HAAGA-HELIA
University of Applied Sciences

Developing Sopima Cross-platform Mobile Application With Xamarin

Svetlana Borisenkova

Bachelor's Thesis
Degree Programme in
Business Information
Technology
2015

HAAGA-HELIA
University of Applied Sciences

Abstract

22.2.2015

Author(s)
Svetlana Borisenkova

Degree programme
Business Information Technology

Thesis title
Developing Sopima Cross-platform Mobile Application with Xamarin

Number of pages and appendix pages
47+13

Every year more and more mobile devices appear in the market. The demand for useful and easy to use mobile applications is growing fast. Businesses more often use mobile applications as a promotional tool and also as a mean to gain customer's trust. This means that the demand for rapid mobile application development will increase accordingly since it is not easy to keep up with such demand for the applications.

The main goal of the thesis project is to develop a cross-platform mobile application for an existing web application that provides contract management services. The application has to work on Windows and Android platforms by sharing as much code as possible using the latest technologies. The iOS platform is out of the thesis scope. The study also focuses on the Xamarin platform and reasons out whether it is a good choice for fast development of native mobile applications. Its key advantages over other mobile development frameworks are specified.

The actual work on the thesis project started in October 2014 and ended in February 2015. The Rapid Application Development method was used to achieve the project goals the as well as the author's personal programming skills and knowledge.

The thesis results in working Android and Windows phone mobile applications. The applications will allow users to have access to the web-based contract management system via their mobile phones which makes contract browsing, viewing, sharing and searching much faster, therefore improving customer experience.

Keywords
Mobile application, Xamarin, cross-platform

Table of contents

Abbreviations and terms

AAD	Azure Active Directory
App	Application
AXML	Active Extensible Markup Language
DLL	Dynamic Link Library. A DLL is a library that contains code and data that can be used by more than one program at the same time (Microsoft, 2015)
HttpClient	The HttpClient library is a set of APIs for .NET which provides a flexible and extensible way to access all things exposed through HTTP. (Microsoft, 2015)
Hyper-V VM	The Hyper-V role in Windows Server 2008 R2 and Windows Server 2008 provides software infrastructure and basic management tools that you can use to create and manage a virtualized server computing environment (Microsoft, 2015)
OS	Operating System
Oracle VM VirtualBox	The world's most popular cross-platform virtualization software enables you to run multiple operating systems on your Mac, PC, Linux, or Oracle Solaris machine (Oracle, 2015)
SDK	Software development kit
SPA	Single-Page Applications are Web apps that load a single HTML page and dynamically update that page as the user interacts with the app. (Microsoft, 2015)
UI	User Interface
Web API	A framework that allows building and exposing HTTP services that can be accessed from different platforms, including mobile
XAML	Extensible Application Markup Language

1 Introduction

It is not a secret that nowadays there are a lot of technologies that allow developers to develop really powerful web applications which are able to satisfy the needs of even the most demanding users. The world has shifted a lot towards using mobile devices instead of old familiar computers and laptops as the source for exploring the Internet.

Quite new trend in web application development called SPA has been around for a while and was able to conquer hearts of many developers. These technologies provide developers with a possibility of developing quite a good responsive UI with high data processing speed. Such applications can be easily used on any device. Still, it cannot compete with native applications for tablets and mobile phones.

This project is about developing a mobile application for an existing web application using the latest technologies. Nowadays, a mobile device has become more than a means for calling and sending messages. People are using their mobile phones more and more to fulfill their everyday needs; therefore, the mobile market is getting bigger and bigger every year. More developers and simply armature are getting into the developing mobile applications to fulfill the growing demand. Famous IT companies are working toward making the life of mobile developers easier. Not that long ago, a totally new platform for developing cross-platform mobile applications called Xamarin was introduced. This platform allows the developers to use only one programming language C# to create mobile applications that will work on different mobile platforms such as: Android, Windows, and IOs. It will be used in order to reach the objectives of the thesis project.

1.1 Need for the application

Sopima is contract management software that provides centralized and secure contract repository, easy contract creation as well as efficient contract handling. Sopima has been developed using the latest technologies. It has been picking up pace quite fast and more and more customers are getting interested in using Sopima in their businesses. There are quite a few strong competitors in the area Sopima operates so it is very important to keep up and be better than they are. One of the keys to win the competition is to keep already existing customers satisfied and get the others interested. In order to do that it is necessary to fulfill needs and demands of the customers and try to predict what they would want. This issue became one of the reasons for developing Sopima mobile application. It will simply allow customers to have their contract with them all the time and be able to search for the needed ones as well as see the latest events in the organization they work

for. Therefore, it will improve the user experience overall and get more creditability from them.

In order to be competitive and fulfill customers' needs, a question of developing a mobile application has aroused. The initial plan was to develop an application that would target only one mobile platform (most likely Windows). After the Xamarin platform has been introduced, it was decided to take a risk and target several mobile platforms at the same time along with exploring this new technology.

1.2 Goals

The goal that has been set for the project is to develop a cross-platform mobile application that is hosted in cloud and is targeting Windows and Android mobile platforms using Xamarin. That will help the customers to improve their user experience while using Sopima services.

At this stage, the application must have a simple UI, basic functionality which includes: log in into the system using Microsoft Azure active directory, displaying latest organization news, displaying contract folders and their contents, and it must be possible to view a contract and search for contracts.

Also this project aims at exploring possibilities and advantages of using Xamarin mobile platform that can be considered as the main technology for developing mobile application as a new service that the company could provide for their customers.

1.3 Scope of the thesis

The scope of the project is to develop a mobile application for Windows and Android mobile platforms. Because of the time restraints and hardware lacking for developing an iOS mobile application has been decided to narrow the scope of the project to targeting at two platforms out of three. Also inside the scope is to introduce a new technology called Xamarin and basics of Windows Azure active directory authorization process.

1.4 Out of the scope

Out of scope is the development of the existing software that serves as the base for the mobile application as well as further development of the mobile application and its publishing to the app stores. Development of the iOS application has also been excluded from the scope.

1.5 Methods

There is quite a list of software development methodologies to choose from, such as: Scrum, Kanban, RAD (Rapid Application Development) just to name a few. It is considered a good practice to develop applications using one or several development methods. It is important to choose the most suitable method in order for the application development to succeed. Each development method has its own requirements to be met before it will take an application to the success. The methods have been studied and the following aspects were taken into consideration before choosing the main development methodology: project scope and size, development team, available initial data, project decisions, and technical requirements.

After all the aspects have been studied, the RAD method has been chosen for the project in order to achieve the set goals and as it seems to be the most suitable method. Rapid application development is a software development methodology that uses minimal planning in favor of rapid prototyping. A prototype is a working model that is functionally equivalent to a component of the product. In order for RAD to work the below listed criteria must be fulfilled (ProjectManagement, 2015).

- Project scope - the goals of the project scope must be well defined and narrow.
- Project data - the data for the project partially or fully exists.
- Project decisions – the decision making does not require a lot of people.
- Project team – the team can be less than six people.
- Project technical architecture – the technologies that will be used are well knows and tested. And the architecture must be clear to implement.
- Project technical requirements – must be reasonable and the used technologies should not be used to their full capacity.

According to the requirements above, five out of six are fulfilled. The exception is that the technologies that are used for the application development are partially known and tested. RAD means fast development and a few technics used to speed the process such as prototyping that initiates the development of a working prototype and improving it along the way. Additionally, by using RAD developers should focus on delivery of the result within the set time box which implies that even the scope of the project can be reduced in order to deliver the result. Among the other technics the iteration is used as approach to improve the application prototype and create a successful application.

Based on the knowledge about RAD, it is possible to say that this is an appropriate method to use for reaching the goals of the project

2 Theoretical background

The theoretical background of the project gives the insight of what kind of principals and technologies will be applied during the development process. Besides, it gives some understanding of the project architecture.

2.1 Xamarin platform

Xamarin is a software platform that provides developers with capability of building native cross platform mobile applications without compromising the native user experience. Developer can use Xamarin studio or Xamarin integrated with Visual studio as their development environment. Currently it supports development of Windows store, Android, iOS and Mac applications using C# programming language, and either Mac or Windows for building them. Simply speaking, now it is possible to do anything a developer would do in Objective-C, Swift or Java but in C#. Not to mention that using Xamarin insights tool allows crash and issue reporting as well as user sessions monitoring.

What is meant by “not compromising the native user experience” is that Xamarin provides the full access to the native APIs and user interface controls to give the most native filling to the apps. Also one of the major benefits of Xamarin is the possibility to share your code between platforms by using either PCL (Portable Class Libraries) or Shared project. Certain aspects of code sharing should be considered before choosing between the two options. Of course, regardless of how much code is possible to share a developer will still have to write a platform specific code. The main mantra when developing with Xamarin is sharing as much code as possible.

Moreover, Xamarin is a commercial product, so you will have to purchase two different licenses for Android and iOS, there is no need to purchase a license for Windows and there is a free version with limited features.

2.2 Code sharing

Code sharing is one of the main reasons for using Xamarin as the main technology in development the applications. When choosing a code sharing method, every developer should pay this subject a special attention. There are two major options available in Xamarin - Shared Asset projects and Portable Class Library projects.

Shared Projects use a single set of files and offers a quick and simple way in which to share code within a solution and generally employs conditional

compilation directives to specify code paths for various platforms that will use it. (Xamarin inc., 2015)

Xamarin suggests that it is a good idea to use shared project when the library will not be shared. If the library sharing is desired then PCL is a better choice. Though there is no strict rules just consider what the needs of the application are and which method fulfils them the most. Besides, it is possible to use both in one project and share your code even further to create Windows 8.1 applications.

In this project the preference was given to the PCL, therefore, it will be discussed further in more details since it is one of the crucial parts of the application.

2.2.1 Portable Class Library

Portable Library projects are automatically enabled in Xamarin Studio on OS X, and are built-in to Visual Studio 2013. When you create a 'regular' Application Project or a Library Project, the resulting DLL is restricted to working on the specific platform it is created for. This prevents you from writing an assembly for a Windows Phone app, and then re-using it on Xamarin.iOS and Xamarin.Android.

When you create a Portable Class Library, however, you can choose a combination of platforms that you want your code to run on. The compatibility choices you make when creating a Portable Class Library are translated into a "Profile" identifier, which describes the platforms the library supports.

The Table 1 below shows very well that Xamarin supports all profiles that are in VS 2013 and also the number of available features which means you can easily use them while building a platform specific application. (Xamarin inc., 2015)

Table 1. Features and profiles (Xamarin inc., 2015)

Feature	.NET Framework	Windows Store Apps	Silverlight	Windows Phone	Xamarin
Core	Y	Y	Y	Y	Y
LINQ	Y	Y	Y	Y	Y
IQueryable	Y	Y	Y	7.5 +	Y
Serialization	Y	Y	Y	Y	Y
Data Annotations	4.0.3 +	Y	Y		Y

According to the official Xamarin web site the PCL has the following pros and cons.

Benefits

- The code written in PCL can be consumed by other projects and libraries.
- Code refactoring affects all the code in the solution whether it is in the platform-specific project or in the PCL.
- The PCL project supports referencing by other projects in a solution. (Xamarin inc., 2015)

It is also very useful if using the MVVM pattern for developing an app, so the **ViewModel** will be created in the PCL project and shared between platform and it reduces the amount of code that needs to be written and of course it is easier to support.

Disadvantages

- Not enough examples about best practices of using PCL.
- Limitation to only APIs which are available on all the platforms.

The last issue can be overcome by using the Provider pattern or Dependency Injection to code the actual implementation in the platform projects against an interface or base class that is defined in the Portable Class Library.

2.3 Xamarin.Android

Let's have a closer look at how it becomes possible to develop native android applications using C# and get full advantage of using .Net framework.

The Android platform is very popular nowadays. The biggest share of mobile devices is android based devices, so it is very important to get the understanding on how it works and, moreover, how it is compiled and runs against .Net APIs.

Android is based on Linux operating system and designed mostly for mobile devices such as smartphones and tablets. Android comes with a lot of open source libraries that provide different services for developing apps.

Looking at the anatomy of android apps, android applications run within the Dalvik virtual machine. Dalvik runs Java written code and applications, after it is been compiled by the compiler, just to provide a short explanation what Dalvik VM is.

Dalvik applies JTI (Just in time compilation). It means that the compilation is done during the run time of the app not before as with Ahead-of-time compilation. To bring .Net to Android and iOS which based on Linux, and develop cross platform applications, an open source platform called Mono was introduced which used by Xamarin to support the cross-platform application development. Mono applications run within the Mono CLR (Common Language Runtime). So, this is how the anatomy of Xamarin Android looks like: there is Android Kernel which is the lowest level of Android; there is Mono RT upon it and .Net APIs. The problem arises since Mono RT runs against Android Kernel which makes it impossible to get the entire application experience at that level, not to mention that Dalvik provides most of the features that are not accessible at the Kernel level, and makes it an Android application, through the Dalvik Java APIs that come from either Java.* or the Android.* namespaces. It means that those facilities are not available directly to native applications. (Xamarin Inc., 2015)

With all these said how do Xamarin Android apps run and have access to all the features needed? A simple answer is that it uses both the Mono CLR and the Dalvik VM. The magic is accomplished through Android bindings which are standard Android classes such as activities or UI classes just to name a few. Also there is such a thing called MCW (Managed Callable Wrappers), so when .Net code is written and Android bindings are referenced, it will allow us to access the classes that residing in Dalvik RT. ACW (Android Callable Wrappers) allows to reference Java.* classes and pass some of that code from Dalvik to Mono.

The following Figure 1 illustrates how the runtimes co-exist

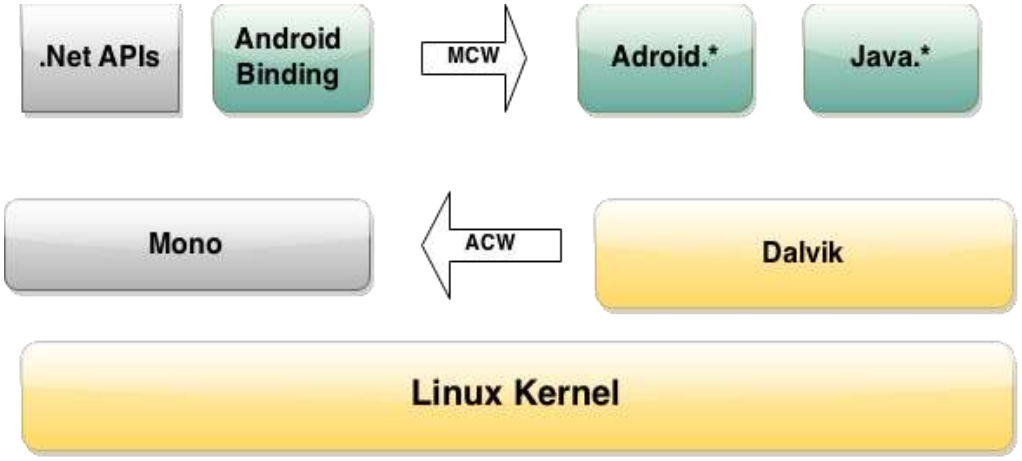


Figure 1. Xamarin Android Architecture (Xamarin inc., 2015)

To sum up, it seems that quite a lot of bindings going on behind the scenes while providing developers with native Android experience when building the applications on Xamarin.Android.

2.4 Windows Phone 8.1

For developing Windows phone 8.1 RT (Run Time) applications there is no need for Xamarin – it comes automatically with VS 2013.

First of all, what it means when we hear a phrase “Windows RT application”. This is how it is defined on the official MSDN web site.

> “A Windows Runtime app is an app that uses the Windows Runtime and runs on a Windows device (such as a PC or tablet) or a Windows Phone. We call a Windows Runtime app that runs on PCs, laptops, and tablets a Windows Store app. We call a Windows Runtime app that runs on the Windows Phone a Windows Phone Store app.” (msdn.microsoft.com)

It provides more features, new UI controls, enhanced navigation support comparing to the previous versions of Windows phone. Moreover, the emulators work fast and give a nice feel of a real device.

2.5 Why Xamarin

Aiming at the development for biggest platforms, there is a choice to make which techniques and tools to use. There are three options to choose from such as:

- Native framework – Eclipse, android studio.
- Hybrid framework – Cordova, PhoneGap, Telerik AppBuilder

- Cross-platform framework - Xamarin

For this project Xamarin has been chosen and here are the reasons why.

- It allows using C# when developing for different platforms in this case Android and Windows, therefore enabling code reuse across the platforms.
- It allows to use all powerful features such as asynchronous programming, HttpClinet for working with Web APIs, LINQ, IEnumerable and still maintain code readability and easy testing.
- Mono .Net implementation allows using common types such as classes, interfaces. Besides, memory management is handled.
- Enabled code sharing which accelerates the development.
- At the end deployed applications are native applications.
- Xamarin provides native UI editors.

All in all, it is a nice platform that provides a lot of possibilities and has all means for providing a good development experience.

3 System requirements specification

Requirements definition is all about representing data that describes how new or existing modified system should work, what users are expecting from it. This one of the most crucial steps in software development since it is important to determine what exactly user wants and expects in order deliver a successful software.

In this case, requirements analyses relates to a new system. Of course, it is a simplified version of the documentations since it includes a lot of steps and consists of many parts.

3.1 Functional requirements

The functional requirements describe how the system operates and what actions performed. The diagram, shown in Figure 2, illustrates the functional requirements for a new system, namely for Sopima mobile application.

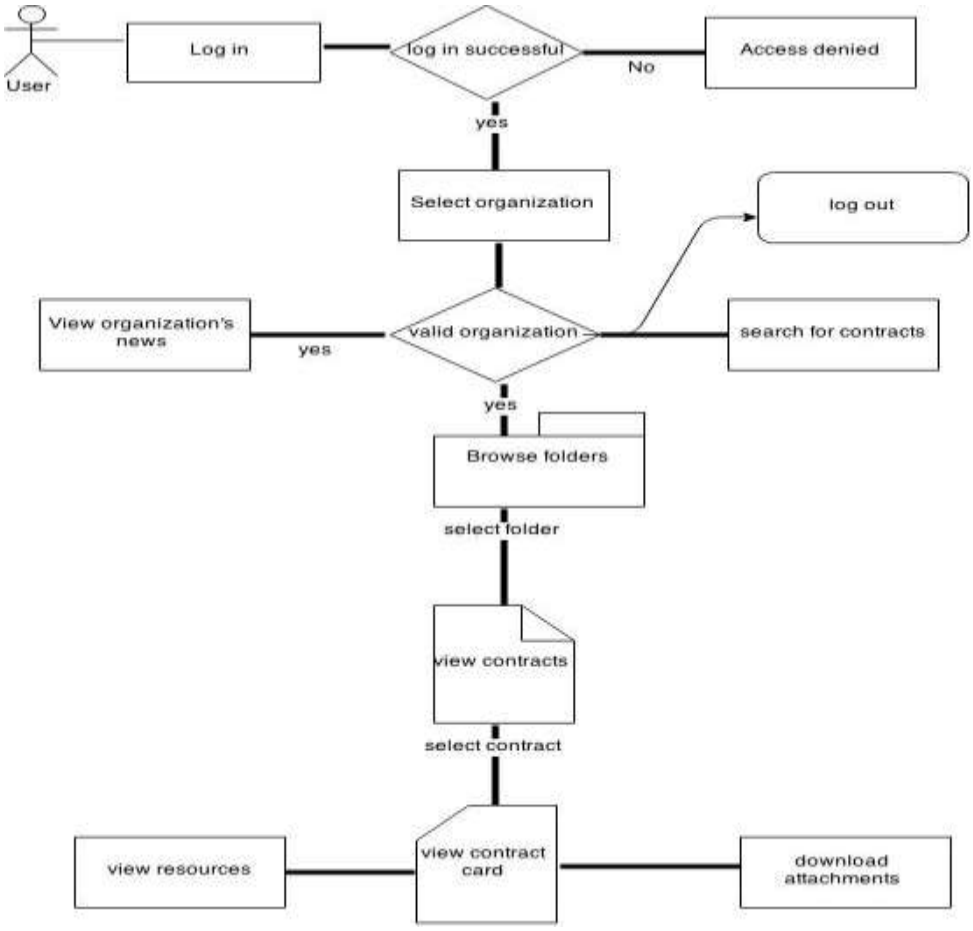


Figure 2. Simplified presentation of system functionality

These are only requirements for the scope of the thesis project. It is a set of basic functionalities.

A user should be able to log in into the system. After that he or she provided with a list of organizations that they belong to. It is possible to select an organization and view organizational latest news, browse folders with contracts, search for contracts, view resources and download any attachments that are in the contracts, and finally log out.

3.2 Non-Functional requirements

Non-functional requirements include such aspects as quality attributes, design and implementation constraint, mostly related to the usability.

The application has to work on both Android and Windows platforms providing native user experience, later on also on iOS.

- Only authorized users can have an access to the data provided by the app.
- The application has to be failure proof.
- The application has to be easy to maintain.
- The application has to have friendly UI and easy to use.
- The application has to have good response time.
- The application has to be easy to modify.

3.3 Reviewing the requirements

After software requirement have been discussed it is time to have a thorough look at the development process itself. Starting from the installation of the working environment and finishing by running the consumer ready applications.

According to the product backlog (Appendix 1), there are quite a lot of features to implement. The most crucial part is to implement the authorization process since without this functionality it won't be possible to get any data for the application. It also states that the development of the iOS app is postponed and will not be covered in the project.

4 Development process

This chapter describes the development process in details. It goes through the development of the application features for both Windows and Android platforms step by step, and points out the main aspects of it.

4.1 Setting the development environment

Xamarin for Visual studio 2013 for Windows 8.1 will be used for the project development. To download Xamarin go to the Xamarin web site and download suited suit. When downloaded, it is time to install Xamarin; there is nothing difficult just a normal program installation. During installation, usually installer does everything for you, it installs the components into their default locations and the VS should have the paths to those locations. In some cases if there is an error after the installation that reads something like “Can’t find Android SDK”, the developer will have to define the paths manually. In order to do that, navigate to Tools → Options → Xamarin → Android or iOS in Visual Studio and set the paths explicitly from there. After that you should have ready environment for Android development.

It is a known drawback that mobile emulators are very slow, so it is good idea to download and install Xamarin Android Player. To see how to install the Android Player read the article “Xamarin Android Player” from http://developer.xamarin.com/guides/android/getting_started/installation/android-player/. It will be needed to install Oracle VM VirtualBox for the player to work. Unfortunately, it works well only when Hyper-V VM, that is needed for Windows phone emulators to work, is off. To turn off/on Hyper-V go to ControlPanel → Programms and Features → Turn Windows features on or off and check or uncheck Hyper-V, the computer restart is required after that. On the contrary, Windows phone emulators work well regardless Oracle VM is on or off.

As a reminder nothing needed for Windows phone development, except updating if necessary Windows phone SDK to 8.1. Also it is now obvious that for Windows phone development your computer has to support Hyper-V VM.

In this case, there were some issues with starting Windows phone emulator because system Firewall would not let Windows phone emulator to start, so some rules have to be added to firewall register in order for it to work.

After all set, it is time to add projects into the solution. To do that right click on the solution explorer → Add → NewProject → Android → and choose the Blank App (Android) like shown in Figure 3.

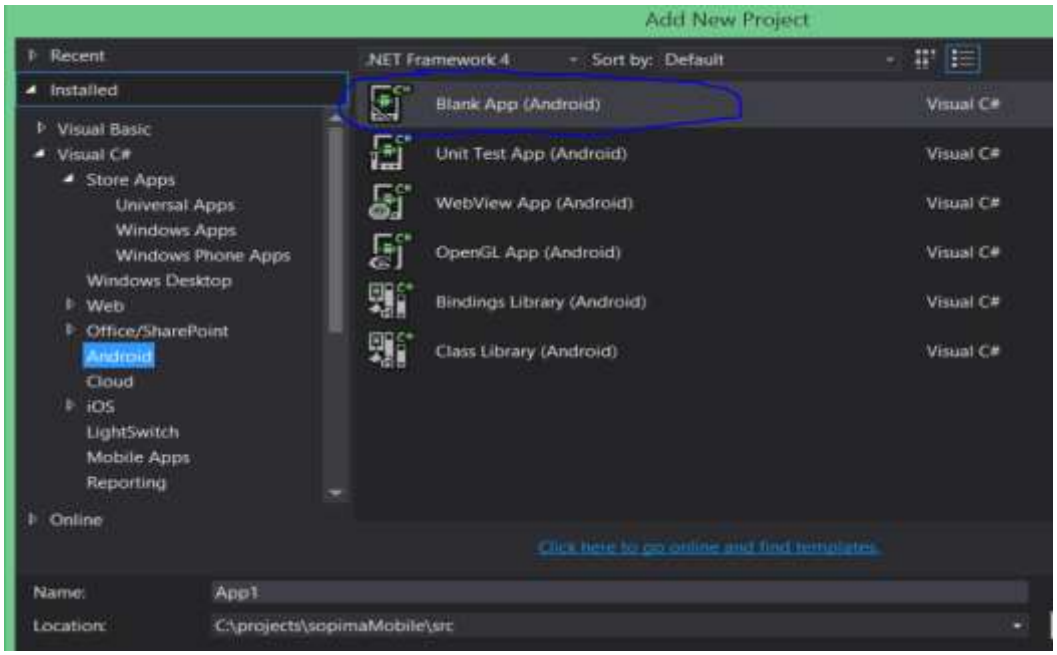


Figure 3. Creating Xamarin Android project

Just give it a name and specify the path. When all set press the OK button and it will create an empty Xamarin Android projects with a standard structure.

The same applies to the Windows phone project, except be careful not to add Silverlight Windows Phone project, in this project we need to select Store apps and from there Windows phone App like shown in Figure 4.

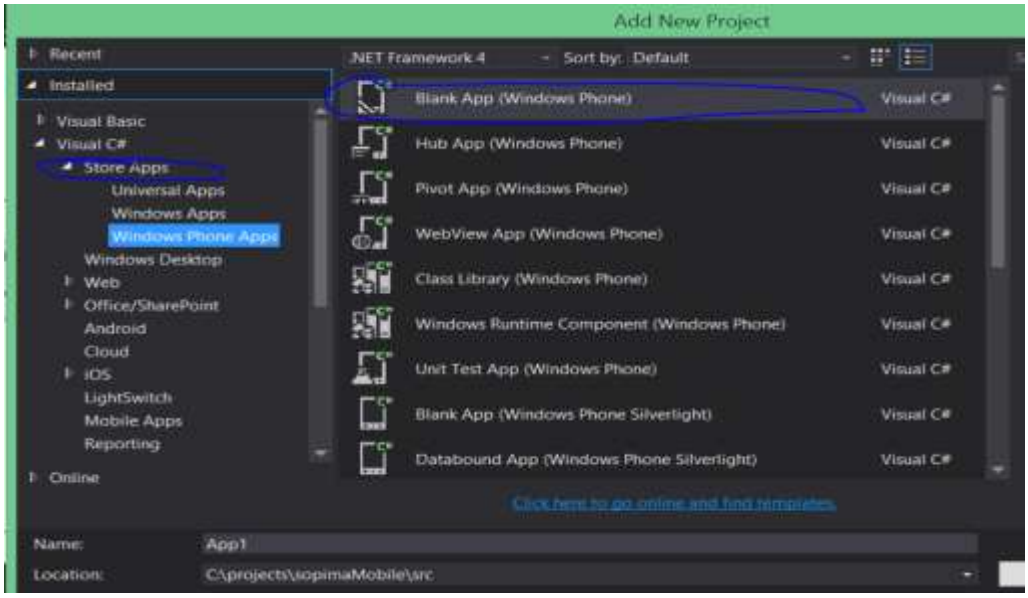


Figure 4. Creating Windows project

Usually, the naming convention when working with Xamarin is formed from the name of the project and the platform name like SopimaMobile.Android and SopimaMobile.Windows.

One more project is required to start the development process and this is PCL class to keep our shared code, and the naming convention, is to add .Core at the end Sopima.Core like shown in Figure 5.

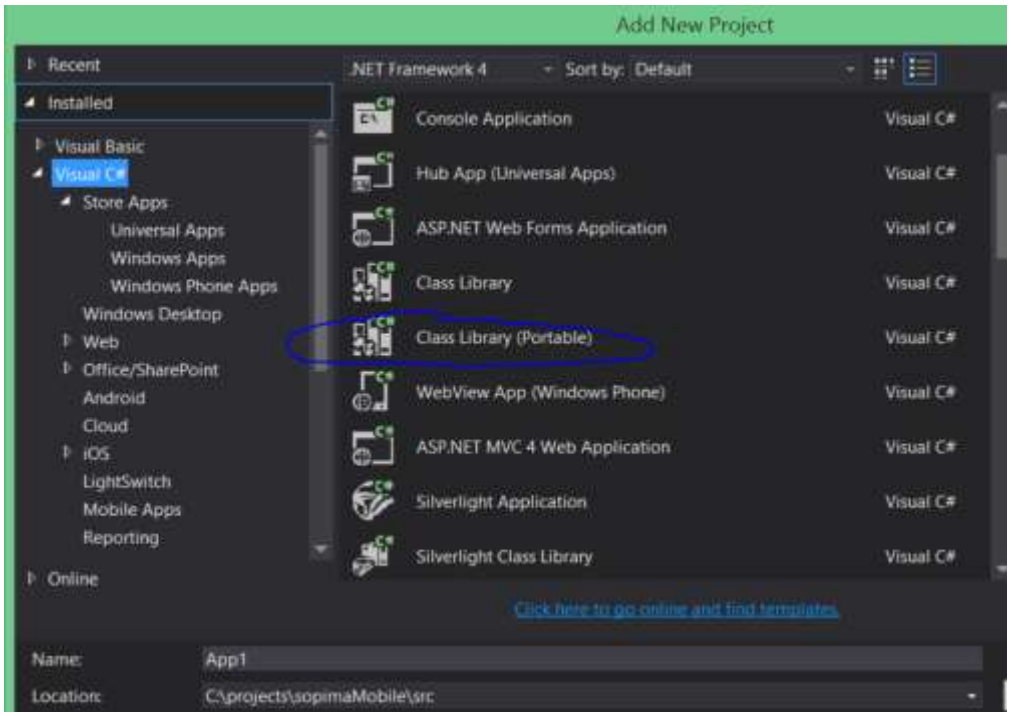


Figure 5. Creating Portable class library project

After the PCL project has been added just add references from the projects so the projects know about the PCL. Right click on the project → Add → Reference, see the Figure 6.

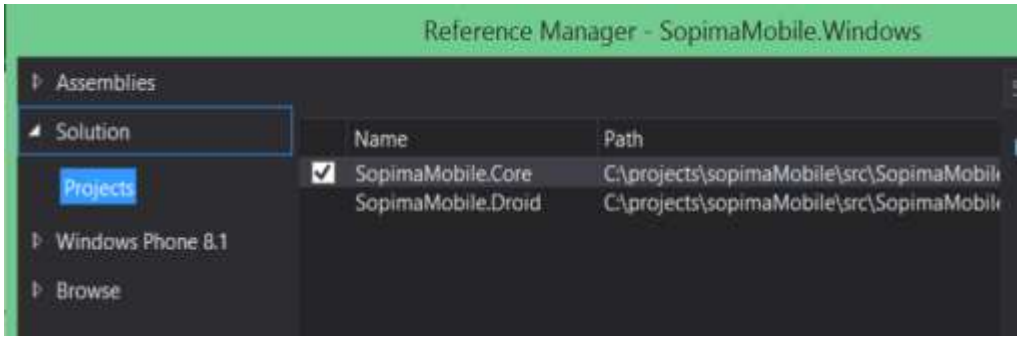


Figure 6. Add reference to PCL project

All set, it is time to start developing the applications.

4.2 Architecture of the software system

A typical cross platform app consists of separate projects for Android, Windows, and iOS and with the corresponding naming convention: Xamarin. Droid, Xamarin. Windows, Xamarin. iOS. Of course, the structure and the names of the project may vary according to the needs. In addition, in order to be able to share the code, it is necessary to add a PCL project which will host the code that will be shared between the projects. After that PCL has to be referenced from each platform's project.

In this case, all models used for the application are placed inside PCL Business Layer "Models folder" so all platform can easily access them. The PCL project structure is generated automatically when adding it to the solution. Figure 7 shows the class diagram for better demonstration of the application Business Layer.

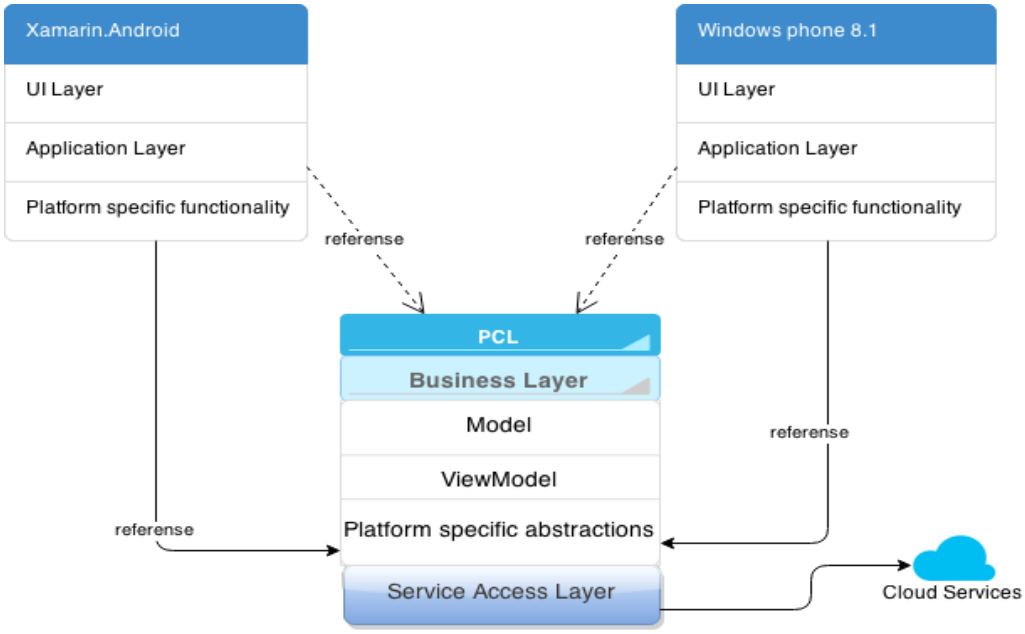
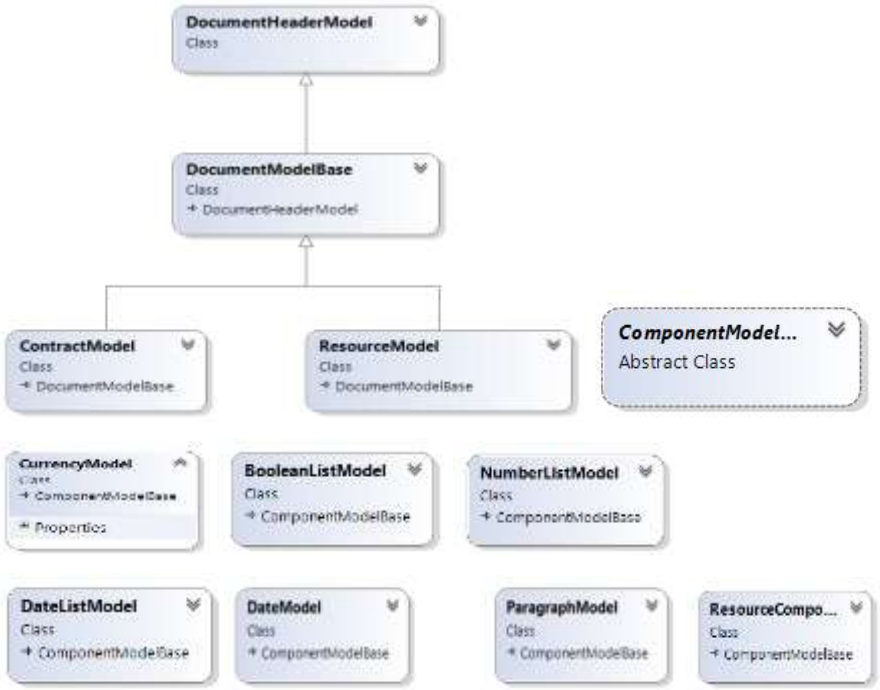


Figure 7. Software architecture

4.3 Sopima mobile core project

The Figure 8 below shows the structure of the share core project, to help get the idea of the project.



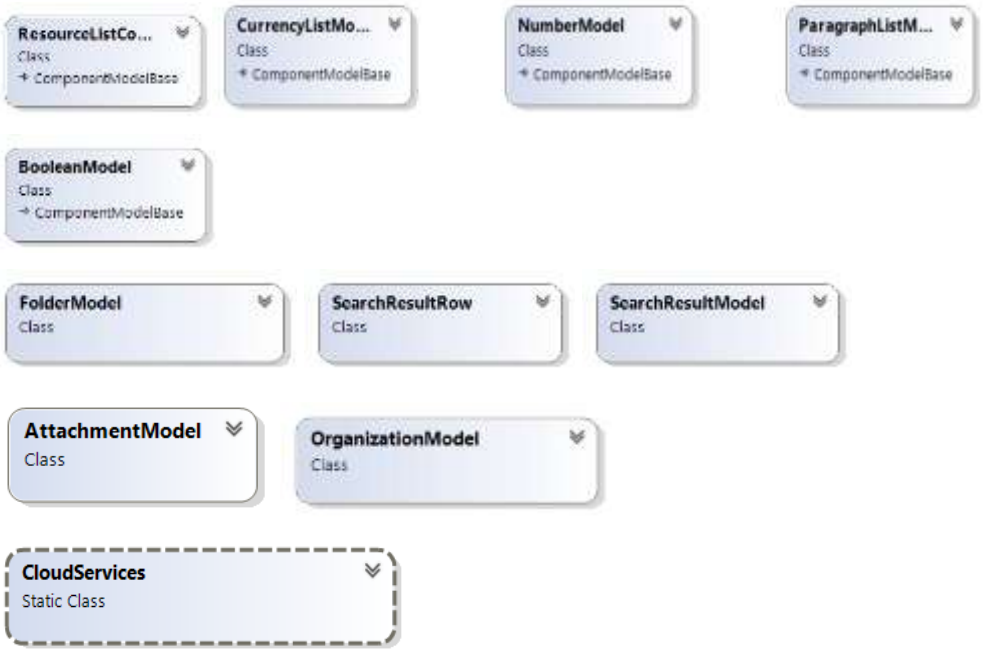


Figure 8. SopimaMobile.Core class diagram

Static class called **CloudServices** (Figure 9) resides all the methods as well as the authentication parameters that will be used for getting data from the Web API.

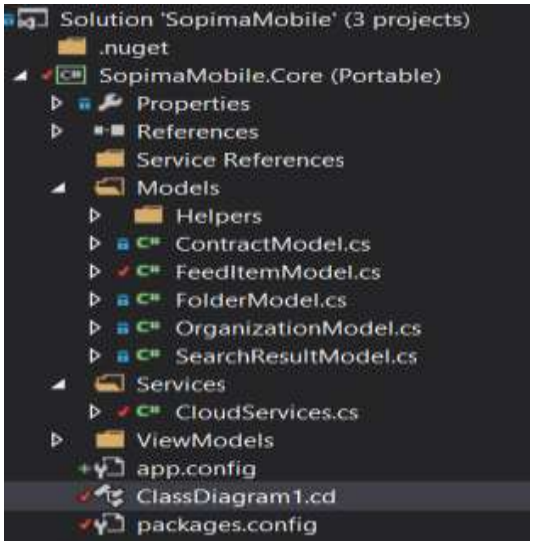


Figure 9. SopimaMobile.Core structure

4.4 Developing main functionality

The next step is to develop the main functionality. The scope of the thesis project includes only the basic read-only functionality which does not make it any easier. The detailed description of the work done during this process, described in the following chapters.

4.4.1 Authorizaion implementation

One of the key functionalities of the application is the authorization process that allows user to log in and log out into the application using their valid accounts. It guarantees that all the data is secured and can be accessed only by the authorized users. The authorization is implemented in the shared PCL project and available to platform-specific projects.

Authorization process flow

Since the AAD has been integrated into the Sopima web application that provides single sign-on functionality for its users, it is a requirement for the mobile application to use the same service for user authentication. The service is hosted in the cloud and allows easy user authentication using assigned corporate credentials.

Only registered users can have access to the application. Users are using their existing accounts and credentials that they have been granted for using the web application. When a client interacts with the application and wants to make data request to the Web API, they send authorization request to the AAD if authorized the client gets a JWT authorization token back and makes a call to the Web API with that token, after that they get the data back from Web API. The Figure 10 nicely displays the authorization process flow.

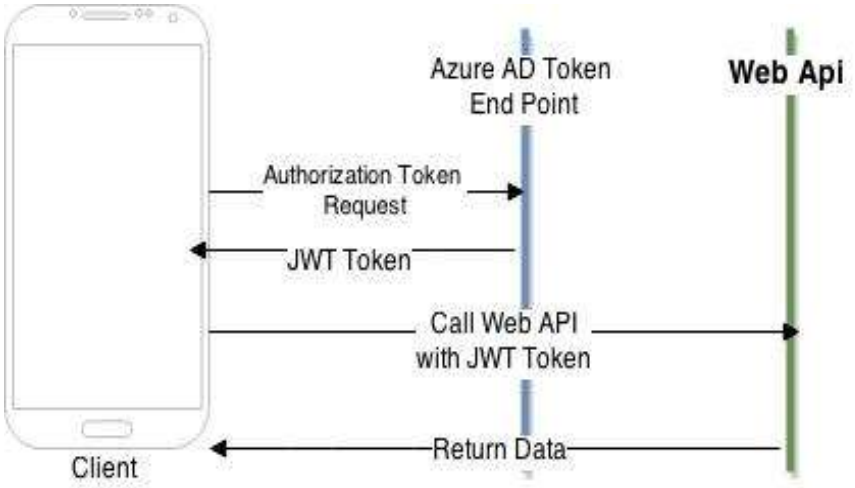


Figure 10. High level authorization & data request flow.

To add authentication to the project, we use Active Directory Authentication Library (ADAL). In order be able to authenticate users, it is necessary to register the application with the Azure Active Directory (AAD). This is done in two simple steps. First of all, you must register your Web API service and expose permissions on it. Secondly, we must register our mobile client application and grant it access to those permissions. After done

with the configurations, it is time for installing ADAL library into the solution projects. The ADAL v3 supports PCL which is exactly what is needed for the project as well as different mobile platforms. The logic for the authorizations is handled in the PCL and shared between platforms. In addition to the shared code, it is important to remember about platform specific parts.

The figure 11 displays all the necessary parameters for the authentication that are available from Azure Active Directory portal.

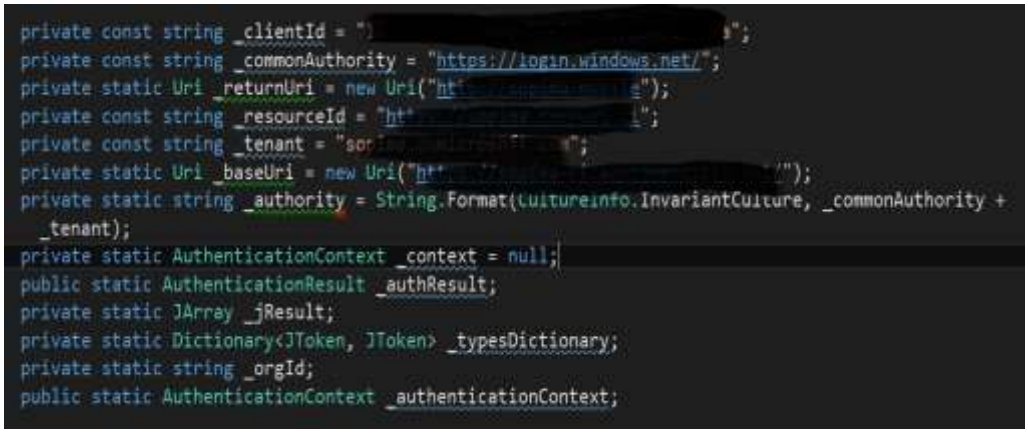


Figure 11. Set of properties required for ADAL authentication

When implementing authorization for Android, it is necessary to override the **OnActivityResult ,** like shown in Figure 12, in order to resume the application since it prompts a user to the standard windows web authorization page in a web browser. Also notice that in the case of Android, 'this' parameter is passed to **AuthorizationParameters(this)** to share the context, but in the case of Windows - without any parameter as new **AuthorizationParameters**().

```
protected override void OnActivityResult(int requestCode, Result re-
sultCode, Intent data)
    {
     base.OnActivityResult(requestCode, resultCode, data);  Authentica-
    tionAgentContinuationHelp-
    er.SetAuthenticationAgentContinuationEventArgs(requestCode, re-
    sultCode, data);
    }
```

Figure 12. **OnActivityResult** method implementation for Android

For Windows Phone, the **OnActivated** method in the **App.xaml.cs** file is modified like shown in Figure 13.

```
protected override void OnActivated(IActivatedEventArgs args)
{
#if WINDOWS_PHONE_APP
  if (args is IWebAuthenticationBrokerContinuationEventArgs)
{     WebAuthenticationBrokerContinuationHelp-
er.SetWebAuthenticationBrokerContinuationEventArgs(args as IWebAuthenticationBrokerCon-
tinuationEventArgs);
  }
#endif
  base.OnActivated(args);
}
```

Figure13. **OnActivated** method implementation for Windows

For more detailed steps how to configure application for successful authorization read the article “Authenticate Xamarin Mobile Apps Using Azure Active Directory” by Mayur Tendulkar to do this go to http://blog.xamarin.com/authenticate-xamarin-mobile-apps-using-azure-active-directory/.

Upon successful authorization a user gets and access token and refresh token as authorization result. It is also possible to cache the token, so users do not have to log in all the time. If the token expires, then request for refresh token. The token is sent with each http request to Web API service in authorization header, or sometimes in URL depending on the Web API design.

This is the basic authentication process that has been implemented for the application. The library can be added to the projects via Package Management console in Visual studio 2013.

```
PM> Install-Package Microsoft.IdentityModel.Clients.ActiveDirectory -Version
2.14.201151115
PM> Install-Package Microsoft.IdentityModel.Clients.ActiveDirectory –Pre
PM> Install-Package AdalXamarinAndroid –Pre
```

Note you will need to reference ADAL from each and every platform and also the versions can differ since it’s constantly being updated.

4.5 Business layer

After the authentication is set and working, it is time to get data for the application which will be stored in so called business layer in the PCL project that usually represent the

models that provide data for the application. In case of MVVM pattern use there will also be ViewModel in the PCL's Business layer.

The Sopima web application is built in using the latest modern technologies including Restful Web Api technology that enables easy data sharing between platforms without exposing the data that is not meant to be accessed and without direct access to the database.

We have a single class in the PCL project Services folder that handles the querying data from Web Api and creating models. To be able to get the data it is necessary to know the Uri to that data. Usually, the web api Uri starts as **api/1/xxx/{id}** where **xxx** is usually the name of the controller and the id in curly brackets represents optional parameter that is very often a unique Id of a single element in a database. Along with the Uri we need to have an access token because only authorized users can be granted the access to the application. It is a good idea to define a default http header that is sent with each and every request to Web Api service. As for how a default Httprequest header is set in Httpclient see the Appendix 7.

Luckily, Xamarin supports Httpclient implementation which is Mono's open-source implementation of System.Net.HttpClient, and it is available in Xamarin.Android and Xamarin.iOS, simply add a reference to System.Net.Http so you can use that. And for Windows platform there are no special requirements.

As seen from the image above HttpClient is used to query data from Web API. Where base address includes the address of the Web API itself hosted in Azure plus common part and a request specific URI. We also tell that we want to receive data in Json format. Json is JavaScript object notation and used for data exchange over the network. The Figure 14 shows a simple example of a simple object represented by Json.

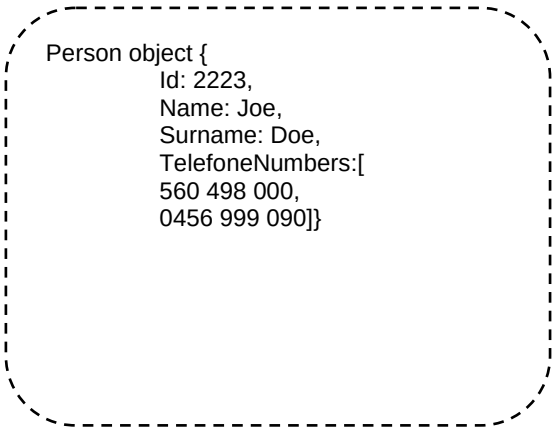


Figure 14. Json object example

It is quite easy to read and write. It is understandable for humans and what is the most important it is easy to parse into the objects. Here is the example of how Json string is deserialized into a model. What is nice about the method, shown in Figure 15, is that it is a generic method so you do not have to specify the return type right away.

```
public static T DeserializeJSon<T>(string jsonString)
    {
        var jsonObject = JValue.Parse(jsonString);



        jsonString = jsonObject.ToString(Formatting.None, null);

        object obj = JsonConvert.DeserializeObject<T>(jsonString);

        return (T) obj;
    }
```

Figure 15. Deserialization method

We simply create a method that accepts the Json string that we got back from web API request and **T** is the object type we would like to receive back. Of course, the models structure should be defined first. Simple as it is.

After that, a specific to each request, method that handles each and every call to the Web API is created, and it utilizes the common method where the access token and selected organization Id are added to the default header. The common method is shown in the Appendix 7.

As the picture above shows **async await** pattern that was introduced with C# 5 and comes in conjunction with Task parallel library. It is supported by Xamarin and is used to send the requests, which is usually the case when getting data via network since it can take time to process and retrieve it. What it means is that the requests are sent asynchronously. It is very important to be able to use this technology because this way we can ensure that we are not blocking the main thread in our case it UI thread so our application UI won't "froze" while we retrieving the data. Note that **async** modifier only indicates that there is code that can be run asynchronously but it doesn't ensure it is important to add the **await** keyword in front of the asynchronously run code otherwise the code is run synchronously.

Task<List<FeedItemModel>> simply indicates the return types is of **FeedItemModel.** This common pattern is also applied to the other methods that are used for getting data from the Web API not to mention that the asynchronous pattern is used across the application and quite a lot.

After models have been created and we are able to query data from the web API, it is time to have a look at how each platform handles this data.

4.5.1 Sopima mobile Windows

The Windows phone application is targeting 8.1 OS and has more capabilities than the previous versions. The SopimaMobile.Windows application has a bit simpler structure than SopimaMobile.Android the complete project structure is shown in Figure 16.

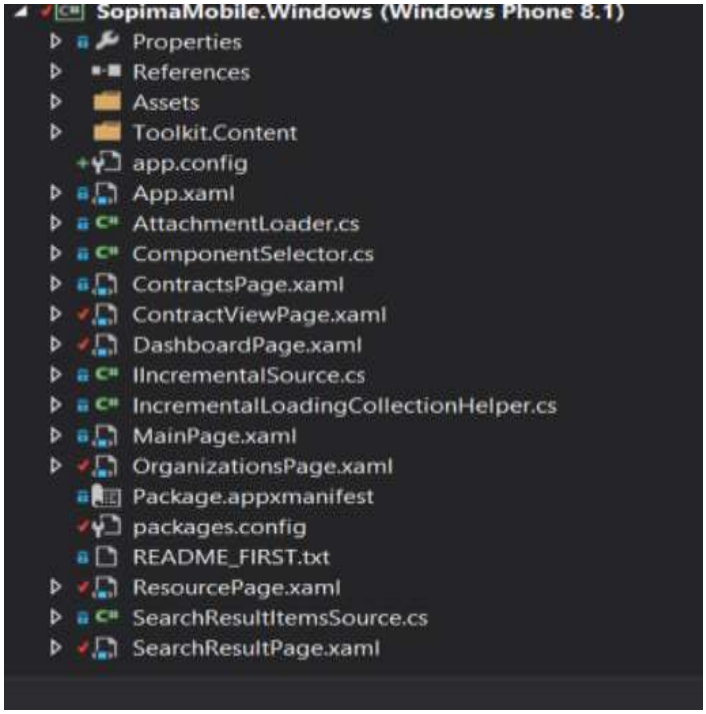


Figure 16. Sopima.Windows project structure

The class diagram shown in Figure 17 reflects the logic behind the system of the SopimaMobile.Windows project. The process flow and features implementation will be discussed in more details further.

Figure 17. SopimaMobile.Windows class diagram

There are eight classes that inherit from Windows Page class and each is represents for a single view and functionality. There are also helper classes that help to display a complex structure of **Contract** and **Resource** models. Furthermore, for downloading files that are represented as attachments to contracts **AttachmentLoader** helper class is used (Appendix 8).

What concerns the UI **ListView** and **ListBox** UI controls will be used to display data for the user they support data binding and provide a lot of useful methods and properties. Each page has **OnNavigatedTo** method where most of the action happens like when navigating to a certain page.

If there are all configurations, and everything is set correctly for the successful authentication, it should be possible to log in into the system. The log in functionality is implemented in the MainPage class by asynchronously calling the **LogIn** method from the shared **CloudServices** class (Appendix 2) with **AuthorizationParameter** as a parameter. The authentication process was discussed in more details in the previous chapters. The method redirects the user to the Microsoft authorization page where the user is required to enter their corporate credentials.

For building **MainPage** layout static IU controls were used. The Figure 18 shows the UI layout of log in page.

Figure 18. SopimaMobile.Windows Log In

Upon successful authorization the user is returned to the application and **List<OrganizationModel>()** is populated and ready to be displayed (Appendix 4). **Frame.Navigate()**method is responsible for navigating between pages of an application, it can also pass parameter to another page, like in case of Organizations. Frame.Navigate(typeof (Organizations), _orgs) passes the list of organizations the user belongs to and have access to (Appendix 4).

The Organizations page class uses the Organizations model to display data using XAML's data-binding features (Appendix 5). The **ListView ItemsSource** is set to the **List<OrganizationModel>()**, that was populated asynchronously in the **MainPage** class and passed as a parameter using the **Frame.Navigate** method. The **{Binding}** or **{Binding Path=}** syntax in the XAML determines how the data is displayed. The result is shown below in Figure 19.

Figure 19. SopimaMoble.Windows Organizatiions

The next page comes into the action when the user selects an organization they want to work with. **ListView** supports an event listener that reacts when an item in the list is clicked. The way it handled programmatically is shown in Figure 20.

```
private void OrganizationsList_SelectionChanged(object sender, Selection-
ChangedEventArgs e)
    {
        SelectedOrganization = _orgs[OrganizationsList.SelectedIndex];
        Frame.Navigate(typeof(DashboardPage));
    }
```

Figure 20. List item SelectionChanged

By using **SelectedIndex** parameter that is provided by the **ListView** control, it is possible to define which item was selected in the list after that the selected **OrganizationModel** is passed as a parameter to the next page called **Dashboard** that is shown in Figure 21 below.

Figure 21. SopimaMobile.Windows dashboard

The Dashboard page has a pivot UI control that provides two options on swiping. Organization's news is displayed on the first page and folders that contain contracts on the next. Besides, on the top a user can see the name of the selected organization. Application navigation bar is introduced on this page. It provides two options: search for contracts and log out. Again the layout is built using a **ListView** UI control.

DashboardPage class gets the passed **OrganizationModel** and makes and synchronous call to the Web API vie **CloudServices** class. When the response is ready and **List<FolderModel>()** and **List<FeedItemModel>** are populated, the **ListView's ItemsSource** is bind to the models and the user sees the result.

The search functionality enables a fast contract search, and it looks like in Figure 22.

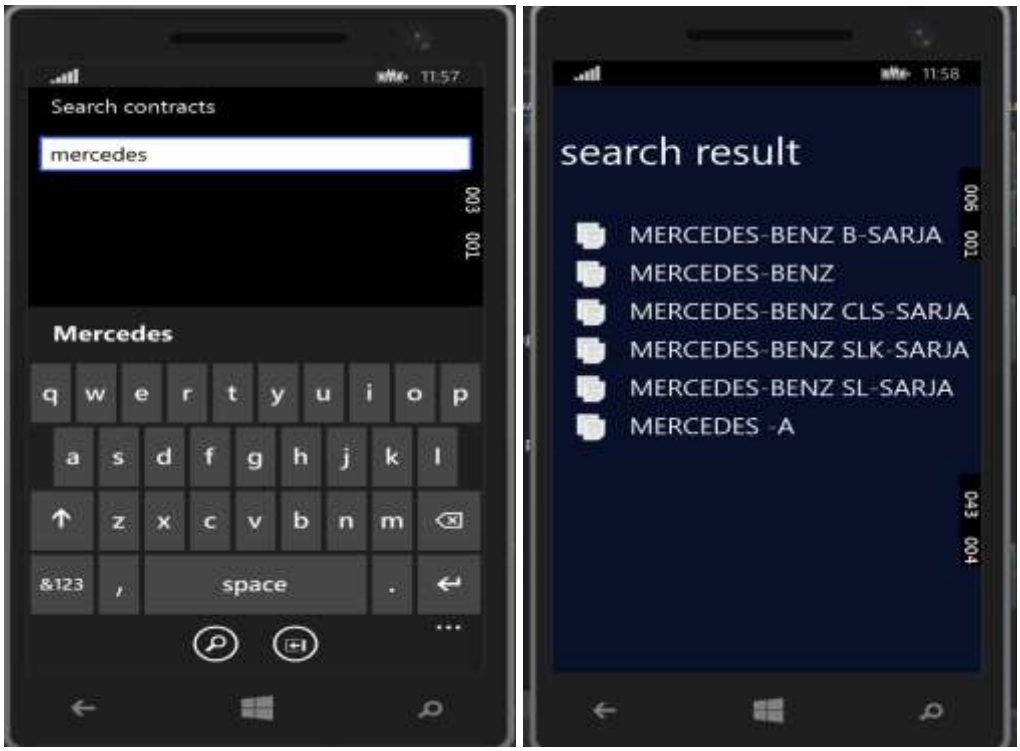


Figure 22. SopimaMobile.Windows search functionality

When the search button is pressed, a search input **TextBox** appears and the user can type the name of the contract or at least some part of it after that on Enter pressed the search starts and upon finished task the search results are displayed to the user. The methods shown in Figure 23 triggers the search to start and sends the search query to the **SearchResultPage** class where it goes further and asynchronously calls Web API and the result model **List<SearchResultRow>()** is bind to the **ListView ItemsSource**.

```
public async void Search_KeyDown(object sender, KeyRoutedEventArgs e)
    {
        if (e.Key == VirtualKey.Enter)
        {
            var query = SearchTextBox.Text;

            Frame.Navigate(typeof(SearchResultPage), query);
        }
    }
```

Figure 23. Search method on Enter pressed

Upon a click on the result list row the navigation takes the user to a single contract view with selected **ContractModel** as a parameter. This view is handled by the **ContractViewPage** class. The Figure 24 shows the UI of a single.

Figure 24. SopimaMobile.Windows single contract view

This is of the most difficult data bindings in the application, because a single contract model has a lot of components. A helper class called **DataTemplateSelector** that inherits from **ContentControl** class that represent a control with a single piece of content. This helper class helps to display the needed section based on the component's type, such as date component, currency component just to name a few, was created to handle different types of contract components. The whole layout is placed inside a **ScrollViewer** UI control since **ListView** is not applicable in this case and scrolling through the contract content is still needed.

On the **ContractViewPage** the user can also go back to the **DashboardPage** or choose to share the contract link with their colleagues by pressing the share button on the application bar which triggers the pop up window with suggested applications for sharing the contract link like shown in Figure 25 below.

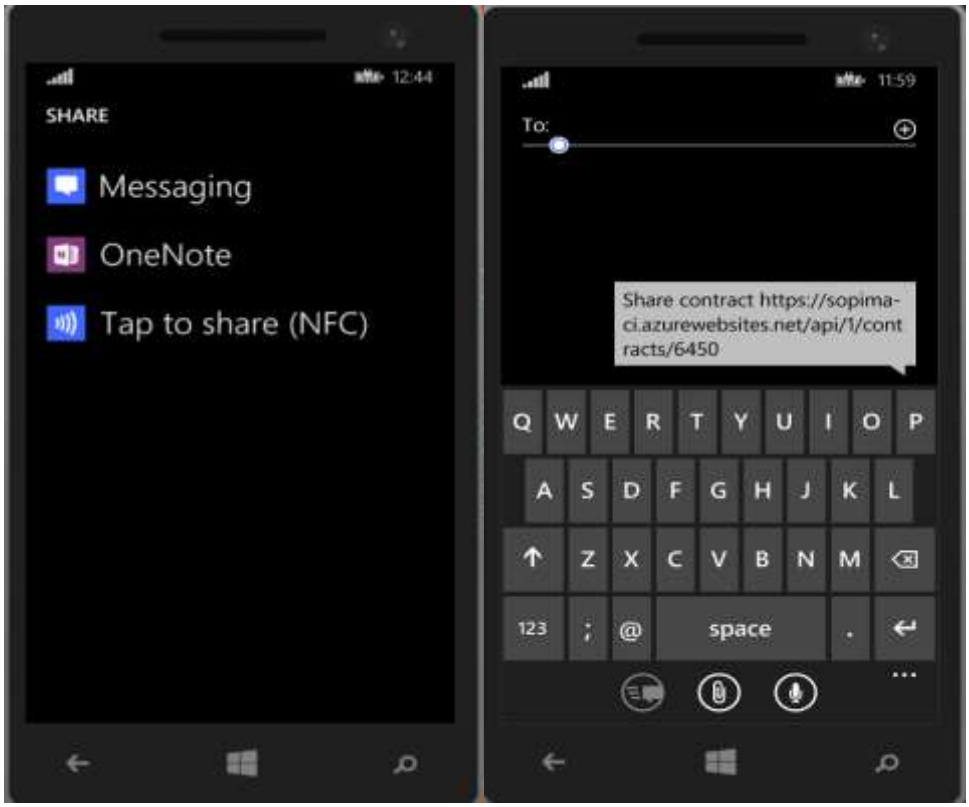


Figure 25. SopimaMobile.Windows sharing a contract

DataTransferManager class initiates the data sharing with other applications. The method displayed in Figure 26 shows how the **ContractViewPage_DataRequested** method handles the contract sharing.

```
void ContractViewPage_DataRequested(DataTransferManager sender, DataRequestedEventArgs args)
    {
        var request = args.Request;
        request.Data.Properties.Title = "Share contract";
        request.Data.Properties.Description = "Share a contract";
        var contractURI = CloudServices.GetContractUrl(selectedContractId);
        request.Data.SetText(contractURI);
    }
```
Figure 26. Contract sharing

One more important feature is the possibility to download the contract's attachments whether it is an image or pdf file. To be able to do that just an attachment URL is needed. The UI result is displayed in Figure 27.

Figure 27. SopimaMobile.Windows downloading contract's attachments

It is easily achieved with the method shown in the Appendix 8. Upon successful download **AttachmentLoadInBrowser** method with an attachment URI as a parameter awaits for the Launcher class to start an associated with link application to open the downloaded file.

Sometimes a contract can have a resource that is associated with the contract and added as a contract component. A resource card, shown in Figure 28, has a similar structure and functionality implementation as a contract card, so it will not be discussed in details here.

Figure 28. SopimaMobile.Windows resource card

Navigation to a **resource TextBlock_Tapped(object sender, TappedRoutedEventArgs e)** event handler is used where the navigation is handled with the usual **Frame.Navigation method** and the data binding to the model.

One more way to get to a certain contract is vie Browse page where by selecting a folder user sees the contracts of the folder and can scroll through the list and filter contracts as shown in Figure 29.

Figure 29. SopimaMobile.Windows browse contracts

ContractsPage class is responsible for the view and providing data for the view. Except familiar ListView control additional helper classes were used in order to implement infinite scrolling effect so when the user scroll to the end of the list, currently the list loads twenty items at time, a new page asynchronously loaded. The scrolling stops when there are no items to load. **IncrementalLoadingCollection<T, I>** that implements **ISupportIncrementalLoading** interface and inherits from **ObservableCollection<I>** is responsible for this functionality it has a method that gets the data per page to load and a **Boolean HasMoreItems** property that is set to true when there are items to load and to false if there no items.

To help a user to find faster contracts from the list of contracts, there is a filter option enabled. The user just types some text into the **TextBox** and the result displayed based on the text. **ContractFilter_FilterTextChanged(object sender, TextChangedEventArgs e)** event handler available for **TextBox** UI control helps to achieve the filtering effect where the filtered result is used to populate the items list and set to the **ListView ItemsSource** for data binding.

This chapter described shortly Windows application architecture and how the app has been developed.

4.5.2 Android

The sample structure for an android application is created automatically when you add it into your project. It looks like a typical android application if you are familiar with native android development environment.

The figure 30 shows the basic structure of the android project.

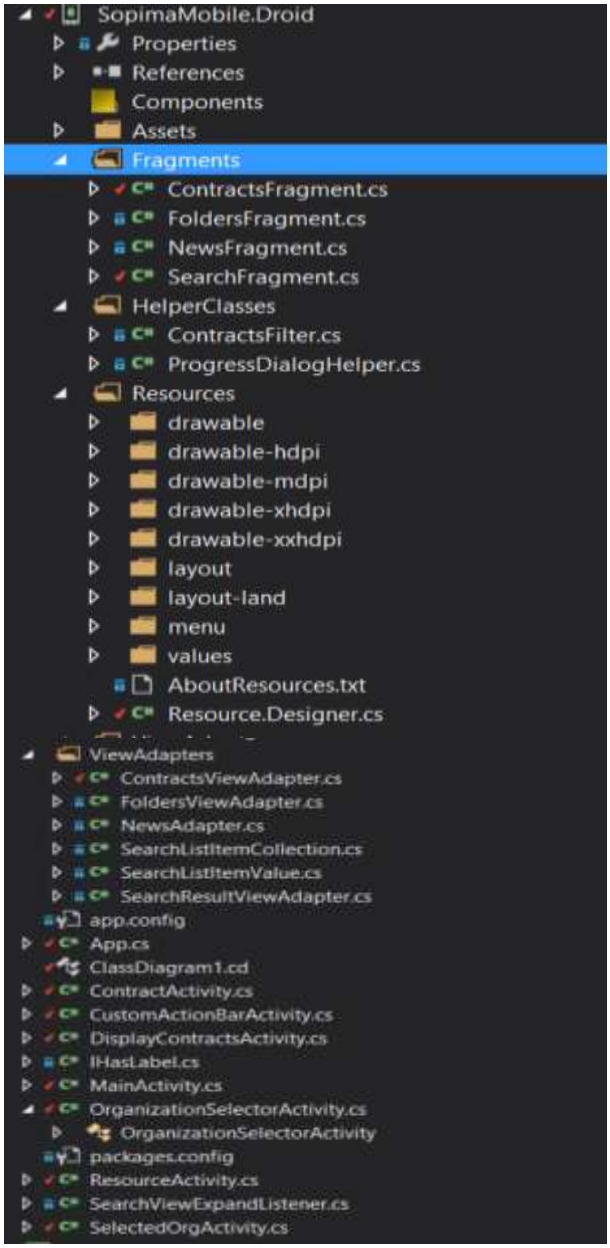


Figure 30. Basic structure SopimaMobile.Droid

The class diagram below in Figure 31 reflects the logic behind the system of the SopimaMobile.Droid project and the description of the diagram follows.

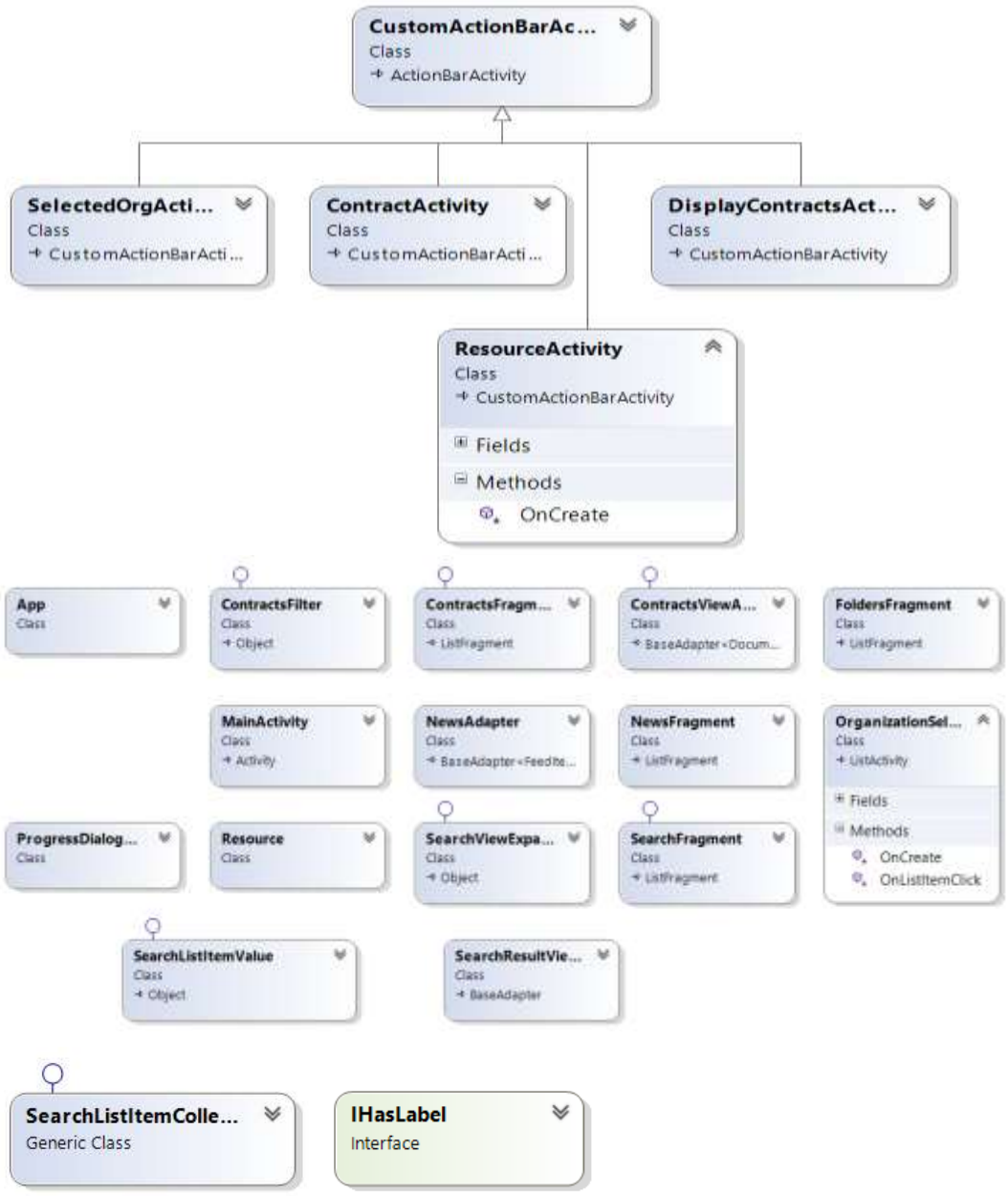


Figure 31. SopimaMobile.Android class diagram

The application logic is the same as in the Windows application, so it won't be discussed here again. In case of Android application the **OnCreate** method, shown in Figure 32, will be used the most.

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

Figure 32. OnCreate method

The Figure 31 shows that there are five main activities in the Android project some of them inherit from the **CustomActionBarActivity** which is responsible for providing custom action bar that is common for all the activities where applicable. Besides the activities, there are a lot of helper classes such as view adapters and fragments that help to pass data to the UI.

The **MainActivity** is responsible for providing the log in layout and handling the event click of the log in button. The Figure 33 shows how authorization request looks like in Sopima mobile Android. The layout is defined in AXML file that is stored in the layout folder along with all other AXML files responsible for the pages' layouts. In order to be associate a certain view with an activity **SetContentView(Resource.Layout.Main)** method with a path to the layout should be set in **OnCreateMethod** of every activity that has a view.

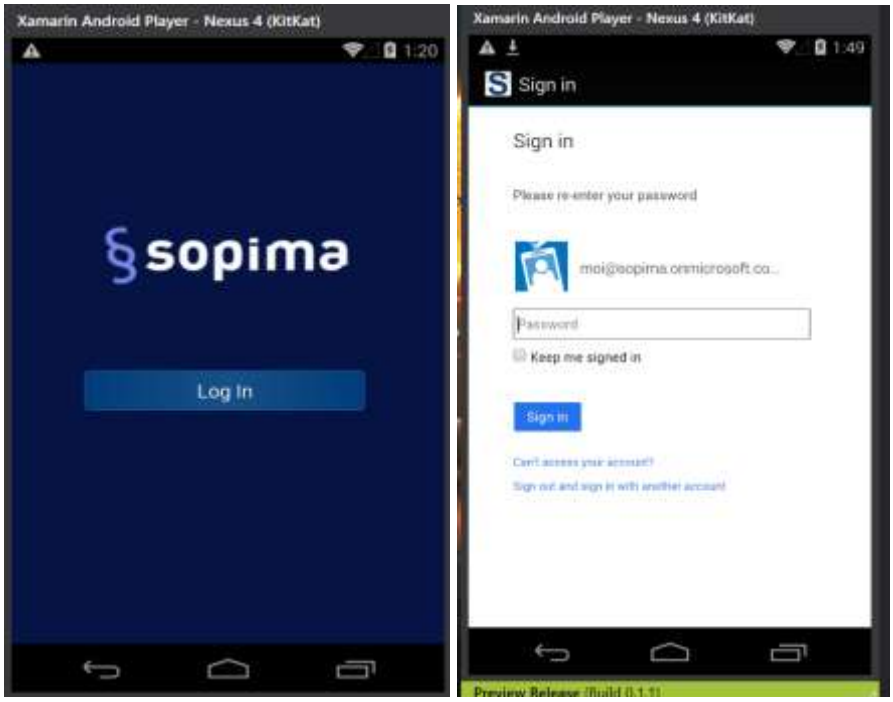


Figure 33. SopimaMobile.Android authorization reqest

When **List<OrganizationModel>** is populated (Appendix 2, Appendix 3) the navigation to the next page is triggered. The Figure 34 shows how the navigation between pages happens in Android.

```
Intent organizationsIntent = new Intent(this, typeof(OrganizationSelectorActivity));
StartActivity(organizationsIntent);
```

Figure 34. Navigation between pages

In **OrganizationSelectorActivity** the view is set and the data retrieved a **ListView** UI control also used to populate a list with the data. There are some standard layouts for the **LisView** control, in most of the cases, it is not enough and a developer will have to create custom **ListView** adapter that would provide custom **ListView** layout based on the developer needs. An example of **ListView** and how it is used can be found from the Appendix 6. The data structure of the list of organizations is quite simple here so it was possible to use a list view with predefined layout though the styling was readjusted.

Figure 35. SopimaMobile.Android display organizations

From the page that is shown in Figure 35, the navigation leads to the **SelectedOrganizationActivity** which is basically the dashboard for the Android application. The layout is shown below in Figure 37.

The page has a tabbed navigation bar with two tabs **News** and **Browse,** there is also a search button, the name of the selected organization and a dropdown menu with options to go the organization selection or log out.

The activity uses Fragments to implement the tabbed app bar. The main idea of the fragments is that it is possible to reuse them throughout the application and place them inside

a page in any place you want by adding **FrameLayout** with an Id to the AXML main view page. A simple example of a Frame layout component is shown below in Figure 36.

```
<FrameLayout
    android:id="@+id/fragmentContainer"
    android:layout_width="match_parent"
    android:layout_height="0dip"
    android:layout_weight="1" />
```

Figure 36. Frame layout Android

The **FoldersFragment** represent the **Browse** tab and the **NewsFragment** the **News** tab whenever a user changes the tab one fragment is replaced by another. In order to create Fragments, the custom fragment class must inherit from **Fragment** or **ListFragment** classes, it dependents on the view that is to be displayed, and the inherited members must be implemented. **NewsViewAdapter** class and **FoldersAdapterClass** are used for supplying data to the **ListView,** because the custom ListView implementation was required. The custom **ViewAdapters** must implement the **BaseAdapter** class which is a member of **Android.Widget**. This type of implementation of custom **ViewAdapters** allows defining any type of layout for a **ListView**, as well as a layout for each fragment created in a separate AXML file. The Folders list Item also provides information about how many contracts there are in a folder. The layout is represented in Figure 37.

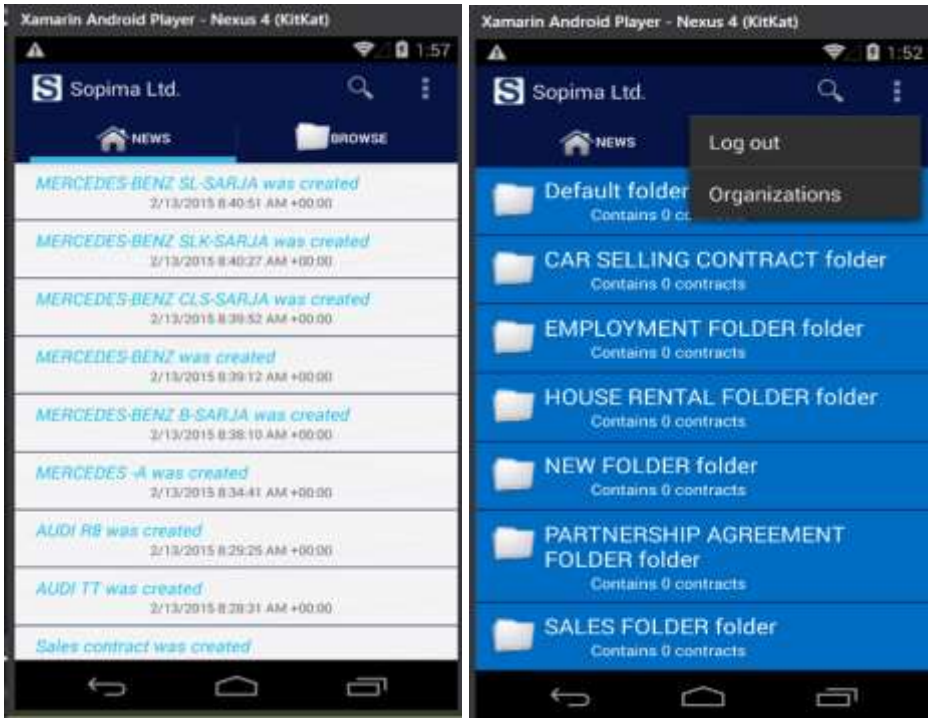


Figure 37. SopimaMoile.Android dashboard

The search functionality is also implemented in **SelectedOrganizationActivity**, and comes as a **ListFragment** as well, it also has a bit more difficult structure and requires some helper classes to be able to display sorted list of contracts. A small example how the input box reacts to the user input and triggers the search function can be found in Appendix 9. The sorting is based on the common template header a contract was created from. The search functionality layout can be seen below in Figure 38.

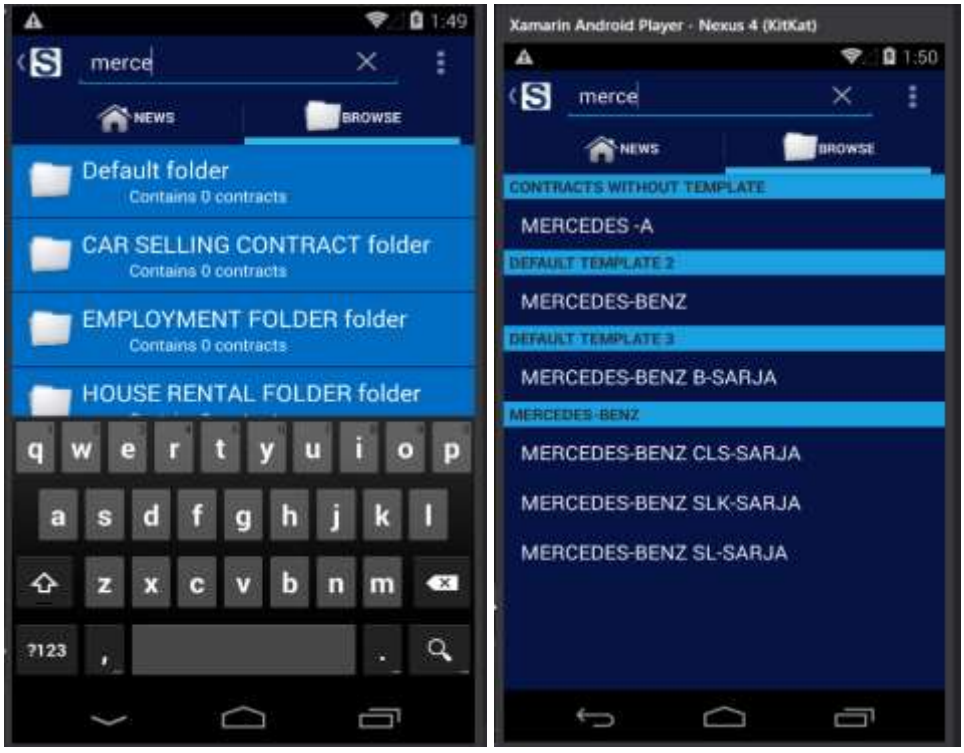


Figure 38. SopimaMobile.Android search for contracts

The same logic as in Windows application the user can access a single contract whether by clicking on a single contract on the search results or by going to a sibgle folder and selecting a contract from there. Also contract filtering and pagination are supported in Android application. The **DisplayContractsActivity** along with the **ContractsViewAdapter** handles the implementation of the functionality. The **CustomViewAdapter** follows the same logic as the **ViewAdpters** that were descussed earlier. Except that it also implements the **IFilterable** interface for filtering the contracts list view. The UI layout can be seen in Figure 39.

Figure 39. SopimaMobile.Android contracts of a folder

After selecting a contract to view, the user is navigated to a single contract view page that is implemented in **ContractActivity** and the layout for this page is mostly defined programmatically in the activity class because of the complex contract structure. It also handles contract link sharing and attachments download.

Android.Support.V7.Widget.ShareActionProvider widget is used for handling contract sharing. The resulted layout is shown below in Figure 40.

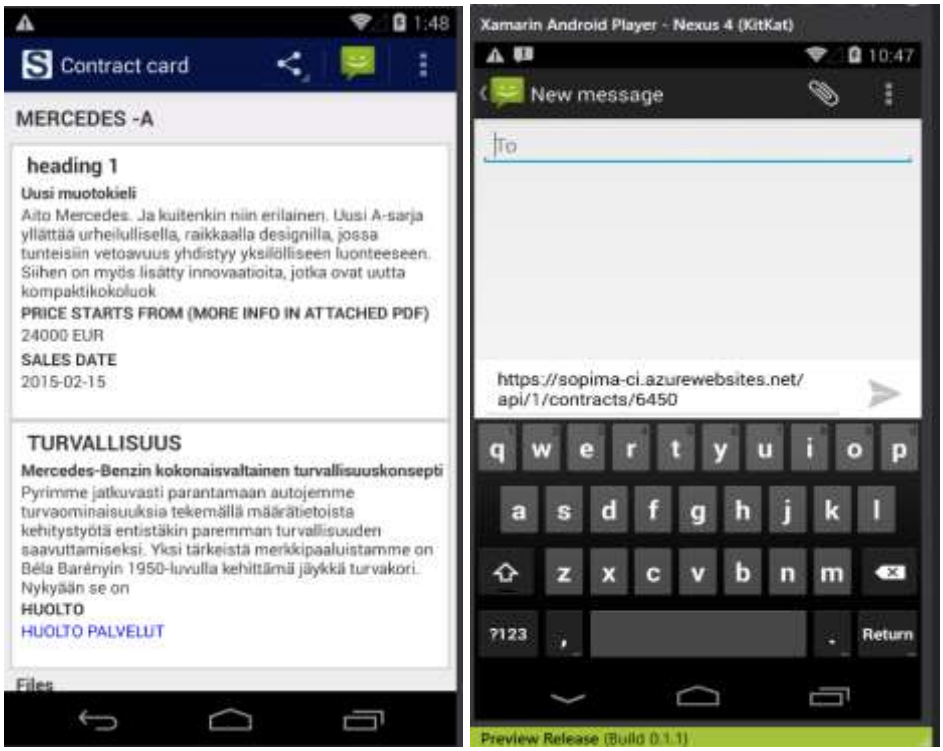


Figure 40. SopimaMobile.Android single contract view

Furthermore, by clicking on a resource link the user is taken to the **ResourceActivity**. The **ResourceActivity** also has similar structure as the ContractActivity and implemented in a similar way.

All in all, it is much more difficult to develop a mobile application that is based on a web application. A lot of technics were applied to make the mobile applications to work similar to the web application.

4.5.3 Layout implementation summary

UI layout implementation is the only project layer where UI is written for each platform separately, again to support the native UI feel. The UI design is implemented by using the native set of tools available in UI designer to give them a native feel to it. Well it is quite time consuming and requires the knowledge of designing with AXML, but since it is possible to share the code behind, probably, it is a good idea to take time and design the native UI. Thought there are some technologies, namely, Xamarin Forms that support the building of cross-platform UI. The Xamarin Forms technology was not used in this project because it does not yet support Windows Store 8.1 platform.

The basic layout was created during the development of the main features and was improved at the final stage.

The Android and Windows application have similar UI layout but of course with the platform specific features and details. The **AXML** language is used for writing the source code for the controls in Android and **XAML** is used in Windows phone.

For designing Sopima.Windows UI, Blend UI design tool was used. The tool is shipped with Visual studio and comes very handy in designing user interfaces for web and desktop applications. It gives a better insight into the layout structure and visibility considering that layouts in **XAML** can have a deep nesting.

As what comes to **SopimaMobile.Droid**, the standard UI builder was used also some parts of UI were defined programmatically since it is easier to manipulate and design it specifically for your needs.

Standard set of user controls is available for creating UIs in Xamarin either by drag-and-drop the controls or writing the source code manually. It is also possible to define controls dynamically in the backend according to dynamically declaring user controls and sometime it comes quite handy.

Of course, when designing the UI it is necessary to take into the consideration different screen sizes and layout orientation.

In Android it is possible to design layout per an orientation by simply creating a new folder in the project called layout-land and placing .axml files that related to the landscape layout. When dealing with the images, there should be created several folders with the appropriate name conventions and different size of images placed accordingly. Note that if there it is not handled appropriately. It will not be accepted to the app store.

In Windows image sizes predefined in **package.appxmanifest** file. In most cases, if a **GridView** or **HubView** is used for building a layout the orientation changing should not affect the view much, though if it is not the case, than again **package.appxmanifest** is used.

ListView UI control is mostly used throughout the application in both Windows and Anroid, since it is able to handle a large set of data providing scrolling along with pagination. It provides a choice of using a standard list view or designing custom styled lists. What concerns Android, in most cases in the application, custom designed **ListViews**

were used, because the data displayed has a more difficult structure. When dealing with Windows it is a bit simpler, because **XAML** supports the data binding quite nicely, so it is easier to work with.

The user interface is implemented in the most simple and standard way for both platforms and according to the each platform standards. The colors and fonts were selected based on the Sopima web application.

5 Project conclusions

The goals of the thesis were reached and the project was successful and ready for further improvement and development. As a result, a functional cross-platform mobile application that works on Windows and Android platforms is ready. The application allows users to view, browse, search, and share their contracts whenever they want even if they do not have access to their computers.

Xamarin mobile platform was also well explored, and the advantages and disadvantages of using the platform became clear.

Achievements

During the development process, new skills and knowledge were acquired along with the improvement of the existing skills such as:

- development process planning
- strategic thinking
- project architecture
- working with cross-platform framework
- programming skills

Methods

The method that was used for achieving the goals of the thesis is rapid application development. Upon the achievements, it is possible to conclude that use of the RAD was the right choice and it lead to the good results.

Resources

Since Xamarin is relatively new technology, not many resources are available yet, though there are enough tutorials and examples for a developer to be able to create a nice mobile application.

Mostly, Xamarin official web site was used as the main source for learning Xamarin as well as existing projects on the GitHub.

Misleading

One of the biggest misleading was not taking into consideration the possible obstacles that can influence the time consumption for developing the applications such as:

- relying on third parties applications and libraries can bring unexpected issues
- learning new technologies along with the development process can be a bit more time consuming

As the result, too wide scope for the project was defined for such short period of time. Considering the time restraints, too ambitious goals were set, therefore, a new strategy plan was developed along the way.

Obstacles

There were also some obstacles during the development process. At the very beginning of the project, it was important to implement the authorization process, since it was not possible to retrieve any data for the mobile application from the Web Api. This means that the mobile apps are dependent on the third party and it is important to configure the Web Api correctly in order for the whole process to work. A lot of time was spent trying to get through and retrieve data from the Web Api because of a small configuration issue in the Web Api itself that did not allow the access to the data. By brainstorming, debugging and attention to the details the issue was resolved and it was possible to query data from the Web Api.

Besides, development of the mobile application for iOS platform was planned, but since it was a bit problematic to get a Mac for the application deployment, it was decided to post-pone the development of iOS mobile application.

Area for improvements

There is still need for code refactoring in order to improve code maintainability and readability which includes creating **ViewModels** for data binding. Moreover, functionality testing should be implemented. Furthermore, security check is crucial for such type of application. Hopefully, Xamarin Forms will support the Windows Store 8.1 platform that it would be possible to design a cross-platform UI as well which can really speed up the development process.

Further development ideas

There is a plan for development of Sopima mobile for iOS platform. Apart from just reading, browsing, and searching contracts there should be possibility for quick editing, deleting, and creating new contracts. More than that, it would be ideal to have some sort of optical character recognition implemented that will help to transit paper contracts straight to electronic version on a mobile phone.

6 Summary

In this thesis project a working mobile cross-platform application has been developed (Windows and Android), using common business logic that is shared between the platforms. To enable cross-platform work environment, Xamarin platform was installed as a part of Visual studio 2013.

The code sharing is implemented by adding a Portable class library project and storing the data model in there. By simply adding a reference from a platform specific project to the PCL, it is possible to manipulate the data models and pass it to the UI.

Azure Active Directory came handy in implementing the authorization process. HttpClient along with asynchronous programming were used to query data from the Web API.
The rapid application development was used as the main development method.
The entire thesis project development process included several phases:

- goals, objectives, and scope definition
- product backlog creation,
- gathering necessary knowledge background
- setting the development environment
- development process
- writing the project report

UI for each platform was developed using the standard user interface controls, available via UI designers.

To conclude, the Xamarin in company of .Net and C# is a powerful tool for developing cross-platform applications without compromising on native features. It has helped a lot in developing Sopima mobile applications and provided nice developer experience.

References

Developer guides. URL: http://developer.xamarin.com/guides/. Accessed on 20.10.2014

Hyper-V. URL: https://technet.microsoft.com/fi-fi/windowsserver/dd448604.aspx. Accessed 16.03.2015

Oracle VM VirtualBox. URL: http://www.oracle.com/us/technologies/virtualization/virtualbox/overview/index.html. Accessed 22.02.2015

Portable HttpClient for .NET Framework and Windows Phone. URL: http://blogs.msdn.com/b/bclteam/archive/2013/02/18/portable-httpclient-for-net-framework-and-windows-phone.aspx. Accessed on 16.03.2015

Process/Project RAD - RAD - Rapid Application Development Process. URL: http://www.projectmanagement.com/content/processes/11306.cfm. Accessed on 22.12.2014

Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET. URL: https://msdn.microsoft.com/en-us/magazine/dn463786.aspx. Accessed on 16.03.2015

Tendulkar, M., 2015. Authenticate Xamarin Mobile Apps Using Azure Active Directory. URL: http://blog.xamarin.com/authenticate-xamarin-mobile-apps-using-azure-active-directory/. Accessed on 31.01.2015

What is DLL? URL: http://support.microsoft.com/en-us/kb/815065. Accessed 16.03.2015

What's a Windows Runtime app? URL: https://msdn.microsoft.com/en-us/library/windows/apps/dn726767.aspx. Accessed 22.12.2014

Xamarin Android Player. URL:http://developer.xamarin.com/guides/android/getting_started/installation/android-player/. Accessed on: 01.11.2014

Appendices

Appendix 1. Product backlog

1.	The application must have authorization feature. A user can log in to the application via Microsoft Azure Active Directory.	Done
2.	The application must work on Android platform. A user is able to use the application on a device based on Android OS.	Done
3.	The application must work on Windows platform. A user is able to use the application on a device based on Windows OS.	Done
4.	The application must work on IOs platform. A user is able to use the application on a device based on IOS.	Postponed
5.	The application must have search feature. A user can search for a contract by typing keywords. Android	Done
6.	The application provides the possibility to choose an organization. A user can select an organization he or she belongs to.	Done
7.	The application provides the possibility to see organization's news. A user can see the latest news of the selected organization.	Done
8.	The application allows browsing the folder with contracts. A user can see folders with contracts.	Done
9.	The application provides the possibility to view contracts. A user can view a contract. Android	Done
10.	The application must have a user friendly UI for Android mobile devices.	Done
11.	The application must have user friendly UI for Windows mobile devices.	Done
12.	The application must have a user friendly UI for IOS devices.	Postponed

Appendix 2. PCL shared LogIn method

```
public static async Task<List<OrganizationModel>> LogIn(IAuthorizationParameters par-
ent)
      {
         dynamic jsonResult;

         try
         {
            _authenticationContext =
               new AuthenticationContext(_authority);

            if (_authenticationContext.TokenCache.ReadItems().Any())
            {
               _authenticationContext =
                  new AuthenticationCon-
text(_authenticationContext.TokenCache.ReadItems().First().Authority);
            }
            else
            {
               _authResult =
                  await _authenticationContext.AcquireTokenAsync(_resourceId, _clientId,
_returnUri, parent);
            }
         }
         catch (Exception ee)
         {
            throw new Exception("Error while authorizing!", ee);
         }
        var organizations = new List<OrganizationModel>();

         try
         {
           var json =  await SendRequest("organizations");
            jsonResult = GetJsonResult(json);
         }
         catch (Exception ex)
         {
            throw new Exception();
         }

         foreach (JToken result in jsonResult)
         {
            organizations.Add(new OrganizationModel
            {
               OrganisationId = (string) result["Id"],
               OrganisationName = (string) result["Name"]
            });
         }
         return organizations;
      }

      private static dynamic GetJsonResult(string json)
      {

         if (JValue.Parse(json) as JArray == null)
         {
            return JObject.Parse(json);
         }

         else {
```

```
        return JArray.Parse(json) as JArray;}
    }

        {
            throw new HttpRequestException();
        }
      }
      return null;
    }
```

Appendix 3. LogIn method implementation Android

```
namespace SopimaMobile.Droid
{
    using System.Collections.Generic;
    using System.Globalization;
    using System.Linq;
    using System.Net;
    using System.Net.Http;
    using System.Net.Http.Headers;
    using System.Threading.Tasks;
    using Android.Graphics;
    using Android.Graphics.Drawables;
    using Android.Provider;
    using Com.Microsoft.Aad.Adal;
    using Core.Models;
    using HelperClasses;
    using Microsoft.IdentityModel.Clients.ActiveDirectory;
    using SopimaMobile.Core.Services;


    [Activity(MainLauncher = true)]
    public class MainActivity : Activity
    {
        private ImageView imLauncher;
        private Button btn_logIn;
        private Button get;
        public static List<OrganizationModel> orgs;

 protected override void OnActivityResult(int requestCode, Result resultCode, Intent data)
        {
            base.OnActivityResult(requestCode, resultCode, data);
            AuthenticationAgentContinuationHelp-
er.SetAuthenticationAgentContinuationEventArgs(requestCode, resultCode,
                data);
        }
protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);

            // Set our view from the "main" layout resource
            SetContentView(Resource.Layout.Main);

            //Set action bar color
            ColorDrawable colorDrawable = new ColorDrawable(Color.ParseColor("#FF061345"));
            ActionBar.SetBackgroundDrawable(colorDrawable);

            //remove icon and title from action bar
            ActionBar.SetDisplayShowHomeEnabled(false);
            ActionBar.SetDisplayShowTitleEnabled(false);

            imLauncher = FindViewById<ImageView>(Resource.Id.im_launcher);
            imLauncher.SetImageResource(Resource.Drawable.sopima_logo);

            // Get our button from the layout resource,
            btn_logIn = FindViewById<Button>(Resource.Id.btn_logIn);

            // and attach an event to it
            btn_logIn.Click += async delegate
            {
```

```
            await ProgressDialogHelper.ShowProgressDialog(this,
                "Please wait...",
                "Logging in...",
                async () => orgs = await CloudServices.LogIn(new AuthorizationParameters(this)));
              // Navigate to next page
Intent organizationsIntent = new Intent(this, typeof(OrganizationSelectorActivity));
            if (orgs.Count != 0)
            {
                StartActivity(organizationsIntent);
            }
            else
            {
                Toast.MakeText(this, "Sorry, you have no organizations registered!",
ToastLength.Long).Show();
            }
        };
    }
```

Appendix 4. LogIn method implementation Windows

```
namespace SopimaMobile.Windows
{
    using Core.Models;
    using Core.Services;
    using global::Windows.UI.Core;
    using Microsoft.IdentityModel.Clients.ActiveDirectory;

    /// <summary>
    /// An empty page that can be used on its own or navigated to within a Frame.
    /// </summary>
    public sealed partial class MainPage : Page
    {

        private List<OrganizationModel> _orgs;
        public MainPage()
        {
            this.InitializeComponent();

            this.NavigationCacheMode = NavigationCacheMode.Required;

        }

        /// <summary>
        /// Invoked when this page is about to be displayed in a Frame.
        /// </summary>
        /// <param name="e">Event data that describes how this page was reached.
        /// This parameter is typically used to configure the page.</param>
        protected override  void OnNavigatedTo(NavigationEventArgs e)
        {
        }

        private  async void Button_Click(object sender, RoutedEventArgs e)
        {
//Wait for the organizations list and navigate to the next page
            _orgs = await CloudServices.LogIn(new AuthorizationParameters());
            Frame.Navigate(typeof (Organizations), _orgs);
        }
    }
```

Apendix 5. Displaying results of the LogIn method call in Windows app

```
<Page
   x:Class="SopimaMobile.Windows.Organizations"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:local="using:SopimaMobile.Windows"
   xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
   xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
   mc:Ignorable="d"
   Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

   <Grid x:Name="OrganizationsGrid" Background="#FF061345">
      <Grid.ColumnDefinitions>
         <ColumnDefinition Width="2*"/>
         <ColumnDefinition Width="3*"/>
      </Grid.ColumnDefinitions>
      <TextBlock HorizontalAlignment="Left" Margin="10,30,0,0" TextWrapping="Wrap"
Text="organizations" CharacterSpacing="50" VerticalAlignment="Top" Height="65" Width="380"
FontSize="50" SelectionHighlightColor="{x:Null}" Grid.ColumnSpan="2" Fore-
ground="#FFE8E8E8"/>
      <ListView x:Name="OrganizationsList" HorizontalAlignment="Left" Height="497" Mar-
gin="10,130,0,0" VerticalAlignment="Top" Width="380" ItemsSource="{Binding}" Fore-
ground="#FFF9F6F6" BorderThickness="1" FontSize="26.667" Grid.ColumnSpan="2" Selection-
Changed="OrganizationsList_SelectionChanged" Background="Transparent">
         <ListView.ItemTemplate>
            <DataTemplate>
               <StackPanel Height="55" Margin="5" Width="500">
                  <StackPanel.Background>
                              <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
                                          <GradientStop Color="#FF104DC3" Offset="1"/>
                                          <GradientStop Color="Transparent"/>
                                          <GradientStop Color="#FB134FC3" Offset="0.987"/>
                                          <GradientStop Color="#A95F87D6"/>
                                          <GradientStop Color="#05F9FAFD"/>
                                          <GradientStop Color="#020D0DFF" Offset="0.476"/>
                                          <GradientStop Color="#F61651C4" Offset="1"/>
                              </LinearGradientBrush>
                  </StackPanel.Background>
                   <TextBlock Text="{Binding OrganisationName}" Margin="5" FontSize="30" Hori-
zontalAlignment="Stretch" VerticalAlignment="Stretch" Height="82" FontFamily="Helvetica Neue
LT Pro">

                   </TextBlock>
               </StackPanel>
            </DataTemplate>
         </ListView.ItemTemplate>
      </ListView>
   </Grid>
</Page>
```

```
namespace SopimaMobile.Windows
{

   using Core.Models;
   using Core.Services;
   using global::Windows.Phone.UI.Input;

   /// <summary>
   /// An empty page that can be used on its own or navigated to within a Frame.
   /// </summary>
```

```
    public sealed partial class Organizations : Page
    {
        private List<OrganizationModel> _orgs;
        public static OrganizationModel SelectedOrganization { get; set; }

        public Organizations()
        {
            this.InitializeComponent();
            this.NavigationCacheMode = NavigationCacheMode.Required;

        }
        /// <summary>
        /// Invoked when this page is about to be displayed in a Frame.
        /// </summary>
        /// <param name="e">Event data that describes how this page was reached.
        /// This parameter is typically used to configure the page.</param>
        protected override void OnNavigatedTo(NavigationEventArgs e)
        {
//Handle the parameter passed from the previous page
            if (e.Parameter as List<OrganizationModel> != null)
            {
                _orgs = e.Parameter as List<OrganizationModel>;
            }

            if (_orgs != null)
            {
                OrganizationsList.ItemsSource = _orgs;

            }
        }

 private void OrganizationsList_SelectionChanged(object sender, SelectionChangedEven-
tArgs e)
        {
            SelectedOrganization = _orgs[OrganizationsList.SelectedIndex];

            CloudServices.SelectedOrganization(SelectedOrganization.OrganisationId);

            Frame.Navigate(typeof(DashboardPage));

        }

    }
}
```

Appendix 6. Displaying results of the LogIn method call in Android app

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:minWidth="25px"
   android:minHeight="25px"
   android:background="#FF061345">
   <ListView
      android:minWidth="25px"
      android:minHeight="25px"
      android:layout_width="match_parent"
      android:divider="@drawable/divider_style"
      android:layout_height="wrap_content"
      android:background="#FF061345"
      android:dividerHeight="4px"
      android:id="@android:id/list" />
</LinearLayout>
```

```
namespace SopimaMobile.Droid
{
   using System.Collections.Generic;
   using Android.App;
   using Android.Content;
   using Android.Graphics;
   using Android.Graphics.Drawables;
   using Android.OS;
   using Android.Views;
   using Android.Widget;
   using Core.Models;
   using Core.Services;


   [Activity(Label = "Organizations")]
   public class OrganizationSelectorActivity : ListActivity
   {
      private ListView listView;
      private List<string> items = new List<string>();
      public static string selectedOrganizationId;
      public static string selectedOrganizationName;

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

         SetContentView(Resource.Layout.Organizations);

         ColorDrawable colorDrawable = new ColorDrawable(Color.ParseColor("#FF061345"));
         ActionBar.SetBackgroundDrawable(colorDrawable);
         TitleColor = new Color(255,255,255);
         listView = FindViewById<ListView>(Android.Resource.Id.List);

         // Get user organizations
         List<OrganizationModel> list = MainActivity.orgs;
         foreach (var organizationModel in list)
         {
            items.Add(organizationModel.OrganisationName);
         }
         var orgList = items.ToArray();
```

```
            //populate listView with the names of the user organizations
            listView.Adapter = new ArrayAdapter<string>(this, An-
droid.Resource.Layout.SimpleListItem1, Android.Resource.Id.Text1, orgList);
            ListView.FastScrollEnabled = true;

        }

        protected override void OnListItemClick(ListView l, View v, int position, long id)
        {
            base.OnListItemClick(l, v, position, id);
            selectedOrganizationName = MainActivity.orgs[position].OrganisationName;
            selectedOrganizationId = MainActivity.orgs[position].OrganisationId;
            CloudServices.SelectedOrganization(selectedOrganizationId);
            Intent organizationIntent = new Intent(this, typeof(SelectedOrgActivity));
            organizationIntent.PutExtra("organizationId", MainActivity.orgs[position].OrganisationId);

            StartActivity(organizationIntent);

        }
    }
}
```

Appendix 7. Sending Web API request

```
    private static async Task<string> SendRequest(string uri)
    {
        // Create request
        var request = new HttpRequestMessage(HttpMethod.Get, uri);
        // Call the service.
        using (var httpClient = new HttpClient())
        {
//Define a default header
httpClient.BaseAddress = new Uri(_baseUri + "api/1/");
            httpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQuali-
tyHeaderValue("application/json"));
            httpClient.DefaultRequestHeaders.Authorization = new Authentication-
HeaderValue("Bearer", _authResult.AccessToken);
            if (_orgId != null)
            {
                httpClient.DefaultRequestHeaders.Add("CurrentOrganizationId", _orgId);
            }
            try
            {
                HttpResponseMessage response = await httpClient.SendAsync(request);
                response.EnsureSuccessStatusCode();

                if (response.IsSuccessStatusCode)
                {
                    return await response.Content.ReadAsStringAsync();
                }
            }
            catch (HttpRequestException ex)
            {
                throw new HttpRequestException();
            }
        }
        return null;
    }
```

Appendix 8. Attachment Loader Windows

```
private async void FilesListBox_SelectionChanged(object sender, Selection-
```

```
ChangedEventArgs e)
      {
         var file = FilesListBox.SelectedItem as File;

         var uri = CloudServices.GetAttachmentUri(file.FileName, selectedContractId);

         await AttachmentLoader.AttachmentLoadInBrowser(uri);

      }

namespace SopimaMobile.Windows
{
  public static class AttachmentLoader
   {
      public static async Task AttachmentLoadInBrowser(string uri)
      {
         // Launch the URI
         var options = new LauncherOptions();
         options.TreatAsUntrusted = true;

         Uri url = new Uri(uri);
         var success = await Launcher.LaunchUriAsync(url, options);
      }
   }
}
```

Appendix 9. Search functionality Android

```
namespace SopimaMobile.Droid
{
```

```
    using Android.Support.V4.View;
    using Android.Views;
    using Android.Widget;

 class SearchViewExpandListener : Java.Lang.Object, MenuItemCom-
pat.IOnActionExpandListener
    {
        private readonly IFilterable _adapter;

        public SearchViewExpandListener(IFilterable adapter)
        {
            _adapter = adapter;
        }
        public SearchViewExpandListener()
        {
        }
        public bool OnMenuItemActionCollapse(IMenuItem item)
        {
            return true;
        }

        public bool OnMenuItemActionExpand(IMenuItem item)
        {
            return true;
        }
    }
}
```