa lietotnēm ir jāspēj darboties uz jebkuras ierīces, kas spēj pārlūkot tīmekli. Atkarībā, no tā, kādas iespējas ir pieejamas lietotāja ierīcē un interneta pārlūkā, progresīvai tīmekļa lietotnei ir pēc iespējas vairāk tās jāizmanto. Tas nozīmē, ka lietotājiem ar vecāku interneta pārlūka versiju vai sliktāku interneta savienojumu joprojām ir pieejams vietnes pamata saturs, bet lietotājiem, kas izmanto jaunāku interneta pārlūku un labāku interneta savienojumu ir pieejama tā pati lapa, tikai ar papildus uzlabojumiem, atkarībā no interneta pārlūka iespējām [3].

2.3. Atpazīstamība

Progresīvās tīmekļa lietotnes ir tīmekļa vietnes, tādēļ meklētājprogrammas spēs tās vienkārši indeksēt un uzrādīt meklējumu rezultātos. Tā ir ievērojama priekšroka pār tradicionālajām lietotnēm, kuras ir daudz sarežģītāk atrast, lietojot meklējumprogrammas (search engines). Bieži vien, ja tīmekļa vietnei ir pieejama arī mobilā lietotne, tad vietne parādīs uznirstošo logu (popup), kas piedāvās lietotājam instalēt mobilo lietotni, lai gan ir veikti pētījumi, kas parāda, ka vietne zaudē potenciālos lietotājus ar šādu agresīvu reklāmas politiku [1, 3].

2.4. Sasaiste

Sasaiste ir vēl viena iezīme, kas mantota no tīmekļa vietnēm – progresīvai tīmekļa lietotnei ir jāizmanto URI, lai norādītu pašreizējo lietotnes stāvokli. Tas ļaus lietotnei saglabāt vai pārlādēt savu stāvokli, ja lietotājs pievieno lapu grāmatzīmēm vai kopīgo lapas URL [3].

2.5. Reaģētspēja

Progresīvas tīmekļa lietotnes lietotāja saskarnei ir vienmēr jāietilpst un korekti jāparāda lapas saturs jebkurā ierīcē, neatkarīgi no tās konstrukcijas un ekrāna izmēra [3].

2.6. Līdzība tradicionālajām lietotnēm

Progresīvajām tīmekļa lietotnēm ir jāizskatās, kā tradicionālai mobilai lietotnei un jābūt veidotai uz lietotnes čaulas (app shell) modeļa [3].

Lietotnes čaulas modelis ir veids, kā izstrādāt progresīvu tīmekļa lietotni, lai tā uzticami un tūlītēji ielādējas lietotāju ekrānos, līdzīgi, kā tas notiek tradicionālajās lietotnēs. Lietotnes čaula ir minimālais HTML, CSS un JavaScript kods, kas nepieciešams, lai nodrošinātu lietotāja saskarni un var nodrošināt tūlītēju, uzticami labu veiktspēju lietotāja atkārtotā apmeklēšanas reizē, kad ir saglabāts interneta pārlūka kešatmiņā.

Lietotnes čaula ir īpaši piemērota vienas lapas lietotnēm, kas balstās uz JavaScript arhitektūru. Šī pieeja paļaujās uz lietotnes čaulas saglabāšanu kešatmiņā (izmantojot servisskriptu), lai spētu ātri startēt lietotni. Pēc tam saturs tiek dinamiski ielādēts katrai lapai, lietojot JavaScript.

Lietojot lietotnes čaulas modeli ar servisskriptu iegūst vairākas priekšrocības:

- Uzticama veiktspēja, kas ir konstanti ātra. Atkārtoti apmeklējumi ir ļoti ātri ielādes ziņā. Statiski resursi un lietotāja saskarne (HTML, JavaScript, attēli, CSS) tiek saglabāti kešatmiņā pirmā apmeklējuma laikā, lai varētu tikt tūlītēji ielādēti atkārtotos apmeklējumos,
- Tradicionālajām lietotnēm līdzīga mijiedarbība. Adaptējot lietotnes čaulas modeli, ir iespējams iegūt tūlītēju, līdzīgu tradicionālajām lietotnēm, navigāciju un mijiedarbību ar lapas elementiem, papildinātu ar bezsaistes atbalstu,
- Ekonomiska datu lietošana. Lietotnes čaulas modelis samazina datu lejupielādes daudzumu.

Lietotnes čaula, lietojot servisskriptu, ir spēcīgs modelis bezsaistes interneta pārlūka kešatmiņas izmantošanai, kā arī dod ievērojamus veiktspējas uzlabojumus, tūlītēji ielādējoties progresīvas tīmekļa lietotnes atkārtotos apmeklējumos [9].

2.7. Savienojamības neatkarība

Pretēji parastām tīmekļa vietnēm, progresīvās tīmekļa lietotnes nav atkarīgas no lietotāja interneta savienojuma. Kad lietotājs apmeklē progresīvo tīmekļa lietotni, tā reģistrēs servisskriptu, kas spēj noteikt un reaģēt uz izmaiņām lietotāja savienojumā. Tas spēj piedāvāt visas vietnes lapas iespējas lietotājiem, kuri ir bezsaistē, tiešsaistē vai ir ar nekvalitatīvu tīmekļa savienojumu.

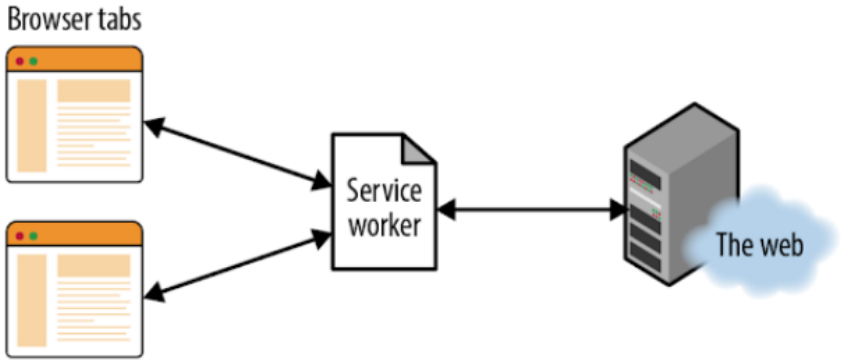
Lietotājiem būtu jāspēj lietot progresīvo tīmekļa lietotni arī situācijās, kad nav interneta savienojuma un pat veikt dažādas darbības, piemēram, publicēt ziņas, zinot, ka šī darbība tiks automātiski izpildīta tiklīdz būs pieejams interneta savienojums, pat ja lietotne un interneta pārlūks ir aizvērts.

Progresīvās tīmekļa lietotnes lietotājam piedāvā jaunu uzticamības slāni, pretī iegūstot lietotāja uzticību, kas līdz šim bija pieejama tikai tradicionālajām lietotnēm. Lietotājs zina, ka spēj jebkurā brīdī atvērt ”WhatsApp” lietotni, uzrakstīt ziņu un aizvērt telefonu, neuztraucoties par interneta savienojumu. Tīmeklis līdz šim nav piedzīvojis šādu uzticību no lietotāja, kas ir viens no iemesliem, kapēc lietotāji bieži dod priekšroku tradicionālajām lietotnēm [1].

2.7.1. Servisskripts

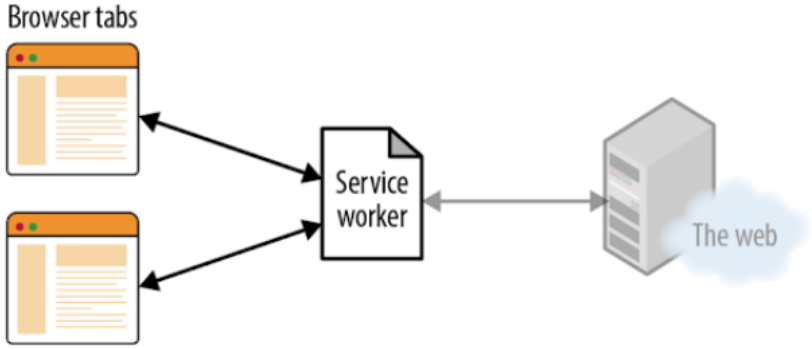
Pirms servisskriptiem eksistēja interneta pārlūku atbalsts, kods darbojās vai nu serverī, vai interneta pārlūka logā. Servisskripti ievieš jaunu līmeni.

Servisskripts tiek reģistrēts, lai kontrolētu vienu vai vairākas vietnes lapas. Tiklīdz instalēts, servisskripts atrodas ārpus jebkura interneta pārlūka loga vai cilnes. Tur skripts spēj klausīties un reaģēt uz tā kontrolēto vietnes lapu notikumiem. Notikumi (events), kā piemēram failu pieprasījumi, var tikt pārtverti, modificēti un atgriezti vietnes lapai.



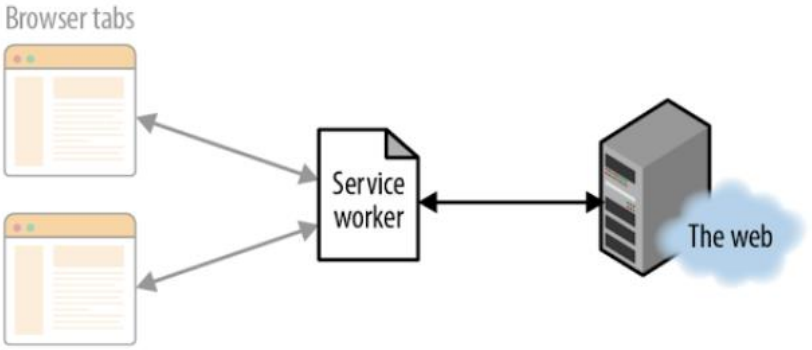
2.1. att. **Servisskripta arhitektūra** [1]

Attēlā 2.1. redzams, ka servisskripts atrodas starp tīmekli un lietotāja interneta pārlūku un spēj komunicēt ar visām šīm komponentēm. Tas nozīmē, ka skripts spēj atbildēt uz pieprasījumiem, neatkarīgi no interneta savienojuma.



2.2. att. **Vietnes lapas komunicē ar servisskriptu bezsaistē** [1]

Attēlā 2.2. ir parādīts, kā servisskripts spēj nodrošināt progresīvās tīmekļa lietotnes darbību bezsaistē. Skripts spēj noteikt bezsaistes režīmu vai lēnas atbildes uz pieprasījumiem no servera un atgriezt saturu, kas saglabāts pārlūka kešatmiņā (cache).



2.3. att. **Servisskripts komunicē ar serveri, kad lietotājs ir pametis lapu** [1]

Attēlā 2.3. ir parādīts koncepts, kā, gadījumā ja lietotājs ir aizvēris visas cilnes ar progresīvo tīmekļa lietotni interneta pārlūkā, joprojām ir slānis, kas var komunicēt ar serveri.

Tas spēj saņemt un parādīt pašpiegādes paziņojumus (push notifications) vai pārliecināties, ka jebkuras lietotāja veiktās darbības tiek piegādātas serverim, pat ja lietotājs aizvēra lietotni, pirms atguva interneta savienojumu.

Servisskriptam ir stāvokļu cikls, kas nav saistīts ar vietnes lapu. Lai instalētu servisskriptu, tas ir jāreģistrē vietnes lapas JavaScript kodā. Reģistrējot skriptu, interneta pārlūks sāks servisskripta instalācijas soli.

Parasti instalācijas stāvoklī tiek kešatmiņā saglabāti statiski resursi, kā piemēram JavaScript, CSS faili un attēli. Ja visi resursi ir veiksmīgi saglabāti parlūka kešatmiņā, tad servisskripts kļūst instalēts, kas ir nākošais stāvoklis pēc instalācijas. Ja kāds fails kļūdas dēļ netiek saglabāts kešatmiņā, tad instalācijas solis neizpildīsies un skripts neaktivizēsies.

Kad servisskripts ir instalējies, tad seko aktivizēšanās stāvoklis, kurā skripts pārņem kontroli pār vietnes lapām. Šajā solī parasti tiek izdzēsta iepriekšējās skripta versijas pārvaldītā pārlūka kešatmiņa.

Pēc tam servisskripta darbība var tikt izbeigta, ja tas kādu laiku netiek lietots, lai taupītu brīvpiekļuves atmiņu vai tas apstrādās tīmekļa pieprasījumus, ja tādi tiek veikti [23].

Servisskripti tiek uzskatīti par galveno progresīvās tīmekļa lietotnes sastāvdaļu. To noturīguma īpašības ļauj progresīvajām tīmekļa lietotnēm piepildīt daudzus lietotāju priekšstatus, par to, ko lietotnēm būtu jāspēj paveikt.

Viena no lielākajām servisskriptu priekšrocībām, ir tāda, ka tie ir parasti JavaScript faili. Tie tiek izstrādāti tāpat, kā jebkurš cits JavaScript kods.

Ieguvums, saprotot un ieviešot servisskriptus un ar tiem saistītās tehnoloģijas, ir liels. Tie ļauj izstrādātājiem izmantot jau eksistējošās tīmekļa tehnoloģijas – JavaScript, HTML, CSS – lai izveidotu tīmekļa lietotnes, kas spēj sacensties vai pat pārspēt tradicionālās mobilās lietotnes [1].

2.8. Atkal-iesaistāmība

Parasti tas ir ticamāk, ka tradicionālo mobilo lietotņu lietotāji atkārtoti izmantos lietotni, nekā, piemēram, tīmekļa vietnes apmeklētājs - tīmekļa vietni. Progresīvajām tīmekļa lietotnēm ir paredzēts sasniegt tādu pašu mērķi ar tādām iespējām, kā pašpiegādes paziņojumi [3].

2.8.1. Pašpiegādes paziņojumi

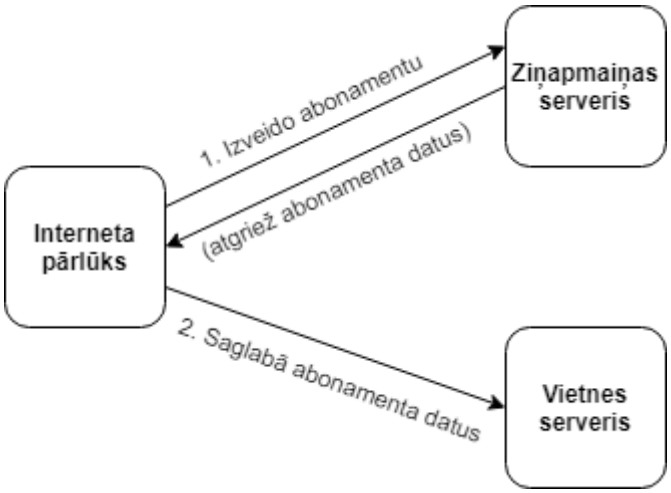
Pašpiegādes paziņojums sastāv no divām komponentēm:

- Ziņa, kas nosūtīta, lietojot “Push API”,
- Paziņojums, kas tiek parādīts lietotājam, lietojot “Notification API”.

“Notification API” ļauj tīmekļa vietnei vai servisskriptam izveidot un kontrolēt sistēmas paziņojumus. Šie paziņojumi tiek rādīti ārpus interneta pārlūka, ierīces lietotāja saskarnē, tādējādi eksistējot ārpus interneta pārlūka loga vai cilnes. Tā kā šie paziņojumi nav atkarīgi no pārlūka loga vai cilnes, tad tie var tikt izveidoti arī pēc tam, kad lietotājs ir jau pametis vietni.

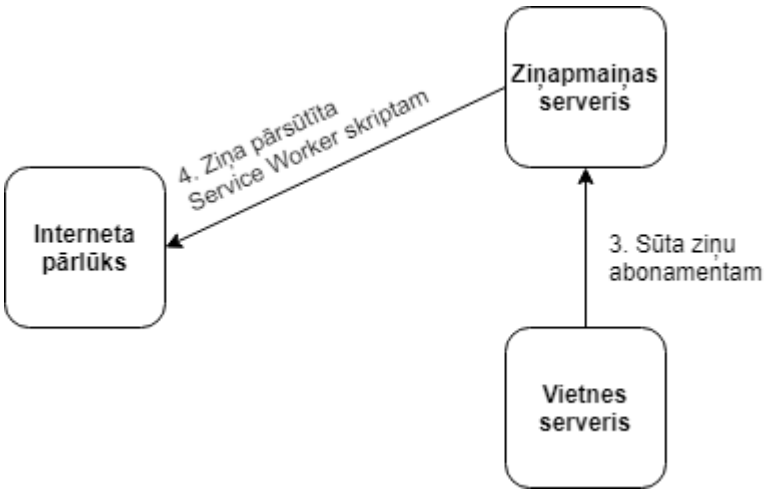
“Push API” ļauj lietotājiem abonēt pašpiegādes ziņojumu saņemšanu lietotnē un ļauj serverim sūtīt ziņas uz lietotāju pārlūkiem jebkurā brīdī. Šīs ziņas apstrādā servisskripts, kas spēj veikt tālākas darbības ar tām, pat ja lietotājs ir pametis lietotni. Vispopulārākā darbība ir šos ziņojumus parādīt lietotājam.

Visi pašpiegādes paziņojumi tiek apstrādāti centrālā ziņapmaiņas (messaging) serverī. Šo serveri uztur interneta pārlūka izstrādātājs un tas seko līdzi visiem lietotnes abonamentiem. Tas nodrošina to, ka pašpiegādes paziņojumi netiek ļaunprātīgi izmantoti un lietotāji nesaņem daudz surogātpasta (spam).



2.4. att. **Pašpiegādes paziņojuma abonamenta reģistrēšana un saglabāšana** [1]

Attēlā 2.4. ir aprakstīti pirmie divi soļi pašpiegādes paziņojuma realizācijai. Pirmais solis ir abonēt konkrēto lietotāju ziņapmaiņai ziņapmaiņas serverī, kurš atgriezīs datus par specifisko abonamentu. Otrais solis ir šos datus saglabāt vietnes serverī. Abas šīs darbības notiek tikai vienu reizi katram lietotājam.



2.5. att. **Pašpiegādes paziņojumu sūtīšana** [1]

Attēlā 2.5. ir parādīts kādas darbības tiek veiktas katru reizi, kad sūta ziņojumu. Vispirms vietnes serveris pārsūta ziņu ziņapmaiņas serverim, izmantojot saglabātos abonamenta datus, kurš to pēc tam pārsūta lietotāja interneta pārlūkam, kur servisskripts to saņem un apstrādā atkarībā no ieviestās funkcionalitātes [1].

2.9. Instalējamība

Progresīvai tīmekļa lietotnei ir jābūt instalējamai uz lietotāja mobilās ierīces. Tiklīdz lietotājs izrāda interesi par progresīvo tīmekļa lietotni, interneta pārlūks automātiski piedāvās pievienot saīsni (shortcut) uz lietotāja ierīces, kas izskata ziņā neatšķiras no citām tradicionālajām mobilajām lietotnēm [1].

Atkarībā no interneta pārlūka, ir dažādi kritēriji, kam jāizpildās, lai pārlūks piedāvātu instalēt lietotni lietotāja ierīcē. Piemēram, "Chrome" interneta pārlūkā ir jāizpildās šādiem kritērijiem:

- Progresīvā tīmekļa lietotne nav jau instalēta uz lietotāja ierīces,
- Lietotājs ir darbojies ar tīmekļa lapu vismaz 30 sekundes,
- Izpilda progresīvās tīmekļa lietotnes kritērijus:
    - Iekļauj tīmekļa lietotnes manifesta (manifest) failu, kas satur:
        - "short_name" vai "name" vērtību,
        - Ikonas, iekļaujot 192 un 512 pikseļu izmēru versijas,
        - "start_url" vērtību.
    - Izmanto HTTPS protokolu (nepieciešams servisskriptiem),
    - Ir reģistrēts servisskripts, kas apstrādā "fetch" notikumus [10].

2.9.1. Lietotnes manifesta fails

Tīmekļa lietotnes manifesta fails ir vienkāršs JSON tipa fails, kas nodod interneta pārlūkam informāciju par lietotni un kā tai būtu jādarbojas, kad ir instalēta uz lietotāja ierīces.

Tīmekļa pārlūki, atkarībā no spējām, var izmantot manifest failu, lai veiktu dažādas darbības. Piemēram, "Chrome" pārlūks automātiski uzģenerēs uzplaiksnījuma ekrānu (splash screen) no manifesta faila "name", "background_color" un "icons" vērtībām, kas tiks parādīts, kad progresīvā tīmekļa lietotne ir palaista no saīsnes, bet saturs vēl nav pieejams [11].

2.10. Drošība

Tā kā progresīvās tīmekļa lietotnes veic daudz personalizētākas darbības un tāpēc, ka visi tīkla pieprasījumi var tikt pārtverti ar servisskriptu, lietotne ir obligāti jāservē ar HTTPS protokolu, lai novērstu MITM (man-in-the-middle) uzbrukumus [1].

3. ANGULAR IETVARS

Angular ietvars atvieglo tīmekļa lietotņu izstrādi. Angular apvieno deklaratīvās veidnes (declarative template), atkarību injicēšanu (dependency injection), pilnu izstrādes rīku nodrošināšanu un integrētās labās prakses elementus, lai nodrošinātu efektīvu izstrādi, ļaujot izstrādāt lietotnes priekš tīmekļa, mobilās ierīces vai galddatora [12].

3.1. Angular lietotnes arhitektūra

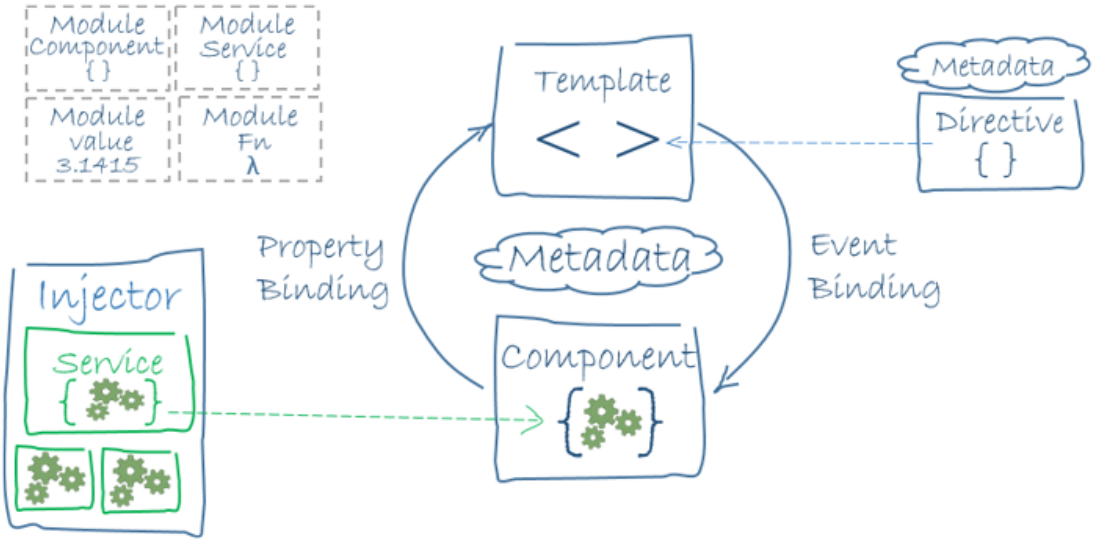
Angular ietvars ir piemērots klienta lietotņu izstrādei HTML un TypeScript valodās. Arī pats ietvara pirmkods ir rakstīts TypeScript valodā. Angular īsteno pamata un papildus funkcionalitāti, kā TypeScript bibliotēku kopu, ko importē izstrādājamajā lietotnē.

Angular lietotnes pamata bloki ir moduļi (modules), kas nodrošina Angular komponenšu (components) kompilācijas kontekstu. Moduļi sakopo loģiski saistītu kodu funkcionālā kopā. Angular lietotnei vienmēr ir vismaz saknes modulis, kas ļauj ielādēt lietotnes kodu un tipiski satur citus moduļus ar papildus funkcionalitāti.

Komponentes definē skatus, kas ir kopas ar ekrāna elementiem, kurus Angular lietotne var apstrādāt un modificēt atkarībā no programmas datiem un loģikas. Katrai lietotnei ir vismaz viena saknes komponente.

Komponentes lieto servisus, kas nodrošina specifisku abstraktu funkcionalitāti, kas nav tieši saistīta ar skatiem. Servisi var tikt injicēti komponentēs kā komponentes atkarības, padarot kodu modulāru (modular), atkalizmantojamu (reusable) un efektīvu.

Komponentes un servisi ir parastas TypeScript klases ar īpašiem "dekorator" elementiem, kas norāda to tipu un metadatus Angular ietvaram, lai norādītu kā tās lietojamas.



3.1. att. **Angular ietvara lietotnes arhitektūra** [12]

Attēlā 3.1. ir parādīta Angular lietotnes arhitektūras diagramma. Komponentes (component) kopā ar veidnēm (template) definē Angular skatu. Komponente komunicē ar veidni, padodot datus, lietojot vērtību saistīšanu (property binding), un veidne komunicē ar komponenti, lietojot notikumu saistīšanu (event binding). Atkarību injektors (dependency injector) nodrošina komponentei pieeju specifisku servisu funkcionalitātei [12].

4. PROGRESĪVAS TĪMEKĻA LIETOTNES IZSTRĀDE ANGULAR IETVARĀ

Darba pētījuma nolūkos ir izstrādāta vienas lapas (single page) tīmekļa lietotne Angular ietvarā. Lietotne ir izstrādāta trīs versijās:

- Parasta Angular lietotne, bez īpašiem, progresīvai tīmekļa lietotnei raksturīgiem uzlabojumiem un funkcionalitātes. Pieejams tiešsaistē https://edgarsjoja.github.io/books-master-build/,
- Progresīva tīmekļa lietotne, kas iegūta, pārveidojot pirmo versiju. Servisskripta funkcionalitāte ir realizēta ar *@angular/service-worker* pakotni. Pieejams tiešsaistē: https://edgarsjoja.github.io/books-pwa-build/,
- Progresīva tīmekļa lietotne, kas saglabā to pašu funkcionalitāti, ko otrā versija, izņemot servisskripta realizāciju. Servisskripta funkcionalitāte ir realizēta manuāli, bez papildus bibliotēku izmantošanas. Pieejams tiešsaistē: https://edgarsjoja.github.io/books-pwa-custom-build/.

Lietotne izmanto "New York Times" grāmatu API [13], lai iegūtu bestselleru kategoriju sarakstus un grāmatas katrā no šiem sarakstiem. Lai iegūtu papildus informāciju par grāmatu, kā piemēram aprakstu, autorus un attēlu, ir izmantots "Open Library Books" grāmatu API [14].

Lietotnes pirmkods ir pieejams publiskā repozitorijā (repository) – (https://github.com/EdgarsJoja/books). Repozitorijā ir pieejami trīs pirmkoda zari, kur katrā no zariem ir realizēta viena no lietotnes versijām.

Šajā nodaļā ir aprakstīts, kādas iespējas ir Angular ietvarā, lai izpildītu otrajā nodaļā aprakstītās progresīvās tīmekļa lietotnes iezīmes un prasības, un kā tās ir realizētas autora izstrādātajā lietotnē.

Lietotnes izstrādei un testēšanai mobilajā ierīcē ir izmantota "Nokia 5" ierīce ar Android operētājsistēmas versiju 7.1.2 un "Chrome" interneta pārlūku.

4.1. Lietotnes funkcionalitātes apraksts

Izvirzītais mērķis lietotnes funckionalitātei ir atbilstība pamata progresīvo tīmekļa lietotņu prasībām, par tās definīciju uzskatot 2. nodaļā aprakstītās minimālās prasības, lai lietotne tiktu piedāvāta instalācijai [33].

Lietotne sastāv no četru veidu lapām:

- Sākumlapa, kur ir pieejama informācija par lietotni,

- Bestselleru kategorijas, kur ir attēlots saraksts ar visām "New York Times" API [13] pieejamajām bestselleru sarakstu kategorijām,
- Bestselleru kategorijas grāmatu saraksts, kurā ir attēlotas visas kategorijas grāmatas,
- Grāmatas informācijas lapa, kur ir redzama informācija par grāmatu, kas iegūta, izmantojot "Open Library Books" API [14].

Lietotnē eksistē navigācijas josla, kurā ir pieejama arī meklēšanas funkcionalitāte, kas filtrē lapās attēlotos sarakstus pēc ievadītā atslēgvārda.

Atbilstoši progresīvo tīmekļa lietotņu pamata prasībām, lietotājam ir jāspēj lietotnes saturu skatīt bezsaistē noteiktu laiku pēc tam, kad lietotne pamesta, ja konkrētās lapas ir tikušas apmeklētas tiešsaistes režīmā. Tāpat arī atkārtotiem lapu apmeklējumiem ir jābūt ātriem, neraugoties uz lietotāja interneta savienojuma kvalitāti, progresīvi izmantojot servisskriptu īpašības, ja interneta pārlūks tos atbalsta.

Lietotnei ir jāizskatās vizuāli labi un pieejamai jebkāda izmēra ekrānos, un pamata funkcionalitātei ir jābūt pieejamai visos populārākajos interneta pārlūkos.

Apmeklējot lietotnes tīmekļa vietni, lietotājam ir jāspēj lietotni ieinstalēt mobilajā ierīcē un palaist to no izveidotās saīsnes.

Lietotnei ir obligāti jāsaņem maksimālais punktu skaits "Lighthouse" [24] rīka auditā progresīvo tīmekļa lietotņu kategorijā un pēc iespējas augstāks rezultāts visās pārējās kategorijās.

4.2. Lietotnes funkcionalitātes realizācija

Lietotnes funkcionalitāte ir realizēta, izmantojot Angular ietvaru. Angular ir izvēlēts kā lietotnes izstrādes ietvars, jo pašreiz ir viens no populārākajiem [34] klienta puses JavaScript/TypeScript ietvariem pasaulē. Salīdzinoši nesen Angular relīzes (release) piezīmēs ir minēts dažādu progresīvo tīmekļa lietotņu elementu atbalsts [29], līdz ar to autoram ir interese tehniskāk apskatīt un noskaidrot izstrādes specifikas un detaļas šiem elementiem.

4.2.1. Lietotnes veiktspēja

Izstrādātā lietotne ir vienas lapas Angular lietotne. Tākā lietotne ir ar samērā mazu un specifisku funkcionalitāti, tad lietotnē eksistē tikai divi moduļi – saknes modulis un lietotnes čaulas modulis, un praktiski visi resursi, kas tiek sūtīti, tiek lietoti katrā lapā. Vairums no tiem ir JavaScript, CSS un fontu faili. Failu izmēru saspiešanas nolūkos, programmas kompilācijas laikā, ir izmantotas Angular komandrindas opcijas *–prod* un *–build-optimizer,* kas neiekļauj

atstarpes, koda komentārus, neizmantotu kodu, saīsina mainīgo un funkciju nosaukumus, u.c. [15].

Par lietotnes serveri ir izvēlēts "GitHub Pages" serviss, kurā ir iespējams uzturēt statiskas tīmekļa vietnes, un, kas izmanto HTTP/2 protokolu, ko ir iespējams redzēt "Chrome" interneta pārlūka tīmekļa pieprasījumu cilnē izstrādātāja rīku sadaļā.

4.2.2. Sasaiste

Angular ietvars lietotnes maršrutēšanai (routing) izmanto ietvarā iebūvēto maršrutēšanas moduli. Katrai no lietotnes lapām tiek piešķirts unikāls ceļš, kas tās identificē. Katru bestselleru grāmatu sarakstu identificē tā nosaukums, veidojot tīmekļa adreses ceļu formātā */lists/<list-name>*, kur *<list-name>* ir saraksta nosaukums. Katras grāmatas lapu definē šis pats tīmekļa adreses ceļš ar papildus, grāmatai unikālu, ISBN parametru, veidojot rezultātā */lists/<list-name>/<isbn>*, kur *<isbn>* ir grāmatas ISBN vērtība. Tādējādi jebkuram lietotnes stāvoklim ir piešķirta unikāla tīmekļa adrese, ko iespējams kopīgot vai saglabāt pārlūka grāmatzīmēs.

4.2.3. Lietotāja saskarne

Lietotns dizaina realizēšanā ir izmantots *@angular/material* modulis [16], kas nodrošina lietotājiem ērtu lietotnes saskarni uz jebkuru ierīci un ekrāna izmēru. Šo moduli ir izstrādājuši Angular ietvara izstrādātāji, un tas ir īpaši optimizēts Angular ietvaram.

"Material" modulis ir veidots, lai atbilstu "Google" materiāla (material) dizaina [30] principiem, kas izstrādāti, lai apkopotu labākos lietotāja saskarnes principus.

4.2.4. Čaulas modelis

Lietotnes progresīvās tīmekļa lietotnes versija izmanto čaulas modeļa principu, lai samazinātu laiku, kad lietotājs ierauga pirmos lapas elementus parādāmies ekrānā. Lai realizētu lietotnes čaulas modeli ir izmantota servera puses renderēšanas (server side rendering) tehnika. Parastās lietotnes versija, kurai nav progresīvo tīmekļa lietotņu uzlabojumi, kā tipiska Angular ietvara lietotne, *index.html* failā izsauc saknes komponenti. Tad, sagaidot, kad attiecīgie JavaScript faili ir ielādējušies, no šī saknes komponentes izsaukuma tiek uzģenerēts HTML kods un lietotājam parādās visa lapa uzreiz.

Servera puses renderēšana, lietotnes pirmkoda kompilēšanas brīdī, kaut kādu definētu pirmkoda daļu jau uzģenerē HTML formātā un ievieto to *index.html* failā. Šī HTML daļa ar CSS kodu veido lietotnes čaulu. Tādējādi lietotāja interneta pārlūkam, veicot tīmekļa

pieprasījumu uz serveri un saņemot *index.html* failu, ir iespējams uzreiz parādīt lietotnes čaulas daļu, negaidot, kamēr ielādējas JavaScript faili, kas pēc tam uzģenerē lapas saturu.

Autora izstrādātajā lietotnē par lietotnes čaulu ir izvēlēta lapas navigācijas sadaļa, kas iekļauj lietotnes nosaukumu, navigācijas elementus, meklēšanas ievades lauku un "New York Times" brenda API ikonu, saskaņā ar API lietošanas noteikumiem [13].

```
...
<app-root></app-root>
...
```

4.1. att. **Angular lietotnes saknes komponentes izsaukšana *index.html* failā.**

Attēlā 4.1. ir parādīts autora izstrādātās lietotnes, bez progresīvo tīmekļa lietotņu uzlabojumiem, *index.html* faila saknes komponentes izsaukšana, kas ietver sevī arī navigācijas elementus. Kad tiek ielādēti JavaScript faili, attiecīgais lapas HTML kods tiek uzģenerēts un parādīts lietotājam pārlūkā.

```
...

<app-root _nghost-c0="" ng-version="5.2.5">
    <router-outlet _ngcontent-c0="" name="app-shell"></router-outlet>
    <app-shell _nghost-c3="" class="ng-star-inserted">

    ...

    <mat-toolbar-row _ngcontent-c4="" class="mat-elevation-z4 mat-toolbar-row">
        <span _ngcontent-c4="">Books</span>
        <span _ngcontent-c4="" class="space-filler"></span>
        <a _ngcontent-c4="" href="http://developer.nytimes.com/" target="_blank"><img _ngcontent-c4=""
src="assets/images/icons/nyt.png"></a>
    </mat-toolbar-row>

    ...

    </app-shell>
</app-root>

...
```

4.2. att. **Servera puses renderēšanas metodes ģenerētais lietotnes čaulas kods.**

Attēlā 4.2. ir parādīta daļa no *index.html* faila satura, kas iegūta izmantojot servera puses renderēšanas metodi lietotnes čaulas ģenerēšanai. Redzams, ka *index.html* failā uzreiz ir pieejams HTML kods, ko interneta pārlūks uzreiz var parādīt, negaidot, kad lejupielādēsies JavaScript faili, kas pēc tam uzģenerē un papildina lapas saturu. Šāda pieeja nozīmē, ka pirmajā lapas apmeklēšanas reizē, tiek pārsūtīts vairāk datu, nekā bez servera puses renderēšanas metodes, taču lietotājs ātrāk vizuāli ierauga vietnes lapu [17, 18, 19].

Servera puses renderēšana izstrādātajā lietotnē ir realizēta ar Angular Universal bibliotēku [19].

Esošajā lietotnē tiek uzģenerēta jauna Angular Universal lietotne ar *ng generate universal <universal_name>* komandu, kur *<universal_name>*, ir lietotnes nosaukums. Tā kā Universal

lietotnei ir gandrīz tie paši faili, kas parastai angular lietotnei, tad tiek izveidoti tikai daži jauni faili. Viens no šiem failiem ir *app.server.module.ts* fails, kas tiks izmantots, lai veiktu servera puses renderēšanu.

Tālāk tiek ģenerēta lietotnes čaula ar komandu *ng generate app-shell <name> --universal-app=<universal_name> --route=<route>,* kur *<route>* ir lietotnes domēna ceļš, priekš kura tiks renderēts HTML kods servera pusē. Šis ceļš nav pieejams lietotnē lietotājam, jo tiek izmantots tikai pirmkoda kompilācijas brīdī, lai uzģenerētu lietotnes čaulu, kas pēc tam tiks ievietota *index.html* failā. Komandas opcija *–universal-app* norāda, kura Angular Universal lietotne tiks izmantota servera puses renderēšanai, kas šajā gadījumā ir tikko uzģenerētā. Potenciāli var eksistēt arī vairākas Universal lietotnes vienā Angular projektā.

Pēc komandas izpildes ir uzģenerēta jauna komponente *app-shell*, kurai ir piešķirts *<route>* maršruts, kas eksistē tikai Angular Universal lietotnē un ir definēts *app.server.module.ts* failā. Jaunajā *app-shell* komponentē tiek ievietots saturs, kurš tiks renderēts HTML kodā, lietotnes kompilācijas (compilation) brīdī [18]. Autora lietotnes gadījumā tiek ievietota *app-navigation* komponente, kas nodrošina lietotnes navigāciju. Daļa no uzģenerētā HTML koda ir parādīta 4.2. attēlā.

4.2.5. Savienojamības neatkarība

Progresīvās tīmekļa lietotnes izstrādei tika izvirzīts mērķis būt spējīgai darboties bezsaistes režīmā, ja lietotājs atkārtoti apmeklē jebkuru lietotnes lapu. Šāda mērķa sasniegšanai ir jāizmanto viena no svarīgākajām progresīvās tīmekļa lietotnes iezīmēm – servisskripts.

Eksistē vairāki veidi, kā izstrādāt šo funkcionalitāti, tādēļ, lai noskaidrotu kādi ir ieguvumi un zaudējumi, no dažādu metožu izmantošanas, ir izveidotas divas autora izstrādātās progresīvās tīmekļa lietotnes versijas, kur vienīgā atšķirība ir veids, kā ir ieviesta servisskripta funkcionalitāte.

4.2.5.1. Servisskripta izstrāde lietojot *@angular/service-worker* pakotni

Angular ietvars kā servisskripta realizāciju noklusējumā izmanto *@angular/service-worker* pakotni. Pakotne izveido *ngsw-config.json* konfigurācijas failu, kurā ir iespējams definēt noteiktus servisskripta darbības principus, kā, piemēram, resursus, kuri jāsaglabā pārlūka kešatmiņā.

Lietotnes kompilācijas laikā, konfigurācijas fails *ngsw-config.json* tiek izmantots, lai uzģenerētu *ngsw.json* failu, kas satur informāciju par konkrētiem failiem, kas saglabājami

kešatmiņā, to glabāšanas ilgumu, maksimālo kešatmiņas apjomu, utt. Uzģenerētais konfigurāciju fails tiek izsaukts *ngsw-worker.js* failā, kas satur servisskripta funkcionalitāti. Attiecīgi *ngsw-worker.js* fails tiek iekļauts *main.<hash>.bundle.js* failā, kas satur visu lietotnes JavaScript funkcionalitāti, kur *<hash>* ir unikāla burtu un ciparu virkne, atkarībā no faila satura. Unikāli failu nosaukumi nodrošina to, ka tad, ja lietotnei ir veiktas izmaiņas pirmkodā, eksistējošais servisskripts lietotāja pārlūkā nespēs kešatmiņā atrast jaunā faila datus un veiks pieprasījumu uz vietnes serveri. Pretējā gadījumā, ja failu nosaukumi nemainītos, tad servisskripts atgrieztu jau saglabātos kešatmiņas datus, tādējādi lietotājam nesaņemot jaunās vietnes izmaiņas. Pēc tam *main.<hash>.bundle.js*, kopā ar citiem JavaScript failiem, tiek iekļauts *index.html* failā [20, 21].

Ieguvums, izstrādājot lietotni, izmantojot šo pakotni, ir tāds, ka servisskripta pamata funkcionalitāti ir iespējams ieviest dažu minūšu laikā.

Diemžēl *@angular/service-worker* pakotne neatbalsta servisskripta funkcionalitātes paplašināšanu, līdz ar to potenciāli lielākām lietotnēm ar specifiskām funkcionalitātes vajadzībām šīs pakotnes izmantošana var nenodrošināt visas funkcionalitātes vajadzības.

Jāmin arī, ka Angular ietvars, un līdz ar to, *@angular/service-worker* pakotne, joprojām ir aktīvā izstrādes stadijā. Tas nozīmē, ka šī servisskripta funkcionalitātes ieviešana var būt apgrūtināta izstrādes gaitā un ar potenciālām kļūdām. Autors lietotnes servisskripta izstrādes gaitā, lietojot *@angular/service-worker* pakotni, vairākkārt piedzīvoja kļūdu paziņojumus un problēmas, kas ir reģistrēti Angular ietvara repozitorijā (https://github.com/angular/angular) pēdējā mēneša vai pat nedēļas laikā.

4.2.5.2. Servisskripta manuāla izstrāde

Kā minēts nodaļā 2.7.1, servisskripti ir parasti JavaScript faili, kas nozīmē, ka tā nav īpaša Angular ietvara funkcionalitāte un var tikt piemērota jebkurai tīmekļa vietnei. Tas nozīmē, ka arī Angular lietotnei var izstrādāt manuālu servisskripta funkcionalitāti, ja lietotnei ir nepieciešama, piemēram, specifiska stratēģija datu glabāšanai kešatmiņā.

Autora izstrādātajā lietotnē manuāla servisskripta funkcionalitāte ir realizēta *sw.js* failā, kurš tiek reģistrēts kā servisskripts *index.html* failā. Failā *sw.js* ir definēts lietotnes lietojamās kešatmiņas nosaukums, kā arī failu nosaukumi, kurus ir nepieciešams saglabāt kešatmiņā servisskripta instalācijas brīdī interneta pārlūkā. Jāņem vērā, ka, tā kā šeit ir jānorāda precīzi failu nosaukumi, tad nav iespējams manuāli norādīt Angular kompilēto failu nosaukumus, kas satur unikālas *<hash>* simbolu virknes, kas kompilācijas brīdī katru reizi var mainīties. Autors izstrādes gaitā nonāca pie diviem iespējamajiem risinājumiem, kā šo problēmu atrisināt:

- Izveidot/lietot skriptu, kas lietotnes kompilācijas brīdī reģistrē izveidoto failu nosaukumus un ievieto konfigurācijas failā [35], kurš pēc tam var tikt izsaukts servisskripta instalācijas brīdī interneta pārlūkā, lai saglabātu kešatmiņā tur norādītos failus,
- Izmainīt failu ģenerēšanu, lai tā nepievienotu unikālu <*hash*> simbolu virkni lietotnes kompilācijas brīdī.

Autors, izstrādājot manuāli servisskriptu, ir izvēlējies otro pieeju – nepievienot <*hash*> simbolu virknes failu nosaukumos. Angular ietvars pirmkoda kompilācijas komandai piedāvā *–output-hashing=*<*hashing_option*> opciju, kur <*hashing_option*> ir viena no vairākām definētām iespējām. Šajā gadījumā tika izvēlēta opcija "none", kas nozīmē, ka neviena uzģenerētā faila nosaukumā nebūs iekļauta <*hash*> simbolu virkne. Piemēram, ģenerētie failu nosaukumi būtu *main.bundle.js* un *inline.bundle.js* utt. Citas opcijas ļauj nepievienot <*hash*> simbolu virkni, piemēram, tikai specifiska tipa failiem.

Šajā gadījumā ir jāievieš papildus funkcionalitāte servisskriptam, kas pārvaldītu pārlūka kešatmiņu, lai novērstu, to, ka lietotājs nesaņem jaunākās lietotnes izmaiņas dēļ eksistējošiem vienādiem failu nosaukumiem, saglabātiem pārlūka kešatmiņā, kā tas ir aprakstīts 4.5.1. nodaļā.

Lai to realizētu, tiek lietota kešatmiņas nosaukuma versionēšana, manuāli pieliekot nosaukumam beigās *-v*<*version*>, kur <*version*> ir lietotnes versijas skaitlis. Interneta pārlūks automātiski atpazīs, ja servisskripta fails ir izmainīts un instalēs jauno skriptu. Instalācijas laikā jaunais skripts saglabās jaunās versijas kešatmiņā definētos failus, kuros ir potenciālās izmaiņas. Šajā brīdī joprojām aktīvs ir vecais servisskripts, tādēļ vēl nevar izdzēst iepriekšējās versijas kešatmiņas datus.

Kad aktivizējas jaunais servisskripts, piemēram, lietotājam aizverot visas cilnes un atverot lietotni atkal, var izdzēst vecās kešatmiņas datus, jo vecais servisskripts vairs nav aktīvs un šī kešatmiņa netiks izmantota [22].

```
...
caches.keys().then(function (keys) {
    return Promise.all(keys.map(function (key) {
        if (key.startsWith(CACHE_PREFIX) && key !== CACHE) {
            return caches.delete(key);
        }
    }))
})
...
```

4.3. att. **Veco kešatmiņu datu izdzēšana servisskripta aktivācijas brīdī**

Attēlā 4.3. ir parādīts, kā autors ir realizējis vecās kešatmiņas datu dzēšanu, kad aktivizējas jauns servisskripts. Tiek pārstaigātas un izdzēstas visas kešatmiņas, kuru nosaukums sākas ar noteiktu prefiksu un nesakrīt ar pašreizējo kešatmiņas nosaukumu.

Lai realizētu pieprasījumu pārtveršanu un apstrādi, servisskripts klausās interneta pārlūka "fetch" notikumus. Tā kā lietotnes saturs nav tāds, kas bieži tiek atjaunināts, ņemot vērā, ka bestselleru sarakstus atjaunina reizi nedēļā vai mēnesī, un grāmatu dati praktiski nemainās, tad kešatmiņas lietošanas stratēģija ir tāda, ka vispirms tiks atgriezti dati no kešatmiņas. Ja dati kešatmiņā nav atrodami, tad tiks veikts pieprasījums uz serveri – vai nu vietnes serveri, vai kādu no lietotajiem API serveriem, kura saņemtie dati tiek nokopēti un ievietoti kešatmiņā, kā arī atgriezti lapai.

```
...
caches.match(e.request).then(function (resp) {
    return resp || fetch(e.request).then(function (response) {
        return caches.open(CACHE).then(function (cache) {
            cache.put(e.request, response.clone());
                return response;
            });
        });
    })
);
...
```

4.4. att. **Interneta pārlūka pieprasījuma apstrādāšana**

Attēlā 4.4. ir parādīta pirmkoda daļa, kas realizē darbu ar kešatmiņu, servisskriptam apstrādājot resursu pieprasījumus.

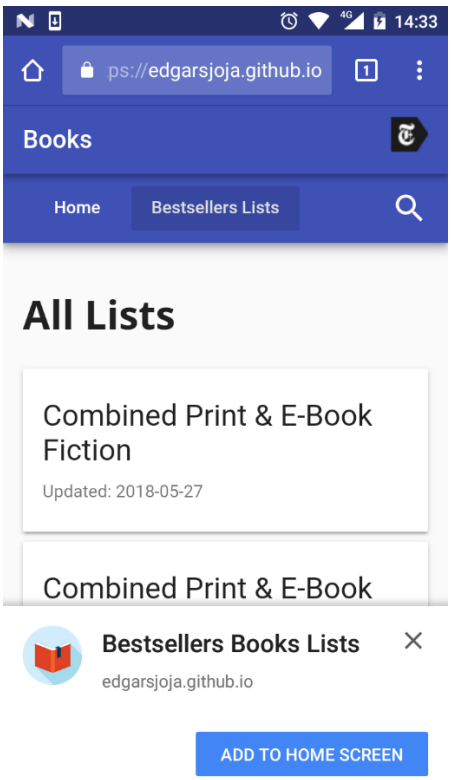
Manuāla servisskripta ieviešana nozīmē, ka ir iespējams izstrādāt funkcionalitāti ļoti specifiskām vietnes vajadzībām, piemēram, atgriezt noteiktu attēlu visiem tīkla attēlu pieprasījumiem.

Tomēr šāda pieeja nozīmē, ka izstrādei ir vajadzīgs vairāk laika, jo ir jāuzraksta arī pamata servisskripta funkcionalitāte, ko jau piedāvā *@angular/service-worker* pakotne, kā piemēram kešatmiņas pārvaldība.

4.2.6. Lietotnes instalējamība

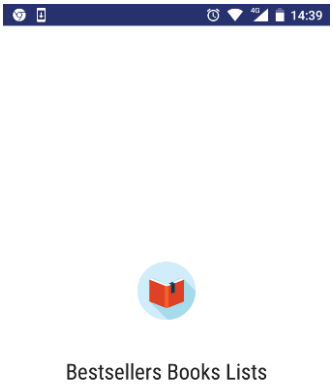
Kā minēts nodaļā 2.9, lietotnei ir jāizpilda noteiktas progresīvo tīmekļa lietotņu prasības, lai tā tiktu piedāvāta lietotājam instalācijai mobilajā ierīcē. Nodaļā 4.2.1. tika aprakstīts, ka autora izstrādātā lietotne izmanto HTTPS protokolu, un nodaļā 4.2.5. ir aprakstīta servisskripta realizācija. Trešais nosacījums lietotnes instalācijai ir manifesta fails, kas satur informāciju par lietotni.

Autora izstrādātajā lietotnē šī informācija tiek glabāta *manifest.json* failā, kas tiek norādīts *index.html* failā. Izstrādātās lietotnes manifesta fails satur lietotnes nosaukumu, aprakstu, definētas ikonas u.c.



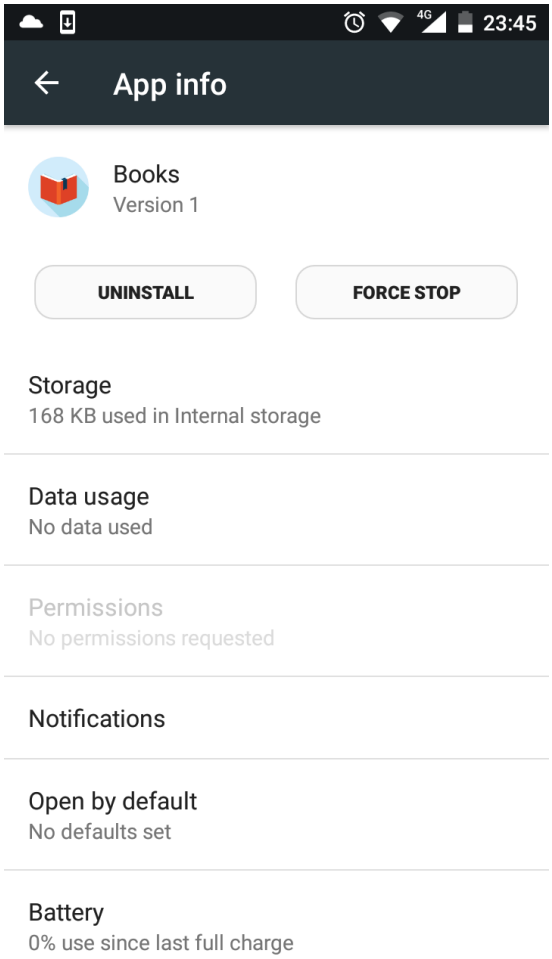
4.5. att. **Lietotne tiek piedāvāta instalācijai mobilajā ierīcē**

Attēls 4.5. ir ekrānuzņēmums (screenshot) mobilajā ierīcē, lietotājam pirmo reizi apmeklējot autora izstrādāto lietotni. Tātad lietotne ir izpildījusi 2. nodaļā aprakstītos kritērijus progresīvās tīmekļa lietotnes definīcijai. Nospiežot "Add To Home Screen" pogu, lietotne tiek instalēta mobilajā ierīcē.



4.6. att. **Izstrādātās lietotnes uzplaiksnījuma ekrāns**

Attēls 4.6. ir ekrānuzņēmums mobilajā ierīcē, kas parādās, lietotājam atverot lietotni no saīsnes mobilajā ierīcē. Uzplaiksnījuma ekrāns “Chrome” interneta pārlūkā ir automātiski ticis ģenerēts no manifesta failā definētajām vērtībām.



4.7. att. **Izstrādātās lietotnes informācijas dati mobilās ierīces lietotņu sarakstā**

Attēlā 4.7. ir parādīts instalētās progresīvās tīmekļa lietotnes informācijas skats, kas pieejams, lietotājam atverot ierīces lietotņu sarakstu mobilajā ierīcē, kur ir redzama informācija par tās izmēru ierīcē, pašpiegādes paziņojumu pārvaldīšanas iestatījumi, mobilo datu lietojums utt.

5. IZSTRĀDĀTĀS LIETOTNES AUDITS “Lighthouse” RĪKĀ

“Lighthouse” rīks ir atvērtā pirmkoda auditu automatizācijas rīks tīmekļa vietņu kvalitātes uzlabošanai. Audits ievāc informāciju par vietnes veiktspēju, progresīvo tīmekļa vietņu prasību ievērošanu, meklētājprogrammu optimizācijām u.c. [24].

Audita realizēšanai tika izmantota “Chrome” interneta pārlūka izstrādātāju rīku “Audit” cilne, kas ievāc informāciju par pašreiz atvērto vietnes lapu.

Tika veikts audits visām trīs lietotnes versijām:

- Parastā lietotne, kas pieejama “master” zarā, pirmkoda repozitorijā,
- Progresīvā tīmekļa lietotne, pieejama “pwa” zarā, pirmkoda repozitorijā, kur servisskripta funkcionalitātes ieviešana ir realizēta ar *@angular/service-worker* pakotni,
- Progresīvā tīmekļa lietotne, pieejama “pwa-custom” zarā, pirmkoda repozitorijā, kur servisskripta funkcionalitātes ieviešana ir realizēta manuāli.

Tabula 5.1. **Izstrādātās lietotnes versiju “Lighthouse” rīka audita rezultāti**

Lietotnes versijas zars repozitorijā	Veiktspēja (performance)	Progresīvā tīmekļa lietotne (progressive web app)	Pieejamība (accessibility)	Labā prakse (best practices)	Meklētājprogrammu optimizācija (SEO)
master	91	45	100	94	100
pwa	92	100	100	100	100
pwa-custom	94	100	100	100	100

Tabulā 5.1. ir redzami “Lighthouse” rīka audita rezultāti katrai no lietotnes versijām. Maksimālais punktu skaits kādā no kategorijām ir 100.

Var novērot, ka veiktspējas kategorija lietotnes versijām īpaši neatšķiras. Visticamāk, tas ir tāpēc, ka lietotne ir samērā maza gan funkcionalitātes gan izmēra ziņā, līdz ar to progresīvo tīmekļa lietotņu optimizācijām nav tik liela efekta, kāds tas varētu būt funkcionalitātes un izmēra ziņā lielākās tīmekļa vietnēs. Neliels uzlabojums veiktspējā, salīdzinot manuālo servisskripta funkcionalitāti ar *@angular/service-worker* pakotnes funkcionalitāti, visticamāk ir tāpēc, ka pakotne izmēra ziņā ir daudz lielāka, nekā manuālā funkcionalitāte, līdz ar to vairāk datu ir jāsūta tīklā.

Kopumā var novērot, ka progresīvās tīmekļa lietotnes versijas uzrāda labāku rezultātu, kas visticamāk būtu vēl ievērojamāks lielākām tīmekļa lietotnēm.

Viens no izvirzītajiem mērķiem izstrādātajai lietotnei bija maksimālā punktu skaita iegūšana "Lighthouse" auditu rīkā progresīvo tīmekļa lietotņu kategorijā. Tabulā 5.1. ir redzams, ka abas lietotnes versijas, kurām ir piemēroti progresīvo tīmekļa lietotņu uzlabojumi ir sasniegušas mērķi, izpildot 11 kritērijus:

- Reģistrē servisskriptu,
- Atbild ar 200 HTTP statusa kodu bezsaistē jeb spēj darboties bezsaistes režīmā,
- Lapai ir saturs, ja JavaScript pārlūkā nav pieejams,
- Izmanto HTTPS protokolu,
- Novirza HTTP pieprasījumus uz HTTPS,
- Lapas ielādes ātrums ir gana ātrs 3G savienojumā,
- Lieotājam tiks piedāvāts instalēt lietotni,
- Konfigurēta individuālam uzplaiksnījuma ekrānam,
- Adreses josla krāsa sakrīt ar brenda krāsu,
- Satur <*meta name="viewport"*> elementu ar "width" vai "initial-scale" vērtību,
- Saturs pareizi ietilpst ekrānā

Audits veikts "Lighthouse" 2.9.1 versijā.

6. IZSTRĀDĀTĀS LIETOTNES VERSIJU SALĪDZINĀJUMS

Kā jau minēts, autora lietotne ir izstrādāta trīs versijās. Šajā nodaļā lietotnes versijas tiek salīdzinātas, lai noskaidrotu, tieši kādi ieguvumi, atšķirības progresīvai tīmekļa lietotnei ir pār parastu tīmekļa lietotni.

6.1. Lietotnes tīmekļa pieprasījumu datu apjoms

Izmēra datu iegūšanai tika izmantota "Chrome" interneta pārlūka tīkla datu cilne, kas parāda pārsūtīto datu izmēru. Katrai lietotnes versijai tika apskatītas četras vietnes lapas – sākumlapa, visu bestselleru sarakstu lapa, viena bestselleru grāmatu saraksta lapa un vienas grāmatas informācijas lapa. Datu izmēru mērvienības ir kilobaiti.

Tabula 6.1. **Tīmekļa pieprasījumu datu apjoms pirmajā lietotnes apmeklējumā**

Lietotnes versijas zars	Sākumlapa	Visu bestselleru saraksts	Bestsellera grāmatu saraksts	Grāmatas informācija	Vidējais rezultāts
master	264	125	11.2	64.8	116
pwa	516	125	11.2	64.8	179
pwa-custom	1400	125	11.1	64.8	400

Tabulā 6.1. ir redzami pārsūtīto tīmekļa pieprasījumu datu apjomi katrai no lietotnes versijām, pirmajā vietnes apmeklēšanas reizē. Redzams, ka ievērojami lielāks datu apjoms ir progresīvās tīmekļa lietotnes versijām. Tas ir izskaidrojams ar to, ka ielādējot lapu pirmo reizi tiek veikti tie paši pieprasījumi, kas parastai lietotnei. Papildus tiem, tiek reģistrēts arī servisskripts, kurš tiek instalēts un instalācijas solī veic pieprasījumus uz definētiem resursiem, lai saglabātu tos pārlūka kešatmiņā.

Pārējo lapu dati ir tikai izmantoto API pieprasījumi, tāpēc pirmajā lietotnes apmeklēšanas reizē nav atšķirību starp versijām.

Tabula 6.2. **Tīmekļa pieprasījumu datu apjoms otrajā lietotnes apmeklējumā**

Lietotnes versijas zars	Sākumlapa	Visu bestselleru saraksts	Bestsellera grāmatu saraksts	Grāmatas informācija	Vidējais rezultāts
master	264	125	11.2	64.8	116
pwa	118	113	0	62.7	73
pwa-custom	0	0	0	0	0

Tabulā 6.2. ir parādīti pārsūtīto tīmekļa pieprasījumu datu apjomi lietotnes otrajā apmeklēšanas reizē visām versijām. Redzams, ka progresīvajām tīmekļa lietotnēm vidējais datu apjoms ir samazinājies, apmeklējot vietnes lapas otrreiz, pretēji pirmajā apmleklējumu reizē, kas redzams 6.1. tabulā.

Pamanāms, ka *pwa-custom* zara lietotnes versijai, kurā servisskripts tika izstrādāts manuāli, tīmeklī dati nav pārsūtīti nemaz. Tas skaidrojams ar to, ka manuāli izstrādājot servisskripta funkcionalitāti tika pielietota katra pieprasījuma saglabāšanas kešatmiņā stratēģija. Piemēram, *pwa* zara lietotnes versijā, izstrādājot servisskripta funkcionalitāti ar *@angular/service-worker* pakotni, konfigurācijā nav norādīta fontu saglabāšana kešatmiņā.

Var secināt, ka atkārtoti lapas apmeklējumi progresīvās tīmekļa lietotnēs vidēji patērēs mazāku datu apjomu tīmekļa pieprasījumos nekā parasta lietotne. Tas ļauj, piemēram, mobilās ierīces lietotājam samazināt mobilo datu lietojumu un līdz ar to, gala izmaksas.

6.2. Lietotnes ielādes laiks

Lietotnes ielādes laika datu iegūšanai tika izmantota "Chrome" interneta pārlūka tīkla datu cilne, kas parāda vietnes lapas ielādes laiku. Tiek apskatīta ielādes (load) notikuma vērtība, kas nozīmē, ka lapa ir pilnībā ielādējusies [25].

Katrai lietotnes versijai tika apskatītas četras vietnes lapas – sākumlapa, visu bestselleru sarakstu lapa, viena bestselleru grāmatu saraksta lapa un vienas grāmatas informācijas lapa. Katra lapas ielāde tiek veikta no jauna, nevis pārejot no vienas uz otru, jo autora izstrādātā lietotne ir vienas lapas lietotne, kas nozīmē, ka ielādes (load) notikums tiek reģistrēts tikai vienu reizi – pirmo reizi atverot vietni. Ielādes laiku mērvienība ir milisekundes.

Tabula 6.3. **Lietotnes ielādes laiks pirmajā lietotnes apmeklējumā**

Lietotnes versijas zars	Sākumlapa	Visu bestselleru saraksts	Bestsellera grāmatu saraksts	Grāmatas informācija	Vidējais rezultāts
master	950	356	496	630	486
pwa	991	692	455	535	668
pwa-custom	986	554	365	655	640

Tabulā 6.3. ir parādīti lietotnes pilnas ielādes laiki milisekundēs katrai no apskatītajām vietnes lapām. Var novērot, ka progresīvajām tīmekļa lietotņu versijām vidējais ielādes laiks ir

nedaudz lielāks nekā parastajai lietotnes versijai, kas visticamāk ir dēļ papildus servisskripta funkcionalitātes.

Tabula 6.3. **Lietotnes ielādes laiks otrajā lietotnes apmeklējumā**

Lietotnes versijas zars	Sākumlapa	Visu bestselleru saraksts	Bestsellera grāmatu saraksts	Grāmatas informācija	Vidējais rezultāts
master	621	351	349	310	407
pwa	421	499	493	1760	793
pwa-custom	406	269	235	225	284

Tabulā 6.3. ir parādīti lietotnes pilnas ielādes laiki milisekundēs katrai no apskatītajām vietnes lapām otrreizējā apmeklējumā. Ir redzams, ka *pwa* zara lietotnes versijas vidējais ielādes laiks ir ievērojami lielāks, salīdzinot ar pārējām versijām. Salīdzinoši ilgais ielādes laiks ir dēļ grāmatas informācijas skata ilgās ielādes. Apskatot tīmekļa pieprasījuma detaļas, tika novērots, ka 1350 milisekundes no šī pieprasījuma ir patērētas "Request to ServiceWorker" darbībai. Tas nozīmē, ka pieprasījums tiek sūtīts servisskriptam [26]. Īsti nav skaidrības, kāpēc šī darbība ir tik ilga. Ir iespējama kļūdaina servisskripta konfigurācija vai arī kļūda pašā skripta funkcionalitātē. Kā jau minēts iepriekš, Angular ietvars un *@angular/service-worker* pakotne joprojām ir aktīvā izstrādes stāvoklī.

Daudz ievērojamāks uzlabojums ielādes laikiem ottreizējā lapu apmeklējumā ir *pwa-custom* zara lietotnes versijai. Tas skaidrojams ar, jau 6.1. nodaļā minēto, saglabāšanas kešatmiņā stratēģiju, kā rezultātā mazāk datu ir jāgaida no tīmekļa un lapa spēj ielādēties ātrāk.

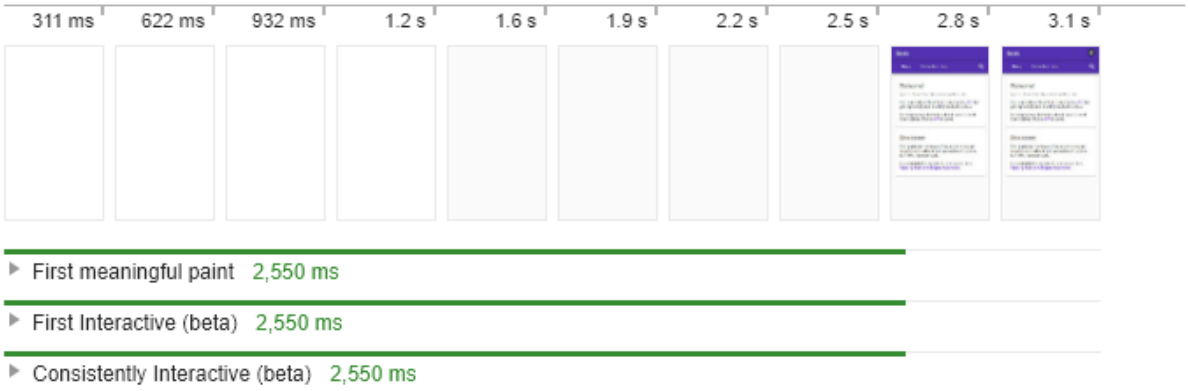
Var secināt, ka ar progresīvo tīmekļa lietotni ir iespējams panākt konstanti mazākus vietnes lapu ielādes laikus, nekā parastai lietotnei. Tomēr ir svarīgi pārliecināties, ka servisskripts funkcionē pareizi un nepalielina lapas ielādes laiku dēļ, piemēram, kļūdainas konfigurācijas.

Ir izdevīgi izmantot servisskripta funkcionalitāti vietnēs, kurās lietotājiem paredzama bieža atgriešanās jau apmeklētās lapās, kā piemēram, autora izstrādātajā lietotnē, kur lietotājs pārvietojas starp grāmatu sarakstiem.

6.3. Lietotnes lapas lietojamības ātrums

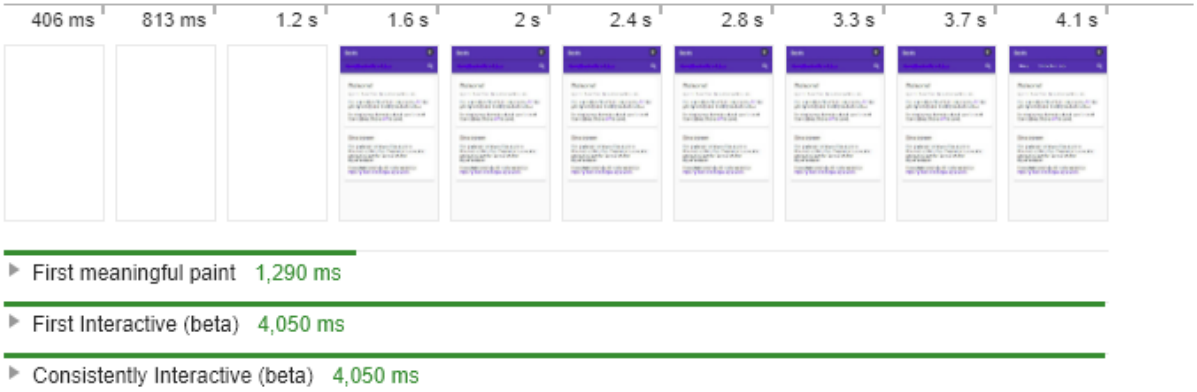
Svarīga vietnes lapas iezīme ir cik ātri lietotājs ierauga pirmos lapas elementus un spēj ar tiem mijiedarboties [27, 28]. Šajā nodaļā tiek apskatīti pirmā nozīmīgā zīmējuma (first meaningful paint) un pirmais mijiedarbojamais lietotnes stāvoklis (first interactive). Datu

iegūšanai ir izmantots “Lighthouse” rīka veiktspējas audits, kas pārbauda lapas, simulējot 3G interneta savienojumu.



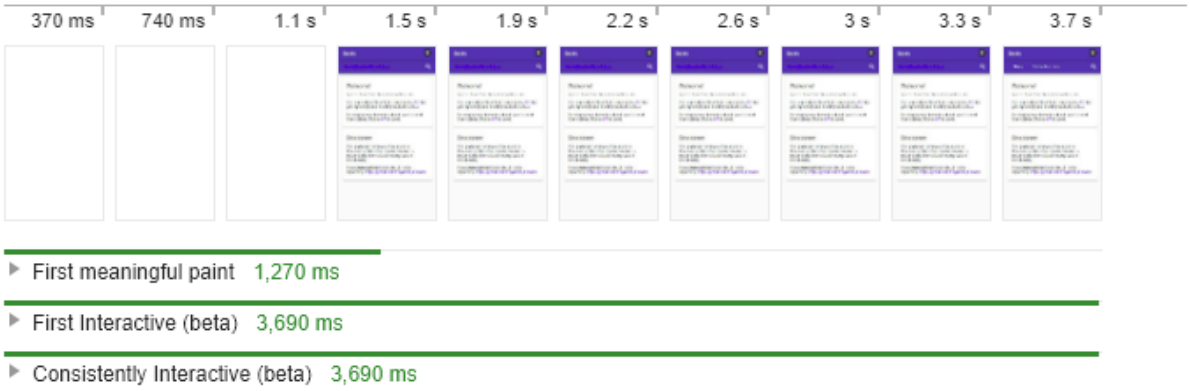
6.1. att. **“Lighthouse” rīka lietotnes *master* zara versijas rezultāti veiktspējas auditā**

Attēlā 6.1. ir redzami audita rezultāti lietotnes *master* zara versijai. Visi nozīmīgie stāvokļi ir notikuši vienā laika brīdī – 2550 milisekundēs, kopš lapas ielādes sākuma. Tas ir tāpēc, ka parastai Angular ietvara lietotnei ir jāsagaida visi resursi no servera, lai tiktu veikta lapas HTML koda ģenerācija un attēlošana lietotājam.



6.2. att. **“Lighthouse” rīka lietotnes *pwa* zara versijas rezultāti veiktspējas auditā**

Attēlā 6.2. ir redzami audita rezultāti lietotnes *pwa* zara versijai. Var novērot, ka pirmais nozīmīgais zīmējums lapā ir noticis divas reizes ātrāk nekā lietotnes *master* zara versijai. Šāds ieguvums ir radies, lietojot čaulas modeli progresīvajām tīmekļa lietotnes versijām. Interneta pārlūks daudz ātrāk spēj parādīt lietotājam jau uzģenerētu HTML kodu, kas saņemts no servera. Pirmais mijiedarbojamais stāvoklis turpretī ir ilgāks, nekā lietotnes *master* zara versijai. Tas skaidrojams ar lielāku JavaScript failu izmēru un lejupielādes laiku dēļ papildus pakotnes servisskripta funkcionalitātes.



6.3. att. **"Lighthouse" rīka lietotnes *pwa-custom* zara versijas rezultāti veiktspējas auditā**

Attēlā 6.3. ir redzami audita rezultāti lietotnes *pwa-custom* zara versijai. Redzams, ka pirmais nozīmīgais zīmējums ir tāds pats, kā lietotnes *pwa* zara versijai, jo lietots tas pats čaulas modelis. Pirmā mijiedarbojamā stāvokļa laiks ir nedaudz samazinājies, jo manuālā servisskripta funkcionalitāte datu apjoma ziņā ir daudz mazāka nekā *@angular/service-worker* pakotnei un netiek iekļauta *main.<hash>.bundle.js* failā, kurā atrodas arī praktiski visa ietvara un lietotnes pamata funkcionalitāta.

Var secināt, ka lietotnes čaulas modelis var radīt iespaidu lietotājam, ka vietnes lapa ielādējas un ir pieejama ātrāk.

Lai uzlabotu Angular progresīvās tīmekļa lietotnes ielādes laiku, servisskripta funkcionalitāti var nodalīt no citiem JavaScript failiem un izsaukt atsevišķi. Tas samazinās pārlūkā HTML ģenerēšanai nepieciešamo datu lejupielādes apjomu.

REZULTĀTI

Darba iztrādes rezultātā ir veikta informācijas izpēte par progresīvajām tīmekļa lietotnēm un Angular ietvaru. Ir definētas svarīgākās progresīvo tīmekļa lietotņu īpašības un to teorētiskie risinājumi, kā piemēram servisskripta lietošana, lai panāktu tīmekļa lietotnes spēju darboties bezsaistē.

Pētījuma rezultātā ir izveidota progresīva tīmekļa lietotne vairākās versijās Angular ietvarā, kā arī veikts tehnisks apraksts funkcionalitātes realizācijai, lai sasniegtu progresīvajām tīmekļa lietotnēm raksturīgas īpašības:

- Ir samazināts tīmeklī sūtāmo datu apjoms ar specifisku Angular ietvara komandrindas komandu lietošanu programmas kompilēšanas brīdī,
- Lietotne tiek servēta, izmantojot "GitHub Pages" servisu, nodrošinot, ka datu apmaiņai tiek lietoti HTTP/2 un HTTPS protokoli,
- Lietotne ir strukturēta tā, lai katrai atšķirīgai lietotnes lapai būtu unikāls URI, izmantojot Angular maršrutēšanas moduli, kas nodrošina iespēju kopīgot un saglabāt lietotnes stāvokļus,
- Ir izmantots Angular ietvara *@angular/material* modulis, kas nodrošina lietotnes satura pareizu attēlošanu jebkāda izmēra ekrānos un seko "Material" dizaina vadlīnijām,
- Ir realizēts lietotnes čaulas modelis, izmantojot Angular Universal bibliotēku, kas nodrošina ātrāku lapas elementu parādīšanos lietotājam atverot lietotni,
- Ir panākta spēja lietotnei darboties vienlīdz ātri bezsaistes režīmā vai sliktas kvalitātes interneta savienojuma gadījumā, lietojot servisskripta funkcionalitāti, kas autora lietotnē ir izstrādāta divos dažādos veidos – lietojot *@angular/service-worker* pakotni, un manuāli izstrādājot servisskripta funkcionalitāti,
- Ir definēts manifesta fails, kas nodrošina, ka lietotne ir instalējama mobilajā ierīcē, ja lietotājs apmeklē tīmekļa vietni.

Ir sasniegts izvirzītais mērķis lietotnei iegūt maksimālo punktu skaitu "Lighthouse" auditu rīka progresīvo tīmekļa lietotņu kategorijā.

Ir veikts 4. nodaļā aprakstīto lietotnes versiju salīdzinājums, lietojot "Lighthouse" auditu datus un "Chrome" interneta pārlūka sniegtos datus par sūtīto tīmekļa datu apjomu, tīmekļa vietnes ielādes laiku, noskaidrojot konkrētu metriku atšķirības starp izstrādātās lietotnes versijām, apstiprinot teorētiskās izpētes daļā doto informāciju par progresīvajām tīmekļa lietotnēm.

SECINĀJUMI

Darba teorētiskās izpētes gaitā tika veikti šādi svarīgākie secinājumi:

- Progresīvās tīmekļa lietotnes ir tīmekļa vietnes, kurām piemīt noteiktas definētas īpašības, kas nav atkarīgas no pamata tehnoloģijas (programmatūras ietvara vai programmatūras pirmkoda valodas), kas lietotas vietnes izstrādē. Tas nozīmē, ka praktiski jebkura tīmekļa vietne var tikt pārveidota, lai atbilstu šīm īpašībām,
- Progresīvo tīmekļa lietotņu mērķis ir uzlabot lietotāja pieredzi tīmekļa vietnē, netieši uzlabojot vietnes apmeklētību, popularitāti, peļņu,
- Eksistē vairākas atšķirīgas definīcijas, par to, ko var uzskatīt par progresīvu tīmekļa lietotni, lai gan visām ir kopīgi noteikti aspekti, kā piemēram, spēja strādāt bezsaistē un instalējamība,
- Progresīvo tīmekļa lietotņu atbalsts ir ieviests visos populārākajos interneta pārlūkos, lai gan tehnoloģija ir samērā jauna, kas nozīmē, ka šī funkcionalitāte var mainīties dažādos aspektos.

Pētījuma veikšanas gaitā tika izdarīti sekojoši secinājumi par progresīvo tīmekļa lietotņu izstrādi Angular ietvarā:

- Angular ietvars ievērojami atvieglo progresīvo tīmekļa lietotņu izstrādi, piedāvājot dažādu funkcionalitāti izstrādes procesu automatizēšanai, lietotojot komandrindas komandas un pakotnes:
    - Komandas *–prod* un *–build-optimizer* spēj samazināt pārkompilētā pirmkoda datu apjomu, tomēr komandrindas dokumentācija ir grūti atrodama dažām specifiskām opcijām,
    - Iebūvētais maršrutēšanas modulis ir ērti izmantojams, lai nodrošinātu, ka katram lietotnes stāvoklim ir unikāls URL. Moduli arī ir iespējams lietot, lai realizētu servera puses renderēšanu,
    - *@angular/material* modulis ir piemērots reaģētspējīga UI izstrādei un atbilst progresīvo tīmekļa lietotņu nosacījumiem saskaņā ar "Lighthouse" auditu,
    - Angular Universal bibliotēka ir piemērota servera puses renderēšanai, izmantojot ietvara maršrutēšanas moduli, tomēr oficiālā dokumentācija nav pilnīga,
    - *@angular/service-worker* pakotne piedāvā pamata servisskripta funkcionalitāti, tomēr nepieļauj servisskripta funkcionalitātes paplašināšanu.

- Angular ietvars atrodas aktīvā izstrādes stadijā, kas nozīmē, ka izstrādātās programmas pirmkods ir jāatjaunina samērā bieži, lai varētu lietotot jaunāko funkcionalitāti,
- Korekti izstrādāta progresīvā tīmekļa lietotne ir instalējama mobilajā ierīcē, kura izmanto Android operētājsistēmu, spēj darboties bezsaistes režīmā un ir ātri un ērti lietojama jebkāda interneta savienojuma kvalitātes apstākļos,
- Viena no svarīgākajām progresīvās tīmekļa lietotnes komponentēm ir servisskripts, kura funkcionalitāte ir atbalstīta visos populārākajos interneta pārlūkos, izņemot "Internet Explorer" pārlūkus.

Salīdzinot autora izstrādātās 4. nodaļā aprakstītās lietotnes versijas, tika veikti sekojoši nozīmīgākie secinājumi:

- Saskaņā ar "Lighthouse" auditu, progresīvai tīmekļa vietnei ir augstāka veiktspēja nekā parastai tīmekļa vietnei ar tādu pašu funkcionalitāti,
- Lietotnes čaulas modeļa lietošana Angular lietotnē var samazināt pirmo elementu parādīšanās laiku ekrānā vismaz divas reizes, uzlabojot lietotāja pieredzi vietnē,
- Progresīvās tīmekļa lietotnes spēj samazināt vidējo mobilo datu lietojumu, vienlaikus palielinot tīmekļa vietnes veiktspēju, kas var būt būtisks faktors reģionos, kur mobilo datu pakalpojumu izmaksas ir samērā augstas,
- Progresīvās tīmekļa lietotnes ir īpaši izdevīgas vietnēs, kur lietotājam ir paradzams samērā liels skaits ottreizēju lapu apmeklējumu.

IZMANTOTĀ LITERATŪRA UN AVOTI

[1] T. Ater, *Building Progressive Web Apps: Bringing the Power of Native to the Browser.* Sebastopol: O'Reilly Media, 2017.

[2] "Progressive Web Apps" [tiešsaiste] [skatīts 14.05.2018]. Pieejams: https://developers.google.com/web/progressive-web-apps/

[3] K. Farrugia, "A Beginner's Guide To Progressive Web Apps" [tiešsaiste] [skatīts 14.05.2018]. Pieejams: https://www.smashingmagazine.com/2016/08/a-beginners-guide-to-progressive-web-apps/

[4] J. Wagner, "Why Performance Matters" [tiešsaiste] [skatīts 14.05.2018]. Pieejams: https://developers.google.com/web/fundamentals/performance/why-performance-matters/

[5] A.Shellhammer, "The need for mobile speed: How mobile latency impacts publishers revenue" [tiešsaiste] [skatīts 14.05.2018]. Pieejams: https://www.doubleclickbygoogle.com/articles/mobile-speed-matters/

[6] I. Grigorik, "Eliminating Unnecessary Downloads" [tiešsaiste] [skatīts 14.05.2018]. Pieejams: https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/eliminate-downloads

[7] I. Grigorik, *High Performance Browser Networking.* Sebastopol: O'Reilly Media, 2013.

[8] A. Osmani, "Automating Image Optimization" [tiešsaiste] [skatīts 14.05.2018]. Pieejams: https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/automating-image-optimization/

[9] A. Osmani, “The App Shell Model” [tiešsaiste] [skatīts 14.05.2018]. Pieejams: https://developers.google.com/web/fundamentals/architecture/app-shell

[10] P. LePage, “Add to Home Screen” [tiešsaiste] [skatīts 15.05.2018]. Pieejams: https://developers.google.com/web/fundamentals/app-install-banners/

[11] M. Gaunt. un P. Kinlan, “The Web App Manifest” [tiešsaiste] [skatīts 15.05.2018]. Pieejams: https://developers.google.com/web/fundamentals/web-app-manifest/

[12] “Angular dokumentācija”, 2010 [tiešsaiste] [skatīts 16.05.2018]. Pieejams: https://angular.io/docs

[13] “The New York Times Developer Network” [tiešsaiste] [skatīts 16.05.2018]. Pieejams: https://developer.nytimes.com/

[14] “Open Library Books API”, 2008 [tiešsaiste] [skatīts 16.05.2018]. Pieejams: https://openlibrary.org/dev/docs/api/books

[15] “Deployment” [tiešsaiste] [skatīts 16.05.2018]. Pieejams: https://angular.io/guide/deployment

[16] “Angular Material”, 2010 [tiešsaiste] [skatīts 17.05.2018]. Pieejams: https://material.angular.io/

[17] “Angular Universal: server-side rendering” [tiešsaiste] [skatīts 17.05.2018]. Pieejams: https://angular.io/guide/universal

[18] “Angular App Shell – Boosting Application Startup Performance” [tiešsaiste] [skatīts 18.05.2018]. Pieejams: https://blog.angular-university.io/angular-app-shell/

[19] “Angular Universal”, 2010 [tiešsaiste] [skatīts 18.05.2018]. Pieejams: https://universal.angular.io/

[20] “Angular service worker introduction” [tiešsaiste] [skatīts 18.05.2018]. Pieejams: https://angular.io/guide/service-worker-intro

[21] “Angular Service Worker – Step-By-Step Guide for turning your Application into a PWA” [tiešsaiste] [skatīts 18.05.2018]. Pieejams: https://blog.angular-university.io/angular-service-worker/

[22] “Using Service Workers” [tiešsaiste] [skatīts 19.05.2018]. Pieejams: https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers

[23] M. Gaunt, “Service Workers: an Introduction” [tiešsaiste] [skatīts 19.05.2018]. Pieejams: https://developers.google.com/web/fundamentals/primers/service-workers/

[24] “Lighthouse” [tiešsaiste] [skatīts 20.05.2018]. Pieejams: https://developers.google.com/web/tools/lighthouse/

[25] K. Basques un M. Kearney, “Measure Resource Loading Times” [tiešsaiste] [skatīts 20.05.2018]. Pieejams: https://developers.google.com/web/tools/chrome-devtools/network-performance/resource-loading

[26] K. Basques, “Network Analysis Reference” [tiešsaiste] [skatīts 20.05.2018]. Pieejams: https://developers.google.com/web/tools/chrome-devtools/network-performance/reference#timing-explanation

[27] “First Meaningful Paint” [tiešsaiste] [skatīts 20.05.2018]. Pieejams: https://developers.google.com/web/tools/lighthouse/audits/first-meaningful-paint

[28] “First Interactive” [tiešsaiste] [skatīts 20.05.2018]. Pieejams: https://developers.google.com/web/tools/lighthouse/audits/first-interactive

[29] S. Fluin, “Angular 5.1 & More Now Available” [tiešsaiste] [skatīts 21.05.2018]. Pieejams: https://blog.angular.io/angular-5-1-more-now-available-27d372f5eb4e

[30] “Material Design” [tiešsaiste] [skatīts 21.05.2018]. Pieejams: https://material.io/design/introduction/#principles

[31] “IEEE Style” [tiešsaiste] [skatīts 21.05.2018]. Pieejams: http://pitt.libguides.com/citationhelp/ieee

[32] “The 2016 U.S. Mobile App Report” [tiešsaiste] [skatīts 21.05.2015]. Pieejams: https://www.comscore.com/Insights/Presentations-and-Whitepapers/2016/The-2016-US-Mobile-App-Report?cs_edgescape_cc=LV

[33] A. Russel, “What, Exactly, Makes Something A Progressive Web App?” [tiešsaiste] [skatīts 21.05.2015]. Pieejams: https://infrequently.org/2016/09/what-exactly-makes-something-a-progressive-web-app/

[34] “Developer Survey Results 2018” [tiešsaiste] [skatīts 25.05.2018]. Pieejams: https://insights.stackoverflow.com/survey/2018/#technology

[35] “assets-webpack-plugin” [tiešsaiste] [skatīts 25.05.2018]. Pieejams: https://github.com/kossnocorp/assets-webpack-plugin

Bakalaura darbs „Progresīvo tīmekļa lietotņu izstrāde Angular ietvarā” izstrādāts LU Datorikas fakultātē.

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: Edgars Joja_____ ___.05.2018.

Rekomendēju/nerekomendēju darbu aizstāvēšanai

Vadītājs: profesors, Dr. dat. Uldis Straujums_____ ___.05.2018.

Recenzents: asociētais profesors Edgars Celms

Darbs iesniegts Datorikas fakultātē 28.05.2018.

Dekāna pilnvarotā persona: vecākā metodiķe Ārija Sproģe_____

Darbs aizstāvēts bakalaura gala pārbaudījuma komisijas sēdē

___.06.2018. prot. Nr. _____

Komisijas sekretārs(-e): _____