VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ BAKALAURO STUDIJŲ PROGRAMA

Daugiaplatformės mobilios programėlės kūrimo analizė panaudojant NativeScript

Cross-platform mobile application development analysis using NativeScript

Bakalauro baigiamasis darbas

Atliko: Darius Pažusinskis (parašas)

Darbo vadovas: lekt. Jonas Ragaišis (parašas)

Darbo recenzentas: doc. dr. Kristina Lapin (parašas)

Vilnius – 2021

Santrauka

Šiame darbe yra analizuojamas daugiaplatformėms mobilioms programoms kurti skirtas NativeScript įrankis. Baigiamajame darbe apžvelgiami daugiaplatformių programų sistemų kūrimo būdai, plačiau aprašomos NativeScript įrankio siūlomos funkcijos ir galimybės. Šio darbo tikslas yra pasiūlyti gaires, kurių pagalba būtų galima įvertinti, kaip naudojantis NativeScript įrankiu jos gali būti įgyvendintos kuriant mobiliąją programėlę. Šiam tikslui įgyvendinti buvo iškelti konkretūs uždaviniai: atlikus tyrimą, nustatyti mobiliųjų programėlių kūrimo metu kylančius iššūkius bei sudaryti gairių sąrašą daugiaplatformei mobiliajai aplikacijai sukurti.

Raktiniai žodžiai: NativeScript, Angular, karkasas, daugiaplatformės mobilios aplikacijos, mobilios aplikacijos kūrimo analizė, Firebase, Android, iOS, WEB.

Summary

This paper analyzes cross-platform mobile application development using NativeScript framework. During the work, the author examines different ways of developing multi-platform application systems and reviews features and capabilities offered by NativeScript in more detailed way. The goal of this thesis is to conduct a list of guidelines for evaluating NativeScript framework during their implementation. To achieve this goal, specific tasks, such as to identify challenges during mobile application development and to create list of guidelines for cross-platform applications development, were set.

Keywords: NativeScript, Angular, framework, cross-platform mobile application, mobile application development analysis, Firebase, Android, iOS, WEB.

Turinys

Įvadas

Tyrimo objektas ir temos aktualumas. Mobilieji telefonai tapo būtinybe daugeliui žmonių visame pasaulyje, mobiliųjų programėlių svarba taip pat tampa didžiulė. Šiai dienai didžiąją rinkos dalį užima iOS ir Android operacines sistemas turintys telefonai. Pagal statcounter[1] portalo duomenis, 2021 metų balandžio mėnesį iOS užėmė 27%, o Android – 72.2% (žiūr. 1 priedą) rinkos. Tai yra net apie 99.2% visos rinkos dalies. Norint pasiekti didžiąją dalį vartotojų, labai svarbu mobilias programėles pritaikyti abejoms minėtoms platformoms. Tradicinis būdas kurti mobiliąsias programėles yra kiekvienai platformai kurti atskirą kodo bazę, kurios reikalauja skirtingų programavimo kalbų žinių. Norint sukurti programėlę Android platformai, programą reiktų rašyti „Kotlin“ kalba, arba „Swift“ - norint sukurti aplikaciją iOS platformai. Toks būdas kurti mobilias programėles yra gana aiškus ir geriausiai optimizuotas atitinkamai operacinei sistemai, tačiau norint apimti abi platformos, programą tektų kurti du kartus [DTC19]. Tai yra viena iš priežasčių, kodėl buvo pradėtos kurti daugiaplatformės mobiliosios programėlės. Vienas iš tokio tipo programėlių kūrimo būdų yra naudojant „NativeScript“ karkasą, kuris siūlo mobiliųjų programėlių kūrimą suteikiant pilną prieigą prie iOS arba Android platformų teikiamų galimybių naudojant „JavaScript“ programavimo kalbą. Tai reiškia, kad viena kodo bazė, parašyta pasirinkta technologija gali būti sukompiliuota tiek į vieną iš minėtų platformų, tiek veikti kaip internetinis puslapis. Toks mobiliųjų programėlių kūrimo būdas kartais yra greitesnis ir pigesnis [DTC19]. Šiame darbe analizuojama mobiliųjų programėlių architektūra, išryškinamos daugiaplatformių programėlių kūrimo gairės bei pagal jas sukuriami reikalavimai tokio tipo sistemoms kurti. Pagal sudarytus reikalavimus buvo tiriama, ar naudojantis NativeScript įrankiu jie gali būti įvykdyti.

Problema. Daugiaplatformių mobiliųjų programėlių kūrimo analize siekiama pateikti lengvesnį, taupantį laiką bei resursus kelią. Toks būdas kurti mobiliąsias programėles leidžia turėti vieną, keliose platformose veikiančią kodo bazę, kurioje daug to paties kodo gali būti pernaudota. Tačiau, sistemos projektavimas yra vienas iš kritinių etapų, norint sukurti daugiaplatformę aplikaciją. Siekiant susiaurinti probleminę sritį, darbo tikslui pasiekti buvo nuspręsta pasirinkti NativeScript įrankį.

Buvo ieškota su NativeScript technologija susijusių mokslinių darbų, tačiau paieškos rezultatų buvo nedaug. Viename iš rastų darbų, kuris buvo parašytas 2017 metais, darbo autorius analizavo daugiaplatformės mobilios programėlės kūrimą lyginant NativeScript ir React Native karkasus. Darbo metu autorius labiau orientavosi į karkasų palyginimą naudojant

[1] https://gs.statcounter.com/os-market-share/mobile/worldwide

metrikas, tačiau nebuvo aptariama architektūrinė dalis bei tam tikrų mobilios programos sluoksnių įgyvendinimo ypatybės [Ves17]. Kitas su NativeScript susijęs darbas aprašė mobilios programos įgyvendinimą panaudojant minėtąjį karkasą. Darbo metu buvo aprašomas planuojamos sukurti programų sistemos dizainas, struktūra ir duomenų modeliai, tačiau apie naudojamo įrankio architektūrinį modelį ir mobilios programos kūrimo gaires taip pat nebuvo paminėta [Šmi17]. Daugiau mokslinių darbų ar tyrimų, atliktų su šiuo įrankiu nerasta. Atlikus literatūrinę analizę pastebėta, jog NativeScript karkasas nėra analizuotas architektūriniu lygiu ir nėra aptartos jo galimybės.

Darbo tikslas: Pasiūlyti gaires, kurių pagalba būtų galima įvertinti, kaip naudojantis NativeScript įrankiu jos gali būti įgyvendintos kuriant mobiliąją programėlę Android ir iOS.

Darbo uždaviniai

- Išanalizuoti teorinę medžiagą siekiant apibūdinti daugiaplatformių programėlių kūrimo būdus;
- Apžvelgti NativeScript įrankį, jo sandarą, siūlomus funkcionalumus;
- Turinio analizės tyrimo pagalba išanalizuoti ir įvardinti svarbiausius kriterijus kuriant mobiliąją programėlę;
- Pagal sudarytus reikalavimus daugiaplatformei mobiliajai programėlei kurti atlikti eksperimentinį tyrimą, kurio metu, naudojantis NativeScript įrankiu, reikalavimai bus bandomi įgyvendinti praktiškai.

Motyvacija. Šią temą autorius pasirinko todėl, kad šiuo metu dar nėra daug mokslinių straipsnių, tiriančių NativeScript karkasą, tad, autoriaus nuomone, ši technologija turėtų sulaukti daugiau dėmesio dėl jos siūlomų technologinių galimybių pilnai pasiekti mobiliųjų platformų programavimo sąsają, galimybę turėti vieną kodo bazę WEB, Android ir iOS platformoms bei dėl turimos galimybės atskirti kodo dalis pagal platformą. Šiuo darbu siekiama prisidėti prie NativeScript populiarinimo parodant jo galimybes bei pasiūlyti gaires mobilios programėlės kūrimui, kurios padės įvertinti įrankio tinkamumą planuojamam projektui.

Laukiami rezultatai

1) Apžvelgtas NativeScript įrankis, siūlomi funkcionalumai, architektūra, galimybės;
2) Atliktas tyrimas, kurio metu sudarytos gairės mobiliųjų programų sistemai kurti buvo taikomos pasinaudojant NativeScript įrankiu.

1. Mobiliosios programėlės kūrimo būdų ir NativeScript įrankio apžvalga

1.1. Mobiliosios programėlės kūrimo būdai

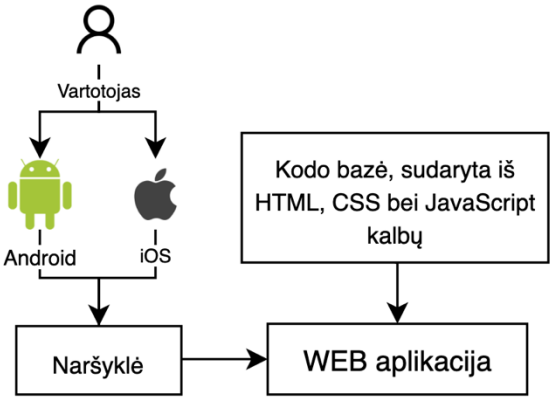
Mobilioji programėlė yra programų sistema, kuri yra pritaikyta veikti daugelyje mobiliųjų įrenginių: mobilieji telefonai (iOS, Android ir kt.), planšetinis kompiuteris ar kitoks nešiojamas įrenginys [IM10]. Jos gali būti atsisiųstos per skirtingas programų platinimo platformas, priklausomai nuo telefono gamintojo ir operacinės sistemos („Apple Store“, „Google Play“ ar kt.). Paprastai, mobiliosios programėlės yra skirtos tam, kad patenkintų vartotojų kasdienius poreikius greičiau ir paprasčiau [IM10].

Egzistuoja keli skirtingi būdai suprogramuoti daugiaplatformę mobiliąją programėlę. Nors galutinis rezultatas atrodo daugiau mažiau panašus, tačiau mobilios programų sistemos tarpusavyje skiriasi kūrimo metodu. Tolimesniuose skyriuose autorius apžvelgs skirtingus mobiliųjų programų įgyvendinimo būdus išskirstant juos į dvi kategorijas: naršyklės pagrindu kuriamos programėlės ir kompiliavimo pagrindu kuriamos programėlės. Vietinė programėlė nebus aptariama, kadangi, kaip buvo minėta šio darbo įvade, ji veikia tik konkrečioje platformoje.

1.1.1. Naršyklės pagrindu kuriamos programėlės

Šiame skyriuje aptariami žiniatinklio technologijų pagalba kuriamų mobiliųjų programėlių tipai. Pirmasis yra itin panašus į tradicinę internetinę svetainę, o antrasis – papildomų technologijų dėka turintis žymiai daugiau galimybių.

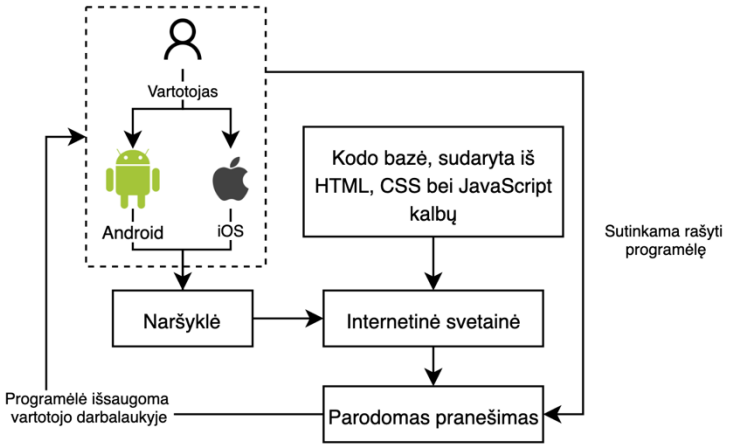
WEB programa yra žiniatinklio technologijų pagalba sukurta programėlė. Tokios programų sistemos kūrimas yra gana paprastas, kadangi programa gali būti pasiekiama iš bet kokios platformos naudojant naršyklę. Žiniatinklio programos dažniausiai yra sukurtos naudojant HTML, CSS bei JavaScript programavimo kalbą (žiūr. 1 pav.). Tokio tipo programėlės privalumai yra keli: gali būti pasiekiama iš daugelio įrenginių, turinčių naršyklę [LLN16] [ALF18]; vartotojui nereikia atsinaujinti programėlės, jeigu buvo atlikti tam tikri pakeitimai [LLN16]; trumpesnis programėlės kūrimo laikas bei mažesnė kaina [ALF18]. Tačiau, taip pat yra ir trūkumų: nėra galimybės pasiekti mobiliųjų įrenginių valdiklių [LLN16] [ALF18]; programėlės našumas mažesnis, ji veikia lėčiau ir yra pilnai priklausoma nuo interneto ryšio [LLN16]; programėlė negali būti patalpinta mobiliųjų programėlių parduotuvėse (pavyzdžiui, App Store arba Google Play) [LLN16]; vartotojo sąsaja gali skirtis priklausomai nuo naudojamo mobiliojo įrenginio [ALF18];



1 pav. WEB pagrindu sukurtos programėlės pasiekiamumas

Progresyvi žiniatinklio programėlė yra panaši į įprastą žiniatinklio kalbų (HTML, CSS, JavaScript) pagalba sukurtą programų sistemą, tačiau, papildomų technologijų dėka turinti daugiau galimybių (pavyzdžiui, progresyvią programėlę galima įkelti į mobiliųjų programėlių parduotuves [BMG17]). Tokio tipo programėlės yra pasiekiamos iš daugelio platformų [BGG18]; jų vartotojo sąsaja panaši į vietinio tipo programėlių [BGG18]; gali būti naudojama neturint interneto ryšio [BGG18]; užsikrauna iki 2-3 kartų greičiau, nei paprastas internetinis puslapis, pritaikytas mobiliųjų telefonų ekranams [RGK19].

Kuomet apsilankoma internetinėje svetainėje, kuri yra progresyvi, vartotojas turi galimybę ją parsisiųsti būdamas naršyklėje (žiūr. 2 pav.). Puslapyje parodomas pranešimas, siūlantis įrašyti šią svetainę į savo mobilųjį įrenginį. Atlikus šį veiksmą, kartu su internetinės svetainės piktograma darbalaukyje bus atsiųsti visi reikalingi failai (pavyzdžiui, šriftai, nuotraukos, HTML, CSS, JavaScript failai ir kt.) programėlės veikimui netgi neturint interneto ryšio [BGG18].

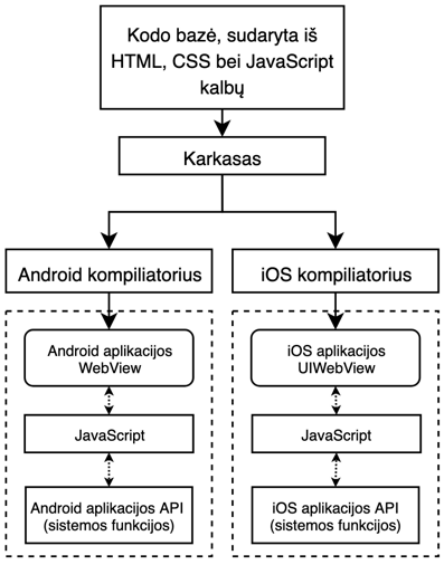


2 pav. Progresyvios programėlės įrašymas iš internetinės svetainės

1.1.2. Kompiliavimo pagrindu kuriamos programėlės

Šiame skyriuje aptariami trys pagrindiniai mobiliųjų programėlių kūrimo būdai, paremti kodo generavimo arba kompiliavimu būdu: hibridinė, model-driven ir cross-compiled. Nors šie metodai tarpusavyje skiriasi, galiausiai jų rezultatas yra panašus – visavertis, vietinio tipo programėlių funkcijas turintis produktas.

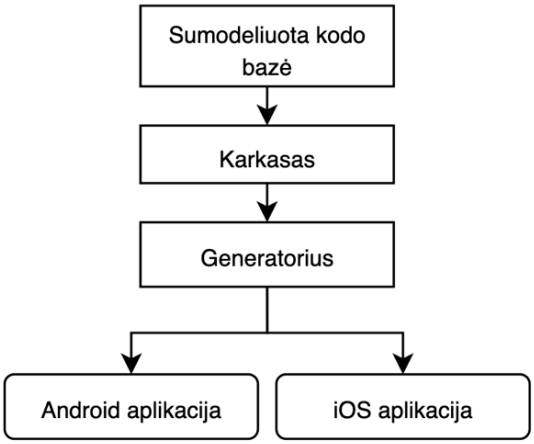
Hibridinė mobilioji programėlė yra mišinys tarp vietinio bei internetinio tipo programėlių. Įprastai jos kuriamos naudojant mobiliųjų programėlių kūrimo karkasą (*angl.* framework) Apache Cordova, kuris palengvina komunikavimo tarp mobiliojo įrenginio platformos WebView[2] ir įrenginio programavimo sąsajos (*angl. API*) procesą [BMG17] JavaScript programavimo kalbos pagalba (žiūr. 3 pav.). Tokios programų sistemos turi gana daug privalumų: jos veikia daugelyje platformų ir naudoja vieną kodo bazę [LLN16] [ALF18]; galima įkelti į mobiliųjų programėlių parduotuves [LLN16][BMG17]; turi prieigą prie mobiliojo įrenginio valdiklių [LLN16]; pasiekiama panaši vartotojo sąsaja į vietinio tipo programėlių [BGG18]; didelis alternatyvių karkasų pasirinkimas [BGG18]. Tačiau, taip įgyvendinti produktai dažnai būna lėtesni [LLN16]; skirtingose platformose negarantuojama identiška vartotojo sąsaja [LLN16] [BGG18], kadangi įvairios operacinės sistemos tą patį kodą interpretuoja skirtingai. Taip pat, hibridinės programų sistemos sukūrimui reikalingos papildomos technologijos ir įrankiai [BGG18].



3 pav. Hibridinės aplikacijos kūrimo struktūra

[2] WebView yra sistema, naudojama apdoroti žiniatinklio turinį mobilaus įrenginio programėlių viduje. https://developer.android.com/reference/android/webkit/WebView.html.

Dar vienas būdas kurti mobiliąją programėlę yra model-driven. Vienas iš modelių kurti programų sistema yra PIM (angl. Platform independent model). Kaip pavaizduota diagramoje (žiūr. 4 pav.), turima sumodeliuota kodo bazė generatoriaus pagalba yra paverčiama į tam tikros platformos programavimo kalbą [LNN16]. Vis dėlto, sugeneruotas kodas turi būti papildomai pakoreguotas rankiniu būdu naudojantis numatyta tos platformos kalba bei SDK[3] įrankiais [LNN16]. Tokiu būdu kuriamos programėlės trūkumai: palaikymas [LNN16]; labai nedidelis karkasų pasirinkimas (vienas iš pastebėtų – MD2[4]) [BGG18]; nepopuliaru [BGG18]. Nors, atrodo, jog model-driven programų sistemų kūrimo būdas nėra lengvas ir dažnai naudojamas, tačiau jis turi ir privalumų: vartotojo sąsaja bei našumas prilygsta vietinio tipo programėlei [LNN16] [BGG18]; gan aukštas abstrakcijos lygis [LNN16];



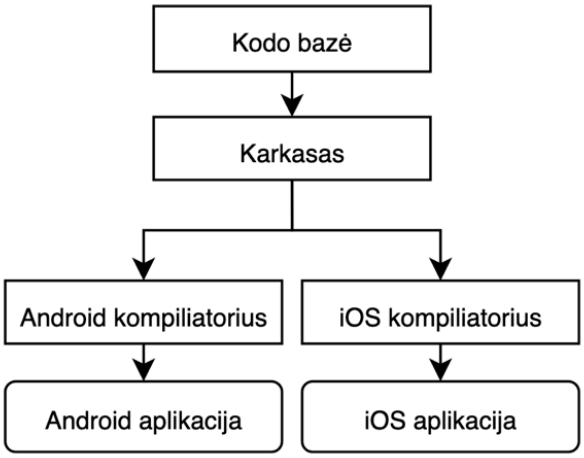
4 pav. Model-driven aplikacijos kūrimas

Cross-compiled mobiliųjų programėlių kūrimas nėra paremtas WebView komponentu ar JavaScript interpretatoriais, kaip kad hibridinėse programėlėse. Vietoj to, naudojama viena programavimo kalba, pavyzdžiui, C# [BMG17]. Parašyta kodo bazė, karkaso pagalba (pavyzdžiui, Xamarin), yra sukompiliuojama į kiekvienai platformai tinkamą formatą, tad galutiniame rezultate programėlė išskirstoma į skirtingoms platformoms skirtą vietinio tipo programų sistemą (žiūr. 5 pav.) [BGG18]. Vis dėlto, tokiu būdu programą parašyti užteks vieno karto, priešingai, nei vietinę, kuomet kiekvienai platformai reikalinga atskira atitinkama kalba parašyta kodo bazė. Toks programėlės kūrimo būdas yra naudingas tada, kuomet siekiama

[3] SDK (angl. Software Development Kit) – programinės įrangos kūrimo rinkinys. Kiekviena platforma suteikia savo specifinį įrangos kūrimo rinkinį, kuris dažnu atveju veikia tik jos ribose.

[4] https://www.wi.uni-muenster.de/sites/wi/files/public/department/pi/publications/heitkoetter/cross-platform-model-driven-development-of-mobile-applications-with-md2.pdf

turėti daugiaplatformę programų sistemą [LLN16] [BGG18], kuri našumo prasme yra panaši į vietinio tipo programėlę [LLN16] [BGG18] [BMG17], turi prieiga prie daugelio mobiliojo įrenginio valdiklių [BGG18] bei norima turėti kuo panašesnę į įprastos mobilios programėlės suteikiamą vartotojo sąsają [LLN16] [BGG18] [BMG17]. Tačiau, tokio tipo programėlės kūrimas kelia sunkumų dėl prieigos prie kai kurių mobiliojo įrenginio valdiklių, kadangi skirtingose platformose tam tikrų funkcijų iškvietimas skiriasi [LNN16].

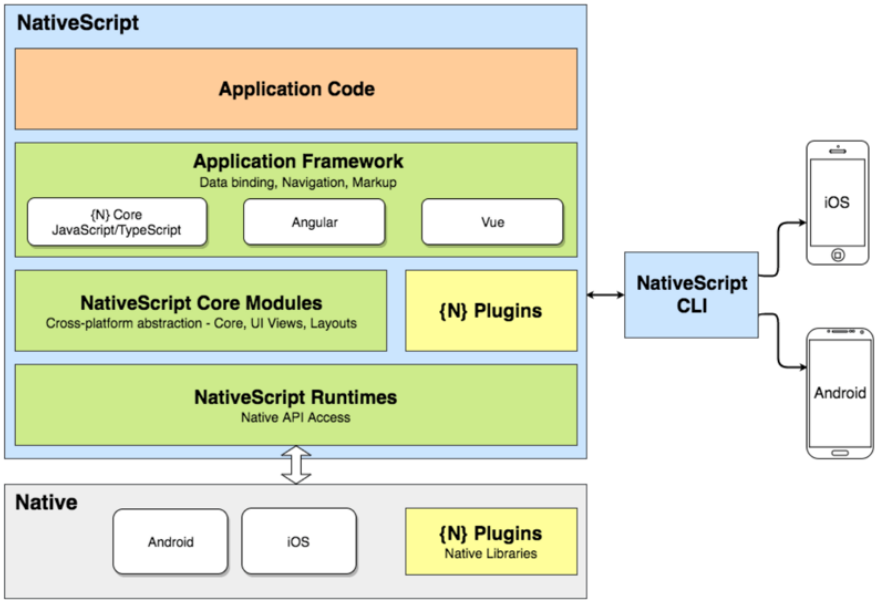


5 pav. Cross-compiled tipo programėlės struktūra

1.2. NativeScript įrankio apžvalga

NativeScript yra atviro kodo įrankis, skirtas daugiaplatformių mobiliųjų programėlių kūrimui naudojant React, Angular, Vue, Svelte karkasus arba VanillaJS[5]. Įrankis pirmą kartą buvo išleistas 2014 metais kompanijos Telerik ir vis dar palaikomas iki šių dienų. Šios technologijos pagalba galima sukurti iOS ar Android platformai skirtą programų sistemą naudojant vieną bendrą kodo bazę [MCL18]. Tuo pačiu NativeScript suteikia galimybę lengvai pasiekti tam tikrus mobiliojo įrenginio valdiklius bei naudoti vietinius platformos komponentus vartotojo sąsajai [DTC19]. Diagramoje galima pamatyti (žiūr. 6 pav.), kad NativeScript variklis sudarytas iš programos kodo, modulių bei „Runtimes“ bloko. Modulius sudaro abstrakcijos, pavyzdžiui, grafiniai vartotojo sąsajos elementai, išdėstymo elementai, tam tikrų programuotojo darbą palengvinančių funkcijų klasės ir pan. „Runtimes“ bloko pagalba įrankis gali tiesiogiai komunikuoti su mobiliųjų įrenginių platformų valdikliais (kamera, GPS ir t.t.). Galiausiai, NativeScript CLI paruošia, sukompiliuoja ir patalpina visą turimą kodo bazę į tam tikrą platformą.

[5] VanillaJS – JavaScript kalba be papildomų plėtinių.



6 pav. NativeScript karkaso techninė dalis. Šaltinis: https://v6.docs.nativescript.org/angular/img/ns-common.png

Viena iš pagrindinių Telerik kompanijos sukurto NativeScript karkaso pranašumų prieš kitas tokio tipo technologijas yra išsiskirianti ekosistema. Daugelis kitų karkasų bendravimui tarp parašyto kodo ir mobiliojo įrenginio platformos naudoja tarpinį komunikacijos sluoksnį, parašytą JavaScript kalba [And16]. Jeigu, pavyzdžiui, programų sistemoje norima integruoti Bluetooth funkcionalumą, tektų ieškoti papildomo plėtinio, kuris atliktų reikalingas funkcijas. Tuo tarpu NativeScript karkasas leidžia tiesiogiai pasiekti specifinio įrenginio platformos modulius [And16].

Winfredas Van Egterenas, „Triodos“ banko komponentų architektas apie „NativeScript“ atsiliepė[6] pasakė, kad naudojant „NativeScript“ ir pasitelkiant internetinių svetainių kūrimo įgūdžius galima kurti mobilias programas neinvestuojant laiko į iOS ir Android platformas. Viena kodo bazė bei mažai papildomo mokymosi reikalaujantis įrankis.

Šis įrankis dalyje skyrių bus analizuojamas Angular karkaso kontekste, kadangi remiantis perskaitytu straipsniu[7], norint užtikrinti optimalią kodo bazės struktūra, naudojant karkaso siūlomus konceptus, integracijas ir kodo dalinimo pernaudojimo architektūra, Angular yra

[6] https://www.telerik.com/docs/default-source/case-studies-documents/triodos-native-script-case-study-final.pdf?sfvrsn=bf9f412b_2

[7] https://www.cmarix.com/blog/why-nativescript-with-angular-makes-the-ideal-solution-for-mobile-app-development/

puikus pasirinkimas. Taip pat NativeScript dokumentacijoje[8] įvardijama, jog ši technologija turi gilią integraciją su Angular, kuri, pasak autorių, yra greičiausia JavaScript sistema.

1.2.1. Tikslinės platformos

Su NativeScript galima kurti tiek iOS, tiek Android platformose veikiančias mobiliąsias programėles. Taip pat, su 5.4 NativeScript CLI[9] versija pristatyta galimybė kurti programų sistemas ir WatchOS[10] platformose. Mobiliąsias programėles iOS platformoms galima kurti tik su MacOS operacinę sistemą turinčiu kompiuteriu, kadangi viena iš reikiamų sistemų norint dirbti su NativeScript yra XCode[11], esanti tik Mac kompiuteriuose. Be jos, taip pat reikia turėti Node ir NativeScript CLI. Tuo tarpu Android mobilioji programėlė gali būti sukurta trijose aplinkose: MacOS, Windows ir Linux. Norint kurti programų sistema kiekvienoje iš aplinkų, būtina savo kompiuteryje turėti Node, NativeScript CLI bei Android Studio.

1.2.2. Vartotojo sąsajos integravimas

NativeScript suteikia sąrašą vartotojo sąsajos komponentų, kurių pagalba lengva sukurti gražiai atrodančią mobiliąją programėlę, kiekvienoje platformoje veikiančią ir atrodančią vienodai. Visi komponentai gali būti koreguojami pagal programuotojo poreikius. Komponentai pagrinde sudaryti iš HTML ir CSS, bet tam tikrais atvejais taip pat įtraukiamas ir JavaScript funkcionalumas. Komponentų dokumentavimo[12] puslapyje galima pamatyti pateiktą kodo pavyzdį (žiūr. 2 priedas) bei aiškų aprašymą apie kiekvieno parametro funkciją. Vartotojo sąsajos komponentų pavyzdžiai: mygtukas, datos pasirinkimo langas, sąrašas, paieškos laukelis, progreso juosta ir kt. Komponentai labai stipriai palengvina programuotojų darbą, kadangi programėlės išdėstymas ir pagrindiniai elementai yra labai lengvai konfigūruojami [And16]. Egzistuoja netgi 6 išdėstymo variantai: Absolute, Dock, Grid, Flexbox, Stack ir Wrap. Kiekvienas iš jų yra atskirai konfigūruojamas.

[8] https://v6.docs.nativescript.org/angular/start/introduction

[9] Command Line Interface – komandų eilutės sąsaja.

[10] Operacinė sistema, naudojama Apple laikrodžiuose.

[11] https://docs.nativescript.org/environment-setup.html#macos-ios

[12] https://docs.nativescript.org/ui-and-styling.html#components

1.2.3. Interakcijos

Be vizualiai matomų komponentų, NativeScript taip pat siūlo ir paruoštas interakcijas. Ši biblioteka siūlo naudoti jau paruoštus įspėjimų bei modalinius langus, taip pat turi integruotas funkcijas gestų suvaldymui – paspaudimas, dvigubas paspaudimas, ilgas paspaudimas, braukimas pirštu ir kt. Tokiu atveju programuotojui nereikia atskirai įgyvendinti funkcionalumo, viskas jau yra paruošta ir aprašyta dokumentacijoje. Pavyzdžiui, norint atlikti kokį nors veiksmą, kuomet programėlės vartotojas laiko nuspaudęs tam tikrą elementą ilgesnį laiką, uždėjus atributą *longPress* ant HTML elemento, sistema ši veiksmą atpažins (žiūr. 7 pav.). Kaip matoma paveikslėlyje, šis veiksmas bus užregistruotas *onLongPress* metode su parametru, suteikiančiu papildomą informaciją apie šį veiksmą.

```
<label text="Long press here" (longPress)="onLongPress($event)"></label>
```

7 pav. Ilgo paspaudimo atpažinimas. Šaltinis: https://docs.nativescript.org/interaction.html#long-press

1.2.4. Mobiliųjų įrenginių valdikliai

Vienas iš didžiausių NativeScript karkaso pranašumų yra tas, jog JavaScript programavimo kalbos pagalba galima tiesiogiai pasiekti mobiliojo įrenginio valdiklius. Jeigu, pavyzdžiui, norint patikrinti iPhone išmaniojo telefono akumuliatoriaus likutį, vietinio tipo aplikacijoje kodas atrodytų taip (žiūr. 8 pav.), tai NativeScript aplinkoje, jį pavertus į JavaScript kalbą jis atrodytų taip (žiūr. 9 pav). Norint pasiekti tam tikrą mobiliojo įrenginio mazgą, reiktų apie jį pasiskaityti atitinkamos platformos dokumentacijoje ir vėliau tą kodą išversti į JavaScript kalbą. Svarbu paminėti, kad ne visada oficialioje mobiliojo įrenginio platformos dokumentacijoje pavaizduotas kodas perrašomas tiesiogiai, kadangi tam tikri programavimo sąsajos iškvietimai gali skirtis. Tokiu atveju įrašomas *@nativescript/types* paketas, kuris padeda išspręsti tokio tipo problemas.

```
float batteryLevel = [[UIDevice currentDevice] batteryLevel];
```

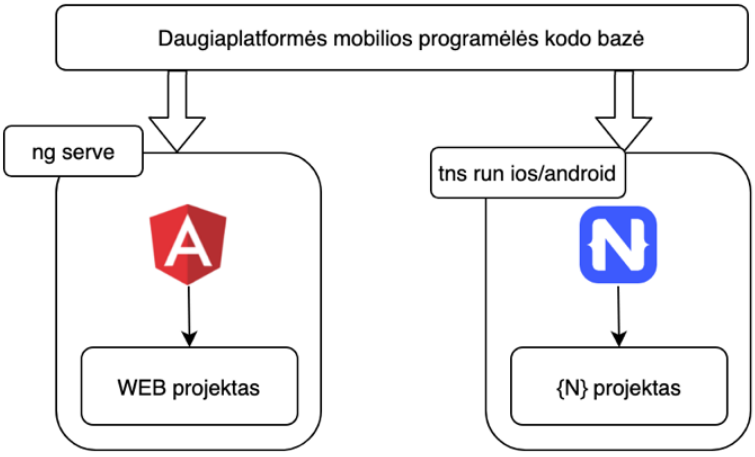
8 pav. iOS platformos kodas baterijos likučiui patikrinti. Šaltinis: https://docs.nativescript.org/native-api-access.html

```
if (global.isIOS) {
  UIDevice.currentDevice.batteryLevel
}
```

9 pav. JavaScript kodas baterijos likučiui patikrinti. Šaltinis: https://docs.nativescript.org/native-api-access.html

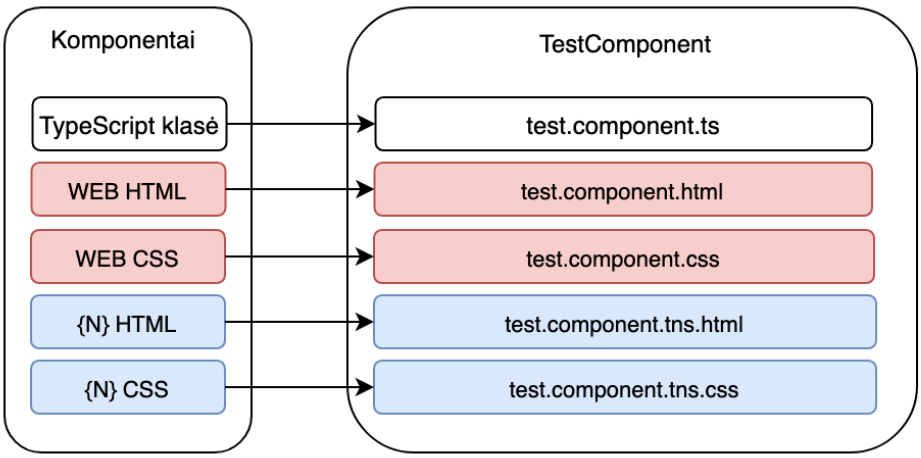
1.2.5. Kodo valdymas bei dalinimas priklausomai nuo platformos

Angular ir NativeScript karkasų kompiliatoriai turi skirtingus projekto paleidimo procesus. Turint vieną daugiaplatformę kodo bazę, sudarytą pagal 3.1.6 skyriuje aprašytą antrąjį architektūrinį modelį, svarbu suprasti paleidimo proceso subtilybes. Pateiktoje diagramoje (žiūr. 10 pav) matoma, jog atlikus skirtingas vykdymo komandas, kreipiamasi į skirtingus kompiliatorius. Įvykdžius komandą *ng serve*, projekte esanti konfigūracija kreipsis į Angular CLI įrankį, kuris į paleidimo procesą įtrauks visus projekte esančius WEB failus. Tuo tarpu įvykdžius komandą *tns run ios/android*, bus kreipiamasi į NativeScript CLI įrankį, kuris į paleidimo procesą įtrauks tik mobiliai programėlei skirtą programinį kodą.



10 pav. Kodo bazės paleidimas skirtingoms platformoms

Angular CLI ir NativeScript CLI failus atskiria pagal jų pavadinimus. Mobilios programos failai yra žymimi su papildoma priesaga **.tns**, tuo tarpu WEB failai išlieka įprastais pavadinimais. Pavyzdys, kaip šiame modelyje atrodo skirtingų kodo dalių ryšiai, pavaizduotas žemiau esančioje schemoje (žiūr. 11 pav.). Kuriant komponentą pavadinimu **TestComponent**, TypeScript klasės failas bus naudojamas tiek mobilioje programėlėje, tiek WEB aplikacijoje. Schemoje taip pat pavaizduotos dvi poros HTML ir CSS failų, kurių vieni turi **.tns** priesagą. Tai reiškia, jog failas, esantis su įprasta .html galūne bus atvaizduojamas WEB programėlėje, o .tns.html – mobilioje programėlėje. Analogiškai bus atskirti ir stiliaus failai. Tai nėra vienintelis būdas atskirti failus. Juos galima išskirstyti ir pagal atskiras platformas naudojant **.android** arba **.ios** priesagas. Ši failų pavadinimų struktūra gali būti pritaikoma ir kitiems klasių failams – servisams, komponentų klasėms, komponentų moduliams ir t.t.



11 pav. Kodo bazės failų išskirstymas pagal atitinkamą platformą

Lyginant su kitais įrankiais, pavyzdžiui, Ionic[13], toks sprendimas nėra siūlomas. Vietoje to, yra galimybė naudoti **Platform**[14] servisą, kuris savyje turi metodą *is()*. Tokiu būdu komponento viduje rašant sąlygą *is(android){}* kodas gali būti atskirtas pagal specifinę platformą. Pagrindinis trūkumas yra tas, jog tokiu atveju programinis kodas smarkiai išsiplėčia ir skaitomumas tampa sudėtingas. Vienas iš sprendimo būdų būtų sukurti kokį nors sprendimą aukštesniame lygyje, tačiau šio tyrimo metu nebus nagrinėjami kitų įrankių problemų sprendimo būdai. Šiuo atveju NativeScript kodo dalinimui pagal specifines platformas siūlo patogesnį būdą.

Panašų sprendimą siūlo ir React Native karkasas. Pastarojo įrankio dokumentacijoje[15] siūlomas sprendimas yra panašus į NativeScript technologijos turimą architektūrą. Jeigu norima atskirti WEB ir mobiliojo įrenginio failus, pastarųjų pavadinimo galūnę reiktų pakeisti į **.native.js**, o jeigu norima specifinį failą naudoti konkrečiai platformai, yra siūlomi atitinkami galūnių variantai: **.ios.js**, **.android.js**. WEB platforma pagal nutylėjimą visada naudos tuos failus, kurie neturi jokių priesagų, pavyzdžiui failo galūnė yra tik **.js**.

[13] https://ionicframework.com/

[14] https://ionicframework.com/docs/angular/platform

[15] https://reactnative.dev/docs/platform-specific-code

2. Metodologija

2.1. Tyrimo metodas

Šiame darbe bus naudojama kokybinių tyrimų turinio analizės metodas, kadangi tiriamoje temoje nėra pakankamai atliktų tyrimų. Šis tyrimas bus padalintas į keturis etapus. Pirmiausia, atliekama literatūros analizė ir identifikuojami mobiliųjų aplikacijų kūrimo iššūkiai. Antrame etape atliekamas sprendimų ieškojimas problemoms, kurios buvo rastos pirmame etape bei sudaromas gairių sąrašas mobiliai programėlei sukurti. Trečią etapą sudaro praktinis NativeScript įrankio įvertinimas bandant įgyvendinti sudarytus reikalavimus. Galiausiai, paskutiniame etape aptariami tyrimo rezultatai.

2.2. Literatūrinė analizė

Mobiliųjų aplikacijų kūrimas yra procesas, kuomet programų sistemos sukuriamos mobiliems įrenginiams. Nors mobiliųjų programų kūrimo procesas yra panašus į standartinių programų, vis tik egzistuoja papildomi reikalavimai pagal kuriuos tradicinių programų sistemų kūrimas skiriasi nuo mobiliųjų programų sistemų. Tam, kad sužinoti mobiliųjų programėlių kūrimo skirtumus nuo tradicinių programų sistemų, buvo ieškoma mokslinių tyrimų bei apklausų, kurių metu buvo aiškinamasi, į ką svarbiausia atkreipti dėmesį kuriant programų sistemą mobiliems įrenginiams.

Analizės metu buvo rasta daugybę naudingos informacijos apie mobilios aplikacijos kūrimo procesą nuo pradžios iki pabaigos. Tačiau, ieškant informacijos, kokie didžiausi iššūkiai kyla kuriant tokias programas, buvo rasta tik nedidelė dalis tyrimų.

Wasserman savo tyrime [Was10], lyginant su darbalaukio programų sistemomis, išskyrė mobiliųjų aplikacijų programoms keliamus papildomus reikalavimus: vartotojo sąsajų programavimas pagal tikslinių platformų taisykles bei energijos suvartojimo įvertinimas. Taip pat, kadangi mobili aplinka, priklausanti nuo įvairių rūšių tinklų, skiriasi nuo tradicinės programų sistemos, tyrimo autorius iškėlė papildomus klausimus:

- Ar interneto ryšio praradimas turės įtakos mobilios programos veiklai?
- Ar esant silpnam arba potencialiam interneto ryšio dingimui yra įgyvendinti sprendimai duomenų tvarkymui, sinchronizavimui?

Šiais klausimais yra pabrėžiamas mobilios aplikacijos veikimo neprisijungus prie tinklo aktualumas.

Dehlinger [DD11], kalbėdamas apie tai, jog tradicinės programų sistemos kūrimo būdai nebetinkami mobiliųjų programėlių kontekste, identifikavo keturis iššūkius iškilusius analizuojant mobiliųjų aplikacijų kūrimą. Du iš jų yra tokie:

- **Perpanaudojama vartotojo sąsaja** – kuriant pernaudojamus vartotojo sąsajos elementus būtina atkreipti dėmesį į ekranų dydžius ir skiriamąją gebą. Pavyzdžiui, Apple platforma turi tik dviejų dydžių pasirinkimus – iPhone ir iPad. Android telefonai, skirtingai nei iOS, turi begalę skirtingų dydžių ekranų įrenginių ir skiriamųjų gebų;
- **Programos pernaudojimas skirtingose platformose** – prieš kuriant mobiliąją programėlę, svarbu nustatyti tikslines platformas. Kaip rašo tyrimo autorius, kūrėjai turi rinktis, ar programuoti mobilias aplikacijas kiekvienai platformai atskirai, ar kurti daugiaplatformę kodo bazę, kurios programinis kodas veiks keliose platformose.

Informacijos ieškojimo metu buvo randamas ne tik skirtumas tarp mobiliųjų bei tradicinių programų kūrimo, bet ir reikalavimų sąrašas mobiliųjų programų sistemų kūrimui. Datta [IDB13] daugiaplatformių mobiliųjų aplikacijų kūrimo būdų palyginimo ir įvertinimo tyrime įvardino tokius reikalavimus, kurie turi būti patenkinti mobilios programėlės kūrimo metu:

- **Patogi vartotojo sąsaja** – anot tyrimo autoriaus, tyrimo metu buvo nustatyta, jog daugiaplatformių programų sistemų kūrimo įrankiai negalėdavo pasiūlyti patogios vartotojo sąsajos įgyvendinimo. Kadangi mobilios programėlės stipriai priklauso nuo teigiamos vartotojo patirties, turi būti pilnai veikianti grafinės dalies biblioteka (interakcijos, animacijos ir kt.);
- **Energijos sunaudojimas** – Datta teigia, jog šiais laikais, kuomet žmonės didžiąją dalį dienos praleidžia naudojantis išmaniaisiais telefonais, sukurtos mobiliosios programėlės turi būti optimizuotos ir kuo mažiau iškraunančios akumuliatorių.

Atlikus pirmąjį tyrimo etapą išryškėjo tam tikri iššūkiai, su kuriais susiduriama kuriant mobiliąją programėlę bei taip pat įvardinti mobilios programų sistemos kūrimui keliami reikalavimai. Lengvesnei analizei rezultatai buvo surašyti į lentelę (žiūr. 1 lentelė). Tarp trijų analizuotų šaltinių pastebima vartotojo sąsajos tendencija. Autoriai minėjo, jog vienodas vartotojo sąsajos atvaizdavimas visose platformose, kodo pernaudojimas ir gera vartotojo patirtis yra svarbi dalis kuriant mobiliąją programėlę. Kiti paminėti dalykai buvo energijos suvartojimo įvertinimas bei programos veikimas neturint interneto ryšio.

1 lentelė. Autorių paminėti reikalavimai mobiliųjų programėlių programoms

Tyrimo autorius	Reikalavimai ir iššūkiai
Wasserman	• Vartotojo sąsaja • Energijos suvartojimo įvertinimas • Programos veikimas neturint interneto ryšio
Dehlinger	• Vartotojo sąsaja • Programos pernaudojimas keliose platformose
Datta	• Patogi vartotojo sąsaja • Energijos sunaudojimas

2015 metais buvo atlikta apklausa[16], kurios metu buvo analizuojama programėlių naudotojų elgsena tam tikrais atvejais. Tyrimo išvados rodo, kad naši, stabili, nedaug resursų naudojanti ir gerą vartotojo patirtį suteikianti mobilioji programa yra raktas į lojalius jos naudotojus. Apklausos metu buvo nustatyta, jog 36% vartotojų nusprendė pašalinti mobiliąją programėlę iš savo išmanaus telefono dėl to, jog ją naudojant greitai mažėdavo baterijos kiekis. Tuo tarpu 20% apklaustųjų teigė, jog mobiliąsias programas pašalino dėl to, kad programėlė naudojo per didelį mobiliojo interneto duomenų kiekį. Galiausiai, paklausus kiek kartų vartotojas, patyręs blogą programėlės veiklą, pabandė dar kartą ja pasinaudoti, net 80% atsakiusiųjų nurodė, kad bando tik iki 3 kartų.

Daroma išvada, jog skaičius projektuojant mobiliąją programėlę turi būti minimalus, kitaip potencialių jos naudotojų skaičius staigiai mažės. Apklausos metu išryškėję kritiniai mazgai buvo našumas, stabilumas, didelis baterijos energijos suvartojimas bei pastebimas internetinių duomenų suvartojimas. Dalis šių problemų sutampa su mokslinių tyrimų analizės metu nustatytais iššūkiais (žiūr. 1 lentelė) kuriant mobilias programėles.

2.3. Gairių mobilios programų sistemos kūrimui sudarymas

Prieš kuriant mobiliąją aplikaciją, labai svarbu įsitikinti, jog kiekvienas programos sluoksnis yra tinkamai sukonstruotas. Net mažiausios problemos, galinčios kilti mobilios

[16] https://techbeacon.com/sites/default/files/gated_asset/mobile-app-user-survey-failing-meet-user-expectations.pdf

programos kūrimo metu, gali pakenkti galutinio rezultato kokybei. Kas yra mobiliųjų programų architektūra? Tai yra struktūrinių elementų ir jų sąsajų rinkinys, iš kurio sudaroma sistema. Galima sakyti, kad tai yra programos griaučiai, o visą mobiliosios programos darbą lemia architektūros kokybė. Praleidę svarbų elementą kurdami mobiliosios programos architektūrą, kyla pavojus iššvaistyti daug laiko ir išlaidų. Mobiliosios programos modelis arba architektūra paprastai yra sudaryta iš kelių sluoksnių – vartotojo patirties, verslo logikos bei duomenų sluoksnio. Kuriant programą mobiliems telefonams, svarbu numatyti gaminamo produkto reikalavimus bei taip nuspręsti, kokia architektūra bus tinkamiausia.

Šiame skyriuje pateikia gairs, į kurias reiktų atkreipti dėmesį kuriant mobiliąją programėlę. Gairės buvo kuriamos analizuojant 2.2 skyriuje Wasserman, Dehlinger ir Datta autorių atliktus tyrimus bei 2015 metais „Dimensional Research“ įmonės vykdytos apklausos, kurios metu buvo apklausiama vartotojų elgsena naudojant mobilias aplikacijas, rezultatus. Besikartojantys rezultatai buvo sugrupuoti į vieną gairę. Pavyzdžiui, vartotojo sąsajos svarbumas buvo paminėtas visuose šaltiniuose, tad akivaizdu, kad kuriant mobilią programų sistemą vienas iš pirmų dalykų, kurie tūrėtų būti apgalvoti, yra vartotojo sąsaja (G1). Antrasis dalykas, pasikartojantis Wasserman tyrime ir apklausoje, buvo susijęs su duomenų tvarkymu. Wasserman tyrime pastebima, jog praradus interneto ryšį, vartotojas ir toliau turėtų turėti galimybę naudotis mobiliąja programų sistema, o apklausoje pabrėžiama, kad dėl per didelio sunaudojamo internetinių duomenų kiekio net 20% apklausoje buvusių dalyvių pašalino programą iš savo mobiliojo įrenginio. Tokiu atveju kuriant mobiliąją programėlę svarbu atkreipti dėmesį į duomenų tvarkymą: vartotojas, net ir neturėdamas interneto ryšio, visada turi turėti galimybę matyti naujausius duomenis; duomenys neturėtų būti siunčiami dažnai ir dideliais kiekiais; pravartu naudoti lokalią duomenų bazę bei kešavimo funkcionalumą, siekiant optimizuoti pastovių duomenų užkrovimą (nuotraukos, šriftai, grafiniai elementai ir kt.). Tad antroji gairė G2 yra duomenų tvarkymo projektavimas. Trečiosios (G3) gairės sukūrimo priežastį lėmė tai, kad Dehlinger tyrimo bei apklausos metu buvo akcentuojamas programėlės kodo pernaudojimo reikalingumas. Dehlinger tyrimo metu pabrėžiama, jog aktualu nuspręsti, ar programa bus daroma vieną kartą ir naudojama keliose platformose, ar du kartus kiekvienai platformai atskirai. Tuo tarpu apklausoje buvo paminėti našumo bei stabilumo kriterijai, kurie tiesiogiai susiję su kodo kokybe turint kuo mažiau besikartojančio kodo. Kadangi darbas orientuotas į daugiaplatformės mobilios programos kūrimą, trečioji gairė nusako kodo bazės, turinčios minimalų pasikartojančio programinio kodo kiekį, projektavimo svarbą kuriant keliose platformose veikiančią programą. Paskutinioji gairė (G4) buvo sukurta dėl Wasserman ir Datta tyrimuose minimo energijos suvartojimo įvertinimo bei apklausos

metu iš 36% vartotojų gauto atsakymo, jog daug telefono akumuliatoriaus energijos suvartojančią programą nusprendžiama pašalinti. Svarbu paminėti, jog kuriant mobilią programų sistemą svarbi ne tik kuo paprastesnė vartotojo sąsaja, bet ir siunčiamų užklausų skaičius tinkle, duomenų atnaujinimo dažnumas ir kt. Kitu atveju gali padidėti ne tik vartotojo naudojamo įrenginio energijos suvartojimas, bet ir procesoriaus apkrova bei operatyvios atminties suvartojimas. Tad paskutinis svarbus reikalavimas kuriant programų sistemą mobiliems įrenginiams yra atsižvelgimas į tikslinių įrenginių galimus techninius apribojimus.

Laikantis šių gairių bus lengviau įsitikinti, jog projektas atitiks 2.2 skyriuje išryškėjusius svarbiausius keliamus reikalavimus kuriant daugiaplatformę programų sistemą mobiliems įrenginiams:

- **G1. Vartotojo sąsaja apgalvota kelių platformų kontekste –** kadangi vartotojai naudoja skirtingus mobiliuosius įrenginius, svarbu įvertinti, jog sukurta vartotojo sąsaja kiekvienam vartotojui gali būti atvaizduojama skirtingai. Tai gali nutikti dėl tokių faktorių kaip skirtinga skiriamoji geba, ekrano dydis ar dėl to, jog kiekviena platformą tą patį kodą gali interpretuoti savaip. Tokiu atveju geriausias sprendimas būtų naudoti nepergrūstą ir kuo paprastesnę vartotojo sąsają arba naudoti trečiųjų šalių siūlomas bibliotekas jai sukurti. Vartotojo sąsajos bilbiotekas naudoti rekomenduojama todėl, kad dažnu atveju jos išsprendžia skirtingo atvaizdavimo problemą ir grafinius elementus skirtingose platformose atvaizduoja vienodai.
- **G2. Duomenų tvarkymo projektavimas** – jeigu kuriant daugiaplatformę mobiliąją programėlę yra reikalinga nuotolinė duomenų bazė, svarbu numatyti atvejus, kuomet mobili įranga interneto ryšį praranda. Šiuo atveju itin svarbu suprojektuoti laikinosios talpyklos, lokalios duomenų bazės bei duomenų sinchronizavimo mechanizmus, turint nepastovią prieigą prie tinklo. O jeigu mobilios programėlės idėja nereikalauja turėti bendrų duomenų tarp jos vartotojų (pavyzdžiui, tai yra žaidimas) ir interneto ryšys programėlės veikimui nėra būtinas, tuomet užtenka suprojektuoti duomenų tvarkymą lokalioje aplinkoje.
- **G3. Programinio kodo pernaudojimas –** planuojant kurti didelį projektą labai svarbu išvengti kodo pasikartojimo, kadangi priešingu atveju didėja rizika turėti neefektyvią ir sunkiau palaikomą programą. Pavyzdžiui, jeigu kuriama daugiaplatformė mobilioji aplikacija iOS ir Android platformoms vienoje kodo bazėje ir tam tikrų programos langų funkcionalumas bei atvaizdavimas sutampa, siūloma turėti tėvinį komponentą, iš kurio kiekvienai platformai skirtas failas

programinį kodą paveldėtų. Tokiu atveju, tas pats funkcionalumas abejose platformose būtų testuojamas tik vieną kartą ir būtų užtikrinama, jog funkcija veikia vienodai visose vietose, kur ji yra naudojama.

- **G4. Atsižvelgimas į įrenginio resursų apribojimus -** kiekvienas funkcionalumo arba vartotojo sąsajos sprendimas turėtų įvertinti limituotą operatyvią atmintį, baterijos talpą bei vidinę atmintį. Pavyzdžiui, ar mobilioji programa privalo atlikti atsarginės duomenų bazės kopiją iš karto, kai tam ateina laikas? O gal šis veiksmas gali būti atliekamas tada, kai programėlė yra neaktyvi ir vartotojas mobilųjį įrenginį krauna? Taip pat, jeigu programėlėje planuojama rodyti vaizdinę medžiagą ir nenorima apkrauti telefono ją apdirbant, ypač svarbu ją pateikti tokiu formatu, kokį geriausiai apdirba konkreti platforma. Pavyzdžiui, iOS ir Android platforma geriausiai apdirba MP4 vaizdinės arba garsinės medžiagos formatą.

3. NativeScript įrankio įvertinimas

Tam, kad galima būtų ieškoti sprendimų reikalavimų įgyvendinimui naudojant NativeScript, autorius sukūrė daugiaplatformės mobilios aplikacijos kodo bazę asmeniniame kompiuteryje. Aplinka buvo paruošta darbui su NativeScript ir Angular karkasais. Mobilios programos testavimui mobiliuose įrenginiuose taip pat buvo pasiruošta – suinstaliuotos bei sukonfigūruotos Android Studio ir XCode programos. NativeScript įrankio bandymai buvo atliekami su Macbook Pro 2019 kompiuteriu, kurio aparatinė įranga leidžia simuliuoti iOS ir Android platformas.

3.1. G1 reikalavimo įgyvendinimas

Vartotojo patirtis yra labai svarbus elementas planuojant kurti mobiliąją programėlę. Tikslas - suteikti teigiamą patirtį, kuri vartotojus palaiko lojalius produktui ar prekės ženklui. Kaip jau buvo minėta 1.2.2 ir 1.2.3 skyriuose, NativeScript siūlo naudoti jau sukurtus vartotojo sąsajai skirtus komponentus, kurie kiekvienoje platformoje atrodys taip kaip jiems priklauso. Kaip rašo H. Shah savo straipsnyje[17], kur lygina Flutter ir NativeScript įrankius, dėl savitos grafinių elementų integracijos NativeScript karkasas siūlo geriausią patirtį ir našumą labai didelių ir kompleksiškų projektų kontekste. Taip pat ir kitame straipsnyje[18], kur lyginami Ionic ir NativeScript karkasai, autorius D. Jaiswal teigia, jog šiame darbe nagrinėjamas įrankis yra žymiai našesnis, kadangi vartotojo sąsajos komponentai yra tiesiogiai talpinami išmaniuose įrenginiuose. Tuo tarpu Ionic grafinės sąsajos našumas yra prastesnis dėl per daug naudojamų tarpinių įrankių, kadangi komponentai yra konvertuojami pasinaudojant žiniatinklio technologijomis, o mobiliuose įrenginiuose jie atvaizduojami naudojant Cordova apvalkalą.

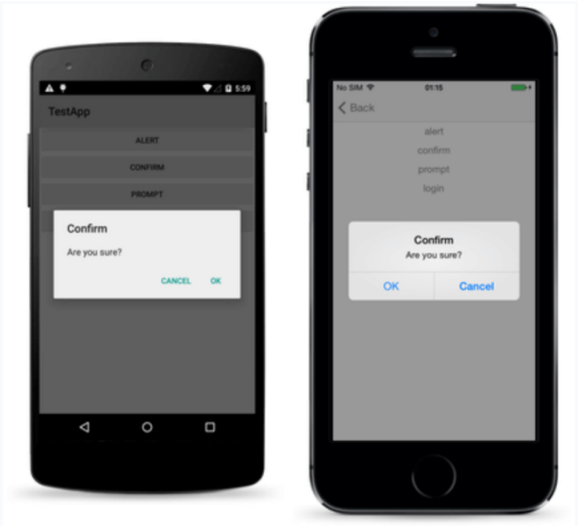
Žemiau esančiuose paveikslėliuose pavaizduoti laiko pasirinkimo (žiūr. 12 pav), sistemos pranešimo lango (žiūr. 13 pav), krovimosi indikatoriaus (žiūr. 14 pav) ir sekcijų navigacijos (žiūr. 15 pav) pavyzdžiai. Android dalyje (kairėje paveikslėlių pusėje), vartotojo sąsajos elementai atvaizduojami atitinkamai pagal telefono numatytą dizainą kaip ir iOS platformoje (dešinėje paveikslėlių pusėje). Nors ir yra siūloma integruota ir jau paruošta grafinių elementų biblioteka, tačiau NativeScript programuotojams palieka laisvę rinktis tarp trečiųjų šalių bibliotekų, pavyzdžiui: Material, Bootstrap ir Kendo. Taip pat yra siūlomas

[17] https://www.simform.com/flutter-vs-nativescript/

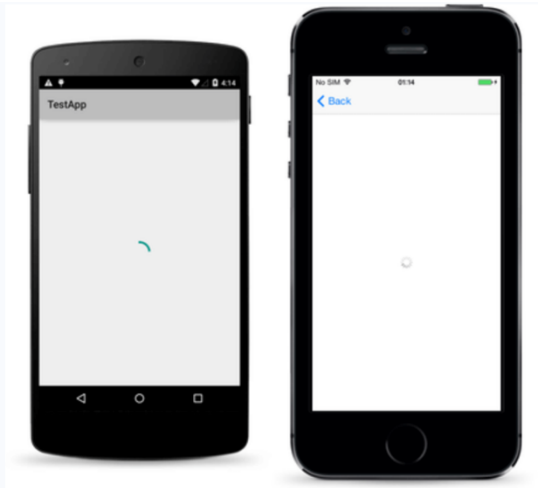
[18] https://medium.com/@DheerajJaiswal6/ionic-vs-nativescript-82d12149c889

oficialus įrankis[19], kurio pagalba galima susikonfigūruoti vartotojo sąsajos biblioteką remiantis Bootstrap ar Material pagrindu bei pasirinkti spalvų gamą. Susikonfigūravus elementus, biblioteką galima atsisiųsti ir importuoti į esamą NativeScript projektą.

12 pav. Laiko pasirinkimo grafinė sąsaja



13 pav. Sistemos pranešimo grafinė sąsaja



14 pav. Krovimosi indikatoriaus grafinė sąsaja

15 pav. Sekcijų navigacijos grafinė sąsaja

3.2. G2 reikalavimo įgyvendinimas

Turint vieną kodo bazę kuriant daugiaplatformę mobiliąją programėlę, taip pat svarbu, kad programėlės turėtų tarpusavyje dalintųsi duomenis. Tam, kad turėti vienodus duomenis visų platformų mobiliosiose programėlėse, reikalingi papildomi sprendimai. Paprastai galima turėti vieną nuotolinį duomenų bazės serverį, į kurį bus saugomi duomenys iš visų platformų

[19] https://themebuilder.telerik.com/

programėlių. Šioje vietoje pasirinkimas gana didelis – Firebase, MongoDB, Couchbase ir kt. Dalis iš paminėtų duomenų bazių įrankių jau yra pritaikytos NativeScript karkasui ir gali būti parsisiųstos iš oficialios parduotuvės[20]. Taip pat svarbu paminėti, kad renkantis duomenų bazės įrankį reikia įvertinti planuojamą vartotojų srautą, įrašų skaičių, paslaugų kainą ir pan. – tokiu atveju bus išrinktas tinkamiausias kuriamam projektui įrankis. Pasirinkus vieną iš duomenų saugojimo įrankių ir įgyvendinus jį NativeScript kodo bazėje, visų platformų programėlės siųs ir gaus duomenis į ir iš to paties taško. Tokiu būdu visų skirtingų platformų programėlių vartotojai matys tą patį.

Didžiausia problema turint nuotolinį duomenų bazės serverį yra ta, jog dingus interneto ryšiui duomenys yra nebepasiekiami. Tokiu atveju vienas iš sprendimų būtų naudoti lokalią duomenų bazę, kurioje vartotojo atlikti veiksmai būtų saugomi telefone [SML20]. iOS ir Android platformos galėtų naudoti, pavyzdžiui, SQLite, RealmDB, ORMLite, Couchbase ar kitas duomenų bazes, o PWA programėlė – localStorage arba IndexedDB. Šis sprendimas leistų programėlėms veikti net ir neturint interneto ryšio. Tuomet, kai interneto ryšys vėl atsirastų, programėlė galėtų atnaujinti nuotolinę duomenų bazę duomenimis, kurie buvo sukaupti lokalioje duomenų bazėje [SGS13]. Nemažiau svarbu yra nuspręsti ir susimodeliuoti lokalios duomenų bazės sprendimą pagal tam tikrus kriterijus. Siekiant išanalizuoti ir pasirinkti tinkamą architektūrą, bus apžvelgtas duomenų sinchronizavimo, saugojimo ir perkėlimo modeliai.

Duomenų sinchronizavimo modeliai yra du: asinchroninis ir sinchroninis [MS12]. Jie skiriasi tuo, jog turint asinchroninį duomenų valdymo sprendimą, duomenys bus tvarkomi atskirai ir nepriklausomai nuo vartotojo. Tai reiškia, kad jis galės ir toliau naudotis mobiliąja programėle, kol fone vyksta tam tikri procesai. Sinchroninis duomenų tvarkymas veikia priešingai – jis neleidžia vartotojui naudotis programėle tol, kol nebaigiamas atlikti prieš tai pradėtas veiksmas [NM16].

Duomenų saugojimo modeliai taip pat yra du: pilnas ir dalinis [MS12]. Šioje vietoje sprendžiama, ar lokalioje duomenų bazėje norima laikyti visus programėlės duomenis, ar tik tuos, kurie yra reikalingi. Nuotolinėje duomenų bazėje pagal nutylėjimą kaupiami visi įmanomi įrašai, tačiau ne visi jie yra reikalingi pilnam mobiliosios programėlės veikimui [NM16].

Paskutinė kategorija yra duomenų perkėlimo modelis. Jų iš viso yra trys: pilnas duomenų perkėlimas, dalinis duomenų perkėlimas remiantis laiko žyma ir dalinis duomenų perkėlimas remiantis algoritmu [MS12]. Pilnas duomenų perkėlimas reiškia, jog tiek iš nuotolinio serverio

[20] https://market.nativescript.org/

į lokalųjį, tiek atvirkščiai, perkeliami visi duomenys be išimčių. Antru atveju tik po tam tikro laiko pasikeitę duomenys yra perkeliami iš serverio į lokalią duomenų bazę arba atvirkščiai. Tokiu atveju yra išsaugomas laikas, kada buvo atlikta paskutinė sinchronizacija [MS12]. Paskutinis modelis yra panašus į antrąjį, tačiau reikalingų duomenų atnaujinimas parenkamas ne pagal laiką, o pagal tam tikrą specifinį algoritmą [NM16].

Skirtingi šių sinchronizavimo modelių deriniai skirtingai veikia programos našumą. Pavyzdžiui, sinchroninis, pilno duomenų saugojimo ir pilno duomenų perkėlimo modelis gali užtrukti daug laiko, jei duomenų bazė yra sukaupus daug duomenų. Tokiu atveju vartotojas bus priverstas laukti duomenų perkėlimo gan ilgą laiką. Kitu atveju, pasirinkus asinchroninį, dalinio saugojimo, ir dalinio duomenų perkėlimo pagal laiko žymę modelį, didelės įtakos vartotojo naudojimuisi programėle nebus. Tuo pačiu bus sukelti tik tie duomenys, kurie atsirado po paskutinio atnaujinimo. Apibendrinant, programos kūrėjas turi pasirinkti programos duomenų tvarkymo modelį pagal kuriamo produkto veikimo principus.

Tam, kad palengvinti duomenų bazės architektūros sudarymą, darbo autorius sudarė klausimyną:

1) Ar programėlė turi nuotolinę duomenų bazę?
2) Ar programėlė turi leisti vartotojui toliau daryti veiksmus, kol duomenų apdorojimo procesas vyksta?
3) Ar programėlės duomenys turi atsinaujinti kaskart ją atsidarius?
4) Ar programėlės duomenys turi atsinaujinti tik vartotojui panorėjus?

Nestabilaus interneto ryšio atveju galima įgyvendinti lokalios ir nuotolinės duomenų bazės sąveiką. Tokiu būdu dingus interneto ryšiui vartotojo kuriami duomenys bus kaupiami, o jam vėl atsiradus – įrašomi į nuotolinę duomenų bazę, iš kurios duomenys jau bus pasidalinti su kitų platformų vartotojais.

Lokalios duomenų bazės integracija panaudojant NativeScript

Kadangi iOS ir Android platformos palaiko SQLite lokalią duomenų bazę, tad ji ir bus naudojama. NativeScript įrankiui ši duomenų bazė jau yra pritaikyta, ją atsisiųsti galima NPM pagalba įvykdžius komandą *npm i –save nativescript-sqlite.* Kadangi šis modulis bus naudojamas tik iOS ir Android platformoms, jis turės būti aprašytas **app.module.tns.ts** faile. Tokiu atveju leidžiant WEB programą, ji neužkraus mobilioms platformoms skirtų modulių (WEB naudoja **app.module.ts** failą).

WEB programos lokalios duomenų bazės integracijai pasirinkta IndexedDB. Tai yra HTML5 naršyklės duomenų saugojimo technologija [Kim15]. Tai yra NoSQL asinchroninė duomenų saugykla, kurioje galima apdoroti didelį duomenų kiekį. IndexedDB palaiko programavimo sąsają, kuri suteikia greitą ir neribotą prieigą prie reikiamų duomenų [Kim15]. Visgi, nėra aišku, ar ši duomenų saugykla vis dar laikoma saugia, kadangi nuo 2006 metų ši dalis nebebuvo tiriama [KEL15].

Anot MDN[21] dokumentacijos, IndexedDB įgyvendinimas dažnu atveju yra komplikuotas, tad siūloma naudoti bibliotekas, palengvinančias šios duomenų bazės funkcionalumo naudojimą – Dexie, ZangoDB, MiniMongo, lovefield ir kt. Šiuo atveju buvo pasirinkta viena iš rekomenduojamų sąraše esančių bibliotekų, pavadinimu Dexie[22]. Pasirinkto įrankio dokumentacijoje[23] galima rasti instrukciją integruojant šį įrankį į Angular pagrindu sukurtą projektą. Parsisiųsti šią biblioteką galima įvykdžius komandą *npm install –save-dev dexie.* Kadangi šis modulis bus naudojamas tik WEB programoje, jis turi būti aprašytas tik **app.module.ts** faile. Tokiu būdu mobilios programos paleidimo metu Dexie modulis nebus užkraunamas.

Nuotolinės duomenų bazės integracija panaudojant NativeScript

Atsiradus interneto ryšiui, svarbu sinchronizuoti turimus lokalius duomenis su nuotoliniame serveryje esančia bendra duomenų baze. Šiai užduočiai išspręsti bus pasitelkta Google Firebase[24] duomenų bazės integracija. Sebastian Witalec konferencijos metu, kaip vienas iš iššūkių buvo paminėtas šios duomenų bazės integravimas. Taip yra todėl, kadangi Angular CLI ir NativeScript CLI paleidimo metu naudoja skirtingus HTTP užklausų interpretavimo modulius. Dėl šios priežasties, reikalingos dvi skirtingos bibliotekos – **@angular/fire** ir **@nativescript/firebase**. Jos atitinkamai gali būti pridėtos įvykdant komandas *npm install –save-dev @angular/fire* ir *npm install –save-dev @nativescript/firebase*. Svarbu paminėti, jog skirtingos bibliotekos reikalingos dėl WEB ir mobiliųjų įrenginių platformose esančių skirtingų protokolų. Užklausų siuntimas ir gavimas vyks į tą pačią duomenų bazę.

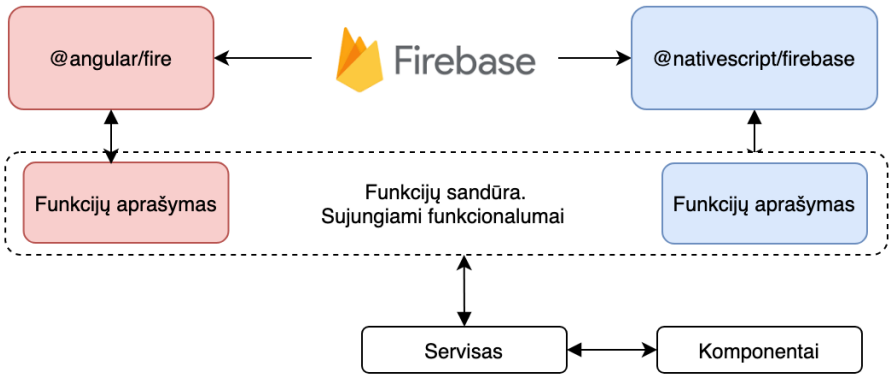
[21] https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API

[22] https://dexie.org/

[23] https://dexie.org/docs/Samples

[24] https://firebase.google.com/

Tokiu atveju reikalingas sprendimas aukštesniame lygyje, kuris leistų naudoti vieną bendrą servisą duomenų išsaugojimui arba gavimui, vietoje dviejų skirtingų atskiroms platformoms. Sebastian Witalec konferencijos metu buvo pristatyta duomenų bazės integravimo architektūra, sprendžianti šią problemą. Kaip matoma, žemiau esančioje diagramoje (žiūr. 16 pav.), sukuriama abstrakcija, kurioje tam tikri aprašyti metodai, priklausomai nuo platformos, kreipsis į vieną ar kitą biblioteką. Tokiu atveju užtenka vieno serviso, kuris bus tiesiogiai kviečiamas iš WEB arba NativeScript komponentų.



16 pav. Duomenų bazės bibliotekų architektūrinis modelis

Įdiegus pristatytą duomenų tvarkymo architektūrą, daugiaplatformė mobili programėlė visose platformose veiks net ir dingus interneto ryšiui. Vartotojai ir toliau galės naudotis programų sistema bei turėti vienodą naujausią informaciją visose platformose.

3.3. G3 reikalavimo įgyvendinimas

Nusprendus projektuoti sluoksniuotą kodo bazę, svarbu nuspręsti koks architektūrinis modelis tiks geriausiai. Atsižvelgiant į tai, jog NativeScript įrankis siūlo galimybę naudoti Angular technologiją, kuri naudoja MVC architektūrą, kodo pernaudojimas ir palaikymas puikiai išsprendžiamas. MVC yra gana paplitęs būdas kurti mobiliųjų programų architektūrą. Iš sluoksnių sudarytas architektūrinis modelis MVC atskiria vartotojo sąsają nuo programos logikos. Kaip rodo pavadinimas, šį modelį sudaro atitinkamai trys pagrindiniai komponentai:

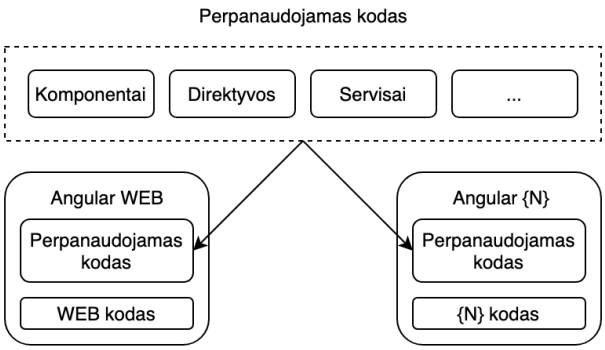
- **Model** – ši dalis yra atsakingas už duomenis, jų valdymą bei taisyklių, kaip modifikuoti ir valdyti duomenis, nustatymą. Modelių paskirtis yra gauti arba gražinti duomenis norimame formate.

- **View** – architektūros dalis, atsakinga už modelyje esančių duomenų atvaizdavimą. Ši dalis neturėtų keisti duomenų ar atlikti tam tikrus loginius veiksmus. Ši dalis skirta atvaizduoti grafinės sąsajos, pavyzdžiui, HTML kodą ir duomenis, gautus iš kontrolerio.
- **Controller** – paskutinioji šios architektūros dalis, atsakinga už komunikaciją tarp modelio ir atvaizdavimo dalių. Pavyzdžiui, programos vartotojui supildžius ir patvirtinus tam tikrą formą, kontroleris persiųs ją modeliui bei tuo pačiu į atvaizdavimo dalį gražins pranešimą, jog forma buvo sėkmingai užpildyta. Kontroleris neturėtų tvarkyti duomenų, jis turi atlikti tik tarpinio sluoksnio rolę.

Tokia programų sistemos struktūra padeda programai vienu metu sutelkti dėmesį į atskiras funkcijas ir užduotis, naudojant šiuos izoliuotus sluoksnius. Tokiu būdu galima daug lengviau pakeisti ar patobulinti programinį kodą, pridedant naujų funkcijų tik tam tikrame sluoksnyje. Apibendrinant, jeigu NativeScript mobilioji programėlė bus kuriama kartu su Angular karkasu, tai šis reikalavimas pagal nutylėjimą bus įvykdytas.

Remiantis Angular eksperto Sebastian Witalec vykusioje konferencijoje pateikta medžiaga, yra du būdai, kuriais galima sukurti pernaudojamą daugiaplatformės mobiliosios programėlės kodo bazę naudojant NativeScript + Angular. Pirmasis sprendimas yra vienoje kodo bazėje turėti du skirtingus projektus, kuriame vienas būtų skirtas mobiliųjų įrenginių platformoms, o kitas talpintų tik WEB failus. Antrasis būdas yra viską turėti vienoje vietoje.

Pirmasis sprendimas yra apibūdinamas kaip viena kodo bazė, aukštame lygyje turinti pernaudojamų kodo elementų bei talpinantis du skirtingus projektus – WEB ir mobiliąją programėlę. Žemiau pateiktoje diagramoje (žiūr. 17 pav.) matoma, jog kodo bazės šakninis aplankas talpina servisus, direktyvas, komponentus bei kitas galimas pernaudoti klases. Tuo tarpu Angular karkaso pagrindu sukurti atskiri WEB ir NativeScript projektai, išvengiant tam tikrų klasių pakartotinio rašymo, šakninėje direktorijoje esančias funkcijas gali panaudoti projekto viduje. Naudojant tokią architektūrą, internetinio puslapio projektas bus paleidžiamas iš vieno projekto, o mobilioji programėlė iš kito. Tokiu atveju išvengiama papildomos konfigūracijos programų sistemų paleidimui.

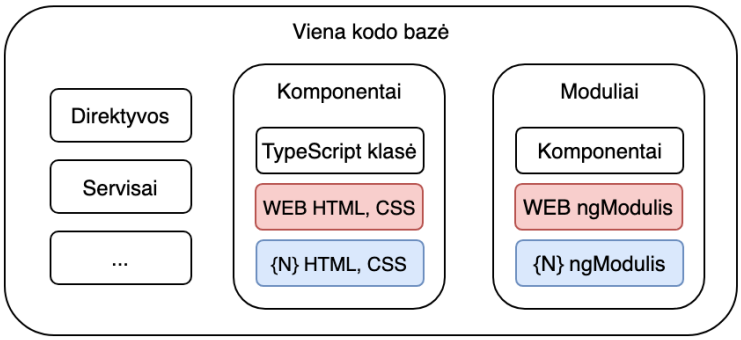


17 pav. Pirmasis architektūrinis kodo bazės modelis

Šis sprendimas, turint du skirtingus projektus vienoje kodo bazėje, turi tokius privalumus kaip lengvas kodo bazės paruošimas, yra tinkamas mažiems projektams bei nereikalauja papildomų žinių, kadangi WEB projektas veiks identiškai, kaip įprastas Angular internetinis sprendimas, o NativeScript projektas – savyje turintis tik mobilioms platformoms reikalingą programinį kodą. Tačiau toks kodo bazės modelis nėra patogus ir lengvai palaikomas esant didesniam projektui, kadangi didėjant puslapių arba mobiliojo įrenginio langų skaičiui, programinis kodas pradeda kartotis bei tampa sudėtinga užtikrinti, jog atlikti pakeitimai vienam projekte buvo taip pat atlikti ir kitame.

Antrasis sprendimas yra apibūdinamas kaip abiejų projektų turėjimas vienoje kodo bazėje. Tokiu atveju tiek WEB, tiek mobiliosios programėlės failai randasi toje pačioje direktorijoje ir lyginant su pirmuoju sprendimu, šiuo atveju išvengiamas kodo dublikavimas bei programų sistemos palaikymas dideliame projekte tampa žymiai paprastesniu. Kaip matoma diagramoje (žiūr. 18 pav.), komponentų viduje turima bendra TypeScript tipo klasė, tačiau WEB ir NativeScript HTML bei stiliaus failai sukurti kiekvienai platformai atskirai. Ta pati struktūra galioja ir moduliams – taip pat kaip ir WEB, mobilioji programėlė talpins tik jai reikalingus plėtinius. Failų atskyrimas pagal platformą yra būtinas, kadangi mobiliųjų įrenginių ir žiniatinklio kodo interpretacija yra skirtinga.

Įdomiausia dalis yra šios daugiaplatformės kodo bazės paleidimas. Norint paleisti WEB projektą, įvykdžius standartinę *ng serve* komandą, kompiliavimo metu NativeScript failai, diagramoje pažymėti mėlynai, bus ignoruoti. Lygiai taip pat leidžiant mobiliąją programėlę įvykdžius *tns run ios/android* komandą bus ignoruoti WEB failai, pažymėti mėlynai.



18 pav. Antrasis architektūrinis kodo bazės modelis

3.4. G4 reikalavimo įgyvendinimas

Kiekviename mobiliosios programėlės projektavime reikėtų atsižvelgti į limituotų resursų aspektą: limituota operatyvi atmintis, ribota baterijos talpa, limituota vidinė atmintis, galimi interneto ryšio trikdžiai dėl duomenų kiekio trūkumo ir pan. J. Swigle atliktoje šių metų daugiausiai parduotų telefono modelių rinkos analizėje[25] matyti, kad tarp penkių populiariausių pasaulyje parduotų telefonų didžiąją dalį sudarė Apple produkcija: iPhone 12 Pro Max, iPhone 12, iPhone 12 Pro ir iPhone 11. Tuo tarpu penktoje vietoje buvo Samsung Galaxy S21 Ultra 5G. Remiantis statistika galima spręsti, jog kadangi nemažą rinkos dalį sudaro galingų išmaniųjų telefonų savininkai, procesoriaus greitis ir operatyvioji atmintis nėra didžiausi mobiliųjų aplikacijų kūrėjų iššūkiai, tačiau rekomenduojama šių aspektų nepamiršti. Tuo tarpu baterijos veikimo laikas dažniausiai yra labiausiai ribojantis veiksnys mobiliuosiuose įrenginiuose. Apšvietimas, skaitymas ir rašymas į atmintį, belaidis ryšys, tinklo srautas ir specializuota aparatūra, pavyzdžiui, GPS, turi įtakos visam energijos suvartojimui. Nors turima ribota vidinė atmintis nėra pagrindė problema, piktnaudžiavimas ja gali paskatinti vartotojo operacinę sistemą paprašyti programos uždaryti arba paaukoti telefone esančius duomenis, tuo pačiu sulėtindamas programos vykdymą. Svarbu optimizuoti mobiliąją programą taip, kad ji reikalautų kuo mažiau resursų iš vartotojo naudojamo telefono.

Šis reikalavimas priklauso nuo labai daug veiksnių: programos kodo kokybės, programos dydžio, naudojamų servisų kiekio, grafinių elementų dydžio ir pan. Kadangi šis reikalavimas priklauso ne tik nuo naudojamo karkaso, jis nebuvo įgyvendinamas NativeScript įrankio pagalba.

[25] https://www.phonearena.com/news/best-selling-phones-q1-2021_id132273

Rezultatai

- Pristatyti daugiaplatformių programėlių kūrimo būdai;
- Apžvelgtas NativeScript karkasas ir jo siūlomos galimybės, funkcionalumai;
- Tyrimo metu identifikuoti reikalavimai, susiję su mobiliųjų programėlių kūrimu;
- Sudarytas gairių sąrašas, į kurias reiktų atkreipti dėmesį kuriant mobiliąją programėlę;
- Gairės aptartos NativeScript kontekste, pavaizduojant įgyvendinimo galimybę bei būdą;

Išvados

Šiame darbe buvo įvertintas daugiaplatformių mobiliųjų programėlių kūrimo įrankis NativeScript. Buvo apžvelgti pagrindiniai daugiaplatformių mobiliųjų aplikacijų kūrimo būdai bei apžvelgtas NativeScript karkasas plačiau aprašant jo siūlomas funkcijas ir galimybes. Tyrimo metu buvo identifikuoti mobiliųjų aplikacijų kūrimo iššūkiai: skirtingose platformose vienodai atrodanti vartotojo sąsaja, mobilios programėlės naudotojo įrenginio energijos suvartojimo įvertinimas, programėlės veikimas neturint arba esant nestabiliam interneto ryšiui bei kodo pernaudojimas keliose platformose. Juos išanalizavus kartu su apklausos metu gautais rezultatais, buvo sukurtas gairių sąrašas, kuris turėtų padėti mobiliųjų aplikacijų kūrėjams atitikti svarbiausius tokios programų sistemos reikalavimus: vartotojo sąsaja apgalvota kelių platformų kontekste, duomenų tvarkymo projektavimas, programinio kodo pernaudojimas, atsižvelgimas į įrenginio resursų apribojimus. Reikalavimai iš gairių sąrašo buvo įgyvendinti praktiškai naudojant NativeScript karkasą.

Šis tyrimas parodė, kad nors šis įrankis nėra populiarus mokslinių darbų kontekste, bet jo pagalba galima sukurti pilnai funkcionuojančią ir didžiąją dalį mobiliųjų programų kūrimo reikalavimų įvykdančią mobiliąją programų sistemą. Dėl įrankio siūlomų technologijų pasirinkimų įvairovės – JavaScript, Angular, Vue, React, Svelte - šis karkasas tikrai vertas dėmesio internetinių platformų programuotojų tarpe. Šio darbo metu išryškėjo NativeScript privalumai: šimtaprocentinė prieiga prie mobiliųjų įrenginių valdiklių, aukšto našumo vartotojo sąsajos komponentų biblioteka, patogi daugiaplatformės mobilios sistemos kodo bazės architektūra bei kodo dalinimas, leidžiantis atskirti reikiamas programinio kodo dalis pagal tam tikrą platformą.

Šaltiniai

[ALF18] A. Ahmand, K. Li, C. Feng, S. M. Asim, A. Yousif, A. S. Ge. An Empirical Study of Investigating Mobile Applications Development Challenges. 2017. Prieiga per internetą: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8326707

[And16] N. J. Anderson. Getting started with NativeScript. 2016.

[BGG18] A. Biørn-Hansen, T. Grønli, G. Ghinea. A Survey and Taxonomy of Core Concepts and Research Challenges in Cross-Platform Mobile Development. 2018. Prieiga per internetą: https://www.researchgate.net/publication/329079262

[BMG17] A. Biørn-Hansen, T. A. Majchrzak and T. Grønli. Progressive Web Apps: The Possible Web-native Unifier for Mobile Development. 2017. Prieiga per internetą: https://www.scitepress.org/Papers/2017/63537/63537.pdf

[DD11] J. Dehlinger, J. Dixon. Mobile Application Software Engineering: Challenges and Research Directions. 2011. Prieiga per internetą: http://bls.buu.ac.th/~s55103/00CourseResource/7-Dehlinger_Dixon.pdf

[DTC19] L. Delia, P. J. Thomas, L. Corbalán, J. F. Sosa. Development Approaches for Mobile Applications: Comparative Analysis of Features: Proceedings of the 2018 Computing Conference, Volume 2. 2019. Prieiga per internetą: https://www.researchgate.net/publication/328657267

[FWC14] H. Flora, X. Wang, S. V. Chande. An Investigation into Mobile Application Development Processes: Challenges and Best Practices. 2014. Prieiga per internetą: https://www.researchgate.net/publication/266743674

[IDB13] I. Dalmasso, S. K. Datta, C. Bonnet, N. Nikaein. Survey, Comparison and Evaluation of Cross Platform Mobile Application Development Tools. 2013. Prieiga per internetą: https://www.researchgate.net/publication/261378649

[IM10] R. Islam, T. Mazumber. Mobile application and its global impact. 2010. Prieiga per internetą: https://www.researchgate.net/publication/308022297

[KEL15] S. Kimak, J. Ellman, C. Laing. Some Potential Issues with the Security of HTML5 IndexedDB. 2016. Prieiga per internetą: https://www.researchgate.net/publication/281066023

[Kim15] S. Kimak. The role of HTML5 IndexedDB, the past, present and future. 2015. Prieiga per internetą: https://www.researchgate.net/publication/304410447

[LLN16] M. Latif, Y. Lakhrissi, E. H. Nfaoui, N. Es-sbai. Cross platform approach for mobile aplication development: a survey. 2016. Prieiga per internetą: https://www.researchgate.net/publication/304371091

[MCL18] N. Murray, F. Coury, A. Lerner, C. Taborda. Ng-book. The complete book on Angular. 2018.

[MS12] Z. Mccormick, D. C. Schmidt. Data synchronization patterns in mobile aplication design. 2012. Prieiga per internetą: https://www.dre.vanderbilt.edu/~schmidt/PDF/PatternPaperv11.pdf

[NM16] R.C. Nickerson, F. B. Mourate-Dussault. Selecting a Stored Data Approach for Mobile Apps. 2016. Prieiga per internetą: https://www.mdpi.com/0718-1876/11/3/16

[RGK19] F. Rakowski, K. Grzybowska, P. Karwatka, A. Kwiecien. Elektroninė knyga The PWA Book. 2019. Prieiga per internetą: https://divante.com/pwabook/chapter/.

[SGS13] M. S. Shriwas, N. Gupta, Dr. A. Sinhal. Efficient Method for Backup and Restore Data in Android. 2013. Prieiga per internetą: https://www.researchgate.net/publication/261231408

[SML20] R. Sjodin, R. Mason, M. Lofty. Integrating Cloud-based NoSQL Technology and Mobile Phone Relational Databases. 2020. Prieiga per internetą: http://www.ccsc.org/publications/journals/RM2020.pdf#page=24

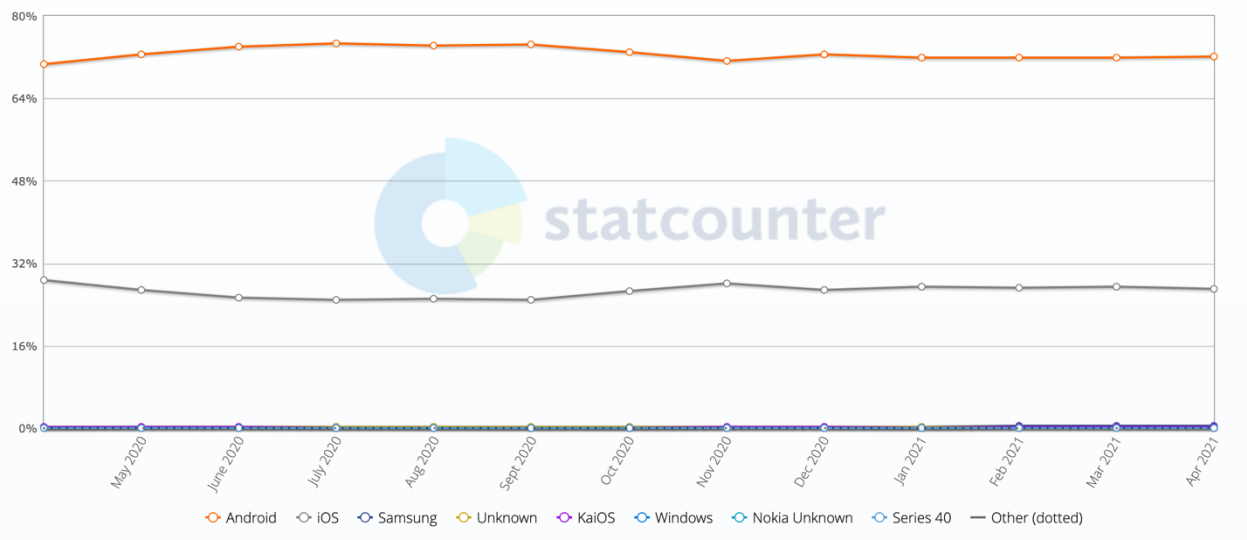
[Šmi17] L. Šmitalova. NativeScript application for Continuous Improvement of Software Engineer's skills. Bachelors thesis. 2017. Prieiga per internetą: https://is.muni.cz/th/j0k8a/thesis.pdf

[Ves17] D. Vesely. Analysis and experiments with NativeScript and React Native framework. Masters thesis. 2017. Prieiga per internetą: https://is.muni.cz/th/bkg22/thesis.pdf

[Was10] I. Wasserman. Software Engineering Issues for Mobile Application Development. 2010. Prieiga per internetą: https://www.researchgate.net/publication/221560176

Priedai

1 priedas



19 pav. Rinkos dalis pagal mobiliųjų telefonų operacinę sistemą. Šaltinis: https://gs.statcounter.com/os-market-share/mobile/worldwide

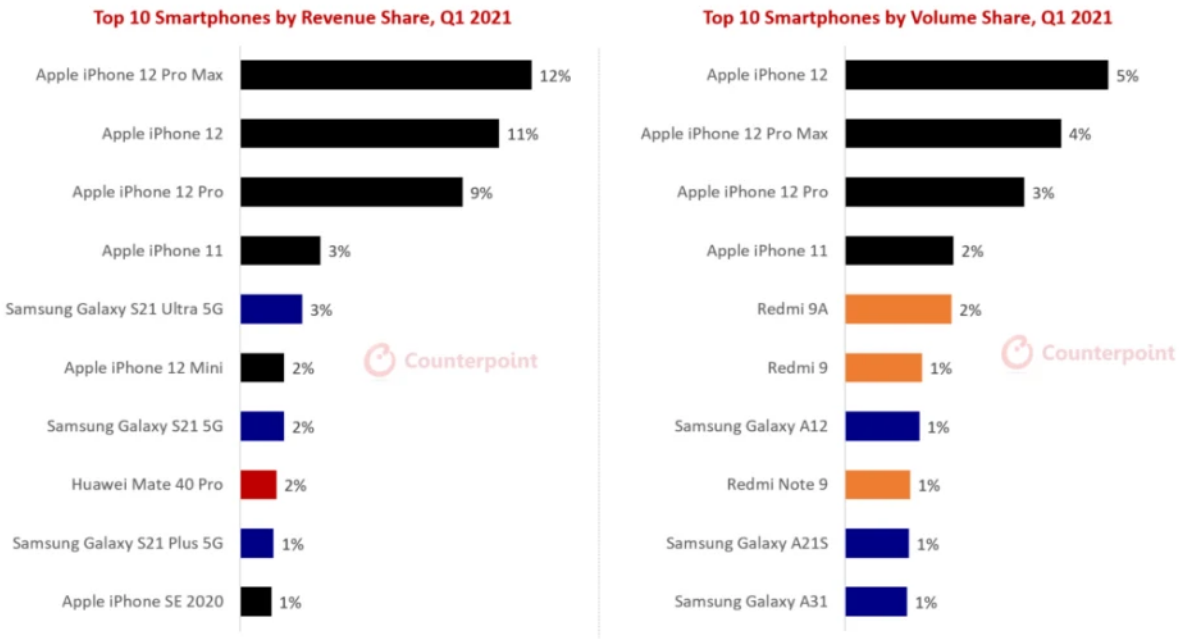
2 priedas

Props

Name	Type	Description
`text`	`String`	Sets the label of the button.
`textWrap`	`Boolean`	Gets or sets whether the widget wraps the text of the label. Useful for longer labels. Default value is `false`.
`isEnabled`	`Boolean`	Make the button disabled or enabled. A disabled button is unusable and un-clickable. Default value is `true`.
`...Inherited`	`Inherited`	Additional inherited properties not shown. Refer to the API Reference

20 pav. NativeScript vartotojo sąsajos elemento naudojimo instrukcija. Šaltinis: https://docs.nativescript.org/ui-and-styling.html#button

3 priedas



21 pav. Parduoduotų telefonų 2021 metai statistika. Šaltinis: https://www.phonearena.com/news/best-selling-phones-q1-2021_id132273