LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

MŪSDIENU IESPĒJAS TĪMEKĻA UN MOBILO VAIRĀKPLATFORMU LIETOTŅU IZSTRĀDĒ

BAKALAURA DARBS

Autore: **Viktorija Savicka**

Studenta apliecības Nr.: vs16048

Darba vadītāja: Dr. dat. Darja Solodovņikova

RĪGA 2020

ANOTĀCIJA

Pieaug nepieciešamība piedāvāt gan mobilās, gan pārlūka lietotnes, lai nodrošinātu labāku lietotāja pieredzi. Vairākplatformu mobilās izstrādes sistēmas tagad atbalsta arī tīmekļa lietotņu izstrādi kopā ar mobilajām ierīcēm. Darbā, pamatojoties uz dažādiem literāriem avotiem un praktisko pieredzi, tika izpētīti dažādas pieejas un rīki, to iezīmes un izmantošanas potenciāls. No tīmekļa vietņu izstrādes viedokļa tika aprakstītas un apsvērtas vairākplatformu lietotņu grupas. Progresīvās tīmekļa lietotnes, Ionic, NativeScript un Flutter tika salīdzinātas pēc izstrādes procesa, ierīču iespēju piekļuves, pārlūka un operētājsistēmas atbalstu, koda koplietošanas, publicēšanas iespējam.

Atslēgvārdi: tīmekļa izstrāde, mobilās lietotnes, vairākplatformu izstrāde, Ionic, NativeScript, Flutter

ABSTRACT

MODERN APPROACHES FOR WEB AND MOBILE CROSS-PLATFORM APP DEVELOPMENT

There is an increasing need to provide both mobile and browser applications to provide a better user experience. Cross-platform mobile development frameworks now begin to support development of web applications along with mobile. This study explores the different approaches and tools available, their features and potential use, based on different literary sources and practical experimentation. Cross-platform app groups were described and considered from perspective of website development. Progressive Web Apps, Ionic, NativeScript and Flutter were compared by development environments, web and mobile, device feature support, browser and operating system support, code reusability, app publishing.

Keywords: web development, mobile apps, cross-platform development, Ionic, NativeScript, Flutter

SATURS

APZĪMĒJUMU SARAKSTS

API (saīs. Application Programming Interface) – lietojumprogrammas interfeiss; funkciju kopums, kas ļauj lietojumprogrammām piekļūt pie operētājsistēmas, pakalpojuma vai citas lietojumprogrammas funkcijām vai datiem.

CSS (saīs. Cascading Style Sheets) – kaskādisku stilu saraksts; dokuments, kas apraksta tīmekļa vietnes vizuālo izskatu.

CLI (saīs. Command Line Interface) - teksta interfeiss, ko izmanto programmatūras darbināšanai.

HTML (saīs. Hyper Text Markup Language) – iezīmēšanas valoda, izmantojot kuru, tiek noteikta tīmekļa vietnes struktūra un atveidojums.

IE11 (saīs. Internet Explorer 11) – interneta pārlūks uz Windows operētājsistēmas.

JS (saīs. JavaScript) – programmēšanas valoda, kas tiek izmantotā tīmekļa vietnēs.

NS (saīs. NativeScript) – Android un iOS mobilo lietotņu izstrādes ietvars.

OS (saīs. operētājsistēma) – programmatūra, kas vada datu organizēšanu un programmu izpildi, nodrošina aparatūras saskaņotu darbību, resursu racionālu izmantošanu ierīcē.

PWA (saīs. Progressive Web App) – lietojumprogrammatūras veids, kas tiek piegādāts caur interneta pārlūku.

TS (saīs. TypeScript) – statiski tipizēta tīmekļa programmēšanas valoda, uzbūvēta uz JavaScript pamata.

UI (saīs. User Interface) – lietotāju saskarne.

Vairākplatformu izstrāde (angl. cross-platform development) - programmatūras izstrādes prakse izstrādāt lietojumprogrammas, kas strādā uz vairākām platformām vai programmatūras vidēm.

Vietēja lietotne (angl. native app) – lietojumprogramma, kas izstrādāta, izmantojot platformas vai ierīces izvēlētu programmēšanas valodu un paredzēta lietošanai konkrētā platformā vai ierīcē.

W3C standarti (saīs. World Wide Web Consortium) - tehnisko specifikāciju un vadlīniju kopums pārlūkprogrammām, tīmekļa vietnēm, tīmekļa pakalpojumiem un meklētājprogrammām.

XML (saīs. eXtensible Markup Language) – iezīmēšanas valoda, kas ļauj lietotājiem definēt savas personiskās iezīmes, lai ar to nodrošinātu noteiktu funkciju izpildi lietojumprogrammā.

IEVADS

Šodien lielāka daļa cilvēku vēlas tūlītēju piekļuvi pie tīmekļa lietotnēm, izmantojot savu viedtālruni. Tomēr, parastās pārlūkprogrammas lietotnes ne vienmēr spēj piedāvāt visas iespējas vai vajadzības, ko lietotāji vēlas redzēt uz saviem tālruņiem, piemērām, piekļuve fotokamerai vai atrašanās vietas izsekošana. Tas ierosina izstrādātājus meklēt citus veidus, kā apmierināt šīs vajadzības. Visbiežāk sastopamais risinājums ir izveidot mobilo lietotni kā paplašinājums kādai esošajai tīmekļa programmatūrai.

Vienlaikus izveidot un uzturēt gan mobilo lietotni, gan tīmekļa lietotni ievērojami palielina izstrādes procesam nepieciešamās pūles un resursus, īpaši tad, kad nepieciešams atbalstīt vairākas mobilās operētājsistēmas. Par risinājumu šī problēmai tiek uzskatīti vairākplatformu lietotņu izstrādes rīki, jo viņi ļauj izveidot lietotnes izmantošanai uz vairākām platformām, vienlaikus atkārtoti izmantojot lielāko daļu no uzrakstītā koda.

Iepriekš veikti vairākplatformu izstrādes pētījumi bieži vien koncentrējas tikai uz lietotņu izstrādi mobilajām ierīcēm. Tā kā liela daļa no mūsu ikdienas darba notiek tieši datora pārlūkā, tad, izstrādājot programmatūru, svarīgi ņemt vērā ne tikai mobilās ierīces, bet arī pārlūka atbalstu. Vairākplatformu izstrādes rīku izstrādātāji atzīst šo problēmu, un modernās vairākplatformu tehnoloģijas tagad savā izstrādes iespēju komplektā iekļauj arī tīmekļa vietņu izstrādi kopā ar mobilām lietotnēm.

Tagad izstrādātājiem ir pieejami vairāki jaudīgi rīki viedtālruņu lietotņu un tīmekļa lietotņu vairākplatformu izstrādei, balstīti uz tīmekļa tehnoloģijām. Lielā daļa iepriekšējo pētījumu šajā jomā ir rakstīti, pamatojoties uz vecākām tehnoloģijām, kas tika uzskatīti par lēnam un zemas veiktspējas. Daži no šīm jaunām rīkiem ir mazāk pazīstami izstrādātāju vidē. Šīs darbs informēs lasītāju par mazāk populārām tehnoloģijām, kurus varētu ņemt vērā, izstrādājot tīmekļa un mobilās lietotnes nākotnē.

Tādējādi darba mērķis ir informēt lasītāju par mūsdienīgiem rīkiem, kas pieejami tīmekļa un mobilo lietotņu vienlaikus izstrādei un palīdzēt izvēlēties piemērotāko izstrādes procesam.

Darbā tika izvirzīti vairāki uzdevumi:

- izpētīt pašreizējās tendences tīmekļa un mobilo lietotņu lietošanā, ka arī vairākplatformu rīku interesi izstrādātāju kopienā;
- aprakstīt dažādus vairākplatformu lietojumprogrammu veidus un rīkus;
- atrast kādas iespējas tīmekļa lietotņu izstrādē piedāvā šī rīki;

- salīdzināt dažus no vairākplatformu izstrādes rīkiem, kas atbalsta vienlaikus tīmekļa un mobilo izstrādi;
- izzināt katras attīstības pieejas un apskatītu rīku īpašās iezīmes, stiprās un vājās puses.

Darbā tiek izmantotas vairākas pētniecības metodes, tostarp - analīzes metode, salīdzināšanas metode, loģiski konstruktīvā metode, eksperimenta metode.

Darbā izmantotie avoti iekļauj sevī zinātniskās publikācijas, grāmatas, statistikas pārskati, programmatūras dokumentācija un interneta materiāli.

Pirmajā sadaļā tiks sniegts ieskats par pašreizējām mobilo ierīču un datoru tīmekļa lietošanas tendencēm, lai uzsvērtu nepieciešamību pie pārlūkprogrammu tīmekļa lietotnēm un mobilām lietotnēm. Turpat arī tiks aprakstīta tīmekļa izstrādātāju kopienas interese par jaunajiem vairākplatformu izstrādes risinājumiem.

Otrajā sadaļā tiks aprakstītas dažādas metodes, kas tiek pielietotas vairākplatformu lietotņu radīšanā, lai gūtu dziļāku izpratni par pašlaik pieejamajām izstrādes tehnoloģijām un to darbības principus, identificētu to pielietojumus ne tikai mobilo lietotņu, bet arī tīmekļa vietņu izstrādei.

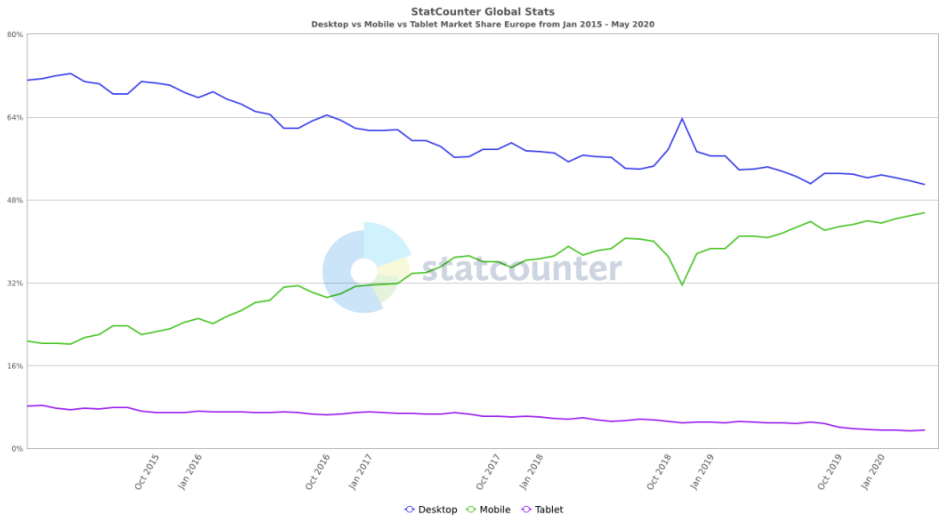
Trešajā sadaļā tiks veikts dažādu veidu vairākplatformu tehnoloģiju un izstrādes ietvaru salīdzinājums. Tas sniegs pārskatu par to pieejamām funkcijām, izstrādes vidēm, pirmkoda atkārtotas izmantojamības potenciālu, atbalstu dažādās pārlūkprogrammās un mobilajās operētājsistēmās. Pamatojoties uz salīdzinājumu rezultātiem, tiks piedāvātas rekomendācijas, lai palīdzētu izstrādātājiem pieņemt lēmumu par piemērotu izstrādes sistēmu, lai izveidotu viedtālruņa lietojumprogrammas vienlaikus kopā ar tīmekļa lietotni, atbilstoši savām vajadzībām.

1. SITUĀCIJAS APRAKSTS

Nodaļā tiks sniegta informācija par pašreizējām tendencēm tirgū saistībā ar mobilo ierīču un personālo datoru izmantošanu. Pasaules digitālais tirgus pastāvīgi mainās un veids, kā mēs šodien strādājam ar internetu, datoriem un mobiliem telefoniem, ļoti atšķiras no tā, kā tas bija pirms vairākiem gadiem. Tāpēc ir svarīgi sekot līdzi šīm izmaiņām.

1.1. Datoru un mobilo ierīču izmantošanas tendences

Pēc starptautiskās statistikas biroja *StatCounter* datiem (*1.1. att.*) redzams, ka mobilo ierīču un planšetdatoru tirgus daļa internetā Eiropā ir pieaugusi no 28% līdz 47% pēdējo 5 gadu laikā [1]. Līdzīgs pieaugums ir arī novērojams tieši Latvijā – mobilo ierīču un planšetdatoru tirgus daļa ir pieauguši no 10% līdz 34% [2]. Tas norāda uz tendenci, ka lielāka un lielāka daļa no interneta lietotājiem sāk vairāk izmantot viedtālruņus, nevis datorus, savam ikdienas darbībām.

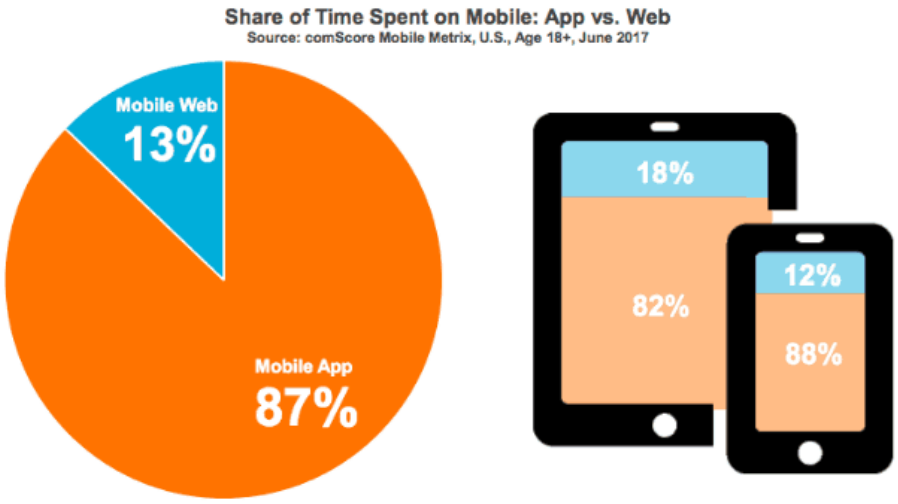


1.1. att. **Datoru (zils), viedtālruņu (zaļš) un planšetdatoru (violeta) tirgus daļa 2015-2020 Eiropas tirgū [1]**

Tomēr būtiski atzīmēt, ka vispārējais pieaugums pēdējā gada laikā ir palēninājies, jo tirgus daļa starp mobilo un galddatoru ir sadalīta uz pusēm. Tas nozīmē, ka abu veidu ierīces ir vienlīdz svarīgas interneta lietotājiem.

Kompānijas *Perificient* pētījums izsecināja, ka interneta lietotāji parasti izmanto datorus sarežģītākiem uzdevumiem, piemēram, veidlapām, pirkumiem vai darbam, bet izmanto mobilās ierīces, piemēram, izklaidei vai pārlūkošanai. Cilvēki parasti pavada divreiz ilgāk laika tīmekļa vietnē uz datora pārlūka, nekā uz mobilās ierīces, fokusējoties uz sarežģītākiem uzdevumiem. Papildus, bieži vien cilvēki uzsāk kādu uzdevumu uz savas mobilās ierīces, pēc tām vēlas turpināt to uz datora [3].

Tajā pašā laikā vienkāršas pārlūka lietotnes mobilajām ierīcēm bieži vien ir nepietiekamas programmu lietotājiem. Pēc *ComScore* veiktajā pētījuma ASV vairāk nekā 80% no mobilajās ierīcēs pavadītā laika tiek tērēti, izmantojot mobilo lietotni, nevis pārlūku. Tas ir tāpēc, ka lietotāju uzskata, ka mobilajām lietotnēm ir labāka veiktspēja ar vairāk iespējam [4].



1.2. att. **Mobilās lietotnēs (oranžs) un mobilā pārlūka (zils) pavadīts laiks, procentos [4]**

Tādējādi ir ļoti svarīgi izstrādāt vairākplatformu lietojumprogrammas, kas ir paredzētas lietošanai uz abām vidēm - gan uz mobilām ierīcēm, gan uz datora pārlūka. Mobilās ierīces ir tikpat populāras, kā datori, un katrai darbības videi ir savs pielietojums. Lai lietotājiem nodrošinātu vislabāk iespējamo pieredzi ar tīmekļa lietojumprogrammatūru, ir jāapsver iespēja izstrādāt risinājumu uz abām vidēm.

1.2. Vairākplatformu izstrādes attīstība un popularitāte

Diemžēl pilnībā vietējo mobilo lietotnes izstrāde, tas ir, izstrādātas izmantošanai tieši noteiktajā operētājsistēmā, īpaši tad, ja ir jāatbalsta gan Android, gan iOS operētājsistēmas, ir ļoti sarežģīts un dārgs uzdevums. Šāda izstrādes pieeja ir dārga, laikietilpīga, un pieprasa lielas pūles no izstrādes komandas. Tas var izraisīt kavēšanos lietotnes izlaišanā, jo izstrādes process notiek atšķirīgi uz katras vides. Ievērojama arī situācija, kad dažas funkcijas ir pieejamas vienā vidē, bet vēl nav pieejamas citā [5, 6].

Lai mazinātu papildu izmaksas un izstrādes laiku, izstrādātāji parasti izmanto vairākplatformu izstrādes ietvarus, kas ļauj izstrādāt lietotnes uz vairākām sistēmām, izmantojot vienu un to pašu pirmkodu, minimāli papildināšot un pielāgojot kodu priekš vienas noteiktās sistēmas.[5, 6] Šī pieeja bieži vien ietekmē lietotnes veiktspēju un ātrdarbību, jo pastāv papildu apstrādes līmenis salīdzinājumā ar vietējām lietojumprogrammām [5].

Izstrādātāju aprindas uzmanību ir piesaistījušas vairākas izstrādes ietvari un tehnoloģijas, lai izveidotu lietotnes uz mobilām ierīcēm, tīmekļa lietotnes un datora lietotnes, strādājot tikai ar vienu pirmkodu. Globālā tīmekļa izstrādātāju aptaujā "*The State of JavaScript*" izstrādātājiem tika jautāts, kādas vairākplatformu izstrādes ietvarus viņi ir izmantojuši un vēlētos izmantot atkārtoti, vai viņi ir dzirdējuši par un ir ieinteresēti [7]. Zemāk redzamajā tabulā (1.1. tabula) ir apkopoti atbildes un to skaits, sakārtoti pēc popularitātes. Atbildes slīprakstā tika sniegtas kā brīva teksta atbildes, un tās nebija daļa no aptaujas pieejamajām izvēles atbildēm, kas varētu ietekmēt aptaujas rezultātus.

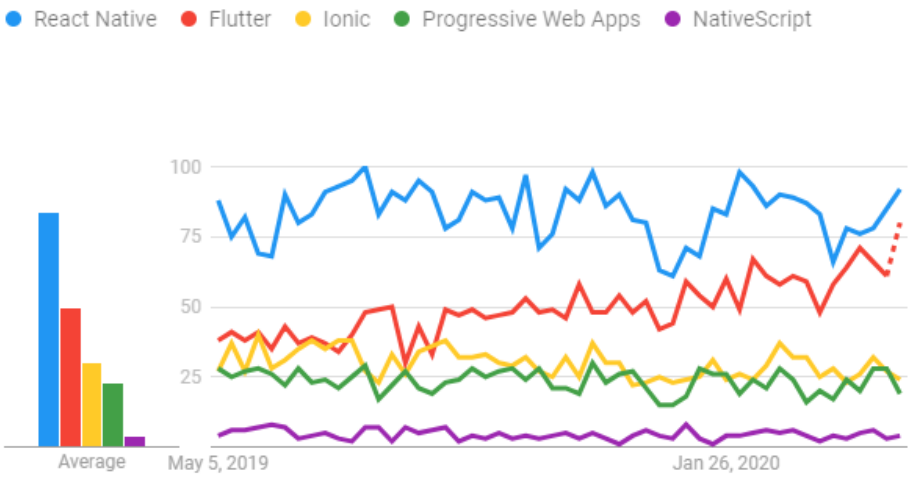
1.1. tabula

Tīmekļa un mobilo vairākplatformu izstrādes ietvaru popularitāte [7]

Rīks	**React Native**	**Ionic**	**Expo**	**Cordova**	**NativeScript**	**Flutter**	**PWA**	**Vue Native**	**App-celerator**
Atbildes	14090	5600	4509	3751	*321*	*185*	*50*	*19*	*13*

Kā redzams, React Native pašlaik ir līderis popularitātē uz tīmekļa tehnoloģijām balstīto vairākplatformu izstādes ietvaru vidū. Ionic un Expo ietvari ir sekojoši pēc popularitātes. Ietvari, kas tika minēti visvairāk ārpus aptaujā dotajām atbildēm, bija NativeScript un Flutter.

Lai iegūtu labāku priekšstatu par šo tehnoloģiju popularitāti, kas netika piedāvātas aptaujā, tika izmantots Google Trends, kas attēlo meklēšanas tendences pasaules visvairāk lietotajā meklētājā. Tehnoloģiju meklēšanas popularitāte ir redzama grafikā, kas apraksta, cik bieži cilvēki meklē informāciju par noteiktu izstrādes platformu pēdējā gada laikā *(1.3. att.)*.



1.3. att. **Meklēšanas popularitāte pēdējā gada laikā - React Native, Flutter, Ionic, NativeScript un PWA [8]**

React Native joprojām ir vispopulārākā tehnoloģija. Flutter ir augošākais popularitātes ziņā pēdējā gadu laikā. Ionic, NativeScript un PWA tehnoloģijas ir mazāk populāri gan aptaujas, gan Google Trends rezultātos.

Var redzēt, ka šobrīd izstrādātājiem ir pieejamas vairākas iespējas vairākplatformu izstrādē, dažas labi zināmas un dažas mazāk. Pirms izstrādājot kādu tīmekļa un mobilo risinājumu, svarīgi apsvērt un salīdzināt iespējas, jo katrai tehnoloģijai ir savas priekšrocības un trūkumi, kā arī jānosaka konkrēti izmantošanas gadījumi dažādiem lietotņu izstādes veidiem.

2. VAIRĀKPLATFORMU IZSTRĀDES PIEEJAS UN IZSTRĀDES RĪKI

Pirms tīmekļa un mobilo vairākplatformu izstrādes ietvaru detalizētākas apskatīšanas un salīdzināšanas, ir svarīgi vispirms izanalizēt un izprot to daudzveidību, struktūru un darbības veidu. Sadaļā tiks sniegts ieskats par vairākplatformu izstrādes tehnoloģiju grupām.

Iepriekšējie darbi par vairākplatformu izstrādes tēmu galvenokārt bija vērsti uz izstrādi starp mobilajām operētājsistēmām. El-Kassas u.c. veiktajā metaanalīzē ir redzams, pastāv plašs diskurss par dažādu platformu attīstības pieeju klasificēšanu – dažas ir detalizētākas, dažas abstraktākas [9].

Lai skaidrāk definētu pašreizējās pieejas un klasificētu arī jaunus tīmekļa un mobilo izstrādes rīkus, par pamatu tika izvēlēta Ksantopulos un Ksinogalos izveidotā klasifikācija. Vairākplatformu mobilos risinājumus klasificē 4 dažādās kategorijās, grupējot tos pēc to implementācijas veida — tīmekļa lietotnes, hibrīdlietotnes, interpretētas lietotnes un uzģenerētas lietotnes [10].

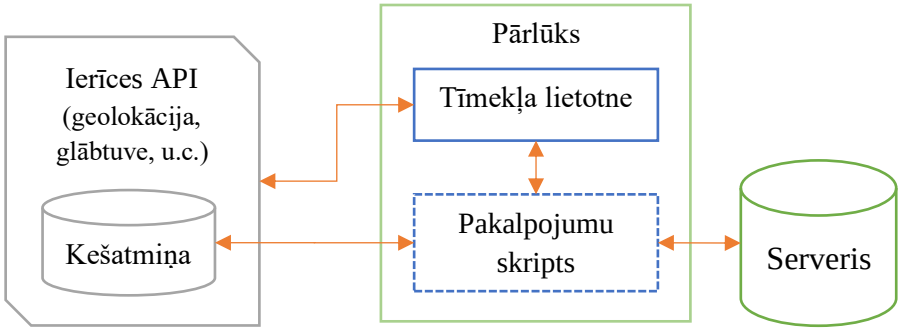
Šī klasifikācijas sistēma tiks izmantota, lai aprakstītu dažādas tīmekļa un mobilo lietojumprogrammu izstrādes pieejas un izstrādes ietvaru tipus.

2.1. Tīmekļa lietotnes

Tīmekļa lietotnes tiek izstrādātas kā tīmekļa vietnes, izmantojot tehnoloģijas HTML, JavaScript un CSS. Šī veida lietojumprogrammas nav nepieciešams instalēt uz mobilās ierīces, lai tās varētu izmantot, jo tām var piekļūt caur pārlūkprogrammu pēc ievadītas adreses [10].

Progresīvās tīmekļa lietotnes (angl. Progressive Web Apps/PWA) ir tīmekļa lietojumprogrammu standarts, kas caur mobilo pārlūku nodrošina vietējai mobilai lietotnei līdzīgu lietotāju pieredzi. Izmantojot PWA tehnoloģijas, iespējams izveidot mobilās lietotnes, kas spēj piekļūt pie mobilās ierīces vai datora funkcijām visās pārlūkprogrammās, kas atbilst W3C standartiem [11].

Progresīvās tīmekļa lietotnes bezsaistes funkcionalitāte tiek ieviesta, izmantojot pakalpojumu skriptus (angl. service workers). Pakalpojumu skripts tiek piereģistrēts pārlūkprogrammā, kad lapa pirmo reizi ielādējas un pēc tām darbojas neatkarīgi no tīmekļa vietnes koda, apstrādā pieprasījumus starp klientu un serveri, saglabā datus kešatmiņā, saglabā pieprasījumus līdz savienojuma atjaunošanai (2.1. att.) [11]. Vietējai lietotnei līdzīga pieredze – neredzama pārlūka saskarne, instalēšana uz ierīces – tiek izveidota, izmantojot JSON "manifesta" failu, kas nosaka, kā tīmekļa lietotnei ir jāizskatās un jāuzvedas lietotājam [12].



2.1. att. **Progresīvas tīmekļa lietotnes darbības procesa diagramma**

PWA tehnoloģiju ieviešana ir vienkāršākais veids, kā pārveidot tīmekļa vietni, lai nodrošinātu mobilo ierīču lietotājiem papildus funkcionalitāti bez nepieciešamības instalēt lietotni no lietotņu veikala.

Lielākā daļa moderno tīmekļa izstrādes ietvaru ļauj automātiski pievienot PWA pamatiespējas tīmekļa lietojumprogrammai, tostarp:

- Angular
- React
- Vue.js
- Polymer

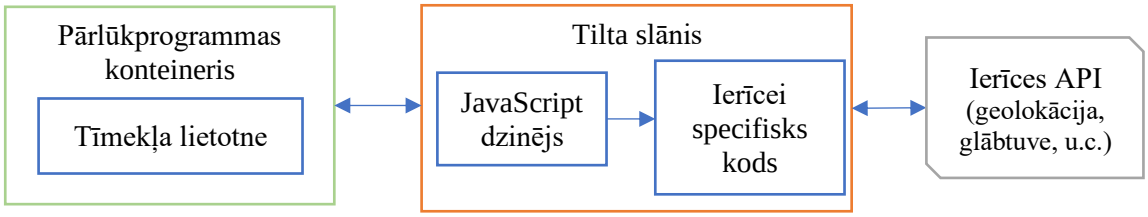
Daudzas tīmekļa un mobilās vairākplatformu izstrādes ietvari nodrošina iespēju pievienot PWA funkcionalitāti izstrādājamās tīmekļa vietnēs.

2.2. Hibrīdlietotnes

Hibrīdlietotnes ir mobilo lietotņu grupa, kas pēc būtības ir tīmekļa un vietējo lietotņu kombinācija. Lietotāja saskarne un programmas kods tiek izstrādāti, izmantojot HTML, CSS un JS, tāpat kā tīmekļa lietotne. Šī saskarne strādā no mobilās ierīces vietēja pārlūkprogrammas konteinera — WebView kontrole uz Android vai WkWebView kontrole uz iOS [5, 10].

Šāda pieeja ļauj piekļūt pie mobilās ierīces funkcionalitātes un vienlaikus atkārtoti izmantot to pašu kodu, kas tiek izmantots pārlūkprogrammas tīmekļa lietotnē. Tā kā šīs lietotnes tiek kompilēti par mobilās operētājsistēmas lietotnes pakotnēm - tos ir iespējams publicēt mobilajā lietotņu veikalā atšķirībā no pārlūkprogrammas lietotnēm [6].

Lai sazinātos ar mobilo ierīci un piekļūtu pie viņu vietējiem ierīces API, kas parasti nav pieejami tīmekļa pārlūkprogrammā, hibrīdlietotnes pielieto tā saucamo abstrakcijas jeb tilta slāni [11]. Šīs tilts ļauj izmantot JavaScript dzinēju, lai caur to izsauktu vietējas ierīces funkcijas, izmantojot JavaScript funkcijas (2.2 att.).



2.2. att. **Hibrīdlietotnes darbības procesa diagramma**

Šis tilta savienojums tiek veidots, izmantojot spraudņus (angl. plugins). Lai izveidotu spraudņu, vispirms tiek uzrakstīta vēlamā funkcionalitāte ierīces vietējā programmēšanas valodā, pēc tam šīs vietējais kods tiek savienots ar kādu JavaScript funkciju, kuru tad varēs izmantot lietotnes tīmekļa daļā [5, 13]. Lielākā daļa izstrādes ietvaru nodrošina visvairāk izmantotās ierīces funkcijas kā spraudņus, bet jaunas, mazāk izmantotas funkcijas vai integrācijas ar trešās puses API var lejupielādēt no spraudņu repozitorija, ko piedāvā izstrādes ietvars.

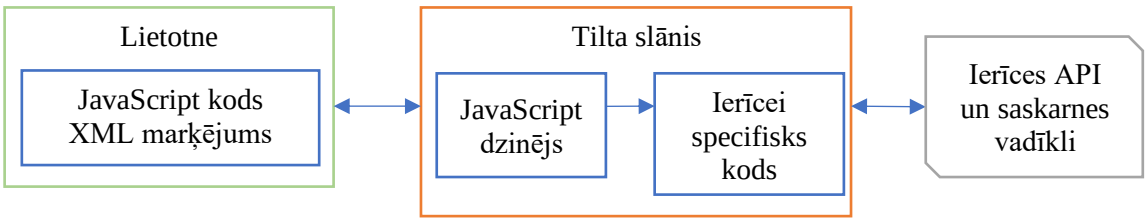
Dažas no pašlaik pieejamajām hibrīdlietotņu izstrādes platformām ir:

- PhoneGap
- Ionic, kas atbalsta tīmekļa izstrādes ietvarus Angular, React
- Quasar izmantojot Vue.js tīmekļa ietvaru
- Onsen UI, kas atbalsta Angular, React un Vue.js

Hibrīdlietotņu izstrādes ietvari piedāvā labu iespēju izstrādāt vairākplatformu mobilās lietotnes, izmantojot pazīstamas tīmekļa tehnoloģijas, un izvairīties no vietēja koda rakstīšanas, pieņemot, ka lietojumprogrammai nav nepieciešama sarežģīta vietējā funkcionalitāte vai lietotāja saskarne.

2.3. Interpretētās lietotnes

Interpretētā mobilā lietotnē programmas kods tiek interpretēts lietotnes darbības laikā, izmantojot JavaScript dzinēju, piemēram, JavaScriptCore vai V8 [10, 14]. Tas pats tilta mehānisms, kas tiek izmantots hibrīdlietotnēs, ir izmantots, lai automātiski uzģenerētu arī lietotāja saskarni lietotnē, izmantojot mobilās ierīces vietējus saskarnes komponentus (2.3. att). Tāpat kā hibrīdlietotnēs, tilta slānis tiek izmantots, lai piekļūtu pie visiem ierīces vietējiem API, savienojot JavaScript dzinēju ar vietēja koda spraudņiem [14, 15].



2.3. att. **Interpretētas lietotnes darbības procesa diagramma**

Ietvari, kas izmanto šādu izstrādes pieeju, bieži reklamē sevi par "vietējiem", kas nav īsti pareizi, jo pašā programmā netiek izmantots platformai specifisks kods. Taču tas arī nav ļoti tālu no patiesības, jo šī pieeja izmanto tikai vietējās mobilās saskarnes elementus un ļauj piekļūt vietējām ierīcēm API caur vietēja koda spraudņiem. Daži izstrādes rīki atļauj arī rakstīt vietējo kodu, izmantojot JavaScript sintaksi, bet šī koda struktūra paliek tāda pati ka ierīcei vietējai programmēšanas valodai [14].

Šī tipa lietotnes parasti tiek rakstītas, izmantojot JavaScript vai tā dialektus lietojumprogrammas loģikai un XML, lai definētu lietotāja saskarnes struktūru mobilās sistēmās [14, 15].

Pašlaik populāri interpretētu lietotņu izstrādes platformas ir:

- React Native izmantojot tīmekļa izstrādes ietvaru React
- Expo izmantojot ietvaru React
- NativeScript, kas atbalsta ietvarus Angular un Vue.js
- Vue Native izmantojot tīmekļa izstrādes ietvaru Vue.js
- Titanium, kas atbalsta ietvarus Angular un Vue.js

Pārlūkprogrammas lietojumprogrammu izveide kopā ar mobilajām lietotnēm, kas izmanto šādu pieeju, ietver sevī XML komponenšu pārvēršanu par HTML un CSS, ko pielieto Expo, vai atsevišķi nodefinēt vietnes komponenta HTML struktūru.

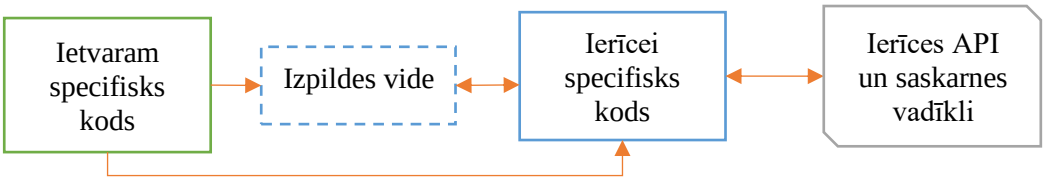
Interpretētā pieeja piedāvā iespēju izstrādāt mobilās lietotnes, kas izskatās un uzvedas tuvi vietēji izstrādātām lietotnēm, bez nepieciešamības vienmēr zināt sistēmas programmēšanas valodu, jo ir pieejami iepriekš uzrakstīti spraudņi.

2.4. Uzģenerētās lietotnes

Uzģenerētās lietotnēs katrai mērķa platformai tiek uzkompilēta platformai raksturīgā lietojumprogrammā, izmantojot vienu un to pašu pirmkodu [10]. Atšķirībā no iepriekšējām pieejām, kas programmu izstrādei izmantoja galvenokārt tīmekļa tehnoloģijas, šī pieejā parasti tiek izmantota kāda cita programmēšanas valoda, kā C#, Dart vai Ruby. Kompilēšanas rezultātā

izveidotā lietotne var saturēt vietējo kodu, vai lietotnē var būt iekļauta platformai raksturīga izpildlaika vide, kuru nodrošina dotais izstrādes ietvars (2.4. att.) [16].

Pašlaik ir ierobežots atbalsts pārlūkprogrammas lietotņu izveidei, izmantojot šo pieeju, taču aktīvi tiek izstrādāti rīki HTML un JavaScript ģenerēšanai [17] vai arī nesen pieejamu WebAssembly koda ģenerēšanai [18].



2.4. att. **Uzģenerētas lietotnes darbības procesa diagramma**

Šāda veida lietojumprogrammas nodrošina augstu veiktspēju, jo, darbinot lietotni mobilajā ierīcē, nav papildu apstrādes slāņa [10], ka arī izmantojot WebAssembly bināro kodu pārlūkprogrammās [18].

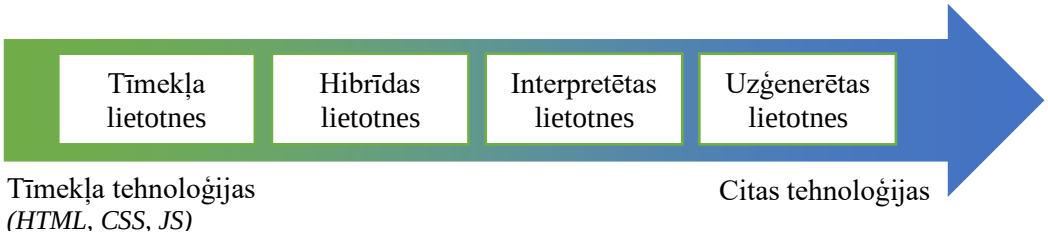
Pastāv vairāki ietvari, kas izmanto šo pieeju tīmekļa un mobilo vairākplatformu izstrādei:

- Flutter izmantojot Dart programmēšanas valodu
- Uno Platform kopā ar Xamarin ietvaru un C# programmēšanas valodu

Esošās tīmekļa vietnes migrēšana uz tīmekļa un mobilo vairākplatformu lietotni ar ietvaru, kas izmanto šo pieeju, var būt problemātiski, jo tiek pielietotas valodas un tehnoloģijas, kas nav izmantotas tīmekļa vidē. Taču pastāv daudzsološas iespējas radīt jaunas, augstas veiktspējas vairākplatformu lietojumprogrammas, izmantojot šī veida izstrādes platformas.

2.5. Kopsavilkums

Vairākplatformu izstrāde tagad attiecās ne tikai izstrādi uz dažādām operētājsistēmām mobilajās ierīces, bet arī uz dažādām vidēm – uz datoriem, tīmekļa pārlūkprogrammām.



2.5. att. **Izstrādes pieeju salīdzinājuma diagramma pēc tīmekļa tehnoloģiju izmantošanas**

Pieejamās tīmekļa un mobilo lietotņu izstrādes pieejas iespējams sakārtot (2.5. att.), sākot no vairāk tīmekļa tehnoloģiju izmantojošās, kas piedāvā lielāku elastīgumu un atpazīstamību

izstrādes procesā, līdz tām, kas izmanto savas programmēšanas valodas un kompilētājus, lai panāktu vislabāko iespējamo veiktspēju.

Dažādie izstrādes ietvari, kas atbalsta tīmekļa un mobilo lietotņu vienlaikus izstrādi, kas tika atrasti šajā nodaļā, ir apkopoti tālāk redzamajā 2.1. tabulā.

2.1. tabula

Vairākplatformu izstrādes rīki un to darbības pieejas

Izstrādes rīks	Izmantotā pieeja lietotnes izstrādē	Atbalstīti tīmekļa izstrādes ietvari	Programmēšanas/ atzīmēšanas valodas
Angular	Tīmekļa lietotne/PWA	-	TS, HTML
React	Tīmekļa lietotne/PWA	-	JS, HTML
Vue.js	Tīmekļa lietotne/PWA	-	JS/TS, HTML
Polymer	Tīmekļa lietotne/PWA	-	JavaScript, HTML
PhoneGap	Hibrīdlietotne	-	JavaScript, HTML
Ionic	Hibrīdlietotne	Angular, React	JS/TS, HTML
Quasar	Hibrīdlietotne	Vue.js	JS/TS, HTML
Onsen UI	Hibrīdlietotne	Angular, React, Vue.js	JS/TS, HTML
React Native	Interpretētā lietotne	React	JS, XML
Expo	Interpretētā lietotne	React	JS, XML
Vue Native	Interpretētā lietotne	Vue.js	JS/TS,
NativeScript	Interpretētā lietotne	Angular, Vue.js	JS/TS, XML/HTML
Titanium	Interpretētā lietotne	Angular, Vue.js	JS/TS, XML
Flutter	Uzģenerētā lietotne	-	Dart
Uno Platform	Uzģenerētā lietotne	-	C#, XAML

Kā ir redzams, pašlaik izstrādātājiem ir pieejama ļoti plaša dažādu izstrādes rīku izvēle, katrs no tām nodrošina dažādu izstrādes preferenču kopumu un katram var būt savas papildu priekšrocības un trūkumi. Svarīgi salīdzināt piedāvātas iespējas, lai izvēlētos sev vispiemērotāko.

3. IZSTRĀDES RĪKU PĀRSKATS UN SALĪDZINĀJUMS

Sadaļā tiks salīdzināti un pielietoti 4 dažādi tīmekļa un mobilās vairākplatformu lietotņu izstrādes tehnoloģijas. Izstrādes ietvari tika paņemti pa vienam no katras kategorijas, kas tika minēti iepriekš 2. sadaļā — tīmekļa lietotne, hibrīdlietotne, interpretēta lietotne un uzģenerēta lietotne.

3.1. Izvēlētas tehnoloģijas

Izstrādes tehnoloģijas tika izvēlētas, balstoties uz viņu jauninājumu un popularitāti izstrādātāju kopienā, kas minēts darba 1. sadaļā.

3.1.1. Progresīva tīmekļa lietotne

Lai tīmekļa vietni varētu veiksmīgi uzskatīt par progresīvo tīmekļa lietotni, tai jāievieš vairākas pamatfunkcijas un jāievēro sekojoši standarti [11, 12]:

- Reaģējamība – vietnei jāpielāgojas jebkuram ekrāna izmēram, jānodrošina pareizi mērogoti saskarnes elementi,
- Uzinstalējamība – vietni ir iespējams “uzinstalēt” un saglabāt to lietotāja tālrunī, nodrošinot saīsni tālruņa palaidējā, tiek nodrošināts ar lietotnes manifestu
- Progresivitāte – vietnei ir jādarbojas jebkurā pārlūkprogrammā uz jebkurās platformas un jebkura pārlūkprogrammas, ieskaitot tajā, kas vēl pilnībā neatbalsta visas PWA iespējas
- Savienojuma neatkarība – tīmekļa lietotne spēj ielādēties un darboties bezsaistē, tiek nodrošināts ar pakalpojumu skriptiem
- Augstā veiktspēja – lietotnei jāielādējas ātri, un lietošanas laikā nedrīkst aizturēties

Lielāka daļa tīmekļa izstrādes ietvaru piedāvā iespējas pie izstrādājamās vietnes pievienot pakalpojumu skriptus un manifestu, lai nodrošinātu PWA pamatiespējas.

3.1.2. Ionic Capacitor

Capacitor[1] ir 2019. gadā izlaists atklātā pirmkoda hibrīdlietotņu dzinējs, kas tiek izmantots vairākplatformu izstrādes ietvarā Ionic. Tā ir pārveidota un uzlabota versija populārā Apache Cordova dzinēja un tiek domāts par tā pēcteci. Galvenās atšķirības salīdzinājumā ar Cordova ir uzlabota veiktspēja, vienkāršots savienojums ar vietējo lietojumprogrammas kodu,

[1] https://ionicframework.com/

iespēja izveidot darbvirsmas lietotnes izmantojot Electron dzinēju, ka arī efektīvāks un vienkāršots izstrādes process [19]. Ionic iespējams izmantot kopā ar citām bibliotēkām un tīmekļa ietvariem - Angular, React, vai Vue.js.

3.1.3. NativeScript

Vispopulārākais no interpretētās pieejas izstrādes rīkiem ir React Native [7, 8]. Tomēr šajā gadījumā tika izvēlēts NativeScript, jo tas pašlaik izraisa izstrādātāju interesi un nav plaši izpētīts citos saistītos darbos.

NativeScript[2] ir atvērtā pirmkoda interpretēts vairākplatformu mobilo lietotņu izstrādes ietvars, ko 2014. gadā izveidoja uzņēmums Telerik/Progress Software. Šīs izstrādes platformas mērķis ir ļaut rakstīt mobilās sistēmas vietējo kodu, neizmantojot tās platformai raksturīgo programmēšanas valodu, bet tikai izmantojot JavaScript vai TypeScript valodu.

Tas piedāvā arī ciešu integrāciju ar Angular un Vue.js izstrādes rīkiem, kas nodrošina iespēju izveidot mobilo lietotni vienlaikus ar tīmekļa lietotni.

3.1.4. Flutter

Flutter[3] ir atvērtā pirmkoda vairākplatformu izstrādes ietvars, ko izveidojusi kompānija Google un pirmo reizi laidusi klajā 2017. gadā.

Tas izmanto uzģenerēšanas pieeju lietotņu izstrādē un ļauj kompilēt vietēju kodu uz sistēmam Android un iOS, kā arī tīmekļa lietotnes (pašlaik beta versijā) no viena un tā paša pirmkoda, kas ir rakstīts, izmantojot Dart programmēšanas valodu. Tā ir vērsta uz augstas veiktspējas lietojumprogrammu nodrošināšanu un ātru renderēšanu uz visām platformām.

[2] https://www.nativescript.org/
[3] https://flutter.dev/

3.2. Izvēlēti salīdzināšanas kritēriji

Riegers un Majčrzaks savā darbā nodrošina plašu kritēriju sistēmu, lai novērtētu vairākplatformu mobilo lietotņu izstrādes tehnoloģijas [21]. Pamatojoties uz šo darbu, tika izveidoti sekojoši kritēriji, pēc kuriem salīdzinātu dažādās pieejas un rīki, kas ir pieejamas tīmekļa un mobilo lietotņu veidošanai:

- Izstrādes vide – apskatīt rīku izstrādes vides, izmantotās programmēšanas valodas;
- Koda koplietošana – izvērtēt iespēju atkārtoti izmantot to pašu kodu, lai izveidotu Android, iOS un tīmekļa programmas vienlaikus un šī koda uzturēšana;
- Piekļuve ierīces funkcionalitātei – salīdzināt izstrādes tehnoloģijas piedāvātas funkcionalitātes strādāt ar ierīces API;
- Atbalstītas vides un pārlūkprogrammas – atbalstītas mobilās platformas un to versijas, atbalstītās pārlūkprogrammas;
- Izplatīšana – lietotnes atklājamība, nopublicēšana lietotņu veikalos.

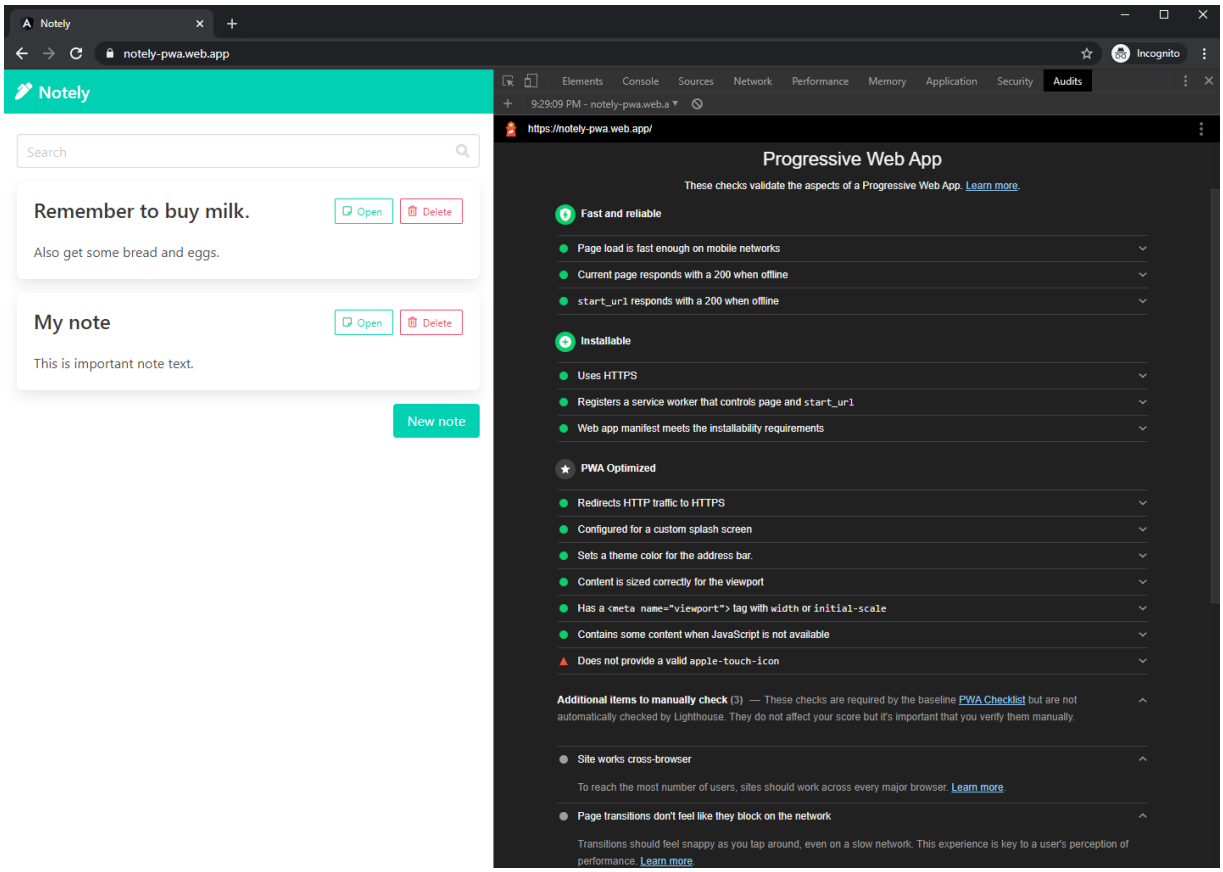
Lai gan šī samazinātā salīdzināšanas kritēriju sistēma nav tik visaptveroša kā sākotnējā, tā spēj pasniegt labu pārskatu par šīm tehnoloģijām un saprast viņu īpatnības, lai palīdzētu identificēt to atšķirīgos izmantošanas gadījumus lietotņu izstrādei.

3.3. Tehnoloģiju salīdzinājums

3.3.1. Izstrādes process

Lai testētu tīmekļa un mobilās izstrādes vidi katram rīkam, tika izveidota vienkārša lietojumprogramma izmantošanai mobilajās ierīcēs un pārlūkprogrammās.

Izmantojot PWA, ir salīdzinoši vienkārši izveidot mobilo lietotni no esošas vienkāršas tīmekļa lietojumprogrammas. Šajā gadījuma tika izmantota Angular ietvars, kas nodrošina bibliotēku `@angular/pwa`, lai automātiski pievienotu pakalpojumu servisus un lietotnes manifestus, kas ir viss, kas ir nepieciešams Android lietotnēm. Papildu konfigurācija ir nepieciešama, lai pievienotu atbalstu arī iOS tīmekļa lietotnēm ar Safari pārlūkprogrammu — pievienojot speciālus metadatus iekš HTML. Chrome Dev Tools nodrošina rīku Lighthouse, lai pārbaudītu, vai tīmekļa lietojumprogramma ir pareizi konfigurēta PWA pamatfunkcionalitātei (3.1. att.). Papildus automātiskai pārbaudei ar šo rīku daži kritēriji ir jāpārbauda arī manuāli, piemēram, pārlūkprogrammas atbalsts un lietotnes veiktspēja.

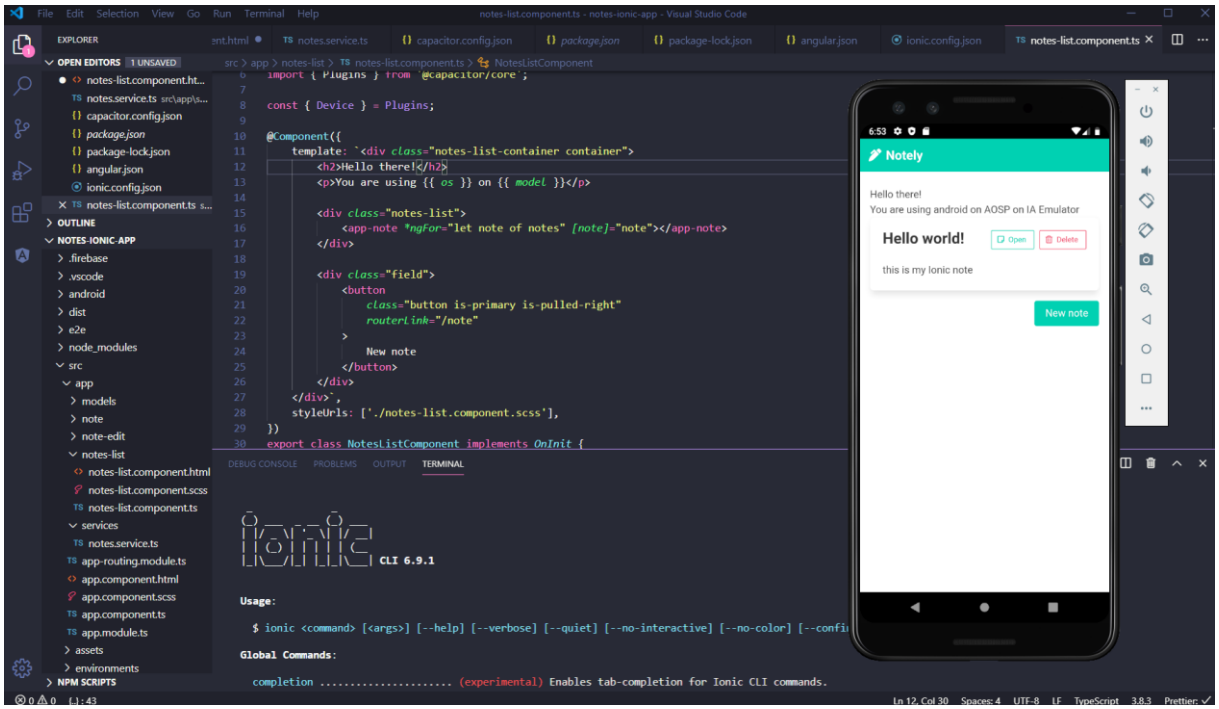


3.1. att. **Ekrānskats no Chrome Lighthouse rīka, kas paredzēts PWA pārbaudei**

Kad šī sākotnējā konfigurācija ir pabeigta, pārējais tīmekļa lietotnes izstrādes process notiek tāpat kā iepriekš, izmantojot HTML, CSS un TypeScript kopā ar Angular ietvaru. Tomēr ir svarīgi regulāri pārbaudīt, vai izstrādājamā lietotne apmierina visus nepieciešamus PWA

kritērijus. Lietotnei ir jāstrādā pareizi bezsaistē un visu veidu pārlūkprogrammās, neatkarīgi no tā, vai šī lietotne ir saglabāta un uzinstalēta, vai nav. Svarīgi ņemt to vērā, turpināšot progresīvas tīmekļa lietotnes izstrādi.

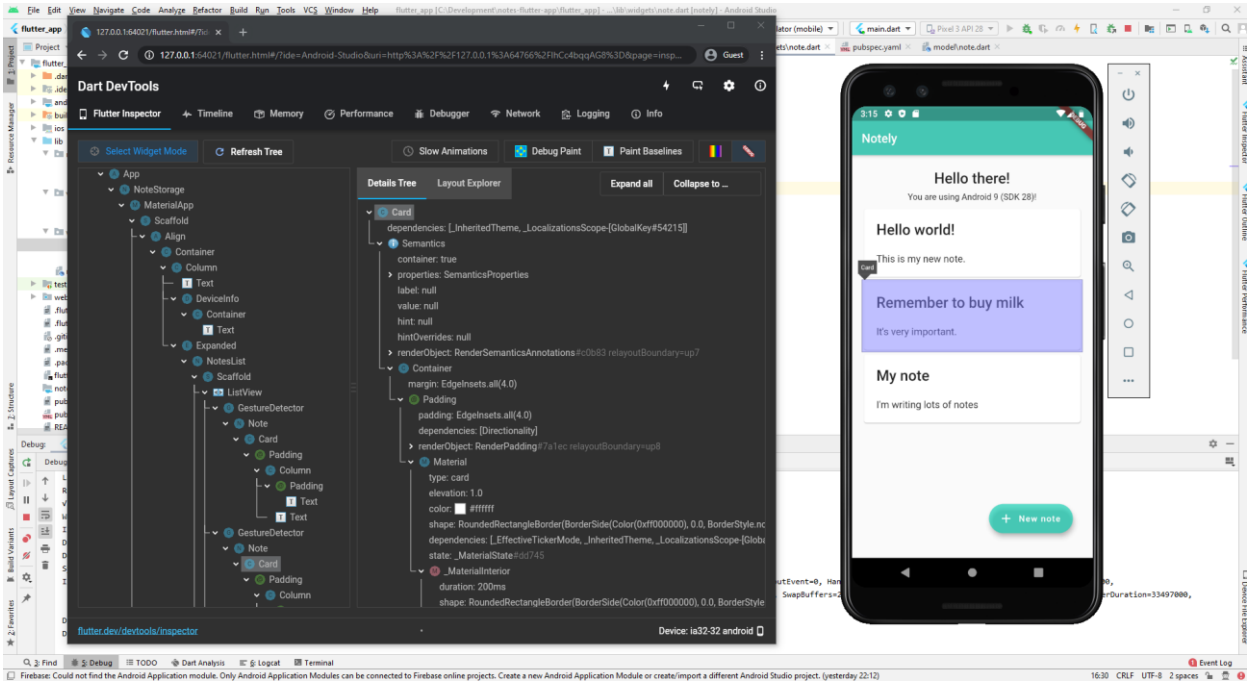
Tīmekļa un mobilo lietotņu izstrāde ar Ionic izstrādes ietvaru notiek, izmantojot teksta redaktoru pēc izvēles un komandrindas rīku (3.2. att.) Lietojumprogrammas projekta struktūra ir tāda pati kā tīmekļa vietnei, kas izmanto Angular ietvaru - sadalīts komponentos un moduļos.



3.2. att. **Piemērs no Ionic izstrādes vides izmantojot Visual Studio Code, Ionic CLI un Android Studio**

Izmantojot Ionic, ir iespējams pārvērst eksistējošo tīmekļa lietotni par mobilo lietotni, kas palaiž šo tīmekļa lietotni ierīces pārlūkprogrammas dzinējā. Lietojumprogrammas izstrādes procesā izstrādātājam nav obligāti nepieciešama piekļuve pie fiziskās ierīces vai emulators, lai testētu tikai lietotāja saskarni. Lietotnes lietotāja saskarni iespējams atveidot ar pārlūkprogrammu izstrādes rīkiem. Tomēr, testējot ierīces iespējas nepieciešams izmantot emulatoru vai fizisko ierīci.

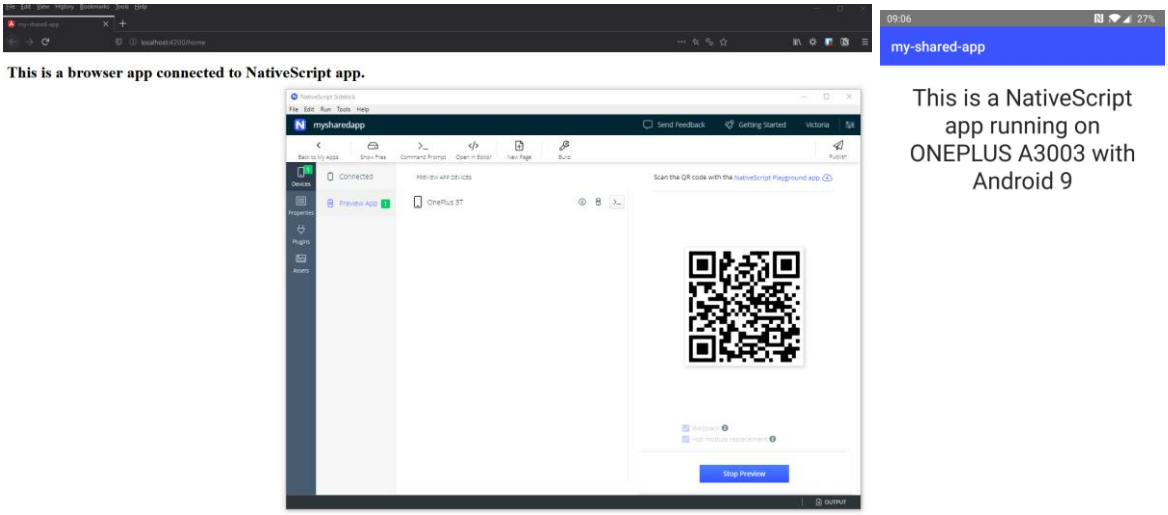
Izmantojot Flutter, nebija iespējams atkalizmantot kodu no iepriekšējiem variantiem, tāpēc no nulles bija jāizveido jauna lietojumprogramma. Ieteicamās izstrādes vides ir Visual Studio Code un CLI vai Android Studio un Flutter spraudni (3.4. att.). Tiek piedāvāts arī Dart DevTools rīks, kas ļauj detalizēti apskatīt katru komponenti un datus uz mobilās ierīces un uz pārlūka. Flutter nodrošina plašu dokumentāciju, apmācības un piemērus, kas palīdz apgūt Dart valodu un rīkus.



3.3. att. **Piemērs no Flutter izstrādes vides izmantojot Android Studio un Dart DevTools**

Gan lietotāja saskarne, gan lietojumprogrammas loģika ir definēta, izmantojot Dart valodu. Valodas struktūra ir ligzdota, un šādā veidā tas var būt salīdzināma ar HTML. Šāda ligzdotā struktūra var apgrūtināt koda izpratni tālāk izstrādes procesā, kas liek izstrādātājiem pareizi sadalīt kodu starp failiem, klasēm un pakotnēm. Tomēr, izstrādes procesā varēja redzēt, ka tīmekļa lietotņu izveides atbalsts šobrīd ir beta versijā, jo, ritinot sarakstu, bija grafiski defekti ar nozudušu tekstu.

NativeScript nodrošina rīku Sidekick, kas palīdz izmantot lietotņu veidnes, kompilēt un darbināt lietojumprogrammu. Tā ļauj arī palaist izstrādes lietojumprogrammas tieši ierīcē bez vajadzības to fiziski savienot ar datoru, izmantojot programmu Playground (3.4 att.).



3.4. att. **Piemērs no NativeScript izstrādes rīka PlayGround**

Nativescript un Angular nodrošina shematikas/skriptus, lai apvienotu tīmekļa un mobilo lietotņu izstrādi vienā projektā. Kods tiek automātiski sadalīts starp mobilajām ierīcēm raksturīgiem un tīmekļa failiem. Izstrādes process turpinās lielākoties atdalīts viens no otra, kas dažos gadījumos var būt vēlamāks, bet arī rada papildu kodu rakstīšanu.

3.3.2. Koda koplietošana

Kopsavilkums par to, kādas iespējas ir pieejamas pirmkoda atkārtotai izmantošanai un uzturēšanai, izmantojot katru tehnoloģiju, ir redzams tabulā zemāk (3.1 tab.), grupēts pēc lietotāja saskarnes koda un lietojumprogrammas funkcionalitātes koda.

3.1. tabula
Tehnoloģiju salīdzinājums pirmkoda koplietošanas ziņā

	PWA	**Ionic**	**NativeScript**	**Flutter**
Lietotāja saskarnes definīcija	Kopējā definīcija	Kopējā definīcija	Definēts atsevišķi	Kopējā definīcija

Izmantojot PWA, tas pats kods tiek izmantots gan tīmekļa lietotnei datora pārlūkiem, gan pārlūkiem uz visām mobilajām ierīcēm. Tomēr lietotāja saskarnes kods ir jāpielāgo darbam gan ar mobilajiem ekrāniem, gan ar datora ekrāniem vienlaikus, parasti plaši izmantojot multivides vaicājumus.

Ionic ietvars izmanto to pašu pieeju UI kodam, jo tas attēlo tīmekļa vietni, kas ietver tādas pašas priekšrocības un riskus kā tīmekļa lietotne. Tas ļauj koplietot lielāku daļu starp vietni un lietotni. Līdzīgu pieeju izmanto arī Flutter ietvars, tomēr HTML/JS/CSS vietā lietotāja saskarne ir rakstīta pilnībā ar Dart valodu [22].

Savukārt izmantojot NativeScript, tīmekļa lietotnēm un mobilajām lietotnēm tiek izmantotas atsevišķas UI definīcijas, jo tiek izmantotas dažādas atzīmēšanas valodas, tomēr stilam joprojām ir kopīgs CSS kods. Koplietot iespējams arī komponenšu loģiku un servisus [23]. Ja lietotnē ir nepieciešams atšķirīgs saskarnes izkārtojums starp datoru un mobilo ierīci, ar vienādu programmas loģiku, tā var būt piemērota pieeja.

Izmantojot Ionic un NativeScript, vienīgie gadījumi, kad lietojumprogrammas loģiskais kods sāk sadalīties starp platformām, ir gadījumi, kad ir nepieciešams sarežģīts vietējs sistēmas kods un nav pieejami esošie spraudņi vēlamajai funkcionalitātei.

3.3.3. Piekļuve ierīces funkcionalitātei

Kopsavilkums par dažiem no ierīces funkcijām, ko atbalsta vai neatbalsta dažādas pārlūkprogrammas un lietotņu izstrādes ietvari, ir redzams 3.2. tabulā.

3.2. tabula

Ierīces API atbalsts dažādas izstrādes vidēs

	Chrome datora pārlūks [24]	Chrome mobilais pārlūks [24]	Ionic mobilā lietotne [25]	NS mobilā lietotne [26]	Flutter mobilā lietotne [27]
Kontakti	-	+	-	+[1]	+[1]
SMS	-	-	+[1]	+[1]	+
Kamera	+	+	+	+	+
Mikrofons	+	+	+[1]	+[1]	+[1]
Bluetooth	+	+	+[1]	+[1]	+[1]
NFC	-	+	+[1]	+[1]	+[1]
Baterija	+	+	+	+[1]	+
Krātuve	+	+	+	+[1]	+
Geolokācija	+	+	+	+	+[1]
Ekrāna orient.	+	+	+	+[1]	+
VR/AR	~	~	-	+[1]	+[1]
Žiroskops	+	+	+[1]	+[1]	+
Akselerometrs	+	+	+	+[1]	+
Paziņojumi	+	+	+	+[1]	+[1]
Pirkstu nospiedums	-	-	+[1]	+[1]	+
Starpliktuve	+	+	+	+[1]	+
Vibrācija	+	+	+	+[1]	+

[1] Pieejams no trešo pušu spraudņu galerijas.

Progresīvas tīmekļa lietotnes piekļuve tālruņa vai datora funkcionalitātei ir pilnībā atkarīga no pārlūkprogrammas atbalsta jaunajiem HTML5 API, kas izveidoti ierīces funkcionalitātes integrēšanai. Šie paši ierobežojumi attiecas arī uz visiem datora pārlūkprogrammas lietotnēm neatkarīgi no izvēlēta izstrādes ietvara – nav iespējams piekļūt kontaktpersonu sarakstam, piemērām, ja datora operētājsistēmai nav tāda API un pārlūkprogramma to nevar atbalstīt. Tāpēc ir svarīgi ņemt vērā saderību starp tīmekļa un mobilās lietojumprogrammām un piedāvāt alternatīvas neatbalstītām funkcijām noteiktā vidē.

Pārējie trīs izstrādes ietvari piedāvā piekļuvi sistēmas vietējam kodam, izmantojot spraudņus, lai piedāvātu ierīces funkcionalitāti. No visām sistēmām Ionic Capacitor nodrošina visvairāk mobilio sistēmu API pamatpakotnē, kas ietver piekļuvi kamerai, failu krātuvei, ģeolokācijai, atļaujām, paziņojumiem, koplietošanas izvēlnei, lietotāja interfeisa logrīkiem. Cita funkcionalitāte, piemēram, autentifikācijas API piekļuve, NFC vai Bluetooth, trešās puses API tiek nodrošināta, izmantojot kopienas izveidotus trešās puses spraudņus.

Flutter un NativeScript nenodrošina tik daudz tālruņa funkciju kā daļa no sava ietvara, bet tā vietā tie paļaujas uz funkcijām, ko izveido izstrādātāju kopiena un koplieto to savās spraudņu un komponentu galerijās. Tā kā šīs sistēmas pašlaik netiek plaši izmantotas, var rasties problēmas ar nepieciešamajiem funkcionalitātes spraudņiem, kas prasa to manuālu rakstīšanu.

Lai manuāli pievienot piekļuvi tālruņa ierīces funkcijām, kas netiek piedāvātas uzreiz, katra tehnoloģija nodrošina atšķirīgu pieeju. NativeScript pieeja tīmekļa izstrādātājiem var būt vispazīstamākā, kā minēts iepriekš, jo tas nodrošina JavaScript ietvarstruktūru vietējā koda rakstīšanai. Izstrādājot funkcionalitāti priekš Android vai iOS, sintakse un struktūra ir ļoti līdzīga vietējās platformas kodam, tiek izmantotas tādas pašas klases, bet rakstīts, izmantojot JavaScript objektus un funkcijas (3.2. att.). Tādā veidā tiek atvieglota pāreja no tikai tīmekļa izstrādes uz mobilo izstrādes procesu, tomēr joprojām jāsaprot valodas bibliotēku īpašības.

```
// Call native Android device APIs with JavaScript!
var date = java.time.LocalDate.now();
var formatter = java.time.format.
    DateTimeFormatter.ofPattern("dd-MMM-yyyy");
var formattedDate = date.format(formatter);

// Call native iOS device APIs with JavaScript!
var formatter = new NSDateFormatter();
formatter.dateStyle = NSDateFormatterMediumStyle;
formatter.timeStyle = NSDateFormatterNoStyle;
formatter.dateFormat = "dd-MM-yyyy";
var date = NSDate.date();
var formattedDate = formatter.stringFromDate(date);
```

```
// ...compared to the same implementation in Java:
Date date = java.time.LocalDate.now()
DateTimeFormatter formatter =
    java.time.format.DateTimeFormatter.ofPattern("dd-MMM-yyyy");
String formattedDate = date.format(formatter);

// ...compared to the same implementation in Objective-C:
NSDateFormatter* formatter = [[NSDateFormatter alloc] init];
formatter.dateStyle = NSDateFormatterMediumStyle;
formatter.timeStyle = NSDateFormatterNoStyle;
formatter.dateFormat = @"dd-MM-yyyy";
NSDate* date = [NSDate date];
NSString* formattedDate = [formatter stringFromDate:date];
```

3.2. att. **Piemērs no NativeScript vietēja koda rakstīšanas iespējām [28]**

Savukārt Ionic un Flutter pieprasa rakstīt vēlamās jaunās funkcijas sistēmas vietējā programmēšanas valodā – Java/Kotlin vai Objective C/Swift, tātad, veidojot mobilo lietotni, iespējams, ka būs nepieciešams, lai izstrādātājs strādātu ar citu, mazāk zināmu valodu. NativeScript savukāri nepiedāvā nekādas iespējas izstrādāt programmas funkcijas ar vietējo programmēšanas valodu, tikai ar JavaScript.

3.3.4. Atbalstītas vides un pārlūkprogrammas

Tālāk redzamajās tabulās ir redzama informācija par katras tehnoloģijas atbalstītajām platformām (3.3. tab.) un tīmekļa pārlūkprogrammām (3.4. tab.).

3.3. tabula
Mobilo operētājsistēmu atbalsts [28, 29]

	PWA	Ionic	NativeScript	Flutter
Android	+[1]	+ *(ver. 5.0+)*	+ *(ver. 4.2+)*	+ *(ver. 4.1+)*
iOS	+[1]	+ *(ver. 11.0+)*	+ *(ver. 9.0+)*	+ *(ver. 8+)*

Flutter atbalsta mobilo operētājsistēmu visvecākās versijas, tāpēc šo ietvaru var izmantot, ja nepieciešams vecāku mobilo ierīču atbalsts. PWA iespējams izmantot neatkarīgi no sistēmas, taču, strādājot ar vecākām operētājsistēmām, jāņem vērā saderība ar vecākām pārlūkprogrammām.

3.4. tabula
Parlūkprogrammu atbalsts [28, 29]

	PWA	Ionic	NativeScript	Flutter
Datora pārlūkprogrammas				
Chrome	+	+	+[1]	+
Safari	+	+	+[1]	+
Edge	+	+	+[1]	+
Firefox	+	+	+[1]	+
IE11	~	-	+[1]	-
Mobilās pārlūkprogrammas				
Chrome (Android)	+	+	+[1]	+
Safari (iOS)	+	+	+[1]	+
Firefox (Android)	+	+	+[1]	+

[1]Izmantojot kopā ar tīmekļa vietni, uzbūvētu ar Angular rīku

Kā var redzēt, visas sistēmas nodrošina labu atbalstu visām modernajām pārlūkprogrammām. Ionic un Flutter ietvari oficiāli neatbalsta novecojušu pārlūku IE11. Ja šāds

atbalsts tomēr ir nepieciešams, iespējams izveidot saderīgu tīmekļa pārlūka lietotni pašiem ar Angular, kas atbalsta IE11, un pēc tam tikai pievienot mobilo lietotņu atbalstu, izmantojot PWA vai NativeScript.

3.3.5. Lietotnes izplatīšana

Lietotņu izplatīšana ir kritērijs, kas jāņem vērā tādēļ, ka PWA darbojas atšķirīgi no pārējām trim izstrādes tehnoloģijām. Ionic, NativeScript un Flutter izveido mobilās lietojumprogrammas, kas paredzēti publicēšanai uz Google Play Store vai Apple App Store. Būtiski, ka lietotņu veikalos lietotāji var atklāt jaunas noderīgas lietojumprogrammas, pārlūkojot kategorijas vai apskatīšot ieteikumus no veikala.

Tas nav viegli iespējams ar progresīvo tīmekļa lietotni. Tā kā PWA ir tikai tīmekļa vietne, tad tā atklājamība ir atkarīga no lietotāja pārlūkošanas ieradumiem, jo internetā nav pieejams pilnīgs progresīvo lietojumprogrammu katalogs. Pēdējā laikā ir kļuvis iespējams pievienot arī progresīvas tīmekļa lietotnes lietotņu veikalos, ja pie lietotnes ir pievienots Trusted Web Activity iesaiņojums. Tomēr lietotņu veikali nelabprāt apstiprina šī tipa lietotnes, uztraucoties par to kvalitāti un potenciālo skaitu [30].

Tomēr, ja izstrādātais projekts jau ir plaši zināma un pielietota tīmekļa lietojumprogramma, piemēram, populārs ziņu kanāls vai programmatūras risinājums, tad atklājamība var nebūt par būtisku problēmu.

REZULTĀTI

Šobrīd statistika liecina, ka mobilo ierīču un galddatoru izmantošana tiešsaistē pašlaik ir nosvērta, un mobilo ierīču izaugsme ir palēninājusies. Iepriekš veiktie pētījumi par lietotāju lietošanas paradumiem parādīja, ka daži uzdevumi ir ieteicamāki datoru pārlūka vidē, citi — mobilajās ierīcēs un pāreja starp vidēm uzdevuma izpildes vidū ir bieži sastopama. Šie fakti parādīja, ka ir svarīgi vienlaikus izstrādāt risinājumus vairākām vidēm.

Darbā tika aprakstītas vairākas vairākplatformu izstrādes pieejas un rīki, kas tos izmanto, koncentrējoties uz rīkiem, kas atbalsta vienlaicīgu tīmekļa un mobilo lietotņu izstrādi. Izstrādes pieejas bija atkarīgas no izmantotajām pamattehnoloģijām, sākot no vairāk tīmekļa tehnoloģiju balstītām līdz mazāk.

- Progresīvas tīmekļa lietotnes ir atkarīgas no pārlūkprogrammām, lai nodrošinātu vietējam mobilai lietotnei līdzīgu pieredzi. Tas piedāvā tīmekļa vietnes ar uzlabotām funkcijām, kas nodrošina labāku lietotāja pieredzi un piekļuvi pie ierīces funkcijām.
- Hibrīdā pieeja atveido tīmekļa lapu mobilajā lietotnē, izmantojot pārlūkprogrammas vadīklas, un izmanto JavaScript tiltu, lai piekļūtu mobilās ierīces funkcijām. To pašu tīmekļa vietni var izmantot arī pārlūkprogrammā.
- Interpretētas lietotnes izmanto JavaScript tiltu, lai atveidotu mobilo saskarni ar ierīces vietējām saskarnes vadīklām un piekļūtu pie ierīces API. Lietotāja saskarne mobilajās lietotnēs tiek definēta citādāk nekā tīmekļa lietotnēs, izmantojot šo pieeju.
- Uzģenerēta pieeja izveido tīmekļa un mobilās lietotnes, izmantojot vienu un to pašu kodu, rakstīts ar citām programmēšanas valodām, kas nav balstītas uz tīmekļa tehnoloģijām. Atbalsts tīmekļa vietņu izstrādei, izmantojot šo pieeju, sākas tikai nesen un ir eksperimentāls.

Vairāki tīmekļa un mobilo lietotņu izstrādes rīki tika salīdzināti pēc atbalstītiem ierīci API, koda koplietošanas potenciālu, pārlūkprogrammu un mobilo ierīču atbalstu, publicēšanas iespējām, veiktspējas.

- Visi rīki nodrošina labu atbalstu visās modernajās pārlūkprogrammās un mobilajās operētājsistēmās.
- Progresīvās tīmekļa lietotnes pašlaik plaši atbalsta datoru un mobilo ierīču pārlūkprogrammas, tas darbojas pietiekami ātri, ja vietne nav pārāk sarežģīta, un ļauj atbalstīt visu veidu ierīces, nestrādājot ar vietējo operētājsistēmas kodu. Parasti tās nevar publicēt lietotņu veikalos un nodrošināt atšķirīgu, mazāk vēlamo lietotāja pieredzi mobilajās ierīcēs.

- Ionic Capacitor ļauj uzlabot tīmekļa vietnes ar piekļuvi pie ierīču API, bet no kurām dažas vēl nav pieejamas ar jaunu Capacitor dzinēju. Tas piedāvā iespēju ātri pārveidot tīmekļa vietni par mobilo lietotni ar piekļuvi pie ierīces funkcionalitātes.
- NativeScript ir atkarīgs no kopienas ieguldījuma ierīču atbalstam. Tajā ir pieejami rīki, kas ļauj savienot to ar Angular vai Vue.js tīmekļa lietojumprogrammām un koplietot viņu lietotnes loģikas kodu. Tas ļauj atsevišķi un neatkarīgi definēt mobilo lietotāja saskarni un tīmekļa vietnes saskarni.
- Flutter ir ģenerētu lietotņu izstrādes rīks, kas piedāvā iespēju izstrādāt kodu, izmantojot tikai vienu programmēšanas valodu, nevis vairākas tehnoloģijas. Pašlaik, kompilējot kodu tīmekļa lietojumprogrammās, rodas dažas problēmas.

Darbs galvenokārt balstījās uz dažādu literatūras un tiešsaistes avotiem, kas ir pietiekami, lai sniegtu vispārēju informatīvu pārskatu par tēmu. Tālāk ir jāveic plašāki eksperimenti ar šiem rīkiem, lai iegūtu konkrētākus mērījumus par rīku veiktspēju, uzturētspēju, un citiem radītajiem.

Vairāki citi aspekti no izstrādes procesa netika ņemti vērā šajā darbā, piemēram, reaģējamība vai drošība, un tāpēc ir nepieciešama tālāka izpēte, lai ciešāk savienotu tīmekļa un mobilo izstrādes procesu. Salīdzināšanas praktiskās daļas nākotnē būtu jāpārbauda arī uz iOS operētājsistēmas jo viņi nebija pārbaudīti ierīces nepieejamības dēļ. Darbā arī netika iekļauti daudzi citi dažādi rīki, kas pieejami izstrādājot tīmekļa lietotnes ar ļoti populārām React ietvaram un Vue.js ietvaram, kurus ir vērts apskatīt.

SECINĀJUMI

Iespējams noteikt vairākus secinājumus par pieejamiem tīmekļa un mobilo lietotņu risinājumiem vienlaikus izstrādei:

1. Progresīvas tīmekļa lietotnes var izmantot, lai nodrošinātu labāku lietotāja pieredzi un papildu funkcijas uz mobilajām ierīcēm, izstrādājot tikai tīmekļa vietni. Tas ir piemērots resursu maztērējošu tīmekļa lietotņu izstrādei, kurām nav nepieciešams daudz iespēju, ar minimālām izmaiņām tīmekļa izstrādes procesā.
2. Hibrīdu un interpretētu lietotņu izstrādes rīkus, piemēram, Ionic un NativeScript, var izmantot, lai izveidotu tādas lietotnes, kas izmanto vairāk no tālruņa funkcionalitātes, bet vienlaikus izmantojot tīmekļa tehnoloģijas.
3. Uzģenerētu lietotņu izstrāde izmantojot, piemēram, Flutter, piedāvā labas iespējas izstrādes procesā, bet nepieciešamas zināšanas jaunajās programmēšanas valodās, kuras parasti netiek izmantotas tīmekļa izstrādei un veidotās tīmekļa lietojumprogrammas pašlaik nav gatavas lietošanai dzīvē.
4. Neatkarīgi no izstrādes pieejas, pārlūkprogrammām pašlaik ir ierobežota piekļuve pie ierīces funkcijām, kas ir jāņem vērā, izstrādājot tīmekļa un mobilās lietotnes. Tomēr pārlūkprogrammas strauji attīstās, lai nākotnē iespējotu vairāk līdzekļu.

Kopumā pastāv liels potenciāls ātri un efektīvi izstrādāt jaunas lietojumprogrammas, kas strādā gan uz pārlūkprogrammām, gan uz mobilajām ierīcēm, un kuri sasniedz pēc iespējas vairāk lietotāju. Pārlūkprogrammas un vairākplatformu rīki ļoti strauji attīstās, un šajā jomā ir jāveic plašāki pētījumi, lai analizētu visās iespējas.

IZMANTOTĀ LITERATŪRA UN AVOTI

[1] "Desktop vs Mobile vs Tablet Market Share Europe", *StatCounter Global Stats*. [Tiešsaistē]. https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/europe#monthly-201501-202005 [Piekļūts: 22. aprīlis, 2020.]

[2] "Desktop vs Mobile vs Tablet Market Share Latvia", *StatCounter Global Stats*. [Tiešsaistē]. https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/latvia#monthly-201501-202005 [Piekļūts: 22. aprīlis, 2020.]

[3] E. Enge, "Mobile vs. Desktop Usage in 2019", *Perficient*, 11. apr., 2019. [Tiešsaistē]. https://www.perficient.com/insights/research-hub/mobile-vs-desktop-usage-study [Piekļūts: 25. aprīlis, 2020.]

[4] "The 2017 U.S. Mobile App Report," *Comscore, Inc.* 24. aug., 2017. [Tiešsaistē]. Pieejams: https://www.comscore.com/Insights/Presentations-and-Whitepapers/2017/The-2017-US-Mobile-App-Report. [Piekļūts: 20. maijs, 2020.]

[5] N. Gok un N. Khanna, *Building Hybrid Android Apps with Java and JavaScript: Applying Native Device APIs*. O'Reilly Media, Inc., 2013.

[6] K. A. More un M. Chandran, "Native Vs Hybrid Apps," jūn., 2016. Pieejams: https://www.semanticscholar.org/paper/Native-Vs-Hybrid-Apps-More-Chandran/99a6e137fe40a7b02fdaa54e32c619dac949f76f.

[7] "The State of JavaScript 2019: Mobile & Desktop". [Tiešsaistē]. Pieejams: https://2019.stateofjs.com/mobile-desktop/ [Piekļūts: 12. maijs 2020.]

[8] "Search Interest for React Native, Flutter, Ionic, Progressive Web Apps, NativeScript", *Google Trends*. [Tiešsaistē]. https://trends.google.com/trends/explore?cat=32&date=today%2012-m,today%2012-m,today%2012-m,today%2012-m,today%2012-m&geo=,,,,&q=React%20Native,%2Fg%2F11f03_rzbg,%2Fg%2F1q6l_n0n0,%2Fg%2F11bzxympx6,%2Fg%2F11c57wr2yt [Piekļūts: 5. maijs 2020.]

[9] W. S. El-Kassas, B. A. Abdullah, A. H. Yousef, un A. M. Wahba, "Taxonomy of Cross-Platform Mobile Applications Development Approaches," *Ain Shams Engineering Journal*, vol. 8, no. 2, pp. 163–190, Jun. 2017, doi: 10.1016/j.asej.2015.08.004.

[10] S. Xanthopoulos un S. Xinogalos, "A comparative analysis of cross-platform development approaches for mobile applications," *Proceedings of the 6th Balkan Conference*

in Informatics on - BCI '13, Thessaloniki, Greece, 2013, lpp. 213, doi: 10.1145/2490257.2490292.

[11] D. A. Hume, *Progressive Web Apps*. Manning Publications, 2017.

[12] S. Richard un P. LePage, "What makes a good Progressive Web App?", *web.dev*, febr. 24, 2020. https://web.dev/pwa-checklist/ [Piekļūts: 10. maijs 2020.]

[13] "Capacitor Plugins", *Ionic Framework*, [Tiešsaistē]. Pieejams: https://capacitor.ionicframework.com/docs/plugins [Piekļūts: 12. maijs 2020.]

[14] "Technical Overview", NativeScript Docs, [Tiešsaistē]. Pieejams: https://docs.nativescript.org/core-concepts/technical-overview [Piekļūts: 12. maijs 2020.]

[15] B. Eisenman, *Learning React Native: building native mobile apps with JavaScript*. Beijing ; Boston: O'Reilly, 2018.

[16] "What is Xamarin?", Microsoft Docs, [Tiešsaistē]. Pieejams: https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin [Piekļūts: 14. maijs 2020.]

[17] "Platforms", Dart [Tiešsaistē]. Pieejams: https://dart.dev/platforms [Piekļūts: 14. maijs 2020.]

[18] "How It Works", Uno Platform [Tiešsaistē]. Pieejams: https://platform.uno/how-it-works/ [Piekļūts: 14. maijs 2020.]

[19] M. Lynch, "Cordova vs Capacitor", Ionic Framework, [Tiešsaistē]. Pieejams: https://ionicframework.com/resources/articles/capacitor-vs-cordova-modern-hybrid-app-development [Piekļūts: 18. maijs 2020.]

[20] "Flutter FAQ", Flutter.dev, [Tiešsaistē]. Pieejams: https://flutter.dev/docs/resources/faq [Piekļūts: 18. maijs 2020.]

[21] C. Rieger un T. A. Majchrzak, "Towards the definitive evaluation framework for cross-platform app development approaches", *Journal of Systems and Software*, sēj. 153, lpp. 175–199, jūl. 2019, doi: 10.1016/j.jss.2019.04.001.

[22] "Flutter for Web Developers", Flutter.dev, [Tiešsaistē]. Pieejams: https://flutter.dev/docs/get-started/flutter-for/web-devs [Piekļūts: 19. maijs 2020.]

[23] "Code Sharing Between Web and Mobile with Angular and NativeScript", NativeScript Team, [Tiešsaistē]. Pieejams: https://www.nativescript.org/blog/code-sharing-between-web-and-mobile-with-angular-and-nativescript [Piekļūts: 20. maijs 2020.]

[24] “What Web Can Do Today” [Tiešsaistē]. Pieejams: https://whatwebcando.today/ [Piekļūts: 20. maijs 2020.]

[25] “Capacitor Community Plugins”, Ionic Framework,[Tiešsaistē]. Pieejams: https://capacitor.ionicframework.com/docs/community/plugins [Piekļūts: 20. maijs 2020.]

[26] “NativeScript Market” [Tiešsaistē]. Pieejams: https://market.nativescript.org/ [Piekļūts: 20. maijs 2020.]

[27] “Flutter Plugins”, GitHub [Tiešsaistē]. Pieejams: https://github.com/flutter/plugins [Piekļūts: 22. maijs 2020.]

[28] “Access Native Device APIs with NativeScript” , NativeScript Team [Tieššaistē]. Pieejams: https://www.nativescript.org/native-api-access [Piekļūts: 22. maijs 2020.]

[29] “Service Workers”, Can I Use [Tiešsaistē]. Pieejams: https://caniuse.com/#feat=serviceworkers [Piekļūts: 21. maijs 2020.]

[30] “Browser Support”, Ionic Framework [Tiešsaistē]. Pieejams: https://ionicframework.com/docs/reference/browser-support [Piekļūts: 22. maijs 2020.]

[31] L. Vu, “Publishing PWAs to Major App Stores: The Whys and Hows”, *SimiCart*, 13. feb., 2020. [Tiešsaistē]. https://www.simicart.com/blog/pwa-app-stores/ [Piekļūts: 14. maijs 2020.]

PIELIKUMI

1. pielikums.

Kods no Ionic Capacitor lietotnes, kas parāda piezīmes un informāciju par ierīci.

```
import { Component, OnInit } from '@angular/core';

import { Plugins } from '@capacitor/core';

const { Device } = Plugins;

@Component({
    template: `<div class="notes-list-container container">
        <h2>Hello there!</h2>
        <p>You are using {{ os }} on {{ model }}</p>

        <div class="notes-list">
            <app-note *ngFor="let note of notes" [note]="note"></app-note>
        </div>

        <div class="field">
            <button
                class="button is-primary is-pulled-right"
                routerLink="/note">
                New note
            </button>
        </div>
    </div>`,
    styleUrls: ['./notes-list.component.scss'],
})
export class NotesListComponent implements OnInit {
    notes: Note[];
    model: string;
    os: string;

    constructor(private notesService: NotesService) {}

    ngOnInit(): void {
        Device.getInfo().then((res) => {
            this.model = res['model'];
            this.os = res['operatingSystem'];
        });

        this.notesService.getAll().subscribe((res) => {
            this.notes = res;
        });
    }
}
```

2. pielikums

Kods no Flutter lietotnes, kas parāda piezīmes un informāciju par ierīci.

```
class App extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return NoteStorage(MaterialApp(
        ...
          body: Align(
            alignment: Alignment.topCenter,
            child: Container(
                padding: const EdgeInsets.fromLTRB(16, 24, 16, 24),
                constraints: BoxConstraints(maxWidth: 800),
                child: Column(
                    children: [
                  Text("Hello there!",
                      style:
                       TextStyle(fontSize:24,fontWeight:FontWeight.w600)),
                  DeviceInfo(),
                  Expanded(child: NotesList()),
                ])), ),
        )));}
class _DeviceInfoState extends State<DeviceInfo> {
  Future<Null> initPlatformState() async {
      String deviceVersion;
      if (Platform.isAndroid) {
        var device = await deviceInfoPlugin.androidInfo;
        deviceVersion    =    'Android    ${device.version.release}    (SDK
${device.version.sdkInt})';
      }
      else if (kIsWeb) {
        deviceVersion = 'a web browser';
      }

    setState(() {
      _deviceVersion = deviceVersion;
    });
```

```
  }

  @override
  Widget build(BuildContext context) {
    return new Container(
      padding: const EdgeInsets.all(8),
      child: Text('You are using $_deviceVersion!')
    );
  }
}
```

Bakalaura darbs „Mūsdienu iespējas tīmekļa un mobilo vairākplatformu lietotņu izstrādē” 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.

Autore: Viktorija Savicka <paraksts>

Rekomendēju/nerekomendēju darbu aizstāvēšanai *(nederīgo svītro vadītājs)*

Vadītāja: Asoc. prof. Dr. dat. Darja Solodovņikova <paraksts> 25.06.2020.

Recenzents: Prof. Dr. dat. Uldis Straujums

Darbs iesniegts Datorikas fakultātē 25.06.2020.

Dekāna pilnvarotā persona: vecākā metodiķe Ārija Sproģe _____

Darbs aizstāvēts bakalaura gala pārbaudījuma komisijas sēdē

____.06.2020. prot. Nr. _____

Komisijas sekretārs(-e): _____