INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Xamarin vs. Android

CARLOS MANUEL MIRANDA MARTINS
Outubro de 2018

POLITÉCNICO
DO PORTO

Xamarin vs. Android

Carlos Martins

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Sistemas Gráficos e Multimédia

Orientador: Dr. Paulo Baltarejo Sousa
Co-Orientador: Pedro Branco

Porto, 14 de Outubro de 2018

Dedicatória

Aos meus pais, irmão e namorada
que sempre me apoiaram quando precisei.
A todos os que, de certa forma, me motivaram
durante este percurso, incluindo eu próprio.

Carlos Martins

Resumo

A Aixtel Engineering produz aplicações móveis apenas para a plataforma Android, mas está a ponderar abranger o desenvolvimento a outras plataformas. Existem diversas abordagens no que se refere ao desenvolvimento de aplicações móveis, mas foi escolhido realizar uma comparação entre o atual desenvolvimento da empresa e o desenvolvimento de aplicações móveis multi-plataforma, mais propriamente, em Xamarin.

O Xamarin permite que as aplicações para as várias plataformas sejam desenvolvidas praticamente, em simultâneo. A grande vantagem em relação ao desenvolvimento nativo é permitir o desenvolvimento em apenas uma linguagem de programação, C#, e apenas um Integrated Development Environment (IDE), Visual Studio.

O Xamarin foi escolhido para a comparação, pois destaca-se das restantes plataformas de desenvolvimento multiplataforma, devido a permitir que, no final, as aplicações sejam nativas. Isto é possível devido à necessidade de criar as interfaces visuais para cada uma delas. O Xamarin também permite desenvolver uma interface comum para várias plataformas, apesar de perder algumas das funcionalidades nativas de cada.

A questão que se coloca neste projeto é se o desenvolvimento em Xamarin é vantajoso em relação ao atual desenvolvimento nativo da empresa, destinado apenas à plataforma Android, quer em termos de performance das aplicações, custos de desenvolvimento e tempo de desenvolvimento de cada aplicação.

Com isto, pretende-se chegar a uma conclusão e perceber se a empresa deve adotar o Xamarin como *framework* de desenvolvimento de aplicações móveis e, assim, abranger as suas aplicações a iOS.

Palavras-chave: Android, Xamarim, desenvolvimento, nativo, multi-plataforma, comparação

Abstract

Nowadays, Aixtel Engineering produces mobile applications only for the Android platform, but is considering extending the development to other platforms. There are several approaches to the development of mobile applications, but it was decided to make a comparison between the current platform of the company and a platform that allows the development of multi-platform mobile applications, more specifically Xamarin.

Xamarin allows multi-platform applications to be developed almost simultaneously. The big advantage over native development is to allow development in just one programming language, C#, and only one IDE, Visual Studio.

Xamarin was chosen for the comparison, as it stands out from the other platforms of multi-platform development, due to allowing the final applications to be native. This is possible because of the need to create the visual interfaces for each of them. Xamarin also lets you develop a common cross-platform interface, even though you may lose some of the native functionality of each platform.

The question that arises in this project is whether the development in Xamarin is advantageous in relation to the current native development of the company, bound only to the Android plataform, both in terms of application perfomance, development costs and time of development of each application.

With this, it's intended to reach a conclusion and realize if the company should adopt Xamarin as the main mobile application development framewokr and thus cover its applications to iOS.

Agradecimentos

Antes de mais, quero agradecer a todos os mencionados na dedicatória. Além disso, quero agradecer a todas as pessoas que estiveram presentes no meu percurso académico e que me acompanharam em todos os momentos.

Por fim, quero agradecer ao meu orientador por todo o apoio.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Código

Lista de Acrónimos

AHP	Analytic Hierarchy Process.
ART	Android Runtime.
CSS	Cascading Style Sheets.
FFE	Fuzzy Front End.
GC	Garbage Collector.
HAL	Hardware Abstraction Layer.
HTML	Cascading Style Sheets.
HTTPS	Hyper Text Transfer Protocol Secure.
IDE	Integrated Development Environment.
JDK	Java Development Kit.
MVC	Model View Controller.
MVP	Model View Presenter.
NCD	New Concept Development.
NPD	New Product Development.
PEST	Política, Económica, Social e Tecnológica.
SDK	Software Development Kit.
SGML	Standard Generalized Markup Language.
SMS	Short Message Service.
SO	Sistema Operativo.
TSG	Technology Stage Gate.
UC	Use Cases.
WORA	Write Once, Run Anywhere.
XML	eXtensible Markup Language.

Capítulo 1

Introdução

O primeiro capítulo desta dissertação apresenta um breve contexto e descrição do problema com o objetivo de enquadrar o leitor no tema em questão. Será exibido um resumo da análise de valor e os objetivos pretendidos. No fim, será apresentada a estrutura do documento.

1.1 Contexto

O Android, desenvolvido pela Google, como Sistema Operativo (SO) foi lançado no seu primeiro *smartphone* em 2008. Após isto, o SO Android teve inúmeras atualizações sendo a mais atual a versão Pie(9.0), lançada em Agosto de 2018. Para o desenvolvimento de aplicações móveis foi lançado, em 2009, o Software Development Kit (SDK) usando a linguagem de programação Java[1] e o eXtensible Markup Language (XML)[1] para a construção das interfaces do utilizador. Em Maio de 2013, foi lançado o Integrated Development Environment (IDE) Android Studio(Android 2018).

A empresa Xamarin foi fundada em Maio de 2011 e adquirida pela Microsoft em 2016. Em 2013 foram lançadas as *frameworks* Xamarin.Android e Xamarin.iOS, que permitem o desenvolvimento de aplicações na linguagem de programação C#[1] para as plataformas Android e iOS respetivamente. Estas aplicações são transformadas em aplicações nativas[2], que são aplicações desenvolvidas para plataformas específicas, e que podem ser obtidas na loja oficial dessa plataforma, através do desenvolvimento de uma interface independente para cada uma das plataformas. Em 2014, surgiu o Xamarin.Forms que simplifica este processo, permitindo uma interface comum a ambas. O número de desenvolvedores que utilizam o Xamarin tem aumentado nos últimos anos (Versluis 2017).

[1] Explicada com detalhe na Secção 3.1.2 do Capítulo 3.
[2] Explicada com detalhe na Secção 3.1.1 do Capítulo 3.

1.2 Problema

A Aixtel Engineering está inserida na área das telecomunicações. A empresa está envolvida em projetos para a instalação de fibra ótica em diversos países. Para auxiliar o desenvolvimento dos projetos, desenvolve plataformas *web* e aplicações móveis que facilitem o trabalho dos seus clientes. Os clientes da Aixtel Engineering são empresas, inseridas na mesma área, que utilizam essas ferramentas para ajudar no desenvolvimento dos projetos. As aplicações móveis são desenvolvidas para o SO Android, e têm como funcionalidade principal a utilização do Google Maps, que facilita o levantamento de dados para a montagem de fibra ótica nas áreas definidas no projeto.

De momento, a empresa está a ponderar a possibilidade de alargar as suas aplicações a outras plataformas, neste caso iOS, com o intuito de abranger um maior número de clientes. Para tal, existem diversas soluções, como o desenvolvimento de aplicações nativas para todas as plataformas, o desenvolvimento de aplicações *web*[2] e o desenvolvimento de aplicações híbridas[2].

Do ponto de vista da empresa, desenvolver as aplicações para cada uma das plataformas seria muito dispendioso, pois seriam necessárias equipas diferentes destinadas ao desenvolvimento em cada plataforma e a compra de *software* e *hardware* necessários para o desenvolvimento, e trabalhoso, dado que seria necessário trabalhar com mais do que uma *framework* e diferentes linguagens de programação.

O desenvolvimento de aplicações *web* também não é o ideal, pois estas não permitem o acesso a certas funcionalidades nativas, dado que necessitam de um *browser* para aceder às mesmas. Isto tem implicações no desempenho das aplicações.

Por isso, a empresa decidiu testar o desenvolvimento de aplicações com uma *framework* multi-plataforma que permita o desenvolvimento de aplicações nativas, mas permitindo o uso de apenas uma *framework* e linguagem de programação. Neste caso, foi escolhido o Xamarin que tem como vantagem o facto de, como mencionado anteriormente, manter as aplicações como nativas e o acesso a funcionalidades que só nativamente são alcançadas.

Para fazer esta escolha será necessário fazer comparações de desempenho entre aplicações desenvolvidas para Android nativamente, em Java, e aplicações desenvolvidas em Xamarin. Além do desempenho, é importante verificar os custos associados a cada um dos desenvolvimentos, bem como, a duração de desenvolvimento das aplicações.

1.3 Objetivos

O principal objetivo desta dissertação é aferir qual a tecnologia mais favorável à empresa, através dos resultados de uma comparação teórica e prática do desenvolvimento de aplicações tanto nativamente em Java, como em Xamarin.

É claro que este objetivo principal tem outros sub-objetivos, pois é necessário que esta comparação seja feita em ambiente empresarial e, como tal, o desenvolvimento de protótipos em ambas as *frameworks* de forma a dar suporte ao estudo, é outro objetivo. Além disso, cada uma das comparações realizadas, quer em termos de desempenho ou custos associados, e a medição final das vantagens e desvantagens de cada *framework* tornam-se outros objetivos.

Destes objetivos, surgem diversas questões. Questões estas que se pretendem responder ao longo desta dissertação e que são:

Q1. Quais as funcionalidades que devem ser exploradas de maneira a obter uma boa avaliação de cada *framework*?

Q2. Qual a *framework* onde é desenvolvida a aplicação com melhor desempenho?

Q3. Qual a *framework* onde os custos do desenvolvimento da aplicação são menores?

Q4. Qual a *framework* onde o tempo de desenvolvimento é menor?

Q5. Quais as vantagens e desvantagens de cada *framework*?

Estas questões levam a uma questão final:

Q6. Qual a *framework* a ser adotada pela empresa?

1.4 Análise de Valor

A realização da análise de valor[3] vai de encontro ao objetivo principal deste trabalho que é a de trazer valor aos serviços prestados pela empresa, neste caso no desenvolvimento de aplicações, tendo em conta os custos associados e sem que haja perda de qualidade dos serviços.

Para tal, a análise de valor irá ser realizada dentro dos modelos New Concept Development (NCD) e Analytic Hierarchy Process (AHP), definir o valor, valor percecionado, valor para o cliente e a proposta de valor. Além disto, será realizado o Modelo de negócio Canvas para completar o modelo de negócio.

[3]Realizada com detalhe na Secção 2.2 do Capítulo 2.

1.5 Estrutura

O presente documento é dividido em três partes fundamentais, introdução, corpo e conclusão. Estas partes são constituídas pelos seguintes capítulos:

- **Capítulo 1 - Introdução**, onde o projeto é contextualizado e introduzido o problema, são apresentados os objetivos, análise de valor e estrutura do documento.
- **Capítulo 2 - Contexto, Problema e Análise de Valor**, onde é realizada uma contextualização mais detalhada e a análise de valor.
- **Capítulo 3 - Estado da Arte e Avaliação de Soluções**, onde é abordado o estado da arte e são analisadas e avaliadas as abordagens definidas.
- **Capítulo 4 - Análise, Design e Desenvolvimento da Solução**, onde é apresentada a análise e o *design* da solução para o problema, bem como, excertos de código relevantes.
- **Capítulo 5 - Comparação das Frameworks**, onde são testadas e comparadas as *frameworks* e respetivas aplicações desenvolvidas.
- **Capítulo 6 - Conclusões**, onde são apresentadas as conclusões da dissertação.

Capítulo 2

Contexto, Problema e Análise de Valor

Este capítulo apresenta uma contextualização e descrição do problema mais detalhada de modo a que o leitor compreenda melhor o tema.

Será apresentada uma solução e será analisado o propósito da mesma. Além disto, será analisado o valor potencial da solução.

2.1 Contexto, Problema e Solução

O Android é um SO baseado no Linux desenvolvido pela Google. No que concerne ao desenvolvimento para Android, o SDK foi lançado em Novembro de 2009 e permite o desenvolvimento de aplicações usando o Java[1], como principal linguagem de programação, e o XML[1] para desenvolver a interface visual. O Android Studio é o principal IDE (Android 2018). Em Julho de 2013, foi atingida, pela primeira vez, a marca de 1 milhão de aplicações móveis disponíveis numa loja oficial, neste caso a Google Play Store. Em Dezembro de 2017, existiam cerca de 3.5 milhões de aplicações (Statista 2017). Hoje em dia, existem cerca de 6 milhões de desenvolvedores para Android e ocupa a maior parte dos desenvolvedores para aplicações móveis (Data 2016).

Em 2009, a base da tecnologia do Xamarin, o Mono Touch, foi lançado, permitindo assim o desenvolvimento de aplicações multi-plataforma, que são aplicações que podem ser instaladas em plataformas distitas, em C#[1]. Esta tecnologia levou à contestação por parte da Apple, no que diz respeito ao desenvolvimento de aplicações para iOS em Mono Touch. Isto pelo facto de não ser usado o desenvolvimento nativo de iOS, levando ao desmantelamento do projeto Mono.

Após estas contestações serem ultrapassadas, foi fundada em Maio de 2011 a empresa Xamarin. Em Fevereiro de 2013, o Xamarin.Android[1], que permite o desenvolvimento nativo para Android, e Xamarin.iOS[1], que permite o desenvolvimento

[1]Explicada com detalhe na Secção 3.1.2.

nativo para iOS, foram lançados (Versluis 2017). A vantagem destas duas *frameworks* é a possibilidade de partilha de código entre ambas, não sendo possível a partilha do código da interface, que é desenvolvida separadamente. Este desenvolvimento distinto permite o acesso a diversas funcionalidades nativas. Em 2014 o Xamarin.Forms[1] foi lançado. O Xamarin.Forms, permite ao desenvolvedor produzir uma interface comum para várias plataformas, mas perdendo algumas das funcionalidades nativas de cada. A empresa foi então adquirida pela Microsoft em 2016. Atualmente, a plataforma de desenvolvimento Xamarin é usada por cerca de 1.4 milhões de desenvolvedores (Xamarin 2018a).

Problema

Como referido no Capítulo 1, existem várias opções para o desenvolvimento de aplicações móveis, tais como, o desenvolvimento de aplicações nativas[2], que são aplicações desenvolvidas especificamente para uma plataforma, como iOS e Android, desenvolvimento de aplicações web[2], que são aplicações desenvolvidas com ferramentas usadas para *web*, são acedidas através de um *browser* e possuem um baixo desempenho, e aplicações híbridas[2], que são aplicações desenvolvidas através de ferramentas usadas para *web*, que podem ser desenvolvidas para várias plataformas mas com acesso a funcionalidades nativas e, normalmente, tem um desempenho pior que as aplicações desenvolvidas nativamente.

A Aixtel Engineering desenvolve plataformas *web* e aplicações móveis que auxiliam a realização de projetos na área das telecomunicações. Atualmente, a empresa produz aplicações móveis apenas para a plataforma Android, tendo como principal funcionalidade a utilização do Google Maps, que facilita o levantamento de dados no terreno delimitado no projeto. Os principais clientes da Aixtel Engineering são empresas que necessitam de fazer esse levantamento de dados para projetar a montagem de fibra ótica. Atualmente, os clientes abrangidos precisam de possuir dispositivos Android para utilizar as aplicações desenvolvidas. Para tal, o desempenho das aplicações no terreno é essencial para o utilizador, no que diz respeito ao uso de bateria e rapidez de uso. Porém, a empresa em questão pretende verificar se o desenvolvimento de aplicações para outras plataformas lhe traz mais vantagens, em termos de custos, desempenho e duração de desenvolvimento, além de abranger mais possíveis clientes que não disponham de dispositivos Android.

Após discussão no seio da empresa e, para chegar a uma conclusão, decidiu-se investigar alternativas ao atual desenvolvimento da empresa, Android nativo. Foi excluído de início o desenvolvimento separado para as diversas plataformas, neste caso Android e iOS, dado os custos de equipas diferentes, *software* e *hardware* associados. Optou-se por uma *framework* destinada ao desenvolvimento multi-plataforma, nomeadamente a *framework* multi-plataforma em Xamarin.

Foi escolhido o desenvolvimento de aplicações móveis multi-plataforma em Xamarin, pois este tem vindo a crescer nos últimos anos, tentando alcançar o desenvolvimento

[2]Explicada com detalhe na Secção 3.1.1 do Capítulo 3.

nativo de Android e iOS. Este crescimento deve-se ao facto do Xamarin permitir que as aplicações para as várias plataformas sejam desenvolvidas, praticamente, em simultâneo, e que, no final, sejam aplicações nativas. Isto torna-se prático pois não são necessárias equipas e ferramentas diferentes para trabalhar com tecnologias específicas como o Android e iOS.

Solução Proposta

A partir deste problema, a solução proposta foi a de realizar uma comparação profunda entre o desenvolvimento para Android nativo e o desenvolvimento multi-plataforma em Xamarin, não se focando apenas nas aplicações, mas também no impacto das mesmas no dispositivo. Para atingir isto, será necessário analisar o meio envolvente do desenvolvimento de aplicações móveis em Android e em Xamarin, nomeadamente, tempo de desenvolvimento das aplicações, custos de desenvolvimento das aplicações, como por exemplo, custos do IDE. Os protótipos desenvolvidos no âmbito desta dissertação não irão ser publicados, mas em relação à empresa, a análise dos custos das lojas associadas será relevante. Também é importante testar se a qualidade e desempenho das aplicações se mantém, quer em termos de gasto de bateria, rapidez e facilidade de uso.

Foi proposto o desenvolvimento de protótipos independentes das aplicações atuais da empresa. No entanto, é importante o desenvolvimento de protótipos robustos que contenham as funcionalidades básicas dessas aplicações e outras funcionalidades que sejam importantes testar. O intuito disto é conseguir testar ao máximo o desempenho das aplicações nos dispositivos executando tarefas complexas.

2.1.1 Propósito

Nesta subsecção será analisado o problema e a solução escolhida, tendo em conta vários pontos de vistas dos vários intervenientes diretos. Esta análise será feita por tópicos.

1. **Propósito**

   O propósito desta solução é responder à questão final **Q6**, referida na Secção 1.3 do Capítulo 1, ou seja, perceber qual das *frameworks* se adequa melhor ao que a empresa pretende. Para tal, é importante perceber como codificar as aplicações em ambas as *frameworks*, com o intuito de conseguir codificar aplicações mais competentes e com mais funcionalidades, de modo a poder fazer uma melhor comparação.

2. **Cliente**

   Neste caso, o único cliente é a empresa, Aixtel Engineering, pois está diretamente ligada e o seu principal interesse é o resultado da solução associada a esta dissertação.

3. **Requisitos do Cliente**

   O primeiro requisito desta solução é apresentar um estudo elaborado de cada uma das *frameworks*, incluindo as suas principais características, facilidade de acesso e custos associados. O segundo requisito, passa por transformar o primeiro em aplicações, uma em cada plataforma. O terceiro, e último requisito, consiste em responder à questão **Q5**, referida na Secção 1.3 do Capítulo 1, que tem como objetivo chegar a uma conclusão sobre as vantagens e desvantagens de cada uma das *frameworks* e chegar a uma decisão final entre as duas.

4. **Definição de Valor para o Aixtel Engineering**

   A possibilidade de expandir as suas aplicações a outras plataformas mantendo, ou até diminuindo, os custos de desenvolvimento e mantendo a qualidade das mesmas, acaba por ser um dos fator-chave nesta solução e a principal fonte de valor para a empresa.

5. **Necessidade de Novas Tecnologias**

   Nesta solução, a *framework* Xamarin, e a linguagem de programação C#, podem ser consideradas novas tecnologias, dado que as tecnologias atualmente utilizadas na empresa são a linguagem de programação Java, com o auxílio do IDE Android Studio e do XML, e que para ser realizada a comparação será necessário o uso de várias tecnologias.

6. **Adaptação de Projetos Existentes**

   A adaptação de outros projetos não foi equacionada nesta solução, visto o problema ser específico da empresa em questão, e do facto de ser uma comparação para a possível substituição das tecnologias usadas na mesma. Como é uma decisão que pode influenciar diretamente a empresa, é pretendido que o desenvolvimento seja controlado por parte da empresa.

2.1.2 Restrições Existentes

Relativamente às restrições tecnológicas, como as tecnologias a ser comparadas e linguagens de programação já foram definidas, o desenvolvimento será restrito a essas, sendo elas, o Java e XML, para o desenvolvimento em Android nativo e o C#, para o desenvolvimento em Xamarin. Além destas, não há outras restrições tecnológicas.

Não há restrições monetárias, pois os IDE escolhidos para este desenvolvimento têm ambos versões gratuitas, sendo eles o Android Studio e o Visual Studio Community, que é a versão gratuita do Visual Studio destinada a estudantes. Além disto, as aplicações finais não irão ser colocadas na loja, por isso não será preciso o pagamento das respetivas taxas.

Quanto à restrição temporal, o tempo de desenvolvimento das aplicações poderá ser demorado, dado ser necessário a adaptação a novas tecnologias e o desenvolvimento de duas aplicações.

2.2 Análise de Valor

A análise de valor, nesta dissertação, é direcionada para o valor proveniente da comparação anteriormente referida na Secção 2.1.

2.2.1 Descrição

A análise de valor é um processo de avaliação sistemático e formal (Nicola, E. P. Ferreira e J. P. Ferreira 2012). Para tal, é necessário perceber o propósito do produto e requisitos do cliente. Estas questões foram respondidas na Subsecção 2.1.1.

O principal objetivo de uma análise de valor é levar ao aumento do valor de um serviço ou produto, com os custos mais baixos possíveis, mas mantendo a qualidade (Nicola, E. P. Ferreira e J. P. Ferreira 2012).

No entanto, a inovação, por parte de uma empresa contem um risco elevado quer económico, quer empresarial. Para efetuar o lançamento de novos produtos ou serviços, deve-se procurar a diminuição desses riscos. O seguimento do processo de inovação, apresentado na Figura 2.1, ajuda na diminuição desses riscos.

O processo apresentado na Figura 2.1, divide-se em três partes fundamentais.

1. **Fuzzy Front End (FFE)** - Ambiente de incerteza e imprevisibilidade, em termos de calendarização, da análise das ideias e da identificação das oportunidades.
2. **New Product Development (NPD)** - Ambiente disciplinado e planeado com foco nos objetivos e no desenvolvimento do produto ou serviço.
3. **Comercialização** - Divulgação e venda do produto ou serviço.

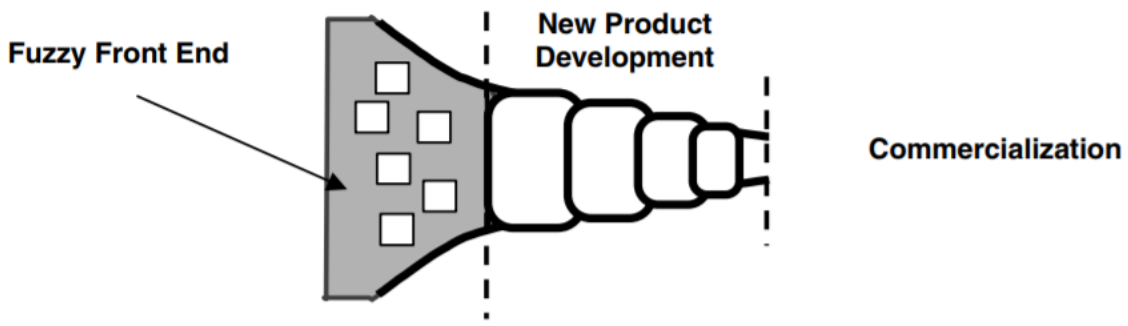

Figura 2.1: Processo de Inovação (Koen et al. 2002).

Nesta dissertação, irá ser abordada a notação do modelo NCD de Koen (Koen et al. 2002), permitindo apresentar os vários estágios do processo de inovação, desde o problema à geração de ideias e sua seleção.

2.2.2 Modelo NCD

O modelo NCD, como apresentado na Figura 2.2, permite uma definição dos padrões-chave do FFE. É composto pelo motor, por cinco elementos essenciais e por fatores que influenciam o processo. As setas de entrada na Figura 2.2 demonstram que o processo pode começar na identificação de oportunidades, bem como, na geração de ideias. A seta de saída representa como os conceitos deixam o modelo e entram nos processos NPD ou Technology Stage Gate (TSG).

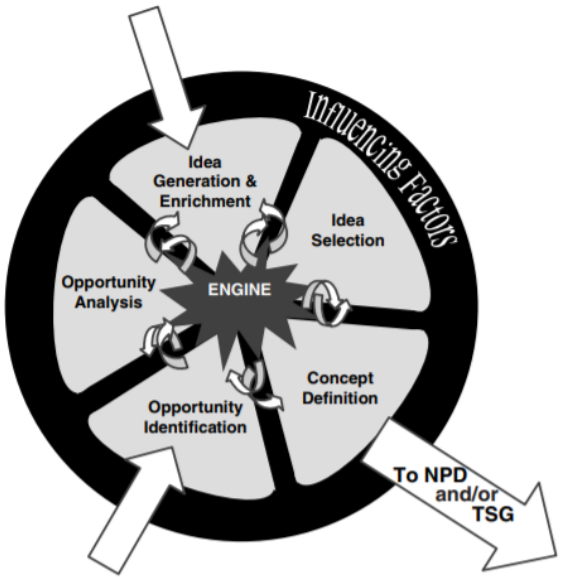

Figura 2.2: NCD (New Concept Development) (Koen et al. 2002).

Motor

O motor, que representa a gestão de nível executivo, dá a ignição aos cinco elementos do Modelo NCD (Koen et al. 2002).

Elementos da Atividade Inovadora

O modelo NCD é caracterizado pelos seguintes cinco elementos:

1. **Identificação de Oportunidade**

   O mercado das aplicações móveis está em constante mudança. Algo que se pode em ter em conta é o constante crescimento do mesmo, levando este mercado a ser uma das maiores tendências nos passados anos. Este crescimento deve-se a diversos fatores, entre os quais o crescimento global da utilização de *smartphones*.

   A cada dia que passa, os *smartphones* são mais utilizados, por utilizadores de todas as idades, para ações simples no seu quotidiano, principalmente na consulta de redes sociais. Além da quantidade, o tempo médio que cada pessoa passa a utilizar um *smartphone*, por dia, também tem vindo a aumentar (ComScore 2017).

   Como mostra a Figura 2.3, é possível verificar esse mesmo crescimento no uso de *smartphones*, ultrapassando mesmo, em meados de 2014 o uso de computadores, que também se encontra em crescimento.

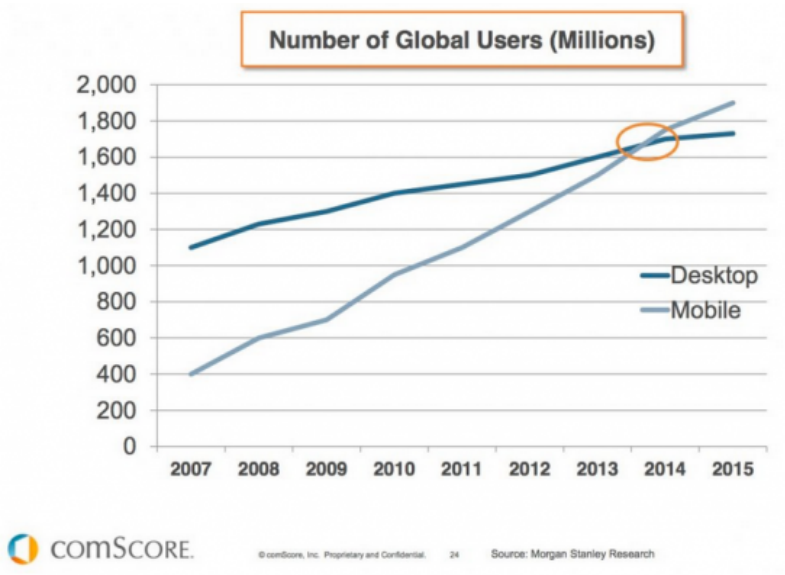

Figura 2.3: Número de Utilizadores Globais de *Smartphones* e Computadores (ComScore 2017)

O crescimento do mercado de aplicações móveis é um dos motivos pelo qual a Aixtel Engineering produz aplicações para o mesmo. No entanto, e como referido anteriormente no Capítulo 1, a Aixtel Engineering produz apenas para a plataforma Android, atingindo apenas uma porção do mercado. Contudo, segundo a StatCounter, a porção abrangida pela Aixtel Engineering ultrapassa mais de metade do mercado, sendo que o SO Android ocupa cerca de 70% dele. Os restantes cerca de 30% estão partilhados pelos outros SO, em que o iOS ocupa maior parte dele (StatCounter 2018).

Outro fator muito importante para a oportunidade é o do uso de Xamarin se encontrar em constante crescimento e adoção, por parte dos desenvolvedores. Além do crescimento individual, o Xamarin também se tem destacado das outras *frameworks* de desenvolvimento multi-plataforma, devido a características como o acesso a funcionalidades nativas. Isto despertou curiosidade por parte da empresa, pois iria permitir abranger o restante mercado de aplicações móveis, apesar de ser necessária a verificação da qualidade das mesmas.

Apesar das aplicações da Aixtel Engineering já abrangerem cerca de 70% do mercado, a possibilidade de poder abranger todo o mercado através do desenvolvimento multi-plataforma, sem aumento de custos e sem perda da qualidade das suas aplicações é algo a ter em conta.

A conjunção destes fatores, levou a Aixtel Engineering a identificar esta oportunidade como favorável e potenciadora de valor para a empresa.

2. **Análise de Oportunidade**

   Para suportar a oportunidade identificada, e como mostrado, recorreu-se à análise de dados estatísticos do mercado, nomeadamente às taxas de utilização global de *smartphones* e computadores, embora nesta situação seja mais relevante os *smartphones*, às taxas diárias de utilização de *smartphones* e aplicações mais utilizadas. O crescimento deste mercado é uma implicação direta no tema desta dissertação, que incide sobre o desenvolvimento de aplicações móveis.

   Além disto foi analisado o mercado do desenvolvimento de aplicações multi-plataforma, aferindo que o Xamarin se encontra em crescimento e distanciamento das outras *frameworks* de desenvolvimento de aplicações multi-plataforma. Este dados foram analisados pela empresa, além do estudo dos possíveis impactos que esta oportunidade poderia ter na empresa, benefícios e custos. Estudo esse que irá ser suportado por esta dissertação.

3. **Geração de Ideias**

   Após a realização da análise de oportunidade e problema associado, e através de um conjunto de *brainstormings*, foram geradas algumas ideias de maneira a acrescentar valor à oportunidade definida.

   A principal ideia é levar o desenvolvimento da empresa a outras plataformas. De maneira a atacar o restante mercado de *smartphones*, e como referido anteriormente, surgiu a ideia da utilização de uma *framework* de desenvolvimento multi-plataforma, particularmente o Xamarin. No entanto, com o objetivo de não perder a qualidade atual das aplicações, foi refletido sobre a realização de um estudo, implicado nesta dissertação. Estudo esse, que deve responder à questão final **Q6**, referida na Secção 1.3 do Capítulo 1, incluindo uma comparação entre o desenvolvimento atual da empresa e o desenvolvimento multi-plataforma.

4. **Seleção de Ideias**

   Das ideias refletidas e com base nos estudos do mercado, foi decidido realizar esse mesmo estudo incidindo sobre a *framework* Xamarin. Dependendo do resultado do mesmo será analisada a possibilidade de executar a ideia principal.

5. **Definição de Conceito**

   A definição de conceito deve surgir após a ideia principal, atacar o mercado das restantes plataformas, estar bem definida. Para tal, como foi referido anteriormente, será necessário efetuar primeiro o estudo definido aquando da geração de ideias.

Fatores Ambientais Externos

A seguinte análise foi feita tendo em conta o que foi referido na Identificação de Oportunidade e os fatores externos da análise Política, Económica, Social e Tecnológica (PEST).

Em relação aos Fatores Económicos, o mercado onde a empresa se encontra está em crescimento, tanto em território nacional, como globalmente. Nos Fatores Sociais pode-se considerar que o estilo de vida tem mudado graças ao avanço tecnológico, quer em termos de educação ou trabalho. No que toca aos Fatores Tecnológicos, mais propriamente, na área móvel, o seu desenvolvimento e respetivo crescimento são uns dos grandes fatores, como foi evidenciado anteriormente.

2.2.3 Valor, Valor Percecionado e Valor para o Cliente

Nesta subsecção será analisada a definição de valor e a quem se destina o mesmo. Além disso, será analisado o valor percecionado, suportado pela tabela de benefícios e custos. Por fim, será analisado o valor para o cliente.

Valor

A definição de valor é dependente de pessoa para pessoa, pois cada um avalia a aquisição de um produto ou serviço tangível ou não tangível de maneira única. A troca de produtos ou serviços é a base de qualquer negócio, como tal a criação de valor para quem usufrui dos mesmo é essencial (Nicola, E. P. Ferreira e J. P. Ferreira 2012).

No caso desta dissertação, o serviço será efetuado para a empresa Aixtel Engineering e é para ela que é necessário a criação de valor. O valor irá tentar ser alcançado através da comparação entre o desenvolvimento atual da empresa e o desenvolvimento em Xamarin.

Tabela 2.1: Benefícios e Sacrifícios

	Serviço	**Relação**
Benefícios	Substituição das atuais tecnologias.	Aumento da Satisfação; Fortalecimento da Imagem; Abranger novas plataformas; Mais clientes.
Sacrifícios:	Custos associados às novas tecnologias.	Possível descontentamento dos atuais colaboradores; Formação para as novas tecnologias.

Valor Percecionado

O valor percecionado varia de cliente para cliente, pois está dependente da expectativa que cada cliente tem em relação a um produto ou serviço (Ulaga e Eggert 2006).

No caso desta dissertação, o valor percecionado é alto para o cliente, pois a empresa poderá ter grandes benefícios e alargar as suas fronteiras e, por sua vez, alcançar mais clientes. Contudo, pode ter grandes sacrifícios caso seja preciso uma alteração drástica das tecnologias atualmente utilizadas para o desenvolvimento na empresa.

A Tabela 2.1, ajuda na fundamentação do valor percecionado desta dissertação, através da análise dos benefícios e sacrifícios do ponto de vista do cliente.

Valor para o Cliente

O valor para o cliente é determinado pela relação entre os benefícios e sacrifícios associados a um produto ou serviço. É necessário pesar cada um deles, de forma a atribuir esse valor (Woodall 2003).

O cliente tem de analisar as vantagens e desvantagens da possível substituição das atuais tecnologias utilizadas para o desenvolvimento. Como referido anteriormente, esta dissertação tem como principal objetivo ajudar o cliente nesta decisão.

2.2.4 Proposta de Valor do Projeto

A solução apresentada nesta dissertação, procura solucionar o problema apresentado na Secção 2.1, aferindo se a Aixtel Engineering deve ou não substituir as atuais tecnologias utilizadas pela empresa.

Esta solução irá criar valor para a empresa, assumindo que além do alargamento das suas fronteiras haverá outras vantagens associadas. No melhor dos casos, a

empresa iria beneficiar com a produção de aplicações com o mesmo ou melhor desempenho que as atuais, custos menores e menor tempo de produção. Todos estes fatores poderão levar ao contentamento dos clientes atuais e eventual crescimento da empresa no mercado. A possibilidade de expansão a outros mercados pode-se tornar uma possibilidade.

Estas vantagens, e eventuais desvantagens, irão ser estudadas ao longo desta dissertação.

2.2.5 Modelo de Negócio Canvas

De forma a suportar este negócio, foi criado o modelo de negócio Canvas. que está representado na Figura 2.4. De seguida, são apresentados os componentes presentes nesse modelo.

Parcerias Chave

A Google e a Microsoft seriam as principais parcerias chave.

Atividades Chave

As atividades chaves serão executar a comparação das diversas tecnologias utilizadas para o desenvolvimento e desenvolver as aplicações para suportar essa comparação.

Recursos Chave

O recurso chave deste negócio é a infraestrutura tecnológica da aplicação.

Estrutura de Custos

O único custo associado a este negócio é associado à colocação das aplicações na Google Play Store.

Proposta de Valor

A proposta de valor é essencialmente o alargamento das fronteiras da empresa.

Relação com os Clientes

A relação com os clientes será uma relação direta.

Canais

Mudança direta na empresa.

Segmentos de Mercado

O segmento de mercado restringe-se à empresa em questão, Aixtel Engineering.

Fontes de Rendimento

No caso das fontes de rendimento não é apresentado nenhuma fonte, pois este serviço será efetuado no seio da empresa. A fonte de rendimento será depois atingida na venda dos produtos, neste caso das aplicações eventualmente desenvolvidas.

Figura 2.4: Modelo de Negócio Canvas

2.2.6 Rede de Valor

"People naturally network as they work so why not model itself as network" (V. Allee)

De acordo com esta citação, a autora Verna Allee, evidencia que naturalmente as pessoas trabalham em rede e, como tal, poderia-se modelar o trabalho como uma rede. Verna Allee foi autora da metodologia "Network Value Analysis", ou análise da rede de valor.

Uma rede de valor é uma teia de relações complexas que geram valor tangível ou intangível. Estas relações terão de ser feitas entre dois ou mais indivíduos, grupo ou organização. Qualquer organização que esteja envolvida na troca de valores tangíveis ou intangíveis é considerada uma rede valor (Allee 2002).

No caso desta dissertação, e relativamente ao seio da empresa, será necessário um relacionamento confiável e estável no que diz respeito à troca de valores entre os membros dos departamentos, mais especificamente no departamento de desenvolvimento.

Após esta solução ser possivelmente implementada na empresa, será necessário que a relação da empresa com os clientes procure estabelecer confiança, de maneira a procurar a estabilidade, credibilidade e segurança nessas relações. Isto seria essencial para satisfazer as necessidades dos seu clientes.

2.2.7 Modelo AHP

O Modelo AHP, criado por (Saaty 1990), ajuda no processo de avaliação e de toma de decisões, ao permitir o uso de multi-critérios qualitativos e quantitativos. Nesta dissertação, iremos usar o Modelo AHP para apurar as melhores tecnologias destinadas ao desenvolvimento, e os critérios a ter em atenção são:

- Desempenho
- Tempo
- Custos

O autor deste Modelo, (Saaty 1990), definiu um escala fundamental, que consiste num conjunto de níveis de importância para comparar os diversos critérios. A Tabela 2.2, demonstra os níveis de importância, a sua definição e explicação.

Tabela 2.2: Escala fundamental - Níveis de importância de comparações (Saaty 1990).

Nível de Importância	**Definição**	**Explicação**
1	Igual importância.	As duas atividades contribuem igualmente para o objetivo.
3	Fraca importância.	A experiência e o julgamento favorecem levemente uma atividade em relação à outra.
5	Forte importância.	A experiência e o julgamento favorecem fortemente uma atividade em relação à outra.
7 **9**	Muito forte importância. Importância absoluta.	Uma atividade é muito fortemente favorecida em relação à outra. A evidência favorece uma atividade em relação a outra com o mais alto grau de certeza.
2, 4, 6, 8	Valores intermediários.	Quando se procura uma condição de compromisso entre duas definições.

Tendo em conta esta escala, pode-se definir o critério do Desempenho como o mais importante, sendo 3 vezes mais importante que os Custos e 6 vezes mais importante que o Tempo. Por sua vez, os Custos são 2 vezes mais importantes que o Tempo. A Matriz 2.5 serve para demonstrar a importância dos critérios.

$$MIC = \begin{array}{c} \\ D \\ T \\ C \end{array} \begin{array}{c} \begin{array}{ccc} D & T & C \end{array} \\ \begin{bmatrix} 1 & 6 & 3 \\ 1/6 & 1 & 1/2 \\ 1/3 & 2 & 1 \end{bmatrix} \end{array}$$

Figura 2.5: Matriz de importância de critérios.

Através da Matriz 2.5 é possível calcular o Vetor 2.6, denominado vetor de prioridades, ou vetor próprio, que nos irá indicar o peso de cada critério. O cálculo deste vetor é efetuado através da divisão de cada elemento de uma coluna da matriz com

a soma dos valores dessa coluna e, no final, com o cálculo da média de cada nova linha da matriz.

$$\begin{matrix} D \\ T \\ C \end{matrix} \begin{bmatrix} 0.67 \\ 0.11 \\ 0.22 \end{bmatrix}$$

Figura 2.6: Vetor Próprio.

Após a decisão dos pesos de cada critério, e tendo como base os resultados do Capítulo 5, é necessário aplicar este conceito às diferentes *frameworks*.

Desempenho

Como indica a Matriz 2.7, o desempenho do desenvolvimento em Android nativo é de fraca importância relativamente a Xamarin, visto que os resultados obtidos na Secção 5.3 do Capítulo 5 indicam que o desempenho é favorecido em Android nativo, mas apenas levemente.

$$D = \begin{matrix} \\ Android \\ Xamarin \end{matrix} \begin{matrix} \begin{matrix} Android & Xamarin \end{matrix} \\ \begin{bmatrix} 1 & 3 \\ 1/3 & 1 \end{bmatrix} \end{matrix}$$

Figura 2.7: Matriz de comparação de desempenho.

Custos

Como indica a Matriz 2.8, os custos do desenvolvimento em Android nativo são de importância absoluta em relação aos custos do desenvolvimento em Xamarin, visto que os resultados obtidos na Secção 5.2 do Capítulo 5 indicam que os custos em Android nativo são favorecidos com o mais alto grau de certeza em relação a Xamarin.

$$C = \begin{matrix} \\ Android \\ Xamarin \end{matrix} \begin{matrix} \begin{matrix} Android & Xamarin \end{matrix} \\ \begin{bmatrix} 1 & 9 \\ 1/9 & 1 \end{bmatrix} \end{matrix}$$

Figura 2.8: Matriz de comparação de custos.

Tempo

Não foi possível analisar o critério de tempo, pois não foram obtidos resultados, tal como indicado na Secção 5.4 do Capítulo 5.

Matriz de Prioridades

Após a análise de cada critério, é necessário calcular o vetor prioridade para cada um deles. Estes vetores serão apresentados na Matriz **??**.

$$MP = \begin{matrix} \\ Android \\ Xamarin \end{matrix} \begin{matrix} \begin{matrix} P & T & C \end{matrix} \\ \begin{bmatrix} 0.75 & 0 & 0.9 \\ 0.25 & 0 & 0.1 \end{bmatrix} \end{matrix}$$

Figura 2.9: Matriz de prioridades.

Classificação das Frameworks

A Matriz 2.10 apresenta a classificação das *frameworks*. Este cálculo é efetuado multiplicando cada elemento da matriz de prioridades com o peso dos critérios anteriormente calculados (0.67, 0.11, 0.22) e somando a linha de maneira a obter o resultado. Como se pode verificar Android possui um resultado de cerca de 70% indo de encontro com o resultado dos dados efetuados no Capítulo 5.

$$CF = \begin{matrix} \\ Android \\ Xamarin \end{matrix} \begin{matrix} \begin{matrix} P & T & C & \textbf{Resultado} \end{matrix} \\ \begin{bmatrix} 0.5025 & 0 & 0.198 & \mathbf{0.7005} \\ 0.1675 & 0 & 0.022 & \mathbf{0.1895} \end{bmatrix} \end{matrix}$$

Figura 2.10: Classificação das Frameworks.

2.3 Sumário

Este capítulo descreve detalhadamente a contextualização e análise do problema. Foi realizada a análise de valor, onde foram seguidos os Modelos NCD e AHP, foi definido o valor, valor percecionado e valor para o cliente e a proposta de valor.

Neste capítulo foram mencionadas as questões **Q5** e **Q6**, que irão ser respondidas no Capítulo 6.

Capítulo 3

Estado da Arte e Avaliação da Solução

Este capítulo tem como principais objetivos a apresentação do estado da arte das tecnologias diretamente interligadas com a solução proposta, definida na Secção 2.1 do Capítulo 2, e a avaliação da mesma.

3.1 Estado da Arte

Nesta dissertação, o Estado da Arte tem como objetivo apresentar o estudo das tecnologias escolhidas para a realização do projeto associado à mesma.

Além disto, serão analisadas as alternativas possíveis a estas tecnologias, mais precisamente, as alternativas à *framework* escolhida para a comparação, o Xamarin.

3.1.1 Tipos de Desenvolvimento de Aplicações Móveis

Nesta subsecção, irão ser explicados os diferentes tipos de desenvolvimento de aplicações móveis, onde se incluem aplicações nativas, aplicações *web* e aplicações híbridas.

Aplicações nativas

Aplicações nativas são aplicações especificamente desenvolvidas para uma plataforma. Aplicações para a plataforma Android são desenvolvidas em Java ou Kotlin, que é uma linguagem de programação que compila na máquina virtual Java, e são disponibilizadas na loja da Google, Google Play Store. Aplicações para a plataforma iOS são desenvolvidas em Objective-C ou Swift e são disponibilizadas na loja da Apple, iStore.

Estas aplicações, dado o facto de serem desenvolvidas nativamente, têm acesso a todas as funcionalidades dos dispositivos e das APIs, sendo possível desenvolver

aplicações mais robustas. A grande desvantagem do desenvolvimento de aplicações nativas é o facto de as aplicações para uma plataforma não poderem ser usadas noutra plataforma distinta e, caso a aplicação seja necessária em diversas plataformas, normalmente ser necessário várias equipas de desenvolvimento, uma para cada plataforma (BuildFire 2016).

Aplicações web

Aplicações *web* são desenvolvidas com tecnologias *web*, como JavaScript, Cascading Style Sheets (HTML) e Cascading Style Sheets (CSS), e funcionam como um *website* dado que necessitam de ser acedidas através de um *browser* no dispositivo. Apesar de poderem ser acedidas nas diversas plataformas e serem de rápido desenvolvimento, possuem uma grande desvantagem que é não conseguir aceder às funcionalidades nativas do dispositivo e às APIs (BuildFire 2016).

Aplicações híbridas

Aplicações híbridas são desenvolvidas com tecnologias *web* e podem ser disponibilizadas para as plataformas Android e iOS nas lojas oficiais de cada plataforma. As aplicações híbridas possuem a grande vantagem de conseguir aceder às funcionalidades nativas dos dispositivos devido ao acesso aos SDK nativos. A grande desvantagem associada às aplicações híbridas é normalmente o desempenho das mesmas e a sensação de utilização que pode não ser semelhante à de utilização de aplicações nativas (BuildFire 2016).

3.1.2 Tecnologias

Nesta subsecção, irão ser estudadas as tecnologias que irão estar envolvidas no decorrer do trabalho realizado no contexto desta dissertação.

Android

O Android é um SO que pertence à Google, baseado no *kernel* do Linux. Este SO foi desenhado, essencialmente para *smartphones* e *tablets*, e o primeiro dispositivo com este SO foi lançado em Setembro de 2008. Atualmente, a Google já alargou as suas fronteiras a outros dispositivos, como por exemplo, Android TV utilizado em televisões, Android Auto utilizado em carros e Android Wear utilizador em *smartwatches* (Android 2018).

A versão mais recente deste SO é a 9.0, denominada Pie. Esta versão é bastante recente tendo sido lançada em Agosto de 2018. Como tal, o número de dispositivos com a versão Pie é praticamente inexistente. Como se pode verificar no gráfico

apresentado na Figura 3.1, e tendo em conta que só estão no Gráfico as versões com mais de 0.1% de utilização, é possível ver que as versões Lollipop(5.0), Marshmallow(6.0) e Nougat(7.0) ainda são bastante utilizadas ocupando um pouco mais de 50% da utilização. Por sua vez, o Oreo(8.0) possui uma utilização de cerca de 20%. Estas versões foram lançadas entre 2014 e 2017, por isso espera-se que o Pie venha a ter a mesma aceitação (Developers 2018a).

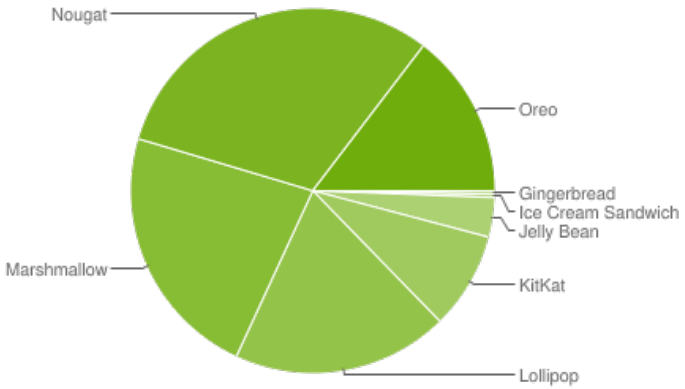

Figura 3.1: Gráfico de Versões Utilizadas em Dispositivos Android (Developers 2018a).

Sempre que uma versão nova é lançada é associada a um número de versão, 9.0 no caso da versão Pie, e é associada a um API *Level*, ou nível de API, API *Level* 28 no caso da versão Pie. Posto isto, cada dispositivo tem uma versão associada e, como tal, um API *Level* associado. Isto permite que, aquando do desenvolvimento de uma aplicação, seja possível definir a API *Level* mínima que a aplicação permite e a API *Level* a que se destina. Normalmente, à medida que são lançadas novas versões certas funcionalidades de versões antigas são descontinuadas e surgem novas funcionalidades, por isso normalmente é definido um API *Level*, ou uma versão, relativamente recente.

Além do *kernel* do Linux, o Android é composto por outros componentes, como por exemplo, o Android Runtime (ART) e a Hardware Abstraction Layer (HAL), como mostra a Figura 3.2.

De seguida irão ser explicados os componentes representados na Figura 3.2:

1. **Linux Kernel**

   Como dito anteriormente, a base do Android é o *kernel* do Linux que executa, por exemplo, funcionalidades como o encadeamento e gestão de memória de baixo nível. Além disto fornece recursos de segurança, drivers para os diversos periféricos e gestão de energia (Developers 2018c).

2. **HAL**

   A HAL é composto por diversos módulos de biblioteca, que implementam uma interface para um componente de hardware específico, como por exemplo o

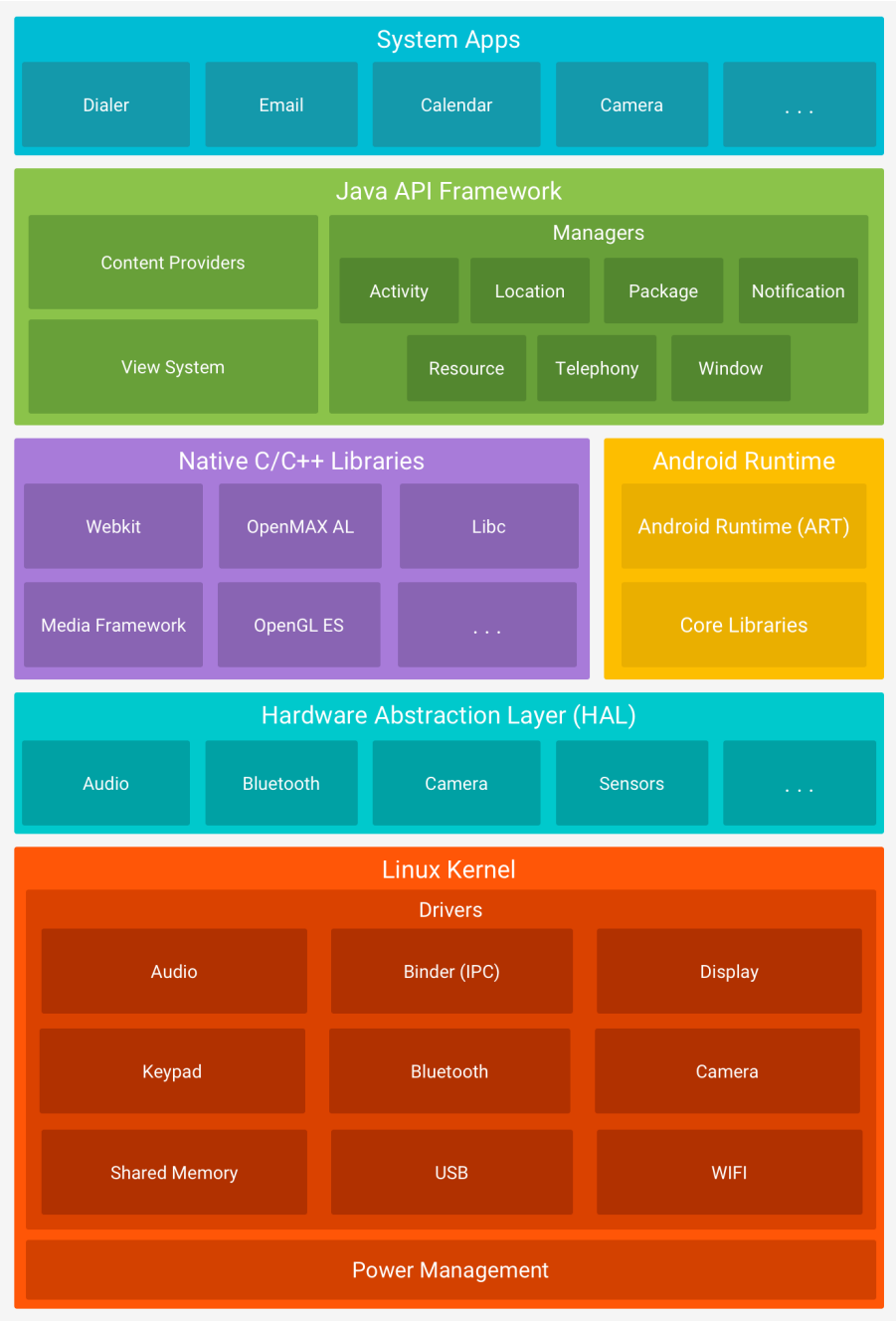

Figura 3.2: Componentes do Android (Developers 2018c).

módulo da câmara, áudio ou *bluetooth*, representados na Figura 3.2. Sempre que é necessário o acesso a algum componente de hardware, por parte de uma *Framework*, é carregado o módulo específico desse componente (Developers 2018c).

3. **ART**

   O ART tem como principal objetivo executar várias máquinas virtuais em dispositivos de baixa memória. Isto é realizado, executando arquivos DEX. O DEX é um formato de *bytecode*, usado com o intuito de consumir o mínimo de memória, e foi projetado exclusivamente para o Android. Os dispositivos com a versão igual, ou superior, à versão 5.0 permitem que cada aplicação execute o seu processo numa instância do ART. Além disto, a otimização do Garbage Collector (GC) é um dos recursos principais do ART (Developers 2018c).

4. **Bibliotecas C/C++ Nativas**

   Certos componentes e serviços do Android, como o ART e HAL mencionados anteriormente, são implementados por código nativo que exige bibliotecas nativas de C e C++. Para isto, o Android possui as Java Framework API que permitem o acesso o acesso a essas bibliotecas como, por exemplo, a Java OpenGL API que permite o acesso à biblioteca OpenGL ES (Developers 2018c).

5. **Estrutura da Java API**

   A Java API é composta por diversos blocos de programação que são necessários para desenvolver aplicações Android. São exemplos disto o sistema de visualização, onde se inclui o XML, o gestor de recursos, que controla o acesso a gráficos e *layouts*, gestor de notificações, que permite notificações nas aplicações, e o gestor de atividade, que controla o ciclo de vida de uma aplicação (Developers 2018c).

6. **Aplicações do Sistema**

   O Android contem de raiz diversas aplicações como, por exemplo, aplicações de aceder ao Email, calendário, câmara e troca de mensagens (Developers 2018c).

Nas aplicações desenvolvidas para Android, o uso de permissões é essencial caso a aplicação aceda a recursos protegidos do dispositivo. Isto foi criado, de maneira a manter a privacidade do utilizador. É necessário definir que permissões a aplicação vai usar, bem como, perguntar ao utilizador se autoriza as mesmas. As permissões dão acesso a dados pessoais, tais como, Short Message Service (SMS) ou ficheiros no dispositivo, ou funcionalidades como acesso à câmara ou Internet (Developers 2018b).

Estas permissões necessitam de ser declaradas no manifesto da aplicação, mais precisamente no ficheiro AndroidManifest.xml. Este manifesto contém informações

relevantes para o sistema, tais como a declaração do nome do pacote Java, declaração de permissões da aplicação, nome da aplicação e declarações de atividades.

Java

A linguagem de programação Java é uma linguagem genérica orientada a objetos. Foi desenhada para ser simples e de maneira a poder ser aprendida e aperfeiçoada por muitos programadores. A linguagem de programação Java tem semelhanças em termos de sintaxe em relação ao C++, embora seja organizada de maneira diferente. Alguns detalhes do C++ foram retirados e foram usadas ideias relativas a outras linguagens (Gosling et al. 2014).

Esta linguagem permite a distinção entre erros de compilação e erros de *runtime*. Esta linguagem é considera uma linguagem de alto nível. Possui gestão automática de armazenamento, normalmente usando o GC. Além disto, a linguagem não permite acessos inseguros como, por exemplo, o acesso a *arrays* sem antes verificar a posição pretendida (Gosling et al. 2014).

A principal diferença no Android é referente às máquinas virtuais. Em Android, como mencionado anteriormente, o *bytecode* é executado através do ART, nas versões a partir da 5.0, e com o Dalvik, nas versões anteriores (Gosling et al. 2014).

XML

O XML é composto por documentos de dados chamados documentos XML e, atualmente, é um formato popular para partilhar informação na Internet. Além disso é um subtipo do Standard Generalized Markup Language (SGML). Os documentos XML têm como objetivo codificar certos documentos em linguagem de máquina, através de um conjunto de regras (Bray et al. 1997).

Cada documento XML deve ser composto por um elemento, designado *Tag*, inicial e, além desse, por uma só *Tag* a servir de raiz. Todos as outras *Tags* devem estar incluídas dentro da *Tag* raiz. Todas as *Tags* do documento devem ser iniciadas e fechadas (Bray et al. 1997).

Com o XML em Android é possível fazer os *layouts* da interface do utilizador, através do vocabulário definido pelo Android, que permite o uso de *Tags* específicas para elementos da interface.

Xamarin

O Xamarin, atualmente que pertence à Microsoft, permite o desenvolvimento de aplicações multi-plataforma, como já foi mencionando anteriormente. Isto inclui o desenvolvimento de aplicações para Android e iOS através de uma única linguagem de programação, um IDE e acesso à *framework* .NET. A linguagem escolhida foi

o C#, que atualmente é uma linguagem bastante popular, e o IDE escolhido foi o Visual Studio (Xamarin 2018a).

Desde já, o desenvolvimento em Xamarin torna-se vantajoso, pois não é necessário a aprendizagem de Objetive-C, destinado à plataforma iOS, e de Java, destinado a Android. O uso de apenas um IDE também é vantajoso e mais eficiente que o uso de vários (Xamarin 2018a).

"Xamarin apps look and feel native because they are."(Xamarin 2018a)

O Xamarin permite que as aplicações finais sejam nativas para todas as plataformas. Isto é possível, devido ao acesso às APIs nativas de cada plataforma, permitindo o acesso a funcionalidades específicas de cada uma. Também permite que as interfaces do utilizador sejam nativas visualmente e que possuam controlos nativos também. O desempenho também é comparável com a nativa, pois o Xamarin compila as aplicações de maneira a que isso aconteça (Xamarin 2018a).

O Xamarin fornece três possibilidades para o desenvolvimento de aplicações móveis:

1. **Xamarin.iOS**

   Permite o desenvolvimento de aplicações nativas para iOS e, portanto, a colocação das mesmas na App Store. Além disto permite o acesso a qualquer API de iOS, a utilização de código já desenvolvido em Objetive-C e o desenvolvimento de aplicações WatchKit, destinadas aos *smartwatches* da Apple (Xamarin 2018a).

2. **Xamarin.Android**

   Tal como o Xamarin.iOS, o Xamarin.Android permite o desenvolvimento de aplicações nativas, mas para Android. Permite também o acesso às APIs de Android, a utilização de código Java e a possibilidade de desenvolver aplicações para Android Wear, os *smartwatches* com o SO Android (Xamarin 2018a).

3. **Xamarin.Forms**

   O Xamarin.Forms permite a possibilidade de criar as interfaces de utilizador em C#. Esta funcionalidade não é possível no Xamarin.Android e Xamarin.iOS de maneira a ter acesso a todos os controlos nativos de cada plataforma (Xamarin 2018a).

Como mostra a Figura 3.3, usando o Xamarin.Forms é possível partilhar quase 100% do código entre as várias plataformas. Se o utilizador optar por escolher o Xamarin.Android ou Xamarin.iOS irá ter de desenvolver as interfaces específicas para cada plataforma. Contudo, a partilha de código ainda é possível, rondando os 75% de código partilhado. As chamadas às APIs de cada plataforma terão sempre de ser feita separadamente (Xamarin 2018a).

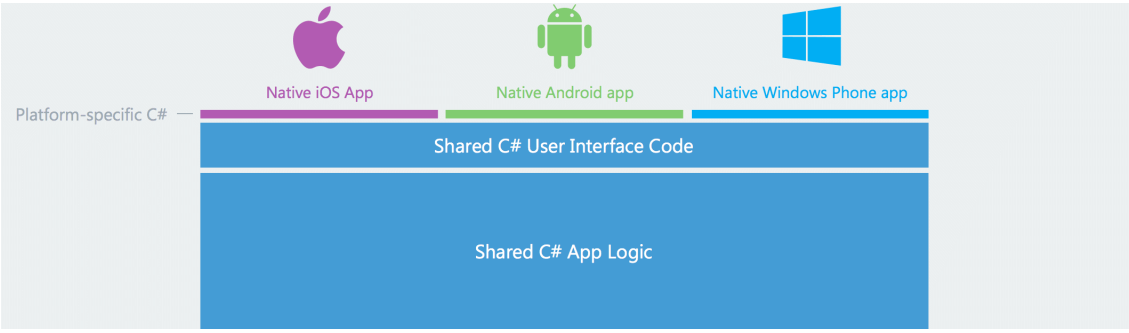

Figura 3.3: Partilha de Código usando Xamarin (Xamarin 2018a).

Partilha de Código

A abordagem mais simples para partilhar código é denominada Shared Projects. As Figuras 3.4 e 3.5 mostram a utilização desta abordagem. Neste caso é criado um projeto com três sub-projetos, um para Android, um para iOS e outro para Windows. Além destes três há um quarto projeto, designado Shared, que contem o código partilhado entre as plataformas. O projeto respetivo a cada plataforma permite o acesso às APIs específicas de cada.

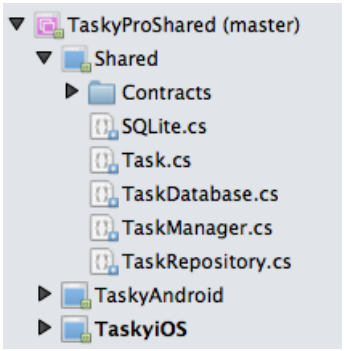

Figura 3.4: Painel do Visual Studio (Xamarin 2018b).

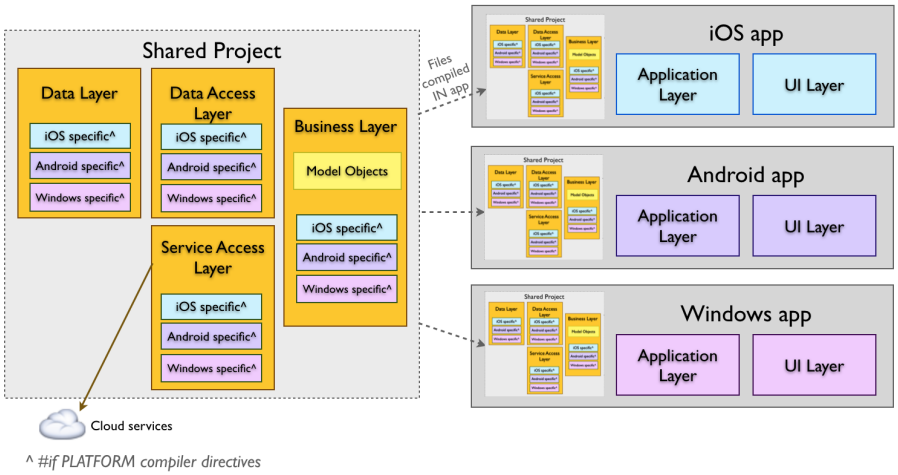

Figura 3.5: Abordagem Shared Projects (Xamarin 2018b).

C#

C#, é uma linguagem de programação, desenvolvida pela Microsoft para a *framework* .NET. Tal como o Java, é orientada a objetos. C# é considerado uma linguagem de alto nível, que contem bases de diversas linguagens, incluindo o C, C++ e Java, usando o melhor de cada uma. Para além destas bases, C# possui características próprias (Liberty 2005)

As principais características retiradas das outras linguagens são o alto desempenho do C, a estrutura orientada a objetos do C++, o GC e a segurança do Java e o desenvolvimento rápido do Visual Basic (Liberty 2005).

3.1.3 Padrões de Arquitetura

Nesta subsecção irão ser analisados os padrões de arquitetura que poderão ser usados para o desenvolvimento das aplicações.

Model View Controller (MVC)

O padrão de arquitetura mais comum no desenvolvimento de aplicações é o MVC, representado na Figura 3.6. Tal como o nome indica, é composto por três componentes fundamentais, o Model, o View e o Controller.

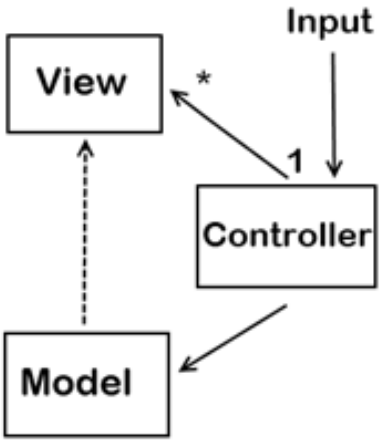

Figura 3.6: MVC

O **Model** pode ser considerado o domínio da aplicação ou a estrutura da aplicação. É onde são estruturados e geridos os dados da aplicação. Estes dados são normalmente apresentados na View e atualizados através do Controller.

A **View** pede os dados ao Model e representa-os graficamente. A ação de mostrar a View é desencadeada pelo Controller.

O **Controller** surge como interface entre os Models e Views e o input do utilizador. Através da receção do input, o Controller interage com os dados da aplicação e, por consequente, com as Views, onde são apresentados os dados, e Models, onde eles são armazenados.

Model View Presenter (MVP)

Uma das alternativas possível ao uso do padrão de arquitetura MVC, seria o padrão MVP, representado na Figura 3.7. O MVP também é composto por três componentes. Este difere do MVC, na medida em que, possui Presenters em vez de Controllers. No entanto, contém igualmente Model e Views.

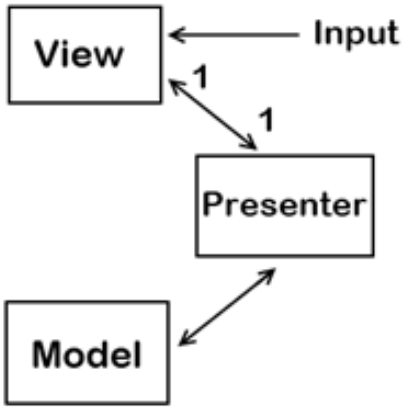

Figura 3.7: MVP

O **Model**, tal como no MVC, estrutura e gere os dados da aplicação. Neste caso apenas interage com o Presenter, fornecendo-lhe os dados e atualizando-os.

A **View**, apresenta os dados do Model que são passados pelo Presenter. Neste caso, o input do utilizador é realizado nas views, sendo depois transmitidos para o Presenter, para atuar nos Models.

O **Presenter**, interage com os Models e com as Views. Obtém os dados no Model e apresenta-os nas Views. Na perspectiva contrária, obtém os inputs do utilizador nas Views, e através deles, modifica os dados nos Models.

3.1.4 Frameworks Alternativas

Esta subsecção tem como principal objetivo estudar alternativas ao *framework* de desenvolvimento multi-plataforma escolhido, o Xamarin. Foram escolhidas outras *frameworks* relevantes no mercado, tais como, Apache Cordova, React Native e Ionic.

Apache Cordova

O Apache Cordova, anteriormente conhecido como PhoneGap, é uma *framework open-source* gratuita destinada ao desenvolvimento de aplicações móveis. Através do Apache Cordova é possível utilizar tecnologias normalmente utilizadas em web, tais como, HTML5, CSS3 e JavaScript para o desenvolvimento móvel multi-plataforma. O Apache Cordova permite o acesso a algumas funcionalidades das APIs nativas como sensores e *network*, através de *plugins* (Cordova 2018).

Esta *framework* permite o desenvolvimento para 8 plataformas, entre as quais, Android, iOS, Windows Phone 8, Blackberry, Ubuntu, Firefox OS, LG WebOS e FireOS.

Tal como apresentado na Figura 3.8, as aplicações finais são híbridas. Isto deve-se ao facto da aplicação correr no que é denominado uma *WebView*. As *WebViews* são nativas dos SO e basicamente permitem correr conteúdo Web no telemóvel através do uso do browser. O Cordova faz isto ao juntar o código Web programado e executa-o na aplicação através da *WebView*. (Cordova 2018).

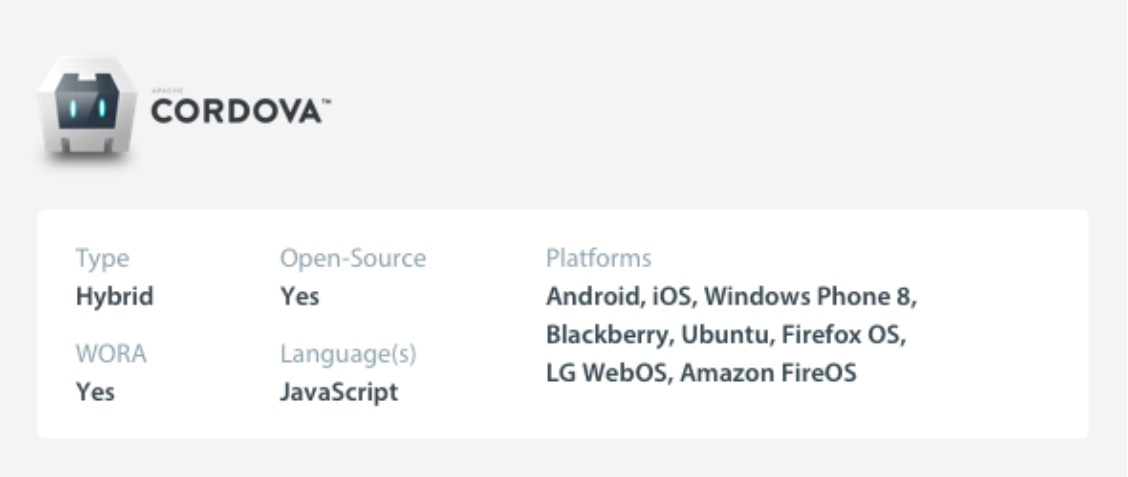

Figura 3.8: Cordova (Idesis 2017).

O facto de ser uma aplicação híbrida, permite que se use o paradigma Write Once, Run Anywhere (WORA). Isto significa que só é necessário uma codificação e que pode então ser corrida nas plataformas suportadas.

De seguida, irão ser apresentadas as suas principais vantagens e desvantagens:

Vantagens

- Acesso a diversas plataformas.
- Só é necessário uma codificação.
- *Updates* que permitem o acesso a novas APIs.

Desvantagens

- Acesso limitado às APIs nativas.
- Uso de *WebView*.
- Desempenho será bastante abaixo das aplicações nativas.
- Desenho das interfaces complicado para iniciantes.

React Native

O React Native é uma *framework open-source* gratuita, que pertence ao Facebook, que surgiu a partir do React como solução para o desenvolvimento de aplicações móveis. Como a base do React Native é o React, quem já souber desenvolver em React tem uma grande vantagem ao desenvolver em React Native. Contudo, é necessário aprender conteúdo específico do React Native, nomeadamente componentes nativos. Como React é uma biblioteca JavaScript, a linguagem principal do React Native é o JavaScript (React 2018).

A Figura 3.9 apresenta algumas das características da *framework*, nomeadamente o facto de as aplicações finais serem nativas e de não usar o paradigma WORA. Isto é devido ao facto de ter de desenvolver as interfaces específicas para cada plataforma, por um lado torna as aplicações nativas, por outro, não permite a reutilização de todo o código.

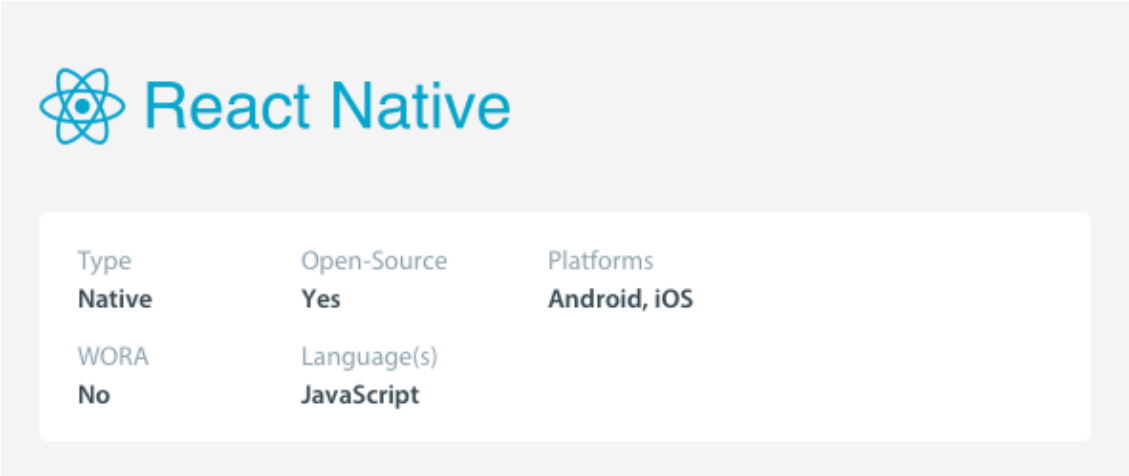

Figura 3.9: React Native (Idesis 2017).

O React Native é uma das *frameworks* mais populares e conceituadas que se destinam ao desenvolvimento de aplicações nativas através de JavaScript. Isto deve-se ao facto de pertencer ao Facebook, levando a que duas das aplicações destinadas a rede sociais mais usadas mundialmente, o Instagram e o Facebook, serem desenvolvidas em React Native.

De seguida, irão ser apresentadas as suas principais vantagens e desvantagens:

Vantagens

- Aplicações nativas.
- Bom desempenho.
- *Framework* atualizada.
- Usado pelo Facebook e Instagram.

Desvantagens

- É necessário a codificação das interfaces para cada plataforma.
- Semelhança ao React pode afastar utilizadores.

Ionic

O Ionic é uma *framework open-source* gratuita, que pertence à empresa Drifty, para o desenvolvimento de aplicação móveis híbridas. Com o Ionic, os desenvolvedores vão poder desenvolver em JavaScript e TypeScript além de possuir a integração da *framework* Angular. O acesso às funcionalidades das APIs nativas de cada plataforma são efetuadas através dos *plugins* do Apache Cordova (Ionic 2018).

Esta *framework* permite o desenvolvimento para 5 plataformas, entre as quais, Android, iOS, Windows Phone 8, Chrome e Desktop.

De acordo com a Figura 3.10, e como referido anteriormente, as aplicações finais são híbridas, ou seja, é usada uma *WebView*. No entanto, traz a vantagem do paradigma WORA, que permite apenas uma codificação.

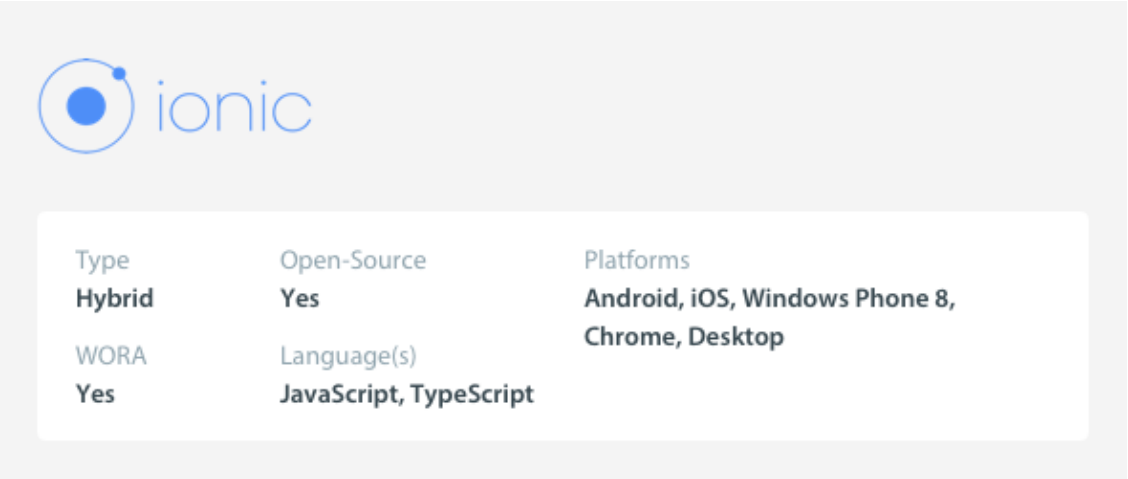

Figura 3.10: Ionic (Idesis 2017).

De seguida, irão ser apresentadas as suas principais vantagens e desvantagens:

Vantagens

- Acesso a diversas plataformas.
- Só é necessário uma codificação.
- *Updates* que permitem o acesso a novas APIs.
- Incorporação da *framework* Angular.

Desvantagens

- Acesso limitado às APIs nativas.
- Uso de *WebView*.
- Desempenho será bastante abaixo das aplicações nativas.
- Depende dos *plugins* nativos do Apache Cordova.
- Certas funcionalidades só são acedidas a partir da versão Pro.

3.2 Avaliação da Solução Proposta

Esta secção tem como principal objetivo a avaliação da solução proposta. Para tal irão ser definidas as grandezas utilizadas na avaliação, as hipóteses a testar, a metodologia de avaliação e o teste estatístico que irá ser usado.

3.2.1 Solução Proposta

No contexto específico desta dissertação, a abordagem a realizar e a solução do problema, já foram definidas pela empresa no momento da realização do projeto. Foi definido que para resolver o seu problema seria necessário comparar a atual *framework* usada para o desenvolvimento de aplicações móveis para a plataforma Android, em Java, com uma *framework* destinada ao desenvolvimento de aplicações móveis multiplataforma. No que se refere à *framework*, foi definido que seria usado o Xamarin para realizar a comparação.

Na Secção 3.1 foram identificadas possíveis alternativas à *framework* definida ao inicio. Foram analisadas as *frameworks* Apache Cordova, que pertence à Apache, React Native, que pertence ao Facebook e o Ionic, que pertence à Drifty. Ao efetuar a análise às *frameworks* referidas, reparou-se que dependo do objetivo das aplicações, todas as *frameworks* teriam as suas vantagens e desvantagens.

Em suma, há alternativas que seriam interessantes estudar em comparação às tecnologias usadas para o desenvolvimento na empresa, mas a *framework* que será testada já foi escolhida. Como tal, será necessário avaliar a solução pretendida, ou seja será necessário avaliar a comparação das diversas tecnologias escolhidas.

3.2.2 Grandezas a Avaliar

De maneira a responder à questão **Q6**, referida na Secção 1.3 do Capítulo 1, decidindo qual das tecnologias utilizadas para o desenvolvimento é mais favorável à empresa, é relevante analisar tanto as *frameworks*, como as aplicações desenvolvidas.

Para tal definiu-se um conjunto de grandezas a avaliar, que correspondem a questões definidas na Secção 1.3 do Capítulo 1:

- **Q2: Desempenho das aplicações:** esta grandeza é de elevada importância, pois o desempenho das aplicações é essencial para os utilizadores. Quanto melhor o desempenho, mais contente o utilizador ficará.
- **Q3: Custos de desenvolvimento:** esta grandeza é importante para a empresa, dado que esta irá perceber se os custos de desenvolvimento irão aumentar ou diminuir caso ocorra a mudança de *framework*.
- **Q4: Tempo de desenvolvimento:** esta grandeza será importante para a empresa perceber se irá ou não demorar mais a desenvolver as suas aplicações e se isso afetará os prazos habituais.

3.2.3 Teste de Hipóteses

Um teste de hipóteses é um teste estatístico com o objetivo de tomar uma decisão entre duas ou mais hipóteses. Neste caso, podemos considerar que μ0 representa a *framework* Android e μ1 a *framework* Xamarin. A hipótese nula 3.1 é a hipótese que se pretende rejeitar, que significa que as tecnologias são iguais no que se refere às grandezas a avaliar.

$$h0 : \mu0 = \mu1 = 50\% \tag{3.1}$$

$$h1 : \mu0 != \mu1 \tag{3.2}$$

Caso isto não aconteça e haja diferenças entre as tecnologias, hipótese 3.2, será importante avaliar as duas de maneira a perceber qual das tecnologias é mais favorável à utilização da empresa. Caso se verifique a hipótese 3.3 significa que Android nativo é melhor no aspeto comparado e, caso se verifique a hipótese 3.4 significa que Xamarin é melhor no aspeto comparado.

$$h2 : \mu0 > \mu1 \tag{3.3}$$

$$h3 : \mu0 < \mu1 \tag{3.4}$$

3.2.4 Métricas

Nesta secção irão ser apresentadas as metodologias que irão ser utilizadas no decorrer avaliação da solução, relativamente às grandezas anteriormente referidas.

Desempenho

Neste contexto, um bom desempenho é essencial nas aplicações e representa a rapidez de resposta que o utilizador necessita para ter uma interação fluída, bem como, um arranque rápido da aplicação, baixo uso de memória e bateria. De maneira a avaliar o desempenho das aplicações será necessário a realização de aplicações com as mesmas funcionalidades em ambas as *frameworks*. Após isso será necessário comparar os tempos de realização de cada funcionalidade, bem como, o tempo de resposta das aplicações. A utilização de memória e tamanho ocupado pelas aplicações também serão relevantes para este estudo.

Custos

Para obter os custos de desenvolvimento de cada *framework* será necessário obter primeiro os possíveis custos das tecnologias associadas e os custos de colocação de aplicações nas lojas, neste caso da Google Play Store. Além disto, será necessário obter o salário dos desenvolvedores de cada *framework*. Posto isto, é importante comparar estes valores com o tempo de desenvolvimento de cada aplicação na determinada *framework*.

Tempo

Com o intuito de avaliar o tempo de desenvolvimento das aplicações em cada *framework* será necessário realizar testes de usabilidade. Estes testes de usabilidade irão indicar aos utilizadores funcionalidades a realizar em cada *framework*. Assim será monitorizado o tempo de realização de cada funcionalidade. É necessário ter em conta que os utilizadores a realizar as funcionalidades nas diferentes *frameworks* possuem o mesmo grau de conhecimento.

3.2.5 Teste Estatístico

De forma a testar as hipóteses acima referidas terá de ser usado um teste estatístico. O teste escolhido foi o **t-test**. O **t-test** é um teste paramétrico que permite comparar as médias de duas amostras diferentes. Este teste permite perceber se existe, ou não, uma variância considerável entre as duas *frameworks* em estudo.

Este teste estatístico pode ser usado, por exemplo, na comparação dos tempos de resposta das funcionalidades de cada aplicação, de maneira a ajudar no apuramento de qual aplicação possui o melhor desempenho. Neste caso irão ser usadas amostras do tempo de 50 cronometragens, em milissegundos, recolhidos na execução dessas tarefas.

3.3 Sumário

Neste capítulo foi efetuado o estudo do estado da arte, onde foram estudadas e apresentadas as tecnologias que irão ser utilizadas ao longo do projeto, além das análise às alternativas às *frameworks* escolhidas.

Foi abordada a solução e as abordagens existentes. Dado que a solução e abordagem já se encontravam definidas de início, foi definido como se iria avaliar a solução. Para tal, foram definidas as grandezas a avaliar, as hipóteses e a metodologia que iria se usar para essa avaliação. No fim, definiu-se o teste estatístico que irá ser usado para comparar as *frameworks*.

Capítulo 4

Análise, Design e Desenvolvimento da Solução

No presente capítulo será apresentado a análise e o *design* destinado às aplicações que irão auxiliar a comparação das tecnologias definidas.

Serão focados os requisitos funcionais e não funcionais, bem como a arquitetura e os padrões que serão usados.

Além disto, serão apresentados excertos de código relevantes de funcionalidades das aplicações desenvolvidas. Irão ser apresentados excertos de cada uma das aplicações, sendo estes comparados entre si caso seja pertinente.

Neste capítulo pretende-se responder à questão **Q1**, na Secção 4.1, e a parte da questão **Q5**, na Secção 4.4. Estas questões foram referidas na Secção 1.3 do Capítulo 1.

4.1 Análise

Nesta secção irão ser detalhadas as funcionalidades escolhidas, bem como os requisitos funcionais e requisitos não funcionais desta aplicação.

4.1.1 Funcionalidades Escolhidas

As funcionalidades escolhidas para as aplicações desenvolvidas no âmbito desta dissertação tiveram em conta estatísticas referentes à Loja oficial da Google, Google Play Store. Não foi possível obter quais as funcionalidades mais utilizadas pelas aplicações no mercado mas, no entanto, através da análise das permissões mais usadas nas aplicações é possível ter uma ideia e fazer uma associação às funcionalidades que requerem essas permissões.

Através de um estudo, realizado em 2014 pelo Pew Research Center, a cerca de 1 milhão de aplicações disponíveis do mercado da Google foram retirados os dados presentes na Figura 4.1, ordenados por percentagem de utilização (Center 2015b).

Top App Permissions in the Google Play Store

Permission	What the Permission Does *"Allows the app to ..."*	Number of apps	% of apps	Hardware Permission or User Information
Full network access	... create network sockets and use custom network protocols. The browser and other applications provide means to send data to the internet, so this permission is not required to send data to the internet.	855,873	83%	Hardware
View network connections	...view information about network connections such as which networks exist and are connected.	714,607	69%	Hardware
Test access to protected storage	... test a permission for USB storage that will be available on future devices. Allows the app to test a permission for the SD card that will be available on future devices	562,442	54%	Hardware
Modify or delete the contents of your USB storage	...write to the USB storage. Allows the app to write to the SD card.	559,941	54%	User info
Read phone status and identity	... access the phone features of the device. This permission allows the app to determine the phone number and device IDs, whether a call is active, and the remote number connected by a call.	361,616	35%	User info
Prevent device from sleeping	... prevent the tablet from going to sleep. Allows the app to prevent the phone from going to sleep.	279,775	27%	Hardware
Precise location (GPS and network-based)	... get your precise location using the Global Positioning System (GPS) or network location sources such as cell towers and Wi-Fi. These location services must be turned on and available to your device for the app to use them. Apps may use this to determine where you are, and may consume additional battery power.	246,750	24%	User info
View Wi-Fi connections	... view information about Wi-Fi networking, such as whether Wi-Fi is enabled and name of connected Wi-Fi devices.	235,093	23%	User info
Control vibration	... control the vibrator.	220,594	21%	Hardware
Approximate location (network-based)	... get your approximate location. This location is derived by location services using network location sources such as cell towers and Wi-Fi. These location services must be turned on and available to your device for the app to use them. Apps may use this to determine approximately where you are.	216,770	21%	User info

Source: Google Play Store, June 18-Sept. 8, 2014.
Note: Descriptions of each permission are how they appear to a user.

PEW RESEARCH CENTER

Figura 4.1: 10 Permissões Mais Utilizadas (Center 2015a)

Analisando a Figura 4.1, é possível perceber que mais de metade das aplicações usam a permissão de acesso a ficheiros, acesso às conexões de rede disponíveis e acesso completo à rede, sendo o acesso completo à rede a permissão mais utilizada. Além destas, o acesso a informações do dispositivo, vibração e acesso à localização encontram-se nas 10 permissões mais utilizadas.

Após esta análise foram escolhidas 7 funcionalidades que tenham em conta estas estatísticas de permissões mais utilizadas:

- **Mapa:** o acesso ao mapa permite utilizar a permissão da localização que normalmente é usada em cerca de 1/4 das aplicações.
- **Leitura e Escrita de ficheiros:** esta permissão é usada em mais de metade das aplicações.
- **Partilha nas Redes Sociais e Visualização de Vídeos:** ambas estas funcionalidades necessitam da permissão mais utilizada, neste caso a de acesso à Internet.
- **Uso da câmara e Sensores:** estas permissões não se encontram nas mais utilizadas, mas foram consideradas relevantes por usarem hardware do dispositivo e o acesso ao mesmo ser mais facilitado em aplicações nativas.

Estas funcionalidades permitem responder à questão **Q1**, aferindo que estas são as funcionalidades que devem ser testadas para obter uma boa avaliação das *frameworks*.

4.1.2 Requisitos Funcionais

No início do análise das aplicações, foram definidos os requisitos funcionais das mesmas. Estes requisitos foram analisados em forma de funcionalidades e transformados em Use Cases (UC), como apresentado na Figura 4.2.

Os UC destas aplicações são:

- **UC01:** Utilizar o mapa para visualizar marcadores ou trajetos.
- **UC02:** Tirar fotografias.
- **UC03:** Partilhar as fotografias nas redes sociais.
- **UC04:** Testar os sensores do telemóvel.
- **UC05:** Ler ficheiros do telemóvel.
- **UC06:** Guardar ficheiros no telemóvel.
- **UC07:** Visualizar vídeos do YouTube incorporados.

Um fator importante na criação destes UCs é o teste de funcionalidades que já sejam utilizadas nas aplicações da empresa. Além disto, é importante a realização

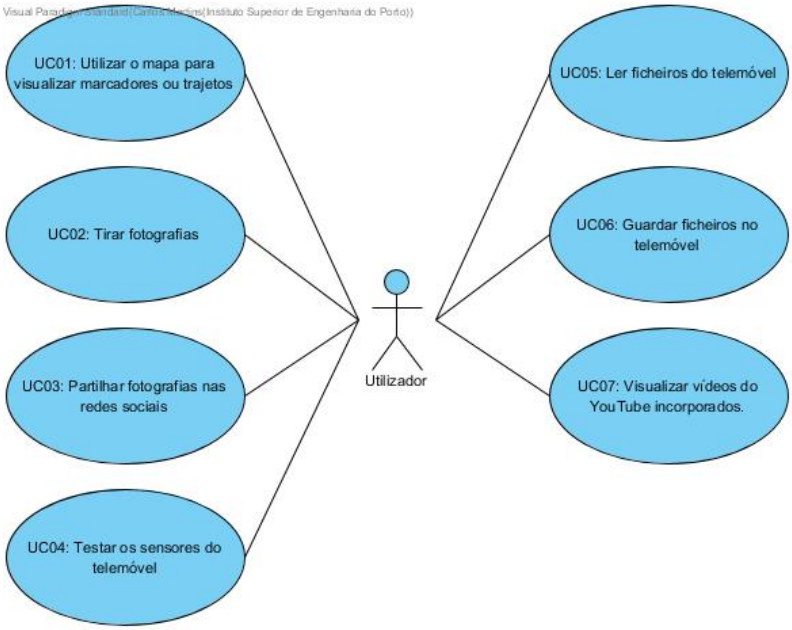

Figura 4.2: Diagrama de Casos de Uso

de uma aplicação robusta que teste funcionalidades executando tarefas que exijam um elevado desempenho por parte do dispositivo.

O primeiro caso de uso é de extrema importância, pois a utilização do mapa, mais propriamente o Google Maps, é a funcionalidade mais importante das aplicações da empresa.

Os restantes casos de uso têm o mesmo propósito do primeiro caso de uso, ou seja, ajudar na comparação das *frameworks* através do teste do desempenho da aplicação na execução de tarefas no dispositivo. Algumas das aplicações da empresa possuem certas funcionalidades semelhantes a estas.

4.1.3 Requisitos Não Funcionais

Quanto aos requisitos não funcionais é importante que a aplicação cumpra os seguintes tópicos:

- **Tecnologias** - as tecnologias envolvidas serão o Android (Java), XML e C#.
- **Desempenho** - as aplicações têm de possuir um bom desempenho e serem robustas.
- **Segurança** - é necessário garantir que as aplicações são segura e que o utilizador apenas tenha acesso ao que é pretendido.
- **Todos os dispositivos** - as aplicações devem poder ser utilizadas em todos os tipos de dispositivos, neste caso *Tablets* e *Smartphones*.

- **Guias de Material Design** - as aplicações devem seguir as regras de *Material Design* definidas pela Google.
- **Multi-plataforma (Xamarin)** - a aplicação desenvolvida em Xamarin terá de ser multi-plataforma.
- **API Level 21+ (Android)** - esta será a versão mínima do Android que irá suportar a aplicação.

4.2 Design

Nesta secção irá ser detalhado o *design* da arquitetura das aplicações a ser desenvolvidas. Será apresentado o Diagrama de Classes, Diagrama de Implantação e Diagramas de Sequência para certas funcionalidades.

4.2.1 Diagrama de Classes

A Figura 4.3, Diagrama de Classes, representa a estrutura e a relação entre as diferentes classes presentes nas aplicações. A Atividade **MenuActivity** será o centro das operações, juntamente com o Adaptador **ActionAdapter** irão decidir de acordo com a ação do utilizador qual Atividade abrir.

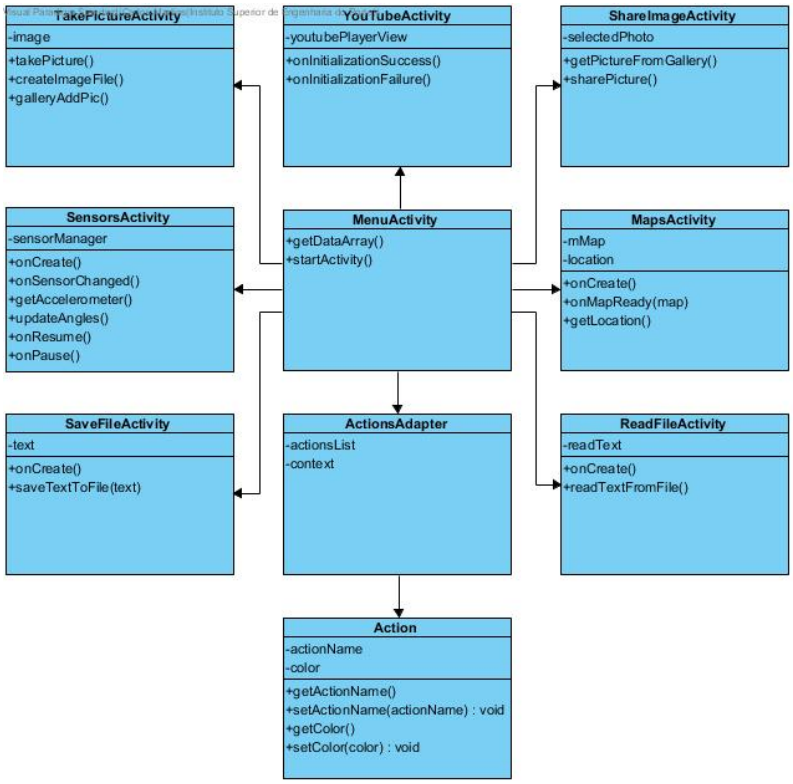

Figura 4.3: Diagrama de Classes

4.2.2 Diagrama de Implantação

O utilizador só interage diretamente com o dispositivo que contenha a aplicação. A aplicação irá permitir ao utilizador a partilha de imagens nas redes sociais como o Facebook, Twitter e Instagram e será feita a conexão com as APIs fornecidas pelas redes sociais. O mesmo se aplica à incorporação de vídeos na aplicação. Mas neste caso a conexão será efetuada com a API do YouTube. Estas conexões são efetuadas em Hyper Text Transfer Protocol Secure (HTTPS) de maneira a serem conexões seguras.

A Figura 4.4, Diagrama de Implantação, representa os componentes presentes no projeto.

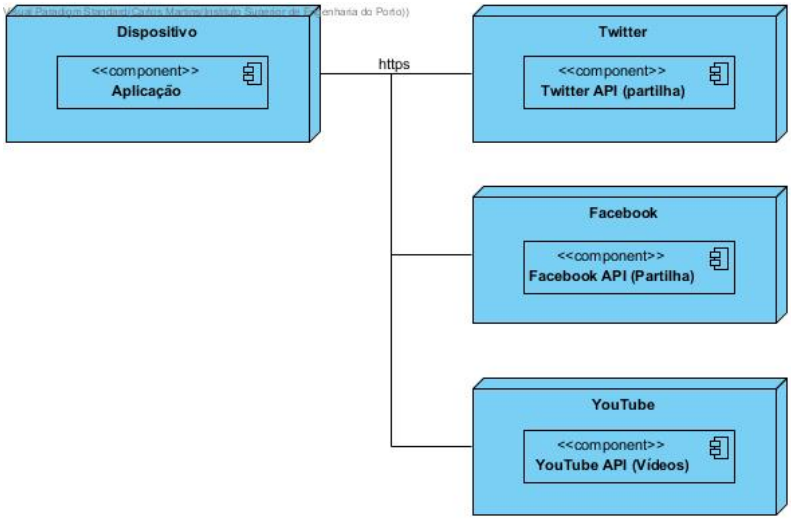

Figura 4.4: Diagrama de Implantação

4.2.3 Diagramas de Sequência

Os diagramas de sequência têm como objetivo delinear como cada UC irá funcionar, enfatizando as diferentes interações entre os objetos.

UC01

Na Figura 4.5 está apresentado o diagrama de sequência do UC01. O processo começa pela interação do utilizador com o menu com o objetivo de abrir o mapa. Após isto, a Atividade que contém o menu irá iniciar a Atividade do mapa, através do método **startActivity()**. A Atividade do mapa irá pedir o fragmento do mapa, **MapFragment**, com o método **getMapAsync()**, que por sua vez obtém o objeto **GoogleMap**, junto com a *callback* **onMapReady()** que irá retornar o mapa à Atividade assim que ele estiver disponível.

Quando o mapa estiver disponível será obtida a última localização do utilizador ao objeto **FusedLocationProviderClient**, através do método **getLastLocation()**. Por fim, será adicionado ao mapa o marcador em Sidney e o mapa será animado para essa localização.

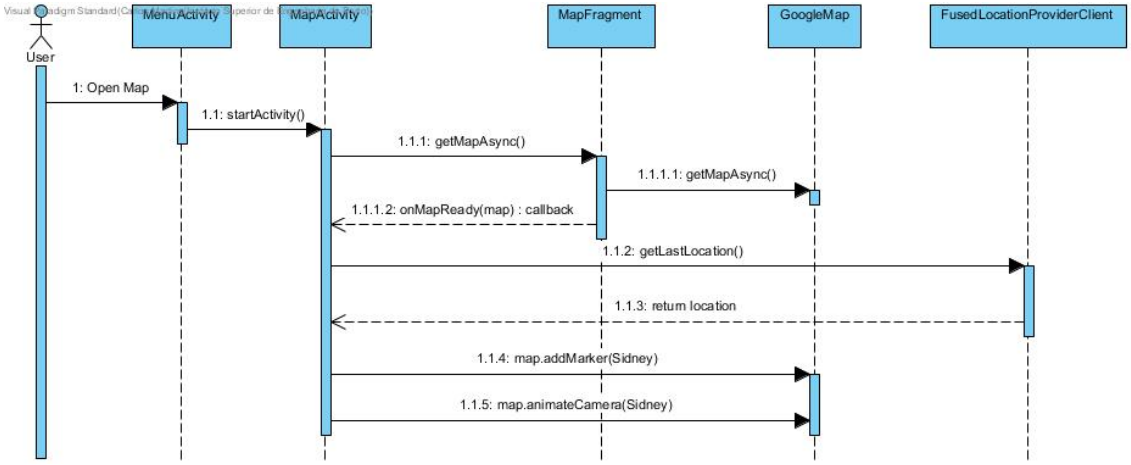

Figura 4.5: Diagrama de Sequência do UC01

UC04

Na Figura 4.6 está apresentado o diagrama de sequência do UC04. O processo começa pela interação do utilizador com o menu com o objetivo de testar os sensores. Após a entrada na Atividade dos sensores, é obtido o **SensorManager** através do método **getSystemService(SENSOR_SERVICE)**. São registados os Listeners dos sensores acelerómetro e campo magnético, com o método **registerListener()**. A cada 3ms verificar-se-à se existe alterações nos valores dos sensores e será chamada a *callback* **onSensorChanged()**.

A *callback* irá trazer um evento que será usado no método **getAccelerometer()**. Este método irá verificar se o dispositivo foi agitado ou não. Usando os valores obtidos por ambos os sensores, será invocado o método **updateOrientationAngles()** que irá obter o ângulo de rotação e atualizar a imagem da bússola de maneira a indicar o Norte.

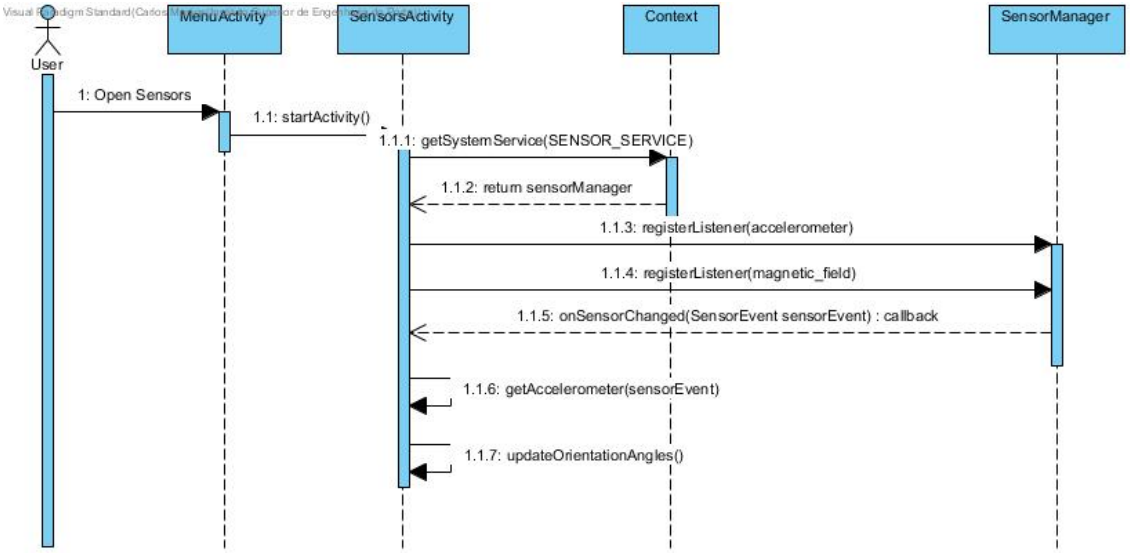

Figura 4.6: Diagrama de Sequência do UC04

UC06

Na Figura 4.7 está representado o diagrama de sequência do UC06. O processo começa pela interação do utilizador com o menu com o objetivo de guardar texto em ficheiro. Dentro da Atividade após o utilizador gravar o texto que escreveu será executado o método **saveFile(text)**. Dentro desse método será criado o **File**. Através do método **new File(path + name)** irá se verificar se o ficheiro nesse caminho e nome existe e, caso não exista, será criado.

Após o ficheiro ser criado, será criada uma *stream*, **FileOutputStream**, que irá gravar o texto no ficheiro através do método **write(text)**. Quando o ficheiro estiver escrito, a *stream* é fechada.

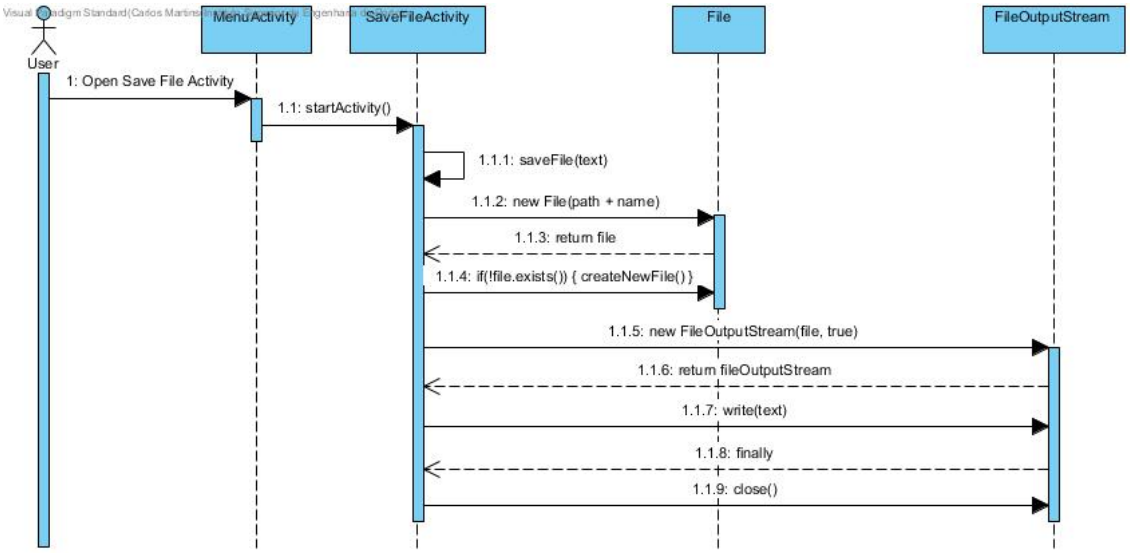

Figura 4.7: Diagrama de Sequência do UC06

UC07

Na Figura 4.8 está representado o diagrama de sequência do UC07. O processo começa pela interação do utilizador com o menu com o objetivo de visualizar um vídeo do YouTube. Após entrar na Atividade será inicializada o **YouTubePlayer** junto com a *callback* **onInitializationSuccess()**. Quando o *player* for inicializado com sucesso, a *callback* irá ser chamada e permitirá carregar um vídeo com o método **loadVideo(videoCode)**.

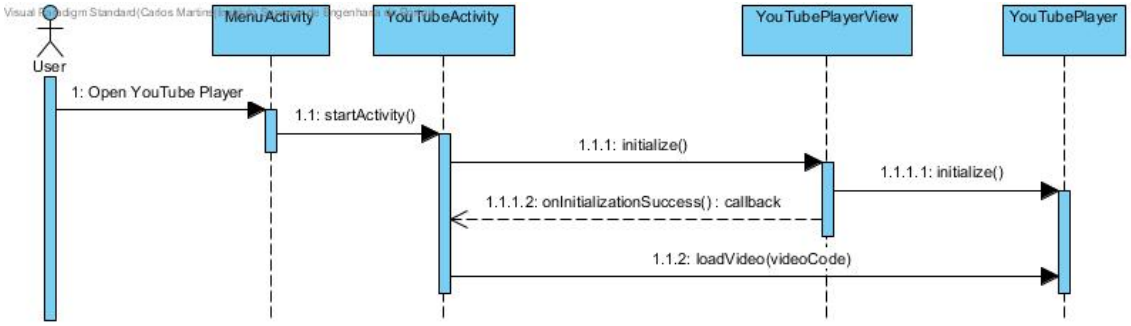

Figura 4.8: Diagrama de Sequência do UC07

4.3 Interface

Nesta secção irá ser apresentado o fluxograma da aplicação, bem como o *design* das interfaces das funcionalidades escolhidas.

4.3.1 Fluxograma

Antes de começar a desenvolver as aplicações, decidiu-se a maneira como a aplicação iria ser apresentada ao utilizador. As funcionalidades, anteriormente representadas como UCs, serão acedidas todas através de um menu principal e cada uma delas terá a opção de voltar a esse menu. Ilustrou-se isto num diagrama de fluxo, representado na Figura 4.9.

4.3.2 Desenho da Interface

O desenho da interface foi realizado tendo em conta as funcionalidades anteriormente referidas e as *guidelines* usadas pela Google na construção das suas aplicações, mais propriamente o **Material Design**. Foram desenhados diversos esboços, um para cada funcionalidade, para ter em conta aquando do desenvolvimento das interfaces da aplicação.

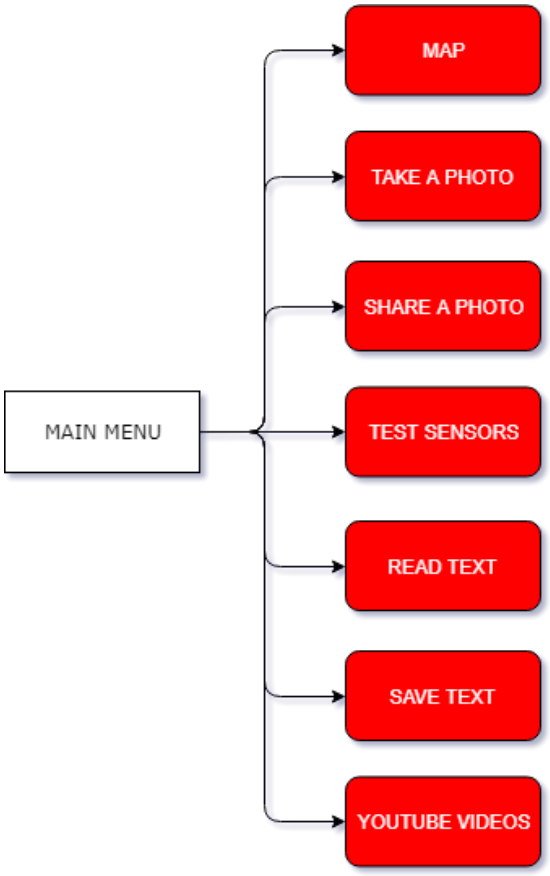

Figura 4.9: Diagrama de Fluxo

Tal como referido anteriormente, irá existir um menu principal, apresentado na Figura 4.10, que dará acesso a todas as funcionalidades da aplicação. Foi desenhado um menu simples, com botões para cada funcionalidade.

Em relação às interfaces das funcionalidades, todas permitem ao utilizador voltar ao menu principal de maneira a poder testar outras funcionalidades.

Os esboços das páginas do mapa, apresentado na Figura 4.11, e visualização de vídeos do YouTube, apresentado na Figura 4.12, são simples e apresentam apenas o mapa e o vídeo, ocupando maior parte do ecrã.

Os esboços das páginas de captura e partilha de imagens, apresentados na Figura 4.13 e Figura 4.14 respetivamente, serão semelhantes, mas a de captura irá permitir ao utilizador usar a câmara do telemóvel através do botão no fundo do ecrã, enquanto que a partilha de imagens irá permitir ao utilizador escolher uma imagem para partilhar aquando da entrada na página e usar o botão no fundo do ecrã para permitir a partilha da fotografia nas redes sociais.

O esboço da página dos sensores, apresentado na Figura 4.15, mostrará uma mensagem quando o utilizador abanar o telemóvel e possuirá uma bússola que mostrará o Norte de acordo com a rotação do telemóvel.

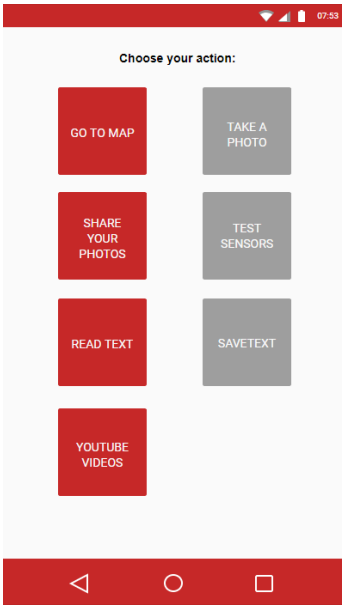

Figura 4.10: Esboço do Menu Principal

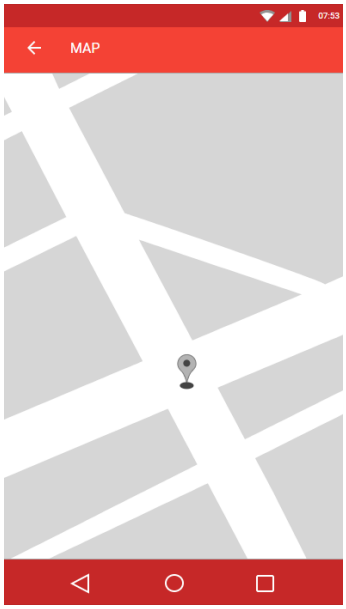

Figura 4.11: Esboço da Página do Mapa

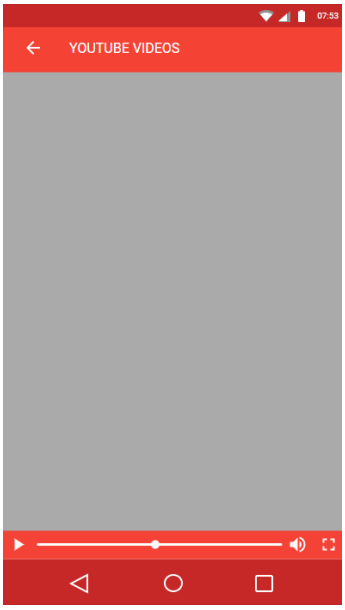

Figura 4.12: Esboço da Página de Visualização de Vídeos

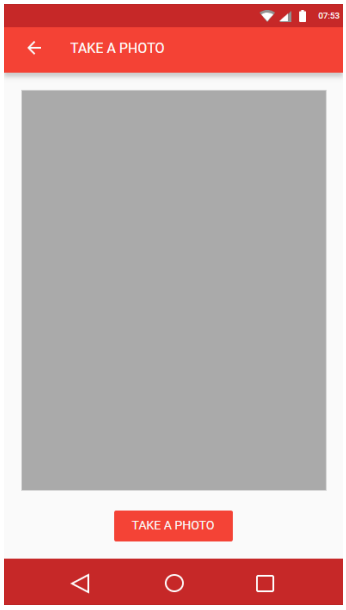

Figura 4.13: Esboço da Página de Tirar Fotografias

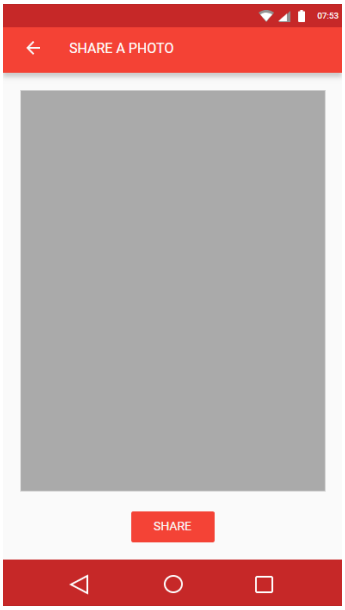

Figura 4.14: Esboço da Página de Partilha de Fotografias

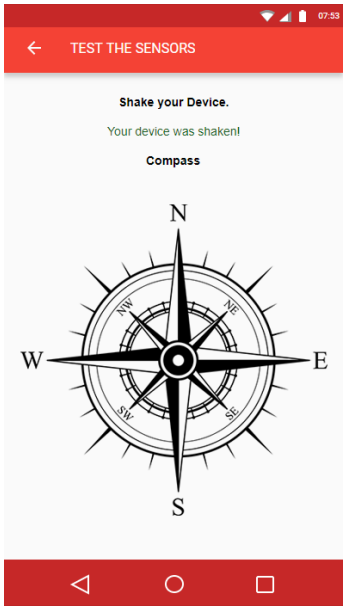

Figura 4.15: Esboço da Página de Teste de Sensores

Os esboços das páginas de escrever e ler texto, apresentados na Figura 4.16 e Figura 4.17, em ficheiros serão semelhantes, mas uma irá permitir guardar o texto escrito através de um botão no topo do ecrã e a outra apenas a leitura do texto anteriormente escrito.

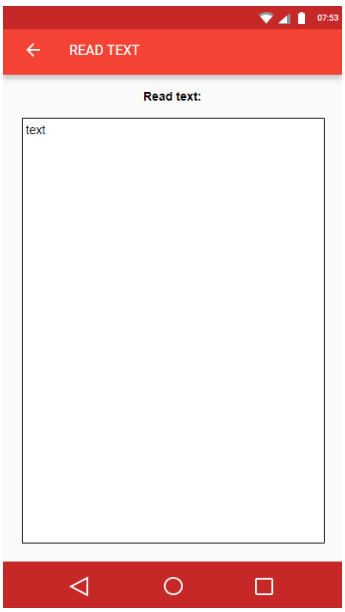

Figura 4.16: Esboço da Página de Leitura de Texto

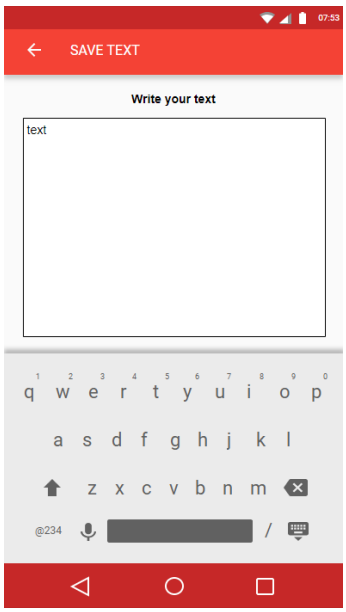

Figura 4.17: Esboço da Página de Escrita de Texto

4.4 Desenvolvimento

Nesta secção serão apresentados excertos de código relevantes de funcionalidades das aplicações desenvolvidas. Irão ser apresentados excertos de cada uma das aplicações, sendo estes comparados entre si caso seja pertinente.

4.4.1 Mapa e Localização

Aquando do planeamento da funcionalidade UC01 decidiu-se utilizar o mapa da Google, Google Maps. Era esperado que a codificação desta funcionalidade em Xamarin fosse de uma dificuldade mais elevada, dado não se estar a trabalhar nativamente, que é o caso de Android nativo.

De seguida, serão apresentados os excertos de código necessários em cada uma das *frameworks*.

Xamarin

```
1 MapFragment mapFrag = (MapFragment)FragmentManager.FindFragmentById
      (Resource.Id.myMap);
2 mapFrag.GetMapAsync(this);
```

Código 4.1: Instanciação do Mapa - Xamarin

No início, apresentado Excerto 4.1, obtém-se o MapFragment, na Linha 1, onde o mapa irá ser contido e após isso obtém-se o mapa assincronamente, na Linha 2. Quando o mapa estiver disponível irá ser invocado o método **OnMapReady()**, disponível no Excerto 4.2, e aí poder-se-á obter a localização bem como adicionar marcadores ao mapa. Neste caso invocou-se o método apresentado no Excerto 4.3, na Linha 4, criou-se uma variável, **sydney**, com a latitude e longitude de Sidney, na Linha 5, e adicionou-se um marcador nessa posição na Linha 6.

```
1 public void OnMapReady(GoogleMap googleMap)
2 {
3   mMap = googleMap;
4   updateLocationUI();
5   LatLng sydney = new LatLng(-34, 151);
6   mMap.AddMarker(new MarkerOptions().SetPosition(sydney).SetTitle("
      Marker in Sydney"));
7 }
```

Código 4.2: Callback OnMapReady - Xamarin

A função com o intuito de obter a localização é relativamente simples, como apresentado no Excerto 4.3, sendo apenas necessário ativar a localização, na Linha 4, e mostrar o botão que permita obter a localização atual do utilizador, na Linha 5.

Isto só será possível fazer se o utilizador der a permissão de acesso à localização, que é verificado nas linhas 2 e 3.

```
1 private void updateLocationUI(){
2   const string permission = Manifest.Permission.AccessFineLocation;
3   if (CheckSelfPermission(permission) == (int)Permission.Granted){
4     mMap.MyLocationEnabled = true;
5     mMap.UiSettings.MyLocationButtonEnabled = true;
6   }
7 }
```

Código 4.3: Obter localização - Xamarin

Android

Em relação ao Android, no Excerto 4.4, obtém-se o SupportMapFragment, na Linha 1, onde o mapa irá ser contido e após isso obtém-se o mapa assincronamente, na Linha 2. Quando o mapa estiver disponível irá ser invocado o método **onMapReady()**, disponível no Excerto 4.5.

```
1 SupportMapFragment mapFragment = (SupportMapFragment)
    getSupportFragmentManager().findFragmentById(R.id.map);
2 mapFragment.getMapAsync(this);
```

Código 4.4: Instanciação do Mapa - Android

No Excerto 4.5, são invocados os métodos **updateLocationUI()** e **getDeviceLocation()**, apresentados no Excerto 4.6, nas Linhas 4 e 5. Criou-se uma variável, **sydney**, com a latitude e longitude de Sidney, na Linha 6, e adicionou-se um marcador nessa posição na Linha 7.

```
1 @Override
2 public void onMapReady(GoogleMap googleMap) {
3   mMap = googleMap;
4   updateLocationUI();
5   getDeviceLocation();
6   LatLng sydney = new LatLng(-34, 151);
7   mMap.addMarker(new MarkerOptions().position(sydney).title("Marker
      in Sydney"));
8 }
```

Código 4.5: Callback onMapReady - Android

No Excerto 4.6 estão presentes os métodos necessários para obter a localização do utilizador.

O método **updateLocationUI()** começa por verificar se a instância do mapa existe, na Linha 2. Depois na Linha 5 verifica se as permissões foram autorizadas e, caso

tenham sido, ativa a localização, na Linha 6, e mostra o botão que permita obter a localização atual do utilizador, na Linha 7. Caso não seja dado a permissão, esta é pedida de novo na Linha 12.

O método **getDeviceLocation()** é responsável por obter a ultima localização possível do utilizador através da Task apresentada na Linha 21. Na Linha 25 verifica-se se a Task foi bem sucedida e, caso tenha sido, na Linha 26 atribui-se à variável global **mLastKnownLocation** a localização do utilizador. Após possuir a localização é responsável por mover o mapa para a posição atual do utilizador, na Linha 28.

```
1  private void updateLocationUI() {
2    if (mMap == null)
3      return;
4    try {
5      if (mLocationPermissionGranted) {
6        mMap.setMyLocationEnabled(true);
7        mMap.getUiSettings().setMyLocationButtonEnabled(true);
8      } else {
9        mMap.setMyLocationEnabled(false);
10       mMap.getUiSettings().setMyLocationButtonEnabled(false);
11       mLastKnownLocation = null;
12       getLocationPermission();
13     }
14   } catch (SecurityException e)  {
15     Log.e("Exception: %s", e.getMessage());
16   }
17 }
18 private void getDeviceLocation() {
19   try {
20     if (mLocationPermissionGranted) {
21       Task locationResult = mFusedLocationProviderClient.
     getLastLocation();
22       locationResult.addOnCompleteListener(this, new
     OnCompleteListener() {
23           @Override
24           public void onComplete(@NonNull Task task) {
25             if (task.isSuccessful()) {
26               mLastKnownLocation = (Location)task.getResult();
27               if(mLastKnownLocation != null) {
28                 mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(new
     LatLng(mLastKnownLocation.getLatitude(), mLastKnownLocation.
     getLongitude()), DEFAULT_ZOOM));
29               }
30             }
31       }});
32     }
33   } catch(SecurityException e)  {
34     Log.e("Exception: %s", e.getMessage());
35   }
36 }
```

Código 4.6: Obter localização - Android

Conclusão

No caso específico do UC01, pode-se verificar que as linhas de código são menores na aplicação realizada na *framework* Xamarin, sendo necessário um método extra para obter a localização em Android.

4.4.2 Câmara

De seguida, serão apresentados os excertos de código necessários para a funcionalidade UC02 em cada uma das *frameworks*.

Xamarin

O Excerto 4.7 representa a ação efetuada no clique do botão para tirar foto. O método **createImageFile()**, invocado na Linha 2, cria um ficheiro do formato **.jpg** numa pasta denominada **"AppTeseXamarin"** na raíz do armazenamento. Após isto é criado um Intent **intent**, na Linha 6, para a Atividade da Câmara que leva como um dos parâmetros o caminho para o ficheiro criado, introduzido na Linha 7. É iniciado a Atividade à espera de um resultado, na Linha 9. Quando esse resultado existir será invocado o método exemplificado no Excerto 4.8.

```
1  takePicture.Click += delegate {
2    File file = createImageFile();
3
4    photoUri = FileProvider.GetUriForFile(Application.Context,
       Application.Context.PackageName + ".provider", file);
5
6    Intent intent = new Intent(MediaStore.ActionImageCapture);
7    intent.PutExtra(MediaStore.ExtraOutput, photoUri);
8    intent.PutExtra("return-data", true);
9    StartActivityForResult(intent, REQUEST_IMAGE_CAPTURE);
10 };
```

Código 4.7: Ação do botão Tirar Fotografia - Xamarin

O método representado no Excerto 4.8 é invocado após o utilizador tirar a fotografia. É usado o método **GetBitmap()**, na Linha 9, que através do caminho para o ficheiro cria um Bitmap **mBitmap**, que representa um conjunto de pixeis que possuem informação de cores, criando uma imagem. Após isto é invocado o método **rotateImageIfRequired()**, na Linha 11, que verifica a orientação da imagem e a roda, caso seja necessário. O Bitmap resultante é colocado numa *view*, na Linha 12, de maneira que o utilizador a consiga visualizar. Por fim, a imagem é adicionada à galeria do dispositivo invocando o método **galleryAddPic()** na Linha 13.

```
protected override void OnActivityResult(int requestCode, [
    GeneratedEnum] Result resultCode, Intent data)
{
  base.OnActivityResult(requestCode, resultCode, data);
  if (requestCode == REQUEST_IMAGE_CAPTURE && resultCode == Result.
   Ok)
  {
    BitmapFactory.Options options = new BitmapFactory.Options();

    Bitmap mBitmap = null;
    mBitmap = MediaStore.Images.Media.GetBitmap(this.
   ContentResolver, photoUri);

    mBitmap = Utility.Utility.rotateImageIfRequired(mBitmap,
   mCurrentPhotoPath);
    photo.SetImageBitmap(mBitmap);
    galleryAddPic();
  }
}
```

Código 4.8: Resultado da Atividade da Câmara - Xamarin

Android

Na *framework* Android, a ação do clique no botão invoca o método **takePicture()** apresentado no Excerto 4.9. Na Linha 2 é criado um Intent **takePictureIntent** pra Atividade da Câmara. Na Linha 6 é invocado o método **createImageFile()** que cria um ficheiro do formato **.jpg** numa pasta denominada **"AppTeseAndroid"** na raíz do armazenamento. Verifica-se se o ficheiro criado existe na Linha 12 e, caso exista, é adicionado ao Intent o caminho para o ficheiro criado, na Linha 16, e inicia-se a atividade à espera de um resultado, na Linha 18. Quando esse resultado existir será invocado o método exemplificado no Excerto 4.10.

```
private void takePicture() {
  Intent takePictureIntent = new Intent(MediaStore.
   ACTION_IMAGE_CAPTURE);
  if (takePictureIntent.resolveActivity(getPackageManager()) !=
   null) {
    File photoFile = null;
    try {
      photoFile = createImageFile();
    } catch (IOException ex) {
      ex.printStackTrace();
    }

    // Continue only if the File was successfully created
    if (photoFile != null) {
      photoURI = FileProvider.getUriForFile(this,
      BuildConfig.APPLICATION_ID + ".provider",
```

```
15          photoFile);
16          takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT, photoURI)
        ;
17          takePictureIntent.putExtra("return-data", true);
18          startActivityForResult(takePictureIntent,
        REQUEST_IMAGE_CAPTURE);
19        }
20      }
21  }
```

Código 4.9: Ação do botão Tirar Fotografia - Android

O método, apresentado no Excerto 4.10, é invocado após o utilizador tirar a fotografia. É criada uma InputStream **input**, na Linha 8, com o caminho para o ficheiro criado. Esta variável é enviada como parâmetro para o método **decodeStream()** que a descodifica e retorna um Bitmap, que é associado à variável **imageBitmap**. O Bitmap resultante é colocado numa *view*, na Linha 11, de maneira que o utilizador a consiga visualizar. Por fim, a imagem é adicionada à galeria do dispositivo invocando o método **galleryAddPic()** na Linha 12.

```
1   @Override
2   protected void onActivityResult(int requestCode, int resultCode,
        Intent data) {
3     if (requestCode == REQUEST_IMAGE_CAPTURE && resultCode ==
        RESULT_OK) {
4       try {
5
6         BitmapFactory.Options options = new BitmapFactory.Options();
7
8         InputStream input = getContentResolver().openInputStream(
        photoURI);
9         Bitmap imageBitmap = BitmapFactory.decodeStream(input, null,
        options);
10
11        photo.setImageBitmap(imageBitmap);
12        galleryAddPic();
13      } catch (FileNotFoundException e) {
14        e.printStackTrace();
15      }
16    }
17  }
```

Código 4.10: Resultado da Atividade da Câmara - Android

Conclusão

No caso específico do UC02, pode-se verificar que as linhas de código são menores na aplicação realizada na *framework* Xamarin, sendo que em Android são necessárias certas verificações extra obrigatórias.

4.4.3 Sensores

De seguida, serão apresentados os excertos de código necessários para a funcionalidade UC04 em cada uma das *frameworks*.

Xamarin

Na Callback de criação da Atividade, **OnCreate()**, é instanciada na Linha 1 uma variável global, **sensorManager**, do gestor de sensores, SensorManager, apresentado no Excerto 4.11. É também guardado numa variável global, **lastUpdate** na Linha 2, o atual tempo em milissegundos, que irá ser usado posteriormente.

```
1 sensorManager = (SensorManager)GetSystemService(SensorService);
2 lastUpdate = Java.Lang.JavaSystem.CurrentTimeMillis();
```

Código 4.11: Instanciação do SensorManager - Xamarin

Na Callback **OnResume()**, apresentada no Excerto 4.12, são registados os Listeners dos sensores necessários. Neste caso serão utilizados os sensores do acelerómetro, Linha 3, e campo magnético, Linha 4, com o objetivo de poder testar se o dispositivo é agitado e de criar uma bússola. Os Listeners irão chamar o método **OnSensorChanged()** sempre que existir um evento num sensor. Isto irá permitir obter os valores dos sensores.

Na Callback **OnPause()**, apresentado no Excerto 4.12, são retirados os Listeners na Linha 9. Isto irá ser necessário para sempre que o utilizador entre na Atividade, registe os Listeners e sempre que o utilizador saia retire os mesmos.

```
1  protected override void OnResume(){
2    base.OnResume();
3    sensorManager.RegisterListener(this, sensorManager.
       GetDefaultSensor(SensorType.Accelerometer), SensorDelay.Normal);
4    sensorManager.RegisterListener(this, sensorManager.
       GetDefaultSensor(SensorType.MagneticField), SensorDelay.Normal);
5  }
6
7  protected override void OnPause(){
8    base.OnPause();
9    sensorManager.UnregisterListener(this);
10 }
```

Código 4.12: Callbacks onResume e onPause - Xamarin

No método **OnSensorChanged()**, apresentado no Excerto 4.13, são copiados os valores dos sensores para dois *arrays* distintos, um para cada sensor nas Linhas 7 e 11, para usar no cálculo da rotação da bússola. Caso o evento disparado seja no acelerómetro, verificado na Linha 3, irá ser calculada a aceleração do dispositivo no

método **getAccelerometer()**, invocado na Linha 5 e apresentado no Excerto 4.14. Sempre que for disparado um evento o ângulo da bússola será atualizado de acordo com os valores dos dois sensores. O ângulo de rotação é calculado no método **updateOrientationAngles()**, invocado na Linha 13 e apresentado no Excerto 4.15.

```
1  public void OnSensorChanged(SensorEvent sensorEvent)
2  {
3    if (sensorEvent.Sensor.Type == SensorType.Accelerometer)
4    {
5      getAccelerometer(sensorEvent);
6
7      sensorEvent.Values.CopyTo(mAccelerometerReading, 0);
8    }
9    else if (sensorEvent.Sensor.Type == SensorType.MagneticField)
10   {
11     sensorEvent.Values.CopyTo(mMagnetometerReading, 0);
12   }
13   updateOrientationAngles();
14 }
```

Código 4.13: Método onSensorChanged - Xamarin

O método **getAccelerometer()**, apresentado no Excerto 4.14, calcula na Linha 9 a raíz quadrada da aceleração e guarda o valor na variável **accelationSquareRoot**. Verifica se esse valor é maior ou igual que 2, na Linha 11, e verifica se o tempo atual menos o valor da variável global **lastUpdate** é menor que 200, na Linha 12. Se for, sai do método. Caso não seja, é apresentada uma mensagem ao utilizador a dizer que o dispositivo foi agitado, na Linha 16.

```
1  private void getAccelerometer(SensorEvent sensorEvent)
2  {
3    IList<float> values = sensorEvent.Values;
4    // Movement
5    float x = values[0];
6    float y = values[1];
7    float z = values[2];
8
9    float accelationSquareRoot = (x * x + y * y + z * z) / (
       SensorManager.GravityEarth * SensorManager.GravityEarth);
10   long actualTime = sensorEvent.Timestamp;
11   if (accelationSquareRoot >= 2) {
12     if (actualTime - lastUpdate < 200) {
13       return;
14     }
15     lastUpdate = actualTime;
16     shakeMessage.Visibility = ViewStates.Visible;
17   }
18 }
```

Código 4.14: Método getAccelerometer - Xamarin

De acordo com o ângulo calculado na Linha 3, será criada uma animação RotateAnimation com esse ângulo, denominada **rotateAnimation**, na Linha 4 e a imagem da bússola será animada na Linha 11 com essa animação.

```
1  private void updateOrientationAngles() {
2    if (SensorManager.GetRotationMatrix(mRotationMatrix, null,
     mAccelerometerReading, mMagnetometerReading)) {
3      float degree = (float)(Java.Lang.Math.ToDegrees(SensorManager.
     GetOrientation(mRotationMatrix, mOrientationAngles)[0]) + 360) %
       360;
4      RotateAnimation rotateAnimation = new RotateAnimation(
5        currentDegree,
6        -degree,
7        Dimension.RelativeToSelf, 0.5f,
8        Dimension.RelativeToSelf, 0.5f);
9      rotateAnimation.Duration = 210;
10     rotateAnimation.FillAfter = true;
11     compassView.StartAnimation(rotateAnimation);
12     currentDegree = -degree;
13   }
14 }
```

Código 4.15: Método updateOrientationAngles - Xamarin

Android

No caso do UC04 o código em Android, apresentado no Excerto 4.16, na Callback de criação da Atividade, **onCreate()**, é instanciada na Linha 1 uma variável global, **sensorManager**, do gestor de sensores, SensorManager. É também guardado numa variável global, **lastUpdate** na Linha 2, o atual tempo em milissegundos, que irá ser usado posteriormente.

```
1 sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
2 lastUpdate = System.currentTimeMillis();
```

Código 4.16: Instanciação do SensorManager - Android

Na Callback **onResume()**, apresentada no Excerto 4.17, são registados os Listeners dos sensores necessários. Neste caso serão utilizados os sensores do acelerómetro, Linha 4, e campo magnético, Linha 5, com o objetivo de poder testar se o dispositivo é agitado e de criar uma bússola. Os Listeners irão chamar o método **onSensorChanged()** sempre que existir um evento num sensor. Isto irá permitir obter os valores dos sensores.

Na Callback **onPause()**, apresentado no Excerto 4.17, são retirados os Listeners na Linha 11. Isto irá ser necessário para, sempre que o utilizador entre na Atividade, registar os Listeners, e sempre que o utilizador saia retirar os mesmos.

```
1  @Override
2  protected void onResume() {
3    super.onResume();
4    sensorManager.registerListener(this, sensorManager.
       getDefaultSensor(Sensor.TYPE_ACCELEROMETER), SensorManager.
       SENSOR_DELAY_NORMAL);
5    sensorManager.registerListener(this, sensorManager.
       getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD), SensorManager.
       SENSOR_DELAY_NORMAL);
6  }
7
8  @Override
9  protected void onPause() {
10   super.onPause();
11   sensorManager.unregisterListener(this);
12 }
```

Código 4.17: Callbacks onResume e onPause - Android

No método **onSensorChanged()**, apresentado no Excerto 4.18, são copiados os valores dos sensores para dois *arrays* distintos, um para cada sensor nas Linhas 5 e 7, para usar no cálculo da rotação da bússola. Caso o evento disparado seja no acelerómetro, verificado na Linha 3, irá ser calculada a aceleração do dispositivo no método **getAccelerometer()**, invocado na Linha 4 e apresentado no Excerto 4.19. Sempre que for disparado um evento o ângulo da bússola será atualizado de acordo com os valores dos dois sensores. O ângulo de rotação é calculado no método **updateOrientationAngles()**, invocado na Linha 9 e apresentado no Excerto 4.20.

```
1  @Override
2  public void onSensorChanged(SensorEvent sensorEvent) {
3    if (sensorEvent.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
4      getAccelerometer(sensorEvent);
5      System.arraycopy(sensorEvent.values, 0, mAccelerometerReading,
       0, mAccelerometerReading.length);
6    } else if (sensorEvent.sensor.getType() == Sensor.
       TYPE_MAGNETIC_FIELD) {
7      System.arraycopy(sensorEvent.values, 0, mMagnetometerReading,
       0, mMagnetometerReading.length);
8    }
9    updateOrientationAngles();
10 }
```

Código 4.18: Método onSensorChanged - Android

O método **getAccelerometer()**, apresentado no Excerto 4.19, calcula na Linha 8 a raíz quadrada da aceleração e guarda o valor na variável **accelationSquareRoot**. Verifica se esse valor é maior ou igual que 2, na Linha 10, e verifica se o tempo atual menos o valor da variável global **lastUpdate** é menor que 200, na Linha 11. Se for, sai do método. Caso não seja, é apresentada uma mensagem ao utilizador a dizer que o dispositivo foi agitado, na Linha 15.

```
1  private void getAccelerometer(SensorEvent event) {
2    float[] values = event.values;
3    // Movement
4    float x = values[0];
5    float y = values[1];
6    float z = values[2];
7
8    float accelationSquareRoot = (x * x + y * y + z * z) / (
       SensorManager.GRAVITY_EARTH * SensorManager.GRAVITY_EARTH);
9    long actualTime = event.timestamp;
10   if (accelationSquareRoot >= 2) {
11     if (actualTime - lastUpdate < 200) {
12       return;
13     }
14     lastUpdate = actualTime;
15     shakeMessage.setVisibility(View.VISIBLE);
16   }
17 }
```

Código 4.19: Método getAccelerometer - Android

No método **updateOrientationAngles()**, apresentado no Excerto 4.20, acordo com o ângulo calculado na Linha 3, será criada uma animação RotateAnimation com esse ângulo, denominada **rotateAnimation**, na Linha 4 e a imagem da bússola será animada na Linha 11 com essa animação.

```
1  public void updateOrientationAngles() {
2    if(SensorManager.getRotationMatrix(mRotationMatrix, null,
       mAccelerometerReading, mMagnetometerReading)) {
3      float degree = (float) (Math.toDegrees(SensorManager.
       getOrientation(mRotationMatrix, mOrientationAngles)[0]) + 360) %
        360;
4      RotateAnimation rotateAnimation = new RotateAnimation(
5        currentDegree,
6        -degree,
7        Animation.RELATIVE_TO_SELF, 0.5f,
8        Animation.RELATIVE_TO_SELF, 0.5f);
9      rotateAnimation.setDuration(210);
10     rotateAnimation.setFillAfter(true);
11     compassView.startAnimation(rotateAnimation);
12     currentDegree = - degree;
13   }
14 }
```

Código 4.20: Método updateOrientationAngles - Android

Conclusão

No caso específico do UC04, pode-se verificar que as linhas de código são idênticas em ambas as aplicações e, como tal, não se pode tirar uma conclusão sobre qual

das duas necessita de mais ou menos código.

4.4.4 Guardar e Ler Ficheiros

De seguida, serão apresentados os excertos de código necessários para a funcionalidade UC05 e UC06 em cada uma das *frameworks*.

Xamarin

O Excerto 4.21, apresenta o método de guardar texto num ficheiro. Começa-se por obter o texto que o utilizador introduziu no campo de texto, na Linha 2. A seguir obtém-se o caminho da raíz do armazenamento do telemóvel, na Linha 4. Verifica-se na Linha 7 se a pasta **"AppTeseXamarin"** existe e, caso não exista, esta é criada na Linha 8. O mesmo é realizado aquando da criação do ficheiro de texto **text.txt**, nas Linhas 11 e 12. Após o ficheiro estar criado ou lido, o texto do utilizador é então gravado no ficheiro na Linha 15, usando a stream FileOutputStream, ou seja, através de um fluxo de dados.

```
1  private void SaveFile() {
2    String text = textInput.Text.ToString();
3    try {
4      String rootPath = Android.OS.Environment.
     ExternalStorageDirectory.AbsolutePath;
5
6      var dir = new File(Android.OS.Environment.
     ExternalStorageDirectory.AbsolutePath, "AppTeseXamarin");
7      if (!dir.Exists()) {
8        dir.Mkdirs();
9      }
10     var file = new File(dir + "/text.txt");
11     if (!file.Exists()) {
12       file.CreateNewFile();
13     }
14     FileOutputStream fileOutputStream = new FileOutputStream(file,
     true);
15     fileOutputStream.Write(Encoding.ASCII.GetBytes(text + Java.Lang
     .JavaSystem.GetProperty("line.separator")));
16   }
17   catch (FileNotFoundException ex) {
18     ex.PrintStackTrace();
19   }
20   catch (IOException ex) {
21     ex.PrintStackTrace();
22   }
23 }
```

Código 4.21: Guarda Texto em Ficheiro - Xamarin

O Excerto 4.22, apresenta o método de ler texto a partir do ficheiro anteriormente gravado. Os primeiros passos são semelhantes. Obtém-se o caminho da raíz do dispositivo na Linha 5 e verifica-se se o ficheiro anteriormente criado existe na Linha 7. Caso exista é usada uma FileStream **fileInputStream** e um InputStreamReader **inputStreamReader** para ir lendo as linhas enquanto existam, no ciclo representado na Linha 14. Enquanto existirem linhas no ficheiro estas são adicionadas, na Linha 15, a uma variável de texto, String, denominada **line**. No final o texto lido é apresentado ao utilizador adicionando-o ao campo de texto, na Linha 22.

```
1  private void ReadTextFromFile()
2  {
3    String line = "";
4    try {
5      String rootPath = Android.OS.Environment.
     ExternalStorageDirectory.AbsolutePath;
6      File file = new File(rootPath + "/AppTeseXamarin/text.txt");
7      if (file.Exists()) {
8
9        System.IO.FileStream fileInputStream = new System.IO.
     FileStream(file.AbsolutePath, System.IO.FileMode.Open);
10       InputStreamReader inputStreamReader = new InputStreamReader(
     fileInputStream);
11       BufferedReader bufferedReader = new BufferedReader(
     inputStreamReader);
12       StringBuilder stringBuilder = new StringBuilder();
13
14       while ((line = bufferedReader.ReadLine()) != null) {
15         stringBuilder.Append(line + Java.Lang.JavaSystem.
     GetProperty("line.separator"));
16       }
17       fileInputStream.Close();
18       line = stringBuilder.ToString();
19
20       bufferedReader.Close();
21
22       textInput.Text = line;
23     }
24   }
25   catch (FileNotFoundException ex) {
26     ex.PrintStackTrace();
27   }
28   catch (IOException ex) {
29     ex.PrintStackTrace();
30   }
31 }
```

Código 4.22: Ler Texto de Ficheiro - Xamarin

Android

O Excerto 4.23, apresenta o método de guardar texto num ficheiro. Começa-se por obter o texto que o utilizador introduziu no campo de texto, na Linha 2. Obtém-se o caminho da raíz do armazenamento do telemóvel, na Linha 4. Verifica-se na Linha 6 se a pasta **"AppTeseAndroid"** existe e, caso não exista, esta é criada na Linha 7. O mesmo é realizado aquando da criação do ficheiro de texto **text.txt**, nas Linhas 10 e 11. Após o ficheiro estar criado ou lido, o texto do utilizador é então gravado no ficheiro na Linha 16, usando a FileOutputStream.

```
1  private void saveFile() {
2    String text = textInput.getText().toString();
3    try {
4      String rootPath = Environment.getExternalStorageDirectory().
     toString();
5      File fileDirectory = new File(rootPath + "/AppTeseAndroid");
6      if (!fileDirectory.exists())
7        fileDirectory.mkdirs();
8
9      File file = new File(fileDirectory + "/text.txt");
10     if (!file.exists()) {
11       file.createNewFile();
12     }
13
14     FileOutputStream fileOutputStream = new FileOutputStream(file,
     true);
15     try {
16       fileOutputStream.write((text + System.getProperty("line.
     separator")).getBytes());
17     } finally {
18       fileOutputStream.close();
19     }
20
21   } catch(FileNotFoundException ex) {
22     ex.printStackTrace();
23   } catch(IOException ex) {
24     ex.printStackTrace();
25   }
26 }
```

Código 4.23: Guarda Texto em Ficheiro - Android

O Excerto 4.24, apresenta o método de ler texto a partir do ficheiro anteriormente gravado. Obtém-se o caminho da raíz do dispositivo na Linha 5 e obtém-se se o ficheiro anteriormente criado na Linha 6. É usada uma FileStream **fileInputStream**, um InputStreamReader **inputStreamReader** e um BufferedReader **bufferedReader** para ler as linhas enquanto existam, no ciclo representado na Linha 13. Enquanto existirem linhas no ficheiro estas são adicionadas, na Linha 15, a uma variável de texto, denominada **line**. No final o texto lido é apresentado ao utilizador adicionando-o ao campo de texto, na Linha 22.

```
1  private void readTextFromFile() {
2    String line = "";
3    try {
4
5      String rootPath = Environment.getExternalStorageDirectory().
     toString();
6      File file = new File(rootPath + "/AppTeseAndroid/text.txt");
7
8      FileInputStream fileInputStream = new FileInputStream (file);
9      InputStreamReader inputStreamReader = new InputStreamReader(
     fileInputStream);
10     BufferedReader bufferedReader = new BufferedReader(
     inputStreamReader);
11     StringBuilder stringBuilder = new StringBuilder();
12
13     while ( (line = bufferedReader.readLine()) != null )
14     {
15       stringBuilder.append(line + System.getProperty("line.
     separator"));
16     }
17     fileInputStream.close();
18     line = stringBuilder.toString();
19
20     bufferedReader.close();
21
22     textInput.setText(line);
23   } catch(IOException ex) {
24     ex.printStackTrace();
25   }
26 }
```

Código 4.24: Ler Texto de Ficheiro - Android

Conclusão

No caso específico do UC05 e UC06, pode-se verificar que as linhas de código são idênticas em ambas as aplicações e, como tal, não se pode tirar uma conclusão sobre qual das duas necessita de mais ou menos código.

4.4.5 Resumo

Para resumir, no UC01 e UC02 as linhas de código necessárias para programar as funcionalidades são menores na *framework* Xamarin do que em Android nativo. Por outro lado, no UC04, UC05 e UC06 as linhas de código são semelhantes em ambas as *frameworks*. Apesar de os testes serem efetuados numa escala pequena, pode-se afirmar que Xamarin utiliza menos linhas de código, mas seria interessante testar com mais funcionalidades e mais métodos de maneira a obter um resultado mais preciso.

4.5 Sumário

Neste capítulo foi efetuado a análise e o *design* da arquitetura e o desenho da interface. Além disto, foram apresentados excertos de código relevantes das diversas funcionalidades desenvolvidas, tendo comparado os excertos entre as *frameworks* para aferir se existia diferenças.

Neste capítulo respondeu-se à questão **Q1**, na Secção 4.1, e a parte da questão **Q5**, na Secção 4.4, na medida em que Xamarin utiliza menos linhas de código que Android, sendo isso uma vantagem para Xamarin e uma desvantagem para Android.

Capítulo 5

Comparação das Frameworks

Neste capítulo serão comparados as duas *frameworks* desenvolvimento de aplicações móveis anteriormente referidos, desenvolvimento em Xamarin e desenvolvimento nativo para Android. No começo irão ser apresentados os programas e SDK necessários para o desenvolvimento e serão comparados os custos desses programas.

Neste capítulo pretende-se responder às questões **Q2**, **Q3**, **Q4** e a parte da questão **Q5**, em todo o capítulo. Estas questões foram referidas na Secção 1.3 do Capítulo 1.

5.1 Setup do Ambiente de Desenvolvimento

Esta secção tem o intuito de apresentar os programas e SDK que os utilizadores necessitam de ter no seu computador para conseguir desenvolver aplicações móveis. Será divida em duas subsecções, uma para cada tecnologia.

5.1.1 Android

Para começar a desenvolver aplicações nativas para Android, é necessário começar por instalar o IDE oficial da Google, o Android Studio. Além do Android Studio é preciso o Android SDK. No entanto, ao efetuar o *download* do IDE o SDK já vem incluído, bem como o Android *Virtual Device* que é um emulador onde o utilizador pode testar as suas aplicações.

As aplicações irão ser programadas na linguagem de programação Java, e como tal é necessário instalar o Java Development Kit (JDK), de maneira a que o Android Studio consiga interpretar e compilar o código.

5.1.2 Xamarin

No caso do desenvolvimento de aplicações em Xamarin, o processo de início é semelhante ao visto para Android. Para começar é essencial a instalação do IDE da Microsoft, o Visual Studio 2017. O Visual Studio 2017 pode ser obtido em

três versões diferentes: **Community**, que é gratuito e é destinado a estudantes ou desenvolvedores individuais, **Professional**, que é destinado a equipas de desenvolvimento pequenas, e **Enterprise**, que é destinado a empresas com equipas de desenvolvimento de todos os tamanhos.

Aquando da instalação do IDE é necessário escolher a opção "Desenvolvimento móvel usando .NET"na escolha do *workload*. Nessa mesma instalação, é possível instalar o Android SDK e o JDK, que irão ser necessários para o desenvolvimento de aplicações para Android.

5.2 Comparação de Custos de Desenvolvimento

Nesta secção irão ser comparados os custos associados aos dois métodos de desenvolvimento com o objetivo de responder à questão **Q3**. Isto inclui os custos dos IDE e custos das Lojas.

5.2.1 Custo dos IDE

O Android Studio é completamente grátis, não sendo necessário pagar para ter acesso a todas as funcionalidades do IDE.

O Visual Studio 2017, por sua vez, possui três versões, apresentadas na Figura 5.1, em que apenas uma delas é gratuita. No desenvolvimento deste projeto irá ser usada a versão **Community**, pois existirá apenas um desenvolvedor. No entanto, em ambiente empresarial, seria necessário a versão **Professional** ou **Entreprise** dependendo do tamanho do grupo de desenvolvimento.

No caso da Aixtel Engineering a versão **Professional** seria a mais adequada, o que traria custos de 539$[1], ou cerca de 460€, por ano por desenvolvedor.

	Assinaturas Anuais		Assinaturas Mensais	
	Visual Studio Enterprise	Visual Studio Professional	Visual Studio Enterprise	Visual Studio Professional
Preço por usuário	2.999 USD/ano	539 USD/ano	250 USD/mês	45 USD/mês
	Compre agora	Compre agora	Compre agora	Compre agora

Figura 5.1: Preços do Visual Studio 2017

[1]Disponível em https://visualstudio.microsoft.com/vs/pricing/.

5.2.2 Custos das Lojas

A Google Play Store possui uma taxa de abertura de conta que ronda os 21€[2]. No entanto este valor só é necessário ser pago aquando da inscrição, não possuindo anualidade nem mensalidade para continuar a publicar.

A Apple Store possui um custo anual de cerca de 84€[3] para empresas, permitindo a publicação de aplicações durante o período de licença.

5.2.3 Conclusão

Posto isto, pode-se aferir que os custos em Android são de 21€ e que esse pagamento é efetuado apenas uma vez na criação da conta para publicar as aplicações na loja. Por outro lado, em Xamarin pode-se somar a esses 21€ o preço do IDE, que é de 460€ por ano e por desenvolvedor, e mais 84€ por ano, caso a empresa pretenda publicar as aplicações na loja da Apple.

Respondendo à questão **Q3**, o desenvolvimento para Android nativo é o mais barato e por uma larga margem. Esta resposta permite responder a parte da questão **Q5**, visto que isto é uma vantagem para Android e uma desvantagem para Xamarin.

5.3 Desempenho

Nesta secção ir-se-á testar o desempenho das duas aplicações desenvolvidas, em diversos aspetos, tais como: tempos de execução, gasto de bateria e tamanho que ocupam. No final, pretende-se responder à questão **Q2**.

5.3.1 Tempos de Execução

De maneira a obter um resultado assertivo nesta comparação irá ser feito um teste estatístico para fundamentar a mesma. O teste estatístico escolhido, foi o **T-Test**, explicado na Secção 3.2 do Capítulo 3. Para tal, irão ser usadas amostras do tempo de 50 cronometragens efetuadas na execução de algumas funcionalidades.

Para o auxílio na execução do teste irá ser usado o **RStudio**, que é um IDE destinado à linguagem de programação **R**, usada na criação de gráficos e cálculos estatísticos. Os dados obtidos foram introduzidos no Excel e gravados num ficheiro de formato **.csv** para ser usado no RStudio. Estes dados estão presentes nos Apêndices, Apêndice A, Apêndice B, Apêndice C e Apêndice D, divididos sob a forma de Tabelas.

[2]Disponível em https://support.google.com/googleplay/android-developer/answer/6112435.
[3]Disponível em https://developer.apple.com/support/compare-memberships/.

Utilizando o RStudio os primeiros passos serão comuns em todos os casos. Primeiro define-se o caminho para a pasta onde se encontra o ficheiro com os dados, através do comando **setwd**. Após isso é lido o ficheiro .csv com os dados para a variável dados, representado na Figura 5.2.

```
5  # Abrir ficheiro csv com os Dados
6  setwd("C:/Users/Carlos Martins/Desktop")
7  dados <- read.csv2(file="dados_tese.csv",header=T)
```

Figura 5.2: Definição do caminho e Leitura do Ficheiro de Dados

De acordo com o que foi definido no Capitulo 3, Secção 3.2 e Subsecção 3.2.3, as hipóteses a testar são **h0: Android = Xamarin = 50%** e que **h1: Android != Xamarin**. Caso a hipótese nula seja rejeitada, verificar-se-á qual das *frameworks* possuí o menor tempo de execução, mantendo a hipótese nula e testando as seguintes hipóteses: **h2: Android < Xamarin** e **h3: Android > Xamarin**. Tal como referido, associou-se que μ0 representa Android e μ1 representa Xamarin. Para executar o **T-Test** será usado um grau de confiança de 95%, ou seja, um alfa de 0.05.

Arranque da Aplicação

Os dados importados são filtrados, com o comando *subset*, por **Framework** e por **Function**, e serão criadas duas variáveis, cada uma com os tempos respectivos à função **Open** e *framework* respectiva, representado na Figura 5.3. Isto irá obter os dados da Tabela A.1, para Android, e Tabela A.2, para Xamarin, presentes no Apêndice A.

```
9  # Dados Android Open
10 android_open <- subset (dados, Framework == "Android" & Function == "Open")
11 # Dados Xamarin Open
12 xamarin_open <- subset (dados, Framework == "Xamarin" & Function == "Open")
```

Figura 5.3: Filtragem dos Dados da Arranque da Aplicação

É imprimido um sumário de cada conjunto de dados, com o comando **summary**. Cada sumário contém dados relevantes, tais como, o valor mínimo e máximo, média e mediana.

Tabela 5.1: Sumário dos Tempos de Execução de Arranque da Aplicação

Framework	Min	Mediana	Máx	Média
Android	781	980.5	1212	957
Xamarin	1001	1355	2472	1383

Analisando a Tabela 5.1 pode-se verificar que na *framework* Android os valores são mais baixos, em que a média e a mediana diferem cerca de 400ms entre as *frameworks*.

Os dados são então testados com o **T-Test**, através do comando **t.test**, com um grau de confiança de 95%, como representado na Figura 5.4.

```
17 # Teste para a igualdade de médias
18 t.test(android_open$i..Time..ms., xamarin_open$i..Time..ms., conf.level=0.95)
```

Figura 5.4: Abertura da Aplicação - 1º T-Test

O **p-value** obtido é 2.2e-16. Como tal, a hipótese nula h0 é rejeitada. Os dados são diferentes por isso é necessário aferir qual dos dois é maior. Para tal irão ser usadas as alternativas "less"e "greater"para perceber se os valores em Android são menores ou maiores que os valores em Xamarin. Primeiro será testado se os tempos em Android são menores que Xamarin usando a alternativa "less"com um grau de confiança de 95%, como representado na Figura 5.5.

```
23 # Teste para médias de P1 menores que P2
24 t.test(android_open$i..Time..ms., xamarin_open$i..Time..ms., alternative="less", conf.level=0.95)
```

Figura 5.5: Abertura da Aplicação - 2º T-Test

Os resultados obtidos indicam um **p-value** de 2.2e-16. Como o **p-value** é menor que o alfa, rejeita-se a hipótese nula h0. Sendo assim, para um grau de confiança de 95% a média de tempos que a aplicação demora a abrir é menor em Android que em Xamarin. Posto isto não é necessário testar a alternativa "greater".

Mapa

Os dados importados são filtrados, tal como no primeiro teste, mas com o filtro **Function** sendo **Map**. Isto irá obter os dados da Tabela B.1, para Android, e Tabela B.2, para Xamarin, presentes no Apêndice B.

Tal como no primeiro teste são imprimidos os sumários de cada conjudo de dados.

Tabela 5.2: Sumário dos Tempos de Execução de Abertura do Mapa

Framework	Min	Mediana	Máx	Média
Android	2567	2641	3794	2691
Xamarin	2316	2346	2534	2357

Analisando a Tabela 5.2, pode-se verificar que os valores na *framework* Android, são mais elevados, com diferenças de cerca de 300ms na média e mediana.

Os dados são então testados com o **T-Test**, mantendo o grau de confiança de 95%, como representado na Figura 5.6.

```
17 # Teste para a igualdade de médias
18 t.test(android_map$i..Time..ms., xamarin_map$i..Time..ms., conf.level=0.95)
```

Figura 5.6: Abertura do Mapa - 1º T-Test

Os resultados obtidos indicam um **p-value** de 2.2e-16. O **p-value** é menor que o alfa. Rejeita-se a hipótese nula h0 e, como tal, é necessário verificar se as médias são maiores ou menos entre as *frameworks*. Primeiro será testada a alternativa "less", como representado na Figura 5.7.

```
23 # Teste para médias de P1 menores que P2
24 t.test(android_map$i..Time..ms., xamarin_map$i..Time..ms., alternative="less", conf.level=0.95)
```

Figura 5.7: Abertura do Mapa - 2º T-Test

O **p-value** obtido foi 1. Como o **p-value** é maior que alfa, não se rejeita a hipótese nula h0, ou seja, os valores em Android não são menores que os valores em Xamarin, mas sabendo que as médias são diferentes pode-se assumir que os valores em Android são maiores que em Xamarin. Para confirmar isto, será efetuado o **T-Test** com a alternativa "greater", como representado na Figura 5.8.

```
23 # Teste para médias de P1 menores que P2
24 t.test(android_map$i..Time..ms., xamarin_map$i..Time..ms., alternative="less", conf.level=0.95)
```

Figura 5.8: Abertura do Mapa - 3º T-Test

O **p-value** obtido foi 2.2e-16. Como o **p-value** é menor que alfa, rejeita-se a hipótese nula h0 e pode-se assumir que os valores em Android são maiores que os valores em Xamarin. Pode-se concluir que a aplicação em Android demora mais que a aplicação em Xamarin a abrir o mapa e mover o mesmo para Sidney.

Câmara

Os dados importados são filtrados, tal como nos outros testes, mas com o filtro **Function** sendo **Cam**. Isto irá obter os dados da Tabela C.1, para Android, e Tabela C.2, para Xamarin, presentes no Apêndice C.

Tal como nos outros teste são imprimidos os sumários de cada conjunto de dados.

Tabela 5.3: Sumário dos Tempos de Execução de Tirar Fotografia

Framework	Min	Mediana	Máx	Média
Android	446	494	579	498
Xamarin	484	537	676	538.4

Analisado a Tabela 5.3 pode-se verificar que os valores em Xamarin são mais elevados, tendo uma diferença de cerca de 40ms na média e mediana.

Os dados são então testados com o **T-Test**, mantendo o grau de confiança de 95%, como representado na Figura 5.9.

```
17  # Teste para a igualdade de médias
18  t.test(android_cam$ï..Time..ms., xamarin_cam$ï..Time..ms., conf.level=0.95)
```

Figura 5.9: Tirar Fotografia - 1º T-Test

Os resultados obtidos indicam um **p-value** de 9.072e-08. Rejeita-se a hipótese nula h0, indicando que os valores das duas *frameworks* são diferentes. Será então realizado um novo **T-Test** usando a alternativa "less", como representado na Figura 5.10.

```
23  # Teste para médias de P1 menores que P2
24  t.test(android_cam$ï..Time..ms., xamarin_cam$ï..Time..ms., alternative="less", conf.level=0.95)
```

Figura 5.10: Tirar Fotografia - 2º T-Test

O **p-value** obtido foi de 4.536e-08. Como o mesmo é menor que alfa, rejeita-se a hipótese nula h0, permitindo concluir que os valores em Android são menores que os valores em Xamarin. Com isto, pode-se concluir que a aplicação produzida na *framework* Android é mais rápida no processo de tirar e guardar fotografias.

YouTube

O procedimento inicial é o mesmo, filtrando os dados com a **Function YouTube** e são imprimidos os sumários dos dados. Isto irá obter os dados da Tabela D.1, para Android, e Tabela D.2, para Xamarin, presentes no Apêndice D.

Tabela 5.4: Sumário dos Tempos de Execução de Abertura do Player do YouTube

Framework	Min	Mediana	Máx	Média
Android	292	407	721	411
Xamarin	309	373	640	384

Analisando a Tabela 5.4 pode-se verificar que os valores na *framework* Android são um pouco mais elevados. Para confirmar isto será realizado o **T-Test** com um grau de confiança de 95%, como representado na Figura 5.11.

```
17  # Teste para a igualdade de médias
18  t.test(android_youtube$ï..Time..ms., xamarin_youtube$ï..Time..ms., conf.level=0.95)
```

Figura 5.11: Player do YouTube - 1º T-Test

Os resultados obtidos neste teste indicam um **p-value** de 0.05452. Como o **p-value** é maior que o alfa não se rejeita a hipótese nula h0. Como tal, para um grau de confiança de 95%, os valores de ambas as *frameworks* são iguais.

Conclusão

Analisando os resultados obtidos nos 4 testes efetuados, pode-se concluir que a aplicação desenvolvida em Android nativo é a mais rápida na execução de 2 tarefas, igual na execução de 1 tarefa e mais lenta na execução da outra tarefa. Posto isto, pode-se concluir implicitamente que o tempo de execução de tarefas na aplicação desenvolvida em Android são menores e, como tal, tem um melhor desempenho nesse aspeto. De maneira a que estes resultados fossem mais precisos seria importante testar as aplicações numa maior escala e em mais dispositivos.

5.3.2 Gasto de Bateria

De maneira a testar o gasto de bateria de cada aplicação será delineado um teste a ser efetuado. Após efetuar o teste e com a aplicação ainda aberta, será verificado nas definições do dispositivo os valores da aplicação.

Este teste será efetuado no dispositivo em três ambientes distintos, com o dispositivo a 100% de bateria, 50% bateria e 20% bateria. Os resultados serão apresentados em **mAh**, milliampere por hora, no que refere à bateria. De notar que o teste terá de ser efetuado com apenas a aplicação de teste a correr e nenhuma em *background* de maneira a não influenciar o teste.

O teste a ser efetuado será:

1. Abrir a aplicação.
2. Abrir o mapa, o mapa irá mover automaticamente para Sidney, como referido anteriormente. A aplicação deve ficar 1 minuto a receber *updates* da localização e dados.
3. Sair do mapa.
4. Tirar 5 fotografias.
5. Partilhar uma fotografia no Messenger.
6. Abrir a funcionalidade dos sensores, e receber *updates* dos mesmo durante 1 minuto.
7. Guardar o texto "Teste"e ler o mesmo.
8. Carregar o vídeo do YouTube e visualizar os primeiros 30 segundos do vídeo.

Após efetuar este teste nos 3 ambientes em cada aplicação serão então analisados os dados.

Tabela 5.5: Sumário dos Gastos de Bateria

Percentagem	**Android**	**Xamarin**
100%	6.28 mAh	9.44 mAh
50%	8.15 mAh	6.56 mAh
20%	8.29 mAh	8.52 mAh
Média	7.57 mAh	8.17 mAh

De acordo com a Tabela 5.5, e analisando estes testes pode-se afirmar que o gasto de bateria em Android é menor, principalmente quando a bateria está no máximo. No entanto, a escala de testes é pequena e seria importante realizar o teste em dispositivos diferentes e em maior escala de maneira a obter uma decisão mais assertiva.

5.3.3 Tamanho da Aplicação

Nesta secção irá ser comparado o tamanho das aplicações desenvolvidas por *framework*. Ambas as aplicações foram instaladas no dispositivo em modo "release"em vez de em modo "debug", caso contrário poderia influenciar o tamanho das mesmas. Além disso não será contado o espaço ocupado por dados ou em *cache*.

No caso da *framework* Android o tamanho da aplicação **3,94 MB**, enquanto que a aplicação produzida na *framework* Xamarin, tem o tamanho de **80,18 MB**, tendo assim uma diferença de **76,24 MB**.

A diferença entre os tamanhos da aplicação é consideravelmente alta. Como tal, pode-se considerar isto como uma desvantagem no desenvolvimento de aplicações em Xamarin, dado que para certas aplicações o armazenamento disponível no dispositivo é imprescindível, responde assim a parte da questão **Q5**.

5.3.4 Conclusão

Concluindo, nos 3 aspetos de desempenho analisados, a aplicação desenvolvida em Android superou, por uma curta margem, a aplicação desenvolvida em Xamarin.

Respondendo à questão **Q2**, e de acordo com os testes analisados, o desenvolvimento para Android nativo produz aplicações com melhor desempenho que o desenvolvimento em Xamarin. Esta resposta permite responder a parte da questão **Q5**, visto que isto é uma vantagem para Android e uma desvantagem para Xamarin.

5.4 Tempo de Desenvolvimento

No que diz respeito ao tempo de desenvolvimento, e à questão **Q4**, a conclusão que se tira não traz grande relevância, pois as aplicações foram desenvolvidas por apenas 1 pessoa. Para chegar a uma conclusão neste métrica seria necessário uma equipa de desenvolvimento maior, com experiência no desenvolvimento de ambas as tecnologias envolvidas.

Posto isto, a questão **Q4** fica sem resposta.

5.5 Sumário

Neste capítulo foi efetuada uma comparação das *frameworks* mais detalhada. Foi comparado a instalação necessária para iniciar o desenvolvimento, os custos das ferramentas associadas e, por fim, o desempenho. Não foi possível comparar o tempo de desenvolvimento dado não existir uma equipa de desenvolvimento destinada a esta dissertação.

Neste capítulo respondeu-se à questão **Q2**, na Secção 5.3, à questão **Q3**, na Secção 5.2 e não se consegui responder à questão **Q4**, na Secção 5.4. Ao longo destas secções, respondeu-se a partes da questão **Q5** no que diz respeito a vantagens e desvantagens das *frameworks*.

Capítulo 6

Conclusões

Neste capítulo irá ser apresentado um breve resumo do contexto e problema e irão ser apresentadas as conclusões desta dissertação. As conclusões serão apresentadas de acordo com as questões definidas na Secção 1.3 do Capítulo 1.

6.1 Contexto

O SO Android foi lançado num dispositivo pela primeira vez em 2008. Atualmente a versão mais recente é a versão Pie(9.0), lançada em Agosto de 2018. Para o desenvolvimento de aplicações móveis é necessário a utilização do SDK, do IDE Android Studio, da linguagem de programação Java e do XML para as interfaces do utilizador.

O Xamarin pertence à Microsoft desde 2016. Atualmente existem as *frameworks* Xamarin.Android e Xamarin.iOS, que permitem o desenvolvimento de aplicações na linguagem de programação C# para as plataformas Android e iOS respetivamente, e o Xamarin.Froms que permite o desenvolvimento em simultâneo para ambas. O Xamarin permite que as aplicações tenham acesso a funcionalidades nativas e sejam disponibilizadas nas lojas de cada plataforma.

6.2 Problema

A Aixtel Engineering está inserida na área das telecomunicações e está envolvida em projetos para a instalação de fibra ótica em diversos países. Esta desenvolve plataformas e aplicações móveis que facilitem o trabalho dos seus clientes no levantamento de dados para a instalação da fibra ótica. Atualmente, as aplicações móveis são desenvolvidas para Android nativo.

A empresa está a ponderar a possibilidade de alargar as suas aplicações a outras plataformas, neste caso iOS, com o intuito de abranger um maior número de clientes. Dentro dos diferentes tipos de desenvolvimento possíveis, a empresa decidiu comparar o atual desenvolvimento com uma *framework* multi-plataforma, Xamarin.

Para decidir qual das *frameworks* deveria utilizar, decidiu comparar ambas em termos de desempenho das aplicações, custos e tempo de desenvolvimento.

6.3 Questão Q1

A questão **Q1**, que foi respondida na Secção 4.1 do Capítulo 4, tinha como objetivo apresentar as funcionalidades que deveriam ser exploradas de forma a obter uma boa avaliação das *frameworks*.

Para tal, recorreu-se a uma análise das permissões mais utilizadas nas aplicações da loja oficial da Google. Após obter as permissões mais utilizadas, foram escolhidas 7 funcionalidades que usassem essas permissões, de maneira a poder analisar funcionalidades que a comunidade use regularmente e assim obter uma melhor avaliação.

6.4 Questão Q2

A questão **Q2**, que foi respondida na Secção 5.3 do Capítulo 5, tinha como objetivo decidir sobre qual das aplicações desenvolvidas iria ter melhor desempenho.

Para avaliar isto, analisaram-se 3 aspetos, onde se incluem o tempo de execução de tarefas, gasto de bateria e tamanho que as aplicações ocupam. Após a analisem, concluiu-se que em todos os aspetos a aplicação desenvolvida em Android nativo é superior, por uma curta margem, à aplicação desenvolvida em Xamarin. No entanto, seria importante testar as aplicações a uma grande escala, de maneira a obter um resultado mais assertivo.

6.5 Questão Q3

A questão **Q3**, que foi respondida na Secção 5.2 do Capítulo 5, tinha como objetivo perceber sobre qual das *frameworks* envolve menores custos.

Analisaram-se os custos dos IDE e das lojas, e concluiu-se que os custos para o desenvolvimento de Android nativo são menores, por uma grande escala. Para desenvolver para Android nativo, o IDE é gratuito e o custo da loja da Google é de 21€ por conta, enquanto que com Xamarin é necessário comprar o IDE, que custa 460€ por ano e por desenvolvedor, e posteriormente a loja da Google e a loja da Apple, que custa 84€ por ano.

6.6 Questão Q4

A questão **Q4** tinha como objetivo decidir sobre qual das *frameworks* possui menor tempo de desenvolvimento. No entanto, não foi possível responder a esta questão, como mencionado na Secção 5.4 do Capítulo 5. Isto deve-se ao facto de esta dissertação estar a ser desenvolvida por apenas uma pessoa e que, para ter dados concretos, seria necessário possuir uma equipa de desenvolvimento com experiência em ambas as tecnologias.

6.7 Questão Q5

A questão **Q5** foi respondida ao longo do Capítulo 4 e do Capítulo 5. Esta questão tinha como objetivo apresentar as vantagens e desvantagens de cada *framework*. As vantagens de uma *framework* irão se refletir nas desvantagens da outra. Estas vantagens e desvantagens irão ter em conta as questões anteriores e os testes efetuados durante esta dissertação. De seguida serão apresentadas as vantagens de desenvolver aplicações em Android nativo.

Androd nativo - Vantagens:

- Menores custos.
- Melhor desempenho.
- Aplicações com menor tamanho.

Xamarin - Vantagens:

- Acesso a diversas plataformas.
- Interface comum para as plataformas.
- Menos linhas de código necessárias.

6.8 Questão Final Q6

A questão final **Q6** é a questão essencial desta dissertação. Ela tem como objetivo decidir sobre qual das *frameworks* deve a empresa optar.

Tal como foi referido no Capítulo 2, os três critérios fundamentais para a empresa eram o custo, tempo e desempenho.

A empresa pretendia manter ou, no melhor dos casos, melhorar o desempenho das suas aplicações, manter ou diminuir os custos associados ao desenvolvimento e migrar as aplicações para outras plataformas de maneira a obter mais clientes. Com os resultados obtidos ao longo desta dissertação, a empresa não iria conseguir obter

os resultados pretendidos, pois em termos de custos e desempenho os resultados indicam que o desenvolvimento em Android nativo é melhor, sendo que a maior vantagem para a empresa caso mudasse para o desenvolvimento em Xamarin, seria o acesso a várias plataformas e a mais clientes.

No entanto, é preciso ter em conta que os testes efetuados foram feitos numa escala pequena num só dispositivo e que, no critério do desempenho, o desenvolvimento para Android nativo é melhor por uma pequena margem. Em relação aos custos, os custos iriam aumentar drasticamente. Supondo que a empresa tinha uma equipa de 5 elementos a desenvolver em Xamarin e a publicar aplicações durante 1 ano, os custos associados seriam de 2384€, 2300€ para os IDE, 84€ para a loja da Apple, enquanto que atualmente a empresa não tem custos previstos, pois já possui a conta da loja da Google.

Concluindo, atendendo à analise e testes desta dissertação seria melhor para a empresa se manter a desenvolver aplicações para Android nativamente. Contudo, caso a empresa tenha possibilidades seria relevante fazer um estudo financeiro para averiguar se os custos-benefícios do desenvolvimento em Xamarin, e disponibilização das suas aplicações em iOS, seriam do interesse da empresa. Além disto, efetuar testes de desempenho a uma grande escala também iria permitir obter resultados mais assertivos nesse aspeto.

6.9 Limitações e Trabalho Futuro

A principal limitação existente no desenvolvimento desta dissertação foi o facto de não existir uma equipa de desenvolvimento, mas sim apenas um desenvolvedor. Apesar das aplicações possuírem funcionalidades relativamente simples de implementar, o desenvolvimento de ambas atrasou-se principalmente pelo facto de serem duas, bem como, o uso de ferramentas novas. Com isto, os testes ficaram um pouco mais reduzidos que o desejado, pois um maior conjunto de dados para testes iria permitir tirar conclusões mais assertivas. Além disto, o facto de não possuir uma equipa de desenvolvimento não permitiu responder à questão **Q4**, como referido na Secção 5.4 e Secção 6.6.

No futuro seria interessante conseguir responder à questão que ficou por responder, bem como, aumentar a escala de testes efetuados e testar mais funcionalidades que podem ter ficado por testar. Num contexto fora da empresa, seria relevante fazer a comparação com o desenvolvimento para iOS nativo, com o intuito de perceber se o desenvolvimento para iOS em Xamarin se encontra ao mesmo nível que o desenvolvimento nativo, visto que as aplicações em iOS são frequentemente mencionadas pela pela seu ótimo desempenho.

6.10 Apreciação Final

Os objetivos pretendidos foram atingidos, à exceção da questão **Q4**. Apesar de não existir resposta a essa questão, a questão final **Q6** foi respondida, decidindo que a empresa Aixtel Engineering se deveria manter a desenvolver para Android nativo. Contudo, foi deixado em aberto um possível estudo mais aprofundado com o intuito de obter um resultado mais assertivo.

No desenvolvimento da aplicação na *framework* Android, verificou-se que a quantidade de fontes *online* com respostas a dúvidas e tutoriais para a implementação de certas funções é enorme contendo resposta para todas as dúvidas que surgiram. Já no desenvolvimento na *framework* Xamarin, o mesmo não acontece. Muito devido ao facto de possuir uma comunidade mais pequena e de ser uma *framework* mais recente. É necessário ter isto em conta, pois torna o desenvolvimento para Android nativo mais fácil para desenvolvedores que não estejam muito dentro da *framework*, ao contrário de Xamarin.

De notar que o desenvolvimento desta dissertação teve como base as boas práticas de Engenharia. Os módulos lecionados no contexto da unidade curricular TMDEI nomeadamente, a pesquisa e escrita técnico-científica, análise de valor de negócio e experimentação e avaliação, contribuíram para o bom desenvolvimento da dissertação e para o enriquecimento profissional.

Bibliografia

Allee, Verna (2002). «A value network approach for modeling and measuring intangibles». Em: *Transparent Enterprise, Madrid. Available at http://www. vernaallee. com*.

Android (2018). *Android*. url: https://www.android.com/ (acedido em 07/02/2018).

Bray, Tim et al. (1997). «Extensible markup language (XML).» Em: *World Wide Web Journal* 2.4, pp. 27–66.

BuildFire (2016). *How To Choose Between Native, Hybrid or Web App For Your Business*. url: https://buildfire.com/choose-native-hybrid-web-mobile-app/ (acedido em 09/10/2018).

Center, Pew Research (2015a). *An Analysis of Android App Permissions*. url: http://www.pewinternet.org/2015/11/10/an-analysis-of-android-app-permissions/ (acedido em 14/07/2018).

– (2015b). *Apps Permissions in the Google Play Store*. url: http://www.pewinternet.org/2015/11/10/apps-permissions-in-the-google-play-store/ (acedido em 14/07/2018).

ComScore (2017). *Mobile's Hierarchy of Needs*. url: https://www.comscore.com/Insights/Presentations-and-Whitepapers/2017/Mobiles-Hierarchy-of-Needs (acedido em 04/02/2018).

Cordova, Apache (2018). *Architectural overview of Cordova platform*. url: https://cordova.apache.org/docs/en/latest/guide/overview/index.html (acedido em 09/02/2018).

Data, Evans (2016). *Mobile Developer Population Reaches 12M Worldwide, Expected to Top 14M by 2020*. url: https://evansdata.com/press/viewRelease.php?pressID=244 (acedido em 06/02/2018).

Developers, Android (2018a). *Dashboards*. url: https://developer.android.com/about/dashboards/index.html (acedido em 07/02/2018).

– (2018b). *Permissões do Sistema*. url: https://developer.android.com/guide/topics/security/permissions (acedido em 09/10/2018).

– (2018c). *Platform Architecture*. url: https://developer.android.com/guide/platform/index.html (acedido em 07/02/2018).

Gosling, James et al. (2014). *The Java Language Specification, Java SE 8 Edition (Java Series)*.

Idesis, Stanley (2017). *Free Cross-Platform Mobile App Development Tools Compared*. url: https://www.outsystems.com/blog/free-cross-platform-mobile-app-development-tools-compared-2017.html (acedido em 09/02/2018).

Ionic (2018). *Build amazing apps in one codebase, for any platform, with the web.* url: https://ionicframework.com/framework (acedido em 09/02/2018).

Koen, Peter A et al. (2002). *Fuzzy front end: effective methods, tools, and techniques*. Wiley, New York, NY.

Liberty, Jesse (2005). *Programming C#: Building. NET Applications with C*. "O'Reilly Media, Inc."

Nicola, Susana, Eduarda Pinto Ferreira e JJ Pinto Ferreira (2012). «A novel framework for modeling value for the customer, an essay on negotiation». Em: *International Journal of Information Technology & Decision Making* 11.03, pp. 661–703.

React (2018). *A framework for building native apps using React*. url: `https://facebook.github.io/react-native/` (acedido em 09/02/2018).

Saaty, Thomas L (1990). «How to make a decision: the analytic hierarchy process». Em: *European journal of operational research* 48.1, pp. 9–26.

StatCounter (2018). *Mobile Operating System Market Share*. url: `http://gs.statcounter.com/os-market-share/mobile/worldwide` (acedido em 05/02/2018).

Statista (2017). *Number of Google Play Store apps 2017.* url: `https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/` (acedido em 07/02/2018).

Ulaga, Wolfgang e Andreas Eggert (2006). «Value-based differentiation in business relationships: Gaining and sustaining key supplier status». Em: *Journal of marketing* 70.1, pp. 119–136.

Versluis, Gerald (2017). «A Brief History of Xamarin». Em: *Xamarin. Forms Essentials*. Springer, pp. 3–18.

Woodall, Tony (2003). «Conceptualising'value for the customer': An attributional, structural and dispositional analysis». Em: *Academy of marketing science review* 2003, p. 1.

Xamarin (2018a). *Mobile Application Development to Build App in C#*. url: `https://www.xamarin.com/platform` (acedido em 08/02/2018).

– (2018b). *Sharing Code Options*. url: `https://developer.xamarin.com/guides/cross-platform/application_fundamentals/code-sharing/` (acedido em 08/02/2018).

Apêndice A

Dados do Arranque da Aplicação

Tabela A.1: Dados do Arranque da Aplicação - Android

Time(ms)	Framework	Function
825	Android	Open
795	Android	Open
956	Android	Open
995	Android	Open
990	Android	Open
1013	Android	Open
786	Android	Open
1037	Android	Open
1000	Android	Open
984	Android	Open
781	Android	Open
993	Android	Open
803	Android	Open
881	Android	Open
985	Android	Open
984	Android	Open
894	Android	Open
993	Android	Open
969	Android	Open
1040	Android	Open
793	Android	Open
1004	Android	Open
977	Android	Open
999	Android	Open
1030	Android	Open
⋮	⋮	⋮

Time(ms)	Framework	Function
⋮	⋮	⋮
984	Android	Open
893	Android	Open
956	Android	Open
789	Android	Open
944	Android	Open
795	Android	Open
961	Android	Open
1013	Android	Open
975	Android	Open
945	Android	Open
818	Android	Open
1024	Android	Open
935	Android	Open
1050	Android	Open
962	Android	Open
965	Android	Open
1100	Android	Open
1062	Android	Open
1101	Android	Open
1046	Android	Open
787	Android	Open
1010	Android	Open
959	Android	Open
1212	Android	Open
1057	Android	Open

Tabela A.2: Dados do Arranque da Aplicação - Xamarin

Time(ms)	**Framework**	**Function**
1316	Xamarin	Open
1330	Xamarin	Open
2133	Xamarin	Open
1399	Xamarin	Open
1370	Xamarin	Open
1336	Xamarin	Open
1093	Xamarin	Open
1265	Xamarin	Open
1405	Xamarin	Open
1355	Xamarin	Open
1344	Xamarin	Open
1369	Xamarin	Open
1262	Xamarin	Open
1409	Xamarin	Open
2472	Xamarin	Open
1208	Xamarin	Open
1429	Xamarin	Open
1304	Xamarin	Open
1265	Xamarin	Open
1336	Xamarin	Open
1560	Xamarin	Open
1203	Xamarin	Open
1328	Xamarin	Open
1379	Xamarin	Open
1429	Xamarin	Open
1399	Xamarin	Open
1001	Xamarin	Open
1369	Xamarin	Open
1362	Xamarin	Open
1333	Xamarin	Open
1361	Xamarin	Open
1360	Xamarin	Open
1380	Xamarin	Open
1364	Xamarin	Open
1290	Xamarin	Open
⋮	⋮	⋮

Time(ms)	**Framework**	**Function**
⋮	⋮	⋮
1272	Xamarin	Open
1320	Xamarin	Open
1366	Xamarin	Open
1368	Xamarin	Open
1320	Xamarin	Open
1400	Xamarin	Open
1369	Xamarin	Open
1355	Xamarin	Open
1300	Xamarin	Open
1380	Xamarin	Open
1346	Xamarin	Open
1349	Xamarin	Open
1800	Xamarin	Open
1345	Xamarin	Open
1339	Xamarin	Open

Apêndice B

Dados UC01

Tabela B.1: Dados UC01 - Android

Time(ms)	Framework	Function
3794	Android	Map
2798	Android	Map
2784	Android	Map
2751	Android	Map
2752	Android	Map
2752	Android	Map
2716	Android	Map
2735	Android	Map
2668	Android	Map
2702	Android	Map
2701	Android	Map
2668	Android	Map
2736	Android	Map
3342	Android	Map
2685	Android	Map
2648	Android	Map
2634	Android	Map
2651	Android	Map
2618	Android	Map
2638	Android	Map
2614	Android	Map
2644	Android	Map
2651	Android	Map
2617	Android	Map
2655	Android	Map
⋮	⋮	⋮

Time(ms)	Framework	Function
⋮	⋮	⋮
2734	Android	Map
2685	Android	Map
2668	Android	Map
2668	Android	Map
2635	Android	Map
2617	Android	Map
2652	Android	Map
2637	Android	Map
2617	Android	Map
2599	Android	Map
2617	Android	Map
2615	Android	Map
2631	Android	Map
2600	Android	Map
2586	Android	Map
2600	Android	Map
2567	Android	Map
2618	Android	Map
2602	Android	Map
2599	Android	Map
2618	Android	Map
2616	Android	Map
2628	Android	Map
2585	Android	Map
2599	Android	Map

Tabela B.2: Dados UC01 - Xamarin

Time(ms)	Framework	Function
2534	Xamarin	Map
2429	Xamarin	Map
2423	Xamarin	Map
2418	Xamarin	Map
2381	Xamarin	Map
2395	Xamarin	Map
2413	Xamarin	Map
2359	Xamarin	Map
2355	Xamarin	Map
2346	Xamarin	Map
2350	Xamarin	Map
2358	Xamarin	Map
2358	Xamarin	Map
2381	Xamarin	Map
2345	Xamarin	Map
2325	Xamarin	Map
2372	Xamarin	Map
2346	Xamarin	Map
2340	Xamarin	Map
2346	Xamarin	Map
2317	Xamarin	Map
2353	Xamarin	Map
2333	Xamarin	Map
2429	Xamarin	Map
2349	Xamarin	Map
2354	Xamarin	Map
2347	Xamarin	Map
2342	Xamarin	Map
2338	Xamarin	Map
2343	Xamarin	Map
2325	Xamarin	Map
2334	Xamarin	Map
2377	Xamarin	Map
2346	Xamarin	Map
2322	Xamarin	Map
⋮	⋮	⋮

Time(ms)	Framework	Function
⋮	⋮	⋮
2331	Xamarin	Map
2375	Xamarin	Map
2336	Xamarin	Map
2357	Xamarin	Map
2322	Xamarin	Map
2330	Xamarin	Map
2316	Xamarin	Map
2348	Xamarin	Map
2336	Xamarin	Map
2370	Xamarin	Map
2320	Xamarin	Map
2343	Xamarin	Map
2323	Xamarin	Map
2329	Xamarin	Map
2344	Xamarin	Map

Apêndice C

Dados UC02

Tabela C.1: Dados UC02 - Android

Time(ms)	Framework	Function
559	Android	Cam
502	Android	Cam
494	Android	Cam
506	Android	Cam
487	Android	Cam
486	Android	Cam
452	Android	Cam
519	Android	Cam
499	Android	Cam
521	Android	Cam
474	Android	Cam
498	Android	Cam
499	Android	Cam
510	Android	Cam
486	Android	Cam
492	Android	Cam
485	Android	Cam
468	Android	Cam
491	Android	Cam
534	Android	Cam
519	Android	Cam
579	Android	Cam
488	Android	Cam
571	Android	Cam
548	Android	Cam
⋮	⋮	⋮

Time(ms)	Framework	Function
⋮	⋮	⋮
464	Android	Cam
498	Android	Cam
496	Android	Cam
514	Android	Cam
564	Android	Cam
517	Android	Cam
490	Android	Cam
454	Android	Cam
559	Android	Cam
480	Android	Cam
462	Android	Cam
531	Android	Cam
466	Android	Cam
485	Android	Cam
449	Android	Cam
446	Android	Cam
498	Android	Cam
489	Android	Cam
495	Android	Cam
471	Android	Cam
490	Android	Cam
461	Android	Cam
457	Android	Cam
502	Android	Cam
494	Android	Cam

Tabela C.2: Dados UC02 - Xamarin

Time(ms)	Framework	Function
676	Xamarin	Cam
561	Xamarin	Cam
552	Xamarin	Cam
544	Xamarin	Cam
497	Xamarin	Cam
536	Xamarin	Cam
527	Xamarin	Cam
485	Xamarin	Cam
676	Xamarin	Cam
527	Xamarin	Cam
491	Xamarin	Cam
515	Xamarin	Cam
511	Xamarin	Cam
500	Xamarin	Cam
523	Xamarin	Cam
516	Xamarin	Cam
508	Xamarin	Cam
506	Xamarin	Cam
484	Xamarin	Cam
550	Xamarin	Cam
515	Xamarin	Cam
512	Xamarin	Cam
580	Xamarin	Cam
519	Xamarin	Cam
489	Xamarin	Cam
487	Xamarin	Cam
551	Xamarin	Cam
556	Xamarin	Cam
578	Xamarin	Cam
553	Xamarin	Cam
556	Xamarin	Cam
565	Xamarin	Cam
551	Xamarin	Cam
526	Xamarin	Cam
561	Xamarin	Cam
⋮	⋮	⋮

Time(ms)	Framework	Function
⋮	⋮	⋮
538	Xamarin	Cam
546	Xamarin	Cam
548	Xamarin	Cam
535	Xamarin	Cam
531	Xamarin	Cam
564	Xamarin	Cam
559	Xamarin	Cam
542	Xamarin	Cam
549	Xamarin	Cam
552	Xamarin	Cam
517	Xamarin	Cam
535	Xamarin	Cam
528	Xamarin	Cam
548	Xamarin	Cam
542	Xamarin	Cam

Apêndice D

Dados UC07

Tabela D.1: Dados UC07 - Android

Time(ms)	Framework	Function
631	Android	YouTube
528	Android	YouTube
416	Android	YouTube
415	Android	YouTube
406	Android	YouTube
481	Android	YouTube
349	Android	YouTube
371	Android	YouTube
359	Android	YouTube
361	Android	YouTube
353	Android	YouTube
440	Android	YouTube
350	Android	YouTube
364	Android	YouTube
349	Android	YouTube
368	Android	YouTube
362	Android	YouTube
431	Android	YouTube
495	Android	YouTube
371	Android	YouTube
393	Android	YouTube
336	Android	YouTube
348	Android	YouTube
372	Android	YouTube
378	Android	YouTube
⋮	⋮	⋮

Time(ms)	Framework	Function
⋮	⋮	⋮
379	Android	YouTube
429	Android	YouTube
354	Android	YouTube
378	Android	YouTube
367	Android	YouTube
434	Android	YouTube
420	Android	YouTube
434	Android	YouTube
403	Android	YouTube
394	Android	YouTube
408	Android	YouTube
426	Android	YouTube
414	Android	YouTube
477	Android	YouTube
415	Android	YouTube
410	Android	YouTube
422	Android	YouTube
422	Android	YouTube
430	Android	YouTube
482	Android	YouTube
438	Android	YouTube
474	Android	YouTube
721	Android	YouTube
303	Android	YouTube
292	Android	YouTube

Tabela D.2: Dados UC07 - Xamarin

Time(ms)	Framework	Function
640	Xamarin	YouTube
503	Xamarin	YouTube
424	Xamarin	YouTube
393	Xamarin	YouTube
357	Xamarin	YouTube
345	Xamarin	YouTube
384	Xamarin	YouTube
315	Xamarin	YouTube
320	Xamarin	YouTube
333	Xamarin	YouTube
323	Xamarin	YouTube
319	Xamarin	YouTube
417	Xamarin	YouTube
344	Xamarin	YouTube
330	Xamarin	YouTube
350	Xamarin	YouTube
329	Xamarin	YouTube
331	Xamarin	YouTube
373	Xamarin	YouTube
335	Xamarin	YouTube
328	Xamarin	YouTube
338	Xamarin	YouTube
330	Xamarin	YouTube
336	Xamarin	YouTube
407	Xamarin	YouTube
356	Xamarin	YouTube
369	Xamarin	YouTube
361	Xamarin	YouTube
369	Xamarin	YouTube
378	Xamarin	YouTube
408	Xamarin	YouTube
400	Xamarin	YouTube
376	Xamarin	YouTube
374	Xamarin	YouTube
380	Xamarin	YouTube
⋮	⋮	⋮

Time(ms)	Framework	Function
⋮	⋮	⋮
381	Xamarin	YouTube
431	Xamarin	YouTube
499	Xamarin	YouTube
422	Xamarin	YouTube
418	Xamarin	YouTube
393	Xamarin	YouTube
411	Xamarin	YouTube
447	Xamarin	YouTube
433	Xamarin	YouTube
436	Xamarin	YouTube
424	Xamarin	YouTube
567	Xamarin	YouTube
332	Xamarin	YouTube
327	Xamarin	YouTube
309	Xamarin	YouTube