Факултет инжењерских наука Универзитета у Крагујевцу

Василије, М. Аврамовић

XAMARIN.ANDROID ПЛАТФОРМА

Израда Андроид апликација у C#

дипломски рад

Крагујевац, 2018.

Факултет инжењерских наука Универзитета у Крагујевцу

Назив студијског програма: Машинско инжењерство

Ниво студија: Дипломске студије по старом програму

Модул: Информатика у инжењерству

Предмет: Програмски језици

Број индекса: 9/99

Василије, М. Аврамовић

XAMARIN.ANDROID ПЛАТФОРМА

Израда Андроид апликација у C#

дипломски рад

Комисија за преглед и одбрану:

1. **Ред. проф. др Ненад Грујовић - ментор**

2. _____ Датум одбране:_____

3. _____ Оцена:_____

У оквиру овог дипломског рада кандидат треба да упозна платформу, изради апликације за оперативни систем Андроид у окружењу Visual Studio, у програмском језику C#, и прикаже предности и мане овог окружења.

Препоручена литература:

[1] Java JDK 7: kompletan priručnik, Mikro knjiga

[2] C# 7.1 i .NET Core 2.0: Moderno međuplatformsko programiranje, Mikro knjiga

[3] Android 4: Izrada aplikacija pomoću paketa Android SDK, Mikro knjiga

Крагујевац, 2018

ментор:

др Ненад Грујовић, редовни професор

Резиме рада:

Циљ овог пројекта је упознавање програмске платформе Xamarin као интегралног дела Visual Studio 2017 развојног окружења, посебно њене компоненте Xamarin.Android. Најпре ћемо припремити окружење за употребу нама потребних компоненти, а затим ћемо у овом окружењу, коришћењем C# програмског језика креирати неколико апликација, и објаснити корак по корак њихов развој и значење кода, упознајући уз пут и начин фунционисања Андроид система.

Кључне речи:

Visual Studio, Xamarin, Xamarin.Android, Android, IDE, C#, aplikacija, kod, layout, UI, korisnički interfejs.

Summary of work:

The aim of this project is to introduce the Xamarin software platform as an integral part of the Visual Studio 2017 development environment, in particular its Xamarin.Android component. First we will prepare the environment for using the necessary components, and then in this environment, using C # programming language, we will create several applications, and explain step by step their development and the meaning of the code, and by that process getting the knowledge of the way the Android system works.

Key words:
Visual Studio, Xamarin, Xamarin.Android, Android, IDE, C#, app, application, code, layout, UI, user interface.

Sadržaj:

1. Uvod

Microsoft Visual Studio (VS) je integrisano programsko okruženje (Integrated Development Environment IDE), kreirano od strane kompanije Microsoft. Visual Studio se koristi za programiranje računarskih programa (Metro UI, desktop), igara, veb-sajtova, veb-servisa i veb-aplikacija na Microsoft Windowsu, Androidu, iOSu itd.

Visual Studio podržava 36 različitih programskih jezika. Ugrađeni jezici uključuju C, C++ i C++/CLI (Visual C++), VB.NET (Visual Basic .NET), C# (Visual C#), F# i TypeScript. Podrška za druge programske jezike kao što su Python, Ruby, Node.js, i M dostupna je putem jezičkih usluga koje se instaliraju odvojeno. Visual Studio takođe podržava XML/XSLT, HTML/XHTML, JavaScript i CSS, Java (i J#).

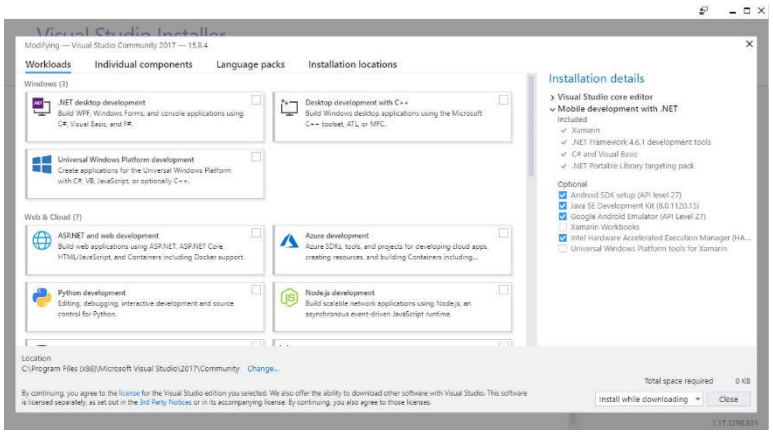
Prošlo je preko dvadeset godina od pojavljivanja programskog paketa Visual Studio 97, koji je predstavljao prvu verziju te moćne programerske alatke (brojna oznaka verzije bila je zapravo 5.0). To je bio prvi pokušaj da se pod jedan krov stave sve programske alatke koje je izdavao Microsoft i da se integracijom omogući unifikacija različitih tehnologija i tako poveća produktivnost. Istina, sve do pojave verzije Visual Studio paketa sa dodatkom .NET u imenu, 2002. godine (verzija 7.0), to su bila veoma različita radna okruženja, bez mnogo zajedničkih dodirnih tačaka.

1.1. Visual Studio 2017

Aktuelna verzija, Visual Studio 2017 (verzija 15.0), dolazi u tri varijante paketa: Community, Professional i Enterprise. Community verzija je u potpunosti besplatna, i namenjena je svima, kako u edukativne tako i u profesionalne svrhe, dok je druge dve komercijalne verzije takođe moguće besplatno preuzeti sa zvaničnog Microsoft sajta, i koristiti u probnom režimu mesec dana, posle čega se moraju platiti za dalji rad. Professional verzija je namenjena profesionalnim programerima, a verzija Enterprise članovima razvojnih timova u kompanijama sa preko 250 računara ili preko milion dolara godišnjeg prihoda.

Dok su ranije verzije Visual Studia nudile poprilično osakaćene besplatne varijante programa, situacija je od ove verzije znatno povoljnija. Sve najvažnije stvari se mogu obaviti i iz Community varijante, dok komercijalne verzije donose neka poboljšanja koja su značajna za korisnike kojima je programiranje u ovom paketu profesija. Glavna razlika između besplatne i profesionalne verzije je što ova druga ima podržan modul CodeLens (postoji još od verzije VS 2013), koji je vrlo udoban u procesu programiranja, ali posao se može obaviti i bez njega. Naravno, Enterprise verzija je najbogatija opcijama i nudi određeni deo alatki koje bi mogle da budu zanimljive programerima za mobilne uređaje, pošto podržava stvari kao što su udaljeni simulator za iOS uređaje ili dodatni instrumenti za platformu Xamarin. Osim toga, ova verzija tradicionalno ima najbogatiji skup funkcija za testiranje aplikacija.

Da bi nam pojednostavili izbor komponenti prilikom instalacije, programeri iz Microsofta su najčešće korišćene profile podelili po kategorijama koje se ovde nazivaju workloads. Glavne kategorije su Windows, Web & Cloud, Mobile & Gaming i Other Toolsets. Kategorija Windows sadrži tri stavke koje se odnose na razvoj UWP (Universal Windows Platform), C++ i .NET aplikacija. Kada izaberemo neku od njih, panel sa desne strane će prikazati šta je od ponuđenih alatki izabrano automatski, a šta je ostavljeno korisniku da izabere po želji.



Slične se stvari ponavljaju i kada je u pitanju kategorija razvoja veb i cloud aplikacija, gde su nam na raspolaganju alatke za razvoj ASP.NET koda, odlična podrška za popularni programski jezik Python, podrška za rad sa cloud sistemom Microsoft Azure i razvoj aplikacija uz pomoć frameworka Node.js. Tu su još tri kategorije koje nisu usko vezane veb razvoj: Data storage and processing, Data science and analytical applications i Office/SharePoint development.

Razvojne alatke za mobilne uređaje i video-igre obuhvataju mnoštvo .NET baziranih tehnologija za kreiranje aplikacija za iOS, Android i Windows na Xamarin platformi, podršku za razvoj Java i JavaScript aplikacija, kreiranje igara putem programskog jezika C++, podršku za Unity framework i razvoj mobilnih aplikacija uz pomoć jezika C++.

Poslednja workloads kategorija sadrži alatke za razvoj Visual Studio ekstenzija, Linux aplikacija na jeziku C++, kao i za razvoj multiplatformskih aplikacija na bazi .NET Core tehnologije. Alternativno je moguće vršiti izbor potrebnih komponenti putem menija Individual Components, gde su taksativno navedene najvažnije podržane funkcije.

1.2. Xamarin

Nakon mnogo godina razvoja, još od 2000. godine, platforma Mono, koja nudi podršku za .NET tehnologiju na sistemima čiji proizvođač nije Microsoft, stasala je u moćnu alatku kojom se koriste stotine hiljada programera, a koja je poznata pod nazivom Xamarin od 2011 godine, kada su inženjeri koji stoje iza Monoa osnovali kompaniju pod nazivom Xamarin. Od 2016. godine, kompanija je akvizicijom postala deo Microsofta.

Xamarin pruža mogućnost efektivnog i jednostavnog programiranja za tri najpopularnije mobilne platforme: Android, iOS i Windows. Pri tome su sve izvorne API funkcije tih operativnih sistema dostupne pozivanju preko Xamarina, pa je brzina izvršavanja koda obično na visokom nivou. Ne treba govoriti kolike su prednosti jednog

takvog okruženja za proizvođače softvera koji istovremeno ciljaju na više različitih platformi.

Xamarin za kodiranje koristi C# programski jezik (koji je veoma rasprostranjen i bogat bibliotekama, a koje se ovde mogu koristiti), i pored programiranja unikatnog korisničkog interfejsa (User Interface, UI) za svaku platformu, omogućava da se osnovna logika u kodu koristi nepromenjena u svim podržanim OS, što obično čini 70 do 85% koda.

Inače, Xamarin se na engleskom najčešće čita „zamarin“, a na somalskom jeziku bi ta reč značila čekić, što asocira na „kovanje“ koda u odgovarajući kalup.

1.3. Praktična iskustva sa resursima potrebnim za rad VS i Xamarin-a

Svi koji su koristili Visual Studio (VS) znaju da je reč o moćnom programerskom sredstvu, ali je to isto tako i dosta sporo radno okruženje kome su u slučaju iole kompleksnijih projekata potrebne desetine sekundi pa i minuti da bi učitao sve fajlove ili da bi izveo njihovo kompajliranje. Iako je jasno da je Majkrosoft uvođenjem brojnih optimizacija ubrzao aktuelnu verziju (po testovima, podizanje programa je čak tri puta brže u odnosu na VS2015), i dalje nije sve tako idealno, i treba se naoružati strpljenjem.

Jedan od „kočničara” je najčešće i antivirus program, pa je korisno izuzeti folder izvršnog fajla Devenv.exe iz procesa proveravanja virusa. Sledeća faza ubrzanja može biti u pozivanju menija **Manage Visual Studio Performance**, gde je moguće isključiti više dodataka koji usporavaju podizanje programa, ukoliko ih ima. Na taj način se postiže smanjenje vremena podizanja programa sa desetina sekundi na svega nekoliko sekundi. Ubrzanje učitavanja projekata je postignuto delimičnim učitavanjem, samo najvažnijih fajlova, dok se ostatak učitava po potrebi, ali opet kada je potrebno prikazati sve resurse, učitavanje ume dodatno da potraje.

Na brzinu u radu sa Visual Studiom veliki povoljan uticaj ima korišćenje solid state diskova (SSD) i za ozbiljan rad sa programom, ovi mediji su od izuzetne koristi.

Isto tako, VS je i veoma zahtevan i kada su u pitanju memorijski resursi. Pre svega prostor na drajvu za instalaciju je kritičan. Prilikom instalacije, ili modifikacije VS-a, biramo samo one komponente koje planiramo da koristimo, pa opet, njihove veličine se kreću od nekoliko gigabajta (GB) do nekoliko desetina GB, i to samo za osnovne postavke komponenti, dok čekiranje svake dodatne podiže instalaciju za prostor čiji se red veličine meri u gigabajtima.

Primera radi, instalacija samo osnovne postavke Xamarin komonente, koja se postiže čekiranjem **Mobile development with .NET** workloada u Visual Studio Installeru bez ikakvih dodataka, zauzima oko 15-16 GB. Uz neophodno apdejtovanje, i eventualnu instalaciju svega par dodatnih paketa za rad sa starijim android platformama (u osnovnom paketu je podrška samo za android SDK 8.1) i emulatora za prikaz i testiranje naših projekata, instalacija VS-a je već zauzela 25-30 GB.

Dodavanjem još komponenti vezanih za Xamarin, i bez ostalih workload-ova VS-a, instalacija VS-a može preći i stotinak gigabajta prostora na hard disku, pa o tome treba povesti računa.

Obzirom da se svaka instalacija obavlja online, direktnim download-om s interneta, veličina fajlova koje je potrebno skinuti uslovljava da proces instalacije dugo traje, zavisno od brzine internet konekcije, čak i satima.

Treba obratiti pažnju, da čak i korisnički projekti, bez obzira na relativnu jednostavnost aplikacije od par desetina linija koda (tipa „Hello World“), prilikom kompajliranja povezuju komponente neophodne za izvršavanje pa folderi sa projektom budu po nekoliko stotina megabajta. Bez dalje optimizacije, takva aplikacija i na android uređaju zauzima par desetina megabajta.

Što se tiče RAM memorije, poboljšan je i rad sa memorijom, u odnosu na starije verzije VS-a, tako da se mogu učitati i veoma obimni projekti bez rušenja radnog okruženja. Međutim, bez obzira što je zvanično dovoljna količina od 4 GB rama za rad VS-a, Xamarin i te kako ume da premaši tu količinu, pogotovu pri pokretanju emulatora koji sami zauzmu značajno preko gigabajta, tako da je neki minimum za iole udoban rad u ovom trenutku bar 8 GB RAM-a.

Kada je procesorska moć u pitanju, zvanično su podržani i stariji procesori oba proizvođača, VS će raditi i na desetak godinama starim CPU-ovima, međutim, opet se praksa malo razlikuje. Korisno je imati novije brže procesore sa što više jezgara zarad bržeg kompajliranja, koje ume da potraje i minutima, ali je za korišćenje android emulatora praktično neophodno da procesori budu Intelovi, jer nije podržana virtuelizacija za AMD mašine, tako da se emulacija na njima izvršava mnogostruko sporije nego na Intelovim procesorima. Pravilan izbor emuliranih uređaja na AMD-ovom CPU se svodi na emulaciju ARM arhitekture, dok se (potpuno nelogično) x86 arhitektura vezuje samo za Intelove procesore i njihov pokušaj se najčešće završava neuspehom.

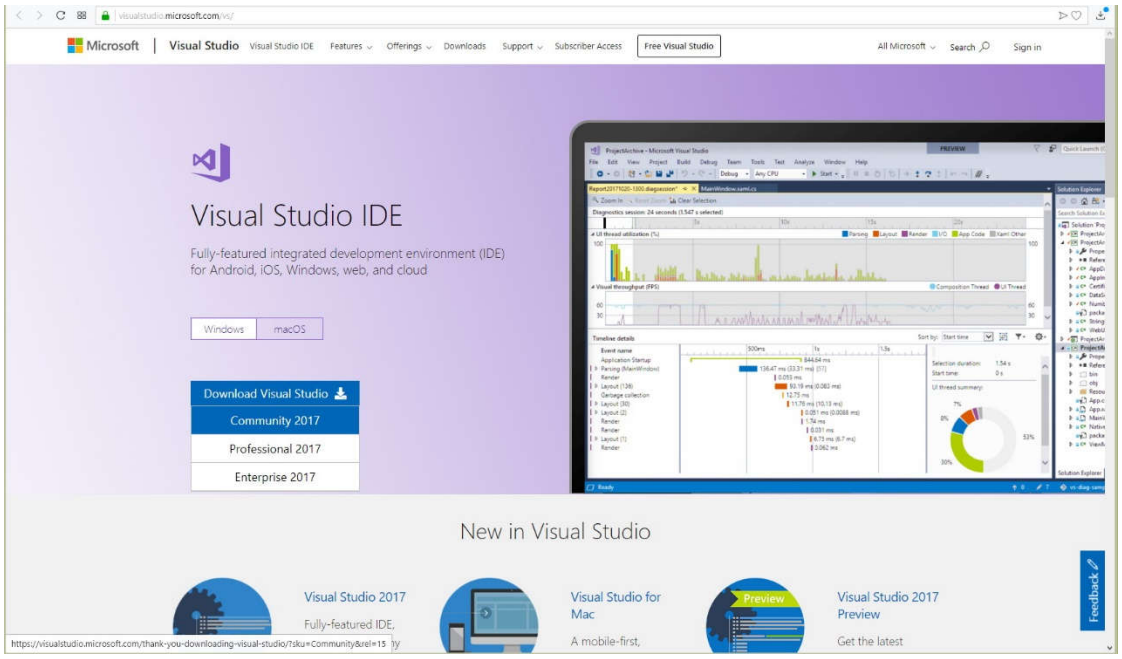
Sličan problem se javlja i kod izbora Windows okruženja za rad. Dok VS 2017 zvanično radi na svim Windows-ima od sedmice ka novijima, Xamarin može praviti različite probleme koji gotovo onemogućuju rad. Preporuka je, a gotovo i uslov, koristiti Windows 10 i to sa obavezno instaliranim poslednjim „velikim“ update-om (Creators Update April 2018 u ovom trenutku), pošto će u suprotnom dizajner korisničkog interfejsa praviti velike probleme, od preklapanja elemenata, do potpunog odbijanja prikaza.

Bez ispunjenja svih ovih uslova koji se ne navode zvanično, rad sa Xamarinom veoma lako postaje noćna mora.

Na kraju, ako je potrebna deinstalacija VS-a, ona se može obaviti iz njegovog instalera, ali treba primetiti da on ne uklanja i sve komponente i update-ove koje je naknadno dodao, tako da je u našem slučaju, od pedesetak gigabajta komponenti koje je VS instalirao, on je uklonio čak manje od polovine svojih fajlova, a ostatak smo ručno pretraživali i uklanjali iz desetina foldera raštrkanih po sistemskom disku, u kategorijama *Program Files* i *Users*, i opet je preteklo par dobro skrivenih gigabajta, koji će ostati na drajvu kao beskoristan zaostatak.

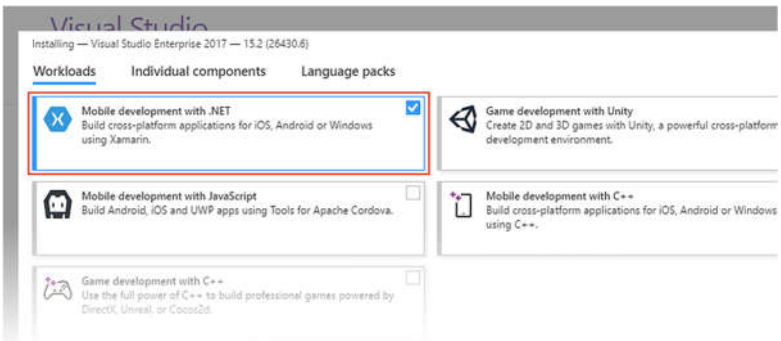
2. Instalacija i podešavanja

Preuzimanje instalacije Visual Studia se može obaviti na zvaničnom sajtu Microsofta (visualstudio.microsoft.com/vs/) gde na stranici Free Visual Studio nalazimo taster Download Visual Studio, koji nam pod sobom omogućava izbor jedne od tri varijante ovog programa. Biramo Community 2017, i nakon klika čekamo kratko da se Visual Studio Installer download-uje s interneta. Microsoft nam nudi da se registrujemo na njihovom sajtu, ali to nije neophodno.



2.1. Instalacija Xamarin komponente Visual Studia

Nakon završetka preuzimanja instalacionog fajla, nalazimo lokaciju gde je fajl skinut i pokrećemo ga, duplim klikom na *vs_community__xxxxxx.xxxx.exe*. Time smo pokrenuli program za instalaciju Visual Studia. Još jednom, u okviru instalera, biramo varijantu koju želimo da instaliramo, i obzirom da želimo besplatnu varijantu, u okviru sekcije Visual Studio Community biramo dugme *Install*. Ukoliko već imamo instalirane komponente Visual Studia, umesto dugmeta *Install*, koristimo dugme *Modify*. Nakon kratkog učitavanja, instaler nam nudi dostupne workloads-ove.



Nama je za korišćenje Xamarina potreban samo *Mobile development with .NET workload*, pa čekiramo njega. Sa desne strane prozora, prikazane su komponente koje će se instalirati, i nudi nam se još opcija, koje nam u ovom

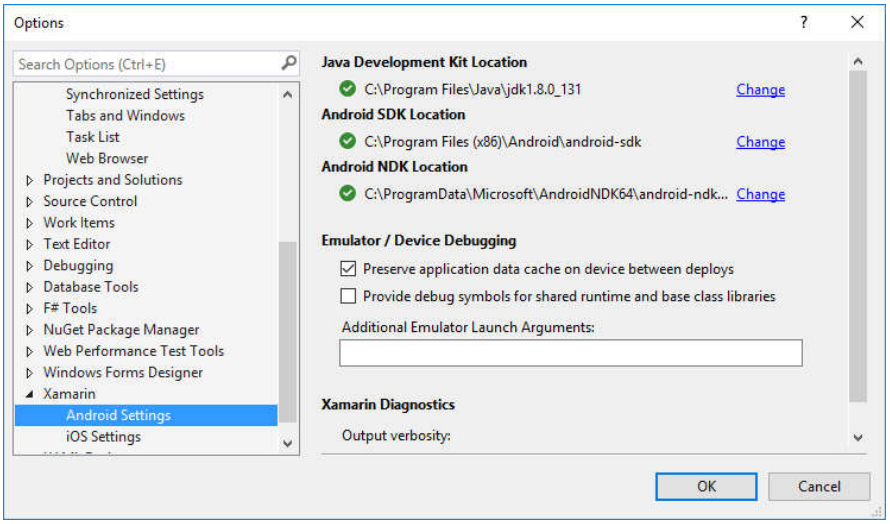
trenutku nisu neophodne, pa ostavljamo sve kako je već podešeno. Ukoliko postoji potreba, u tabu *Individual components* možemo birati još veliki broj pojedinačnih komponenti koje su navedene taksativno, ali nam za sada ništa od toga nije neophodno. Iznad dugmeta *Install* nam se prikazuje veličina potrebnog prostora za instalaciju izabranih komponenti. Pritiskom na dugme *Install* pokrećemo instalaciju, i izabrane komponente počinju da se download-uju s interneta i instaliraju. Na progres baru možemo pratiti napredak procesa instalacije, a i ne moramo, obzirom da instalacija može potrajati i preko sat vremena, a ne zahteva dalje intervencije korisnika.

Po završetku instalacije, program nam opet nudi mogućnost da kreiramo korisnički nalog na Mikrosoftovom sajtu kako bi smo mogli da koristimo Azure i druge napredne servise kao i sinhronizacije podataka, ali nam sve to i nije neophodno u ovom trenutku, pa možemo jednostavno pritisnuti *Not now, maybe later* i pokrenuti Visual Studio.

2.2. Podešavanje Xamarin-a

Pri prvom pokretanju Visual Studia, nudi nam se mogućnost prilagođavanja okruženja, zavisno od toga čime prevashodno želimo da se bavimo, i koje boje pozadina želimo da gledamo, ali sve to možemo menjati i kasnije pa možemo ostaviti i ponuđena podešavanja.

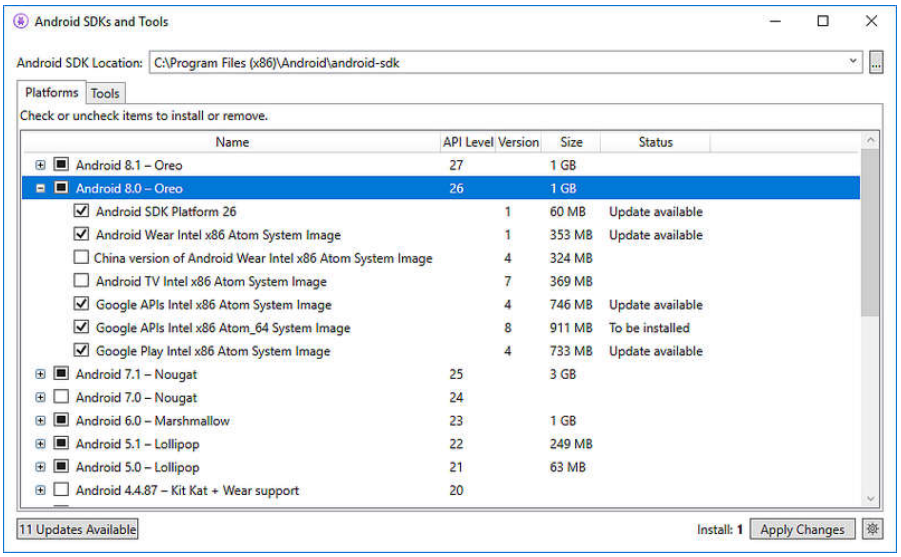
Nakon podizanja programa, bavimo se opcijama koje nam jesu neophodne za rad sa Xamarinom. Za rad u ovom okruženju su nam neophodni Java Development Kit (JDK) i Android SDK za pravljenje aplikacija, i ako smo instalirali Xamarin na prethodno opisan način, Visual Studio je to odradio za nas i podesio lokacije. Ako nismo, moramo ručno postaviti putanje na odgovarajuće fajlove, što možemo proveriti i podesiti u opcijama koje nalazimo kliktanjem na **Tools > Options > Xamarin > Android Settings**.



Ukoliko su putanje za sve tri komponente (čak i ako poslednja putanja stoji prazna) štiklirane u zelenom krugu, podešavanja su dobra.

Android koristi različite Android API verzije, da odredi kompatibilnost aplikacije sa različitim verzijama Android operativnog sistema. Zavisno od toga na kojim verzijama Androida želimo da se naša aplikacija izvršava, verovatno ćemo morati da instaliramo dodatne Android SDK komponente, a možda i dodatne komponente

emulatora i alate. Za to koristimo *Android SDK Manager*, koji nalazimo na putanji **Tools > Android > Android SDK Manager**.



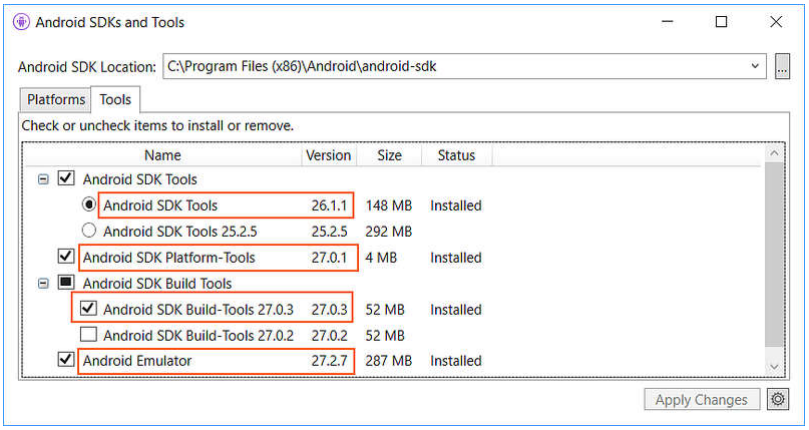
Na početku su instalirani samo komponente za posljednju verziju Androida (Android 8.1 u ovom trenutku) i to ne sve moguće. Po potrebi ćemo u ovom prozoru često dodavati nove komponente, kako platformi tako i dodatnih alata u tabu *Tools*, čekiranjem željenih komponenti, kao i raditi update-ovanje postojećih komponenti na dugme *Updates Available*. Potvrda svih instalacija se vrši na dugme *Apply Changes*. Za veliki broj instalacija je potrebno prihvatiti uslove licence. Svaka instalacija ide direktno sa interneta i, obzirom da su paketi reda veličine više stotina megabajta do preko gigabajta, može da potraje.

2.3. Podešavanje Android emulatora

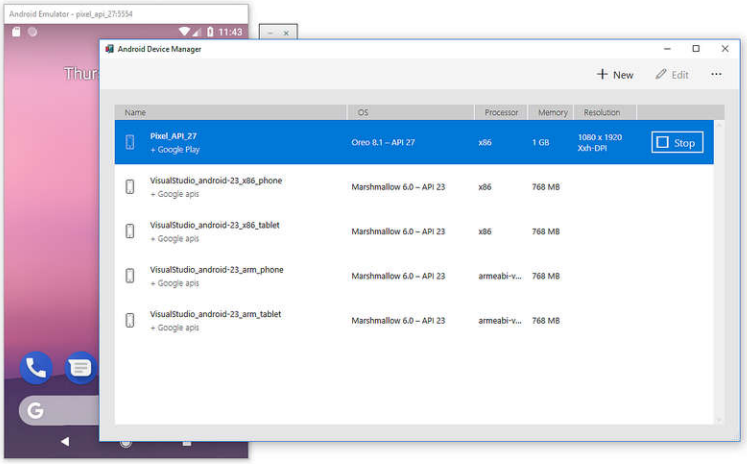
Android emulator može raditi u širokom rasponu konfiguracija da bi simulirao različite uređaje. Svaka sačuvana konfiguracija predstavlja jedan virtuelni uređaj. Kada pokrenemo test naše aplikacije na emulatoru, biramo unapred pripremljen Virtualni uređaj koji simulira rad stvarnog fizičkog Android uređaja kao što su Nexus ili Pixel telefoni.

Za rad sa emulatorom, na VS moraju biti instalirani poslednji update-ovi kako VS-a tako i Xamarina, kao i Android SDK. Ako smo instalaciju odradili na do sad opisan način, ti uslovi su zadovoljeni. Ono što nam još treba, Instaliraćemo korišćenjem Android SDK menadžera, u Tools tabu:

- **Android SDK Tools version 26.1.1** ili noviji
- **Android SDK Platform-Tools 27.0.1** ili noviji
- **Android SDK Build-Tools 27.0.3** ili noviji
- **Android Emulator 27.2.7** ili noviji

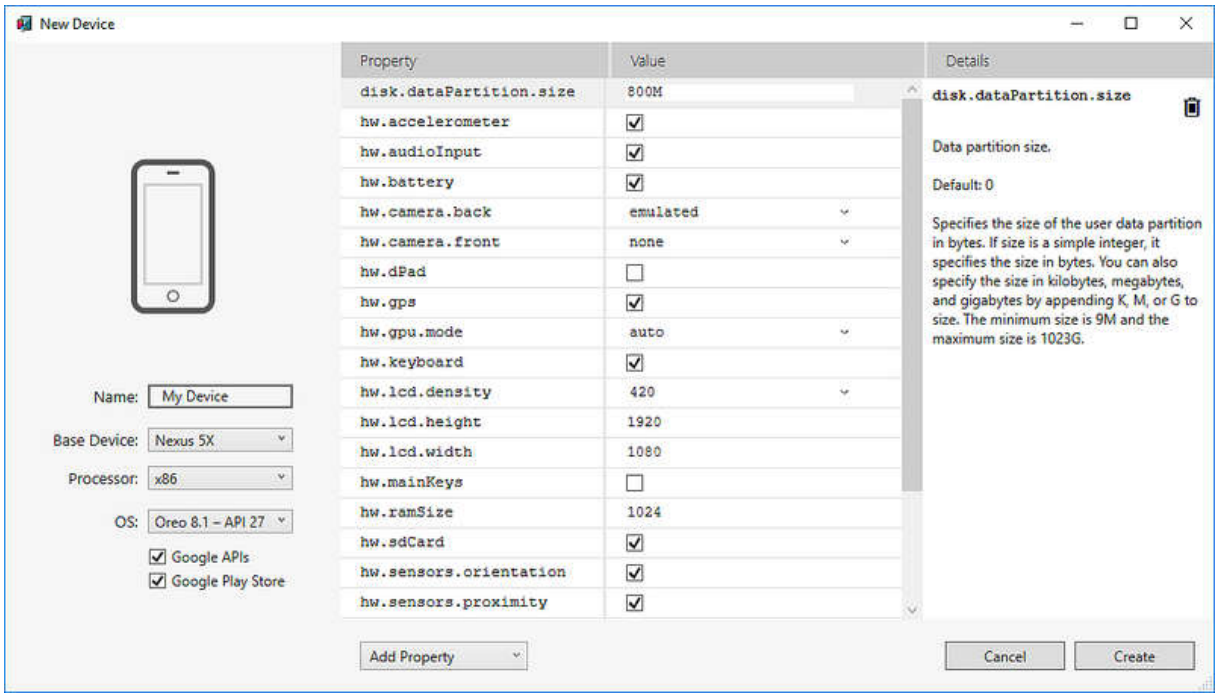


Android Device Manager pokrećemo iz menija **Tools** kliktanjem na **Tools > Android > Android Device Manager**:



U prozoru su prikazani trenutno konfigurisani uređaji, njihovo ime, verzija androida, procesor, veličina memorije, kao i rezolucija. Pritiskom na dugme *Start* pored konfiguracije pokrećemo emulator, a na dugme *Stop* koje se pojavljuje nakon toga ga zaustavljamo. Na dugme *New* pravimo novi virtuelni uređaj, na *Edit* menjamo konfiguraciju postojećeg.

Kreiraćemo novi Android Virtual Device (AVD), virtuelni uređaj, na taster *New*. Opcija je zaista puno a objasnićemo samo neke bitnije.



- *Name* je polje za unos imena našeg uređaja.

- *Base Device* služi za izbor uređaja od ponuđenih koji želimo da emuliramo, koji se razlikuju po osnovnim hardverskim postavkama.

- *Processor* predstavlja izbor fizičkog procesora čiji rad simuliramo. Teoretski, najbrže će raditi emulacije x86 procesora, ili x86_64 ako simuliramo 64-bitni procesor. Za njih je moguće uključiti hardversku akceleraciju, i u vezi konfigurisanja mogućih akceleracija postoji opširna tema o podešavanjima kako VS-a, tako i Windows Virtualizacije, koja je obrađena na zvaničnom sajtu https://docs.microsoft.com/en-us/xamarin/android/get-started/installation/android-emulator/hardware-

acceleration?pivots=windows i najčešćim problemima https://docs.microsoft.com/en-us/xamarin/android/get-started/installation/android-emulator/troubleshooting?pivots=windows ali i brojnim forumima i stranicama posvećenim tome. Međutim, ukoliko ne radite na PC-ju baziranom na novijem Intel procesoru, lako se može desiti da uprkos svim ovim podešavanjima emulator i dalje ne radi kako treba, pa je često sigurniji izbor emulacija ARM procesora, ali je izvršavanje daleko sporije.

- *OS* nudi izbor android verzija, postavka je na je najnoviji, ali nam nekada treba i emulacija rada na starijim, zašta je opet potreban download starijih Android SDK.

- *Google APIs* i *Google Play Store* nam nude simulaciju rada Guglovih servisa, i dostupni su samo za neke *Base Device*.

Ostala polja u središnjem delu prozora predstavljaju propertije i nude nam detaljnija podešavanja hardvera i ponašanja našeg uređaja, i nećemo se sada baviti njima pojedinačno. Klikom na *Create* dugme kreiramo naš virtualni uređaj, posle čega nam opet treba potvrda saglasnosti sa uslovima licence ukoliko se instaliraju nove komponente, i posle nekog vremena potrebnog za instalaciju, naš novi uređaj je u listi uređaja i spreman za upotrebu.

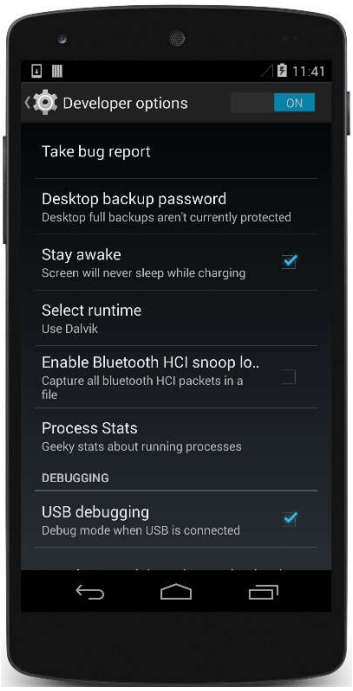
Sa selektovanim uređajem u listi, možemo kliknuti na *Edit* dugme, ako želimo da mu promenimo neka svojstva. Time se otvara identičan prozor kao u prethodnom slučaju i važe ista pravila. Primetimo još u gornjem desnom uglu dugme sa tri tačke, koje otvara dodatne opcije. Korisna nam je *Duplicate and Edit* ako želimo da kreiramo sličan uređaj sa manjim izmenama opcija, i *Delete* ukoliko želimo da uklonimo neki uređaj. Reveal in Explorer otvara u eksploreru lokaciju gde su fajlovi konfiguracije uređaja, dok *Factory Reset* resetuje stanje uređaja na početno, u trenutku puštanja u rad, ne menjajući nikakva podešavanja postavljena pri kreiranju ili editovanju uređaja.

2.4. Povezivanje fizičkog Android uređaja

Ukoliko nam je rad virtualnog Android uređaja suviše spor, ili nepouzdan zbog pucanja programa, ili jednostavno želimo da testiramo naš program na pravom Android uređaju, moguće je povezati fizički Android uređaj sa Visual Studiom i pokrenuti izvršavanje programa na njemu.

Moguće je teoretski povezati bilo koji Android uređaj za potrebe testiranja, ali pre nego što povežemo USB kablom Android uređaj i PC, moramo obaviti neka pripremna konfigurisanja da bi veza Visual Studia i Androida proradila.

Prvi korak je omogućavanje *USB Debugging* opcije na Android uređaju. Ta opcija se nalazi u *Settings*-u uređaja, u *Developer options* meniju. Međutim, kod novijih Android operativnih sistema, počev od verzije Android 4.2, taj meni je fabrički sakriven. Da bi ga otkrili, potrebno je da odemo na lokaciju **Settings > About phone**, i kliknemo *Build number* stavku 7 puta kako bi otkrili *Developer Options* tab.

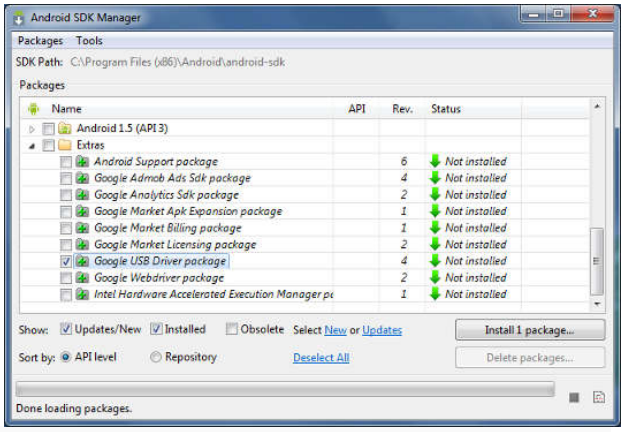


Nakon 7 kliktanja na *Build number*, *Developer Options* nam postaje dostupan i sada odlazimo na lokaciju **Settings > System > Developer Options** gde čekiramo opciju *USB Debugging.*

Sledeći neophodan korak je instalacija USB drajvera za Android uređaj, i on se može razlikovati za različite operativne sisteme i Android uređaje različitih proizvođača, pa ćemo ovde navesti postupak okvirno, za Google Nexus uređaje.

Pokrećemo **android.bat** aplikaciju koja se nalazi u **[Android SDK install path]\tools** direktorijumu. Po default-u Xamarin instalira Android SDK na sledeću lokaciju: `C:\Users\[username]\AppData\Local\Android\android-sdk`

Instalaciju *Google USB Driver* paketa možemo obaviti iz *Android SDK Manager*-a, koji nalazimo ekspanzijom *Extras* foldera, kao na slici.



Drajveri su download-ovani na lokaciju `[Android SDK install path]\extras\google\usb\_driver`

Default lokacija Xamarin.Android instalacije je:

`C:\Users\[username]\AppData\Local\Android\android-sdk\extras\google\usb_driver`

Nakon download-a USB Driver-a, potrebno ih je instalirati. Na Windows 7 taj proces bi izgledao ovako:

1. Konektujemo Android uređaj na kompjuter USB kablom.
2. Desni klik na ikonicu Computer na desktopu ili u Windows Exploreru i biramo opciju *Manage.*
3. Biramo *Devices* na levoj strani.
4. Nalazimo i raširimo *Other Devices* na desnoj strani.
5. Desni klik na ime uređaja i biramo *Update Driver Software* čime se startuje Hardware Update Wizard.
6. Biramo *Browse my computer for driver software* i klikćemo na *Next*

7. Klikćemo dugme *Browse* i nalazimo lokaciju USB drajvera (u ovom slučaju **[Android SDK install path]\extras\google\usb_driver**)
8. Kliknemo dugme *Next* i sačekamo da se završi instalacija.

Za druge uređaje USB drajvere treba tražiti na sajtu proizvođača uređaja.

Međutim, iako zvanično jesu dovoljni samo USB drajveri, veoma često u praksi ni sa njima Visual Studio ne vidi naš uređaj, iako ga sam Windows uredno prepoznaje i radi sa njim. Ponekad je potrebno instalirati i dodatni paket proizvođača za rad s uređajem iz Windows-a. U našem konkretnom slučaju, za LG telefon, bilo je potrebno instalirati LG PC Suite, i update-ovati ga na poslednju verziju kako bi konekcija sa Visual Studiom proradila.

Podrazumeva se da smo i Visual Studio update-ovali.

Uređaj je moguće povezati i preko WiFi konekcije, a uputstvo možete naći na adresi: https://docs.microsoft.com/en-us/xamarin/android/get-started/installation/set-up-device-for-development

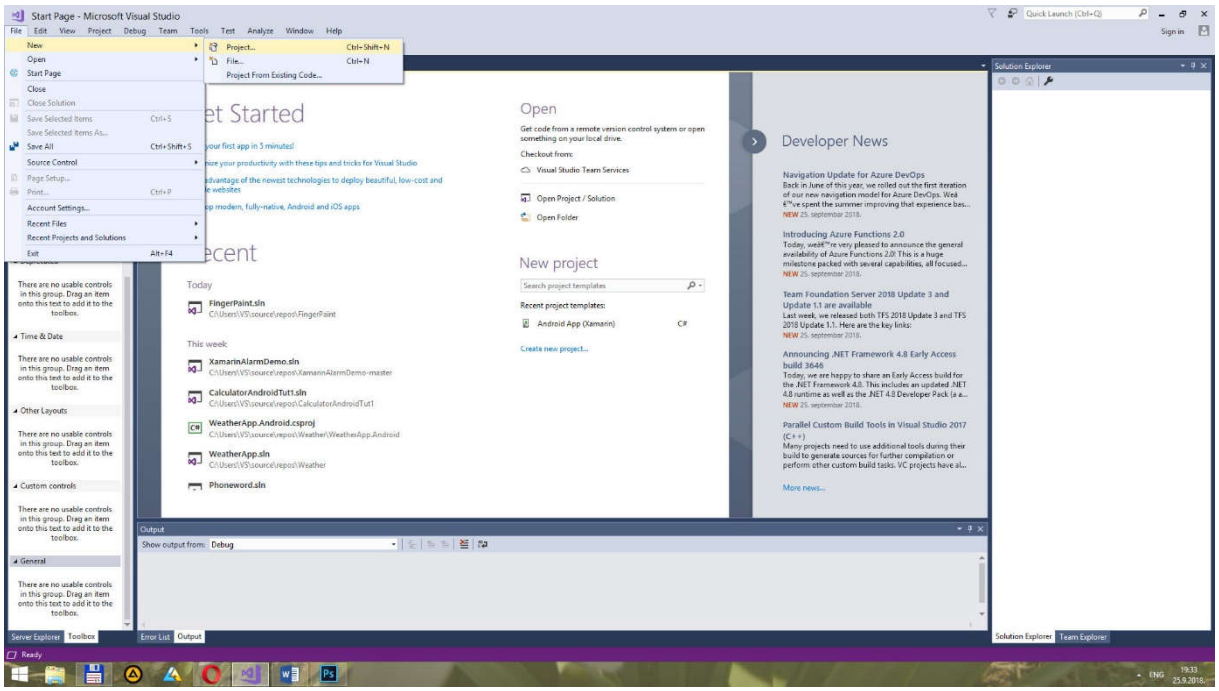
Ako su prethodni koraci korektno završeni, konektujemo Android uređaj, i u VS-u pri pokretanju izvršavanja aplikacije, umesto emulatora biramo u padajućem meniju naš Android uređaj. Naravno, da bi uopšte naš kod mogao da se izvršava na tom uređaju, u projektu mora biti definisano da minimalna verzija androida bude jednaka ili manja od verzije na našem uređaju, ali o tome više reči kasnije.

3. Naša prva aplikacija

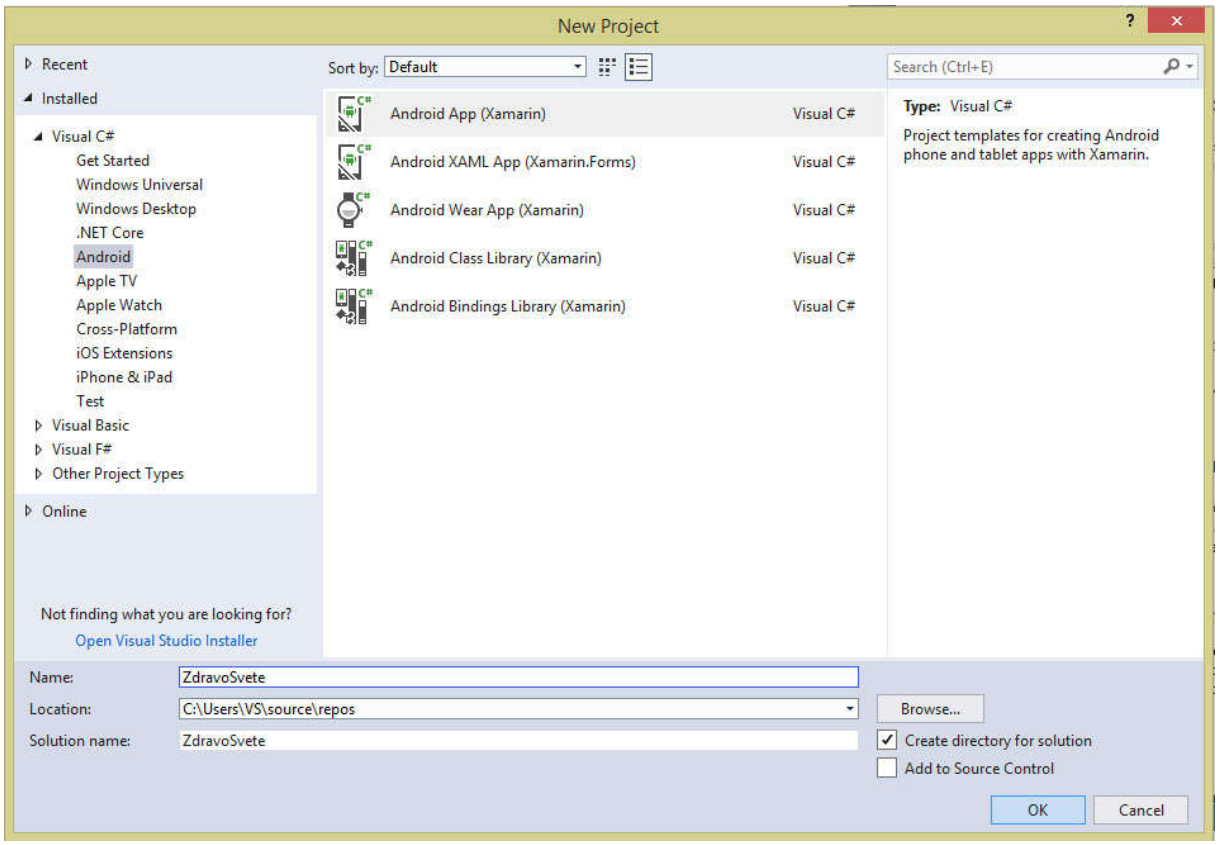
Tradicionalno prva aplikacija koju pravimo u svakom novom okruženju i jeziku, jeste **HelloWorld,** pa da ne preskačemo tradiciju, napravićemo je i ovde. Nećemo je previše objašnjavati u ovom trenutku, samo ćemo demonstrirati njen rad. Malo ćemo je uz put lokalizovati na naš govorni jezik.

3.1. Pokretanje HelloWorld projekta

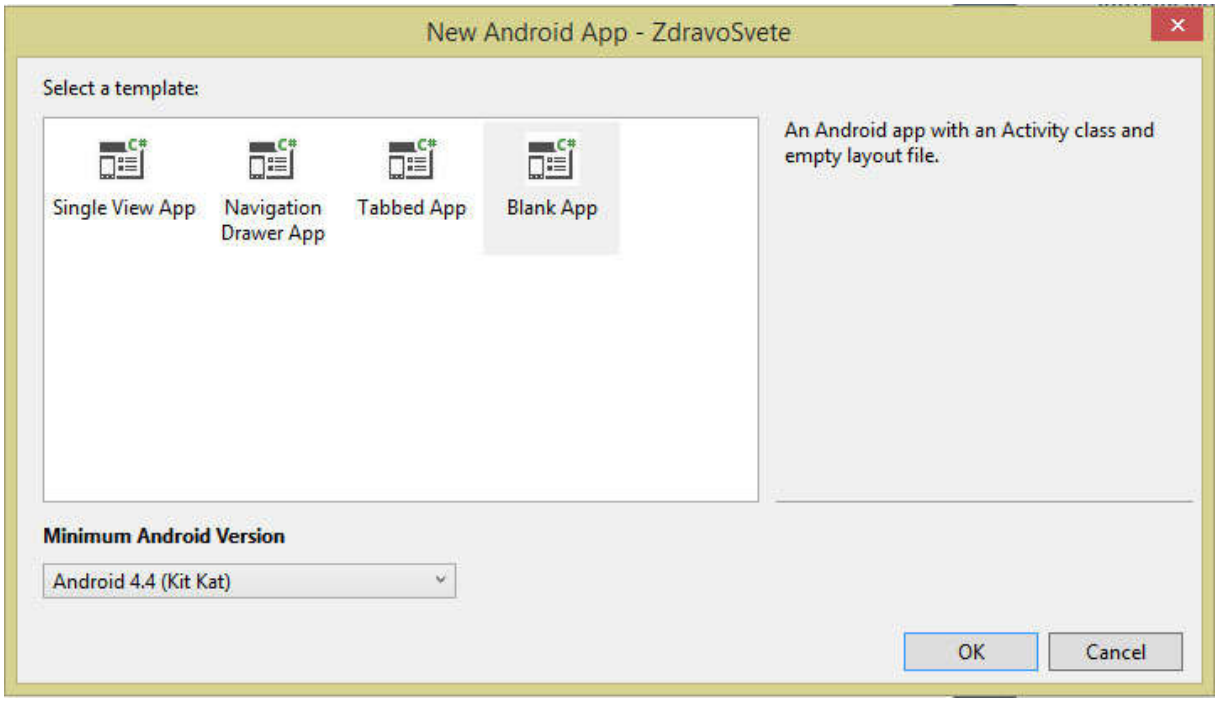
Startujemo Visual Studio. Nov projekat započinjemo kliktanjem na **File > New > Project**.



U **New Project** dijalogu, biramo **Android App** šablon (template). Dajemo ime projektu *ZdravoSvete* pa pritiskamo **OK**.



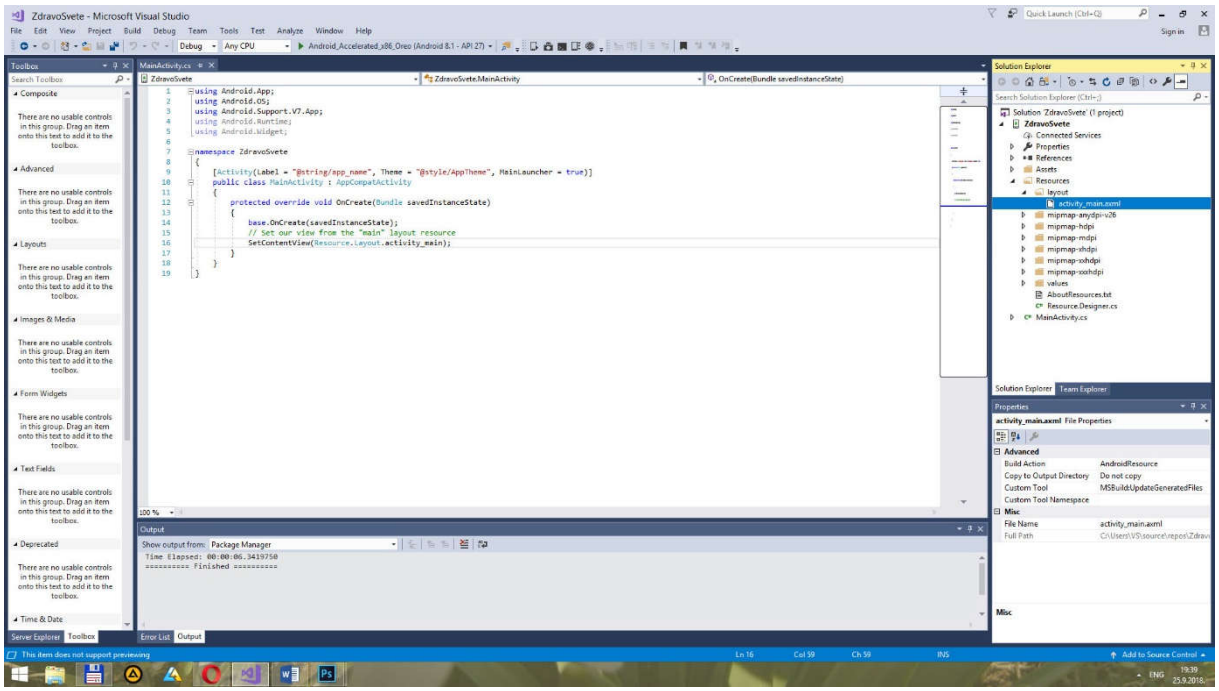
U **New Android App** dijalogu, biramo **Blank App**. Obratimo pažnju na padajući meni **Minimum Android Version**. Ako planiramo da aplikaciju pokrećemo na svom Android uređaju (telefon, tablet..) onda ovde moramo izabrati verziju operativnog sistema koja je na našem uređaju ili stariju od te.



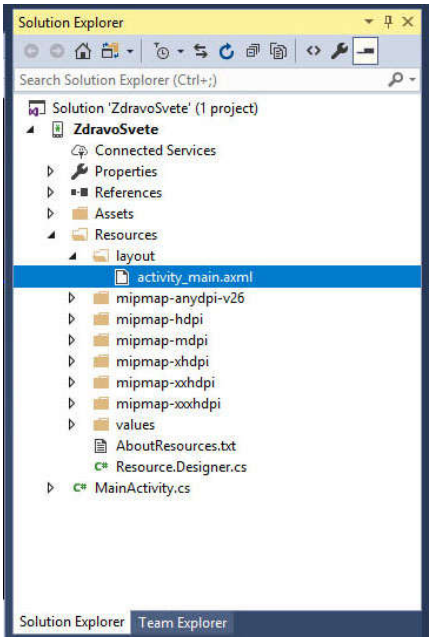
Na kraju pritiskamo **OK** da bi startovali projekat.

3.2. Kreiranje vizuelnog dela aplikacije

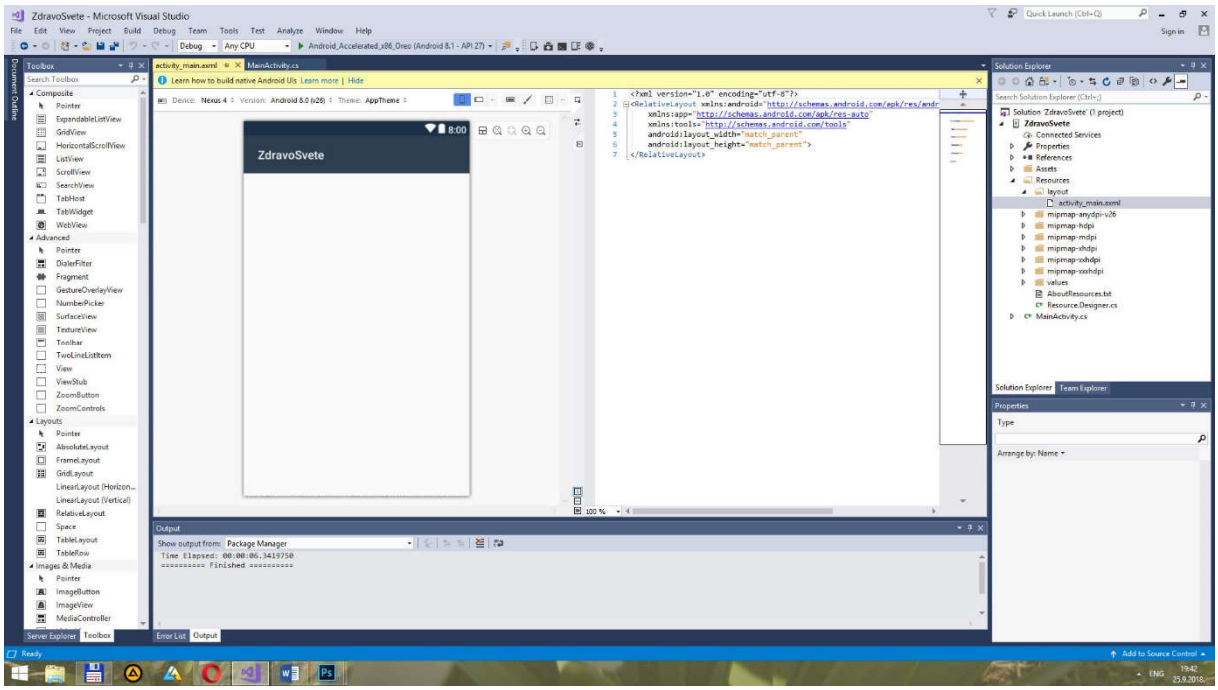
Nakon što je projekat kreiran, otvara se Visual Studio IDE, i otvara nam kod, fajl **MainActivity.cs,** kao na slici.



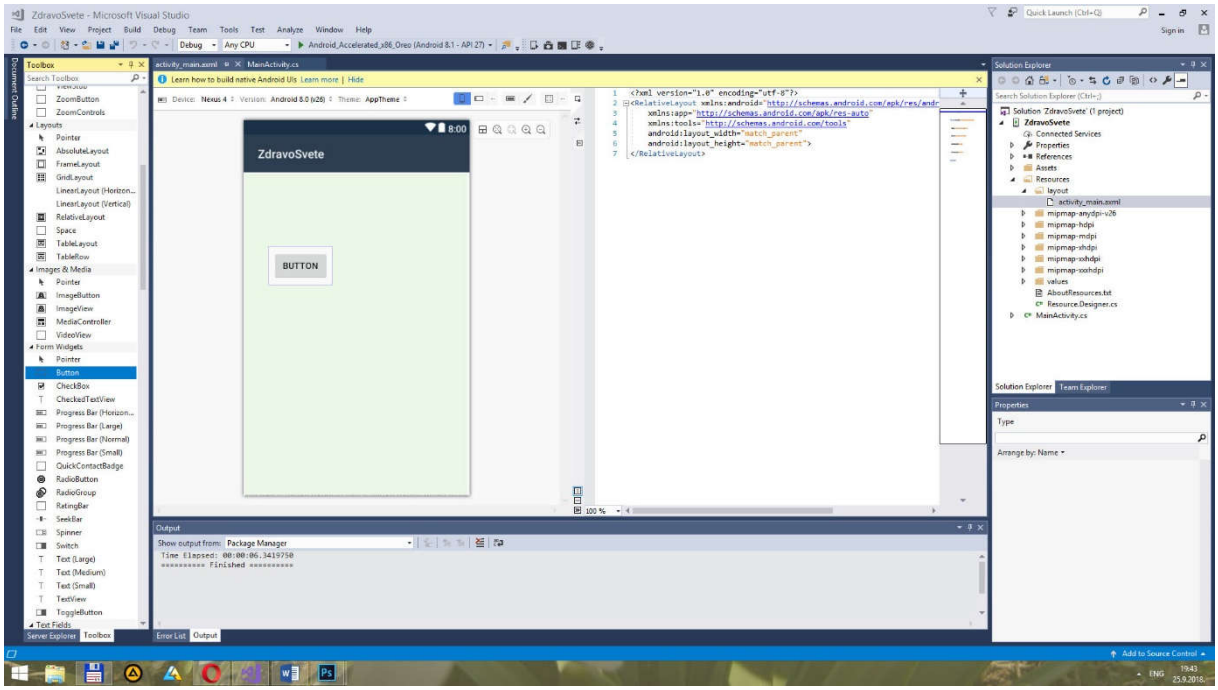
Obratimo pažnju na **Solution Explorer** prozor sa desne strane. Proširimo najpre **Resources** folder duplim klikom na njega, ili jednim klikom na strelicu ispred, a zatim i **layout** folder unutar njega.



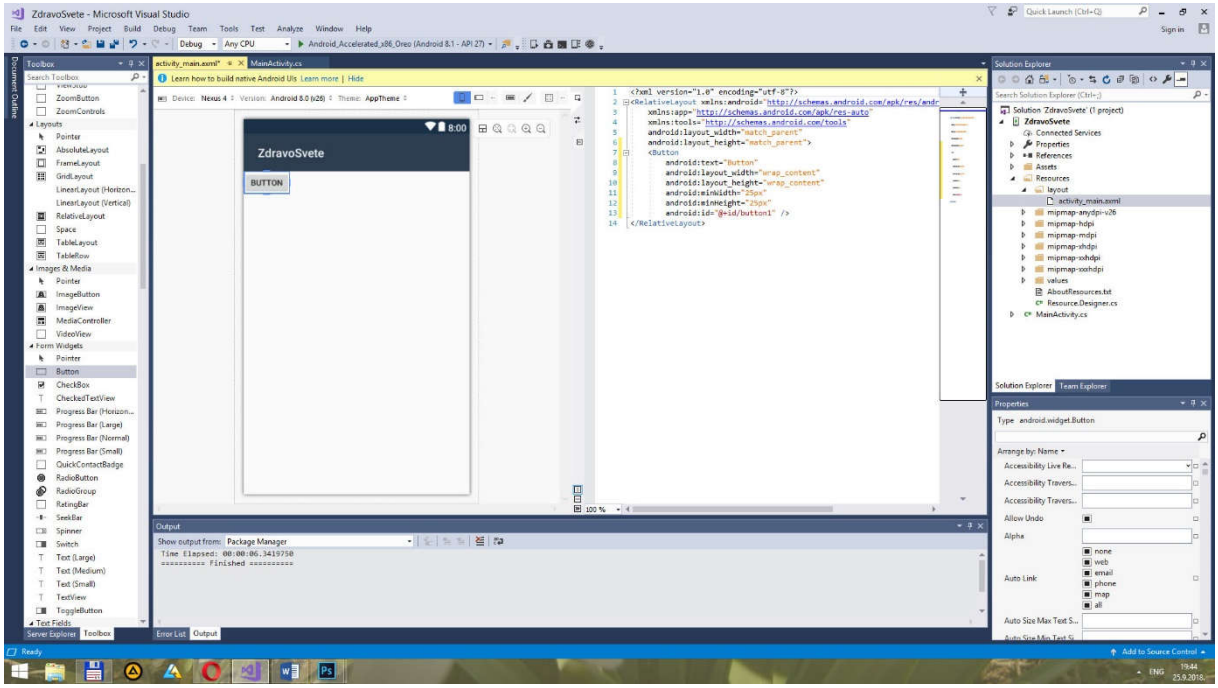
Duplim klikom na **activity_main.axml** otvaramo taj fajl u Android Designer-u. Tu možemo da dizajniramo izgled naše aplikacije.



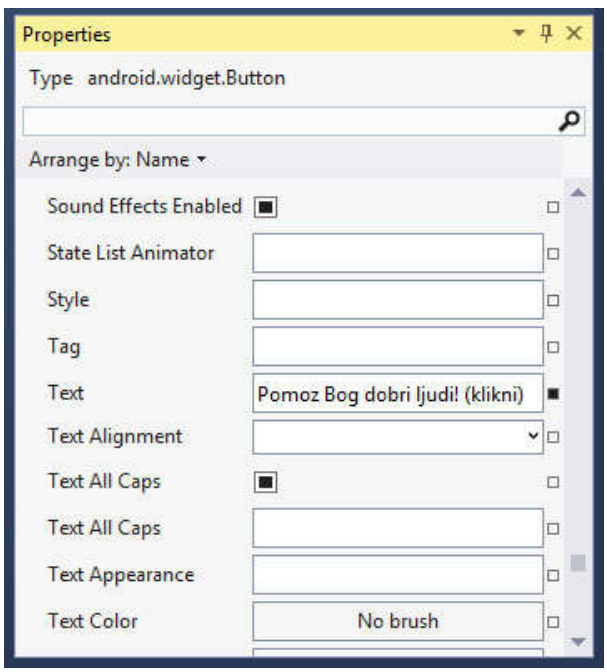
Kliknemo na **Toolbox** natpis (uz levu ivicu prozora) da bi otvorili Toolbox prozor, ako već nije otvoren, i iz njega tražimo alat **Button**, prevlačimo ga i spuštamo na dizajnersku površinu u centralnom delu Visual Studia.

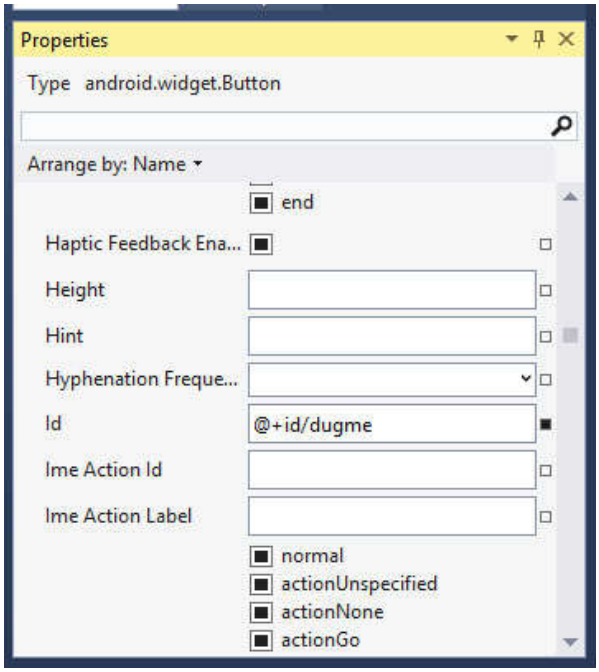


Primećujemo u desnom delu dizajnerske površine da se sadržaj fajla u XML jeziku promenio. Objasnićemo to kasnije, za sada samo registrujemo da postoje promene.



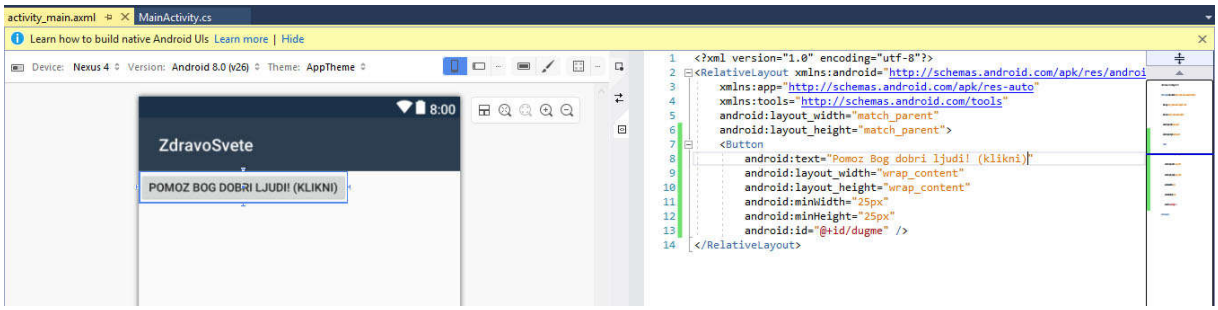
Sa selektovanom kontrolom **Button** na dizajnerskoj površini (ako nije selektovana, kliknite na nju), u prozoru **Properties** (donji desni ugao) pronalazimo svojstvo **Text**, i menjamo njen sadržaj koji glasi *Button* u *Pomoz Bog dobri ljudi! (klikni)*.





Pronalazimo u istom prozoru i svojstvo **Id** i menjamo mu sadržaj u *@+id/dugme*.

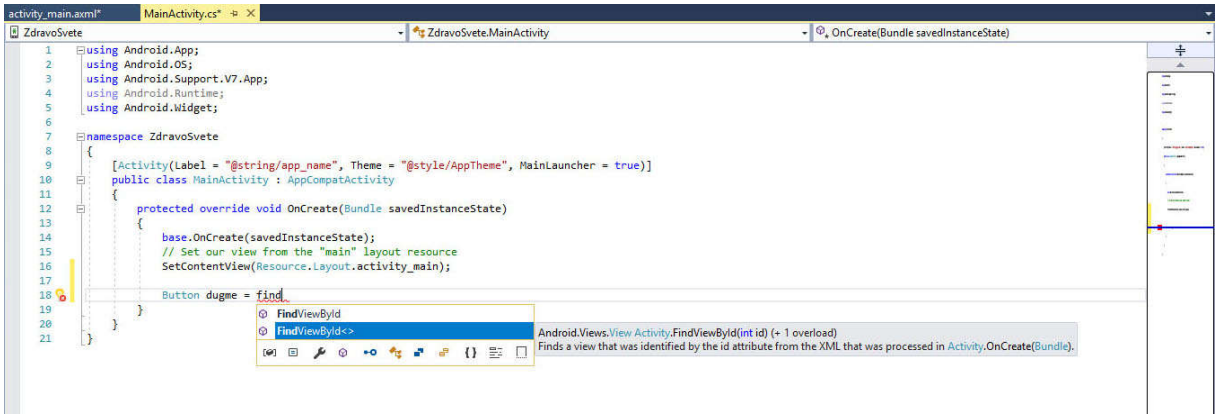
Vidimo da se izgled naše aplikacije, to jest tekst u dugmetu, promenio. Primetimo odgovarajuće promene i u otvorenom XML fajlu.



Time smo završili dizajn naše aplikacije. Vraćamo se u kod klikom na tab prozora **MainActivity.cs**.

3.3. Pisanje koda

U MainActivity klasi, nalazimo **OnCreate**() metodu. Kod za našu aplikaciju kucamo unutar **OnCreate**() metode, a ispod poziva metode **SetContentView(Resource.Layout.Main)**. Primetimo kako dok kucamo slovo po slovo, Visual Studio IDE nam nudi dopunu teksta iz spiska komandi, klasi, metodi, promenljivih, i drugih pojmova koje prepoznaje u datom kontekstu.



Najpre unosimo sledeću liniju koda, koja predstavlja način povezivanja objekata u C# (objekat **dugme1** klase *Button*) sa objektima kreiranim u korisničkom interfejsu, u ovom slučaju sa dugmetom koje smo dodali na dizajn naše aplikacije.

```
Button dugme1 = FindViewById<Button>(Resource.Id.dugme);
```

Primećujemo da se povezivanje obavlja pomoću metode *FindViewById()* koja pronalazi odgovarajući objekat interfejsa na osnovu njegovog svojstva *Id*, koje smo imenovali u dizajnu, u ovom slučaju **dugme**.

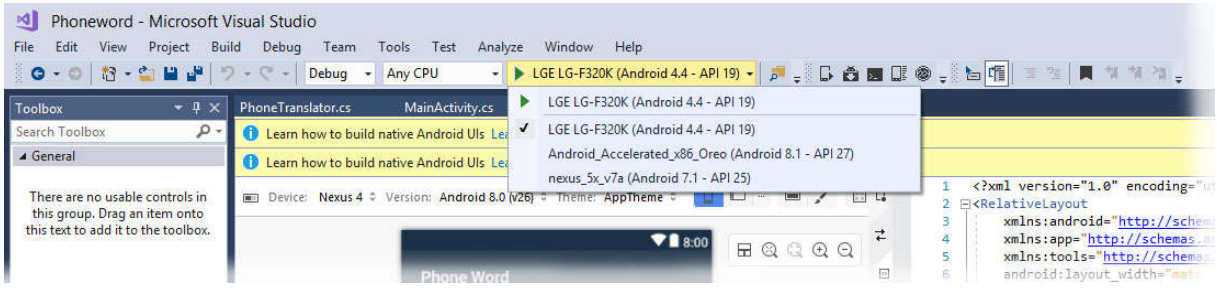
Zatim, ispod prethodnog, unosimo kod koji će se izvršavati na pritisak dugmeta.

```
            dugme1.Click += delegate
            {
                dugme1.Text = "A zdravo i tebi, zete";
            };
```

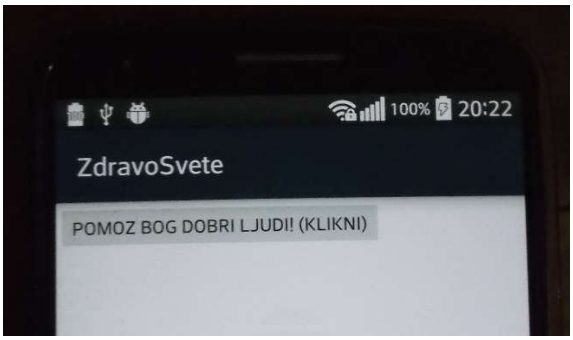
Na pritisak dugmeta, menja se njegovo svojstvo *Text*. I to je sav kod naše aplikacije.

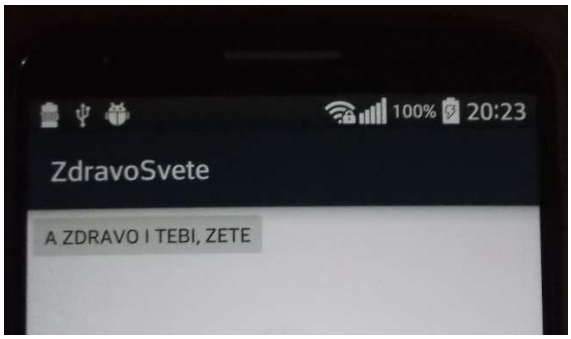
3.4. Testiranje aplikacije

Aplikaciju testiramo pokretanjem Android emulatora ili Android uređaja (ako je povezan) koji biramo iz padajućeg menija i pritiskom na zeleni trouglić pored koji označava komandu za startovanje izvršavanja.



Na slikama vidimo rezultat izvršavanja programa na Android uređaju.



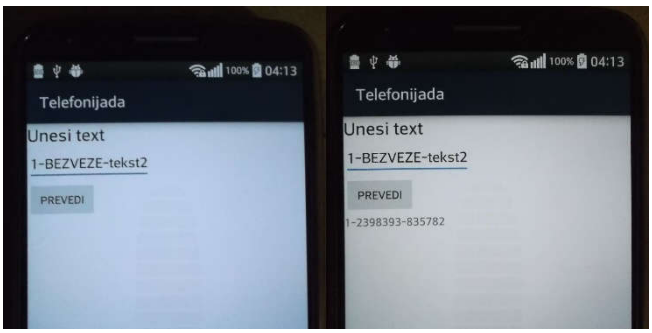


4. Android aplikacija detaljno

U ovom poglavlju napravićemo dve verzije jedne aplikacije koje će demonstrirati rad u Xamarin.Android okruženju i neke njegove mogućnosti, i detaljno ih objasniti.

4.1. Korak po korak do aplikacije

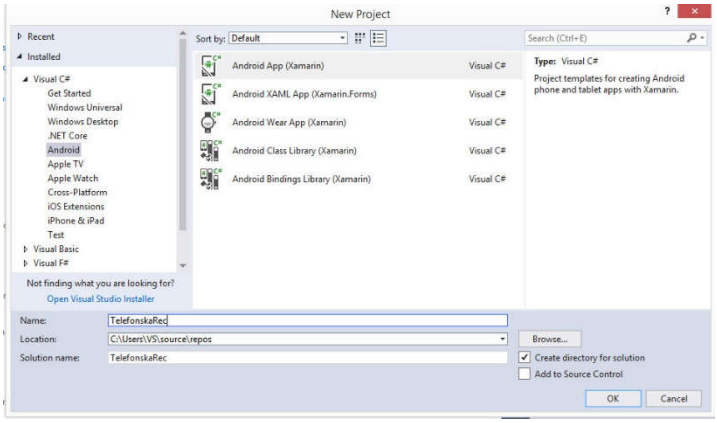
Opisaćemo najpre postupno, „od nule“, proces kreiranja relativno jednostavne Android aplikacije koja je tek nešto malo složenija od prethodne „HelloWorld“ aplikacije. Ovaj program prevodi uneti alfanumerički niz znakova u (telefonski) broj, koristeći se rasporedom sa tastaturi starih telefona. Rad gotovog programa je demonstriran na slici.



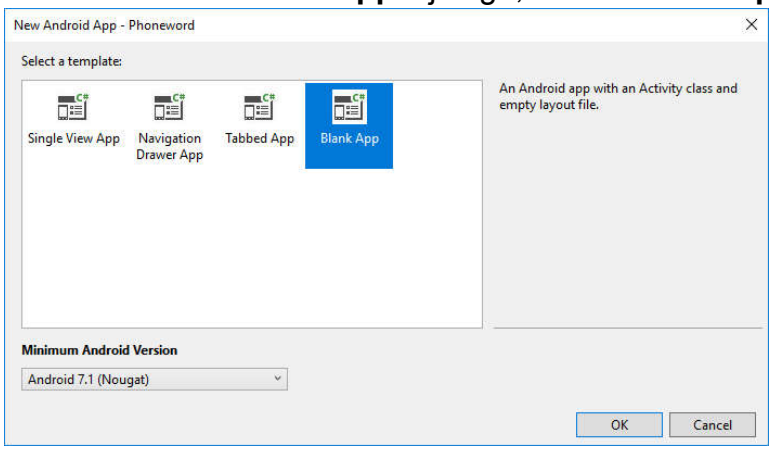
4.1.1. Pokretanje novog Android projekta

Startujemo Visual Studio. Nov projekat započinjemo kliktanjem na **File > New > Project**.

U **New Project** dijalogu, biramo **Android App** šablon (template). Dajemo ime projektu *TelefonskaRec* pa pritiskamo **OK**.

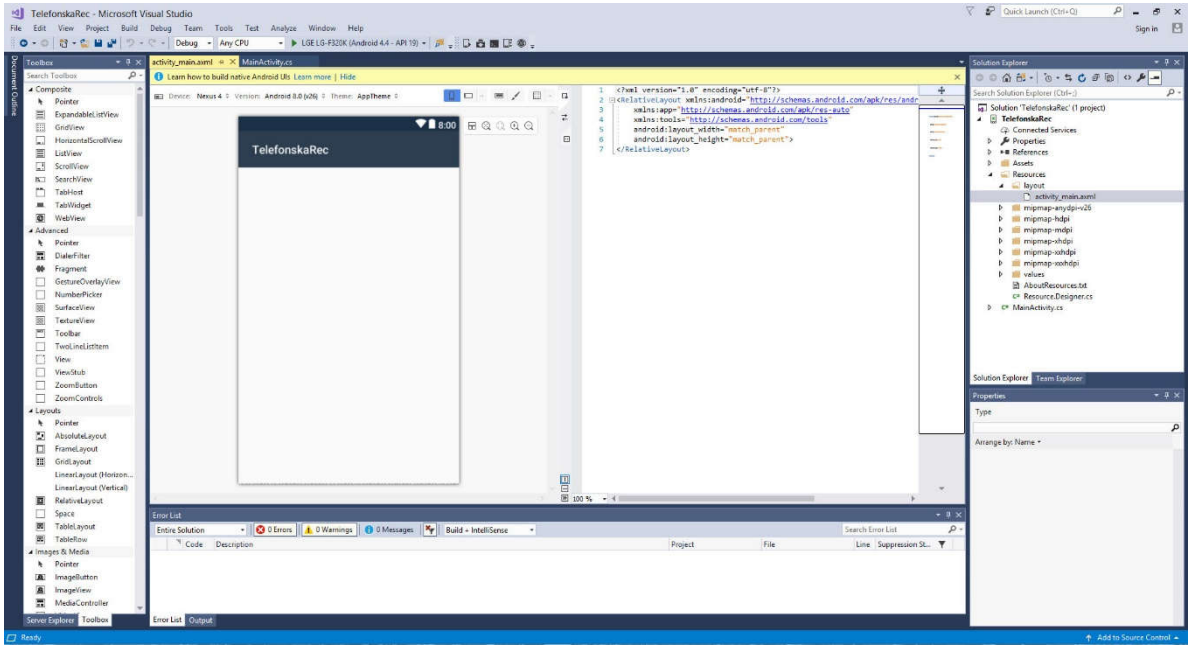


U **New Android App** dijalogu, biramo **Blank App**. Obratimo pažnju na padajući meni **Minimum Android Version**. Izborom u toj stavki ciljamo na kojoj će najstarijoj platformi naš projekat moći da radi. Za novije platforme postoji veći broj mogućnosti, ali se program neće moći izvršavati na starijim uređajima. Na kraju pritiskamo **OK** da bi startovali projekat.

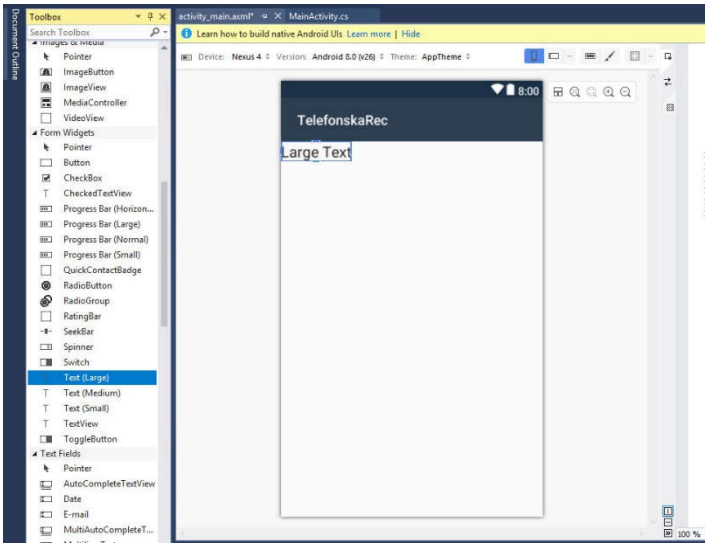


4.1.2. Kreiranje šeme elemenata (Layout) korisničkog interfejsa

Nakon što je projekat kreiran, obratimo pažnju na **Solution Explorer** prozor sa desne strane. U njemu se mogu pronaći svi resursi našeg projekta, svi pojedinačni fajlovi uključeni u projekat, kao i svi strukturni elementi našeg koda. Proširimo najpre **Resources** folder a zatim i **layout** folder unutar njega. Duplim klikom na **activity_main.axml** otvaramo taj fajl u Android Designer-u. Sada vidimo kako izgleda naša aplikacija i vizuelno.

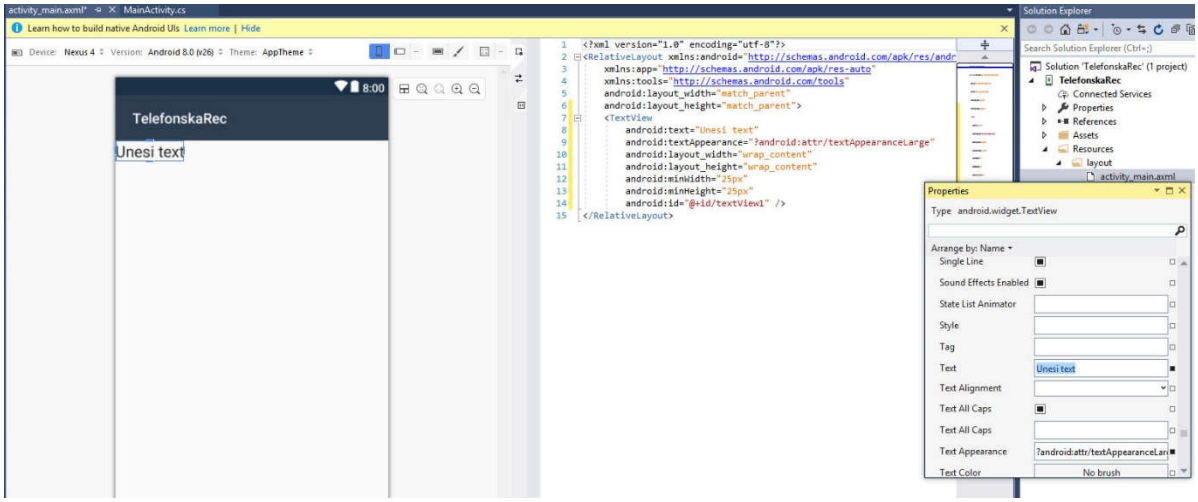


Kliknemo na **Toolbox** natpis (uz levu ivicu prozora, ukoliko nije otvoren nalazimo ga u **View > Toolbox**) da bi otvorili Toolbox prozor, i iz njega tražimo alat (u Form Widget kategoriji) **Text (Large)**, prevlačimo ga i spuštamo na dizajnersku površinu u centralnom delu Visual Studia.



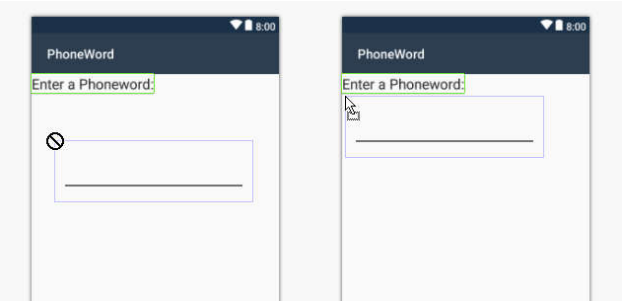
Ukoliko imamo problem da pronađemo neki alat u *Toolbox*-u, možemo ga potražiti po imenu pomoću *Search* opcije toolboxa. Takođe, ukoliko nam *Toolbox* previše zaklanja dizajnersku radnu površinu, možemo ga pomoću stilizovane špenadle (Auto Hide, u gornjem desnom uglu *Toolbox* prozora) pin-ovati da stoji uvek otvoren, čime će dobiti svoje stalno mesto pored (umesto toga što je bio preko) dizajnerske površine, ali time smanjujemo njenu površinu, a ona nam treba češće nego *Toolbox*. Naravno, *Toolbox* možemo i unpin-ovati kada nam više nije potreban.

Sa selektovanom kontrolom **Text (Large)** na dizajnerskoj površini, u prozoru **Properties** (donji desni ugao) pronalazimo svojstvo **Text** (ako je potrebno, i ovde postoji *Search* koji se odnosi na sva svojstva selektovane kontrole), i menjamo njen sadržaj koji glasi *Large Text* u *Unesi text*.

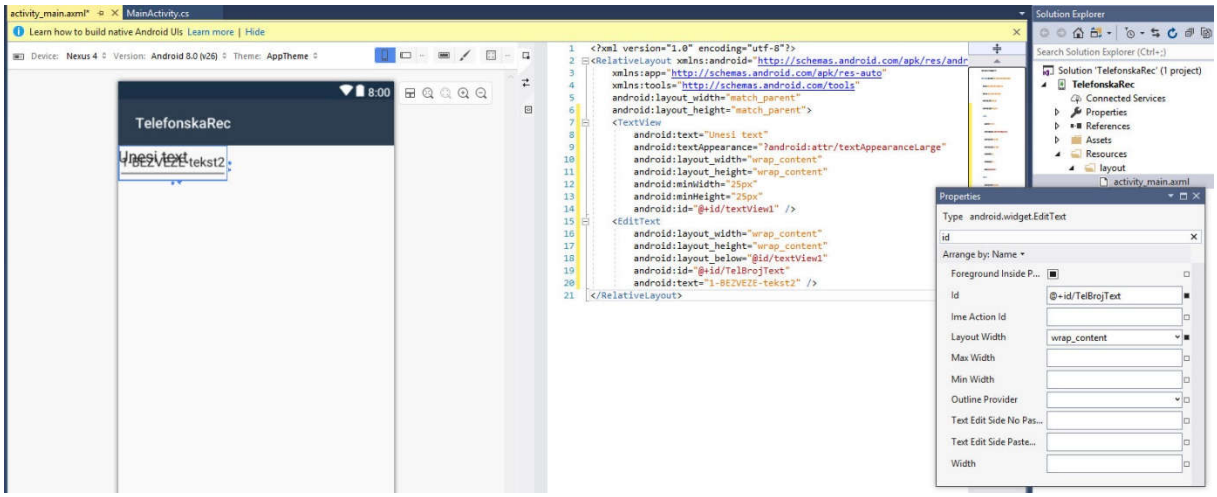


Obratimo pažnju, da se nakon završetka unosa teksta i klika van tog Text polja, u okviru otvorenog **activity_main.axml** fajla, koji zapravo u XML-u jeziku pamti kompletan opis dizajna našeg projekta, u okviru `<TextView ... />` taga koji opisuje našu **Text (Large)** kontrolu, automatski se promenila linija `android:text="Large Text"` u liniju `android:text="Unesi text"`. Na taj način je u tom XML fajlu sačuvana naša izmena u dizajnu. Kao što vidimo, izmene svojstva kontrola možemo vršiti i direktno u ovom XML fajlu, bez pretrage po *Properties* prozoru, ukoliko znamo koje svojstvo želimo da dodelimo ili izmenimo, što nakon dužeg rada u Xamarinu i pamćenja često korišćenih svojstava značajno ubrzava proces dizajna.

Sledeći korak u dizajnu naše aplikacije je da nađemo u *Toolbox*-u **Plain Text** widget (u okviru sekcije *Text Fields*) i prevučemo ga na dizajnersku površinu, pazeći da ga postavimo tačno ispod prethodnog **Text (Large)** widget-a, u suprotnom ako pokušamo na bilo kom drugom mestu, neće doći do postavljanja te kontrole na dizajnersku površinu (važi i za bilo koju drugu kontrolu, uvek vodimo računa da nov widget postavimo tačno ispod prethodnog widget-a), kao što se vidi na slici.

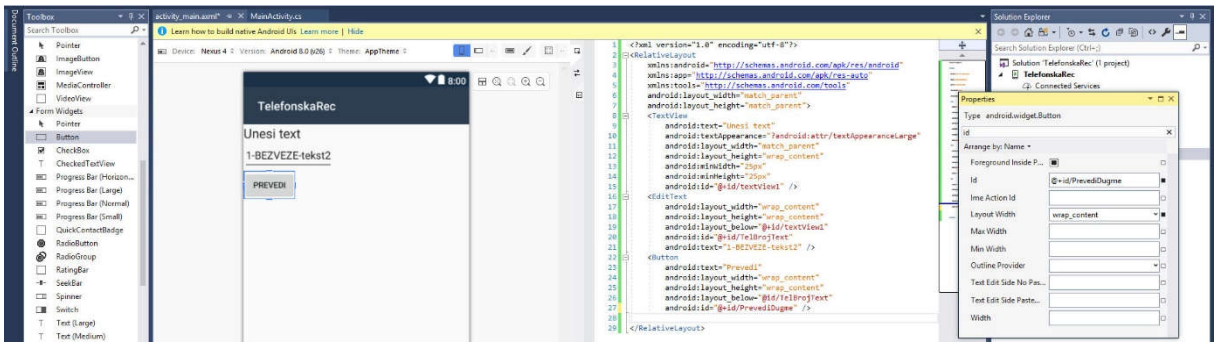


Sa selektovanim **Plain Text** widget-om na dizajnerskoj površi, u **Properties** prozoru (ili u okviru XML fajla, u okviru odgovarajućeg taga) nalazimo svojstvo **Id** ove kontrole i menjamo ga u *@+id/TelBrojText*. Zatim, kao u prethodnom slučaju, nalazimo svojstvo **Text** i menjamo ga u *1-BEZVEZE-tekst2*.

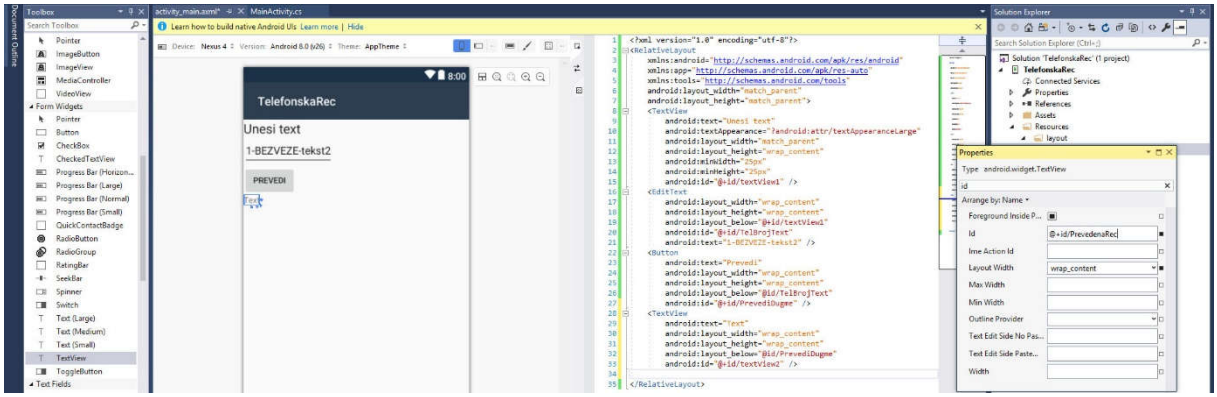


Primetimo sada u novo generisanom XML kodu, da se naš **Plain Text** alat zapravo vodi pod tagom `<EditText … />` i da je automatski dobio svojstvo `android:layout_below="@id/textView1"`, koje govori o pozicioniranju ove kontrole u odnosu na prethodnu kojoj nismo menjali svojstvo **Id** pa se generički zove *@+id/textView1*. Takođe primećujemo i značaj svojstva **Id** jer ono služi za unikatnu identifikaciju svakog elementa.

Sledeći element koji nam je potreban je kontrola **Button**, koju iz *Toolbox*-a prevlačimo na dizajnersku površ, po istom principu, lepeći je tačno ispod prethodne kontrole, a onda joj dodeljujući u svojstvu **Text** vrednost *Prevedi*, a za svojstvo **Id** vrednost *@+id/PrevediDugme*.



Istim postupkom, dodajemo u dizajn i kontrolu **TextView** iz *Toolbox*-a.



Tu kontrolu smo smestili ispod **Button**-a, njen **Text** property smo obrisali, to jest ostavili prazan string (primećujemo da je zato to njeno svojstvo nestalo iz XML-a, koji

štedi prostor time što ne čuva svojstva kojima nije dodeljena nikakva vrednost) i dodeljujemo svojstvu **Id** vrednost *@+id/PrevedenaRec*.

Možemo u XML kodu primetiti da se i ova kontrola u XML-u vodi pod istim nazivom kao i prva. Takođe, osim svojstava na koje smo do sad namerno uticali, primećujemo da sve kontrole imaju ista svojstva koja su automatski dodeljena, a koja određuju njihovu širinu i visinu, i koja su automatski podešena da budu samo tolika koliko je potrebno da obuhvate unet sadržaj, `android:layout_width="wrap_content"` i `android:layout_height="wrap_content"`.

Sadržaj **activity_main.axml** fajla koji u XML-u u potpunosti opisuje izgled naše aplikacije je dat u sledećem listingu:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        <TextView
                android:text="Unesi text"
                android:textAppearance="?android:attr/textAppearanceLarge"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:minWidth="25px"
                android:minHeight="25px"
                android:id="@+id/textView1" />
        <EditText
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_below="@+id/textView1"
                android:id="@+id/TelBrojText"
                android:text="1-BEZVEZE-tekst2" />
        <Button
                android:text="Prevedi"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_below="@id/TelBrojText"
                android:id="@+id/PrevediDugme" />
        <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_below="@id/PrevediDugme"
                android:id="@+id/PrevedenaRec" />
        <Button
                android:text="@string/translationHistory"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_below="@id/PrevedenaRec"
                android:id="@+id/IstorijaDugme"
                android:enabled="false" />
</RelativeLayout>
```

Na kraju (a valjalo bi i češće) obavezno snimamo naš rad klikom na **File > Save** za tekući fajl, ili **File > Save All** za ceo projekat ili prečicama tih komandi koje je korisno zapamtiti **Ctrl + S** ili **Ctrl + Shift +S**.

Možemo malo eksperimentisati u prozoru *Properties* pa recimo širinu nekog od elemenata promeniti iz padajućeg menija u recimo *match_parent* što će uticati da se ta kontrola raširi celom širinom ekrana. Možemo menjati margine i druga svojstva koja imaju u imenu *Layout* i videti kako ta svojstva utiču na izgled kontrole, ili čekirati neko

od *Gravity* svojstava pa videti kako to utiče na pozicioniranje elementa u odnosu na ekran, menjati *Text Size* ili *Text Style* i razna druga svojstva koja utiču na izgled. Treba zapamtiti šta smo menjali da bi eventualno ispravili ono što nam ne odgovara, mada je za to često korisno i **Undo** dugme koje poništava jednu ili više naših poslednjih akcija, kao i dugme **Redo**, koje poništava dejstvo **Undo**-a..

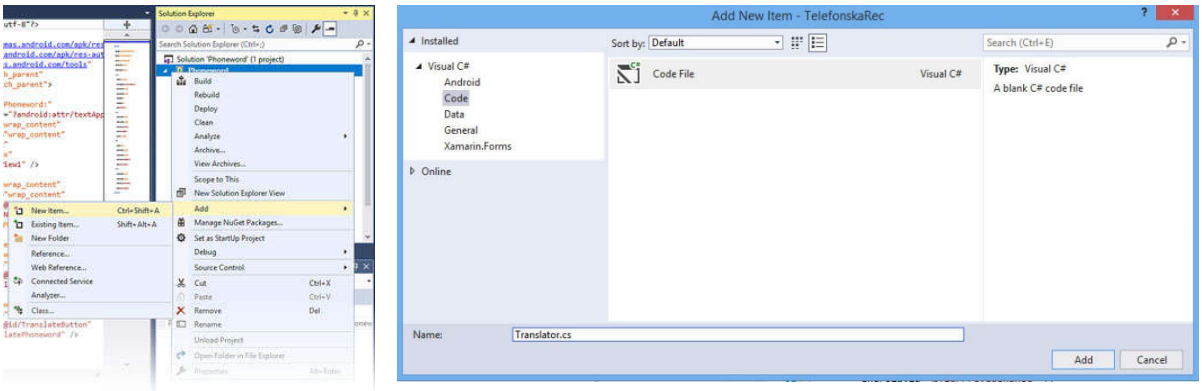
Neki od problema koji se mogu relativno često javiti u radu sa dizajnom interfejsa jeste da se prikaz dizajna više ne vidi (uz komentar da nedostaje neka komponenta) ukoliko smo menjali verzije Androida, teme ili nešto drugo. U tom slučaju ponekad pomognu promene podešavanja na vrhu dizajnerskog ekrana, najčešće promena *Theme* u (*Default theme*) iz padajućeg menija.

Dešava se povremeno, da prilikom dodavanja kontroli na dizajnersku površinu, elementi počnu da se preklapaju, najčešći uzrok je nepažljivo povezivanje redosleda kontroli ili promena njihovog **Id**-a, pa se vizuelnom kontrolom XML-a može pronaći greška i ispraviti. Mada, dešava se nekada isti vizuelni problem i kad je sve ispravno povezano, ali se on obično sam razreši osvežavanjem prilikom startovanja izvršavanja programa ili na drugi način.

4.3.3. Kod za translaciju

Završivši dizajn interfejsa, prelazimo na sledeći korak, da dodamo kod koji će se baviti nekim radnjama koji ova aplikacija treba da obavlja, prevođenjem alfanumeričkog koda u telefonski broj. Kompletan kod, obzirom na jednostavnost, može bez problema da se nalazi i u okviru već postojećeg MainActivity.cs fajla, ali pošto želimo da sledimo principe objektnog programiranja, i omogućimo da taj kod bude univerzalno upotrebljiv, pa i za druge platforme, kod za translaciju ćemo odvojiti u poseban fajl, odnosno posebnu klasu namenjenu samo tome.

Dodaćemo novi fajl u naš projekat tako što ćemo u *Solution Exploreru* kliknuti desnim dugmetom miša na naslov projekta *TelefonskaRec*, i izabrati **Add > New Item** kao na slici. U dijalogu koji se otvorio biramo sa leve strane **Visual C# > Code** i u srednjem delu najzad **Code File**. Dajemo mu ime **Translator.cs** i pritiskamo dugme **Add.**



Na taj način smo kreirali novu praznu C# klasu. Unećemo u nju sledeći kod:

```
using System.Text;
using System;
namespace Core
{
    public static class Prevodilac
    {
        public static string UBroj(string unetaRec)
        {
            if (string.IsNullOrWhiteSpace(unetaRec))
                return "";
            else
                unetaRec = unetaRec.ToUpperInvariant();

            var novBroj = new StringBuilder();
            foreach (var znak in unetaRec)
            {
                if (" -0123456789".Sadrzi(znak))
                {
                    novBroj.Append(znak);
                }
                else
                {
                    var resultat = PrevediUBroj(znak);
                    if (resultat != null)
                        novBroj.Append(resultat);
                }
            }
            return novBroj.ToString();
        }
        static bool Sadrzi(this string ključniString, char znak)
        {
            return ključniString.IndexOf(znak) >= 0;
        }
        static int? PrevediUBroj(char znak)
        {
            if ("ABC".Sadrzi(znak))
                return 2;
            else if ("DEF".Sadrzi(znak))
                return 3;
            else if ("GHI".Sadrzi(znak))
                return 4;
            else if ("JKL".Sadrzi(znak))
                return 5;
            else if ("MNO".Sadrzi(znak))
                return 6;
            else if ("PQRS".Sadrzi(znak))
                return 7;
            else if ("TUV".Sadrzi(znak))
                return 8;
            else if ("WXYZ".Sadrzi(znak))
                return 9;
            return null;
        }
    }
}
```

Naša klasa *Prevodilac* sadrži nekoliko funkcija (metoda), i *static* je tipa. Takođe, i sve funkcije su *static* tipa što znači da se mogu pozivati bez prethodnog kreiranja objekta, direktno pozivom na ime klase i funkcije unutar nje. Takođe, sve je *public* dostupnosti, bez restrikcija.

Funkcija koju ćemo pozivati iz glavnog programa se zove **UBroj()** i vraća *string* tip podataka, a takođe i prima kao ulaz *string* koji se zove **unetaRec.** Najpre se u njoj proverava da li je ulazni string prazan ili sadrži samo prazan prostor, i ako je taj uslov ispunjen izlazimo iz funkcije i vraćamo prazan string. U suprotnom ulazni string **unetaRec** konvertujemo u isti string u kome su sva slova velika, metodom stringa **ToUpperInvariant().**

Definišemo novu promenljivu **novBroj** koja je tipa *StringBuilder* što je tip podataka klase slične klasi *string*, samo jednostavniji, brže se obrađuje i uglavnom služi onome što mu ime i govori.

Pravimo petlju koja za svaki karakter **c** (tipa *varchar*) iz stringa **unetaRec** proverava da li se on sadrži u stringu " **-0123456789**", tj. da li je on cifra ili znak "**-**" ili " " (praznina) pozivanjem metode **Sadrzi**(), i ako je ispunjen uslov, dodaje taj karakter **c** direktno u **novBroj**. Ukoliko nije ispunjen uslov, pozivamo funkciju **PrevediUBroj()** čiju vraćenu vrednost dodeljujemo integer promenljivoj **result**. Ukoliko **result** ima vrednost različitu od *null*, tu vrednost dodajemo na **novBroj**.

Na kraju funkcije, **novBroj** konvertujemo u string i vraćamo tu vrednost.

Sledeća funkcija je **Sadrzi()**, koja će predstavljati novu metodu *String*-a, na izlazu vraća *bool* vrednost, a prima string iz koga se poziva i karakter koji se proverava. Funkcija služi za proveru da li se karakter sadrži u stringu iz koga se poziva metoda, proverom pomoću sistemske metode klase *String* **IndexOf()** da li je nađeni indeks koji vraća mesto karaktera u stringu veći ili jednak nuli, ako jeste, ispunjen je uslov da se karakter sadrži u stringu pa vraća vrednost *TRUE*, u suprotnom vraća *FALSE*. Dakle, izraz iza naredbe *return* je logički, poređenje.

Na kraju tu je i funkcija **PrevediUBroj()** koja prima karakter a vraća integer, cifru koja se bira zavisno od uslova u kom od navedenih stringova se nalazi ulazni karakter.

Ovaj fajl na kraju snimimo sa **CRTL + S** i možemo ga zatvoriti.

4.1.3. Povezivanje sa interfejsom

Sada naš kod moramo povezati sa elementima interfejsa, da bi korisnik mogao da utiče na dešavanja u programu, odnosno da bi on bio interaktivan. Počećemo sa povezivanjem **PrevediDugme**-ta. U MainActivity klasi, nalazimo **OnCreate**() metodu. Kod za dugme dodajemo unutar **OnCreate**() metode, a ispod poziva metode **SetContentView(Resource.Layout.Main)**. Dakle prvo lociramo to mesto u kodu:

```
using Android.App;
using Android.OS;
using Android.Support.V7.App;
using Android.Runtime;
using Android.Widget;

namespace TelefonskaRec
{
    [Activity(Label = "@string/app_name", Theme = "@style/AppTheme", MainLauncher = true)]
    public class MainActivity : AppCompatActivity
    {
        protected override void OnCreate(Bundle savedInstanceState)
        {
            base.OnCreate(savedInstanceState);
            // Set our view from the "main" layout resource
            SetContentView(Resource.Layout.activity_main);

            // Nov kod pišemo OVDE!!!
        }
    }
}
```

A zatim na to mesto dodajemo reference na kontrole korisničkog interfejsa koje smo kreirali u Android dizajneru i povezujemo sa objektima odgovarajućih klasa u C#:

```
// Preuzimamo UI controle iz učitanog layout-a
EditText telBrojTekst = FindViewById<EditText>(Resource.Id.TelBrojText);
TextView prevedenaRec = FindViewById<TextView>(Resource.Id.PrevedenaRec);
Button dugmePrevedi = FindViewById<Button>(Resource.Id.PrevediDugme);
```

Na kraju dodajemo kod koji će se izvršavati na pritisak dugmeta **Prevedi**:

```
            //  kod za dugmePrevedidugme
            dugmePrevedi.Click += (sender, e) =>
            {
                //  prevodimo korisnikov alphanumericki telefonski broj u numericki
                string prevedenBroj = Core.Prevodilac.UBroj(telBrojTekst.Text);
                if (string.IsNullOrWhiteSpace(prevedenBroj))
                {
                    prevedenaRec.Text = string.Empty;
                }
                else
                {
                    prevedenaRec.Text = prevedenBroj;
                }
            };
```

Kod je jednostavan, string **prevedenBroj** dobija vrednost iz funkcije **UBroj()** (koju pozivamo s kompletnom putanjom, tj. imenom namespace-a *Core* i klase *Prevodilac)*, kojoj smo prosledili svojstvo (property) *Tekst* objekta koji je povezan sa kontrolom interfejsa za unos teksta, čime smo praktično prosledili sadržaj teksta unetog u kontrolu. Zatim proveravamo da li je string **prevedenBroj** prazan ili je samo prazan prostor i ako jeste, svojstvu *Text* objekta povezanog sa kontrolom interfejsa za prikaz rezultata dodeljujemo prazan string, a ako nije ispunjen taj uslov, toj kontroli prosleđujemo string **prevedenBroj.**

Završili smo sav posao i ostaje nam samo da snimimo ceo projekat i pokrenemo kompajliranje i izgradnju aplikacije selektovanjem **Build > Rebuild Solution** iz menija ili prečicom **CTRL-SHIFT-B**.

Ukoliko kompajler prijavi greške, ispravljamo ih dok se aplikacija ne kompajlira uspešno. Ukoliko se javi greška: *Resource does not exist in the current context*, proverimo da li se ime namespace-a u fajlu **MainActivity.cs** poklapa sa imenom projekta. Ako se i dalje pri kompajliranju javlja greška, proverite da li su instalirani poslednji update-ovi za Visual Studio.

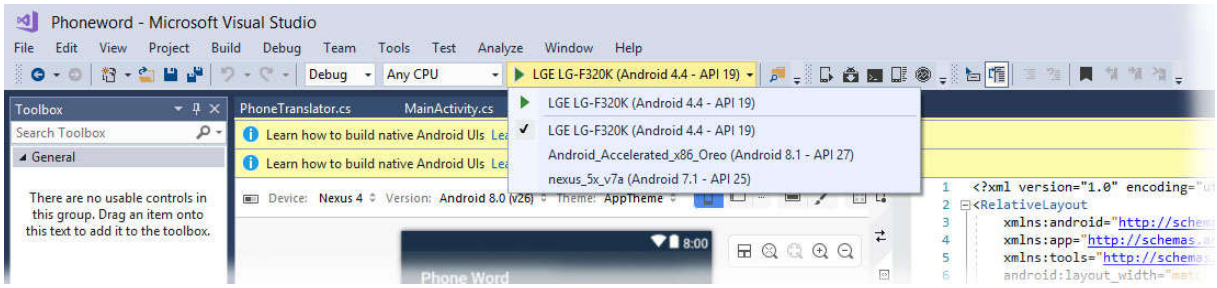
4.1.4. Davanje imena aplikaciji

Sada kada smo završili aplikaciju, možemo joj dati i ime. Ona je automatski dobila isto ime kao i projekat *TelefonskaRec* , ali mi ga možemo i promeniti. Otvaramo u *Solution Exploreru* folder **Resurses**, zatim unutar njega folder **values** i duplim klikom otvaramo fajl **strings.xml**. U njemu menjamo **app_name** string i novo ime aplikacije *Telefonijada*.

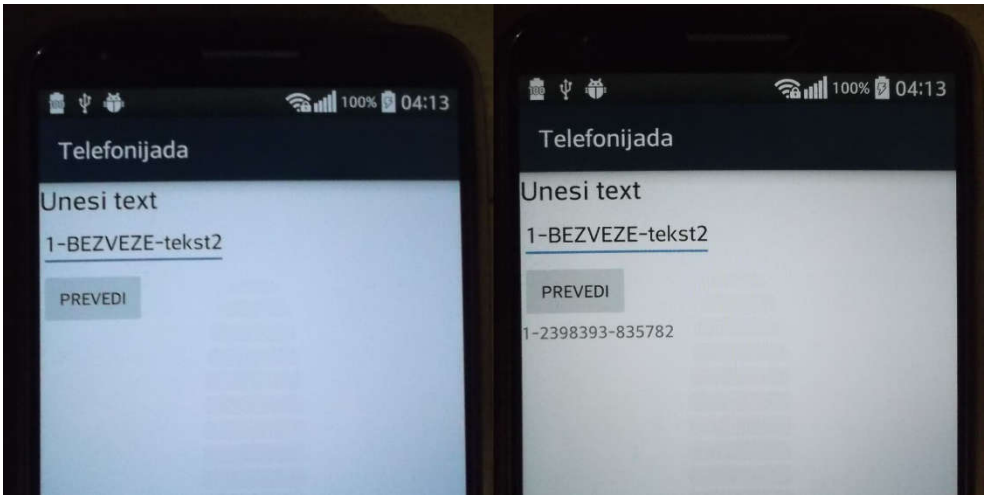
```
<resources>
    <string name="app_name">Telefonijada</string>
    <string name="action_settings">Settings</string>
</resources>
```

4.1.5. Testiranje aplikacije

Aplikaciju testiramo pokretanjem Android emulatora ili (ako imamo povezan fizički uređaj) Android uređaja koji biramo iz padajućeg menija i pritiskom na zeleni trouglić pored koji označava komandu za startovanje izvršavanja.



U polje za unos teksta unesemo neki alfanumerički tekst, i pritiskom na dugme **Prevedi** dobijamo otprilike ovakav rezultat izvršavanja programa:



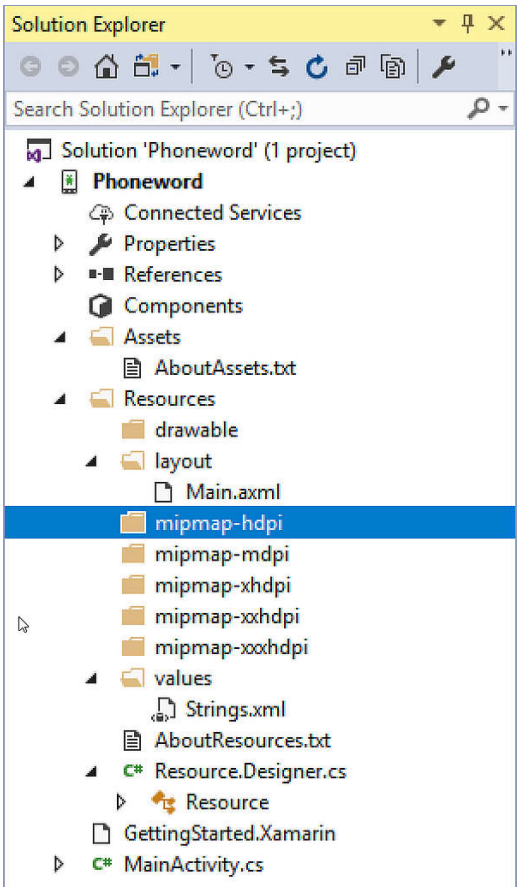
4.2. Dublje razumevanje naše Xamarin aplikacije

Dok smo kreirali prethodnu aplikaciju, uz put smo upoznali neke koncepte, alate i korake potrebne za izgradnju te aplikacije, a sada ćemo se pozabaviti detaljnije načinom na koji Android aplikacija funkcioniše.

4.2.1. Anatomija Xamarin.Android aplikacije

Solution Explorer sadrži strukturu direktorijuma i svih fajlova vezanih za projekat. Kada smo pokrenuli nov Solution (rešenje, može sadržati više projekata) *TelefonskaRec* , Android projekat *TelefonskaRec* je smešten u Solution Explorer.

Obratimo pažnju na elemente unutar projekta i njihovu namenu.



- Properties - Sadrži AndroidManifest.xml fajl koji opisuje sve što je potrebno za Xamarin.Android aplikaciju, ime, broj verzije i dozvole. U istom folderu je takođe i AssemblyInfo.cs, koji je fajl sa metapodacima .NET-a, koji možemo popuniti nekim osnovnim informacijama o našoj aplikaciji.

- References - Sadrži komponente potrebne za izgradnju i pokretanje aplikacije. U njoj možemo videti .NET komponente kao što su System, System.Core, i System.Xml, kao reference na Xamarin Mono.Android komponente.

- Assets - Sadrži fajlove potrebne za rad aplikacije, uključujući fontove, lokalne podatke, tekstualne fajlove. Ovim fajlovima je moguće pristupiti kroz generisanu klasu Assets. Više informacija o Asset-ima na linku Using Android Assets.

- Resources - Sadrži resurse aplikacije kao što su stringovi, slike i šeme rasporeda elemenata (layouts). Ovim resursima se može pristupiti kroz generisanu klasu Resourse. Više detalja na linku Android Resources. U ovaj folder je uključeno i sažeto uputstvo o resursima u **AboutResources.txt** fajlu.

U **Resources** folderu su folderi **drawable**, **layout**, **mipmap** (više njih) i **values**, kao i fajl **Resource.Designer.cs**.

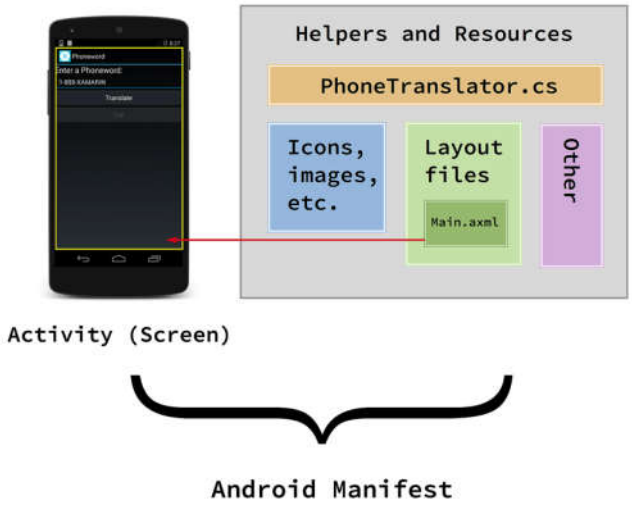
- **drawable** – Sadrži resurse kao što su slike (više detalja na drawable resources).
- **mipmap** – Sadrži fajlove za ikonice za startovanje aplikacije, za ekrane različitih gustina tačaka, u png formatu, **Icon.png**.
- **layout** – Sadrži Android dizajner fajlove (.axml) koji definišu korisnički interfejs (user interface, UI) za svaki ekran ili aktivnost. Po default-u se kreira layout koji se zove **Main.axml**, odnosno **activity_main.axml**, zavisno od verzije.
- **values** – Sadrži XML fajlove koji čuvaju neke jednostavne veličine kao što su stringovi, integer-i, boje. Na primer, po default-u je kreiran fajl **Strings.xml** koji čuva stringove kojima je definisano na primer ime aplikacije.
- **Resource.designer.cs** – Poznat i kao resurs klasa, ovaj fajl je parcijalna klasa koja čuva unikatne ID-ove dodeljene svakom resursu. Generiše se automatski od strane Android alata, i regeneriše se po potrebi. Ne bi ga trebalo ručno menjati jer će Android novim upisom pobrisati sve naše promene.

4.2.2. Fundamenti Android aplikacija

Android aplikacije nemaju singularnu startnu tačku, ne postoji jedinstvena linija koda u aplikaciji koju operativni sistem poziva da bi startovao aplikaciju. Aplikacija se zapravo startuje kada Android instancira neku od klasa aplikacije, tokom čega se učitava čitav proces aplikacije u memoriju.

Ova unikatna odlika Androida je korisna pri dizajniranju složenih aplikacija ili interakciji sa Android operativnim sistemom, ali i čine stvari kompleksnim u slučaju jednostavne aplikacije kao što je *TelefonskaRec* . U ovom delu ćemo se baviti samo aplikacijom koja koristi najčešći ulaz u aplikaciju, prvi ekran.

Kada aplikaciju *TelefonskaRec* otvorimo prvi put u emulatoru ili uređaju, operativni sistem kreira prvu aktivnost (Activity). *Activity* je specijalna klasa Androida koja odgovara jednom ekranu aplikacije i odgovorna je za iscrtavanje i pokretanje korisničkog interfejsa. Kada Android kreira prvu aktivnost aplikacije, učitava i čitavu aplikaciju.

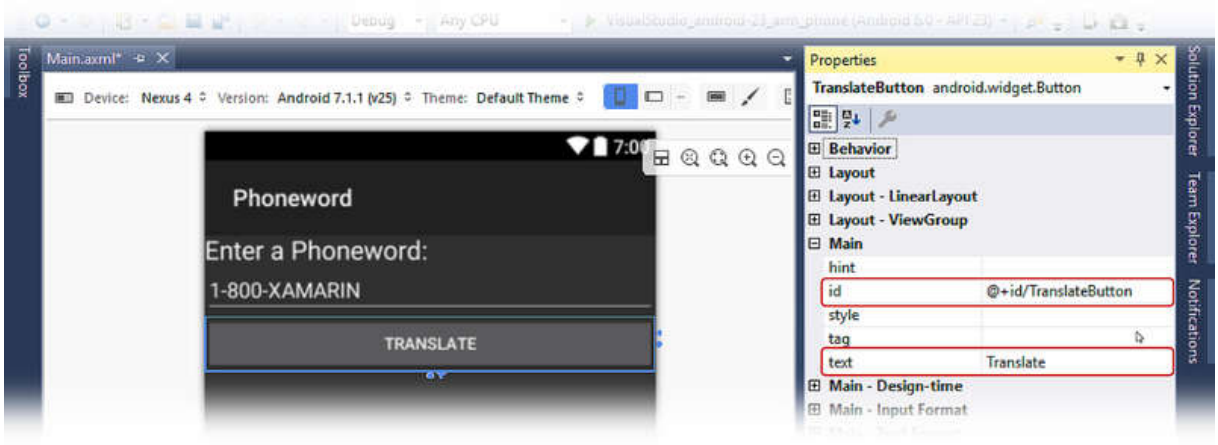


Obzirom da ne postoji linearna progresija kroz aplikaciju (ona se može startovati iz više pristupnih tačaka), Android ima jedinstven način vođenja računa koje klase i fajlovi pripadaju aplikaciji. U našem primeru, svi delovi koji čine aplikaciju su registrovani u XML fajlu **Android Manifest**, koji vodi računa o sadržaju, svojstvima i dozvolama i prezentuje ih operativnom sistemu. Našu aplikaciju možemo zamisliti kao jedan Activity (ekran) i kolekciju resursa povezanih **Android Manifest** fajlom.

4.2.3. Korisnički interfejs (UI)

Kao što smo rekli, interfejs za prvi ekran se nalazi u fajlu **activity_main.axml**, koji je dizajnerski fajl. Ime *Main* je proizvoljno, layout fajl se mogao zvati i drugačije. Kada ga otvorimo u VS, otvara se vizuelni editor za Android layout fajlove, zvani Android Designer.

U našoj aplikaciji setovali smo ID dugmeta **Prevedi** na *@+id/PrevediDugme*.



Kada setujemo svojstvo *id* **Prevedi** dugmeta, Android dizajner je mapirao **Prevedi** kontrolu u *Resours* klasu i dodelio joj *resourse ID* of Prevedi . Zahvaljujući ovom mapiranju vizuelne kontrole u klasu moguća je upotreba **Prevedi** dugmeta i drugih kontrola u kodu aplikacije. Reprezentacija kontrole u kodu se povezuje sa vizuelnom reprezentacijom kontrole preko svojstva **id**.

Sve elemente koje smo kreirali u Android dizajneru, možemo videti u XML fajlu, kao što smo već pokazali, i odatle koristiti dalje.

4.2.4. Aktivnosti (Activity class) u Android aplikaciji

Activity klasa sadrži kod koji pokreće korisnički interfejs, i odgovorna je za interakciju sa korisnikom. *TelefonskaRec* aplikacija ima samo jedan ekran (Activity). Klasa koja pokreće taj ekran je nazvana MainActivity i snalazi se u fajlu **MainActivity.cs**. Ime MainActivity u Androidu (za razliku od uobičajenog C# programiranja) nema specijalan značaj, i ne bi bilo bitno i da se nazove drugačije, mada se konvencionalno prva aktivnost u aplikaciji obično zove MainActivity.

Kada otvorimo **MainActivity.cs** fajl, vidimo da je MainActivity klasa podklasa (izvedena klasa od) klase Activity, i da je krasi Activity atribut ispred nje:

```
[Activity (Label = "TelefonskaRec", MainLauncher = true)]
public class MainActivity : Activity
{
  ...
}
```

U novijim verzijama se taj atribut neznatno razlikuje, ali mu je poenta ista:

```
[Activity(Label = "@string/app_name", Theme = "@style/AppTheme", MainLauncher = true)]
public class MainActivity : AppCompatActivity
{ ... }
```

Activity Attribute registruje Activity unutar Android Manifest-a, čime obaveštava Android da je ova klasa deo *TelefonskaRec* aplikacije o kojoj brine ovaj manifest.

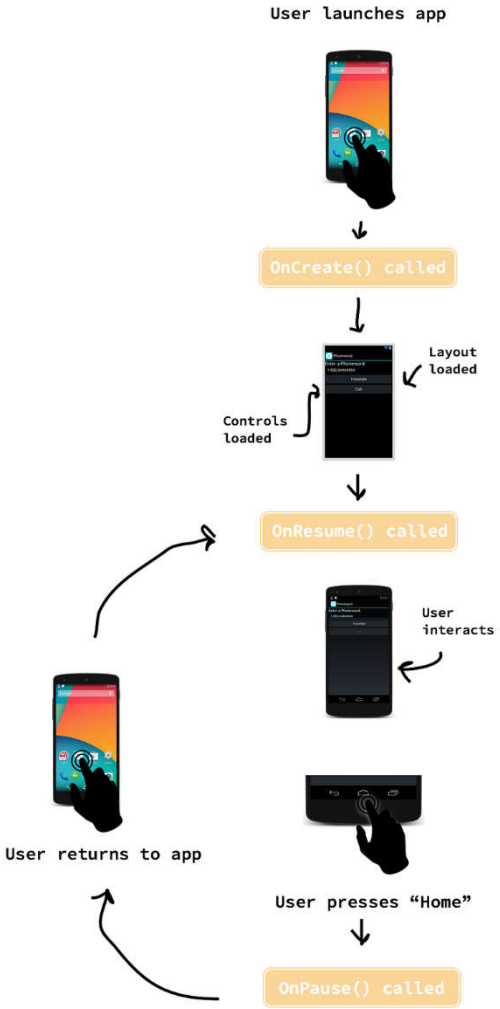
Label svojstvo setuje tekst koji će se prikazati na vrhu ekrana, i dok je u starijoj verziji tekst dodeljen direktno, u novijoj se čita iz **Resourses / values / strings.xml** fajla gde smo mi prethodno promenili ime aplikacije.

Theme svojstvo (kod novije verzije), slično prethodnom, predstavlja referencu na resursni fajl **Resourses / values / styles.xml** u kome možemo menjati temu (ukupan izgled) ovog Activity-ja. Ovo svojstvo može biti uzrok nekompatibilnosti između različitih verzija programa i dovesti do neprikazivanja layout-a u dizajnerskoj površini, a ponekad i nemogućnosti izvršenja aplikacije zbog nedostatka resursa. Zato je preporuka obavezno update-ovati Xamarin i VS pre otvaranja projekta.

MainLauncher svojstvo govori Androidu da prikaže ovaj Activity kada se aplikacija startuje. Ovo svojstvo je važno kada imamo više Activity-ja (ekrana) u aplikaciji, čime ćemo se pozabaviti kasnije.

4.2.5. Životni vek Aktivnosti

Aktivnosti u Androidu prolaze kroz različite faze životnog ciklusa, zavisno od njihovih interakcija sa korisnikom. Aktivnost može biti kreirana, startovana, pauzirana, nastavljena, uništena i tako dalje. Activity klasa sadrži metode koje sistem poziva u određenim trenucima životnog veka Aktivnosti. Na dijagramu desno se vidi tipičan životni ciklus Aktivnosti, kao i odgovarajuće metode određenim stanjima aktivnosti.



Izmenom (override) metoda klase Activity vezanih za životni ciklus, možemo kontrolisati njeno ponašanje, kako se učitava, kako reaguje na korisnikove akcije, pa i to šta se dešava kada nestane sa ekrana uređaja. Na primer možemo uraditi *override* neke od metoda životnog ciklusa sa ovog dijagrama kako bi izvršili neke važne zadatke:

- **OnCreate** – Kreira prikaze, inicijalizuje promenljive, i izvodi druge pripremne radnje potrebne da se izvrše pre nego što

korisnik vidi Activity. Ovaj metod se poziva samo jednom, kada se Activity učita u memoriju.

- **OnResume** – Izvršava sve zadatke koji se moraju desiti svaki put kada se taj Activity vrati na ekran uređaja.
- **OnPause** – Izvršava sve zadatke koji se moraju desiti svaki put kada taj Activity napusti ekran uređaja.

Dodavanjem našeg novog koda u metode životnog ciklusa Activity, mi radimo *override* bazne implementacije te metode. Ulazimo u postojeći metod (koji već u sebi ima neki kod) i proširujemo taj metod sopstvenim kodom. Iz našeg metoda pozivamo baznu implementaciju tog metoda, kako bi osigurali da se originalni kod izvrši pre našeg novog koda.

Životni ciklus Aktivnosti je važan i kompleksan deo Android sistema. Više o Aktivnostima možemo pročitati u vodiču Activity Lifecycle, a mi ćemo se sada malo više fokusirati ne jednu metodu, na prvu fazu životnog ciklusa Aktivnosti, metodu *OnCreate*.

3.2.5.1. OnCreate metod

Android poziva *OnCreate* metodu klase *Activity* kada kreira Aktivnost, a pre nego što je ekran prikazan korisniku. Možemo *override*-ovati tu metodu da bi kreirali prikaz i pripremili svoj Activity prema potrebama korisnika.

```
protected override void OnCreate (Bundle bundle)
{
    base.OnCreate (bundle);
    // Setujemo naš prikaz iz "activity_main" layout resurs fajla
    SetContentView (Resource.Layout.activity_main);
    // Nov kod pišemo OVDE!!!
}
```

U našoj aplikaciji *TelefonskaRec* , prva stvar koju odrađujemo u OnCreate metodi učitavanje korisničkog interfejsa kreiranog u Android dizajneru. Da vi učitali UI, pozivamo funkciju SetContentView i prosleđujemo joj *resource layout name* za layout fajl *activity_main.axml.*

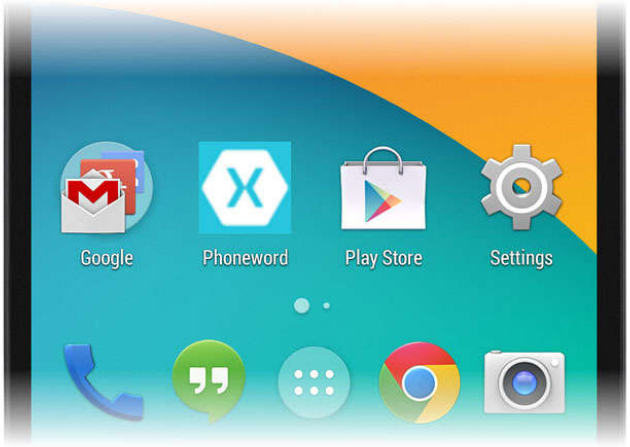
Kada se MainActivity startuje, ona kreira prikaz baziran na sadržaju fajla *activity_main.axml.* Primećujemo da ime layout fajla (*activity_main.axml*) odgovara imenu Activity (MainActivity). Ovo nije neophodno iz perspektive Androida, ali kako budemo dodavali više Activity ekrana aplikaciji, nalazimo da ova konvencija olakšava uparivanje koda sa layout fajlom.

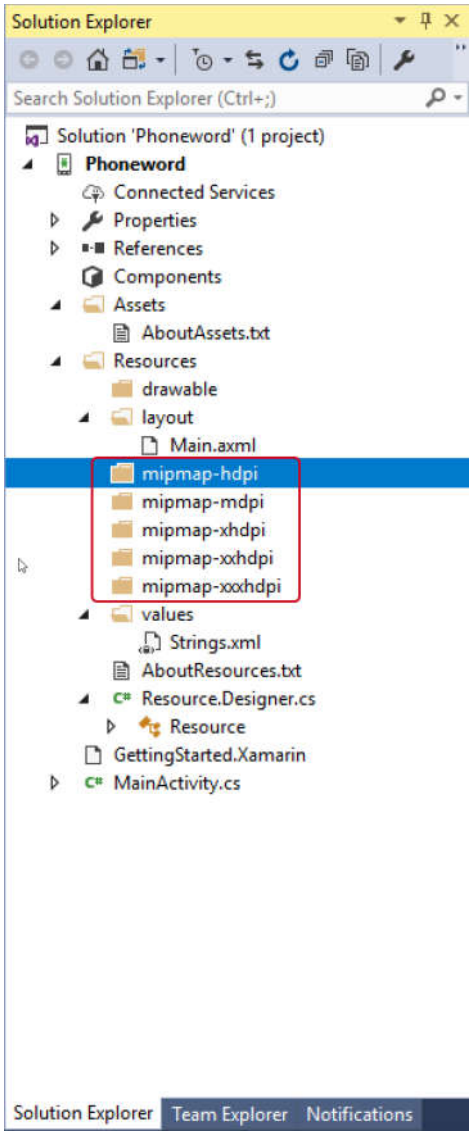
Nakon što je layout fajl pripremljen, tražimo kontrole pomoću funkcije *FindViewById()* kojoj prosleđujemo resurs ID željene kontrole, kao što smo već objasnili u odeljku 3.3.4. Povezivanje sa interfejsom.

Kada smo napravili reference na kontrole u layout fajlu, programiramo ih da reaguju na korisnikove akcije, što smo već objasnili u istom odeljku.

4.2.6. Ikonice za različite uređaje

Android uređaji dolaze sa različitim veličinama ekrana i rezolucijama, pa ne izgledaju sve slike dobro na svim ekranima. Na slici desno je ikonica male gustine na ekranu velike gustine, pa zato deluje mutno u odnosu na ostale ikone.





Da bi se to predupredilo, dobra praksa je dodavanje ikonica različitih rezolucija u **Resourse** folder. U njemu je Android napravio set različitih verzija **mipmap** foldera, da bi baratao ikonicama različitih gustina, *mdpi* za srednje (medium dot per inch), *hdpi* za guste (high dpi), i *xhdpi, xxhdpi, xxxhdpi* za veoma velike gustine ekrana. Ikonice različitih veličina su smeštene u odgovarajuće **mipmap** foldere tako da Android može sam da izabere ikonicu odgovarajuće gustine, kao na slici ispod.

Obzirom da nemamo svi umetničke sposobnosti ili dizajnera pri ruci, a želimo da naše ikonice ipak ne budu generičke, kako bi se na neki način isticale u odnosu na druge, njih možemo nabaviti, između bezbroj drugih načina, i na neki od sledećih:

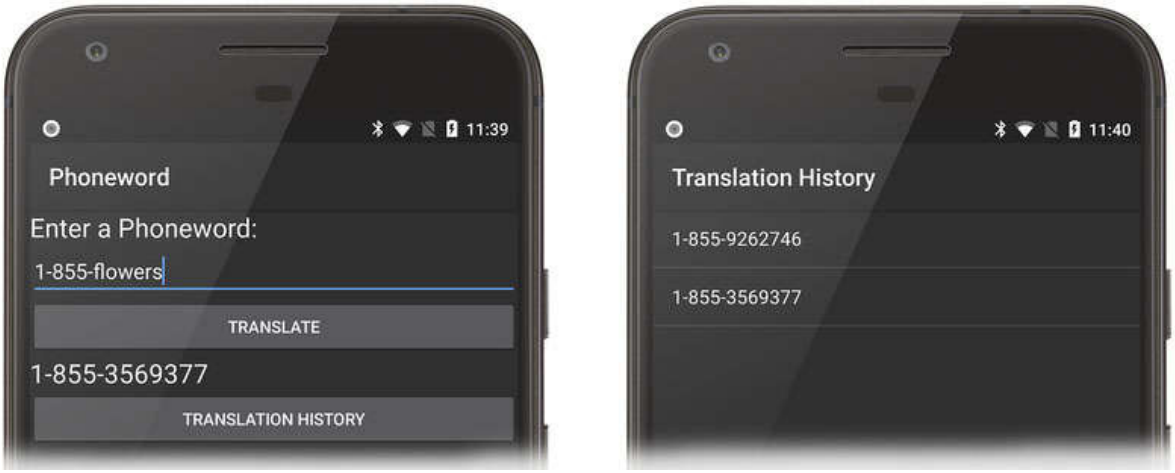
- Visual Studio – Direktno u ovom IDE-u možemo kreirati jednostavne setove ikonica za našu aplikaciju.

- Android Asset Studio – Web-baziran, i izvršava se u Internet browser-u, generator svih tipova Android ikonica, sa linkovima ka drugim korisnim alatima.
- Glyphish – Kvalitetni unapred napravljeni setovi ikonica za besplatan download ili kupovinu.
- Fiverr – Nudi nam izbor dizajnera koji će dizajnirati set ikonica za nas, po različitim cenama, počev od 5$.

Više informacija o veličinama ikonica i zahtevima, možemo naći u Android Resources vodiču.

4.3. Više-ekranska (multiscreen) aplikacija

Kao što smo već pominjali, aplikacija može imati i više od jednog Activity-ja, više ekrana. U ovom poglavlju ćemo proširiti našu *TelefonskaRec* aplikaciju tako da ima i drugi ekran, koji će čuvati istoriju brojeva prevedenih ovom aplikacijom.

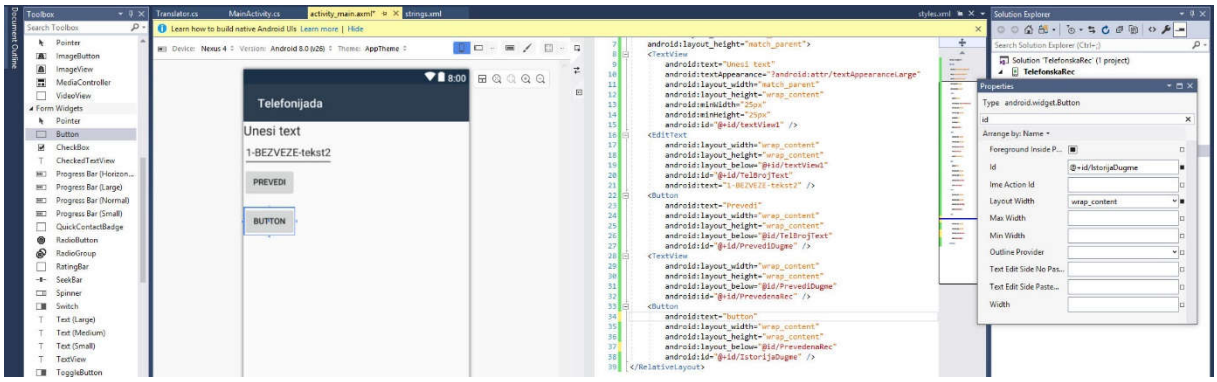


4.3.1. Nadogradnja layout-a aplikacije

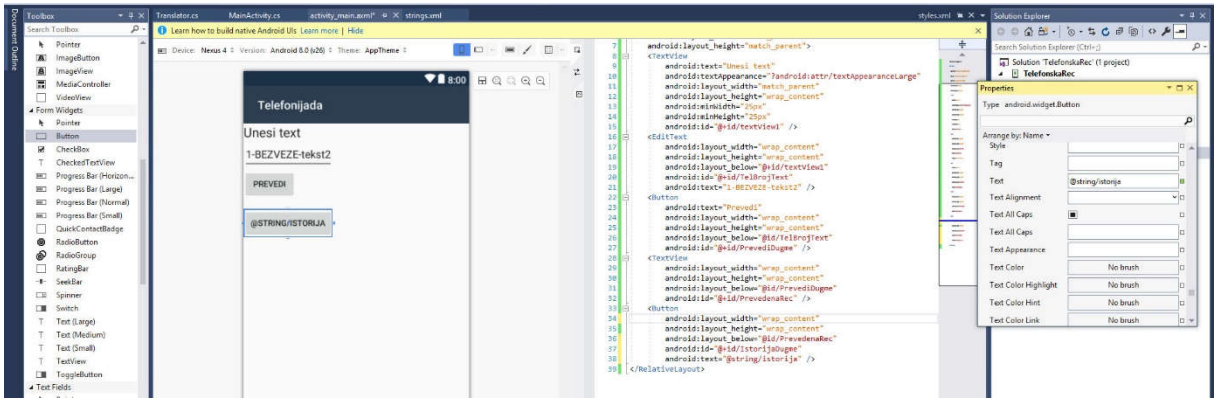
Ako već nismo, otvaramo *TelefonskaRec* aplikaciju u Visual Studiju i otvaramo duplim klikom fajl *activity_main.axml* iz *Solution Explorera*.

Iz *Toolbox*-a prevlačimo **Button** na dizajnersku površ i smeštamo ga odmah ispod *PrevedenaRec* **TextView**-a. U *Properties* panelu menjamo **Id** dugmeta na *@+id/IstorijaDugme*.

Primetimo, ako nismo do sad, da kada kreiramo nov **Id**, između znaka **@** i nastavka **id/imeKontrole** postoji znak **+**, kao u primeru *@+id/IstorijaDugme*, a kada pozivamo postojeći **Id** neke kontrole, znaka **+** nema, stoji samo *@id/IstorijaDugme*.



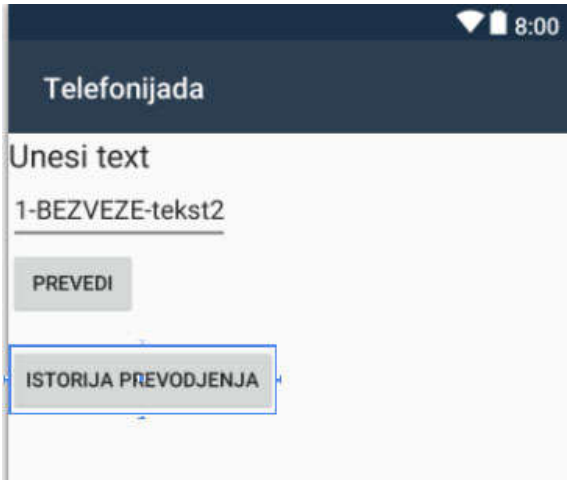
Zatim, setujemo **Text** svojstvo dugmeta na *@string/istorija*. Android dizajner će tekst u početku ispisati bukvalno baš tako, jer ne postoji resurs sa tim imenom, ali ćemo kasnije napraviti par izmena kako bi se sve korektno prikazalo.



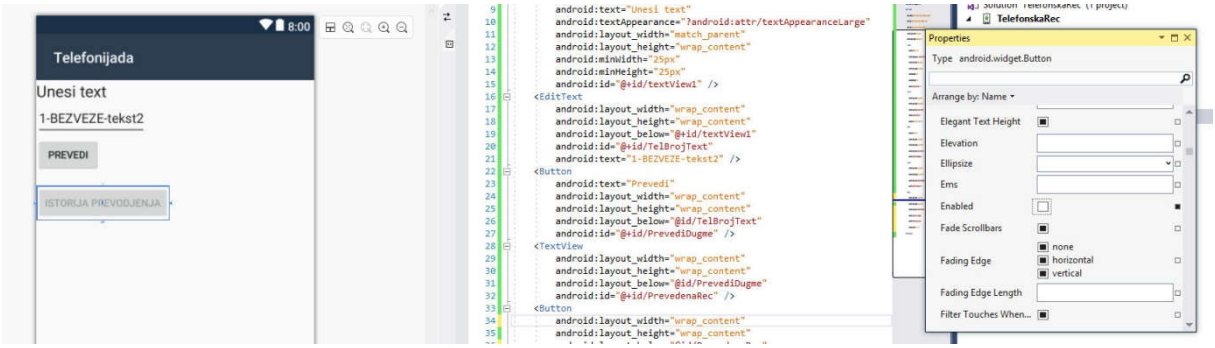
U *Solution Explorer*-u duplim klikom otvaramo **Resources** / **values** / **Strings.XML** resurs fajl. U njemu dodajemo *string* imena **istorija** i njegovu vrednost, pa ga sačuvamo, kao što je prikazano u listingu:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="istorija">Istorija prevodjenja</string>
    <string name="ApplicationName">TelefonskaRec</string>
</resources>
```

Sada sa prepoznavanjem novog resursa, **Istorija Prevodjenja** dugme menja svojstvo *Text* u skladu sa novom vrednošću stringa.

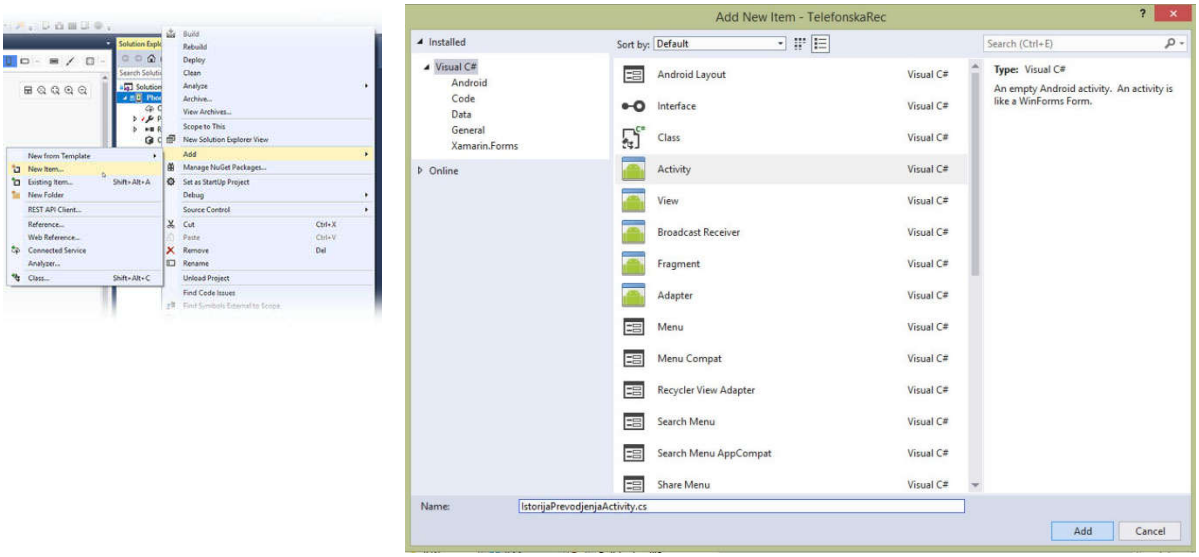


Sa izabranim dugmetom **Istorija Prevodjenja** na dizajnerskoj površi, tražimo **enabled** svojstvo u *Properties* panelu i setujemo njenu vrednost na **false,** ili de-čekiramo kako bi onesposobili dugme, što će prouzrokovati da se dugme zatamni u dizajnerskoj površi.



4.3.2. Kreiranje nove Aktivnosti

Kreiramo novu aktivnost koja će pokretati drugi ekran. U *Solution Explorer*-u desni klik na *TelefonskaRec* projekat otvara meni u kome biramo **Add > New Item...**



U **Add New Item** dialogu, biramo **Visual C# > Activity** (ili **Visual C# > Android > Activity** ako nam je potrebno da smanjimo pretragu samo na Android stavke), imenujemo Activity fajl u **IstorijaPrevodjenjaActivity.cs**, i naravno dugme **Add**.

Zamenjujemo šablonski kod u fajlu **IstorijaPrevodjenjaActivity.cs** sledećim:

```
using System;
using System.Collections.Generic;
using Android.App;
using Android.OS;
using Android.Widget;

namespace TelefonskaRec
{
    [Activity(Label = "@string/translationHistory", Theme = "@style/AppTheme")]
    public class IstorijaPrevodjenjaActivity: ListActivity
    {
        protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);
            //  preuzimanje liste brojava iz atacmenta Namere
            var telBrojevi = Intent.Extras.GetStringArrayList("phone_numbers") ?? new string[0];
            //  prikazivanje liste brojeva pomoću listView-a
            this.ListAdapter = new ArrayAdapter<string>(this,
                                    Android.Resource.Layout.SimpleListItem1, telBrojevi);
        }
    }
}
```

U ovom fajlu, kreirali smo klasu izvedenu iz ListActivity klase i popunili je programski, tako da ne mora da se pravi poseban layout sa dizajnom za ovu Aktivnost, jer će ceo Activity, odnosno ekran, biti jedna lista. Kasnije ćemo ovaj kod detaljnije objasniti.

Naravno, kao i uvek, fajl snimimo, pa možemo i da zatvorimo. Primetimo ako već nismo, snimanje se pored navedenih načina (iz menija, prečicom sa tastature) može obaviti i klikom na odgovarajuću ikonicu u *Standard Toolbar*-u koji se nalazi ispod menija.

4.3.3. Kod za istoriju prevođenja

Ovaj deo aplikacije skuplja telefonske brojeve koje je korisnik preveo na prvom ekranu, i prosleđuje ih drugom ekranu. Brojevi se skladište kao lista stringova. Da bi radili sa tim listama (kao i *Namerama* (Intents) koje ćemo koristiti kasnije) moramo uključiti još dve biblioteke, pomoću *using* direktiva koje dodajemo na početak koda, tj. vrh fajla **MainActivity.cs.**

```
using System.Collections.Generic;
using Android.Content;
```

Sledeći korak, kreiramo praznu listu koja će biti popunjena telefonskim brojevima, na početku *MainActivity* klase, kao u listingu:

```
[Activity(Label = "@string/app_name", Theme = "@style/AppTheme", MainLauncher = true)]
public class MainActivity : AppCompatActivity
{
    static readonly List<string> telBrojevi = new List<string>();
    ...// OnCreate(), i ostalo.
}
```

Zatim u *MainActivity* klasu dodajemo kod za referenciranje na **dugmeIstorije** iz interfejsa, u nastavku prethodnih referenci, tj. odmah posle deklarisanja *dugmePrevedi*.

```
Button dugmeIstorije = FindViewById<Button> (Resource.Id.IstorijaDugme);
```

Na kraju *OnCreate()* metode dodajemo kod za akciju **dugmeIstorije**, koji nećemo komentarisati u ovom trenutku.

```
            dugmeIstorije.Click += (sender, e) =>
            {
                //  kreiramo Nameru za slanje TranslationHistory Activity-u
                var namera = new Intent(this, typeof(IstorijaPrevodjenjaActivity));
                //  priključujemo Nameri listu telefonskih brojeva
                namera.PutStringArrayListExtra("phone_numbers", telBrojevi);
                //  pokretanje Aktivnosti
                StartActivity(namera);
            };
```

Nadograđujemo i kod za **dugmePrevedi**, da dodaje telefonski broj na listu **telBrojevi**, unutar *Click* handler-a za **dugmePrevedi**, tako da kod unutar njegovog *else* bloka sada izgleda ovako:

```
                else
                {
                    //  prikazujemo prevedeni broj
                    prevedenaRec.Text = prevedenBroj;
                    //  dodajemo ovaj broj na listu telefonskih brojeva
                    telBrojevi.Add(prevedenBroj);
                    //  omogućavamo da dugme postane aktivno
                    dugmeIstorije.Enabled = true;
                }
```

Snimamo i **Build**-ujemo aplikaciju da bi bili sigurni da nema grešaka.

Na kraju možemo da pokrenemo aplikaciju i testiramo njen rad.

4.4. Detaljnije objašnjenje ovde korišćenih koncepata

Pokušaćemo detaljnije da objasnimo Android navigaciju i arhitekturu, kako bi mogli kreirati kompleksnije aplikacije. Pojasnićemo koncepte Aplikacijskih Blokova (*Application Building Blocks*), Android navigacije Namerama (*Intents*), i istraživaćemo dodatne opcije Android hardverske navigacije. Analiziraćemo nov kod u *TelefonskaRec* aplikaciji i razviti bolji pogled na odnose između aplikacije i operativnog sistema, i drugih aplikacija.

4.4.1. Android aplikacijski blokovi

Android aplikacija se sastoji od kolekcije specijalnih Android klasa zvanih Aplikacijski Blokovi (*Application Blocks*) upakovanih zajedno sa resursima aplikacije, slikama, temama, pomoćnim klasama, itd. koji su koordinisani XML fajlom zvanim *Android Manifest*.

Aplikacijski blokovi čine kičmu Android aplikacije jer nam omogućavaju da uradimo stvari koje ne bismo normalno postigli regularnim klasama. Dve najvažnije su *Activities* i *Services*:

- **Activity** – Aktivnost odgovara ekranu sa korisničkim interfejsom konceptualno je slična veb stranici u veb aplikaciji. Na primer u newsfeed aplikaciji, login ekran bi bio prva Aktivnost, skrolujuća lista vesti bi bila druga Aktivnost, detaljna stranica za svaku vest bi bila treća Aktivnost. Više o Aktivnostima možemo naći u vodiču Activity Lifecycle.
- **Service** – Android Servisi podržavaju Aktivnosti preuzimajući dugotrajne procese i izvršavajući ih u pozadini. Servisi nemaju korisnički interfejs, i koriste se za zadatke koji nisu vezani za ekran, npr. puštanje pesme u pozadini ili upload-ovanje slika na server. Za više informacija videti vodiče Creating Services i Android Services.

Android aplikacija ne mora koristiti sve tipove Blokova, a često koriste više blokova istog tipa. Npr. *TelefonskaRec* aplikacija se sastoji iz dve Aktivnosti (ekrana) i nekoliko resurs fajlova. Jednostavan muzički plejer bi mogao imati nekoliko Aktivnosti i Servis za puštanje muzike kada je aplikacija u pozadini.

4.4.2. Namere (Intents)

Još jedan fundamentalni koncept u Android aplikacijama je Namera (Intent). Android je dizajniran oko principa najmanjih privilegija, aplikacija ima pristup samo Blokovima potrebnim za rad, a oni imaju limitiran pristup Blokovima koji čine operativni sistem ili druge aplikacije. Slično tome, Blokovi su labavo spojeni, dizajnirani su da imaju malo znanja i ograničen pristup drugim Blokovima, čak i kada su deo iste aplikacije.

Da bi komunicirali, Aplikacijski Blokovi šalju asinhrone pouke koje se zovu Namere napred i nazad međusobno. Namera sadrži informaciju o Bloku koji je prima i ponekad neke podatke. Namera poslata iz jedne komponente aplikacije okida neko dešavanje u drugoj komponenti aplikacije, povezujući te dve komponente i omogućava im komunikaciju. Šaljući Namere napred i nazad, navodimo dva Bloka da koordiniraju kompleksne akcije kao što je na primer pokretanje aplikacije kamere da slika i snimi, sakupljanje lokacijskih podataka, ili navigacija iz jednog ekrana u drugi.

Android Manifest čuva informacije o svim Aplikacijskim Blokovima u jednoj aplikaciji, o zahtevima, dozvolama, povezanim bibliotekama, svemu što je potrebno operativnom sistemu da bi znao kako da izvrši aplikaciju. **Android Manifest** takođe radi i sa Aktivnostima i Namerama da bi kontrolisao koje su akcije odgovarajuće za datu Aktivnost. Ove napredne odlike Android Manifesta su pokrivene vodičem Working with the Android Manifest.

4.4.3. Android navigacija

U jedno-ekranskoj verziji *TelefonskaRec* aplikacije, korišćene su samo jedna Aktivnost, jedna Namera, i Android Manifest uz još poneki resurs kao što su ikonice. U multi-ekranskoj verziji, dodata je još jedna Aktivnost, koja se pokreće iz prve Aktivnosti korišćenjem Namere.

Najpre je kreirana Namera, prosleđen joj je trenutni Kontekst (*this*, koji se odnosi na trenutni Kontekst) i tip Aplikacijskog Bloka koji tražimo (**IstorijaPrevodjenjaActivity**):

```
var namera = new Intent(this, typeof(IstorijaPrevodjenjaActivity));
```

Kontekst (*Context*) je interfejs za globalnu informaciju o aplikacijskom okruženju, on omogućava novo-kreiranom objektu da zna šta se dešava s aplikacijom. Ako Nameru posmatramo kao poruku, obezbeđujemo joj ime primaoca (*IstorijaPrevodjenjaActivity*) i primaočevu adresu (*Context*).

Android obezbeđuje opciju priključenja jednostavnog podatka Nameri (kompleksnijim podacima se drugačije barata). U našem primeru **PutStringArrayExtra()** metoda se koristi da se Nameri priključi lista telefonskih brojeva, i **StartActivity()** se poziva za primaoca Namere, kao što se vidi u listingu EventHeandler-a za klik na dugme *dugmeIstorije:*

```
            dugmeIstorije.Click += (sender, e) =>
            {
                //  kreiramo Nameru za slanje IstorijaPrevodjenjaActivity-u
                var namera = new Intent(this, typeof(IstorijaPrevodjenjaActivity));
                //  priključujemo Nameri listu telefonskih brojeva
                namera.PutStringArrayListExtra("phone_numbers", telBrojevi);
                //  pokretanje Aktivnosti
                StartActivity(namera);
            };
```

4.4.4. Dodatni koncepti predstavljeni u aplikaciji

Navešćemo ostale koncepte koje smo uveli u TelefonskaRec aplikaciji a koje nismo do sad pojasnili.

String Resources – U dizajneru, svojstvo *Text* kontrole *IstorijaDugme* smo setovali na *"@string/istorija* ". Sintaksa *@string* znači da je vrednost stringa smeštena u string resurs fajl, **Strings.xml**. Zbog toga smo u taj fajl morali dodati vrednost stringa *istorija*:

```
<resources>
    <string name="istorija">Istorija Prevođenja</string>
    <string name="app_name">Telefonijada</string>
    <string name="action_settings">Settings</string>
</resources>
```

Za više informacija o string resursima a i ostalim Android resursima, konsultovati Android Resources guide.

ListView i **ArrayAdapter** – *ListView* je komponenta korisničkog interfejsa koja omogućava jednostavan način prikazivanja skrolujućih listi redova podataka. *ListView* instanca zahteva *Adapter* koji će je popuniti sa podacima prikazanim u redovima. Bio nam je dovoljan jedan red koda da bi smo u korisničkom interfejsu *IstorijaPrevodjenjaActivity*-ja prikazali listu telefonskih brojeva, i to tako što smo svojstvu *ListView*-a ***ListAdapter*** dodeli novi adapter, koji je prilagodio u matricu listu telefonskih brojeva **telBrojevi.**

```
this.ListAdapter = new ArrayAdapter<string>(this,
                   Android.Resource.Layout.SimpleListItem1, telBrojevi);
```

Listu brojeva **telBrojevi** smo prethodno preuzeli iz Namere, (Intent-a).

```
var telBrojevi = Intent.Extras.GetStringArrayList("phone_numbers") ?? new string[0];
```

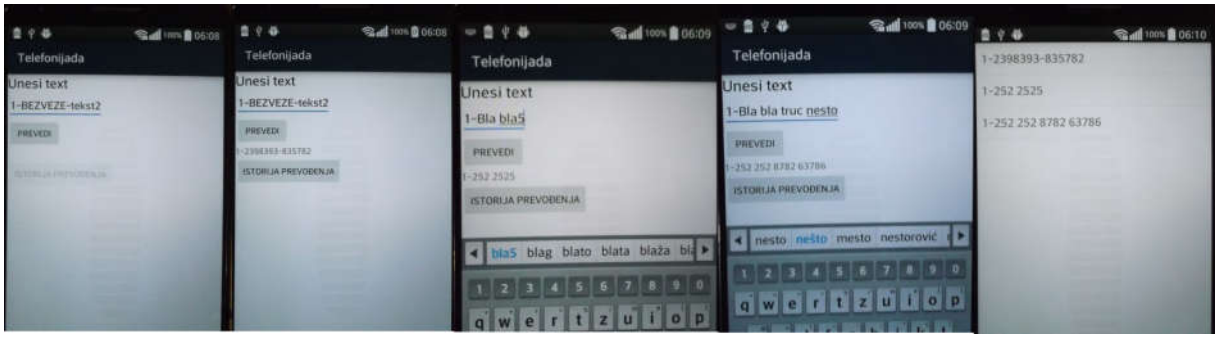
ListViews i Adapters su opširna tema, pokrivena detaljno ListViews and Adapters vodičem. Takođe, Populating a ListView With Data vodič se bavi specifično upotrebom ugrađenih *ListActivity* i *ArrayAdapter* klasama za kreiranje i popunjavanje ListView bez definisanja posebnih layout-ova, kao što je urađeno u TelefonskaRec aplikaciji.

4.4.5. Listing kompletnog koda Aktivnosti MainActivity

Kompletan kod **MainActivity.cs** fajla proširene verzije aplikacije TelefonskaRec je dat u listingu:

```
using System.Collections.Generic;
using Android.Content;
using Android.App;
using Android.OS;
using Android.Support.V7.App;
using Android.Runtime;
using Android.Widget;

namespace TelefonskaRec
{
    [Activity(Label = "@string/app_name", Theme = "@style/AppTheme", MainLauncher = true)]
    public class MainActivity : AppCompatActivity
    {
        //  kreiramo listu telefonskih brojeva
       static readonly List<string> telBrojevi = new List<string>();
        //  metoda koja se poziva pri kreiranju Aktivnosti
        protected override void OnCreate(Bundle savedInstanceState)
        {
            base.OnCreate(savedInstanceState);
            //  setujemo prikaz iz "activity_main" layout resursa
            SetContentView(Resource.Layout.activity_main);
            //  preuzimamo UI kontrole iz učitanog layouta
            EditText telBrojTekst = FindViewById<EditText>(Resource.Id.TelBrojText);
            TextView prevedenaRec = FindViewById<TextView>(Resource.Id.PrevedenaRec);
            Button dugmePrevedi = FindViewById<Button>(Resource.Id.PrevediDugme);
            Button dugmeIstorije = FindViewById<Button>(Resource.Id.IstorijaDugme);
            //  kod za dugmePrevedi
            dugmePrevedi.Click += (sender, e) =>
            {
                //  prevodimo korisnikov alphanumericki telefonski broj u numericki
                string prevedenBroj = Core.Prevodilac.UBroj(telBrojTekst.Text);
                if (string.IsNullOrWhiteSpace(prevedenBroj))
                {
                    prevedenaRec.Text = string.Empty;
                }
                else
                {
                    //  prikazujemo prevedeni broj
                    prevedenaRec.Text = prevedenBroj;
                    //  dodajemo ovaj broj na listu telefonskih brojeva
                    telBrojevi.Add(prevedenBroj);
                    //  omogućavamo da dugme postane aktivno
                    dugmeIstorije.Enabled = true;
                }
            };
            //  kod za dugmeIstorije dugme
            dugmeIstorije.Click += (sender, e) =>
            {
                //  kreiramo Nameru za slanje IstorijaPrevodjenjaActivity-u
                var namera = new Intent(this, typeof(IstorijaPrevodjenjaActivity));
                //  prikljućujemo Nameri listu telefonskih brojeva
                namera.PutStringArrayListExtra("phone_numbers", telBrojevi);
                //  pokretanje Aktivnosti
                StartActivity(namera);
            };
        }
    }
}
```

Slike aplikacije u izvršavanju na Android telefonu:



5. Primeri Android aplikacija

Daćemo u nastavku par primera android aplikacija, koje nećemo toliko detaljno obrađivati kao na početku, ali su tu ipak uz par komentara da demonstriraju rad Xamarin platforme kao i programiranje za Android.

5.1. Kalkulator

Kalkulator (digitron) je jedna od najčešće kreiranih aplikacija kada je potrebno demonstrirati rad na nekoj platformi, zbog svoje jednostavnosti ali ipak nesumnjive funkcionalnosti.

Kompletan tutorijal za izradu ove aplikacije možete naći na YouToube adresama:

1. https://youtu.be/LgNXUicjUOA - izrada korisničkog interfejsa
2. https://youtu.be/91WA8N0SppI - kodiranje kalkulatorske logike

Kreiramo nov Android projekat, dajemo mu ime, biramo praznu aplikaciju i minimalnu verziju Androida.

5.1.1. Dizajn (layout) kalkulatora

Otvaramo *activity_main.axml* fajl u *Resourses / layout* folderu u Solution Exploreru. I XML pogledu na dizajn layout-a, menjamo zaglavlje, XML tag **RelativeLayout** u **LinearLayout** kako bi radili sa linearnim layout-om, gde komponente nisu vezane svaka s prethodnom već imaju linearan raspored, i dodajemo svojstvo *android:orientation="vertical"* tako da XML kod sada izgleda ovako:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
</LinearLayout>
```

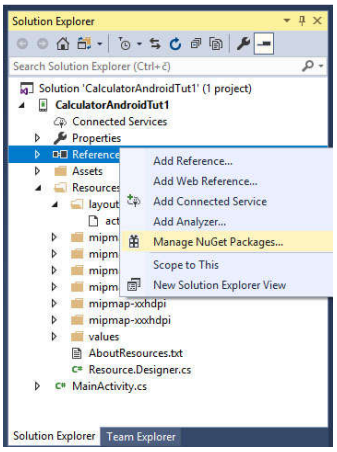
Želimo da imamo prikaz teksta u koji će ići vrednosti koje se unose i rezultata računa, koji može da se skroluje ukoliko je tekst predugačak da stane na ekran. Zato u *layout* dodajemo novu kontrolu, *HorizontalScrollView*, unutar koje će biti tekstualni prikaz. Ostavljamo njena svojstva with na match parent, kako bi zauzela celu površinu layout-a, height na 0, jer ne želimo da kontrola bude vidljiva, da ima visinu, i weight 1, koji je atribut koji koristi *LinearLayout* kao koreni element, što znači da *HorizontalScrollView* zauzima isti prostor kao i layout.

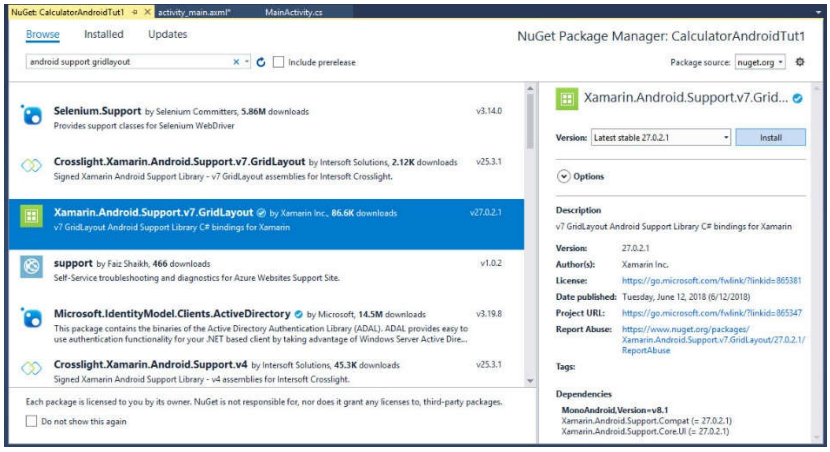
Sada dodajemo kontrolu *TextView*, i primećujemo da je ona unutar *HorizontalScrollView* kontrole. Menjamo njeno svojstvo *Id* na *“calculator_text_view“*. Menjamo njenu širinu na *"wrap_content"*. Menjamo svojstvo *Padding* na vrednost

10dp, što nam govori da će tekst u *TextView* kontroli imati razmak od ivica 10 piksela. Postavljamo *TextSize* na *50sp*, jer želimo da tekst bude krupniji, a *sp* (scalable dependent pixel) jedinica veličine se razlikuje od *dp* (density dependent pixel) jer dozvoljava skaliranje teksta ako dođe do promena na ekranu, tipa promene rezolucije, dok će veličine definisane u *dp* sačuvati fiksan broj piksela, bez obzira na promene. *Text* property smo promenili na *"123"* samo radi vidljivosti na dizajn površi. Kod koji smo dodali izgleda ovako:

```
<HorizontalScrollView
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_weight="1"
      android:minWidth="25px"
      android:minHeight="25px">
      <TextView
            android:text="123"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:id="@+id/calculator_text_view"
            android:padding="10dp"
            android:textSize="50sp" />
</HorizontalScrollView>
```

Da bi napravili šemu tastera kalkulatora, od koristi će nam biti kontrola *GridLayout*. Ali da bi ona radila i na starijim uređajima, potrebno je da uključimo podršku za nju. Da bi dodali *support* biblioteke, potrebno je da desnim klikom kliknemo na *References* u *Solution Explorer-u* i izaberemo *Manage NuGet Packages*.





U ovom dijalogu kliknemo na dugme Browse, i u liniji za pretragu kucamo *android support gridlayout*, što će nam suziti pretragu na pakete s tim imenom. Biramo onaj s potpisom Xamarin-a, i pritiskamo dugme Install. Naravno, prihvatimo uslove licence.

Sada nam je potrebno da u našem layoutu napravimo novi *namespace*, što ćemo uraditi odmah ispod postojećih u XML-u. Oznaka za *namespace* u XML-u **xmlns**. Dodaćemo: `xmlns:app=http://schemas.android.com/apk/res-auto`. Međutim, obratimo pažnju, ako smo na početku pri definisanju projekta stavili minimalnu verziju androida ispod 5.0, ovaj *namespace* je najverovatnije već dodat. Ovaj *namespace* sadrži klase

koje su nam potrebne za rad sa *GridLayout*-om na starijim Android operativnim sistemima.

Ovaj put ne možemo *GridLayout* kontrolu dodati iz *Toolboxa* već ćemo njenu kompatibilnu verziju ručno dodatu u XML kodu, ispod zatvorenog taga *HorizontalScrollView*. Dodajemo XML kod koji treba da izgleda ovako:

```
    <android.support.v7.widget.GridLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="4"
        app:orientation="horizontal"
        app:rowCount="5"
        app:columnCount="4">
    </android.support.v7.widget.GridLayout>
```

Dakle podešavanja su slična kao za *HorizontalScrollView* kontrolu, sem što *layout_weight* dobija vrednost 4. Takođe, dodali smo `app:orientation="horizontal"` svojstvo, koje kao što vidimo nije svojstvo iz Android *namespace*, već iz *namespace* aplikacije, koji smo prethodno pravili, a kome naš *GridLayout* pripada. Takođe, setovali smo mu svojstva za broj redova i broj kolona.

Ono što nam je potrebno u digitronu za interakciju jesu dugmad. Dodaćemo za početak jedno, direktno u XML-u, unutar *GridLayout*-a. Visinu i širinu dugmeta nećemo podešavati, jer ćemo i ovde koristiti svojstvo *weight* kojim ćemo definisati razmere dugmeta. Međutim, za razliku od samog layout-a, ovde želimo da se širina i visina dugmeta menjaju dinamično. Opet za razliku od layout-a ovde i kolone i redovi imaju svoj *weight*, pa ih oba postavljamo na 1. Podešavamo veličinu teksta na 25sp.

Na kraju dodajemo i ime metoda koji će biti korišćen kada se dugme pritisne, što ćemo programirati kasnije, za sada ga samo imenujemo `android:onClick="ButtonClick"`. To bi izgledalo ovako:

```
            <Button
                  android:layout_with="0dp"
                  android:layout_height="0dp"
                  app:layout_rowWight="1"
                  app:layout_columnWight="1"
                  android:textSize="25sp"
                  android:onClick="ButtonClick"
            />
```

Sada kada imamo jedno dugme, možemo ga jednostavno iskopirati 20-ak puta. Ali ukoliko bi želeli nešto da promenimo u dizajnu dugmeta, to bi značilo da 20 puta menjamo to isto. Zato je bolje da napravimo poseban Stil (Style) dugmeta koji će se primenjivati na svu dugmad.

Nov *Style* dodajemo tako što otvaramo u *Solution Explorer-u Resources / values*, desni klik, *Add New Item*, otvorimo nov dokument i nazovemo ga po želji, mada je *styles.xml* uobičajeno ime. U novijim verzijama Xamarina, taj fajl već kreiran unapred pa ga samo otvorimo. Ukoliko pravimo novi fajl, u njemu dodajemo *Resources* element, pa unutra njega pravimo nov *Stye* tag. Ukoliko fajl već postoji, samo dodajemo unutar Resources elementa, posle postojećeg stila. Kreiramo *style name*, i kopiramo u njega

sadržaj vezan za Button iz dizajnerskog axml fajla. Naravno, to ne može tako da radi pa ćemo ga prilagoditi.

Dakle kreiramo posebne *item* elemente i unutar njih smeštamo pojedinačna svojstva. Primetimo da kada su svojstva vezana za Android, u *item*-u stoji *android:* a kada je svojstvo vezano za aplikaciju, nema naziva *namespec*-a već se direktno navodi svojstvo. Na kraju, sadržaj **style.xml** fajla treba da izgleda ovako:

```
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>
  <style name="button_calculator">
    <item name="android:layout_width">0dp</item>
    <item name="android:layout_height">0dp</item>
    <item name="layout_rowWeight">1</item>
    <item name="layout_columnWeight">1</item>
    <item name="android:textSize">25dp</item>
    <item name="android:onClick">ButtonClick</item>
  </style>
```

Sada se vraćamo u dizajn, axml fajl, i brišemo sve što smo podešavali za Button. Sve što nam sada treba u okviru Button taga je svojstvo *Style* kome dodeljujemo ime stila koji smo kreirali u *style.xml* fajlu `style="@style/button_calculator"` i svojstvo *Text* u kome ispisujemo tekst koji će pisati na dugmetu, u ovom slučaju DEL. Njemu dodajemo svojtvo `app:layout_columnSpan="4"` koje govori da će se to dugme razvući preko četiri kolone.

Kopiramo dugme, brišemo mu *columnSpan* svojstvo, jer ovo dugme treba a bude samo u jednoj ćeliji grid-a, i menjamo *Tekst* u vrednost “7“. Ovo kopiramo za svu dugmad, menjajući samo tekst.

Fajl **activity_main.axml** treba da izgleda ovako:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <HorizontalScrollView
              android:layout_width="match_parent"
              android:layout_height="wrap_content"
              android:layout_weight="1"
              android:minWidth="25px"
              android:minHeight="25px">
              <TextView
                     android:text="123"
                     android:layout_width="wrap_content"
```

```
                    android:layout_height="match_parent"
                    android:id="@+id/calculator_text_view"
                    android:padding="10dp"
                    android:textSize="50sp" />
    </HorizontalScrollView>
    <android.support.v7.widget.GridLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="4"
        app:orientation="horizontal"
        app:rowCount="5"
        app:columnCount="4">
        <Button
            style="@style/button_calculator"
            app:layout_columnSpan="4"
            android:text="DEL" />
        <Button
            style="@style/button_calculator"
            android:text="7" />
        <Button
            style="@style/button_calculator"
            android:text="8" />
        <Button
            style="@style/button_calculator"
            android:text="9" />
        <Button
            style="@style/button_calculator"
            android:text="÷" />
        <Button
            style="@style/button_calculator"
            android:text="4" />
        <Button
            style="@style/button_calculator"
            android:text="5" />
        <Button
            style="@style/button_calculator"
            android:text="6" />
        <Button
            style="@style/button_calculator"
            android:text="×" />
        <Button
            style="@style/button_calculator"
            android:text="1" />
        <Button
            style="@style/button_calculator"
            android:text="2" />
        <Button
            style="@style/button_calculator"
            android:text="3" />
        <Button
            style="@style/button_calculator"
            android:text="-" />
        <Button
            style="@style/button_calculator"
            android:text="." />
        <Button
            style="@style/button_calculator"
            android:text="0" />
        <Button
            style="@style/button_calculator"
            android:text="=" />
        <Button
            style="@style/button_calculator"
            android:text="+" />
    </android.support.v7.widget.GridLayout>
</LinearLayout>
```

5.1.2. Kod kalkulatora

Otvaramo *MainActivity.cs* fajl.

Deklarišemo promenljive koje ćemo koristiti u ovoj Aktivnosti, unutar klase *MainActivity* na samom početku.

Prva je *TextView* **calculatorText** koja će prikazivati unete brojeve i rezultate, ne više od dva broja istovremeno, pošto je ovo jednostavan kalkulator.

Deklarisaćemo i inicijalizirati matricu string tipa **numbers** koja će čuvati unete brojeve, dimenzije 2 polja.

Deklarisaćemo i string koji će čuvati trenutno aktivirani operator, koji ćemo nazvati **@operator**. Simbol @ je dodat zato sto je reč operator ključna reč u C#.

Unutar metode *OnCreate()* povezujemo naš **calculatorText** *TextView* sa kontrolom interfejsa *TextView* preko njenog *Id*-a.

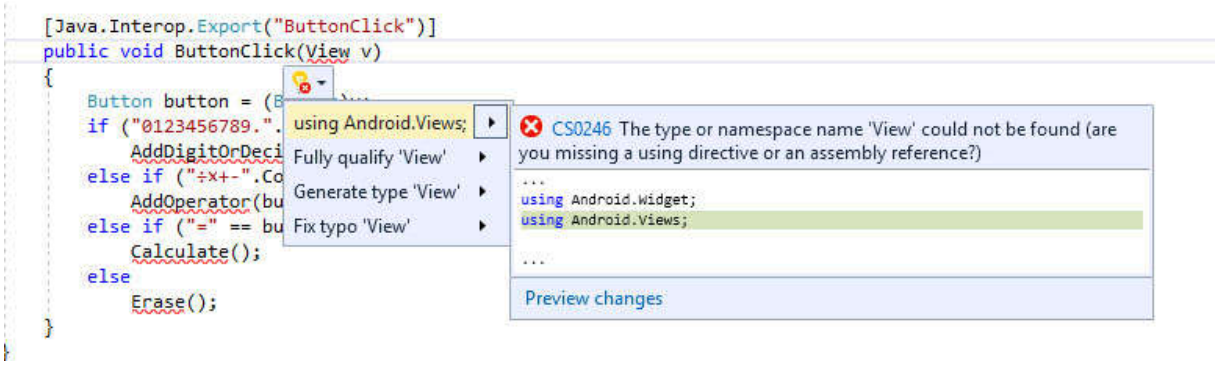
Kod do sada u *MainActivity.cs* fajlu izgleda ovako:

```
using Android.App;
using Android.OS;
using Android.Support.V7.App;
using Android.Runtime;
using Android.Widget;

namespace CalculatorAndroidTut1
{
    [Activity(Label = "@string/app_name", Theme = "@style/AppTheme", MainLauncher = true)]
    public class MainActivity : AppCompatActivity
    {
        private TextView calculatorText;

        private string[] numbers = new string[2];
        private string @operator;

        protected override void OnCreate(Bundle savedInstanceState)
        {
            base.OnCreate(savedInstanceState);
            // Set our view from the "main" layout resource
            SetContentView(Resource.Layout.activity_main);

           calculatorText = FindViewById<TextView>(Resource.Id.calculator_text_view);
        }
    }
}
```

Pošto smo u dizajnu interfejsa naveli ime metode koja će se pozivati na klik dugmeta, sada implementiramo njen kod. Moramo je nazvati onako kako smo je nazvali u dizajneru, tj. fajlu *styles.xml.* Smeštamo je odmah posle metode *OnCreate()* (iako bi smo mogli da kreiramo poseban fajl sa kodom kao u slučaju *TelefonskaRec* aplikacije, ovaj put nećemo to raditi) i dodajemo sledeći kod u nju:

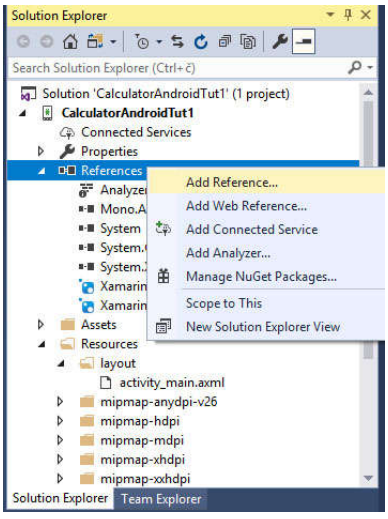
```
        [Java.Interop.Export("ButtonClick")]
        public void ButtonClick(View v)
        {
            Button button = (Button)v;
            if ("0123456789.".Contains(button.Text))
                AddDigitOrDecimalPoint(button.Text);
            else if ("÷x+-".Contains(button.Text))
                AddOperator(button.Text);
            else if ("=" == button.Text)
                Calculate();
            else
                Erase();
        }
```

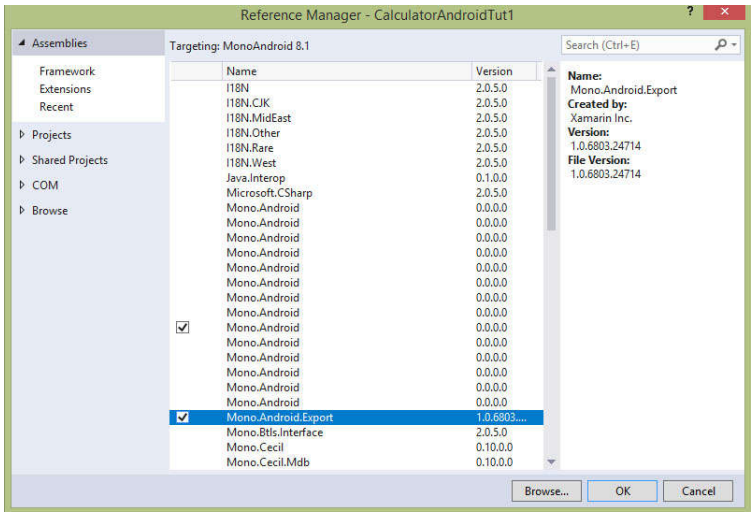
Funkcija ne vraća ništa, a prima samo jedan argument **v** tipa *View*. Visual Studio će nam izbaciti grešku, jer tip *View* ne postoji u dosadašnjim *namespace*-ovima koje koristimo, ali ako posle otkucanog View pritisnemo dugme CTRL, pojaviće se ispod View mali padajući meni iz koga biramo `using Android.Views` opciju koja nam automatski ubacuje tu instrukciju na početak fajla, čime se u naš projekat uključuje i taj *namespace*, pa *View* postaje prepoznat tip podataka.



Da bi funkcija **ButtonClick()** mogla da radi, moramo ispred nje dodati atribut koji je određuje u resursima kao akciju, i to je u ovom slučaju `[Java.Interop.Export("ButtonClick")]`.

Da bi radili sa Export funkcijom Jave, moramo dodati referencu na Mono.Expot.dll. Da bi to postigli, u *Solution Explorer*-u desnim klikom na References biramo stavku Add Refference, i u dijalogu biramo Mono.Android.Export.

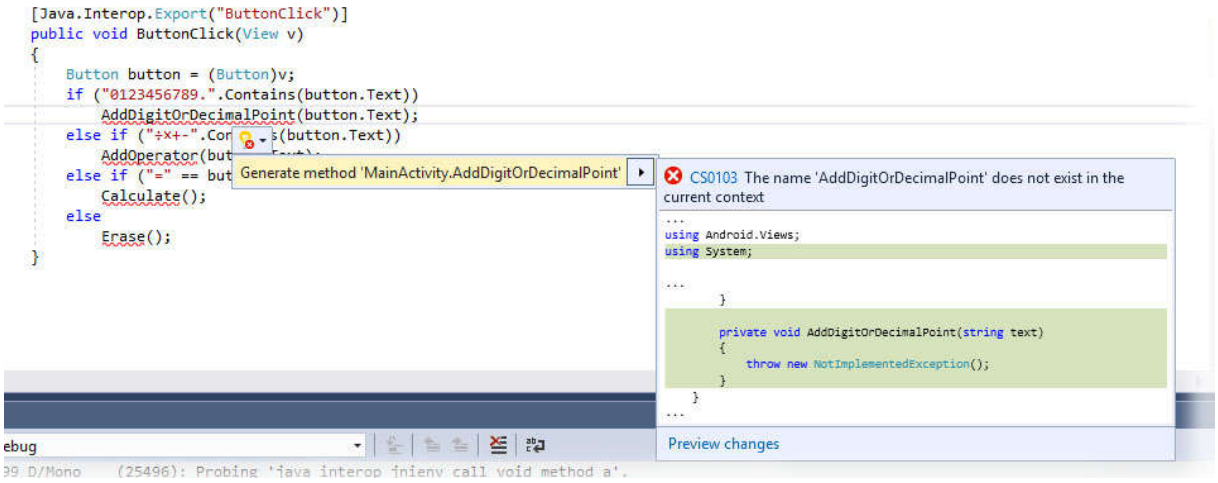




Treba nam da očitamo vrednost svojstva *Text* argumenta **V** koji primamo, ali on kao promenljiva tipa *View* nema to svojstvo, pa ga zato kastujemo u tip (*Button*), i tu vrednost dodeljujemo novodeklarisanoj promenljivoj *Button* tipa.

Radimo proveru da li se svojstvo *Text* objekta *Button* sadrži u stringu "0123456789.", tj. da li je pritisnuto dugme cifre ili decimalne tačke, i ako jeste taj tekst (cifru ili tačku) prosleđujemo kao argument funkciji **AddDigitOrDecimalPoint()**

Pošto funkcija **AddDigitOrDecimalPoint()** ne postoji, kao i malopre klikom na njeno ime i pritiskom na CTRL otvara se mali padajući meni gde nam VS nudi da kreira tu funkciju za nas, što i činimo klikom na taj izbor.



Ispod **ButtonClick()** funkcije je kreirana nova funkcija **AddDigitOrDecimalPoint**().

Pravimo novi uslov, u kome proveravamo da li se *button.Text* sadrži u stringu "÷×+-" i ako je uslov ispunjen pozivamo funkciju **AddOperator()**. Kako ni ta funkcija ne postoji, generišemo je na isti način kao u prethodnom slučaju.

Sledeća provera je da li je *button.Text* jednak stringu "=" i u tom slučaju pozivamo funkciju **Calculate**(). I nju generišemo na opisan način.

Poslednji slučaj je ako nije ispunjen ni jedan od prethodnih uslova, a ostalo je samo jedno dugme koje nismo obradili, pozivamo funkciju **Erase()**, i generišemo je.

U novim za nas generisanim funkcijama smo promenili ime argumenta iz **text** u **value** i obrisali nepotrebne *throw new NotImplementedException()* komande.

Dodajmo sledeći kod u **AddDigitOrDecimalPoint**() funkciju.

```
        private void AddDigitOrDecimalPoint(string value)
        {
            int index = @operator == null ? 0 : 1;

            if (value == "." && numbers[index].Contains("."))
                return;
            numbers[index] += value;

            UpdateCalculatorText();
        }
```

Funkcija će kao što joj ime i kaže, trenutno pritisnutu cifru ili tačku, a koju je funkcija primila kao argument, da doda u string koji će postati broj za računsku operaciju.

Pošto možemo imati dva broja upisana u kalkulator u jednom trenutku, imamo i dva člana niza **numbers[ ]** u koji treba da ih smestimo, i odmah na početku treba da odredimo u koji od ta dva člana niza smeštamo trenutnu cifru ili tačku, odnosno koji indeks za taj niz treba da izaberemo, 0 ili 1. Pošto su uneti brojevi razdvojeni operatorom, navedeno određivanje postižemo proverom da li je stisnut neki operator, ako nije, to jest vrednost promenljive **@operator** je *null*, onda smeštamo simbol u prvi broj, tj. indeks niza će biti 0, u suprotnom indeks je 1.

Sledeća provera je da li je trenutni **value** tačka, i ako se tačka već sadrži u broju, izlazimo iz funkcije bez daljeg izvršavanja koda, kako bi sprečili dupliranje decimalne tačke.

Ako već nismo izašli iz funkcije, **value** dodajemo na string **number[index]** i pozivamo funkciju **UpdateCalculatorText()**, a koja takođe još ne postoji pa je kreiramo. Kada je funkcija kreirana, možemo iz njenom poziva da se prebacimo u nju brzo pritiskom na taster **F12**, pa ćemo to i uraditi.

Sada kada smo kreirali funkciju **UpdateCalculatorText()**, koja će osvežavati prikaz unetim vrednostima ako postoje, pa hajde da odmah napišemo kod za nju.

```
private void UpdateCalculatorText() => calculatorText.Text = $"{numbers[0]} {@operator} {numbers[1]}";
```

Svojstvu *Text* promenljive **calculatorText** dodeljujemo string koji se sastoji od vrednosti *string* promenljivih **{numbers[0]} {@operator} {numbers[1]}**, sa pauzama između njih. Pošto je cela funkcija jedan izraz, od nje smo napravili *Expression* body, tako što smo umesto da izraz stavimo u blok unutar te funkcije, stavili u isti red sa imenom funkcije i razdvojili operatorom **=>** čime smo formirali jedan *Expression* (izraz).

Sad možemo da odradimo kod funkcije **AddOperator()**.

```
        private void AddOperator(string value)
        {
            if (numbers[1] != null)
            {
                Calculate(value);
                return;
            }

            @operator = value;

            UpdateCalculatorText();
        }
```

Kod je jednostavan, proveravamo da li je već unet drugi broj i ako jeste pozivamo funkciju **Calculate()** kojoj prosleđujemo string iz argumenta, **value**, i izlazimo iz funkcije. Ako nismo izašli iz funkcije izvršava se kod ispod, tj. promenljiva **@operator** dobija vrednost **value**, i pozivamo funkciju **UpdateCalculatorText(),** da bi osvežili prikaz na ekranu dodajući uneti operator.

Na ovaj način smo postigli da ukoliko već postoje uneta dva broja, pritiskom na novi taster operatora prvo sračunamo postojeći izraz, pa tek posle možemo na taj rezultat dodavati novi operator.

Primetimo da u ovom pozivu funkcije **Calculate()** prosleđujemo string **value** koji prikazuje operator, pa to moramo dodati u deklaraciji funkcije **Calculate()**, argument tipa string. Ali ako se secamo, pri pozivu iste funkcije **Calculate()** iz funkcije **ButtonClick(),** nismo prosledili nikakav argument funkciji **Calculate()**.

Najlakši način da argument postane opcionalan je da mu u okviru deklaracije odmah dodelimo default vrednost, *null*, koja će ostati ako ne primi nijednu drugu, tj. ako pri pozivu funkcije **Calculate()** ne prosledimo argument.

Unesimo sledeći kod u funkciju **Calculate()**:

```
private void Calculate(string newOperator = null)
{
    double? result = null;
    double? first = numbers[0] == null ? null : (double?)double.Parse(numbers[0]);
    double? second = numbers[1] == null ? null : (double?)double.Parse(numbers[1]);

    switch (@operator)
    {
        case "÷":
            result = first / second;
            break;
        case "x":
            result = first * second;
            break;
        case "+":
            result = first + second;
            break;
        case "-":
            result = first - second;
            break;
    }

    if (result != null)
    {
        numbers[0] = result.ToString();
        @operator = newOperator;
        numbers[1] = null;
        UpdateCalculatorText();
    }
}
```

Najpre deklarišemo promenljivu **result** tipa *double* koja će čuvati rezultat operacije, i dodeljujemo joj inicijalnu vrednost *null*, što nam je potrebno za kasnije. Pošto promenljive vrednosnih tipova kao što je *double* ne mogu da imaju *null* vrednost, učinićemo je *null*-abilnim *double* tipom, što postižemo dodavanjem znaka **?** uz *double*.

Slično radimo i za promenljivu **first** kojoj odmah i dodeljujemo vrednost pomoću sledećeg upita: proveravamo da li je vrednost stringa **numbers[0]** *null*, i ako jeste i **first** dobija vrednost *null*, a ako nije dobija vrednost stringa **numbers[0]** koji prethodno moramo da *Parse*-ujemo u tip *double*, a koji opet *Cast*-ujemo u *null*-abilni tip *double?* sve u istoj liniji.

Identičan postupak sprovodimo i za promenljivu **second**.

Nakon toga izračunavamo **result** zavisno od izabrane operacije, odnosno vrednosti promenljive **@operator**.

Ako je **result** različit od vrednosti *null*, što bi se desilo da je iz bilo kojih razloga da je **first** ili second bio **null**, želimo da ga prikažemo na ekranu našeg kalkulatora, što ćemo učiniti konvertovanjem **result**-a u string, i dodeljivanjem stringu **numbers[0]**. Promenljivoj @operator dodeljujemo vrednost argumenta funkcije **Calculate()**, ako ga je bilo (ako nije, primio je dafault vrednost *null*). *Null*-ujemo drugu vrednost, string **numbers[1]**. Na kraju pozivamo funkciju *UpdateCalculatorText()*, čime prikazujemo na ekranu rezultat izvršene operacije i uneti novi operator, ako postoji.

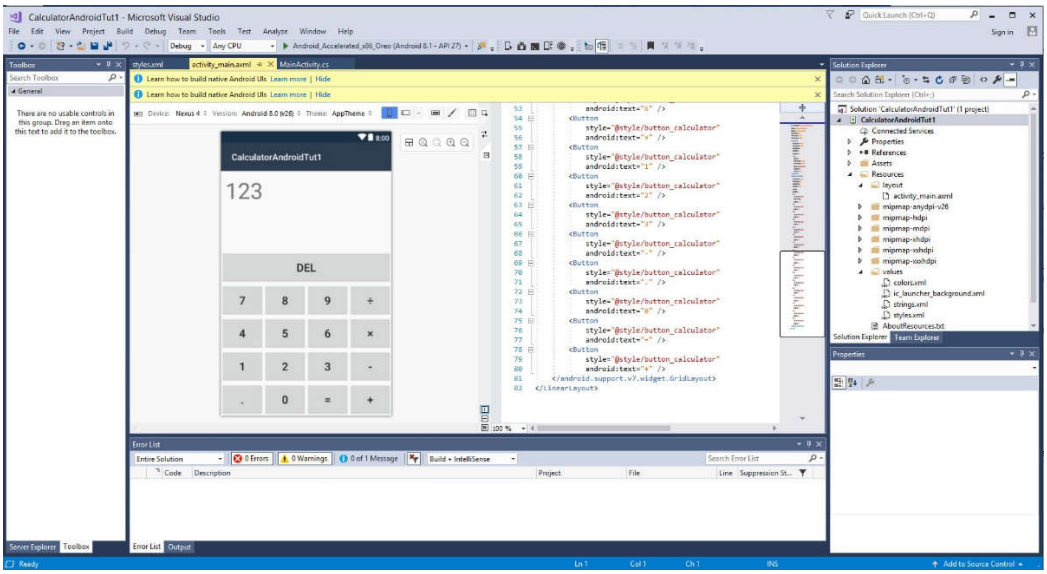
Ostalo nam je još jedno dugme, odnosno njegova funkcionalnost neobrađena. Unosimo sledeći kod za funkciju **Erase().**

```
        private void Erase()
        {
            numbers[0] = numbers[1] = null;
            @operator = null;
            UpdateCalculatorText();
        }
```

Nema potrebe previše komentarisati ovu funkciju, sve promenljive koje se prikazuju na ekranu su *null*-ovane i pozvana je funkcija za update-ovanje prikaza na ekranu.

Time je naš digitron konačno završen, i možemo ga testirati u emulatoru ili na našem Android uređaju.

Za kraj napomena, layout prikaz bi trebalo da izgleda kao na sledećoj slici, a ako ne izgleda, update-ujte Visual Studio na poslednju verziju**, Help > Check for Updates**, i ispratite proceduru. U svakom slučaju, čak i ako se ne prikazuje dobro na dizajnerskoj površini, radiće kad se pokrene.

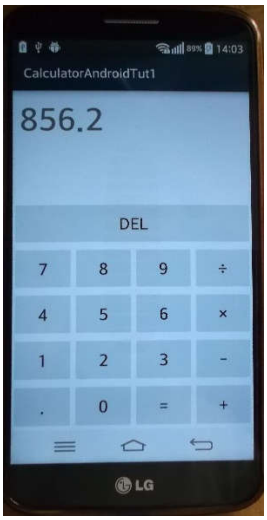


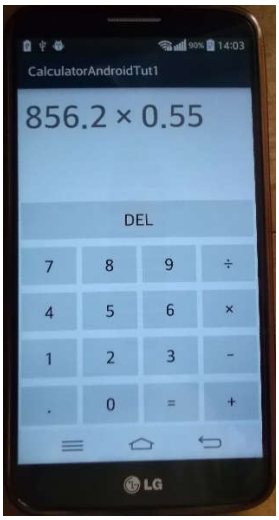
Kompletan kod ove aktivnosti Main je dat u sledećem listingu:

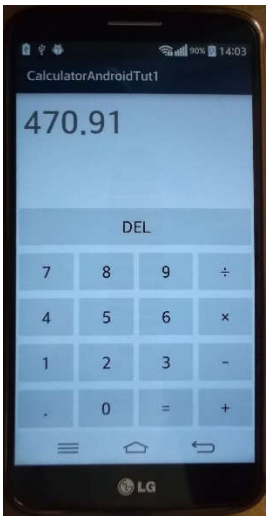
```
using Android.App;
using Android.OS;
using Android.Support.V7.App;
using Android.Runtime;
using Android.Widget;
using Android.Views;
using System;

namespace CalculatorAndroidTut1
{
    [Activity(Label = "@string/app_name", Theme = "@style/AppTheme", MainLauncher = true)]
    public class MainActivity : AppCompatActivity
    {
        private TextView calculatorText;

        private string[] numbers = new string[2];
        private string @operator;

        protected override void OnCreate(Bundle savedInstanceState)
        {
            base.OnCreate(savedInstanceState);
            // Set our view from the "main" layout resource
            SetContentView(Resource.Layout.activity_main);

            calculatorText = FindViewById<TextView>(Resource.Id.calculator_text_view);
        }

        [Java.Interop.Export("ButtonClick")]
        public void ButtonClick(View v)
        {
            Button button = (Button)v;
            if ("0123456789.".Contains(button.Text))
                AddDigitOrDecimalPoint(button.Text);
            else if ("÷×+-".Contains(button.Text))
                AddOperator(button.Text);
            else if ("=" == button.Text)
                Calculate();
            else
                Erase();
        }

        private void AddDigitOrDecimalPoint(string value)
        {
            int index = @operator == null ? 0 : 1;

            if (value == "." && numbers[index].Contains("."))
                return;

            numbers[index] += value;

            UpdateCalculatorText();
        }

        private void AddOperator(string value)
        {
            if (numbers[1] != null)
            {
                Calculate(value);
                return;
            }

            @operator = value;

            UpdateCalculatorText();
        }
```

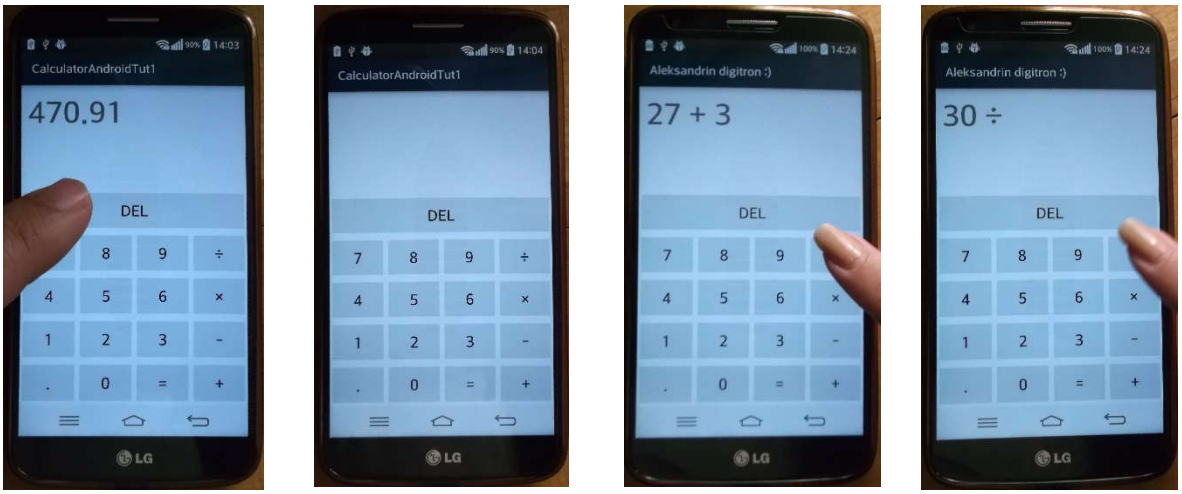
```
        private void Calculate(string newOperator = null)
        {
            double? result = null;
            double? first = numbers[0] == null ? null : (double?)double.Parse(numbers[0]);
            double? second = numbers[1] == null ? null : (double?)double.Parse(numbers[1]);

            switch (@operator)
            {
                case "÷":
                    result = first / second;
                    break;
                case "x":
                    result = first * second;
                    break;
                case "+":
                    result = first + second;
                    break;
                case "-":
                    result = first - second;
                    break;
            }

            if (result != null)
            {
                numbers[0] = result.ToString();
                @operator = newOperator;
                numbers[1] = null;
                UpdateCalculatorText();
            }
        }

        private void Erase()
        {
            numbers[0] = numbers[1] = null;
            @operator = null;
            UpdateCalculatorText();
        }

        private void UpdateCalculatorText() => calculatorText.Text = $"{numbers[0]} {@operator} {numbers[1]}";
    }
}
```

Najzad i par slika izvršavanja aplikacije na Android uređaju koje demonstriraju mogućnosti aplikacije. Vidi se da smo u nekom trenutku promenili ime aplikacije.









5.2. Multi-touch aplikacija

Ova aplikacija je zapravo deo tutorijala u Microsoft Documents, poglavlje *Docs/Xamarin/Xamarin.Android/Application Fundamentals/Touch/Multi-Touch*.

Za njeno potpuno razumevanje nam nedostaje znanje iz prethodnih poglavlja, pa ćemo je opisati samo u kratkim crtama bez tendencije da objasnimo baš sve, već samo načelno način rada. Aplikacija demonstrira mogućnosti korišćenja multi-touch sposobnosti rada ekrana (praćenje više istovremenih dodira), i pri tome to čini u okviru simpatičnog programa za crtanje po ekranu.

5.2.1. Dizajn aplikacije

Dizajn je relativno jednostavan, u okviru ekrana imamo dugme i dve Spinner kontrole. Spinner kontrole predstavljaju vrstu padajućeg menija i služe za izbor jedne od ponuđenih stavki, slično Combo box-u u C#.

Iz jednog Spinner-a biramo željenu boju, iz drugog debljinu „olovke“, odnosno linije koju crtamo.

Dugme služi da obriše kompletan nacrtan sadržaj.

Preko najvećeg dela ekrana se prostire kontrola fingerpaint.FingerPaintCanvasView, čija je namena da prikaže naš crtež, odnosno linije koje vučemo po ekranu.

XML dizajna je dat u listingu:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

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

        <Spinner
            android:id="@+id/colorSpinner"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:prompt="@string/color_spinner_prompt" />

        <Spinner
            android:id="@+id/widthSpinner"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:prompt="@string/width_spinner_prompt" />

        <Button
            android:id="@+id/clearButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/clearText" />

    </LinearLayout>

    <fingerpaint.FingerPaintCanvasView
         android:id="@+id/canvasView"
         android:layout_width="match_parent"
         android:layout_height="match_parent" />
</LinearLayout>
```

Primećujemo da dugme svoje svojstvo *Text* vuče iz string fajla. To rade i svojstva *Prompt* kontrole Spinner. *Prompt* odgovara svojstvu *Text*, samo za Spinner kontrolu, odnosno to je ono što će prikazivati kada se kontrola otvori. Pogledajmo string.xml fajl:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="ApplicationName">FingerPaint</string>

    <string name="color_spinner_prompt">Color</string>

    <string-array name="colors_array">
        <item>Red</item>
        <item>Green</item>
        <item>Blue</item>
        <item>Cyan</item>
        <item>Magenta</item>
        <item>Yellow</item>
        <item>Black</item>
        <item>Gray</item>
        <item>White</item>
    </string-array>
```

```
    <string name="width_spinner_prompt">Width</string>

    <string-array name="widths_array">
        <item>Thin</item>
        <item>Thinish</item>
        <item>Medium</item>
        <item>Thickish</item>
        <item>Thick</item>
    </string-array>

    <string name="clearText">Clear</string>
</resources>
```

Kao što vidimo za *Prompt* postoji niz *item*-a koji će biti prikazani i koji su smešteni u *string-array*.

5.2.2. Kod aplikacije

Za događaje vezane za dodir ekran, generiše se Id kod svaki put kada prst dodirne ekran, i postaje nevažeći kada se podigne prst.

Program koji prati svaki prst, gotovo uvek sadrži rečnik (dictionary) da prati promene u radu touch-a. Ključ rečnika je Id kod koji identifikuje svaki prst, Vrednost rečnika zavisi od aplikacije. U ovoj aplikaciji, svaki potez prstom, od pritiska do puštanja ekrana, je asociran sa objektom koji sadrži sve informacije potrebne da se prikažu linije nacrtane tim prstom. Za te svrhe je dizajnirana mala klasa *FingerPaintPolyline*:

```
class FingerPaintPolyline
{
    public FingerPaintPolyline()
    {
        Path = new Path();
    }

    public Color Color { set; get; }

    public float StrokeWidth { set; get; }

    public Path Path { private set; get; }
}
```

Svaka poli-linija ima boju, širinu olovke, i Android Graphic Path objekat koji akumulira i prikazuje više tačaka linije koja je nacrtana.

Ostatak koda koji objašnjavamo se sadrži *View* izvedbi nazvanoj *FingerPaintCanvasView*. Ta klasa sadrži rečnik objekata *FingerPaintPolyline*, tokom vremena kada su crtane jednim ili više prstiju.

```
Dictionary<int, FingerPaintPolyline> inProgressPolylines = new
Dictionary<int, FingerPaintPolyline>();
```

Rečnik omogućava pogledu da brzo pristupi *FingerPaintPolyline* informacijama za određeni prst.

FingerPaintCanvasView klasa takođe sadrži i listu objekata za poli-linije koje su završene.

```
List<FingerPaintPolyline> completedPolylines = new
List<FingerPaintPolyline>();
```

Objekti u listi su u redosledu kojim su crtani.

FingerPaintCanvasView Override-uje dve metode definisane pogledom, *OnDraw()* i *OnTouchEvent()*. U override-u prve, pogled iscrtava kompletne poli-linije, a u drugoj one koje su još u progresu.

Override *OnTouchEvent()* počinje preuzimanjem *pointerIndex* vrednosti svojstva *ActionIndex*, koje se razlikuje za različite prste ali nije konzistentna za različite događaje. Zato se koristi *pointerIndex* da bi dobio *id* vrednost iz *GetPointerId*() metode koja je konzistentna kroz više događaja.

```
public override bool OnTouchEvent(MotionEvent args)
{
    // Get the pointer index
    int pointerIndex = args.ActionIndex;

    // Get the id to identify a finger over the course of its progress
    int id = args.GetPointerId(pointerIndex);

    // Use ActionMasked here rather than Action to reduce the number of
possibilities
    switch (args.ActionMasked)
    {
        // ...
    }

    // Invalidate to update the view
    Invalidate();

    // Request continued touch input
    return true;
}
```

Override koristi *Action Masked* property u *switch* izrazu umesto *Action*, zato što kod multi-toucha svojstvo Action ima vrednost *MotionEventsAction.Down* za prvi prst koji dodirne ekran, a onda vrednosti Pointer2Down i Pointer3Down za drugi i treći. Za četvrti i peti dobija numeričke vrednosti koje i ne odgovaraju članovima *MotionEventsAction* enumeracije, što otežava pregled.

Slično se dešava i kada prst napušta kontakt s ekranom.

Svojstvo *ActionMask* uzima manje brojeva vrednosti jer je namenjeno radu u saradnji sa *ActionIndex* svojstvom za diferencijaciju kod više prstiju. Kada prsti dodirnu ekran, svojstvo dodeljuje *MotionEventActions.Down* za prvi prst i *PointerDown* za ostale. Pri dizanju prsta dodeljuje se *Pointer1Up* za ostale a *Up* za prvi prst.

Kada se koristi *ActionMasked*, *ActionIndex* se razlikuje za ostale prste koji dodiruju i napuštaju ekran, ali nam obično ne treba ta vrednost za upotrebu, osim kao argument za druge metode u *MotionEvent* objektu. Jedna od najvažnijih metoda za muli-touch je GetPointerId, pozivana u gornjem kodu. Ona vraća vrednost koja se može koristiti kao ključ u rečniku da se veže za određen događaj ili prst.

OnTouchEvent procesira *MotionEventActions* i *PointerDown* identično, kreirajući nov *FingerPaintPolyline* objekat i dodajući ga u rečnik.

```
public override bool OnTouchEvent(MotionEvent args)
{
    // Get the pointer index
    int pointerIndex = args.ActionIndex;
    // Get the id to identify a finger over the course of its progress
    int id = args.GetPointerId(pointerIndex);
    // Use ActionMasked here rather than Action to reduce possibilities
    switch (args.ActionMasked)
    {
        case MotionEventActions.Down:
        case MotionEventActions.PointerDown:
            // Create a Polyline, set the initial point, and store it
            FingerPaintPolyline polyline = new FingerPaintPolyline
            {
                Color = StrokeColor,
                StrokeWidth = StrokeWidth
            };
            polyline.Path.MoveTo(args.GetX(pointerIndex),
                                 args.GetY(pointerIndex));
            inProgressPolylines.Add(id, polyline);
            break;
        // ...
    }
    // ...
}
```

Primećujemo da *pointerIndex* se takođe koristi da pribavi poziciju prsta unutar pogleda. Sve informacije o dodiru su povezane sa vrednošću *pointerIndex.* Id unikatno identifikuje prste preko više poruka, pa se koristi da bi se kreirao unos u rečniku.

Slično *OnToucheEvent* override barata sa *MotionEventActions.Up* i *Pointer1Up* identično, transferujući kompletan poly-line u kolekciju completedPolylines da bi se mogao iscrtati tokom *OnDraw* override metode. Kod takođe uklanja Id unos iz rečnika.

```
public override bool OnTouchEvent(MotionEvent args)
{
    // ...
    switch (args.ActionMasked)
    {
        // ...
        case MotionEventActions.Up:
        case MotionEventActions.Pointer1Up:

            inProgressPolylines[id].Path.LineTo(args.GetX(pointerIndex),
                                                args.GetY(pointerIndex));

            // Transfer the in-progress polyline to a completed polyline
            completedPolylines.Add(inProgressPolylines[id]);
            inProgressPolylines.Remove(id);
            break;

        case MotionEventActions.Cancel:
            inProgressPolylines.Remove(id);
            break;
    }
    // ...
}
```

Između *down* i *up* događaja, načelno ima puno *MotionEventActions.Move* događaja. Oni su grupisani u jedinstven poziv *OnTouchEvent* i njima se barata drugačije od *Up* i *Down* događaja. Vrednost *ActionIndex* svojstva se mora ignorisati. Umesto toga, metod mora se nabaviti mnoštvo pointerIndex vrednosti u petlji između 0 i *Pointer.Count* svojstva, i onda pribaviti Id za svaku *pointerIndex* vrednost.

```
public override bool OnTouchEvent(MotionEvent args)
{
    // ...
    switch (args.ActionMasked)
    {
        // ...
        case MotionEventActions.Move:
            // Multiple Move events are bundled, so handle them differently
          for (pointerIndex = 0; pointerIndex < args.PointerCount; pointerIndex++)
          {
              id = args.GetPointerId(pointerIndex);
              inProgressPolylines[id].Path.LineTo(args.GetX(pointerIndex),
                                                  args.GetY(pointerIndex));
          }
          break;
       // ...
    }
    // ...
}
```

Na ovaj način se omogućava da program prati svaki prst i crta rezultate na ekran. Kompletan kod FingerPaintCanvasView klase:

```
using System;
using System.Collections.Generic;
using Android.Content;
using Android.Util;
using Android.Views;
using Android.Graphics;

namespace FingerPaint
{
    public class FingerPaintCanvasView : View
    {
        // Two collections for storing polylines
        Dictionary<int, FingerPaintPolyline> inProgressPolylines = new Dictionary<int,
FingerPaintPolyline>();
        List<FingerPaintPolyline> completedPolylines = new List<FingerPaintPolyline>();
        Paint paint = new Paint();
        public FingerPaintCanvasView(Context context) : base(context)
        {
            Initialize();
        }

        public FingerPaintCanvasView(Context context, IAttributeSet attrs) :
            base(context, attrs)
        {
            Initialize();
        }

        void Initialize()
        {
        }
        // External interface accessed from MainActivity
        public Color StrokeColor { set; get; } = Color.Red;
        public float StrokeWidth { set; get; } = 2;
```

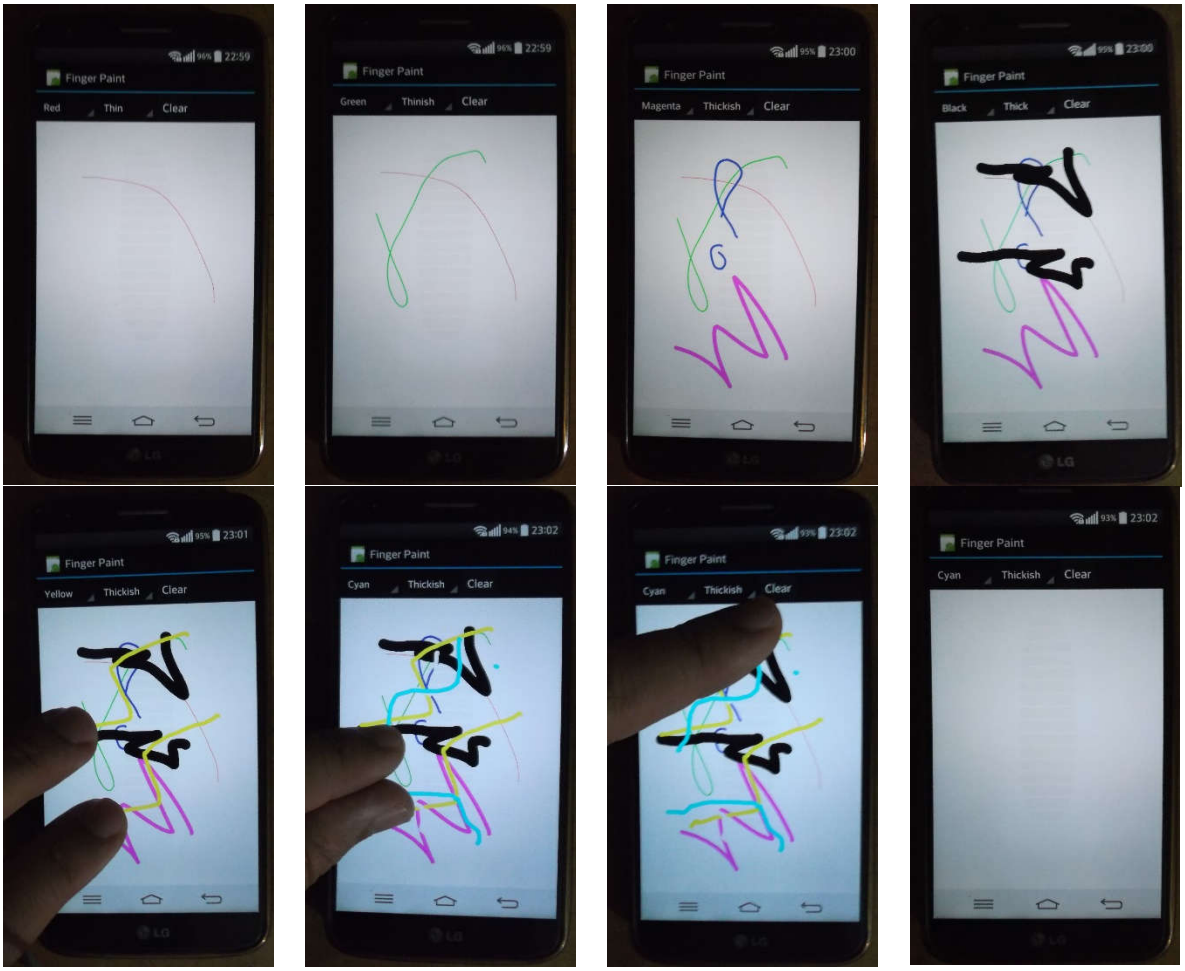
```
        public void ClearAll()
        {
            completedPolylines.Clear();
            Invalidate();
        }

        // Overrides
        public override bool OnTouchEvent(MotionEvent args)
        {
            // Get the pointer index
            int pointerIndex = args.ActionIndex;

            // Get the id to identify a finger over the course of its progress
            int id = args.GetPointerId(pointerIndex);

            // Use ActionMasked rather than Action to reduce the number of possibilities
            switch (args.ActionMasked)
            {
                case MotionEventActions.Down:
                case MotionEventActions.PointerDown:

                    // Create a Polyline, set the initial point, and store it
                    FingerPaintPolyline polyline = new FingerPaintPolyline
                    {
                        Color = StrokeColor,
                        StrokeWidth = StrokeWidth
                    };

                    polyline.Path.MoveTo(args.GetX(pointerIndex),
                                         args.GetY(pointerIndex));

                    inProgressPolylines.Add(id, polyline);
                    break;

                case MotionEventActions.Move:

                    // Multiple Move events are bundled, so handle them differently
                    for (pointerIndex = 0; pointerIndex < args.PointerCount; pointerIndex++)
                    {
                        id = args.GetPointerId(pointerIndex);

                        inProgressPolylines[id].Path.LineTo(args.GetX(pointerIndex),
                                                            args.GetY(pointerIndex));
                    }
                    break;

                case MotionEventActions.Up:
                case MotionEventActions.Pointer1Up:

                    inProgressPolylines[id].Path.LineTo(args.GetX(pointerIndex),
                                                        args.GetY(pointerIndex));

                    // Transfer the in-progress polyline to a completed polyline
                    completedPolylines.Add(inProgressPolylines[id]);
                    inProgressPolylines.Remove(id);
                    break;

                case MotionEventActions.Cancel:
                    inProgressPolylines.Remove(id);
                    break;
            }
```

```
            // Invalidate to update the view
            Invalidate();

            // Request continued touch input
            return true;
        }

        protected override void OnDraw(Canvas canvas)
        {
            base.OnDraw(canvas);

            // Clear canvas to white
            paint.SetStyle(Paint.Style.Fill);
            paint.Color = Color.White;
            canvas.DrawPaint(paint);

            // Draw strokes
            paint.SetStyle(Paint.Style.Stroke);
            paint.StrokeCap = Paint.Cap.Round;
            paint.StrokeJoin = Paint.Join.Round;

            // Draw the completed polylines
            foreach (FingerPaintPolyline polyline in completedPolylines)
            {
                paint.Color = polyline.Color;
                paint.StrokeWidth = polyline.StrokeWidth;
                canvas.DrawPath(polyline.Path, paint);
            }

            // Draw the in-progress polylines
            foreach (FingerPaintPolyline polyline in inProgressPolylines.Values)
            {
                paint.Color = polyline.Color;
                paint.StrokeWidth = polyline.StrokeWidth;
                canvas.DrawPath(polyline.Path, paint);
            }
        }
    }
}
```

Kompletan kod Main.cs fajla:

```
using System;
using Android.App;
using Android.Graphics;
using Android.Widget;
using Android.OS;

namespace FingerPaint
{
    [Activity(Label = "Finger Paint", MainLauncher = true, Icon = "@drawable/icon")]
    public class MainActivity : Activity
    {
        FingerPaintCanvasView fingerPaintCanvasView;

        protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);

            // Set the view from the Main.axml layout resource
            SetContentView(Resource.Layout.Main);
           // Get a reference to the FingerPaintCanvasView from the Main.axml file
           fingerPaintCanvasView = FindViewById<FingerPaintCanvasView>(Resource.Id.canvasView);

           // Set up the Spinner to select stroke color
           Spinner colorSpinner = FindViewById<Spinner>(Resource.Id.colorSpinner);
           colorSpinner.ItemSelected += OnColorSpinnerItemSelected;

           var adapter = ArrayAdapter.CreateFromResource(this, Resource.Array.colors_array,
Android.Resource.Layout.SimpleSpinnerItem);
           adapter.SetDropDownViewResource(Android.Resource.Layout.SimpleSpinnerDropDownItem);
           colorSpinner.Adapter = adapter;

           // Set up the Spinner to select stroke width
           Spinner widthSpinner = FindViewById<Spinner>(Resource.Id.widthSpinner);
           widthSpinner.ItemSelected += OnWidthSpinnerItemSelected;

           var widthsAdapter = ArrayAdapter.CreateFromResource(this, Resource.Array.widths_array,
Android.Resource.Layout.SimpleSpinnerItem);
           widthsAdapter.SetDropDownViewResource(Android.Resource.Layout.SimpleSpinnerDropDownItem);
           widthSpinner.Adapter = widthsAdapter;

           // Set up the Clear button
           Button clearButton = FindViewById<Button>(Resource.Id.clearButton);
           clearButton.Click += OnClearButtonClick;
       }

       void OnColorSpinnerItemSelected(object sender, AdapterView.ItemSelectedEventArgs args)
       {
           Spinner spinner = (Spinner)sender;
           string strColor = (string)spinner.GetItemAtPosition(args.Position);
           Color strokeColor = (Color)(typeof(Color).GetProperty(strColor).GetValue(null));
           fingerPaintCanvasView.StrokeColor = strokeColor;
       }

       void OnWidthSpinnerItemSelected(object sender, AdapterView.ItemSelectedEventArgs args)
       {
           Spinner spinner = (Spinner)sender;
           float strokeWidth = new float[] { 2, 5, 10, 20, 50 } [args.Position];
           fingerPaintCanvasView.StrokeWidth = strokeWidth;
       }

       void OnClearButtonClick(object sender, EventArgs args)
       {
           fingerPaintCanvasView.ClearAll();
       }
   }
}
```

Na slikama je demonstriran rad aplikacije:



6. Zaključak

Ovim zaključujemo osnovni kurs upoznavanja Xamarin.Android platforme. U njemu smo postupno kreirali jednu jednostavnu aplikaciju i kroz taj čin naučili neke bitne koncepte Android programiranja, otkrili neke mogućnosti Xamarin platforme kao dela Visual Studio-a, i u zbiru stekli solidnu osnovu za početak razvoja aplikacija za Android.

Ipak, mogućnosti ove platforme su značajno veće od onoga što možemo predstaviti za kratko vreme, i u zato treba nastaviti sa proučavanjem ove platforme. U tu svrhu nam mogu pomoći dokumenti na zvaničnom Microsoft-ovom sajtu https://docs.microsoft.com/en-us/xamarin/#pivot=platforms&panel=Android.

Mi smo u ovom radu obradili oblast koja otprilike pokriva prvo poglavlje ove opširne dokumentacije, pod nazivom *Getting Started with Android*. Ostatak dokumentacije koja bi upotpunila poznavanje Xamarin.Android platforme je raspoređen po sledećim poglavljima:

- Getting Started
- Application Fundamentals
- User Interface
- Platform Features
- Xamarin.Essentials
- Data and Cloud Services
- Deployment and Testing
- Advanced Concepts and Internals
- Wear

Naziv poglavlja uglavnom dovoljno rečito govori o onome šta u tom poglavlju možemo očekivati da naučimo, mada je svakako preporučljivo čitati ih redom jer postupno uvode nove koncepte. Jedino da pojasnimo da se u poslednjem poglavlju *Wear* govori o nosivim Android gadget-ima, poput pametnih satova.

U okviru te dokumentacije se na kraju nalazi i nekoliko urađenih primera sa datim kodom koji demonstriraju neke od objašnjenih koncepata u praktičnom radu.

Osim ovih primera, i na YouTube servisu, kao i na drugim lokacijama na internetu, mogu se naći brojni tutorijali za kreiranje Android aplikacija u Xamarinu.

I za kraj, večito pitanje:

> „Kom se carstvu privoljeti?“
>
> Visual Studio – Xamarin ili Android Studio?

Prednosti Xamarina su:

- Mogućnost korišćena istog koda na Windowsu, Androidu i iOS-u
- Pisanje UI samo jednom korišćenjem Xamarin Formi
- Programiranje u C#
- **.Net** podrška
- Mogućnost korišćenja Java biblioteka u C#

Mane su mu:

- Manji broj biblioteka za Android
- Komplikovano priključivanje biblioteka, pogotovo Java-inih
- Layout dizajner slabiji sa opcijama, često greši i ne prikazuje elemente ispravno
- Rad sa emulatorima veoma problematičan
- Velika hardverska zahtevnost
- VELIKO zauzeće prostora

Prednosti Android Studia:

- Bolja dokumentacija
- Veći broj biblioteka
- Lako povezivanje biblioteka
- Bolji dizajner layout-a, više opcija
- Nešto bolji XML editor
- Mogućnost korišćenja novog naprednog jezika, Kotlin, specijalizovanog za Android, interoperatibilan sa Javom, koji je pravljen po uzoru na C# onoliko koliko se svojevremeno C# ugledao na Java jezik
- Hardverski nezahtevniji
- Mnogo manje zauzeće skladišnog prostora

Mane:

- Vezan je za samo jednu platformu

Kao i uvek, odluka je na Vama.

Literatura:

[1] https://sr.wikipedia.org/sr-el/Majkrosoft_vižual_studio

https://en.wikipedia.org/wiki/Xamarin

[2] https://www.sk.rs/2017/07/sktr01.html

[3] https://docs.microsoft.com/en-us/xamarin/android/

[4] https://www.youtube.com/c/resocoder

[5] https://github.com/xamarin/monodroid-samples/tree/master/PhonewordMultiscreen/Phoneword

[6] https://www.youtube.com/watch?v=rkNikCa5D48