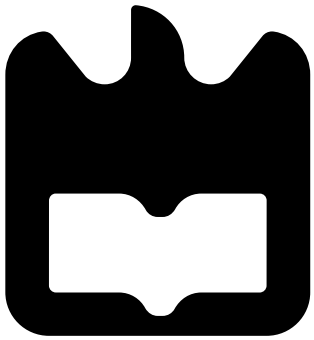
Universidade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática,
2016

Tiago Manuel Carvalho Henriques

Aplicação móvel para enriquecer a experiência da exposição "70 Cavaquinhos, 70 Artistas"

Extending the "70 Cavaquinhos, 70 Artistas" exhibition with mobile applications



Tiago Manuel Carvalho Henriques

Aplicação móvel para enriquecer a experiência da exposição "70 Cavaquinhos, 70 Artistas"

Extending the "70 Cavaquinhos, 70 Artistas" exhibition with mobile applications

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Professor Doutor Ilídio Castro Oliveira, Professor Auxiliar do Departamento de Eletrónica e Telecomunicações da Universidade de Aveiro

o júri / the jury

presidente / president

Professor Doutor Joaquim João Estrela Ribeiro Silvestre Madeira
Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

vogais / examiners committee

Professor Doutor Ilídio Fernando de Castro Oliveira
Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

Professor Doutor Telmo Eduardo Miranda Castelão da Silva
Professor Auxiliar do Departamento de Comunicação e Arte da Universidade de Aveiro

agradecimentos / acknowledgements

Gostaria de agradecer em primeiro lugar ao Professor Ilídio Oliveira, orientador deste projecto, por toda a disponibilidade e apoio prestado, pelos conselhos e orientação fornecida.

Agradecer à Associação Museu Cavaquinho pela oportunidade, e em especial ao impulsionador deste projeto, Júlio Pereira, pela disponibilidade demonstrada e contributo fornecido ao longo de toda a realização deste. Também ao Marcelo Baptista e João Cruz pelas indicações fornecidas na parte de design.

Por fim agradecer à minha família, em especial à minha irmã, e amigos por todo o incansável apoio.

Resumo

As aplicações móveis têm cada vez mais impacto e relevância no quotidiano, acrescentando valor tanto a nível profissional como a nível pessoal. A grande variedade de aplicações existente fornece apoio nas mais diversas áreas, desde entretenimento, prática de exercício, melhoria de produtividade, valorização pessoal, entre muitas outras.

As exposições são espaços dedicados à expressão artística, nas mais variadas formas. O apoio às exposições tem sido concretizado na sua maioria através de panfletos ou mensagens escritas e, mais recentemente, também com aplicações móveis.

Neste contexto, o objetivo deste projeto consiste no desenvolvimento de uma aplicação móvel capaz de fornecer suporte à exposição "70 Cavaquinhos, 70 Artistas", numa forma complementar à exposição. Em particular, procura-se uma aplicação capaz de fomentar o interesse do público para esta exposição em específico, sem, no entanto, deixar de considerar a vantagem da aplicação ser construída de forma a ser adaptada para outras exposições.

Procurou-se explorar a natureza interactiva dos dispositivos móveis com características como a integração de um sistema de leitura de identificadores ou a inclusão de um jogo relacionado com a exposição.

Foram desenvolvidas duas aplicações para duas plataformas diferentes, em Java para Android e em Swift para iOS, mantendo, no entanto, o conteúdo semelhante. Como seria de esperar, foram realizadas adaptações de funcionalidades e características especificamente para cada sistema operativo móvel. Foi, ainda, desenvolvida uma base de dados comum às duas aplicações - Firebase. A aplicação em Android pode ser já encontrada na Play Store, enquanto que a aplicação em iOS se encontra pronta a ser publicada.

As duas aplicações desenvolvidas cumprem os critérios inicialmente estabelecidos, fornecendo informação adicional e relevante para uma boa experiência, não só aquando da visita à exposição, mas também antes e depois da mesma. Adicionalmente, estas aplicações podem ser adaptadas facilmente a outras exposições, mudando apenas os dados inseridos no *back-end*.

Abstract

Mobile applications have been gaining impact and relevance in people's quotidian, adding value not only in a professional level but also in a personal one. The wide range of mobile applications provides support in distinctive areas, from entertainment, to exercise, work, personal achievement, among others.

Exhibitions focus on the artistic expression, in the most diverse ways. The support to the exhibitions has been achieved mainly through flyers or written messages, however more recently also with mobile applications.

In this context, the aim of this project concerns the development of a mobile application able to provide support to the "70 Cavaquinhos, 70 Artistas" exhibition, in a complementary way. In particular, the application should be able to foment the public's interest on this specific exhibition, considering, however, the advantage of building an application easily adapted to other exhibitions.

The interactive nature of the application was explored through the integration of specific features, as a reading system or an exhibition-related game.

Two applications were developed for two different platforms, in Java for Android and in Swift for iOS, maintaining, however, a similar content. As expected, some adjustments of functionalities and specific features for each operating system had to be implemented. A common backend for both applications was also developed - Firebase. The Android application can be already found on Play Store, while the iOS application is ready to be published.

The two developed applications fulfil the initial established criteria, providing additional and relevant information about the exhibition, thus offering a better experience not only during the exhibition itself but also before and after it. Additionally, these applications can be easily adapted to other exhibitions, just by modifying the data inserted in the backend database.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The continuous improvement of mobile devices into more powerful and small equipments, combining portability and design, has led to the development of rich smartphones [1]. They allow people to do several and diverse tasks at the same time, often removing the need of a computer. The increased development of applications (apps) has been trying to fulfil needs, from the smaller and simpler tasks to the most complex. As expected, and with the wide range of apps and functionalities available, people became more dependent on technology and on what it can bring, with this dependency increasing year after year.[2]

Applications have been one of the more explored market niche in the last years, representing millions of dollars[3]. There are thousands of applications, in all fields and for all needs. Any idea can be taken and developed into something: applications have been developed to ease communications, to improve productivity, to entertain people or to foment exercise practice; essentially to allow users to explore their interests and hobbies.

1.1 Collaboration between the Associação Museu Cavaquinho and the University of Aveiro

The Associação Museu Cavaquinho[1] is an association that aims at spreading the history of *cavaquinho* while fomenting its practice, systematically collecting and organising related information. The University of Aveiro has been collaborating with this association, helping in *cavaquinho*'s promotion and related areas, including design and technological aspects.

The Association already has an online platform where anyone interested can get information about *cavaquinho*. Nevertheless, a wider presence in other platforms as Android or iOS is still lacking, being this the main reason behind the development of this project.

Integrated in its mission, the Association promotes activities to increase the awareness of the population for the rich heritage of *cavaquinho* in the Portuguese culture. A good example is the exhibition "70 Cavaquinhos, 70 Artistas", with contribution from several recognised artists.

[1] http://www.cavaquinho.pt

This exhibition entitled "70 Cavaquinhos, 70 Artistas" comprises the work of 70 artists that express their art by any means in a *cavaquinho*. Instruments are then displayed carefully around the room, in order to give visitors the best experience. A book has also been published containing all different pieces in detail.

Firstly inaugurated in Mosteiro dos Jerónimos by the Collective and Travelling National Exhibition, on November 27th 2014, this exhibition has travelled since through several Portuguese cities (Theatro-Circo - Braga, Sala Cidade - Coimbra, amongst others) counting with more than 40000 visitors [4].

Figure 1.1: Exhibition in Braga - Theatro-Circo, 2015 and in Viana do Castelo - Museu do Traje, 2015.
(source: http://www.cavaquinhos.pt/en/Exhibition-Web.htm)

1.2 Objectives

The existing exhibition follows the classic gallery-style format, in which the works are displayed in a room, according to the curated setup.

The aim of this work consists on the development of a mobile application to serve as complement for the "70 Cavaquinhos, 70 Artistas" physical exhibition, allowing the users to visit a digital version.

The application should:

- allow the user to browse a digital gallery with the same works seen in the exhibition, during, before or after an actual visit, thus better complementing the exhibition.
- support both Android and iOS platforms.

A backend support should also be developed to allow updating and creating new content for the exhibition, without changing any application code.

1.3 Dissertation Structure

Chapter 2 focuses on related work, introducing the mobile developed systems and technologies. We also review other works that have been done in the field of mobile-enabled exhibitions support.

Chapter 3 focuses on the requirements that need to be fulfilled in order to develop a successful mobile application. Both general and specific requirements are presented focusing on the ones relevant for an application in this area. The system architecture is explored in the Chapter 4, detailing the overall architecture both in Android and iOS.

Chapter 5 focuses on the implementation, describing the work performed for the Android and iOS applications and the backend services that support both apps.

Chapter 6 presents the accomplished results, displaying the tests that were performed in order to improve the final result, both in terms of compatibility and usability.

The conclusion discusses the results, presenting some insights about future work to be developed in the scope of this project.

Chapter 2

State of the Art

2.1 Mobile Applications Development

Mobile devices appeared in people's life with a big impact[5]. Firstly developed, computers came with several benefits and numerous possibilities. The need of mobility, however, expanded the research in this field, leading to the development of mobile computers, considered a huge step forward in the technological world. As the needs of the users increased, so did the evolution of technology: smaller devices, as smartphones, were developed in order to allow people to perform some tasks without the need of an actual computer[6].

When mobile phones first appeared in 1973 [7], they were developed with the only goal of making calls. Afterwards, with the advance of technology, some features were progressively added, meeting the needs of the users. Sending written messages became a reality in 1992 [8], while using the phone for entertainment became possible in 1994 [9], allowing users to play some limited games. Many other features have been introduced in order to fulfil the needs of a growing market, as improved screens and incorporated cameras, even attempting to replace the need for other devices.

Still, the huge step only came in 2007, when the first smartphone was successfully released [10], and touchscreen became more used. Since then, a whole new world of possibilities has been explored, allowing the development of one of the key concepts today in this mobile world: mobile applications.

With these changes, the mobile operating system became a key concept on the phone. In fact, more important than good hardware is a software that can explore the full potential of the existing hardware[1]. Therefore, it was expected that the large technological companies would start developing their own operating systems (OS). Apple released iOS in 2007 [10], only to be used by their products, while Google launched Android 1.0 in 2008 [11], allowing its use in several other companies' devices. Microsoft replaced Windows Mobile for the Windows Phone in 2010, and then again later the Windows Mobile for the Windows 10 Mobile in 2014 [12]. Several other companies have done the same, attempting to get ahead of the competition in this area[5](Table 2.1).

Mobile OS	**Company**	**Current Version**
iOS	Apple Inc.	9.3.2
Android OS	Open Hanset Alliance	6.0.1
Windows Phone	Microsoft	Windows Mobile 10
BlackBerry 10	BlackBerry Ltd.	10.3.2.2876
Firefox OS	Mozilla Foundation	2.2.0

Table 2.1: Short description of the main mobile OS currently on the market.

Apart from the smartphone, the development of new gadgets has been a reality. Although the idea is not recent, only in 2010 the first tablet was successfully released by Apple [13]. These devices' innovation arise as a compromise between laptops and smartphones, both regarding size and performance. As a result, new applications and new operating systems have been explored.

Up to now, two main competitors have been driving the world of mobile application forward: Apple with iOS and Google with Android, the latter providing the OS for almost any device on the market. Together they form 98,4% of all the market share sales [14][15](Figure 2.1): Google dominates with 80,7%, mostly due to the giant number of manufacturers using Android as their main OS, while Apple gets the remaining 17,7%, a positive number when considering its use only by Apple products. Microsoft is the following competitor with Windows Phone, however with a very small share of 1,1%, which is not nearly enough to attract developers to this OS [8]. Any other OS has not been able to stand in this competitive market.

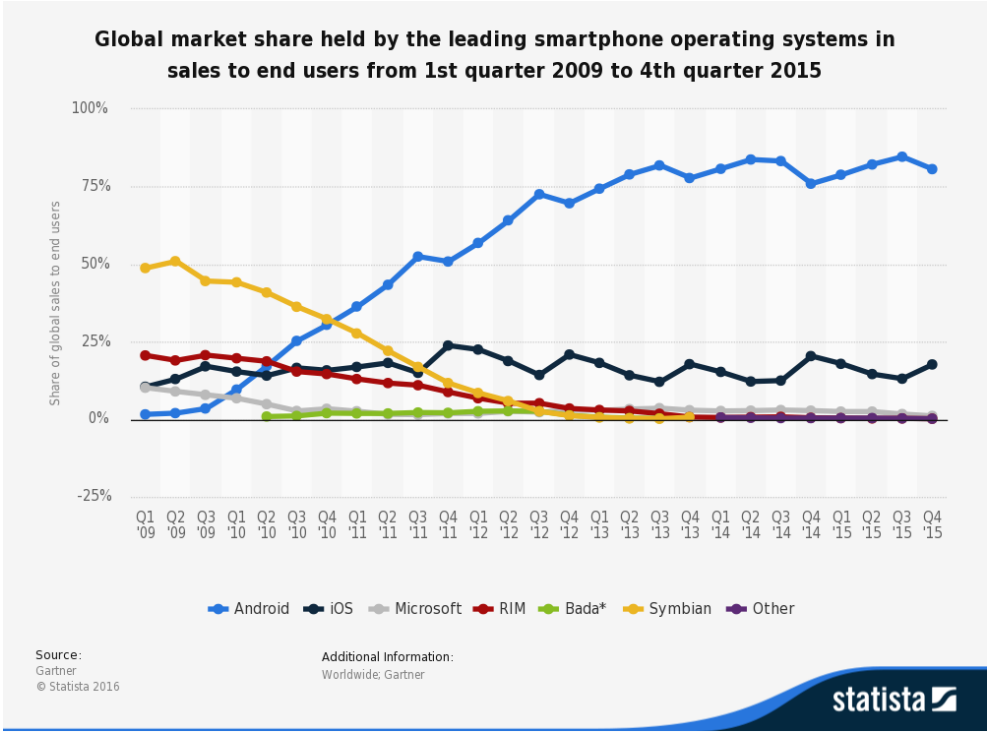


Figure 2.1: Global market share held by the leading operating systems in sales to end users, using information from a Gartner report
. (source: http://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/)

Based on the data from Figure 2.1, iOS and Android are the obvious platforms when developing an application, allowing more than 90% of the people to have access to the developed resource.

The fast development of mobile communications and associated features led to an expansion of the market in order to satisfy the increasing needs of the costumers. Applications started to be not only developed and sold by the large companies but also by individual developers. Apple opened the App Store for this purpose as well as Google, firstly called Google Play and now Play Store.

The continuous development of Android translates into a high number of applications developed in this OS, with more apps on Google Play than on the App Store [16](Figure 2.2). Together they surpass the billion and a half applications [17]. This difference regarding published applications can be partially explained considering the submission process that all iOS applications have to pass when comparing to Android platforms, where everything can be published with little revision.

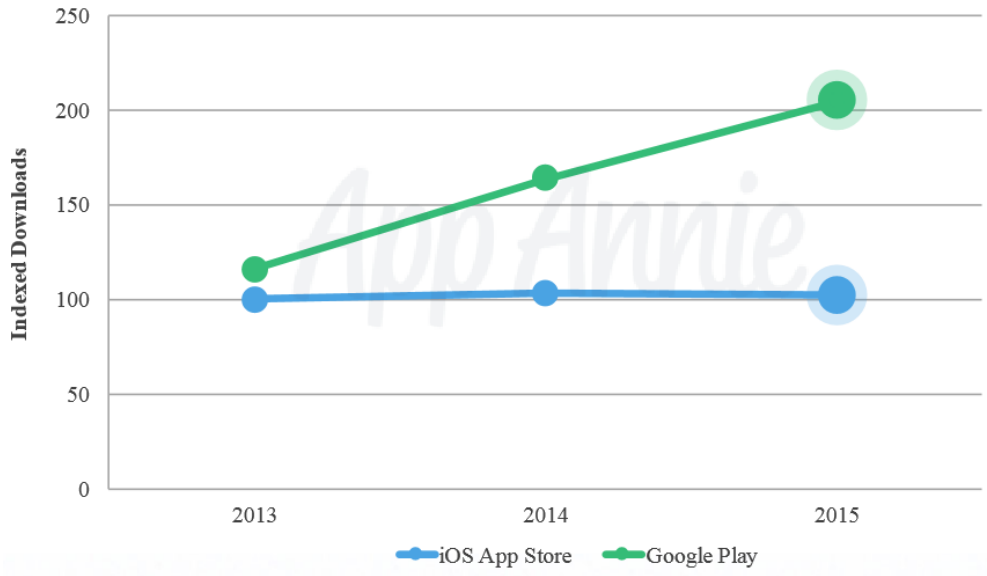


Figure 2.2: Indexed downloads from Android and iOS
(source: http://go.appannie.com/report-app-annie-2015-retrospective).

2.1.1 iOS platform

Developed by Apple in 2007 and originally called iPhone OS, this OS was initially designed only to suit the iPhone. Afterwards, it was extended to support other Apple devices such as the iPad or iPod Touch. Based on direct manipulation and multi-touch, this framework allows the production of different applications by taking advantage of the device resources, as the camera, accelerometer or GPS.

iOS has always been associated, as every Apple device, to a clean design. As a result, the user is offered a simple and intuitive interface, in which every detail counts. Unlike other mobile operating system, iOS does not have a menu: instead it presents the user all its applications divided in screens, accessible through swipes. The spotlight was also introduced in iPhone OS 3.0, allowing users to search media, apps, emails, messages and similar content. A notification center with two views (Today and Notification) is available as well as a control center, where several aspects of the OS can be configured (Figure 2.3) such as the Internet connection state or the bluetooth connection.

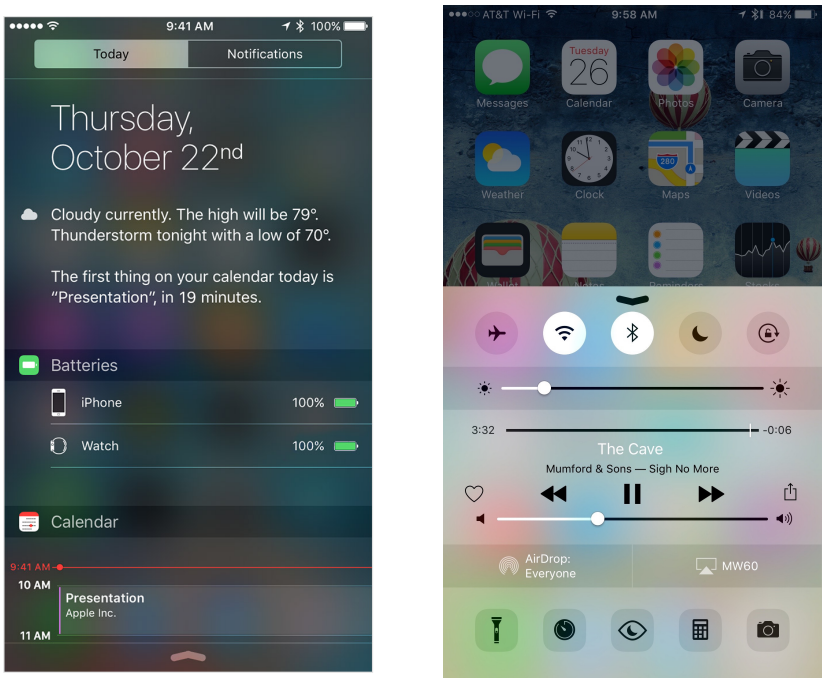


Figure 2.3: iOS notification center and control center.

With the advance of technology, several other features have been introduced: the concept of folder became a reality in iOS 4.0 as well as multitasking, which brought an enormous advance in how users interact with the device. Another remarkable feature of Apple devices is Siri (Speech Interpretation and Recognition Interface), a software program that works as an intelligent personal assistant. Introduced in Apple devices since the iPhone 4S, this program has been continuously improved in order to provide users a more accurate interaction.

Currently, iOS runs in iPhone 4S, iPad 2, iPad Pro and any later releases as well as in all models of the iPad Mini and 5th-generation iPod touch and later. When it was released, the chosen development language was Objective-C[18], an Object Oriented Language already used by Apple for other frameworks. More recently, in 2014, Apple released a new language to be used for programming their devices: Swift[19]. This is also an Object Oriented Language, however with some improvements and a lighter syntax. At this point both development languages are accepted, and therefore, the developer may choose the most suitable one for his purpose.

Although iOS shares the Darwin foundation with OS X, it is not fully Unix-compatible. It shares some frameworks, such as Core Foundation and Foundation Kit, however iOS uses the Cocoa Touch rather than the Cocoa [20]. The released SDK only runs in OS X, being this a restrictive factor when developing applications, since not everyone has the suitable hardware. Xcode[21] is provided freely by Apple as the development software, although others can be used, namely AppCode[22].

Regarding the internal architecture, iOS acts as an intermediary between the underlying hardware and the applications created. These applications do not access directly to the underlying hardware, but rather through well-defined system interfaces, making it easy to write applications independently of the device hardware. These layers are the Cocoa Touch, Media, Core Services and Core OS, as shown in Figure 2.4.

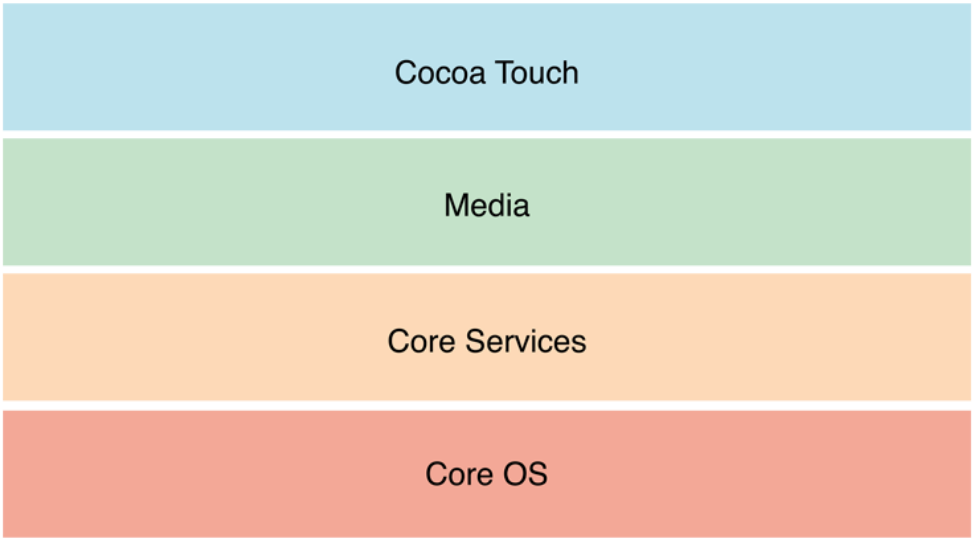


Figure 2.4: Internal layers of the iOS system.

Specifically, the Cocoa Touch layer provides the basic infrastructure and support for key technologies, such as multitasking, touch-based input or push notifications. The Media layer contains the graphics, audio and video technologies used to implement multimedia experiences and build applications. Some of the referred technologies include the UIKit graphics, the Core Graphics framework or the Core Animation. The following layer, the Core Services layer, contains fundamental system services for applications. Two of the most important services are the Core Foundation and Foundation Framework, which are the basic types that all applications use. This layer also contains other individual technologies to support features, such as location, iCloud or social media. Lastly, the Core layer contains the low-level features that most technologies are built upon. These technologies are implemented recurrently, including when dealing with security or directly when communicating with external hardware. It contains several important frameworks, as the Accelerate Framework, the Local Authentication Framework or the Security Framework.

Similarly to other operating systems, iOS updates are important to keep up with the recent developments in this field, with iOS already in a 9.3 version (as of May, 2016). The continuous release of updates allows Apple to implement new discoveries and features, increasing the user coverage and satisfaction. In fact, more than 96% of the users have a device with one of the last two released versions, with 84% being already in the last version [23](Figure 2.5).

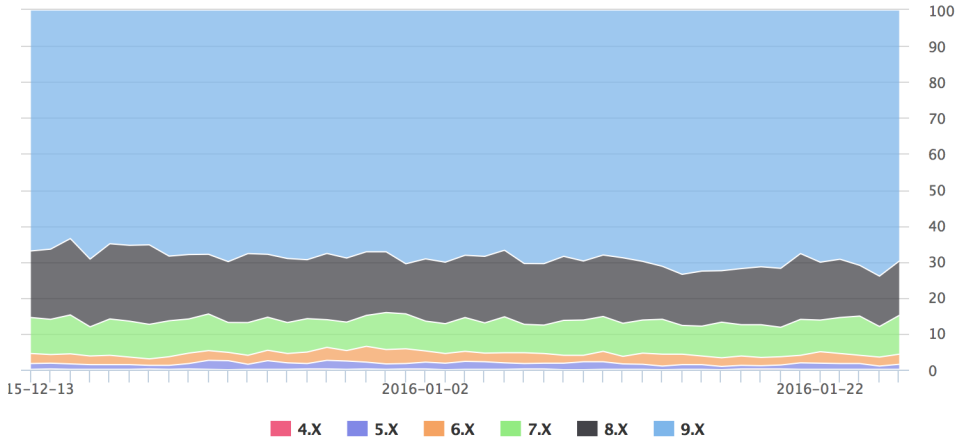


Figure 2.5: Distribution of users for all active iOS versions (percentage) (source: https://david-smith.org/iosversionstats/).

2.1.2 Android platform

Initially released in 2008, Android is nowadays the dominant mobile operating system on the market. Acquired by Google in 2005, this operating system uses multi-touch and direct manipulation, in order to get the full potential of the different types of manufacturers devices. Being behind iOS shortly after its released, Android overcame Apple's operating system in 2013, remaining ahead up to now.

Android is an open-source operating system, which gives substantial advantages when comparing to other closed ones: the comprehensive knowledge of the developers community about the system allows them to share more information and solve problems easier.

Android uses Java, an Object Oriented Language, as its main programming language in conjunction with XML (Extensible Markup Language) to define layouts. The easiness of these languages and syntax are advantages that make it easy to learn and develop an application for this OS.

Android's layout can vary depending on the devices: it is usually associated with a menu, allowing the users to have an environment space to put some widgets and their favorite applications (Figure 2.6). This is, in fact, the biggest advantage of this OS: the customisation. A search bar is also provided to search for files or applications, as well as a notification center, where the notifications are presented and where the user can configure some aspects of the device.

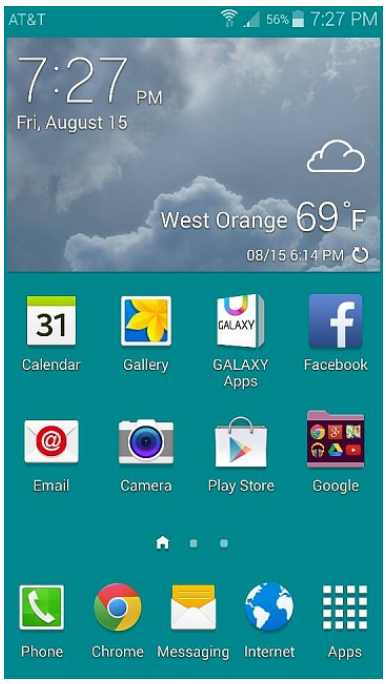


Figure 2.6: Example of Android home screen.

Regarding the specific Android architecture (Figure 2.7), it is divided in four layers: Applications, Application Framework, Libraries and Linux Kernel[24].

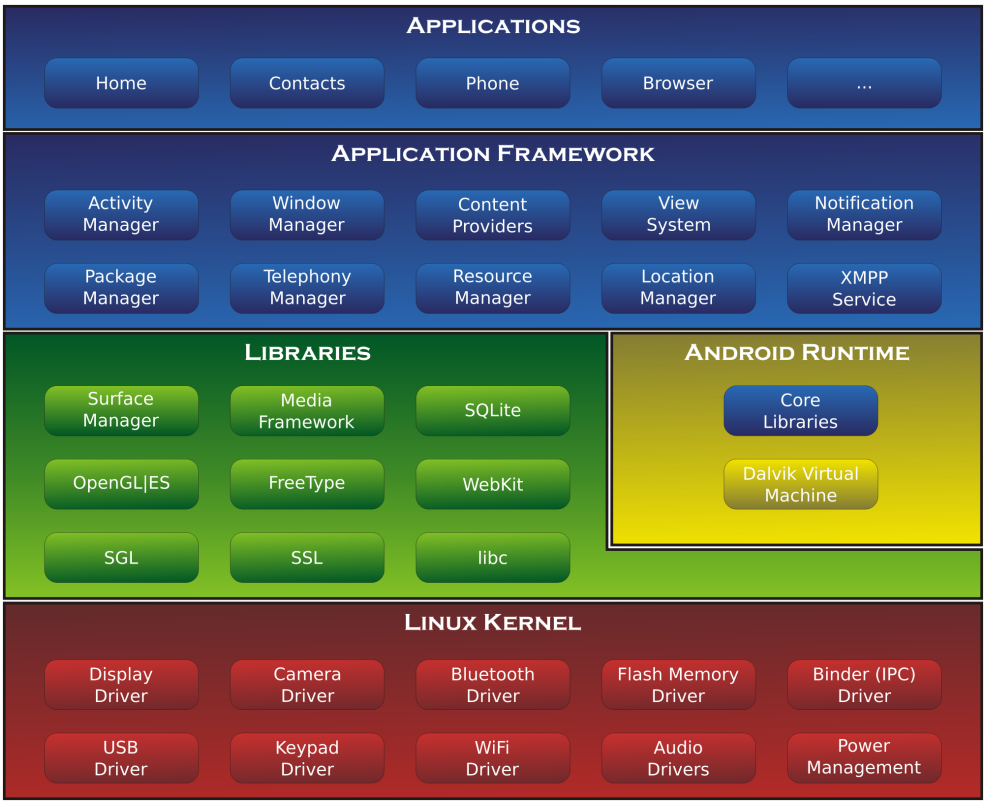


Figure 2.7: Internal Android architecture
(source: http://www.zdnet.com/article/how-android-works-the-big-picture/).

The top layer is the Applications layer, where applications are installed. Some of the pre-existing applications include the Contacts Books, Browser or Games. The Application Framework layer uses libraries from the Libraries layer in order to provide several higher-level services to applications in the form of Java classes, as the Activity Manager or the Notification Manager. As referred, the following layer, Libraries, comprises the libraries used by the Application Framework, such as the libc, SSL or SQLite. It also includes some Java-based libraries that are specific for the Android development, as android.app, android.database, among others. The bottom layer of this architecture is the Linux Kernel. This layer is responsible for creating a level of abstraction to the device hardware and it contains all the essential hardware drivers as the Camera Driver, the Display Driver or the Audio Drivers. This layer also handles network related aspects as well as other tasks related to drivers.

Another important section on the architecture is the Android Runtime. This is available on the Libraries layer and provides a key component called Dalvik Virtual Machine[24], which is a Java Virtual Machine designed and optimised for Android. This Dalvik VM enables every Android application to run on its own process, making use of Linux core features. It also provides a set of core libraries which enable Android application developers to write applications using the standard Java programming language.

Unlike iOS, Android is not associated with a specific device, and therefore a hardware company can easily get the Google's released version and adapt to make its own, giving rise to different versions across devices in order to take specific advantages of their resources, although the operating system is the same[25].

Given the high number of devices running this OS, some manufacturers begin to develop their own versions of Android, leading to a dispersion in terms of updates and commercially available versions [26]. As of June 6, 2016, only 10.1% of the users possess the last version of Android (Figure 2.8). In fact, most of the users remain using the 4.4 version (31.6%), released in 2013. To achieve the 95% users, an application has to be developed for the Jelly Bean version. As a consequence, developers are not able to extract the full potential of updates released by Google, since they still have to develop applications for old versions in order to assure compatibility. This aspect is frequently solved by the community, which develop compatibility libraries, allowing some of the more recent features to be compatible with older versions.

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	2.0%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	1.9%
4.1.x	Jelly Bean	16	6.8%
4.2.x		17	9.4%
4.3		18	2.7%
4.4	KitKat	19	31.6%
5.0	Lollipop	21	15.4%
5.1		22	20.0%
6.0	Marshmallow	23	10.1%

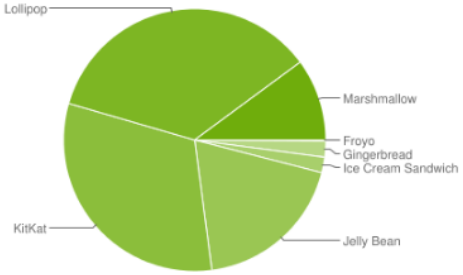


Figure 2.8: Distribution of Android devices by software version as of June 6, 2016 (source: https://developer.android.com/about/dashboards/index.html).

2.1.3 Native, Web and Hybrid approaches for mobile applications

There are three main architecture alternatives for mobile applications: native, web based or a combination of the two, named hybrid. The definition of the architecture requires several factors to be considered, including the quality aimed for the final product, the complexity of the features that the application needs to be able to support or the available budget, among others.

Native Applications

A native application needs to be developed in accordance to the OS in which it will be implemented. Therefore, although the final product is the same, there is the need of developing different applications, one for each target OS. As a consequence, the time and resources needed to develop the applications increase, since there is an extra need of knowledge of two distinct programming languages, Java and Objective-C or Swift. On the other hand, the developer will have more control over the application, being able to develop more complex features. If the budget is not an issue, this method should be selected to develop the application.

Web Based Applications

A web application is written in web code, using the same language as those applications - namely HTML, CSS, Javascript and others, depending on the used platform - but with increased interactivity. This application will load in a browser, like any other website, and therefore there is no need to install it. With this method, a single application can be developed and maintained since it can be used in all devices. This is the most compelling advantage of this method over the native one, since it leads to reduced work and consequently a lower budget associated.

This method, however, has limited resources and reduced features can be explored, since an internet connection is required to access most of them. Moreover, distribution of these applications through app stores as App store and Play Store is hampered. As indicated by Figure 2.9 via a humor strip, if the application is simple enough there is no need to increase the complexity of the system, as this is a perfectly valid and simpler alternative.

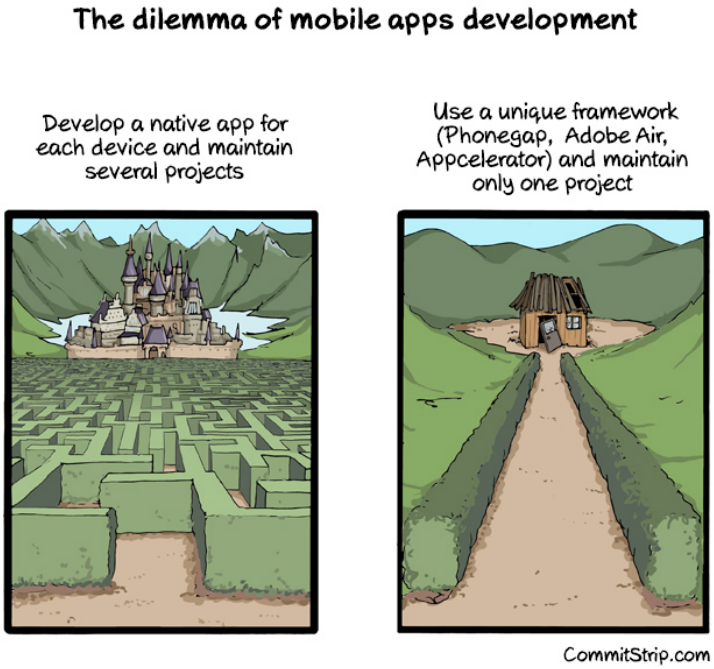


Figure 2.9: The dilemma of mobile applications development (source: http://www.commitstrip.com/en/2014/08/18/the-dilemna-of-mobile-apps-development/).

Hybrid Applications

A hybrid application consists in a web application that is then wrapped inside a thin native container that provides access to native platform features, allowing the application to access more features of the device than a web one. This means that most of the code is similar in both platforms, being then translated to the native language. As a result, most of the advantages of the web way are still there, but with a more refined user experience.

However, it can never reach the experience of a native application, as it is still slower and dependable on the browser speed. It requires a lower budget, like the web applications, but has a higher cost, since it takes substantial work to run this type of application in different platforms. In some cases, it could even require a higher budget than developing native applications.

2.2 Mobile applications to extend the exhibition experience

An exhibition allows artists to expose their work, any form of art, to other people. Apart from the exposed art pieces, contextualization of the exhibition as well as other related aspects, are also important issues to be considered[27].

A well-developed application in terms of functionality may not be enough to provide the user a good experience. Acknowledgement of the users' expectations and preferences is also relevant in the design of an application able to establish a virtual connection between the visitor and the exhibition itself[28]. The application should then follow a more visitor-focused approach[29][30], considering the interaction with the cultural resource[27][31] as well as with other visitors[27]. Additionally, a well-accepted museum/exhibition application should stimulate the users' interest in knowing more about the exhibition[32].

Mobile applications have the potential to personalize the user's museum/exhibition experience according to their preferences, extending the museum/exhibition offer to a much broader range of use-case scenarios[28]. Some studies reveal that visitors enjoy using interactive exhibits[27], with managers from the most important museums around the world believing in the importance of technology in the exhibitions[27]. According to the Museums Association Mobile Survey, from 2013, 50% of the surveyed museums in the UK had already mobile experiences and other 19% were planning on developing some[27]. The tendency is to all of them end up developing their own museums apps.

Technologies can encourage people to go to museums. The possibility of accessing the gallery objects before the visit stimulates the visit[33] as well as creating a connection that remains after the trip will increase the chances of repeating the tour[28]. As such, these are two important issues to be considered in each application. Virtual exhibitions also allow people with mobility issues or time restrictions to still be able to see the exhibition [34][35]. The evolution of new information technology is also believed to create new opportunities to promote and value the cultural heritage[35].

The actual technological state of the museums and exhibitions reveals two main elements that are used to transmit information to the users: the Quick Response (QR) code and mobile applications[27][36].

QR codes are used in many museums and exhibitions and can be used to trigger an action on a mobile device that can lead the user to multiple sources[27] (Table 2.1).

Text	Give the opportunity to creators of each exhibitions to write everything they believe the visitors should know
Games	In quizzes and treasure hunts, leading or not to prizes.
Interaction with and between visitors	Comments at the end of the exhibition, allowing the museums to have feedback and that way improve the service provided.
Research	Coded links to important pages to provide more information.
Website promotion	QR codes can drive more traffic to the museum's website.
Promotion of the gift shop	QR codes can be used to link elements in the exhibition to items in the gift shop
Video	QR codes can be used to provide the user with videos of reconstructions of the object.
Choosing a path	Users can use QR codes to choose their own path through the exhibition.
Augmented reality	QR codes can be used to provide sensory input as sights, sounds and concepts.

Table 2.2: Uses of the QR codes in exhibitions.

Mobile applications can be used for many purposes[27] (Table 2.4).

Location awareness	The application can determine the visitor location and create custom tours.
Gaming	The gaming model could be highly effective because produces behaviors that are otherwise rare in conventional museum-going experience.
Crowdsourcing	Allow the users the change to provide their own content to a museum exhibit
Polling	Could be used to start conversation among visitors and encourage them to think critically.

Table 2.3: Uses of mobile apps in exhibitions.

Although this dissertation focuses on mobile applications, actions triggered by a QR code reading can also be integrated in the application. In order to develop a successful mobile application, the purposes explained on the Table 2.4 need to be filtered taking into account the user's preferences. The Table 2.5 shows some of the requirements that users consider important in such app[28].

Museum	1. How to get to the museum. 2. Description of the museum structure 3. Time of opening and closing 4. Tickets booking and/or purchasing 5. Additional services 6. Contacts 7. Timetable of planned exhibitions 8. Real-time news on new products
Artworks	1. Museum artworks photo-gallery. 2. Audio-guides with detailed information about each artwork 3. Explanatory files of the artworksg 4. 3D virtual tour 5. Virtual tour bookmarks
Map	1. Museum map. 2. Suggestions for guided tours
Accessibility and Usability	1. Multilingual option. 2. Available off-line 3. Friendly and efficient user interface 4. Free download 5. Accessible on any device (iOS, Android, and so on)

Table 2.4: Requirements identified by users of a mobile app for museums.

From these requirements, people gave particular relevance to: how to get to the museum, contacts, museum artworks photo-gallery, multilingual option and having a friendly and efficient interface[28].

One of the available applications is Juliet - the Mobile Educational Virtual Exhibition[37]. This application allows its users to navigate through a gallery of objects, see each of them individually, use a QR Code Scanner to read a code that gets the user to the specific object and also consult some information about the exhibition (Figure 2.10).

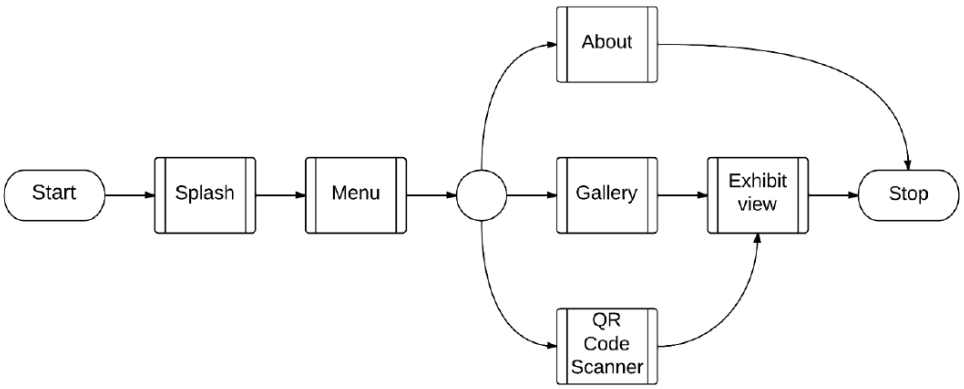


Figure 2.10: Flow of activities in the Juliet mobile application.

When searching in the App Store and Play Store for exhibitions, an enormous number of applications showed up in the results, from museums to temporary exhibitions or movies. One of the most complete was the MoMA app [1]. This is an application that serves as support for exhibitions in the Museum of Modern Art, presenting a high level of interactivity with the user, not only due to the possibility of navigation between elements of the exhibitions, but also due to the audio associated with each object. Visually, this application firstly presents the user a menu, where it is possible to see some of the latest news as well as to choose an object of the collection to see. A menu at the bottom presents the main options, whereas the rest of them are accessible through the menu button (Figure 2.11 most left). The collection section allows the user to navigate through the several objects of the gallery, without leaving the main page, leading him to that object once it is selected (Figure 2.11 center left). This application also enables its users to search for a specific part of the exhibition (Figure 2.11 center right), leading the user directly to it, saving time. It allows the user to navigate per and between floors, where he can see the objects in their exact locations (Figure 2.11 most right). Other aspects of this application include the possibility of adding objects to the user's favorites, taking pictures and sharing or adding them to the user's favorites, and navigating through the collection by category, chronological order or by floor.

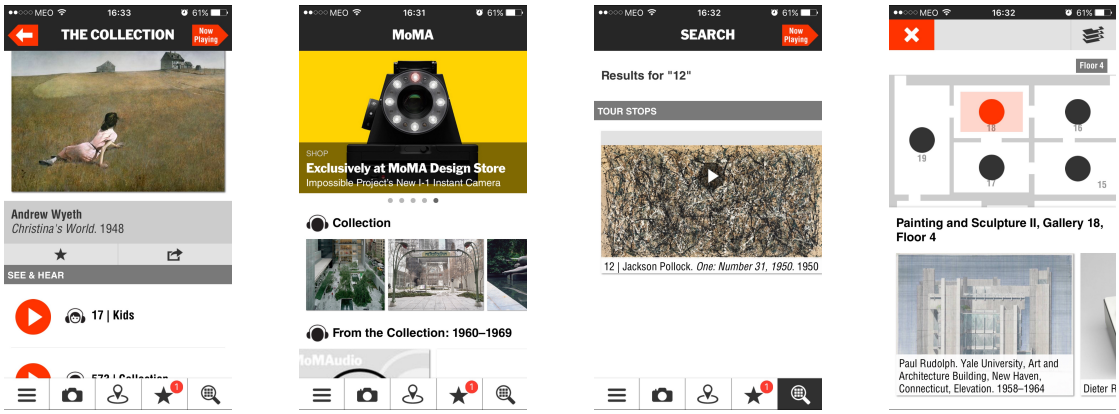


Figure 2.11: MoMA application screens.

The British museum application [2]is another example, although less interactive and more directed to the exhibition. It initially presents the user a menu, allowing him to navigate between halls, to explore through a set of maps or simply to see the objects (Figure 2.12 left). This application also supports search, although by artwork's name, leading the user directly to it. The most relevant aspect of this application is the possibility of seeing the object as if the user were in the exhibition itself. An interactive gallery view is presented to the user, allowing him to swipe left or right to look around the place (Figure 2.12 center). Regarding the artworks, there is also the possibility of observing all of them in a grid view (Figure 2.12 right), leading the user to each one when clicked.

[1]https://www.moma.org
[2]https://itunes.apple.com/gb/app/british-museum-guide/id551275212?mt=8

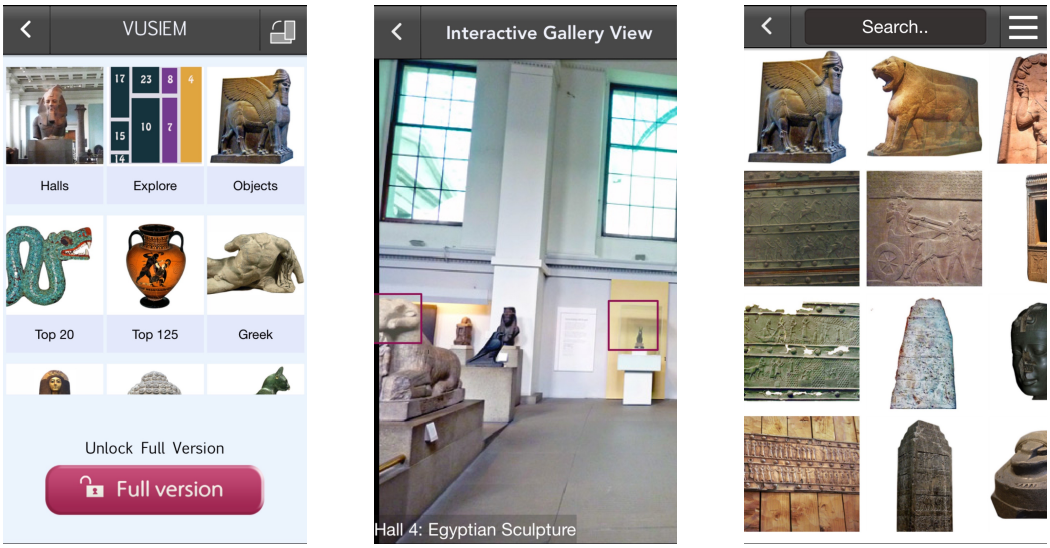


Figure 2.12: British Museum application screens.

In the iOS market there are also a huge number of applications with the same design, made by the same company - eTips LTD [3]. Some of those applications include Louvre Museum Visitor Guide, British Museum Visitor Guide, Vatican Museums Visitor Guide or Natural History Museum of London Visitor Guide. All of these applications have the same layout, presenting the user a main page with a slide out menu, where the user can swipe left or right to navigate through some of the most important topics (Figure 2.13 left). Some of the functionalities include the possibility to save favorites, see the objects in a grid view (Figure 2.13 center) and each of them specifically (Figure 2.13 right). Nevertheless, most of these applications content is paid.

[3]https://itunes.apple.com/us/developer/etips-ltd/id330954824

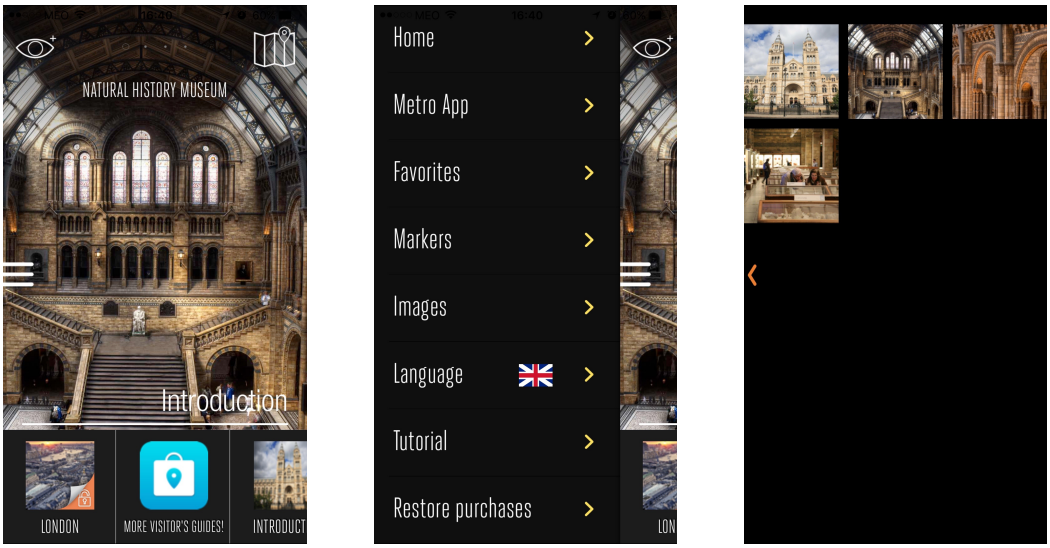


Figure 2.13: Natural History Museum of London Visitor Guide application views.

The Artist Colony [4]is another application, however supporting a more specific type of exhibition, similarly to the one aimed by this project. This is a very simple application that presents the user a menu where it is only possible to choose one of the topics exposed (Figure 2.14 left). Once selected a specific object, a grid view with the paintings related to that topic is shown (Figure 2.14 right), presenting no more than that painting in full screen when selected.

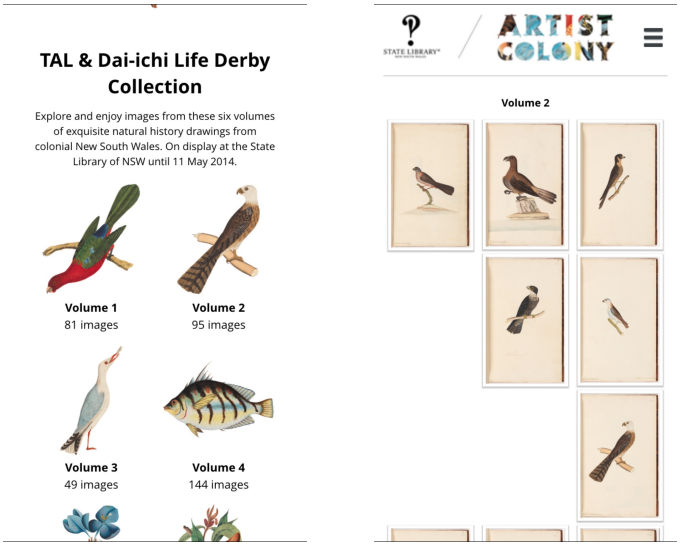


Figure 2.14: Artist Colony Exhibition application views.

[4]https://itunes.apple.com/au/app/artist-colony-exhibition/id839892399?mt=8

Other applications present some improvements, not only in topics directly associated with the exhibition but also regarding how the user can receive the exposed information. If the MoMA application allows the users to listen to the content, the Museum SLB app [5], for example, provides a section with gesture language for those who cannot read (Figure 2.15).

Figure 2.15: Museu SLB application view.

Since nowadays almost all museums or exhibitions have an application to serve as digital support, the number of similar apps has been increasing. However, some improvements can still be made by adding features able to deepen the visitors involvement in the exhibition.

Table 2.5 shows the features of each of the previous described applications.

[5] http://museubenfica.slbenfica.pt/pt-pt/home/museuhome/museuinterativo/aplicacaomobile.aspx

Application:	**Main Features**
Juliet	- Possibility of seeing an object in detail - Use of QR codes - Information about the exhibition
MoMA	- Possibility of seeing an object in detail - Use of audio guides - Use of search bar - Possibility of adding favorites - Information about the exhibition
British Museum	- Possibility of seeing an object in detail - Possibility of seeing a map with the objects in it - Use of Virtual Reality - Information about the exhibition
Museum of London Guide	- Possibility of seeing an object in detail - Information about the exhibition - Possibility of adding favorites
Artist Colony Exhibition	- Possibility of seeing an object in detail - Information about the exhibition
Museum SLB	- Possibility of seeing an object in detail - Use of Audio Guides - Use of Gesture Language - Information about the exhibition

Table 2.5: Description of some of the features of the described applications.

Chapter 3

Requirements

3.1 Introduction to the application domain

The focus of this project comprises the development of an application able to provide support for a real exhibition. In this context, the application domain comprises everything related to the exhibition.

The purpose of an exhibition consists on showing something to the people who visit it, and therefore, the objects need to be the focus. The application should provide several images of the same object as well as some information about the objects and the exhibition. The interactivity with the exhibition also plays an important role, since a simple replica of the real exhibition with the same information as the real one would not add value.

In the real exhibition "70 Cavaquinhos, 70 Artistas" the art pieces, *cavaquinhos*, are displayed following certain rules in order to make the exhibition more enjoyable[38]. Figure 3.2 shows an example of the physical space of the exhibition.

Figure 3.1: Exhibition in Braga - Theatro-Circo, 2015 and in Viana do Castelo - Museu do Traje, 2015.
(source: http://www.cavaquinhos.pt/en/Exhibition-Web.htm)

3.2 Functional requirements (scenarios)

The system requires interaction with essentially two distinct actors: the application visitor and the database administrator/curator, that is the person responsible for gathering and adding the right information in the database. While the visitor will have no responsibilities other than use the application, the curator will take crucial part in the system operation, continuously updating the information. In Figure 3.2 all the use cases[39] associated with the visitor are shown.

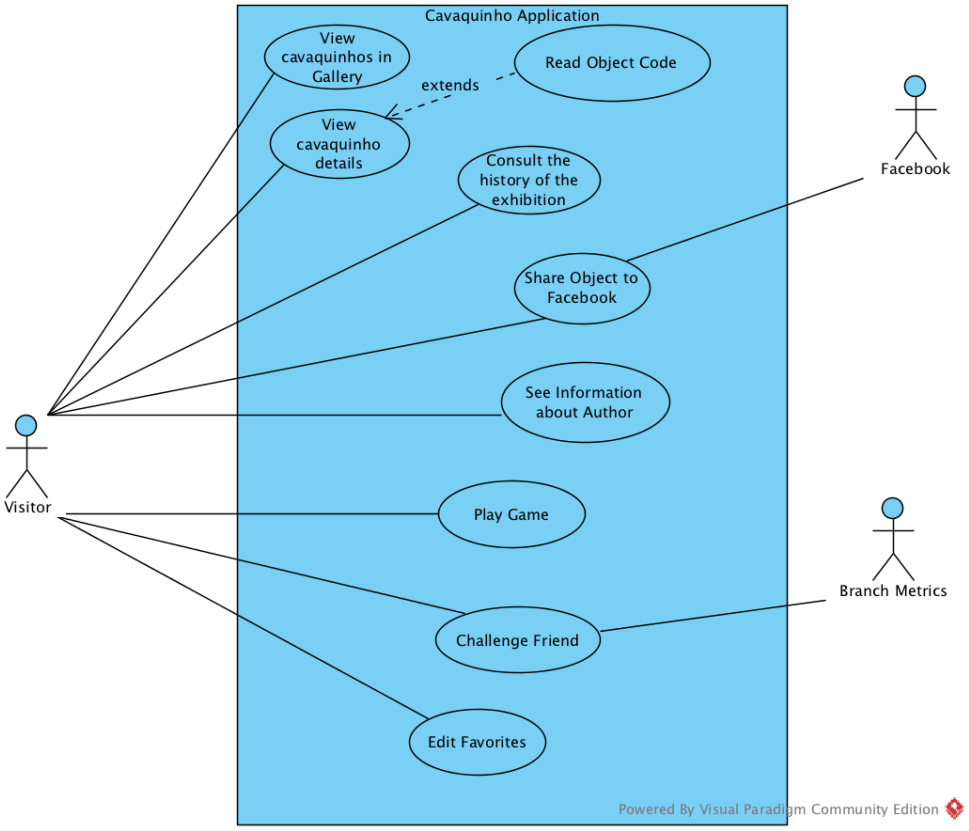


Figure 3.2: Use cases for the visitor.

The Facebook and Branch Metrics are external services used in the Share Object to Facebook and Challenge Friend. The use cases are briefly described in the following text; more detail on the underlying scenarios are provided in annex.

Use Case 1 - View Objects Gallery

The visitor opens the application and the gallery is presented as the main page, allowing him to navigate through the *cavaquinhos* of the exhibition. If the visitor does not have a stable Internet connection and is using the application for the first time, he is not be able to open the application. If the download of the data has already been done, the application does not require the connection.

Use Case 2 - View Specific Object

The visitor opens the application and the gallery appears. He can then select one of the *cavaquinhos* and see it with more detail along with some more information. The visitor can also open the menu and select the Read Code option to read a QR code of that specific *cavaquinho*, which leads him immediately to it.

Use Case 3 - Share Object To Facebook

When the visitor is presented with the gallery, after selecting one of the *cavaquinhos* (or reading the code of a *cavaquinho*), he can open the provided menu and select the Share to Facebook option. If the visitor has not yet made the login, this is not be possible, otherwise he can add some text to the image and share it in Facebook.

Use case 4 - See Favorites

The visitor opens the menu and selects the Favorites option, seeing then the *cavaquinhos* he has already added to his list, although only if he is logged in in the application. Otherwise, the visitor sees a message that asks him to make the login.

Use case 5 - View Information about an author

The visitor opens the menu of the application and selects the Authors List option. In this screen he can see the list of the authors and click in any of them to acquire more information about him.

Use Case 6 - Edit Favorite

When the visitor is presented with the gallery, after selecting one of the *cavaquinhos* (or reading the code of a *cavaquinho*), he can open the provided menu and select the Add to Favorites option. If the visitor has not yet made the login, this is not be possible, otherwise that *cavaquinho* is added to his favorites. Once that *cavaquinho* is on the visitors list, he can access his favorites and remove it.

Use case 7 - See information about the exhibition

The visitor opens the menu and selects the This Exhibition option, which leads him to a screen where he can read information about the exhibition.

Use case 8 - Challenge friend

When the visitor is presented with a *cavaquinho*, after selecting one of the *cavaquinhos* (or reading the code of a *cavaquinho*), he can open the provided menu and select the Challenge a Friend option. If the visitor has not yet made the login, this is not possible, otherwise he is redirected to a screen where he can select a part of that *cavaquinho* and challenge a friend, via Facebook, link or message.

Use case 9 - Play game

The visitor opens the menu and selects the Play game option. This action opens a page where the he is faced with the list of the *cavaquinhos* and the piece to guess to which one does it belong to. The visitor can shake his device to reorganize the list. When he selects the right *cavaquinho*, he is redirected to the a page with the information about that *cavaquinho*. The visitor can then press the back button and repeat the challenge as many times as he wants.

This application is strongly based in images, so we applied a strategy to the image management. The images are initially on the web and the application gets them in run-time. For this to be possible, a stable connection is required, which limits the application usability. Nevertheless, the user is also given the opportunity of downloading the images to his device, allowing him to use the application without Internet. Some of the previous described use cases require one of these two possibilities, while others always require an Internet connection since they perform changes in the database.

The curator actor updates the supporting backend (Figure 3.3).

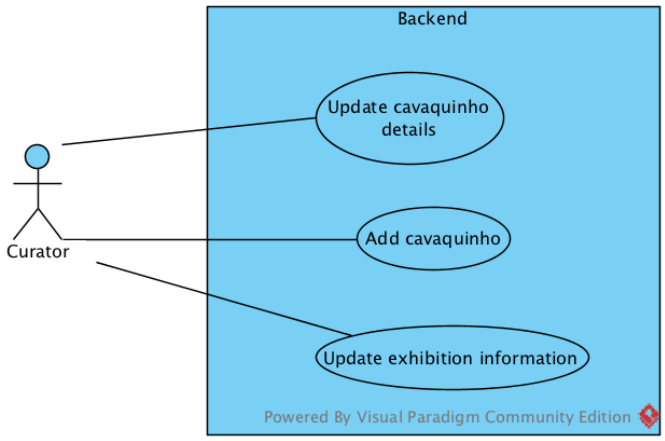


Figure 3.3: Use cases for the curator.

These use cases are briefly described in the following text; more detail on the underlying scenarios are provided in annex.

Use case 10 - Update *cavaquinho* details

The curator accesses the backend and changes information about a specific *cavaquinho*.

Use case 11 - Add *cavaquinhos*

The curator accesses the backend and adds a *cavaquinho* by adding a new entry to the *cavaquinhos* list, providing all the specific information required.

Use case 12 - Update exhibition information

The curator accesses the backend and changes information about the exhibition or about the authors list.

Based on the use cases both for the user and the admin, the application requirements and features are shown in the following tables. The requirement/feature name and the associated level of priority are shown, ranging from MUST - a requirement/feature that must be met, SHOULD - a requirement/feature that should be present, but that does not keep the application to fulfil its purpose, and NICE - a requirement/feature that adds value to the application, however is not essential.

Compatibility and portability requirements	**Priority**
Mobile Application for Android	MUST
Mobile Application for iOS	MUST
Application runs in smartphones (both Android and iOS)	MUST
Application runs in tablets (both Android and iOS)	MUST
Application runs in portrait mode (both Android and iOS)	MUST
Application runs in landscape mode (both Android and iOS)	SHOULD
Application supports English language	MUST
Application supports Portuguese language	SHOULD

Table 3.1: Compatibility and portability requirements.

The application itself should provide the user some specific actions. The following features should be met.

Application Feature	**Priority**
Present a gallery with all objects	MUST
Present specific information about an object	MUST
Allow the user to share an object on Facebook	SHOULD
Allow the user to see several images of the same object	SHOULD
Allow the user to zoom an object	NICE
Allow the user to add an object to his favorites	SHOULD
Allow the user to see information about an author	MUST
Allow the user to read a code to get to an object detailed view	MUST
Allow the user to remove objects from his favorites list	SHOULD
Allow the user to see the information about the exhibition	MUST
Allow the user to download the images to the device internal storage	SHOULD
Allow the user to put/remove author names in gallery images	NICE
Allow the user to challenge a friend through a game	SHOULD
Allow the user play a random game	SHOULD
Login in the application with Facebook	MUST

Table 3.2: Selected application features.

The application should provide the user an enjoyable experience, being at the same time user-friendly. When dealing with images, images taking too long to load may hamper the good usage of the application. To overcome this, the application is expected to load the images from the Internet or from the device according to the users' will. When using the Internet, some cache system should be present to decrease the loading times. The application should also use the minimum possible RAM and CPU usage.

3.3 Generalization opportunities

Although focused on the "70 Cavaquinhos, 70 Artistas" exhibition, we developed this application considering a general perspective, in order to allow the easy adjustment to other exhibitions. Its development focused primarily on the structure rather than on the content, a strategy that allows to change the content with small or none changes in the mobile application itself.

In this context, while some of the previously described use cases were defined specifically for this exhibition, others were developed considering the possible applicability in applications related with other exhibitions. As an example, the logic to access the object details by reading a marker is included in the solution, though suppressed for the "70 Cavaquinhos, 70 Artistas" exhibition by configuration.

Considering the aim of this application, a good visualisation of the objects is crucial to successfully reach the target users: the correct observation of each item has to be guaranteed, in order to captivate the user. Apart from this, several other features could be implemented to enrich the app, providing more information about a specific object or about the entire exhibition. The reading of the QR code leading directly to a specific item or the implementation of a game concerning the exhibited pieces are two examples of additional features that increase the usefulness of the application and the interactivity with the visitor.

In order to adjust this application, specifically developed for this exhibition, to other deployments, some requirements need to be considered. Some of these requirements include the way the information and images are loaded, considering carefully the static and dynamic components. In this solution the load of the images from the Internet and subsequent download allows the total change on the content of the application without changing the mobile application. These aspects have a huge impact in the portability of the application and on its usage in other scenarios and exhibitions.

Chapter 4

System Architecture

4.1 Overall system proposal

The development of an application both in Android and iOS operating systems was considered necessary for the development of this project. As the information is the same in both apps, a single backend service could be shared. Figure 4.1 shows an overall perspective of the proposed system, where applications in both systems share the same backend service.

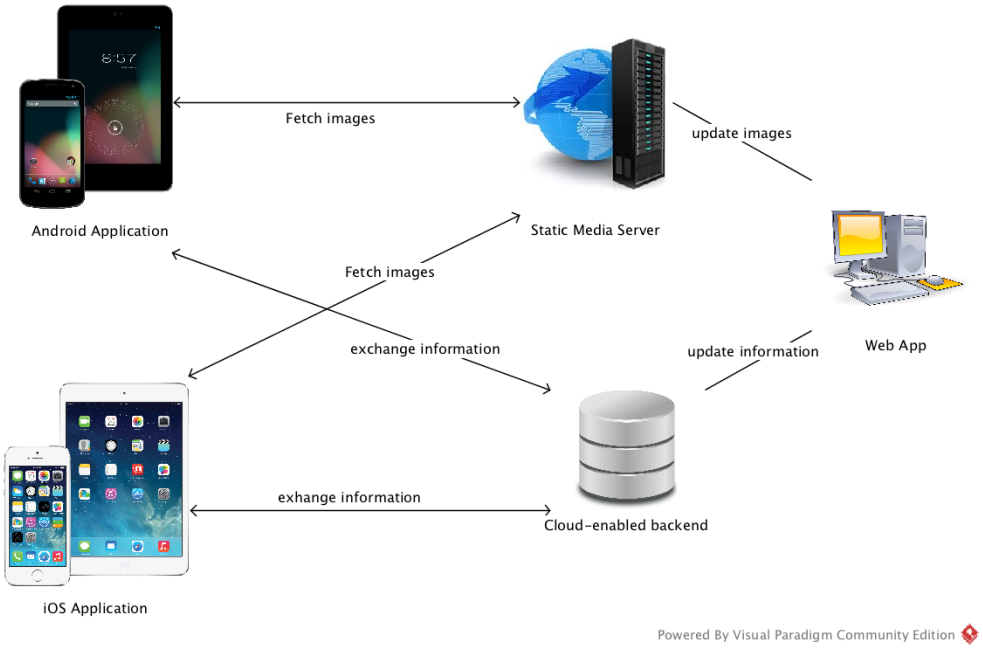


Figure 4.1: Overall architecture of the system.

Context independency was one of the requirements of this project, in order to allow the implementation of any type of object (*cavaquinhos* in this case) in the application, while avoiding the need of changes in the structure. For this to be possible, all the information related to the exhibition needs to be application independent.

We propose an architecture that includes two native applications, for Android and iOS, sharing a common data repository, available on the cloud. For efficient data access the backend should expose REST services. This backend will include the exhibition information, being fetched by both applications, including the links for the images of all the objects. A Static Media Server will store those images that will then be accessed via http requests by both applications to improve efficiency. A Web App will manage both the Cloud-enabled backend and the Static Media server, adding and updating information as required.

4.2 MVC

Model-View-Controller (MVC)[40] is a very common pattern in the software design. It divides the application development in three distinct parts: Model, View and Controller (Figure 4.2). This pattern in used in both mobile application since it simplifies the development process.

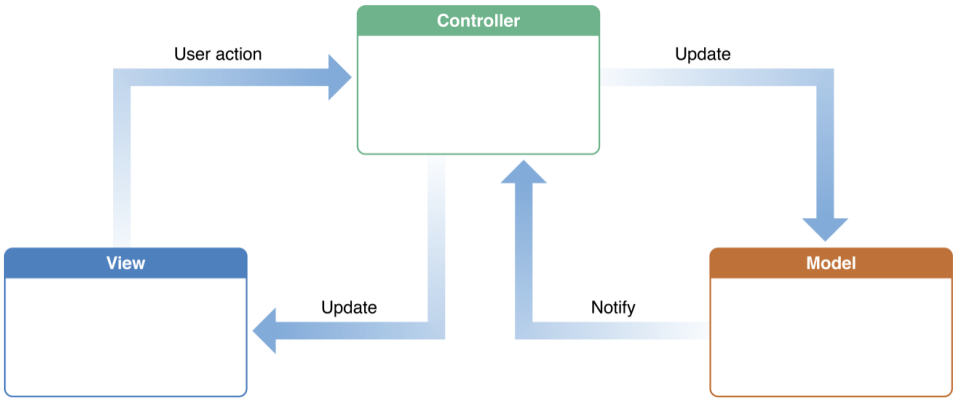


Figure 4.2: Model View Controller (MVC) pattern.
(source:
https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html)

The Model object comprises the data specific to an application and defines the logic and computation that manipulates that specific data. It is possible to have multiple Model objects and have connections between them. However, no explicit connections should be established between the Model objects and the View objects, as this information should be delivered by the Controller objects. The View objects should only present the application data to the users and allow them to manipulate it. These objects know how to draw themselves and how to respond to actions. The Controller object acts as an intermediary between one or more Model objects and View objects of an application. These objects are then the connection, being also able to perform setup and coordinating tasks for an application and manage the life cycle of other objects.

4.3 Android application

We developed the Android application following the Model-View-Controller (MVC) pattern. Figure 4.3 shows a simple example of how this pattern is applied.

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/aboutDetailedDescription"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="20dp"
        android:text="Description"
        android:layout_marginBottom="10dp"
        android:paddingBottom="50dp"
        />

</LinearLayout>
```

```
// Gets the database instance.
data = Data.getInstance();

// Gets the history structure from the DB.
History history = data.getHistory();

// Gets the text view and assigns it the corresponding value.
TextView text = (TextView) findViewById(R.id.aboutDetailedDescription);
text.setText(history.getText());
```

Figure 4.3: Example of the MVC pattern in Android.

In the left side of the Figure 4.3 we have the View, in which the elements are displayed in a view as the developer defines, giving an id to each element that needs data from the Model part. On the right side of Figure 4.3, we have the Controller, that will get the information from the Model and assign it to the View. The data in the Figure is represented by the Data object. The Controller then fetches the History object and in the last line (in yellow) assigns the information to the textview (that was fetched by its id).

Figure 4.4 shows the implemented internal architecture for the Android application.

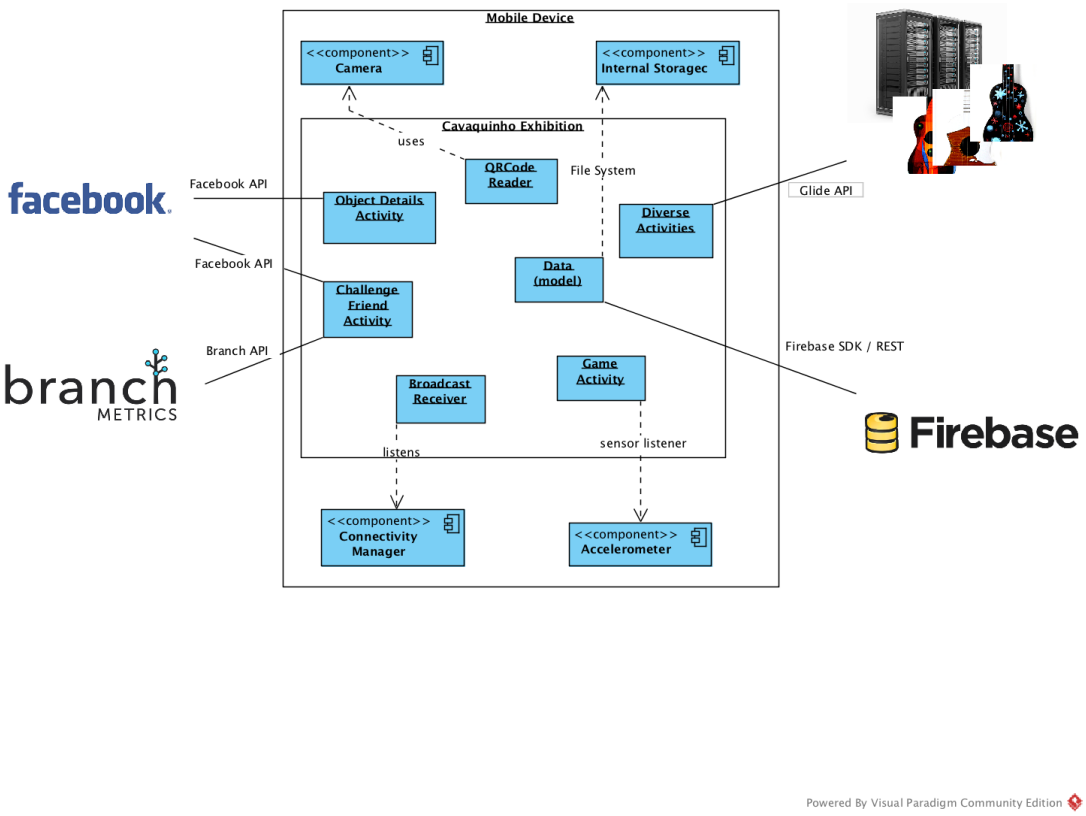


Figure 4.4: Android application architecture.

In the above Figure we can see some of the main activities and components used by the application. The camera, used by the QRCode reader activity to read the QR code of the *cavaquinho*, the Accelerometer used by the Game activity to detect a shake movement, the Connectivity Manager used by a Brodcast Receiver in order to detect changes in the network state, and the application Internal Storage, used by the Data singleton to write and read information.

The application makes use of four external components. The Data class uses Firebase API to access the Firebase and get all the necessary information to run the app. Every activity with images use that information to get the links and, using the Glide API, load the image from the server. Both Object Details Activity and Challenge a Friend activities use the Facebook API to share an image or a challenge to a friend's feed. The Challenge a Friend activity also uses the Branch API to form the link that is then shared via Facebook.

4.4 iOS application

In iOS, the MVC pattern defines not only the role objects play in the application, but it also defines the way objects communicate with each other. Separating these three types of objects are abstract boundaries. Apple considers the MVC model crucial to get a good design for Cocoa application, as the benefits of this pattern are numerous.

Regarding the internal architecture, as the application is the same and the same principles are applied, the architecture is very similar to the Android architecture, with some changes (Figure 4.4).

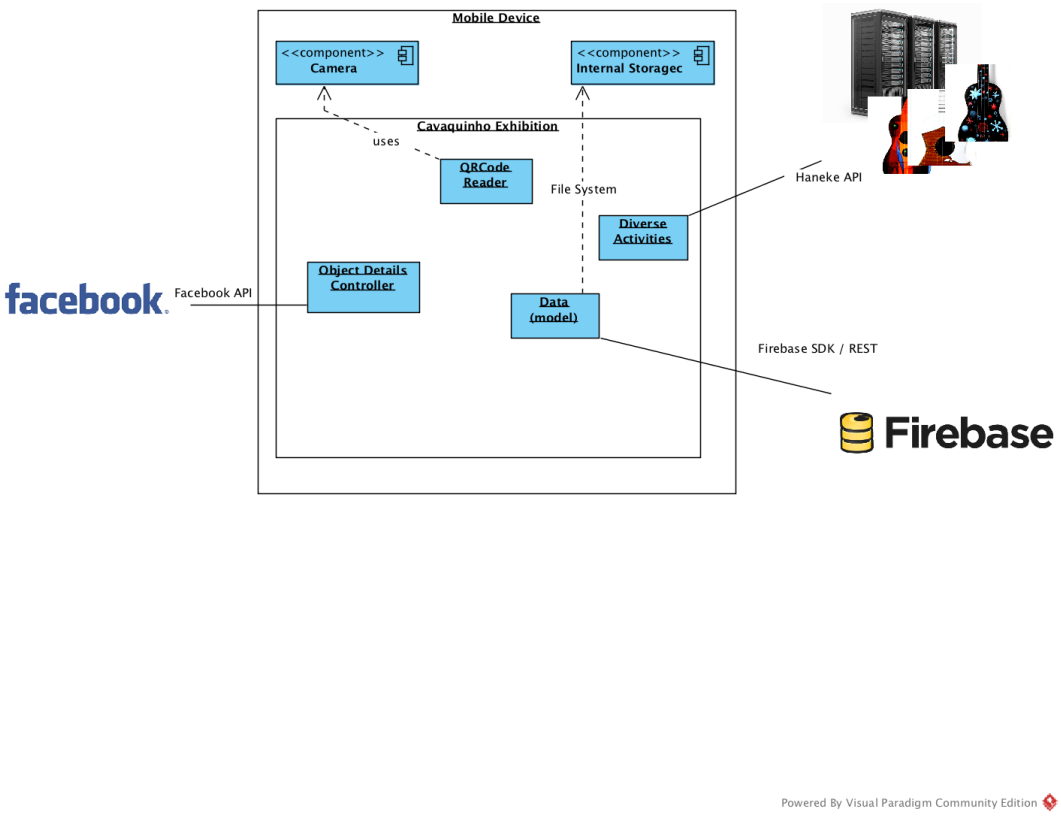


Figure 4.5: iOS application architecture.

In this Figure we can see some of the main controllers and components used by the application. The camera, used by the QRCode reader controller to read the QR code of the *cavaquinho* and the application Internal Storage, used by the Data singleton to write and read information.

As for the external components, the Data class uses Firebase API to access the Firebase and get all the necessary information to run the app. For every view that loads an image, the corresponding controller gets the links and, using the haneke API, loads the image from the server. Object Details Controller uses the Facebook API to share an image to a friend's feed.

4.5 Backend services

Both applications require a backend service and, for this project, the solution found was the usage of a Backend-as-a-Service.

Backend-as-a-Service is a cloud computing category that consists on companies that make easier for developers to setup, use and operate a cloud backend for their mobile, tablet and web apps. This service provides features such as push notifications or integration with social networks. This type of service is provided through a customized Software Development Kit (SDK) and Application Programming Interfaces (API). To use these services, the developer only needs to integrate the SDK in his project being then able to connect with the backend to get or add information.

Between the several available options in the market, we found Firebase to be the most suitable one. This is a powerful platform that allows the construction of cross-platform mobile and web applications, providing both an Android and iOS SDKs. It also provides a REST API service. Firebase is a NoSQL database that uses JSON (JavaScript Object Notation) to properly store and structure all the information as it is crucial to efficiently retrieve all the information. This database is easily administered, since it provides a user-friendly browser interface. Developed by Google, Chrome also provides an extension to better manipulate the database, providing an easier way to manipulate JSON.

This service relies on a real time database, providing authentication and security. It is used in the project to include all the exhibition related information, allowing multiple devices connections (Figure 4.6).

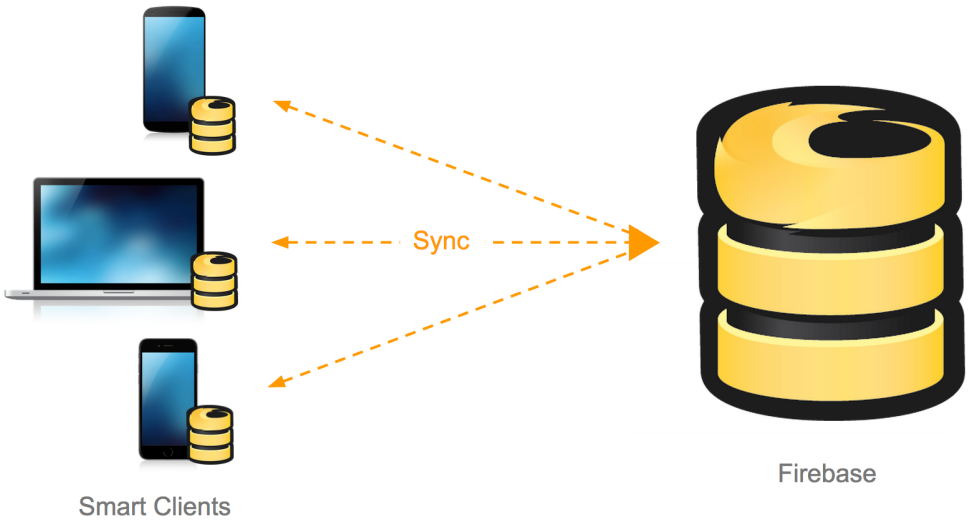


Figure 4.6: Backend (Firebase) respresentation.

In spite of the fact that Firebase stores all the information related to the application, image storing was not supported at the time we developed this project. As consequence, the images are stored in a different server, being only the image URL on the Firebase database.

4.6 Data and Image synchronization strategy

Both mobile applications developed in this project make an heavy use of image management, since the image is a crucial part on the application itself. The data contains 70 *cavaquinhos*, each of them with four images (that can be more), which comprises a total of 280 images. Additionally, one of the goals of this project consisted on allowing the usage of the application both online and offline. For this to be possible, a strategy for the management of the information had to be implemented, in order to keep the information up-to-date with the one in the database. Thus when the database manager changes something, that change is reflected in the application right away. The mobile application has however to be capable of determining whether there is a need of updating the downloaded information. This can be achieved by writing in the private internal storage of the app, keeping track of the state.

The chosen strategy for updating the information is simple (Figure 4.7): the first time the user initiates the app, the download with all the information starts (with exception of the images) and that information, containing the authors names, biographies, exhibition information among others, is stored in files. At the same time, a version attribute is fetched from the database, with the rest of the information, and that version is stored too. From this point forward, the behaviour is always the same: if the user has no Internet access, the current version in the app is fetched, and if existent, the app runs with that version; if not, it warns the user that an Internet connection is required. If the user has Internet access, the version on the database is fetched, and compared with the one in the app. If the version is the same, it runs with the content in the files, otherwise, it downloads the new version and re-writes on the files and updates the version attribute.

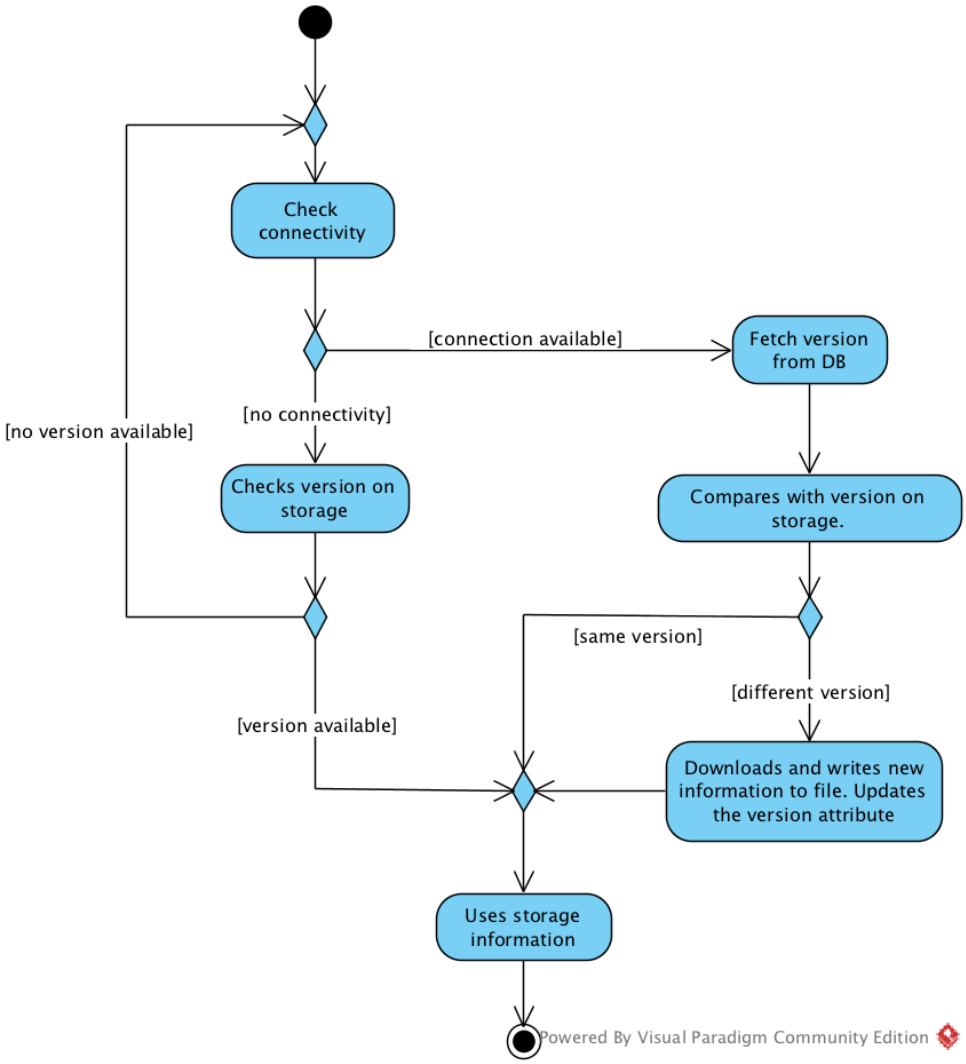


Figure 4.7: Information management strategy applied.

In what concerns images, we followed a different approach. Considering the large number of images in the app, the pre-defined behaviour for the application to run consists on loading the images from the Internet, using some kind of cache system. In this case, the Glide[41] (Android) or Haneke[42] (iOS) library takes care of the load and cache system, with some configurations. However, with this strategy, the user loses the possibility of using the app offline, with possible performance problems if the Internet connection is weak. The user has the possibility of downloading all the images to the internal storage of the device, so that the app can be used in an offline mode. The applied strategy, as can be seen from Figure 4.8, is very similar to the one applied in the data update. When the user firstly chooses to download the images, the version attribute is fetched from the database and stored in the internal storage alongside with all the metadata of the images. From this point forward, that information is used, allowing the user to make use of the app without Internet. When the user chooses to download again, it is firstly checked if a new version is available and if so, the information is replaced and the version attribute updated.

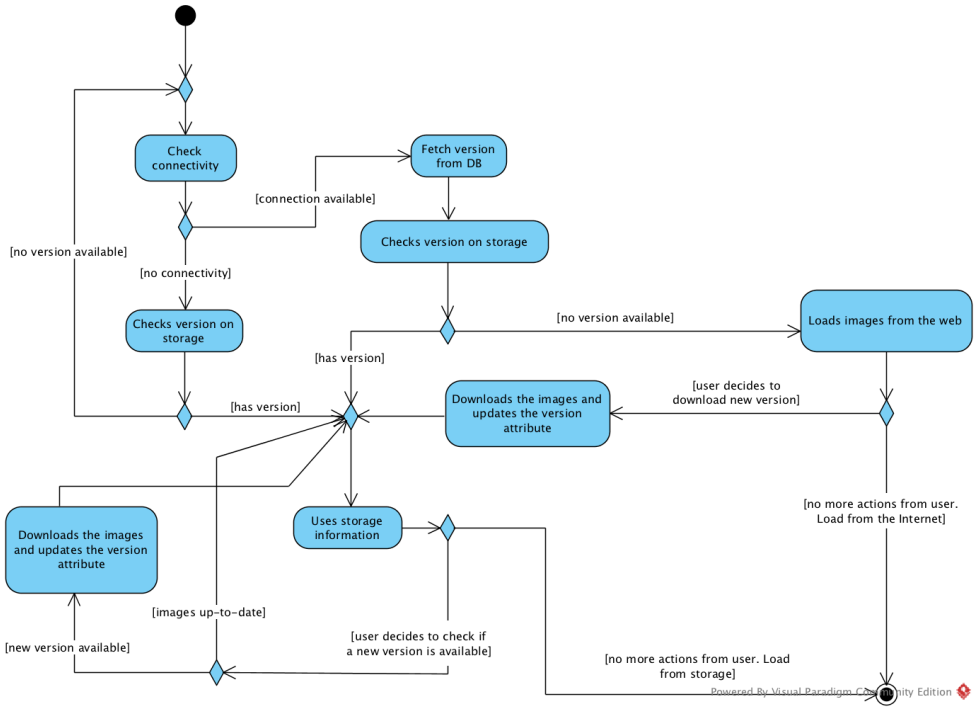


Figure 4.8: Image management strategy applied.

Chapter 5

Implementation

This chapter comprises all the aspects related with the implementation process of the project. We give a detailed explanation about Android and iOS implementations as well as about the backend service. All the tools are described as well as what makes this application suitable for its purpose.

5.1 Evolutionary prototypes

Two aspects are crucial when developing an Android application: the activity lifecycle and the fragments lifecycle. They allow maintaining the application in a stable state, with the information up-to-date. Besides these two components of the Android world, others are also important to accomplish certain tasks, as the Receivers or Notification Manager. Next, the main aspects of the Android version of this application will be described.

This was an incremental work, as some features were defined from the beginning while others were only added afterwards. Even some features that were initially defined were then removed. Nevertheless, we still present some of those features since they can be suitable for a future version of the application or for some other exhibitions.

One of the most relevant features is the possibility to draw on top of a pre-defined *cavaquinho*, as can be seen from Figure 5.1. This allows the user to paint, erase and reset the picture, for later to save or share on the Internet. This is a feature that can be improved in order to allow some kind of interactivity between users.

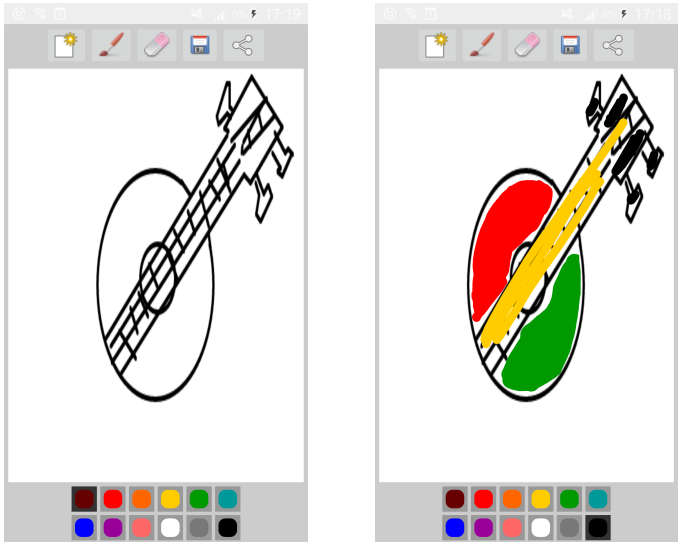


Figure 5.1: Example of the small paint editor created for the first developed Android version.

Another advancement, afterwards withdrawn, was the rearrangement of the gallery items when the device was shaken. Although this shake logic was not implemented in the gallery, it was later used in another part of the application, as can be seen forward in this section.

Figure 5.2: Gallery view of a previous developed Android version.

5.2 Android Implementation

In this section, we describe the implementation of the more important aspects of the solution.

The application follows the Google Design Guidelines[43] as well as the menu, although with a small difference. We used a DrawerLayout element, in order to avoid showing the navigation bar and get more screen space for the *cavaquinhos*. Nevertheless, to inform the user about the presence of the menu, we placed a button in the top left corner of the screen. The DrawerLayout already takes care of the open/close logic, however, as the menu button was not integrated, the appearing and disappearing of this button was processed: every time a user closes the menu or comes back from another screen it should become visible, while when the user opens the menu by using either the swipe gesture or the button itself, it should become invisible (Figure 5.3).

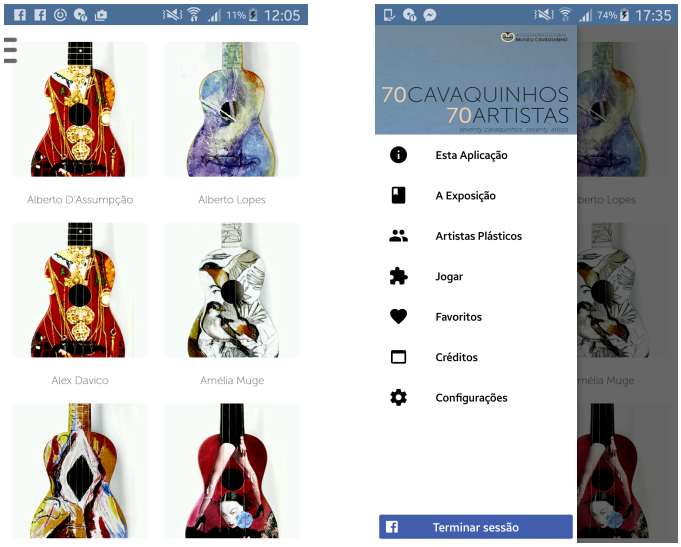


Figure 5.3: Applications' main page with and without the menu open.

The DrawerLayout is divided into two main elements (Listing 5.1): a FrameLayout where the page content should be placed and a RelativeLayout with the drawer elements. In the activity, we assigned a grid to the FrameLayout and a list to the ListView inside the RelativeLayout. Then we configured this list with the redefinition of a list adapter, in order to get the items with an image and text. To complete this menu, we assigned an item click listener to the ListView in order to detect the user click in any option and perform the required action.

```
<android.support.v4.widget.DrawerLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:id="@+id/drawer_layout"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".MainActivity" >

        <FrameLayout
            android:id="@+id/main"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:background="@android:color/white">

            <include layout="@layout/content_main" />

        </FrameLayout>

        <RelativeLayout ... >

            <LinearLayout ...>

                <ImageView .../>

                <ListView .../>

            </LinearLayout>

            <com.facebook.login.widget.LoginButton ...>

        </RelativeLayout>


    </android.support.v4.widget.DrawerLayout>
```

Listing 5.1: Xml of the implemented DrawerLayout element.

We implemented the qrcode code reading functionality using the external library ZXing Android Embedded[1], which only requires the call and initialisation of the IntentIntegrator function and the configuration of some parameters (optional) (Listing 5.2). The final result obtained is then treated and associated (or not) with a specific *cavaquinho*, opening the respective page in order for the user to know more about it. This feature is not present in the final version by configuration.

```
IntentIntegrator integrator = new IntentIntegrator(MainActivity.this);
integrator.setCaptureActivity(AnyOrientationCaptureActivity.class);
integrator.setPrompt("Scan something");
integrator.setOrientationLocked(false);
integrator.setBeepEnabled(false);
integrator.initiateScan();
```

Listing 5.2: Excerpt of the code used for the QR code reader function.

From Figure 5.4 left, it is possible to see all the different options provided by the application. We achieved both the implementation of Favorites and Authors List views by redefining the BaseExpandableListAdapter, as can be seen in Figure 5.4. This class accepts a list of elements for the header and another list of elements for the content, having two different methods to expand both of them. On the layout part, the ExpandableListView was the only required element. Afterwards, assigning the redefined adapter to this element is all it is needed.

[1]https://github.com/zxing/zxing

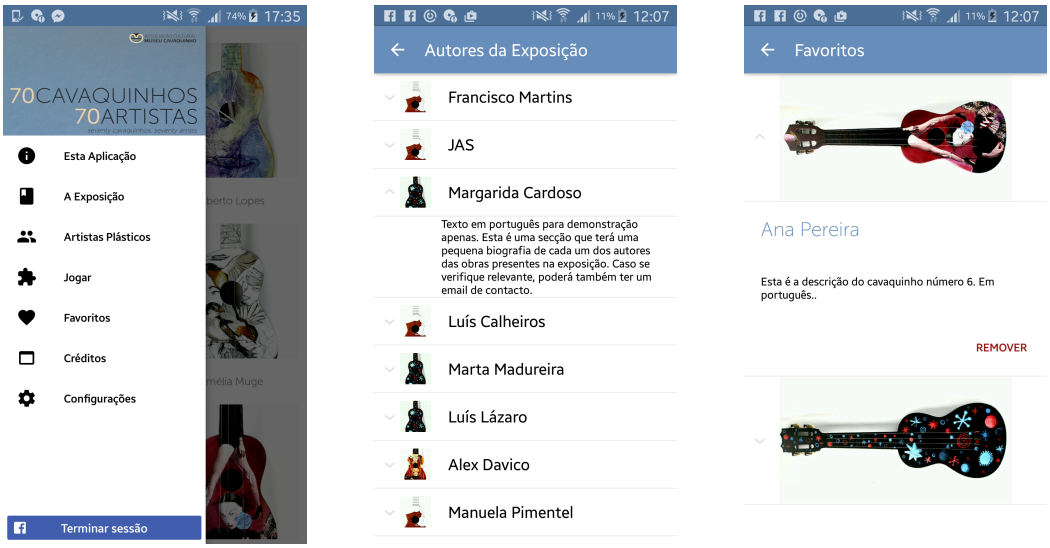


Figure 5.4: Main menu, Authors and Favorites pages.

Later, we integrated in the application the Play Game option. This consists on a grid view with all the objects and a fragment of a picture (Figure 5.5), generated randomly by the app. The user has then to drag and drop the fragment on the object it belongs to. If the correlation between the fragment and the object was correctly established, another page with that object opens up.

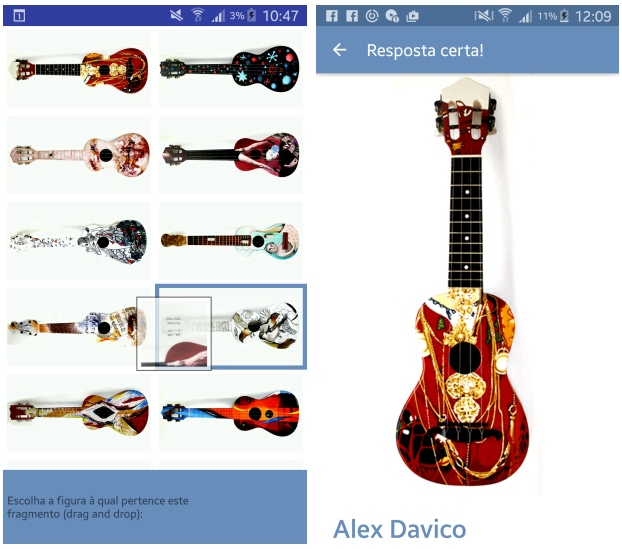


Figure 5.5: Game page and the page that opens on a right answer.

We implemented a rearrangement of the *cavaquinhos* when shaking the device in this Play Game option, by creating a class implementing the SensorEventListener interface. In this class, we defined a threshold as well as a time between movement detection of the smartphone and a reset time, instantiating it in the class where the movement needed to be detected and assigning a listener to it. We used the SensorManager to help calculate the gravity force, which was then compared with the defined threshold to determine if there was a shake or not. In Listing 5.3 the corresponding code is shown. The SensorManager and a Sensor instances also need to be instantiated. In order to achieve this, a request in the manifest has to be performed to the device accelerometer.

```
float x = event.values[0];
float y = event.values[1];
float z = event.values[2];

float gX = x / SensorManager.GRAVITY_EARTH;
float gY = y / SensorManager.GRAVITY_EARTH;
float gZ = z / SensorManager.GRAVITY_EARTH;

// gForce will be close to 1 when there is no movement.
float gForce = (float)Math.sqrt(gX * gX + gY * gY + gZ * gZ);

if (gForce > SHAKE_THRESHOLD_GRAVITY) {
    final long now = System.currentTimeMillis();
    // ignore shake events too close to each other (500ms)
    if (mShakeTimestamp + SHAKE_SLOP_TIME_MS > now) {
        return;
    }

    // reset the shake count after 3 seconds of no shakes
    if (mShakeTimestamp + SHAKE_COUNT_RESET_TIME_MS < now) {
        mShakeCount = 0;
    }

    mShakeTimestamp = now;
    mShakeCount++;

    mListener.onShake(mShakeCount);
}
```

Listing 5.3: Used code to detect the shake movement of the device.

It is also through this menu that the user can perform the login and logout of the application. We used the SDK provided by Firebase to perform the login, implementing the Facebook login independently. The Facebook generated token is used to perform the Firebase login. As the Facebook SDK keeps track of this access token between sessions, a verification occurs when the user tries to connect with Facebook to confirm whether the user is already logged in to Facebook and if it is necessary to get the token for the Firebase to use. Listing 5.4 shows the code that allows executing the login.

```
public void onFacebookAccessTokenChange(AccessToken token) {
    if (token != null) {
        ref.authWithOAuthToken("facebook", token.getToken(), new Firebase.AuthResultHandler() {
            @Override
            public void onAuthenticated(AuthData authData) {
                // The Facebook user is now authenticated with your Firebase app
                Map<String, String> map = new HashMap<String, String>();
                map.put("provider", authData.getProvider());
                if (authData.getProviderData().containsKey("displayName")) {
                    map.put("displayName", authData.getProviderData().get("displayName").toString())
    ;
                }
                ref.child("users").child(authData.getUid()).setValue(map);

                setAuthData(authData);
                setFavoritesListener();
            }

            @Override
            public void onAuthenticationError(FirebaseError firebaseError) {
                // there was an error
            }
        });
    } else {
    /* Logged out of Facebook so do a logout from the Firebase app */
        ref.unauth();
    }
}
```

Listing 5.4: Method that performs the Firebase login using Facebook.

All the other functionalities of the application are accessible in the detailed view of each object. When selecting an object, a new page is opened. We defined that the image should take almost all the space, as it is the main element. As an object has several images, this page includes a ViewPager in conjugation with a CirclePageIndicator to allow the user to swipe through those images. The uses code can be seen in Listing 5.5.

```
<android.support.v4.view.ViewPager
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/pager"
    android:layout_width="match_parent"
    android:layout_height="@dimen/view_pager_view_pager_portrait_height"
    android:orientation="vertical"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:context=".ScreenSlidePagerActivity"
    tools:showIn="@layout/activity_screen_slide_pager_activity"
    android:background="@color/textColor"/>

<com.viewpagerindicator.CirclePageIndicator
    android:id="@+id/titles"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent"
    android:padding="@dimen/imageFragment_selector_padding"
    android:background="@color/backgroundColor"
    app:fillColor="@color/imageFragment_pageSelector_fill__color"
    app:pageColor="@color/imageFragment_pageSelector_page__color"
    app:strokeColor="@color/imageFragment_pageSelector_stroke__color"/>
```

Listing 5.5: Xml used for the gallery style image show.

We implemented a floating button to give the user the same actions when he is on the detailed page: the possibility of adding it to the user's favorites, playing a game or sharing an image of the object to Facebook. We implemented the latter action by importing the official SDK, initialising it and creating an instance of a CallbackManager and ShareDialog. When pressing the sharing button, a bitmap image or URL was assigned to a SharePhoto instance, being then build the content to share. This process can be seen in Listing 5.6.

[1]https://github.com/chrisbanes/PhotoView

```
if (ShareDialog.canShow(ShareLinkContent.class)) {
    SharePhoto photo = new SharePhoto.Builder()
            .setBitmap(bitmap)
            .build();
    SharePhotoContent content = new SharePhotoContent.Builder()
            .addPhoto(photo)
            .build();

    ShareDialog.show(getActivity(), content);
}
```

Listing 5.6: Used function to share an image to Facebook.

To add Favorites a simple call to the database adding that object is needed. The Challenge a friend option was achieved with the help of another library[2], that provides firstly a page where the user can crop the object and then another page to visualise the result and share it with friends (Figure 5.6).

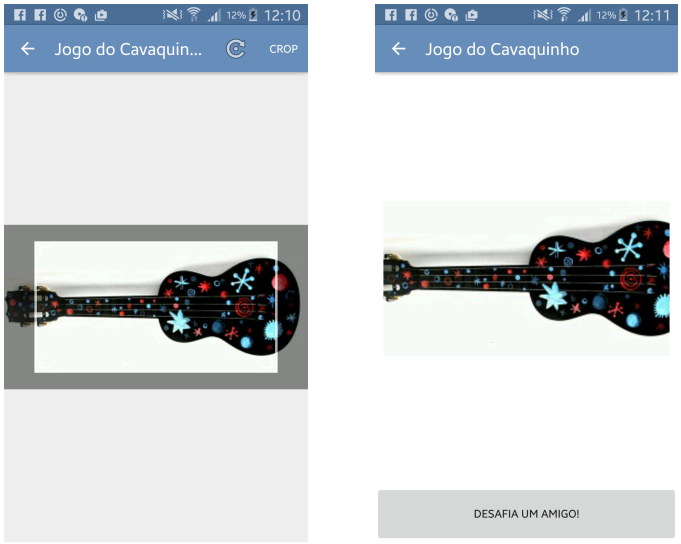


Figure 5.6: Challenge a Friend pages.

This Share option will publish a branch link, provided by the Branch Metrics[3]. When clicking on the link, identification of the device occurs, acting differently depending on Android, iOS or web. If in Android, it will check if the user has the application installed, and if so, will open up the game page with that game. If the app is not installed, the user will be redirected to the Play Store to download the application. The same process happens for iOS. The application will then have to recognise when the app is opening from a link or not. This is done by the code in the next Figure (Listing 5.7), alongside with a reference in the manifest to the activity that is supposed to open - game activity.

[2]https://github.com/ArthurHub/Android-Image-Cropper
[3]https://branch.io

```
@Override
public void onStart() {
    super.onStart();

    final Branch branch = Branch.getInstance();
    branch.initSession(new Branch.BranchReferralInitListener() {
        @Override
        public void onInitFinished(JSONObject referringParams, BranchError error) {
            if (error == null) {
                // params are the deep linked params associated with the link that the user
                // clicked before showing up
                Log.i("BranchConfigTest", "deep link data: " + referringParams.toString());

                try {
                    Log.i("BranchConfigTest", "deep link data: " +
                            branch.getLatestReferringParams().getString("imagecode"));
                } catch (JSONException e) {
                    e.printStackTrace();
                }
            }
        }
    }, this.getIntent().getData(), this);
}

@Override
public void onNewIntent(Intent intent) {
    this.setIntent(intent);
}
```

Listing 5.7: Used function when challenging a friend.

Given the high image content of the application, the user memory needed to be considered. The solution found consisted on the use of the Glide library[4], a fast and efficient open source media management and image loading framework for Android. This library wraps media decoding, memory and disk caching into a simple and easy to use interface, also providing image transformations. Its use is demonstrated in Listing 5.8. In this version, all images started to be loaded using this library.

```
Glide.with(getContext())
    .load(Uri.fromFile(outFile))
    .asBitmap()
    .error(R.drawable.ic_error)
    .placeholder(R.drawable.waiting)
    .transform( new RotateTransformation( context, 90f ))
    .into(image);
```

Listing 5.8: Example of the Glide code used to load an image.

This library can be used with both application resources or with an image URL, and allows the use of a placeholder while the image loads and another for when the load fails. It can also be used to download the images when the user chooses to. The downloaded files are then saved in the internal storage of the device (to be used when needed), being used from that point forward, as this library also allows the load of images from a URI. As explained in the architecture section, this is a process that passes through a series of validations, in order to properly load the images either from the internal storage or from the server.

We also implemented a broadcast receiver to warn the application if the state of the Internet changed. In fact, it initiates the application if the user is in the splash page without Internet and with no version on the internal storage, and in the meantime turns on the connection. The used code can be seen in Listing 5.9.

[4]https://github.com/bumptech/glide

```
private final BroadcastReceiver myReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {

        if (intent.getExtras() != null) {
            final ConnectivityManager connectivityManager =
                    (ConnectivityManager)context.getSystemService(Context.CONNECTIVITY_SERVICE);
            final NetworkInfo ni = connectivityManager.getActiveNetworkInfo();

            if (ni != null && ni.isConnectedOrConnecting()) {
                if (!dataInit) {
                    Data data = Data.getInstance(context);
                    dataInit= true;
                }
            } else if (intent.getBooleanExtra(ConnectivityManager.EXTRA_NO_CONNECTIVITY,
                    Boolean.FALSE)) {
                Log.d(TAG, "There's no network connectivity");
            }
        }
    }
};
```

Listing 5.9: Code used to implement the broadcast receiver.

Since there are some provided features by the application that can be misconceived, we developed a tutorial that is shown to the user on the first time he launches the application.The Challenge a Friend option can also pass undetected to the user. The solution to this problem was to provide a tutorial explaining how to challenge a friend. Both this tutorials are accessible in the This Application page.

During the development of this application several libraries were used. A short description of each of them is presented in Table 5.1

Library	Description	Use in the application
zxing	ZXing ("zebra crossing") is an open-source, multi-format 1D/2D barcode image processing library implemented in Java, with ports to other languages.	Used to the QR code reading logic
PhotoView	PhotoView aims to help produce an easily usable implementation of a zooming Android ImageView.	Used in the zooming of images.
Glide	Glide is a fast and efficient open source media management and image loading framework for Android that wraps media decoding, memory and disk caching, and resource pooling into a simple and easy to use interface.	Used in the loading of images from URLs or from the internal storage and in the rotation of some images.
FloatingActionButton	Yet another implementation of Floating Action Button for Android with lots of features.	Used to construct the FAB button in the detailed items.
Retrofit	Type-safe HTTP client for Android and Java by Square, Inc.	Used to process request to the database.
ViewPagerIndicator	Paging indicator widgets that are compatible with the ViewPager from the Android Support Library to improve discoverability of content.	used to apply the dots below the images in the detailed item screen, allowing the user to position himself
Android Image Cropper	Powerful (Zoom, Rotation, Multi-Source), customizable (Shape, Limits, Style), optimized (Async, Sampling, Matrix) and simple image cropping library for Android.	Used in the cropping of the image on the challenging a friend action
Butter Knife	Field and method binding for Android views which uses annotation processing to generate boilerplate code for you.	used to simplify some of the classes of the application
AppIntro	AppIntro is an Android Library that helps you make a cool intro for your app, like the ones in Google apps.	Used to develop a tutorial.

Table 5.1: Libraries used in the application development.

5.3 iOS Development

When developing in Android, the Activity lifecycle and Fragment lifecycle are two essential concepts; however, when in iOS programming there is no parallel to these concepts, but instead the App Life Cycle. Applications are a sophisticated interplay between code and system frameworks, with those system frameworks providing the basic infrastructure that all applications need to run. It is possible to customise those infrastructures to give the application the desired design and features.

iOS frameworks rely on design patterns such as the Model-View-Controller and delegation in their implementation. The good understanding of those designs is crucial to the successful creation of an application.

Regarding the implementation itself, we developed this iOS application after the Android version. As a consequence, and although the general aspects were already defined - leading to only one version of the iOS app, some features were not possible to be developed in the iOS environment or were simply too complex. As a result, this version presents some differences when compared to the Android version.

While in Android there is a grid element, in iOS the correspondent element is a UICollection, harder to configure since there is no specific number of columns. Therefore, we configured this aspect by getting the screen dimensions and changing the object width each time the device had a change of orientation. This was achieved by overriding the viewWillTransitionToSize function, as can be seen in Listing 5.10.

```
override func viewWillTransitionToSize(size: CGSize, withTransitionCoordinator coordinator:
    UIViewControllerTransitionCoordinator) {

    // Caled to force the change of cell size when rotating the devide
    super.viewWillTransitionToSize(size, withTransitionCoordinator: coordinator)
    self.collectionView!.performBatchUpdates(nil, completion: nil)
}

func collectionView(collectionView: UICollectionView, layout collectionViewLayout:
    UICollectionViewLayout, sizeForItemAtIndexPath indexPath: NSIndexPath) -> CGSize {

    // Gets the screen dimensions
    let screenSize: CGRect = UIScreen.mainScreen().bounds
    let screenWidth = screenSize.width
    let screenHeight = screenSize.height

    var size = screenWidth/2-15

    // Defines the cell size in order to fill 2 cells if in portrait or 3 if in landscape
    if UIDevice.currentDevice().orientation.isLandscape.boolValue {
        size = max(screenWidth, screenHeight)/3-10
    } else {
        size = min(screenWidth, screenHeight)/2-10
    }

    return  CGSize(width: size, height: size)
}
```

Listing 5.10: Code that defines the size of a cell, considering the device screen dimensions.

This configuration leads to an initial screen with objects displayed into two or three columns, depending on the portrait or landscape mode adopted by the device (Figure 5.7). It is also possible to see the slide out menu implementation, with the same icons used in the Android implementation.

We achieved this specific menu implementation by using the SWRevealViewController[8] library with some modifications. Although the menu was designed to have the menu accessible in every page, allowing the user to navigate from one page to another, in this implementation a main page with the images was preferred, with the other pages being accessible from that menu. Those pages once open have a back button in the top left corner allowing to get the user back to the initial page. This way the similarity with the Android application is higher.

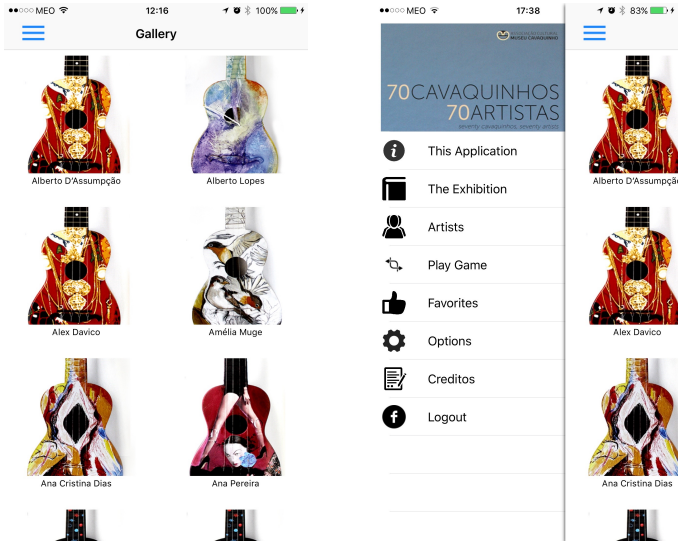


Figure 5.7: Gallery views of the developed iOS version.

In terms of code, this implementation required a main page and a list to apply to the menu. This list was configured to have configurable cells, in order to allow each one to have its own title and icon. The connection with the other pages is then easily achieved using segues from the cell to the specific controller in the Storyboard.

The QRCode reader is also present in the application, implemented using the AVFoundation. We created a controller that implemented the AVCaptureMetadataOutputObjectsDelegate protocol, in order to intercept any metadata found in the input device. We used the AVCaptureDevice class to initialise a device object and provide the video as the media type parameter. AVCaptureMetadataOutput was also used to set the output device to the capture session (Listing 5.11).

```
objCaptureSession = AVCaptureSession()
objCaptureSession?.addInput(objCaptureDeviceInput as! AVCaptureInput)
let objCaptureMetadataOutput = AVCaptureMetadataOutput()
objCaptureSession?.addOutput(objCaptureMetadataOutput)
objCaptureMetadataOutput.setMetadataObjectsDelegate(self, queue: dispatch_get_main_queue())
objCaptureMetadataOutput.metadataObjectTypes = [AVMetadataObjectTypeQRCode]
```

Listing 5.11: Excerpt of code used for the QR code reader function.

Afterwards, we added a Video Preview layer for the QRCode Read and display on the device, achieved using the AVCapturePreviewLayer class. Some other aspects were also defined, such as the border width and the color shown when the code is detected on screen. We also added the QRcode view to the main view.

For the detection of the result, we implemented the captureOutput function. Since this function captures everything and even when there is a result, the reader keeps searching for other input (leading to the read of the same code several times), some validations need to be performed. Listing 5.12 shows how these aspects were solved.

```
func captureOutput(captureOutput: AVCaptureOutput!, didOutputMetadataObjects
    metadataObjects: [AnyObject]!, fromConnection connection: AVCaptureConnection!) {

    if metadataObjects == nil || metadataObjects.count == 0 {
        vwQRCode?.frame = CGRectZero
        return
    }
    let objMetadataMachineReadableCodeObject = metadataObjects[0] as!
     AVMetadataMachineReadableCodeObject
    if objMetadataMachineReadableCodeObject.type == AVMetadataObjectTypeQRCode {
        let objBarCode = objCaptureVideoPreviewLayer?
            .transformedMetadataObjectForMetadataObject(objMetadataMachineReadableCodeObject
            as AVMetadataMachineReadableCodeObject) as! AVMetadataMachineReadableCodeObject
        vwQRCode?.frame = objBarCode.bounds;
        if objMetadataMachineReadableCodeObject.stringValue != nil {
            // We get a result
            if let value = objMetadataMachineReadableCodeObject.stringValue.integerValue {
                if value>=0 && value<singleton.getNumberTotalImages() {
                    pageIndex = Int(objMetadataMachineReadableCodeObject.stringValue)
                    if (!called) {
                        called = true
                        singleton.setQRcodeRead(pageIndex)
                        self.performSegueWithIdentifier("codeRead", sender: nil)

                    }
                }

            }

        }
    }
}
```

Listing 5.12: Another excerpt of code used for the QR code reader function.

Once the proper result is read, the associated page needs to be open. This was a particularly difficult task to accomplish, since after the opening of the associated page the user should be able to return to the initial page instead of the QRcode reader. Since iOS development works with controllers (putting each new task on top of the previous one), it was needed to open the main page again and direct the user to the specific object. This problem was solved using the created singleton. As it is possible to observe in the code in Listing 5.12, when the reader gets a positive match, the setQRCodeReader method is called, passing the code and afterwards performing a segue to the main page. Once in there, it is checked in the database whether the code was read and if so, it gets the code and preforms another segue to the specific object.

Regarding the page with the object (Figure 5.8), this page also presents a gallery view with images of the same object, the title and description of the object, and a Share and Favorites buttons. The same ViewPager interaction was also applied, allowing the user to swipe left or right to get to the previous or next object. We implemented this behaviour with a UIPageViewController, which deals with the managing of this element, defining the current one to be displayed and preparing the previous and next one, through the viewControllerBeforeViewController and viewControllerAfterViewController methods. This manager calls the specific page view controller, which will then configure all the elements to put in the page.

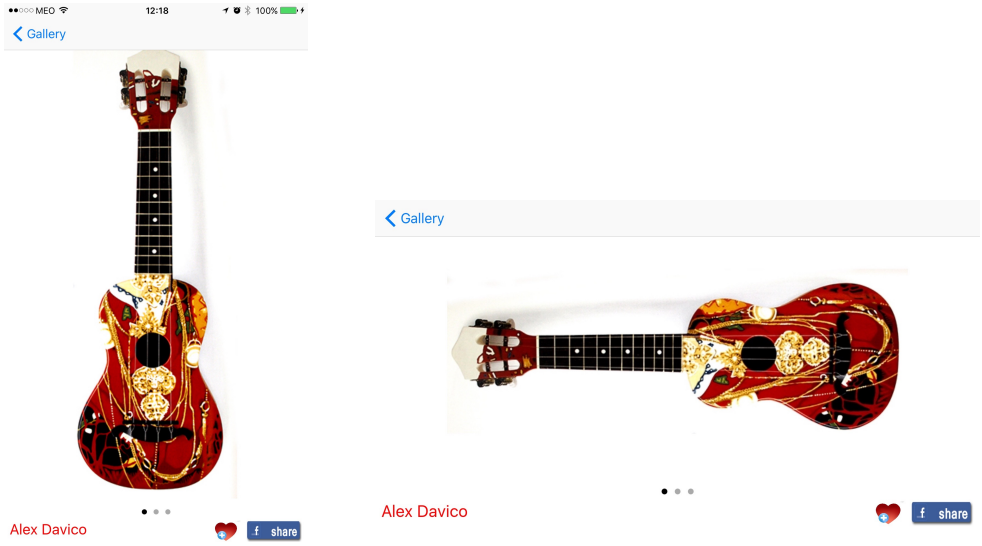


Figure 5.8: Item views of the developed iOS version.

We developed the gallery view based on the use of the ImageSlideshow[9] library, which uses the Alamofire[10] library to load the images using the image URL. For its implementation, it is only necessary to add a view to serve as slideshow extending the ImageSlideShow class. Then we configured some aspects about the slideshow and the images were able to be presented.

This page also allows a zoom in in any of the presented images. To achieve that, we defined a gesture recogniser to recognise the click on the image and a new controller. In this controller, it is opened a new view and the proper scales are dealt with through the updateMinZoomScaleForSize and updateConstraintsForSize methods, as can be seen in Listings 5.13.

```
private func updateMinZoomScaleForSize(size: CGSize) {
    let widthScale = size.width / imageView.bounds.width
    let heightScale = size.height / imageView.bounds.height
    let minScale = min(widthScale, heightScale)

    scrollView.minimumZoomScale = minScale
    scrollView.zoomScale = minScale
}

private func updateConstraintsForSize(size: CGSize) {

    let yOffset = max(0, (size.height - imageView.frame.height) / 2)
    imageViewTopConstraint.constant = yOffset
    imageViewBottomConstraint.constant = yOffset

    let xOffset = max(0,(size.width - imageView.frame.width) / 2)
    imageViewLeadingConstraint.constant = xOffset
    imageViewTrailingConstraint.constant = xOffset

    view.layoutIfNeeded()
}
```

Listing 5.13: Used methods to deal with the zoom of the image.

The final result can be seen in the Figure 5.9.

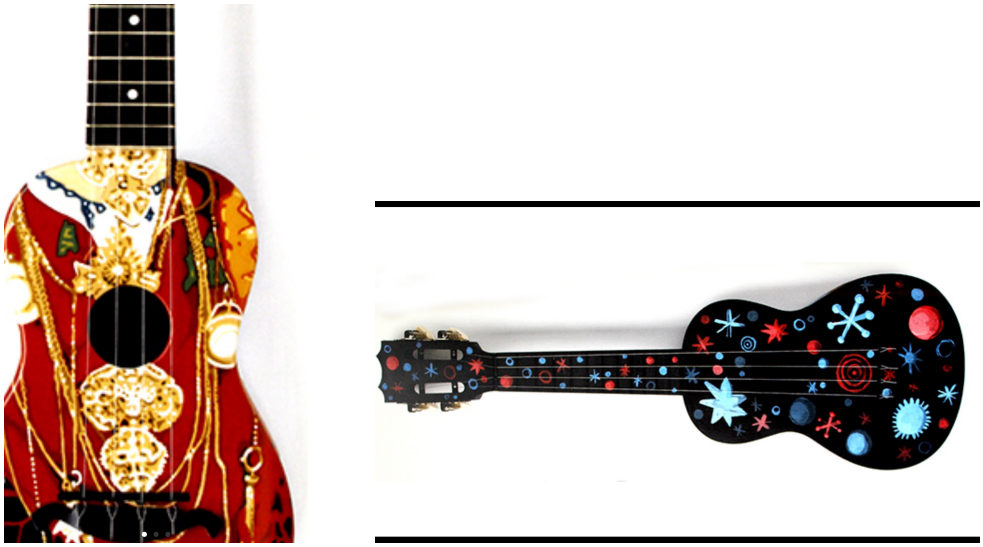


Figure 5.9: Zoom views of the developed iOS version.

In order to get a robust view able to be displayed in a device of any screen size, we defined some constraints. In this specific view, once it is prepared to be loaded, the height of this slideshow is defined to be equal to the minimum screen dimension minus 80 (found commitment).

We created both the Authors List and Favorites by using an expandable list view. This can be done by having a normal list view and adding a constraint defining the cell height. Then, when the cell is clicked on, it will open to a predefined height (Listing 5.14), in this case 200.

```
override func setSelected(selected: Bool, animated: Bool) {
    super.setSelected(selected, animated: animated)

    let constant: CGFloat = selected ? 200.0 : 0.0

    if !animated {
        descHeightConstraint.constant = constant
        expandablePart.hidden = !selected

        return
    }

    UIView.animateWithDuration(0.3, delay: 0.0, options: [.AllowUserInteraction, .
     BeginFromCurrentState], animations: {
        self.descHeightConstraint.constant = constant
        self.layoutIfNeeded()
        }, completion: { completed in
            self.expandablePart.hidden = !selected
    })
}
```

Listing 5.14: Excerpt of code used to apply the expandable list view.

In the Favorites page, the cell header will contain the object image while the title and description of the object appear only when clicked (Figure 5.10). One of the predefined behaviours for lists in iOS is the delete act, which is associated to a left swipe that shows a message of delete, as can be seen in Figure 5.10 on the right. Therefore, no remove button was necessary to implement for the removal of favorites. Once again, and in order to provide a proper execution in any device, we defined the image height programmatically as being a third of the screens biggest dimension minus 20.

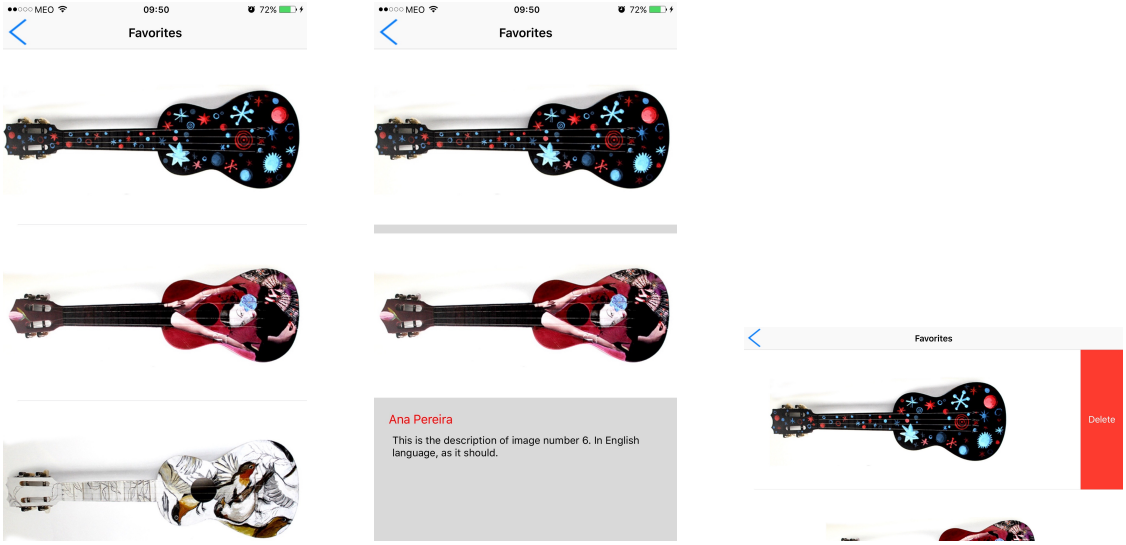


Figure 5.10: Favorites views of the developed iOS version.

Regarding the Authors List page, the header comprised the name of the author, while the text about him appear on the expandable part (Figure 5.11). The predefined remove swipe was disabled as it would make no sense in this case.

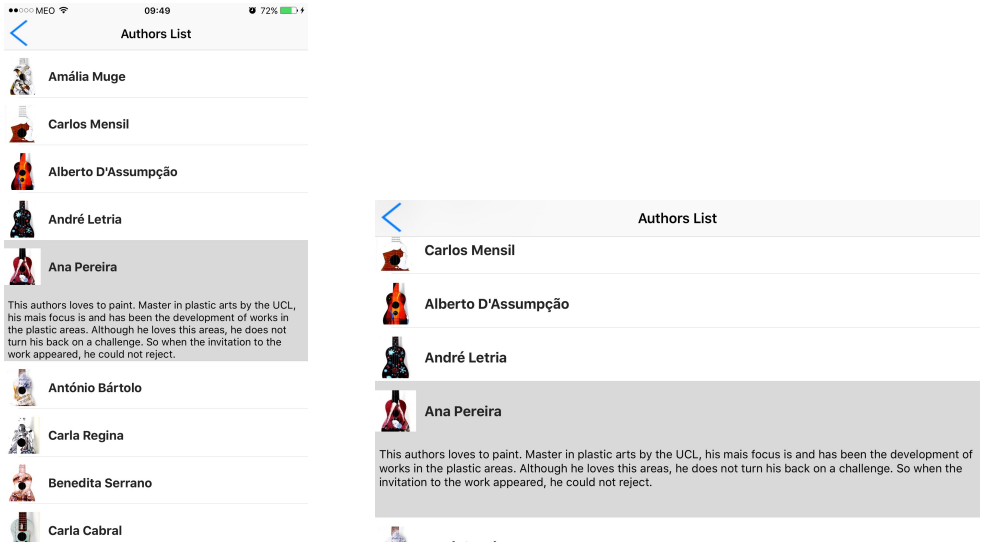


Figure 5.11: Zoom view of the developed iOS version.

In this iOS version, only the Firebase SDK was used with the database. The data can be managed with different types of events, as it is supposed to add, remove or update data. Listing 5.15 shows how this can be done, in this case, to get the images from the database.

```
func getImagesFromDB(splashView: SplashView) {
    var url = ""
    if NSLocale.currentLocale().objectForKey(NSLocaleLanguageCode)! as! String == "pt" {
        url = "https://radiant-inferno-748.firebaseio.com/images_pt/list"
    }
    else {
        url = "https://radiant-inferno-748.firebaseio.com/images/list"
    }

    // Firebase part
    let favs = Firebase(url: url)
    var count: Int = 0
    favs.observeEventType(.Value, withBlock: { snapshot in
        let json = JSON(snapshot.value)

        for (key,subJson):(String, JSON) in json {
            print("\(key) ->")
            var arrayOthers: Array<String> = []

            for (_,other):(String, JSON) in subJson["otherImages"] {
                arrayOthers.append(other["img"].stringValue)
            }

            let image = Image(title: subJson["title"].stringValue,
                id: subJson["id"].stringValue,
                description: subJson["description"].stringValue,
                image: subJson["imageUrl"].stringValue,
                otherImages: arrayOthers)

            print(image.getTitle())
            self.images[count] = image
            count += 1
        }

        splashView.startApp()

        }, withCancelBlock: { error in
            print(error.description)
    })
}
```

Listing 5.15: Excerpt of code used to get information from the Firebase.

Similarly to the Android version, we added a page with the options to remove/place the names alongside the objects in the collectionview as well as an option to download the images to internal storage. The download of the images will be done to files, being then loaded from the disk to the image view directly using the contentsOfFile function from UIImage.

We also implemented the Game option: a new page with a collection view of all objects and the partial image is shown. When establishing the correct correspondence, a new page is opened. Regarding the challenging of a friend, this option was not possible to be implemented, since the using of the Branch metrics required a service only available to paid developers. Nevertheless, the application is ready to include such feature.

Figure 5.12: Game view of the iOS version.

On the development of this application several libraries were used. A short description of each of them is presented in Table 5.2

Library	Description	Use in the application
ImageSlideshow	iOS / Swift image slideshow with circular scrolling, timer and full screen viewer.	Used in the zooming of images and in the viewpager logic.
Haneke	Haneke is a lightweight generic cache for iOS and tvOS written in Swift 2.0. It's designed to be super-simple to use.	Used in the loading of images from URLs.
SwiftyJSON	SwiftyJSON makes it easy to deal with JSON data in Swift.	Used to process JSON data.
Toast-Swift	Toast-Swift is a Swift extension that adds toast notifications to the UIView object class. It is intended to be simple, lightweight, and easy to use.	Used to construct toast messages.
Alamofire	Alamofire is an HTTP networking library written in Swift.	Used to load images to the viewpager.

Table 5.2: Libraries used in the iOS application development.

5.4 Backend Development

The backend of this project is implemented using Firebase, a database that stores data as JSON (JavaScript Object Notation). This is a lightweight data-interchange format, easy to read and write. It is built in two structures: in a collection of name/value pair, that is understood in several languages as an object, record, struct, dictionary, hash table, keyed table, keyed list or associate array; and in an ordered list of values, that is recognized in most languages as an array, vector, list or sequence. All modern languages support them in one way or another.

Using these structures allows constructing the most varied structures. In this project, several sets of data were needed, such as the *cavaquinho* list, information about the exhibition, the artists list, the users' registry in the application and the favorites associated with each one of them. For some of these objects, it is necessary to have as many replicated structures as supported languages by the application, with each of the structures with the right information in the correct language.

Regarding the images, we created two identical structures: one for the Portuguese language and another for the English one. The chosen language is then determined at run-time by the application and the right call is made to the database. This structure needs to keep an array of objects, each of them having an image URL, title, description, a unique id and a list of other images. This structure can be seen in Figure 5.13.

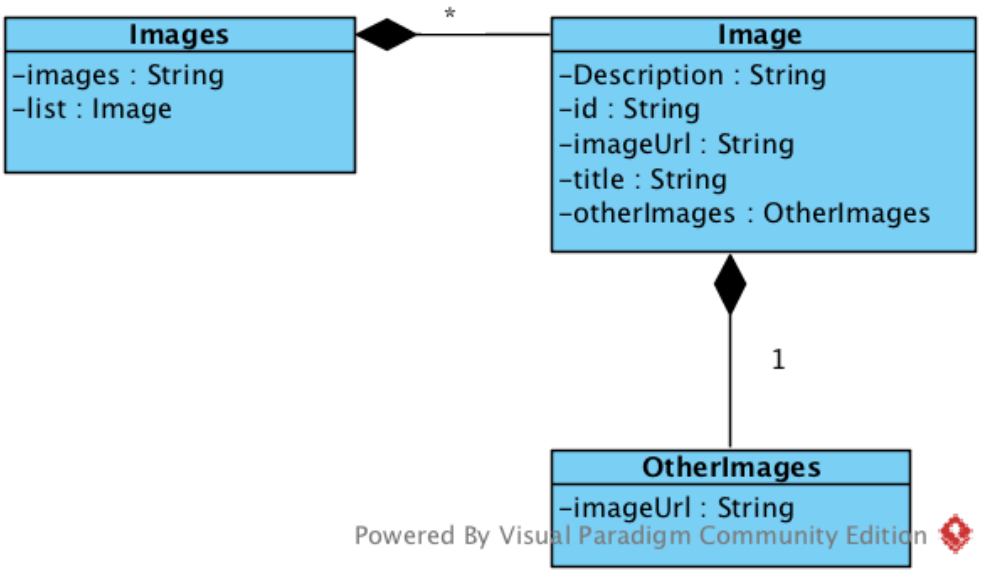


Figure 5.13: Representation of the objects list.

Additionally, there is a structure named images_pt exactly similar to the latter, however with information in the Portuguese language. This structure allows each item to change without the need of changing the code. Other fields can also be added, providing more information if needed.

The Authors List is also stored in the database. This list has a similar structure as the one used for the images, however instead of having an image URL, title, id and list of other images, each item has simply the author name and a short biography. The structure can be seen in Figure 5.14.

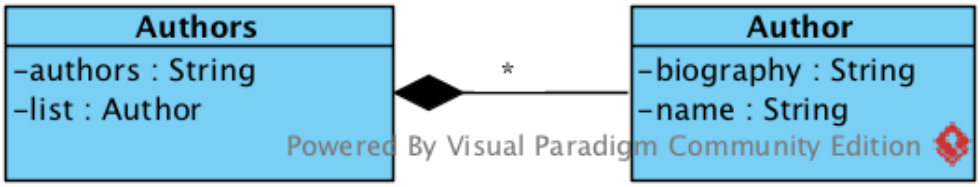


Figure 5.14: Representation of the authors list objects.

The Authors List also has the corresponding version in the Portuguese language: authors_pt.

The remaining information is stored in the database only when the user logs in in the app. Only after logging in, a field called users is created automatically by the Firebase, storing the information associated with that specific user, namely the display name and the provider, in this case Facebook. This data is stored under the corresponding unique user id, provided by Facebook.

The last piece of information to be stored is the user's Favorites. Once the user is logged in and has an associated id, his favorites start being stored under a favorites field. For each added favorite, the corresponding id and the main image URL are also stored, as can be seen from Figure 5.15.

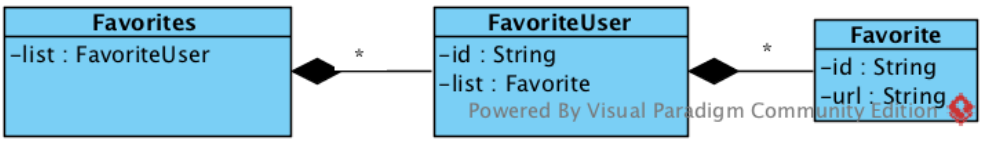


Figure 5.15: Representation of the user favorites.

Firebase also provides a section with security and rules that can be applied. In this case, the important thing is to guarantee that a user can only change his own favorites. For this, the rule applied was the one seen in Figure 5.16, which guarantees exactly that. No more critical information is stored in the database, as there was no necessity for applying a more restrictive policy.

```
"rules": {
  "users": {
    "$user_id": {
      // grants write access to the owner of this user account
      // whose uid must exactly match the key ($user_id)
      ".write": "$user_id === auth.uid"
    }
  },
    ".read": true,
    ".write": true
}
```

Figure 5.16: Firebase rules applied.

5.5 How to reuse the solution for other exhibitions

As it is, the application could be reused in a different exhibition, as long as the implemented interactions would make sense in the new scenario.

It would be required to create new tokens for the new application with respect to the cloud services used, as described in the following text, and to load the new Firebase instance with the appropriate data. Other artwork-centred exhibition could make specific deployments with modest effort, thought no administration interface was developed for this purpose.

To reuse this application with another content, a developer has to:

1. Change the desired content in the backend, maintaining the structure.
2. Change the app key of the Facebook SDK.
3. Change the app key of the Branch (only in the Android version).

If the developer wants to use a different Firebase database, he has to:

1. Change the Firebase link in the Data class.
2. Change the Firebase app key.
3. Change the app key of the Facebook SDK.
4. Change the app key of the Branch (only in the Android version).

This last case is only possible if the developer also keeps the structure of the already implemented database.

Chapter 6

Results and validation

In this section, we present the final results regarding the two applications developed in different platforms: one in Android and another in iOS.

This chapter also includes the tests performed to validate both applications: the compatibility tests, which prove that the application can run in devices with different screen dimensions and different software versions (to a point), and the usability tests, where feedback from people that used the application is presented.

6.1 Available Prototypes

6.1.1 Android

The presented Android prototype is also available in the Play Store, where it can be downloaded and fully tested.

When the user opens up the application, a splash screen first appears, in order to check if a connection is available or if the application has the needed information downloaded in the internal memory. If neither happens, a warning text is issued to the user informing that an Internet connection is required in order to execute the application. If one of the conditions is met, the app starts running. The main page is open and the user is presented with the object gallery (Figure 6.1 centre and right) and a menu, from which every functionality of the app can be accessed.

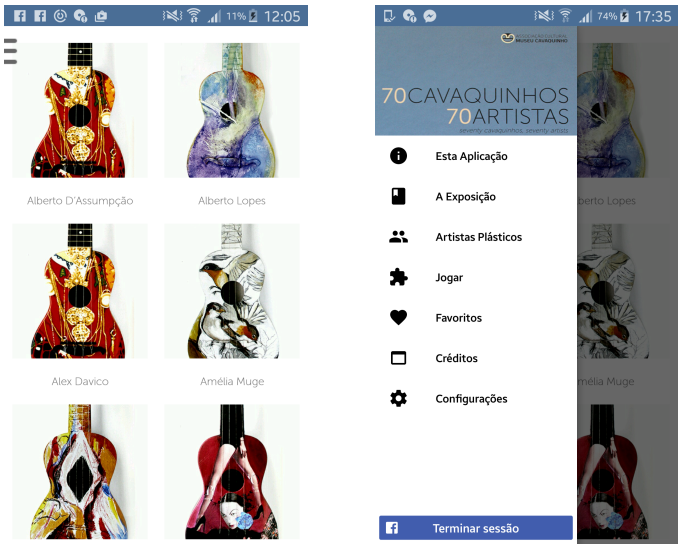


Figure 6.1: Gallery and menu views of the Android version.

The main feature is the view of a concrete item. The usage of this function follows the steps described in Figure 6.2: after the gallery view is exposed, a specific object is selected from the gallery, directing the user to a new page where more information is disclosed. Apart from observing the image in more detail (through a click in the images), specific buttons to access specific functionalities, namely Sharing of the object on Facebook, adding the object to Favorites or Challenging a friend, are shown.

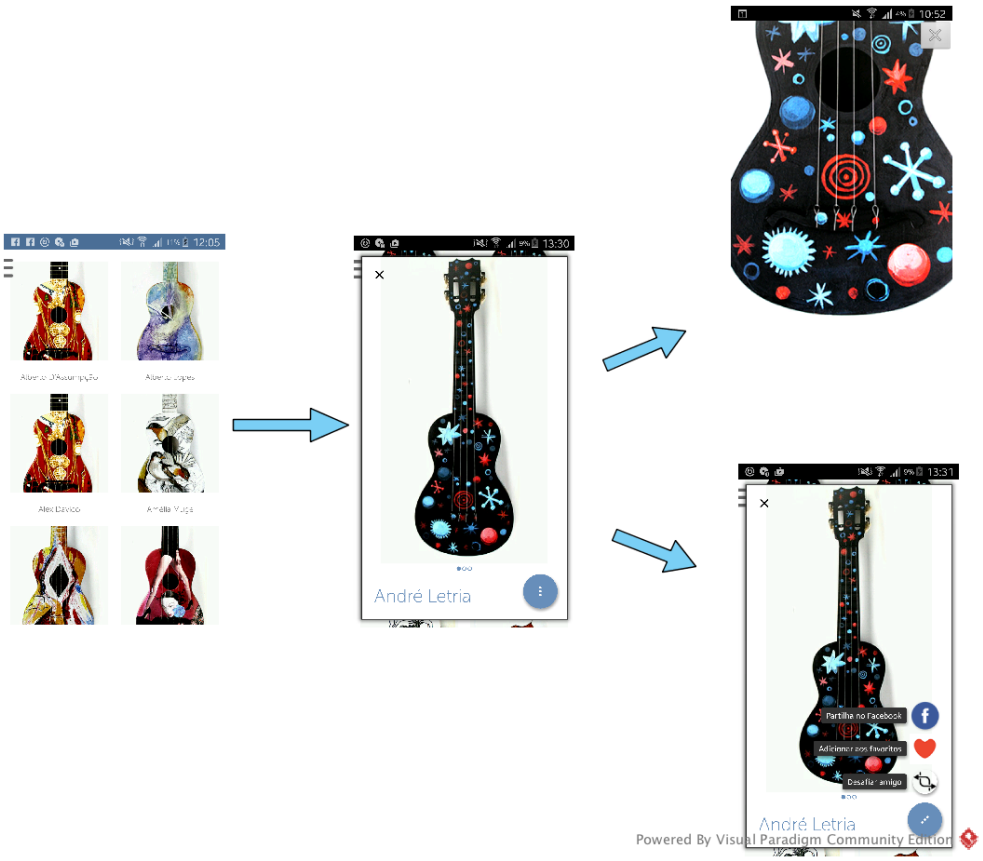


Figure 6.2: Workflow to view item and perform zoom on the Android app.

By choosing the sharing to Facebook option, the user is directed through a flow that can be seen in Figure 6.3, where a page first appears with the image loaded and where he can add a caption to the figure. The image can then be shared in a specific group, in a friend's feed or in his own wall.

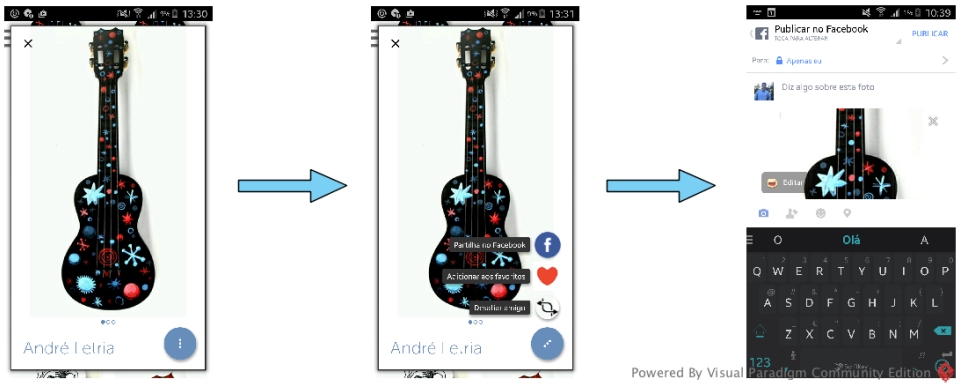


Figure 6.3: Workflow to share a picture of an object to Facebook on the Android app.

If the action to be performed corresponds to adding a specific object to the user's Favorites list, a message is shown informing the user that the image has been added to his favorites (Figure 6.4).

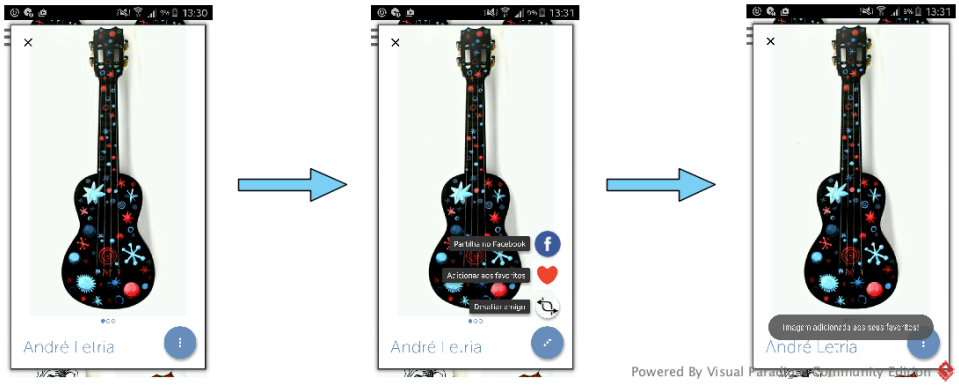


Figure 6.4: Workflow of adding an object to favorites on the Android app.

If, on the other hand, the user aims to Challenge a friend, the action flow becomes more complex. First, the user will have to define the portion of the figure to share, being able to not only crop the image but also rotate it. Afterwards, he will see the result and will have the option to share not only to a frien's wall on Facebook, as illustrated in Figure 6.5, but also by any means he wants to, as this share is made through a link.

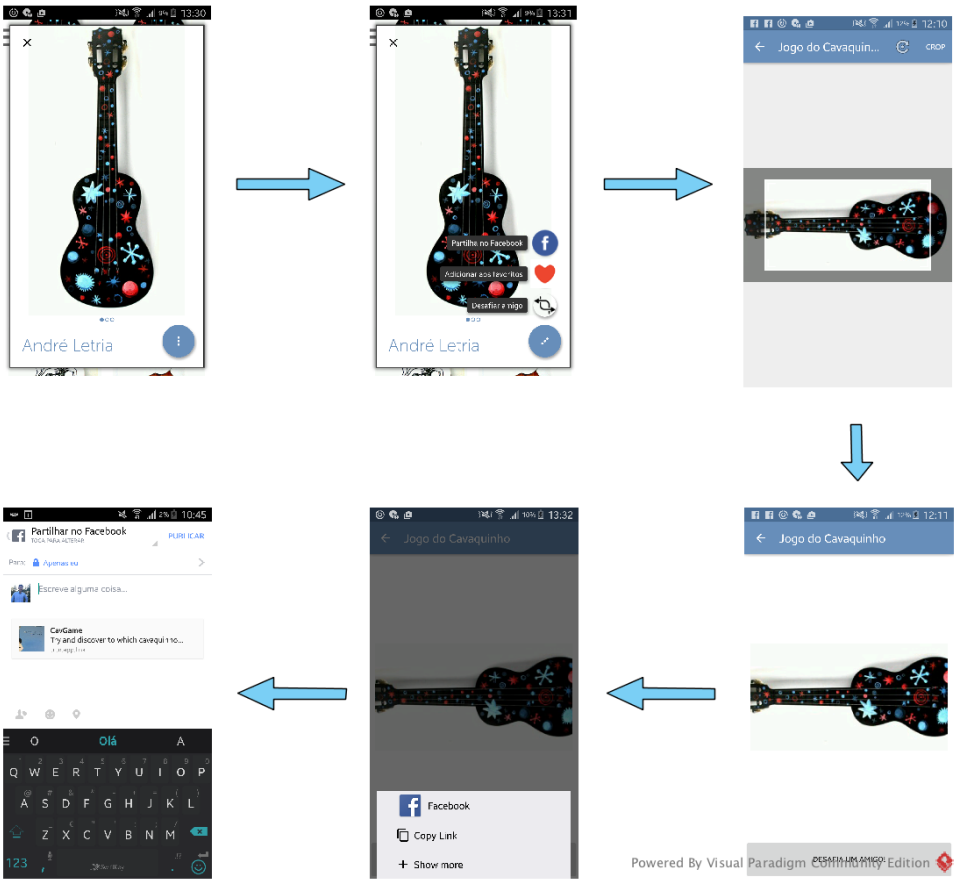


Figure 6.5: Workflow of challenging a friend on the Android app.

When in the main page, some other features are accessible through the main menu. One of the main features of this app is the possibility to read an object code, which allows the user to go directly to that objects page. As can be seen from Figure 6.6, this can be achieved by choosing the Read code option, which will open a view to read a QRcode. If a valid code is read, the corresponding page will open. As this feature is excluded from the final version, the flow in Figure 6.6 shows a menu without the Read Code option.

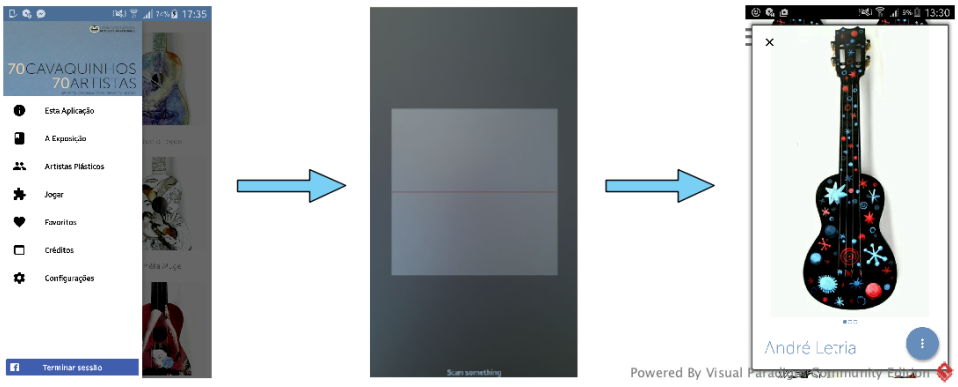


Figure 6.6: Workflow of reading a code to go directly to an object on the Android app.

Another of the main features is the possibility of playing a game (Figure 6.7). If chosen, this option will lead to another page where the user has all the images in a grid view and will have to drag and drop the portion of the image to the correct object. Once the correct answer is achieved, the corresponding page will open so the user can get more information about the item or simply return to the main gallery.

Figure 6.7: Workflow of Play a game on the Android app.

Other features accessible through the main menu include the Artists list, Favorites, The Exhibition, This Application, Favorites and Options. The first feature will present the user a simple page with the authors exposing their works in the exhibition and some information regarding them. The Favorites page will have the user favorites, with an image, text and the possibility of removing any item from that list. The The Exhibition page presents an overview of the exhibition and specific information regarding it. The This Application page will present the user with information about the application and the possibility of accessing two tutorials: one to learn how to use the application and another to learn how to challenge a friend. The Options page will present some configurations allowed: the possibility of having the main gallery without the authors' name and the possibility of downloading all the images and information to the device's internal storage, so that the application can run offline and with lower loading times. All these flows can be seen in the next Figure 6.8.

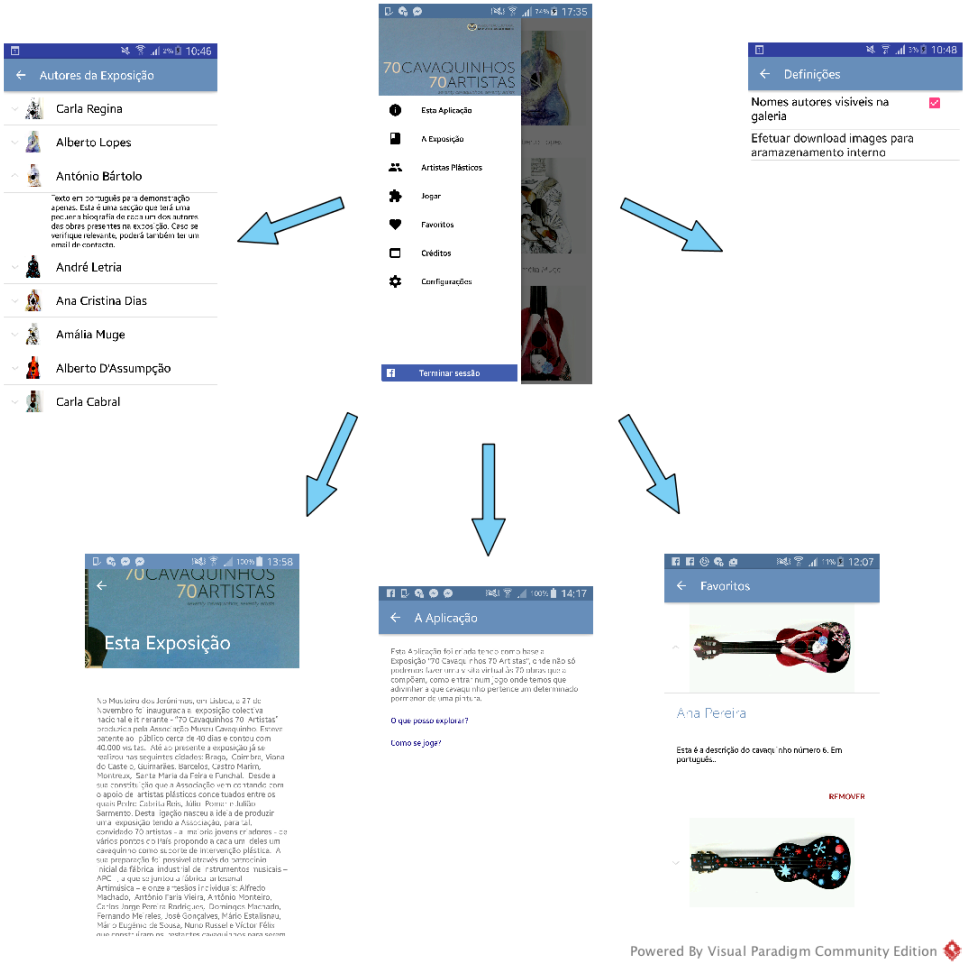


Figure 6.8: Workflow of the rest of the pages on the Android app.

6.1.2 iOS

The available iOS prototype is very similar to the Android one, as the application purpose is the same. We made some changes, as it will be perceived afterwards, since some aspects were impossible to be replicated in the iOS platform.

Similarly to the Android version, when the user opens up the iOS application, a splash screen first appears, in order to check if a connection is available or if the application has the needed information downloaded in the internal memory. If neither happens, a warning sign is issued to the user, indicating that an Internet connection is required in order to run the application. If one of the conditions is met, the app starts normally. The main page is then open and the user is presented with the object gallery (Figure 6.9) and a menu, from which the user can access every functionality of the app.

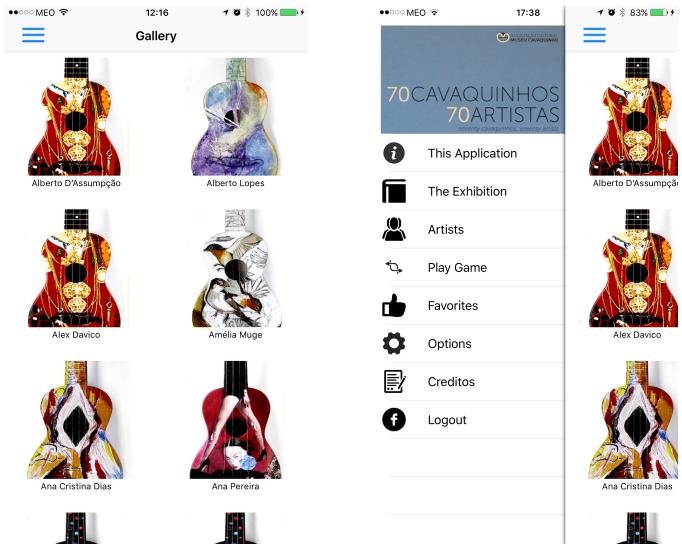


Figure 6.9: Gallery and menu views of the iOS version.

The main functional is the view of a concrete item. As can be seen in Figure 6.10, the flow is simple: the app is started showing the object gallery from which the user can choose a specific object to observe in more detail. In this new page, several images of the same object can be observed and more options regarding that object can be accessed, namely sharing on Facebook and adding that object to Favorites.

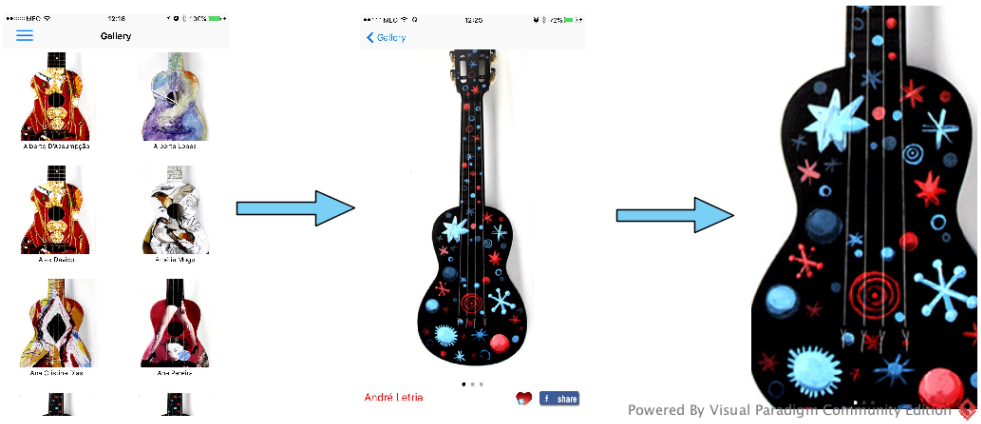


Figure 6.10: Workflow to view item and perform zoom on the iOS app.

When choosing the Share option, a new page opens up, allowing the user to add a caption and to choose where to share the object (feed, friends' feed, group page,). If the chosen option is add to users' favorites, a pop up warning appears informing him that that object has been added to his favorites. Both these flows can be seen in Figure 6.11.

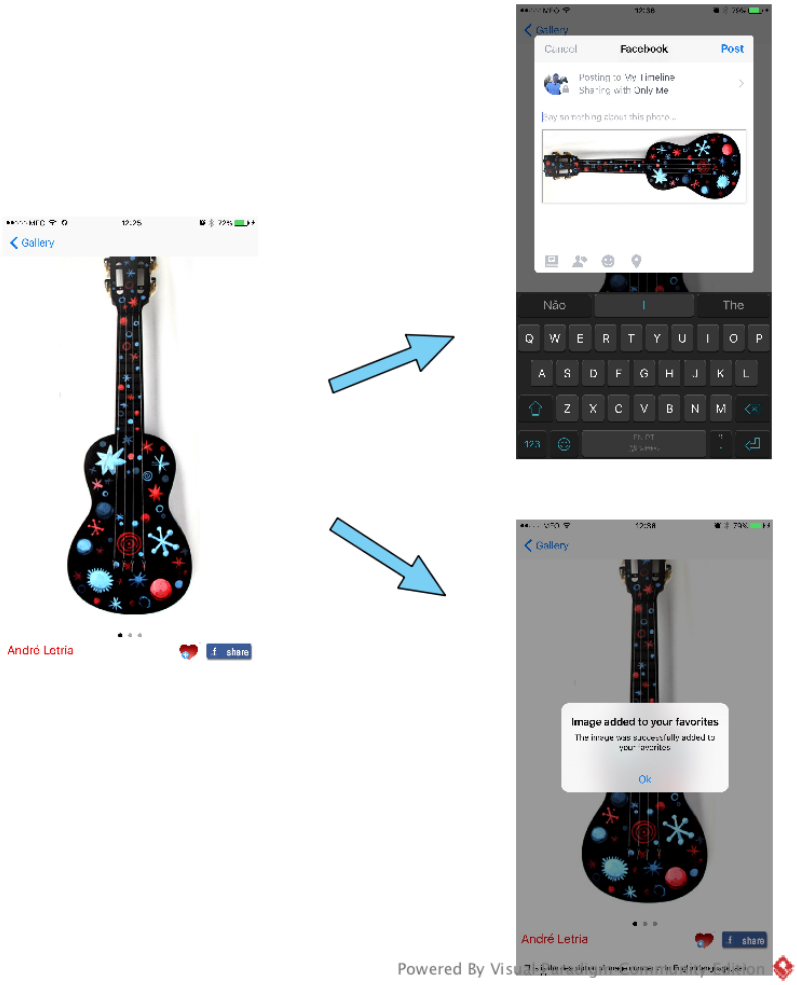


Figure 6.11: Workflow to share to Facebook and add an object to favorites on the iOS app.

The main page offers other features to access through the main menu. Once again, similarly to the Android version, the read of a QRcode is possible, allowing the user to go directly to the objects page. As can be seen from Figure 6.12, if present, this can be achieved by choosing the Read Code option, which will lead to a new window to read the code. If valid, the corresponding page will be opened. This feature was also excluded from this specific application, so the corresponding menu option is not present in Figure 6.12.

Figure 6.12: Workflow of reading a code to go directly to an object on the iOS app.

Although the Challenge friend option in not available to the iOS version, the play game feature was possible to be implemented. This option can be accessed from the main menu, and will lead the user to a page with a collection view, where he can click in the correct picture. Once this is achieved, a pop up appears congratulating the user, leading him to the gallery page. This working flow can be seen in Figure 6.13.

Figure 6.13: Workflow of reading a code to go directly to an object on the iOS app.

As in the Android version, this application also has other simple pages that can be accessed, namely the Authors list, Favorites, This Application, The Exhibition and Options page. The flow for each of these pages can be seen in Figure 6.14.

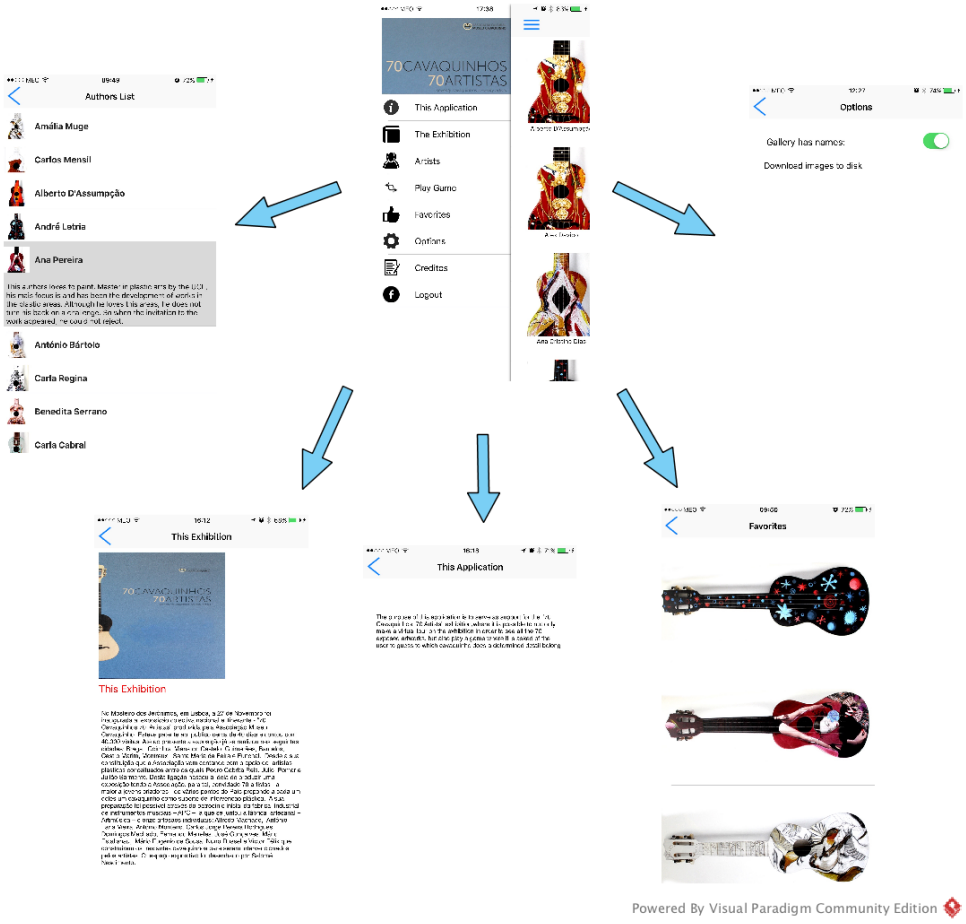


Figure 6.14: Workflow of the rest of the pages on the iOS app.

6.2 Compatibility tests

A crucial aspect of the application development is the compatibility tests. The good design and appealing features are neglected if the application can only run in the last released devices, with the last software versions or in small devices. The main goal of an application is to reach all its potential users, and therefore the application must be developed for as many versions and screen sizes as possible.

As the number of supported versions decreases, the possibility of the new features to be included in the application also decreases, since they are released only for the latest versions. In this context, a compromise had to be made, in order to have a good application with valuable features however compatible with the majority of the devices in the market.

6.2.1 Android

For the Android platform, the choice of a minimum API is a complex aspect. The market is full of devices running on different versions and for that reason successful applications should be developed to include at least 90% of the devices. To reach this number, API 15 was chosen, covering more than 95% of the users. This aspect is defined by choosing a minimum SDK in which the application will run, which in this case is the SDK 15 (Figure 6.15).

```
compileSdkVersion 23
buildToolsVersion "23.0.2"
defaultConfig {
    applicationId 'pt.deti.cavexh_v6'
    minSdkVersion 15
    targetSdkVersion 23
    versionCode 2
    versionName '2.0'
}
```

Figure 6.15: Android versions for which the application was developed.

Other aspect that needs consideration is the device size. The application needs to have a good design both in small and big devices, allowing the application to run in all smartphones and even in tablets. Table 6.1 shows some of the devices in which the application was successful tested and some relevant specifications of those devices.

Device	API	Target	Resolution(pixels)
Samsung Galaxy S3	16	4.4	720 x 1280
Samsung Galaxy S4	16	4.4	1080 x 1920
Nexus 10	16	4.1	2560 x 1600
Nexus One	16	4.0.3	480 x 800
BQ Aquaris E4	19	4.4.2	3264 x 2448
Samsung Galaxy S7	23	6.0	1440 x 2560

Table 6.1: Device specifications of the testing devices.

The next Figures show the application running in a small and big devices, maintaining the overall aspect.

Figures 6.16, 6.17 and 6.18 show the app running in Nexus 10 API 15.

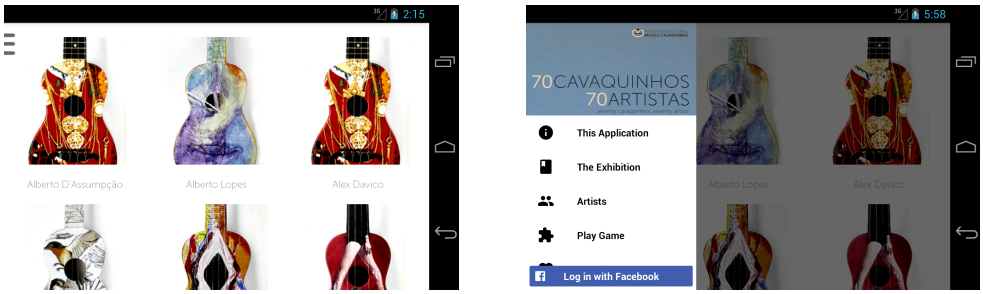


Figure 6.16: Gallery and menu views running in a Nexus 10, in landscape mode.

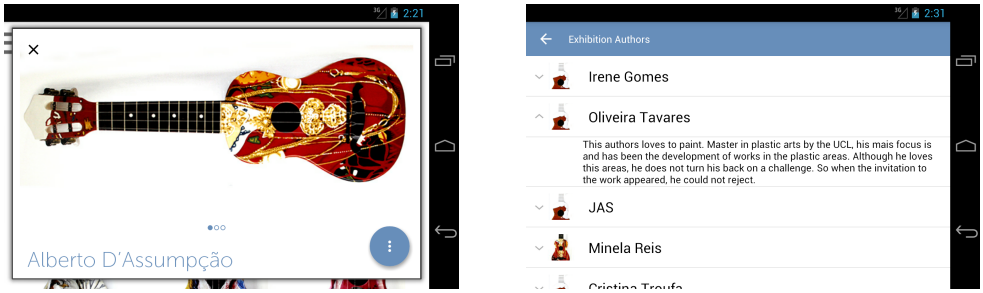


Figure 6.17: Detailed Item and Authors List views running in a Nexus 10, in landscape mode.

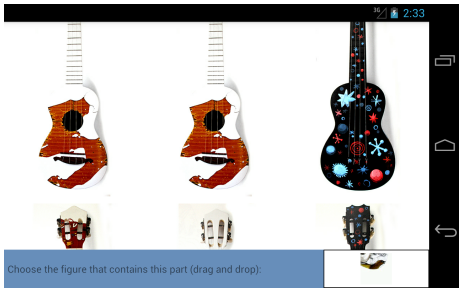


Figure 6.18: Play Game view running in a Nexus 10, in landscape mode.

Figures 6.19 and 6.20 show the app running in Nexus 0ne, API 16.

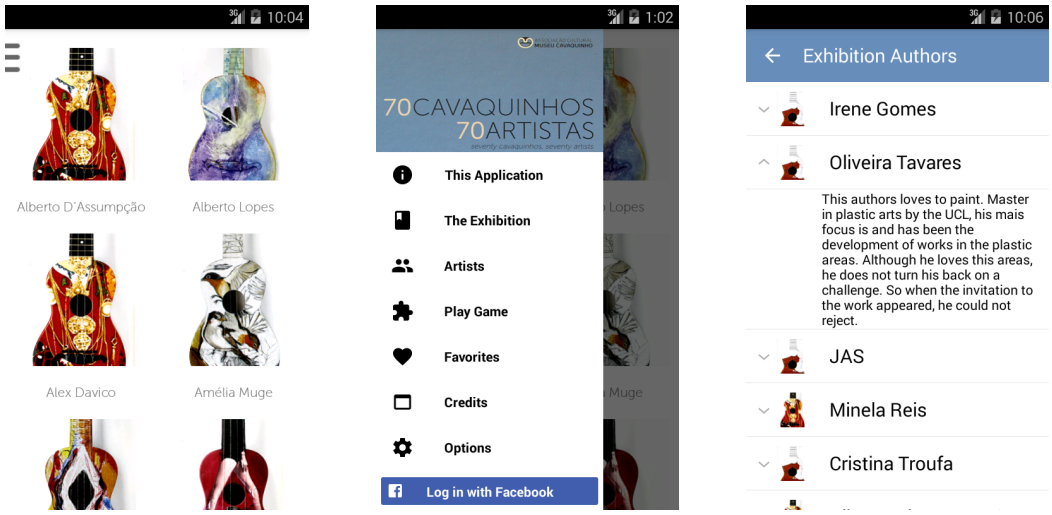


Figure 6.19: Gallery, Menu and Authors List views running in a Nexus One, in portrait mode.

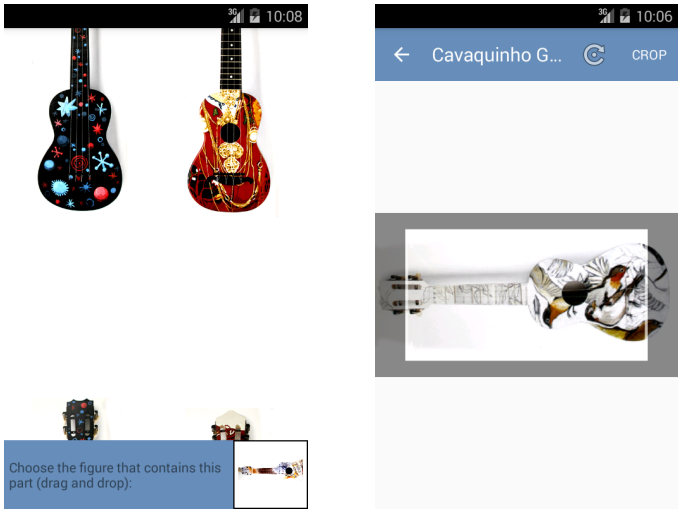


Figure 6.20: Challenge Friend and Play Game views running in a Nexus One, in portrait mode.

6.2.2 iOS

In opposite to the Android platform, software version in iOS is not considered a problem. Since it is a close and proprietary software, Apple updates their own devices and when a new version is released, it is available for almost all the devices within a week. More than 90% of the users is already in one of the last two released versions, 8.x or 9.x, therefore the application was developed for devices running in iOS 8.x or later.

Regarding the screen sizes, the solution when designing the iOS application was to make use of constraints. The steps taken in order to achieve this are described in the implementation section. Table 6.2 shows some of the devices in which the application was successful tested and some relevant specifications of those devices.

Device	iOS Version	Resolution(pixels)
iPhone 6s Plus	9.3	1242 2208
iPhone 4s	8.0	640 x 960
iPhone 5	9.0	640 x 1136
iPad Mini	8.0	1536 x 2048
iPad Pro	9.0	2048 x 2732

Table 6.2: Device specifications of the testing devices.

On the next Figures, it is shown the application running on an iPad and on an iPhone 4s, the smaller one available.

Figures 6.21 and 6.22 show the app running on iPad.

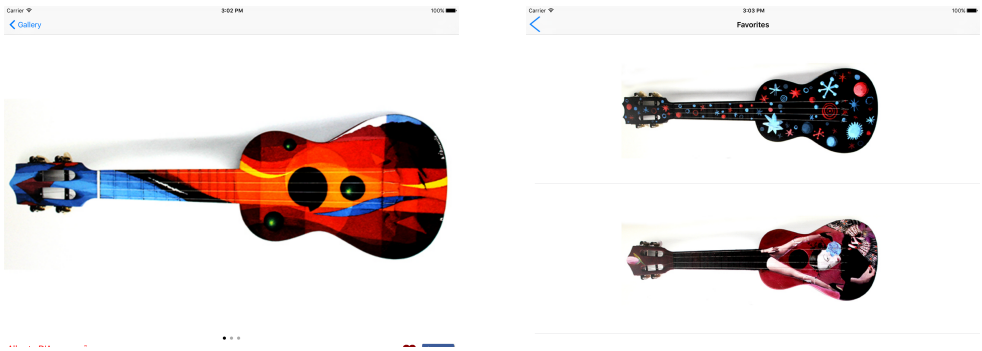


Figure 6.21: Item and Favorites views running on iPad, in landscape mode.

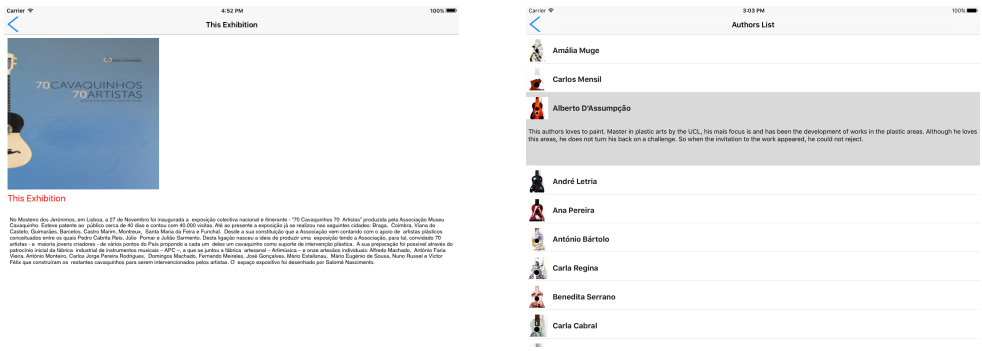


Figure 6.22: The Exhibition and Authors List views running on iPad, in landscape mode.

Figures 6.23 and 6.24 showing the app running on iPhone 4s.

Figure 6.23: Gallery, Menu and Favorites List views running in iPhone 4s, in portrait mode.

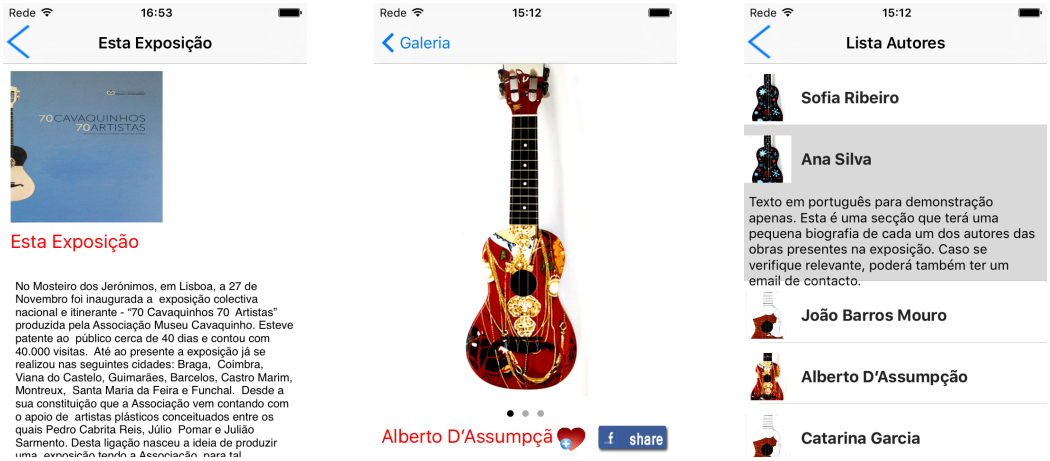


Figure 6.24: The Exhibition, Item and Authors List views running in iPhone 4s, in portrait mode.

6.3 Pilot Use

The usability tests are performed in order to perceive if the application fulfils all the requisites: if it is self-explanatory, if it performs what is supposed to, if it achieves the goal it was designed to. In order to do this, users have to try the app and give feedback regarding their usability and performance, in order to detect flaws in the design and interaction.

For this purpose, the Android application was introduced in the Play Store as a beta testing application (https://play.google.com/apps/testing/pt.deti.cavexh_v6). In a first phase, some people were invited to use the application; in a second phase, a link was generated for everyone with access to it to be able to download and use the application. For the pilot use, a series of questions were prepared (Table 6.3) in order to fully characterize the performance of the application, at the same time providing valuable feedback on the application usability.

These tests were made in a first phase, where the feedback was important to improve and add some features. The asked questions were the following.

Scope:	Number	Android Application
Purpose:	1	Can users navigate the gallery?
	2	Can users select an Object
	3	Can users navigate through the image gallery inside the detailed view?
	4	Can users navigate across detailed views (left and right swipe in detailed view)
	5	Can users share an object on Facebook?
	6	Can users add an object to their Favorites?
	7	Can users open an object through QR code?
	8	Can users access the The Exhibition page?
	9	Can users access the Favorites page?
	10	Can users access the Authors list page?
	11	Can users make login?
	12	Can users remove an object from their Favorites list?
	13	Can users see more information about an author by clicking on his name?
	14	Can users see more information about a favorite by clicking on it?
	15	Can users make logout?
	16	Can users Challenge a friend?
	17	Can users play the provided game?

Table 6.3: Questions asked to the testing users.

Besides these questions, each user has a section to provide additional feedback, in case they have suggestions. Some parameters appear without answer because either the features were not yet included at the time of the test or were later removed.

The results can be seen in Table 6.4.

Number:	User 1	User 2	User 3	User 4
1	Yes	Yes	Yes	Yes
2	Yes	Yes	Yes	Yes
3	Yes	Yes	Yes	Yes
4	Yes	Yes	Yes	Yes
5	Yes	Yes	Yes	Yes
6	Yes	Yes	Yes	Yes
7	Yes	Yes	-	-
8	Yes	Yes	Yes	Yes
9	Yes	Yes	Yes	Yes
10	Yes	Yes	Yes	Yes
11	Yes	Yes	Yes	Yes
12	Yes	Yes	Yes	Yes
13	Yes	Yes	Yes	Yes
14	Yes	Yes	Yes	Yes
15	Yes	Yes	Yes	Yes
16	-	-	Yes	Yes
17	-	-	Yes	Yes
Feedback:	- Improve Layout Design - Possibility of navigation in offline mode	- Possibility of navigation in offline mode	- Improve Favorites and Authors List	-

Table 6.4: Questions asked to the Android testing users.

This pilot tests allowed to perceive that the flow of the application was being fully understood by the users. The focus of improvement was in the user interface of the application. Later with the publishing of the application as open beta in the Play Store, the feedback started to be received via email.

Concerning the test and usage of the application, we integrated Fabric[1], a mobile platform with modular kits that can be mixed and matched to build the best apps. This platform allows the detection of crashes in the users' usage, sending a report with the specific error that occurred, helping this way in the correction of possible bugs. This platform also provides a section with statistical usage, as the number of users, session times, among others.

Concerning the iOS application, due to impossibility of publish the application in the App Store (no premium account), no beta testing was possible to be accomplished. Only some tests were possible to be made physically on the developers' device, as the latter is the only device where the application runs without being on the App Store. The same questionnaire was applied, with the results shown in Table 6.5:

[1] https://fabric.io

Number:	User 1	User 2	User 3
1	Yes	Yes	Yes
2	Yes	Yes	Yes
3	Yes	Yes	Yes
4	Yes	Yes	Yes
5	Yes	Yes	Yes
6	Yes	Yes	Yes
7	Yes	Yes	Yes
8	Yes	Yes	Yes
9	Yes	Yes	Yes
10	Yes	Yes	Yes
11	Yes	Yes	Yes
12	Yes	Yes	Yes
13	Yes	Yes	Yes
14	Yes	Yes	Yes
15	Yes	Yes	Yes
16	-	-	-
17	-	-	Yes
Feedback:	- Improve Layout Design - Possibility of navigation in offline mode	- Possibility of navigation in offline mode	- Improve Favorites and Authors List

Table 6.5: Questions asked to the iOS testing users.

Chapter 7

Conclusions

In this chapter, the conclusions of this dissertation are described as well as what could be achieved and what could not be implemented. Some ideas regarding future work are also addressed, in order to better complement or improve the obtained work.

7.1 Developed project

This dissertation project comprises the development of an application to provide digital support for the "70 Cavaquinhos, 70 Artistas" exhibition. This application aims to increase the interactivity between the visitors and the exhibition through the implementation of dynamic and informative features. In order to reach as many users as possible, this application was developed in different platforms, namely Android and iOS, since they represent almost all the users in the market.

Being aware of other applications already in the market, with the same purpose, the need of developing an application that followed the guidelines for museum/exhibitions was key to the successful development of the app. Therefore, some general features were included in the application, such as the way to represent the objects in the exhibition, a specific page with information regarding the exhibition's history, details about the authors or the possibility of adding favorites to the user pages.

New elements not commonly seen in similar applications were implemented, e.g.; the possibility of reading id tokens to direct the user to a specific content. The share of an image on Facebook was another feature included, in order to not only allow publishing a simple object on a friend's wall, but also to serve as a way to disseminate the exhibition and the application itself.

The major increment in the application was the possibility to play a game, in order to increase the interactivity between the visitor and the exhibition. The possibility of challenging a friend, even if with a device from another mobile operating system, allows the users to enjoy the exhibition. It helps once again the spreading of the exhibition and the application. The possibility of playing a game without being challenged is another feature that was accomplished.

The images play a crucial role in the smooth execution of the application. The developed system allows the execution of the application with or without making the download of the information and images to the internal storage of the device, with advantages and disadvantages associated with both. Another aspect that can also be configured by the user is the possibility of having a gallery with or without the names on it.

The application is very similar both in Android and iOS platforms, even if some differences can be noticed. Due to the lack of an Apple premium account, the challenging of a friend feature was excluded of the iOS version. Apart from that, the same elements are present, even if in some cases, not in the same way.

It is, nevertheless, important to notice that even though this application was developed to serve as support of the "70 Cavaquinhos, 70 Artistas" exhibition, the process followed during the development of the app allows it to be easily adapted for different exhibitions, just by changing the information and images on the backend.

7.2 Future Work

The requirements of the project were met, but improvements can always be made, through the addition of a new functionality or through the enhancement of an existing one.

One example consists on the possibility of using a beacon as replacement or complement for the QR code reader. With this strategy, the user could just pass near the artwork in the exhibition and be redirected to the specific object. The use of the camera to recognize the object is another possibility.

The use of virtual and augmented reality can also be explored in order to provide the visitor with a more intense experience of the exhibition. Adding a map as a 3D model to allow the possibility of navigating the exhibition while not physically present in the applications is another aspect to consider.

The iOS version could also be improved in some aspects when compared with the Android version. With a paid developer account, access to some premium features would be possible, allowing the integration of the challenging of a friend in order to allow other users to play the game.

References

[1] Y.-K. Lee, C.-T. Chang, Y. Lin, and Z.-H. Cheng, "The dark side of smartphone usage: Psychological traits, compulsive behavior and technostress," *Computers in Human Behavior*, vol. 31, pp. 373–383, 2014.

[2] N. M. Suki and N. M. Suki, "Dependency on smartphones: An analysis of structural equation modelling," *Jurnal Teknologi*, vol. 62, no. 1, 2013.

[3] T. Louis. How much do average apps make? [Online]. Available: http://www.forbes.com/sites/tristanlouis/2013/08/10/how-much-do-average-apps-make

[4] A. C. e Museu Cavaquinho. Museu cavauinho exhibition-web. [Online]. Available: http://www.cavaquinhos.pt/en/Exhibition-Space.htm

[5] M. Kenney and B. Pon, "Structuring the smartphone industry: Is the mobile internet os platform the key?" *Journal of Industry, Competition and Trade*, vol. 11, pp. 239–261, 2011.

[6] M. Boulos, S. Wheeler, C. Tavares, and R. Jones, "How smartphones are changing the face of mobile and participatory healthcare: an overview, with example from ecaalyx," *BioMedical Engineering OnLine*, 2011.

[7] M. Cooper, R. Dronsuth, A. Leitich, J. Lynk, J. Mikulski, J. Mitchell, R. Richardson, and J. Sangster, "Radio telephone system," Sep. 16 1975, uS Patent 3,906,166. [Online]. Available: http://www.google.com/patents/US3906166

[8] V. Shannon. 15 years of text messages, a 'cultural phenomenon'. [Online]. Available: http://www.nytimes.com/2007/12/05/technology/05iht-sms.4.8603150.html?pagewanted=all

[9] A. Elmenthaler. Hagenuk mt-2000 with tetris. [Online]. Available: https://web.archive.org/web/20130617124048/http://www.handy-sammler.de/Handys/Hagenuk_MT-2000.htm

[10] A. P. Info. Apple reinvents the phone with iphone. [Online]. Available: http://www.apple.com/pr/library/2007/01/09Apple-Reinvents-the-Phone-with-iPhone.html

[11] Android. Android history. [Online]. Available: https://www.android.com/history

[12] IGN. Windows phone history. [Online]. Available: http://www.ign.com/wikis/windows-phone/History

[13] A. P. Info. Apple launches ipad. [Online]. Available: http://www.apple.com/pr/library/2010/01/27Apple-Launches-iPad.html

[14] Statista. Global market share held by the leading smartphone operating systems in sales to end users from 1st quarter 2009 to 1st quarter 2016. [Online]. Available: http://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/

[15] Gartner. Gartner. [Online]. Available: http://www.gartner.com/newsroom/id/3215217

[16] A. Annie. App annie 2015 retrospective monetization opens new frontiers. [Online]. Available: http://blog.appannie.com/app-annie-2015-retrospective/

[17] Statista. Number of apps available in leading app stores as of july 2015. [Online]. Available: http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

[18] A. Hillegass and M. Ward, *Objective-C Programming: The Big Nerd Ranch Guide*, ser. Second Edition. Pearson Technology Group, 2011.

[19] A. Inc., *The Swift Programming Language (Swift 2.2)*, ser. First Edition. Apple Inc., 2014.

[20] S. G. Kochan, *Programming in Objective-C*, ser. Sixth Edition. Addison Wesley, 2014.

[21] A. Developer. Xcode. [Online]. Available: https://developer.apple.com/xcode/

[22] JetBrains. Appcode. [Online]. Available: https://www.jetbrains.com/objc/

[23] A. Developer. App store. [Online]. Available: https://developer.apple.com/support/app-store/

[24] R. Meier, *Professional Android 4 Application Development*, ser. Third Edition. Wrox, 2012.

[25] A. O. S. Project. Android open source project. [Online]. Available: https://source.android.com/

[26] D. Android. Dashboards. [Online]. Available: https://developer.android.com/about/dashboards/index.html

[27] S. Medi and N. Pavlovi, "Mobile technologies in museum exhibitions," *TURIZAM*, vol. 18, no. 4, pp. 166–174, 2014.

[28] F. Palumbo, G. Dominici, and G. Basile, "Designing a mobile app for museums according to the drivers of visitor satisfaction," *Recent Advances in Business Management and Marketing*, pp. 1–8, 2013.

[29] P. Kotler, "Marketing management: Analysis, planning, implementation and control," *Prentice Hall*, 1991.

[30] K. McLean, "Planning for people in museum exhibitions," *Malloy Lithographing*, 1996.

[31] S. McArthur and C. Hall, "Heritage management in australia and new zealand. the human dimensions," *Oxford University Press*, 1996.

[32] J. Donald, "The measurement of learning in the museum," *Canadian Journal of Education*, vol. 16, no. 3, pp. 371–382, 1991.

[33] F. G. Filip, "Information technologies in cultural institutions," *Studies in Informatics and Control*, vol. 6, no. 14, pp. 385–400, 1996.

[34] C. Ciurea, C. Coreriu, and C. Tudorache, "Implementing mobile applications for virtual exhibitions using augmented reality," *Journal of Mobile, Embedded and Distributed Systems*, vol. 6, no. 3, 2014.

[35] M. Patel, M. White, K. Walczak, and P. Sayd, "Digitization to presentation - building virtual museum exhibitions," *Proceedings of International Conference on Vision, Video and Graphics*, 2003.

[36] C. Ciurea, A. Zafiroiu, and A. Grosu, "Implementing mobile virtual exhibition to increase cultural heritage visibility," *Informatica Economic*, vol. 18, no. 2, pp. 24–31, 2014.

[37] C. Ciurea and C. Tudorache, "New perspectives on the development of virtual exhibitions for mobile devices," *Economy Informatics*, vol. 14, no. 1, pp. 31–38, 2014.

[38] M. cavaquinho. Exhibition 70 cavaquinhos 70 artistas. [Online]. Available: http://www.cavaquinhos.pt/en/Exhibition-Space.htm

[39] S. Ambler, *The elements of UML 2.0 Style*, ser. First Edition. Cambridge University Press, 2005.

[40] E. Freeman and E. Freeman, *Head First Design Patterns*, ser. First Edition. O'reilly, 2004.

[41] S. Judd. Glide. [Online]. Available: https://github.com/bumptech/glide

[42] H. Pique, J. Romano, L. Ascorbe, and O. Blanc. Haneke. [Online]. Available: https://github.com/Haneke/HanekeSwift

[43] Google. Google design. [Online]. Available: https://design.google.com/

Annex - Use Cases Specification

In this attachment the application use cases are described in more detail.

Use Case 1 - View *Cavaquinhos* Gallery

Name:	**View *Cavaquinhos* Gallery**
Actor	User
Pre-Conditions	An Internet connection must be established
Sequence of Events	1. The user opens the application 2. The gallery appears to the user
Alternative Sequence	

Table 7.1: Description of the View *Cavaquinhos* Gallery use case.

Use Case 2 - View Specific *Cavaquinho*

Name:	**View Specific *Cavaquinho***
Actor	User
Pre-Conditions	An Internet connection must be established
Sequence of Events	1. The user opens the application 2. The gallery appears to the user 3. The user clicks in one of the *cavaquinhos* 4. A new page with that specific *cavaquinho* is open
Alternative Sequence	1. The user opens the application 2. The user opens the menu 3. The user chooses the "Read code" option 4. The user scans the respective code 5. A new page with the specific *cavaquinho* is open

Table 7.2: Description of the View Specific *Cavaquinho* use case.

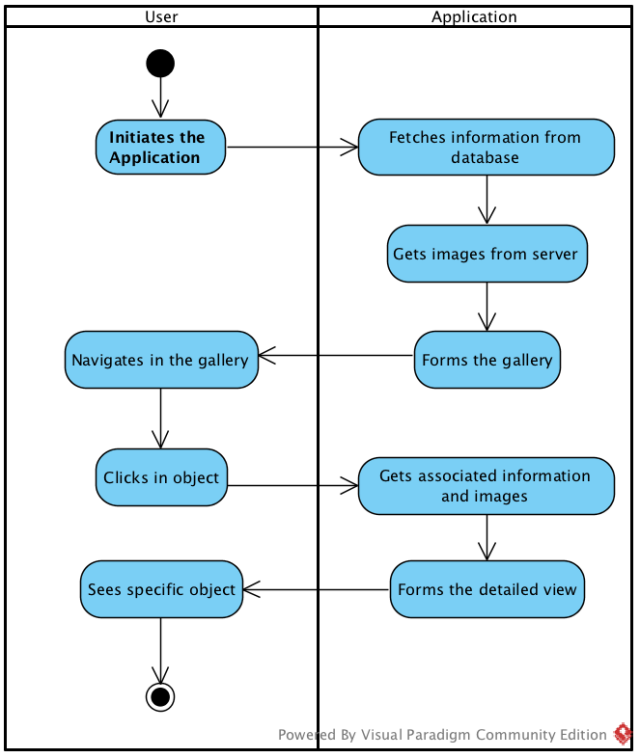


Figure 7.1: Activity diagram for the View Specific *Cavaquinho* use case.

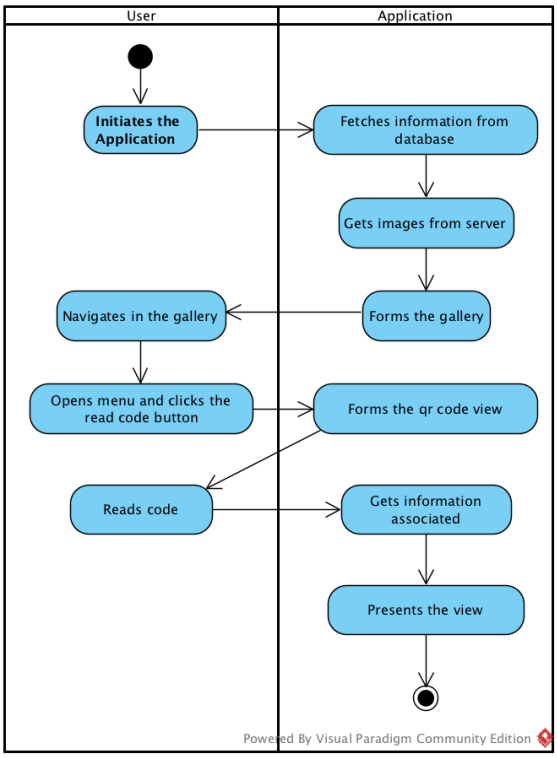


Figure 7.2: ALternative sequence diagram for the Read *Cavaquinho* Code use case.

Use Case 3 - Share *Cavaquinho* to Facebook

Name:	**Share *Cavaquinho* to Facebook**
Actor	User
Pre-Conditions	An Internet connection must be established The user must own a Facebook account The device must have the Facebook native application installed
Sequence of Events	1. The user opens the application 2. The gallery appears to the user 3. The user clicks in one of the *cavaquinhos* 4. A new page with that specific *cavaquinho* is open 5. The user clicks in the Share button 6. A new page is presented with the image loaded to add a description 7. The user confirms the share
Alternative Sequence	1. The user opens the application 2. The user opens the menu 3. The user chooses the "Read code" option 4. The user scans the respective code 5. A new page with the specific *cavaquinho* is open 6. The user clicks in the Share button 7. A new page is presented with the image loaded to add a description 8. The user confirms the share

Table 7.3: Description of the Share *Cavaquinho* to Facebook use case.

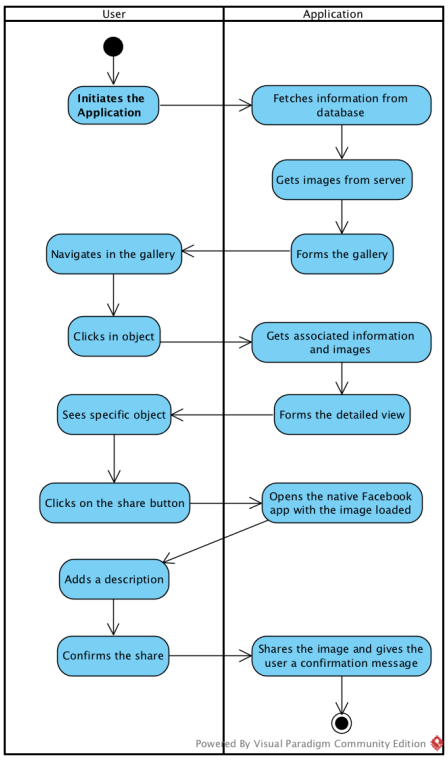


Figure 7.3: Activity diagram for the Share *Cavaquinho* to Facebook use case.

Use Case 4 - See Favorites

Name:	**See Favorites**
Actor	User
Pre-Conditions	An Internet connection must be established The user must be logged in in the application
Sequence of Events	1. The user opens the application 2. The user opens the menu 3. The user chooses the "Favorites" option 4. A new page with the user favorites is open
Alternative Sequence	

Table 7.4: Description of the See Favorites use case.

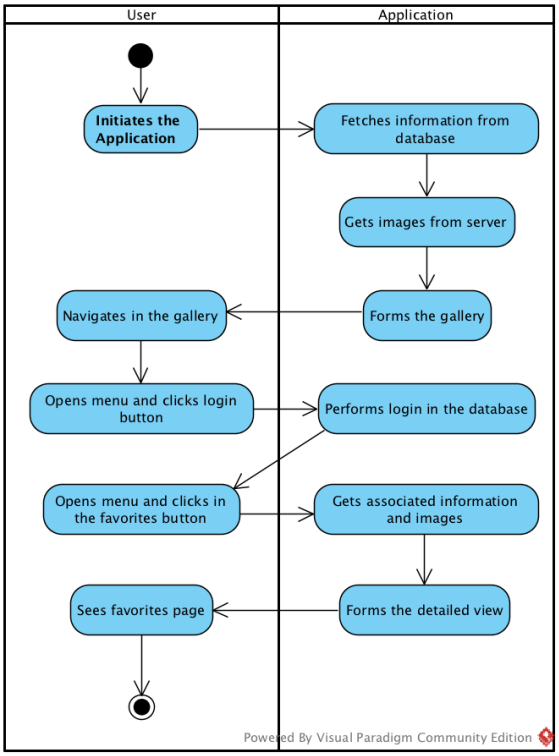


Figure 7.4: Activity diagram for the See Favorites use case.

Use Case 5 - View information about author

Name:	**View Information about Author**
Actor	User
Pre-Conditions	An internet connection must be established
Sequence of Events	1. The user opens the application 2. The user opens the menu 3. The user chooses the Authors List option 4. A new page with the authors information is open
Alternative Sequence	

Table 7.5: Description of the View Information about Author use case.

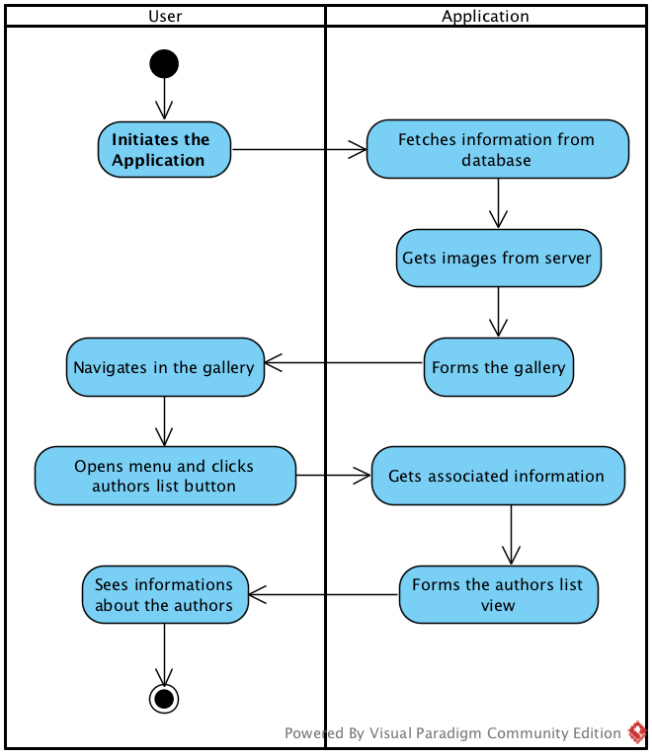


Figure 7.5: Activity diagram for the View Information about Author use case.

Use Case 6 - Edit Favorite (add action)

Name:	**Edit Favorite (add action)**
Actor	User
Pre-Conditions	An Internet connection must be established The user must be logged in in the application
Sequence of Events	1. The user opens the application 2. The gallery appears to the user 3. The user clicks in one of the *cavaquinhos* 4. A new page with that specific *cavaquinho* is open 5. The user clicks in the add to favorites button 6. The *cavaquinho* is added to the user favorites
Alternative Sequence	1. The user opens the application 2. The user opens the menu 3. The user chooses the "Read code" option 4. The user scans the respective code 5. A new page with the specific *cavaquinho* is open 6. The user clicks in the Add to Favorites button 7. The *cavaquinho* is added to the user favorites

Table 7.6: Description of the Edit Favorite use case (add action).

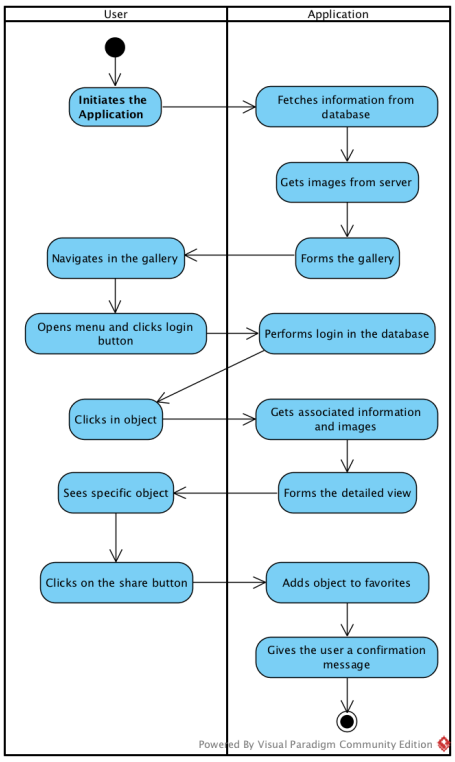


Figure 7.6: Activity diagram for the Add Favorite use case.

Use Case 6 - Edit Favorite (remove action)

Name:	**Edit Favorite (remove action)**
Actor	User
Pre-Conditions	An Internet connection must be established The user must be logged in in the application
Sequence of Events	1. The user opens the application 2. The user opens the menu 3. The user chooses the Favorites option 4. A new page with the users favorites is open 5. The user chooses one to remove
Alternative Sequence	

Table 7.7: Description of the Edit favorite use case (remove action).

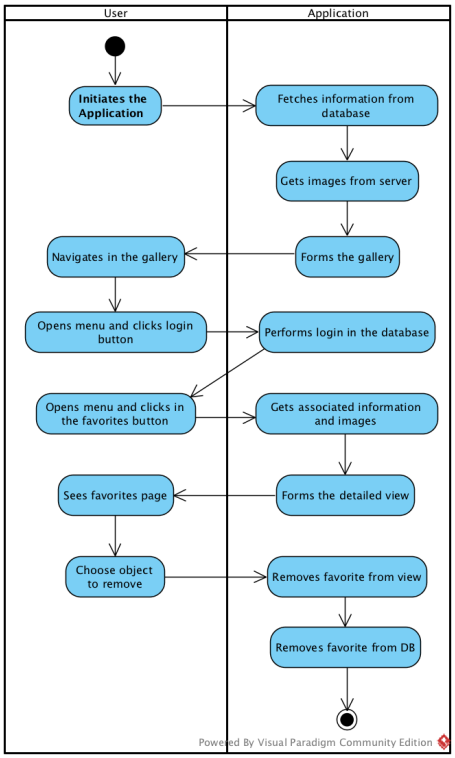


Figure 7.7: Activity diagram for the Remove favorite use case.

Use Case 7 - See information about the exhibition

Name:	**See information about the exhibition**
Actor	User
Pre-Conditions	An internet connection must be established
Sequence of Events	1. The user opens the application 2. The user opens the menu 3. The user chooses the History option 4. A new page with the history information is open
Alternative Sequence	

Table 7.8: Description of the See information about the exhibition.

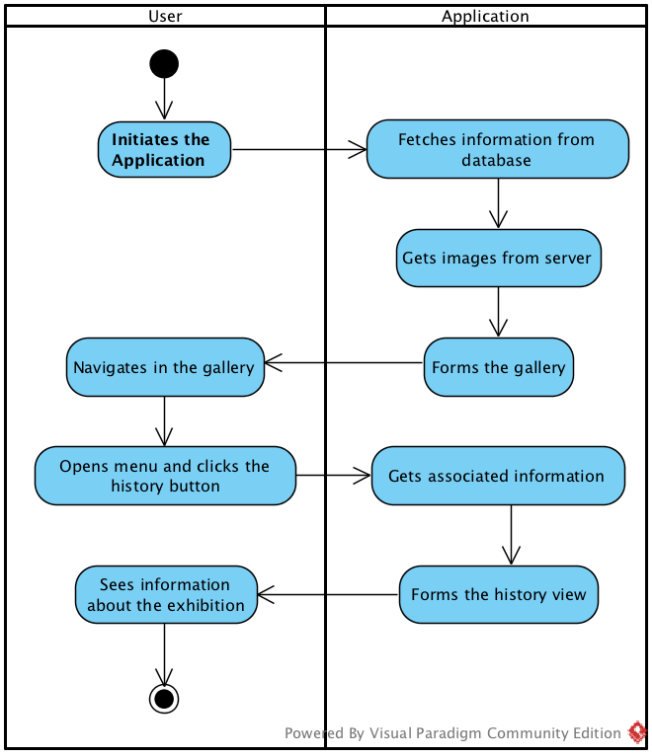


Figure 7.8: Activity diagram for the See exhibition history use case.

Use case 8 - Challenge friend

Name:	**Challenge a Friend**
Actor	User
Pre-Conditions	An Internet connection must be established The user must own a Facebook account The device must have the Facebook native application installed
Sequence of Events	1. The user opens the application 2. The gallery appears to the user 3. The user clicks in one of the *cavaquinhos* 4. A new page with that specific *cavaquinho* is open 5. The user clicks in the Challenge Friend button 6. A new page is presented with the image loaded crop the image 7. A new page with the result 8. The user adds a caption 9. The user confirms the challenge
Alternative Sequence	1. The user opens the application 2. The user opens the menu 3. The user chooses the "Read code" option 4. The user scans the respective code 5. A new page with the specific *cavaquinho* is open 6. The user clicks in the Challenge Friend button 7. A new page is presented with the image loaded crop the image 8. A new page with the result 9. The user adds a caption 10. The user confirms the challenge

Table 7.9: Description of the Challenge a Friend use case.

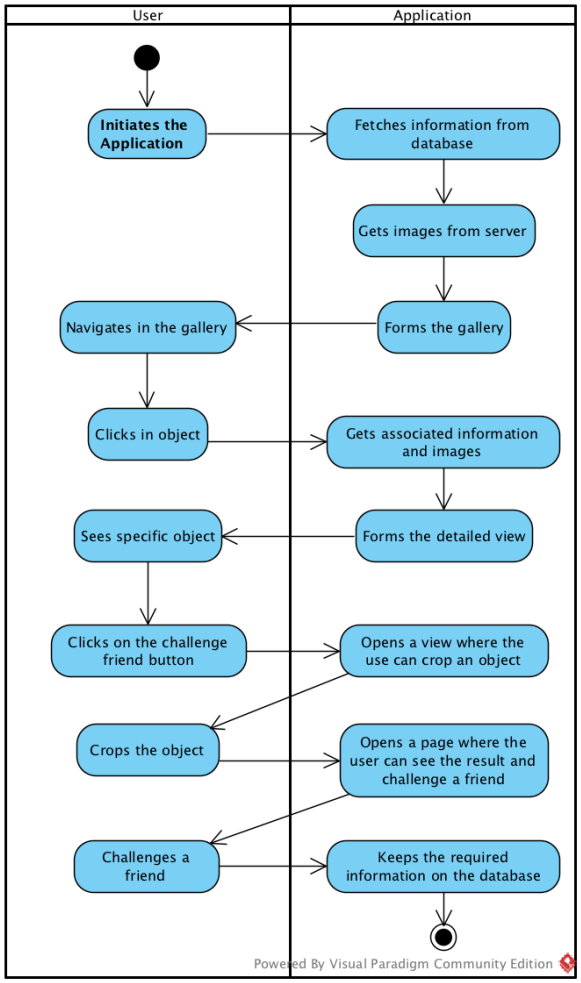


Figure 7.9: Activity diagram for the Challenge a Friend use case.

Use case 9 - Play game

Name:	**Play Game**
Actor	User
Pre-Conditions	An internet connection must be established
Sequence of Events	1. The user opens the application 2. The user opens the menu 3. The user chooses the Play Game option 4. The user plays the game 4. When right, a new page with the associated information is open
Alternative Sequence	

Table 7.10: Description of the Play Game use case.

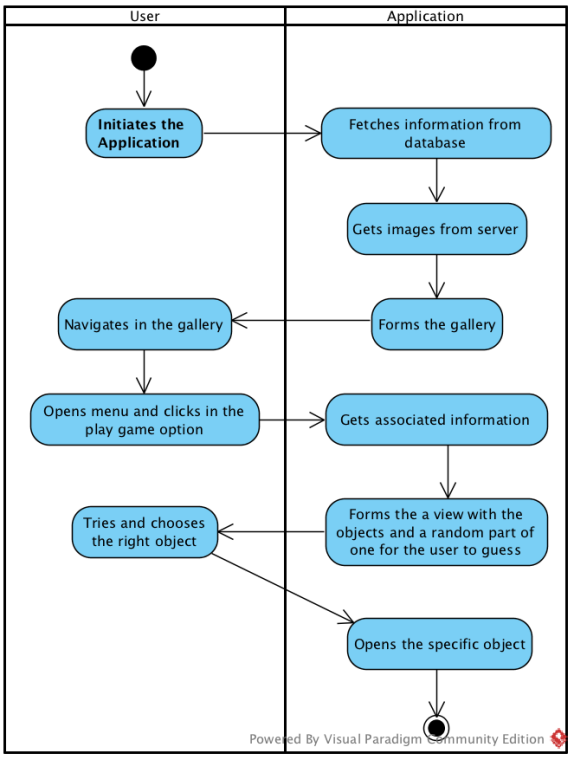


Figure 7.10: Activity diagram for the Play Game use case.

Use Case 10 - Update *cavaquinho* details

Name:	**Update *cavaquinho* details**
Actor	DB Admin
Pre-Conditions	An Internet connection must be established The admin must have login access to the used database
Sequence of Events	1. The Administrator performs the login in the database 2. The Admin opens the developer console 3. The Admin updates a *cavaquinho*
Alternative Sequence	

Table 7.11: Description of the Update *cavaquinho* details use case.

Use Case 11 - Add *cavaquinhos*

Name:	**Add *cavaquinhos***
Actor	DB Admin
Pre-Conditions	An Internet connection must be established The admin must have login access to the used database
Sequence of Events	1. The Administrator performs the login in the database 2. The Admin opens the developer console 3. The Admin adds a *cavaquinhos* to the database
Alternative Sequence	

Table 7.12: Description of the Add *cavaquinhos* use case.

Use Case 12 - Update exhibition information

Name:	**Update exhibition information**
Actor	DB Admin
Pre-Conditions	An Internet connection must be established The admin must have login access to the used database
Sequence of Events	1. The Administrator performs the login in the database 2. The Admin opens the developer console 3. The Admin updates information about the exhibition
Alternative Sequence	

Table 7.13: Description of the Update exhibition information use case.

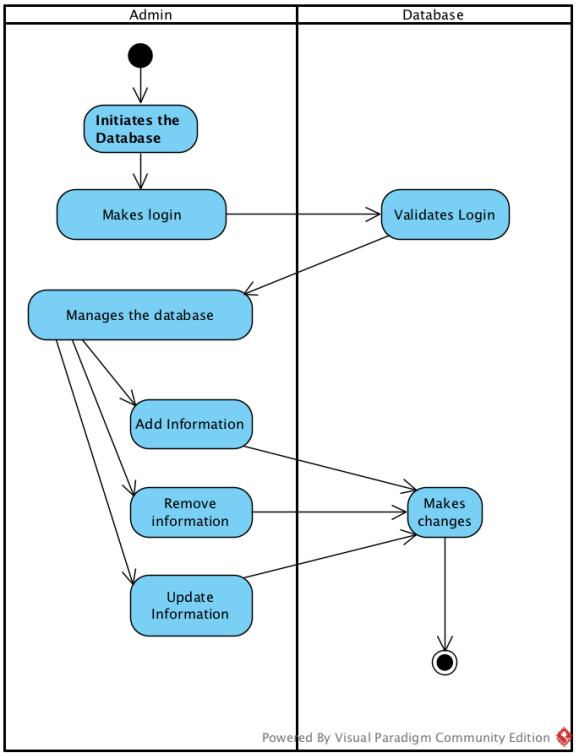


Figure 7.11: Activity diagram for the Manage database information use case (Add, Remove, Update information).