UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Luka Vidmar

Primerjava izvornega in hibridnega razvoja mobilne aplikacije

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

Ljubljana, 2018

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Luka Vidmar

Primerjava izvornega in hibridnega razvoja mobilne aplikacije

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: viš. pred. dr. Igor Rožanc

Ljubljana, 2018

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Zaradi obilice različnih mobilnih platform in verzij je razvoj povsod delujoče mobilne aplikacije za razvijalca velik izziv. V diplomski nalogi preučite in primerjajte razvoj izvorne mobilne aplikacije za Android z razvojem hibridne mobilne rešitve, ki zmore generirati tudi aplikacije za druge platforme. V ta namen najprej predstavite značilnosti sodobnih mobilnih naprav ter izvorni in hibridni način razvoja mobilnih aplikacij. Na zgledu sestavljene mobilne aplikacije, ki vsebuje različne tipične značilnosti tovrstnih aplikacij, prikažite razvoj obeh različic ter ju primerjajte med seboj. Primerjavo opravite po različnih kriterijih tako z vidika razvijalca kot uporabnika.

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Luka Vidmar z vpisno številko 63110041 sem avtor diplomskega dela z naslovom:

Primerjava izvornega in hibridnega razvoja mobilne aplikacije

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom viš. pred. dr. Igorja Rožanca,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 29. januarja 2018 Podpis avtorja:

Zahvaljujem se viš. pred. dr. Igorju Rožancu za mentorstvo pri diplomski nalogi, zraven pa še staršem in vsem, ki so mi nudili podporo in pomoč med študijem.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
RAM	random access memory	bralno pisalni pomnilnik
ROM	read only memory	bralni pomnilnik
API	application programming interface	aplikacijski programski vmesnik
IDE	integrated development environment	integrirano razvojno okolje

Povzetek

Naslov: Primerjava izvornega in hibridnega razvoja mobilne aplikacije

Cilj diplomske naloge je ugotoviti razlike med izdelavo in samim delovanjem hibridne in izvorne aplikacije. Da bi razumeli le-te, moramo poznati delovanje pametne mobilne naprave, njeno zgodovino in razvoj. Na mobilnih napravah so različni operacijski sistemi, kot so Android, iOS in Windows Phone. Vsak izmed teh ima svoj programski jezik, v katerem je razvit; Android ima Javo, iOS ima C in Windows Phone ima C#. Z različnimi razvojnimi okolji pa se razvijajo mobilne aplikacije; Android ima priporočljivo razvojno okolje Eclipse ali Android Studio, iOS ima Xcode in Windows Phone ima Visual Studio. Hibridni razvoj mobilne aplikacije je možen na več načinov, vsak pa ima svoje jezike in priporočljivo razvojno okolje. Aplikacije se lahko razvijajo v izvornem in hibridnem načinu, vendar ima vsak svoje prednosti in pomanjkljivosti. Za izpostavitev le-teh, moramo ustvariti aplikacijo, ki je razdeljena na 11 delov; kamera, tri v vrsto, kača, vibriranje, baterija, zanke, slike, označevanje besedila, delo z objekti in shramba v pomnilnik in datoteko, rekurzija in iskanje v binarnem drevesu in povezanem seznamu, igranje klavirja. S temi deli, se vidi razlika med hibridno in izvorno aplikacijo v hitrosti delovanja, izgledu uporabniškega vmesnika, klicih vgrajenih funkcij pametne mobilne naprave, zvoku, času izdelave aplikacije, urejenosti strukture datotek in delovanja na različnih platformah.

Ključne besede: hibridna aplikacija, izvorna aplikacija, primerjava hibridne in izvorne aplikacije

Abstract

Title: A comparison of native and hybrid mobile application development.

The goal of diploma is to determine the differences between making and using hybrid and original application. To understand these, at first we have to understand the operation of mobile devices, their history and evolution. There are multiple different operational systems on mobile devices such as Android, iOS or Windows Phone. Each one of them has its own programming language in which it has been developed; Android has Java, iOS has C and Windows Phone has C#. However with different development environments different mobile applications are evolving; Android has development environment Eclipse or Android Studio, iOS has Xcode and Windows Phone has Visual Studio. Hybrid development is possible in several different ways, but each one has its own set of languages and recommended development environment. The application can be developed in original or hybrid way, therefore each one has its own set of advantages and imperfections. To expose these, we have to create application, which is divided on 11 parts, camera, snake, vibration, battery, for loop, pictures, highlighting text, work with objects and storage in memory or file, recursion and search in binary tree and connected list, playing piano. With these parts, the difference can be shown between hybrid or original application in speed, looks of users interface, calls of built-in functions of smart mobile device, sound, time required for making, structure of files and proper function on different platforms.

Keywords: hybrid application, original application, the comparison of hybrid and original application

Poglavje 1 Uvod

Živimo v svetu, v katerem so pametne mobilne naprave vedno bolj in bolj nepogrešljive. Z nami so v vsakem trenutku. Pametne mobilne naprave ne služijo več samo pošiljanju sporočil in klicev, kot je bilo sprva mišljeno, ampak tudi fotografiranju, druženju s prijatelji, igranju iger, oglaševanju, brskanju po spletu, skratka skoraj vsemu. Trenutno pametne mobilne naprave uporablja približno 4,3 milijarde uporabnikov, kar pomeni ogromen trg za programerje. Seveda to prinaša tudi slabe lastnosti; zaradi tako velikega trga se ogromno programerjev poda v te vode in imamo posledično na voljo toliko aplikacij in iger, da je težko dobiti tako idejo, ki bi še lahko uspela.

Z razvojem pametnih mobilnih naprav so se razvile tudi različne platforme, kot so na primer Android, Windows Phone in iOS. Cilj programerja je, da za mobilno aplikacijo porabi čim manj časa, čim manj denarja, hkrati pa mora biti hitra, lepo oblikovana, da pritegne uporabnika, in seveda zanimiva. Predvsem zaradi hitrosti razvoja aplikacije in lažjega programiranja le-te, so se na trgu pojavili različni hibridni razvoji aplikacij, kot so na primer PhoneGap, Xamarin, Ionic itd.

Da bi programerjem olajšali izbiro razvoja aplikacije, bomo primerjali hibridni in izvorni razvoj aplikacije. Zanimala nas bo hitrost delovanja aplikacije, izgled uporabniškega vmesnika, poskusili bomo uporabiti različne vgrajene funkcije pametne mobilne naprave, kako se obnaša zvok, čas za izdelavo aplikacije, urejenost datotek in delovanje aplikacije na različnih platformah.

V poglavju 2 bomo opisali pametno mobilno napravo ter njeno zgodovino in razvoj. V poglavju 3 bomo našteli različne operacijske sisteme za pametno mobilno napravo in opisali programski jezik, v katerem je operacijski sistem spisan in najbolj pogosto uporabljeno razvojno okolje. Nato bomo v poglavju 4 prikazali razvoj hibridne in nato še izvorne aplikacije. V zadnjem poglavju 6 pa bomo še opisali glavne razlike, ki smo jih pridobili s primerjavo hibridne in izvorne aplikacije.

Seveda pa testiranje ne bo popolno, ker obstaja še veliko stvari, ki bi jih lahko testirali, kot je na primer povezovanje aplikacije na bazo podatkov, primerjava z aplikacijo za iOS, obdelovanje zvoka ali slike in podobno.

Poglavje 2 Pametna mobilna naprava

Da bomo razumeli, kako deluje pametna mobilna naprava, si moramo najprej pogledati, kaj to sploh je, njeno zgodovino in njen razvoj skozi leta.

2.1 Pametna mobilna naprava

Pametna mobilna naprava je naprava z naprednim operacijskim sistemom. Vsebuje funkcije računalniškega operacijskega sistema in funkcije, ki so uporabne za telefonijo.

Sodobne funkcije:

- -koledar
- -ura
- -kamera
- -prenos podatkov med napravami
- -internet
- -raznorazne igre in aplikacije

Funkcije uporabne za telefonijo:

- -klicanje
- -pošiljanje sporočil
- -shranjevanje stikov

Sestavljene so iz skoraj istih komponent kot računalnik. Imajo matično ploščo, RAM, procesor, grafični čip, pomnilnik in ekran, ki so prikazani na sliki 2.1. Pri nakupu pametne naprave najpogosteje odločajo naštete komponente, zadnje čase pa je velik poudarek še na velikosti ekrana in kvaliteti kamere.

Slika 2.1: Komponente pametne mobilne naprave.

Leto ali dve nazaj bi lahko rekli, da so mobilne naprave žepne velikosti. Zaradi velikega poudarka na udobnosti pri brskanju po spletu, se pametne mobilne naprave vedno bolj večajo. Pametne mobilne naprave dosegajo velikosti 6 ", tablični računalniki pa tja do 12.9 ". Običajno vsebujejo barvne zaslone na dotik, ki zajemajo do 95 % prednje plošče naprave. Zaradi virtualne tipkovnice standardna tipkovnica ni več potrebna. Seveda pa nam velik zaslon s seboj prinaša tudi pomanjkljivosti. Prva izmed njih je velika poraba baterije, kar pomeni, da večina uporabnikov polni aparat vsakodnevno. Obstajajo tudi izjeme, ki zdržijo nekaj dni, vendar jih je zelo malo, seveda pa je to odvisno tudi od uporabnikove rabe mobilne naprave. Velik zaslon je tudi zelo občutljiv na udarce, kar pomeni, da zaslon ob padcu po vsej verjetnosti poči [1] [2].

2.2 Zgodovina

Podjetje International Business Machines Corp je leta 1994 izdalo prvi mobilni telefon, ki je že vseboval aplikacije in zaslon na dotik s pisalom. Prikazan je na sliki 2.2. Nadeli so mu ime IBM Simon, prodali pa so jih okrog 50 000 [3].

Slika 2.2: IBM Simon.

Leta 1996 je Nokia Communicator izdala mobilni telefon, ki je po zunanjosti izgledal kot običajen telefon. Bil je preklopni aparat, ki je vseboval LCD zaslon v skoraj takšni velikosti, kot je bil aparat sam. Prikazan je na sliki 2.3. Na spodnji strani je vseboval QWERTY tipkovnico [3].

Slika 2.3: Telefon, ki ga je izdala Nokia Communicator.

Leta 2000 je prišel na trg Ericsson R380, slika 2.4. Bil je prvi pametni mobilni telefon, ponašal pa se je tudi s skromno težo in normalno velikostjo. Uporabljal je nov operacijski sistem OS [3].

Slika 2.4: Ericsson R380.

Leta 2002 je izšel Palm Treo, ki je prikazan na sliki 2.5 in ga je razvilo podjetje HandSpring, ki ga je kasneje kupil Palm. Pomemben je predvsem zato, ker je že vseboval večopravilnost. Med klicanjem je uporabnik lahko gledal koledar. Podpiral je tudi direktno klicanje iz seznama kontaktov in pošiljanje e-sporočil. Imel je že barven zaslon in nameščen Palm OS [3].

Slika 2.5: Palm Treo.

Blackberry Quark 6210, slika 2.6, je izšel leta 2003. Pomemben je bil, ker je bil prvi, ki je imel glasovne klice. Pred tem je bilo potrebno na telefon priklopiti žične slušalke. Nameščen je imel Blackberry OS [3].

Slika 2.6: Blackberry Quark 6210.

Leta 2007 je na trg prišel iPhone prve generacije, ki ga lahko vidimo na sliki 2.7. Zasnoval in razvil ga je Apple. Podpiral je že GPRS in EDGE prenos podatkov. Na pogled je bil zelo privlačne oblike in pa intuitiven uporabniški vmesnik. Bil je prvi pametni mobilni telefon, ki je uporabljal zaslon na dotik s prsti kot glavno interakcijo med uporabnikom in telefonom namesto pisal, tipkovnice in miške [3].

Slika 2.7: iPhone prve generacije.

HTC Dream, slika 2.8, je izšel leta 2008. Poganjal je Linux, ki pa je temeljil na Android operacijskem sistemu. Bil je konkurenca Blackberry operacijskemu sistemu, Symbianu in pa iOS-u. Ponašal se je s prilagodljivim grafičnim vmesnikom in integriranimi Google storitvami, kot je Gmail ali pa Android Market [3].

Slika 2.8: HTC Dream.

Leta 2010 je Samsung izdal Galaxy S Android, ki je bil prva naprava tretje Android serije pametnih mobilnih telefonov, ki jih je razvil Samsung. Bil je še prvi večji konkurent iPhonu [3]. Prikazan je na sliki 2.9.

Slika 2.9: Samsung Galaxy S.

Leta 2011 je bila izdana Motorola Razr. Bil je najtanjši pametni telefon na svetu. Njegova debelina je bila 7.1 mm [3]. Prikaz debeline na sliki 2.10.

Slika 2.10: Motorola Razr.

Leta 2013 je izšel iPhone 5s, ki je prikazan na sliki 2.11. Bil je najbolj prodajan telefon leta, ponašal pa se je s prepoznavanjem prstnega odtisa, ki je bil nameščen na tipki `domov`. Prepoznava prstnega odtisa se je uporabljala za odklepanje zaslona, avtentikacijo za App Store in nakupe prek iTunes [3].

Slika 2.11: iPhone 5s.

2.3 Razvoj

Leta 1999 je japonsko podjetje NTT DoCoMo izdalo prvi pametni mobilni telefon, razširil pa se je šele leta 2000. Leta 2012, je večina pametnih mobilnih telefonov že podpirala visoko hitrost mobilnega štiripasovnega 4G LTE omrežja, podpirali pa so tudi že senzorje gibanja in mobilno plačevanje. Proti koncu leta 2012 je imela pametni mobilni telefon že okrog milijarda uporabnikov [4] [5]. Od leta 2013 si je 65 % uporabnikov mobilne tehnologije že lastilo pametni mobilni telefon, začetek leta 2016 pa že 79 % samo v ZDA [6].

Poglavje 3 Izvorni razvoj aplikacij

Izvorno aplikacijo lahko naredimo na različnih mobilnih operacijskih sistemih, zato si bomo ogledali vse mobilne operacijske sisteme, to so Android, Windows Phone in iOS, programske jezike, v katerih so spisani in najbolj pogosto uporabljeno razvojno okolje.

3.1 Android

3.1.1 Android

Android je mobilni operacijski sistem, ki ga je razvilo podjetje Android Inc, od leta 2005 pa je v lasti Googla [7]. Predstavljen je bil leta 2007, prva različica Androida pa je prišla na trg leta 2008, in sicer na aparatu HTC Dream [8]. Je odprtokodni sistem, ki temelji na jedru Linux, napisan pa je v programskem jeziku Java. Ker je odprtokodni, je tudi brezplačen, kar predstavlja veliko prednost razvijalcem. Tako so razvojne skupine razvile tudi različico CyanogenMod, ki zagotavlja novejše različice Androida za starejše naprave, ki več ne prejemajo posodobitev [9] [10]. Seveda ne smemo pozabiti tudi na podjetja, ki vsako novo verzijo Androida še malo priredijo za svojo znamko telefona. Razpoznavni znak Androida je prikazan na sliki 3.1.

Slika 3.1: Android ikona.

Od julija 2013 je imel Google Play Store že več kot milijon Android aplikacij, maja 2013 pa so študije pokazale, da je 71 % razvijalcev že naredilo aplikacijo za Android, leta 2015 pa je več kot 40 % polno zaposlenih razvijalcev delalo na platformi Android. Konec septembra 2015 je Android imel več kot 1.4 bilijona aktivnih pametnih naprav z Android platformo [11] [12].

Zaradi preprostega osnovnega sistema in brezplačne uporabe se je Android razvijal do te mere, da ga nameščajo na različnih napravah, kot so pametna ura, pametni telefoni, pametne tablice, avtomobilski radiji, televizije itd.

Prednosti Androida: [13]

- na namizju nam ves čas prikazuje opozorila, če pa je aparat v mirovanju, se nam prižge LED lučka, ki utripa
- večopravilnost; v nekem trenutku imamo lahko odprtih več različnih aplikacij
- Google Android App Market, kjer imamo na voljo več tisoč aplikacij
- omogoča nam dostop do spleta in ustvarjanje WIFI točke
- samodejno posodabljanje prek interneta
- omogoča nalaganje različnih ROM-ov
- integriran z Google dejavnostmi
- brezplačen in odprtokodni sistem

Slabosti Androida: [13]

- različna velikost in resolucija ekranov, kar otežuje razvoj grafične oblike aplikacije
- za posodobitve in večino aplikacij potrebuje internet, ki pa ni dostopen povsod
- hitro ranljiv (hitro lahko dobi virus)
- zastonjske aplikacije vsebujejo veliko reklam
- zaradi aplikacij, ki tečejo v ozadju, se hitreje izprazni baterija
- s posodobitvami se povečajo zahteve in posledično začne delovati počasi

3.1.2 Razvoj

Ker je operacijski sistem Android nastal v programskem jeziku Java, bomo ta jezik opisali in si ogledali še njegovo najbolj pogosto uporabljeno razvojno okolje, to je Android Studio in Eclipse.

3.1.2.1 Java

Java je objektno usmerjen, prenosljivi programski jezik, ki ga je razvil James Gosling s sodelavci v podjetju Sum Microsystems. Razvita je bila leta 1991 še pod imenom OAK, kar pomeni hrast, razvili pa so jo kot menjavo za programski jezik C++ [14]. Sprva je bila mišljena za interaktivno televizijo, vendar je bilo to za takrat še prezahtevno [15]. Sintaksa je zelo podobna programskemu jeziku C in C++ [16]. Leta 1995 je bila izdana prva različica Jave 1.0, zadnja različica 8.0 pa je bila izdana leta 2014. Trenutno je na razpolago različica 8-update 111. Zaradi dobre varnosti so Javo kmalu po izdaji prve različice začeli uporabljati za spletne aplikacije, kar pa sprva sploh ni bilo mišljeno [17]. Javo vzdržuje in posodablja Oracle - Sun Microsystems.

Programski jezik Java je dobil ime po vrsti kave, kar lahko razberemo tudi iz logotipa, ki je prikazan na sliki 3.2.

Slika 3.2: Java logotip kave.

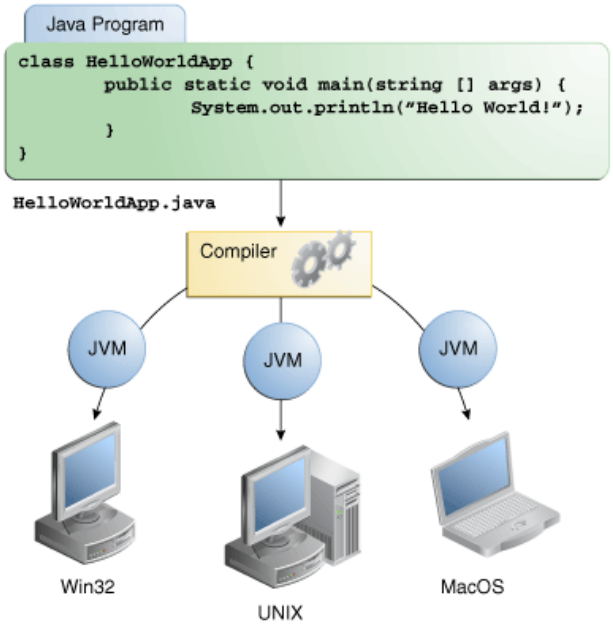
Poznamo 3 vrste Jave: [18]

- J2SE standardna različica Jave za osebne računalnike
- J2ME različica Jave za mini naprave (mobiteli, pametni televizorji …)
- J2EE poslovna različica Jave

Glavne prednosti Jave: [18]

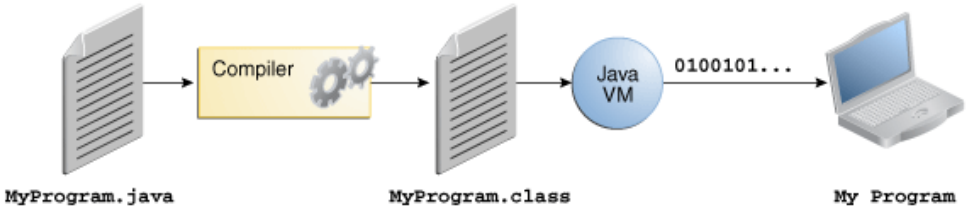
- objektno usmerjen jezik
- neodvisen od platforme
- primeren tudi za razvoj spletnih aplikacij
- varna
- relativno enostavna

Velika prednost Jave je, da je neodvisna od platforme oziroma kot pravijo za Javo: »write once, run anywhere,« kar pomeni, da omogoča programerjem, da lahko Javo poganjajo na različnih platformah [19], kar je prikazano na sliki 3.3.



Slika 3.3: Ista aplikacija je zmožna teči na različnih platformah.

Naš prvi program v programskem jeziku Java začnemo izdelovati tako, da vso izvorno kodo napišemo v nek urejevalnik (prikaz na sliki 3.5) in nato shranimo v datoteko s končnico `.java`. Ime datoteke se mora ujemati z imenom glavnega razreda. To datoteko moramo nato s pomočjo prevajalnika prevesti v višji programski jezik oziroma strojni jezik, tako da dobimo novo datoteko s končnico `.class`. Prikaz na sliki 3.6. Ta datoteka je nared za izvajanje [20]. Razvojni proces je prikazan na sliki 3.4.



Slika 3.4: Razvojni proces.

```
public class firstProgram{
    public static void main (String [] args){
        System.out.println("Java in the best!");
    }
}
```

Slika 3.5: Preprost program.

```
D:\Desktop>javac firstProgram.java

D:\Desktop>java firstProgram
Java in the best!

D:\Desktop>
```

Slika 3.6: Prevajanje in zagon programa.

3.1.2.2 Android studio

Android Studio je integrirano razvojno okolje IDE in je namenjeno za razvoj Android aplikacij, ki temelji na IntelliJ IDEA [21]. Logotip je prikazan na sliki 3.7. Izšel je 16. maja 2013, in sicer na Google I/O konferenci, prva stabilna različica pa je izšla šele junija 2014, verzije 1.0 [22]. Je prosto dostopen pod Apache licenco. Mišljen je bil kot zamenjava za Eclipse Android Development Tools (ADT) kot Googlov primarni IDE za razvoj Android aplikacij [23].

Na voljo je za 3 različne operacijske sisteme:

- Windows
- Linux
- Mac OS X

Slika 3.7 Android Studio logotip.

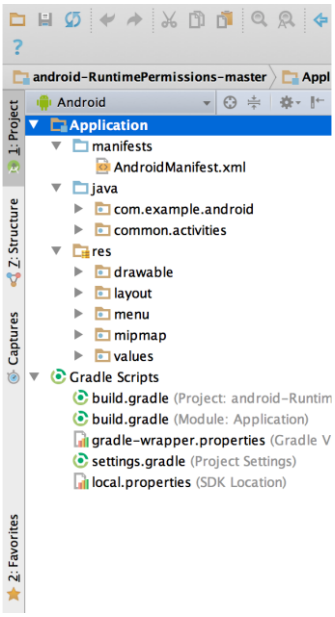
Android Studio nam ponuja veliko funkcij, ki povečujejo produktivnost pri izdelavi aplikacij, kot so: [24]

- prilagodljiv Gradle sistem
- hiter emulator z veliko funkcijami
- enotno okolje, v katerem lahko razvijamo aplikacije za vse Android naprave
- spremembe se samodejno prevajajo brez gradnje novega APK-ja
- samodejno dopolnjuje ukaze oziroma nam jih ponuja
- integriran GitHub
- obsežna orodja za testiranje
- vgrajena podpora za Google Cloud Platform, kar omogoča lažjo integracijo Google Cloud Messaging

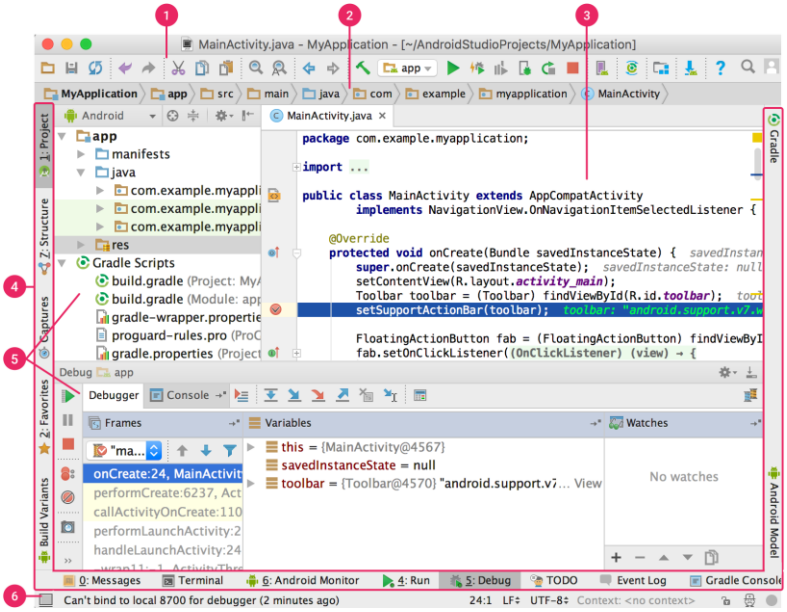
Projekt je zgrajen vedno iz treh glavnih delov: [24]

- Manifest: vsebuje AndroidManifest.xml datoteko
- Java: vsebuje Java kodo, vključno z JUnit testi
- Res: vsebuje vse vire brez kode, kot je na primer XML postavitev, UI strings in slike

  Zgradba dokumenta je prikazana na sliki 3.8, na sliki 3.9 pa imamo izgled uporabniškega vmesnika.



Slika 3.8: Zgradba projekta.



Slika 3.9: Izgled uporabniškega vmesnika.

- 1 - Orodna vrstica
- 2 - Navigacijska vrstica, za sprehajanje po datotekah
- 3 - Okno za urejanje in pisanje kode

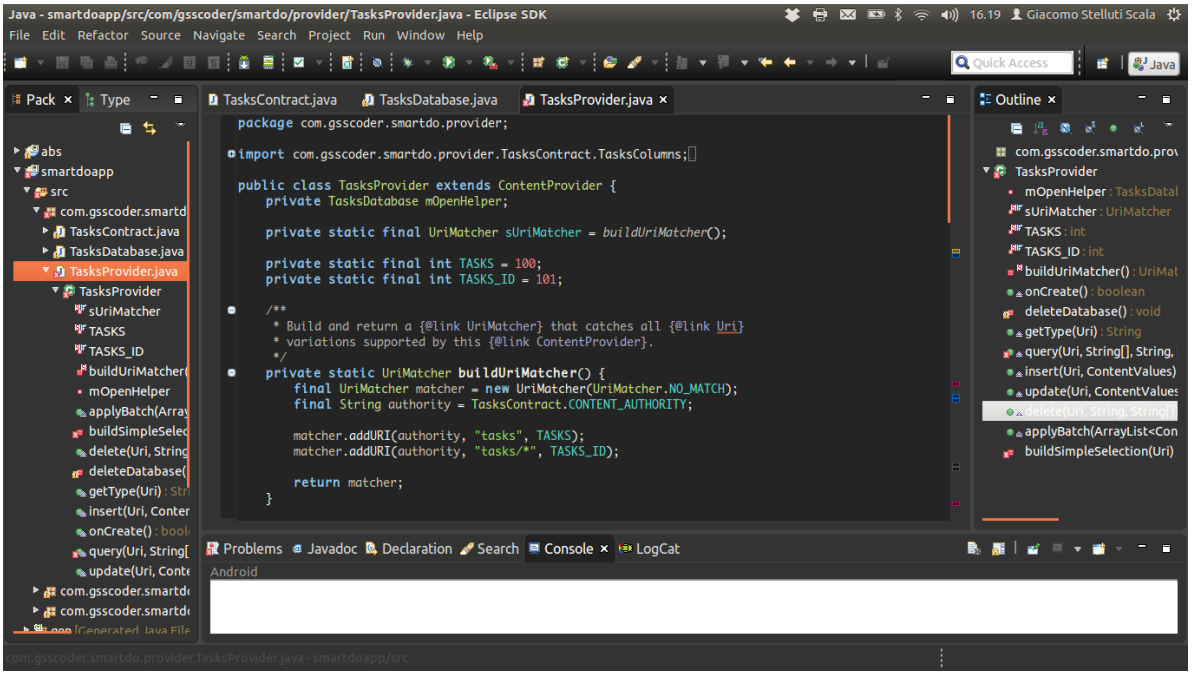
- 4 - Vrstica z orodji je na zunanji strani okna, ki vsebuje gumbe za razširitev posameznih orodij
- 5 - Okno z orodji omogoča dostop do različnih opravil, kot je vodenje projekta, iskanje, verzija itd.
- 6 - Statusna vrstica, ki nam kaže status našega projekta in trenutne napake ali opozorila

3.1.2.3 Eclipse

Eclipse je integrirano razvojno okolje IDE, ki se ga uporablja za programiranje. Eclipse je večinoma napisan v programskem jeziku Java, za katerega je tudi namenjen, lahko pa se ga uporablja tudi za razvoj aplikacij v drugih programskih jezikih z uporabo različnih vtičnikov, kot so Ada, ABAP, C, C++, COBOL, D, Fortran, Haskell, JavaScript, Julia, Lasso, Lua, NATURAL, Perl, PHP, Prolog, Python, R, Ruby, Rust, Scala, Clojure, Groovy, Scheme in Erlang [25].

Slika 3.10: Eclipse logotip.

Sama osnova razvojnega okolja (slika 3.11) izvira iz IBM VisualAge. Je prost in odprtokodni razvojni sistem [26]. Logotip je prikazan na sliki 3.10.



Slika 3.11: Razvojno okolje Eclipse.

Eclipse je leta 2014 zasedal 48 % IDE trga [25].

Prednosti uporabe Eclipsa:

- zastonj in odprtokodni
- podpira veliko programskih jezikov
- integrirani testi JUnit
- samodejno posodabljanje
- samodejno dopolnjevanje kode
- refactoring
- samodejno iskanje sintaktičnih napak

3.2 Windows Phone

3.2.1 Windows Phone

Windows Phone je mobilni operacijski sistem, ki ga je Microsoft razvil za pametne telefone. Izšel je kot nadomestek prejšnjega sistema Windows Mobile, ki je bil v osnovi namenjen za podjetniški trg, Windows Phone pa je namenjen potrošniškemu trgu. Izšel je oktobra 2010, in sicer pod imenom Windows Phone 7 [27]. Windows Phone operacijski sistem je povezan tudi z drugimi Microsoftovimi aplikacijami, kot so Bing, SkyDrive, Xbox in pa Microsoft Office Mobile. Leta 2011 je Nokia začela nameščati Windows Phone na svoje pametne telefone. Najbolj znana serija telefonov je Nokia Lumia, ki je leta 2013 predstavljala 83,3 % prodanega deleža Windows Phone uporabnikov [28].

3.2.2 Microsoft in Nokia

11. februarja 2011 sta Steve Ballmer (Microsoft CEO) in Stepehl Elop (Nokia CEO) na tiskovni konferenci v Londonu napovedala partnerstvo med družbama. Nokia bi začela opuščati trenutni operacijski sistem Symbian in začela uvajati Windows Phone [29]. Osredotočeni so bili, da naredijo nov mobilni operacijski sistem, ki bi postal konkurenca iOS-u in Androidu oziroma kot so dejali *it is now a three horse race*, kar pomeni, da je sedaj dirka treh konjev [30]. Elop se je odločil za Windows Phone namesto za Android, in sicer zaradi ene same besede in to je diferenciacija. Zaradi poznega pristopa k Androidu bi imeli hude probleme z diferenciacijo [31].

Nekaj glavnih integracij Nokiinih in Microsoftovih storitev: [32]

- Bing bo postal glavni raziskovalec na Nokiah
- integracija Nokia Maps z Bing Maps
- integracija Nokia Ovi Store z Windows Phone Store

3.2.3 Verzije

3.2.3.1 Windows Phone 7

15. februarja 2010 je bil napovedan na Mobile World kongresu v Barceloni, javno pa je izšel 8. novembra 2010 v Združenih državah Amerike [33].

Leta 2011 je izšla verzija 7.5 Mango, ki je že vsebovala Internet Explorer 9, ki je podpiral iste stvari kot namizna različica, na primer večopravilnost, imel je dostop do Windows Live SkyDrive in pa Twitter [34].

Leta 2012 je izšla še manjša posodobitev pod imenom Tango, s katerim so odpravili nekaj nepravilnosti, znižali pa so tudi minimalne strojne zahteve, tako da so ga lahko naložili tudi uporabniki z malo slabšimi mobilnimi aparati [35].

Leta 2013 je izšla še zadnja različica, in sicer 7.8. Izdali so jo, ker Windows Phone 8 ni bilo mogoče namestiti na aparatih, ki so imeli nameščen Windows Phone 7 in so zato na tej različici dodali nekaj funkcionalnosti iz različice 8, kot so na primer več različnih ozadij ali pa različne barve tem [36]. Logotip je prikazan na sliki 3.12.

Slika 3.12: Windows Phone 7 logotip.

3.2.3.2 Windows Phone 8

29. oktobra 2012 je Microsoft izdal drugo generacijo mobilnega operacijskega sistema, in sicer Windows Phone 8. Logotip je prikazan na sliki 3.13. Dosegli so, da so aplikacije narejene za Windows Phone 8 delovale tudi na Windows Phone 7, in sicer tako, da so v novem operacijskem sistemu vgradili jedro iz sistema Windows NT [37].

Slika 3.13: Windows Phone 8 logotip.

3.2.3.3 Windows Phone 8.1

2. aprila 2014 je bil napovedan Windows Phone 8.1, 10. aprila pa je bila izdana prva različica, namenjena za razvijalce [38]. Vseboval je že center za obvestila, Internet Explorer 11, ki je imel samodejno sinhronizacijo z Windowsom 8.1, med drugim pa tudi možnosti dodajanja aplikacij na začetni menu. Uvedli so še zahtevo za gumb za `kamero, nazaj, začetek in iskanje`. Dodali so tudi glasovni asistent Cortana, ki je nadomestila Bing [39]. Logotip je prikazan na sliki 3.14.

Slika 3.14: Windows Phone 8.1 logotip.

3.2.3.4 Windows 10 Mobile

21. januarja 2015 je bila izdana nova različica, in sicer za vse mobilne operacijske sisteme na tablicah in pametnih telefonih, ki temeljijo na ARM arhitekturi. Logotip je prikazan na sliki 3.15. Glavni cilj je bil čim bolj združiti Windows 10 Mobile z Windows 10 na računalnikih po funkcionalnostih, če torej priključimo na pametno tablico še en zaslon in dodamo miško in tipkovnico, bi morali imeti večino funkcionalnosti, ki jih podpira sam računalnik. Dodali so tudi Skype, posodobili Office in dodali Microsoft Edge [40] [41].

Slika 3.15: Windows Phone 10 logotip.

3.2.4 Uporabniški vmesnik

Uporabniški vmesnik, prikazan na sliki 3.16, temelji na Microsoft Metro obliki, ideja pa izhaja iz uporabniškega vmesnika na napravi Zune HD [42]. Začetni zaslon je sestavljen iz premikajočih se živih ploščic, ki so bile tudi navdih za Windows 8. Te ploščice so povezave do različnih aplikacij, funkcij itd. Velikost ploščic in oblika se samodejno menjuje, npr. pri prejeti e-pošti se ploščica samodejno spremeni in nam da vedeti, da smo dobili novo e-sporočilo.

Windows Phone podpira večopravilnost in zazna več dotikov naenkrat. Aparati imajo vgrajene OLED zaslone [43]. To pomeni, da črne pike na zaslonu ne oddajajo svetlobe, zato je tudi privzeta barva črna, saj nam s tem podaljša življenjsko dobo baterije. Seveda lahko po želji spremenimo barvo ikon, barvo teme, ozadje namizja …

Slika 3.16: Windows Phone uporabniški vmesnik.

3.2.5 Razvoj

Windows Phone je napisan v programskem jeziku C#, zato si bomo ogledali ta jezik in razvojno okolje Visual Studio.

3.2.5.1 C#

C# je več paradigni programski jezik. Obsega močno tipizacijo ter imperativno, deklarativno, funkcijsko, generično, komponentno orientirano in objektno orientirano programiranje. Ima avtomatično upravljanje s pomnilnikom, kar pomeni, da čistilnik spomina poskuša brisati vse objekte, ki nimajo več povezav, so nedostopni oziroma jih program ne potrebuje več. Vsebuje tudi zmožnost reflekcije [44].

Razvil ga je Microsoft v okviru razvoja .NET ogrodja, kasneje pa je bil odobren kot standard s strani organizacij ECMA-334 in pa ISO/IEC 23270:2006 [44].

Januarja 1999 je Anders Hejlsberg izoblikoval ekipo za razvoj novega jezika. Sprva so jezik hoteli poimenovati COOL, kar pa je kratica za C Object Oriented Language, kar v slovenščini pomeni C Objektno Usmerjen Jezik. Jezik so nato leta 2000 spremenili v C# zaradi razlogov, ki so bili takrat na tržišču [45].

Slika 3.17: C# logotip.

Ime C# izhaja iz notnega zapisa, kjer # viša noto za pol tona. Podobno kot C++, kjer ++ pomeni povečavo za 1. Ker je C# nadgradnja C++, lahko C++ dodamo še dva plusa in če postavimo štiri pluse v mrežo dva po dva, dobimo znak # [44]. Prikaz logotipa na sliki 3.17.

C# se zgleduje po programskih jezikih C, C++ in pa po Javi. Razvit je bil z namenom, da pobere najboljše stvari teh treh jezikov in popravi pomanjkljivosti. Eden izmed pomembnih primerov so generični tipi, ki so še bolj dodelani kot v Javi, pridobimo pa bolj optimizirano strojno kodo, večjo varnost pri definiciji tipov, ter striktno upoštevanje kovariančnosti in kontravariančnosti [44].

Smernice pri razvoju C# jezika: [44]

- preprost, modern, univerzalen, objektno usmerjen

- močan poudarek na preverjanju podatkovnih tipov, preverjanju velikosti tabel, odkrivanju poskusov uporabe ne deklariranih podatkovnih tipov in avtomatskem zbiranju in brisanju smeti

- namenjen za razvoj programskih komponent, primernih za uporabo v distribuiranih okoljih

- zelo pomembna je prenosljivost programske kode

- podpora za internacionalizacijo

- primeren za razvijanje aplikacij za velike, visoko razvite sisteme, male sisteme, pa vse do majhnih namenskih funkcij

- čeprav so C# aplikacije ekonomične glede spomina in zahtev procesorske moči, jezik ni bil namenjen kot tekmovalec jeziku C v uspešnosti in velikosti

Na sliki 3.19 so prikazane verzije C#, slika 3.18 pa prikazuje primer kode spisane v C#.

```
using System;

class Program
{
    static void Main(String[] args)
    {
        Console.WriteLine("Hello, world!");
    }
}
```

Slika 3.18: Primer izpisa »Hello, world!«

Version	Language specification			Date	.NET Framework	Visual Studio
	ECMA	ISO/IEC	Microsoft			
C# 1.0	December 2002	April 2003	January 2002	January 2002	.NET Framework 1.0	Visual Studio .NET 2002
C# 1.1 and 1.2			October 2003	April 2003	.NET Framework 1.1	Visual Studio .NET 2003
C# 2.0	June 2006	September 2006	September 2005[note 3]	November 2005	.NET Framework 2.0	Visual Studio 2005
C# 3.0	None[note 4]		August 2007	November 2007	.NET Framework 2.0 (Except LINQ/Query Extensions)[35] .NET Framework 3.0 (Except LINQ/Query Extensions)[35] .NET Framework 3.5	Visual Studio 2008 Visual Studio 2010
C# 4.0			April 2010	April 2010	.NET Framework 4	Visual Studio 2010
C# 5.0	In Progress[36]	None[note 4]	June 2013	August 2012	.NET Framework 4.5	Visual Studio 2012 Visual Studio 2013
C# 6.0	None[note 4]		None	July 2015	.NET Framework 4.6	Visual Studio 2015

Slika 3.19: Verzije C#.

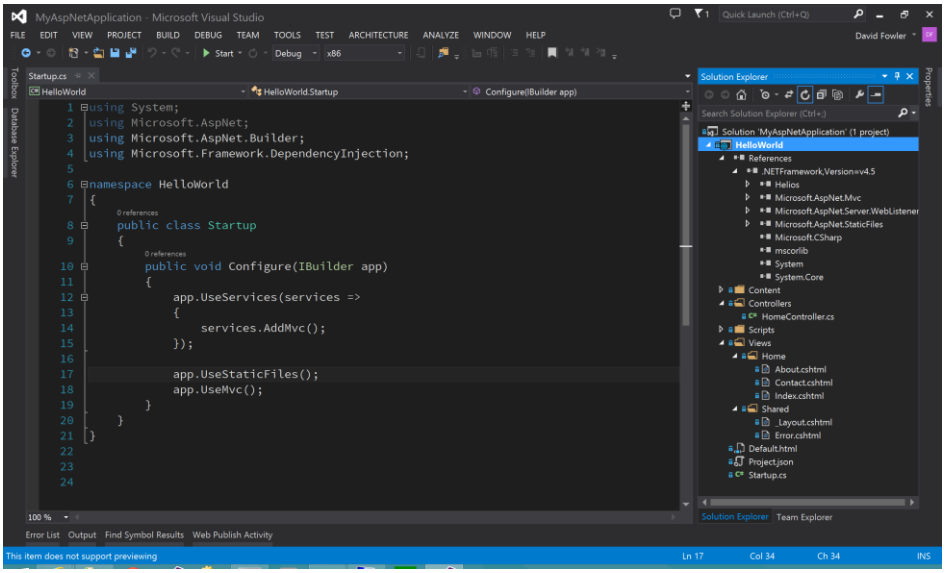
3.2.5.2 Visual Studio

Visual Studio je integrirano razvojno okolje, ki ga je razvilo podjetje Microsoft. Logotip lahko vidimo na sliki 3.20. Namenjeno je za razvoj računalniških programov za sistem Windows in pa tudi spletnih strani, spletnih aplikacij, spletnih storitev in aplikacij, ki stojijo na osnovi ogrodja .NET. Vsebuje tudi urejevalnik kode, prikazan na sliki 3.22, ki podpira samo dopolnjevanje kode oziroma ukazov. Integriran je tudi iskalnik napak in razhroščevalnik. Vgrajena ima tudi orodja, s katerimi lahko sestavljamo grafični uporabniški vmesnik, podporo za oblikovanje spletnih strani in grafični prikaz podatkovne baze. Podpira tudi raznorazne vtičnike, ki izboljšujejo delovanje na vseh ravneh. Primer vtičnika bi bil Xamarin, preko katerega lahko razvijemo aplikacijo za Android in iOS. Visual Studio ne podpira samo jezik C#, ampak tudi C, C++, Python, Ruby, Node.js, M, XML/XSLT, HTML/XHTML, JavaScript, CSS in Javo [46] [47]. Na sliki 3.21 imamo tudi naštete različne verzije Visual Studia.

Slika 3.20: Visual Studio logotip.

Product name	Codename	Version number	Version of cl.exe	Supported .NET Framework versions	Release date
Visual Studio 97	*Boston*	5.0	N/A	N/A	February 1997
Visual Studio 6.0	*Aspen*	6.0	12.00	N/A	June 1998
Visual Studio .NET (2002)	*Rainier*	7.0	13.00	1.0	February 13, 2002
Visual Studio .NET 2003	*Everett*	7.1	13.10	1.1	April 24, 2003
Visual Studio 2005	*Whidbey*	8.0	14.00	2.0, 3.0	November 7, 2005
Visual Studio 2008	*Orcas*	9.0	15.00	2.0, 3.0, 3.5	November 19, 2007
Visual Studio 2010	*Dev10/Rosario*	10.0	16.00	2.0 – 4.0	April 12, 2010
Visual Studio 2012	*Dev11*	11.0	17.00	2.0 – 4.5.2	September 12, 2012
Visual Studio 2013	*Dev12*	12.0	18.00	2.0 – 4.5.2	October 17, 2013
Visual Studio 2015	*Dev14*	14.0	19.00	2.0 – 4.6	July 20, 2015
Visual Studio 2017[64]	*Dev15*	15.0	19.10	2.0 – 4.6.2; Core 1.0	TBA

Slika 3.21: Visual Studio verzije.



Slika 3.22: Izgled Visual Studio.

3.3 IOS

3.3.1 IOS

iOS, nekdanji iPhone OS, je mobilni operacijski sistem, ki ga je razvil Apple za svoje naprave. iOS je nameščen na vseh Applovih izdelkih, kar pomeni na iPadu, iPhonu in iPod touche. Na vsaki napravi lahko na zadnji strani vidimo logotip iOS-a, ki je prikazan na sliki 3.23. Prvi najbolj priljubljen mobilni operacijski sistem je Android, takoj za njim pa je iOS, če se osredotočimo na pametne telefone in pametne tablice, s tem da je na tablicah najbolj popularen Android šele od leta 2013 [48].

iOS je bil predstavljen leta 2007, in sicer le za iPhone, septembra 2007 so ga predstavili še za iPod touche in januarja 2010 še za iPad. Junija 2016 je iOS imel na voljo že 2 milijona aplikacij, od tega jih je bilo okrog 725 tisoč narejenih v izvorni kodi [49].

Slika 3.23: iOS logotip.

iOS podpira zaznavo več dotikov naenkrat, vsebuje pa še drsnike, stikala in gumbe. Zaznati zna tresenje pametne mobilne naprave, pozicijo telefona, da lahko prilagodi pokončni oziroma ležeči način, novejši pa imajo vgrajene tudi GPS senzorje, višinomer ipd. [49].

Novejše različice sistema vsebujejo iCloud. To je program, ki sinhronizira podatke na vseh Applovih napravah, tako da lahko do podatkov dostopamo na vseh napravah [49].

iOS ideje so se začele leta 2005, ko je Steve Jobs, ustanovitelj podjetja Apple, začel načrtovati iPhone. Na voljo je imel 2 možnosti, ali zmanjšati sistem iz Mac-a ali pa povečati sistem iz iPad-a. Tako je na internem natečaju, na katerem sta bili Macintosh in iPod ekipa pod vodstvom Scott Forstall in Tony Fadell, bil razvit iPhone OS. Forstall je bil kasneje tudi odgovoren za razvoj App Store in iTunes [49].

Verzije:

- iPhone OS 1
- iPhone OS 2
- iPhone OS 3
- iOS 4
- iOS 5
- iOS 6
- iOS 7
- iOS 8
- iOS 9
- iOS 10

Začetni zaslon, ki se nam prikaže ob pritisku gumba `domov` po odklepu telefona in prižigu telefona, je sestavljen iz več delov. Na vrhnji strani je statusna vrstica, ki nam prikazuje podatke, kot so ura, moč signala ali pa stanje baterije. Spodnja stran vsebuje *dock* vrstico, kjer lahko dodajamo različne aplikacije po svoji želji, običajno so aplikacije, ki jih največkrat uporabljamo. Preostali del zaslona je namenjen različnim aplikacijam, povezavam do aplikacij itd. Ozadje zaslona lahko po želji spreminjamo. Od razvoja pa vse do verzije 7 so uporabljali

Helvetica pisavo, z različico iOS 7 pa je prišla tudi možnost, da uporabnik preklopi na pisavo Neue Bold, iOS 9 pa je prešel na pisavo San Francisco [49].

Večopravilnost je prišla z izdajo verzije iOS 4, in sicer Junija 2010. Opravila, ki jih je podpiral iOS 4, so bila: [49]

- zvok v ozadju – aplikacija je tekla v ozadju, vse dokler se je predvajala glasba oziroma videi
- prenos zvoka preko IP – aplikacija se je samodejno ugasnila, če ni bilo nobenega klica
- lokacija v ozadju – aplikacija je bila obveščena, če se je spremenila lokacija
- potisna obvestila
- lokalna obvestila – aplikacije, ki so bile vezane na čas
- dokončevanje opravil – aplikacija je prosila sistem za dodatni čas za dokončevanje trenutne naloge
- hitro preklapljanje med aplikacijami – aplikacija ni izvedla nobene kode in se je lahko odstranila iz pomnilnika ob vsakem času

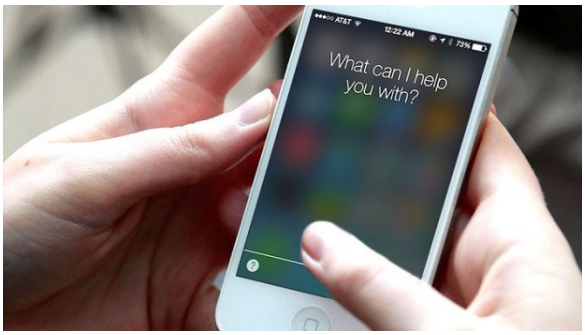
iOS 5 je dodal še tri nova opravila: [49]

- kiosk za časopise – aplikacija je lahko v ozadju prenesla podatke, da so lahko bili na voljo za uporabnika ob vsakem času
- zunanja dodatna oprema – aplikacija je lahko komunicirala z drugo napravo in prenašala podatke ob vsakem času
- Bluetooth – aplikacija je lahko v ozadju upravljala z Bluetooth tehnologijo

S prihodom iOS 7 je večopravilnost prišla v vse stvari, torej so vse aplikacije lahko tekle v ozadju [49], kar je prikazano na sliki 3.24.

Slika 3.24: iOS večopravilnost.

S prihodom iOS4 plus je prišla tudi aplikacija Siri (slika3.25). To je osebni asistent, ki ga uporabljamo prek zvoka oziroma pogovora z njim. Zna napisati sporočilo, poklicati osebo, odpreti aplikacijo, brskati po spletu, poiskati lokacijo in nas usmerjati in odgovoriti na osnovna vprašanja. S prihodom iOS 7 je bil posodobljen tudi Siri in s tem pridobil na večji hitrosti odziva, vseboval je že Wikipedijo, Bing in Twitter, spremenili pa so mu tudi zvok, in sicer v bolj človeškega [49].



Slika 3.25: Siri.

3.3.2 Razvoj

iOS je bil razvit v programskem jeziku C, ki ga bomo opisali in si ogledali še razvojno okolje Xcode.

3.3.2.1 C

Med letoma 1969 in 1973 je Dennis Ritchie v AT&T Bellovih laboratorijih razvil programski jezik C, katerega logotip je prikazan na sliki 3.26. Skupaj s Kenom Thompsonom sta najprej razvila prevajalnik za Unix operacijski sistem v zbirnem jeziku. C se je začel hitro pojavljati tudi v drugih operacijskih sistemih. Uporablja se ga od super računalnikov, mikro krmilnikov, pa vse do vgrajenih sistemov [50].

Slika 3.26 Programski jezik C.

Programski jezik C je nizkonivojski imperativni standardizirani računalniški programski jezik tretje generacije. Podpira strukturalno programiranje, leksično območje spremenljivke in rekurzijo, statični sistem tipov pa preprečuje mnogo nenameravanih operacij. [51].

Je imperativni jezik, ki je bil izdelan za prevajanje na preprostem prevajalniku in zagotavljanju nizkonivojskega dostopa do polnilnika [51].

Skupaj z Javo, C++ in PHP je eden izmed najbolj priljubljenih programskih jezikov, ni pa najprimernejši za učenje programiranja, čeprav večkrat zaide tudi tja [52].

C++ je razširitev jezika C.

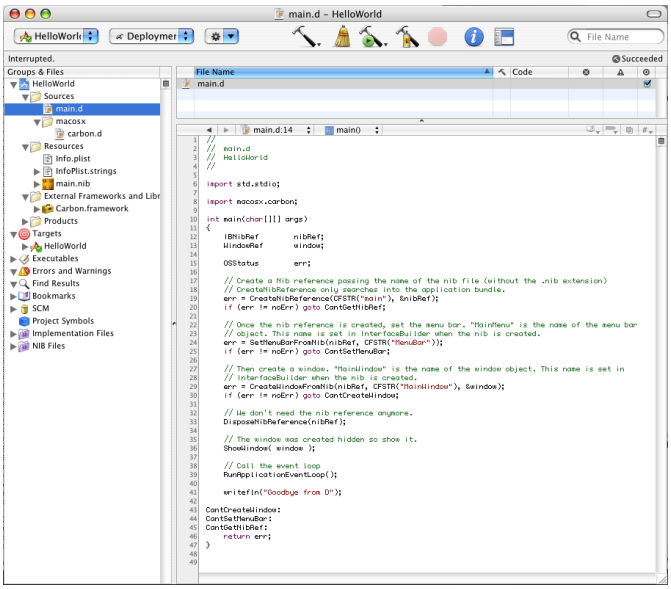
3.3.2.2 Xcode

Xcode je integrirano razvojno okolje, ki ga je razvil Apple, z njim pa lahko ustvarjamo aplikacije za Mac, iPhone, iPad, Apple Watch in Apple TV. Izšel je leta 2007, trenutno pa imamo zunaj že različico 8, ki jo lahko prenesemo zastonj iz App Store, preko spletne strani Apple Developer pa lahko prenesemo tudi vse ostale verzije [53]. Logotip lahko vidimo na sliki 3.27.

Slika 3.27: Xcode logotip.

Xcode podpira izvorne kode različnih programskih jezikov, to so C, C++, Objective-C, Objective-C++, Java, AppleScript, Python, Ruby, ResEdit (Rez), Swift, Pascal, Ada, C#, Perl, in D [53].

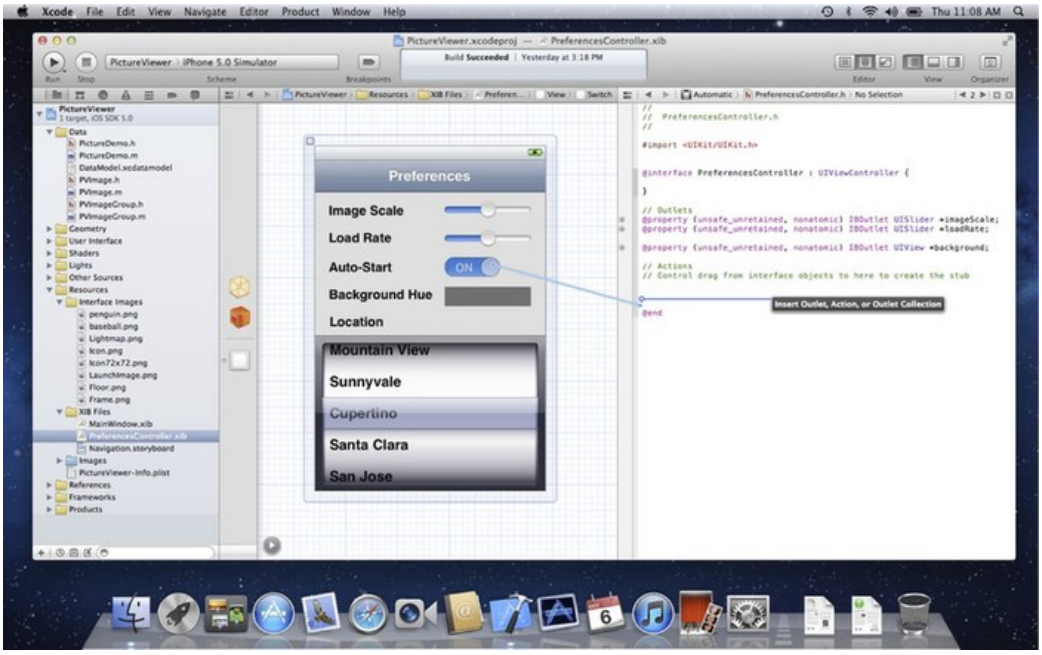
Xcode ponuja razvijanje uporabniškega vmesnika (slika 3.28), pisanje kode in razhroščevanje kode vse na enem mestu. Zna analizirati projekt in tako zaznati logične in sintaktične napake [54].



Slika 3.28: Xcode izgled.

Xcode lahko namestimo samo na Mac računalnikih, kar je kar velik minus [54]. Lahko si pa na Windows namestimo program VMWare ali pa VirtualBox in si virtualno namestimo OS X Lion.

Xcode nam ponuja tudi izdelavo uporabniškega vmesnika brez pisanja kode, ampak samo s kliki. Ko uporabniški vmesnik grafično dokončamo, ga lahko v urejevalniku še dokončamo z vsemi funkcijami. Omogoča nam tudi delo v dveh oknih, in sicer eno okno za grafični urejevalnik, drugo pa za izvorno kodo [55]. Delo v dveh oknih je prikazano na sliki 3.29.



Slika 3.29: Xcode, delo v dveh oknih.

Poglavje 4 Hibridni razvoj aplikacij

Hibridna aplikacija je lahko narejena na več različnih načinov. V nadaljevanju si bomo ogledali PhoneGap, Xamarin in Ionic ter opisali programske jezike, ki so potrebni za vsak pristop in najbolj pogosto uporabljeno oziroma priporočljivo razvojno okolje.

4.1 PhoneGap

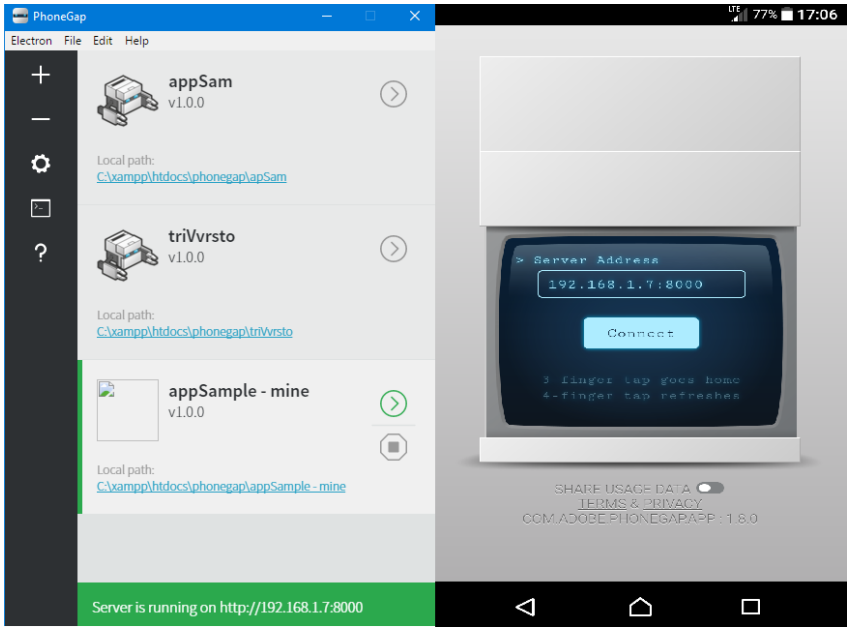
4.1.1 PhoneGap

Za izdelavo mobilne aplikacije je potrebno znati več različnih programskih jezikov in uporabljati različna razvojna okolja. PhoneGap je orodje, s katerim lahko s porabo standardnih spletnih tehnologij ustvarimo aplikacije, ki jih lahko poganjamo na različnih mobilnih platformah, kot so Android, WindowsPhone in iOS [56]. Uporablja jezike HTML5, JavaScript in CSS3, s katerimi ustvarimo hibridno aplikacijo, ki ni povsem spletna aplikacija, saj ima tudi dostop do API naprave.

PhoneGap je razvilo podjetje Nitobi, od leta 2011 pa je lastnik Adobe Systems. Programska oprema se je najprej imenovala PhoneGap, nato Apache Callback, trenutno pa uporablja ime Apache Cordova. Je odprtokodna programska oprema [57].

4.1.2 Namestitev

Za pravilni razvoj PhoneGap aplikacije najprej potrebujemo razvojno okolje. Iz uradne spletne strani PhoneGap-a je potrebno prenesti dve aplikaciji. Prvo aplikacijo namestimo na računalnik, drugo pa na pametno mobilno napravo, za katero izdelujemo aplikacijo. Prikaz aplikacij si lahko ogledamo na sliki 4.1. Aplikacijo za mobilno napravo lahko dobimo tudi v *Trgovini play*. Ko imamo aplikacije nameščene, je potrebno ustvariti nov projekt s pritiskom na gumb `plus`. Ko je projekt ustvarjen, ga zaženemo s pomočjo gumba, ki vsebuje puščico v desno smer. Nato zaženemo še aplikacijo na mobilni napravi, kjer je potrebno vnesti naslov serverja. Ta naslov najdemo na dnu aplikacije z računalnika. S pritiskom na tipko `Connect` se aplikacija zažene. Potrebno je paziti, da imamo računalnik in mobilno napravo v istem omrežju, sicer to ne deluje [56].



Slika 4.1 Levo Aplikacija za računalnik, desno za mobilno napravo.

4.1.3 Razvoj

Pri razvoju aplikacije v PhoneGap-u lahko uporabljamo več različnih ogrodij in programskih jezikov. Ogledali si bomo ogrodje jQuery Mobile in programske jezike HTML, CSS in JavaScript.

4.1.3.1 jQuery Mobile

Ogrodje jQuery Mobile temelji na standardu HTML5 in tehnologiji JavaScript. Uporablja se ga za izdelavo hibridnih mobilnih aplikacij. Vnaprej ima pripravljenih veliko komponent, kot so gumbi, razne tabele, znaki za informacije, različna opozorila itd. S pomočjo ogrodja PhoneGap lahko aplikacijo zapakiramo in zaženemo na mobilnih napravah [57].

4.1.3.2 HTML

HTML ali Hyper Text Markup Language je spletni oziroma označevalni jezik. Uporabljamo ga za izdelavo osnovne spletne strani. Iz njega lahko določimo zgradbo dokumenta. Sestavljen je iz značk, okrog značke pa imam špičaste oklepaje. Značke so vedno v parih; prva pomeni začetek nekega dela, ukaza, druga pa konec. Značka za konec ima vedno za špičastim oklepajem znak, na primer `</body>`. Lahko se tudi gnezdijo [58].

4.1.3.3 CSS

CSS oziroma Cascading Style Sheets je spletni jezik, ki dopolnjuje jezik HTML. Z njim lahko določamo, kako naj spletna stran izgleda, HTML-ju dodamo stil in s tem določimo, kako naj se prikaže na spletni strani. Spreminjamo lahko barve, postavitve, velikosti itd. [59].

4.1.3.4 Javascript

Javascript, skrajšana oblika JS, je objektni skriptni jezik, namenjen za izdelavo interaktivnih spletnih strani. Je zelo podoben Javi, ampak ni bil razvit v odvisnosti od le-te. Zelo dobro sodeluje z jezikom HTML, saj se poveže z njegovimi objekti in zagotovi nadzor nad njimi [60].

4.2 Xamarin

4.2.1 Xamarin

Xamarin je orodje za izdelavo hibridnih mobilnih aplikacij. Logotip je prikazan na sliki 4.2. To je odprtokodni sistem za izdelavo aplikacij za platforme Android, iOS in Windows Phone. Je edinstven, saj za izdelavo aplikacij uporablja samo en programski jezik in sicer programski jezik C#. Lastnik orodja je Microsoft. Za izdelavo mobilne aplikacije običajno uporabljamo Visual Studio. Pri obliki aplikacije si lahko zelo pomagamo z Visual Studiem, saj nam izgled aplikacije ni potrebno sprogramirati, ampak lahko kar s klikom izbiramo elemente za našo aplikacijo [61].

Xamarin nam ponuja dva izdelka in sicer Xamarin.iOS in Xamarin.Android. Oba sta zgrajena na vrhu »Mono« odprtokodne različice .NET framework, ki temelji na objavljenih standardih. Aplikacije za iOS prevajalnik pretvori vse do zbirnega jezika ARM, aplikacije za Android pa do »Intermediate Language« IL. Xamarin sam poskrbi za dodeljevanje pomnilnika, brisanje smeti itd. [61].

Slika 4.2 Xamarin

4.3 Ionic

4.3.1 Ionic

Ionic je odprtokodno orodje za izdelavo mobilnih hibridnih aplikacij, ki uporablja jezik HTML5. Ker uporablja HTML5 ogrodje, potrebuje še ali Cordovo ali pa PhoneGap, da lahko steče na mobilni platformi. Podobno kot jQuery Mobile tudi Ionic uporabljamo bolj za razvijanje samega izgleda oziroma uporabniškega vmesnika. Zraven jezika HTML5 lahko uporabljamo tudi HTML, CSS JavaScript in AngulaeJS, kot da bi ustvarili spletno stran, le da tu ustvarjamo samostojno aplikacijo [62]. Logotip je prikazan na sliki 4.3.

Ionic je ustanovilo podjetje Drifty Co. leta 2012, natančneje Max Lynch, Ben Sperry in Adam Bradley. Ustvarjen je bil, da bi omogočili spletnim programerjem boljši in lažji način razvoja aplikacij. Do leta 2015 je bilo razvitih že več kot 1,3 milijona aplikacij [62].

Slika 4.3 Ionic

Poglavje 5 Razvoj aplikacije

Aplikacijo, ki bo vsebovala vklop kamere, igro tri v vrsto, kačo, ki se premika po zaslonu, vklop različnih vibracij mobilne naprave, lastnosti, ki jih lahko dobimo o bateriji, čas merjenja različnih ugnezdenih zank, prikaz slik in označevanja besedila, objekte, ki se bodo shranjevali v pomnilnik in datoteko, rekurzijo in iskanje v binarnem drevesu in povezanem seznamu ter igranje klavirja, bomo razvili v hibridnem in nato še v izvornem načinu. Vsak del je drugačen od drugega in je namenjen za testiranje, kako se bo obnesel v hibridnem oziroma izvornem razvoju.

5.1 Hibridni razvoj aplikacije za Android v PhoneGapu

Kot smo že zgoraj zapisali, bomo predstavili hibridni razvoj aplikacije po delih z orodjem PhoneGap.

5.1.1 Sestava aplikacije

Aplikacija, narejena z orodjem PhoneGap, ima več različnih map. Glavna mapa je `www`, v kateri imamo mapo `css`, ki vsebuje datoteko, ki določa izgled aplikacije, mapo `img`, v kateri imamo slike, mapo `js`, v kateri je datoteka ki določa funkcionalnost aplikacije, mapa `sound`, v kateri imamo zvočne posnetke in pa še datoteka `.html`, v kateri je določena zgradba aplikacije.

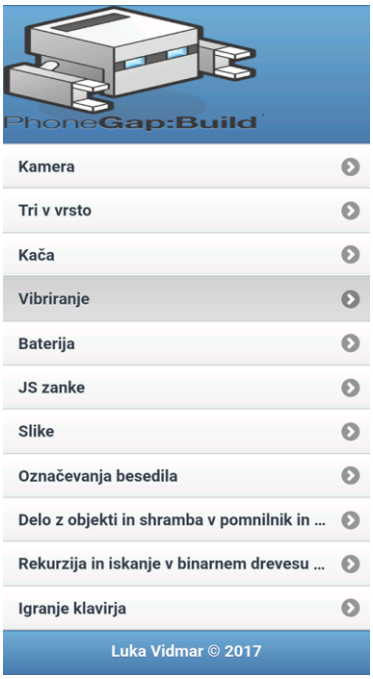
5.1.1.1 Izgled prve strani aplikacije

Vsaka stran v aplikaciji je zgrajena iz glave, vsebine in noge. V naši aplikaciji imamo na prvi strani v glavi sliko PhoneGap-a, v ostalih straneh pa ime trenutnega dela aplikacije. V nogi je napis ime in priimek ustvarjalca aplikacije in letnica nastanka aplikacije. Sestavo aplikacije lahko vidimo na sliki 5.1.

```
<div id="page" data-role="page" data-theme="b">
    <div data-role="header" data-theme="b">
        <div class="logo"><img src="img/phonegap.png" width="250" height="130" alt=""/></div>
    </div>
    <div data-role="content">
        <ul data-role="listview" data-theme="c">
            <li><a href="#camera">Kamera</a></li>
            <li><a href="#triVvrsto">Tri v vrsto</a></li>
            <li><a href="#kaca">Kača</a></li>
            <li><a href="#dregljaji">Dregljaji</a></li>
            <li><a href="#baterija">Baterija</a></li>
            <li><a href="#js">JS zanke</a></li>
            <li><a href="#pic">Slike</a></li>
            <li><a href="#ozn">Označevanja besedila</a></li>
            <li><a href="#objekti">Delo z objekti in shramba v pomnilnik in datoteko</a></li>
            <li><a href="#rekurzija">Rekurzija in iskanje v binarnem drevesu in povezanem seznamu</a></li>
            <li><a href="#klavir">Igranje klavirja</a></li>
        </ul>
    </div>
    <div data-role="footer"  data-theme="b">
        <h4>Luka Vidmar &copy; 2017</h4>
    </div>
</div>
```

Slika 5.1 Sestava aplikacije v ozadju.

Prva stran aplikacije ima 11 gumbov (slika 5.2), ki nas usmerijo v del aplikacije, za katerega je gumb namenjen. S pomočjo jQuery mobile se nam ni bilo potrebno veliko ukvarjati z izgledom aplikacije, saj je le-ta narejen tako, da ima že izgrajene komponente in njihov izgled.



Slika 5.2 Izgled prve strani.

5.1.1.2 Kamera

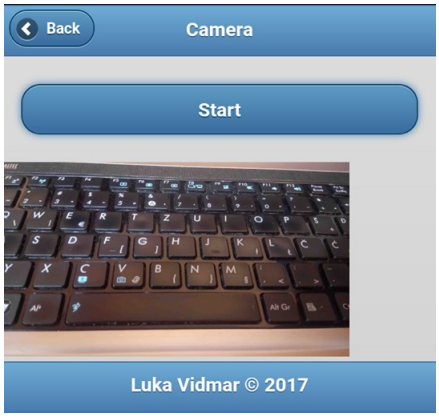
Kamera je eden izmed delov aplikacije, ki spremeni navadno spletno aplikacijo v mobilno, saj uporabljamo PhoneGap dodatke, ki poskrbijo, da se ob kliku na gumb odpre kamera. Zanima nas, kako se bo kamera obnesla tudi na drugih napravah in drugih platformah. Koda za kamero je razdeljena na 4 dele.

```javascript
$("#cam").click(function(){

    event.preventDefault();
    if (!navigator.camera) {
        alert("Camera API not supported", "Error");
        return;
    }

    navigator.camera.getPicture(onSuccess, onFail, { quality: 25,
        destinationType: Camera.DestinationType.FILE_URI,
        sourceType : Camera.PictureSourceType.CAMERA,
        allowEdit : false,
        encodingType: Camera.EncodingType.JPEG,
        popoverOptions: CameraPopoverOptions,
        saveToPhotoAlbum: true
    });

    function onSuccess(imageData) {
        var image = document.getElementById("picture");
        image.src = imageData;
    }

    function onFail(message) {
        alert('Failed because: ' + message);
    }
```

```html
<div id="camera" data-role="page" data-theme="b" data-add-back-btn="true">
    <div data-role="header" data-theme="b">
        <h1>Camera</h1>
    </div>
    <div data-role="content">
        <input type="button" value="Start" id="cam" />
    </div>
    <img src="" width="80%" id="picture">
    <div data-role="footer" data-theme="b">
        <h4>Luka Vidmar &copy; 2017</h4>
    </div>
</div>
```

Slika 5.3 Levo JavaScript koda, desno HTML koda za kamero.

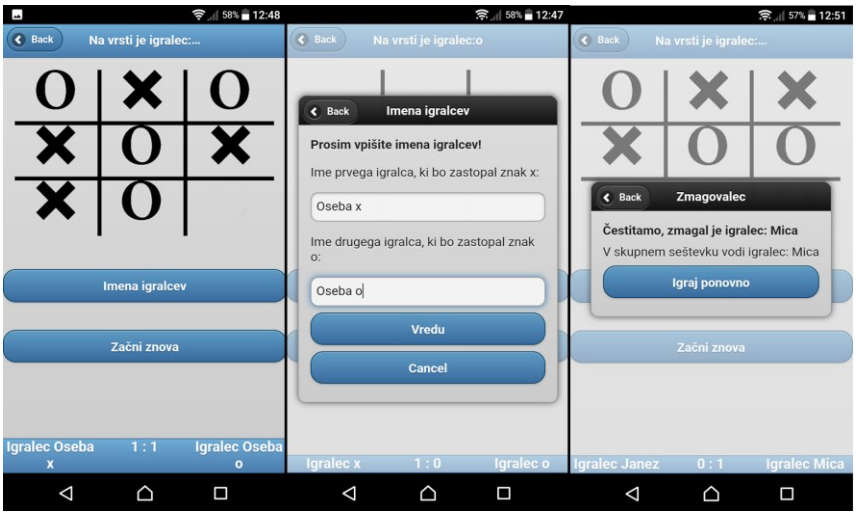
Prvi del kode poskrbi za opozorilo, če naprava ne podpira kamere. V drugem delu je potrebno poskrbeti za nastavitve kamere, torej kvaliteto slike, velikost slike, kam naj sliko shrani, oblika slike in izbiro, če lahko sliko tudi urejamo. Tretji del se izvede, če izpolnjujemo vse pogoje. Vklopi se kamera, posname sliko, nato lahko sliko urejamo in shranimo. Slika se nam prikaže pod gumbom za vklop kamere in pa v tudi v albumu slik. Zadnji del pa poskrbi za izpis napake, če do te pride. Del kode je prikazan na sliki 5.3, izgled pa je prikazan na sliki 5.4.



Slika 5.4 Izgled strani za kamero.

5.1.1.3 Tri v vrsto

Pri tej igri nas zanima, kako se bodo prikazale dimenzije kvadratov na različnih platformah in pa seveda tudi znaki, ki se pojavljajo v kvadratih. Poudarek je tudi na obvestilih, ki se pojavljajo ob koncu igre. Tri v vrsto je igra za dva igralca, v kateri imamo mrežo 3 x 3 polja in v njih prvi igralec vstavlja znak x, drugi pa znak o (slika 5.5). Cilj te igre je, da eden od igralcev naredi zaporedje treh enakih znakov, torej levo, desno, gor, dol oziroma diagonalno. Igra je narejena tako, da si lahko izberemo imena igralcev, lahko jo ponastavimo, v nogi strani pa se nam prikazuje trenutni rezultat.



Slika 5.5 Izgled aplikacije Tri v vrsto

Prvi del aplikacije, torej mreža 3 x 3, je izgrajena iz ukaza `GRID`, ki ga lahko vidimo na sliki 5.6. Najprej razdelimo stran na 3 dele in nato vsak del razdelimo še na tri. Vsakemu delu je potrebno s CSS ukazi še ustrezno dodati obrobo.

```
<div class="ui-grid-b"  >
    <div class="ui-block-a ui-grid-border-bottom ui-grid-border-right" id="11" ><img id="slika11"  style="height:50px"/></div>
    <div class="ui-block-b ui-grid-border-bottom ui-grid-border-right" id="12" ><img  id="slika12" style="height:50px"/></div>
    <div class="ui-block-c ui-grid-border-bottom" id="13" /></div>
</div>

<div class="ui-grid-b"  >
    <div class="ui-block-a ui-grid-border-bottom ui-grid-border-right" id="21" ><img id="slika21" style="height:50px"/></div>
    <div class="ui-block-b ui-grid-border-bottom ui-grid-border-right" id="22" ><img id="slika22" style="height:50px"/></div>
    <div class="ui-block-c ui-grid-border-bottom" id="23" ><img id="slika23" style="height:50px"/></div>
</div>

<div class="ui-grid-b"  >
    <div class="ui-block-a ui-grid-border-right" id="31" ><img id="slika31" style="height:50px"/></div>
    <div class="ui-block-b ui-grid-border-right" id="32" ><img id="slika32" style="height:50px"/></div>
    <div class="ui-block-c " id="33" ><img id="slika33" style="height:50px"/></div>
</div>
```

Slika 5.6 Mreža 3 x 3.

Za urejanje imena igralcev in obvestilo o zmagi in izenačenju smo uporabili kar ukaz `DIALOG`, ki je prikazan na sliki 5.7.

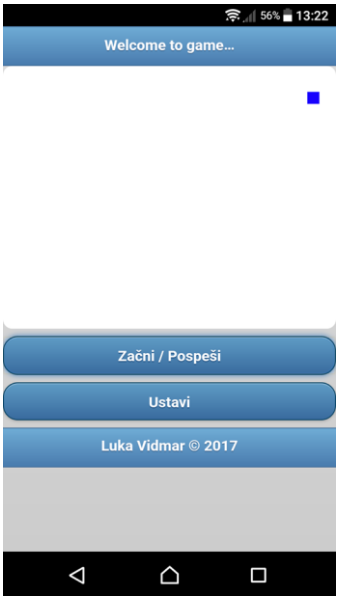
```
<a href="#dialogIgralci" data-rel="popup" data-position-to="window" data-transition="pop"  type="button"  id="igralci">Imena igralcev</a>
<div data-role="popup" id="dialogIgralci" data-overlay-theme="b" data-theme="b" data-dismissible="false" style="max-width:400px;">
    <div data-role="header" data-theme="a">
        <h1>Imena igralcev</h1>
    </div>
    <div role="main" class="ui-content">
        <h3 class="ui-title">Prosim vpišite imena igralcev!</h3>

        <label for="text-basic"><p>Ime prvega igralca, ki bo zastopal znak x:</p></label>
        <input type="text" name="text-basic" id="imex" value="">

        <label for="text-basic"><p>Ime drugega igralca, ki bo zastopal znak o:</p></label>
        <input type="text" name="text-basic" id="imeo" value="">

        <a href="#"  type="button" data-rel="back" id="igralciPodani">Vredu</a>
        <a href="#"  type="button" data-rel="back" data-transition="flow">Cancel</a>
    </div>
</div>
```

Slika 5.7 *Dialog* za izbiro imen igralcev.

Igra je sprogramirana tako, da mrežo 3 x 3 označimo s števili od 1 do 9. Nato naredimo tabelo, v kateri imamo števila od 1 do 9, zraven števil pa imamo še dodatno številko 0, 1 ali 2. 0 pomeni da je mesto še nezasedeno, 1 pomeni da ga je zasedel prvi igralec, 2 pa drugi igralec. Ko igralec izbere poljuben kvadrat, se zažene funkcija, ki pregleda, če je kvadrat prazen, če je še prost se tam pojavi njegov znak in takoj za tem se izvede funkcija, ki pregleda če je mogoče prišlo do treh zaporednih znakov, da dobimo zmagovalca. Če pride do tega, se igra zaključi in zmagovalcu se doda nova zmaga, sicer pa se igra nadaljuje, vse dokler niso vsi kvadrati zasedeni.

5.1.1.4 Kača

Aplikacija kača je namenjena, da vidimo, kako hitro se lahko kvadrat premika po zaslonu. Sestavljena je iz kvadrata, ki ima širino celega zaslona, dolžino pa le polovico zaslona (slika 5.8). V njem imamo še dodaten kvadrat, ki se giba znotraj prejšnjega kvadrata. Imamo še 2 gumba, in sicer prvi za začetek premikanja kvadrata in pospeševanja premika, drugi gumb pa je za ustavitev kvadrata.



Slika 5.8 Izgled aplikacije kača.

V ozadju aplikacije smo najprej dobili, kje se prvi kvadrat nahaja. Ko imamo koordinate kvadrata, smo postavili vanj še manjši kvadrat. Za premikanje kvadrata smo uporabili funkcijo `setInterval()`, ki smo ji nastavili čas intervala neskončno, s tem smo pridobili, da se funkcija nikoli ne ustavi in tako se kvadrat vedno premika. V tej funkciji pa imamo še štiri pogojne stavke, ki pazijo, da manjši kvadrat ne zaide izven velikega. Ko pride manjši kvadrat do kota velikega kvadrata, se sproži pogojni stavek in mu spremeni pot v drugo stran. Prikaz na sliki 5.9.

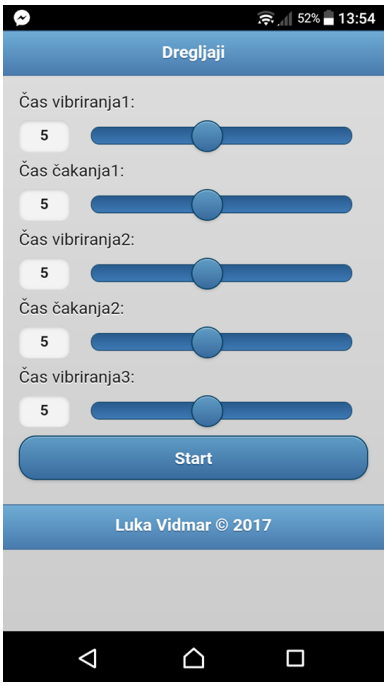
```
if(left2 <= left + 30 && top2 <= top + 30){
    a = 0;
    b = 1;
}
if(left2 <= left + 30 && top2 >= bottom - 30){
    a = 1;
    b = 0;
}
if(left2 >= right - 30 && top2 >= bottom - 30){
    a = 0;
    b = -1;
}
if(top2 <= top + 30 && left2 >= right - 30){
    a = -1;
    b = 0;
}

$('.apple').css({left: (left2 + a), top: (top2 + b)});
$('.apple').css({left: (left2 + a), top: (top2 + b)});
```

Slika 5.9 *If-stavek* za premik kvadrata.

Za ustavitev premika kvadrata imamo gumb `ustavi`, s katerim sprožimo funkcijo `clearInterval()` in s tem se neskončni interval ustavi.

5.1.1.5 Vibriranje

Pri vibriranju bomo kot pri kameri ponovno testirali, če lahko z enim ukazom odpremo vibriranje na različnih platformah. Vibriranje je sestavljeno iz 5-ih drsnikov, s katerimi nastavljamo vibriranje. Prikaz izgleda na sliki 5.10. S prvim, tretjim in petim drsnikom nastavljamo čas vibriranja, z drugim in četrtim pa čas zastoja med vibriranjem. Na dnu strani imamo še gumb za vklop.



Slika 5.10 Izgled aplikacije za vibriranje.

Za vklop vibracije smo uporabili PhoneGap dodatek za vibracijo pametne mobilne naprave. Sestavljen je iz dveh delov. V prvem je potrebno nastaviti hitrosti vibracije in čas med njimi, v drugem pa je samo klic funkcije za vklop vibracije s parametri časa, vibriranja in čakanja med vibriranjem, kar je prikazano na sliki 5.11.

```
$("#dreg").click(function(){

    var one = document.getElementById("sliderDreg1").value;
    var two = document.getElementById("sliderWait1").value;
    var three = document.getElementById("sliderDreg2").value;
    var four = document.getElementById("sliderWait2").value;
    var five = document.getElementById("sliderDreg3").value;

    navigator.accelerometer.getCurrentAcceleration(function(acceleration){alert('Acceleration X: ' + acceleration.x + '\n' +
        'Acceleration Y: ' + acceleration.y + '\n' +
        'Acceleration Z: ' + acceleration.z + '\n' +
        'Timestamp: '      + acceleration.timestamp + '\n');}, function (){console.error});

    console.log(navigator.vibrate([one*1000, two*1000, three*1000, four*1000, five*1000]));

});
```

Slika 5.11 Dodatek za vklop vibracije.

5.1.1.6 Baterija

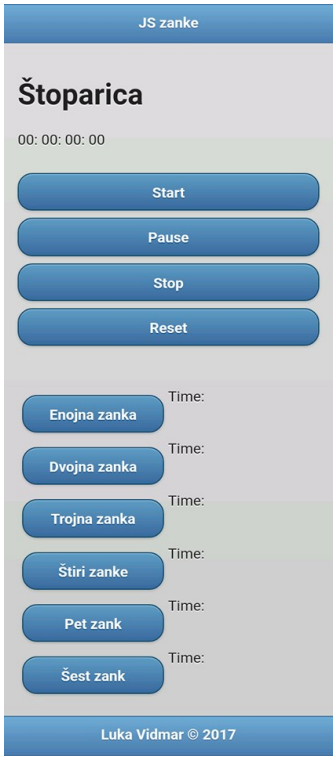
Za baterijo prav tako uporabljamo PhoneGap dodatek. Ko kliknemo na gumb `baterija`, sprožimo opozorilo, ki nam sporoči, koliko je baterija napolnjena. Dodatek je sestavljen iz treh delov. Prvi del vsebuje funkcijo `onchargingchange()`, ki se sproži, ko se aparat priključi oziroma izključi iz napajanja. Drugi del vsebuje funkcijo `onlevelchange()`, ki se sproži, ko se status napolnjenosti baterije spremeni in nas o tem obvesti. Ostane nam še tretji del, ki se zažene ob kliku na gumb `baterija` in nam sporoči trenutno napolnjenost baterije. Ti trije deli so prikazani na sliki 5.12.

```
navigator.getBattery().then(function(battery) {
    battery.onchargingchange = function() {
        alert("Baterija se je začela ali končala polnit. \nNapolnjenost: "+(this.level*100) +"%");
    };
});

navigator.getBattery().then(function(battery) {
    battery.onlevelchange = function() {
        alert("Napolnjenost baterija se je spremenila. \nNapolnjenost: "+(this.level*100) +"%");
    };
});

$("#Battery").click(function(){

    navigator.getBattery().then(function(battery) {

        alert("Napolnjenost baterije: "+battery.level*100+"%.");
    });

});
```

Slika 5.12 Dodatek za baterijo.

5.1.1.7 JS zanke

JS zanke je del aplikacije, s katero bomo testirali hitrost delovanja zank. V prvem delu imamo štoparico, v drugem delu pa zanke. Imamo od enojne vse do šest gnezdenih zank (slika 5.13).



Slika 5.13 Izgled *Js* zank.

Čas delovanja zanke se meri s pomočjo funkcije `Date.now()`. Ko stisnemo na gumb, na primer `Trojna zanka`, se najprej v spremenljivko shrani trenutni čas. Takoj za tem izvedemo 3 ugnezdene zanke in za tem si spet v spremenljivko shranimo trenutni čas. Ta dva časa nato odštejemo in dobimo čas, ki ga je aplikacija potrebovala za izvedbo teh zank. Koda je prikazana na sliki 5.14.

```
$("#three").click(function(){
    var t1 = Date.now();
    for (var i = 0; i < 10; i++) {
        for (var j = 0; j < 10; j++) {
            for (var k = 0; k < 10; k++) {
                $("#izpis").append(i + " " +j +" "+ k);
            }
        }
    }
    var t2= Date.now();

    var mili = t2-t1;
    var sec = (t2 - t1)/1000;
    $("#threeblock").html("Čas: "+sec +" sekund ali "+mili+" milisekund");
    $("#izpis").append("<p> </p>");

});
```

Slika 5.14 Izvedba merjenja časa zank.

5.1.1.8 Slike

Del aplikacije, namenjen slikam, je ustvarjen za preverjanje prikaza slike na mobilni napravi. Izbrali smo tri različne slike, in sicer podolgovato, ozko in zelo majhno sliko. Na sliki 5.15 vidimo del programske kode, kako je prikazan del aplikacije slike.

```
<div id="pic" data-role="page" data-theme="b" data-add-back-btn="true">
    <div data-role="header" data-theme="b">
        <h1>Pictures</h1>
    </div>
    <div data-role="content">
        <h3>Ogled nekaj slik</h3>
        <h4>Podolgovata slika:</h4>
        <p><img src="img/siroka.jpg"/></p>
        <h4>Ozka slika:</h4>
        <p><img src="img/podolgovata.jpg"/></p>
        <h4>Majhna slika:</h4>
        <p><img src="img/icon.jpg"/></p>
    </div>
    <div data-role="footer"  data-theme="b">
        <h4>Luka Vidmar &copy; 2017</h4>
    </div>
</div>
```

Slika 5.15 Del aplikacije slike.

5.1.1.9 Označevanja besedila

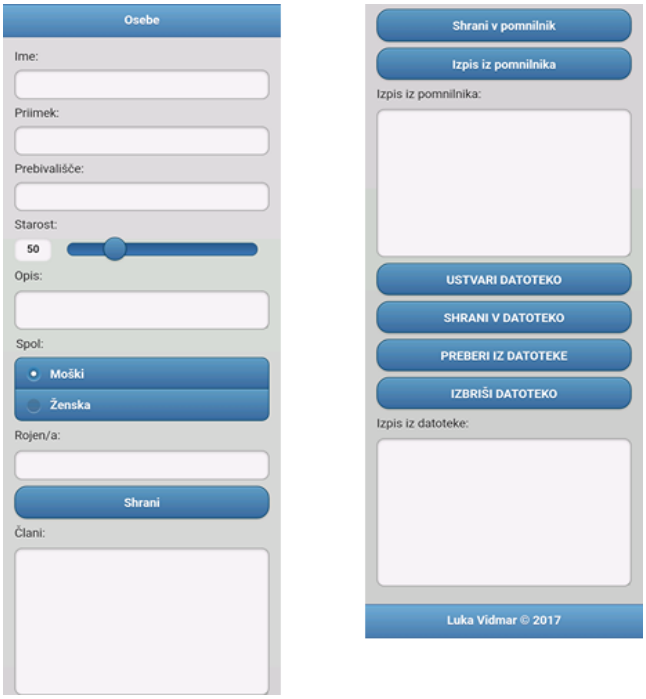
Del za označevanja besedila je namenjen poskusu, kako se bodo besedila označila na različnih platformah. V prvem delu imamo naštevanje programskih jezikov, ki se označujejo spredaj s piko, v drugem delu imamo številke, ki se številčno označujejo, v tretjem pa imamo še primer *teletype* besedila in pa primer odebeljenega besedila. Prikaz označevanja je na sliki 5.16.

```
<div data-role="page" id="ozn" data-theme="b" data-add-back-btn="true">
    <div data-role="header" data-theme="b">
        <h1>Označevanja</h1>
    </div>
    <div data-role="content">
        <p>Programski jeziki:</p>
        <ul>
            <li>Java</li>
            <li>C</li>
            <li>C++</li>
            <li>C#</li>
        </ul>
        <h2>Številčenje</h2>
        <ol>
            <li>Prvi</li>
            <li>Drugi</li>
            <li>Tetji , ki se ponavljajo besede: TetjiTetjiTetjiTetjiTetjiTetjiTetjiTetjiTetjiTetjiTetjiTetjiTetjiTetjiTetjiTetjiTetji</li>
        </ol>
        <p><tt>Primer teletype besedila.</tt></p>
        <p><strong>Opomba:</strong> Primer odebeljenega besedila.</p>
    </div>
    <div data-role="footer" data-theme="b">
        <h4>Luka Vidmar &copy; 2017</h4>
    </div>
</div>
```

Slika 5.16 Primer označevanja besedila.

5.1.1.10 Delo z objekti in shramba v pomnilnik in datoteko

V tem delu aplikacije smo testirali, kako se obnese ustvarjanje objektov, ki jih lahko začasno shranimo v lokalni pomnilnik oziroma za stalno shranimo v datoteko. Poudarek je bil na tem, če bo shranjevanje delovalo tudi na drugih napravah in različnih platformah. Zraven gumbov in območij za besedilo pa smo še dodali drsnik, izbiro med dvema spoloma in pa izbiro datuma. Izgled je prikazan na sliki 5.17.



Slika 5.17 Izgled dela aplikacije za shranjevanje in ustvarjanje objektov.

Za objekt smo si izbrali kar osebo, ki ima atribute; `ime`, `priimek`, `prebivališče`, `starost`, `opis`, `spol` in `rojstvo`. To lahko vidimo na sliki 5.18.

```
var oseba = new Object();
oseba.ime = ime;
oseba.priimek = priimek;
oseba.prebivalisce = prebivalisce;
oseba.starost = starost;
oseba.opis = opisOsebe;
oseba.spol = spol;
oseba.rojstvo = datum;
```

Slika 5.18 Ustvarjanje objekta oseba.

Ob pritisku na gumb `shrani`, se oseba shrani v spremenljivko podatkovnega tipa `string` in se nato izpiše v okencu člani. S pritiskom na gumb `shrani v pomnilnik`, pokličemo funkcijo `localStorage.setItem()` in tja shranimo spremenljivko, v kateri so shranjeni člani. Prikazano na sliki 5.19.

```
$("#savetopomnilnik").click(function () {
    document.getElementById("pomnilnikk").value = "";
    var inside = document.getElementById("clani").value;
    localStorage.setItem("key",inside);
    alert("Uspešno shranjeno v pomnilnik.");

});
```

Slika 5.19 Shranjevanje v lokalni pomnilnik.

Za branje iz lokalnega pomnilnika pa pokličemo funkcijo `localStorage.getItem()`, ki je prikazana na sliki 5.20.

```
$("#pomnilnik").click(function () {
    document.getElementById("pomnilnikk").value = localStorage.getItem("key");
    alert("Uspešno pridobljeno iz pomnilnika.");
});
```

Slika 5.20 Branje iz lokalnega pomnilnika.

Nazadnje pa imamo še ustvarjanje, shranjevanje, branje in brisanje datoteke. Funkcije za delo z datotekami imajo tri dele (slika 5.21). Prvi del so nastavitve funkcije, v našem primeru opis datoteke, drugi del je funkcija, ki se zgodi, če je vse potekalo brez napak, torej ustvarjanje

datoteke, branje, shranjevanje in brisanje datoteke, v tretjem delu pa je funkcija, ki se izvede ob napaki in nam napako tudi javi.

```
function writeFile() {
    var tekst = document.getElementById("clani").value;
    var type = window.TEMPORARY;
    var size = 5*1024*1024;
    window.requestFileSystem(type, size, successCallback, errorCallback)

    function successCallback(fs) {
        fs.root.getFile('log.txt', {create: true}, function(fileEntry) {

            fileEntry.createWriter(function(fileWriter) {
                fileWriter.onwriteend = function(e) {
                    alert('Shranjeno.');
                };

                fileWriter.onerror = function(e) {
                    alert('Napaka pri shranjevanju: ' + e.toString());
                };

                var blob = new Blob([tekst], {type: 'text/plain'});
                fileWriter.write(blob);
            }, errorCallback);
        }, errorCallback);
    }

    function errorCallback(error) {
        alert("ERROR: " + error.code)
    }
    document.getElementById("textarea").value = "";
}
```

Slika 5.21 Primer funkcije za shranjevanje v datoteko.

5.1.1.11 Rekurzija in iskanje v binarnem drevesu in povezanem seznamu

Ta del aplikacije je bil ustvarjen za merjenje hitrosti pri delu z velikimi količinami podatkov. Podatki se shranjujejo v binarno drevo in povezan seznam. Zanima nas, kolikšna bo razlika pri iskanju v binarnem drevesu in povezanem seznamu med hibridno in izvirno aplikacijo. Izgled aplikacije lahko vidimo na sliki 5.22.

Slika 5.22 Izgled dela aplikacije za rekurzijo, binarno drevo in povezan seznam.

V prvem delu najprej vpišemo številko, na kateri se bo podatek nahajal, torej ključ podatka. Ko pritisnemo gumb `shrani`, se izvedejo funkcije za shranjevanje podatkov v binarno drevo in shranjevanje podatkov v povezan seznam. Najprej ustvarimo dva objekta; prvega za povezan seznam in drugega za binarno drevo (slika 5.23).

```
var rootNormal = new Object();
rootNormal.key = null;
rootNormal.name = null;
rootNormal.next = null;
```

```
var newNode = new Object();
newNode.key = key;
newNode.name = name;
newNode.leftChild = null;
newNode.rightChild = null;
```

Slika 5.23 Levo objekt za povezan seznam, desno za binarno drevo.

Pri shranjevanju v povezan seznam potrebujemo objekt, ki lahko shrani ključ, podatek in pa naslednji objekt, torej naslednji podatek. Za binarno drevo pa potrebujemo ključ in podatek, ki ga shranjujemo, in pa še levi in desni otrok. Shranjevanje pri povezanem seznamu poteka tako, da imamo zanko, ki se sprehodi skozi seznam, in sicer tako da pregleda, če ima sam objekt že povezavo na naslednjega, in ko pride do konca, mu doda nov objekt. Pri shranjevanju v binarno drevo pa se sklicujemo na levega in desnega otroka. Ustvarimo funkcijo, ki najprej pogleda ključ pri objektu, ki ga shranjujemo, in če je ključ večji od korena drevesa, se spustimo k desnemu otroku, če je manjši, pa k levemu otroku. To ponavljamo dokler ne pridemo do

objekta, ki nima več otroka. Samo shranjevanje se najlepše izvede z rekurzijo, ki je prikazana na sliki 5.24.

```
if(parseInt(key) < parseInt(focusNode.key)){

    if(focusNode.leftChild == null ){
        focusNode.leftChild = newNode;
        bool = false;
    }else{
        focusNode = focusNode.leftChild;
    }
}
if(parseInt(key)> parseInt(focusNode.key)){
    if(focusNode.rightChild == null ) {
        focusNode.rightChild = newNode;
        bool = false;
    }else{
        focusNode = focusNode.rightChild;
    }

}
```

Slika 5.24 Del kode, kjer se odločimo za levega oziroma desnega otroka.

Za izpis drevesa smo se odločili kar za premi obhod, ki ga lahko vidimo na sliki 5.25. To je obhod binarnega drevesa, kjer za korenom najprej izpišemo leve otroke, šele nato pa gremo na desne.

```
function preorder() {
    var result = [];
    var node = rootBinary;
    var preorderr = function(node) {
        result.push(node.key);
        node.leftChild && preorderr(node.leftChild);
        node.rightChild && preorderr(node.rightChild);
    };
    preorderr(node);
    return result;
}
```

Slika 5.25 Premi obhod binarnega drevesa.

Pri iskanju ključa v binarnem drevesu in povezanem seznamu se izkaže, da se binarno drevo izkaže za občutno hitrejše kot povezan seznam. Pri iskanju se izkaže, da ima binarno drevo čas $O(\log n)$ oziroma v najslabšem primeru $O(n)$, povezan seznam pa ima vedno $O(n)$. Iskanje pri povezanem seznamu se izvede tako, da se premikamo od objekta do objekta, dokler ne pridemo do iskanega. V binarnem drevesu pa vedno najprej pogledamo ključ iskanega objekta in ključ korena v drevesu in se potem pomikamo levo, če je iskani ključ manjši, oziroma desno, če je iskani ključ večji od trenutnega objekta. Funkcija je skoraj ista kot pri dodajanju objekta. Čas iskanja elementa smo izmerili s funkcijo `Date.now()`, torej smo si shranili čas pred klicem

funkcije in po klicu funkcije, ta dva časa odšteli in dobili čas iskanja objekta. Funkcije lahko vidimo na sliki 5.26.

```
function findInBinaryTree() {
    var key = document.getElementById("findTree").value ;

    var focusNode = rootBinary;

    while(focusNode.key != key){

        if(key < focusNode.key){
            focusNode = focusNode.leftChild;
        }else{
            focusNode = focusNode.rightChild;
        }
    }
    document.getElementById("findetBinaryTree").value = focusNode.name;
}

function findInConnectedList() {
    var key = document.getElementById("findTree").value ;
    var run = true;
    var data = "Ta številka ne obstaja";
    var focus = rootNormal;

    while(run){
        if(focus.key == key){
            data = focus.name;
            run = false;
        }else{
            if(focus.next == null){
                run = false;
            }else{
                focus = focus.next;
            }
        }

    }
    document.getElementById("findetConnectedList").value = data;
}
```

Slika 5.26 Funkcije za iskanje objektov.

Pri gumbu za izbris binarnega drevesa in povezanega seznama nam ni potrebno storiti veliko. Koren povezanega seznama in binarnega drevesa samo nastavimo na ničlo. Ker Java vsebuje *Garbage Collector*, nam sama poskrbi za izbris nedejavnih objektov.

Gumb `generiraj drevo` nam ustvari nov povezan seznam in novo binarno drevo. V sami kodi aplikacije je v tabeli shranjenih več tisoč številk, skozi katere se sprehodimo in preprosto pokličemo funkcijo za shranjevanje v binarno drevo in povezan seznam. To funkcijo smo ustvarili, ker sam človek težko doda toliko številk v program in bi bilo težko testirati razliko v času iskanja med hibridno in izvorno aplikacijo, pa tudi številke morajo biti iste in v istem zaporedju tudi shranjene.

Nazadnje pa imamo še kontrolo, ki je namenjena uporabniku, da vidi, kaj se trenutno izvaja oziroma če se je ukaz že izvedel.

5.1.1.12 Igranje klavirja

Igranje klavirja je igra, ki je namenjena preizkusu zvoka. Zanima nas, kako hitro bomo lahko pritiskali na tipko in se bo ta zvok še proizvedel, med drugim pa tudi, če bo sam izgled klavirja ostal enak, ko bomo aplikacijo namestili na drugo napravo. Izgled aplikacije je prikazan na sliki 5.27.

Slika 5.27 Del aplikacije za igranje klavirja.

Del aplikacije za igranje klavirja je zgrajen iz ukaza `grid`. Sliko razdelimo na vrhu na 20 delov, spodaj pa na 10 delov. Na vrhu jih še malo pomaknemo v desno in ustrezne obarvamo v črno, da dobimo poltone. Spodaj jih obarvamo samo 7 in dodamo imena tonov. Ob vsakem pritisku na ton oziroma tipko se izvede HTML5 ukaz in začne se predvajati zvok oziroma ton določene tipke (slika 5.28).

```
function zvok(name) {
    var fileName = 'sound/'+name+'.wav';
    $('#divVideo audio source').attr('src', fileName);
    $("#divVideo audio")[0].load();
    $("#divVideo audio").trigger('play')
}
```

Slika 5.28 HTML5 ukaz za predvajanje zvoka.

5.2 Izvorni razvoj aplikacije za Android v Javi

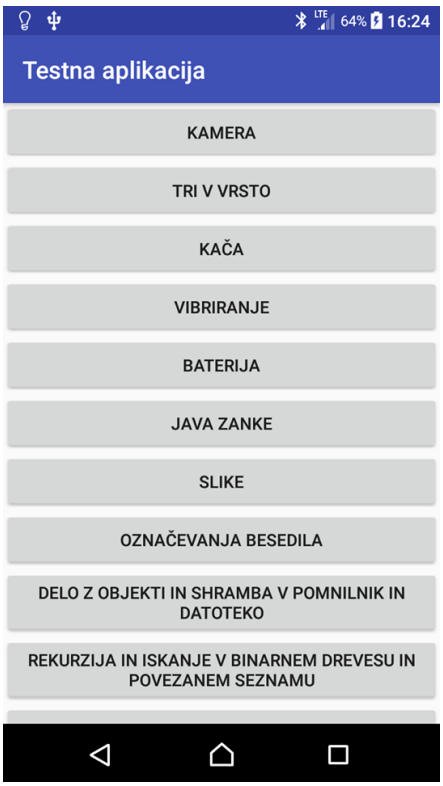
V prejšnjem podpoglavju smo predstavili hibridni razvoj aplikacije, sedaj pa bomo še predstavili izvorni razvoj aplikacije v programskem jeziku Java za operacijski sistem Android.

5.2.1 Sestava aplikacije

Aplikacija narejena v Javi ima 3 glavne mape. Prva je manifest, v kateri je datoteka `AndroidManifest.xml`. V njej so shranjene vse glavne stvari aplikacije, kot je na primer tema aplikacije ali pa katere datoteke `activity` vsebuje aplikacija itd. Nato imamo mapo Java in v njej *com.example.hsport.imeAplikacije.java*, v kateri so shranjene vse `.java` datoteke. Na koncu imamo še mapo `layout`, v kateri so shranjene datoteke, ki določajo izgled raznim delom aplikacije. Torej imamo za vsako stran v aplikaciji datoteko `.java`, v kateri imamo sprogramirano, kaj bo ta stran delala in datoteko `.xml`, v kateri imamo sprogramiran izgled strani.

5.2.1.1 Prva stran aplikacije

Prva stran aplikacije je zgrajena iz samih gumbov, ki nam pokličejo naslednjo aktivnost. Izgled gumbov lahko vidimo na sliki 5.29.



Slika 5.29 Prva stran aplikacije

Prva stran je zgrajena iz drsnika, nato smo ji določili linearno postavitev in dodajali še gumbe. Vse to lahko dodajamo tako, da pišemo kodo v datoteko (slika 5.30), lahko pa izberemo drugi, grafični način, kjer samo izberemo elemente in jih prenesemo na to stran.

```
<ScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <Button
            android:id="@+id/cameraButton"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Kamera"
            android:onClick="camActivity" />
```

Slika 5.30 Izgled kode za grafični vmesnik za prvo stran.

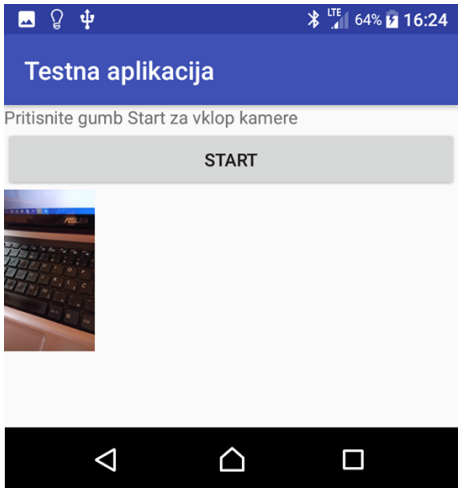
Na koncu je potrebno samo še v Javi narediti funkcijo, ki se sproži ob kliku na gumb (slika 5.31).

```
public void camActivity(View v){
    Intent intent = new Intent( packageContext: this, Kamera.class);
    startActivity(intent);
}
```

Slika 5.31 Funkcija za priklic nove aktivnosti.

5.2.1.2 Kamera

Aktivnost kamera je prav tako zgrajena iz drsnika, linearne postavitve (slika 5.33), v njej pa imamo okvir za besedilo `textView`, gumb in pa okvir za sliko `imageView`. Izgled je prikazan na sliki 5.32.



Slika 5.32 Izgled aktivnosti kamere.

V ozadju pa smo morali narediti novo funkcijo `onClick()` za vklop kamere, ki jo prikličemo ob pritisku na gumb `START` in pa funkcijo, ki se izvede, če se je izvedla funkcija `onClick()`, ta pa nam sliko prikaže na zaslonu. Na sliki 5.33 lahko vidimo funkcije za vklop kamere in gradnike grafičnega vmesnika.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    if(requestCode == CAM_REQUEST){
        Bitmap bitmap = (Bitmap) data.getExtras().get("data");
        imgTakenPic.setImageBitmap(bitmap);
    }
}

class btnTakePhotoClicker implements  Button.OnClickListener{

    @Override
    public void onClick(View view) {
        Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
        startActivityForResult(intent,CAM_REQUEST);
    }
}
```

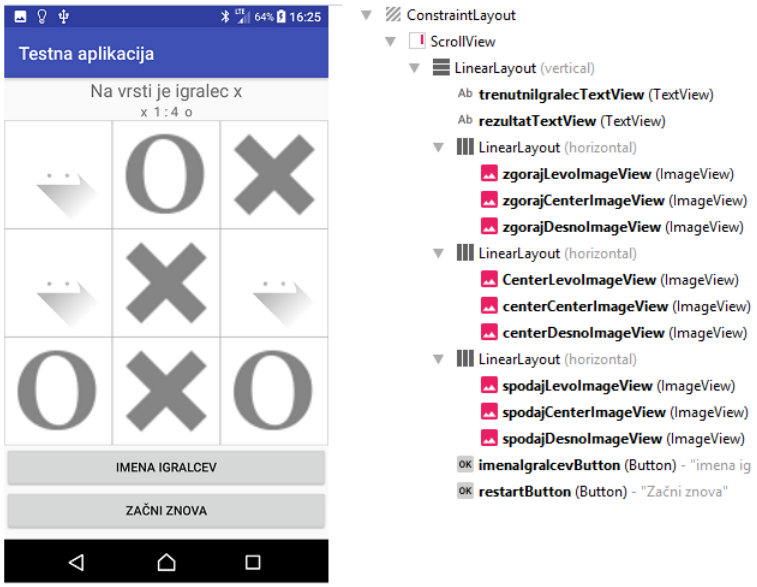
```
<ScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical" >

        <TextView
            android:id="@+id/textView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Pritisnite gumb Start za vklop kamere" />

        <Button
            android:id="@+id/startCameraButton"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Start" />


        <ImageView
            android:id="@+id/imageView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_below="@+id/button"
            android:scaleType="centerCrop" />
```

Slika 5.33 Levo: Funkcije za vklop kamere. Desno: Gradniki grafičnega vmesnika.

5.2.1.3 Tri v vrsto

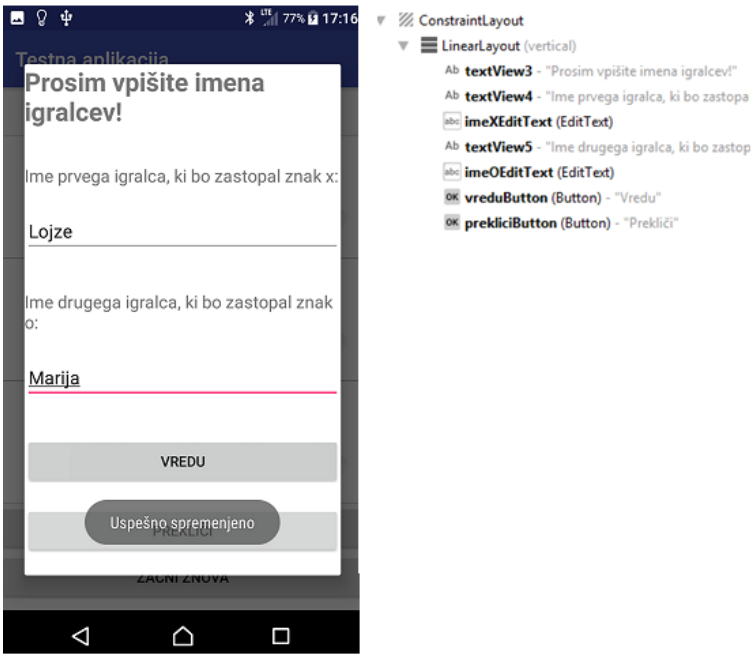
Za igro tri v vrsto smo uporabili drsnika, vertikalen linearni položaj, v njem pa imamo dva okvirja za besedilo `textView`, kar si lahko ogledamo tudi na sliki 5.34. V prvem izpisujemo, kateri igralec je trenutno na vrsti, v drugem pa rezultat igre. Za tem imamo 3 vertikalne linearne

položaje, s katerimi naredimo kvadrate 3 x 3, v njih pa imamo okvir za sliko. Na koncu nam ostaneta še dva gumba, in sicer prvi za določanje imena igralcev, drugi pa za ponastavitev igre.



Slika 5.34 Izgled postavitve igre tri v vrsto.

Koda, ki se izvaja v ozadju, je ista kot pri hibridni aplikaciji. Ob kliku na enega izmed kvadratov, se v tem kvadratu prikaže znak igralca in za tem se pokliče funkcija, ki pregleda, če je prišlo do zmage. Ob pritisku na zadnja dva gumba smo morali ustvariti `popup Dialog()`. Dialog je zgrajen in nove `.xml` datoteke, ki mu določa obliko. Na primer dialog za spreminjanje imena igralcev ima 5 okvirčkov za besedilo, ki so prikazani na sliki 5.35, od tega je v treh besedilo že napisano, dva pa sta prazna za imena naših igralcev. Na koncu imamo še gumb `vredu` in `prekliči`.



Slika 5.35 Dialog za določanje imena igralcev.

Na sliki 5.36 lahko vidimo, da ob pritisku na gumb `imena igralcev` uporabimo funkcijo `AlertDialog.Builder()`, ki nam prikliče aktivnost za spreminjanje imen igralcev. V tej funkciji pogledamo imena igralcev in jih takoj spremenimo. Za konec uporabimo še funkcijo `Toast.makeText()`, ki nam prikaže obvestilo ob uspešni menjavi imen.

```
b.setOnClickListener ((view) → {
        AlertDialog.Builder mBuilder = new AlertDialog.Builder( context: triVvrsto.this);
        View mView = getLayoutInflater().inflate(R.layout.activity_dialog_spremeni_ime,  root: null);
        final EditText imexEditText = (EditText) mView.findViewById(R.id.imeXEditText);
        final EditText imeoEditText = (EditText) mView.findViewById(R.id.imeOEditText);
        final Button vreduSpremeniImeButton = (Button) mView.findViewById(R.id.vreduButton);
        Button prekliciSpremeniImeButton = (Button) mView.findViewById(R.id.prekliciButton);

        vreduSpremeniImeButton.setOnClickListener((view) → {
                igralecX = imexEditText.getText().toString();
                igralecO = imeoEditText.getText().toString();
                Toast.makeText( context: triVvrsto.this,  text: "Uspešno spremenjeno", Toast.LENGTH_SHORT ).show();
                TextView trenutniIgralecTextView = (TextView)findViewById(R.id.trenutniIgralecTextView);
                TextView rezultatTextView = (TextView)findViewById(R.id.rezultatTextView);
                trenutniIgralecTextView.setText("Na vrsti je igralec " +igralecX);
                rezultatTextView.setText(igralecX + "  "+igralecXtock+" : "+igralecOtock + "  "+igralecO);
        });
```

Slika 5.36 Koda za menjavo imen igralcev.

5.2.1.4 Kača

Kača je aktivnost, v kateri se kvadratek premika po zaslonu v obliki kvadrata. Sestavljena je iz relativne postavitve (slika 5.37), na kateri imamo sliko, ki se bo premikala po zaslonu, spodaj v levem kotu gumb za začetek premikanja, v desnem kotu pa gumb za prekinitev premikanja.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <ImageView
        android:id="@+id/image"
        android:layout_width="50dp"
        android:layout_height="50dp"
        android:src="@drawable/icon" />

    <Button
        android:id="@+id/stopButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:text="Stop"
        android:onClick="ustavi"/>

    <Button
        android:id="@+id/startButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:text="Start"
        android:onClick="pomik"/>

</RelativeLayout>
```

Slika 5.37 Sestava aktivnosti kača.

Da se kvadratek začne premikati, smo morali najprej izvedeti velikost našega zaslona, in sicer s funkcijami :

```
- Display display = getWindowManager().getDefaultDisplay();
- Point size = new Point();
- display.getSize(size);
- width = size.x;
- height = size.y;
```

Ko smo dobili velikost zaslona, smo si določili še do kje se lahko kvadratek premika (torej največji in najmanjši dovoljen x in največji in najmanjši dovoljen y), da se kvadratek ne izgubi iz zaslona. Nato smo naredili še funkcijo za pomik, ki pokliče drugo funkcijo za pomik, ki vsebuje rekurzijo. To si lahko ogledamo na sliki 5.38. Narejena je tako, da najprej preveri, če smo stisnili na tipko »stop« in če nismo, najprej preveri, v katero smer se mora premaknit, nato se tja pomakne, počaka pol sekunde in nato spet pokliče sama sebe. V primeru da pridemo do meje našega zaslona, se smer premikanja kvadratka spremeni.

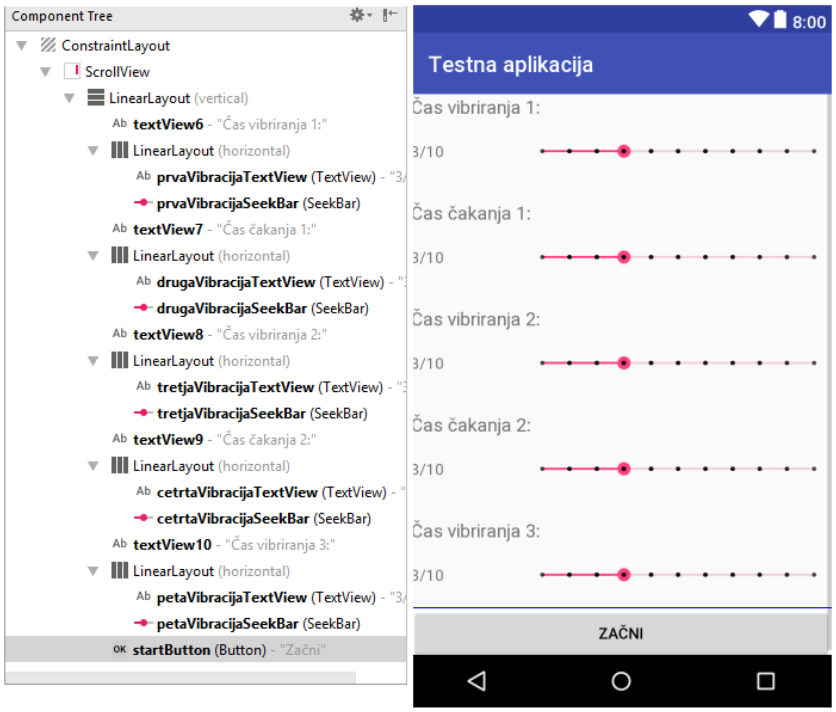
```
public void pomik2(){
    Handler handler1 = new Handler();
    if(ustavi){
    }else {
        handler1.postDelayed(new Runnable() {

            @Override
            public void run() {

                if (smer == 1) {
                    pojdiDol();
                    pomik2();
                }
                if (smer == 2) {
                    pojdiDesno();
                    pomik2();
                }
                if (smer == 3) {
                    pojdiGor();
                    pomik2();
                }
                if (smer == 4) {
                    pojdiLevo();
                    pomik2();
                }

            }
        }, delayMillis: 50 );
    }

}
```

```
public void pojdiDol(){
    y = image.getY();
    if(y < maxBottom){
        image.setY(y+5);
    }else{
        smer = 2;
    }
}

public void pojdiDesno(){
    x = image.getX();
    if(x < maxRight){
        image.setX(x+5);
    }else{
        smer = 3;
    }
}

public void pojdiGor(){
    y = image.getY();
    if(y > maxTop){
        image.setY(y-5);
    }else{
        smer = 4;
    }
}

public void pojdiLevo(){
    x = image.getX();
    if(x > maxLeft){
        image.setX(x-5);
    }else{
        smer = 1;
    }
}
```

Slika 5.38 Funkcija za pomik kvadratka.

5.2.1.5 Vibriranje

Za vibriranje smo naredili novo aktivnost, ki vsebuje drsnik po strani, linearno postavitev, 5 okvirjev za besedilo `textView`, 5 drsnikov `seekBar`, ki se pomikajo levo in desno in še dodatnih 5 okvirjev zraven drsnikov za številko `textView`, ki jo bomo izbrali s pomočjo drsnika. Nazadnje imamo še gumb in ob pritisku nanj sprožimo vibriranje aparata. Gradniki so prikazani na sliki 5.39. Aparat začne vibrirati v takem vrstnem redu, kot smo ga izbrali s pomočjo drsnikov.



Slika 5.39 Izgled aktivnosti za vibriranje.

V Java datoteki smo naredili funkcijo, ki se sproži s pritiskom na gumb `začni`. V njej smo najprej prebrali zaporedje vibracije, ki smo jih nastavili z drsniki, nato smo poklicali funkcijo `Vibrator.vibrate(milisekunde)`, ki je prikazana na sliki 5.40. Med klicanjem funkcije za vibracijo smo uporabili še funkcijo `Handler.postDelayes()`, ki nam omogoča čakanje med eno in drugo vibracijo.

```
public void vibrate(View v){
    Vibrator vib = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);
    // Vibrate for 500 milliseconds
    vib.vibrate( milliseconds: vibriranje1*1000);
    Handler handler = new Handler();
    handler.postDelayed(new Runnable() {
        public void run() {
            // Actions to do after 5 seconds
            Vibrator vib = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);
            vib.vibrate( milliseconds: vibriranje2*1000);
        }
    }, delayMillis: cakanje1*1000+vibriranje1*1000);

    handler = new Handler();
    handler.postDelayed(new Runnable() {
        public void run() {
            // Actions to do after 5 seconds
            Vibrator vib = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);
            vib.vibrate( milliseconds: vibriranje3*1000);
        }
    }, delayMillis: cakanje1*1000+vibriranje1*1000+cakanje2*1000+vibriranje2*1000);

}
```

Slika 5.40 Funkcija za izvajanje zaporedja vibracije.

5.2.1.6 Baterija

Za status baterije ni bilo potrebno ustvarjati nove aktivnosti. V glavni aktivnosti, torej na prvi strani, smo povezali gumb `baterija` na novo funkcijo, ki nam izpiše podrobnosti o bateriji.

Slika 5.41 Izgled izpisa podrobnosti o bateriji.

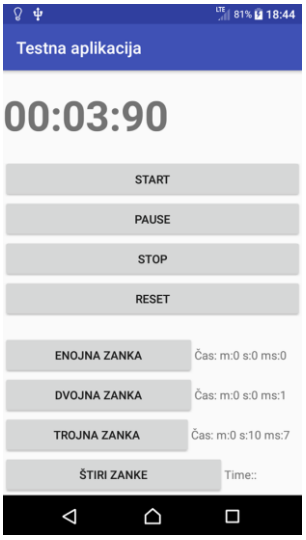
Za pridobitev podatkov o bateriji smo uporabili funkcijo `BroadcastReceiver batteryInfoReceiver()`, ki nam s pomočjo funkcije `BatteryManager()` pridobi željene podatke. Funkcija je prikazana na sliki 5.42. Za izpis podatkov smo uporabili kar funkcijo `Toast.makeText()`, ki nam izpiše podatke v majhnem kvadratu oziroma obvestilu, kot je prikazano na sliki 5.41.

```
private BroadcastReceiver batteryInfoReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {

        int  health= intent.getIntExtra(BatteryManager.EXTRA_HEALTH, defaultValue: 0);
        int  level= intent.getIntExtra(BatteryManager.EXTRA_LEVEL, defaultValue: 0);
        int  plugged= intent.getIntExtra(BatteryManager.EXTRA_PLUGGED, defaultValue: 0);
        boolean  present= intent.getExtras().getBoolean(BatteryManager.EXTRA_PRESENT);
        String  technology= intent.getExtras().getString(BatteryManager.EXTRA_TECHNOLOGY);
        int  temperature= intent.getIntExtra(BatteryManager.EXTRA_TEMPERATURE, defaultValue: 0);
        int  voltage= intent.getIntExtra(BatteryManager.EXTRA_VOLTAGE, defaultValue: 0);

        Toast.makeText( context: MainActivity.this,

                text: "Health: "+health+"\n"+
                        "Level: "+level+"\n"+
                        "Plugged: "+plugged+"\n"+
                        "Technology: "+technology+"\n"+
                        "Temperature: "+temperature+"\n"+
                        "Voltage: "+voltage+"\n", Toast.LENGTH_SHORT ).show();
    }
};
```

Slika 5.42 Funkcija za pridobitev podatkov o bateriji.

5.2.1.7 Java zanke

Osnova za prikaz aktivnosti Java zank je drsnik in linearna-vertikalna postavitev. Na vrhu strani imamo prikaz časa za štoparico, nato imamo gumbe `start`, `pause`, `stop` in `reset` za upravljanje štoparice, sledi šest gumbov, s katerimi lahko sprožimo od 1 pa vse do 6 gnezdenih zank, zraven pa imamo še okno za izpis časa, ki ga je potrebovala zanka. Na koncu imamo še eno okno, v katerem izpisujemo številke, skozi katere se je sprehodila zanka (slika 5.43).



Slika 5.43 Izgled aktivnosti Java zank.

Za merjenje časa smo uporabili funkcijo `SystemClock.uptimeMillis()`. Pri štoparici imamo 4 funkcije. Prva je funkcija začni, ki si v spremenljivko shrani trenutni čas s pomočjo zgoraj omenjene funkcije. Nato pokliče drugo funkcijo, ki skrbi za izpis trenutnega časa. Čas se računa tako, da trenutni čas odštejemo od časa, ki smo si ga shranili, ko smo zagnali štoparico. Tako dobimo milisekunde, ki jih samo še pretvorimo v ure, minute in sekunde. To ponavljamo, dokler ne pritisnemo na gumb `pavza` ali pa `stop`. Ko pritisnemo na pavzo, se štoparica ustavi in ta čas moramo odbiti, ko štoparica začne spet teči. Gumb `stop` pa naredi enako kot `pavza`, le da se štoparica nato ponastavi na ničlo. Zadnja funkcija za ponastavitev štoparice vse čase, ki smo jih imeli shranjene, pobriše in nato še štoparico nastavi na ničlo.

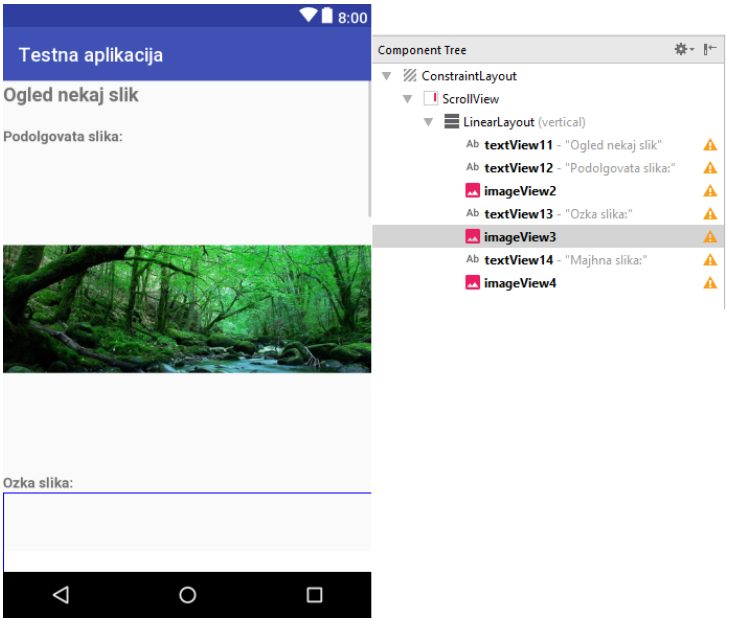
Na sliki 5.44 lahko vidimo funkcijo `System.currentTimeMillis()` za merjenje časa pri izvedbi zank, ki nam prav tako vrne trenutni čas v milisekundah. Ko se zanka izvede, odštejemo trenutni čas od časa pred začetkom zanke in tako dobimo čas, ki ga je potrebovala zanka za izvedbo.

```
Button firstButton = (Button)findViewById(R.id.zanka1Button);

firstButton.setOnClickListener((view) → {
        long firstTime = System.currentTimeMillis();
        TextView izpis = (TextView)findViewById(R.id.izpisTextView);
        izpis.setText("");

        for(int i = 0; i < 10; i++){
            izpis.setText(izpis.getText() + " "+i+" /");
        }

        long secTime = System.currentTimeMillis();

        long elapsed = secTime - firstTime;



        TextView izpisTime = (TextView) findViewById(R.id.time1TextView);
        izpisTime.setText(cas(elapsed));
});
```

Slika 5.44 Merjenje časa za izvedbo zanke.

5.2.1.8 Slike

Za prikaz slik smo izbrali drsnik, da se bomo lahko pomikali po strani od slike do slike, nato smo izbrali vertikalno postavitev slike, `textView` za napise nad slikami in `imageView`. za prikaz slike. Razporeditev je prikazana na sliki 5.45.

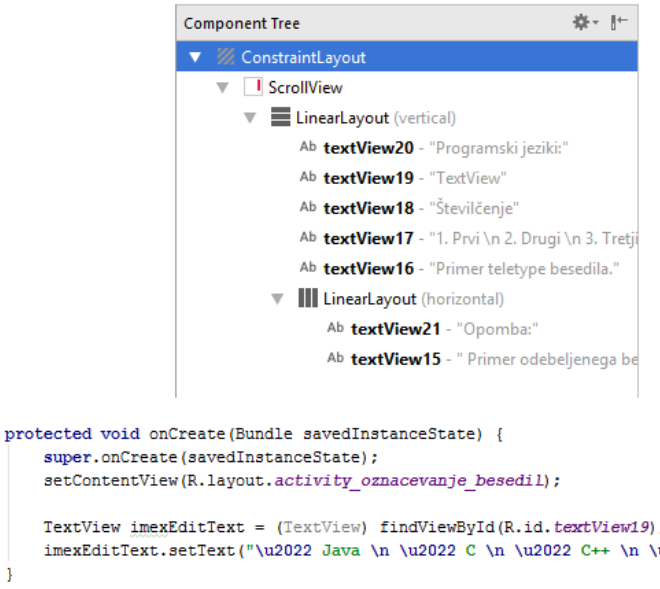


Slika 5.45 Izgled strani za prikaz slik.

5.2.1.9 Označevanje besedila

Za označevanje besedila smo uporabili gradnike `textView`, ki so linearno vertikalno poravnani z drsnikom pri strani. Za prikaz pike pri naštevanju programskih jezikov smo morali

uporabiti kombinacijo številk in črk `\u2022` iz *UNICODE* kodiranja, kar je prikazano na sliki 5.46.

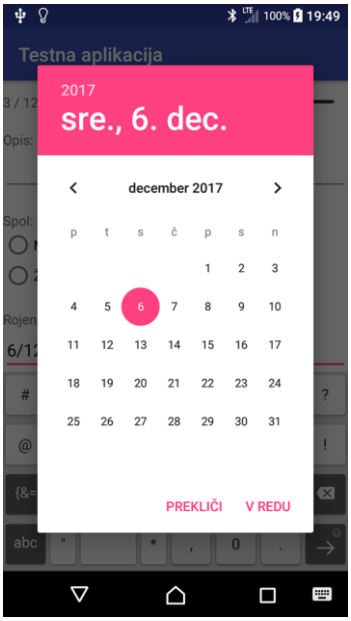



```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_oznacevanje_besedil);

    TextView imexEditText = (TextView) findViewById(R.id.textView19);
    imexEditText.setText("\u2022 Java \n \u2022 C \n \u2022 C++ \n \u2022 C# \n ");
}
```

Slika 5.46 Gradniki strani in funkcija za prikaz naštevanja programskih jezikov.

5.2.1.10 Delo z objekti in shramba v pomnilnik in datoteko

Aktivnost za delo z objekti vsebuje drsnik in na njem imamo vertikalno linearno razporeditev. Najprej imamo okvirje za vpis podatkov o osebi, torej ime, priimek, prebivališče, nato imamo drsnik za izbiro starosti, za izbiro spola pa imamo kar izbirni gumb, ki dopusti samo eno izbiro. Nazadnje imamo še okvir za izbiro datuma rojstva, ob kliku nanj pa se prikaže koledar, ki je na sliki 5.47.



Slika 5.47 Izbira datuma rojstva.

Na koncu imamo gumba za shraniti osebe v pomnilnik in za izpis iz njega. Imamo še gumbe za ustvariti datoteko, vanjo shraniti osebe, prebrati osebe iz datoteke in gumb za izbrisati datoteko.

Za ustvarjenje novega objekta, imenovanega oseba, smo naredili nov razred z imenom oseba. Ta razred ima konstruktor za dodajanje raznih atributov, kot so `ime`, `priimek`, `prebivališče` itd. in pa funkcijo za izpis osebe. Prikazan je na sliki 5.48.

```
public class oseba {

    String ime = "";
    String priimek = "";
    String prebivalisce = "";
    int starost = 0;
    String opis = "";
    String spol = "";
    String datum = "";


    public void oseba(String ime, String priimek, String prebivalisce, int starost, String opis, String spol, String datum){
        this.ime = ime;
        this.priimek = priimek;
        this.prebivalisce = prebivalisce;
        this.starost = starost;
        this.opis = opis;
        this. spol = spol;
        this.datum = datum;
    }

    public String toString(){
        return ("Ime: "+this.ime + "\nPriimek: "+this.priimek+"\nPrebivališče: "+this.prebivalisce+"\nStarost: "+this.starost+'
    }

}
```

Slika 5.48 Razred oseba.

Za prikaz koledarja smo uporabili »dialog«. Ustvarili smo funkcijo `nastaviDatum()`, ki pokliče dialog z id-jem 2, ta pa nato pokliče naslednjo funkcijo `DatePickerDialog()`, ki nam odpre dialog. To lahko vidimo na sliki 5.49.

```
public void nastaviDatum(View view) {
    showDialog( id: 2);
}

@Override
protected Dialog onCreateDialog(int id) {
    if (id == 2) {
        return new DatePickerDialog( context: this, myDateListener, leto, mesec, dan);
    }
    return null;
}

private DatePickerDialog.OnDateSetListener myDateListener = new DatePickerDialog.OnDateSetListener() {
            @Override
            public void onDateSet(DatePicker arg0, int arg1, int arg2, int arg3) {
                prikaziDatum(arg1,  mesec: arg2+1, arg3);
            }
        };

private void prikaziDatum(int leto, int mesec, int dan) {
    datumEditText.setText(dan + "/" + mesec + "/" + leto );
}
```

Slika 5.49 Funkcije za prikaz koledarja.

Shranjevanje oseb v pomnilnik lahko izvedemo na več načinov. Podatke lahko shranimo v začasno datoteko s pomočjo funkcije `File.createTempFile()` ali pa naredimo kar globalno spremenljivko. Za klic globalne spremenljivke moramo vedno dodati spredaj še ime razreda, v katerem se spremenljivka nahaja. Kodo si lahko ogledamo na sliki 5.50.

```
public void shraniVpomnilnik(View v){
    String osebje = "";
    for(int i = 0; i < indexOsebe; i++){
        osebje = osebje + "\n"+osebe[i].toString();
    }
    MainActivity.osebe = osebje;
    Toast.makeText(getApplicationContext(),  text: "Uspešno shranjeno!"  ,Toast.LENGTH_SHORT).show();
}

public void preberiIzPomnilnika(View v){
    IzpisPomnilnikaTextView.setText(MainActivity.osebe);
    Toast.makeText(getApplicationContext(),  text: "Izpisano!"  ,Toast.LENGTH_SHORT).show();
}
```

Slika 5.50 Shranjevanje v globalno spremenljivko.

Za shranjevanje osebe v pravo datoteko moramo uporabiti funkcijo `FileOutputStream()` in nato `openFileOutput()`, da datoteka nastane. Ko imamo datoteko, lahko s pomočjo funkcije `OutputStreamWriter()` začnemo pisati vanjo. Ko s tem končamo, moramo vedno datoteko zapreti s `file.close()`. Ostane nam še branje iz datoteke, ki pa ga zvedemo s funkcijo `InputStreamReader()`, nato pokličemo še funkcijo `BufferedReader()` in

začnemo brati z ukazom `file.readLine()`. Za izbris datoteke lahko uporabimo funkcijo `file.delete()`. Shranjevanje imamo prikazano na sliki 5.51.

```
public void shraniVdatoteko(View v){
    String osebje = "";
    for(int i = 0; i < indexOsebe; i++){
        osebje = osebje + "\n"+osebe[i].toString();
    }
    try {
        OutputStreamWriter osw = new OutputStreamWriter(fOut);
        osw.write(osebje);
        osw.flush();
        osw.close();
        Toast.makeText(getApplicationContext(), text: "Uspešno shranjeno v datoteko.",Toast.LENGTH_SHORT).show();
    }catch (Exception e){
        Toast.makeText(getApplicationContext(), text: "Prišlo je do napake pri shranjevanju v datoteko." +e.getMessage(),Toast.LENGTH_SHORT).show();
    }
}

public void preberiIzDatoteke(View v){
    izpisDatotekeTextView.setText("");
    String line = "";
    try {
        FileInputStream fIn = openFileInput( name: "osebe.txt");
        InputStreamReader isr = new InputStreamReader(fIn);
        BufferedReader in = new BufferedReader(isr);
        while((line = in.readLine()) != null) {
            izpisDatotekeTextView.setText(izpisDatotekeTextView.getText() + "\n" + line);
        }
        Toast.makeText(getApplicationContext(), text: "Uspešno prebrano." + line,Toast.LENGTH_SHORT).show();
    }catch (Exception e){
        Toast.makeText(getApplicationContext(), text: "Prišlo je do napake pri branju iz datoteke." +e.getMessage(),Toast.LENGTH_SHORT).show();
    }
}

public void izbrisiDatoteko(View v){
    izpisDatotekeTextView.setText("");
    try {
        File dir = getFilesDir();
        File file = new File(dir, child: "osebe.txt");
        boolean deleted = file.delete();
        Toast.makeText(getApplicationContext(), text: "Datoteka uspešno izbrisana.",Toast.LENGTH_SHORT).show();
    } catch (Exception e) {
        // File permission problems are caught here.
        Toast.makeText(getApplicationContext(), text: "Prišlo je do napake pri brisanju datoteke."+e.getMessage(),Toast.LENGTH_SHORT).show();
    }

}
```

Slika 5.51 Shranjevanje, branje in brisanje datoteke.

5.2.1.11 Rekurzija in iskanje v binarnem drevesu in povezanem seznamu

Za prikaz aktivnosti smo uporabili drsnik z linearno, vertikalno razporeditvijo. Najprej moramo vnesti neko številko in podatek, ki ji pripada. S klikom na gumb `shrani` se nam ustvari nov objekt (slika 5.52), kateremu lahko dodamo atribute; `številko`, `podatek` in nato še `levega` in `desnega otroka` za binarno shranjevanje v drevo ali pa samo vezavo na naslednji objekt za povezan seznam.

```
public class objektStevilkaPodatek {

    int stevilka;
    String podatek = "";
    objektStevilkaPodatek next = null;
    objektStevilkaPodatek leftChild = null;
    objektStevilkaPodatek rightChild = null;


    public void dodajStevilkaPodatek(int stevilka, String podatek){
        this.stevilka = stevilka;
        this.podatek = podatek;

    }

    public String toString() { return ("Stevilka: "+this.stevilka + "\nPodatek: "+this.podatek); }

}
```

Slika 5.52 Objekt za shranjevanje številke in podatka.

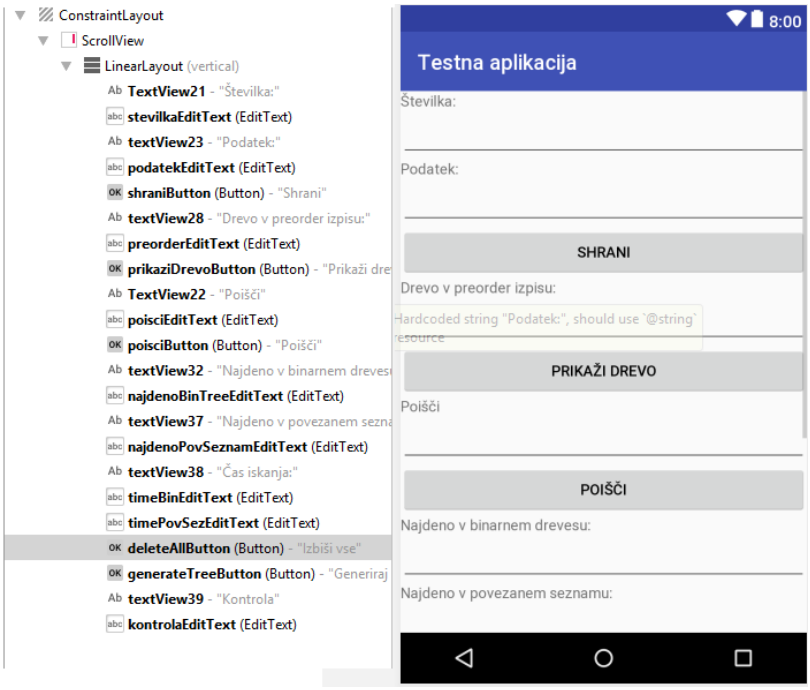
Pod gumbom za shranjevanje (slika 5.54) imamo tudi gumb za izpis. Ta gumb nam binarno drevo izpiše v premem obhodu. Nato imamo gumb za iskanje objekta, kar pomeni da moramo najprej vnesti željeno številko in na podlagi le-te se poišče objekt v binarnem drevesu in v povezanem seznamu, ta podatek pa se nato izpiše. Pod podatkom se nam izpiše še čas iskanja elementa, ki ga je potreboval algoritem za iskanje v binarnem drevesu in povezanem seznamu. Funkcijo za merjenje časa med iskanjem elementa si lahko ogledamo na sliki 5.53.

```
public void poisci(View v){
    kontrolaEditText.setText(kontrolaEditText.getText() + "\nIskanje v binarnem drevesu se je začelo.");
    long firstTime = System.currentTimeMillis();
    findInBinTree();
    long secTime = System.currentTimeMillis();
    long elapsed = secTime - firstTime;
    timeBinEditText.setText("Binarno drevo: "+cas(elapsed)+"::"+elapsed);
    kontrolaEditText.setText(kontrolaEditText.getText() + "\nIskanje v binarnem drevesu se je končalo.");
    kontrolaEditText.setText(kontrolaEditText.getText() + "\nIskanje v povezanem seznamu se je začelo.");
    long firstTime1 = System.currentTimeMillis();
    findInLinkedList();
    long secTime2 = System.currentTimeMillis();
    long elapsed3 = secTime2 - firstTime1;
    timePovSezEditText.setText("Povezan seznam: "+cas(elapsed3) +"::"+elapsed3);
    kontrolaEditText.setText(kontrolaEditText.getText() + "\nIskanje v povezanem seznamu se je končalo.");
    poisciEditText.setText("");
    Toast.makeText( context: rekurzija.this,  text: "Iskanje je uspešno zaključeno.", Toast.LENGTH_SHORT ).show();

}
```

Slika 5.53 Funkcija za merjenje časa med iskanjem elementa.

Pod izpisanimi časi imamo gumb za brisanje vseh podatkov. Če pritisnemo na ta gumb, se nam vsi okvirji izpraznijo in vsi podatki, ki smo jih imeli shranjene, se nam izbrišejo. Gumb `generiraj` drevo nam ustvari novo drevo s 7809 objekti. Da bomo čas lahko primerjali s

hibridno aplikacijo, so ti objekti dodani v istem zaporedju kot tam. Na koncu imamo še okvir za kontrolo, v katerem se nam izpisuje vsak proces, ki se trenutno izvaja.



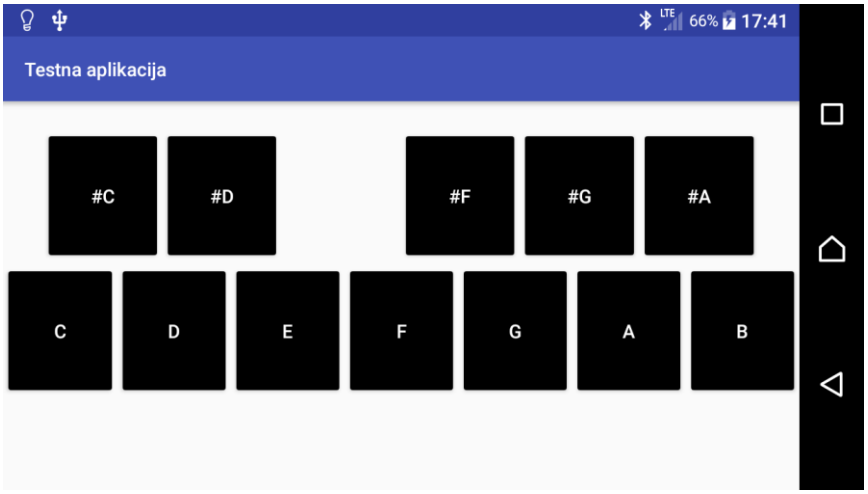
Slika 5.54 Izgled aktivnosti za shranjevanje podatkov v binarno drevo in povezan seznam.

5.2.1.12 Igranje klavirja

Klavir smo sestavili iz 13-ih gumbov. Izbrali smo dve linearni horizontalni postavitvi gumbov; v prvi imamo 6 gumbov, ki so višaji, v drugi pa 7 gumbov. Ker višaja za ton e nimamo, smo ta gumb skrili. Da se zgornji gumbi poravnajo na sredino s spodnjimi, smo uporabili še funkcijo `android:paddingRight` in `android:paddingLeft`. Gumbom smo nastavili še črno ozadje, bele črke in pa malo višjo višino. To lahko vidimo na sliki 5.55 in 5.56.

```
<Button
    android:id="@+id/ccButton"
    android:layout_width="wrap_content"
    android:layout_height="100dp"
    android:layout_weight="1"
    android:text="#c"
    android:backgroundTint="#000000"
    android:textColor="#ffffff"
    android:onClick="onClick"/>
```

Slika 5.55 Lastnosti gumba.



Slika 5.56 Izgled aktivnosti igranje klavirja.

Ob vsakem pritisku na gumb se sproži funkcija, ki najprej pogleda, kateri gumb smo pritisnili, nato na podlagi tega gumba zaigra ustrezen ton. Za predvajanje zvoka smo uporabili funkcijo `MediaPlayer.create()`, ki v spremenljivko najprej shrani pot do zvoka, nato pa `MediaPlayer.start()`, ki začne predvajati zvok. Funkcije si lahko ogledamo na sliki 5.57.

```
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.cButton:
                final MediaPlayer c = MediaPlayer.create( context: piano.this, R.raw.c);
                c.start();
            break;
        case R.id.ccButton:
                final MediaPlayer cc = MediaPlayer.create( context: piano.this, R.raw.cc);
                cc.start();
            break;
        case R.id.dButton:
                final MediaPlayer d = MediaPlayer.create( context: piano.this, R.raw.d);
                d.start();
            break;
```

Slika 5.57 Del kode, ki na podlagi pritisnjenega gumba sproži ustrezen zvok.

Poglavje 6 Primerjava aplikacij

Končano hibridno in izvorno aplikacijo bomo primerjali po hitrosti, izgledu uporabniškega vmesnika, klica vgrajenih funkcij, zvoku, času izdelave ene in druge aplikacije, urejenosti in strukturi datotek in pa delovanju na različnih platformah.

6.1 Hitrost

Zagon same aplikacije je pri hibridni in izvorni obliki približno enak, hibridna aplikacija se sicer odpira počasneje, ampak je to zanemarljivo majhna razlika.

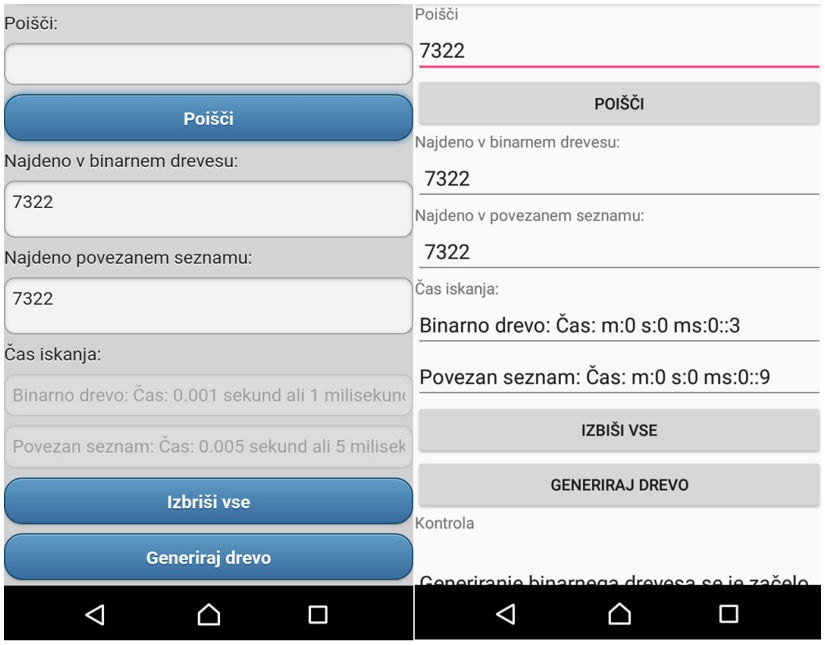
Pri aktivnosti tri v vrsto se pozna razlika pri hitri izbiri kvadratov. Ko pri izvorni aplikaciji kliknemo na kvadrat, da se prikaže križec ali pa krožec, se to zgodi takoj neglede na hitrost pritiskov, pri hibridni aplikaciji pa se slika malo počasneje pokaže, če hitro klikamo po kvadratih. Če na primer kliknemo kolikor hitro lahko na tri kvadrate, se ob tretjem pritisku postavlja komaj druga slika.

Velika razlika je pri aktivnosti kača. Ko pritiskamo na gumb `pospeši`, se pospešuje samo do neke stopnje. Razlika je tako očitna, da lahko pri hibridni aplikaciji sledimo, kako se kvadratek premika po zaslonu, pri izvorni aplikaciji pa pride do take hitrosti premika kvadratka, da ga s prostim očesom ne zaznamo več točno, zaznavamo bolj kote, kjer spremeni smer.

Pri aktivnosti zanke, kjer merimo hitrost izvajanja zank, se hibridna aplikacija obnese veliko boljše. Pri hibridni aplikaciji lahko izmerimo čas 5 gnezdenih zank, 6 jih ne zmore več in aplikacija se podre. Pri izvorni aplikaciji pa se problem pojavi že pri 4 gnezdenih zankah, ki jih aplikacija ne more več izvesti. Čas izvedbe zank si lahko ogledamo tudi na sliki 6.1.

Slika 6.1 Levo čas izvorne aplikacije, desno čas hibridne aplikacije.

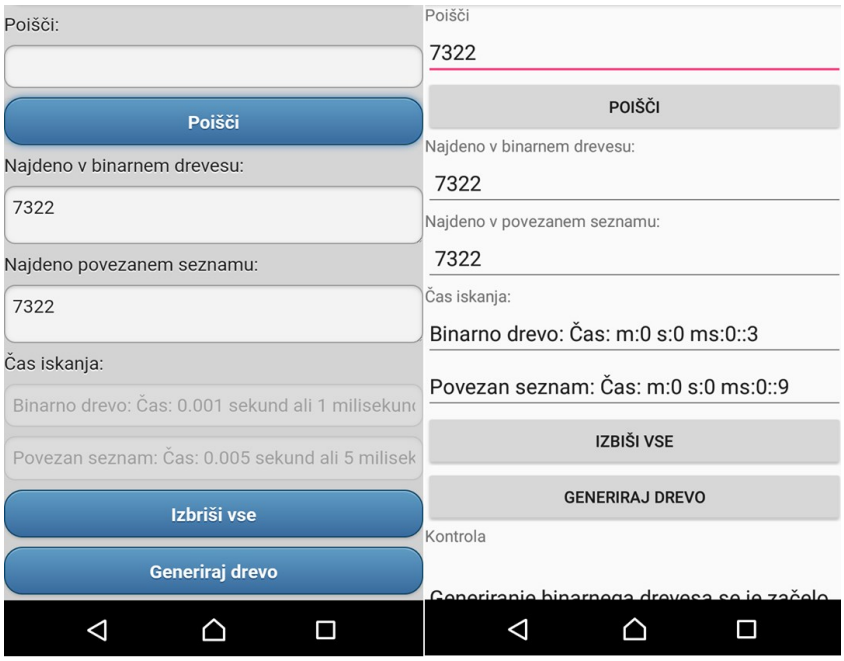
Ostane nam še aktivnost, pri kateri s pomočjo rekurzije iščemo objekte, ki smo si jih shranili, in sicer neko številko in podatek, ki mu pripada. S pomočjo gumba `generiraj drevo` bomo ustvarili binarno drevo, ki ima 7809 objektov in sicer v obeh aplikacijah vnesene v istem zaporedju. Za iskanje si bomo izbrali objekt s številko 7322, ki smo ga zadnjega vnesli. Izkaže se, da je čas pri izvorni aplikaciji veliko boljši; manj kot 1 milisekundo oziroma 0,003, pri hibridni pa kar 5 milisekunde pri povezanem seznamu. To lahko vidimo na sliki 6.2.



Slika 6.2 Levo čas pri hibridni aplikaciji, desno pri izvorni aplikaciji.

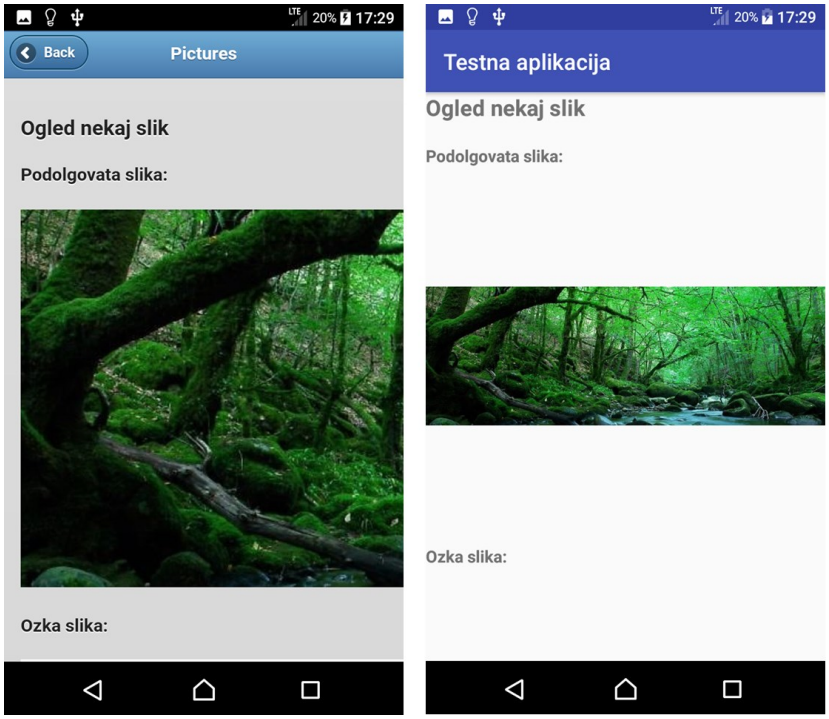
6.2 Uporabniški vmesnik

Izgled uporabniškega vmesnika se med aplikacijami zelo razlikuje. Med izdelavo aplikacij nismo dali veliko pozornosti na obliko uporabniškega vmesnika. Pazili smo samo na postavitev elementov, oblike elementov pa nismo spreminjali. Gumbi so veliko lepše izoblikovani v hibridni aplikaciji, prav tako je privzeta tema na hibridni aplikaciji lepša, saj vsebuje tudi nekaj barv in takoj spremeni sam izgled in udobnost uporabniškega vmesnika. Primer je na sliki 6.3.



Slika 6.3 Levo izgled hibridne aplikacije, desno izvorne aplikacije.

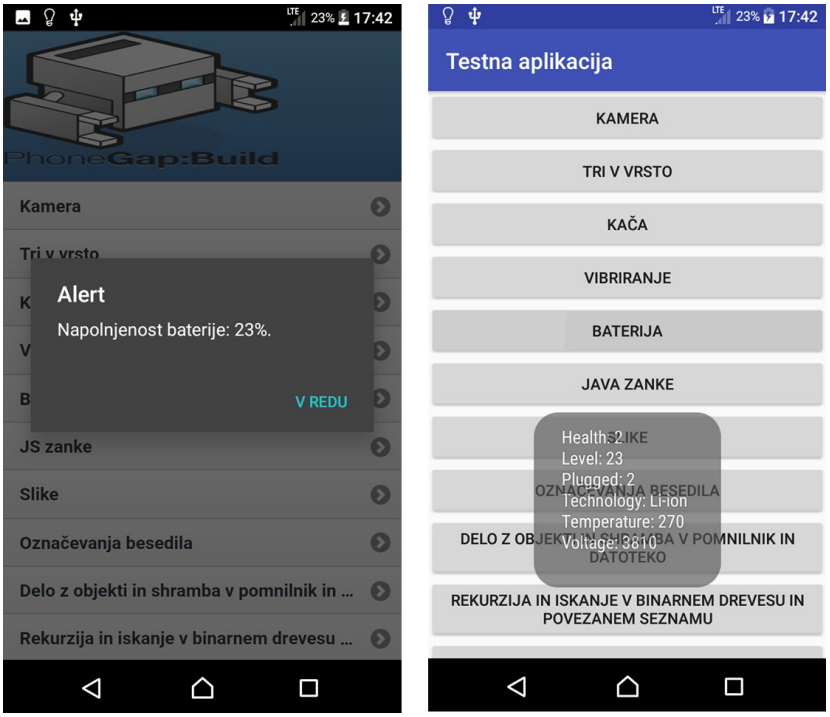
Prav tako lahko opazimo tudi razliko pri aktivnosti slike, kjer smo dodajali različno velike slike. Za visoke slike lepo poskrbi drsnik ob strani zaslona in nam s pomikanjem gor in dol po zaslonu lepo prikaže sliko na obeh aplikacijah. Problem pa nastane pri široki sliki. Izvorna aplikacija sama poskrbi za velikost tako, da jo skrči na dimenzije, ki ustrezajo zaslonu. Pri hibridni aplikaciji pa slika ostane enako široka, kot je bila, in nam desni del slike preprosto odreže. To lahko vidimo na sliki 6.4.



Slika 6.4 Izgled aktivnosti slike; levo hibridna, desno izvorna aplikacija.

6.3 Klic vgrajenih funkcij pametne mobilne naprave

PhoneGap je dobro poskrbel tudi za dostop vgrajenih funkcij pametne mobilne naprave, kot so na primer kamera, vibracija, baterija, NFC, sporočila itd. S hibridno aplikacijo lahko dostopamo do vseh teh funkcij, ne moremo pa zaznati vseh stvari povezanih s temi funkcijami kot jih lahko pri izvorni aplikaciji. Dober primer je pridobitev podatkov o bateriji; pri hibridni aplikaciji lahko dostopamo do trenutne napolnjenosti baterije, lahko tudi izvemo, če se baterija polni, če smo jo izklopili iz polnjenja, ne moremo pa dostopati do podatkov, kot na primer do tehnologije baterije, temperature baterije, voltaže baterije. Primer izpisa podatkov o bateriji je prikazan na sliki 6.5.



Slika 6.5 Podatki o bateriji; levo hibridna, desno izvorna aplikacija.

6.4 Zvok

Pri izvorni aplikaciji se ob dotiku na gumbe sproži zvok klika, ki ga imamo nastavljenega na pametni mobilni napravi. Problem nastane pri aktivnosti igranje klavirja, ker so toni sestavljeni iz gumbov, kar privede do tega, da se ob kliku na gumb sproži zvok tona in zvok klika na gumb. Pri hibridni aplikaciji tega ni in se sproži samo zvok tona, manjka pa zvok klika na gumb, ko stisnemo ostale gumbe.

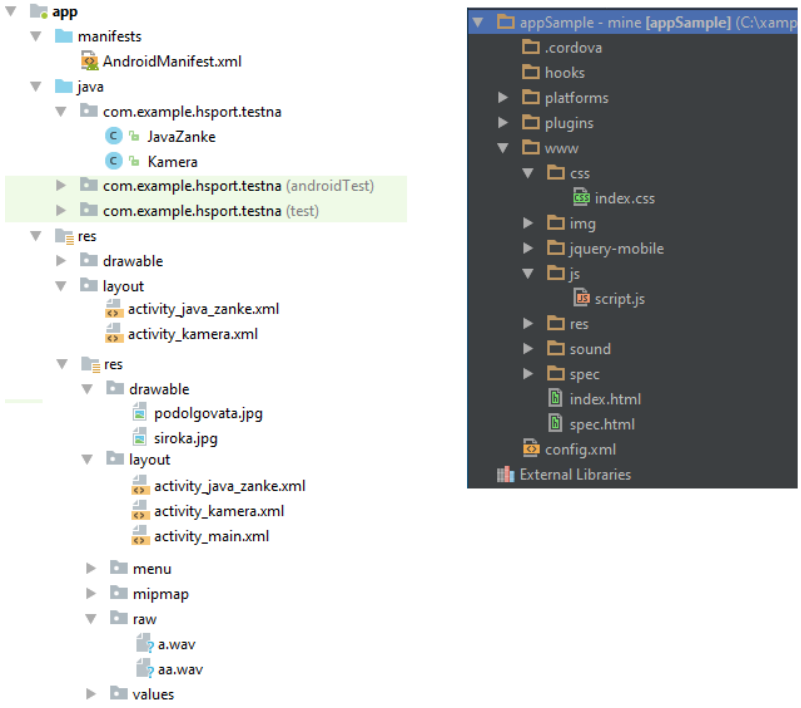
6.5 Čas izdelave aplikacije

Če poznamo samo programske jezike, ki jih uporabljamo za te aplikacije, in ničesar drugega, se veliko prej naučimo narediti hibridno aplikacijo, pa tudi čas nastanka aplikacije je manjši pri hibridni aplikaciji. Ko pri hibridni aplikaciji neko stvar naredimo oziroma popravimo, le osvežimo brskalnik na pametni mobilni napravi in stvar je že prikazana, za razliko od izvorne aplikacije, kjer se aplikacija najprej na novo prenese na pametno mobilno napravo in se potem na novo namesti in zažene. Če si za primer vzamemo uporabo vibracije, lahko pri programiranju hibridne aplikacije to funkcijo samo pokličemo, ker je že vgrajena in nam vrne podatke, ki jih želimo, pri izvorni aplikaciji pa se moramo najprej naučiti, kako sama funkcija deluje in kako lahko preko nje pridemo do željenih podatkov, kar nam vzame veliko več časa. Če pa se

začnemo programskih jezikov učiti na novo, bomo prej naredili izvorno aplikacijo, saj imamo tukaj samo en programski jezik, v hibridni aplikaciji pa najmanj 3 programske jezike.

6.6 Urejenost

Pri urejenosti smo se osredotočili na datoteke, torej koliko datotek je, kje so, kaj je v njih in kako izgleda koda v njih. Pri izvorni aplikaciji imamo datoteke lepo razporejene, tako da so aktivnosti v svoji mapi, manifest v svoji mapi, zvoki, slike v svoji mapi, razredi v svoji mapi itd. Seveda je to enako urejeno tudi pri hibridni aplikaciji, velika razlika pa je v tem, da imamo pri izvorni aplikaciji za vsako aktivnost svoj razred oziroma datoteko in v njej kodo za delovanje točno te aktivnosti, prav tako imamo tudi pri obliki aktivnosti za vsako aktivnost svojo datoteko, ki ji določa obliko. Pri hibridni aplikaciji pa imamo samo dve datoteki, in sicer je ena za obliko vseh aktivnosti in druga za samo delovanje aktivnosti. Izgled strukture datotek si lahko ogledamo na sliki 6.6. Če imamo neko stvar za popraviti, se veliko lepše znajdemo v izvorni aplikaciji in tudi prej najdemo del, ki ga želimo popraviti. Seveda lahko tudi pri hibridni aplikaciji te stvari razdelimo na različne dele, ampak imamo potem veliko več dela in popravljanja. Dober primer tega je, da dokler imamo vse v eni datoteki nam gumbi `nazaj`, torej iz trenutne aktivnosti na prejšnjo aktivnost, lepo delujejo, ko pa kodo razdelimo v različne datoteke, nam gumbi ne delujejo več. Pri izvorni aplikaciji je velik plus tudi to, da lahko pri izdelavi oblike aplikacije izbiramo elemente iz seznama in jih dodajamo na našo aktivnost, torej nam ni potrebno skrbeti, kako je oblika sprogramirana, ampak samo primemo element in ga povlečemo na virtualen zaslon.



Slika 6.6 Struktura datotek na levi strani izvorne in desni strani hibridne aplikacije.

6.7 Delovanje na različnih platformah

Pri izvorni aplikaciji vemo, da jo lahko namestimo samo na platformo, za katero je bila izdelana, v našem primeru jo lahko namestimo samo na pametni mobilni aparat, ki ima za operacijski sistem Android. Aplikacija lahko deluje na različnih verzijah Androida. Ko prvič ustvarimo aplikacijo, moramo izbrati tudi najmanjšo verzijo, na kateri bo aplikacija še delovala, tako da če jo namestimo na verzijo, ki je nižje od minimalne zahtevane, aplikacija ne bo delovala oziroma jo sistem ne bo pustil namestiti. Aplikacijo smo namestili na različne Android mobilne naprave z verzijo 7.0, ker smo izbrali minimalno verzijo 7.0 in aplikacija je delovala povsod.

Hibridna aplikacija bi morala delovati na različnih platformah in različnih verzijah sistema. Aplikacijo smo namestili na različne android verzije in naprave.

- Tablični računalnik Lenovo Tab 10 s sistemom Android verzije 6.0.1: deluje vse.
- Tablični računalnik Overmax SteelCore 10+II Android verzije 4.1.1: deluje vse.
- Pametna mobilna naprava Sony Xperia XA Android verzije 7.0: deluje vse.
- Pametna mobilna naprava Sony Ericsson Arc S Android verzije 4.1.1: pride do napake pri razčlenjevanju paketa, aplikacija se ne more namestiti.

- Pametna mobilna naprava Huawei P9 Lite Android verzije 7.0: deluje vse.

Poglavje 7 Sklepne ugotovitve

Orodja za izdelavo hibridnih aplikacij so in bodo vedno v zaostanku z orodji za izvorne aplikacije. Vedno, ko dobimo novo verzijo nekega sistema, dobimo z njim tudi vsa orodja in dostope do vseh funkcij. Pri hibridnih aplikacijah pa moramo vedno še nekaj časa počakati, da nam funkcije nove verzije sistema vgradijo v orodje oziroma sprogramirajo funkcije za dostop, ki jih bomo lahko potem programerji uporabljali.

Hibridna in izvorna implementacija imata dobre in slabe strani in na podlagi le-teh moramo sami presoditi, katera vrsta implementacije naše aplikacije bi bila najboljša izbira.

Pri izdelavi testne aplikacije smo prišli do različnih ugotovitev (tabela 1).

- Hitrost je pri obeh aplikacijah podobna. Med njima so zelo majhne razlike, bi pa bilo vseeno bolj pametno uporabljati izvorni način implementacije, če bomo izdelovali aplikacijo, ki mora biti hitra in točna.

- Pri uporabniškem vmesniku smo ugotovili, da ima hibridna aplikacija lepšo obliko z manj dela pri oblikovanju. Če nam je pomemben videz uporabniškega vmesnika in hitrost izdelave le-tega, bi se tako bilo boljše odločiti za hibridni razvoj.

- Klic vgrajenih funkcij pametnih mobilnih naprav deluje na obeh implementacijah, bi pa bila definitivno boljša izbira izvorna implementacija, če bi morali narediti aplikacijo, ki uporablja neko funkcijo, kot je na primer NFC in z njo delali tudi druge stvari, ki niso namenjene za to funkcijo (kot bi bilo branje čipa iz bančne kartice z uporabo funkcije NFC).

- Pri zvoku je vseeno, katero implementacijo bomo izbrali.

- Če hočemo aplikacijo narediti čim hitreje, bo boljša izbira hibridna implementacija.

- Tudi če moramo narediti aplikacijo, ki mora delovati na različnih sistemih, je boljša izbira hibridna implementacija.

Odločitev za izdelavo hibridne ali izvorne aplikacije ni preprosta in jo je težko definirati. Najbolje je, da si najprej aplikacijo začrtamo in določimo stvari, ki so nam pri izdelavi le-te

pomembne. Nato jih primerjamo, kako se obnesejo pri hibridni ali pa izvorni aplikaciji, in šele zatem lahko določimo, katero implementacijo bomo izbrali. Če bi hoteli hitro narediti kakšno preprosto aplikacijo, kot bi bil recimo opis znamenitosti v nekem mestu, bi bilo bolje izbrati hibridno implementacijo, ker bi bila sama izdelava aplikacije hitrejša, uporabniški vmesnik bi bil že lepo oblikovan, odzivnost aplikacije bi bila dovolj dobra in delovala bi na različnih platformah. Če bi izbrali izvorni razvoj, bi porabili več časa pri izdelovanju, aplikacija pa bi delovala samo na eni platformi. Če pa bi imeli neko zahtevno aplikacijo, ki potrebuje dobro odzivnost in dostop do vseh podatkov neke funkcije, ki jo ima pametna mobilna naprava (na primer dostop do oddaljenega strežnika preko interneta), pa bi bilo bolje uporabiti izvorni način implementacije, saj bi imeli dostop do vseh funkcij in tudi odzivnost bi bila zelo dobra.

	HIBRIDNA APLIKACIJA	IZVORNA APLIKACIJA
HITROST		**X**
UPORABNIŠKI VMESNIK	**X**	
KLIC FUNKCIJ		**X**
ZVOK	**X**	**X**
ČAS IZDELAVE	**X**	
UREJENOST		**X**
RAZLIČNE PLATFORME	**X**	

Tabela 1: Prednosti hibridne in izvorne aplikacije.

Uspelo nam je ugotoviti prednosti hibridne in izvorne aplikacije, lahko pa bi dodali še veliko drugih testov. Lahko bi testirali hibridno aplikacijo tudi na drugih platformah in ne samo na Androidu, tudi izvorno aplikacijo bi lahko naredili za druge platforme, lahko bi se v aplikaciji poskusili povezati z bazo podatkov prek spleta ali pa uporabili še kakšne druge vgrajene funkcije pametne mobilne naprave.

Poglavje 8 Uporabljeni viri

[1] „*Phone Scoop*,“ [Elektronski]. Available: http://www.phonescoop.com/glossary/term.php?gid=131. [Poskus dostopa 5 december 2016].

[2] A. Nusca, „*Smartphone vs. feature phone arms race heats up; which did you buy?*,“ 20 8 2009. [Elektronski]. Available: http://www.zdnet.com/article/smartphone-vs-feature-phone-arms-race-heats-up-which-did-you-buy/. [Poskus dostopa 5 december 2016].

[3] „*20 years of the smartphone: an evolution in pictures,*“ [Elektronski]. Available: http://www.telegraph.co.uk/technology/mobile-phones/11037225/20-years-of-the-smartphone-an-evolution-in-pictures.html?frame=3007865. [Poskus dostopa 6 december 2016].

[4] P. Budmar, „*Why Japanese smartphones never went global*,“ PC World, 11 Julij 2011. [Elektronski]. Available: http://www.pcworld.idg.com.au/article/430254/why_japanese_smartphones_never_went_global/. [Poskus dostopa 6 december 2016].

[5] D. Reisinger, „*Worldwide smartphone user base hits 1 billion*,“ 17 Oktober 2012. [Elektronski]. Available: https://www.cnet.com/news/worldwide-smartphone-user-base-hits-1-billion/. [Poskus dostopa 6 december 2016].

[6] „*ComScore Reports January 2016 U.S. Smartphone Subscriber Market Share*,“ 3 Marec 2016. [Elektronski]. Available: https://www.comscore.com/Insights/Rankings/comScore-Reports-January-2016-US-Smartphone-Subscriber-Market-Share. [Poskus dostopa 6 december 2016].

[7] B. Elgin, „*Google Buys Android for Its Mobile Arsenal*,“ 17 Avgust 2005. [Elektronski]. Available:

https://www.bloomberg.com/businessweek/technology/content/aug2005/tc20050817_0949_tc024.htm. [Poskus dostopa 2016 december 7].

[8] B. Cha, „*All T-Mobile retail stores to carry G1*,“ 23 Januar 2009. [Elektronski]. Available: https://www.cnet.com/news/all-t-mobile-retail-stores-to-carry-g1/. [Poskus dostopa 7 december 2016].

[9] A. M. I. M. ISAAC, „*Android OS Hack Gives Virtual Early Upgrade*,“ 4 November 2011. [Elektronski]. Available: https://www.wired.com/2011/04/cyanogenmod-android/. [Poskus dostopa 7 december 2016].

[10] J. Kopstein, „*Access Denied: why Android's broken promise of unlocked bootloaders needs to be fixed,*“ 20 November 2012. [Elektronski]. Available: http://www.theverge.com/2012/11/20/3666668/access-denied-android-unlocked-bootloaders. [Poskus dostopa 7 december 2016].

[11] „*Developer Economics Q1 2015: State of the Developer Nation*,“ 17 Februar 2015. [Elektronski]. Available: https://www.developereconomics.com/reports/developer-economics-q1-2015. [Poskus dostopa 7 december 2016].

[12] J. CALLAHAM, „*Google says there are now 1.4 billion active Android devices worldwide*,“ 29 September 2015. [Elektronski]. Available: http://www.androidcentral.com/google-says-there-are-now-14-billion-active-android-devices-worldwide. [Poskus dostopa 7 december 2016].

[13] H. SOFFAR, „*The advantages and disadvantages of Android mobile phones*,“ 10 Junij 2015. [Elektronski]. Available: http://www.online-sciences.com/technology/the-advantages-and-disadvantages-of-android-mobile-phones-2/. [Poskus dostopa 7 december 2016].

[14] A. Handy, „*Twenty years of Java through its creator's eyes*,“ 20 Maj 2015. [Elektronski]. Available: http://sdtimes.com/twenty-years-of-java-through-its-creators-eyes/. [Poskus dostopa 8 december 2016].

[15] „*The History of Java Technology*,“ [Elektronski]. Available: http://www.oracle.com/technetwork/java/javase/overview/javahistory-index-198355.html. [Poskus dostopa 8 december 2016].

[16] H. Kabutz, „*Once Upon an Oak ...,*“ 15 Julij 2003. [Elektronski]. Available: http://www.artima.com/weblogs/viewpost.jsp?thread=7555. [Poskus dostopa 8 december 2016].

[17] „*Java SE versions history*,“ 21 Marec 2014. [Elektronski]. Available: http://www.codejava.net/java-se/java-se-versions-history. [Poskus dostopa 8 december 2016].

[18] „*J2SE vs J2ME vs J2EE….What's the difference?*,“ [Elektronski]. Available: http://www.geeksforgeeks.org/j2se-vs-j2me-vs-j2ee-whats-the-difference/. [Poskus dostopa 8 december 2016].

[19] N. Langley, „*Write once, run anywhere?*,“ Maj 2002. [Elektronski]. Available: http://www.computerweekly.com/feature/Write-once-run-anywhere. [Poskus dostopa 8 december 2016].

[20] „*The Java™ Tutorials,*“ [Elektronski]. Available: https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html. [Poskus dostopa 8 december 2016].

[21] „*Meet Android Studio*,“ [Elektronski]. Available: https://developer.android.com/studio/intro/index.html. [Poskus dostopa 12 december 2016].

[22] B. Jakuben, „*Why Google Loves Developers*,“ 16 Maj 2013. [Elektronski]. Available: http://blog.teamtreehouse.com/why-google-loves-developers. [Poskus dostopa 12 december 2016].

[23] R. S. John, „*Android Studio vs. Eclipse: What You Need To Know*,“ [Elektronski]. Available: https://www.airpair.com/android/android-studio-vs-eclipse. [Poskus dostopa 12 december 2016].

[24] „*Meet Android Studio*,“ [Elektronski]. Available: https://developer.android.com/studio/intro/index.html. [Poskus dostopa 12 december 2016].

[25] O. white, „*IDEs vs. Build Tools: How Eclipse, IntelliJ IDEA & NetBeans users work with Maven, Ant, SBT & Gradle*,“ 14 oktober 2014. [Elektronski]. Available:

https://zeroturnaround.com/rebellabs/ides-vs-build-tools-how-eclipse-intellij-idea-netbeans-users-work-with-maven-ant-sbt-gradle/. [Poskus dostopa 12 december 2016].

[26] G. Cernosek, „*A brief history of Eclipse*,“ 15 November 2005. [Elektronski]. Available: http://www.ibm.com/developerworks/rational/library/nov05/cernosek/. [Poskus dostopa 12 december 2016].

[27] C. Ziegler, „*Microsoft talks Windows Phone 7 Series development ahead of GDC: Silverlight, XNA, and no backward compatibility,*“ 3 april 2010. [Elektronski]. Available: https://www.engadget.com/2010/03/04/microsoft-talks-windows-phone-7-series-development-ahead-of-gdc/. [Poskus dostopa 13 december 2016].

[28] E. PROTALINSKI, „*Nokia now controls 90% of the Windows Phone 8 market, with the low-end Lumia 520 grabbing over 35% shar*e,“ 27 November 2013. [Elektronski]. Available: http://thenextweb.com/microsoft/2013/11/27/nokia-now-controls-90-windows-phone-8-market-low-end-lumia-520-accounting-35-share/. [Poskus dostopa 13 december 2016].

[29] M. D. Team, „*Open Letter from CEO Stephen Elop, Nokia and CEO Steve Ballmer, Microsoft*,“ 11 Februar 2011. [Elektronski]. Available: https://blogs.windows.com/devices/2011/02/11/open-letter-from-ceo-stephen-elop-nokia-and-ceo-steve-ballmer-microsoft/#cAM2SLX3FluHaao1.97. [Poskus dostopa december 13 2016].

[30] L. Dignan, „*Nokia to rely on Microsoft's Windows Phone 7: 'This is now a three horse race'*,“ 11 Februar 2011. [Elektronski]. Available: http://www.zdnet.com/article/nokia-to-rely-on-microsofts-windows-phone-7-this-is-now-a-three-horse-race/. [Poskus dostopa 13 december 2016].

[31] R. Cheng, „*Nokia on the edge: Inside an icon's fight for survival,*“ 18 December 2012. [Elektronski]. Available: https://www.cnet.com/news/nokia-on-the-edge-inside-an-icons-fight-for-survival/. [Poskus dostopa 13 december 2016].

[32] „*Nokia and Microsoft Announce Plans for a Broad Strategic Partnership to Build a New Global Mobile Ecosystem,*“ 10 Februar 2011. [Elektronski]. Available: https://news.microsoft.com/2011/02/10/nokia-and-microsoft-announce-plans-for-a-broad-strategic-partnership-to-build-a-new-global-mobile-

ecosystem/#sm.0007x8kamkn4cwf10mq1m15vgvrud#7gtDvqSRXcvMJCB8.97. [Poskus dostopa 13 december 2016].

[33] „*Microsoft Unveils Windows Phone 7 Series,*“ 15 Februar 2010. [Elektronski]. Available: https://news.microsoft.com/2010/02/15/microsoft-unveils-windows-phone-7-series/#sm.0007x8kamkn4cwf10mq1m15vgvrud#JL2651PZRYsVpfv8.97. [Poskus dostopa 14 december 2016].

[34] J. Dolcourt, „*Windows Phone 7.5 Mango review: Up to speed,*“ 27 September 2011. [Elektronski]. Available: https://www.cnet.com/news/windows-phone-7-5-mango-review-up-to-speed/. [Poskus dostopa 14 december 2016].

[35] T. Warren, „*Microsoft details full Windows Phone 'Tango' 256MB RAM device limitations,*“ 8 Marec 2012. [Elektronski]. Available: http://www.theverge.com/2012/3/8/2853948/windows-phone-tango-256mb-ram-restrictions. [Poskus dostopa 14 december 2016].

[36] D. Reisinger, „*Microsoft to end Windows Phone 7.8 and 8 support in 201*4,“ 18 Marec 2013. [Elektronski]. Available: https://www.cnet.com/news/microsoft-to-end-windows-phone-7-8-and-8-support-in-2014/. [Poskus dostopa 14 december 2016].

[37] T. Holwerda, „*Microsoft: Windows Phone 8 To Use NT Kernel*,“ 2 Februar 2012. [Elektronski]. Available: http://www.osnews.com/story/25574/Microsoft_Windows_Phone_8_To_Use_NT_Kernel/. [Poskus dostopa 14 december 2016].

[38] T. Warren, „*Microsoft begins sharing Windows Phone 8.1 with developers*,“ 10 februar 2014. [Elektronski]. Available: http://www.theverge.com/2014/2/10/5399264/microsoft-begins-sharing-windows-phone-8-1-with-developers. [Poskus dostopa 14 december 2016].

[39] „*Windows Phone 8.1 Features*,“ [Elektronski]. Available: http://www.windowscentral.com/windows-phone-81-features. [Poskus dostopa 14 december 2016].

[40] J. Rogerson, „*January 21 may be the date Microsoft reveals Windows 10 for phones*,“ 13 januar 2015. [Elektronski]. Available: http://www.techradar.com/news/phone-

and-communications/mobile-phones/january-21-may-be-the-date-microsoft-reveals-windows-10-for-phones-1280196. [Poskus dostopa 14 december 2016].

[41] M. J. Foley, „*First Windows 10 mobile preview, due in February, key to Microsoft's OneCore vision*,“ 20 januar 2015. [Elektronski]. Available: http://www.zdnet.com/article/first-windows-10-mobile-preview-due-in-february-key-to-microsofts-onecore-vision/. [Poskus dostopa 14 december 2016].

[42] S. Basu, Real World Windows 8 Development, Apress: Apress, 2013.

[43] M. A., „*5 best non-Samsung smartphones with OLED displays,*“ 13 julij 2015. [Elektronski]. Available: http://www.phonearena.com/news/5-best-non-Samsung-smartphones-with-OLED-displays_id71449. [Poskus dostopa 14 december 2016].

[44] S. Dall, „*INTRODUCTION TO C# 0 C# is a multi-paradigm programming language which is based on objectoriented and component-oriented programming disciplines.*.“ [Elektronski]. Available: http://docplayer.net/21380085-Introduction-to-c-0-c-is-a-multi-paradigm-programming-language-which-is-based-on-objectoriented-and-component-oriented-programming-disciplines.html. [Poskus dostopa 15 december 2016].

[45] „*The A-Z of Programming Languages: C#*,“ 1 oktober 2008. [Elektronski]. Available: http://www.computerworld.com.au/article/261958/-z_programming_languages_c. [Poskus dostopa 15 december 2016].

[46] J. Avery, „*What Is Visual Studio*,“ 22 avgust 2005. [Elektronski]. Available: http://archive.oreilly.com/pub/a/windows/2005/08/22/whatisVisualStudio.html. [Poskus dostopa 16 december 2016].

[47] „*Welcome to Visual Studio 2015*,“ [Elektronski]. Available: https://msdn.microsoft.com/en-us/library/dd831853.aspx. [Poskus dostopa 16 december 2016].

[48] Egham, „*Gartner Says Worldwide Tablet Sales Grew 68 Percent in 2013, With Android Capturing 62 Percent of the Market,*“ 3 marec 2016. [Elektronski]. Available: http://www.gartner.com/newsroom/id/2674215. [Poskus dostopa 9 januar 2017].

[49] V. Staff, „*iOS: A visual history*,“ 16 september 2013. [Elektronski]. Available: http://www.theverge.com/2011/12/13/2612736/ios-history-iphone-ipad. [Poskus dostopa 9 januar 2017].

[50] „*The History of the C Language*,“ [Elektronski]. Available: https://www.codingunit.com/the-history-of-the-c-language. [Poskus dostopa 10 januar 2017].

[51] „*Programski jezik C*,“ [Elektronski]. Available: https://sl.wikipedia.org/wiki/Programski_jezik_C. [Poskus dostopa 10 januar 2017].

[52] „*The History of the C Language,*“ 7 januar 2017. [Elektronski]. Available: http://www.tiobe.com/tiobe-index//. [Poskus dostopa 10 januar 2017].

[53] „*Xcode*,“ [Elektronski]. Available: https://en.wikipedia.org/wiki/Xcode. [Poskus dostopa 11 januar 2017].

[54] V. Berce, „*Jabolko*,“ 23 december 2011. [Elektronski]. Available: http://www.jabolko.org/www.jabolko.org/snelkurs/468-uvod-v-xcode.html. [Poskus dostopa 11 januar 2017].

[55] „*Tools you'll love to use.*,“ [Elektronski]. Available: https://developer.apple.com/xcode/ide/. [Poskus dostopa 11 januar 2017].

[56] A. S. Inc, „*PhoneGap*,“ 2016. [Elektronski]. Available: https://phonegap.com/about/. [Poskus dostopa 17 november 2017].

[57] P. G. David Zakelšek, „*Pregled orodij za razvoj heterogenih*,“ VIRIZ, Maribor, 2014.

[58] R. Shannon, „*What is HTML?,*“ [Elektronski]. Available: http://www.yourhtmlsource.com/starthere/whatishtml.html. [Poskus dostopa 17 november 2017].

[59] J. Krynin, „*thoughtco*,“ 7 maj 2015. [Elektronski]. Available: https://www.thoughtco.com/what-is-css-3466390. [Poskus dostopa 17 november 2017].

[60] S. Chapman, „*Thoughtco*,“ 12 september 2017. [Elektronski]. Available: https://www.thoughtco.com/what-is-javascript-2037921. [Poskus dostopa 17 november 2017].

[61] Microsoft, „*Xamarin*,“ [Elektronski]. Available: https://developer.xamarin.com/guides/cross-platform/getting_started/introduction_to_mobile_development/. [Poskus dostopa 17 november 2017].

[62] Ionic, „*Ionic*,“ [Elektronski]. Available: https://ionicframework.com/. [Poskus dostopa 14 december 2017].