Graduation Project – Agent VI innoVi

Amikam Goldfarb
Nadav Ramot
Gil Ronen

Project's git:

www.github.com/goldami1/GraduationProject

Table of Contents

Brief Introduction

AgentVI App

- AgentVI App is intended to provide support in the mobile platform to Innovi AgentVI clients.
    - Live stream from sensors.
    - Visibility of events.
    - Visibility of sensors.
    - Filtration based on sites.
    - Marking areas of interest.
    - Tagging events.
    - Sensors health history visibility.
    - Notifications based on real time events.

Xamarin

- Xamarin enables to create code once (most of the code, depends of functionality), and to build and run it on all platforms (Android, iOS, Windows Phone, etc). – Xamarin is a Cross-Platform dev platform.
- UI and App's Views/Pages are written in Xamarin Shared project both in XML code and in C# code-behind the xaml.
- There's no difference between the two mentioned writing techniques – as the XML code is decoded into C# prior during the build process. However, XML code, that actually is called XAML code, is much easier to define and to write.
- Great intractable for this matter can be found in the following link: www.youtube.com/watch?v=93ZU6j59wL4
- This lecturer and others have practical and easy to learn video series. (UDEMY/Coursera/etc) – recommended investment.
- XAML code is lastly decoded to Native code for each project's platforms, and although it's possible to develop app's interfaces directly in Native code for each platform, This big advantage of using Xamarin.Forms in building once from the same shared code to all platforms and making this code to work almost exactly as the Native one (very close in terms of visibility up to some difference in Controls, Fonts, Colors, Templates and etc…) entirely worth it.
- As a great advantage – in every part of the code – it's possible to run platform-specific query by querying the Device.RuntimePlatform value.

Basically, Xamarin supports most of smartphone resources (camera, microphone, push-messages, etc). Furthermore, in terms of support, MSDN website, Xamarin wide development community.

Getting Started

Proxy

- The first weeks were spent to learn and understand the http protocol and to choose the best C# libraries compatible with Xamarin to make API calls – for more details on this subject see "Recommended Libraries".
- Knowing the API's various data types and methods was crucial for optimal planning of the proxy structure
- To test API calls easily without the need to write code it is recommended to use a tool such as Postman.
- Creating a dummy proxy for working offline

Mobile

- First few weeks were spent in understanding the correct architecture for a Cross-Platform Xamarin project, in investigating, choosing, and testing C# libraries that are needed for our project. Most of the time, it was a "try and see" process, as there're a lot of nugets (libraries) which don't provide exactly what they intend to.
- There was an initial learning curve in understanding how and when to work with dependency injection design pattern in Xamarin's MVVM Shared project.
- First Phase of project was mostly centered around building UI basic design – in terms of learning and building xaml code.
- At the end of the first phase, it was crucial to create minimal integration with proxy. It was one of the most valuable milestones – as there were a lot of desynchronization between how the code was written, and how the project was built at first, and how it was supposed to be – so the two projects would communicate and run in harmony.
- For iOS development – MacOS Virtual Machine (Possible with VMWare on Windows) is needed to be installed and configured. Development can be done in Visual studio for Mac, or in Window's Visual studio – while MacOS machine is configured as a building endpoint for iOS project (in Visual Studio for Windows).

Recommended Libraries

Proxy

- We have found that the best libraries to use for API calls while developing under Xamarin are Json.net and HttpClient. Attempting to use different tools such as System.Json and HttpWebRequest/HttpWebResponse resulted in various exceptions / Compiler errors due to incompatibility issues with Xamarin. The Json,net library can be easily installed as a Nuget package in visual studio.
- Some standard .Net libraries under the System,Net.Http namespace may also come in handy.

Mobile

- Libraries that were used for developing the Xamarin App were:
  - Plugin.DeviceOrientation
  - Plugins.Popup
  - Plugin.Settings
  - FFImageLoading
  - InfiniteScrolling
  - Standard .Net libraries for Xamarin.Forms platform.
  - Octane.VideoPlayer
- Plugin.DeviceOrientation library turned out to be a time-consuming bottleneck – thus, a performance consideration and fine-tuning was required to achieve good app performance.
- Octane VideoPlayer library was firstly used in first Phase, and it was used to display videos received from server's API. However, after API update – videos were received in HLS way (instead of mp4 file), which isn't supported by this library. Thus, further consideration and fix was needed.

Saving Sensitive Data

- Implementation of saving sensitive data feature in our App is done by the help of settings plugin.
- In this way, specific properties are saved directly into each platforms native settings APIs (NSUserDefaults, SharedPreferences, etc…).
- This ensures the fastest, most secure, reliable and effective creation and editing settings in our app.
- App.Current.Properties obj in Xamarin actually serializes and deserializes items to disk. This process is done without any security considerations.
- On other hand, saving directly to native platforms settings is more reliable. This type of saving data achieved by additionaly using the settings plugin.

Displaying Videos

- As server's API was intended at first to provide video interface (recording) through a file located under server's domain – pages which supported video appearance – were using VideoPlayer nugget which provides a responsive UI component for Video display and control.
- Following an API update, which changed the way how videos are received – there was needed additional thinking and investigation – as the component that was used for this use case - turned out invalid and unsuitable.
- New video interface of server's API (both for video-stream, and video-recording) became an HLS (Http Live Streaming) one.
- To provide support in our App for this new way of receiving video – there were two possibilities.
    - Either to inherit from native video players and implement a cross-platform component which supports the HLS video playing
    - Either to use browser's engine (which supports playing HLS videos as part of HTML5 support)
- As the second option turns out to be the quickest and bug-free workaround – it was our path decision.
- In terms of "nice to have" it's possible to inject into WebView JS code which can provide additional behavior for the video player page that is shown.

Filtering & Caching

- During App's development, it was substantial that proxy's http requests can cause performance issues as these tightly depends on cellphone's connectivity state.
- For untightening this bond – a caching module was implemented for solving this matter.
- This module is periodically polling the most up-to-date hierarchy data.
  Until this hierarchy isn't fully fetched – App uses the hierarchy achieved from API.

Xamarin Forms / Dev&Dep Env. Issues

Proxy

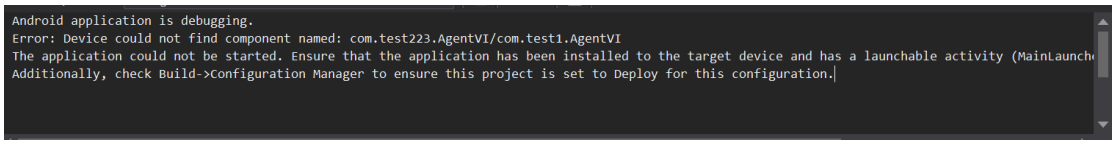
- SSL certificate issue: If the API does not have an official SSL certificate an exception is thrown while sending the http request due to security issues. Attempting to bypass this error by changing the Handler from ServicePointManager.ServerCertificateValidationCallback results in an exception in Xamarin project but works in .Net core projects. We have not been able to discover the reason for this so far. For this reason it is essential for the API to have an official certificate to avoid this problem.

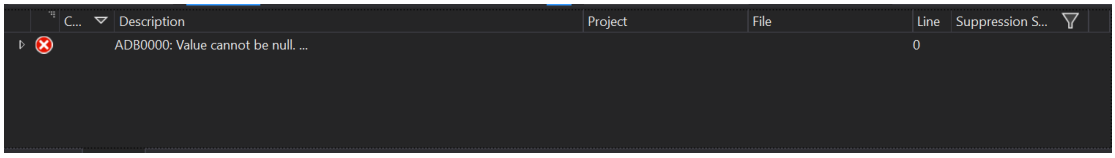
Mobile

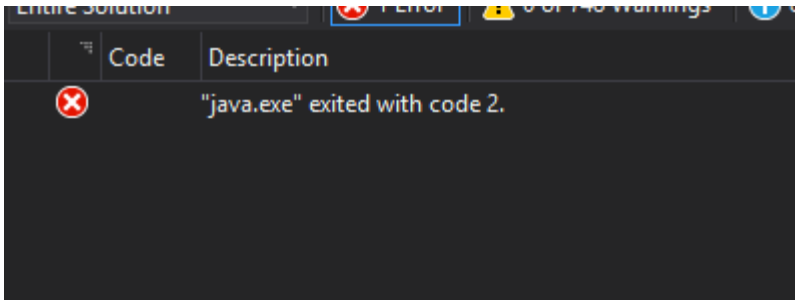
- TLS certificate issue: if API does not support an official certificate, it causes an inconsistent behavior in project's different platforms.
  - In android – a warning appears during HLS (https) video playing.
  - In iOS – player does not play the content at all. This gap leads toNSExceptionDomains exceptions.
    - Until iOS 9 – a workaround of temporary approving this insecure connection was possible. For later iOS versions this workaround was blocked.
- As having a workaround for this matter is considered as bad practice, and mostly isn't possible lately - PI should provide a TLS certificate.

- Multiple Xamarin components and especially Page Templates aren't Cross-Platform as they intend to.
    - TabbedPage in iOS creates a page with tabs bar in it's buttom (which includes both image and text). While in Android – tabs bar is located in the upper part, and tabs lack icons support.
    - Button text in iOS displayed exactly as the input it receives. While in Android it, button displays it text in capital letters.
    - Label control in Shared project doesn't support an underline, while in native projects this control does support this functionality.
- As a reason of all this inconsistency – custom renderers and custom controls were implemented to keep the UI close to project's characterization.
- Features that use reflection prevent from using Xamarin Live Player. However, the debugger can be used instead. Thus, Xamarin Live Player is relevant mostly only for debugging design and xaml Phase.
- Even though project is rebuilt - Xamarin.Forms sometimes use the old code for the rebuilding process. To fix this matter – clean the solution, remove obj and bin folders, and proceed once again with the rebuilding process.
- Views aren't bindable. Meaning, a page that is inserted into a ContentView control – loose its binds. This issue was a real burden in developing the way our UI works. Following aspects describe our considerations and solution:
    - On the one hand, TabbedPage is designed for entirely different content per tab. Meaning – no shared/static content between tab views.
    - On the other hand, our application's view is mostly static on its bottom part and upper part (tabs layout, header layout).
    - Thus, TabbedPage couldn't be used (additionally to its inconsistency between different platforms).
    - Instead, a ContentPage is used to display app's view (header+content+footer). While content in this page is changed dependently on the current binding and user's interaction with the app.
    - This functionality was implemented as a slave functionality to Xamarin's NavigationPage.
    - This "Slave Navigation" is based on a stack which holds pairs of Page and its binding context, each time a view is changed – main page's binding context is updated related to the current state of the Slave NavigationPage.

- Internal properties aren't bindable. however, public once are indeed bindable.
- Following upgrade of XF version to >= 2.5 – the custom renderer of listview that was implemented to support caching strategy – was depreciated.
- Images and resources should be placed in root dir and composed only of small letters and underlines. Furthermore, all resources should be marked as AndroidResource for being built into the APK file.
- Upon deploy for Android, there's an environmental-independent error - "ResolveLibraryProjectImports" task failed unexpectedly error. To solve the problem - you should clean the solution. Then, build all solution's projects from top to bottom, each one at a time.
- During development phase, VS occasionally doesn't recognize the debugging platform (device) and/or doesn't allow build process. This issue is mostly appeared in following ways:







- To fix this matter – simply a solution clean, and bin+obj dirs clean are needed. Or an explicit change of Activity's name (in Android) is needed.

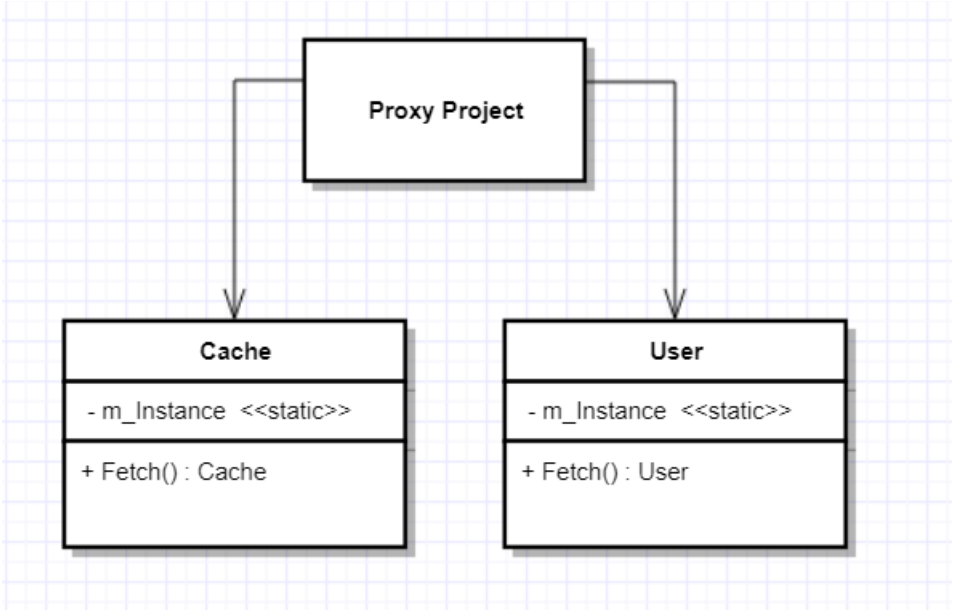
Design Patterns

Proxy

Singleton

There are two singleton classes in the proxy project: User, Cache

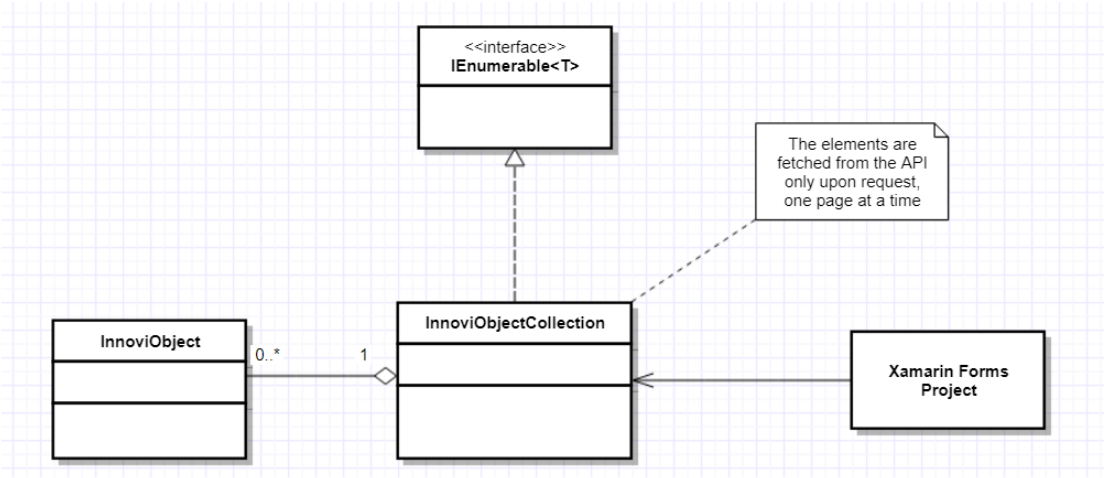
Both classes need to have one and only one instance which is accessible to various other classes in the proxy project.



Virtual Object Proxy

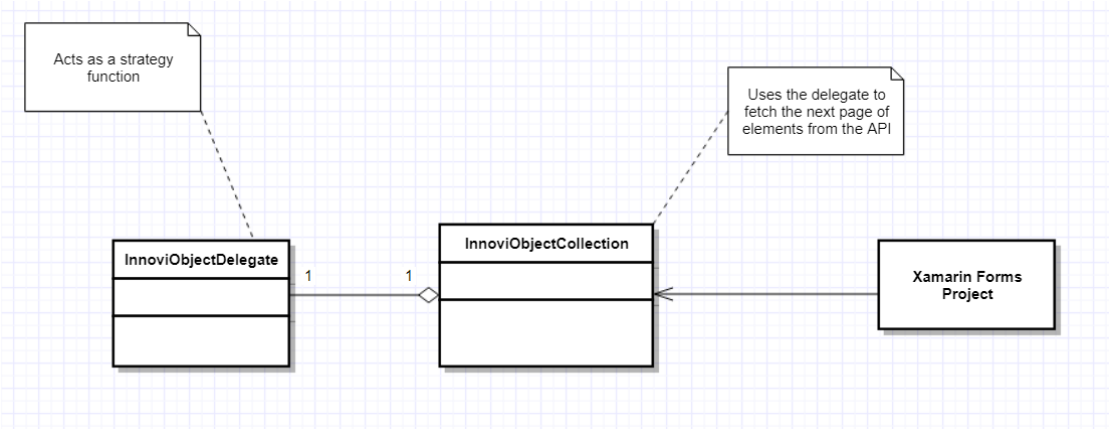
Since the data is fetched by http requests from a remote server, a need arose to have an enumerable collection which performs lazy creation of its elements on demand.

The InnoviObjectCollection class implements this by using a custom enumerator.

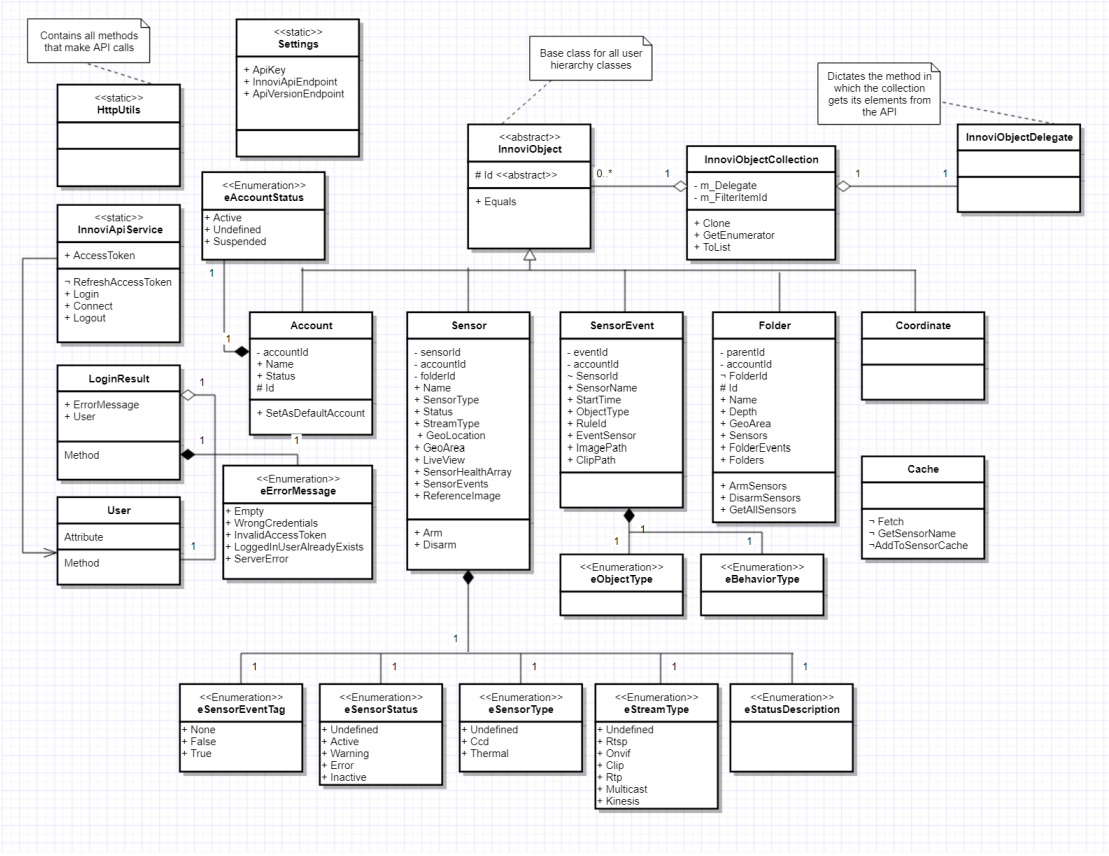


Strategy

The InnoviObjectCollection class needs to make different http requests depending upon the actual innovi elements they are holding. For this reason, it was necessary to implement a strategy design pattern (using a delegate instead of a strategy object). The delegate is given to the collection in the constructor and determines the way in which it fetches its next page of elements from the API server.



Proxy Class Diagram



Mobile

- Singleton Design Pattern for ServiceManager
- MVVM Design Pattern for Mobile project
- Observer Design Pattern
    - For Navigation between pages while maintaining binding context feature in View Control. Design Pattern is implemented in MainPage ViewModel in cooperation with its view (for visual updates).
    - For Navigation – in term of updating and populating the currently appeared page. (To replace the functionality of onAppeared method)
- Dependency Injection Design Pattern – for running different code for different platforms. Implemented to get device's rotation sensor data / update rotation state.
- Façade Design Pattern – Implemented under ILoginService and IFilterService interfaces. Intended for easier content access across the whole project.

    - Further details and schemes can be seen in project's git.