Spring 4-2021

ZHVILLIMI I MOBILE APLIKACIONEVE DUKE PËRDORUR CROSS-PLATFORM TEKNOLOGJI

Gent Bicaj
University for Business and Technology - UBT

Programi për Shkenca Kompjuterike dhe Inxhinierise

ZHVILLIMI I MOBILE APLIKACIONEVE DUKE PËRDORUR CROSS-PLATFORM TEKNOLOGJI

Shkalla Bachelor

Gent Bicaj

Prill / 2021
Prishtinë

Programi për Shkenca Kompjuterike dhe Inxhinierise

Punim Diplome
Viti akademik 2015-2016

Gent Bicaj

ZHVILLIMI I MOBILE APLIKACIONEVE DUKE PËRDORUR CROSS-PLATFORM TEKNOLOGJI

Mentori: MSc. Blerim Zylfiu

Prill / 2021

Ky punim është përpiluar dhe dorëzuar në përmbushjen e kërkesave të pjesshme për Shkallën Bachelor

ABSTRAKT

Zhvillimi dhe mirëmbajtja e aplikacioneve mobile për shumë platforma në të njëjtën kohë mund të jetë e gjatë. Meqenëse aplikacionet duhet të zhvillohen me teknologjitë e zhvillimit të secilës platformë, dizajnerët e softuerëve duhet të zhvillojnë dhe mbajnë shumë kode burimore të ndara për një aplikacion. Kjo ka shtyrë shumë kompani dhe komunitete të krijojnë framework dhe mjete të reja zhvillimore për aplikacionet mobile të cilat lejojnë zhvilluesit të shkruajnë dhe mbajnë një bazë të vetme të kodit. Baza e kodit më pas përpilohet në aplikacione për secilën platformë dhe aplikacionet pastaj mund të publikohen.

Ky punim përqendrohet në teknologjitë e zhvillimit cross-platform, posaçërisht në ato që përdorin funksionalitetin e secilës platformë për të ofruar një përvojë përdoruesit. Qëllimi i këtij studimi është të zbulojë nëse teknologjitë cross-platform janë një mundësi e mirë për zhvillimin modern të aplikacioneve mobile si nga përvoja e zhvillimit ashtu edhe nga përvoja e përdoruesit.

Gjetja kryesore e hulumtimit ishte React Native dhe teknologjitë e tjera cross-platform mund të jenë një mundësi praktike për zhvillimin e aplikacioneve mobile, është e paqartë kur teknologjitë cross-platform kanë një avantazh të rëndësishëm. Në aplikacione të vogla ndryshimi midis të dyve nuk është i theksuar dhe në aplikacione më komplekse teknologjitë cross-platform nuk kanë domosdoshmërisht shumë përfitime në krahasim me teknologjitë vendase.

MIRËNJOHJE/FALENDERIME

Shpreh mirënjohjet e mia më të thella ndaj të gjithë mësuesëve, profesorëve dhe edukatorëve që më kanë përcjell gjatë gjithë jetës. Mirënjohje dhe falenderimin me te vecant ia kushtoj familjes time që më kanë përkrah dhe mirëkuptuar në cdo hap të jetës. Për të mos harruar falenderimin dhe mirënjohjen ndaj prof. Blerim Zylfiu që me shumë zell dhe pa përtesë më ka përkrahur dhe më ka udhëzuar gjatë kohës të shkrimit të kësaj teme.

PËRMBAJTJA

LISTA E FIGURAVE

1. HYRJE

Tregu i aplikacioneve mobile është bërë një nga degët më të mëdha të industrisë së softverit që nga fillimi i tij. Sipas raporteve nga We Are Social dhe Hootsuite, në vitin 2020, 4.2 miliardë përdorues unikë zotëronin smartphone në mbarë botën [1]. Në vitin 2019 65% e të gjithë trafikut në internet erdhi nga smartphone [1] dhe një raport tjetër nga App Annie thotë se në vitin 2020 numri i përgjithshëm i shkarkimeve të aplikacioneve mobile ishte mbi 185 miliardë [2].

Nevoja për zhvillimin e aplikacioneve mobile vazhdon të rritet por, në të njëjtën kohë, rasti unik i përdorimit për këto aplikacione paraqet disa probleme për zhvilluesit sepse secili sistem operativ i smartphone-it ka platformë të veten, e pavarur nga të tjerët për sa i përket ekosistemit të aplikimit , përvojën e synuar të përdoruesit dhe teknologjitë e përdorura për zhvillim. Meqenëse baza e mundshme e përdoruesve për shumicën e aplikacioneve mobile mbulon shumë platforma, zhvilluesit duhet të zhvillojnë aplikacione të veçanta për secilën platformë, duke ndjekur udhëzimet e dizajnit të platformave dhe duke përdorur teknologjitë e tyre të zhvillimit.

Kjo kërkesë për ekspertizë në teknologjitë e shumta zhvillimore dhe arkitekturat e aplikacioneve ka krijuar një nevojë për mjete zhvillimi që thjeshtojnë procesin e zhvillimit të aplikacioneve mobile. Meqenëse pengesa kryesore është të krijosh një aplikacion të plotë veçmas për secilën platformë, janë shfaqur teknologji të reja që lejojnë zhvilluesit të ndajnë pjesë të bazës kodike midis platformave. Këto të ashtuquajtura teknologji cross-platform premtojnë të zgjidhin çështjet e zhvillimit të aplikacioneve, por aktualisht nuk ka një konsensus të qartë nëse ata e përmbushin atë premtim.

Qëllimi i këtij studimi është të përcaktojë nëse React Native dhe cross-platform framework-at në përgjithësi, janë një mundësi e vlefshme për zhvillim të aplikacioneve. Platformat mobile të përfshira në këtë hulumtim janë Android (Google) dhe iOS (Apple). Platforma të tjera ekzistojnë, por këto dy mbulojnë shumicën dërrmuese të përdoruesve. Qasje të ndryshme për zhvillimin e aplikacioneve do të prezantohen dhe krahasohen përmes një rishikimi të literaturës dhe pas kësaj një aplikacion i vogël do të dizajnohet dhe zhvillohet me React Native. Përvoja e zhvillimit dhe aplikimi përfundimtar do të vlerësohen në krahasim me situata të ngjashme për të parë nëse lindin ndryshime të rëndësishme.

2. DEKLARIMI I PROBLEMIT

Zhvillimi i aplikacioneve është një degë e zhvillimit të softverit që përqendrohet në aplikacionet e krijuara për pajisjet mobile. Rëndësia e disa veçorive është më e theksuar sesa në zhvillimin e softverit në desktop ose në web. Gjegjësisht efikasiteti i energjisë dhe aftësia për të punuar brenda kufizimeve të pajisjeve të smartphone-ve, veçanërisht madhësia e vogël e ekranit, janë të rëndësishme, por mbi të gjitha përvoja e përdoruesit bëhet e rëndësishme.

Përdorshmëria është sigurisht e rëndësishme sa herë që zhvillon një softuer por me aplikacionet mobile "pamja dhe ndjesia" e aplikacionit është një nga shqetësimet më të mëdha në çdo projekt. Çdo platformë ka një udhëzues të dizajnit dhe zhvilluesit e aplikacioneve inkurajohen të hartojnë përvojën e përdoruesit të aplikacioneve të tyre duke ndjekur këto udhëzues sa më afër që të jetë e mundur. Kjo rezulton në aplikacione që përdoruesi nuk duhet të mësojë t'i përdorë, por përkundrazi ndjehen intuitiv dhe të njohur sepse përdoruesi tashmë ka mësuar projektin e platformës gjatë përdorimit të sistemit operativ dhe aplikacioneve të para-instaluara.

Përvoja e përdoruesit është gjithashtu një nga ndryshimet më të mëdha midis një aplikacioni mobile dhe një aplikacioni modern ueb. Me evolucionin e aplikacioneve të përgjegjshme në internet që përshtaten në një ekran të çdo madhësie, kufiri midis një aplikacioni mobile dhe një aplikacioni web është bërë më pak i qartë. Ndonjëherë mund të jetë një mundësi më e mirë për t'i shërbyer përdoruesit një aplikacioni web sesa t'i bësh ata të shkarkojnë një aplikacion nga tregu i aplikacioneve të platformës. Sidoqoftë, në disa raste, një aplikacion mobile ka përparësi ndaj homologut të saj web dhe nëse aplikacioni kërkon përdorimin e veçorive të pajisjeve të tilla si kamera ose akselerometri, mbështetja për ata në aplikacionet web nuk është mjaft e pjekur në këtë pikë.

Ngjashmëria midis aplikacioneve web dhe mobile ka çuar gjithashtu në një keqkuptim në lidhje me zhvillimin e aplikacioneve në mobile. Ata që nuk janë të njohur me zhvillimin nativ të telefonave mund të mendojnë se një aplikacion mobile duhet të shkruhet vetëm një herë dhe më pas mund të lëshohet për të gjitha platformat. Kjo tradicionalisht nuk është e vërtetë për aplikacionet mobile dhe madje edhe në rastin e zhvillimit cross-platform, jo gjithçka mund të ndahet.

3. SHQYRTIMI I LITERATURËS

3.1 Zhvillimi i mobile aplikacioneve duke përdorur native teknologjitë

Zhvillimi native i aplikacioneve mobile i referohet mënyrës tradicionale të zhvillimit të aplikacioneve, duke përdorur gjuhët programuese vendase të platformave dhe arkitekturat e programeve për të shkruar bazat e kodeve që më pas përpilohen në aplikacione vërtet vendase. Ndërsa kjo metodë është përdorur që nga fillimi i telefonave inteligjentë, mjetet dhe teknologjitë kanë evoluar gjatë viteve.

Për Android, gjuha native e programimit ka qenë Java ose C ++ [3] por që nga viti 2017 Android gjithashtu ka mbështetur Kotlin. Objektiv-C ishte gjuha e vetme për zhvillimin e iOS deri në vitin 2014 [4], kur Apple prezantoi gjuhën e tyre të programimit Swift për publikun [5].

Zhvillimi native i aplikacioneve i detyron zhvilluesit jo vetëm të mësojnë dhe të përdorin gjuhët vendase të programimit, por gjithashtu i shtyn zhvilluesit të përdorin arkitekturën e rekomanduar të programeve për secilën platformë.

Për iOS arkitektura e rekomanduar është Model-View-Controller (MVC), ndërsa Android aplikacionet zakonisht përdorin arkitekturën ModelView-ViewModel (MVVM) [4] [3]. Si pasojë, kodet native kanë shumë pak të përbashkëta me njëra-tjetrën, dhe shpesh njohja e native platforme ofron shumë pak ndihmë për të kuptuar tjetrën.

Argumenti më i fortë për fuqinë e zhvillimit native është përvoja e përdoruesit. Meqenëse zhvilluesit po shkruajnë kodin native, ata mund të përdorin API-të vendase (Application Programming Interfaces) dhe të mbështesin librari të platformave, si dhe të krijojnë paraqitjet e tyre të ndërfaqes së përdoruesit (UI) duke përdorur përbërësit native të UI të siguruar nga platforma. Si rezultat, pamja dhe ndjesia native e kërkuar është relativisht e lehtë për t'u arritur kur përdorni teknologjitë native të zhvillimit.

Çështja më e madhe me zhvillimin e aplikacioneve native është, sigurisht, bazat e kodeve të ndara. Zhvillimi i të njëjtit aplikacion për shumë platforma kërkon njohuri të metodave specifike të zhvillimit të platformës, udhëzuesve të dizajnit dhe teknologjive. Zhvillimi dhe mirëmbajtja e aplikacioneve native mobile për shumë platforma është e krahasueshme me ekzekutimin e shumë projekteve individuale të softuerit në të njëjtën kohë, kështu që shpesh kërkohen më shumë zhvillues dhe personel tjetër i projektit në krahasim me një projekt të vetëm aplikimi. Kjo rrit si koston e zhvillimit ashtu edhe mirëmbajtjen ndërsa rritet numri i platformave të mbështetura.

3.2 Zhvillimi i mobile aplikacioneve duke përdorur cross-platform teknologjitë

Cross-Platform zhvillimi në përgjithësi i referohet aktit të zhvillimit të softuerit të pavarur nga platforma. Ky lloj programi mund të ekzekutohet në pajisje të shumta dhe sisteme operative dhe duhet të funksionojë në të njëjtën mënyrë, pavarësisht nga ambienti në të cilin po ekzekutohet.
Zhvillimi në cross-platform ka ekzistuar për pothuajse aq kohë sa vetë smartphonët. IPhone origjinal u lëshua në 2007 [6] dhe framework-at e parë cross-platform për zhvillimin e mobile aplikacioneve u lëshuan në fillim të vitit 2008 [7].
Sidoqoftë, shumica e teknologjive të hershme të zhvillimit cross-platform për aplikacionet mobile ishin lloje të ndryshëm mbështjellësish për web aplikacione, duke lejuar që programi kompjuterik i shkruar në HTML, CSS dhe JavaScript të ekzekutohet si një aplikacion individual [7].
Më vonë, teknologjitë më moderne të zhvillimit filluan të shfaqeshin, duke i lejuar zhvilluesit të krijojnë cross-platform mobile aplikacione të cilat u ngjanin shumë më shumë homologëve të tyre vendas. Nevoja për një përvojë të përdoruesit vendas është po aq e fortë në aplikacionet cross-platform sa aplikacionet vendase, kështu që zhvilluesit duhet ende të jenë të njohur me platformat me të cilat po punojnë. Mund të bëhet një argument që zhvilluesit cross-platform nuk kanë nevojë për një njohuri kaq të thellë të secilës platformë si zhvilluesit vendas, por disa zhvillime specifike të platformës duhet të bëhen nëse qëllimi është të fitoni një përvojë.
Cross-platform aplikacionet mund të kategorizohen në mënyra të ndryshme bazuar në mënyrën se si janë zhvilluar. Një studim mbi qasjet e zhvillimit në cross-platform, i botuar në 2012, ndan aplikacionet mobile cross-platform në katër kategori kryesore: mobile web aplikacionet, aplikacionet hibride, cross-platform aplikacione dhe aplikacione të interpretuara [8].

3.2.1 Hybrid Development Frameworks

Framework-at hibride janë një koleksion i teknologjive të zhvillimit të web aplikacioneve. Dallimi kryesor midis një web aplikacioni dhe një aplikacioni hibrid është se ndërsa një web aplikacion do të përdoret gjithmonë përmes një shfletuesi, framework-at hibride prodhojnë aplikacione të cilat janë instaluar gjithmonë në smartphone [8].
Karakteristikat dhe interface-i i përdoruesit të një aplikacioni hibrid janë zhvilluar me HTML, CSS dhe JavaScript, ashtu si çdo web aplikacion. Framework-u hibrid pastaj e mbështjell web aplikacionin në një konteiner që krijon një aplikacion të instalueshëm për platformat e ndryshme mobile. Disa framework hibride veprojnë si një mbështjellës shumë i hollë për web aplikacionet,

duke mos ofruar asnjë qasje në veçoritë e pajisjeve të tilla si kamera ose vendndodhja. Kjo ishte veçanërisht e vërtetë për framework-at më të hershme cross-platform [7], por shumica e opsioneve moderne lejojnë që zhvilluesi të përdorë edhe veçoritë të pajisjeve [8].

Pothuajse të gjitha framework-at hibride, të tilla si PhoneGap, Apache Cordova dhe Ionic, kanë të njëjtën pengesë: përvojën e përdoruesit. Një mobile aplikacion supozohet të duket dhe të ndjehet sikurse në platformat vendase, por ky qëllim është jashtëzakonisht i vështirë për tu arritur kur në fund të fundit jeni duke zhvilluar një web aplikacion. Interface-at e ndara të përdoruesit sigurisht që mund të zhvillohen për platforma të ndryshme, por kjo plotësisht e mposht qëllimin e zhvillimit cross-platform. Ripërdorimi i kodit bëhet margjinal dhe performanca e aplikacionit do të jetë gjithmonë më e keqe sesa një aplikacion i zhvilluar në platformë të veçantë [8].

Framework-at hibride nuk kanë ndryshuar shumë gjatë gjithë këtyre viteve dhe ende vuajnë nga të njëjtat disavantazhe si në vitin 2012. Meqenëse përvoja e përdoruesit të aplikacionit nuk përputhet me pritjet e vendosura nga kanali i shpërndarjes, aplikacionet që janë zhvilluar me framework-a hibride shpesh konsiderohen inferiore ndaj web aplikacioneve [8].

3.2.2 Cross-Compiled Native Frameworks

Native framework-at lejojnë zhvilluesit të shkruajnë kodin burimor në një gjuhë të përbashkët programimi, e cila më pas përpilohet në binarët vendas për secilën platformë duke përdorur një cross-compiler. Përgjegjësia e cross-compiler është të përshkruaj cilët përbërës të ndërfaqes së përdoruesit vendas dhe tiparet e platformës native duhet të përdoren [8].

Një aplikacion i zhvilluar me një cross-compiled framework mund të jetë shumë afër një aplikacioni mobile plotësisht vendas, por përjashtime ekzistojnë. Përparësitë kryesore të framework-ave cross-compiled janë performanca dhe përvoja e përdoruesit. Interface i përdoruesit jepet në secilën pajisje duke përdorur përbërës vendas, kështu që aplikacioni do të duket vendas për sa kohë që zhvilluesit marrin parasysh udhëzimet për hartimin e platformës. Ndërsa asnjë framework Cross-compile nuk mund të përputhet mjaftueshëm me performancën e aplikacioneve plotësisht vendase, testet e performancës tregojnë se frameworkat cross-compile arrijnë performancë adekuate në shumicën e situatave [9] [10]. Shembuj të cross-compiled framework janë Microsoft's Xamarin dhe Xamarin.Forms dhe Google’s Flutter [11]. Aplikacionet Xamarin dhe Xamarin.Format janë shkruar në C # dhe një kombinim i C# dhe XAM përkatësisht. Flutter aplikacionet janë shkruar në Dart. Flutteri është gjithashtu një përjashtim nga framework-at cross-compiled për sa i përket implementimit të interface-it të përdoruesit. Flutter është një framework

relativisht i ri me lëshimin e tij fillestar në 2017 dhe sapo ka arritur në daljen e parë të qëndrueshme të Flutter 1.0 [12]. Dallimi i dukshëm midis tij dhe framework-ave të tjerë është se Flutter nuk përdor ndonjë përbërës vendas të interface-it së përdoruesit për të dhënë pikëpamjet e tij. Në vend të kësaj, ajo ka një librari me miniaplikacione të implementuara nga ekipi i Flutter [13]. Kjo lejon që Flutter të abstragojë përbërësit në të njëjtën mënyrë, pavarësisht nga platforma, pasi nuk mbështetet në zbatimin dhe arkitekturën specifike të platformës. Sidoqoftë, pengesa për këtë qasje është që zhvilluesit duhet të mbështeten te mirëmbajtësit e Flutter për të mbajtur të azhurnuara të gjitha widgetet e tyre. Niveli i abstraksionit të interface-it së përdoruesit është rritur, dhe si i tillë, ripërdorimi i kodit në interface-at e përdoruesit është bërë më i lehtë. Disa karakteristika të platformës vendase janë abstraguar mjaftueshëm për të lejuar edhe ripërdorimin e kodit, por tipare më të ndërlikuara kërkojnë ende zbatim specifik të platformës.

3.2.3 Native Scripting Framework-at

Framework-at origjinale të skriptimit përdorin një interpretues për të ekzekutuar kodin e shkruar gjatë kohës së ekzekutimit. Çdo gjuhë skriptimi mund të përdoret, por framework-at më të zakonshme moderne përdorin një variant të JavaScript, gjuha më e njohur për zhvillimin e web aplikacioneve [14].

Aplikacionet skriptuese përdorin gjuhën e skriptimit për logjikën e tyre të biznesit dhe dizajnin e interface-it së përdoruesit, por përdorin përbërësit e interface-it së përdoruesit të platformës vendase për dhënien e aplikacionit dhe thirrin API-të e platformës vendase për të gjitha tiparet e mundshme. Përparësitë e kësaj qasjeje janë të ngjashme me përparësitë e framework-: interface i përdoruesit do të duket vendase në platformë nëse dizajnohet siç duhet, dhe performanca e aplikacionit është shumë afër një aplikacioni vendas. Kjo rezulton në një përvojë të përdoruesit shumë afër asaj të aplikacioneve vendase pa pasur nevojë për një kuptim të thellë të vetë platformave.

Pengesat më të mëdha të framework-ave të skriptimit vendas janë gjuha e përdorur dhe zbatimi i veçorive vendase. JavaScript dhe shumë gjuhë të tjera të skriptimit nuk janë të përshtatshme në mënyrë ideale për të gjitha rastet e përdorimit për shkak të veçorive të tyre të qenësishme si gjuhë skriptimi, përkatësisht karakteristikave dinamike dhe përkufizimeve të tipit të dobët. Kjo shpesh do të çojë në gabime në kod duke u vërejtur vetëm gjatë kohës së ekzekutimit, megjithëse disa nga këto çështje mund të zgjidhen duke përdorur librari shtesë mbështetëse të krijuara për gjuhët.

3.3 React Native

React Native është një framework origjinal i skriptimit për zhvillimin e cross-platform mobile aplikacioneve. Framework u krijua fillimisht nga Facebook dhe u lëshua në 2015 vetëm për iOS, por një komunitet aktiv që atëherë ka shtuar mbështetjen për Android dhe ka bërë shumë kontribute të tjera në projektin me burim të hapur. Aplikacionet React Native zhvillohen duke përdorur JavaScript, më saktësisht EcmaScript 2015 (ES2015 ose ES6) [15]. EcmaScript është një superset i JavaScript, duke shtuar veçori të shumta në JavaScript dhe duke rregulluar disa nga gabimet e bëra fillimisht në specifikimet JavaScript. EcmaScript gjithashtu ka zëvendësuar JavaScript të pastër në web aplikacionet moderne dhe të gjithë shfletuesit modernë mbështesin një version të tij [16]. Përveç ES2015, React Native zbaton pjesë të versionit tjetër të EcmaScript, ES2016 ose ES7

3.3.1 Arkitektura

Arkitektura e React Native përbëhet nga një makinë virtuale JavaScript, ura React Native dhe modulet native. Kjo arkitekturë është paraqitur në Figurën 1. Kodi ekzekutohet në makinerinë virtuale JS së bashku me çdo librari të palëve të treta të përdorura. Thirrjet e modulit native drejtohen përmes urës së React Native në API-të vendase dhe librari të palëve të treta dhe rezultatet kalohen përmes urës nëse është e nevojshme. Kjo lejon përdorimin e JavaScript për të zhvilluar aplikacionin duke përdorur akoma përbërësit dhe tiparet e UI-ve native të platformave [17].

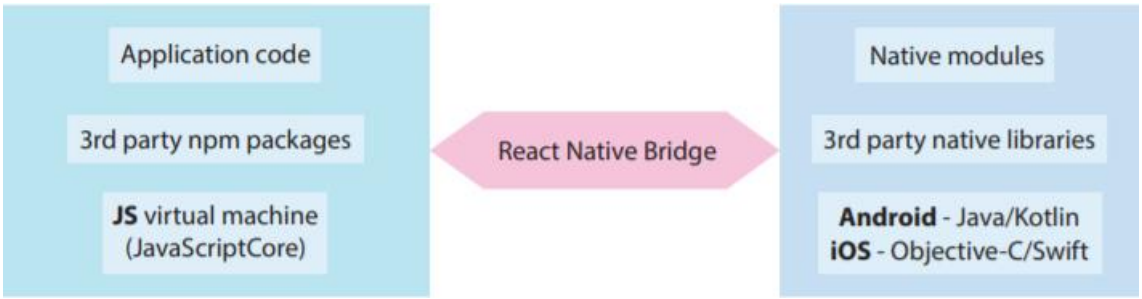

Figura 1- Arkitektura e React Native

Vlen të përmendet se React Native po pëson një ndryshim të arkitekturës në kohën e shkrimit të kësaj teme pasi arkitektura e re e React Native Fabric është në zhvillim e sipër dhe ka të ngjarë të dalë në 2019/2020 [18].

Arkitektura e JavaScript bazohet në React të Facebook, një framework i bazuar në përbërës për ndërtimin e interface-ave për web aplikacione. Elementi themelor i një React aplikacioni është një përbërës, i cili është një klasë individuale JavaScript që gjithmonë jep një pjesë të interface-it së përdoruesit. Komponenti mund të marrë gjithashtu veti (të njohura si rekuizita në React) dhe të

ketë një gjendje të brendshme. Më pas komponentë të shumtë kompozohen për të krijuar interface më komplekse të përdoruesit dhe të dhënat mund të kalojnë në zinxhirin e përbërësit si rekuizita, nga prindi tek fëmija [17].

Ndryshimi i të dhënave në një React aplikacion mund të jetë mjaft sfidues, pasi rrjedha e të dhënave është njëdrejtimëshe dhe të dhënat duhet të mutohen vetëm në nivelin më të lartë atje ku ekziston. Të dhënat e azhurnuara më pas duhet të përcillen si pjesë përbërëse të fëmijëve. Në aplikacionet e pastra React kjo arrihet përmes kalimit të funksioneve të kthimit në telefon të përbërësit të fëmijëve si rekuizita. Atëherë përbërësi i fëmijës mund të thërrasë një funksion kthyese për të mutuar të dhënat. Ndërsa përbërësi prind që ka gjendjen e nivelit më të lartë merr përgjigjen, ajo do të shndërrojë të dhënat dhe të dhënat e azhurnuara u kalojnë automatikisht komponentëve të fëmijëve. Zgjidhje të tjera ekzistojnë në formën e Node moduleve të siguruara nga Node Package Manager (npm). Një nga Modulet më të popullarizuara të Node për trajtimin e gjendjes së aplikacionit dhe mutacionit të të dhënave është Redux, i cili siguron një gjendje të centralizuar që çdo komponent mund të ketë qasje dhe thirrjen e funksioneve të ndryshimit. As React e as React Native nuk ofrojnë ndonjë zgjidhje për të naviguar midis pikëpamjeve të ndryshme në aplikacion. Përsëri, Node modulet ekzistojnë për të zgjidhur navigimin në mënyra të ndryshme, por një nga më të zakonshmit është React Navigation. React Navigation ofron lloje të ndryshme të navigimit, të tilla si navigimi në stack ose navigimi i tab-ave, për situata të ndryshme.

3.3.2 Karakteristikat

Shumica e tipareve të zakonshme janë zbatuar tashmë në React Native dhe mund të thirren direkt nga kodi JavaScript. Ndonjëherë edhe pse zhvilluesit duhet të zbatojnë një veçori më komplekse sesa mund të sigurojnë API-të React Native, siç është gjurmimi i akselemetrit në një proces. Në këto situata React Native lejon zhvilluesit të zbatojnë funksionalitetin e tyre në të njëjtën mënyrë si vetë implementimi i React Native. Krijohet një modul me një interface JavaScript që mund të thirret brenda kodit. Kodi specifik i platformës është shkruar për modulin në gjuhën programuese amtare të secilës platformë dhe urat React Native bën thirrjen nga JavaScript në modulet vendase.

3.4 Krahasimi i React Native me zhvillimin native

Për shkak të fushës së kufizuar të këtij punimi nuk mund të bëhej një krahasim i plotë midis zhvillimit React Native dhe zhvillimit Native. Në vend të kësaj, një aplikacion i vogël u dizenjua dhe u zhvillua me React Native. Përvoja e zhvillimit të React Native dhe çështjet e mundshme që ndodhën gjatë zhvillimit krahasohet me mënyrën se si ato situata do të ishin zgjidhur në një aplikacion native, ose nëse të njëjtat çështje do të kishin ekzistuar fare. Përvoja e përdoruesit e një aplikacioni React Native vlerësohet si përmes aplikacionit të testit, ashtu edhe studimeve paraprake. Aplikimi i testit i zhvilluar për këtë tezë është mjaft i thjeshtë, kështu që nuk do të gjenden ndryshime dramatike në përdorshmërinë ose performancën në krahasim me një aplikacion native.

3.4.1 Dizajnimi dhe zhvillimi i aplikacionit për testim

Koncepti për aplikacionin testues është një aplikacion që lejon përdoruesit të skanojnë barkodet, të kërkojë në një bazë të dhënash për informacion në lidhje me produktin e skenuar, të shndërrojë informacionin në një citim të formatuar dhe të lejojë përdoruesin të ndajë citimin si tekst përmes cilitdo kanal që ata e gjejnë më shumë të përshtatshëm.
Kërkesat themelore për aplikacion ishin të thjeshta. Kur një përdorues hap aplikacionin, në fillim përdoruesit i kërkohet leje për të përdorur kamerën e pajisjes. Përdoruesi më pas mund të drejtohet në pamjen e skanimit ku një barkod mund të vendoset në pamjen e kamerës duke përdorur pamjen e kamerës. Me skanimin e një barkodi, aplikacioni lejon përdoruesin të konfirmojë që barkodi është i saktë. Nëse është, aplikacioni vazhdon në një pamje kërkimi dhe fillon të kërkojë në bazën e të dhënave për rezultatin që përputhen me barkodin. Pamja e kërkimit tregon statusin e kërkimit dhe rezultatet përfundimtare, duke lejuar përdoruesin të zgjedhë se cili nga rezultatet ishte i saktë. Sapo përdoruesi të zgjedhë një nga rezultatet, aplikacioni shkon në një pamje ndarëse ku krijon një citim të formatuar dhe lejon përdoruesin ta ndajë atë përmes Share API të platformës.

Disa kufizime ishin vendosur për aplikacionin përpara zhvillimit të tij:

- Citimi është formatuar në formatin BibTeX
- Përdoruesit nuk i lejohet të redaktojë citimin në aplikacion sepse duhet të zhvillohet një formë redaktimi shumë komplekse, ose aplikacioni do të rrezikonte të linte që përdoruesi të prishte formatimin e citimit.

- Google Books është përdorur si bazë e shënimeve

Teknologjia kryesore e përdorur për zhvillimin e aplikacionit është React Native, por shumë shpesh zhvilluesit shtojnë librari dhe mjete të shumta të palëve të treta për të komplimentuar React Native dhe për ta bërë më të lehtë zhvillimin. Në këtë aplikacion, teknologjia e përdorur është e paraqitur më poshtë:

- React Native për pjesën më të madhe të aplikacionit
- Libraria Redux për kontrollimin e gjendjes dhe kalimin e të dhënave dhe veprimeve si rekuizione te komponenti
- Libraria React Navigation
- Expo barcode librarinë
- TypeScript

Kodi burimor i zhvilluar me këtë teknologjie është larg nga pure React Native por është gjithashtu më tregues i një shembulli të botës reale të zhvillimit me React Native. Sidomos pasi të dhënat që merren me aplikimin bëhen jo-parëndësishme, shumë prej këtyre librarive bëhen thelbësore për të shmangur gabimet në zhvillim. Vetë zhvillimi i aplikacionit u bë duke përdorur metodologjinë agile: modulet e vogla të aplikacionit u zhvilluan një nga një, së pari në një Produkt Minimal të Qëndrueshëm (MVP), dhe më pas u zgjeruan pasi modulet e tjerë të aplikacionit kërkonin më shumë karakteristika prej tyre. Ndërsa React Native lehtëson zhvillimin shumë të shpejtë dhe përsëritjet e provave, prototipi ishte i vazhdueshëm, por procesi i zhvillimit mund të ndahet në pesë pika dhe në prototipa përkatës të cilët bazohen në versionet e mëparshme, të renditura më poshtë:

- Pamja kryesore me kontrollimin e lejes së kamerës, navigimi në pamjen e skanerit dhe skanimi i një barkodi
- Skanimi i një barkodi dhe lejimi i përdoruesit ta konfirmojë atë, duke kërkuar në Google Books API
- Kërkimi i pamjes dhe navigimi tek ai nga pamja e skanerit, Kërkimi i Google Books API për barkodin, lejimi i përdoruesit të zgjedhë se cili rezultat do të përdorët dhe krijimi i citimit

- Pamja e citimit dhe navigimi në të, implementimi i Share API, stilimi i interface-it së përdoruesit për Android dhe ndërtimi i parë i lëshimit për Android
- Përshtatja e disa stileve për iOS, shtimi i disa grafikëve dhe finalizimi i aplikacionit për sa i përket kësaj teme, dhe një ndërtim i dytë për Android.

Versionet Android të aplikacionit u testuan në një pajisje dhe u ndërtuan paketa APK (Android Application Package) për të instaluar aplikacionin në pajisje të shumta. Versionet e iOS u testuan në emulators, por asnjë pajisje provë nuk ishte në dispozicion. Asnjë paketë IPA (iOS App Store Package) nuk u ndërtua sepse Apple kërkon një shume për të paguar në programin e tyre të zhvilluesit për të qenë në gjendje të ndërtojmë aplikacione.

Google Books API doli të ishte më e vështirë për të punuar sesa ajo që dukej fillimisht. Pengesa kryesore e API është se nuk garanton ekzistencën e ndonjë informacioni jetik për funksionalitetin e aplikacionit. Për shembull, informacioni i autorit të disa librave nuk ekzistonte në të dhënat e kthyera nga API edhe pse faqja përkatëse në internet e Librave të Google e tregonte qartë atë. Kjo do të thoshte se informacioni nga API nuk mund të mbështetej për krijimin e citimeve dhe në vend të kësaj citimet duhej të merreshin nga një API i veçantë që Google përdor në shërbimin e tyre Libra për eksport citimesh. Të gjitha çështjet me Google Books mund të ishin shmangur duke kaluar në një API më të përshtatshëm si ai i ofruar nga ISBNdb, por fatkeqësisht asnjë API i lirë mund të dukej më mirë.

3.5 Përvoja e Zhvillimit

Është shumë e lehtë të vendosni mjedisin e zhvillimit duke përdorur mjetet e Expo dhe është po aq e lehtë të merrni versionin e zhvillimit të aplikacionit në një pajisje provë ose një emulator. Marrja e ekzekutimit të aplikacionit fillestar "Hello world" është një proces i shpejtë, por instalimi i të gjithë bibliotekave mbështetëse npm dhe bërja e tyre për të punuar në aplikacion kërkon shumë më shumë përpjekje.

Zhvillimi i një aplikacioni me React Native karakterizohet nga prototipi i shpejtë. Duket se shumë nga përparësitë e kornizës përqendrohen në bërjen më të lehtë dhe më të shpejtë të testimit të një aplikacioni të azhurnuar. Një nga pikat kryesore të shitjes së React Native dhe shumë kornizave të tjera të skenarit vendas është ringarkimi i drejtpërdrejtë, i njohur gjithashtu si ringarkimi i nxehtë.

Gjatë zhvillimit të aplikacionit, një pajisje ose një emulator është i lidhur me një server zhvillimi që ekzekuton aplikacionin. Kurdoherë që një nga skedarët e kodit burim JavaScript ndryshon, serveri i zhvillimit do të ribundojë kodin JavaScript dhe do t'ia dërgojë pajisjes dhe pajisja rimbush menjëherë aplikacionin me kodin e azhurnuar. Kjo zvogëlon sasinë e kohës së shpenzuar në pritje të një ndërtimi për të përfunduar në mënyrë që të provojë një ndryshim të ri, por gjithashtu inkurajon zhvilluesin drejt përdorimit të një stili të zhvillimit të provave dhe gabimeve ku aplikacioni rindërtohet pas çdo ndryshimi të vogël. Ky mund të jetë një efekt i mirë ose i keq i funksionit.

Facebook dhe shumë zhvillues të tjerë të teknologjisë cross-platform tregtojnë framework-at e tyre me një parullë që mund të përmblidhet si "mëso një herë, shkruaj kudo". Kjo do të thotë që zhvilluesi duhet të ketë nevojë vetëm për njohjen e një gjuhe programimi për të zhvilluar një aplikacion. Ky pretendim duket të jetë i vërtetë të paktën në rastin e aplikimit të provës dhe aplikacioneve të tjera me kompleksitet të ngjashëm. Nëse komponentët vendas nuk duhet të zhvillohen, aplikacionet React Native mund të shkruhen thjesht në JavaScript gjë që mund ta bëjë këtë teknologji tërheqëse për zhvilluesit e internetit që kërkojnë të futen në zhvillimin e aplikacioneve.

Një pyetje tjetër kryesore, dhe një çështje që ka edhe më shumë rëndësi për qëndrueshmërinë financiare të zhvillimit të lëvizshëm cross-platform, është ripërdorimi i kodit. Facebook pretendon se rreth 75% e kodit React Native mund të ndahet në të gjithë platformat edhe kur zhvillohen aplikacione që janë krijuar për të kryer sa më natyrshëm që të jetë e mundur.

Në aplikacionin e provës, ripërdorimi i kodit është i shkëlqyeshëm sepse kodi specifik i platformës duhej të shkruhej vetëm për një në pesë përbërës të UI-së dhe logjika e biznesit të aplikacionit, në thelb e gjithë logjika dhe përpunimi i të dhënave, ndahet plotësisht. Niclas Hansson dhe Tomas Vidhall gjetën në hulumtimin e tyre se ky pretendim mund të konsiderohet i vërtetë ose i rremë në varësi të mënyrës se si e shikojmë ne shprehjen. Në të gjithë projektin e tyre 62% e kodit u nda, por nëse shikojmë versionet Android dhe iOS si aplikacione të veçanta, mbi 75% e kodit të përdorur për secilën platformë u nda nëpër platforma. Nëse një aplikacion i ngjashëm në internet do të zhvillohej duke përdorur librarinë Redux për menaxhimin e gjendjeve, logjika e biznesit mund të ndahej në të gjithë projektet pasi në asnjë mënyrë nuk lidhet me shtresën e ndërfaqes së përdoruesit, e cila mund të konsiderohet si një bonus.

Pjesa më interesante e projektit dhe ajo që mund të tregojë shumë më tepër për React Native në përgjithësi, ishte implementimi i skanerit të barkodit. Fillimisht kjo mendohej të ishte sfida më e

madhe e projektit, por duke bërë më shumë hulumtime mbi këtë temë, dukej sikur Expo tashmë ka një librari që funksionon pikërisht atë që ishte e nevojshme. U përdor libraria Expo dhe në fund skaneri i barkodit doli të ishte një nga tiparet më të lehta të aplikacionit për sa i përket implementimit. Skanuesi i barkodit gjithashtu zbulon atë që mund të konsiderohet një forcë ose një e metë në hartimin e React Native, përkatësisht librarive të palëve të treta të mbështetjes. Skanuesi i barkodit ishte i lehtë për tu zbatuar sepse Expo tashmë kishte zhvilluar një librari që funksiononte mirë për këtë qëllim dhe e kishte botuar atë si burim të hapur. Nëse nuk do të ishte ky rasti, modulet vendase do të duhej të zhvilloheshin për të dy iOS dhe Android për të qenë në gjendje të përdorin veçoritë vendase dhe si i tillë projekti do të kërkonte njohuri të paktën tre gjuhësh të ndryshme programimi: JavaScript për React Native , Swift ose Objektivi-C për iOS dhe Java ose Kotlin për Android. Vështirësia rritet më tej sepse do të nevojiten edhe njohuritë për metodën e preferuar të secilës platformë për skanimin e barkodit. Jo vetëm kaq, krijimi i një moduli si ky do të kërkonte gjithashtu njohuri të arkitekturës së React Native. Kjo menjëherë do të mohonte një nga pikat e forta më të mëdha të teknologjive ndër-platformare.

Çështje të tjera që shfaqen kur përdoren librari të palëve të treta janë mirëmbajtja dhe jetëgjatësia. Apple dhe Google gjithmonë i mbajnë SDK të azhurnuara me versionet më të fundit të sistemit operativ dhe ato lëshojnë pamje paraprake të zhvilluesve përpara një azhurnimi të madh të sistemit operativ në mënyrë që zhvilluesit e aplikacioneve vendase të jenë në gjendje të mbajnë softuerin e tyre të azhurnuar. Me teknologji cross-platform ne nuk presim që Apple dhe Google ta bëjnë këtë, por gjithashtu ne u besojmë autorëve ndër-platformë që të mbajnë kornizat e tyre të azhurnuara dhe ne u besojmë palëve të treta që të mbajnë libraritë e tyre të azhurnuara. Ndonjëherë kjo nuk ndodh, dhe kur një azhurnim i madh godet konsumatorët, aplikacionet do të pushojnë së punuari derisa dikush tjetër të rregullojë problemet brenda librarisë ose framework-ut të tyre.

Kjo ndodhi me React Native dhe Android 8.0 (Oreo) në 2017 kështu që nuk është vetëm një kërcënim hipotetik. Matias Martinez dhe Sylvain Lecomte gjithashtu zbuluan këto çështje në hulumtimet e tyre mbi cilësinë e aplikacioneve cross-platform.

Libraritë e palëve të treta mund të përdorin gjithashtu veçori të amortizuara për të arritur atë që duan, të tilla si skanimi i barkodit. Expo ka bërë mirë në këtë drejtim dhe librarive të tyre BarCodeScanner përdor më të rejat Google Machine Vision API dhe Apple AVFoundation për të bërë skanimin e tyre, por zhvilluesit mund të shpresojnë që Expo të vazhdojë mirëmbajtjen dhe zbatimin e API-ve të reja edhe në të ardhmen. Libraritë e palëve të treta që përdorin veçori të

amortizuara mund të bëhen një problem kryesor gjatë një azhurnimi të sistemit operativ, pasi që këto karakteristika mund të hiqen.

4. METODOLGJIA

Për të realizuar këtë punim janë përdorur disa metoda të cilat kanë kontribuar në studim më të thelluar dhe më të detajizuar të temës që e kemi përzgjedhur. Metoda këto me anë të të cilave është arritur rezultati aktual. Më konkretisht gjatë këtij studimi janë përdorur këto metoda:

1. **Shfletimi i literaturës** – metodë e cila është përdorur që të bëhet i mundur kërkimi i projekteve, materialit, fakteve, librave dhe hulumtimeve të ngjajshme. Kjo metodë ka kontribuar në zgjerimin e diapazonit në këtë lami dhe është arritur niveli i duhur për t'u realizuar ndërtimi i aplikacionit me temë.
2. **Implementimi –** metodë e cila ka bërë të mundur zhvillimin e aplikacionit me anë të cilës e përcaktojmë se a kemi arritur rezultatin e dëshiruar.

5. REZULTATET

5.1 Zhvillimi i aplikacionit

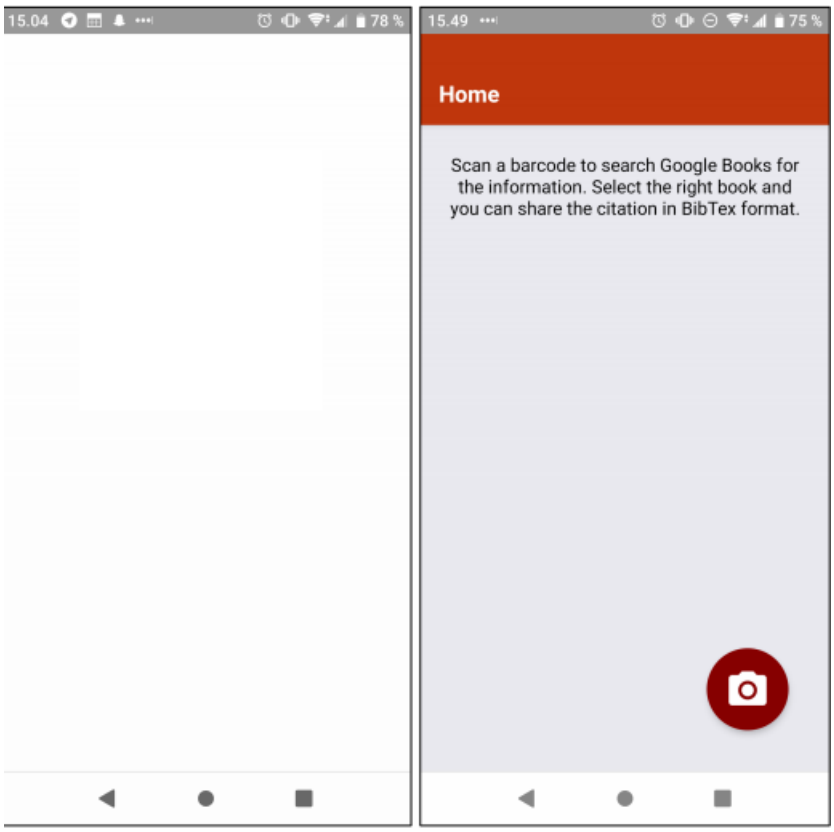

Figura 2 - Pamja e HOME

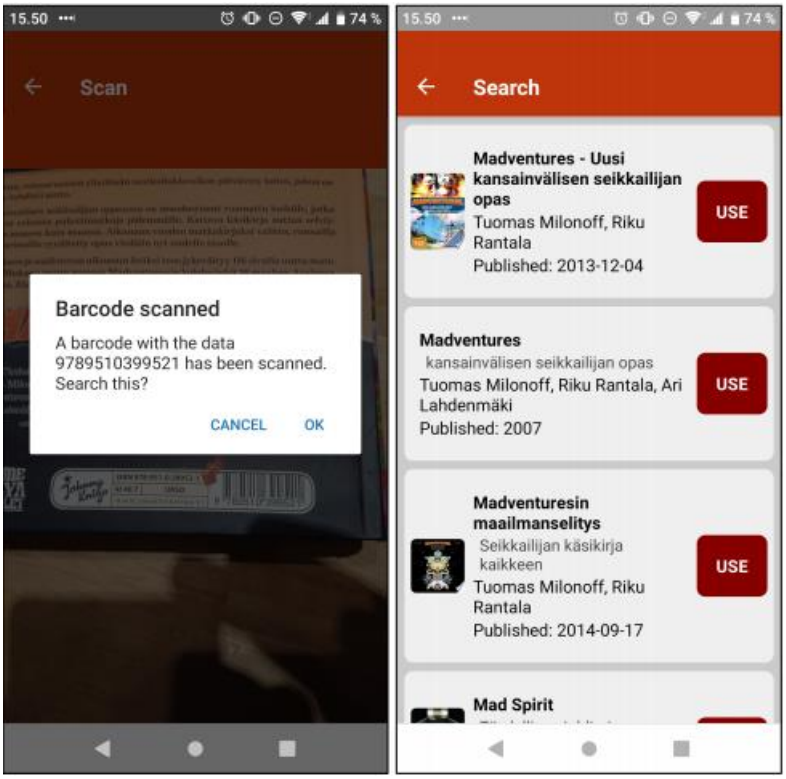

Figura 3- Skenimi me barcode

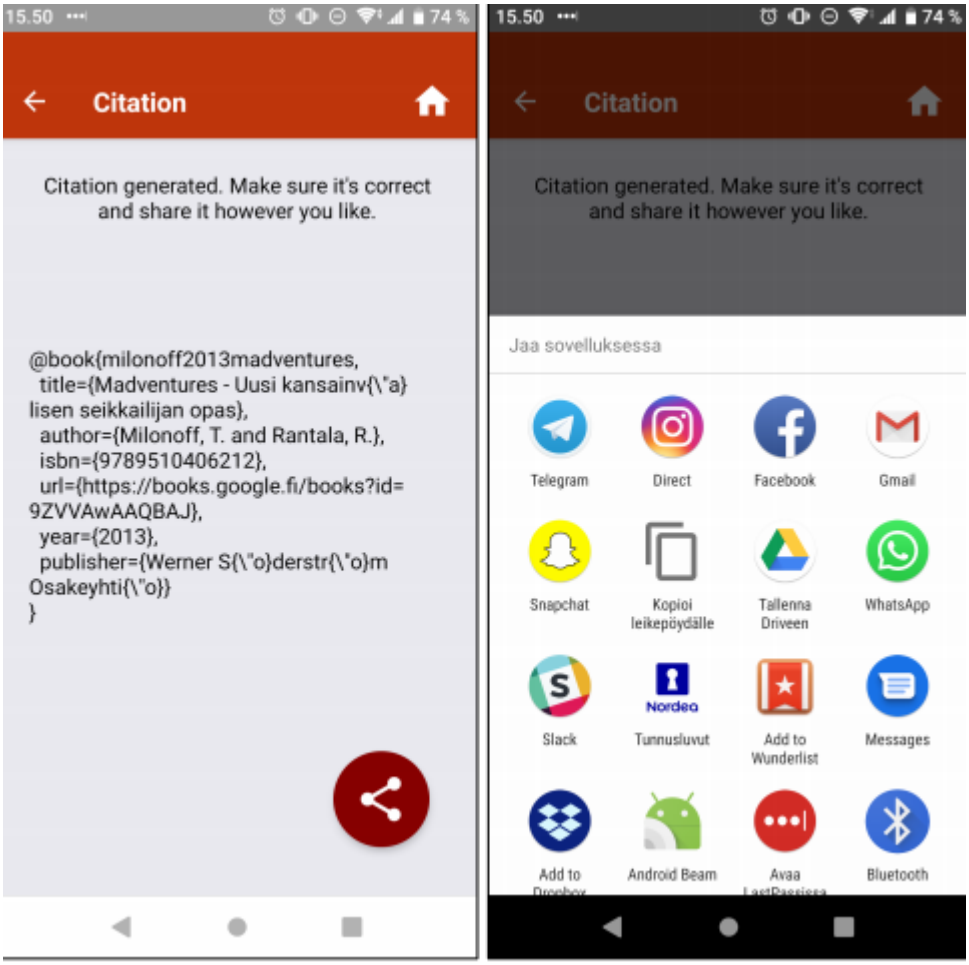

Figura 4 - Gjenerimi i citimit

6. DISKUTIME DHE PËRFUNDIME

Në përgjithësi rezultatet e studimit duken premtuese për React Native. Të gjithë qëllimet kryesore për aplikacionin e zhvilluar u arritën dhe, rezultati përfundimtar kishte pamjen dhe ndjesinë e një aplikacioni vendas. Sidoqoftë, duket se kërkime të mëtejshme ende duhet të bëhen para se të mund të nxirren konkluzione të forta për zhvillimin e lëvizshëm ndër-platformor si një e tërë. Teknologjitë e zhvillimit vazhdojnë të evoluojnë gjithashtu me një ritëm të shpejtë, kështu që shumica e kërkimeve të kaluara bëhen të vjetruara me kalimin e kohës. Shumë aspekte të zhvillimit të aplikacioneve nuk u diskutuan fare brenda kësaj teze por ato janë akoma faktorë të rëndësishëm për projekte të suksesshme zhvillimore. Këto përfshijnë testimin në formën e testimit të njësisë dhe testimin e automatizuar UI, botimin e një aplikacioni ndër-platformor dhe automatizimin e ndërtesës, testimin dhe botimin e proceseve. Bazuar në rezultatet React Native dhe teknologjitë mobile ndër-platformë në përgjithësi, mund të jenë një mundësi e vlefshme për zhvillimin e aplikacioneve vendase të lëvizshme. Kërkesa të caktuara nuk ndryshojnë, dhe është akoma përgjegjësia e projektuesve dhe zhvilluesve për të krijuar përvojën e përdoruesit vendas për secilën platformë, dhe si e tillë, zhvillimi i celularëve kërkon akoma njohuri të përvojës specifike të përdoruesit dhe udhëzimeve të dizajnit për secilën platformë. Zhvillimi i aplikacionit duke përdorur teknologji ndër-platformë mund ta bëjë procesin e zhvillimit më të lehtë dhe më pak kohë në disa mënyra, dhe mirëmbajtja e aplikacionit duhet gjithashtu të rritet për shkak të ripërdorimit të kodit. Edhe pse teknologjitë ndër-platformore mund të jenë një mundësi e vlefshme për zhvillimin e celularëve, vetëm bazuar në këtë studim është e paqartë se kur ata kanë një avantazh të qartë mbi zhvillimin vendas. Në aplikacione të thjeshta, ndryshimet në kohën e zhvillimit dhe mirëmbajtjen nuk janë të rëndësishme, dhe në aplikacione më komplekse, abstraksioni i rritur mund të mos ofrojë përfitime për zhvilluesit. Në fakt, në disa raste, kur duhet të zbatojmë shumë karakteristika vendase të platformës për aplikacionin, teknologjitë ndër-platformë mund të veprojnë vetëm si një pengesë për përparimin e aplikacionit.

Çuditërisht duket se çështja më e madhe e mundshme për React Native dhe teknologji të ngjashme vjen nga qëndrueshmëria. Në një projekt React Native jo vetëm që mbështetet te Google dhe Apple për të mbajtur të azhurnuara SDK-të e tyre, por gjithashtu mbështetet në të gjithë grupin e organizatave të palëve të treta dhe njerëzit për të mbajtur kornizat, libraritë dhe mjetet e tyre të azhurnuara gjithashtu. Ky nuk është problem kur gjithçka funksionon, dhe komuniteti me burim

të hapur ka një forcë të caktuar në këtë drejtim. Çështjet reale shfaqen kur edhe një nga kornizat, libraritë ose mjetet e palëve të treta bie nga mbështetja dhe mirëmbajtja.

7. REFERENCAT

[1] N. McDonald, «We Are Social: Digital in 2018,» 01 2020. [Në linjë]. Available: https://wearesocial.com/us/blog/2020/01/globaldigital-report-2020.. [Qasja 15 11 2020].

[2] S. L dhe C. S, «App Annie 2017 Retrospective,» 01 2020. [Në linjë]. Available: https://www.appannie.com/en/insights/marketdata/app-annie-2020-retrospective/. [Qasja 15 11 2020].

[3] «Google, Android Developer Documentation,» [Në linjë]. Available: https://developer.android.com/index.html. [Qasja 11 2020].

[4] «Apple, iOS Developer Documentation,» [Në linjë]. Available: https://developer.apple.com/. [Qasja 11 2020].

[5] «Apple, Swift Has Reached 1.0, Sept. 2014,» [Në linjë]. Available: https://developer.apple.com/swift/blog/?id=14. [Qasja 11 2020].

[6] «Apple, Apple Reinvents the Phone with iPhone,» [Në linjë]. Available: https://www.apple.com/newsroom/2007/01/09Apple-Reinvents-the-Phone-with-iPhone/. [Qasja 11 2020].

[7] H. G, S. G dhe D. A, «Cross-platform mobile development, Mobile Learning Environment,» *Cambridge*, vëll. i 16, nr. 9, pp. 158-171, 2011.

[8] R. C. P. R dhe T. S. B., «A study on approaches to build cross-platform,» në *2012 Annual IEEE India Conference (INDICON)*, 2012.

[9] A. «Performance Comparison: Xamarin.Forms, Xamarin.iOS, Xamarin.Android,» [Në linjë]. Available: https://www.altexsoft.com/blog/engineering/performancecomparison-xamarin-forms-xamarin-ios-xamarin-android-vs-androidand-ios-native-applications/. [Qasja 11 2020].

[10] B. K, «Cross-Platform Mobile App Development with Flutter — Xamarin — React Native: A Performance Focused Comparison,» [Në linjë]. Available: https://medium.com/@korhanbircan/crossplatform - mobile - app - development - with - flutter - xamarin - react -. [Qasja 12 2020].

[11] P. J, T. G dhe B. C, Xamarin: Cross-Platform Mobile Application, Birmingham: Packt Publishing, 2016.

[12] L. S, «Announcing Flutter beta 1: Build beautiful native apps,» [Në linjë]. Available: https://medium.com/flutter-io/announcing-flutter-beta-1-build-beautiful-native-apps-dc142aea74c0. [Qasja 12 2020].

[13] «Google, Flutter documentation,» [Në linjë]. Available: https://flutter.io/. [Qasja 12 2020].

[14] H. N dhe V. T, « Effects on performance and usability for cross-platform,» 2016.

[15] « Facebook, React Native Documentation,» [Në linjë]. Available: https://facebook.github.io/react-native/index.html. [Qasja 12 2020].

[16] I. E, «ECMAScript,» [Në linjë]. Available: https://www.ecma-international.org/ecma262/9.0/index.html. [Qasja 12 2020].

[17] N. V, Building Mobile Apps with JavaScript, Birmingham: Packt Publishing, 2017.

[18] P. N, «Chain React 2018 - React Native Fabric Architecture,» [Në linjë]. Available: http://blog.nparashuram.com/2018/07/chain-react-2018-react-native-fabric.html. [Qasja 12 2020].

8. APPENDIX

```
import React, { Component } from 'react';
import {
  Image,
  Share,
  StyleSheet,
  Text,
  TouchableOpacity,
  View
} from 'react-native';
import { NavigationContainerProps, NavigationScreenProp } from 'react-navigation';
import { connect } from 'react-redux';

import { ApplicationState } from '../store';
import { globalStyles } from '../util/styleConstants';
import PlatformButton from './PlatformButton';

type StateProps = {
  citation: string | undefined
};

type Props = Readonly<NavigationContainerProps & StateProps>;

class CitationView extends Component<Props> {
  static navigationOptions = ({ navigation }: { navigation: NavigationScreenProp<{}> }) =>
{
    return {
      title: 'Citation',
      headerRight: (
        <TouchableOpacity
          onPress={() => navigation.popToTop()}
        >
```

```
        <Image source={require('../assets/home.png')} style={{ width: 32, height: 32, marginRight: 18 }} />
      </TouchableOpacity>
    )
  }
};

_shareCitation() {
  const { citation } = this.props;
  if (citation === undefined) {
    return;
  }

  Share.share({ message: citation });
}

render() {
  return (
    <View style={styles.container}>
      <Text style={[globalStyles.bobyText, styles.bodyText]}>
        Citation generated. Make sure it's correct and share it however you like.
      </Text>

      <Text style={globalStyles.bobyText}>
        {this.props.citation}
      </Text>

      <PlatformButton image={'share'} textIos={'Share'} onPress={() => this._shareCitation()} />
    </View>
  );
}
```

```
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'space-between',
    margin: 24
  },
  bodyText: {
    textAlign: 'center'
  }
});

const mapStateToProps = (state: ApplicationState) => {
  return {
    citation: state.bookApi.citation
  };
};

export default connect<StateProps, {}, {},
ApplicationState>(mapStateToProps)(CitationView);

import { BarCodeScanner, Constants, Permissions } from 'expo';
import React, { Component, ReactNode } from 'react';
import { Alert, StyleSheet, Text, View } from 'react-native';
import { NavigationContainerProps } from 'react-navigation';
import { connect } from 'react-redux';
import { bindActionCreators, Dispatch } from 'redux';

import { ApplicationState } from '../store';
import { fetchCitation } from '../store/bookApi';
```

```
import { cameraPermission } from '../store/scanner';
import { colors, globalStyles } from '../util/styleConstants';
import PlatformButton from './PlatformButton';

type StateProps = {
  hasCameraPermission: boolean | undefined
};

type DispatchProps = {
  cameraPermission: typeof cameraPermission,
  fetchCitation: typeof fetchCitation
};

type Props = Readonly<NavigationContainerProps & StateProps & DispatchProps>;

class MainView extends Component<Props> {
  static navigationOptions = {
    title: 'Home',
  };

  async componentDidMount() {
    const result: Permissions.PermissionResponse = await
Permissions.askAsync(Permissions.CAMERA);
    this.props.cameraPermission(result.status === 'granted');
  }

  _navigateToScanner() {
    const { hasCameraPermission, navigation } = this.props;

    if (navigation == undefined) {
      return;
    }
```

```
    if (hasCameraPermission !== true) {
      Alert.alert(
        'No camera permission',
        'The app has not been granted permission to use the camera.',
      );
      return;
    }

    navigation.navigate('Scanner');
  }

  render() {
    const { hasCameraPermission } = this.props;

    let status: ReactNode = null;

    if (hasCameraPermission === undefined) {
      status = <Text style={[globalStyles.bobyText, styles.bodyText]}>'Requesting for camera permission'</Text>;
    }
    if (hasCameraPermission === false) {
      status = <Text style={[globalStyles.bobyText, styles.bodyText]}>'No camera permission'</Text>;
    }

    return (
      <View style={styles.container}>
        <Text style={[globalStyles.bobyText, styles.bodyText]}>
          Scan a barcode to search Google Books for the information. Select the right book and you can share the citation in BibTex format.
        </Text>
```

```
        {status}

        <PlatformButton image={'camera'} textIos={'Scan'} onPress={() =>
this._navigateToScanner()} />
      </View>
    );
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'space-between',
    margin: 24
  },
  bodyText: {
    textAlign: 'center'
  },
  button: {
    backgroundColor: colors.primaryColorD,
    padding: 14,
    borderRadius: 40,
    alignSelf: 'flex-end',
    marginBottom: 12,
    marginRight: 12
  },
  buttonImage: {
    width: 48,
    height: 48
  }
```

```
});

const mapStateToProps = (state: ApplicationState) => {
  return {
    hasCameraPermission: state.scanner.hasCameraPermission
  };
};

const mapDispatchToProps = (dispatch: Dispatch) => bindActionCreators({
  cameraPermission,
  fetchCitation
}, dispatch);

export default connect<StateProps, DispatchProps, {},
ApplicationState>(mapStateToProps, mapDispatchToProps)(MainView);

import React, { SFC } from 'react';
import {
  Image,
  ImageRequireSource,
  ImageStyle,
  Platform,
  StyleSheet,
  Text,
  TouchableOpacity,
} from 'react-native';

import { colors } from '../util/styleConstants';

type ImageObject = {
  [key: string]: ImageRequireSource;
}
```

```
const images: ImageObject = {
  cameraWhite: require('../assets/camera-white.png') as ImageRequireSource,
  cameraDark: require('../assets/camera-dark.png') as ImageRequireSource,
  shareWhite: require('../assets/share-white.png') as ImageRequireSource,
  shareDark: require('../assets/share-dark.png') as ImageRequireSource
};

const androidPostFix = 'White';
const iosPostFix = 'Dark';

const androidImageSize = 40;
const iosImageSize = 36;

type Props = {
  textIos: string,
  image: 'camera' | 'share',
  onPress: () => void
};

const PlatformButton: SFC<Props> = (props: Props) => {
  if (Platform.OS === 'android') {
    return (
      <TouchableOpacity style={styles.androidButton} onPress={() => props.onPress()}>
        <Image
          source={images[props.image + androidPostFix]}
          style={styles.androidButtonImage as ImageStyle}
        />
      </TouchableOpacity>
    );
  }
  else {
```

```
    return (
      <TouchableOpacity style={styles.iosButton} onPress={() => props.onPress()}>
        <Text style={styles.iosButtonText}>{props.textIos}</Text>
        <Image
          source={images[props.image + iosPostFix]}
          style={styles.iosButtonImage as ImageStyle}
        />
      </TouchableOpacity>
    );
  }
}
export default PlatformButton;

const styles = StyleSheet.create({
  androidButton: {
    backgroundColor: colors.primaryColorD,
    elevation: 4,
    padding: 16,
    borderRadius: androidImageSize,
    alignSelf: 'flex-end',
    marginBottom: 12,
    marginRight: 12
  },
  androidButtonImage: {
    width: androidImageSize,
    height: androidImageSize
  },
  iosButton: {
    flexDirection: 'row',
    borderColor: colors.primaryColorD,
    borderWidth: 2,
    padding: 16,
```

```
    borderRadius: 24,
    alignSelf: 'center',
    alignItems: 'center',
  },
  iosButtonImage: {
    width: iosImageSize,
    height: iosImageSize
  },
  iosButtonText: {
    flex: 1,
    fontSize: 22,
    color: colors.primaryColorD,
    textAlign: 'center',
    marginLeft: iosImageSize
  }
});

import { BarCodeScanner } from 'expo';
import React, { Component } from 'react';
import { Alert, StyleSheet, Text, View } from 'react-native';
import { NavigationContainerProps } from 'react-navigation';
import { connect } from 'react-redux';
import { bindActionCreators, Dispatch } from 'redux';

import { ApplicationState } from '../store';
import { BarCodeScannerResult, scanResultDone, scanSuccess } from '../store/scanner';
import delay from '../util/delay';

type StateProps = {
  isBusy: boolean,
  scanResultData: string | undefined,
  hasCameraPermission: boolean | undefined
```

```
};

type DispatchProps = {
  scanSuccess: typeof scanSuccess,
  scanResultDone: typeof scanResultDone
};

type Props = Readonly<NavigationContainerProps & StateProps & DispatchProps>;

class ScannerView extends Component<Props> {
  static navigationOptions = {
    title: 'Scan',
  };

  componentDidUpdate(prevProps: Props) {
    const { scanResultData } = this.props;

    // Gotcha: if using navigation.navigate, the call does not unmount view
    if (scanResultData !== undefined && !prevProps.isBusy) {
      Alert.alert(
        'Barcode scanned',
        `A barcode with the data ${scanResultData} has been scanned. Search this?`,
        [
          { text: 'Cancel', onPress: () => this._alertCancelPressed(), style: 'cancel' },
          { text: 'OK', onPress: () => this._alertOkPressed() },
        ],
        { onDismiss: () => this._alertCancelPressed() }
      );
    }
  }

  render() {
```

```
  const { hasCameraPermission, isBusy } = this.props;

  if (hasCameraPermission !== true) { // Should never happen
    return <Text>No access to camera</Text>;
  }

  return (
    <View style={{ flex: 1 }}>
      <BarCodeScanner
        onBarCodeRead={
          isBusy ? // This will only prevent events when this view is focused
          undefined :
          (res: BarCodeScannerResult) => {
            this.props.scanSuccess(res)
          }
        }
        style={StyleSheet.absoluteFill}
      />
    </View>
  );
}

async _alertCancelPressed() {
  await delay(1000);
  this.props.scanResultDone();
}

_alertOkPressed() {
  if (this.props.navigation == undefined) {
    return;
  }
  // TODO: reactivate scanner when necessary
```

```
    this.props.navigation.replace('Search');
    this.props.scanResultDone();
  }
}

const mapStateToProps = (state: ApplicationState) => {
  return {
    isBusy: state.scanner.isBusy,
    scanResultData: state.scanner.scanResultData,
    hasCameraPermission: state.scanner.hasCameraPermission
  };
};

const mapDispatchToProps = (dispatch: Dispatch) => bindActionCreators({
  scanSuccess,
  scanResultDone
}, dispatch);

export default connect<StateProps, DispatchProps, {},
ApplicationState>(mapStateToProps, mapDispatchToProps)(ScannerView);



import React, { Component } from 'react';
import {
    ActivityIndicator,
    FlatList,
    Image,
    ImageStyle,
    StyleSheet,
    Text,
    TouchableOpacity,
    View
```

```
} from 'react-native';
import { NavigationContainerProps } from 'react-navigation';
import { connect } from 'react-redux';
import { bindActionCreators, Dispatch } from 'redux';

import { ApplicationState } from '../store';
import { fetchBook, fetchCitation } from '../store/bookApi';
import { colors, globalStyles } from '../util/styleConstants';

type StateProps = {
  scanResultData: string | undefined,
  searchedBooks: BookData[],
  isBusy: boolean
};

type DispatchProps = {
  fetchBook: typeof fetchBook,
  fetchCitation: typeof fetchCitation
};

type Props = Readonly<NavigationContainerProps & StateProps & DispatchProps>;

class SearchView extends Component<Props> {
  static navigationOptions = {
    title: 'Search',
  };

  componentWillMount() {
    if (this.props.scanResultData === undefined) {
      return;
    }
    this.props.fetchBook(this.props.scanResultData);
```

```
  }

  _renderLoadingView() {
    return (
      <View style={[styles.container, styles.loading]}>
        <Text style={[globalStyles.bobyText, styles.loadingText]}>Searching Google Books for {this.props.scanResultData}</Text>
        <ActivityIndicator size={'large'} />
      </View>
    );
  }

  _renderListItem(item: BookData) {
    return (
      <View style={styles.listItem}>
        {
          item.volumeInfo.imageLinks !== undefined && item.volumeInfo.imageLinks.smallThumbnail !== undefined &&
          <Image
            style={styles.image as ImageStyle} // TODO: fix type
            source={{uri: item.volumeInfo.imageLinks.smallThumbnail}}
            resizeMode={'contain'}
          />
        }
        <View style={styles.listItemInfo}>
          <Text style={[globalStyles.bobyText, {fontWeight: 'bold'}]}>{item.volumeInfo.title}</Text>
          {
            item.volumeInfo.subtitle !== undefined &&
            <Text style={styles.subtitle}>{item.volumeInfo.subtitle}</Text>
          }
          <Text style={globalStyles.bobyText}>
```

```
          {
            item.volumeInfo.authors !== undefined &&
            item.volumeInfo.authors.join(', ')
          }
        </Text>
        <Text                                    style={globalStyles.bobyText}>Published:
{item.volumeInfo.publishedDate}</Text>
      </View>

      <TouchableOpacity style={styles.button} onPress={() => this._itemSelected(item)}>
        <Text style={styles.buttonText}>USE</Text>
      </TouchableOpacity>
    </View>
  )
 }

 _renderResultsView() {
  return (
    <View style={styles.container}>
      <FlatList
        data={this.props.searchedBooks}
        renderItem={(obj: {item: BookData}) => this._renderListItem(obj.item)}
        keyExtractor={(item: BookData) => item.id}
        ListHeaderComponent={<View style={styles.listHeader} />}
      />
    </View>
  );
 }

 render() {
  if (this.props.isBusy) {
    return this._renderLoadingView();
```

```
    }
    else {
      return this._renderResultsView();
    }
  }

  _itemSelected(item: BookData) {
    this.props.fetchCitation(item.id);

    const { navigation } = this.props;

    if (navigation == undefined) {
      return;
    }

    navigation.navigate('Citation');
  }
}

const styles = StyleSheet.create({
  button: {
    padding: 18,
    borderRadius: 8,
    backgroundColor: colors.primaryColorD
  },
  buttonText: {
    fontSize: 16,
    fontWeight: 'bold',
    color: 'white'
  },
  container: {
    flex: 1,
```

```
    backgroundColor: '#ccc'
  },
  image: {
    width: 50,
    height: 90,
    borderRadius: 4
  },
  listHeader: {
    marginTop: 8
  },
  listItem: {
    flex: 1,
    flexDirection: 'row',
    justifyContent: 'flex-start',
    alignItems: 'center',
    marginHorizontal: 8,
    marginBottom: 8,
    paddingVertical: 12,
    paddingHorizontal: 6,
    borderRadius: 8,
    backgroundColor: '#f0f0f0'
  },
  listItemInfo: {
    flex: 1,
    flexDirection: 'column',
    flexWrap: 'wrap',
    margin: 8
  },
  loading: {
    justifyContent: 'flex-start',
    alignContent: 'center',
    paddingHorizontal: 32
```

```
  },
  loadingText: {
    marginVertical: 40,
    textAlign: 'center'
  },
  subtitle: {
    fontSize: 15,
    color: '#444',
    paddingLeft: 6
  }
});

const mapStateToProps = (state: ApplicationState) => {
  return {
    scanResultData: state.scanner.scanResultData,
    searchedBooks: state.bookApi.searchedBooks,
    isBusy: state.bookApi.isBusy
  };
};

const mapDispatchToProps = (dispatch: Dispatch) => bindActionCreators({
  fetchBook,
  fetchCitation
}, dispatch);

export default connect<StateProps, DispatchProps, {},
ApplicationState>(mapStateToProps, mapDispatchToProps)(SearchView);

import { Action } from 'redux';
import { ThunkAction, ThunkDispatch } from 'redux-thunk';

import assertNever from '../util/assertNever';
```

```
import readFile from '../util/readFile';
import { ApplicationState } from './';

// Not an authentication key, used for statistics.
// TODO: restrict api key usage to Android/iOS apps only
const apiKey: string = 'AIzaSyAiUlbaPvpP3QC-qnjl7_MXPQ7HY0RDgqg';

export type BookApiState = Readonly<{
  isBusy: boolean,
  searchedBooks: BookData[],
  citation: string | undefined,
  error: Error | undefined
}>;

const defaultState: BookApiState = {
  isBusy: false,
  searchedBooks: [],
  citation: undefined,
  error: undefined
};

// TS 'typeof' returns the exact value as type for implicitly typed 'const'
const FETCH_BOOK_START = 'refgen-app/bookApi/FETCH_BOOK_START';
const FETCH_BOOK_SUCCESS = 'refgen-app/bookApi/FETCH_BOOK_SUCCESS';
const FETCH_CITATION_START = 'refgen-app/bookApi/FETCH_CITATION_START';
const FETCH_CITATION_SUCCESS = 'refgen-
app/bookApi/FETCH_CITATION_SUCCESS';
const FETCH_ERROR = 'refgen-app/bookApi/FETCH_ERROR';

interface FetchBookStartAction extends Action {
  type: typeof FETCH_BOOK_START;
};
```

```
interface FetchBookSuccessAction extends Action {
  type: typeof FETCH_BOOK_SUCCESS;
  payload: BookData[];
};

interface FetchCitationStartAction extends Action {
  type: typeof FETCH_CITATION_START;
};

interface FetchCitationSuccessAction extends Action {
  type: typeof FETCH_CITATION_SUCCESS;
  payload: any // TODO: Type this
};

interface FetchErrorAction extends Action {
  type: typeof FETCH_ERROR;
  payload: Error;
};

type KnownAction = FetchBookStartAction |
                   FetchBookSuccessAction |
                   FetchCitationStartAction |
                   FetchCitationSuccessAction |
                   FetchErrorAction;

export default function reducer (state: BookApiState = defaultState, action: KnownAction):
BookApiState {
  switch (action.type) {
    case FETCH_BOOK_START: {
      return {
        ...state,
```

```
      isBusy: true
    };
  }
  case FETCH_BOOK_SUCCESS: {
    return {
      ...state,
      isBusy: false,
      searchedBooks: action.payload
    };
  }
  case FETCH_CITATION_START: {
    return {
      ...state,
      isBusy: true
    };
  }
  case FETCH_CITATION_SUCCESS: {
    return {
      ...state,
      isBusy: false,
      citation: action.payload
    };
  }
  case FETCH_ERROR: {
    return {
      ...state,
      isBusy: false,
      error: action.payload
    };
  }
  default: {
    assertNever(action);
```

```
      return state;
    }
  }
}

export function fetchBook(barCode: string): ThunkAction<void, undefined, undefined, KnownAction> {
  return async (dispatch: ThunkDispatch<undefined, undefined, KnownAction>) => {
    dispatch({ type: FETCH_BOOK_START });

    try {
      const response = await fetch(`https://www.googleapis.com/books/v1/volumes?q=${barCode}&key=${apiKey}`);
      if (!response.ok) {
        throw new Error(response.statusText);
      }

      const responseData = await response.json();

      if (responseData.totalItems === 0) {
        throw new Error('No books found');
      }

      const payload: BookData[] = responseData.items.map((responseBook: BookResource) => {
        const book: BookData = {
          id: responseBook.id,
          selfLink: responseBook.selfLink,
          volumeInfo: {
            title: responseBook.volumeInfo.title,
            subtitle: responseBook.volumeInfo.subtitle != undefined ? responseBook.volumeInfo.subtitle : undefined,
```

```
            authors: responseBook.volumeInfo.authors,
            publisher: responseBook.volumeInfo.publisher != undefined ? responseBook.volumeInfo.publisher : undefined,
            publishedDate: responseBook.volumeInfo.publishedDate,
            industryIdentifiers: responseBook.volumeInfo.industryIdentifiers,
            printType: responseBook.volumeInfo.printType,
            imageLinks: responseBook.volumeInfo.imageLinks != undefined ? responseBook.volumeInfo.imageLinks : undefined,
            language: responseBook.volumeInfo.language,
            infoLink: responseBook.volumeInfo.infoLink
          },
          accessInfo: {
            country: responseBook.accessInfo.country,
            viewability: responseBook.accessInfo.viewability,
            publicDomain: responseBook.accessInfo.publicDomain,
            epub: responseBook.accessInfo.epub,
            pdf: responseBook.accessInfo.pdf
          }
        };
        return book;
      })

      dispatch({ type: FETCH_BOOK_SUCCESS, payload });
    }
    catch(error) {
      dispatch({ type: FETCH_ERROR, payload: error });
    }
  }
}

export function fetchCitation(id: string):ThunkAction<void, ApplicationState, undefined, KnownAction> {
```

```
  // e.g. https://books.google.fi/books/download/Nälkäpeli.bibtex?id=pB0GDQEACAAJ&hl=fi&output=bibtex
  return async (dispatch: ThunkDispatch<ApplicationState, undefined, KnownAction>, getState: () => ApplicationState) => {
    dispatch({ type: FETCH_CITATION_START });

    // TODO: Localize hl parameter, add a setting?
    try {
      const response = await fetch(`https://books.google.fi/books/download/?id=${id}&hl=fi&output=bibtex`);
      if (!response.ok) {
        throw new Error(response.statusText);
      }

      const data = await response.blob();

      const text = await readFile(data);

      dispatch({ type: FETCH_CITATION_SUCCESS, payload: text });
    }
    catch (error) {
      dispatch({ type: FETCH_ERROR, payload: error });
    }
  }
}

import { combineReducers } from 'redux';

import bookApi, { BookApiState } from './bookApi';
import scanner, { ScannerState } from './scanner';
```

```
export interface ApplicationState {
  scanner: ScannerState,
  bookApi: BookApiState
};

export const reducers = combineReducers({
  scanner,
  bookApi
});

import { Action } from 'redux';

import assertNever from '../util/assertNever';

export type BarCodeScannerResult = {
  type: string,
  data: string
};

export type ScannerState = Readonly<{
  isBusy: boolean,
  scanResultData: string | undefined,
  hasCameraPermission: boolean | undefined,
  error: Error | undefined
}>;

const defaultState: ScannerState = {
  isBusy: false,
  scanResultData: undefined,
  hasCameraPermission: undefined,
  error: undefined
};
```

```
// TS 'typeof' returns the exact value as type for implicitly typed 'const'
const CAMERA_PERMISSION = 'refgen-app/scanner/CAMERA_PERMISSION';
const SCAN_RESULT_DONE = 'refgen-app/scanner/SCAN_RESULT_DONE';
const SCAN_SUCCESS = 'refgen-app/scanner/SCAN_SUCCESS';

interface CameraPermissionAction extends Action {
  type: typeof CAMERA_PERMISSION;
  payload: boolean;
};

interface ScanResultDoneAction extends Action {
  type: typeof SCAN_RESULT_DONE;
}

interface ScanSuccessAction extends Action {
  type: typeof SCAN_SUCCESS;
  payload: string;
};

type KnownAction = CameraPermissionAction |
                   ScanResultDoneAction |
                   ScanSuccessAction;

export default function reducer (state: ScannerState = defaultState, action: KnownAction):
ScannerState {
  switch (action.type) {
    case CAMERA_PERMISSION: {
      return {
        ...state,
        hasCameraPermission: action.payload
      };
```

```
    }
    case SCAN_RESULT_DONE: {
      return {
        ...state,
        isBusy: false
      };
    }
    case SCAN_SUCCESS: {
      if (state.isBusy) {
        return state;
      }

      return {
        ...state,
        isBusy: true,
        scanResultData: action.payload
      };
    }
    default: {
      assertNever(action);
      return state;
    }
  }
}

export function cameraPermission(payload: boolean): CameraPermissionAction {
  return {
    type: CAMERA_PERMISSION,
    payload
  };
}
```

```
export function scanResultDone(): ScanResultDoneAction {
  return {
    type: SCAN_RESULT_DONE
  };
}

// Only use scanned data, also available type and target (target exists on Android, not sure about iOS)
// TODO: Fix BarCodeScannerCallback type in DefinitelyTyped repo, check target in iOS
export function scanSuccess(payload: BarCodeScannerResult): ScanSuccessAction {
  return {
    type: SCAN_SUCCESS,
    payload: payload.data
  };
}
```