Tallinna Ülikool
Digitehnoloogiate instituut

Cross-platform mobiilirakenduste arendus kasutades Xamarin'i

Seminaritöö

Autor: Priit Mattus
Juhendaja: Jaagup Kippar

Autor: ……,,……,, 2015

Juhendaja: ……,,……,, 2015

Instituudi direktor: ……..,,……..,, 2015

Tallinn 2015

Autorideklaratsioon

Deklareerin, et käesolev seminaritöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

…….. ……..

(kuupäev) (autor)

Sisukord

Sissejuhatus

Seminaritöö eesmärgiks on luua eestikeelne õppematerjal, mis tutvustaks cross-platform mobiilirakenduste arendamist Xamarini keskkonnas. Õppematerjali käigus kasutan viimast stabiilset Xamarini versiooni, milleks on Xamarin 5.7.

Õppematerjal on suunatud tarkvaraarendajatele, kes soovivad luua Androidi- ja IOS-rakendusi. Õppematerjali käigus antakse õppurile edasi baasteadmised MvvmCros projekti seadistamiseks ning lihtsama rakenduse loomiseks. Õppematerjalist jätsin välja Windows Phone platvormile arenduse, sest Windows Phone kasutajate hulk on hetkel väga väike. Teema valiku põhjuseks on tööalane kogemus Xamarini ja MvvmCrossi kasutamisel ning hetkel puudub Eesti keelne õppematerjal Xamarini ja MvvmCrossi kasutamiseks. Soovin tõsta teadlikust Xamarini arendusvahendite olemasolust. Xamarin on ainuke cross-platform arendus vahend millel on võrdsed võimalused natiivse arendusega. Samuti on arendajal võimalik arendada kasutades C# keelt. See loob suure eelise sest arendaja ei pea oskama java ning objective-C keelt. Xamarini puuduseks on see, et veebis võib olla näiteid ja juhendeid raskem leida kuid see probleem väheneb, sest arendajate kogukond on kiirelt kasvav.

Ennem materjali kasutamist on vajalikud baasteadmised C# keeles. Materjal sisaldab juhendit arendamiseks Xamarin Studio keskkonnas. Selleks, et arendada IOS rakendusi, on vaja kasutada ka Mac OSX operatsioonisüsteemiga arvutit.

Mõisted

Data binding – Tehnika millega seotakse kokku kahe elemendi vahelised andmed

Model-View-ViewModel(Mvvm) – Programmeerimismuster, mis eraldab loogika ja kasutajaliidese. Jagab koodi kolme komponendi vahel – model, view ja viewmodel. Komponendid ei ole omavahel tugevalt seotud, see võimaldab komponentide vahetust, komponendi sisest koodi muutmist ilma teisi komponente mõjutamata ja komponente on võimalik arendada sõltumatult.

Cross-Platform development – Tarkvaraarendus liik, mis on suunatud mitmele platvormile korraga.

NuGet Package Manager – Paketi haldur Microsofti arenduskeskkonnas. Lihtsustab pakettide loomist ja kasutamist.

.NET framework – Microsofti arendatud üldotstarbeline arendusplatvorm mistahes rakenduse arendamiseks.

Portable Class Library (PCL) – Projekti tüüp milles on võimalik arendada koodi rohkem kui ühele .NET raamistiku platvormile.

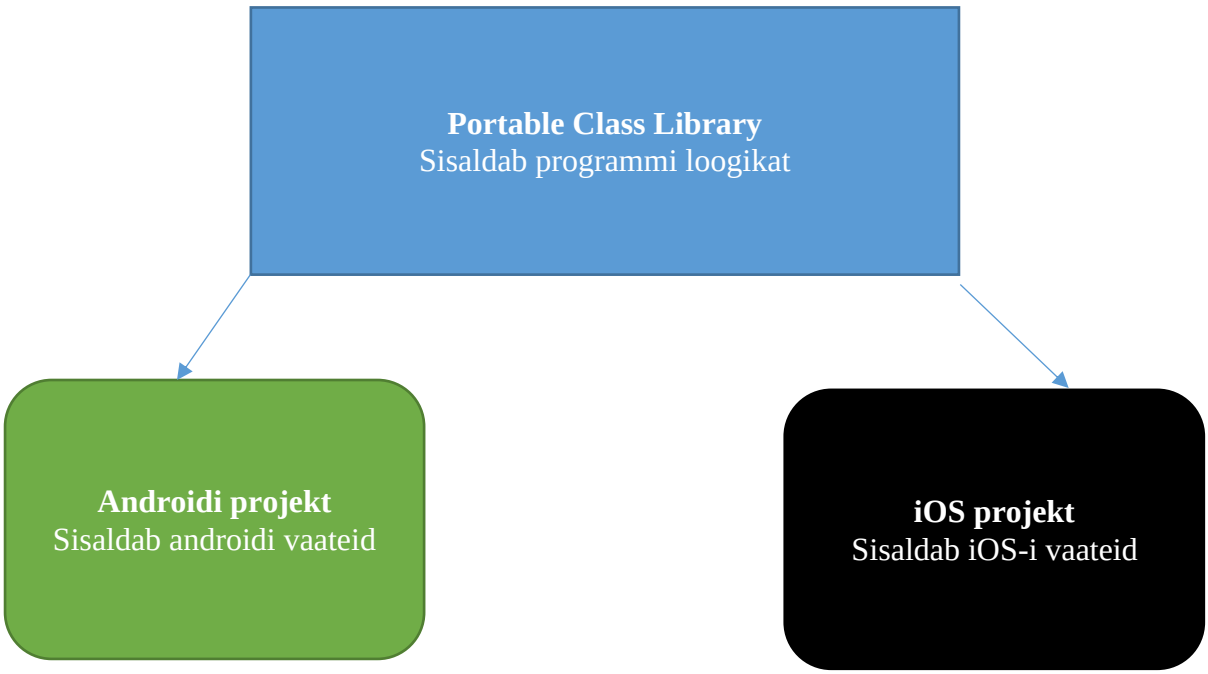
1. Mis on Xamarin?

Xamarin San Francisos paiknev tarkvarafirma, mis loodi 2011 aasta mais. Xamarini loojateks on insenerid kes valmistasid MonoTouch-i ja Mono for Android-i. MonoTouch ja Mono for Android on cross-platform implementatsioonid Common Langugage infrastruktuurist(CLI) . MonoTouch on mõeldud iOS rakenduste arendamiseks C# keeles ja Mono for Android rakenduste arendamiseks androidile C# keeles.

Xamarin võimaldab C# keele koodibaasi kasutades arendada natiivseid rakendusi iOS, Apple OSX, Android ja Windows platformile, jagades platformide vahel koodi. Xamarini kasutavad üle 500 tuhande tarkvaraarendaja üle maailma. Arendamine on võimalik Xamarini enda loodud Xamarin Studio arenduskeskonnas ja ka Visual Studios. Xamarin toetab iOS, android ja Windows Phone platvorme. Võrreldes konkurentidega (nt Phonegap) on Xamarin ainuke arendusvahend kus igal platvormil kasutatakse natiivseid komponente. See tagab selle, et lõppkasutaja on tuttav kõigi kasutajaliidsel olevate elementidega ning rakendus töötab sama kiirest kui natiivselt arendatud rakendus.

2. Mis on MvvmCross?

MvvmCross on model view viewmodel(mvvm) muster cross-platform rakenduste loomiseks, mis võimaldab programmikoodi suurt taaskasutust erinevatele platformidele arendamiseks. MvvmCrossi kasutamiseks tuleb luua portable class library tüüpi projekt mis sisaldab enamus programmi loogikast ja platvormi põhine projekt mis sisaldab endas vaateid ja platformile spetsiifilist koodi nagu näiteks telefoni kaamera kasutamine. Kuna enamus loogikast erinevatel platvormidel on sama, näiteks serveriga suhtlus ja andmebaasiga suhtlemine, siis on vaja see loogika kirja panna ainult üks kord. See annab cross-platform arendamisele suure eelise natiivse arenduse ees. Andmete sidumine programmi loogika ja vaadete vahel käib kasutades data bindingut. Väga hea on kasutada MvvmCrossi ka sellisel juhul kui arenduses on Windowsi rakendus ning arenduskeeleks on C# ja tahetakse juurde teha mobiili rakendusi. Sel juhul on võimalik taaskasutada kogu koodi mobiilile arendamisel.



Joonis 1Mvvm cross

3. Töökeskkonna seadistamine

Õppematerjais käsitletakse rakenduste arendamist Androidile ja iOS-ile. Selleks, et arendada rakendust Visual Studios on vaja Windows 8 või uuemat operatsioonisüsteemi ja Visual Studio 2012 või uuemat versiooni. Vaja on ka Mac OSX operatsioonisüsteemiga arvutit millega on võimalik rakenduse kompileerimine iOS operatsioonisüsteemile. Rakendusi saab arendada ka Xamarin Studiot kasutades. Xamarin Studio toetab Windowsi kui ka Mac OSX operatsioonisüsteeme. Materjalis olev Xamarin Studio juhend on koostatud kasutades Mac OSX viimast versiooni milleks on El Capitan. Mac operatsioonisüsteemil arendades tuleb installeerida lisaks Xamarini tarkvarale ka XCode.

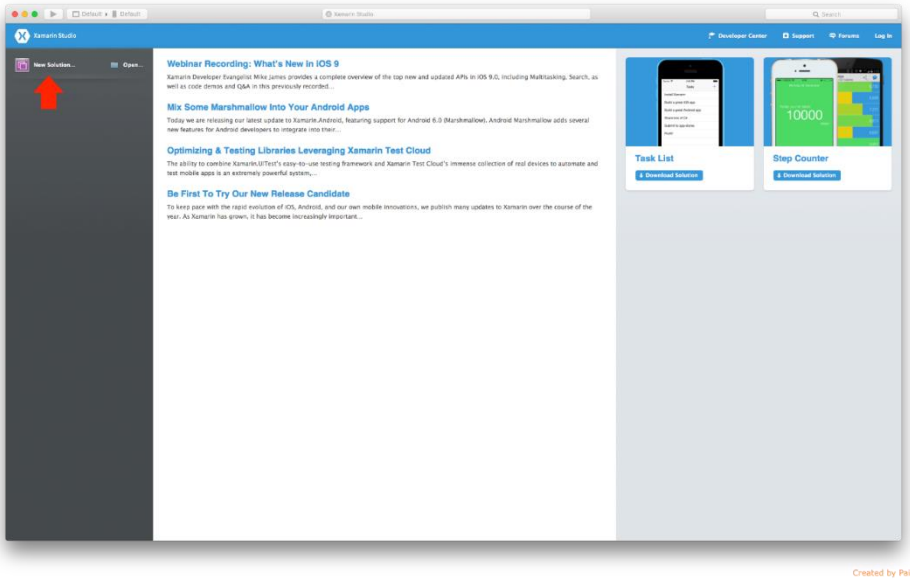
3.1 Xamarini tööriistad

Xamariniks vajaliku tarkvara saab alla laadida Xamarini ametlikult veebilehelt http://xamarin.com/ . Allalaadimiseks tuleb lehel registreeruda ning peale seda on võimalik tarkvara alla laadida. Vajalik on ka androidi emulaator. Emulaatoriks võib kasutada ka Google poolt loodud emulaatoried kui ka Xamarini enda loodud emulaatorit. Materjalis olevates näidetes on kasutatud Xamarini emulaatorit. Emulaatori saab alla laadida aadressilt https://xamarin.com/android-player . iOS emuleerimiseks kasutame XCode-iga kaasa tulevad iOS emulaatorit.

4. Projektide seadistamine

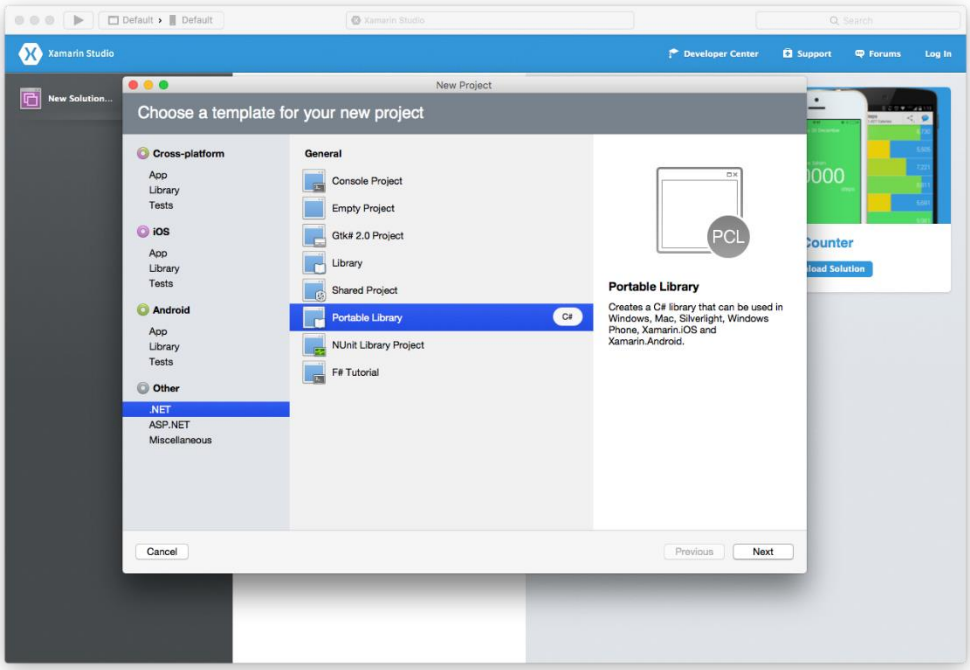
4.1 Portable class library loomine

Loome uue solutioni vajutades Xamarin Studios üleval vasakul olevat nuppu New Solution.



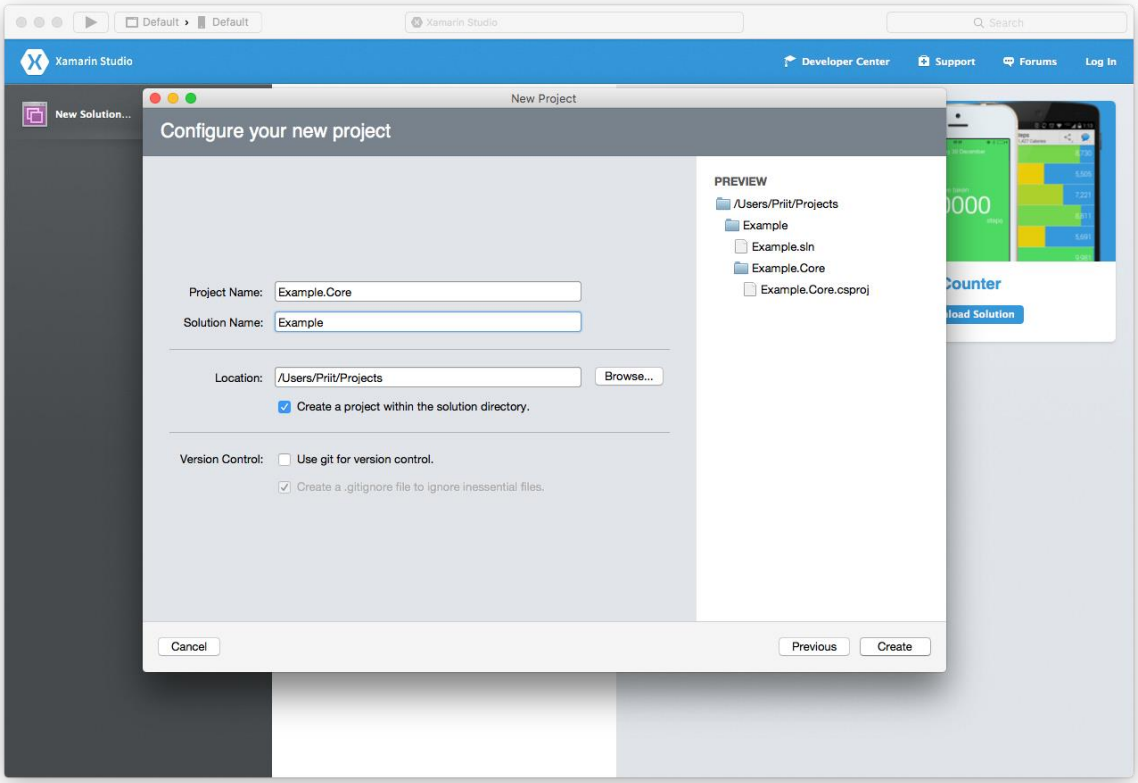
Joonis 2Uue solutioni loomine

Projekti tüübiks valime vasakult Other kategooriast .NET ja Portable Library.



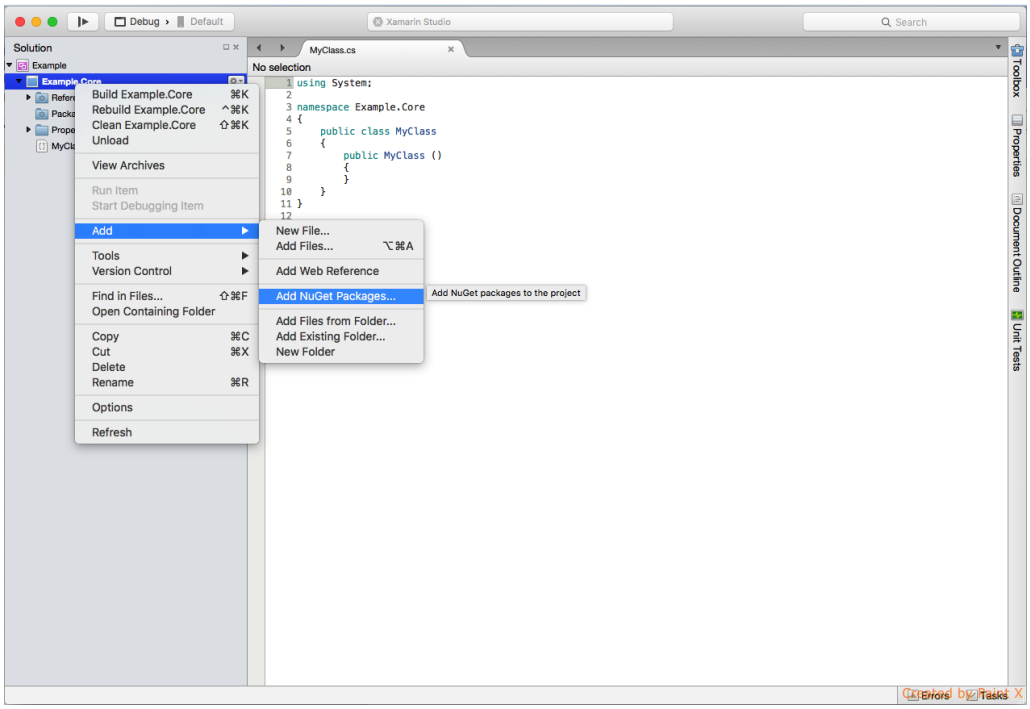
Joonis 3Portable Class Library projektitüübi valimine

Vajutame Next ning anname Projektile ja Solutionile nime. Projekti nimeks Example.Core ja solutioni nimeks Example.



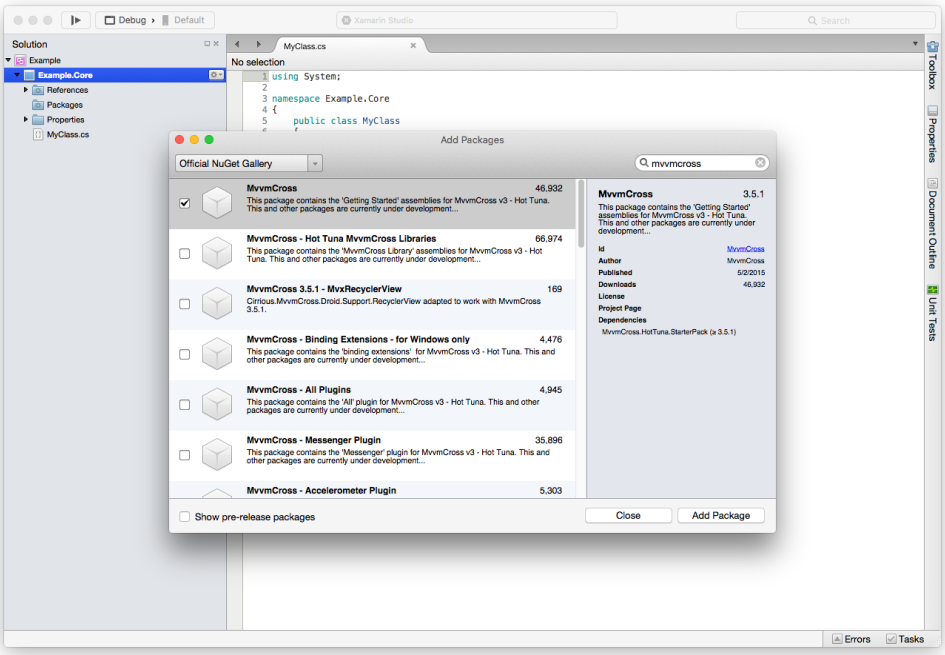
Joonis 4 Portable Class Library projektile nime andmine

Selleks, et oleks võimalik kasutada MvvmCross-i tuleb lisada Example.Core projektile MvvmCross-i raamistik. Raamistiku saab lisada kasutades Nuget package manageri. Vajutame parema klahviga Example.Core projektile ning valime Add -> Add NuGet Packages...



Joonis 5NuGet package lisamine

Paremal üleval olevasse lahtrisse kirjutame otsisõnaks mvvmcross ning valime tulemustest MvvmCross ning vajutame Add Package.

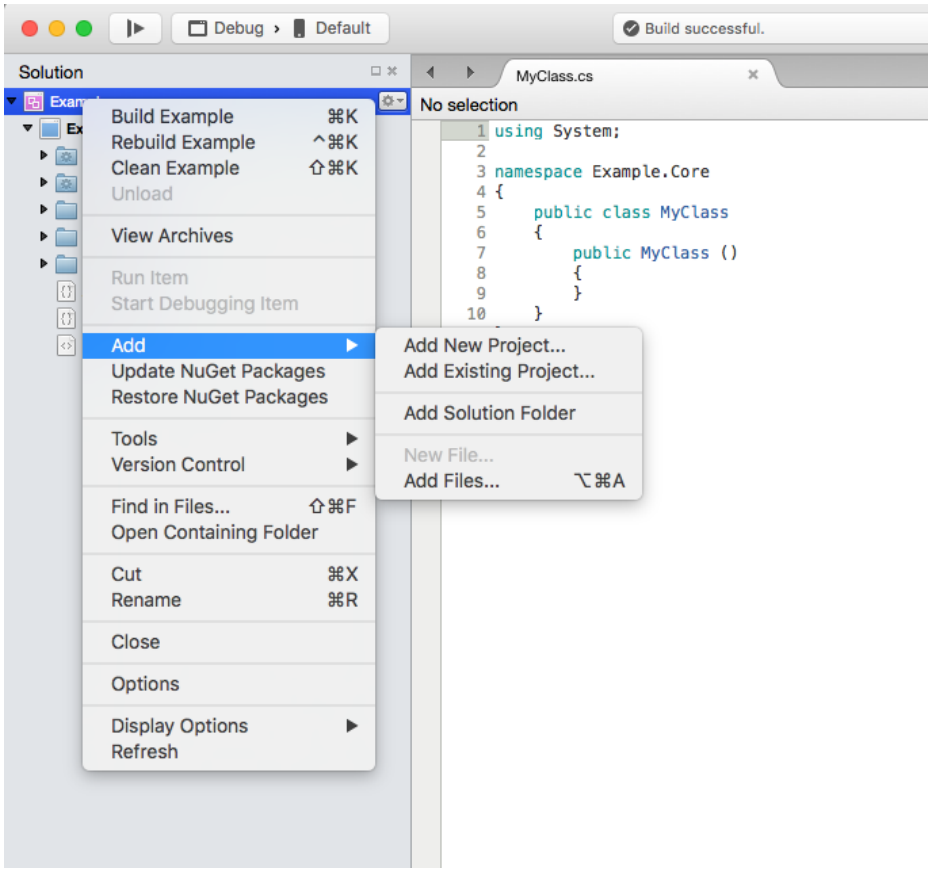


Joonis 6 MvvmCross package valimine

Sellega on Example.Core projekti algne seadistus lõppenud.

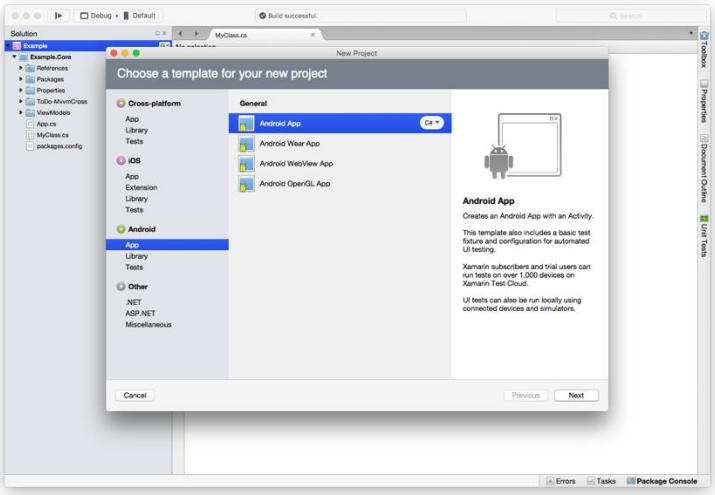
4.2 Androidi projekt

Lisame solutionisse uue projekti. Selleks vajutame parema klahviga solutionile ning valime Add -> Add New Project.



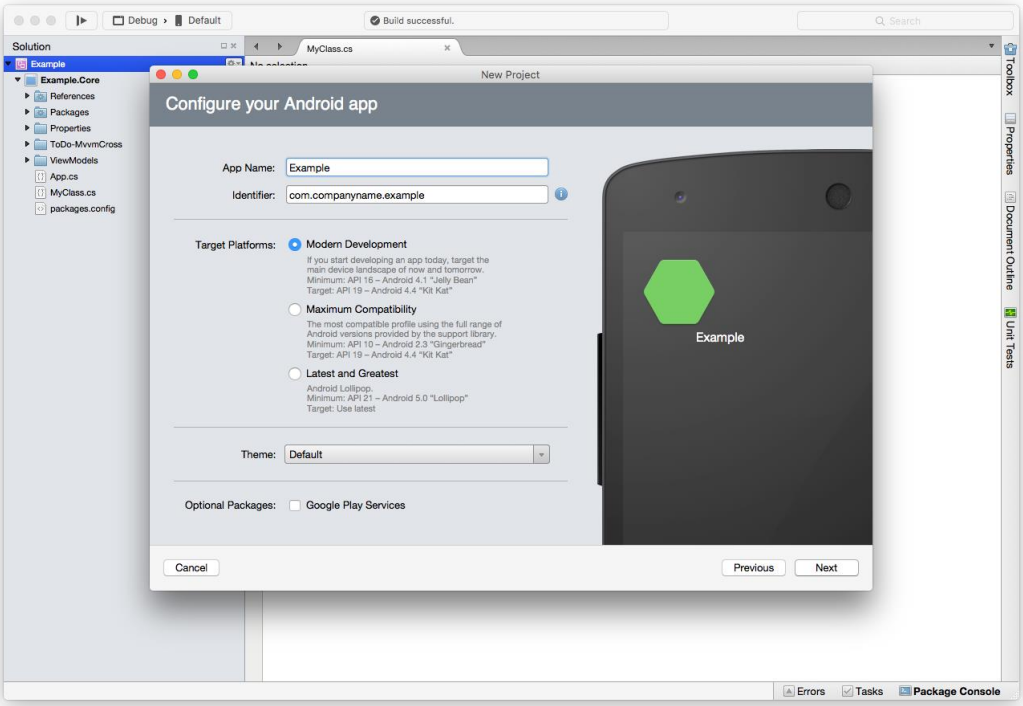
Joonis 7 Androidi projekti lisamine

Valime vasakult Androidi kategooriast App ning tüübiks Android App.



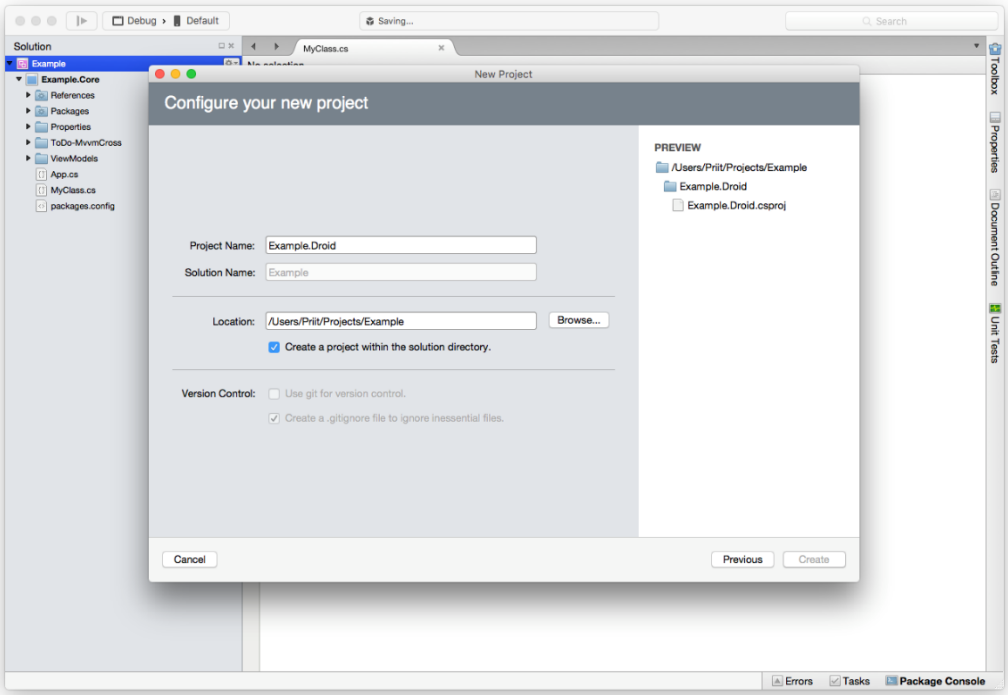
Joonis 8 Androidi projekti tüübi valimine

Anname rakendusele nime Example. Ülejäänud valikuid muutma ei pea. Vajutame Next.



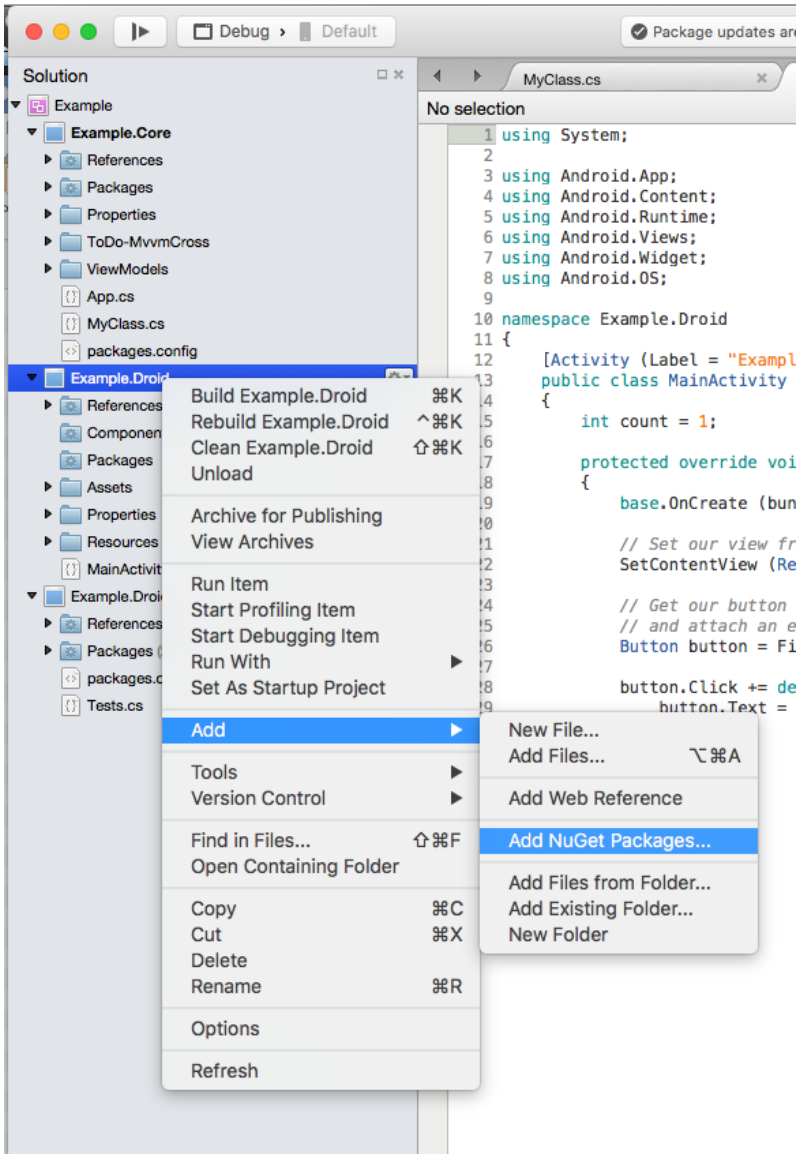
Joonis 9 Androidi rakenduse nimetamine

Projektile kirjutame nimeks Example.Droid ja vajutame Create.



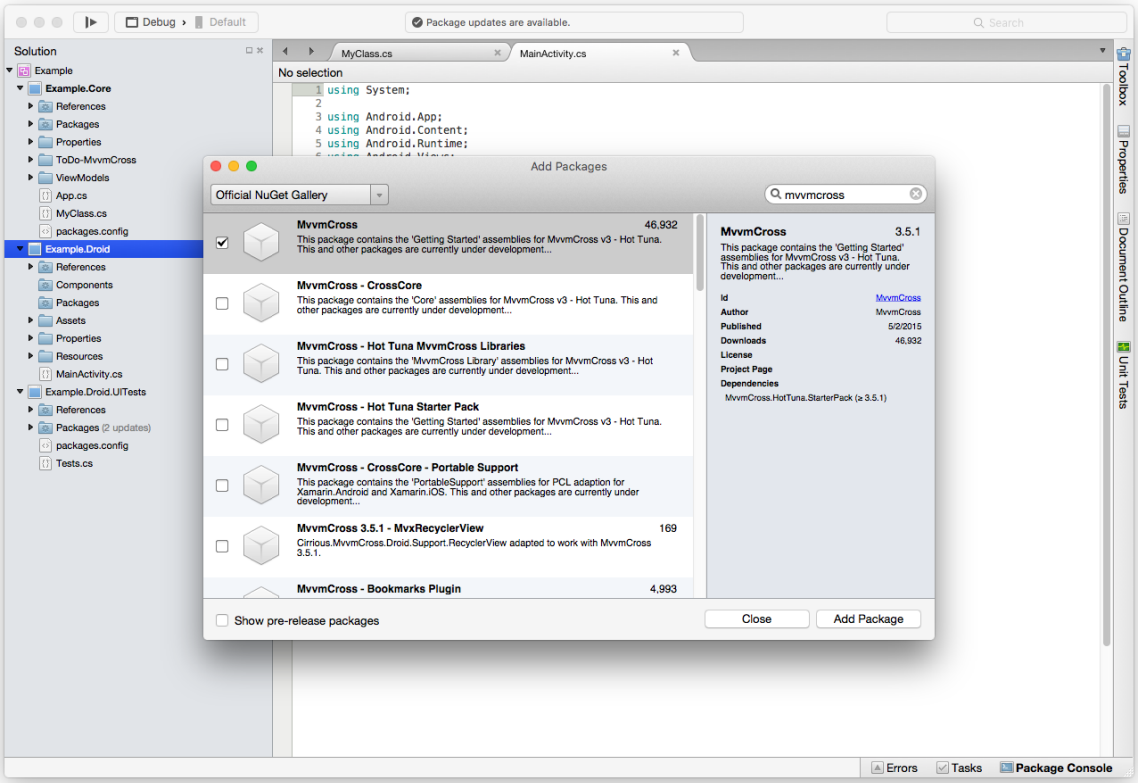
Joonis 10 Androidi projekti nimetamine

Selleks, et saaksime kasutada androidi projektis MvvmCross-i lisame MvvmCross-i kasutades NuGet package manageri. Selleks vajutame parema klahviga Example.Droid projektile ning valime Add -> Add NuGet Packages...



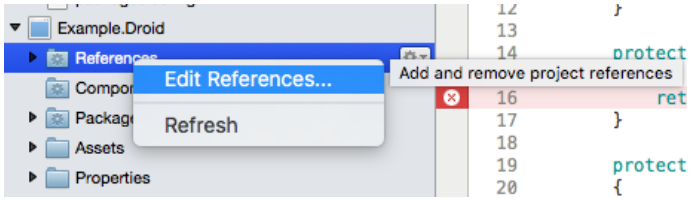
Joonis 11 Androidi projektile NuGet package lisamine

Otsingu lahtrisse kirjutame mvvmcross ning valime tulemustest MvvmCross. Vajutame Add Package.

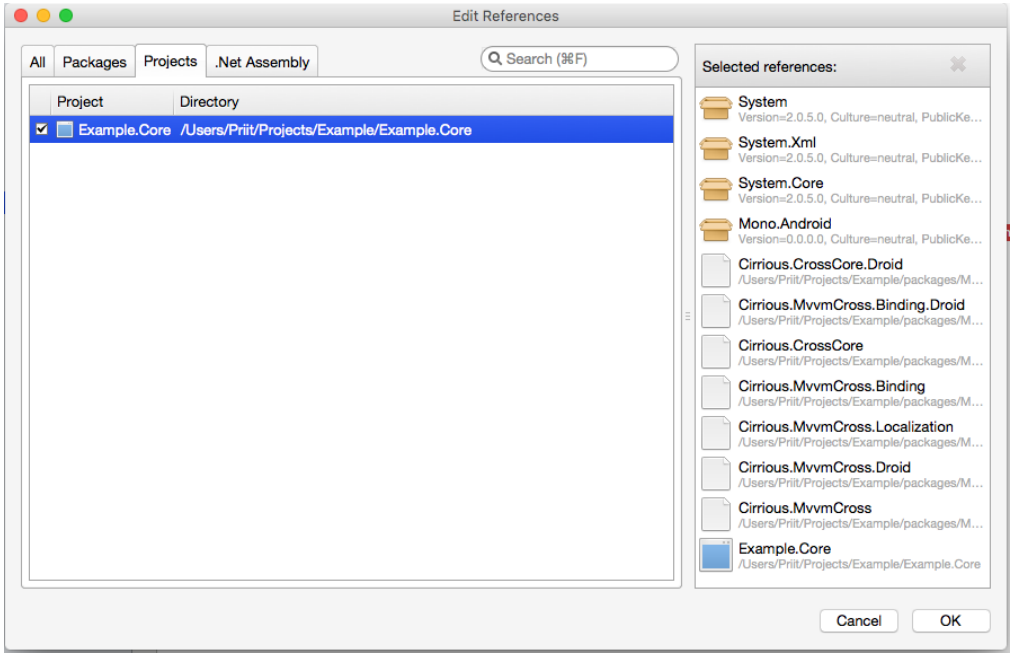


Joonis 12 Androidi projektile õige NuGet package valimine

Lisame Example.Droid projektile viite Example.Core projektile. Selleks vajutame parema klahviga Example.Droid olevale References kaustale ning valime Edit References.

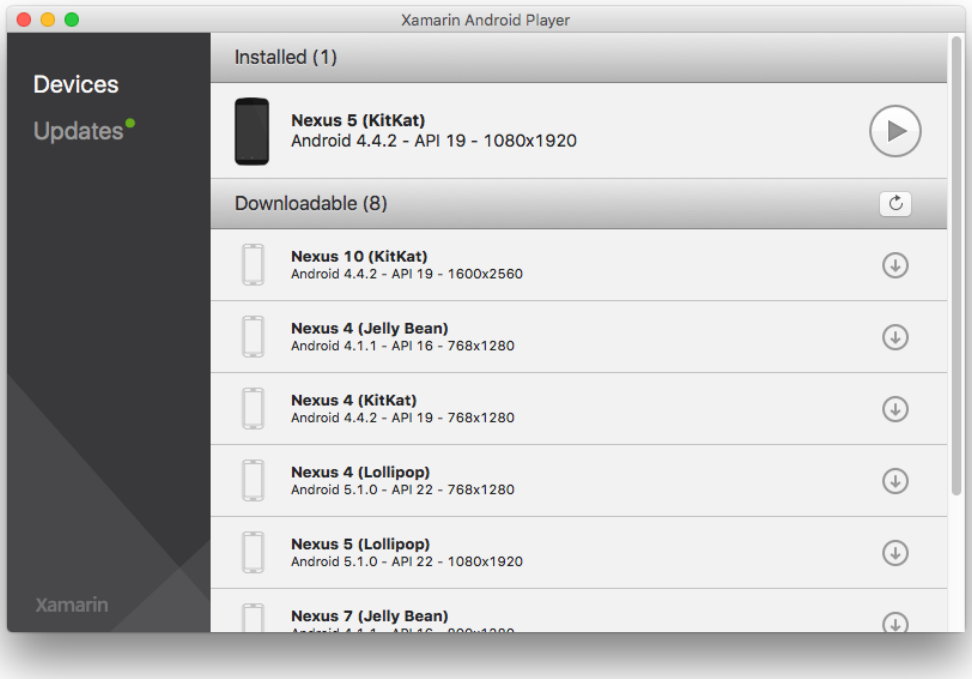


Valime Projects lahtri ning märgistame Example.Core projekti. Vajutame OK.



Joonis 13 Androidi projektis viite lisamine PCL projektile

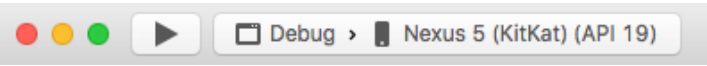
Avame Xamarin Player rakenduse. Laeme alla emulaatori millel on API level 19. Peale seda käivitame installeeritud emulaatori.



Joonis 14 Androidi emulaator

Eemaldame projektist faili nimega MainActivity.cs

Määrame Example.Droid stardi projektiks. Selleks vajutame parema klahviga Example.Droid projectile ning valime Make As Startup Project. Valime ülevalt installeeritud emulaatori ning vajutame Run nuppu.



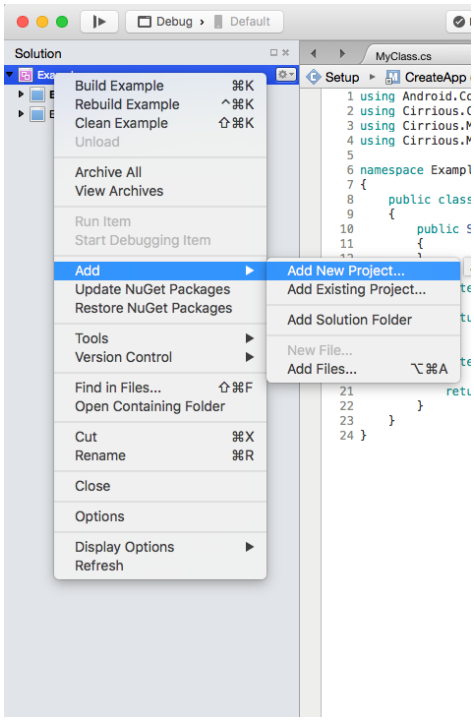
Joonis 15 Xamarin Studio käivitusriba

Kui kõik õnnestus ilmub emulaatorisse meie rakendus.

Nüüd on androidi projekt valmis arenduseks.

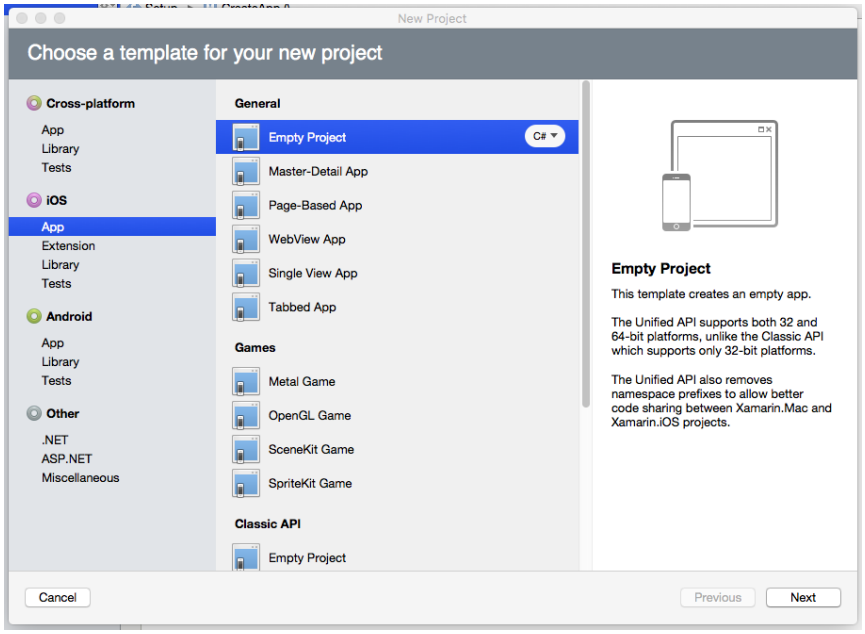
4.3 iOS projekti seadistus

Lisame solutionisse iOS-i projekti. Selleks vajutame parema klahviga Example solutionil ning valime Add -> Add New Project.



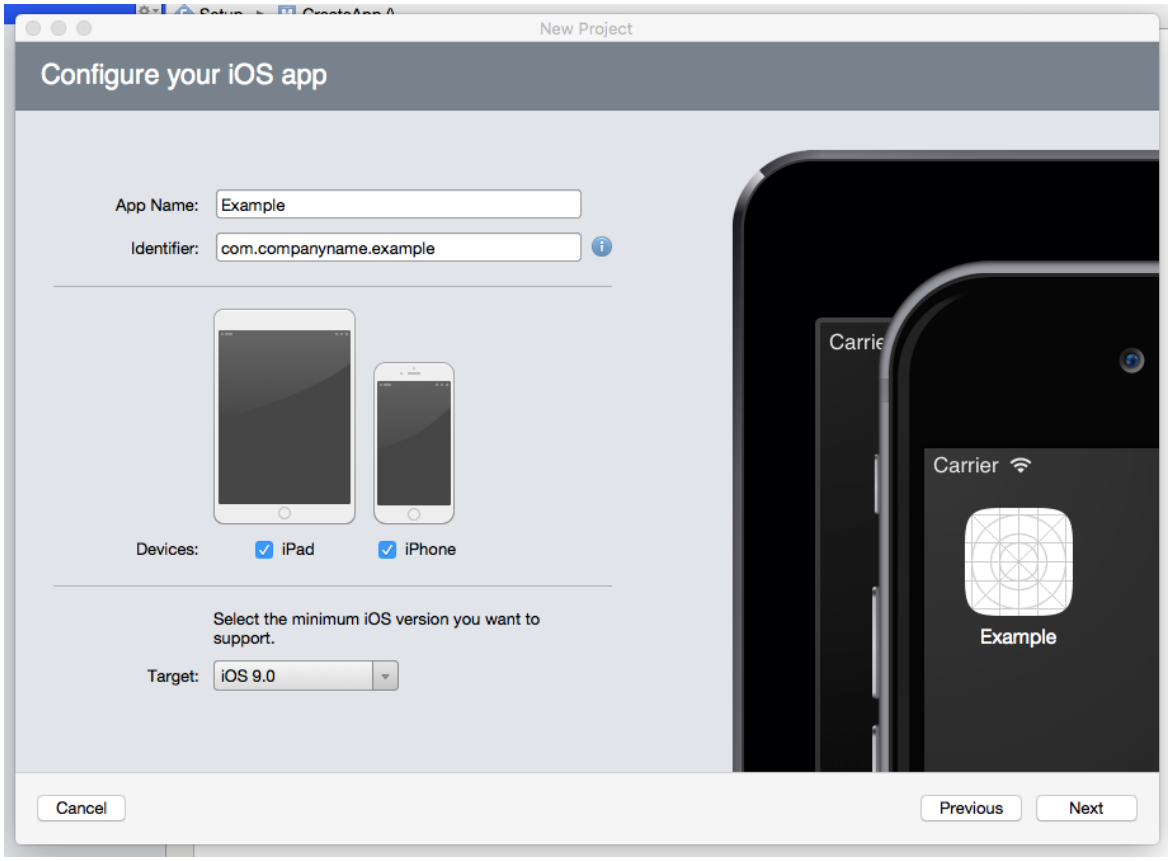
Joonis 16 iOS projekti lisamine

Valime iOS kategooriast App ja tüübiks Empty Project ning vajutamie Next.



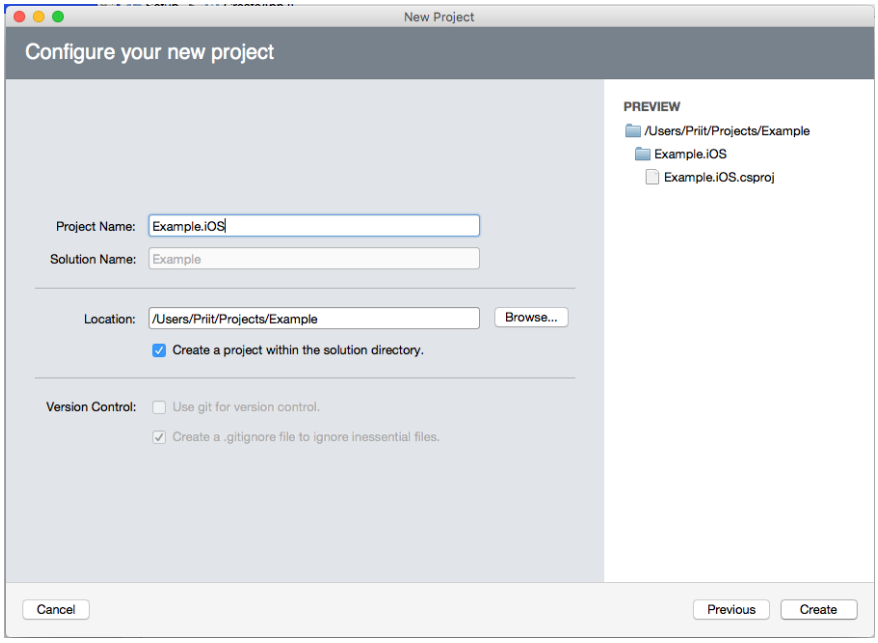
Joonis 17 iOS projekti tüübi valimine

Järgmises aknas peame andma rakendusele nime. Nimeks kirjutame Example. Identifier genereeritakse meie jaoks ise, kuid võime ka ise identifier-i määrata. Devices kategooria alt saame määrata kas soovime rakendust arendada iPadile, või iPhonele. Selles näites valime mõlemad. Targetiks tuleb määrata minimaalne iOS-i versioon millele arendada tahame. Selles näites valin kõige uuema milleks on iOS 9.0. Vajutan Next.



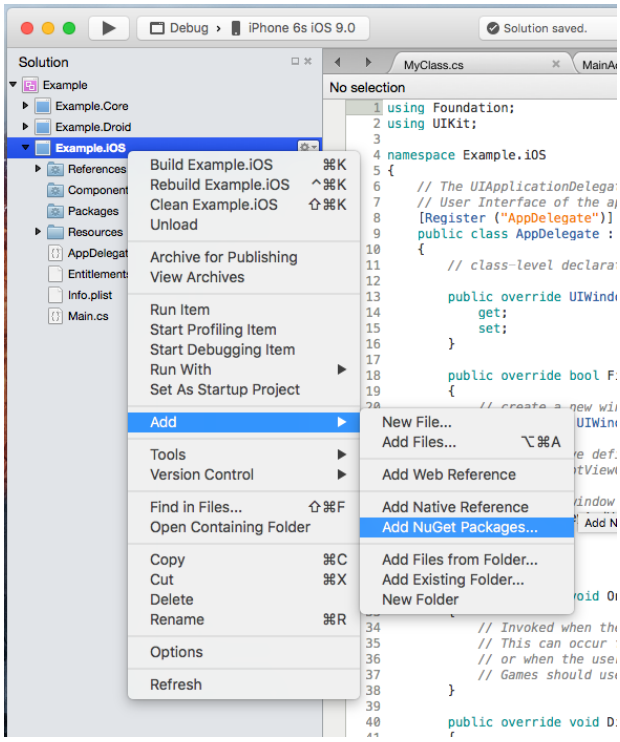
Joonis 18 iOS rakendusele nime andmine

Järgmises aknas tuleb anda projektile nimi. Projektile anname nimeks Example.iOS. Vajutame Create.



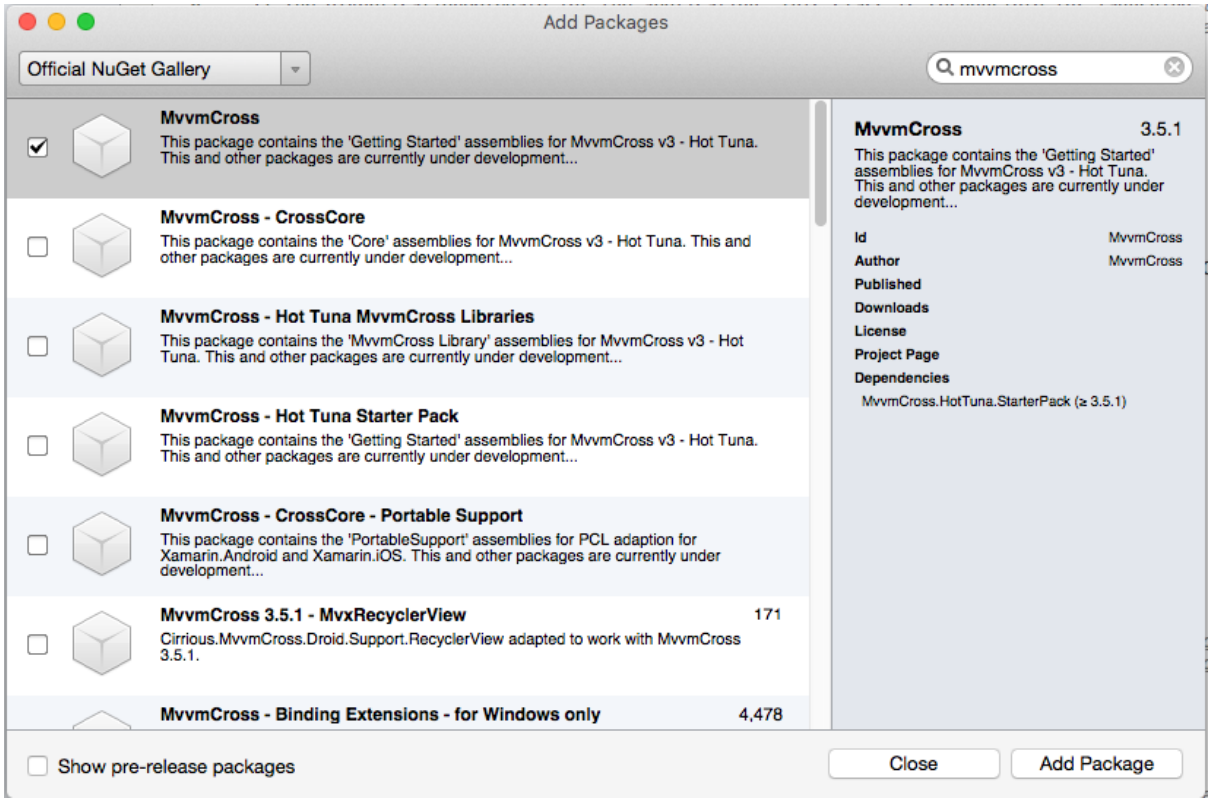
Joonis 19 iOS projektile nime andmine

Lisame iOS projektile MvvmCross raamistiku. Selleks kasutame NuGet Package Manager – i . Selleks vajutame parema klahviga Example.iOS projektil ning valime Add ->Add NuGet Packages...



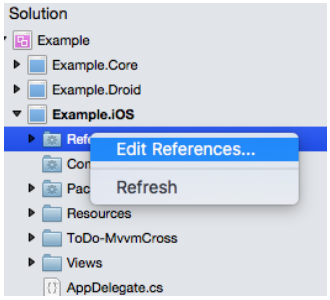
Joonis 20 iOS projektile NuGet package lisamine

Nuget Package Manager-i aknas kirjutame otsisõnaks MvvmCross. Valime allalaadimiseks paketi MvvmCross. Vajutame Add package.



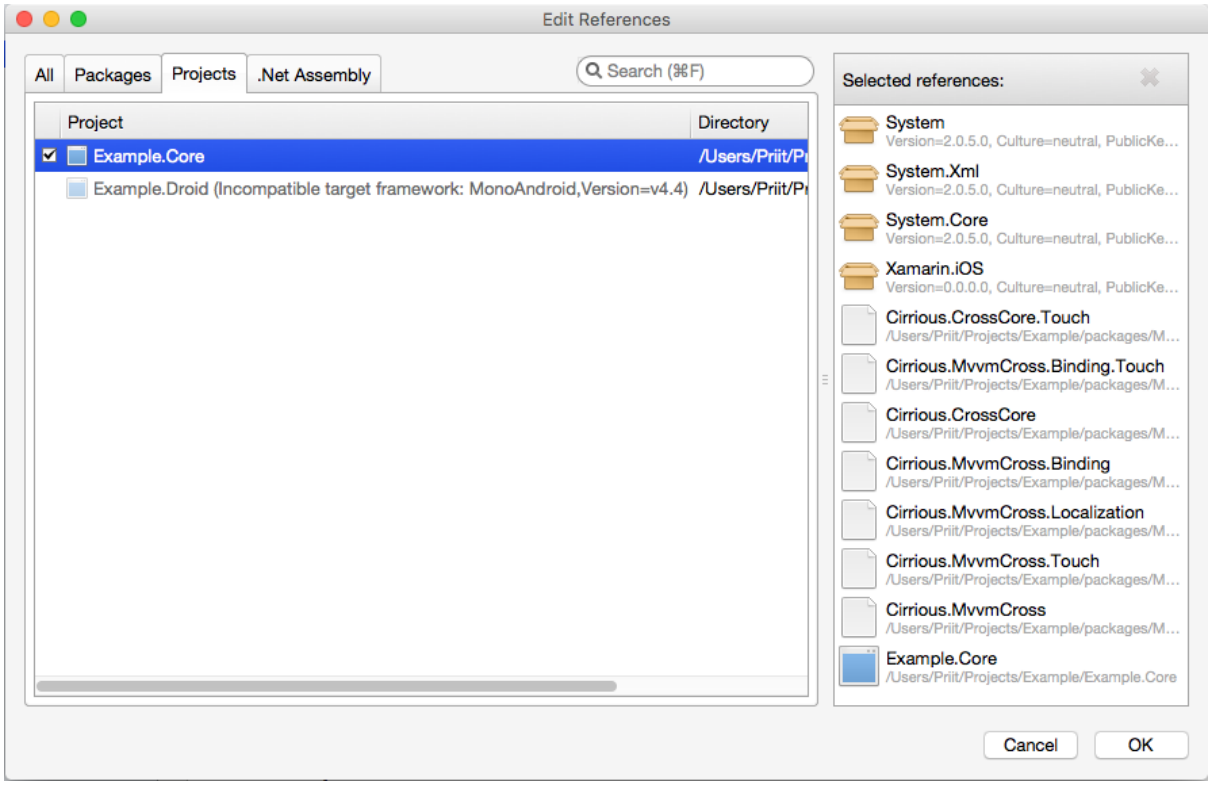
Joonis 21 iOS projektile õige NuGet package valimine

Lisame Example.iOS projektile viite Example.Core projektile. Selleks vajutame parema klahviga Example.iOS projektis References kaustale ning valime Edit References.



Joonis 22 iOS projekti viidete muutmine

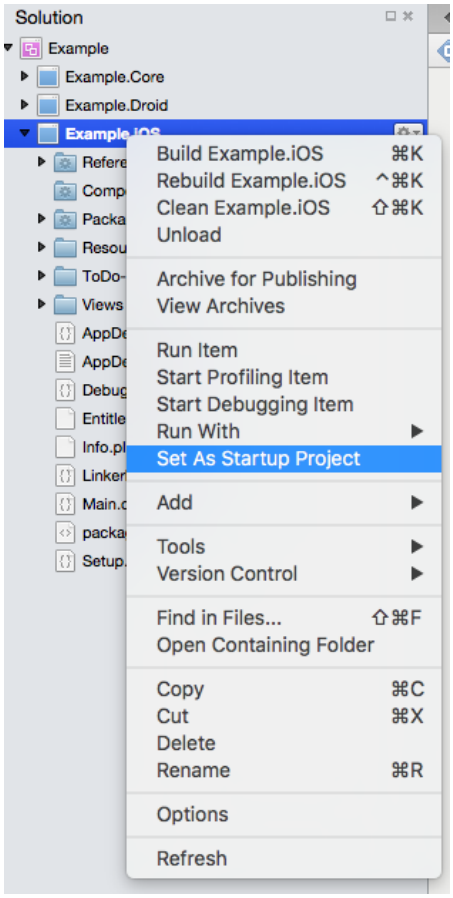
Valime Projects kategooria ning märgistame Example.Core projekti ning vajutame OK.



Joonis 23 iOS projektis viite lisamine PCL projektile

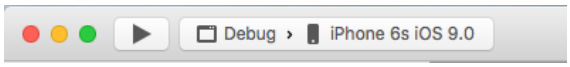
MvvmCross paketi lisamine muudab üldjuhul enamus vajalike failide sisu kuid vahel tuleb muuta mõnda faili ka käsitsi. iOS projektis tuleb MvvmCross raamistikuga kaasa kaust ToDo-MvvmCross. Sealt seest leiame juhendi failide muutmiseks. Antud juhul tuleb käsitsi muuta AppDelegate.cs sisu. Selleks on kaasa tulnud tekstifail AppDelegate.cs.txt. Kopeerime AppDelegate.cs.txt sisu AppDelegate.cs sisu asemele. Peale seda on iOS projekt valmis arenduseks.

Katsetame kas oleme olnud edukad, määrame Example.iOS projekti stardi projektiks. Selleks vajutame parema klahviga Example.iOS projektil ning valime Set As Startup Project.



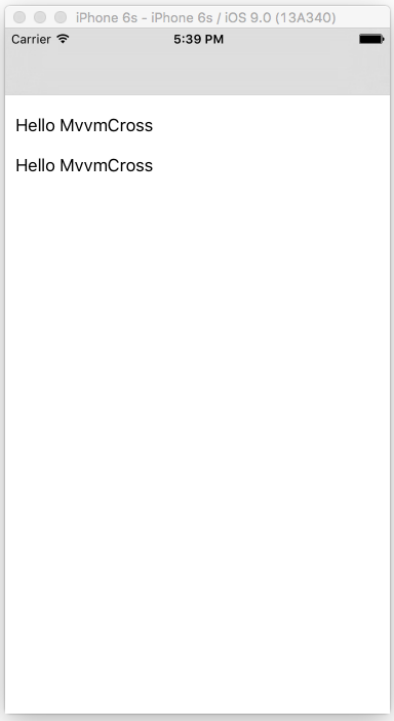
Joonis 24 iOS projekti stardiprojektiks määramine

Valime sobiva emulaatori milles rakendus käivitada ning vajutame Run nuppu.



Joonis 25 Xamarin Studio käivitusriba

Ekraanile peaks ilmuma emulaator ning käivituma meie arendatud näidisrakendus.



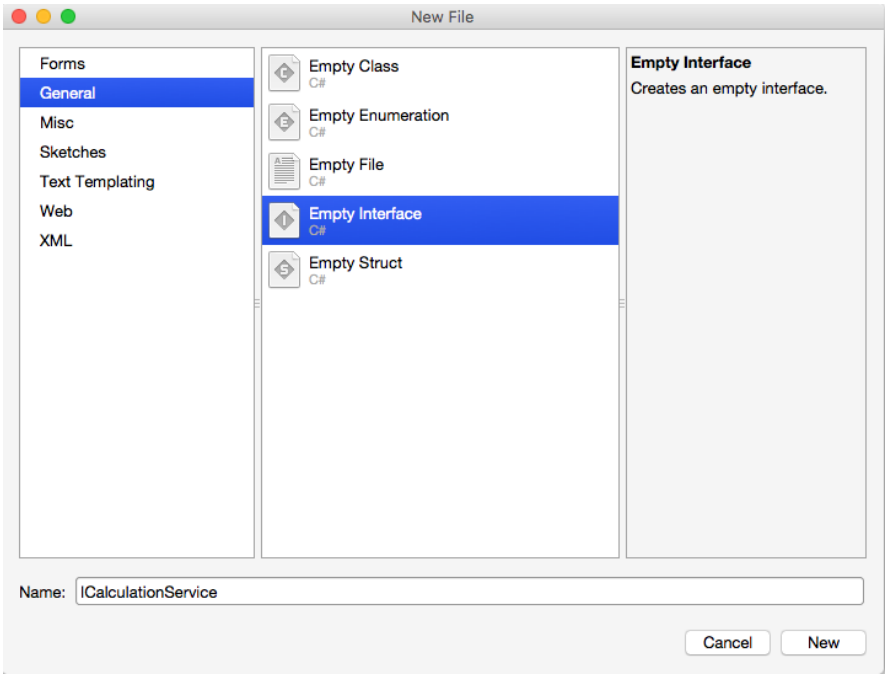
Joonis 26 iOS emulaator

5. Jootrahakalkulaator

Näite eesmärgiks on luua jootraha kalkulaator. Kakulaatoris saame määrata arve suuruse ning mitu protsenti sellest jootrahaks soovime anda.

5.1 Portable class library

Loome Example.Core projekti uue kausta nimega Services. Selleks vajutame parema klahviga Example.Core peale ning valime Add->New Folder. Selle kausta sisse lisame uue interface, selleks vajutame Services kausta peale parema klahviga ning valime Add->New File. Valime General kategooria alt Empty Interface. Nimeks kirjutame ICalculationService ning vajutame New.



Joonis 27 ICalculationService interface loomine

Loome interface meetodile Tip. Tüübiks double ning parameetriteks double subTotal ja double generosity. Peale seda peaks nägema välja ICalculationService selline:

```
using System;

namespace Example.Core
{
    public interface ICalculationService
    {
        double Tip(double subTotal, double generosity);
    }
}
```

Koodinäide 1 ICalculationService interface loomine

Loome Services kausta klassi nimega CalculationService. Selleks vajutame parema klahviga Services kasutale ja valime Add->New file. Valime General kategooriast Empty Class ning nimeks kirjutame CalculationService. Vajutame New.

CalculationService klassis lisame pärinemise ICalculationService-ile. Implementeerime ICalculationServices loodud meetodi ning kirjutame meetodisse jootraha arvutamiseks loogika.

Eemaldame automaatselt genereeritud konstruktori. CalculationService peaks välja nägema selline:

```
using System;

namespace Example.Core
{
    public class CalculationService : ICalculationService
    {
        public double Tip(double subTotal, double generosity)
        {
            return subTotal*generosity/100.0;
        }
    }
}
```

Koodinäide 2 CalculationService loomine

Järgmiseks avame faili nimega FirstViewModel.cs , mis asub ViewModels kaustas. Kustutame hetkel failis oleva property Hello. Teeme uue private readonly ICalculationService tüüpi objekti ning anname talle nimeks _calculationService.

```
private readonly ICalculationService _calculationService;
```

Koodinäide 3 ICalculationService objekti loomine

Loome FirstViewModel klassile konstruktori ning parameetritesse kirjutame ICalculationService calculationService. Konstruktoris initsialiseerime calculationService.

```
public FirstViewModel(ICalculationService calculationService)
{
    _calculationService = calculationService;
}
```

Koodinäide 4 FirstViewModel konstruktor

Lisame FirstViewModel faili neli double tüüpi property-t -Tip, Total, SubTotal ja Generosity:

```
private double _subTotal;
        public double SubTotal
        {
            get { return _subTotal; }
            set
            {
                _subTotal = value;
                RaisePropertyChanged(() => SubTotal);
            }
        }
        private double _generosity;
        public double Generosity
        {
            get { return _generosity; }
            set
            {
                _generosity = value;
                RaisePropertyChanged(() => Generosity);
            }
        }
        private double _tip;
        public double Tip
        {
            get { return _tip; }
            set
            {
                _tip = value;
                RaisePropertyChanged(() => Tip);
            }
        }
        private double _total;
        public double Total
        {
            get { return _total; }
            set
            {
                _total = value;
                RaisePropertyChanged(() => Total);
            }
        }
```

Koodinäide 5 SubTotal, Generosity, Tip, Total property-d

Loome meetodi Recalc :

```
private void Recalc()
        {
            Tip = _calculationService.Tip(SubTotal, Generosity);
            Total = SubTotal + Tip;
        }
```

Koodinäide 6 Recalc meetod

Lisame Recalc meetodi välja kutsumise SubTotal ja Generosity propertytesse :

```
private double _subTotal;
        public double SubTotal
        {
            get { return _subTotal; }
            set
            {
                _subTotal = value;
                RaisePropertyChanged(() => SubTotal); Recalc();
            }
```

```
        }

        private double _generosity;
        public double Generosity
        {
            get { return _generosity; }
            set
            {
                _generosity = value;
                RaisePropertyChanged(() => Generosity); Recalc();
            }
        }
```

Koodinäide 7 Recalc meetodi lisamine SubTotal ja Generosity propertytesse

Määrame SubTotal ja Generosity property-le konstruktoris algväärtused ning kutsume välja meetodi Recalc.

```
public FirstViewModel(ICalculationService calculationService)
{
    _calculationService = calculationService;
    _generosity = 20;
    _subTotal = 100;
    Recalc();
}
```

Koodinäide 8 Property-te algväärtustamine ja Recalc meetodi välja kutsumine konstruktoris

Peale seda peaks FirstViewModel nägema välja järgmiselt:

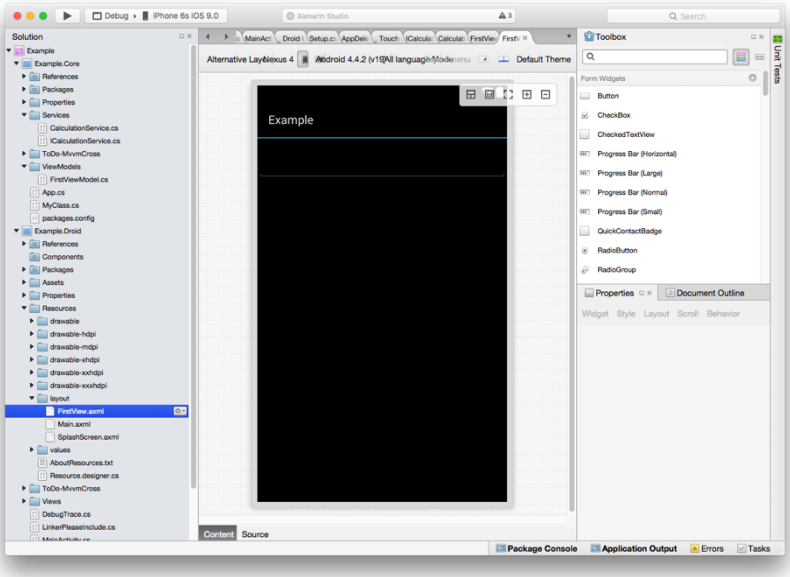
```
using Cirrious.MvvmCross.ViewModels;

namespace Example.Core.ViewModels
{
    public class FirstViewModel
        : MvxViewModel
    {
        private readonly ICalculationService _calculationService;

        public FirstViewModel(ICalculationService calculationService)
        {
            _calculationService = calculationService;
            _generosity = 20;
            _subTotal = 100;
            Recalc();

        }

        private void Recalc()
        {
            Tip = _calculationService.Tip(SubTotal, Generosity);
            Total = SubTotal + Tip;
        }


        private double _subTotal;
        public double SubTotal
        {
            get { return _subTotal; }
            set
            {
                _subTotal = value;
                RaisePropertyChanged(() => SubTotal); Recalc();
            }
        }
```

```
        private double _generosity;
        public double Generosity
        {
            get { return _generosity; }
            set
            {
                _generosity = value;
                RaisePropertyChanged(() => Generosity); Recalc();
            }
        }

        private double _tip;
        public double Tip
        {
            get { return _tip; }
            set
            {
                _tip = value;
                RaisePropertyChanged(() => Tip);
            }
        }
        private double _total;
        public double Total
        {
            get { return _total; }
            set
            {
                _total = value;
                RaisePropertyChanged(() => Total);
            }
        }

    }
}
```

Koodinäide 9 FirstViewModel tervikuna

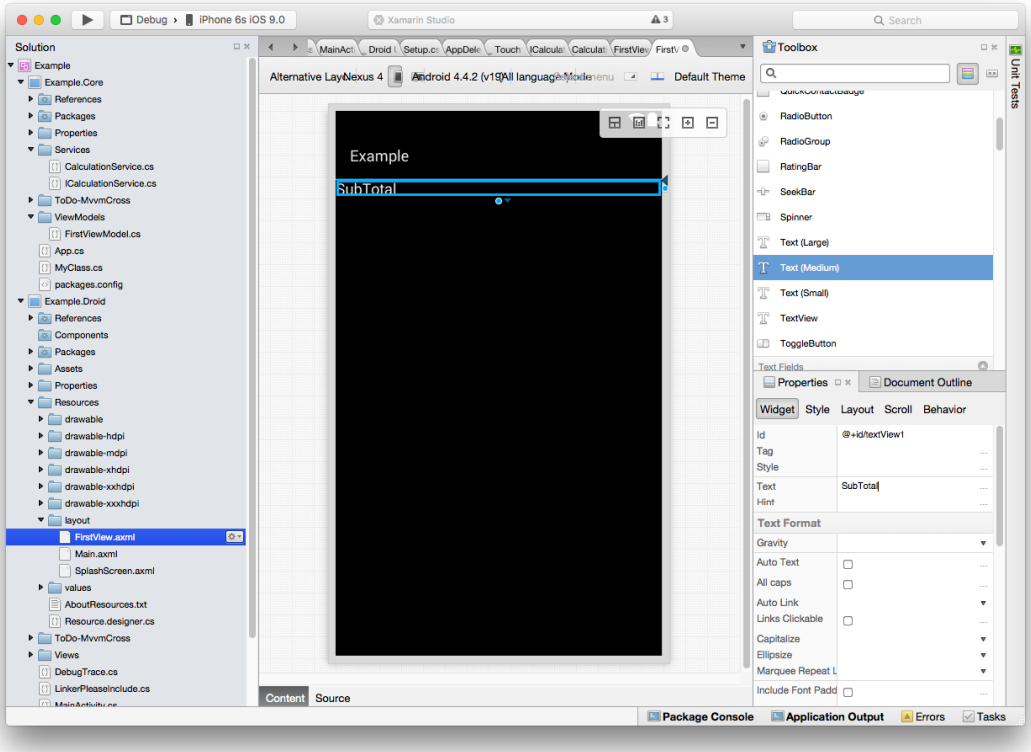
5.2 Androidi disain

Loome androidi projektile kasutajaliidese. Avame Example.Droid projektist faili nimega FirstView.axml, mis asub Resources -> Layout kaustas.



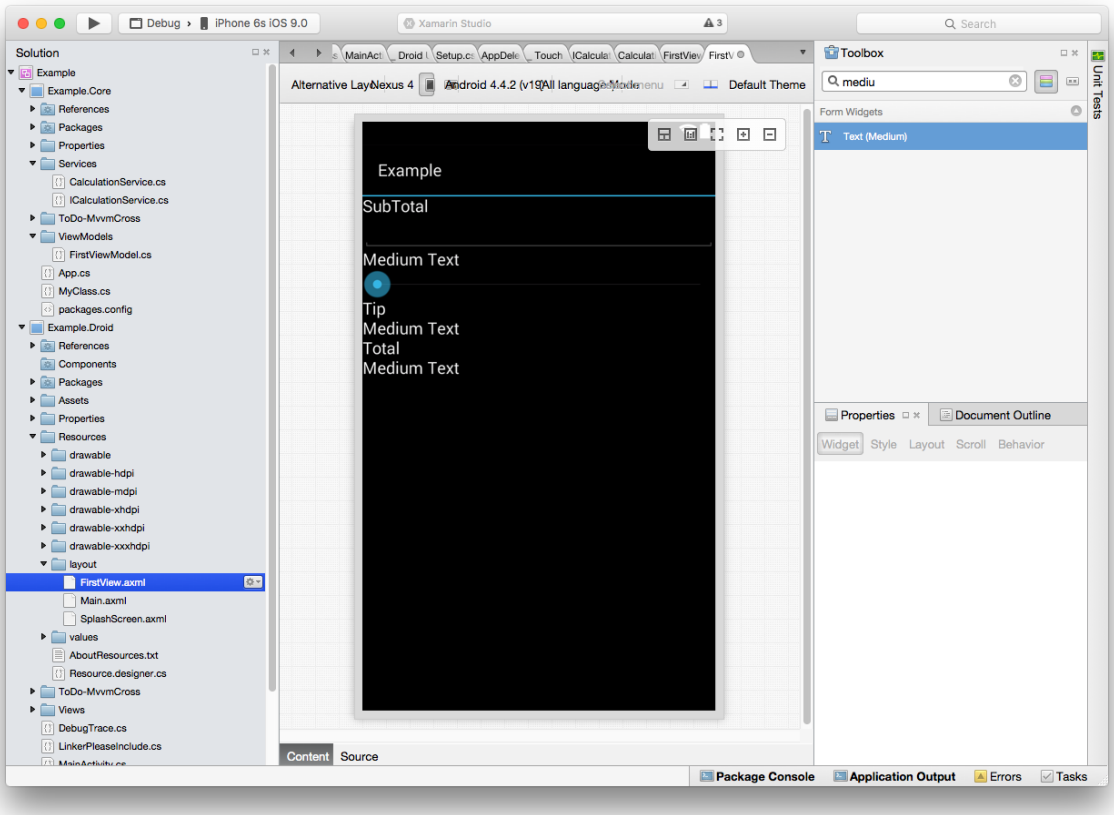
Joonis 28 Xamarin Studio androidi disaineri ekraanitõmmis

Kustutame hetkel ekraanil olevad EditTexti ja TextView vajutades nendele peale ning peale seda delete klahvi. Vasakult Toolbox-ist lohistame ekraanile form widget kategooria alt Text(Medium). Paremal pool asuvas aknas properties paanil otsime üles rea kuhu on kirjutatud text. Muudame Medium Text ümber SubTotal-iks.



Joonis 29 Xamarin Studio androidi disaineri tõmmis

Seejärel lohistame ekraanile toolboxist Text Fields kategooria alt plain text. Peale seda lohistame ekraanile veel ühe Medium Text-i ja SeekBar-i. Nimetame nimetame teise Medium Texti ümber Generosity-ks. Lisame veel neli Medium Texti. Esimese nimeks kirjutame Tip ja kolmada nimeks Total



Joonis 30 Xamarin Studio androidi disaineri ekraanitõmmis

Peale seda lähme Source vaatele, kuhu saab alt vasakult valides Source. Seome elemendid väärtustega.

Selleks lisame EditTexti parameetritele:

```
local:MvxBind="Text SubTotal"
```

SeekBarile parameetritele lisame:

```
local:MvxBind="Progress Generosity"
```

Tip-i TextView-le järgnevale TextView parameetritele lisame:

```
local:MvxBind="Text Tip"
```

Total TextView-le järgnevale TextView parameetritele lisame:

```
local:MvxBind="Text Total"
```

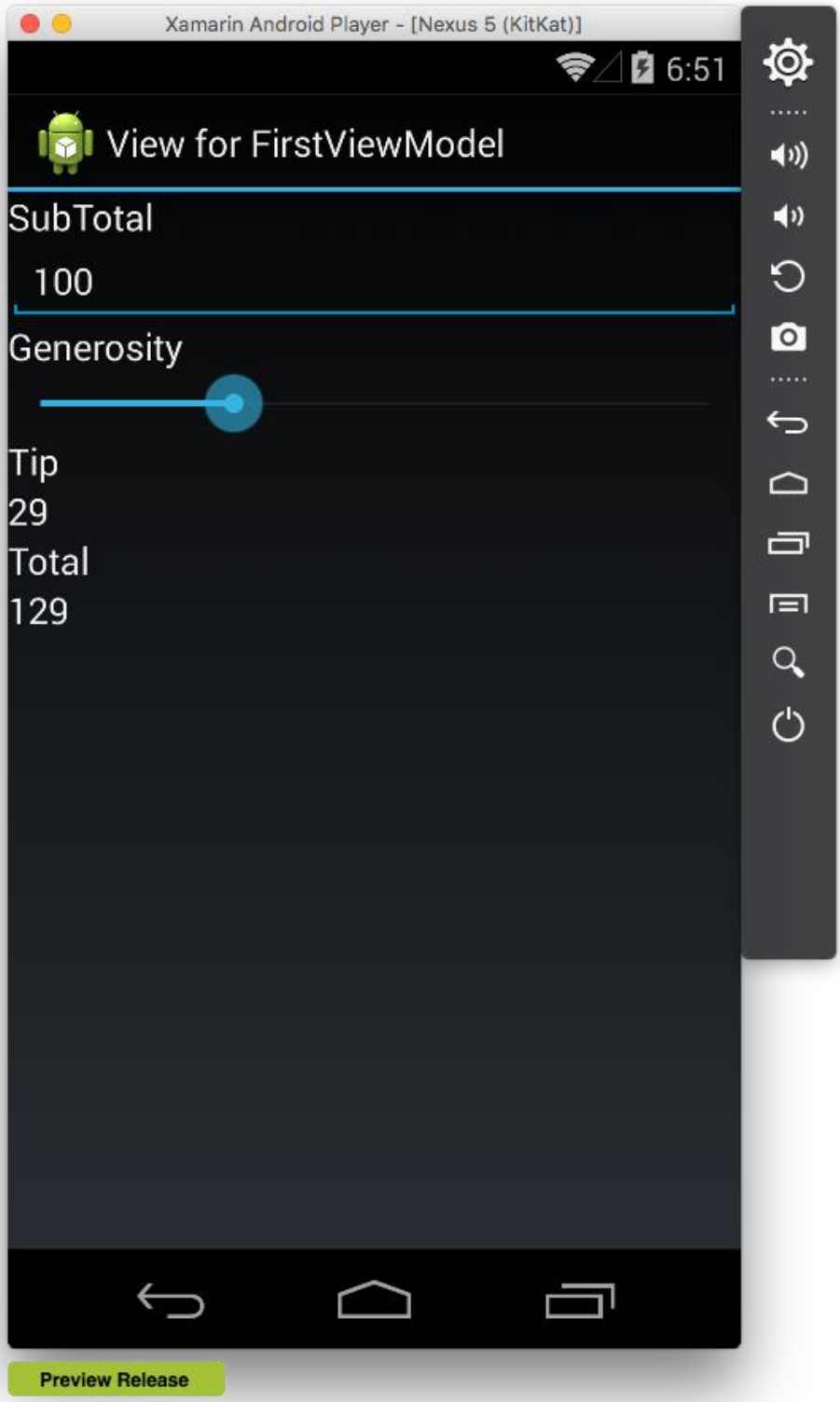
FirstView.axml peaks välja nägema järgmiselt:

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

```
        android:text="SubTotal"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/textView1" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        local:MvxBind="Text SubTotal"
        android:id="@+id/editText1" />
    <TextView
        android:text="Generosity"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/textView2" />
    <SeekBar
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        local:MvxBind="Progress Generosity"
        android:id="@+id/seekBar1" />
    <TextView
        android:text="Tip"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/textView3" />
    <TextView
        android:text="Medium Text"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        local:MvxBind="Text Tip"
        android:id="@+id/textView4" />
    <TextView
        android:text="Total"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/textView5" />
    <TextView
        android:text="Medium Text"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        local:MvxBind="Text Total"
        android:id="@+id/textView6" />
</LinearLayout>
```

Koodinäide 10 FirstView.axml tervikuna

Peale seda määrame startup projektiks Example.Droid projekti ning käivitame programmi. Emulaatorisse peaks ilmuma jootrahakalkulaator.



Joonis 31 Androidi emulaatori ekraanitõmmis näidisrakendusest

5.3 iOS disain

Loome iOS rakendusele kasutajaliidese. Avame Example.IOS projektis faili FirstView.cs .

Kustutame ViewDidLoad meetodis üleliigse . Alles peaks jääma:

```
public override void ViewDidLoad()
        {
            View = new UIView { BackgroundColor = UIColor.White };
            base.ViewDidLoad();
        }
```

Koodinäide 11 ViewDidLoad meetod

Lisame viite System.Drawing teegile:

```
using System.Drawing;
```

Lisame kasutajaliidese elemendid :

```
var label = new UILabel(new RectangleF(10, 80, 300, 40));
            label.Text = "SubTotal";
            Add(label);

            var subTotalTextField = new UITextField(new Rectangle(10, 120, 300, 40));
            Add(subTotalTextField);

            var label2 = new UILabel(new RectangleF(10, 160, 300, 40));
            label2.Text = "Generosity";
            Add(label2);

            var slider = new UISlider(new RectangleF(10, 200, 300, 40));
            slider.MinValue = 0;
            slider.MaxValue = 100;
            Add(slider);

            var label3 = new UILabel(new RectangleF(10, 240, 300, 40));
            label3.Text = "Tip";
            Add(label3);

            var tipLabel = new UILabel(new RectangleF(10, 280, 300, 40));
            Add(tipLabel);

            var label4 = new UILabel(new RectangleF(10, 320, 300, 40));
            label4.Text = "Total";
            Add(label4);

            var totalLabel = new UILabel(new RectangleF(10, 360, 300, 40));
            Add(totalLabel);
```

Koodinäide 12 Kasutajaliidese elemendid

Lisame viite Example.Core.ViewModels teegile:

```
using Example.Core.ViewModels;
```

Seome elemendid väärtustega:

```
var set = this.CreateBindingSet<FirstView, FirstViewModel>();
            set.Bind(subTotalTextField).To(vm => vm.SubTotal);
            set.Bind(slider).To(vm => vm.Generosity);
            set.Bind(tipLabel).To(vm => vm.Tip);
            set.Bind(totalLabel).To(vm => vm.Total);
            set.Apply();
```

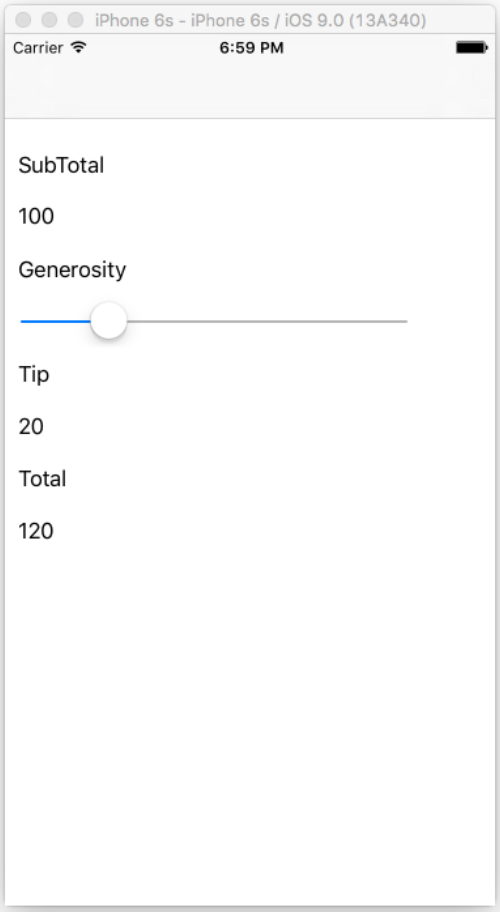
Koodinäide 12 Bindingute loomine

Kogu FirstView.cs koos peaks nägema välja järgmiselt

```
using Cirrious.MvvmCross.Binding.BindingContext;
using Cirrious.MvvmCross.Touch.Views;
using CoreGraphics;
using Foundation;
using ObjCRuntime;
using UIKit;
using System.Drawing;
using Example.Core.ViewModels;

namespace Example.iOS.Views
{
    [Register("FirstView")]
    public class FirstView : MvxViewController
    {
        public override void ViewDidLoad()
        {
            View = new UIView { BackgroundColor = UIColor.White };
            base.ViewDidLoad();

            var label = new UILabel(new RectangleF(10, 80, 300, 40));
            label.Text = "SubTotal";
            Add(label);

            var subTotalTextField = new UITextField(new Rectangle(10, 120, 300, 40));
            Add(subTotalTextField);

            var label2 = new UILabel(new RectangleF(10, 160, 300, 40));
            label2.Text = "Generosity";
            Add(label2);

            var slider = new UISlider(new RectangleF(10, 200, 300, 40));
            slider.MinValue = 0;
            slider.MaxValue = 100;
            Add(slider);


            var label3 = new UILabel(new RectangleF(10, 240, 300, 40));
            label3.Text = "Tip";
            Add(label3);


            var tipLabel = new UILabel(new RectangleF(10, 280, 300, 40));
            Add(tipLabel);

            var label4 = new UILabel(new RectangleF(10, 320, 300, 40));
            label4.Text = "Total";
            Add(label4);

            var totalLabel = new UILabel(new RectangleF(10, 360, 300, 40));
            Add(totalLabel);

            var set = this.CreateBindingSet<FirstView, FirstViewModel>();
            set.Bind(subTotalTextField).To(vm => vm.SubTotal);
            set.Bind(slider).To(vm => vm.Generosity);
            set.Bind(tipLabel).To(vm => vm.Tip);
            set.Bind(totalLabel).To(vm => vm.Total);
            set.Apply();

        }

    }
}
```

Koodinäide 12 FirstView.cs kood tervikuna

Määrame stardi projektiks Example.iOS ning käivitame programmi. Emulaatorisse peaks ilmuma jootrahakalkulaator



Joonis 32 iOS rakenduse ekraanitõmmis koos näidisrakendusega

Ülesanded

1. Lisa CalculationService-sse funktsioon mis teisendab summat mõnda teise rahaühikusse
2. Lisa disainile eraldi väli teisendatud summa näitamiseks.
3. Lisa FirstViewModelisse property teisendatud summa jaoks.
4. Lisa Recalc meetodisse vajalik kood property uuendamiseks
5. Ühenda disain koodiga kastuades bindingut

Kokkuvõte

Antud seminaritöös tutvustati õppijale Xamarini ja MvvmCrossi algset projektide seadistamist ning arendasime koos ka esimese rakenduse jootrahakalkulaatori näitel. Juhendi lõpus on ka enesekontrolliks ülesanded.

Eesmärgiks oli teha õppematerjal mida saab kasutada arendaja kes pole varem kokku puutunud Xamariniga kuid omab elementaarteadmisi C# keeles. Juhend peaks andma algteadmised Xamarini ja MvvmCrossi kasutamiseks.

Autori jaoks osutus kõige raskemaks inglise keelsete väljendite tõlkimine eesti keelde. Lisaks oli keeruline kirja panna menüüdes liikumist ja nuppude vajutamist. Töösse on lisatud võimalikult palju pilte, et kõik nupuvajutused ning menüüvalikud oleksid arusaadavad.

Kasutatud allikad

Microsoft (2012). The MVVM Pattern. Kasutamise kuupäev 1.11.2015. Allikas: https://msdn.microsoft.com/en-us/library/hh848246.aspx

Lodge, S (2015). MvvmCross. Kasutamise kuupäev 1.11.2015. Allikas: https://github.com/MvvmCross/MvvmCross

.NET Foundation (2015). Nuget. Kasutamise kuupäev 1.11.2015. Allikas: https://www.nuget.org/

Microsoft (2015). .NET. Kasutamise kuupäev 1.11.2015. Allikas: https://msdn.microsoft.com/en-us/vstudio/aa496123.aspx

Xamarin(2015). Create native iOS, Android, Mac and Windows apps in C#. Kasutamise kuupäev 1.11.2015. Allikas: https://www. xamarin.com/platform

r