Fleet Management Cross Platform Application

Bruno Miguel Nascimento Carola

ABSTRACT

This document aims to showcase the development of a cross-platform mobile application focusing on Google's Android operating system and Apple's iOS. It's a car fleet management application called FleetMobile from Tecmic S.A.. This application was only available for Android systems, so this document intends to show the entire process of migrating it to the iOS system and consequently to a multiplatform system, displaying the decision making process and choices made along the way, working simultaneously with the growth, needs and demands of the technological world and users of the FleetMobile app.

This paper addressed the need urgency to make code reusable and a way to facilitate future maintenance and upgrade actions evenly for both target systems of interest, using various techniques and technologies, from Mvvm-Cross, Xamarin to Android and iOS. Additionally, Signal-R technology was used to satisfy the requirement of having real-time information on some application features.

Index Terms

Mobile Application, Fleet Management, Cross-Platform, iOS, Android, Design Pattern, Real Time Communications, Xamarin, Mvvm-Cross

INTRODUCTION

Tecmic - Tecnologia De Microeletrónica, a Portuguese multinational company founded in 1988, specialized in fleet control and management, developed in 2015 a native Android application called FleetMobile that extends the iZiTraN Web product. It allows real-time knowledge of the precise position of vehicles in a fleet on a map and their information, such as detailed data on driving times, kilometers traveled, fuel consumption, vehicle history and more. FleetMobile also allows fleet operators to contact their drivers via text messages and receive alerts of various incidents that may arise, as well as deal with them.

This project focuses on meeting the requirements and needs of users of the FleetMobile application. It studies the best methodology for modularly and interoperable developing this application to minimize future maintenance and upgrade costs on all required platforms (Android and iOS). It involves the study of several peculiar paradigms such as:

- The best tools for multiplatform development.
- The study of the use of iOS guidelines to interfaces already developed in Android.
- The best technology for real time communication.
- The best design pattern and techniques to make code reusable which will also facilitate future maintenance and upgrade actions evenly for both target systems of interest (Android and iOS).

STATE OF ART

Frameworks and design patterns

Native Development – Xcode

Currently, as already mentioned, the FleetMobile application is developed on Android using the Android Studio framework. It is possible to follow a line where the work already done is maintained and the iOS version is developed natively.

In 2003, Apple introduced an integrated development environment called XCode. In newer versions, this environment comes with modern but powerful features that provide a complete package for developers. XCode provides developers with an interface design mode, an intuitive and user-friendly development environment, as well as a UI testing environment. XCode's most powerful feature is the runtime, which continuously tracks and alerts programmers about syntax errors, providing recommendations and memory alerts. In addition, with the latest version, XCode provides a large number of extensions that can be easily and intelligently integrated into XCode.

In addition to a strong design component, Apple has developed a strong testing and simulation component. It enables the testing and simulating of the debug mode application on a simulator of any Apple-made device, from iPhone 4 to the latest iPhone, Smartwatches and iPods.

XCode supports several programming and scripting languages, including C, C ++, Objective-C, Objective-C ++, Java, AppleScript, Python, Ruby, ResEdit, and Swift. Nonetheless, Apple has decided to make Swift the main programming language of XCode. [2] - Swift is a modern and user-friendly language that is designed by users for users. It has the power of low-level programming languages like C or C ++ and the smoothness of high-level languages like C # or JavaScript. In addition, Swift is light and comes with a fairly complete predefined library.

Xamarin

Xamarin is a multiplatform framework that presents many features for building mobile applications, either more natively with Xamarin.Droid and Xamarin.iOS, or with Xamarin.Forms.

This framework combines the full power of native platforms by adding several powerful features by itself. Using the C # language, API, and data structures, on average it is possible to develop code where 90% of solution code is shared across all major platforms and with Xamarin.Forms interfaces it is possible to share approximately 100%. [3]

In Xamarin there are several methodologies that can be used to share cross-platform projects code such as .Net Standard Libraries, Shared Projects and Portable Class Libraries, better known as PCL, which will be detailed below. [4]

- **.NET Standard Libraries** - provide a well-defined set of base class libraries that can be referenced in different types of projects, including multiplatform projects such as Xamarin.Android and Xamarin.iOS. .NET Standard 2.0 is recommended as it provides maximum compatibility with existing .NET Framework code.
- **Shared Projects** – allows the writing of common code that is referenced by several different application projects. The code is compiled as part of each reference project and may include compiler directives to help incorporate platform-specific functionality into the shared code base.
- **PCL - Portable Class Library** – When an Application Project or Library Project is created, the resulting DLL is restricted to the specific platform for which it was created. A combination of platforms on which to run the code can also be chosen. However, these libraries are obsolete in newer versions of Visual Studio.

Apart from these methodologies there are several plugins and frameworks that can be used in both Xamarin Studio and Visual Studio, such as:

NuGet is a very popular Package Manager in the C # developer community that is available in both Xamarin Studio and Visual Studio. It allows developers to create, share, and consume .Net project code and libraries, providing all the tools they need for each of these functions.

Mvvm-Cross is a framework based on the MVVM design pattern (Model-View-ViewModel) which allows an increased code reusability on all platforms where the application will be developed.

The **MVVM** design pattern, **Figure 1**, separates the logic of a View object into two separate objects, one called **View** and the other called **ViewModel**. The **View** object is responsible for the user Interface and **ViewModel** is all about the main classes where all the common logic to all platforms is described.

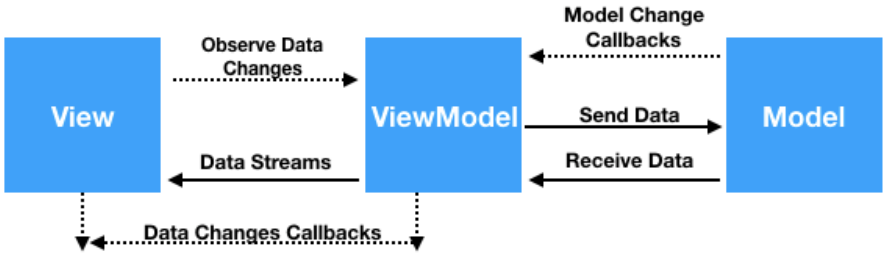

Figure 1 – MVVM design pattern;

PhoneGap

PhoneGap is a mobile application development framework based on an open source project (Apache Cordova). This framework allows developers to build HTML, CSS and JavaScript applications only once and deploy them to different mobile devices without losing the capabilities of a native application.

This framework offers multiplatform development with the freedom of custom development. As a cloud-based service, all applications will be built with the latest SDK for each target platform. PhoneGap Build supports multiple platforms including Android, iOS, Windows Phone and webOS with a single codebase.

The resulting applications are hybrid in the sense that they are neither native mobile nor purely web-based applications. Instead of using the native UI of each platform, all layout rendering is done through webviews, on the other hand these are not just web applications as they have access to the native APIs of each device.

React Native Framework

React and React Native both use an architecture that follows a flow methodology to handle the passing of all data in the application called **Flux**. This was created by Facebook as an alternative to the conventional model MVC - Model View Controller. The biggest difference from Flux to MVC is that it uses a one-way model to pass data between the various layers making it easier to identify errors - as depicted in **Figure 2.**

The **Flux architecture** has reduced data passing complexity compared to MVC by not using bidirectional data binding and by requiring all actions to pass through the centralized Dispatcher - as seen with the Action to Dispatcher binding in **Figure 2**. Dispatcher handles all changes from different data sources by updating the Store - application data layer - so the View - apresentation layer - will be updated and redrawn according to the new changes.

React Development - React Native uses a lot of Web development logic, when using HTML logic, the language is Javascript and has a style component identical to CSS.

React Native, unlike React, does not use standard HTML tags, instead it created similar components.

Real-time Communication Technologies

The FleetMobile application has a wide range of features and real-time data on the fleet, such as indicators of its location, speed, current status (off, stopped, in transit, in motion), kilometers traveled, among others.

SignalR

The SignalR library simplifies the process of adding real-time communication functionality to the ASP.NET framework adopted in existing web services.

ASP.NET SignalR is an ASP.NET library that simplifies the process of adding real-time functionality to applications. Real-time features consist in having the content resulting from running code on the server-side available instantly to the client-side, rather than the server waiting for client requests to get new information. This technology automatically selects the best method of transport based on the analysis of the client connection, choosing between **WebSockets**, **Long Pooling** and **Pooling**.

SignalR contains two models of client-server communication - **Persistent connections** and **Hubs**.

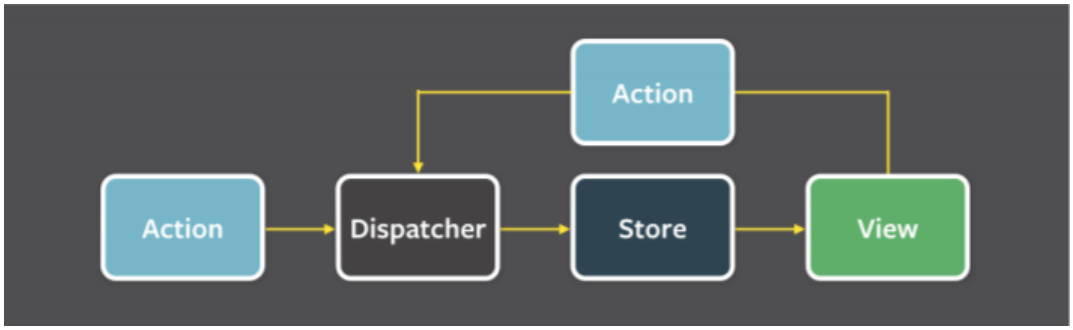

Figure 2 – Information flux in a React app that uses Flux Architecture to build interfaces;

Topics Overview

With its strong compatibility with multiplatform development coupled with freedom in choosing the IDE, Xamarim is one of the biggest alternatives. This tool lets you share approximately 96% of the code across all platforms and features a community of around 1.4 million developers. It is based on the .NET framework and, as already mentioned, C# is a very mature and evolved language which can be used in parallel with many useful development technologies such as LINQ, Lambdas and asynchronous programming.

Unlike conventional web-based hybrid development solutions, a cross-platform application created using Xamarin can still be classified as native. Its performance has been constantly improved from update to update to get as close to native as possible, a little due to the fact that it is open source and has a constantly growing community.

PROPOSED SOLUTION

The solution was partly decided by the Tecmic development team, the most important points being detailed below:

1. This application is developed using Xamarin in Visual Studio and not in Xamarin Studio, and against PhoneGap and Xcode.
2. To design iOS interfaces will be used XCode, which must be imported into Visual Studio, having the native components of XCode and Apple Guidelines in the developed solution, having the best of both worlds.
3. It was decided to use the Mvvm-Cross plugin available in the NuGet component, described in the previous section, therefore using the MVVM design pattern. All in the effort to make code reusable for other applications and to streamline future maintenance and upgrade actions evenly for both target systems of interest (Android and iOS).
4. In terms of Real Time communication technology, it was decided to use SignalR. The communication model option was for Hubs to use, due to the development team's experience and use in most applications, even recommended by Microsoft itself.

This solution has been supplemented over time with more plugins and enhancements.

IMPLEMENTATION

Among the main operating systems for smartphones, the company opted for the mobile application to be developed on the Android platform with use in versions equal to or greater than KitKat 4.4. Meanwhile for the iOS systems the app will be available equal or superior to iPhone 5 and minimum system version 10.3 but keeping in mind that it is recommended to have the latest update.

For the development of the entire solution of FleetMobile application was used Microsoft Visual Studio tool, similarly to the developed Web services was also used Microsoft Visual Studio tool adopting the ASP.NET framework.

Protocols and Communication Techologies

REST (Representational State Transfer) was used for communication. REST is simple to understand and has as one of its key features a strong compatibility with any client or server that supports HTTP / HTTPS, being flexibility the biggest advantage of the REST architecture and the fact that it gives the developer the option to choose the most appropriate format according to the specific needs. The most common formats are JavaScript Object Notation (JSON), eXtensible Markup Language (XML), and plain text, but in theory any format can be used. This leads to another performance advantage, given that Web services can use the JSON format leads to increased performance due to communication package size. That is, they require less bandwidth, which is very useful for mobile communications. However, it is worth highlighting the growth in the use of binary protocols, a subject much debated in the Internet of Things (IoT) theme. [1]

Due to the needs of various users to have constantly updated information about vehicles, messages, trips and alarms, these same real-time features of the application, it was used the ASP.NET framework API, Signal R [5]. This framework was chosen to reduce the number of connections and requests to the server, to help determine how the server handles all requests and to ensure that the information arrives in a timely manner reducing any constraints on real-time communications.

System Infrastructure

This project arose from the need for an iOS version of the Fleetmobile application from Tecmic S.A.. As such, it was naturally integrated into the existing system infrastructure. Following is **Figure 3** which describes this same infrastructure called iZiTraN.

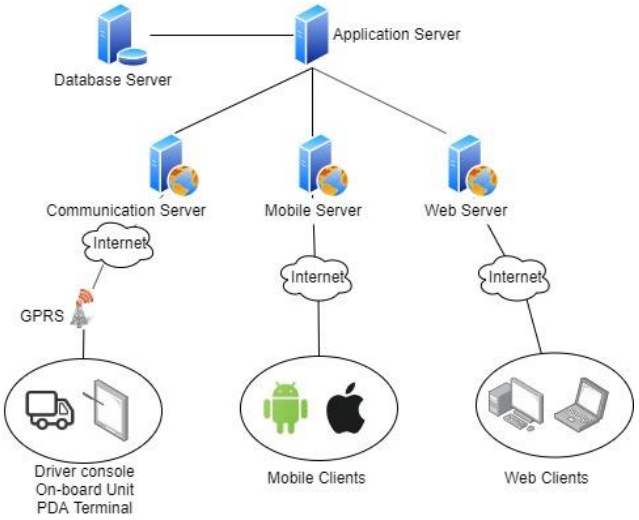

Figure 3 – iZiTraN system infrastructure architecture;

.

The iZiTraN infrastructure consists in the various Tecmic devices and equipment present in the fleet vehicles. In the lower left corner of the figure are represented the Tecmic equipments that allow to deduce and remove all the necessary information for each vehicle. (XTraN and iZiTraN web, among others). Requests from these clients are sent to the Web Server which in turn communicates with the host, Application Server. In the lower center section are the Mobile Clients, which communicate with the Mobile Server.

Mobile Application Architecture

The multiplatform version, in turn, was developed using the Xamarin framework integrated with the Mvvm-Cross framework, in a logic of data modulation and its flow between layers. So the multiplatform FleetMobile application has the following architecture, **Figure 4**.

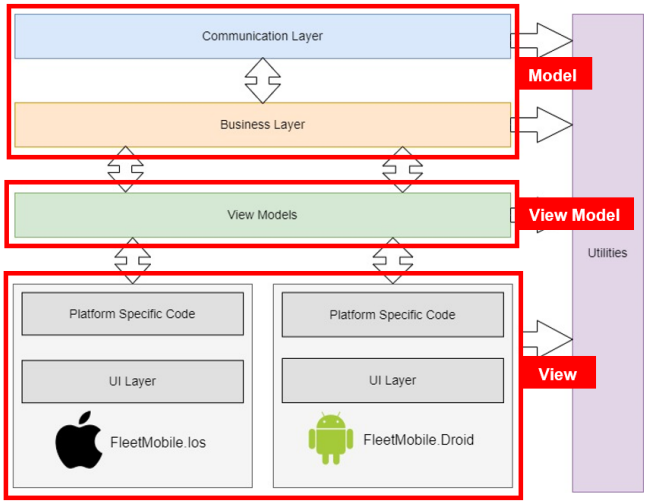

Figure 4 – FleetMobile app architecture using Xamarin e Mvvm-Cross;

Xamarim features an architecture where all code is approximately 96% reusable and common to both Android and iOS platforms and where the business, communications and data access layers are the same, as detailed in **Figure 4** with "Business Layer" and "Communication Layer" layers. On the other hand, the non-common layers are the platform specific "FleetMobile.Droid" and "FleetMobile.Ios", considered as the view layers. In red is represented the integration with the Mvvm-Cross framework.

Interface Guidelines

For the development of interfaces of any application or system it is essential to follow the various usability and design guidelines of the various platforms. For a multiplatform mobile application it is then necessary to validate the different guidelines, in this particular case the guidelines of both Android and iOS.

Generally speaking there are some basic principles that help creating impact and determining the identity of an app, be it Android or iOS. They are - **Aesthetic Integrity**, **Consistency**, **Feedback**, **Metaphors** and **User Control**. Beyond these are three principles that Apple claims that differentiate iOS platforms from all other platforms - **Clarity, Deference and Depth**.

Graphic Interfaces

All Android interfaces were already created and as such were reused. Regarding the interface design for the iOS platform, XCode was used using the XCode Interface Builder. This tool uses the native iOS UIKit framework, which allows applications to get a consistent interface to the total system while also offering a great level of customization. They are adaptable, allowing developers to create a single application ready for any iOS device and

screen, moreover UIKit elements automatically update when the system introduces updates.

Afterwards the interfaces developed in XCode were imported into the Visual Studio project, thus having coherent interfaces that followed Apple's styles and guidelines.

Prior to implementation, some non-functional prototypes were designed to study possible changes between existing Android interfaces and new iOS interfaces, as the iOS and Android guidelines are somewhat different. After their validation, the final interfaces were implemented following the guidelines already mentioned.

TESTS AND ANALYSIS

This section presents the tests that were performed, namely unit tests, integration, performance and usability tests.

Unit Tests

The development of these tests consists in validating the code by inputting and outputting data in a small, isolated code excerpt. These tests were performed throughout each iteration, in order to test each functionality developed by validating whether or not it returns the expected result.

Unit tests, as already mentioned, performed throughout each iteration, were divided into four iterations corresponding to the end of each of the following modules, “Vehicles”, “Travel”, “Alarms” and “Messages”. When each of these modules was closed these tests were then performed. Said tests obtained the expected result.

Integration Tests

Integration tests are software tests used to test integration between multiple modules and layers. In these tests full functionality was tested, specifically, for example, sending a message and receiving the reply, as well as updating the speed and geographical position over time in a vehicle in “moving” state. Using the message example, with this test it was possible to test both the functionality itself and the SignalR framework by testing the reception and notification of the response.

Performance Tests

Performance testing was also quite important, assuming there are customers with fleets of thousands of cars, which means that, for example, getting conversations from these vehicles can be quite a costly operation if not implemented in an exemplary manner. Therefore, tests were made in this perspective, which obtained the results represented in **Figure 5.**

Usability Tests

Throughout the development of this application several usability tests were planned and performed, initially with a group of employees of a Tecmic S.A .client (10 people). In a second phase to more advanced users, a selected group of company employees (15 people), as they have a strong knowledge about the business model where the application fits. Several task guides were then written for them to perform.

These tests were partially in vain because the entire audience of this application already had a huge experience on the native Android version of this application, whereby the results were 100% successful.

Results Discussion

With unit tests it was possible to validate small and isolated code excerpts of the various functionalities aiming to ensure the cohesion and reliability of their code. The system passed accurately in all tests.

Regarding the integration tests, with sending and receiving messages, it was possible to verify that the integration between all layers - layers that concern to the integration between Mvvm-Cross, Xamarin and the ASP.Net framework - was successful. It was also possible to test the integration with the SignalR framework in the reception of the message.

Based on the results of the load tests, **Figure 5**, it was possible to see that in most list screens - message list, travel list or even the alarm list - when the number of fleet vehicles scales to hundreds or even for the thousands it had excessive loading times for fluid use. As such, data access layers were optimized to bring only fields that are visible in the interface, thus presenting practically instantaneous load values.

	Número de veiculos						
	10	20	50	100	500	1000	10000
Renderizar Veículos no Mapa	≈1seg	≈1seg	≈1seg	≈2seg	≈5seg	≈10seg	≈22seg
Carregar Lista Viagens	≈1seg	≈1seg	≈3seg	≈12seg	≈30seg	≈62seg	≈170seg
Carregar Lista Alarmes	≈1seg	≈1seg	≈1seg	≈2seg	≈5seg	≈16seg	≈93seg
Carregar Lista Mensagens	≈1seg	≈5seg	≈15seg	≈24seg	≈41seg	≈93seg	≈218seg

Figure 5 – Performance tests;

It is also important to note that in addition to these tests, at the time of application submission in the various stores of each platform (Apple - AppleStore, Android - Play Store), especially in the Apple Store, the app was subjected to several intensive tests by Apple, for at least two days, passing with distinction in all and as such is now available on the Apple Store.

CONCLUSION

This project addressed the need for Tecmic S.A. customers to have a version of the FleetMobile mobile app for their iOS devices. For the development of said app were detailed requirements to be studied and met starting with the best tools for multiplatform development, study of the use of iOS guidelines to interfaces already developed on Android, which is the best technology for real-time communication and what is the better design pattern and techniques for making code reusable and facilitating future maintenance and upgrade actions evenly for both target systems of interest (Android and iOS).

As presented in this paper, we studied technologies, frameworks and design patterns that would allow the development of a multiplatform application. It was then chosen, in the development phase, the Xamarin and the Mvvm-Cross framework for multiplatform development with the aid of the SignalR framework for real-time application functionality. A scalable architecture that meets customer needs was achieved, as well as the enabling of future additions to the customer's business process. This architecture has achieved the intended goals of having a system with reusable code that facilitates future maintenance and update actions uniformly for both platforms. Based on the native android version of this app and apart from the designed architecture, it was also possible to have an iOS interface that followed the iOS guidelines without major changes.

This project resulted in the publication and availability of the FleetMobile application for iOS systems, which is the main requirement of this project, fulfilling simultaneously the need for the company's customers to have an iOS version of the application and all the objectives detailed by the company before the development of the same. However, the system presents potential for evolution and need for future work.

Future Work

As mentioned in the previous section, this project has great potential for evolution, however there is also a need for future work. That’s what this last section will focus on with the following enumerated, keeping in mind that the first points still come from the first native version. [1]:

- The addition of more modules, such as energy efficiency, driving times and driver scoring, using the Gamification concept [1];
- The addition of voice communications to fleet vehicles through the mobile device voice API. This addition was not considered a priority because many vehicles do not yet have voice communication via the Tecmic drivers console [1];
- Identifying the actions that the application can execute offline in order to implement a database in the mobile application. This point was not considered a priority since the main purpose of the application is to provide real-time information such as locations, alarms, messages, among others [1];

- Improve some details of the application interface design;
- Finalize the development of the Android version on this multiplatform system and consequently run all necessary tests and analysis. This point was not a priority because the main requirement was the availability of the iOS version;
- Make the Android version available on the Android app store - Play Store.

REFERENCES

1. João Miguel Henriques Domingos – Fleet Mobile Management – Relatório de Estágio – 2015. [accessed Oct 7 2019]
2. Triet Le, iOS Development with Swift programming language, Available from https://www.theseus.fi/bitstream/handle/10024/124255/Triet-Le-Thesis-Final-05042017.pdf?sequence=1 [accessed Oct 7 2019]
3. Mukesh Prajapati & Dhananjay Phadake & Archit Poddar, STUDY ON XAMARIN CROSS-PLATFORM FRAMEWORK, https://www.academia.edu/37016885/Study_on_Cross_-Platform_Mobile_App_Development_With_Xamarin [accessed Jul 26 2019]
4. Craig Dunn & Brad Umbaugh, Sharing code overview, https://docs.microsoft.com/en-us/xamarin/cross-platform/app-fundamentals/code-sharing . [accessed Aug 20 2019]
5. Techbrij, SignalR, https://techbrij.com/database-change-notifications-asp-net-signalr-sqldependency . [accessed Sep 06 2019]