Sam Kazerouni 26658415
Omer Deljanin 27064438
Eric Leon-Rodas 26648754
Asma Laaribi 29326197
Jonathan Phillips 26699596
Joseph Daaboul 27077890

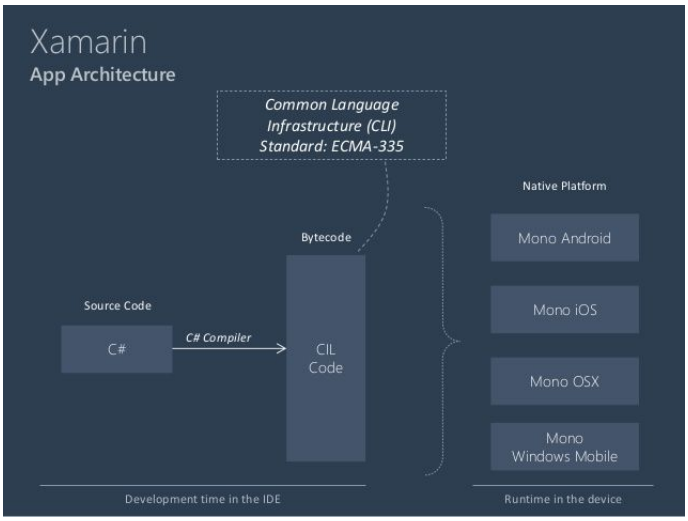
Iteration 2

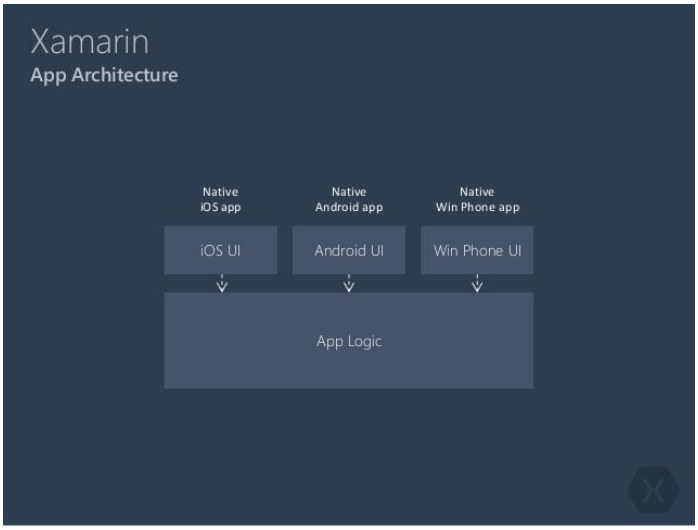
Glimpse is a location based application that aims to provide its users with the most up to date information related to local promotions and sales. Local businesses often have small scale promotions or one-day deals that are communicated to clients in person as they enter the store. This app will give these businesses a platform to advertize such sales to a wider audience. Buyers will be able to see the sales that are taking place in the stores or businesses close by. Glimpse will help businesses by giving them exposure to potential clients who are passing by.

Velocity

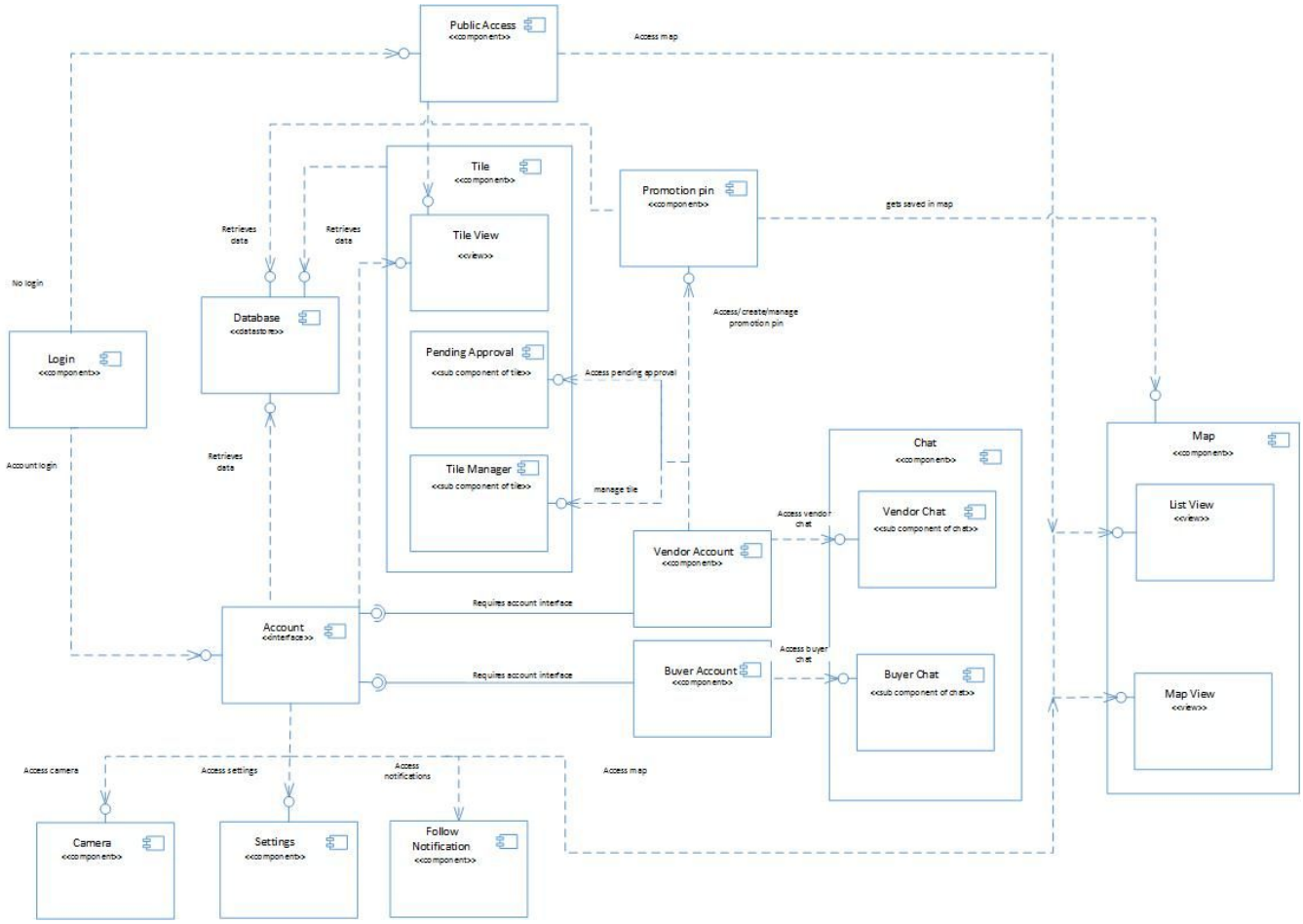
Iteration 1 Velocity: 0
Iteration 2 Velocity: 0 //No Unit Tests were Completed, Stories moved to Iteration 3.

Architecture and Diagrams





[1]



The business logic of the system will revolve around the account type used in the login phase. Each system component will behave differently depending on the type of

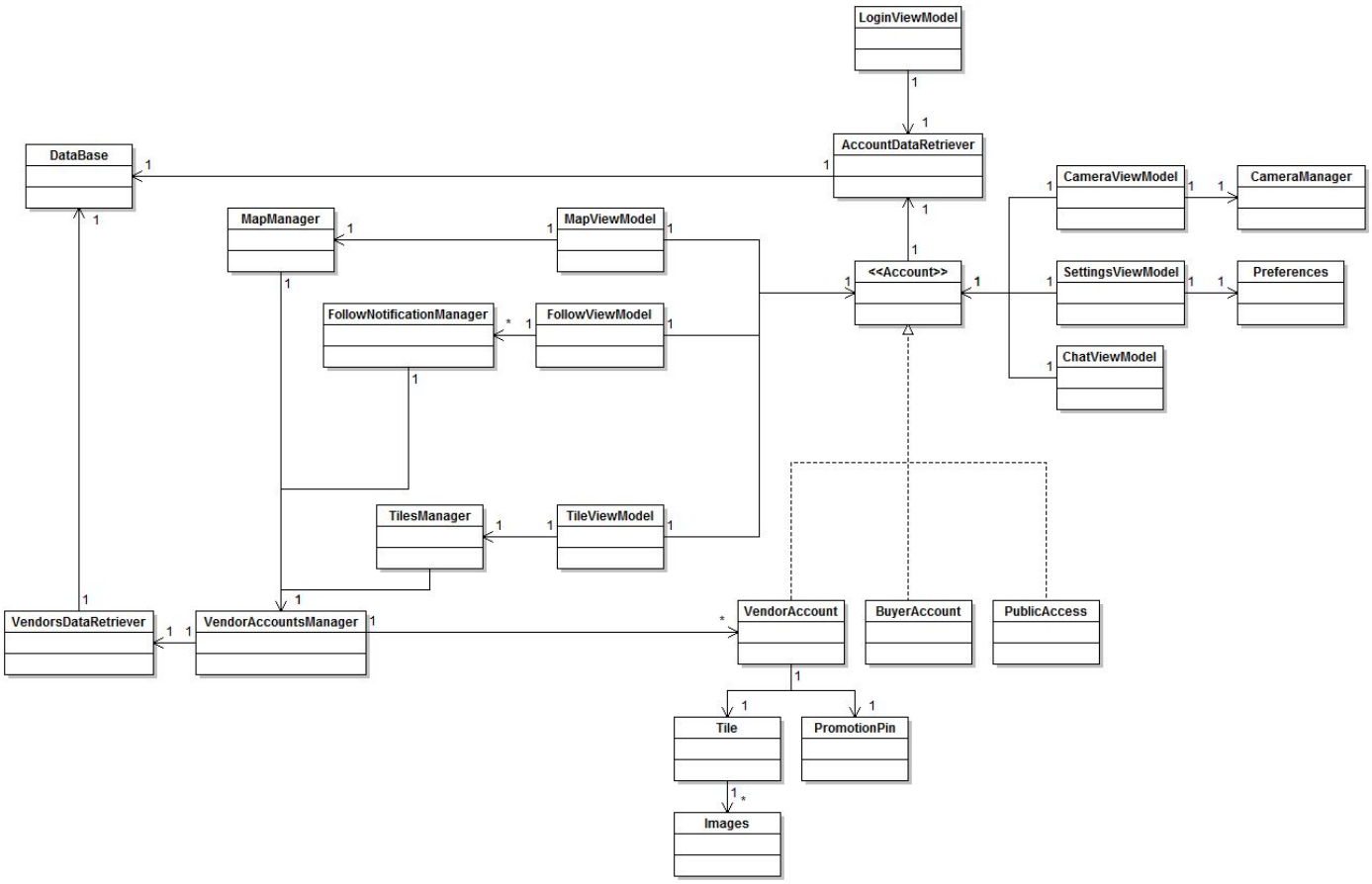
[1] http://www.slideshare.net/PrabhaKularathna1/xamarin-beyond-thebasics

account logged in. The components include the following: login, map, tile, notification, settings, and chat. The user interface views will be coded differently using mono and mono for android. However, each component will reference a view model which will contain the properties that will bind to different control elements of the view. The properties of the view models will perform business logic that will be common to both views coded in mono and mono for android.

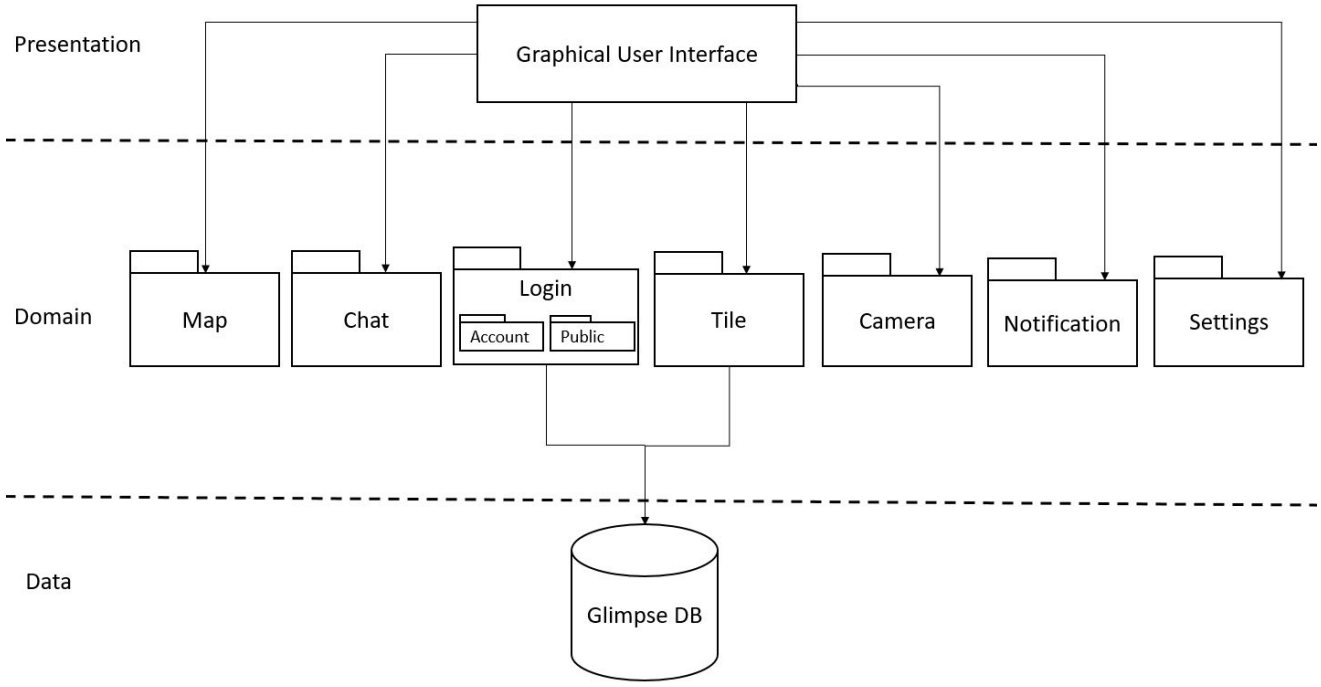
The different account types will be the *VendorAccount*, the *BuyerAccount* and the *PublicAccess*. Vendor accounts contain a the *PromotionPin* and the *Tile* objects.

The map, follow notification and tile components will use a *VendorDataRetriever* to access the vendor's account information from the database.

Moreover, the camera, settings, and chat components will not use the database to retrieve data. The camera will be able to access the photo library of the user's phone. The settings will store the user's settings locally on their phone and finally the chat will simply manipulate data loaded on the app.



The layer diagram of our application is represented below and separates Glimpse into 3 tiers: user interface, domain logic, and data tier. The user will interact with the application through the presentation layer. Information related to the Account will be stored and retrieved through the database. The logic tier coordinates the application and is composed of 7 main concepts: the map, the chat feature, the tile display, the camera, notifications, and the login process. Moreover it processes information between the data layer and the presentation layer.



Plan to Release 1 (Modified)

Iteration 1 (0 points)

During this iteration, we focused on the planning and preparation of the project. Therefore, we set up our GitHub page, planned our architecture, carefully selected our frameworks and focused on the goals of our application. We also did several low level prototypes in order to have a better picture of our application and how we can come up with something different than some of the competitors out there.

Iteration 2 ~~(32 points)~~ 0 points

During this iteration, we completed setting up our mvvmcross application framework which took more time than expected. Many team members had trouble setting up Visual Studio 2015 with their emulators as we had many compatibility problems. Some issues included environment variable modifications, emulators not loading projects properly, visual studio debugging tools not functioning, framework compatibility issues. All these issues delayed our iteration user stories. Several stories have been implemented however not yet tested as the Unit Testing project is not fully setup yet. Some of these issues were solved and documented on GitHub in the Wiki. We created a Setup Fixes/Issues folder for reference and also a project setup procedure highlighting everything from the start. There has been some work done on UI and business logic for the Vendor Account however nothing has been tested. All these

user stories will be pushed to the next iteration because they have not yet been tested. This means that some user stories from Iteration 3 will be removed.

Iteration 3 (Release 1) (32 + 21 points)

During this iteration, we are focused on implementing part of the main functionality of our application. We will start first by implementing the map and loading promotion pins on it.

Infrastructure

	Apache Cordova	**Xamarin (Selected choice)**	Native	Ionic
Performance	Good	**Good**	Best	Good
Cross-Platform	yes	**yes**	no	yes
Difficulty based on our skills	Medium	**Easy**	Hard	Medium
Open source	yes	**yes**	Yes for android, No for iOS	yes
Plugins compatibility	Medium	**High**	High	Medium
IDE	Mainly Netbeans	**Visual Studio**	Android Studio	Mainly Atom,WebStorm
Testing	Jasmine	**Automated. Xamarin.UITest (C#)**	Android -JUnit iOS - KIF	Jasmine
Language	HTML, JS,CSS	**C#**	Java	AngularJS, HTML, JS,CSS
Documentation	Medium	**High**	High	Medium

To determine what technology we should pursue for our application, we considered several important factors. First and foremost, we had to consider options

that were open source. With that being our primary focus , we were able to limit our options to four technologies (Cordova,Xamarin,Ionic,Native*). To be efficient, we want to develop using a technology that supports cross-platform development. All our options support that except native solutions. Although they are superior in performance, we would be losing time to be proficient in each platform thus diminishing returns. Cordova and Ionic are both web-based technologies & based off the individual skills on our team, we are most proficient using C# which makes Xamarin a better choice. Xamarin also has the superior documentation and testing frameworks.

*Our platform focus for native applications are Android & iOS. Android being open-source while iOS closed.

Application Framework (MvvmCross)

MvvmCross is a cross-platform mvvm framework that allows developers to create cross platform apps. Model-View-ViewModel is an software architectural pattern. MVVM is a variation of Martin Fowler's Presentation Model design pattern. Like Fowler's Presentation Model, MVVM abstracts a view's state and behavior. However, whereas the Presentation Model abstracts a view (i.e., creates a view model) in a manner not dependent on a specific user-interface platform.Several reasons why we are using

MVVM and Presentation Model both derive from the model–view–controller pattern (MVC). MVVM facilitates a separation of development of the graphical user interface (either as markup language or GUI code) from development of the business logic or back-end logic (the data model). The view model of MVVM is a value converter meaning the ViewModel is responsible for exposing (converting) the data objects from the model in such a way objects are easily managed and consumed. In this respect, the view model is more model than view, and handles most if not all of the view's display logic The view model may implement a mediator pattern, organizing access to the back-end logic around the set of use cases supported by the view.

We took advantage of all the benefits that are provided by MVVMCross, such as bindings, converters, IoC, etc. For the Portable Class Libraries, the other option to sharing code is file-linking. PCL is neat, tidy, and requires no overhead when your common code library changes. File-linking is a manual process and can begin to get cumbersome after constant project changes. Furthermore, extensibility of the application can be easily done using the concept of plugins. Plugins can be added to the framework which can make our application grow much faster.

The disadvantage of using MvvmCross framework is that there a learning curve.

https://mvvmcross.com/

Database (Selected choice: SQLite)

SQLite

For our database, we have chosen SQLite as it is very simple to use and integrate with our Xamarin Mono Applications (Android and iOS). iOS and Android both have the SQLite database engine "built in" and access to store and retrieve data is simplified by Xamarin's platform. It allows us to save and retrieve objects from database without writing SQL statements. It is efficient in the sense that it contains a thin wrapper on SQLite which is very fast and won't cause any performance bottleneck of our queries. It also features simple methods for executing CRUD operations and queries safely. It also works with our data model without forcing us to change our classes using a reflection-driven ORM layer.

ADO.NET

This uses SQLite database and is available for iOS and Android however it uses the ADO.NET-like syntax. It requires us to write SQL statements which can become tricky if the queries become complex. Many of our team members do not have expert SQL knowledge therefore we decided this is a risky technology to go with.

Realm Xamarin

It does not use ORM, Realm data models are defined using traditional C# classes with properties. However it is very easy to use and setup, similar to SQLite. In terms of speed, Realm claims they are faster however depending on how SQLite is used it can match the speed of Realm. Both Realm and SQLite work nicely with Cross platform applications. Realm is still new in the market and SQLite has been there for very long, therefore in order to be safe SQLite would be our selected choice.

Testing Framework (Selected choice: NUnit)

For the unit testing, we are presented with possible libraries to test with: NUnit and XUnit.

By looking at the advantages and disadvantages of each library, we have reached the conclusion that it doesn't really matter which one we use. They both give the tools for the developer to write good tests and the quality of the test won't change based on which library we use but rather on our skill at testing. We have chosen to go with NUnit since it seems more of a standard within C# and Xamarin. Xamarin provides tools for better UI tests which are implemented with NUnit. Also, tools such as Resharper provide better support for NUnit and run well within Visual Studio, so we are satisfied with it.

For UI testing, Xamarin has an internal library called Xamarin.UITest to validate the ui of iOS and Android devices. We are not planning to release on other devices such as Windows Phone so this will do the job for us. This library will be useful since it can be used to test the UI on all different versions of iOS and Android as well as being used as Acceptance tests. It can also be automated which is great to be able to do daily tests at night after everyone commits their code for example.

Name Conventions

Standard C# Coding Convention:
https://msdn.microsoft.com/en-us/library/ff926074.aspx