Criador automático de aplicações nativas Android e iOS

Filipe Ribeiro Matos

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática, Área de Especialização em Arquiteturas, Sistemas e Redes

Orientador: Doutor Paulo Baltarejo Sousa

Júri:

Presidente:

Doutor João Paulo Jorge Pereira

Vogais:

Doutor Luis Miguel Rosario da Silva Pinho

Doutor Paulo Manuel Baltarejo de Sousa

Porto, Outubro 2014

Resumo

Nos últimos anos tem-se verificado um acentuado aumento na utilização de dispositivos móveis a nível internacional, pelo que as aplicações desenvolvidas para este tipo específico de dispositivos, conhecidas por *apps*, têm vindo a ganhar uma enorme popularidade. São cada vez mais as empresas que procuram estar presentes nos mais diversos sistemas operativos móveis, com o objetivo de suportar e desenvolver o seu negócio, alargando o seu leque de possíveis consumidores.

Neste sentido surgiram diversas ferramentas com a função de facilitar o desenvolvimento de aplicações móveis, denominadas *frameworks* multi-plataforma. Estas frameworks conduziram ao aparecimento de plataformas *web*, que permitem criar aplicações multi-plataforma sem ser obrigatório ter conhecimentos em programação.

Assim, e a partir da análise de vários criadores *online* de aplicações móveis identificados e das diferentes estratégias de desenvolvimento de aplicações móveis existentes, foi proposta a implementação de uma plataforma *web* capaz de criar aplicações nativas Android e iOS, dois dos sistemas operativos mais utilizados na atualidade.

Após desenvolvida a plataforma *web*, designada MobileAppBuilder, foi avaliada a sua qualidade e as aplicações criadas pela mesma, através do preenchimento de um questionário por parte de 10 individuos com formação em Engenharia Informática, resultando numa classificação geral de "excelente". De modo a analisar o desempenho das aplicações produzidas pela plataforma desenvolvida, foram realizados testes comparativos entre uma aplicação da MobileAppBuilder e duas homólogas de dois dos criadores *online* estudados, nomeadamente Andromo e Como. Os resultados destes testes revelaram que a MobileAppBuilder gera aplicações menos pesadas, mais rápidas e mais eficientes em alguns aspetos, nomeadamente no arranque.

Palavras-chave: aplicações móveis nativas, android, ios, plataforma web, *build tools*, processo de compilação

Abstract

In the past years there has been a growth in the use of mobile devices at an international level, which caused applications developed to this type of devices, known as *apps*, to experience an increase in popularity. There are more and more companies looking for being present in the variety of existing mobile operating systems, in order to support and develop their business, widening their range of possible consumers.

Therefore, many tools emerged with the purpose of making the development of mobile applications easier, called cross platform frameworks. These led to the appearing of web platforms that allow the creation of cross platform applications without the need of knowledge in computer programming.

Thus, from the analysis of the identified online mobile application creators and of the existing different mobile application development strategies, it was proposed the implementation of a web platform capable of creating native mobile applications for Android and iOS, the more common operating systems nowadays.

After the development of the web platform, named MobileAppBuilder, its quality was surveyed trough a questionnaire and the apps generated through it were evaluated, by comparative tests with two of the online creators studied, namely Como and Andromo.

In order to evaluate the quality of the platform, 10 individuals with a degree in Computer Science were surveyed, with the platform getting an overall rating of “excellent”. The tests done to the applications generated through the platform revealed that the MobileAppBuilder produces less heavy, faster and more efficient applications that the other platforms tests, for example in the application start-up.

Keywords: mobile native applications, android, ios, web platform, build tools, compilation process

Agradecimentos

Concluída a dissertação, gostaria de deixar algumas palavras de agradecimento às pessoas que tornaram a sua realização possível.

Em primeiro lugar, ao meu orientador, o Professor Paulo Baltarejo Sousa pela partilha de saberes e disponibilidade que sempre demonstrou.

À minha família e amigos pelo incentivo e carinho e em especial à Inês por ter estado lá nos bons e maus momentos . A todos eles, o meu mais sincero agradecimento.

À Coca-Cola, o mais doce dos elixires ao alcance da humanidade.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Código Fonte

Lista de Comandos

Lista de Gráficos

Acrónimos

AAPT Android Asset Packaging Tool

API Application Programming Interface

APK Android Application Package

BSD Berkeley Software Distribution

CMS Content Management System

CSS Cascading Style Sheets

DLL Dynamic-link Library

DNS Domain Name System

DVM Dalvik Virtual Machine

FTP File Transfer Protocol

GPS Global Positioning System

GSM Global System for Mobile Communications

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IDE Integrated Development Environment

IPA iOS Application Archive

JSON JavaScript Object Notation

LAMP Linux, Apache, MySQL, PHP

MMS Multimedia Messaging Service

OS Operating System

PDO PHP Data Objects

RSS Really Simple Syndication

SDK Software Development Kit

SMS Short Message Service

SQL Structured Query Language

SSL Secure Sockets Layer

TLS Transport Layer Security

UI User Interface

URI Uniform Resource Identifier

URL Uniform Resource Locator

W3C World Wide Web Consortium

XML Extensible Markup Language

1 Introdução

O primeiro protótipo de telemóvel surgiu em 1973 pela mão de Martin Cooper, membro executivo da multi-nacional Motorola (Shiels, 2003). Em 1984 surgiu no mercado o dispositivo DynaTAC que, com um peso de quase 1kg e 20cm de comprimento, possuía uma capacidade de trinta minutos de chamada após dez longas horas de carga (Motorola, 2014).

Passaram somente quarenta anos desde a criação do primeiro protótipo de telemóvel, mas a rápida evolução destes dispositivos leva a acreditar que o tempo em que um telemóvel apenas servia para efetuar chamadas e enviar mensagens de texto se encontra num passado longínquo. Em determinadas zonas do globo, em particular no mundo ocidentalizado e países ditos desenvolvidos, é altamente improvável estar num espaço público sem ver alguém a tocar no ecrã do seu *smartphone*, uma espécie de "computador de mão" com um sistema operativo integrado que oferece mais interfaces de comunicação que um computador tradicional, mas que apresenta algumas restrições impostas pela utilização de uma bateria, nomeadamente na capacidade de processamento.

A procura associada a estes dispositivos nunca foi tão elevada quanto é atualmente. Durante o ano de 2013 foram vendidas 968 milhões de unidades em todo o mundo, um aumento de 42.3% comparativamente ao ano de 2012, no qual foram vendidas 680 milhões de unidades. Pela primeira vez na história, o número de *smartphones* vendidos ultrapassou o número de telemóveis tradicionais, representando 54% do total das vendas de dispositivos móveis (1.8 biliões de unidades) em 2013 (Gartner-1, 2014).

Consequentemente, também se encontra em elevado crescimento o mercado das aplicações para estes dispositivos móveis, que hoje dominam o mercado. A procura de ferramentas por parte das empresas que lhes permitam estar presentes no mercado através destas plataformas nunca foi tão grande. Em 2012 foram efetuados 64 milhões de *downloads* de aplicações através da Play Store [1] e Apple Store[2], valor que aumentou para 102 milhões em 2013, gerando uma receita de aproximadamente 26 biliões de dólares, mais 8 biliões de dólares relativamente a 2012 (Gartner-2, 2013).

Apesar do brutal crescimento observado, as caraterísticas destes dispositivos conferem-lhes algumas limitações que precisam de ser devidamente consideradas no desenvolvimento de aplicações, pois resultam em possíveis implicações técnicas, ergonómicas e económicas para o utilizador.

Neste sentido, projetar a interface gráfica de uma aplicação móvel pode ser um verdadeiro desafio, considerando que os tamanhos de ecrã variam de dispositivo para dispositivo, assim como as resoluções. Se os dados de uma aplicação forem muito complexos e o utilizador tiver de interagir com elevado número de objetos no ecrã (como por exemplo campos de entrada, listas, ou caixas de seleção), é aconselhável dividir esses dados por diferentes ecrãs ou *tabs*.

Uma interface gráfica demasiado complexa e algoritmos demasiado intensivos, isto é, que necessitem de grande capacidade de computação, podem facilmente tornar uma aplicação pouco "amiga do utilizador".

O desempenho das aplicações executadas em dispositivos móveis, denominadas aplicações móveis, vai depender de vários factores, principalmente para as que interagem com servidores remotos. Para a performance de um computador tradicional, o volume de dados que consome a partir de um servidor remoto não é importante, dependendo apenas da ligação à Internet utilizada. Já para dispositivos móveis, esta questão é completamente diferente, pois a ligação à Internet vai ser estabelecida através de redes sem fios, sendo a capacidade de comunicação inferior e assim caracterizada por uma maior latência.

---

[1] Plataforma de distribuição digital suportada pela Google que permite efetuar o *download* de aplicações para o Android.

[2] Plataforma de distribuição digital suportada pela Apple que permite efetuar o *download* de aplicações para o iOS.

Nos últimos anos, a tecnologia associada aos dispositivos móveis tem vindo a evoluir de forma significativa, principalmente ao nível do armazenamento e processamento de informação. No entanto, a performance alcançada é ainda baixa, especialmente quanto à duração das baterias disponíveis, considerado atualmente o maior obstáculo à melhoria do desempenho destes dispositivos.

A segurança é outro fator a ter em conta, uma vez que nos dispositivos móveis são armazenados dados sensíveis do utilizador, os quais não devem ser acedidos por estranhos, nomeadamente em caso de roubo. Alguns dispositivos incluem um leitor de impressões digitais, no sentido de garantir ao utilizador uma barreira de segurança extra. Tipicamente as aplicações que contêm dados sensíveis possuem um sistema de *login*, tendo o utilizador de apresentar um conjunto de credenciais para poder utilizar a mesma.

A elevada procura demonstrada por parte das empresas no sentido de estarem presentes nas diferentes plataformas móveis conduziu ao aparecimento de várias ferramentas que suportam o desenvolvimento de aplicações multi-plataforma, isto é, instrumentos que permitam criar aplicações para mais do que um sistema operativo, por exemplo, tanto para Android como para iOS. É esperado que este mercado, de origem recente, passe de uma receita de 3.2 biliões de dólares em 2013 para 3.6 biliões de dólares em 2014 (ResearchAndMarkets, 2014). Quando comparados estes valores com os observados entre 2012 e 2013, que corresponderam a uma taxa de crescimento de 97%, é possível verificar que o mercado para estas ferramentas está a atingir a sua maturidade e já ultrapassou o seu pico de popularidade. Existem vários fatores que ajudam a justificar esta tendência, tais como o aumento da popularidade das aplicações com código-fonte nativo e as limitações das ferramentas multi-plataforma, que colocam cada vez maiores desafios aos programadores (ResearchAndMarkets, 2014).

A necessidade de aceder a estas ferramentas levou ao aparecimento de plataformas *online*, que permitem a criação de aplicações móveis através de uma ligação à internet. As mesmas foram objeto de estudo na presente dissertação, sendo que apenas uma plataforma se revelou capaz de gerar aplicações nativas para Android. O trabalho a realizar sobre o qual assenta esta dissertação de mestrado pretende apresentar uma plataforma *web* capaz de criar aplicações nativas para Android e iOS.

1.1 Contribuição

O trabalho relatado na presente dissertação terá as seguintes contribuições:

- Estado da arte referente às diferentes estratégias de desenvolvimento de aplicações móveis;
- Estado da arte relativo aos vários criadores de aplicações móveis existentes atualmente;
- Proposta de um sistema capaz de criar aplicações para dispositivos que tenham como sistema operativo o Android ou iOS;
- Especificação dos detalhes de implementação da solução proposta;
- Uma plataforma capaz de gerar aplicações que obtenham o maior rendimento possível do dispositivo em que são executadas, pelo facto de serem nativas;
- Um sistema expansível e capaz de tirar o máximo partido de todas as características de *hardware* dos dispositivos móveis;
- Uma plataforma que permite criar aplicações nativas para Android e iOS em tempo recorde quando comparado ao tempo dispendido por uma equipa de programadores para criar uma aplicação semelhante.

1.2 Estrutura do Documento

A estrutura do presente documento encontra-se dividida em cinco capítulos:

- Capítulo 1 - Introdução - Descreve de forma breve as ideias-chave do projeto desenvolvido.
- Capítulo 2 – Estado da Arte – Engloba o resumo dos sistemas operativos Android e iOS, o estudo das diferentes estratégias de desenvolvimento de aplicações móveis e ainda a análise a alguns dos criadores de aplicações móveis existentes.
- Capítulo 3 – MobileAppBuilder – Está dividido em duas secções: a análise, onde se encontram identificados os requisitos funcionais e não funcionais, a arquitetura do sistema proposto e o modelo de dados que irá suportar a plataforma desenvolvida; e a implementação, onde são descritos os detalhes de implementação da solução proposta.

- Capítulo 4 – Análise à MobileAppBuilder – Apresenta os resultados e a discussão dos testes de performance realizados a aplicações produzidas através da plataforma desenvolvida e de um inquérito realizado por um grupo de indivíduos formados em Engenharia Informática.
- Capítulo 5 – Conclusão – Apresenta as conclusões do estudo efetuado no âmbito desta dissertação e faz referência a trabalho futuro a ser realizado com base nos resultados obtidos.

2 Estado da Arte

Em qualquer projeto é necessário na fase de levantamento de requisitos estabelecer um primeiro contacto com os conceitos relacionados com o tema a abordar, antes de iniciar o estudo do problema a resolver, sendo muito importante procurar e analisar outros projetos e soluções existentes. Através dos mesmos é possível apurar quais as melhores práticas a utilizar para a implementação e quais as funcionalidades fundamentais e secundárias quedevem fazer parte da solução a desenvolver. Este processo de investigação, que consiste em pesquisar tanto quanto possível toda a informação e tudo o que já existe feito ou desenvolvido na área em estudo, denomina-se "Observar o Estado da Arte".

Neste capítulo são abordados os temas estudados no âmbito desta dissertação, tais como os sistemas operativos móveis utilizados (Android e iOS), as estratégias de desenvolvimento de aplicações móveis que existem atualmente, bem como a análise efetuada a algumas ferramentas que permitem a criação de aplicações móveis.

2.1 Sistemas Operativos Móveis

Longe vão os dias em que um telemóvel apenas servia para efetuar chamadas e enviar mensagens de texto esporadicamente. Atualmente, os dispositivos móveis (*smartphones*, *PDAs* e *tablets*) são autênticos computadores de mão, com um sistema operativo integrado, que fornece uma plataforma para programadores onde é permitido instalar e executar aplicações específicas.

O primeiro smartphone foi apresentado pela International Business Machines (IBM) em Novembro de 1992, que com o ROM-DOS como sistema operativo e sem qualquer botão físico permitia a interação com o utilizador através do seu ecrã tátil (Grush, 2012).

No final dos anos 80 surgiu o sistema operativo EPOC (Kilpatrick, 2014), desenvolvido pela empresa Psion Software, que em 1998 se viria a tornar na Symbian Ltd. (Operating-System, 2004) juntamente com a Ericsson, Nokia e Motorola. Nesse mesmo ano lançaram para o mercado o Symbian OS, que em 2007 conquistou 62,3% da quota do mercado dos sistemas operativos para dispositivos móveis (Gartner-3, 2009). Contudo, em Junho de 2008 a Nokia anunciou a aquisição da Symbian Ltd. e a criação de uma nova organização sem fins lucrativos, a Symbian Foundation, com o objetivo de garantir a compatibilidade do sistema operativo Symbian OS com diferentes dispositivos (Virki, 2008). Em Fevereiro de 2010, a plataforma Symbian foi oficialmente anunciada como sendo *open source* (WatBlog, 2010). Um ano mais tarde, a Nokia anunciou uma mudança na sua estratégia, abandonando o Symbian como sistema operativo principal e adotando o Microsoft Windows Phone (Litchfield, 2011). Recentemente, a multi-nacional apresentou o Asha, um sistema operativo que tem como objetivo principal tornar os telemóveis de baixo custo o mais funcional possível, para que utilizadores de mercados emergentes e economias menos fortes tenham um melhor acesso a conteúdos *smart*, como redes sociais e ligação à Internet (TekSapo, 2013).

Para competir com o Symbian OS, a Microsoft lança o Windows CE (Compact Edition) em finais de 1996 (HPFactor, 2001), um sistema operativo que se viria a tornar na base de todos os sistemas operativos móveis desenvolvidos pela Microsoft. Devido à sua interface complexa e à consequente dificuldade de interação com os mesmos, os sistemas operativos móveis da Microsoft nunca tiveram uma adesão acentuada, motivo que levou a empresa a introduzir no Windows Phone 7 (lançado a Outubro de 2009) (Fried, 2009) o estilo de *design* Metro (Cnet, 2012), conhecido por dispor no ecrã o menu principal e as várias aplicações em blocos quadrados e retangulares de diferentes tamanhos. Em Outubro de 2012 foi lançado o Windows Phone 8, a versão mais recente do sistema operativo da Microsoft. Segundo um estudo da IDC (International Data Corporation), no terceiro trimestre de 2013 o número de dispositivos móveis vendidos com Windows Phone aumentou 156% relativamente ao terceiro trimestre do ano anterior (IDC, 2013).

Ainda em 2007, a Apple lançou o seu primeiro *smartphone*, designado iPhone, que tinha integrado o sistema operativo iOS 1.0. Apesar de ser um dispositivo caro, o mesmo não

suportava 3G, *multi-tasking* nem *Multimedia Messaging Service* (MMS) (TheVerge, 2013), entre outras funcionalidades. Continha um ecrã tátil na totalidade da sua face frontal, ícones grandes e uma interface significativamente interativa e amiga do utilizador.

Também em 2007, foi anunciada a criação do Open Handset Alliance (OHA), um consórcio atualmente composto por 84 empresas (entre elas Google, Sony, Dell e Intel) com o objetivo de criar alguma padronização para dispositivos móveis em termos de sistema operativo. Inicialmente desenvolvido pela Android Inc. (comprada pela Google em 2005 (Elgin, 2005)), o Android OS chegou ao mercado em 2008, sendo um sistema operativo livre e *open-source*, baseado no *kernel* do Linux e desenhado para dispositivos móveis (*smartphones*, *tablets* e *netbooks*).

O estudo "IDC Worldwide Mobile Phone Tracker", publicado a 12 de Novembro de 2013 pela IDC, revela os dados presentes na Figura 1.

Top Four Operating Systems, Shipments, and Market Share, Q3 2013 (Units in Millions)

Operating System	3Q13 Shipment Volumes	3Q13 Market Share	3Q12 Shipment Volumes	3Q12 Market Share	Year-Over-Year Change
Android	211.6	81.0%	139.9	74.9%	51.3%
iOS	33.8	12.9%	26.9	14.4%	25.6%
Windows Phone	9.5	3.6%	3.7	2.0%	156.0%
BlackBerry	4.5	1.7%	7.7	4.1%	-41.6%
Others	1.7	0.6%	8.4	4.5%	-80.1%
Total	**261.1**	**100.0%**	**186.7**	**100.0%**	**39.9%**

Figura 1 – Quota de mercado dos sistemas operativos móveis no terceiro trimestre de 2013

Após uma análise dos dados presentes na Figura 1 é possível verificar que o Android OS domina 81% do mercado atual, o que torna imperativo suportar sistemas Android quando se desenvolve para dispositivos móveis. O sistema operativo com a segunda maior quota de mercado é o iOS, que se encontra a perder espaço para concorrentes diretos como Windows Phone e Android.

A plataforma desenvolvida no âmbito da presente dissertação irá suportar a criação de aplicações Android e iOS, estando descritos nos sub-capítulos seguintes conceitos e caraterísticas associadas a estes sistemas operativos.

2.1.1 Android

Baseado no kernel 2.6 do Linux, o Android é um sistema operativo livre e *open-source* que permite o desenvolvimento de aplicações na linguagem de programação Java. Construído de modo a que os programadores possam desenvolver aplicações que tirem o máximo partido das funcionalidades que os dispositivos têm para oferecer (tais como efetuar chamadas, enviar mensagens e utilizar a câmara), proporciona aos utilizadores uma experiência mais rica e coesa (OpenHandsetAlliance, 2014) comparativamente a sistemas operativos móveis lançados anteriormente.

O Android não diferencia aplicações do telefone e aplicações desenvolvidas por terceiros. O sistema operativo utiliza uma máquina virtual personalizada com o objetivo de otimizar a memória e recursos de hardware dos dispositivos móveis.

Para o desenvolvimento de aplicações Android é necessário utilizar o *Software Development Kit* (SDK), composto pela *Application Programming Interface* (API) de classes Java e pelo emulador, isto é, um dispositivo móvel virtual que permite executar e testar aplicações desenvolvidas. As aplicações Android desenvolvidas através do SDK resultam num ficheiro do tipo *Android Application Package* (APK). Além do SDK é necessário um ambiente de desenvolvimento, nomeadamente: Eclipse (Eclipse, 2014), Netbeans (Netbeans, 2014), IntelliJ DEA (JetBains-IDEA, 2014) ou outro. Atualmente, a Google disponibiliza o *Android Development Tools* (ADT) *Bundle*, composto pelo ambiente Eclipse e o SDK. Ainda em fase de amadurecimento encontra-se o Android Studio, um ambiente de desenvolvimento baseado no IntelliJ IDEA e que se destaca pelo suporte à *build tool* (conceito abordado numa fase mais adiantada do presente documento) Gradle (Gradle, 2014).

2.1.1.1 Arquitetura

A Arquitetura do Android (Figura 2) encontra-se dividida em cinco partes (Pereira & Silva, 2009): Kernel Linux, bibliotecas, ambiente de execução, *framework* e aplicações:

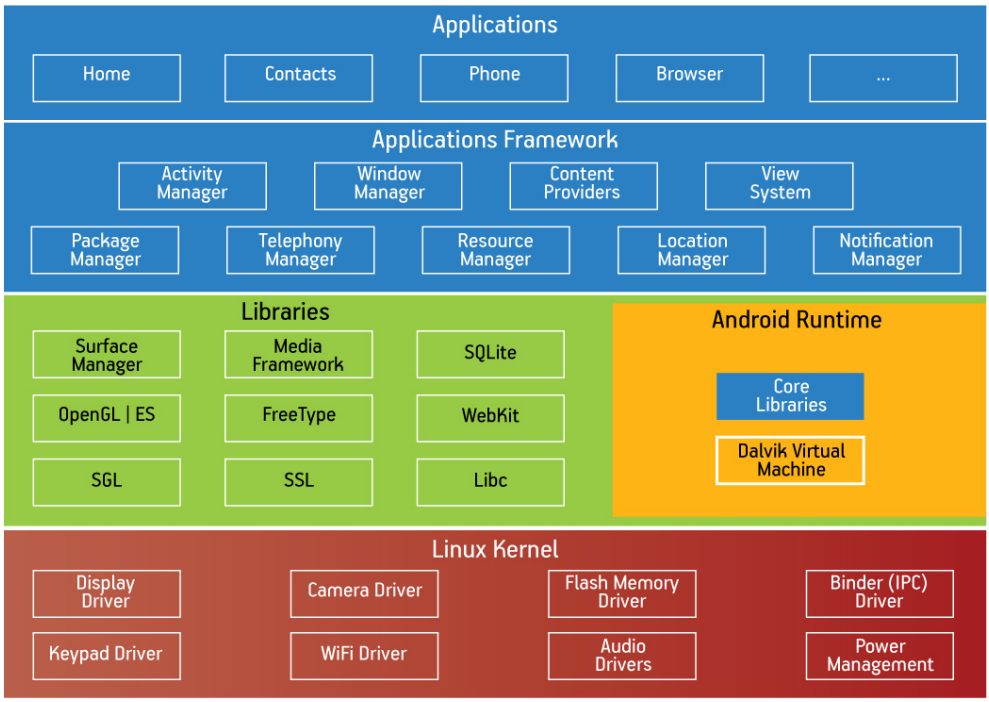


Figura 2 – Arquitetura Android

Kernel Linux

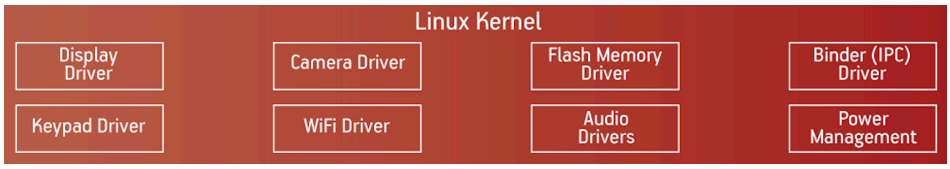


Figura 3 - Arquitetura Android: Linux Kernel

O Android utiliza internamente o kernel do Linux 2.6 para tarefas fundamentais de segurança, gestão de memória, gestão de processos e de gestão de protocolos de rede, entre outros aspetos. No fundo, esta é uma camada de abstração entre o *hardware* e os outros componentes da plataforma (Figura 3). De modo a otimizar o kernel do Linux para dispositivos móveis foi necessário realizar várias alterações ao mesmo, como por exemplo a introdução de novos *device drivers* e de um mecanismo para terminar processos quando existe pouca memória disponível (*lowmem killer*) (Tess, 2013).

Bibliotecas

Acima do kernel do Linux estão as bibliotecas nativas do Android (Figura 4), escritas em C/C++ e específicas para cada arquitetura de *hardware*. Neste conjunto de bibliotecas encontram-se a biblioteca padrão do C (Libc) e as bibliotecas que suportam os formatos de áudio e vídeo

mais populares, a visualização de gráficos 2D e 3D, as funções de aceleração de *hardware* e a renderização 3D, entre outras. Algumas das bibliotecas nativas mais importantes são:

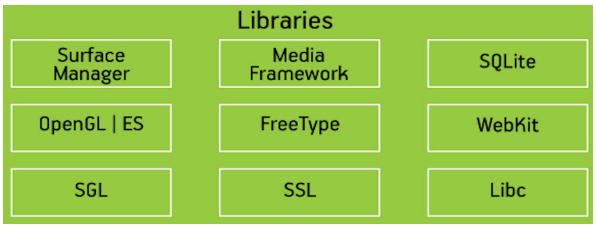


Figura 4 – Arquitetura Android: Bibliotecas

- Surface Manager - A biblioteca Surface Manager é utilizada na composição do gestor de janelas com um *buffer off-screen*, isto é, em vez de desenhar diretamente no *buffer* do ecrã os *bitmaps* são desenhados num *buffer* fora do ecrã, que é posteriormente combinado com outros *buffers*, desenhando o ecrã final que o utilizador vê. Esta implementação permite a criação de transparências e diferentes efeitos para transições entre janelas (AndroidAppMarket, 2012).
- *Structured Query Language* (SQL) Engine - O SQL Engine permite o armazenamento persistente de dados de aplicações. O Android inclui o SQLite, um motor de bases de dados relacionais leve.
- Browser Engine - O Browser Engine é utilizado para mostrar o conteúdo de páginas HTML (*HyperText Markup Language*). O Android inclui o WebKit (WebKit, 2014), um motor de navegação *open-source* utilizado nos *browsers* Google Chrome e Safari, entre outros.

Todas as bibliotecas referidas estão disponíveis na camada superior da Arquitectura Android (Applicationn Framework).

Ambiente de Execução

Ao lado das bibliotecas nativas do Android e acima do kernel do Linux encontra-se o ambiente de execução (Figura 5), composto por uma máquina virtual Dalvik (DVM – *Dalvik Virtual Machine*) e as principais bibliotecas do Java. Apesar de as aplicações para Android serem desenvolvidas recorrendo à linguagem de

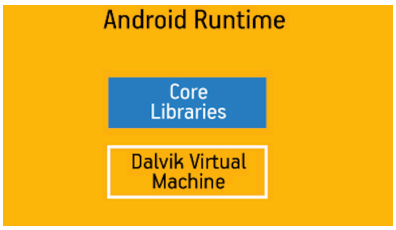


Figura 5 – Arquitetura Android: Ambiente de Execução

programação Java, neste sistema operativo não existe nenhuma máquina virtual Java (JVM), mas sim uma máquina virtual Dalvik, otimizada para dispositivos com baixa capacidade de processamento e memória.

Em Android cada aplicação é executada dentro do seu próprio processo, isto é, dentro da sua própria instância da máquina virtual Dalvik. Esta, ao contrário da JVM, não executa ficheiros .class mas sim apenas ficheiros .dex, que são gerados através dos ficheiros .class em tempo de compilação, sendo este um formato de *bytecode* desenhado para ocupar menos espaço.

Application Framework

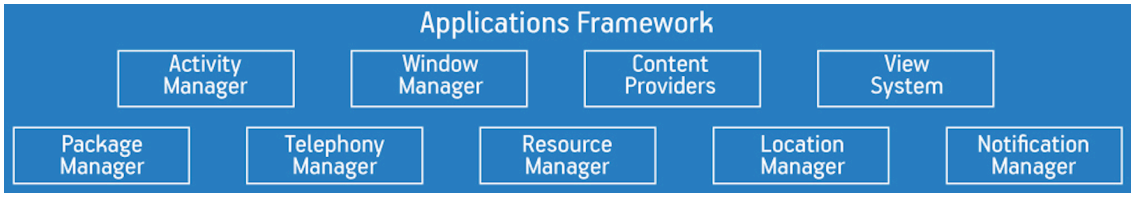


Figura 6 - Arquitetura Android: Application Framework

Acima das bibliotecas nativas e do Android *Runtime* encontra-se o *Application Framework* (Figura 6). Esta camada fornece acesso a blocos de programação utilizados nas aplicações desenvolvidas para este sistema operativo, o que simplifica a reutilização de componentes. Entre as APIs disponíveis, destacam-se:

- Activity Manager - Controla o ciclo de vida das aplicações e mantém uma "*back stack*" de forma a permitir a navegação do utilizador, será abordada em maior detalhe numa fase posterior do presente documento.
- Content Providers - Consiste numa interface padrão utilizada para a partilha de dados entre aplicações.
- Location Manager – Oferece acesso a serviços de localização do sistema que permite às aplicações obter atualizações periódicas da localização geográfica do dispositivo, entre outras funcionalidades.
- Notification Manager - Permite que as aplicações mostrem alertas personalizados na barra de estados, como por exemplo novas mensagens de texto.

- Resource Manager - Oferece acesso a recursos que não sejam código-fonte, tais como *strings*, imagens, *layouts*, entre outos.

Aplicações

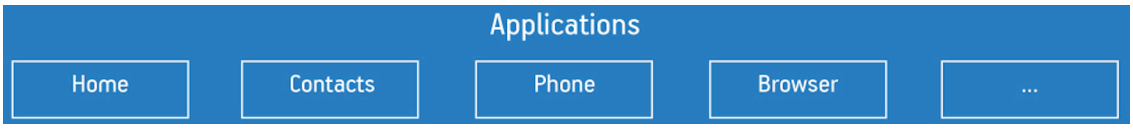


Figura 7 - Arquitetura Android: Applications

A camada aplicações (Figura 7) é a mais alta da arquitetura da plataforma Android e é composta, como o nome indica, por todas as aplicações contidas no dispositivo, entre elas: *home*, contactos, telefone, navegador *web*, cliente de e-mail, mapas e calendário.

2.1.1.2 Activity Manager

Nos sistemas operativos convencionais, nomeadamente Linux e Windows, é permitido ao utilizador ter várias aplicações abertas, cada uma com a sua janela. O utilizador escolhe que janela (ou janelas) quer ver naquele momento, navegando de forma fácil entre as mesmas.

Em Android, o conceito de janelas é diferente do convencional. Existe apenas uma aplicação a correr no ecrã do dispositivo, que ocupa todo o ecrã, à exceção da barra de estados. Quando o utilizador liga o *smartphone*, a primeira aplicação que vê é a *home application* (Figura 8).

Cada ecrã visível para o utilizador é internamente representado pela classe Activity. Cada aplicação é representada por uma ou mais Activities e um processo de Linux. Uma Activity tem o seu próprio ciclo de vida que não se encontra ligado ao ciclo de vida do processo da aplicação.

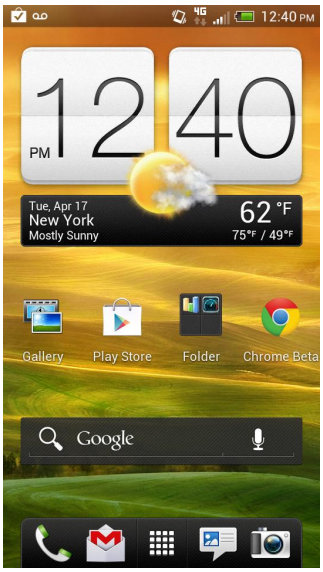


Figura 8 – Android Home Application

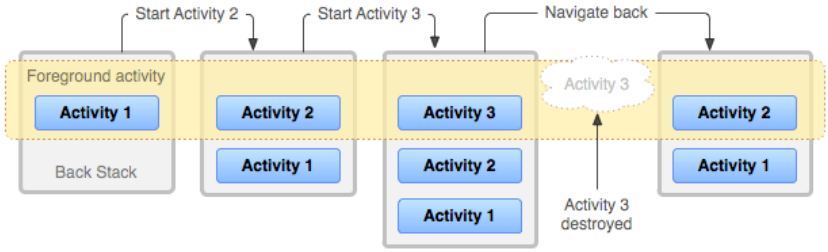


Figura 9 – Funcionamento da Activity *stack*

Quando um utilizador inicia uma aplicação, o Android coloca-a no ecrã. A partir dessa aplicação o utilizador pode abrir um novo ecrã ou até mesmo abrir outra aplicação. Toda esta informação de programas e ecrãs é guardada na *application stack* através do Activity Manager. Em qualquer altura o utilizador pode pressionar o botão de retroceder e voltar ao ecrã anterior, o que se encontra registado na *stack* (Figura 9).

2.1.1.3 Elementos da Aplicação

Todos os componentes de uma aplicação devem encontrar-se declarados dentro de um determinado ficheiro XML (*eXtensible Markup Language*), o AndroidManifest.xml. De seguida encontra-se uma breve descrição dos vários componentes que uma aplicação desenvolvida em Android pode instanciar e executar.

Activities

Uma Activity é fundamentalmente um ecrã de uma aplicação com a qual o utilizador poderá interagir. É o componente mais utilizado de todos e é constituído por um conjunto de *Views*, que respondem a eventos previamente programados. Durante o seu tempo de vida, uma Activity pode estar num de sete estados diferentes.

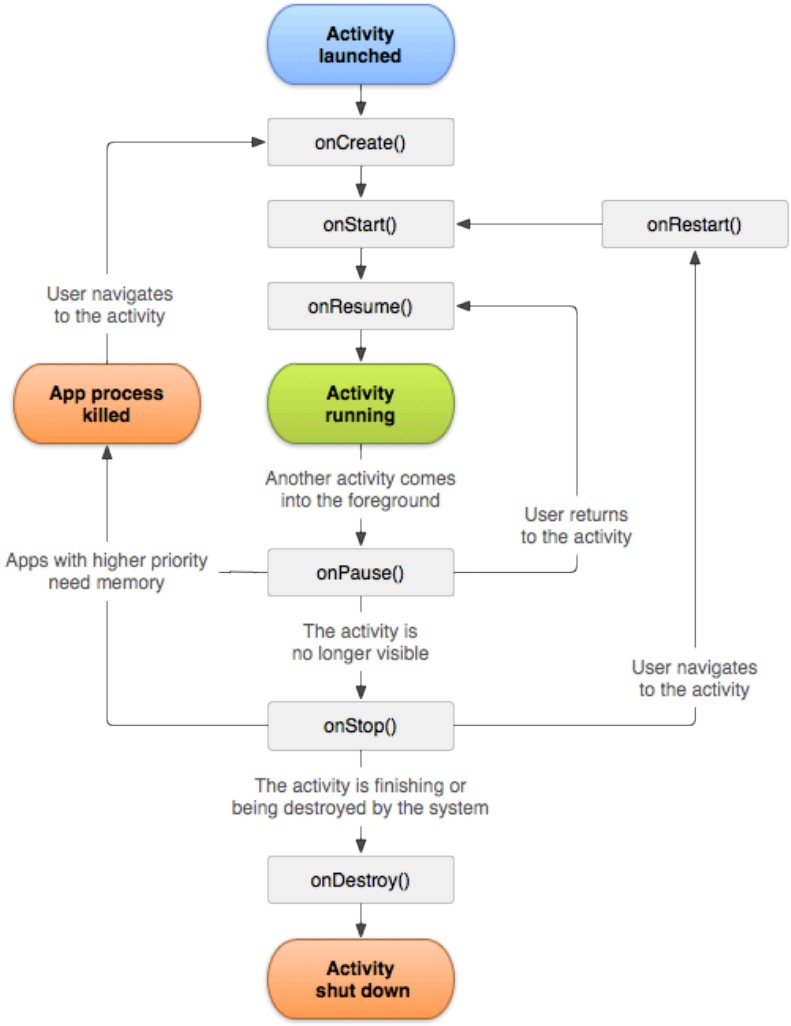


Figura 10 – Ciclo de vida de uma Activity

De seguida são descritos os métodos invocados quando uma Activity se encontra num dos sete estados ilustrados na Figura 10:

- onCreate() – Este método é invocado quando a Activity é criada pela primeira vez. Aqui devem ser feitas todas as configurações estáticas, como por exemplo criar *views*, atribuir dados a listas, etc.
- onStart() – Chamado quando a Activity se torna visível para o utilizador
- onResume() – Este método é executado quando o utilizador pode interagir com a Activity, ou seja, quando esta se encontra no topo da *stack* das Activities.
- onPause() – Invocado quando o sistema está perto de chamar a próxima Activity, permite terminar acções que estão a decorrer (animações e reprodução de vídeo) e armazenar informação de forma permanente no caso de o utilizador sair da aplicação.

- onStop() – Chamado quando a Activity deixa de se encontrar visível para o utilizador, devido a uma outra Activity estar a ser exibida ao utilizador.
- onRestart() – Executado quando a Activity estiver parada e prestes a ser iniciada novamente.
- onDestroy() – É a última chamada antes da Activity ser destruída. Isto acontece porque a Activity está a terminar (alguém invoca o método *finish*) ou o sistema está a destruir recursos para libertar espaço.

Todos estes estados devem ser *overridden* na classe da Activity para que o Android os invoque na altura apropriada.

Intents

Intent é um mecanismo utilizado para descrever uma ação abstrata que deve ser efetuada quando se pretende por exemplo enviar um e-mail, escolher um contacto, abrir o *browser*, etc. Cada Intent está associado a um componente, que é chamado quando é feito o tratamento do Intent.

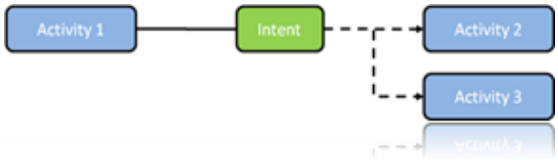


Figura 11 – Funcionamento Intent

Um Intent pode ser utilizado com os métodos: *startActivity* para iniciar uma nova Activity (

Figura 11); *broadcastIntent* para enviar uma mensagem *broadcast*; ou com os métodos *startService* ou *bindService* para comunicar com um serviço que esteja a ser executado em *background*.

Services

Um Service é um componente da aplicação que não possui interface gráfica (não há interação direta com o utilizador) e que pode executar operações de longa execução em *background*. Um exemplo simples de utilização de um serviço é o de um reprodutor de música: o reprodutor é iniciado pelo utilizador através de uma Activity e a música continua a tocar mesmo quando o utilizador abre uma Activity diferente ou outra aplicação. Para que o reprodutor continue a tocar é necessária a utilização de um serviço que seja "independente" da Activity.

Content Providers

Um Content Provider é um mecanismo de partilha de informação entre aplicações. O Android oferece vários Content Providers, como por exemplo o dos contactos, que permite que outras aplicações acedam a nomes, endereços, números, etc.

Broadcast Receivers

Broadcast Receiver é um componente do Android que responde a determinados eventos, geralmente enviados pelo sistema. Quando algum evento acontece, o sistema operativo envia uma mensagem em *broadcast* de modo a que seja recebida pelos Broadcast Receivers. Alguns exemplos de mensagens recebidas são avisos de bateria fraca ou alertas de que o telemóvel está a receber uma chamada, entre outras situações.

Resources

Um recurso pode ser uma *string*, um *bitmap* ou qualquer outro pedaço de informação de que a aplicação precise. Em tempo de compilação, todos os recursos são carregados para a aplicação. O compilador de recursos (AAPT - *Android Asset Packaging Tool*) processa todos os recursos de acordo com o seu formato e diretório em que estão armazenados. Após compilados todos os recursos, é gerada uma classe de nome "R", que contém os identificadores de todos os recursos da aplicação.

Android Manifest

Todas as aplicações devem ter um ficheiro designado AndroidManifest.xml no seu diretório raiz. Este ficheiro apresenta ao sistema operativo Android informação essencial sobre a aplicação que o sistema tem de ter antes de poder executar código. Entre outros aspectos, o manifesto tem como funções:

- Dar o nome ao Java *package* da aplicação. O *package name* funciona como identificador único para a aplicação.
- Descrever os componentes da aplicação, nomeadamente as Activities, serviços, *broadcast receivers* e *content providers* que compõe a aplicação. Nomeia as classes que implementam cada um dos componentes e indica as suas capacidades. Estas declarações permitem que o sistema operativo Android saiba em que consistem os componentes e sob que condições poderão ser executados.
- Determinar que processos irão hospedar componentes da aplicação.

- Declarar que permissões a aplicação tem de ter de modo a poder aceder a áreas protegidas da API e interagir com outras aplicações.
- Declarar as permissões que outras aplicações têm de ter para poder interagir com os componentes da aplicação.
- Definir o nível mínimo de Android API que a aplicação requer.

2.1.2 iOS

O iOS é um sistema operativo móvel que deriva do Mac OS X (baseado no Darwin BSD (OpenSource-Apple, 2014)), sendo desenvolvido pela Apple e distribuído exclusivamente em dispositivos da marca: iPhone, iPad, iPod e Apple TV.

Em Janeiro de 2007, durante a Macworld Conference & Expo, a Apple apresentou o sistema operativo iPhone OS, originalmente desenhado apenas para o iPhone e que não permitia a instalação de aplicações desenvolvidas por terceiros (Cohen, 2007). A elevada aceitação que o iPhone teve no mercado levou a Apple a anunciar o lançamento do SDK em Março de 2008 (Dalrymple, 2008), abrindo à comunidade de programadores a possibilidade de desenvolver aplicações para este sistema operativo. Com o lançamento do iPad em 2010 (Apple, 2010), o iPhone OS passou a designar-se iOS (Patel, 2010).

Atualmente no desenvolvimento de aplicações iOS é utilizada a linguagem de programação Objective-C, que obriga à utilização do Xcode (Developer-Apple-1, 2014), um ambiente de desenvolvimento que contém a mais recente versão do SDK e o iOS Simulator, um emulador que permite que as aplicações sejam testadas durante o processo de desenvolvimento. A compilação de uma aplicação iOS resulta num ficheiro do tipo *iOS Application Archive* (IPA) e para que a mesma possa ser testada num dispositivo físico é necessário possuir uma conta de "*Apple developer*", que implica o pagamento de uma quantia anual de $99, aproximadamente 78€ (DeveloperApple-2, 2014).

2.1.2.1 Objective-C

O Objective-C, desenvolvido no início dos anos 80 por Brad Cox e Tom Love, foi adquirido pela NeXT em 1988. Em 2006, a Apple anunciou o lançamento do Objective-C 2.0 que incluía, para além de melhorias ao nível da sintaxe e da performance, o suporte a sistemas de 64-bit. (TechnoBuffalo, 2011)

Esta linguagem de programação consiste numa fina camada por cima do C que permite a qualquer compilador de Objective-C compilar código C. Todas as operações não orientadas a objetos derivam do C, ao invés das operações orientadas a objetos, que são semelhantes ao sistema de mensagens existente na linguagem de programação Smalltalk (Smalltalk, 2014). Em Objective-C, uma instância de um objeto não chama métodos, envia uma mensagem (Código Fonte 1).

```
[obj method: argument];
```

Código Fonte 1 – Envio de uma mensagem em Objetive-C

Tal como na linguagem de programação C, o Objective-C obriga a que a implementação de uma classe esteja separada da declaração dos métodos que esta contém. Assim, também a declaração da interface se encontra definida no ficheiro .h (*header file*) e a implementação da interface no ficheiro .m.

Até Julho do presente ano de 2014, o desenvolvimento de aplicações nativas para iOS era feito exclusivamente com recurso ao Objective-C. Com o anúncio do iOS 8 na Worldwide Developers Conference 2014, a Apple divulgou a criação de uma nova linguagem de programação, o Swift (DeveloperApple-3, 2014). Apesar da pouca maturidade, com a sua sintaxe concisa e expressiva, o Swift promete angariar mais interessados para o desenvolvimento de aplicações para iOS.

2.1.2.2 Arquitetura

A arquitetura do iOS é formada por quatro camadas, sendo que cada uma oferece um conjunto de *frameworks* que podem ser utilizados durante o desenvolvimento de aplicações. As aplicações desenvolvidas comunicam com interfaces do sistema bem definidas, que por sua vez comunicarão com o *hardware*. As interfaces do sistema operativo facilitam o desenvolvimento de aplicações e garantem o seu funcionamento de forma consistente em dispositivos com diferentes capacidades de *hardware*. As quatro camadas que compõem este sistema operativo são: Core OS, Core Services, Media e

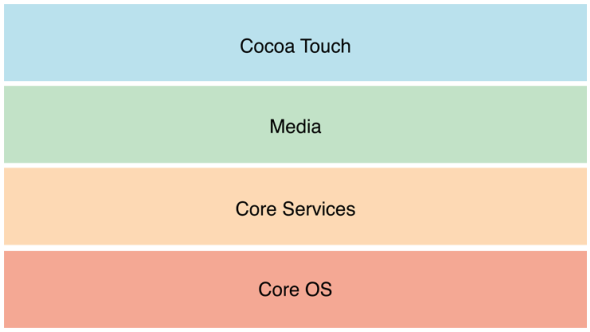


Figura 12 – Arquitetura iOS

Cocoa Touch.

Na arquitectura do iOS as camadas inferiores contêm serviços e tecnologias fundamentais, suportando as camadas superiores que implementam serviços e tecnologias mais sofisticadas. De seguida serão abordadas de forma detalhada as quatro camadas indicadas na Figura 12.

Core OS

A camada Core OS (Figura 13) contém funcionalidades sobre as quais a maioria das tecnologias de nível superior é construída. Esta camada é utilizada diretamente pelos programadores apenas quando existe a necessidade de lidar explicitamente com a segurança da aplicação ou de comunicar com um acessório de *hardware* externo. Os *frameworks* disponíveis nesta camada são: Accelerate Framework, Core Bluetooth Framework, External Accessory Framework, Generic Security Services Framework e Security Framework.

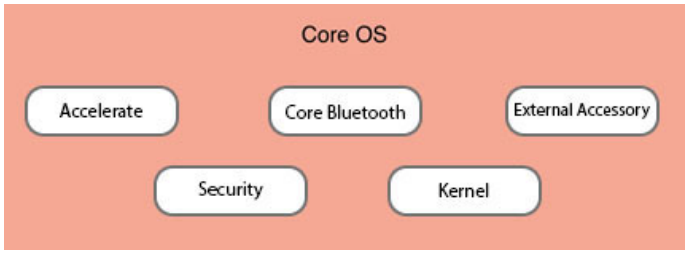


Figura 13 – Arquitetura iOS: Core OS Layer

Da mesma camada faz também parte o nível de sistema, componente que engloba o kernel do sistema operativo, os *drivers* de *hardware* e as interfaces UNIX, que comunicam com o *hardware*. O kernel é responsável pela gestão do sistema virtual de memória, *threads*, sistema de ficheiros, redes, comunicação entre processos, entre outros aspetos. Por razões de segurança, o acesso ao kernel e aos *drivers* é restrito a um conjunto limitado de *frameworks* de sistema e de aplicações (DeveloperApple-4, 2014).

Biblioteca Libsystem

O iOS fornece um conjunto de interfaces de acesso a recursos do sistema operativo através da biblioteca Libsystem. As interfaces baseadas em C dão suporte a *Threading*, *sockets* BSD, acesso ao sistema de ficheiros, serviços DNS (*Domain Name System*), alocação de memória, entre outros.

Accelerate Framework

O *framework* Accelerate contém interfaces utilizadas para executar o processamento de sinais digitais (DSP), de cálculos de álgebra linear e de imagem. O código deste *framework* encontra-se otimizado para as configurações de *hardware* presentes nos dispositivos iOS.

Security Framework

Este *framework* fornece interfaces para a gestão de certificados, chaves públicas e privadas, políticas de confiança e para a geração de números aleatórios criptograficamente seguros, com o objetivo de garantir a segurança dos dados utilizados pela aplicação.

Core Services

A camada Core Services (Figura 14) contém serviços do sistema que são fundamentais para a execução das aplicações. Entre eles são considerados serviços chave a Core Foundation e a Foundation Framework, que definem os tipos básicos de dados que todas as aplicações utilizam. Esta camada inclui também tecnologias individuais que suportam funcionalidades como localização, iCloud, "social media" e *networking*. (DeveloperApple-5, 2014)

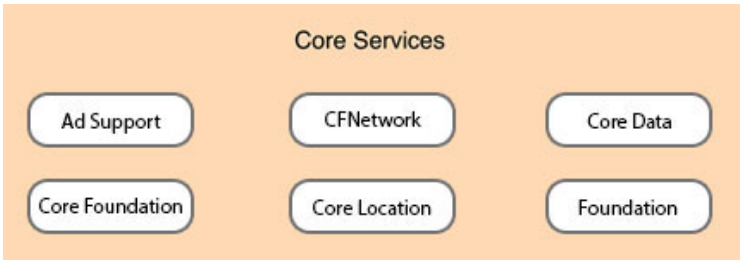


Figura 14– Arquitetura iOS: Core Services Layer

De seguida são abordadas algumas das principais funcionalidades de alto nível disponíveis nesta camada do sistema operativo.

Armazenamento iCloud

O armazenamento iCloud permite à aplicação gravar documentos e dados do utilizador em servidores da Apple, que possibilita assim aos utilizadores aceder a esses

itens a partir de todos os seus computadores e aparelhos iOS. Tornar os documentos do utilizador acessíveis através do iCloud significa que os utilizadores podem visualizar ou editar os mesmos a partir de qualquer dispositivo, sem a necessidade de explicitamente sincronizar ou transferir ficheiros.

Proteção de Dados

A proteção de dados permite que aplicações que utilizem dados sensíveis do utilizador tirem partido da encriptação incorporada em alguns dispositivos. Quando a aplicação define um ficheiro específico como protegido, o sistema armazena esse ficheiro no disco em formato encriptado. Enquanto o aparelho estiver bloqueado, os conteúdos do ficheiro estão inacessíveis tanto para a aplicação como para qualquer potencial intruso.

Suporte de Partilha de Ficheiros

O suporte de partilha de ficheiros permite à aplicação disponibilizar ficheiros do utilizador no iTunes (versão 9.1 ou posteriores). Uma aplicação que declare suportar a partilha de ficheiros, torna o conteúdo do seu diretório /Documents disponível para o utilizador, que pode desta forma mover ficheiros para dentro e fora deste diretório. Esta funcionalidade não permite que a aplicação partilhe ficheiros com outras aplicações no mesmo aparelho.

Compras dentro da Aplicação (*In-App Purchase*)

Esta funcionalidade possibilita a venda de conteúdos e serviços específicos da aplicação a partir da mesma. Pode ser implementada utilizando o *framework* Store Kit, que disponibiliza a infraestrutura necessária para processar transações financeiras através da conta do iTunes do utilizador.

SQLite

A biblioteca SQLite permite a integração na aplicação de uma base de dados SQL leve, sem necessidade de recorrer a um servidor remoto. A partir da aplicação é possível criar uma base de dados local e gerir as tabelas e registos existentes.

Suporte XML

O *framework* Foundation fornece a classe NSXMLParser, de modo a receber os elementos de um document XML. O suporte adicional para manipulação de conteúdo XML é disponibilizadopela biblioteca de libxml2. Esta biblioteca open-source permite analisar ou escrever informação XML rapidamente e transformar conteúdo XML em HTML.

Depois de uma análise às funcionalidades oferecidas pelos Core Services, abordamos agora alguns dos *frameworks* que fazem parte desta camada.

Ad Support Framework

O *framework* Ad Support (AdSupport.framework) dá acesso a um identificador que as aplicações podem utilizar para efeitos de publicidade.

CFNetwork Framework

O *framework* CFNetwork (CFNetwork.framework) consiste num conjunto de interfaces de alta performance com base na linguagem de programação C que utiliza abstrações orientadas a objetos para trabalhar com protocolos de redes.

Com o CFNetwork é possível: utilizar *sockets* BSD (*Berkeley Software Distribution*), criar ligações encriptadas utilizando SSL (Secure Sockets Layer) ou TLS (*Transport Layer Security*), resolver *hosts* DNS, trabalhar com servidores HTTP (*Hypertext Transfer Protocol)*, autenticar servidores HTTP/HTTPS (*Hypertext Transfer Protocol Secure)* e trabalhar com servidores FTP (*File Transfer Protocol*).

Core Data Framework

Core Data (CoreData.framework) é uma tecnologia de gestão do modelo de dados de uma aplicação *Model-View-Controller*. Esta tecnologia está destinada à utilização em aplicações nas quais o modelo de dados é altamente estruturado.

O Core Data reduz significativamente a quantidade de código fonte necessário para a criação da aplicação. Fornece as seguintes funcionalidades:

- Armazenamento de objetos, isto é classes, numa base de dados SQLite para obter uma boa performance;

- Uma classe NSFetchedResultsController para gerir resultados para *table views*;
- Suporte à validação de valores de propriedades/campos;
- Suporte para alterações de propagação;
- Garantia de que as relações entre objetos se mantêm consistentes;
- Suporte para agrupamento, filtragem e organização de dados na memória.

Core Foundation Framework

O *framework* Core Foundation (CoreFoundation.framework) consiste num conjunto de interfaces baseadas na linguagem de programação C que oferecem a gestão básica de dados e funcionalidades de serviço para iOS. Este *framework* inclui suporte para diversas funcionalidades, como por exemplo: coleções de tipos de dados (*arrays*, *sets*, entre outros), gestão de *strings*, gestão da data e hora e manipulação de URLs (URL - *Uniform Resource Locator*).

Core Location Framework

O Core Location (CoreLocation.framework) proporciona informação de localização e direção à aplicação. Para adquirir informações da localização (como longitude e latitude) atual do utilizador, o *framework* utiliza o GPS (*Global Positioning System*) integrado, informações Wi-Fi ou células GSM (*Global System for Mobile Communications)*, através de A-GPS – GPS Assistido.

Foundation Framework

O *framework* Foundation (Foundation.framework) disponibiliza interfaces em Objective-C que permitem aceder a muitas das funcionalidades encontradas no Core Foundation Framework. O Foundation Framework dá suporte a funcionalidades como gestão de *strings*, gestão de data e hora, *threads*, correspondência de expressões regulares e suporte de *cache*.

Media

A camada Media (Figura 15) contém tecnologias necessárias para criação e visualização de gráficos, ficheiros áudio e vídeo, que facilitam o desenvolvimento de experiências multimédia numa aplicação (DeveloperApple-6, 2014).

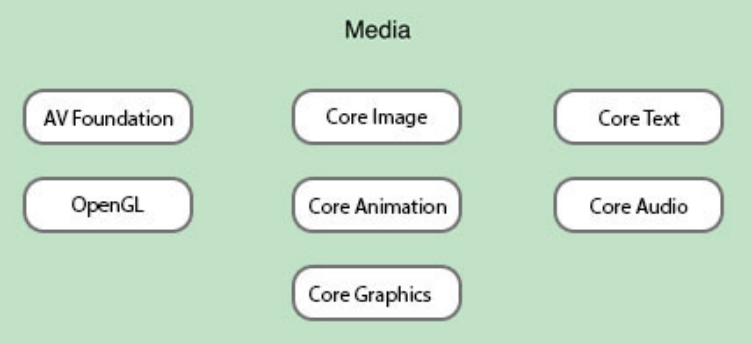


Figura 15 – Arquitetura iOS: Media Layer

Tecnologias de Gráficos

O *framework* mais utilizado desta camada é o UIKit (*User Interface Kit*), que fornece várias tecnologias de suporte a gráficos no sentido de promover a interatividade com o utilizador. Este *framework* proporciona formas eficientes de renderização de imagens e conteúdos baseados em texto. Algumas das tecnologias disponíveis nesta camada são:

- Core Graphics – Lida com a renderização de vetores com base em imagens 2D;
- OpenGL ES – Oferece suporte para renderização 2D e 3D utilizando aceleração de *hardware*, é um *framework* frequentemente utilizado no desenvolvimento de jogos;
- Image I/O – Fornece interfaces de leitura e criação de imagens na maior parte dos formatos existentes;
- Assets Library – Suporta o acesso a fotos e vídeos armazenados no dispositivo, sendo geralmente utilizado quando se objectiva integrar conteúdos do utilizador na aplicação.

Tecnologias de Áudio

As tecnologias de áudio disponíveis na camada Media conferem a capacidade de reproduzir e gravar áudio de alta qualidade. Dos *frameworks* que esta camada oferece, destacam-se:

- Media Player – Dá acesso à biblioteca do iTunes do utilizador da aplicação e suporta a reprodução de faixas de áudio e listas;

- AV Foundation – Oferece um conjunto de interfaces que facilitam a gestão de reprodução e gravação de áudio;
- Core Audio – Contém interfaces simples e sofisticadas que permitem a reprodução de sons de alerta do sistema e permitem também efetuar *stream* de áudio, entre outras funcionalidades.

Tecnologias de Vídeo

As tecnologias de vídeo do iOS dão suporte à gestão de conteúdos de vídeo, com a finalidade de os incorporar, reproduzir e capturar. Permite assim, entre outras funções, a reprodução de *streaming* de conteúdos vídeo na Internet. Os *frameworks* que suportam as tecnologias de vídeo são apresentados de seguida:

- Classe UIImagePickerController – Faz parte do *framework* UIKit e é utilizada para fornecer ao utilizador um ecrã onde seja possível selecionar ficheiros media;
- Media Player – Conjunto de interfaces que podem ser utilizadas para reproduzir filmes em ecrã completo ou parcial, oferecendo ao utilizador alguns controlos de reprodução;
- AV Foundation – Oferece interfaces de reprodução e gravação de vídeo, sendo geralmente utilizado em situações onde é necessário um maior controlo sobre a reprodução ou gravação do vídeo, como por exemplo em aplicações de realidade aumentada;
- Core Media – Fornece tipos de dados de baixo nível e interfaces de manipulação de *media*, não sendo necessário para a maior parte das aplicações utilizar este *framework* diretamente.

As tecnologias de vídeo do iOS suportam vários padrões de compressão e de reprodução para vários formatos, como por exemplo mov, mp4, m4v e 3gp.

Cocoa Touch

A camada Cocoa Touch (Figura 16) contém *frameworks* essenciais para a construção de aplicações e define a infra-estrutura das tecnologias fundamentais.

Figura 16 – Arquitetura iOS: Cocoa Touch Layer

Consideram-se exemplos de tecnologias fundamentais: o suporte a multi-tarefas, o serviço de notificações *push*, os serviços de *User Interface* (UI) (que incluem o reconhecimento de gestos ou a preservação do estado do ecrã) (DeveloperApple-7, n.d.). Algumas das tecnologias mais utilizadas ao nível da UI são:

- TextKit – Conjunto de classes utilizadas para a manipulação de texto e tipografia, sendo possível com este Kit colocar texto em parágrafos, colunas, páginas ou fazer o texto fluir sobre imagens;
- Storyboard – Permite a criar toda a UI num só local de modo a que seja possível visualizar todas as *views* e *viewcontrollers* e entender como funcionam em conjunto, consistindo também uma parte importante da *storyboard* a definição de *segues*, pois os mesmos determinam a transição de um ecrã para outro;
- Gesture Recognizers – Esta tecnologia permite detetar e definir gestos comuns do sistema operativo, como *swipes* e *pinches* (ver Figura 17).

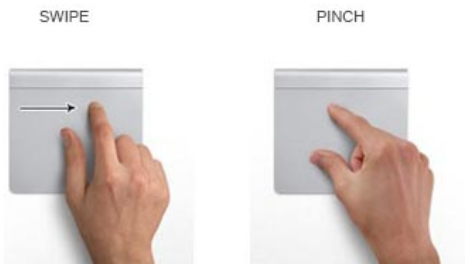


Figura 17 – Ilustração dos gestos *swipe* e *pinch*.

Nesta camada, os *frameworks* com maior destaque são:

- iAd Framework - Permite a integração de publicidade na aplicação;
- Map Kit Framework – Usado para a manipulação de mapas;
- Message UI Framework - Facilita o envio de mensagens de texto como SMS (*Short Message Service*) e e-mail;

- UIKit Framework - Fornece uma infra-estrutura fundamental para a implementação de gráficos e aplicações orientadas a eventos, incluindo funcionalidades como: a gestão básica da aplicação (incluindo o *loop* principal de execução), a capacidade de multi-tarefa, o suporte para texto e conteúdos *web*, informações sobre a bateria, o suporte às funções copiar/colar/cortar, a partilha de conteúdos através de e-mail/Twitter/Facebook e o suporte de acessibilidade para utilizadores com deficiências, entre outras.

2.2 Estratégias de desenvolvimento de aplicações móveis

Com o aumento da utilização de dispositivos móveis por parte dos consumidores, as aplicações desenvolvidas para este tipo específico de dispositivos, conhecidas por *apps*, têm vindo a ganhar cada vez mais popularidade. No mundo empresarial existem cada vez mais organizações a utilizar esta ferramenta no sentido de suportar e desenvolver o seu negócio.

A distribuição de aplicações para dispositivos móveis passa maioritariamente pelas chamadas "*app stores*", uma espécie de loja *online* onde os utilizadores podem encontrar o *software* de que precisam. O tipo de distribuição pode variar, existindo aplicações gratuitas e pagas.

As aplicações desenvolvidas para dispositivos móveis têm caraterísticas específicas quando comparadas com as de computadores tradicionais, uma vez que os dispositivos móveis contêm *hardware* próprio, tal como sensores, câmara e GPS. Adicionalmente, a maioria dos dispositivos móveis possui ecrãs relativamente pequenos, com o toque a servir de método de interação por parte do utilizador (Pastore, 2013).

2.2.1 Tipos de aplicações móveis

Consoante a estratégia de desenvolvimento, as aplicações existentes para dispositivos móveis dividem-se em três tipos: nativas, *web* e híbridas. As aplicações nativas são desenvolvidas na linguagem de programação definida pela plataforma e permitem utilizar todo o potencial das funcionalidades do dispositivo, garantindo uma interação excelente com o utilizador. A principal desvantagem desta abordagem é que reduz o número de possíveis utilizadores, pois a aplicação precisa de ser desenvolvida de forma diferente para cada sistema operativo em que a queiramos suportar.

Aplicações para dispositivos com o sistema operativo iOS são desenvolvidas com a linguagem de programação Objective-C, para Android com Java e para Windows Mobile com C#. A Figura 18 ilustra o processo de desenvolvimento da mesma aplicação para diferentes sistemas operativos.

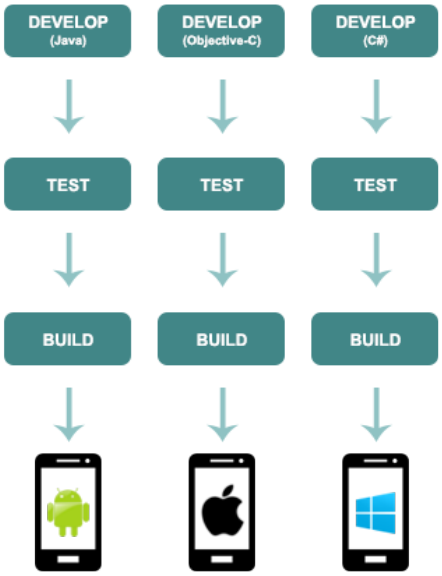


Figura 18 – Etapas do desenvolvimento de aplicações nativas

Figura 19 – Etapas do desenvolvimento de aplicações *web*

As aplicações *web* (Figura 19) caracterizam-se por serem executadas num navegador de Internet. Não necessitam de instalação, nem mesmo de código nativo. Esta particularidade permite aos programadores criar uma aplicação cujo código é escrito apenas uma vez e que se encontra dividido em três componentes: HTML5, JavaScript e CSS (*Cascading Style Sheets*). Utilizando esta abordagem, não é necessário desenvolver a mesma aplicação para vários sistemas operativos, facilitando o processo de desenvolvimento e manutenção (Wolf & Huffstadt, 2013).

As aplicações híbridas (Figura 20) combinam a portabilidade das aplicações *web*, através da utilização de HTML5 e CSS, com a facilidade de acesso a recursos do dispositivo, resultado da utilização de APIs nativas. Um estudo efetuado pela Gartner em Abril de 2013 (Gartner-4, 2013) aponta para que em 2016, 60% das aplicações empresariais (*business-to-employee*) serão híbridas, referindo que em 2015 o HTML5 será a linguagem mais utilizada no desenvolvimento de aplicações móveis. A promessa da evolução do HTML5 leva a que esta abordagem seja cada vez mais tida em consideração por parte das empresas.

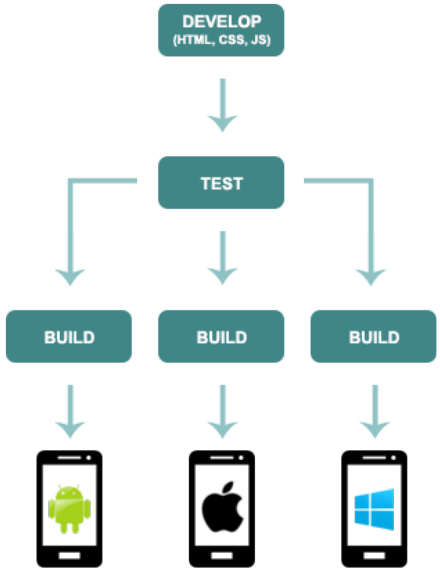


Figura 20 – Etapas do desenvolvimento de aplicações híbridas

A possibilidade de desenvolver o código de uma aplicação móvel apenas uma vez e poder instalar a mesma em diferentes sistemas operativos tem um enorme potencial. Contudo, o facto de a aplicação ser criada através da utilização de *frameworks* que trabalham como *middle layer* reduz significativamente a sua performance, comparativamente às aplicações nativas.

2.2.2 O paradigma multi-plataforma

Em teoria as aplicações híbridas são uma excelente abordagem, uma vez que permitem que o seu código seja desenvolvido num *framework* e executado em dispositivos com sistemas operativos diferentes, sem serem necessários gastos adicionais no desenvolvimento.

O PhoneGap (PhoneGap, 2014), comprado em 2011 pela Adobe, é o mais conhecido *framework* multi-plataforma. Embora permita desenvolver o *front-end* com recurso apenas a tecnologias *web*, possibilita a utilização de funcionalidades da API nativa (como por exemplo, utilizar a câmara) através de uma biblioteca Javascript, que combina o *front-end* com contentores de código nativo. Para que o *front-end* seja visualizado em HTML, as aplicações utilizam os componentes UIWebView para iOS e WebView para Android.

Nas aplicações *web* e híbridas, a interface com o utilizador é desenhada através de HTML, o que torna difícil manter o chamado "*look and feel*" das aplicações nativas. No sentido de tentar colmatar essa falha dos *frameworks* multi-plataforma podem se utilizar *frameworks* de UI, como por exemplo o jQTouch (JqtJS, 2014) e o jQuery Mobile (JQueryMobile, 2014). O Apache Cordova, um dos mais recentes *frameworks* multi-plataforma que tem por base o PhoneGap, vem já com *frameworks* de UI incorporadas, como o jQuery Mobile, Dojo Mobile (DojoToolKit, 2014) e o Sencha Touch (Sencha, 2014).

2.2.2.1 Os requisitos das *frameworks* mutiplataforma

Quando considerado o uso de um *framework* multi-plataforma, existem alguns detalhes que é necessário ter em conta, nomeadamente em termos de *design*, limitações técnicas (naturalmente ligadas à atualização das plataformas nativas), suporte técnico relativo ao *framework*, entre outros.

No *design* da aplicação é necessário identificar quais as funcionalidades que a aplicação terá de suportar, de modo a avaliar se será necessário utilizar código nativo ou desenvolver *plugins* que não existam para complementar o *framework*. No caso de ser necessário, é possível que o tempo e dinheiro poupados na utilização desta abordagem sejam mínimos quando comparados com uma abordagem nativa de desenvolvimento.

2.2.2.2 Problemas no *design* de aplicações

Quando se pretende desenvolver uma aplicação móvel para mais do que um tipo de plataforma é preciso ter consciência de que estas apresentam diferenças significativas entre si. É um facto que existem diferentes requisitos para uma aplicação que se adapte na perfeição a um iPad ou para uma que se adapte na sua plenitude a um Samsung Galaxy. Há diferenças na localização das *tabs* no ecrã (fundo ou topo), no tamanho dos *icons*, nas resoluções de ecrã e respetivas densidades, entre outros exemplos, devendo tudo isto ser tido em consideração, obrigando o programador a conhecer as diferentes plataformas para as quais pretende desenvolver.

2.2.2.3 Limitações das *frameworks* multi-plataforma

Outro aspeto a considerar é a evolução das funcionalidades dos sistemas operativos móveis. Cada *framework* multi-plataforma suporta apenas um conjunto de funcionalidades de cada sistema operativo, limitando em alguns casos as potencialidades das aplicações desenvolvidas através dos mesmos.

Quando uma nova funcionalidade de um determinado sistema operativo é lançada, é necessário esperar que a mesma seja incluída na API do *framework* para que possa ser utilizada. Alguns *frameworks* não permitem que se adicionem novas funcionalidades, o que pode conduzir à perda de oportunidades.

Dependendo de quais sejam os requisitos de uma aplicação, pode ser necessário o desenvolvimento de *plugins* diferentes para cada sistema operativo. Esta necessidade pode levar a uma acentuada perda de recursos monetários. A única forma de garantir que uma aplicação é construída com as tecnologias mais apropriadas e eficientes é construí-la nativamente, isto é, com recurso à linguagem de programação definida para a plataforma.

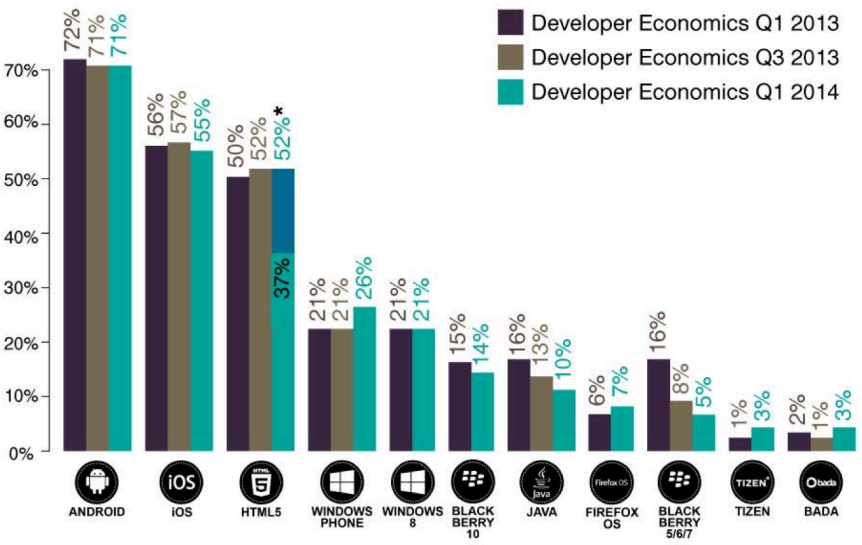
2.2.2.4 Contexto da aplicação

Um fator indubitavelmente importante e que deve obrigatoriamente ser tido em conta é o contexto da aplicação. Se uma aplicação apresenta conteúdos de uma rede social, incorpora funcionalidades baseadas na localização do utilizador, utiliza a câmara e/ou animações gráficas e necessita de diferentes *designs* para *tablets* e *smartphones*, então deve ser efetuada uma análise cuidadosa. O GPS e a câmara são funcionalidades que estão incorporadas em grande parte dos *frameworks* multi-plataforma, mas cada sistema operativo trata de diferente forma cada uma dessas funcionalidades, o que requer algum cuidado quando se desenvolve a aplicação.

2.2.3 Frameworks para aplicações multi-plataforma

A ideia de ser possível desenvolver o código para uma aplicação móvel apenas uma vez e instalar a mesma em diferentes sistemas operativos tem vindo a ganhar cada vez mais adeptos, já que é extremamente vantajosa por permitir chegar a um maior número de utilizadores. Hoje em dia são mais de uma centena, as ferramentas multi-plataforma disponíveis no mercado.

Um inquérito realizado no âmbito de um estudo da Developer Economics (VisionMobile, 2014) no primeiro trimestre de 2014, revelou que 52% dos inquiridos (7.000 programadores de aplicações móveis provenientes de 127 países) desenvolve aplicações móveis com recurso ao HTML5 (Figura 21).



*This figure includes developers who develop hybrid apps and apps developed with HTML5 but translated to native code.

Figura 21 – Gráfico comparativo da utilização das tecnologias de desenvolvimento de aplicações móveis

A partir do mesmo estudo é possível verificar que 30% dos programadores inquiridos utilizam *frameworks* multi-plataforma, como o PhoneGap, Titanium Appcelerator e o Adobe AIR.

2.2.3.1 PhoneGap

O PhoneGap, que é provavelmente a solução multi-plataforma mais conhecida da atualidade, permite desenvolver aplicações híbridas e *web*. Desenvolvido inicialmente pela Nitobi, foi comprado em Outubro de 2011 pela Adobe. Mais tarde, o seu código fonte seria entregue à Apache Software Foundation, dando origem ao Apache Cordova, que hoje distribui o PhoneGap de forma gratuita (Brian, 2012).

Este *framework* é utilizado com vista ao desenvolvimento de aplicações para dispositivos móveis, recorrendo a tecnologias *web*. Permite construir aplicações para *smartphones* e *tablets* utilizando HTML, CSS e JavasScript, sem ser necessário recorrer a linguagens de programação como Objective-C, Java ou C#.

Às aplicações desenvolvidas através do PhoneGap é adicionada uma camada de código nativo, que permite a instalação das mesmas nos dispositivos através das lojas de aplicações em múltiplas plataformas (Sonmez, 2013). A maioria dos sistemas operativos móveis fornece um componente do tipo *web view*, utilizado para exibir conteúdos HTML locais ou de um servidor

remoto. O contentor nativo gerado pelo PhoneGap carrega as páginas HTML desenvolvidas pelo programador para o componente *web view* do sistema operativo, exibindo o resultado do HTML como interface para o utilizador. No fundo, estas aplicações são executadas localmente no *browser* do dispositivo, utilizando as APIs de Javascript para efetuar chamadas ao código nativo do sistema operativo.

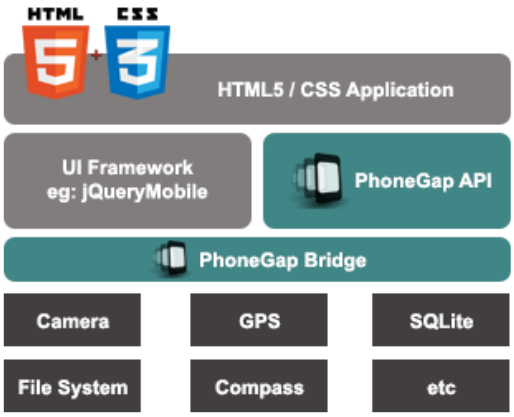


Figura 22 – Arquitetura PhoneGap

O sistema de mensagens da *web view* utilizado para comunicar o código nativo através da PhoneGap Bridge (Figura 22) é a chave desta tecnologia, uma vez que permite que uma aplicação *web* instalada localmente invoque código nativo.

Pontos fortes do PhoneGap

- É suportado por todos os sistemas operativos que contenham uma *web view*;
- Suporta HTML5 e CSS3;
- Em teoria basta desenvolver o código da aplicação uma única vez para diferentes sistemas operativos – "*code once, run everywhere*";
- Permite a utilização de bibliotecas para desenhar a UI da aplicação, como por exemplo o Sencha Touch ou o MooTools (MooTools, 2014).

Pontos fracos do PhoneGap

- A qualidade da UI depende do motor de renderização de cada sistema operativo, isto é, por exemplo, o motor do iOS é bastante potente comparativamente ao utilizado em Android, que é considerado aceitável;
- Os problemas relacionados com *cross-browser* nas aplicações *web* tradicionais também precisam de ser tratados no PhoneGap;

- A UI desenvolvida não contém controladores nativos;
- É impossível alcançar a performance e a capacidade de resposta das aplicações nativas.

2.2.3.2 Titanium Appcelerator

Outro *framework* amplamente conhecido é o Titanium Appcelerator, que permite o desenvolvimento de aplicações *web*, híbridas e nativas[3] (Environment, 2014) em Javascript, através da sua API (Traeg, 2014). O Titanium é desenvolvido com base em dois princípios relacionados com o desenvolvimento de aplicações móveis:

- Existe um conjunto de funcionalidades que os mais diversos sistemas operativos oferecem, as quais podem ser normalizadas;
- Cada sistema operativo tem funcionalidades e convenções de UI específicas que devem ser respeitadas e tratadas isoladamente, para se obter uma melhor experiência de utilização.

Com base nestes princípios, é possível afirmar que o Titanium Appcelerator não procura fornecer uma API que permita desenvolver o código da aplicação uma única vez para diferentes sistemas operativos – "*code once, run everywhere*". O grande objetivo desta *framework* é possibilitar a reutilização de código fonte através da sua API em Javascript (Whinnery, 2012).

Uma aplicação desenvolvida com o Titanium Appcelerator em tempo de execução é composta por três componentes:

- O código fonte Javascript, convertido num conjunto de símbolos;
- A implementação nativa da API fornecida pelo Titanium;
- Um interpretador de Javascript utilizado para avaliar o código fonte da aplicação desenvolvida em tempo de execução, sendo que em iOS o interpretador utilizado é o JavaScriptCore e em Android o V8 ou o Rhino.

[3] A afirmação de que o Titanium permite o desenvolvimento de aplicações nativas pode ser contestada pelo facto de estas utilizarem o Titanium kernel, não executando diretamente na Dalvik VM.

Ao invés de ser compilado para a linguagem de programação Objective-C ou Java, o código fonte desenvolvido é avaliado em *runtime*. Assim, como é possível verificar através da arquitetura ilustrada na Figura 23, o Titanium não utiliza nenhum componente *web view* para que as suas aplicações sejam executadas em dispositivos móveis, utilizando um ambiente de execução de Javascript, criado quando a aplicação é iniciada.

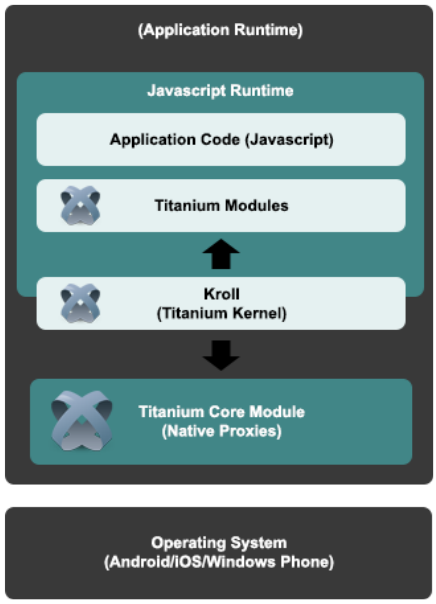


Figura 23 – Arquitetura Titanium Appcelerator

Na maioria dos casos não é utilizado HTML ou CSS nas aplicações desenvolvidas com o Titanium, uma vez que são fornecidas APIs para a utilização de componentes de interface nativos.

Pontos fortes do Titanium Appcelerator

- Permite a utilização de componentes visuais nativos, garantindo o "*look and feel*" das aplicações nativas;
- Oferece melhor performance e capacidade de resposta relativamente ao PhoneGap.

Pontos fracos do Titanium Appcelerator

- Atualmente só é possível desenvolver aplicações para Android, iOS e Windows Phone;
- Não é perfeita para programadores habituados a tecnologias *web*, uma vez que o desenvolvimento de aplicações é feito através de APIs Javascript.

- A performance e capacidade de resposta ficam ainda aquém das verificadas para as aplicações nativas;
- Muitos programadores preferem programar nativamente em Objective-C ou Java do que em Javascript.

2.2.3.3 Xamarin

O último *framework* a ser abordado designa-se Xamarin (Xamarin, 2014), e permite o desenvolvimento de aplicações em C# para as plataformas Android, iOS e Windows Phone, através da partilha entre as mesmas de grande parte do código fonte. Ao desenvolver código com o Xamarin, o programador utiliza uma camada de abstração que se encontra acima dos SDKs nativos, o que resultará numa aplicação nativa[4]. Esta abordagem limita a quantidade de código que será partilhado entre as diferentes plataformas (Sonmez, 2013).

A solução mais utilizada para partilhar código entre projetos multi-plataforma consiste na criação de um projeto partilhado, o que possibilita que o código fonte escrito seja referenciado por diferentes projetos de aplicação (DeveloperXamarin-1, 2014). O desenvolvimento de uma aplicação para as diferentes plataformas implica assim a criação de um projeto partilhado e de um projeto para cada plataforma, como se encontra ilustrado na Figura 24.

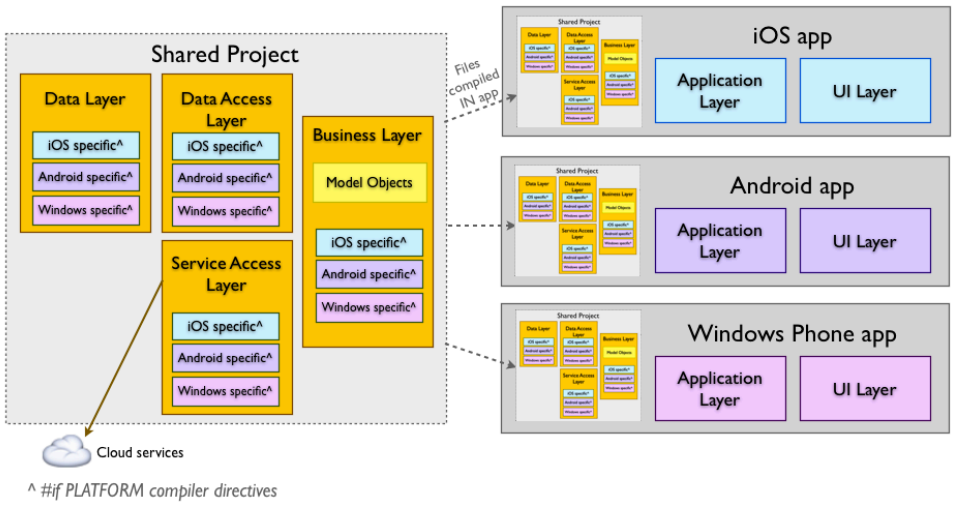


Figura 24 – Arquitetura Xamarin

[4] As aplicações Android desenvolvidas com o Xamarin são executadas na MonoVM, uma implementação em C# da Dalvik VM, podendo assim não ser consideradas aplicações nativas.

Ao contrário dos outros projetos, o projeto partilhado não gera qualquer aplicação (formato DLL - Dynamic-link library), pois será compilado como parte de cada projeto que o usa como referência. O código do projeto partilhado pode conter diretivas de compilação que permitem ativar e desativar secções de código, consoante a plataforma da aplicação para a qual o mesmo será compilado.

Para gerar as aplicações, no caso de iOS o código C# é compilado para a linguagem ARM Assembly, sendo que para Android o código é compilado com a MonoVM (MonoProject, 2014), uma máquina virtual que implementa as APIs da Dalvik Virtual Machine em C#, permitindo a execução nativa das aplicações desenvolvidas (DeveloperXamarin-2, 2014).

O Xamarin tem o seu próprio IDE (*Integrated Development Environment*), o Xamarin Studio, mas a utilização do seu SDK pode ser feita recorrendo ao Microsoft Visual Studio, através da instalação de um plugin. Tal como as outras plataformas analisadas, o Xamarin não é uma ferramenta na qual possa ser depositada total confiança, uma vez que foram reportadas 45 falhas apenas durante o mês de Agosto, das quais apenas cerca de 17 tinham sido resolvidas até ao final do mês de Setembro (Bugzilla-Xamarin, 2014).

Pontos fortes do Xamarin

- Os programadores apenas precisam de conhecer a linguagem de programação C# para programar para iOS e Android;
- Permite aproveitar entre 60% a 70% do código desenvolvido (Sonmez, 2013);
- Permite a criação de aplicações nativas.

Pontos fracos do Xamarin

- Não permite a partilha de código da UI entre Android, iOS e Windows Phone;
- Tem associada uma comunidade pequena, o que pode resultar numa morosa resolução de problemas;
- É necessário o pagamento de uma licença para desenvolver aplicações com esta ferramenta.

2.2.4 Desenvolvimento de aplicações com SDK nativo vs frameworks multi-plataforma

O interesse em desenvolver aplicações multi-plataforma levou a um aumento no número de ferramentas existentes que permitem a criação de aplicações para várias plataformas e com código fonte partilhado para as mesmas. Para a criação de aplicações híbridas e nativas, as frameworks multi-plataforma analisadas utilizam uma camada para comunicar com o sistema operativo, o PhoneGap Bridge no caso do PhoneGap, o Titanium Kernel em aplicações do Titanium Appcelerator e a MonoVM nas aplicações desenvolvidas a partir do Xamarin.

Na Tabela 1 são apresentadas as principais vantagens e desvantagens do desenvolvimento de aplicações utilizando o SDK nativo ou as *frameworks* multi-plataforma.

Tabela 1 – Vantagens e desvantagens de desenvolvimento com SDK nativo vs *frameworks* multi-plataforma

	SDK Nativo	*Frameworks* Multi-plataforma
Vantagens	• Bibliotecas atualizadas; • Acesso direto a apoio técnico (ex: documentação); • Estabilidade; • App Store de confiança para os utilizadores; • Melhores resultados a nível de UI; • Permite tirar todo o proveito do ecrã; • Permite obter a melhor performance e tempo de resposta possível.	• Rápido desenvolvimento de aplicações; • Em teoria basta escrever o código fonte uma vez e executa em várias plataformas.
Desvantagens	• Nem todas são *open-source*; • Linguagem de programação diferente; • Desenvolvimento relativamente lento; • Requer maior orçamento e mais experiência.	• Nem todas são *open-source*; • Acesso a apoio técnico limitado; • Instável; • O design da UI depende da plataforma e é ainda bastante limitado.

Através da análise da Tabela 1 é possível verificar que o desenvolvimento de aplicações recorrendo aos SDKs nativos permite obter uma aplicação estável que apresenta um melhor desempenho e uma UI familiar para o utilizador. Grande parte da comunidade de programadores *mobile* desenvolve aplicações utilizando SDKs nativos, o que facilita o acesso a recursos como tutoriais e fóruns. Este aspeto é também fundamental quando se pretende encontrar bibliotecas atualizadas e uma documentação significativamente extensa. A principal

desvantagem desta opção prende-se com o facto de o desenvolvimento das aplicações ser um processo moroso, resultando na necessidade de lhe ser associado um custo mais elevado. Por contraste, o desenvolvimento de aplicações com *frameworks* multi-plataforma pode revelar-se significativamente mais rápido, pois em teoria basta escrever o código uma vez para que a aplicação desenvolvida seja executada em diferentes sistemas operativos. As principais desvantagens relacionam-se com acesso limitado aos recursos dos dispositivos, uma fraca documentação e comunidade de reduzida dimensão.

São inúmeros os pontos que precisam de ser considerados quando se desenvolve uma aplicação móvel. A capacidade de usufruir de todas as potencialidades do dispositivo, obter a melhor performance possível e ter acesso a uma documentação extremamente detalhada pode pesar no momento de tomar decisões, comparativamente a um acesso limitado aos recursos de um dispositivo, uma performance aceitável apenas para alguns casos e a vantagem de teoricamente ser possível escrever o código fonte apenas uma vez para todos os sistemas operativos.

2.2.5 Notas finais

Atualmente a maior parte das aplicações móveis são nativas, principalmente porque as mesmas oferecem a melhor experiência de utilização possível ao utilizador. No entanto, não existe uma resposta certa para a estratégia de desenvolvimento que se deve seguir.

Em 2012, Mark Zuckerberg afirmou que um dos seus maiores erros teria sido apostar em demasia no HTML5 para as aplicações móveis (Olanoff, 2012). O fundador e diretor executivo do Facebook declarou também que desde a mudança nas aplicações de iOS para código nativo, o número de *feed stories* carregadas através da aplicação duplicou. O Financial Times, por exemplo, disponibiliza uma aplicação *web* para dispositivos móveis com iOS (iPhone e iPad) e uma aplicação nativa para dispositivos com Android.

Antes de optar por uma estratégia de desenvolvimento, há alguns fatores que devem ser tomados em conta. A análise das funcionalidades da aplicação é extremamente importante, uma vez que nos permite saber se a sua implementação poderá ser realizada através de um *framework* multi-plataforma ou não. Devem também ser considerados os requisitos de *hardware*, fator que pode afastar algumas das estratégias disponíveis.

Relativamente à performance, no caso de o objetivo ser oferecer uma aplicação que reaja rapidamente às ações de *input* do utilizador, o desenvolvimento de uma aplicação nativa acaba por ser o caminho certo.

Um fator importante mas poucas vezes considerado é a distribuição da aplicação. Apenas aplicações híbridas e nativas são passíveis de ser colocadas nas conhecidas "*app stores*". Restrições de *design*, suporte à plataforma, acesso às APIs, curva de aprendizagem, produtividade, ferramentas de desenvolvimento, e método de *debugging* são exemplos de fatores que podem e devem ser considerados.

Na generalidade dos casos a estratégia de desenvolvimento selecionada baseia-se predominantemente nos custos e tempo de desenvolvimento.

2.3 Criadores automáticos de aplicações móveis existentes

A crescente popularidade dos dispositivos móveis e o aparecimento de *frameworks* multi-plataforma levaram a um aumento do número de plataformas *online* disponíveis, com a finalidade de permitir a criação de aplicações móveis. Atualmente, qualquer pessoa sem nenhum conhecimento prévio a nível de programação é capaz de criar aplicações para dispositivos móveis, através destas plataformas. Nesta secção será feita uma análise a alguns dos *websites* desse tipo disponíveis atualmente, sendo que apenas um dos exemplos permite criar aplicações nativas.

2.3.1 Andromo

Figura 25 – Logótipo Andromo

O Andromo (Figura 25) é, entre as plataformas analisadas, a única que permite a criação de aplicações nativas para Android (Andromo, 2014). A grande desvantagem que apresenta relativamente às outras é o facto de apenas permitir a criação de aplicações para o sistema operativo da Google.

No primeiro passo para a criação de uma aplicação é necessário introduzir o nome, versão, *package name*, icon e mercado em que a aplicação será distribuída (Google Play, Amazon AppStore ou Samsung Apps).

Posteriormente é possível definir os ecrãs da aplicação (designados Activities nesta plataforma), estilos (cor do texto, imagem de fundo, estilo da *action bar* e design do ecrã principal - *dashboard*), configurações relativas a publicidade (permite a integração de diferentes serviços de publicidade, como por exemplo o AdMob (GoogleAdMob, 2014), AppBrain (AppBrain, 2014), entre outros) e a configuração de outros serviços, como por exemplo o Google App Analytics (GoogleAnalytics, 2014), que permite acompanhar as atividades do utilizador na aplicação, desde a sua instalação até às compras realizadas através da mesma.

Uma aplicação criada através do Andromo pode conter até 19 Activities, sendo que os dados comuns entre as mesmas são o título, a descrição, o *icon* e a posição no *dashboard*. Entre as várias opções disponíveis é possível criar as seguintes Activities: acerca, contactos, Facebook, mapa, galeria de fotografias, leitor de RSS (*Rich Site Summary*), Twitter, *website* e Youtube.

Em cada Activity existe um conjunto de parâmetros a preencher. Por exemplo, para o leitor de RSS é necessário introduzir o URL do *feed*, a ação que irá ser desencadeada quando o utilizador carrega num item, o estilo de cada item e ainda a existência ou não de publicidade para cada item RSS, podendo o utilizador visualizar dois tipos de publicidade no mesmo ecrã caso esteja definido algum serviço de publicidade na aplicação.

Após construída a aplicação, é disponibilizado para *download* o APK, que necessita posteriormente de ser instalado no dispositivo.

2.3.2 AppMachine

Figura 26 – Logótipo AppMachine

O AppMachine (Figura 26) permite a criação de aplicações *web* para dispositivos móveis que executem Android, iOS ou Windows Phone (AppMachine, 2014). Desde Dezembro de 2013

que esta ferramenta utiliza internamente o *framework* multi-plataforma de Javascript designado Famo.us (Famo.us, 2014) para a criação de aplicações móveis (John, 2014).

A criação de uma nova aplicação inicia-se com a introdução no formulário inicial do AppMachine do nome desejado para a aplicação e de um URL, que pode ser de um *website*, do Twitter, do Facebook ou do canal do Youtube. O AppMachine analisa o URL fornecido e recolhe todos os dados que considerar necessários, introduzindo-os na aplicação. De seguida é pedido que seja selecionado um tema para a aplicação, ou seja, o aspeto que se deseja que a aplicação gerada apresente. No *dashboard* (painel de administração), no menu relativo à aplicação, é possível encontrar as seguintes opções:

- Conteúdo – Nesta opção é permitido adicionar vários ecrãs à aplicação, que terão dois campos em comum: título e *icon*. No AppMachine os ecrãs designam-se blocos, remetendo, por exemplo, para os blocos de brincar que podem ser empilhados uns em cima dos outros e permitem assim fazer construções. Do mesmo modo, como ilustrado na Figura 27, os ecrãs das aplicações são blocos que organizados formam construções, isto é, aplicações. Para além dos campos comuns a todos os ecrãs, é possível definir para cada um o seu estilo (cor de texto, imagem de fundo, entre outros aspetos) e uma imagem superior, de cabeçalho. Existem cerca de 35 ecrãs disponíveis, divididos em 3 categorias (Basic, Plus e Pro) e passíveis de serem adicionados à aplicação. Alguns exemplos são: contactos, eventos, Facebook, Twitter, FAQ, galeria de fotos, pontos de interesse, RSS, URLs, video, leitor de QR Code, rádio, carrinho de compras, entre outros.

Figura 27 – Blocos no AppMachine

- Design – Permite a definição dos estilos de navegação no menu principal, da cor do texto, dos botões, do tipo de transição entre ecrãs, da imagem de fundo e do estilo de fonte. É possível alterar as imagens de todos os botões da aplicação, desde o botão "regressar" até ao botão "cancelar". Merecedor de destaque é o separador "Logic", que permite adicionar ações ao ecrã principal. As ações (que incluem efetuar chamada, partilhar no Facebook ou Twitter, enviar e-mail) podem ser despoletadas

em três estados diferentes do ecrã: *Init* (quando se inicia o ecrã), *Open* (quando está visível) e *Close* (quando se fecha o ecrã).

- Configuração – Nesta opção é possível definir uma imagem de arranque (*splash screen*), a moeda corrente a ser utilizada pela aplicação, o identificador e segredo da aplicação no Facebook, assim como dados relativos ao Twitter e Google Analysis.
- Promoção – Aqui são disponibilizados elementos úteis à promoção da aplicação, tais como: código QR, *website* promocional e suporte à partilha da aplicação através das mais variadas redes sociais.
- Publicar – Esta opção é utilizada para definir parâmetros de configuração de publicação da aplicação nas várias lojas (AppStore, Google Play e WP Store) entre os quais as imagens dos ecrãs, a categoria, a descrição, o título e o idioma.

Durante o processo de construção da aplicação é possível visualizar do lado direito do *website* uma previsão do aspeto que terá a aplicação, pormenor que pode ser altamente facilitador.

Dos criadores analisados, o AppMachine é considerado um dos mais completos, revelando-se um verdadeiro *Content Management System* (CMS) de aplicações móveis. Esta ferramenta auxilia o gestor da aplicação desde a sua criação, publicação e promoção até à sua manutenção.

2.3.3 Como AppMaker

Figura 28 – Logótipo Como

Fundada em 2010 (Como, 2014), a Como (Figura 28) conta com mais de um milhão de aplicações criadas através do seu *website*, afirmando acrescentar a este valor mais 4,500 aplicações diariamente (Anon., 2014). As aplicações geradas são do tipo *web* e podem ser colocadas em três mercados: AppStore, Google Play e Amazon. Esta ferramenta auxilia o utilizador na publicação nos vários mercados, bem como na manutenção e promoção da aplicação.

O passo inicial para a criação de uma aplicação é semelhante ao do AppMachine, uma vez que é também requerida a introdução do nome da aplicação, a categoria (música, eventos, negócios, entre outras) e o URL de uma página do Facebook ou de um *website*.

A construção da aplicação encontra-se dividida em dois separadores: "Páginas da Aplicação" e "Estilos e Navegação". No separador "Estilos e Navegação" é possível definir um tema, o *layout* de navegação entre ecrãs, o esquema de cores, imagem de fundo e ícone da aplicação. O separador "Páginas da Aplicação" é utilizado para efetuar a gestão dos ecrãs que a aplicação irá conter. Encontram-se disponíveis 27 ecrãs, divididos em quatro categorias: Contact, Monetization, Social e Media. Na categoria Contact constam os ecrãs mapa, acerca e envio de e-mail. Na categoria Monetization as opções passam por ecrã de *links*, página *web*, cupões de desconto, catálogo de produtos, entre outros. A categoria Social oferece a integração de Facebook, Twitter, eventos e RSS, enquanto da categoria Media fazem parte ecrãs como álbum de fotografias, vídeo e áudio.

Durante o processo de construção é possível visualizar como a aplicação irá aparecer em *smartphone* e *tablet* Android, iPhone 4 e 5, iPad e ainda Amazon Kindle Fire HD.

2.3.4 Mobile Roadie

Figura 29 – Logótipo Mobile Roadie

O Mobile Roadie (Figura 29) foi lançado em Março de 2009 (MobileRoadie, 2014) e desde cedo na sua existência pôde contar com as aplicações móveis de artistas famosos, como Taylor Swift e Katy Perry (Kim, 2011), o que contribuiu para a sua visibilidade no mercado. Em Setembro de 2011, as aplicações criadas através desta plataforma já contavam com um total de mais de 10 milhões de *downloads*, em 56 países.

O *dashboard* desta plataforma divide-se em cinco áreas:

- Construir – Permite adicionar os mais variados ecrãs, como notícias, galeria de vídeos ou fotos, eventos, acerca, twitter, *links*, entre outros. Existem ecrãs especiais, como a

incorporação de uma página *web* ou de um ecrã de votações, que apenas estão disponíveis para assinantes do plano Pro. Para cada ecrã é possível personalizar a sua aparência. No ecrã de notícias, por exemplo, o conteúdo pode ter várias fontes, nomeadamente: *website* do Mobile Roadie (a plataforma oferece a possibilidade de publicar notícias), RSS, Twitter ou Google News.

- Design – Nesta área são definidos os estilos da aplicação, nomeadamente da página principal, de navegação, de cores e de ordem dos ecrãs.
- Gerir – Utilizada para introduzir alguns dados importantes como as credenciais da App Store e do Google Play, com o objectivo de que a publicação da aplicação nos diferentes mercados seja possível. A ativação ou desativação de anúncios também é feita nesta área.
- Participar – Possibilita gerir alguns dos conteúdos da aplicação, tais como envio notificações, utilização de ferramentas promocionais, ou envio de mensagens delimitado pela localização do utilizador.
- Análise – Nesta área encontram-se informações relativas aos *downloads* da aplicação, à sua rentabilização, utilizadores, entre outros aspetos.

Comparativamente às outras plataformas, o Mobile Roadie destaca-se por permitir que a aplicação exporte ou importe dados vindos de *websites* criados através do Blogger, Wordpress, Joomla! ou Drupal. Além da publicação nos mercados, também é possível exportar a aplicação criada para dispositivos móveis no formato de *website*.

A aplicação Mobile Roadie Connect, disponível para iOS e Android, permite a pré-visualização das aplicações de um determinado utilizador, mediante autenticação do mesmo.

3 MobileAppBuilder

Depois de efetuado o estudo do Estado da Arte, será abordada a análise do problema e o processo de implementação da solução proposta. Da análise fazem parte o levantamento de requisitos (funcionais e não funcionais), o desenho da arquitetura do sistema proposto e o modelo de dados que o irá suportar. Na implementação serão abordados os detalhes do desenvolvimento da solução proposta dos projetos Android, iOS e do *website*.

3.1 Análise

Na presente secção são identificados os requisitos necessários para a correta utilização por parte dos utilizadores da plataforma a desenvolver, requisitos esses que vão guiar o processo de desenvolvimento da mesma e possibilitar que fique totalmente de acordo com as necessidades dos utilizadores.

Assim, será efetuada uma síntese dos requisitos necessários para o desenvolvimento da plataforma, com o objetivo de demonstrar ao leitor a estrutura lógica da solução a implementar.

3.1.1 Levantamento de Requisitos

O levantamento de requisitos abrange o processo de recolha de toda a informação necessária para a definição das funcionalidades que o *software* deve ter, do ambiente interativo desejado e do âmbito pretendido para a gestão de aplicações.

3.1.1.1 Requisitos Funcionais

Após a realização do estudo do Estado da Arte e a familiarização com a temática da criação de aplicações nativas e respetivos conceitos, procedeu-se à identificação das funcionalidades que deveriam ser desenvolvidas para a construção da plataforma, as quais se encontram listadas de seguida:

Adicionar ecrãs

A plataforma deverá suportar a criação de aplicações que possam conter os seguintes ecrãs:

- Eventos - Ecrã que permite a visualização de eventos cujo conteúdo seja obtido através do acesso a um *webservice*. Cada evento deve ter um título, data e local (composto por morada, cidade, entre outros dados).
- Contactos - Ecrã que contém os contactos básicos de uma entidade: e-mail, telefone, *website* e morada.
- *Web* – Permite, através da aplicação, a visualização e navegação numa página *web*.
- Links - Listagem de *links* com interesse a que o utilizador pode aceder. Cada *link* deverá ter um título e uma descrição.
- Twitter - Ecrã com os mais recentes *tweets* de um determinado utilizador. O objetivo deste ecrã é permitir a integração da plataforma com uma rede social.
- Info - Ecrã com uma imagem e texto de tema livre, isto é, funciona como um ecrã de informação adicional que é totalmente personalizável pelo utilizador, podendo adquirir diversas funções (por exemplo um ecrã de "acerca").
- FAQ - Listagem das questões efetuadas com mais frequência e respetivas respostas, com a finalidade de esclarecer os utilizadores.
- RSS – Ecrã com a listagem dos itens RSS de um determinado *website*. A cada item deve estar associado um título, descrição, data e URL, de modo a que seja possível a sua leitura integral quando clicado.

- Partilha de foto - Ecrã que permite ao utilizador tirar uma fotografia e proceder posteriormente ao envio da mesma para um servidor *web*. O principal foco deste ecrã é tirar partido das funcionalidades que o *hardware* dos dispositivos móveis oferece.

Todos os ecrãs disponibilizados deverão ter em comum os atributos título, subtítulo e ícone, cujo preenchimento deverá ser obrigatório.

Suporte à personalização dos estilos

A plataforma deverá suportar a personalização dos estilos da aplicação, como por exemplo: o ícone da aplicação, o estilo do menu principal, a imagem ou cor de fundo, a cor dos diferentes tipos de texto (normal, títulos, entre outros) e a cor dos botões.

Configurar publicidade

Esta funcionalidade possibilitará ao utilizador definir se a aplicação irá ter publicidade em iOS e/ou Android. A publicidade para iOS utilizará a *framework* iAd e para Android o serviço AdMob, cujo identificador de unidade – AdUnitID - é necessário para a sua utilização.

Produzir aplicação iOS e Android

A partir da indicação do utilizador para produzir/compilar o projeto criado através da plataforma, as aplicações nativas para iOS e Android são geradas. Pelo facto de o principal foco do presente trabalho ser o Android, a aplicação deverá estar pronta para ser colocada na Google Play. O resultado final deste processo deverá ser um ficheiro APK e/ou um IPA, para Android e iOS respetivamente.

3.1.1.2 Requisitos não Funcionais

Os requisitos não funcionais em geral relacionam-se com padrões de qualidade, como o desempenho, a usabilidade, a segurança. São requisitos mínimos para um *software* que se pretende que seja de qualidade.

Desempenho

Qualquer operação efetuada na plataforma deverá ser processada com eficiência, pois o utilizador pretende utilizar uma ferramenta que responda rapidamente às suas necessidades e que não seja morosa a executar as suas tarefas.

Segurança

Os acessos à plataforma apenas serão possíveis mediante a introdução de um nome de utilizador e palavra-passe. As palavras-passe e outros dados sensíveis deverão encontrar-se encriptados na base de dados.

Usabilidade

A plataforma a ser desenvolvida deverá possuir uma interface intuitiva e agradável para o utilizador, de forma a garantir ao mesmo um fácil acesso e uso das funcionalidades desenvolvidas, bem como ter compatibilidade com os principais *browsers* utilizados no mercado.

Implementação

A plataforma deverá ser desenvolvida sobre o ambiente LAMP (*Linux, Apache, MySQL and PHP*), isto é, utilizando o sistema operativo Linux, um servidor Apache, um sistema de gestão de base de dados MySQL e a linguagem de programação PHP. No desenvolvimento do *website* deverá recorrer-se ao *framework* PHP Symfony2 e a interface (HTML/CSS) deverá seguir as normas da W3C (*World Wide Web Consortium*), uma organização que tem a finalidade de estabelecer padrões para a criação e interpretação de conteúdos para a *web*.

Hardware e Software

A plataforma deverá encontrar-se alojada numa máquina com o sistema operativo Mac OS X e Xcode instalado, para permitir a compilação da aplicação iOS.

Padrão de desenvolvimento

O código fonte do *website*, aplicações Android e iOS deverá ser desenvolvido seguindo o padrão *Model-View-Controller* (MVC). O MVC é um modelo de arquitetura de *software* composto por três camadas:

- *Model* – Esta camada é independente da interface apresentada ao utilizador e é responsável pela representação dos dados, oferecendo meios de acesso (leitura e escrita) aos mesmos.
- *Controller* – Processa e responde a eventos, geralmente ações do utilizador, e interage diretamente com o *Model* para satisfazer essas ações. A validação e

filtragem de dados introduzidos pelo utilizador são realizadas no *Controller*. Esta camada não é responsável pela obtenção dos dados (responsabilidade do *Model*) nem pela sua exibição (responsabilidade da *View*), serve para controlar as outras duas camadas como um todo.

- *View* – Responsável pela exibição de dados, é com esta camada que o utilizador interage.

Este padrão é importante, pois promove a reutilização de código, facilita a sua manutenção e permite manter o código organizado/limpo.

3.1.1.3 Conclusão do Levantamento de Requisitos

Com base no estudo do Estado da Arte e nas funcionalidades identificadas no levantamento de requisitos que devem ser desenvolvidas para a construção da plataforma, é possível concluir que nenhuma das ferramentas estudadas satisfaz o problema na totalidade, pois nenhuma permite a criação de aplicações nativas tanto para Android como para iOS.

3.1.2 Sistema Proposto

Na presente secção é descrita a arquitetura do sistema proposto e o modelo de dados para o desenvolvimento da plataforma, com base no levantamento dos requisitos e no estudo de conceitos relacionados com o tema em estudo.

3.1.2.1 Arquitetura

De seguida é apresentada a arquitetura do sistema proposto, sendo abordadas em detalhe algumas das suas caraterísticas. A arquitetura encontra-se dividida em cinco componentes (Figura 30), com o *website* a funcionar como ponto central, pois é através do mesmo que o utilizador irá realizar as tarefas identificadas no levantamento de requisitos.

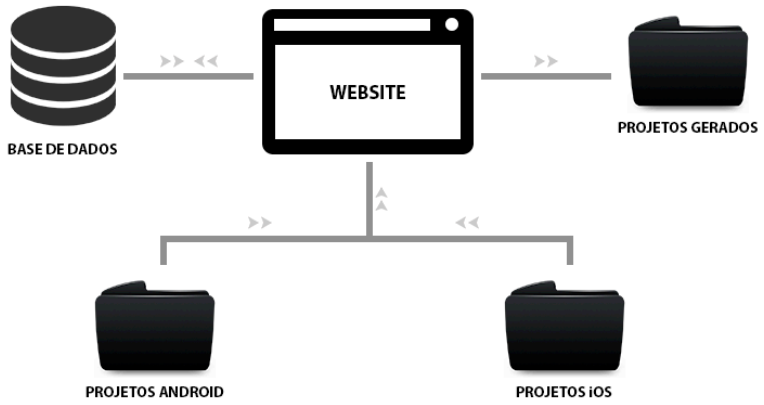


Figura 30 – Arquitetura do Sistema Proposto

De seguida são abordados os componentes que fazem parte da arquitetura do sistema proposto.

Website

O *website*, que funcionará como componente central do criador automático de aplicações Android e iOS, será desenvolvido através do Symfony (um *framework* PHP), sendo utilizada uma base de dados para suportar a gestão das aplicações.

Para o utilizador gerir uma aplicação necessita de criar e editar os conteúdos da mesma, o que passa por adicionar e editar ecrãs que deseja que constem na aplicação e editar os seus estilos gerais, desde a definição da cor do texto ao upload de imagens de fundo.

Através do *website* é possível que o utilizador defina a existência ou não de publicidade na sua aplicação e ordene a execução do processo de compilação da aplicação criada para Android e/ou iOS.

Base de Dados

Uma vez que terá de ser utilizado o Symphony para o desenvolvimento do *website*, a base de dados será gerada através de uma biblioteca denominada Doctrine, que permite a criação da mesma através de objectos/classes PHP. Mais detalhes sobre a implementação da base de dados serão abordados numa fase mais avançada deste documento. Na base de dados irão constar dados que permitem:

- Saber que módulos se encontram disponíveis para o utilizador adicionar à sua aplicação;
- Conhecer os dados de um módulo de uma determinada aplicação;
- Identificar os recursos (imagens, ficheiros de texto) de um determinado módulo;
- Facilitar o processo de compilação realizado pelo *website*, contendo entre outros dados, o caminho físico relativo a um módulo de um projeto Android ou o nome da Activity principal desse módulo.

Projetos Android

O componente projetos Android resume-se a um diretório onde constam vários projetos Android criados através do Android Developer Tools, um *plugin* para o Eclipse.

Na Figura 31 está ilustrado o diagrama dos projetos Android, com os diferentes projetos criados para a geração de aplicações Android.

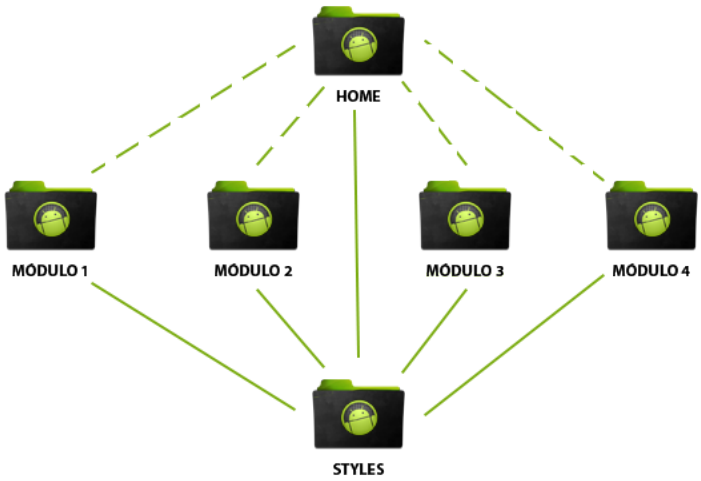


Figura 31 – Diagrama Projetos Android

O projeto Home é o único do tipo aplicação, todos os outros são definidos como bibliotecas. Este projeto contém o ecrã principal da aplicação, que permite ao utilizador navegar entre os módulos que a aplicação tem disponíveis.

A biblioteca Styles será adicionada a todos os projetos e irá conter dados relativos aos estilos gerais da aplicação.

As bibliotecas correspondentes aos módulos apenas serão adicionadas ao projeto HOME em tempo de compilação, o que permitirá que a aplicação gerada possa apenas conter os módulos que irá utilizar. Cada uma destas bibliotecas irá conter ficheiros relativos ao funcionamento da aplicação, quer sejam ecrãs, recursos ou código-fonte.

Projeto iOS

Uma vez que o foco principal do presente trabalho é o desenvolvimento de aplicações Android, para os projetos iOS será adoptada uma estratégia diferente, sendo desenvolvida apenas uma aplicação. Embora a mesma incorpore todos os módulos, apenas serão disponibilizados na aplicação os que farão parte da mesma.

Projetos Gerados

O componente projetos gerados funciona como diretório e é utilizado no processo de compilação das aplicações, sendo copiados para o mesmo os projetos Android e iOS que irão dar origem à aplicação pretendida. Em cada aplicação gerada irá constar o código fonte utilizado, um ficheiro onde serão registadas informações relativas a cada passo do processo de compilação para uma depuração e o resultado final (ficheiro .apk ou .ipa).

3.1.2.2 Modelo de Dados

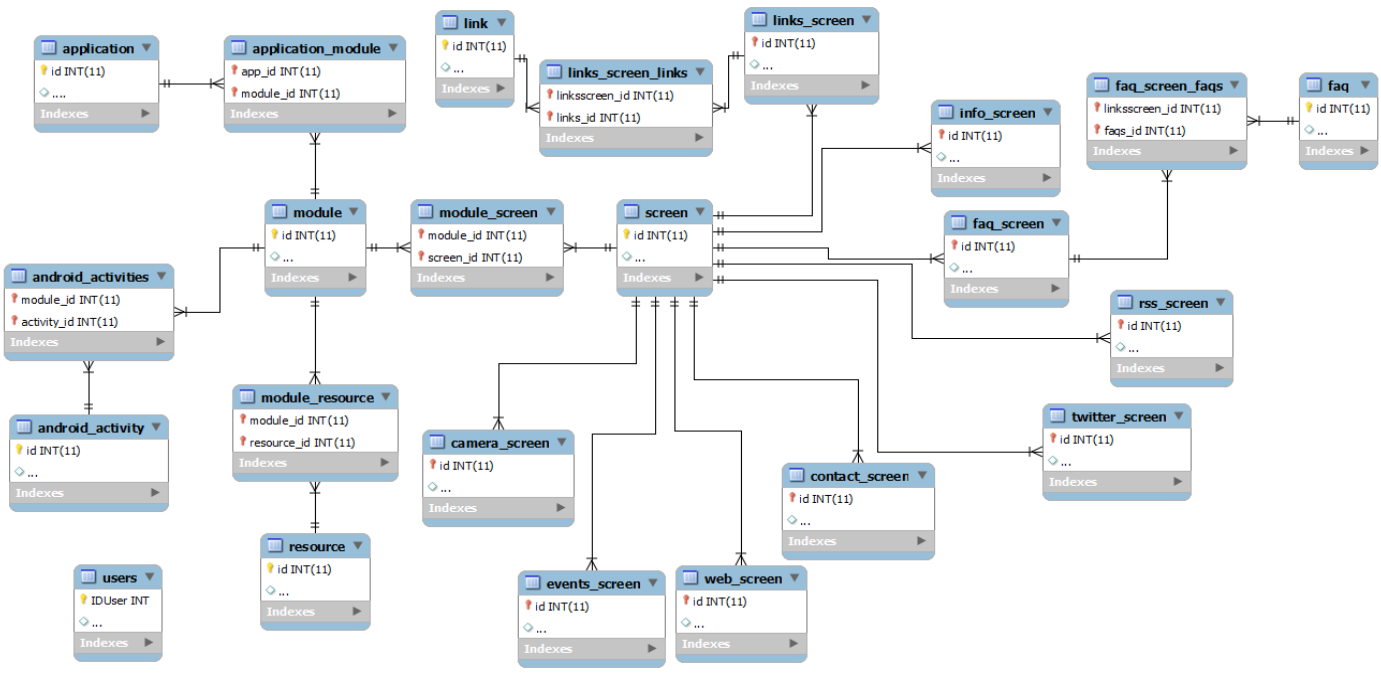


Figura 32 – Tabelas do Modelo de Dados do *Website*

De modo a cumprir os objetivos para os quais foi desenhado, um sistema informático necessita de possuir informação, que utiliza nas suas tarefas. Por este motivo e de modo a conseguir criar um sistema funcional, foi elaborado um modelo de dados para a plataforma a desenvolver, com base nos requisitos identificados.

Na Figura 32 está ilustrado o modelo de dados, que é explicado com maior detalhe nos parágrafos seguintes e justificada a existência de algumas tabelas e campos que fazem parte do mesmo.

Aplicação

A tabela "application" (Figura 33) representa uma aplicação criada através do *website*. Nesta tabela são armazenados os dados relativos aos estilos (`Cfg_LayoutType`, `Cfg_TextColor`, Icon, entre outros), o identificador da aplicação em Android (`AndroidPackageName`) e o nome da aplicação (`AppName`). O campo `AppNameSlug` irá conter o nome da aplicação sem espaços em branco. Os campos `MonetizationAndroidState`, `MonetizationAndroidAdUnitID` e `MonetizationIosState` definem parâmetros relacionados com a rentabilização da aplicação em termos de publicidade.

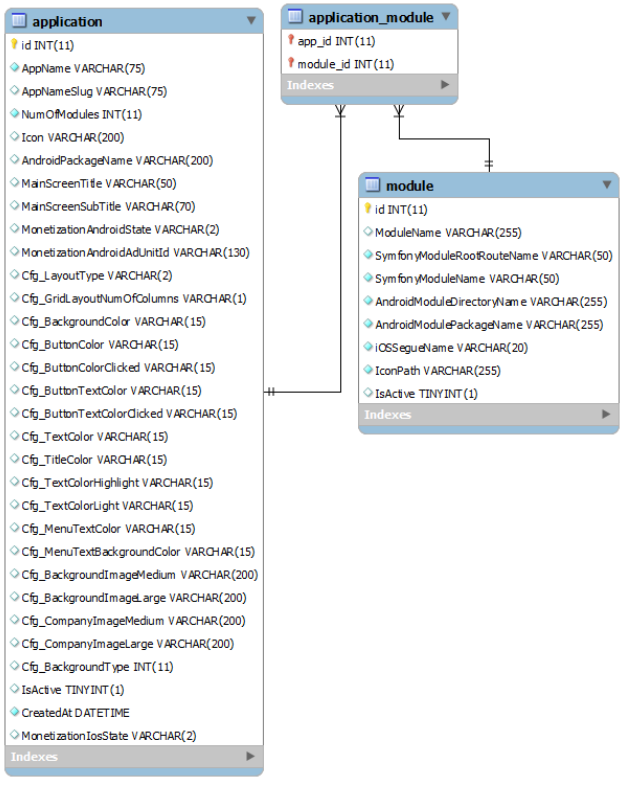


Figura 33 – Relação entre tabela aplicação e módulo

Módulos

Uma aplicação pode conter um ou mais módulos, sendo que cada módulo se encontra diretamente relacionado com um ou mais ecrãs, Activities Android ou recursos.

Para cada módulo (Figura 34) está definido um nome, o nome *root* atribuído em Symfony, o nome do *segue* utilizado em iOS e os dados relativos a Android, nomeadamente o *package name* e a localização do diretório onde se encontra o projeto. Uma vez que o ficheiro AndroidManifest necessitará de declarar todas as Activities que utiliza, foi criada uma relação entre o módulo e a respetiva Activity. Os módulos irão conter diferentes recursos, tais como imagens e ficheiros de configuração, sendo os mesmo do tipo .properties para Android e .plist para iOS.

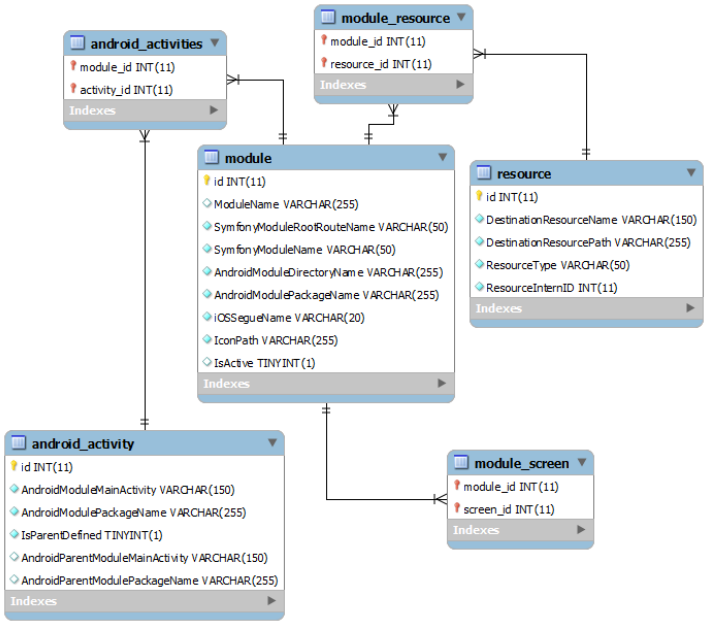


Figura 34 – Tabela módulo e relação com ecrã, Activity e recurso.

Ecrãs

Cada registo da tabela "screen" (Figura 35 – Tabela ecrã) irá representar um ecrã, tendo de ser definido para cada um o título, subtítulo e ícone.

De seguida estão descritos os ecrãs passíveis de serem adicionados a

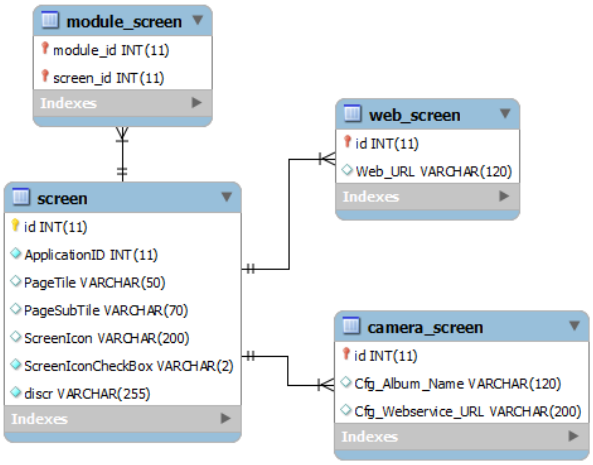


Figura 35 – Tabela ecrã

uma aplicação:

- Eventos - Para o ecrã de eventos é necessário ter definido o endereço eletrónico do *webservice* que irá ser utilizado e o número de eventos que irão aparecer no ecrã. É também essencial definir o identificador do *webservice* (`EventContentSourceType`), de modo a ser possível utilizar mais do que um *webservice* diferente, com origem no EventBrite ou no LastFM, por exemplo.
- Contactos - Desta tabela fazem parte os campos relativos aos dados de contactos de uma entidade, tais como: morada, nome da empresa, endereço do *website*, e-mail, telefone, longitude e latitude. Em Android, para apresentar um mapa utilizando a longitude e latitude será necessário armazenar a API *key* que será utilizada para recorrer ao Google Maps.
- *Web* - Ao ecrã *web* deverá ser atribuído um título, subtítulo e ícone, que serão armazenados na tabela "screens", bem como o endereço URL do *website* a apresentar.
- Links - Este ecrã irá conter uma listagem de links, sendo estabelecida uma relação com a tabela "link" para ter acesso aos mesmos, da qual fazem parte os campos que caraterizem um link (nome, URL e descrição). A tabela "links_screen" será composta por um título e uma imagem, que poderá encontrar-se definida em dois tamanhos: médio (`Cfg_HeaderImageMedium`) e grande (`Cfg_HeaderImageLarge`).
- Twitter - A tabela "twitter_screen" irá conter quatro campos: o nome do utilizador da conta a partir da qual serão obtidos os *tweets* (`TwitterScreenName`), o número máximo de *tweets* a ser carregado, a API *key* e a API *secret*, ambas necessárias para obter os *tweets* de um utilizador.
- Info - Para criar um ecrã info será necessário armazenar dados relativos a uma imagem e ao conteúdo HTML, isto é, o texto a ser visualizado.
- FAQ - À semelhança do ecrã links, o ecrã FAQ será composto por uma listagem de *faqs – Frequently Asked Questions*. Para o efeito, será criada uma relação com a tabela "faq", da qual fazem parte os campos pergunta (`Faq_Question`) e resposta (`Faq_Answer`). A tabela "faq_screen" terá na sua composição uma imagem de tamanho médio (`Cfg_HeaderImageMedium`) e outra de tamanho grande (`Cfg_HeaderImageLarge`), utilizadas para o cabeçalho do ecrã.
- RSS - A tabela "rss_screen" será composta pelos seguintes campos:

- `RSS_ContentURL` – URL do *webservice* a partir do qual serão carregados os itens RSS;
- `RSS_MaxNumOfItems` – o número máximo de itens RSS a serem visualizados no ecrã;
- `RSS_LayoutType` – o tipo de *layout* a ser carregado para o utilizador, uma vez que a intenção será disponibilizar mais do que um *layout*;
- `RSS_HighlightItemTitleTextColor` – cor do texto do título de um item RSS;
- `RSS_HighlightItemTitleBGColor` – cor de fundo do título de um item RSS.

- Partilha de foto - Para o ecrã de partilha de foto deverão ser armazenados os dados de dois campos: nome do álbum onde serão armazenadas as fotos no dispositivo e endereço do *webservice* para onde será feito o upload da fotografia captada.

3.1.3 Conclusão da Análise

A análise efetuada no âmbito do presente estudo foi dividida em duas fases. A primeira, o levantamento de requisitos, serviu para identificar as funcionalidades que a plataforma tem que conter, de forma a satisfazer todos os critérios exigidos.

A segunda fase consistiu no desenho de um sistema proposto, com base nos requisitos funcionais e não funcionais identificados, no estudo das arquiteturas dos sistemas operativos em questão e da análise aos diversos criadores automáticos de aplicações móveis existentes na atualidade. O desenho da arquitetura e do modelo de dados revelou-se muito importante para o desenvolvimento da plataforma, pois permitiu a definição dos objetivos a atingir na fase seguinte.

3.2 Implementação

Terminada a fase de análise, procedeu-se à implementação de todos os componentes identificados na subsecção Arquitetura do Sistema Proposto. Ao longo do presente sub-capítulo é abordada a fase de desenvolvimento da plataforma e dos projetos Android e iOS, sendo apresentadas e descritas as principais funcionalidades e indicada a forma como foram implementadas.

3.2.1 Tecnologias Utilizadas

Durante o desenvolvimento do projeto foram utilizadas diversas tecnologias no sentido de criar uma solução capaz de resolver o problema proposto, estando enumeradas de seguida:

- Ambientes de desenvolvimento: Xcode e Android Developer Tools (eclipse);
- Linguagens de programação: PHP, Objective-C e Java;
- Sistema de Gestão de Base de Dados: MySQL;
- Sistema Operativo: Mac OS X v10.9.2;
- Outras tecnologias e ferramentas: Symfony2, HTML, CSS, JavaScript, jQuery, phpMyAdmin, XML, JSON (*JavaScript Object Notation*) e Bootstrap.

A utilização destas tecnologias em particular deve-se à necessidade de desenvolver a plataforma em ambiente LAMP, e à preferência dada ao uso de ferramentas livres.

3.2.2 Android

Nesta subsecção são abordados os detalhes de implementação dos projetos Android e alguns conceitos relacionados. Os projetos serão desenvolvidos utilizando a API 16, o que significa que as aplicações serão compiladas para a versão 4.1.x (Jelly Bean) do Android, ou superior. A versão das *build tools* utilizadas nos projetos criados foi a 18.1.1, cujo lançamento ocorreu em Setembro de 2013.

3.2.2.1 Conceitos Relacionados

De forma a proporcionar ao leitor uma melhor compreensão dos passos tomados para pôr em prática a solução para o problema colocado, encontram-se explicados alguns dos conceitos relacionados com a implementação dos projetos Android.

Build Tools

As *build tools* são ferramentas que permitem a criação de rotinas, isto é, tarefas repetitivas, como por exemplo: compilar código fonte, executar testes ou criar documentação relativa ao *software*. Geralmente as *build tools* não são executadas em ambiente gráfico, ou seja, estão apenas acessíveis via linha de comandos.

Em Android, o desenvolvimento e compilação do projeto são feitos através do Eclipse, que não permite a execução das seguintes tarefas (Quigley, 2010):

- Adicionar passos de compilação personalizados;
- Utilizar um sistema automatizado de compilação;
- Utilizar configurações de compilação (*build configurations*);
- Criar o projeto final através de um comando (*release*).

Tipicamente, o processo de compilação de um projeto em Java inclui a compilação do código para Java *bytecode*, a criação de um ficheiro .jar para distribuição e a criação de documentação Javadoc. Exemplos de *build tools* são o Apache Ant, o Apache Maven e o Gradle.

ANT

O Ant (Another Neat Tool) é uma *build tool* que utiliza XML para descrever o processo de compilação (por omissão o nome script de compilação tem o nome de build.xml). Esta ferramenta é significativamente conhecida no meio dos programadores de Java, e permite a criação de aplicações Android sem utilizar IDE's de desenvolvimento, como por exemplo o Eclipse (Quigley, 2010).

No Ant os passos de compilação designam-se tarefas, que se encontram definidas dentro de *targets*, sendo os mais conhecidos designados o *debug* e o *release*. Num projeto Android, após a execução de comandos como o ant debug ou ant release, é criado um ficheiro .apk na pasta bin/ do projeto. Esta ferramenta permite (Vogel, 2013):

- Criar o pacote final (*release package*) pronto para ser colocado no mercado;
- Criar configurações de compilação em ficheiros de propriedades;
- Criar um pacote final com apenas um comando;
- Adicionar um ficheiro build.xml a um projeto Android já existente, através do comando android update project --path “caminho_do_projeto”;
- Editar os ficheiros de código fonte com base nas configurações.

Relativamente ao último ponto referido, suponhamos que no nosso projeto existe uma classe de configuração onde estão definidas algumas variáveis, como por exemplo uma *debugging*

flag (true|false), a mesma terá diferentes valores consoante a aplicação seja compilada para o *target debug* ou para o *release*. Assim, na versão *release* não faz sentido a aplicação fazer *log* para a consola, no entanto, na versão *debug* já faz (Quigley, 2010).

Para criar um novo projeto Android com o ficheiro build.xml, é necessário executar o seguinte comando:

```
android create project --name YourProjectName --path C:\dev\YourProject --
target android-16 --package com.company.testproject --activity MainActivity
```

Comando 1 – Comando para criar um projeto Android

Este comando permite definir um nome para o projeto, o caminho para o diretório no qual serão criados os seus ficheiros, a versão do Android para o qual a aplicação será desenvolvida, o *package name* e ainda o nome da Activity principal.

Todas as aplicações Android que são executadas num dispositivo físico precisam de ser assinadas. Quando desenvolvido no Eclipse, um projeto Android é assinado com uma chave de *debug*. Contudo, quando o objetivo é colocar a aplicação no mercado é necessário assinar a aplicação com uma chave real. Este procedimento pode ser facilmente executado através do Ant.

Proguard

O ProGuard é uma ferramenta utilizada para otimizar e ofuscar código fonte através da remoção de porções de código não utilizadas e/ou através da alteração do nome de classes, atributos e métodos (DeveloperAndroid-1, 2014).

Em Android, a sua utilização resulta num ficheiro .apk de pequena dimensão, tornando o processo de *reverse engineering* da aplicação mais trabalhoso e pouco apetecível. Esta ferramenta encontra-se integrada no ambiente de desenvolvimento para Android e é executada apenas quando o código é compilado no modo *release* (seja pelo Ant ou pelo Eclipse) (Rafael, 2012).

Quando é criado um novo projeto de Android, é colocado automaticamente na pasta do projeto um ficheiro proguard.cfg, que contém as definições do modo como esta ferramenta irá otimizar e ofuscar o código fonte. Para ativar o ProGuard é necessário definir a localização

relativa ou absoluta do proguard.cfg no ficheiro project.properties, presente na pasta do projeto (DeveloperAndroid-1, 2014):

```
proguard.config = proguard.cfg;
```

Código Fonte 2 – Configuração do ProGuard

Embora exista a possibilidade de usar as definições por omissão do ProGuard, estas devem ser determinadas para cada projeto, uma vez que esta ferramenta pode em alguns casos eliminar código fonte necessário. Após a sua execução são criados quatro ficheiros de texto (DeveloperAndroid-1, 2014):

- dump.txt - Descrição da estrutura interna de todas as classes do ficheiro .apk;
- mapping.txt - Listagem do mapeamento feito entre as classes originais e ofuscadas;
- seeds.txt - Listagem de todas as classes que não foram ofuscadas;
- usage.txt - Lista o código do .apk que foi modificado.

Google Play Services

O Google Play Services é uma biblioteca que apresenta um conjunto de funcionalidades que permitem melhorar a experiência na utilização de aplicações Android (DeveloperAndroid-2, 2014). Este componente torna possível a realização de processos fundamentais, como a autenticação nos serviços Google, sincronização dos contactos, incorporação de mapas, compras na aplicação (*in-app purchase*), utilização do Google+ e do AdMob para publicidade, entre outros serviços.

3.2.2.2 Biblioteca de Estilos

A biblioteca de estilos será um constituinte de todos os projetos desenvolvidos para Android, tendo como função armazenar diversos dados, tais como a cor de texto e imagem ou cor de fundo. Para o efeito foi criada a classe `Styles_config` (localizada em uebSTYLES_lib/src/net/filipematos/uebstyles_lib/), que conta apenas com as seguintes declarações de *Strings*:

```
public class Styles_config{
  public static final String BACKGROUND_IMAGE_NAME = "bg_image";
  public static final String BACKGROUND_COLOR = "#F0F0F0";
  public static final String TEXT_COLOR = "#000000";
  public static final String TEXT_COLOR_LIGHT = "#CCCCCC";
  public static final String TITLE_COLOR = "#33C4E8";
                      ...
  public static final String AD_UNIT_ID="ca-app-pub-
5057887816328772/6984244444";
  public static final String IS_MONETIZATION_ON="1";
}
```

Código Fonte 3 – Conteúdo do ficheiro Styles_config.java localizado no diretório uebSTYLES/src/…

Os valores das variáveis existentes nesta classe são atualizados em tempo de compilação, através da chamada de um *target* Ant. Nesse sentido foi criado no diretório principal do projeto o ficheiro build.xml, que contém instruções a serem executadas pelo Ant e os ficheiros Styles_config.java e styles_config.properties, localizados dentro do diretório uebSTYLES_lib/styles_config/.

```
CONFIG.BACKGROUND_IMAGE_NAME=n/a
CONFIG.BACKGROUND_COLOR=#000000
CONFIG.TEXT_COLOR=#FFFFFF
CONFIG.TEXT_COLOR_LIGHT=#CCCCCC
CONFIG.TITLE_COLOR=#05D0C1
...
CONFIG.AD_UNIT_ID=ca-app-pub-5057887816328772/6984244444
CONFIG.IS_MONETIZATION_ON=1
```

Código Fonte 4 – Excerto do código do ficheiro styles_config.properties

O ficheiro styles_config.properties é um ficheiro de propriedades, cujo conteúdo é definido pelo *website* quando o utilizador desejar obter a sua aplicação.

```
// @GENERATED@
public class Styles_config {
  public static final String BACKGROUND_IMAGE_NAME =
"@CONFIG.BACKGROUND_IMAGE_NAME@";
  public static final String BACKGROUND_COLOR =
"@CONFIG.BACKGROUND_COLOR@";
  public static final String TEXT_COLOR = "@CONFIG.TEXT_COLOR@";
  public static final String TEXT_COLOR_LIGHT =
"@CONFIG.TEXT_COLOR_LIGHT@";
  public static final String TITLE_COLOR = "@CONFIG.TITLE_COLOR@";
  ...
  public static final String AD_UNIT_ID="@CONFIG.AD_UNIT_ID@";
  public static final String
IS_MONETIZATION_ON="@CONFIG.IS_MONETIZATION_ON@";
}
```

Código Fonte 5 - Conteúdo do ficheiro Styles_config.java localizado no diretório uebSTYLES/styles_config/…

O ficheiro Java representado na Código Fonte 5 é igual ao que será utilizado em tempo de execução, diferenciando-se apenas pelo valor das suas variáveis, que consistem em *tokens* com referência para o nome das propriedades existentes no ficheiro styles_config.properties.

No ficheiro build.xml foi definido um *target* designado `buildLib`, cuja função é copiar o ficheiro uebSTYLES_lib/styles_config/Styles_config.java para o diretório uebSTYLES_lib/src/net/filipematos/uebstyles_lib/, alterando os *tokens* das variáveis pelos valores do ficheiro de propriedades.

```
<target name="copyConfigProperties"
 description="Copy the configuration file, replacing tokens in the file" >

    <copy encoding="utf-8" overwrite="true"
        file="styles_config/Styles_config.java"
        todir="${config.target.path}" >

        <filterset filtersfile="styles_config/${config.properties.file}"
/>
    </copy>

    <tstamp />

    <echo message="Copy of config properties finished. ${TSTAMP}" />
</target>
```

Código Fonte 6 – Definição do target copyConfigProperties do ficheiro build.xml

A execução do `buildLib` acaba por invocar o *target* `copyConfigProperties`, cuja função foi descrita anteriormente. A propriedade `${config.target.path}` contém o caminho do ficheiro de destino, neste caso src/net/filipematos/uebstyles_lib/, e a propriedade `${config.properties.file}` contém o nome do ficheiro de propriedades a utilizar (styles_config.properties).

Para além dos dados diretamente relacionados com os estilos de apresentação da aplicação, foram também armazenados os dados relativos à publicidade, uma vez que esta biblioteca será obrigatoriamente utilizada por todos os projetos e a inclusão ou não de publicidade na aplicação afeta a apresentação da mesma.

O acesso aos dados que a biblioteca armazena poderia ser realizado através da classe `Styles_config`, pois as variáveis são públicas. No entanto, para permitir a outras bibliotecas saber, por exemplo, se o fundo do ecrã é uma cor ou imagem (**Error! Reference source not found.**), foi criada a classe `AppStyles`.

```
public static String getBackgroundImageName(){
 return Styles_config.BACKGROUND_IMAGE_NAME;
}

public static boolean hasBackgroundImage(){
 if (Styles_config.BACKGROUND_IMAGE_NAME.contains("n/a"))
        return false;
 return true;
}

public static int getBackgroundColor(){
 return Color.parseColor(Styles_config.BACKGROUND_COLOR);
}
```

Código Fonte 7 – Excerto de código da classe AppStyles

Todos os projetos Android contêm um ou dois ficheiros de propriedades, tendo sido necessária a criação de apenas um no caso da biblioteca de estilos, o ficheiro styles_config. No entanto, com o objetivo de separar logicamente os parâmetros que cada módulo requer, foram criados dois ficheiros para as outras bibliotecas: a de configuração (que contém dados como URL ou a API *key* para obter os tweets de um determinado utilizador) e a de valores (que contém o título do ecrã, o sub-título, entre outros dados). O processo de compilação é igual ao efetuado na biblioteca de estilos, para ambos os ficheiros.

3.2.2.3 Módulos

Foram desenvolvidos para Android nove projetos do tipo biblioteca, que representam os módulos disponíveis. Estes projetos partilham um *target* com a mesma denominação (`buildLib` – *Build Library*) e o mesmo processo de compilação, já descrito anteriormente relativamente à biblioteca de estilos. De seguida serão abordados detalhes de implementação de apenas alguns dos módulos, uma vez que o seu processo de compilação é idêntico.

Publicidade

Atualmente existem inúmeros serviços que tornam possível a rentalibização da aplicação, através do acesso às suas *frameworks*. O serviço pelo qual se optou é disponibilizado e suportado pela Google, o AdMob. Encontra-se incorporado em todos os módulos, à exceção do módulo *web*, sendo necessário para a sua utilização indicar na aplicação a referência da biblioteca do Google Play Services. Este processo pode ser realizado em interface gráfica ou através da introdução no ficheiro project.properties, que se encontra na raiz de cada projeto, o seguinte conteúdo:

```
android.library.reference.2=../uebSTYLES_lib/google-play-services_lib
```

Código Fonte 8 – Adicionar a referência de uma biblioteca a um projeto Android

Adicionalmente é necessário inserir a permissão para utilização da Internet e introduzir entre os elementos `<application>` do ficheiro AndroidManifest.xml o seguinte conteúdo:

```
<activity android:name="com.google.android.gms.ads.AdActivity"
 android:configChanges="keyboard|keyboardHidden|orientation|screenLayout|ui
Mode|screenSize|smallestScreenSize"/>
```

Código Fonte 9 – Elemento <activity> no AndroidManifest que utilizará publicidade

Estas três etapas conferem a qualquer aplicação a capacidade de utilizar publicidade nos seus ecrãs. Na Activity onde é desejado incluir publicidade é necessário introduzir código em quatro estados da mesma: `onCreate()`, `onPause()`, `onResume()` e `onDestroy()`.

```
public class MainActivity_contacts extends Activity {

 private AdView mAdView;

 @Override
 protected void onCreate(Bundle savedInstanceState) {
        LinearLayout ll = (LinearLayout)
this.findViewById(R.id.main_layout);
        //Carregar ADS
        if(AppStyles.isMonetizationOn()){
               mAdView = new AdView(this);
               mAdView.setAdUnitId(AppStyles.getAdUnitId());
               mAdView.setAdSize(AdSize.BANNER);
               mAdView.loadAd(new AdRequest.Builder().build());
               LinearLayout.LayoutParams params = new
LinearLayout.LayoutParams(LinearLayout.LayoutParams.MATCH_PARENT,
LinearLayout.LayoutParams.WRAP_CONTENT);
               ll.addView(mAdView, params);
        }
 }
 @Override
 protected void onPause() {
        if(mAdView != null)
               mAdView.pause();
        super.onPause();
 }

 @Override
 protected void onResume() {
        super.onResume();
        if(mAdView != null)
               mAdView.resume();
 }

 @Override
 protected void onDestroy() {
        if(mAdView != null)
               mAdView.destroy();
        super.onDestroy();
 }
}
```

Código Fonte 10 – Implementação de publicidade no ecrã de contactos

O Código Fonte 10 demonstra a incorporação de publicidade no módulo de contactos. No método `onCreate()` é inicializado um objeto do tipo `AdView` que executa os métodos `setAdUnitID` (para identificar a conta do AdMob), `setAdSize` (com o objetivo de definir o tamanho do anúncio) e ainda `loadAd` (que utiliza uma *thread* para carregar o anúncio em *background*). Posteriormente o objeto do tipo `AdView` é adicionado ao *layout* principal do ecrã, através do método `addView`.

Biblioteca de Contactos

Na biblioteca de contactos encontram-se armazenados vários tipos de dados, tais como as coordenadas (latitude e longitude) de um local, endereços de e-mail, *websites*, entre outros. Estes dados encontram-se divididos em duas classes: `Contacts_config` e `Contacts_values`. O valor das suas variáveis é atualizado em tempo de compilação, através do recurso aos ficheiros de propriedades criados pelo *website*. Este processo é igual ao descrito anteriormente para a biblioteca de estilos. No método `onCreate()` da Activity principal deste módulo (MainActivity_contacts.java) encontram-se definidos o título e subtítulo do ecrã.

As coordenadas geográficas contidas na classe `Contacts_values` que são utilizadas para colocar um ponteiro no mapa são verificadas e, caso não sejam válidas, são alteradas para as coordenadas de localização do ISEP, que foi definida como referência geográfica.

Para fornecer ao utilizador um mapa é necessário utilizar a biblioteca do Google Play Services e adicionar ao AndroidManifest.xml dentro do elemento `<manifest>` o seguinte conteúdo:

```
<uses-feature
  android:glEsVersion="0x00020000"
  android:required="true" />
<uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"
/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"
/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Código Fonte 11 – Permissões necessárias para a utilização do mapa

O primeiro elemento declara a necessidade de utilizar OpenGL para a visualização do mapa. Os restantes elementos declaram permissões de acesso ao GPS. Dentro do `<application>` é ainda necessário adicionar os elementos:

```
<meta-data
 android:name="com.google.android.gms.version"
        android:value="@integer/google_play_services_version" />
<meta-data
 android:name="com.google.android.maps.v2.API_KEY"
 android:value="AIzaSyCG5BdLXNW6FANAIkmWX37HiA2Q4bclIdM"/>
```

Código Fonte 12 – Declaração no AndroidManifest de elementos necessários à utilização de mapa

O primeiro elemento é utilizado para indicar a versão do Google Play Services e o segundo contém a referência da API *key*, utilizada para permitir o acesso ao Google Maps. Após configurados todos estes detalhes, segue-se a implementação do mapa, que começa por adicionar ao *layout* o elemento Fragment da classe `MapFragment`.

```
<fragment android:id="@+id/map"
 android:layout_width="match_parent"
    android:layout_height="300dp"
    class="com.google.android.gms.maps.MapFragment" />
```

Código Fonte 13 – Declaração do elemento <fragment> no AndroidManifest

É através deste elemento que o mapa será carregado e visualizado pelo utilizador no ecrã. No método `onCreate()` da Activity `MainActivity_contacts` encontra-se o excerto de Código Fonte 14.

```
map = ((MapFragment)
getFragmentManager().findFragmentById(R.id.map)).getMap();
if (map!=null){
 Marker companyMarker = map.addMarker(new MarkerOptions()
        .position(LocationLatLng)
        .title(Contacts_values.CompanyName)
        .snippet(Contacts_values.Address)
        );

 CameraUpdate yourLocation =
CameraUpdateFactory.newLatLngZoom(LocationLatLng, 11.0f);
 map.animateCamera(yourLocation);
}
```

Código Fonte 14 – Conteúdo do método onCreate da classe MainActivity_contacts

A primeira instrução tem a função de carregar para o objeto `map` (do tipo `GoogleMap`) o Fragment do mapa definido anteriormente no *layout*. De seguida são definidas a localização de um ponteiro cujas coordenadas se encontram no objeto `LocationLatLng` (do tipo `LatLng`) e a posição e *zoom* do mapa, através do método `newLatLngZoom`.

Biblioteca de Eventos

Na biblioteca de eventos encontram-se armazenados dados como o título e subtítulo da Activity principal e o título e subtítulo da Activity de um evento na classe `Events_values`. Já na classe `Events_config`, encontram-se as seguintes variáveis:

- `CONTENT_SOURCE` – identificador do *webservice* que será consumido, uma vez que nesta biblioteca será permitido carregar eventos de três serviços diferentes (EventBrite, artista da LastFM ou local da LastFM);
- `CONTENT_URL` – o endereço do *webservice*;
- `MAX_NUM_OF_EVENTS` – o número máximo de eventos apresentados ao utilizador.

Uma vez que esta biblioteca irá consumir dados de um *webservice*, foi necessário adicionar permissões de acesso ao estado da rede e uso da Internet. Do modelo de dados da biblioteca fazem parte as classes Event e Venue, utilizadas para armazenar os dados que serão apresentados ao utilizador.

O facto de a biblioteca ser capaz de consumir dados de três fontes diferentes obrigou à declaração de uma interface (`WSParser`) e à implementação de três *parsers* (`WSParserEventBrite`, `WSParserLastFMArtist` e `WSParserLastFMVenue`).

Em qualquer dos casos as comunicações com o serviço e o tratamento dos dados são realizados em *background*, recorrendo-se à *inner* classe `SitesDownloadTask` implementada na Activity `MainActivity_events`.

O processo de preenchimento da listagem de eventos é idêntico ao da biblioteca de *links*. A ação é desencadeada quando o utilizador pressiona um item da lista, sendo aberta na biblioteca uma nova Activity, onde o utilizador pode encontrar informações detalhadas relativas ao evento selecionado. A Activity `SingleEvent` permite visitar o URL do evento ou do local, se o utilizador assim o desejar. Quando o evento tem uma imagem associada, esta é carregada para o *layout* desta Activity.

```
eventIMG = (ImageView) findViewById(R.id.eventImageView);
if(checkIfNullOrEmpty(event.getEventImageURL())){
 eventIMG.setImageResource(R.drawable.default_event_img);
}else{
 eventIMG.setTag(event.getEventURL());
 new ImageDownloader().execute(event.getEventImageURL());
}
```

Código Fonte 15 – Excerto de código do método onCreate da Activity SingleEvent

O elemento `<ImageView>` do *layout* do ecrã do evento é carregado para o objeto `eventIMG`. Caso a variável `eventImageURL` do evento (objeto `Event`) seja vazia, é colocada uma imagem por omissão. Caso contrário é utilizada a *inner* classe `ImageDownloader` para tratar do processo de *download* da imagem.

```
private class ImageDownloader extends AsyncTask<String, Void, Bitmap> {

 @Override
 protected Bitmap doInBackground(String... param) {
        return downloadBitmap(param[0]);
 }
 @Override
 protected void onPostExecute(Bitmap result) {
        eventIMG.setImageBitmap(result);
 }
 private Bitmap downloadBitmap(String url) {
        ...
 }
}
```

Código Fonte 16 – Excerto da classe ImageDownloader

Esta classe, quando termina a sua tarefa em *background*, define no elemento `<ImageView>` do *layout* do ecrã do evento a sua imagem, tarefa efetuada no método onPostExecute.

Biblioteca de Partilha de Fotografias

O desenvolvimento da biblioteca teve início quando foram adicionadas ao ficheiro AndroidManifest as permissões necessárias.

```
<uses-feature android:name="android.hardware.camera" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"
/>
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Código Fonte 17 – Permissões necessárias ao funcionamento da biblioteca de partilha de fotografias

O Código Fonte 17 ilustra assim a adição das permissões de acesso à câmara do dispositivo, à leitura e escrita de ficheiro, à Internet e ao estado da ligação.

O *layout* deste ecrã quando carregado para o visor do dispositivo é composto apenas por um botão, que permite ao utilizador tirar uma fotografia.

```
Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
```

Código Fonte 18 – Intent para abrir a aplicação câmara

A acção `ACTION_IMAGE_CAPTURE` do Intent criado quando pressionado o botão permite ao utilizador aceder à camara. A imagem recebida é armazenada no dispositivo, sendo necessário garantir que durante esse processo o álbum da aplicação se encontra criado.

Após armazenada a fotografia do utilizador, são adicionados ao *layout* do ecrã a foto e um botão que permite o seu envio. Quando pressionado, este botão chama em *background* a classe `HttpUploader`, que trata do processo de envio da imagem para o *webservice*, cujo endereço se encontra na classe `Cam_config`.

O processo de envio de uma imagem consiste em fazer *decode* da mesma para *bitmap* e comprimir, resultando num objeto `OutputStream`. Esse objeto é convertido num *array* de *bytes* e posteriormente numa *string*, que acaba por ser colocada na mensagem POST de HTTP enviada para o *webservice*. A resposta à mensagem enviada é colocada num objeto `HttpResponse`, podendo ser confirmado se o envio da imagem foi efetuado com sucesso ou não.

3.2.2.4 Projeto HOME

O projeto de Android HOME é, entre os desenvolvidos, o único que se encontra definido como aplicação, sendo composto por um ecrã que funcionará como menu de navegação entre os vários ecrãs que os módulos oferecem. Este é o ponto principal da aplicação a ser gerada através da plataforma.

É necessário que o ficheiro AndroidManifest presente neste projeto declare todas as permissões e usos requeridos pelas funcionalidades que os módulos precisam (por exemplo OpenGL para os mapas, API *key* do Google Maps, câmara, entre outras), bem como indique a referência para cada uma das Activities que irá apresentar. Nesse sentido, as Activities criadas

nas bibliotecas relativas a cada módulo (contactos, eventos, twitter, entre outros) devem encontrar-se declaradas no AndroidManifest.

De modo a que a aplicação criada possa ser colocada no mercado foi criada uma *keystore* (filKS.keystore), que permite assinar a aplicação durante o processo de compilação. A indicação de que é necessário utilizar uma determinada keystore para assinar a aplicação deve estar presente no ficheiro ant.properties, que se encontra na raiz do projeto.

```
java.encoding=UTF-8
key.store=./home_config/filKS.keystore
key.alias=xalias
key.store.password=xxxxxx
key.alias.password=xxxxxx
```

Código Fonte 19 – Conteúdo do ficheiro ant.properties

Tal como todas as bibliotecas Android desenvolvidas no presente trabalho, este projeto HOME conta com as classes `Home_config` (com tipo de *layout* e número de colunas, caso seja `GridLayout`) e `Home_values` (com o título e subtítulo do ecrã), e ainda com um ficheiro de propriedades processado em tempo de execução que contém informação sobre os módulos que a aplicação irá disponibilizar.

```
MODULE_0=net.filipematos.uebcontacts_lib#MainActivity_contacts#Contactos#ic
_contacts
MODULE_1=net.filipematos.uebinfo_lib#MainActivity_info#Info#ic_info
MODULE_2=net.filipematos.uebwebpage_lib#MainActivity_webpage#Web#ic_web
MODULE_3=net.filipematos.uebevents_lib#MainActivity_events#Eventos#ic_event
s
NUM_OF_MODULES=4
```

Código Fonte 20 – Conteúdo do ficheiro modules.properties

No ficheiro modules.properties é possível encontrar várias informações acerca de cada módulo, tais como: o seu *package name*, nome da Activity principal, título para apresentar no menu e ícone. Este ficheiro é carregado e processado pelo método `onCreate()` da `MainActivity` do projeto Home, que utiliza o seu conteúdo para criar um `ArrayList` preenchido com objetos da classe `Module`. Sempre que o utilizador selecionar um dos módulos no menu é executado o método `startActitivy`, responsável pela criação de um Intent que permite abrir o ecrã pertencente ao módulo selecionado, podendo contudo ser apresentados diferentes *layouts*.

```
private void startActivity(int pos){
 try{
        String target = modulesArray.get(pos).getPackageName() + "." +
modulesArray.get(pos).getActivityName();

        Intent intent = new
Intent().setClassName(getApplicationContext(),target);
        startActivity(intent);
 }catch (ActivityNotFoundException ex){
        Toast.makeText(this.getApplicationContext(), ex.toString(),
Toast.LENGTH_LONG).show();
 }
}
```

Código Fonte 21 – Método startActivity implementado na classe MainActivity

A variável `modulesArray` está relacionada com o `ArrayList` referido anteriormente e é utilizada para obter o *package name* e o nome da Activity do módulo selecionado, utilizados para formar o Intent que levará à visualização do ecrã principal do módulo.

De modo a carregar as Activities dos módulos, as mesmas devem encontrar-se declaradas no AndroidManifest e ser importadas para a `MainActivity`, através da *keyword* `import`. Estes dois processos são efetuados pela plataforma através da invocação de *targets* Ant que se encontram definidos no ficheiro custom_rules.xml, localizado na raiz do projeto.

Targets Ant

Através da linha de comandos a plataforma desenvolvida invocará três *targets* definidos no ficheiro custom_rules.xml, nomeadamente:

- `add-main-activity-import` – *Target* que recebe como parâmetro o nome de uma classe que representa uma Activity, que será utilizado para adicionar um `import` à `MainActivity` do projeto Home.
- `add-activity-manifest` –*Target* que pode receber um ou dois parâmetros, o nome da Activity e o da parent Activity, que serão utilizados para adicionar o elemento `<activity>` ao AndroidManifest, permitindo a navegação para estas Activities. O segundo parâmetro apenas é utilizado para ecrãs cujo ecrã anterior não seja a `MainActivity`, tornando necessária a declaração do seu *parent* com o objetivo de oferecer uma navegação entre ecrãs consistente.
- `buildAPK` – T*arget* principal, é através da sua chamada que a aplicação Android será compilada.

```
<target name="buildAPK">
 <!-- Parameters of buildAPK target: -Dapp_name -Dpackage_name and –
Dgmaps_api_key -->
 <property name="app_name" value="Generated App"/>
 <property name="package_name" value="net.filipematos.ueb.pckg"/>
 <property name="gmaps_api_key"
value="AIzaSyCG5BdLXNW6FANAIkmWX37HiA2Q4bclIdM"/>
 <!-- Update code of Home_config and Home_values files   -->
 <antcall target="loadPropertiesFiles" />
 <!--  Change Application Name on AndroidManifest.xml -->
 <antcall target="add-application-name">
       <param name="app_name" value="${app_name}"/>
 </antcall>
 <!--  Change Google Maps API KEY on AndroidManifest.xml -->
 <antcall target="add-gmaps-key">
       <param name="gmaps_api_key" value="${gmaps_api_key}"/>
 </antcall>
 <!--  Change Application Package Name on AndroidManifest.xml -->
 <antcall target="add-manifest-package-name">
       <param name="package_name" value="${package_name}"/>
 </antcall>
 <!--  Replaces in the application source code the old package name
with the new package name  -->
 <antcall target="change-source-code-package-name">
       <param name="package_name" value="${package_name}"/>
 </antcall>
 <antcall target="clean" />
 <antcall target="release" />
</target>
```

Código Fonte 22 – Excerto do ficheiro custom_rules.xml

Quando invocado, o `buildAPK` recebe três parâmetros e a sua implementação consiste em executar outros *targets* definidos no mesmo ficheiro. Além de serem atualizados os valores das classes `Home_config` e `Home_values` (`loadPropertiesFiles`), são também alterados o *package name* e o nome da aplicação no AndroidManifest (`add-application-name`) e adicionada a API *key* do Google Maps (`add-gmaps-key`). De seguida é executado o *target* `change-source-code-package-name`, que vai alterar o *package name* de todos os ficheiros de código fonte, de modo a permitir que duas aplicações criadas através da plataforma possam coexistir no mesmo dispositivo. O *target* `clean` apaga todos os ficheiros dos diretórios bin/ e gen/ localizados na raiz do projeto. Para finalizar é executado o *target* `release`, que compila o projeto, resultando numa aplicação assinada com a *keystore* e pronta a ser colocada no mercado.

3.2.3 iOS

No presente subcapítulo são abordados os aspetos relacionados com a implementação do projeto iOS, que irá permitir criar aplicações para este sistema operativo através da plataforma. O projeto foi desenvolvido para a versão iOS 7 e foi utilizado o ambiente de desenvolvimento Xcode 6. Uma vez que o foco principal do trabalho desenvolvido foi o Android, para iOS foi apenas criado um projeto que permitisse provar o conceito, com todos os módulos implementados à exceção do Twitter e de partilha de fotos.

A plataforma vai ter como função colocar dados da aplicação no projeto iOS, existindo assim a necessidade de definir alguns ficheiros plist (*property list*) com determinadas configurações. Com a finalidade de armazenar os dados relativos aos estilos da aplicação foi criado o ficheiro styles.plist.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
<key>MAIN_SCREEN_TITLE</key>
<string>FIIFIFI</string>
<key>MAIN_SCREEN_SUB_TITLE</key>
<string>Bem vindo!</string>
<key>LAYOUT_TYPE</key>
<integer>2</integer>
<key>TEXT_COLOR</key>
<string>000000</string>
...
<key>AD_UNIT_ID</key>
<string>n/a</string>
<key>IS_MONETIZATION_ON</key>
<true/>
</dict>
</plist>
```

Código Fonte 23 – Excerto do ficheiro styles.plist

Os ficheiros plist são do tipo XML, e têm uma finalidade idêntica à dos ficheiros de propriedades em Java. Têm como vantagem permitir a utilização de diferentes tipos de dados, entre os quais *strings*, *integers*, "booleanos" e dicionários de dados. No ficheiro modules.plist encontra-se um dicionário de dados que contém informações relativas aos módulos que a aplicação irá fornecer.

```
<plist version="1.0">
<dict>
 <key>num_of_modules</key>
 <integer>7</integer>
 <key>modules</key>
 <dict>
        <key>name_0</key>
        <string>Info</string>
        <key>icon_name_0</key>
        <string>ic_info</string>
        <key>segue_name_0</key>
        <string>push_info</string>
        ...
 </dict>
</dict>
</plist>
```

Código Fonte 24 – Excerto do ficheiro modules.plist

Os ficheiros styles.plist e modules.plist são carregados e tratados quando a aplicação é lançada no dispositivo, através do controller `FMMenuViewController`. Este *controller* é o primeiro a ser executado quando a aplicação arranca, pressupondo três tarefas:

- Preencher o `NSMutableArray *modules` com objetos do tipo `FMModule`, através do processamento do ficheiro modules.plist;
- Inicializar o `NSMutableArray *modules` e um objeto do tipo `FMAppConfig`, que irá conter todos os dados que se encontram disponíveis no ficheiro styles.plist;
- Lançar para o ecrã o menu pretendido através de um *segue*.

3.2.3.1 Transição entre ecrãs

A adoção de *segues* como ferramenta para a transição entre ecrãs obriga à definição dos mesmos na *storyboard*, de forma a que se encontrem preparados para ser invocados através de um *controller*. Cada ecrã tem um *controller* associado que implementa o método `setAppConfig`, invocado por outro *controller* que pretende lançar o primeiro.

```
@implementation FMMenuType1ViewController
-(void)setAppConfig:(FMAppConfig *)appConfig
{
    _appConfig = appConfig;
}
- (void)viewDidLoad
{
...
}
@end
```

Código Fonte 25 – Excerto da classe FMMenuType1ViewController

Quando na aplicação se transita entre ecrãs pela primeira vez é obrigatório o envio do objeto `FMAppConfig`, criado no `FMMenuViewController`. Uma vez que este objeto contém

informações relativas aos estilos da aplicação, cada *controller* vai precisar deste objeto para apresentar de forma correta o respetivo ecrã. Como é possível verificar pelo *controller* `FMMenuType1ViewController` ilustrado no Código Fonte 25, este detalhe torna indispensável que cada *controller* tenha um método `setAppConfig` e um objeto do tipo `FMAppConfig` declarado.

3.2.3.2 Módulos

Ao contrário do que sucede em Android, em que cada módulo representa um projeto isolado, em iOS a aplicação corresponde a um projeto, encontrando-se nesse projeto a implementação de todos os módulos. Cada módulo tem associado um ficheiro .plist, que contém os dados necessários para a correta apresentação do ecrã ao utilizador. Este ficheiro é processado na função `viewDidLoad` do *controller* de cada módulo. De seguida serão abordados detalhes de implementação de apenas alguns dos módulos.

Publicidade

Atualmente existem inúmeros serviços que oferecem a capacidade de rentabilização de aplicações iOS, nomeadamente o iAd, oferecido pela Apple. Para que uma aplicação seja capaz de utilizar este serviço, é necessário incorporar a iAd.framework no projeto da aplicação. Na vista (*view*) onde são apresentados os objetos do ecrã é obrigatório adicionar o elemento `ADBannerView`, que permite a apresentação de publicidade

Todos os *controllers* que implementam publicidade necessitam que seja importado para o seu ficheiro .h (*header file*) o `<iAd/iAd.h>`, que o mesmo contenha um objeto `ADBannerView` e que seja implementado o protocolo `ADBannerViewDelegate`.

```
#import <iAd/iAd.h>
#import "FMAppConfig.h"

@interface FMEventsViewController :
UIViewController<ADBannerViewDelegate, ...>

@property(nonatomic, strong)FMAppConfig *appConfig;
@property(nonatomic, strong)NSMutableArray* eventsArray;
@property (weak, nonatomic) IBOutlet ADBannerView *adView;
@property(nonatomic, strong)NSString* screen_title;
...
@end
```

Código Fonte 26 – Excerto do *header file* da classe FMEventsViewController

A implementação do protocolo `ADBannerViewDelegate` obriga o *controller* a reagir às alterações que ocorram ao objeto `ADBannerView`. Para garantir o seu correto funcionamento, o protocolo obriga à implementação de duas funções:

```
-(void) bannerViewDidLoadAd:(ADBannerView *)banner{
          [UIView beginAnimations:nil context:nil];
    [UIView setAnimationDuration:1];
    [banner setAlpha:1];
    [UIView commitAnimations];
}

-(void) bannerView:(ADBannerView *)banner
didFailToReceiveAdWithError:(NSError *)error{
    [UIView beginAnimations:nil context:nil];
    [UIView setAnimationDuration:1];
    [banner setAlpha:0];
    [UIView commitAnimations];
}
```

Código Fonte 27 – Funções que o protocolo ADBannerViewDelegate obriga a implementar

A primeira é invocada quando um novo anúncio publicitário se encontra pronto para ser visualizado, e o segundo quando ocorre algum erro ao tentar carregar um novo anúncio. No Código Fonte 27, é possível verificar que a única diferença entre ambos é a invocação do `setAlpha`, cujo parâmetro é 1 quando o anúncio está pronto (visível) e 0 quando ocorreu um erro (invisível).

Para além das duas funções que o protocolo `ADBannerViewDelegate` obriga a implementar, é necessário adicionar o Código Fonte 28 à função `viewDidLoad` de todos os *controllers* que apresentem publicidade.

```
if(self.appConfig.is_monetization_on){
  self.adView.delegate = self;
}else{
  self.adView.delegate = nil;
}
```

Código Fonte 28 – Excerto da função viewDidLoad de todos os *controllers* iOS

Fundamentalmente é no bloco de código ilustrado que é verificado se a aplicação contém publicidade. Em caso afirmativo, é indicado ao objeto `ADBannerView` que será controlado pelo *controller* atual (*self*), caso contrário não sofrerá controlo (*nil*).

Módulo contactos

No ecrã deste módulo são apresentados ao utilizador dados de contacto e um mapa. Este conteúdo é obtido através do processamento do ficheiro contacts_config.plist, que armazena dados como as coordenadas (latitude e longitude) de um local, endereço de e-mail, *website*, entre outros.

Para que uma aplicação seja capaz de utilizar um mapa é necessário incorporar a MapKit.framework no projeto da mesma. Na vista (*view*) onde se encontram os objetos do ecrã é obrigatório adicionar o elemento `MKMapView`, de modo a que o mapa seja apresentado.

O *controller* que apresenta o mapa deve importar o ficheiro `<MapKit/MapKit.h>` e implementar o protocolo `MKMapViewDelegate` no seu *header file*, bem como declarar a relação do mesmo com o objeto adicionado à vista que ele controla. A função `viewDidLoad` do *controller* `FMContactsViewController` invoca a `setupMap`, que é responsável por configurar o mapa que será apresentado ao utilizador.

```
-(void)setupMap:(NSString *)notationTitle{
 self.mapView.delegate = self;

 [_mapView setMapType:MKMapTypeStandard];
 [_mapView setZoomEnabled:YES];
 [_mapView setScrollEnabled:YES];
  ...
 CLLocationCoordinate2D coordinate1;
 coordinate1.latitude = _latitude;
 coordinate1.longitude = _longitude;

 FMMapAnnotation *annotation = [[FMMapAnnotation alloc]
initWithCoordinate:coordinate1 title:notationTitle];
 [self.mapView addAnnotation:annotation];
}
```

Código Fonte 29 – Função setupMap do *controller* FMContactsViewController

A primeira instrução ilustrada no Código Fonte 29 tem como objetivo indicar ao objeto `MKMapView` que o responsável por ele é o *controller* atual (*self*). De seguida, são definidos parâmetros de configuração para o mapa, tais como a possibilidade de fazer *zoom* ou *scroll*. O objeto `coordinate1` representa as coordenadas sobre as quais será colocado um ponteiro, tendo sido desenvolvida para colocar o mesmo no mapa a classe `FMMapAnnotation`, que implementa o protocolo `MKAnnotation`.

```
@interface FMMapAnnotation : NSObject <MKAnnotation>
```

Código Fonte 30 – Declaração da classe FMMapAnnotation

Módulo Eventos

Em iOS, o módulo eventos é composto por três ecrãs: um que lista todos os eventos carregados através do *webservice* do EventBrite, outro com os detalhes do evento, e um último onde é possível visualizar os detalhes do local (inclui mapa). Para desenvolver estes três ecrãs foram construídos três *controllers*: o `FMEventsViewController`, o `FMEventDetailsViewController` e o `FMEventVenueViewController`, respetivamente.

Para suportar os dados que serão apresentados nestes ecrãs foram criados os objetos `FMEvent` e `FMVenue`, que representam um evento e um local.

Listagem dos eventos

Na storyboard do projeto, o ecrã da listagem de eventos tem definido, entre outros, um elemento do tipo `UITableView`. Este objeto permite apresentar no ecrã uma tabela dividida em secções e com várias linhas. As linhas são representadas pela classe `FMEventsTableViewCell`, que possui três propriedades que determinam três elementos visuais: *label* do título, data e local do evento.

O *controller* deste ecrã (`FMEventsViewController`) implementa os protocolos `UITableViewDelegate` e `UITableViewDataSource`, necessários para que possa controlar a tabela.

Dentro da função `viewDidLoad` do `FMEventsViewController` é processado o events_config.plist, de modo a obter o endereço URL do *webservice* do EventBrite, entre outros dados. Posteriormente é indicado ao elemento `UITableView` que este será controlado e preenchido pelo `FMEventsViewController` (*controller* atual).

```
[self.tbl_events setDelegate:self];
[self.tbl_events setDataSource:self];
```

Código Fonte 31 – Definição do responsável da tabela tbl_events

O *download* e processamento dos dados em formato JSON fornecidos pelo *webservice* é efetuado em *background*, através do acesso à classe `FMDownloadManager`.

```
-(void)getEventBriteEvents{
 [FMDownloadManager download:self.url
withCompletionBlock:^(NSData *result) {
        ...
        self.eventsArray = [[NSMutableArray alloc]init];
        ...
        FMEvent *event = [[FMEvent alloc]init];
        event.title = [dicEvent valueForKeyPath:@"title"];
        event.startDate = [dicEvent
valueForKeyPath:@"start_date"];
        ...
        [self.eventsArray addObject:event];
        ...
        [self.tbl_events reloadData];
    }];
}
```

Código Fonte 32 – Excerto da função getEventBriteEvents do FMEventsViewController

Após efetuar o *download*, é realizado o tratamento do JSON recebido através do acesso ao *webservice*, que resulta num *array* de objetos `FMEvent`. A variável `dicEvent` é do tipo `NSDictionary`, e representa um evento em JSON. Após concluído o tratamento do JSON é pedido que seja atualizado o conteúdo da `UITableView`.

```
- (void)tableView:(UITableView *)tableView
didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    NSString * segueName = @"push_single_event";
    [self performSegueWithIdentifier:segueName sender:self];
}

- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section
{
    return self.eventsArray.count;
}

- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell = [tableView
dequeueReusableCellWithIdentifier:@"EventsViewCell"
forIndexPath:indexPath];
    FMEventsTableViewCell* eventsCell =
(FMEventsTableViewCell * ) cell;

    FMEvent *object = self.eventsArray[indexPath.row];
    eventsCell.lblEventTitle.text = object.title;
    eventsCell.lblEventTitle.textColor = [self.ut
colorWithHexString:self.appConfig.highlight_text_color];
    ...
    return cell;
}
```

Código Fonte 33 – Excerto da classe FMEventsViewController

As quatro funções ilustradas no encontram-se implementadas no *controller* `FMEventsViewController`, uma vez que os protocolos `UITableViewDelegate` e `UITableViewDataSource` assim o obrigam. A primeira é utilizada para indicar quantas secções a tabela irá ter. Uma vez que a tabela irá conter apenas eventos, haverá apenas uma secção.

A segunda função é invocada quando uma linha da tabela é pressionada pelo utilizador. Quando selecionada uma linha da tabela, é apresentado ao utilizador um ecrã com detalhes do evento. Para efetuar essa transição é criado um *segue*, que funciona de forma semelhante ao explicado anteriormente no contexto do `FMMenuViewController`.

A função `tableView:numberOfRowsInSection` determina quantas linhas irá ter cada secção da tabela, sendo que neste caso em particular o número de linhas é igual ao número de elementos que existem no *array* de eventos (`eventsArray`).

A última e quarta função, a `tableView cellForRowAtIndexPath`, é responsável por definir a apresentação da linha da tabela no índice (`indexPath`). Cada linha da tabela é identificada pela palavra "`EventsViewCell`", que é utilizada para obter o objeto do tipo `UITableViewCell`. O objeto, após convertido para `FMEventsTableViewCell`, preenchido e personalizado, é enviado como retorno para a tabela.

Detalhes do evento

Neste ecrã são apresentados ao utilizador detalhes acerca do evento, nomeadamente uma ligação para o seu *website* e um ecrã com pormenores do local. Estas operações são disponibilizadas através de dois botões visíveis no ecrã do dispositivo.

```
- (IBAction)btn_EventURL:(id)sender {
    [[UIApplication sharedApplication] openURL:[NSURL
URLWithString:self.event.url]];
}

- (IBAction)btn_VenueDetails:(id)sender {
    NSString * segueName = @"push_venue_details";
    [self performSegueWithIdentifier:segueName sender:self];
}
```

Código Fonte 34 – Excerto da classe FMEventDetailsViewController

A primeira função apresentada no Código Fonte 34 é invocada quando o utilizador pressiona o botão que permite visualizar o *website* do evento. A instrução dentro da função lança um pedido às aplicações instaladas para abrir o URL, como por exemplo o Safari ou outra aplicação semelhante.

A segunda função executa o *segue* com o identificador "`push_venue_details`", que representa a transição para o ecrã dos detalhes do local do evento. Durante este processo é enviado para o ecrã com a informação do local os objetos `FMAppConfig` e `FMVenue`.

Detalhes do local

O ecrã de detalhes do local contém vários elementos, entre eles um mapa. A implementação do mapa é idêntica à descrita anteriormente para o módulo de contactos, sendo que o *controller* deste ecrã necessita de implementar o protocolo `MKMapViewDelegate`.

3.2.4 Website

Na presente secção é descrito o processo de implementação do *website*, o ponto central do criador automático de aplicações Android e iOS a ser construído. Serão abordados, entre outros aspetos, conceitos relacionados com o tema em questão, o diagrama de classes, os módulos utilizados e o *layout* definido.

3.2.4.1 Conceitos relacionados

Para facilitar a compreensão do leitor relativamente às etapas executadas no desenvolvimento do *website*, encontram-se explicados no presente subcapítulo alguns conceitos relacionados com o tema.

Symfony 2

A versão inicial do Symfony, anunciada em Fevereiro de 2007 (Zaninotto, 2007), foi dos primeiros *frameworks* PHP a surgir que suportam uma *debug toolbar*. Em Julho de 2011 é lançado o Symfony 2 (Symfony-1, 2014), um conjunto de bibliotecas independentes que podem ser utilizadas no desenvolvimento de um projeto em PHP. O facto de ser composto por mais de vinte bibliotecas leva a que o Symfony seja muitas vezes descrito como sendo um *full-*

stack web framework. As bibliotecas, denominadas componentes Symfony, são sempre uma ferramenta útil, independentemente do projeto *web* desenvolvido. De seguida encontram-se descritos os componentes considerados mais importantes para o desenvolvimento do *website* (Symfony-2, 2014):

- HttpFoundation – contém, entre outras, as classes Request e Response, utilizadas para lidar com as sessões de utilizadores e *upload* de ficheiros;
- Routing – sistema de encaminhamento rápido e poderoso que permite mapear um URI (*Uniform Resource Identifier*) específico (ex: /contact) a partir de instruções que indicam como esse pedido deve ser tratado (um exemplo de instrução é invocar a função contactAction());
- Form – um *framework* completo e flexível que permite criar formulários e processar o seu preenchimento;
- Validator – sistema que permite definir regras relativamente aos dados e posteriormente validar se os dados inseridos pelos utilizadores seguem as mesmas;
- ClassLoader – uma biblioteca que permite carregar classes PHP automaticamente, o que possibilita a utilização dessas classes sem as importar *a posteriori*;
- Templating – um conjunto de ferramentas que permite, entre outras tarefas, o suporte à herança entre *templates* e a sua renderização.
- Security – uma biblioteca utilizada para lidar com todos os tipos de segurança passíveis de existir dentro de uma aplicação *web*;
- Translation – um *framework* utilizado para traduzir texto (*strings*) dentro da aplicação.

Todos os componentes referidos acima são de baixo acoplamento e podem ser utilizados em qualquer projeto *web*, mesmo que não seja utilizado o Symfony no desenvolvimento. O processo de criação do *website* foi efetuado através da linha de comandos, bem como a definição das entidades que representam o modelo de dados e a criação dos respetivos formulários CRUD (*Create, Read, Update e Delete*), fator que na maioria dos casos causa a diminuição do tempo necessário para o desenvolvimento.

Doctrine

Lançado em Setembro de 2008 (DoctrineProject-1, 2008), o Doctrine é composto por um conjunto de bibliotecas PHP focadas no armazenamento de dados e mapeamento de objetos.

Os principais projetos desta ferramenta são o Object Relational Mapper (ORM) e o Database Abstraction Layer (DBAL). O Doctrine baseia-se em tecnologias de persistência de objetos como o Hibernate (Java) e o ActiveRecord (Ruby on Rails).

Object Relational Mapper (ORM)

Uma das principais funcionalidades desta camada, que se encontra acima da DBAL, é possibilitar a consulta de numa linguagem SQL proprietária e orientada a objetos, denominada Doctrine Query Language (DQL). Esta camada confere aos programadores uma alternativa ao SQL original, permitindo obter flexibilidade na programação sem a necessidade de ter código duplicado (DoctrineProject-2, 2014).

Database Abstraction Layer (DBAL)

A DBAL é uma fina camada em torno da API PDO (*PHP Data Objects*) que oferece diversas funcionalidades, como por exemplo a manipulação da base de dados através de uma API orientada a objetos. O facto de esta camada abstrair a API PDO possibilita o desenvolvimento e personalização de *drivers* de acesso à base de dados. A DBAL suporta atualmente vários sistemas de gestão de base de dados, tais como o MySQL, Microsoft SQL Server, PostgreSQL, SQLite, entre outros, e pode ser utilizada independentemente do ORM (Object Relational Mapper) (DoctrineProject-3, 2014).

A ferramenta Doctrine encontra-se integrada no Symfony, podendo ser utilizada com outros *frameworks* PHP, nomeadamente com o Zend e o CodeIgnitor.

Twig

O Twig é um motor de renderização de *templates* para PHP caracterizado por ser flexível, seguro e rápido. O seu lançamento oficial ocorreu em 2011 (Twig, 2011), e atualmente requer para a sua execução a versão 5.2.4 do PHP, ou superior. Este motor permite o desenvolvimento de *templates* recorrendo a uma sintaxe significativamente menos detalhada do que a utilizada em PHP.

```
<?php echo $var ?>
```

Código Fonte 35 – Exibição do conteúdo da variável $var em PHP

O Código Fonte 35 – Exibição do conteúdo da variável $var em PHP, apresentado em PHP, permite que seja visualizado no ecrã o conteúdo da variável $var. O mesmo objetivo é alcançável com o Twig com o seguinte código:

```
{{ var }}
```

Código Fonte 36 – Exibição do conteúdo da variável var com Twig

Este motor de renderização de *templates* encontra-se integrado por omissão no Symfony e foi utilizado na implementação da plataforma desenvolvida.

YAML

O YAML (YAML Ain't Markup Language) é uma linguagem de codificação de dados (Evans, 2014). Uma vez que a sua sintaxe é significativamente menos pormenorizada do que as linguagens de marcação mais comuns, como o XML ou o HTML, é considerado por muitos programadores como uma linguagem de marcação rápida.

O YAML caracteriza-se por ser relativamente fácil de ler por humanos e por contar com uma sintaxe simples, atributos que fazem com que permita mapear os tipos de dados mais comuns na maioria das linguagens de programação. Adicionalmente, utiliza uma notação baseada na identação do código.

Assim, atualmente é possível utilizar o YAML em várias linguagens de programação, como por exemplo em C/C++, Objective-C, PHP, Python, Java ou Javascript.

3.2.4.2 Gerar o Bundle da plataforma

Em Symfony, um *bundle* (pacote) é um diretório com uma estrutura bem definida que pode conter classes, *controllers*, *templates* ou recursos *web*. Para o desenvolvimento da plataforma foi necessário criar um *bundle* através da linha de comandos.

```
php app/console generate:bundle --namespace=Ueb/AppBuilderBundle --
format=yml –dir=~Sites/uebAppBuilder/
```

Comando 2 – Comando Symfony utilizado para criar um *bundle*

O Comando 2 permite a criação de um *bundle*, onde estão definidos o *namespace*, o diretório onde irá ser criado o *bundle* e o formato dos ficheiros de configuração, como por exemplo de

encaminhamento. O formato utilizado pode variar (XML, PHP, anotações), tendo-se optado pelo YAML.

3.2.4.3 Diagrama de classes

O diagrama de classes representa a estrutura da plataforma gerada, sendo possível verificar através do mesmo a forma como as classes se relacionam entre si. A sua construção tem por base o objectivo de satisfazer os requisitos funcionais definidos na fase de análise. O diagrama de classes ilustrado no Anexo 1 – Excerto do diagrama de classes do *Website*apresenta apenas as classes que fazem parte do modelo. Ainda assim, encontra-se de alguma forma incompleto devido à sua longa extensão, não estando representadas as classes: `ContactScreen`, `FaqScreen`, `Faq`, `InfoScreen`, `RSSScreen`, `TwitterScreen`, `WebScreen` e `CameraScreen`.

Cada uma das classes utilizadas na plataforma foi desenvolvida em quatro passos:

1. Definir os atributos da entidade através da criação de um ficheiro YAML – ORM;
2. Executar o comando que permite criar a classe PHP através do ficheiro ORM;
3. Criar o comando para gerar uma tabela na base de dados que represente a entidade descrita no ficheiro ORM;
4. Executar o comando que gera os formulários para as quatro operações básicas a que qualquer entidade se encontra afeta: criar, atualizar, ler e apagar. Este passo engloba a criação de um ficheiro de configuração de encaminhamento, *templates* e um *controller* para processar estas operações.

3.2.4.4 Entidade Aplicação

A entidade aplicação representa uma aplicação móvel para Android e iOS, sendo necessário que contenha dados como o nome da aplicação que será gerada, o título do ecrã principal, a cor do texto, entre outros aspetos. O primeiro passo dado para criar esta entidade foi definir no ficheiro Application.orm.yml os seus atributos, as relações com outras entidades, e ainda os *callbacks* para os diferentes estados em que a entidade se poderá encontrar.

O valor que se segue à *keyword* table define qual o nome da tabela que representará esta entidade na base de dados. O identificador terá o nome `id` e o seu valor será incrementado automáticamente.

Como existe a necessidade de esta entidade ter um *array* de módulos, foi definida uma relação de "muitos-para-muitos" com a entidade `Module`. Os detalhes da tabela que irá

relacionar as entidades Aplicação e Módulo também se encontram especificados no Application.orm.yml.

```
    Ueb\AppBuilderBundle\Entity\Application:
        type: entity
        table: application
        id:
          id:
            type: integer
            generator: { strategy: AUTO }
        fields:
          AppName:
            type: string
            length: 75
          Cfg_TextColor:
            type: string
            length: 15
            nullable: true
     ...
          CreatedAt:
                type: datetime
        # ******************************
        # Table Relationships
        # ******************************
        manyToMany:
            modules:
                targetEntity: Module
                joinTable:
                    name: application_module
                    joinColumns:
                        app_id:
                            referencedColumnName: id
                    inverseJoinColumns:
                        module_id:
                            referencedColumnName: id

        lifecycleCallbacks:
            prePersist: [ preUpload, setCreatedAtValue, setSlugName ]
            preUpdate: [ preUpload, setSlugName]
            postPersist: [ upload ]
            ...
```

Código Fonte 37 – Conteúdo do ficheiro Application.orm.yml

Por último, após a *keyword* `lifecycleCallbacks` encontram-se discriminadas as funções que serão invocadas nos diferentes estados em que a entidade se poderá encontrar. Na sua maioria estão relacionadas com o *upload* de várias imagens, como é o caso do ícone da aplicação ou da imagem de fundo.

Gerar a classe Aplicação

Para que a classe PHP Application seja gerada através do Application.orm.yml, é necessária a execução do seguinte comando:

```
php app/console doctrine:generate:entities UebAppBuilderBundle
```

Comando 3 – Comando Symfony utilizado para gerar as entidades de um determinado bundle

Ao ser invocado, este comando cria todas as classes das respetivas entidades que se encontrem definidas no diretório uebAppBuilder/src/Ueb/AppBuilderBundle/Resources/config/doctrine/.

Gerar a tabela Aplicação na base de dados

A criação na base de dados das tabelas que representam as entidades é feita através da linha de comandos, existindo duas formas de o fazer, utilizando dois comandos diferentes.

```
php app/console doctrine:database:create
```

Comando 4 – Comando Symfony que permite a criação da base de dados

O Comando 4 permite criar a base de dados de raiz, o que faz com que na maioria dos projetos seja executado apenas uma vez.

```
php app/console doctrine:database:update --force
```

Comando 5 – Comando Symfony para atualizar a base de dados

O comando `doctrine:database:update` atualiza a base de dados em vigor, pressupondo assim que a base de dados tenha sido criada anteriormente à sua utilização. Qualquer um dos dois comandos permite que seja adicionado o parâmetro `--dump-sql`, que apresenta no ecrã o script SQL que será executado para realizar a operação desejada.

Gerar formulários CRUD

É também através da execução de uma linha de comandos que são criados os formulários que permitem criar, atualizar, ler e apagar uma aplicação.

```
php app/console doctrine:generate:crud --
entity=UebAppBuilderBundle:Application --format=yml
```

Comando 6 – Comando Symfony utilizado para criar os formulários CRUD da entidade Application

A introdução na consola do comando apresentado resulta no preenchimento dos vários parâmetros que a mesma requer, à medida que os mesmos vão sendo pedidos, como por exemplo o prefixo de encaminhamento /app.

No caso particular da entidade aplicação, a execução do comando cria vários ficheiros, nomeadamente a classe `ApplicationController`, o ficheiro de encaminhamento (application.yml) dentro do diretório uebAppBuilder/src/Ueb/AppBuilderBundle /Resources/config/routing/, e quatro vistas (*views*): index.html.twig, new.html.twig, edit.html.twig e show.html.twig. As quatro vistas permitem, respectivamente: visualizar uma listagem de aplicações (*index*), criar uma nova aplicação (*new*), atualizar ou apagar a mesma (*edit*) e consultar os seus detalhes (*show*).

```
    app:
        pattern:  /
        defaults: { _controller: "UebAppBuilderBundle:Application:index" }
    ...
    app_create:
        pattern:  /create
        defaults: { _controller: "UebAppBuilderBundle:Application:create" }
        requirements: { _method: post }

    app_edit:
        pattern:  /{appname_slug}/edit
        defaults: { _controller: "UebAppBuilderBundle:Application:edit" }
    ...
    UebAppBuilderBundle_app:
        resource:
    "@UebAppBuilderBundle/Resources/config/routing/application_manage.yml"
        prefix:   /{appname_slug}/manage
```

Código Fonte 38 – Excerto do conteúdo do ficheiro de encaminhamento application.yml

O Código Fonte 38 representa um pequeno excerto do conteúdo do ficheiro de encaminhamento application.yml, relativo à entidade aplicação. Neste ficheiro encontram-se identificadas as várias rotas existentes, às quais está associado um nome (ex: `app_new`), o URL padrão e o *controller* que irá lidar com a operação pedida. Em alguns casos encontra-se especificado o tipo de pedido HTTP que deve ser aceite para satisfazer a operação requerida (ex: `app_create`).

A primeira rota no Código 42 define que o acesso à plataforma (ex: http://localhost/uebAppBuilderBundle/) será tratado pela função `indexAction`, que se encontra definida no `ApplicationController` do UebAppBuilderBundle.

```
namespace Ueb\AppBuilderBundle\Controller;
...
class ApplicationController extends Controller
{
    public function indexAction()
    {
        $em = $this->getDoctrine()->getManager();

        $entities = $em->getRepository('UebAppBuilderBundle:Application')-
>findAll();

        return $this-
>render('UebAppBuilderBundle:Application:index.html.twig', array(
            'entities' => $entities,
        ));
    }
  ...
}
```

Código Fonte 39 – Excerto do ApplicationController

Esta função obtém todos os objetos do tipo aplicação existentes na base de dados e que foram enviados para o *layout* index.html.twig. O *layout* irá utilizar a variável entities para apresentar ao utilizador uma listagem de todas as aplicações criadas através da plataforma.

```
{% extends '::base.html.twig' %}
...
{% block body -%}
    <h1>Application list</h1>
    <table class="records_list table" >
        <thead>
            ...
        </thead>
        <tbody>
        {% for entity in entities %}
            <tr>
                <td><a href="{{ path('app_manage', { 'appname_slug':
entity.AppNameSlug }) }}">{{ entity.id }}</a></td>
                <td>{{ entity.AppName }}</td>
                <td>{{ entity.Icon }}</td>
                <td>
                    <a href="{{ path('app_manage', { 'appname_slug':
entity.AppNameSlug }) }}" class="btn btn-info">manage</a>
                </td>
            </tr>
        {% endfor %}
        </tbody>
    </table>

    <a href="{{ path('app_new') }}" class="btn btn-default">
        Create a new entry
    </a>
{% endblock %}
```

Código Fonte 40 – Excerto do ficheiro *layout* index.html.twig da entidade Aplicação

A primeira linha do Código Fonte 40 ilustrado acima, indica que o atual *layout* é uma extensão do ficheiro base.html.twig, que se encontra localizado no directório uebAppBuilder/app/Resources/views/. A instrução `{% block body -%}` indica que o bloco com o nome body existente no *base layout* irá ser reescrito no base_management.html.twig. Este excerto de código itera todas as aplicações (entities) através da instrução for, e apresenta o id, nome e ícone da aplicação. Além da tabela, é apresentado ao utilizador um botão que permite a criação de uma nova aplicação `{{ path('app_new') }}`. A rota `app_new` encontra-se definida no application.yml, localizado em uebAppBuilder/src/Ueb/AppBuilderBundle /Resources/ config/routing/, e a função que tratará o pedido será a `newAction`, definida no `ApplicationController`.

```
    public function newAction(){
     $entity = new Application();
     $form   = $this->createCreateForm($entity);

     return $this->render('UebAppBuilderBundle:Application:new.html.twig',
    array(
            'entity' => $entity,
            'form'   => $form->createView(),
            'icon_path' => $this->DEFAULT_APP_ICON_PATH,
     ));
    }

    private function createCreateForm(Application $entity)
    {
     $form = $this->createForm(new ApplicationType(), $entity, array(
            'action' => $this->generateUrl('app_create'),
            'method' => 'POST',
     ));

     $form->add('submit', 'submit', array('label' => 'Create',
            'attr' => array('class' => 'btn btn-success')));

     return $form;
    }
```

Código Fonte 41 – Excerto de código do ApplicationController

A função `newAction` instância um novo objeto da classe `Application`, faz uma chamada à função `createCreateForm` e posteriormente, através da função render, pede que seja apresentado ao utilizador o *layout* new.html.twig, enviando como parâmetros o objeto `Application`, o formulário e o ícone, por omissão da aplicação.

A função `createCreateForm` cria um formulário HTML (recorrendo à classe `ApplicationType`), que apenas aceita pedidos HTTP do tipo POST, sendo a submissão do formulário tratada pela função `createAction()`, do `ApplicationController` (rota `app_create`).

```
$form->handleRequest($request);
if ($form->isValid()) {
  $em = $this->getDoctrine()->getManager();

  $em->persist($entity);
  $em->flush();

  return $this->redirect($this->generateUrl('app_manage',
array('appname_slug' => $entity->getAppNameSlug())));
}
```

Código Fonte 42 – Excerto da função createAction do ApplicationController

O excerto apresentado no Código Fonte 42 é retirado da função `createAction()`, na qual é validado o formulário submetido e armazenado na base de dados o objeto `$entity`, do tipo Application. Posteriormente o utilizador é encaminhado para a rota `app_manage`, onde poderá realizar diversas tarefas, tais como adicionar ecrãs ou configurar a aparência da aplicação.

3.2.4.5 Módulos

Na plataforma desenvolvida, uma aplicação pode conter um ou mais módulos, sendo que um módulo representa um ecrã dentro da aplicação. De modo a forçar todos os ecrãs a implementar funções determinantes para o correto funcionamento da plataforma e oferecer funções comuns a todos os ecrãs, foi criada a classe abstrata Screen.

```
namespace Ueb\AppBuilderBundle\Entity;

abstract class Screen
{
  abstract public function getScreenTitle();
  abstract public function writeAndroidResources(Module $module,
$project_path, $log_file_path);
  abstract public function writeIOSResources(Module $module, $project_path,
$log_file_path);
  abstract protected function getConfigPropertieFileContent();
  abstract protected function getValuesPropertieFileContent();
  ...
}
```

Código Fonte 43 – Definição da classe abstrata Screen

Esta classe obriga que todas as outras classes implementem as funções que se encontram declaradas no Código Fonte 43. A primeira das cinco funções descritas é utilizada para retornar o título do ecrã.

A segunda e terceira são invocadas para serem criados todos os recursos relacionados com o ecrã, como por exemplo imagens ou ficheiros de propriedades (.properties ou .plist, para Android e iOS respetivamente).

As últimas duas funções são invocadas pela função `writeAndroidResources`, e têm como retorno o conteúdo que será escrito nos ficheiros de propriedades existentes em todas as bibliotecas desenvolvidas em Android (por exemplo: events_config.properties e events_values.properties).

```
class EventsScreen extends Screen
{
      public function writeAndroidResources(Module $module, $project_path,
     $log_file_path)
         {
             $resources = $module->getResources();
             foreach ($resources as $resource) {
                 switch ($resource->getResourceInternID()) {
                     case 0: //PROPERTIES CONFIG
                         $configFileContent = $this-
     >getConfigPropertieFileContent();
                         if(!$this->writePropFile($resource,
     $configFileContent,$log_file_path, $project_path))
                                 return false;

                         break;
                           ...
                 }
             }
             return true;
         }

      protected function getConfigPropertieFileContent() {
             $content= 'CONFIG.CONTENT_SOURCE='. $this-
     >getEventContentSourceType() . "\n"
                     . 'CONFIG.CONTENT_URL='. $this->getEventURL() . "\n"
                     . 'CONFIG.MAX_NUM_OF_EVENTS='. $this-
     >getEventMaxNumOfItems()
                     ;
             return $content;
         }
       ...
     }
```

Código Fonte 44 – Excerto da implementação da classe EventScreen

O Código Fonte 44 representa um pequeno excerto do código relativo à implementação da classe `EventsScreen`, que estende a classe `Screen`, facto que obriga à implementação das funções `writeAndroidResources` e `getConfigPropertieFileContent`, entre outras.

A implementação dos formulários CRUD dos vários ecrãs foi desenvolvida de forma semelhante à explicada anteriormente para os formulários da entidade Aplicação.

3.2.4.6 Layout

Para o desenho da interface da plataforma foram criados dois ficheiros, o base.html.twig e o base_management.html.twig, que definem a aparência global da plataforma e estão ambos localizados em uebAppBuilder/app/Resources/views/.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>{% block title %}Welcome Page!{% endblock %}</title>

        {% block stylesheets %}
        <link
href="{{ asset('bundles/uebappbuilder/css/bootstrap.css') }}"
rel="stylesheet">
        ...
        {% endblock %}
        ...
    </head>
    <body>
        <div id="wrapper">
            {% block sidebar_header %}
            {% endblock %}

               <div id="page-wrapper">
                      {% block body %}
                      {% endblock %}
               </div>
        </div>
        {% block javascripts %}
        <script type="text/javascript"
src="{{ asset('bundles/uebappbuilder/js/jquery-2.1.0.js') }}">
        </script>
        {% endblock %}
        ...
    </body>
</html>
```

Código Fonte 45 – Excerto do conteúdo do ficheiro base.html.twig

No primeiro ficheiro (base.html.twig) encontra-se definida a estrutura geral da plataforma, que está dividida em vários blocos que podem ser personalizados dentro de outros *layouts*. O segundo ficheiro (base_management.html.twig) vai implementar os blocos title e sidebar_header do base.html.twig. Na plataforma, grande parte dos *layouts* estende o `base_management`, à excepção dos que correspondem aos ecrãs. Os *layouts* que permitem adicionar ou editar um ecrã são uma extensão do screens.html.twig.

3.2.4.7 Processo de compilação

O processo de compilação de aplicações Android e iOS é executado por duas classes, que consistem numa implementação da interface `AppBuilderInterface`.

```
interface AppBuilderInterface {
    public function build(\Ueb\AppBuilderBundle\Entity\Application
$application = null);
}
```

Código Fonte 46 – Declaração da interface AppBuilderInterface

Esta interface foi criada com o objetivo de obrigar ambas as classes a implementar a função build e a permitir que no futuro seja possível estender a compilação de aplicações móveis a outros sistemas operativos.

Android

A criação de uma aplicação Android através da plataforma é um processo bastante complexo e que compreende um elevado número de etapas.

A primeira etapa consiste em gerar um nome para atribuir a um diretório a ser criado, e dentro do qual será efetuado todo o processo de compilação. Após criado o diretório de destino, é copiado para este o projeto HOME, seguidamente atualizado através do comando `android update project --path CAMINHO_DO_PROJETO`.

Módulos

Na etapa seguinte é efetuado o tratamento dos módulos que irão fazer parte da aplicação, através da sua iteração. Para cada módulo são realizadas as seguintes tarefas:

- Copiar a biblioteca Android que representa o módulo para o diretório destino;
- Escrever os ficheiros de propriedades que a biblioteca contém (ex: events_config.properties e events_values.properties);
- Executar o *target* Ant: `buildLib`;
- Atualizar a biblioteca através do comando `android update lib-project --path caminho_da_biblioteca`;
- Adicionar a biblioteca ao projeto HOME: `android update project --path caminho_do_projeto --library caminho_da_biblioteca`;
- Iterar as Activities que fazem parte do módulo e de seguida:
    - Executar o target Ant `add-activity-manifest`, com o objetivo de adicionar o elemento `<activity>` ao AndroidManifest.xml do projeto HOME;

- Executar o *target* Ant `add-main-activity-import`, que adiciona o `import` da Activity ao MainActivity.java do projeto HOME;
- Colocar na variável `$ModulesPropertiesFileContent` conteúdo relativo ao módulo.

Após o processamento de todos os módulos da aplicação, o conteúdo da variável `$ModulesPropertiesFileContent` é escrito no ficheiro modules.properties do projeto HOME (Código Fonte 47).

```
if($this->processModules($application)){
 $this->ModulesPropertiesFileContent = $this-
>ModulesPropertiesFileContent
        . 'NUM_OF_MODULES=' . $application->getModules()-
>count();
 Util::writeToFile($this->project_path
        . $this->main_AndroidModuleDirectoryName . '/'.
$this->main_ModulesPropertiesFile,
        $this->ModulesPropertiesFileContent);
 ...
}
```

Código Fonte 47 – Excerto de código da classe AndroidAppBuilder

Na etapa que se segue é processada a biblioteca de estilos.

Biblioteca de Estilos

O processamento da biblioteca de estilos encontra-se dividido em cinco passos, que se encontram identificados de seguida:

- Copiar a biblioteca de estilos para o diretório de destino;
- Escrever o ficheiro de propriedades que a biblioteca irá utilizar – styles_config.properties;
- Copiar para a biblioteca a imagem de fundo, no caso de existir;
- Executar o *target* Ant `buildLib`;
- Copiar os ícones de menu de cada módulo.

Finalizado o processamento da biblioteca de estilos, o ícone da aplicação é copiado para o projeto HOME e são criados os ficheiros de propriedades home_config.properties e home_values.properties.

```
$cmd = 'cd ' . $this->project_path . $this-
>main_AndroidModuleDirectoryName
                . '; ant'
        . ' -Dapp_name="' . $application->getAppName() . '"'
        . ' -Dpackage_name="' . $application->getAndroidPackageName() .
'"';
if($gmaps_api_key != ''){
   $cmd = $cmd  . ' -Dgmaps_api_key="' . $gmaps_api_key. '"';
}
$cmd = $cmd. ' buildAPK';

if($this->execCommand($cmd,
  'Android HOME project - ant buildAPK target wasn\'t builded.',
  'Android HOME project - ant buildAPK target was builded.'))
{
 return array (1, $this->log_file_path, $this->project_path . $this-
>main_AndroidModuleDirectoryName . '/bin/');
```

Código Fonte 48 – Excerto de código daclasse AndroidAppBuilder (2)

Para terminar, é executado o *target* Ant `buildAPK` que, conforme a aplicação contenha ou não um ecrã de contactos, pode ser invocado com o parâmetro `Dgmaps_api_key`. Após a sua construção, o comando `$cmd` é executado através da função `execCommand`, que recebe como parâmetros o comando a ser executado e as mensagens a serem escritas no ficheiro de *log*, que variam em caso de sucesso ou insucesso do processo de compilação. Em caso de sucesso, é enviado para a classe `ApplicationController` um *array*, no qual a primeira posição indica se o processo de compilação foi executado com sucesso ou não. A segunda posição refere-se à localização do ficheiro de *log* do processo de compilação e a terceira ao caminho do ficheiro APK gerado.

iOS

O processo de compilação de uma aplicação iOS que se encontra implementado na plataforma é significativamente mais simples do que o de uma aplicação Android. Cada passo do procedimento é acompanhado pela escrita de uma mensagem num ficheiro de *log*, que dará indicação se foi executado com sucesso ou não.

O primeiro passo consiste na geração de um nome para atribuir a um diretório de destino, onde se realizarão as seguintes etapas. De seguida é copiado para o mesmo o projeto iOS desenvolvido e criado o ficheiro styles.plist, que contém dados relativos à aparência da aplicação iOS.

Na etapa seguinte é armazenado numa *string* ($modules_plist_content) o conteúdo que irá ser escrito no ficheiro modules.plist. Em simultâneo são realizadas as seguintes tarefas:

- Iterar todos os módulos que a aplicação contém;
- Adicionar à variável `$modules_plist_content` conteúdo específico do módulo;
- Escrever os recursos do módulo, nomeadamente o ficheiro de configuração do mesmo (por exemplo links_config.plist) e imagens de cabeçalho no caso dos ecrãs de Info, FAQ ou Links.

Terminada a iteração dos vários módulos, é finalizado o conteúdo da variável `$modules_plist_content` e a mesma é escrita no ficheiro modules.plist.

O passo seguinte consiste em copiar para o projeto iOS as imagens que este irá utilizar como ícone da aplicação, imagem de fundo e ícones dos módulos que serão apresentados no menu.

Através do comando `sed` é alterado o conteúdo do ficheiro uebHomeiOS-Info.plist, de modo a adicionar à aplicação o seu nome e *package name*.

Para terminar o processo são executados dois comandos, o primeiro para gerar um ficheiro do tipo xarchive e o segundo para gerar um IPA.

```
$cmd = 'cd '.$this->project_path . $this->main_IOSDirectoryName .';
xcodebuild -scheme uebHomeiOS -archivePath '.$application-
>getAppNameSlug().'.xcarchive archive';
if($this->execCommand($cmd,
  'error creating application xcarchive file.',
  'iOS application xcarchive file created successfully.'))
{
$cmd = 'cd '.$this->project_path . $this->main_IOSDirectoryName .';
xcodebuild -exportArchive -exportFormat IPA -archivePath '.$application-
>getAppNameSlug().'.xcarchive -exportPath '.$application-
>getAppNameSlug();
 if($this->execCommand($cmd,
        'error creating application IPA file.',
        'iOS application IPA file created successfully.'))
 {
        return array (1, $this->log_file_path, $this->project_path .
$this->main_IOSDirectoryName . '/' . $application->getAppNameSlug() .
'.ipa');
 }
}
```

Código Fonte 49 – Excerto de código da classe IosAppBuilder

Após construído o comando `xcodebuild`, do qual resultará um ficheiro .xarchive, é invocada a função `execComand`, responsável por executar o comando e escrever no ficheiro de log. Caso o

segundo comando ilustrado no Código Fonte 49 com a função de gerar um ficheiro IPA seja executado com sucesso, é enviado um *array* de retorno para o `ApplicationController`, classe que invocou a função `build` da classe `IosAppBuilder`. O valor da primeira posição deste *array* indica se o processo de compilação foi executado com sucesso ou não. Na segunda posição é enviada a localização do ficheiro de *log* que contém informação detalhada sobre o processo de compilação e na terceira o caminho para o ficheiro IPA gerado.

3.2.5 Conclusão da Implementação

Nesta secção foi relatado todo o processo de desenvolvimento e implementação do projecto, tendo em conta as funcionalidades identificadas e descritas na Análise.

O processo envolveu a criação de projetos e bibliotecas Android, assim como a definição de *targets* Ant que permitissem a sua posterior utilização. Implicou a criação de um projeto iOS com a maioria dos módulos incorporados e também a construção da plataforma *web*, que engloba o processo de compilação de aplicações Android e iOS.

Embora todos os procedimentos tenham sido cuidadosamente planeados, o desenvolvimento da plataforma e das aplicações foi um trabalho demorado e pesado, tendo sido ultrapassadas muitas dificuldades ao longo de todo o processo. O resultado foi a criação de um sistema complexo e robusto que executa as tarefas que se pretendia de forma eficiente.

4 Análise à MobileAppBuilder

Após o processo de análise e implementação da MobileAppBuilder, considerou-se importante realizar alguns testes de performance às aplicações produzidas através da plataforma desenvolvida, bem como inquirir acerca da qualidade da mesma. Assim, no presente capítulo são abordados os métodos utilizados nos testes de performance e na avaliação da plataforma, estando também descritos os respetivos resultados e discussão.

4.1 Testes performance

Com o objetivo de compreender se a performance das aplicações geradas através da plataforma desenvolvida seria superior à das geradas por outras plataformas *online*, foram realizados testes às aplicações, sendo contudo importante referir alguns aspetos que condicionaram a realização dos mesmos. Das plataformas *online* analisadas no capítulo "Estado da Arte", apenas duas permitem descarregar o APK final, necessário para testar as aplicações. Desta forma, somente foram realizados testes comparativos para as plataformas Andromo e Como, que pelo facto de não oferecerem a possibilidade de testar em iOS levaram a que os testes fossem realizados unicamente para Android.

Para o efeito foi então criada uma aplicação Android na plataforma desenvolvida e nas duas plataformas *online* identificadas (Andromo e Como). A aplicação criada foi igual para as três plataformas, tendo-se optado por utilizar somente os ecrãs comuns às mesmas, resultando na inclusão de quatro ecrãs: mapa, RSS, Twitter e *website*.

Para testar a performance das aplicações foram medidos os tempos de arranque da aplicação e de apresentação dos quatro ecrãs no dispositivo. No processo participaram dois indivíduos, o primeiro com a função de arrancar a aplicação e abrir os ecrãs e a segunda a de cronometrar os procedimentos recorrendo ao cronómetro de um dispositivo móvel. Para cada aspecto a testar foram realizadas 15 medições, todas com as aplicações sem *cache*, sendo a mesma apagada antes de cada nova medição. A bateria de testes não é extensa nem extremamente rigorosa, em parte devido a restrições temporais e técnicas.

Pelo facto de o espaço em disco ser para alguns utilizadores um fator importante, antes de abordar em detalhe os resultados dos testes realizados é importante indicar o tamanho das três aplicações produzidas nas diferentes plataformas e dos respetivos APK, que se encontram registados na Tabela 2.

Tabela 2 – Tamanho do APK e da aplicação instalada

Tamanho (Mb)	MobileAppBuilder	Andromo	Como
APK	1.3	2.2	16.3
Aplicação Instalada	3.6	4.82	26.4

Os 26.4Mb utilizados pela aplicação produzida pelo Como podem ser algo inconcebível para o utilizador quando comparados com o espaço ocupado pelas aplicações referentes às outras duas plataformas, tendo em conta principalmente que a aplicação é composta por apenas quatro ecrãs. É possível verificar através da Tabela 2 que a aplicação produzida pela MobileAppBuilder é a que tem um APK de menor tamanho e que necessita de menos espaço em disco para ser instalada, sendo uma vantagem para os utilizadores com preocupação acerca do espaço ocupado nos seus dispositivos.

4.1.1 Arranque da aplicação

Para testar o arranque da aplicação foi cronometrado o tempo desde que é selecionado o ícone da aplicação até ao momento em que a aplicação se encontra disponível para ser utilizada pelo utilizador. A média dos resultados obtidos encontra-se ilustrada no **Error! Reference source not found.**.

A proximidade de valores entre a aplicação produzida através da plataforma desenvolvida no âmbito da presente dissertação de mestrado e a produzida através do Andromo deve-se ao facto de ambas serem completamente nativas, ao invés da Como, que não utiliza tecnologias nativas. A pequena diferença entre as primeiras está relacionada com diferenças no mecanismo utilizado no arranque de aplicações.

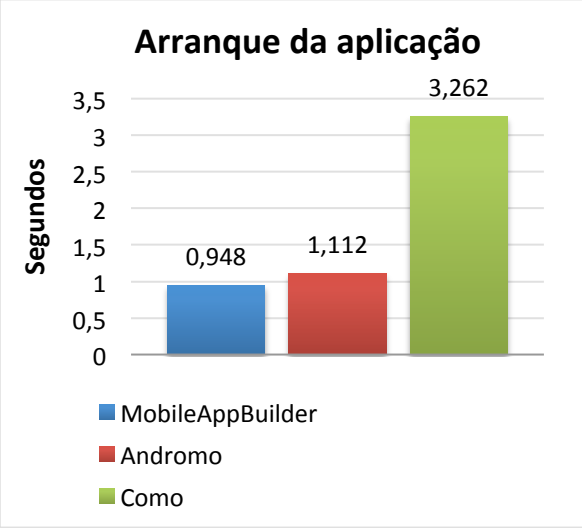


Gráfico 1 – Arranque da aplicação

4.1.2 Apresentar ecrã com mapa

Neste teste foi medido o intervalo de tempo desde o clique no ícone do ecrã de mapas do menu das aplicações produzidas até à apresentação do ecrã com mapa completamente carregado e ponteiro definido estando os resultados indicados no Gráfico 2.

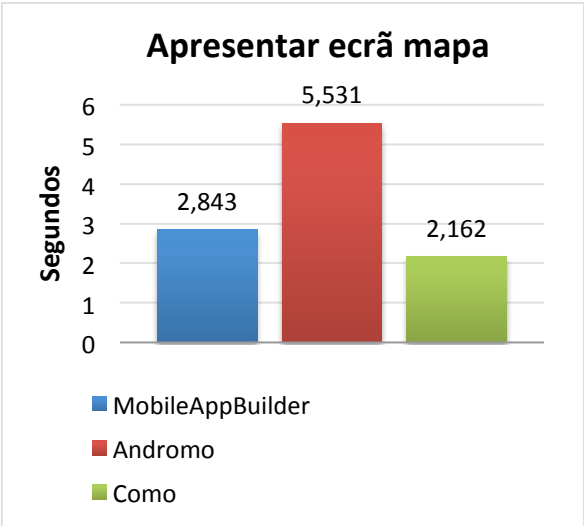


Gráfico 2 – Apresentar ecrã mapa

As discrepâncias observadas entre as plataformas relacionam-se em parte com o facto de na aplicação da MobileAppBuilder ser apresentado um ecrã com um mapa e diferentes *labels* preenchidas, enquanto no mesmo ecrã nas aplicações das outras plataformas apenas é apresentado ao utilizador um mapa.

O fraco resultado obtido pelo Andromo deve-se ao facto da aplicação produzida pelo mesmo não utilizar o Google Maps para apresentar o mapa, mas sim uma outra *framework*.

Relativamente à diferença observada entre os resultados obtidos para as aplicações da MobileAppBuilder e do Como, a mesma está relacionada com o facto de esta última aplicação

apresentar uma vista 2D do mapa, que exige menor computação que o ecrã com uma vista satélite apresentado pela MobileAppBuilder, aliado ao já referido preenchimento de labels que se verifica também para esta aplicação.

4.1.3 Apresentar listagem de feeds RSS

Os resultados apresentados no Gráfico 3 referem-se ao intervalo de tempo entre o clique no ícone de RSS do menu da aplicação e o preenchimento total do espaço de ecrã com os *feeds* carregados de um RSS.

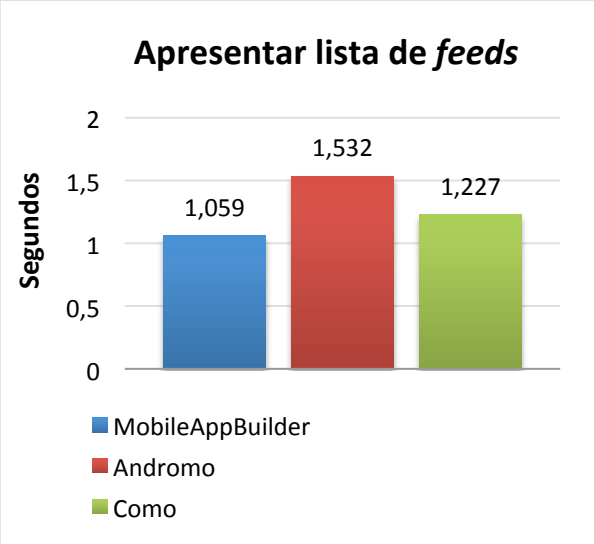


Gráfico 3 – Apresentar lista de feeds

Através da análise dos resultados é possível verificar que a aplicação com melhor desempenho foi a correspondente à MobileAppBuilder, seguindo-se a referente ao Como. Esta desigualdade deve-se ao facto de a aplicação gerada através do Como não tratar diretamente o *feed* RSS, e portanto há a necessidade de definir na plataforma qual o tempo de atualizaçãoo do *feed*, o que significa que o tratamento do RSS é feito pelo servidor e o seu conteúdo é enviado para a aplicação móvel sempre que esta o requisita. Este conceito denomina-se *mobile cloud computing* e na sua base está a utilização de servidores *web* capazes de executar uma parte das tarefas que dispositivos como *smartphones* tenham a necessidade de efetuar, com o intuito de permitir aos mesmos conservar bateria.

4.1.4 Apresentar listagem de tweets

Para este teste foi cronometrado o tempo desde que é selecionado o ícone correspondente à listagem de *tweets* no menu, até à sua visualização na totalidade do ecrã.

Tal como acontece no ecrã com uma listagem de *feeds*, a aplicação do Como obtém melhores resultados (Gráfico 4) o que se deve à utilização de um servidor para efetuar o tratamento dos *tweets* de um determinado utilizador. Relativamente à aplicação produzida pela MobileAppBuilder e Andromo, a diferença observada comparativamente ao ecrã de *feeds* RSS é consequência do processo de autenticação no Twitter.

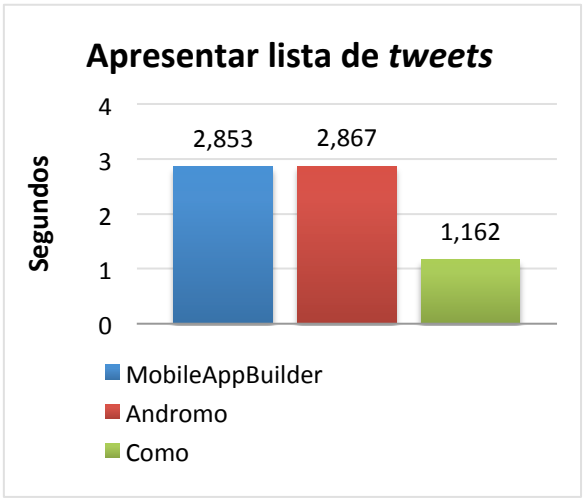


Gráfico 4 – Apresentar lista de tweets

4.1.5 Apresentar página web

Os resultados apresentados no Gráfico 5 referem-se ao período de tempo ocorrido entre a seleção do ícone correspondente ao ecrã *web* no menu da aplicação, até à apresentação completa da página *web*.

Os resultados obtidos neste teste são bastante semelhantes para as três plataformas, o que leva a supor que, tal como a aplicação produzida na MobileAppBuilder, também as outras aplicações testadas utilizam o componente WebView. Contudo, na aplicação produzida através da MobileAppBuilder o componente WebView encontra-se com o Javascript ativo, o que não é possível afirmar para as aplicações produzidas pelo Andromo e Como, fator que pode ser diferenciador na interpretação dos resultados visualizados no Gráfico 5.

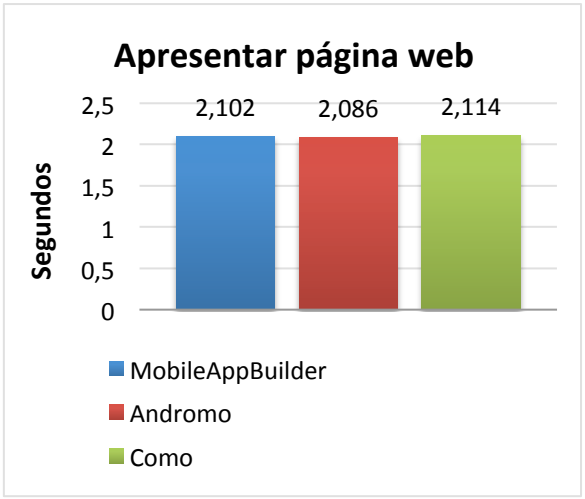


Gráfico 5 – Apresentar página *web*

4.2 Questionário

Neste subcapítulo é apresentada uma compilação dos resultados obtidos no âmbito do questionário Anexo 2 – Questionário efetuado a um pequeno grupo de dez indivíduos formados em Engenharia Informática.

A resposta ao questionário foi precedida pela realização de um exercício (Anexo 3 – Enunciado do Exercício) na plataforma desenvolvida, que supõe a criação de uma aplicação com quatro ecrãs e a configuração de vários detalhes. Terminado o exercício, a aplicação gerada seria instalada num dispositivo móvel Android ou iOS de modo a que o inquirido tivesse a oportunidade de experimentar a aplicação que criou e estar assim melhor preparado para responder ao questionário que lhe seria entregue. O mesmo é composto por 11 perguntas Anexo 2 – e está divido em três partes: plataforma *web*, aplicações produzidas e sugestões.

Compiladas as respostas obtidas relativamente à plataforma *web*, é possível afirmar que 60% dos inquiridos consideraram a linguagem utilizada na plataforma bastante adequada. Metade dos inquiridos considerou a interface da plataforma muito simples e bem estruturada, enquanto 40% considerou a mesma boa. Ainda relativamente à interface, durante a realização do exercício proposto a plataforma nunca *crashou* e as ferramentas de configuração dos aspetos visuais da aplicação foram classificadas como boas para 50% dos inquiridos e excelente para 40%.

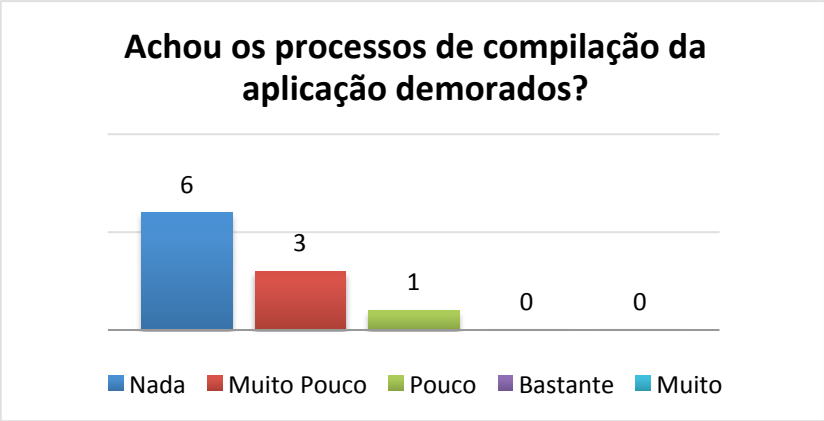


Gráfico 6 – Respostas à pergunta "Achou os processos de compilação da aplicação demorados?"

No Gráfico 6 estão indicados os resultados relativos à opinião dos inquiridos quanto à duração do processo de compilação da aplicação, verificando-se que 60% dos inquiridos considerou o

processo nada demorado, 30% muito pouco demorado e 10% pouco demorado. Assim, é possível concluir que o tempo de compilação é rápido e satisfatório para os utilizadores.

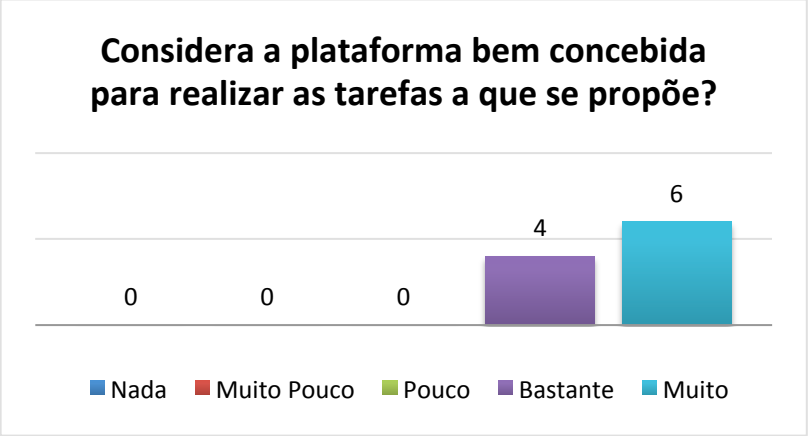


Gráfico 7 – Resposta à pergunta "Considera a plataforma bem concebida para realizar as tarefas a que se propõe?"

Relativamente à capacidade da plataforma para cumprir as tarefas a que se propõe, a resposta dos inquiridos (Gráfico 7) é positiva, com 60% a considerar a plataforma muito bem concebida e 40% bastante bem concebida.

Para terminar a secção do questionário relativa à plataforma *web*, quanto à utilidade da mesma, a maioria dos inquiridos considerou a plataforma muito útil para o desenvolvimento de aplicações móveis, como é possível verificar no Gráfico 8.

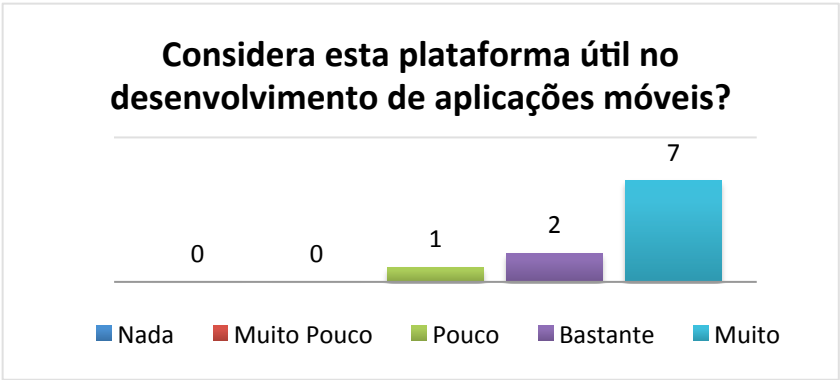


Gráfico 8 – Respostas à pergunta "Considera esta plataforma útil no desenvolvimento de aplicações móveis?"

Relativamente às aplicações geradas foram efetuadas três questões, das quais são analisados os resultados de seguida.

Como é possível verificar no Gráfico 9, grande parte dos inquiridos (80%) considera que o tempo de arranque da aplicação produzida pela plataforma *web* para Android e iOS é excelente, o que constitui um aspecto muito positivo.

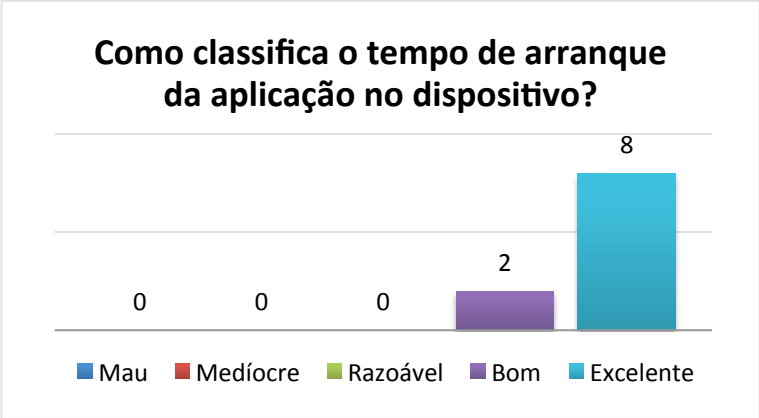


Gráfico 9 – Respostas à questão "Como classifica o tempo de arranque da aplicação no dispositivo?"

Quando ao aspeto visual da aplicação (Gráfico 10), 10% dos inquiridos considerou que este correspondeu pouco ao idealizado aquando da configuração da aplicação através da plataforma *web* desenvolvida, 40% considerou que o aspeto correspondeu bastante e 50% que correspondeu muito ao idealizado.

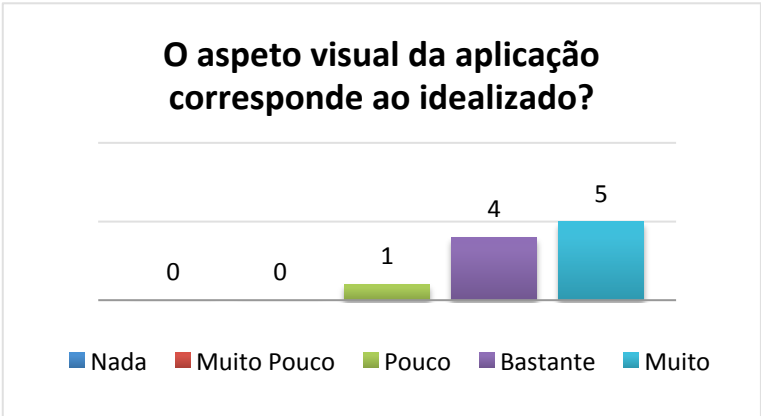


Gráfico 10 – Resposta à pergunta "O aspeto visual da aplicação corresponde ao idealizado?"

A terceira e última pergunta relacionada com as aplicações produzidas objetiva avaliar a sua performance.

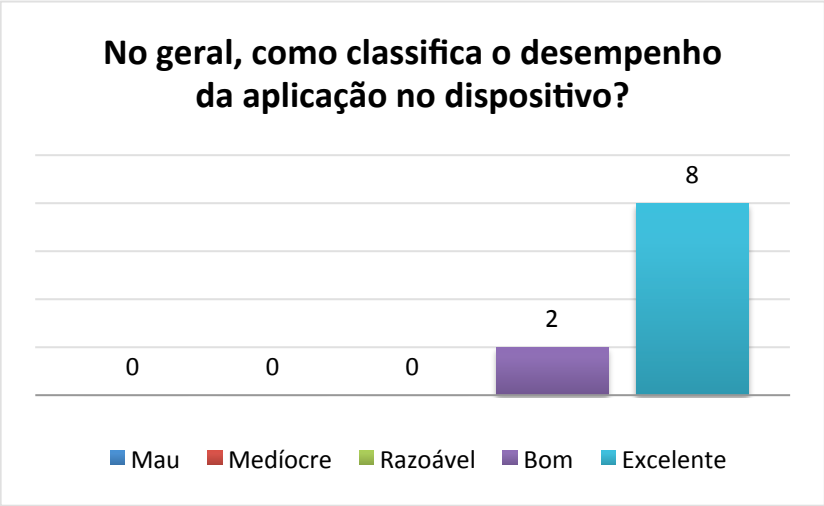


Gráfico 11 – Resposta à pergunta "No geral, como classifica o desempenho da aplicação no dispositivo?"

Os resultados obtidos quanto à classificação do desempenho das aplicações produzidas em dispositivos móveis (Gráfico 11) são bastante positivos, com 80% dos inquiridos a classificar o desempenho das aplicações como excelente.

A última parte do questionário visava a recolha de sugestões e a obtenção de *feedback* que permitisse identificar que trabalho futuro poderia ser desenvolvido. Foi sugerida a incorporação na plataforma de uma pré-visualização da aplicação final durante o processo de configuração dos ecrãs e estilos da aplicação, bem como a possibilidade de adicionar mais ecrãs e selecionar mais tipos de menus para a mesma. Outra sugestão consiste na definição de diferentes temas para a aplicação e conferir ao utilizador da plataforma *web* a possibilidade de escolher para a sua aplicação um dos temas pré-definidos.

5 Conclusões e Trabalho Futuro

Durante a última década o mercado dos dispositivos móveis tem vindo a crescer exponencialmente, captando a atenção da comunidade de programadores e do mundo empresarial. Com este crescimento surgiu o mercado das aplicações móveis, que levou ao aparecimento de diferentes oportunidades e negócios, como é o caso dos criadores *online* de aplicações móveis, plataformas que permitem gerar aplicações para dispositivos móveis com relativa facilidade e rapidez.

Estas plataformas foram objeto de análise no presente estudo e, embora o seu aparecimento seja relativamente recente, são inúmeras as soluções já existentes. No entanto, a generalidade destas ferramentas peca por gerar aplicações híbridas e *web* (apenas uma das plataformas analisadas é capaz de criar aplicações Android nativas), que se caraterizam por apresentar um baixo rendimento e diversas limitações ao nível do uso de *hardware* e interface visual quando comparadas com aplicações nativas.

Assim, uma vez que encontrar uma plataforma *web* capaz de produzir aplicações nativas para Android e iOS é extremamente difícil, foi proposto o desenvolvimento de uma plataforma que pudesse preencher esta lacuna. O desenvolvimento de um sistema com as caraterísticas que se pretendem envolve um conjunto de aspetos que foram amplamente analisados e documentados no decorrer da presente dissertação de mestrado.

Os dispositivos móveis apresentam variadas limitações, nomeadamente ao nível de *hardware* e de capacidade de computação, o que leva a que o desenvolvimento de aplicações para os

mesmos tenha de ser feito de forma extremamente cuidadosa e rigorosa, de forma a oferecer ao utilizador final uma interface gráfica agradável e tempos de resposta aceitáveis.

O resultado final desta dissertação é assim um sistema que permite a geração de aplicações nativas Android e iOS em tempo recorde, quando comparado com o tempo despendido por uma equipa de programadores a trabalhar para desenvolver aplicações semelhantes de raiz. A MobileAppBuilder pode ser utilizada por uma equipa de programadores como ferramenta para desenvolver um projeto base (*kickstart*) que posteriormente possa ser adaptado às necessidades de cada cliente, ou como um criador *online* idêntico a todos aqueles que foram analisados no âmbito da presente dissertação, seja por um programador ou por um indivíduo sem conhecimentos em programação.

Os resultados obtidos nos testes comparativos realizados à aplicação gerada pela MobileAppBuilder demonstram que esta plataforma apresenta caraterísticas que a tornam competitiva relativamente às outras plataformas existentes. Um dos aspetos importantes é o facto da aplicação da MobileAppBuilder ser de menor tamanho do que as produzidas pelas outras plataformas testadas, facto que pode ser importante para diversos utilizadores, pois os dispositivos móveis (*smartphones* e *tablets*) têm um espaço de disco limitado, que quanto mais ocupado está, pior o desempenho do dispositivo. Outro fator importante é a aplicação produzida na MobileAppBuilder apresentar tempos menores de arranque e de apresentação de alguns ecrãs do que as outras produzidas pelas plataformas testadas, o que melhora a experiência do utilizador.

Já o questionário realizado, permitiu reconhecer que a plataforma criada agrada aos utilizadores, com classificações positivas no geral. No entanto, o número de indivíduos inquiridos foi limitado, consequência de constrições temporais. Seria extremamente vantajoso realizar inquéritos a mais indivíduos que cumpram as condições impostas, de modo a obter um maior e melhor *feedback* e assim ir de encontro às necessidades e preferências dos utilizadores.

Embora todos os objetivos tenham sido alcançados, existem melhorias possíveis de realizar no futuro. Seria importante efetuar a implementação em iOS dos módulos de partilha de fotografia e do Twitter, bem como o desenvolvimento de um mecanismo que ofereça a capacidade de produzir aplicações iOS apenas com o código fonte necessário, à semelhança do que acontece com Android, no qual os vários módulos se encontram divididos em

bibliotecas. Por sugestão dos inquiridos na avaliação da plataforma, deveria ser incorporada na mesma uma previsão do ecrã que o utilizador está a configurar, de modo a que este consiga percecionar a aparência da aplicação móvel antes da mesma ser gerada.

Concluindo, penso que a realização da presente dissertação promoveu a aplicação de conhecimentos adquiridos no âmbito das várias unidades curriculares frequentadas, bem como o desenvolvimento de competências no domínio da pesquisa e da escrita, que resultaram num significativo enriquecimento de competências profissionais.

6 Bibliografia

AndroidAppMarket, 2012. [Online]
Disponível em: http://www.android-app-market.com/android-architecture.html
[Acedido em 2 Setembro 2014].

Andromo, 2014. [Online]
Disponível em: http://andromo.com/
[Acedido em 5 Setembro 2014].

Anon., 2014. [Online]
Disponível em: http://www.como.com/about-us/
[Acedido em 5 Setembro 2014].

AppBrain, 2014. [Online]
Disponível em: http://pt.appbrain.com/
[Acedido em 27 Setembro 2014].

Apple, 2010. *Apple Launches iPad.* [Online]
Disponível em: http://www.apple.com/pr/library/2010/01/27Apple-Launches-iPad.html
[Acedido em 12 Outubro 2014].

AppMachine, 2014. [Online]
Disponível em: http://www.appmachine.com/
[Acedido em 5 Setembro 2014].

Brian, 2012. *PhoneGap, Cordova, and what's in a name?.* [Online]
Disponível em: http://phonegap.com/2012/03/19/phonegap-cordova-and-what%E2%80%99s-in-a-name/
[Acedido em 3 Setembro 2014].

Bugzilla-Xamarin, 2014. [Online]
Disponível em: https://bugzilla.xamarin.com/buglist.cgi?classification=Xamarin&chfieldto=2014-08-31&query_format=advanced&chfield=[Bug%20creation]&chfieldfrom=2014-08-01&product=Android
[Acedido em 28 Setembro 2014].

Cnet, 2012. *Why Metro now rules at Microsoft.* [Online]
Disponível em: http://www.cnet.com/news/why-metro-now-rules-at-microsoft/
[Acedido em 12 Outubro 2014].

Cohen, P., 2007. [Online]
Disponível em: http://www.macworld.com/article/1056750/macworldexpo.html
[Acedido em 26 Setembro 2014].

Como, 2014. *Como.* [Online]
Disponível em: http://www.como.com/
[Acedido em 5 Setembro 2014].

Dalrymple, J., 2008. *Apple unveils iPhone SDK.* [Online]
Disponível em: http://www.macworld.com/article/1132400/iphonesdk.html
[Acedido em 12 Outubro 2014].

DeveloperAndroid-1, 2014. *ProGuard.* [Online]
Disponível em: http://developer.android.com/tools/help/proguard.html
[Acedido em 20 Setembro 2014].

DeveloperAndroid-2, 2014. *Google Play Services.* [Online]
Disponível em: https://developer.android.com/google/play-services/index.html
[Acedido em 20 Setembro 2014].

Developer-Apple-1, 2014. *Xcode - Apple developer.* [Online]
Disponível em: https://developer.apple.com/xcode/
[Acedido em 26 Setembro 2014].

DeveloperApple-2, 2014. *About the iOS Technologies.* [Online]
Disponível em:
https://developer.apple.com/library/ios/documentation/miscellaneous/conceptual/iphoneostechoverview/Introduction/Introduction.html
[Acedido em 4 Setembro 2014].

DeveloperApple-3, 2014. [Online]
Disponível em: https://developer.apple.com/swift/
[Acedido em 27 Setembro 2014].

DeveloperApple-4, 2014. *Core OS Layer.* [Online]
Disponível em:
https://developer.apple.com/library/ios/documentation/miscellaneous/conceptual/iphoneostechoverview/CoreOSLayer/CoreOSLayer.html#//apple_ref/doc/uid/TP40007898-CH11-SW1
[Acedido em 4 Setembro 2014].

DeveloperApple-5, 2014. *CoreServicesLayer - Apple Developer.* [Online]
Disponível em:
https://developer.apple.com/library/ios/documentation/miscellaneous/conceptual/iphoneostechoverview/CoreServicesLayer/CoreServicesLayer.html#//apple_ref/doc/uid/TP40007898-CH10-SW5
[Acedido em 3 Setembro 2014].

DeveloperApple-6, 2014. *Media Layer.* [Online]
Disponível em:
https://developer.apple.com/library/ios/documentation/miscellaneous/conceptual/iphoneostechoverview/MediaLayer/MediaLayer.html#//apple_ref/doc/uid/TP40007898-CH9-SW8
[Acedido em 4 Setembro 2014].

DeveloperApple-7, n.d. *Cocoa Touch Layer.* [Online]
Disponível em:
https://developer.apple.com/library/ios/documentation/miscellaneous/conceptual/iphoneos

techoverview/iPhoneOSTechnologies/iPhoneOSTechnologies.html#//apple_ref/doc/uid/TP40007898-CH3-SW1
[Acedido em 4 Setembro 2014].

DeveloperXamarin-1, 2014. *Shared Projects.* [Online]
Disponível em: http://developer.xamarin.com/guides/cross-platform/application_fundamentals/shared_projects/
[Acedido em 28 Setembro 2014].

DeveloperXamarin-2, 2014. *Understanding the Xamarin Mobile Platform.* [Online]
Disponível em: http://developer.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications/part_1_-_understanding_the_xamarin_mobile_platform/
[Acedido em 28 Setembro 2014].

DoctrineProject-1, 2008. *Doctrine 1.0 Released.* [Online]
Disponível em: http://www.doctrine-project.org/2008/09/01/doctrine-1-0-released.html
[Acedido em 20 Setembro 2014].

DoctrineProject-2, 2014. *Object Relational Mapper.* [Online]
Disponível em: http://www.doctrine-project.org/projects/orm.html
[Acedido em 20 Setembro 2014].

DoctrineProject-3, 2014. *Doctrine DBAL.* [Online]
Disponível em: http://docs.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/introduction.html
[Acedido em 20 Setembro 2014].

DojoToolKit, 2014. [Online]
Disponível em: http://dojotoolkit.org/features/mobile
[Acedido em 27 Setembro 2014].

Eclipse, 2014. [Online]
Disponível em: https://www.eclipse.org/
[Acedido em 26 Setembro 2014].

Elgin, B., 2005. [Online]
Disponível em: http://www.webcitation.org/5wk7sIvVb
[Acedido em 2 Setembro 2014].

Environment, T. M. D., 2014. *Titanium Mobile.* [Online]
Disponível em: http://www.appcelerator.com/titanium/
[Acedido em 9 Outubro 2014].

Evans, C., 2014. *YAML Official.* [Online]
Disponível em: http://www.yaml.org/
[Acedido em 20 Setembro 2014].

Famo.us, 2014. [Online]
Disponível em: http://famo.us/
[Acedido em 27 Setembro 2014].

Fried, I., 2009. *Windows 7 to launch October 22.* [Online]
Disponível em: http://www.cnet.com/news/windows-7-to-launch-october-22/
[Acedido em 26 Setembro 2014].

Gartner-1, 2014. *Gartner Says Annual Smartphone Sales Surpassed Sales of Feature Phones for the First Time in 2013.* [Online]
Disponível em: http://www.gartner.com/newsroom/id/2665715
[Acedido em 2 Setembro 2014].

Gartner-2, 2013. *Gartner Says Mobile App Stores Will See Annual Downloads Reach 102 Billion in 2013.* [Online]
Disponível em: http://www.gartner.com/newsroom/id/2592315
[Acedido em 2 Setembro 2014].

Gartner-3, 2009. *Gartner Says Worldwide Smartphone Sales Reached Its Lowest Growth Rate With 3.7 Per Cent Increase in Fourth Quarter of 2008.* [Online]
Disponível em: http://www.gartner.com/newsroom/id/910112
[Acedido em 12 Outubro 2014].

Gartner-4, 2013. *Gartner Recommends a Hybrid Approach for Business-to-Employee Mobile Apps.* [Online]

Disponível em: http://www.gartner.com/newsroom/id/2429815
[Acedido em 3 Setembro 2014].

GoogleAdMob, 2014. *Google-AdMob.* [Online]
Disponível em: https://www.google.com/ads/admob
[Acedido em 27 Setembro 2014].

GoogleAnalytics, 2014. [Online]
Disponível em: http://www.google.com/analytics/mobile/
[Acedido em 27 Setembro 2014].

Gradle, 2014. [Online]
Disponível em: http://www.gradle.org/
[Acedido em 26 Setembro 2014].

Grush, A., 2012. [Online]
Disponível em: http://www.androidauthority.com/ibm-simon-birthday-134255/
[Acedido em 1 Setembro 2014].

HPFactor, 2001. [Online]
Disponível em: http://www.hpcfactor.com/support/windowsce/wce1.asp
[Acedido em 2 Setembro 2014].

IDC, 2013. *Android Pushes Past 80% Market Share While Windows Phone Shipments Leap 156.0% Year Over Year in the Third Quarter.* [Online]
Disponível em: http://www.idc.com/getdoc.jsp?containerId=prUS24442013
[Acedido em 2 Setembro 2014].

JetBains-IDEA, 2014. [Online]
Disponível em: http://www.jetbrains.com/idea/
[Acedido em 26 Setembro 2014].

John, 2014. *Famo.us and AppMachine Meetup Renews Future For Web Apps.* [Online]
Disponível em: http://www.appmachine.com/blog/famo-us-appmachine-meetup-renews-future-web-apps/
[Acedido em 5 Setembro 2014].

JqtJS, 2014. [Online]
Disponível em: http://jqtjs.com/
[Acedido em 27 Setembro 2014].

JQueryMobile, 2014. [Online]
Disponível em: http://jquerymobile.com/
[Acedido em 27 Setembro 2014].

Kilpatrick, R., 2014. *Psion MC 200/400/600.* [Online]
Disponível em: http://www.old-computers.com/museum/computer.asp?c=737&st=1
[Acedido em 12 Outubro 2014].

Kim, T., 2011. *Mobile Roadie surpasses 10 million app downloads.* [Online]
Disponível em: http://blog.mobileroadie.com/2011/09/mobile-roadie-surpasses-10-million-app-downloads/
[Acedido em 5 Setembro 2014].

Litchfield, S., 2011. [Online]
Disponível em:
http://www.allaboutmeego.com/news/item/12584_Nokias_new_strategy_and_struct.php/
[Acedido em 2 Setembro 2014].

MobileRoadie, 2014. [Online]
Disponível em: http://.mobileroadie.com/
[Acedido em 5 Setembro 2014].

MonoProject, 2014. *The Mono Runtime.* [Online]
Disponível em: http://www.mono-project.com/docs/advanced/runtime/
[Acedido em 28 Setembro 2014].

MooTools, 2014. [Online]
Disponível em: http://mootools.net/
[Acedido em 27 Setembro 2014].

Motorola, 2014. [Online]
Disponível em: https://www.motorola.com/us/consumers/about-motorola-

us/About_Motorola-History-Timeline/About_Motorola-History-Timeline.html#1980
[Acedido em 28 Setembro 2014].

Netbeans, 2014. [Online]
Disponível em: https://netbeans.org/
[Acedido em 26 Setembro 2014].

Olanoff, D., 2012. *Mark Zuckerberg: Our Biggest Mistake Was Betting Too Much On HTML5.* [Online]
Disponível em: http://techcrunch.com/2012/09/11/mark-zuckerberg-our-biggest-mistake-with-mobile-was-betting-too-much-on-html5/
[Acedido em 3 Setembro 2014].

OpenHandsetAlliance, 2014. [Online]
Disponível em: http://www.openhandsetalliance.com/android_overview.html
[Acedido em 2 Setembro 2014].

OpenSource-Apple, 2014. [Online]
Disponível em: http://www.opensource.apple.com/
[Acedido em 27 Setembro 2014].

Operating-System, 2004. *Symbian (EPOC) Operating System.* [Online]
Disponível em: http://www.operating-system.org/betriebssystem/_english/bs-epoc.htm
[Acedido em 12 Outubro 2014].

Pastore, S., 2013. *Mobile operating systems and apps development.* Veneza, Italia, s.n.

Patel, N., 2010. *iPhone OS 4 renamed iOS 4, launching June 21 with 1500 new features.* [Online]
Disponível em: http://www.engadget.com/2010/06/07/iphone-os-4-renamed-ios-gets-1500-new-features/
[Acedido em 12 Outubro 2014].

Pereira, L. C. & Silva, M. L., 2009. Arquitetura Android. In: *Android para desenvolvedores.* s.l.:Brasport Livros e Multimédia.

PhoneGap, 2014. [Online]
Disponível em: http://phonegap.com/
[Acedido em 27 Setembro 2014].

Quigley, M., 2010. [Online]
Disponível em: http://www.androidengineer.com/2010/06/using-ant-to-automate-building-android.html
[Acedido em 20 Setembro 2014].

Rafael, P., 2012. *Protegendo seu aplicativo Java com o Proguard.* [Online]
Disponível em: http://www.viamais.net/blog/protegendo-seu-aplicativo-java-com-o-proguard/#.VBzJKRazn2p
[Acedido em 20 Setembro 2014].

ResearchAndMarkets, 2014. *Cross Platform Mobile Development Tools: Market Analysis & Forecast - Third Edition.* [Online]
Disponível em: http://www.researchandmarkets.com/reports/2896434/cross-platform-mobile-development-tools-market
[Acedido em 28 Setembro 2014].

Sencha, 2014. [Online]
Disponível em: http://www.sencha.com/products/touch
[Acedido em 27 Setembro 2014].

Shiels, M., 2003. *A chat with the man behind mobiles.* [Online]
Disponível em: http://news.bbc.co.uk/2/hi/uk_news/2963619.stm
[Acedido em 28 Setembro 2014].

Smalltalk, 2014. [Online]
Disponível em: http://www.smalltalk.org/
[Acedido em 27 Setembro 2014].

Sonmez, J., 2013. *Which Cross Platform Mobile Development Platform Should You Choose?.* [Online]
Disponível em: http://architects.dzone.com/articles/which-cross-platform-mobile
[Acedido em 3 Setembro 2014].

Symfony-1, 2014. *The Release Process.* [Online]
Disponível em: symfony.com/doc/current/contributing/community/releases.html
[Acedido em 20 Setembro 2014].

Symfony-2, 2014. *Symfony and HTTP Fundamentals.* [Online]
Disponível em: http://symfony.com/doc/current/book/http_fundamentals.html
[Acedido em 20 Setembro 2014].

TechnoBuffalo, 2011. *Introduction to iOS development.* [Online]
Disponível em: http://www.technobuffalo.com/2011/03/27/introduction-to-ios-development-anoverview-
[Acedido em 4 Setembro 2014].

TekSapo, 2013. [Online]
Disponível em: http://tek.sapo.pt/tek_mobile/equipamentos/novo_sistema_operativo_da_nokia_chega_a_portu_1328060.html
[Acedido em 2 Setembro 2014].

Tess, 2013. [Online]
Disponível em: http://www.programering.com/a/MjNzADMwATE.html
[Acedido em 2 Setembro 2014].

TheVerge, 2013. [Online]
Disponível em: http://www.theverge.com/2011/12/13/2612736/ios-history-iphone-ipad
[Acedido em 2 Setembro 2014].

Traeg, P., 2014. *Four Ways To Build A Mobile Application, Part 4: Appcelerator Titanium.* [Online]
Disponível em: http://www.smashingmagazine.com/2014/03/10/4-ways-build-mobile-application-part4-appcelerator-titanium/
[Acedido em 3 Setembro 2014].

Twig, 2011. *Twig 1.0.0 released!.* [Online]
Disponível em: http://blog.twig.sensiolabs.org/post/4140267310/twig-1-0-0-released
[Acedido em 20 Setembro 2014].

Virki, T., 2008. *Nokia buys Symbian, opens up smartphone software.* [Online]
Disponível em: http://www.reuters.com/article/2008/06/24/us-symbian-nokia-idUSHEL00651520080624
[Acedido em 12 Outubro 2014].

VisionMobile, 2014. *Developer Economics Q1 2014: State of the Developer Nation,* Londres, Inglaterra: VisionMobile.

Vogel, L., 2013. *Apache Ant - Tutorial.* [Online]
Disponível em: http://www.vogella.com/tutorials/ApacheAnt/article.html
[Acedido em 20 Setembro 2014].

WatBlog, 2010. [Online]
Disponível em: http://www.watblog.com/2010/02/06/symbian-os-now-fully-open-source/
[Acedido em 2 Setembro 2014].

WebKit, 2014. [Online]
Disponível em: https://www.webkit.org/
[Acedido em 26 Setembro 2014].

Whinnery, K., 2012. *Comparing Titanium and PhoneGap.* [Online]
Disponível em: http://www.appcelerator.com/blog/2012/05/comparing-titanium-and-phonegap/
[Acedido em 3 Setembro 2014].

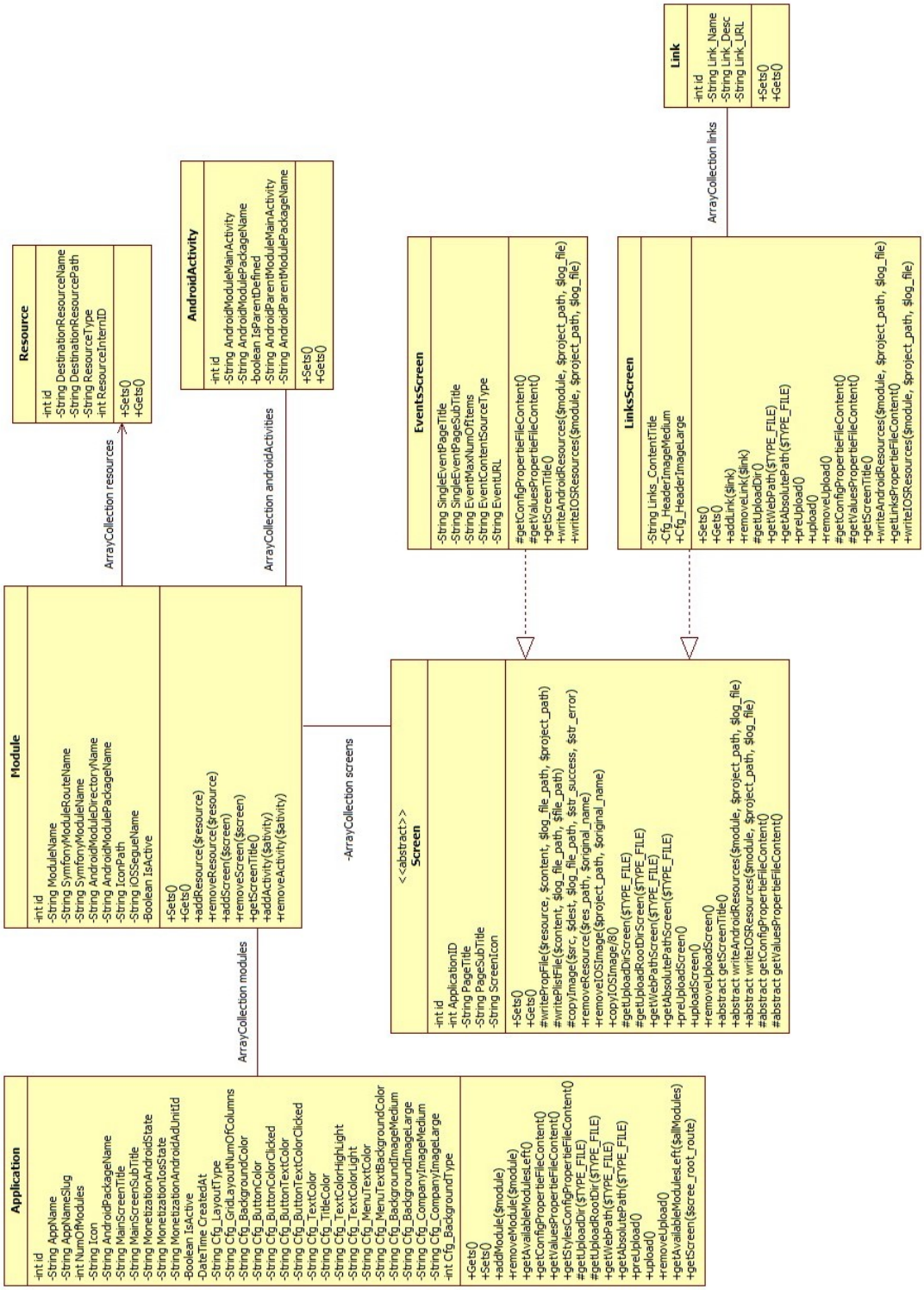
Wolf, F. & Huffstadt, K., 2013. Mobile Enterprise Application Development - a Cross-Platform Framework. *FHWS Science Journal*, Volume 1.

Xamarin, 2014. [Online]
Disponível em: http://xamarin.com/
[Acedido em 28 Setembro 2014].

Zaninotto, F., 2007. *symfony 1.0 released.* [Online]
Disponível em: http://symfony.com/blog/symfony-1-0-released
[Acedido em 20 Setembro 2014].

7 Anexos

7.1 Anexo 1 – Excerto do diagrama de classes do *Website*



7.2 Anexo 2 – Questionário

Departamento de Engenharia Informática	**QUESTIONÁRIO** **- DISSERTAÇÃO MESTRADO – MEI - ISEP -**

Aluno:	Filipe Matos - 1090573	**Orientador:**	Paulo Baltarejo - PBS
Área mestrado:	Arquiteturas, Sistemas e Redes		
Titulo dissertação:	Criador automático de aplicações nativas Android e iOS		

1. Questionário

O preenchimento do presente questionário pressupõe a realização do exercício na plataforma MobileAppBuilder. Assinale com um X o valor da escala que considera mais adequado à avaliação dos itens assinalados.

1 - Nada/Mau | 2 – Muito Pouco/Mediocre | 3 - Pouco/Razoável | 4 - Bastante/Bom | 5 – Muito/Excelente

PLATAFORMA WEB					
	1	2	3	4	5
A linguagem encontrada na plataforma é adequada?					
Considera o *interface* da plataforma simples e bem estruturado?					
O *interface* da plataforma congelou ou crashou?					
Como classificaria a configuração de todos os aspetos visuais da aplicação que desejava?					
Achou os processos de compilação da aplicação demorados?					
Considera a plataforma bem concebida para realizar as tarefas a que se propõe?					
Considera esta plataforma útil no desenvolvimento de aplicações móveis?					
APLICAÇÕES PRODUZIDAS					
	1	2	3	4	5
Como classifica o tempo de arranque da aplicação no dispositivo?					
O aspeto visual da aplicação corresponde ao idealizado?					
No geral, como classifica o desempenho da aplicação no dispositivo?					

1

epartamento de ENGENHARIA INFORMÁTICA Instituto Superior de Engenharia do Porto	**QUESTIONÁRIO** **- DISSERTAÇÃO MESTRADO – MEI - ISEP -**

SUGESTÕES

Nome (facultativo):	

7.3 Anexo 3 – Enunciado do Exercício

Departamento de Engenharia Informática	**EXERCÍCIO** **- DISSERTAÇÃO MESTRADO – MEI - ISEP -**

Aluno:	Filipe Matos - 1090573	**Orientador:**	Paulo Baltarejo - PBS
Área mestrado:	Arquiteturas, Sistemas e Redes		
Título dissertação:	Criador automático de aplicações nativas Android e iOS		

1. Plataforma MobileAppBuilder

Previamente à apresentação do enunciado do exercício a realizar é descrita de forma sucinta a plataforma web denominada MobileAppBuilder, sendo abordados conceitos importantes para a resolução do mesmo.

O MobileAppBuilder é uma plataforma web que permite a criação de aplicações nativas Android e iOS sem haver a necessidade de escrever uma única linha de código.

A plataforma encontra-se dividida em quatro separadores, representando cada um deles uma etapa do processo de criação de uma aplicação móvel.

Para o orientar antes da realização do exercício, encontra-se descrita cada uma das seções:

- **Application Info** – permite alterar o nome, *package name* e ícone da aplicação. Contem a listagem dos ecrãs que fazem parte da aplicação e possibilita ordenar a plataforma a efetuar o processo de produção da aplicação para Android e/ou iOS.
- **Screens** – permite adicionar ecrãs à aplicação e configurar alguns parâmetros, como o título do ecrã e subtítulo (comuns a todos os ecrãs), entre outros específicos de cada ecrã.
- **Styles** – oferece a capacidade de editar os aspetos visuais da aplicação, como o *layout* do menu principal, cor ou imagem de fundo, cor de texto, entre outros.
- **Monetization**– possibilita a configuração de anúncios publicitários na aplicação Android e iOS.

Dentro de cada separador, após efetuar as alterações necessárias e antes de passar para o separador seguinte, deve clicar no botão "Save" que se encontra no fundo da página, de modo a gravar as alterações efectuadas.

2. Enunciado do Exercício

O enunciado do exercício a realizar encontra-se descrito de seguida.

Crie uma aplicação na plataforma MobileAppBuilder com o nome "ISEP", com o *package name* designado pt.ipp.isep.dei e com um ícone à sua escolha.

Adicione um mínimo de quatro ecrãs à aplicação e configure os parâmetros de cada um de acordo com as suas preferências.

Ative a opção de publicidade para a aplicação Androide introduza o Ad Unit Id "ca-app-pub-5057887816328772/6984244444". Indique se deseja ou não publicidade na aplicação iOS.

Configure os estilos da aplicação livremente, excepto a cor dos títulos do texto, que deve ser cor de laranja.

Para terminar, dê indicação à plataforma para gerar a sua aplicação para Android e iOS.