Mestrado em Engenharia Informática

Dissertação

Relatório Final

SoFly goes mobile

Diogo Manuel Ferreira Mendonça

dmfm@student.dei.uc.pt

Orientador do departamento:

Fernando Barros

Orientador da empresa:

Virgílio Esteves

Data: 6 de Julho de 2015

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Mestrado em Engenharia Informática

Dissertação

Relatório Final

SoFly goes mobile

Diogo Manuel Ferreira Mendonça

dmfm@student.dei.uc.pt

Orientador do departamento:

Fernando Barros

Orientador da empresa:

Virgílio Esteves

Data: 6 de Julho de 2015

Elementos do júri:

Júri Arguente: Luís Filipe Vieira Cordeiro

Júri Vogal: Jorge Miguel Sá Silva

Resumo em Português

O presente documento descreve as atividades desenvolvidas no Estágio Curricular integrado no Mestrado de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

A BroadScope tem em desenvolvimento uma aplicação web para empresas que permite recolher informação sobre a sua influência nas redes sociais assim como os principais intervenientes nesta.

Foi objetivo do estágio desenvolver uma solução para *tablets* com as mesmas funcionalidades da aplicação web. O projeto permitiu estudar quais as melhores soluções para este problema da empresa e, no fim, criar uma aplicação modular que respondeu aos requisitos impostos pela empresa. Foram também criados componentes que permitem ter uma experiência visual semelhante à que se encontra na versão web.

Resumo em Inglês

The current document describes the activities that took place during the Curricular Internship that is part of the Master's Degree in Informatics Engineering of the Science and Technology Faculty of the University of Coimbra.

BroadScope is currently developing a web application for companies that allows them to collect information regarding their influence on social networks as well as the main players.

This internship's objective was to develop a solution for tablets with the same features as the web application. This project encompassed studying the best solutions to the problem and in the end creating a modular application while respecting the requirements demanded by the company. Furthermore, additional components were created to mimic a visual experience similar to the web applications'.

Índice

Termos e Acrónimos

API	*Application Programming Interface* (Interface de Programação de Aplicações)
CDN	*Content Delivery Network* (Rede de Fornecimento de Conteúdo)
CLI	*Command Line Interface* (Linha de Comandos)
CRUD	*Create, Retrieve, Update, Delete* (Criar, Reaver, Atualizar, Remover)
CSS	*Cascading Style Sheets* (Folhas de Estilos)
CTO	*Chief Technology Officer* (Director-Chefe Tecnológico)
CTP	*Community Technology Preview*
DEI	Departamento de Engenharia Informática
GUI	*Graphical User Interface* (Interface Gráfica do Utilizador)
HTML	*HyperText Markup Language*
JIT	*Just-In-Time* (Tradução Dinâmica)
JSON	*JavaScript Object Notation*
KPI	*Key Performance Indicator* (Indicador-Chave de Desempenho)
MBaaS	*Mobile Backend as a Service* (Backend Móvel como Serviço)
MVC	*Model View Controller* (Modelo Vista Controlador)
MVP	*Model View Presenter* (Modelo Vista Presenter)
MVVM	*Model View ViewModel* (Modelo Vista ViewModel)
PCL	*Portable Class Library*
Runtime	Tempo de Execução de Programa
SASS	*Syntatically Awesome Style Sheets*
SDK	*Software Development Kit* (Kit de Desenvolvimento de Software)
SMS	*Short Message Service* (Serviço de Mensagens Curtas)
SQL	*Structured Query Language* (Linguagem de Consulta Estruturada)
Tombstoning	Estado preservado da aplicação quando se navega para fora desta
UI	*User Interface* (Interface de Utilizador)
XIB	*XCode Interface Builder*
XML	*eXtensible Markup Language*

1 Informação Geral

1.1 Motivação

O trabalho descrito neste relatório foi desenvolvido no âmbito do estágio curricular para obtenção do grau de Mestre em Engenharia Informática pela Universidade de Coimbra nas instalações da empresa BroadScope.

A BroadScope encontra-se de momento a desenvolver uma solução própria denominada SoFly Analytics. Esta solução visa permitir reunir informações das diversas redes sociais e analisá-las em tempo quase real, permitindo obter *feedback* em tempo útil.

Atualmente e em simultâneo também se encontra a ser desenvolvido por outro estagiário do DEI parte desta solução.

1.2 Objetivos do Estágio

É objetivo deste estágio estudar, especificar e desenvolver uma aplicação móvel para *tablets* de suporte a este novo produto da empresa. A aplicação terá de respeitar algumas característica, requisitos gerais e específicos já definidos pela empresa assim como outros que venham a ser aplicados durante o desenvolvimento do produto em si.

Na primeira fase deste estágio pretende-se estudar as várias soluções e tecnologias existentes para a criação da aplicação em questão, através de investigação e criação de provas de conceito, de modo a facilitar a escolha sobre a solução que mais valor trará para a empresa, tanto ao nível de funcionalidades como à satisfação das condições requeridas.

Numa segunda fase, estando ciente das melhores soluções para o problema em questão, pretende-se proceder ao desenvolvimento da referida aplicação respeitando os requisitos definidos até ao momento.

1.3 Agradecimentos

Este trabalho não seria possível sem o apoio prestado pela fantástica equipa da BroadScope que, ao longo deste processo, sempre se esforçou por me ajudar e disponibilizar os recursos necessários para este trabalho.

Gostaria também de mencionar e agradecer o acompanhamento e preocupação com o meu trabalho demonstradas pelo Doutor Fernando Barros durante a duração do estágio.

1.4 Organização do documento

Este documento encontra-se dividido em vários capítulos de forma a facilitar a sua leitura e a proporcionar uma sequência lógica de informação.

O presente documento encontra-se estruturado com os seguintes capítulos:

1. Informação Geral

2. Gestão do Projeto

3. Projeto

4. Desenvolvimento

5. Conclusão e Trabalho Futuro

6. Referência

Este primeiro capítulo, no qual nos encontramos, contém a introdução ao estágio assim como informações gerais sobre o projeto a realizar.

No segundo capítulo são abordados em primeiro lugar os detalhes referentes à gestão do projeto, tais como a equipa, metodologias utilizadas e planeamento deste.

O terceiro capítulo centra-se na preparação realizada antes do desenvolvimento do projeto propriamente dito. Nesta parte são descritas as tecnologias estudadas e os testes realizados sobre estas, bem como os requisitos que necessitam de respeitar. São também explicitadas quaisquer dificuldades encontradas e a sua resolução. Finalmente, são apresentados os resultados dos testes e as conclusões destes.

Já o quarto capítulo engloba todo o trabalho realizado na segunda metade do estágio, ou seja, o desenvolvimento do produto. São referidas as metodologias e padrões de desenvolvimento utilizados assim como a arquitetura da solução criada e as especificidades desta. Também são mencionados os desafios encontrados neste ponto do trabalho e a solução encontrada para estes, quando possível.

No quinto capítulo é feita uma conclusão e análise retrospetiva ao trabalho realizado durante o estágio e é feita uma breve referência sobre o que se espera ser o trabalho futuro deste projeto findo o período de estágio.

Por fim, o sexto capítulo contém as referências consultadas durante o período de estágio, até ao presente momento.

2 Gestão do Projeto

2.1 Equipa

Fizeram parte da equipa do projeto deste estágio o coordenador da empresa, que também ocupa o papel de CTO desta, assim como o estagiário.

2.2 Coordenação e Reuniões

Durante o estágio existiram dois conjuntos distintos de reuniões entre os diversos intervenientes deste. Por um lado, e devido à metodologia de desenvolvimento interno da empresa, existiram sempre reuniões diárias entre o estagiário e o coordenador da empresa onde se discutia o trabalho realizado e a realizar, tal como eram discutidos quaisquer pontos específicos ou dúvidas sobre estes; por outro lado, existiram também reuniões com uma periodicidade aproximadamente mensal entre o estagiário e ambos os coordenadores (da empresa e do departamento), onde era discutido também ao pormenor o trabalho realizado até ao momento e a realizar futuramente (estas reuniões também serviram para receber *feedback* e recomendações do coordenador do DEI).

Além das reuniões acima mencionadas e com uma periodicidade de duas semanas, foram elaborados e entregues relatórios do tipo 15/5 a ambos os coordenadores. Estes relatórios seguiram a estrutura recomendada pela plataforma [1] do DEI, sendo que cada um deveria conter uma descrição sobre o trabalho feito nas duas semanas anteriores à data do relatório, quaisquer desvios que ocorreram face ao planeamento apresentado para essas duas semanas, bem como o trabalho planeado para as duas semanas seguintes.

2.3 Metodologia

O estágio (e estagiário) também seguiu os métodos de desenvolvimento de software adotados pela empresa. Fazendo parte do grupo de métodos de desenvolvimento ágeis, a empresa emprega uma versão mais simplificada da metodologia *Scrum* [2]. Esta metodologia é caracterizada principalmente por se focar em pequenas iterações sobre o produto e, devido a isto, ser facilmente adaptável a alterações que possam ocorrer, tanto nos objetivos como nas tarefas a realizar pelos elementos da equipa.

O *Scrum* é baseado em *sprints* que representam períodos de tempo regulares e constantes em que se realizam evoluções sobre o produto. Além dos *sprints*, as reuniões diárias de ponto de situação são o outro facto importante da aplicação da metodologia. Diariamente a equipa de desenvolvimento reúne-se durante pouco tempo, tipicamente limitada temporalmente por um período de cinco minutos, onde cada participante descreve o que fez no dia anterior, o

que irá fazer no dia corrente e também tenta identificar quaisquer impedimentos que possam existir e que afetem este novo planeamento. Adicionalmente, no fim de cada dia de trabalho, é atualizado na ferramenta de gestão de projeto o trabalho realizado pelo membro da equipa nesse dia, isto é, é reportado no sistema de controlo de evolução das tarefas o tempo despendido e em que é que esse tempo foi efetivamente gasto.

2.4 Planeamento

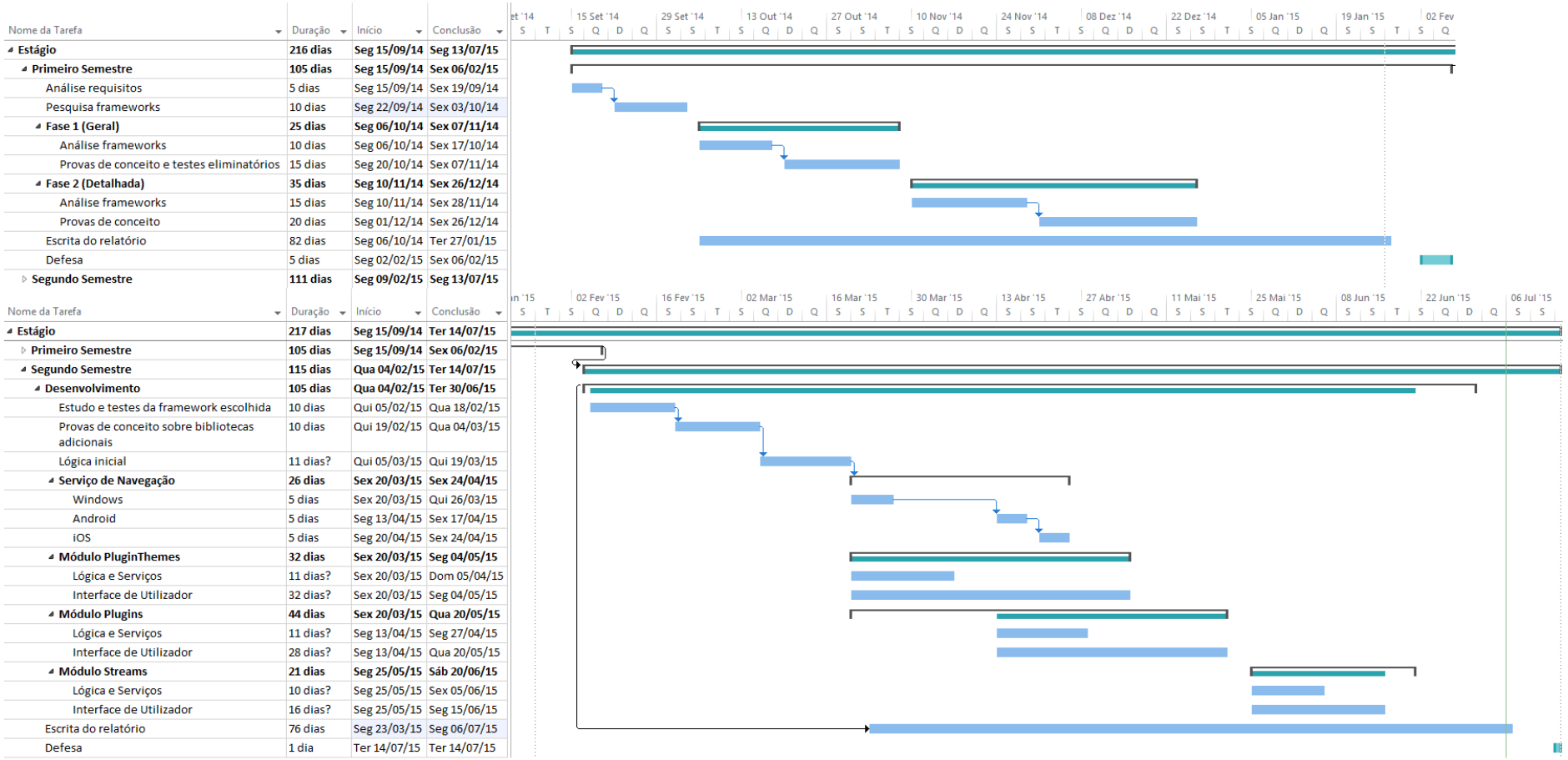
Como se pode ver pelo diagrama de *Gantt* que se apresenta de seguida (Figura 1), o primeiro semestre de trabalho do estágio foi dividido em três secções. A primeira foi caracterizada principalmente pela pesquisa das soluções existentes para o problema, sendo esta já influenciada pela existência de alguns requisitos encontrados (ou já transmitidos ao estagiário).

A segunda fase incluiu o estudo detalhado das *frameworks* encontradas na fase anterior e a aprovação ou reprovação destas, através da aplicação dos critérios definidos pelos requisitos.

Por último, a terceira fase (e a mais trabalhosa) envolveu toda a análise detalhada sobre as *frameworks* mais promissoras e os testes funcionais e provas de conceito que foram criados sobre estas.

Relativamente aos trabalhos a realizados durante o segundo semestre, o plano delineado no primeiro semestre foi maioritariamente respeitado. Como é normal nas etapas de desenvolvimento de um produto, é comum encontrarem-se problemas inesperados que podem alterar até certo ponto o planeamento efetuado. O risco de tal acontecer neste caso era relativamente superior, dado que o produto principal ainda se encontra em desenvolvimento. No entanto, a metodologia ágil implementada pela empresa foi um facto abonatório face aos imprevistos encontrados no decorrer da segunda metade do estágio.

Inicialmente e após algum estudo e planeamento inicial sobre como melhor desenvolver e estruturar a aplicação, começou-se pela implementação dos módulos Plugin e PluginThemes nas três plataformas. Após esta fase inicial houve uma grande restruturação da API que levou a alterações profundas na lógica da aplicação. Finda esta parte, retomou-se o desenvolvimento do módulo final sobre as Streams.



Nome da Tarefa	Duração	Início	Conclusão
Estágio	**216 dias**	**Seg 15/09/14**	**Seg 13/07/15**
Primeiro Semestre	**105 dias**	**Seg 15/09/14**	**Sex 06/02/15**
Análise requisitos	5 dias	Seg 15/09/14	Sex 19/09/14
Pesquisa frameworks	10 dias	Seg 22/09/14	Sex 03/10/14
Fase 1 (Geral)	**25 dias**	**Seg 06/10/14**	**Sex 07/11/14**
Análise frameworks	10 dias	Seg 06/10/14	Sex 17/10/14
Provas de conceito e testes eliminatórios	15 dias	Seg 20/10/14	Sex 07/11/14
Fase 2 (Detalhada)	**35 dias**	**Seg 10/11/14**	**Sex 26/12/14**
Análise frameworks	15 dias	Seg 10/11/14	Sex 28/11/14
Provas de conceito	20 dias	Seg 01/12/14	Sex 26/12/14
Escrita do relatório	82 dias	Seg 06/10/14	Ter 27/01/15
Defesa	5 dias	Seg 02/02/15	Sex 06/02/15
Segundo Semestre	**111 dias**	**Seg 09/02/15**	**Seg 13/07/15**

Nome da Tarefa	Duração	Início	Conclusão
Estágio	**217 dias**	**Seg 15/09/14**	**Ter 14/07/15**
Primeiro Semestre	**105 dias**	**Seg 15/09/14**	**Sex 06/02/15**
Segundo Semestre	**115 dias**	**Qua 04/02/15**	**Ter 14/07/15**
Desenvolvimento	**105 dias**	**Qua 04/02/15**	**Ter 30/06/15**
Estudo e testes da framework escolhida	10 dias	Qui 05/02/15	Qua 18/02/15
Provas de conceito sobre bibliotecas adicionais	10 dias	Qui 19/02/15	Qua 04/03/15
Lógica inicial	11 dias?	Qui 05/03/15	Qui 19/03/15
Serviço de Navegação	**26 dias**	**Sex 20/03/15**	**Sex 24/04/15**
Windows	5 dias	Sex 20/03/15	Qui 26/03/15
Android	5 dias	Seg 13/04/15	Sex 17/04/15
iOS	5 dias	Seg 20/04/15	Sex 24/04/15
Módulo PluginThemes	**32 dias**	**Sex 20/03/15**	**Seg 04/05/15**
Lógica e Serviços	11 dias?	Sex 20/03/15	Dom 05/04/15
Interface de Utilizador	32 dias?	Sex 20/03/15	Seg 04/05/15
Módulo Plugins	**44 dias**	**Sex 20/03/15**	**Qua 20/05/15**
Lógica e Serviços	11 dias?	Seg 13/04/15	Seg 27/04/15
Interface de Utilizador	28 dias?	Seg 13/04/15	Qua 20/05/15
Módulo Streams	**21 dias**	**Seg 25/05/15**	**Sáb 20/06/15**
Lógica e Serviços	10 dias?	Seg 25/05/15	Sex 05/06/15
Interface de Utilizador	16 dias?	Seg 25/05/15	Seg 15/06/15
Escrita do relatório	76 dias	Seg 23/03/15	Seg 06/07/15
Defesa	1 dia	Ter 14/07/15	Ter 14/07/15

FIGURA 1 – DIAGRAMA DE GANTT DO TRABALHO REALIZADO

3 Projeto

A Empresa

Este estágio curricular resulta da proposta feita pela BroadScope, empresa que iniciou atividade em 2012 e se encontra sediada na Incubadora do Instituto Pedro Nunes em Coimbra. Esta é constituída por uma equipa jovem e multidisciplinar com especial enfoque em tecnologias Microsoft®. A empresa realiza trabalhos nas áreas de soluções web, aplicações distribuídas, móveis e de *desktop*, tendo a qualidade dos seus colaboradores sido comprovada e reconhecida através da atribuição de prémios e certificações internacionais tais como Microsoft® Certified Professional, Microsoft® Certified Technology Specialist e Microsoft® Most Valuable Professional, assim como o leque de clientes onde se incluem a Microsoft®, Indra®, Leadership Business Consulting, Qolpac, entre outros.

Apesar de a BroadScope ser uma consultora tecnológica, também já deu origem a uma segunda empresa denominada SoFly, nome este que é partilhado com o produto criado pela mesma.

O Produto

O SoFly Analytics, ou apenas SoFly como diminutivo, na sua fase atual, é uma aplicação web em ASP.NET MVC em C# (*C Sharp*) para o segmento empresarial que, fazendo uso de informação recolhida em várias redes sociais e através da aplicação de algoritmos proprietários, consegue proporcionar ao cliente uma visão mais específica e um conjunto de métricas sobre o alcance e influência que este tem para com o seu público-alvo. A plataforma pretende oferecer a grupos de *media* e marketing uma melhor maneira de compreenderem as suas audiências através da análise das informações presentes nas redes sociais, de forma a guiarem o conteúdo das mesmas, ou seja, permitir a estas equipas aprender, adaptar e responder de um aforma ágil e rápida. Esta solução também visa permitir orientar as temáticas disponíveis e discutidas nas redes sociais.

Como o público-alvo e as diferentes interações estão espalhadas por várias redes sociais, não é prático nem fiável e muito menos eficiente controlá-los, organizá-los e analisá-los manualmente. O SoFly pretende permitir aos seus clientes compreender melhor este público-alvo através de uma análise automática levando o cliente a tomar decisões de negócio melhor fundamentadas.

Durante o período de estágio foram desenvolvidos para a aplicação web (com a respetiva web API) três módulos com diferentes funcionalidades mas com forte ligação entre eles. Cada um destes módulos permite fazer a gestão sobre um tipo de objeto de negócio: Stream, Plugin e PluginTheme. A Stream representa uma fonte de dados de uma rede social, podendo ser criada com base numa página do Facebook (inclui *posts*, comentários, *likes*, *shares* e *views*), ou em outras fontes de dados de outras redes sociais tais como Twitter, Instagram, entre outras. O Plugin é uma representação gráfica de informação visual que o cliente pode criar

para interagir ou visualizar o conteúdo de uma Stream. O PluginTheme é a customização visual que é permitida aplicar sobre um Plugin (podem ser reutilizadas em vários Plugins).

Na fase inicial em que o produto se encontra, apenas é permitida a criação de Streams sobre páginas do Facebook. A criação de Plugins também se encontra limitada a um tipo, que representa um *post* do Facebook com toda a informação associada a este: comentários, *likes*, imagem, texto, assim como a possibilidade de o utilizador comentar. Após a sua criação e configuração, é apresentado ao cliente uma pequena quantidade de código HTML e JavaScript que este pode utilizar para apresentar o Plugin em qualquer website seu.

Trabalho a Realizar

Estando nós atualmente na era de dispositivos móveis, é necessário para a empresa apresentar uma solução móvel para *tablets* a par com a aplicação web para *desktop*, e foi este o desafio colocado ao estagiário.

O produto a ser desenvolvido no momento já possui uma web API que se encontra em desenvolvimento, sendo que faz apenas parte do projeto de estágio a criação da aplicação móvel respeitando as limitações e requisitos impostos.

De seguida, é apresentada uma imagem (Figura 2) que ilustra a arquitetura da solução a desenvolver. É então possível observar que o trabalho do aluno está representado na área "Mobile Solutions".

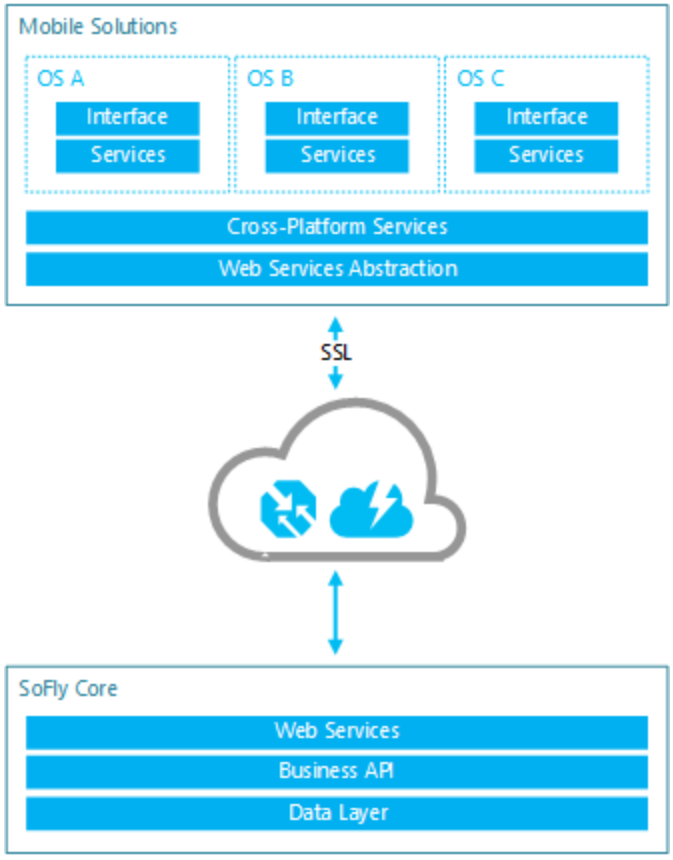


FIGURA 2 – ARQUITETURA DA SOLUÇÃO E CORE DO PRODUTO

Começando pelo *core* da solução, esta representação foi criada de forma simplista pois, além de se encontrar fora do âmbito deste documento (assim como a secção de acesso ao SoFly Core), é constituída por muitos outros serviços que não influenciam o projeto de estágio.

O SoFly Core é constituído então por uma camada de dados (*Data Layer*) que trata de gerir tudo o que está relacionado com a persistência e acesso a dados (controlo de ligações, replicação, cópias de segurança, validação, entre outras funcionalidades). No nível seguinte encontra-se a API de negócio (*Business* API). Esta é responsável pelas transformações das entidades em objetos de negócio e vice-versa, assim como da validação destes e camada de negócio associada. Por último, temos a camada de serviços web (Web *Services*) que trata de expor estes objetos e outra informação a aplicações que os queiram consumir, com toda a lógica associada (incluindo autenticação e encriptação de dados). É com esta camada, normalmente referida por API, que a aplicação a desenvolver interage.

Ao centro da figura temos a secção de acesso ao SoFly Core que, no nosso caso, também inclui um gestor de tráfego (*traffic manager*) e uma CDN. O primeiro permite, por exemplo, encaminhar os pedidos dos utilizadores para um servidor mais próximo destes, assim como possui também capacidades de *failover*, entre outras características. O segundo é constituído por um grande grupo de servidores espalhados geograficamente cujo objetivo é a distribuição de conteúdo estático aos utilizadores, focando-se na elevada disponibilidade e performance.

No topo da imagem temos a representação de alto nível do que é pretendido criar neste estágio: a solução móvel do produto SoFly Analytics. A partilha das camadas de serviços web (Web *Services*) e serviços multi-plataforma (*Cross-Platform Services*) vem reforçar a escolha de uma *framework* que possibilite a reutilização de código entre diversos sistemas operativos (representados por A, B e C).

Começando pela primeira das duas camadas multi-plataforma, a camada de serviços web será responsável pela realização e gestão dos pedidos aos serviços web do SoFly Core. A camada dos serviços multi-plataforma será responsável por toda a parte da lógica de negócio que possa existir associada ao produto a criar, bem como abstrair, sempre que possível, a realização de tarefas comuns aos diversos sistemas operativos.

Posteriormente, cada sistema operativo móvel terá serviços dedicados (*Services*), os quais incluem acesso às APIs e funcionalidades nativas do dispositivo em questão e interface de utilizador (*Interface*) própria.

3.1 Requisitos e Casos de Uso

Antes de se iniciar o trabalho, haviam sido já definidos por parte da empresa um conjunto de requisitos funcionais que a solução a criar terá de respeitar:

- Terá de ser possível ao utilizador gerir os PluginThemes associados à sua subscrição. Esta gestão engloba a listagem, criação, edição e eliminação destes.

- O utilizador, durante a edição ou criação de um PluginTheme, tem de ser apresentado com feedback visual das alterações e configurações que escolher para este.

- O utilizador não poderá apagar um PluginTheme se este estiver a ser usado por um Plugin em funcionamento.

- Terá de ser possível ao utilizador gerir os Plugins associados à sua subscrição. Esta gestão engloba a listagem, criação, edição e eliminação destes.

- Durante a configuração de um Plugin (criação ou edição) o utilizador deve poder observar o resultado visual das suas escolhas no que toca às configurações possíveis.

- Terá de ser possível ao utilizador gerir as Streams associados à sua subscrição. Esta gestão engloba a listagem, criação, edição, visualização e eliminação destas.

- Ao ver os detalhes de uma Stream, deve ser mostrado ao utilizador todo o conteúdo recolhido sobre esta para que este possa observar os resultados das análises efetuadas.

- Encomendas apagadas pelo utilizador permanecerão numa lista à parte durante um certo período de tempo até serem automaticamente eliminadas ou o utilizador, manualmente, as altere para o estado Ativo ou *Purged*.

Relativamente aos casos de uso que a aplicação deve respeitar, foram definidos os seguintes:

- Um utilizador que pretende usar a aplicação num *tablet* e espera obter a mesma experiência e especto familiar que a utilização através do computador proporcionou.

- Voltando a usar a aplicação, o utilizador espera que esta se lembre de informação específica relativamente a si mesmo, tais como os seus dados de autenticação, preferências de apresentação da informação, entre outros.

- Se, em qualquer momento do ciclo de vida/execução/funcionamento da aplicação, a conectividade aos serviços for inexistente ou falhar, esta situação deverá ser transparente (ou causar o mínimo impacto) à experiência do utilizador e a aplicação deverá tentar recuperar e retomar a ligação aos serviços sem a intervenção deste.

Com vista aos módulos atuais e futuros que venham a ser implementados, foram definidos os seguintes requisitos do produto (deve-se notar que nem todos estes desafios podem/devem ser resolvidos nesta parte do estágio.):

- Dado que se trata de uma aplicação de *analytics* em tempo real, comunicação constante com os serviços centrais do produto é uma obrigatoriedade. No entanto, a existência de uma ligação à internet por parte de um dispositivo móvel nem sempre é garantida. Podem ocorrer falhas de cobertura, tanto da rede *Wi-Fi* como da rede de dados, assim como pode esgotar-

se o saldo de dados disponível. É então necessário que a aplicação saiba responder a estas perdas de conectividade da maneira correta: notificações ao utilizador, saiba realizar degradação controlada e manter as funcionalidades que ainda forem possíveis, tentar restabelecer a conexão aos serviços e suportar a restituição dos dados no período em que a ligação falhou.

- Com a quantidade de dados que se pretende mostrar, será necessária uma maneira de armazenar estes, quer para futuras consultas (suportar histórico), quer para auxiliar a recuperação destes em caso de falha temporária da ligação. Também será necessário salvaguardar dados do utilizador assim como quaisquer dados de configuração da aplicação.

- A quantidade de diferentes dispositivos móveis existentes no mercado leva a que, querendo suportar a maioria destes, seja necessário encontrar uma solução que não só facilmente se adapte a vários tamanhos de ecrã como suporte diferentes sistemas operativos móveis. É então necessário encontrar uma solução que ajuste a disposição dos elementos da aplicação consoante o tamanho ou orientação do dispositivo, exibindo, alterando ou removendo componentes do ecrã. Ao ter de suportar os três sistemas operativos móveis, é de extrema importância que a solução escolhida seja capaz de partilha de código entre as várias versões da aplicação, característica esta que oferece desde início um conjunto grande de vantagens: redução do tempo de desenvolvimento; redução da quantidade de código que é necessário criar, manter e otimizar; camada de negócio é apenas atualizada uma vez.

- A tecnologia utilizada pelo núcleo da solução para comunicação assíncrona em tempo real é o SignalR [26], pelo que a aplicação móvel terá que suportar o protocolo que, atualmente, apenas existe implementado em JavaScript e C#.

- Sendo uma aplicação para clientes empresariais, a autenticação é um facto importante, pelo que a solução terá de suportar autenticação dos pedidos ao servidor através do protocolo OAuth, assim como utilização de SSL para autenticação e possivelmente outros pedidos.

- Passando as funcionalidades principais da aplicação por *analytics* / *reporting* de dados em tempo real, é necessária que a solução escolhida seja capaz de apresentar vários tipos de representações gráficas, assim como dados em formato de lista ou tabela. É então também importante que a *framework* escolhida tenha várias bibliotecas de representação gráfica de onde seja possível selecionar a que melhor suporta os requisitos necessários da aplicação.

É importante notar que, apesar de nos capítulos anteriores se referir que a aplicação é para *tablets*, até ao final da primeira metade do estágio era objetivo da empresa que a aplicação a desenvolver fosse para telemóveis. Assim sendo, todo o trabalho exposto no subcapítulo 3.2 e restantes do capítulo 3, assim como as afirmações do capítulo 4, foram realizadas ainda com o objetivo de desenvolver a solução para telemóveis. Tendo isto em consideração, é importante salientar que nenhum deste trabalho nem das conclusões apresentadas podem ser consideradas inválidas ou inúteis, pois tanto o Windows Phone 8® como o Windows 8® funcionam sobre a mesma versão de sistema operativo, o Win RT®. No limite apenas se

poderiam colocar em causa algumas afirmações feitas relativamente ao suporte visual do Windows Phone dado que, apesar de partilharem o mesmo código de linguagem visual, existem algumas pequenas diferenças que poderiam fazer com que uma ou outra biblioteca de gráficos não seja suportada, não sendo no entanto este caso, pois todas são suportadas corretamente em ambos os temos.

3.2 Estado da Arte

Ao procurar soluções que respeitem os pontos mencionados no subcapítulo 3.1 é possível encontrar o que parece ser um leque variado de soluções. No entanto, ao aprofundar o estudo de cada uma das tecnologias encontradas é possível perceber que nem todas possuem as características que se pretendem.

3.2.1 Apache™ Cordova™

O Cordova™ [3] é um conjunto *open source* de APIs em JavaScript, suportados pela fundação Apache™, que permite aceder às funções nativas dos telemóveis (sensores, câmaras, entre outros). Esta solução faz uso de tecnologias web tais como o HTML e CSS além do JavaScript já mencionado para gerar a aplicação localmente no telemóvel (não é necessário um servidor para alojar a aplicação).

Esta *framework* segue o paradigma "write once, run anywhere" no sentido de que o código escrito em JavaScript, principalmente as chamadas às APIs de funções nativas, é consistente em todos os dispositivos, sendo poucas as situações em que é necessário escrever código específico para uma plataforma.

Relativamente à parte visual da aplicação, podem ser usadas *frameworks* de interface de utilizador já conhecidas e usadas em outro tipo de aplicações web: jQuery Mobile [4], Dojo Mobile [5], Sencha Touch [6], entre outras.

Aplicações criadas com Cordova™ acabam por ser aplicações web a correr num browser próprio dentro de uma aplicação nativa e, devido a esta última parte, é possível serem publicadas nas lojas de aplicações pertencentes a cada sistema operativo móvel.

Esta *framework* está disponível para os seguintes sistemas operativos: Windows Phone®, iOS™, Android™, BlackBerry®, Palm® WebOS™, Bada e Symbian™. Com toda esta versatilidade tem de existir algum custo e, neste caso, o ponto fraco encontra-se na performance das aplicações que, no caso de grandes exigências gráficas, apresentam poucas capacidades. Além da *framework*, a fundação Apache™ também disponibiliza um diretório de vários *plug-ins* criados para a *framework* que visam facilitar algumas operações de alguns programadores.

3.2.2 PhoneGap™

O PhoneGap™ [7] é o antecessor do Apache™ Cordova™. Dois anos depois de ser criada, em 2011, a empresa foi comprada pela Adobe® e o core da *framework* foi doado à fundação Apache™ onde ficou com o nome de Cordova™.

Ambas as *frameworks* são muito parecidas e suportam os mesmos dispositivos, no entanto, a Adobe® construiu uma série de serviços à volta do PhoneGap™ que podem ser o fator diferenciador para quem fizer a escolha: o PhoneGap™ Build permite a compilação da aplicação para os vários sistemas operativos pretendidos sem a necessidade de instalação dos SDKs no computador; o Edge Inspect ajuda os programadores a testar a aplicação em vários dispositivos através de *debugging* remoto e também da capacidade de tirar várias *screenshots* aos dispositivos; e o PhoneGap™ Enterprise é uma ferramenta de gestão empresarial para gerir o processo de produção e publicação das aplicações, assim como ferramentas de *analytics* sobre as próprias aplicações.

Também é possível em ambas as *frameworks* alterar entre a execução de código na linguagem nativa e em JavaScript, dando a oportunidade ao programador de usar algum código pré-existente ou alguma biblioteca otimizada para a linguagem nativa.

Apesar de ter sido comprado, o PhoneGap™ continua a ser *open source* com grande influência da comunidade de utilizadores. Este também pode usar o mesmo conjunto de *plug-ins* disponíveis para o Cordova™.

Um dos pontos fortes de ambas estas *frameworks* é a fácil utilização da grande maioria das bibliotecas já existentes para JavaScript. A escolha entre o Cordova™ e PhoneGap™ pode resumir-se apenas aos serviços extra (pagos) que este último disponibiliza.

3.2.3 Telerik® AppBuilder®

O AppBuilder® [8] é semelhante ao PhoneGap™ em vários aspetos: também é baseado em PhoneGap™/Cordova™ e também oferece um conjunto de serviços extra consoante a subscrição escolhida. Sendo o núcleo da *framework* semelhante às duas anteriores é igualmente possível fazer uso dos *plug-ins* existentes para estas assim como de outras bibliotecas em JavaScript.

A principal diferença relativamente às outras duas *frameworks* prende-se então nos serviços extra disponibilizados: assim como a ferramenta Build do PhoneGap™, o AppBuilder® também permite a compilação das aplicações para os vários sistemas operativos na *cloud* não sendo necessária a instalação dos SDKs no computador ou a utilização de hardware próprio (computador com Mac OS™ para iOS™, por exemplo); o LiveSync permite que alterações ao código da aplicação sejam instantaneamente aplicadas às aplicações a correr nos dispositivos de teste; é possível desenvolver código para o AppBuilder® através de vários ambientes diferentes (web browser, cliente *desktop*, extensão

para o Microsoft® Visual Studio®, Sublime Text); o AppPrototyper permite fazer protótipos da aplicação de baixa ou alta-fidelidade, com interações e sistema de revisões; a Telerik® também fornece vários serviços *cloud* com interfaces específicas para o AppBuilder® tais como bases de dados, SMS, correio eletrónico, notificações *push*, entre outros; a plataforma de Mobile Testing permite efetuar testes nos vários sistemas operativos sem alterar código assim como partilhar os resultados destes com a equipa de desenvolvimento; o AppManager permite gerir a disponibilização e instalação das aplicações de modo privado assim como definir as permissões dos utilizadores destas, permitindo também gerir as atualizações às aplicações nos dispositivos dos clientes; por fim, a ferramenta de Analytics permite obter um conjunto lato de informação sobre a aplicação (quais as características mais usadas, obter relatórios de erros das aplicações, analisar a performance das várias ações da aplicação, receber notificações em caso de um evento específico).

O AppBuilder® também permite integrar outras *frameworks* / paradigmas de programação já existentes nas suas aplicações, como é o caso do Knockout.js [9], Angular.js [10], Backbone.js [11], entre outros.

Relativamente à interface de utilizador das aplicações construídas com o AppBuilder®, é possível usar também um grande número de *frameworks* existentes: jQuery Mobile, Kendo Ui® [12], Ionic [13], entre outras.

3.2.4 Appcelerator® Titanium™

Ao contrário das *frameworks* anteriores, o Titanium™ [14], dependendo da plataforma onde pretende correr, pode criar uma aplicação nativa ou híbrida (híbrida no caso do Windows Phone®, nativa para iOS™, Android™ e BlackBerry®). Nesta *framework* também é deixado de lado o HTML e CSS para a criação dos componentes visuais da aplicação e é utilizada uma *framework* chamada Alloy [15] que segue um padrão MVC: as Views são criadas com XML, o estilo destas é definido através de ficheiros JSON semelhantes ao CSS e os controladores são escritos em JavaScript. Para controlo das interfaces de utilizador nativas é utilizada a Titanium™ API que expõe os métodos necessários para aceder a estas.

O Titanium™ também difere das outras *frameworks* no seu funcionamento. Nas anteriores, baseadas em Cordova™ (PhoneGap™ e AppBuilder®), é criada uma aplicação nativa com uma web *View* onde são expostos métodos em JavaScript que a aplicação *wrapper* consegue interpretar. Quanto ao Titanium™, no momento de execução da aplicação existem três componentes principais: o código escrito em JavaScript, a implementação da Titanium™ API na linguagem da plataforma e um interpretador de JavaScript que vai ser utilizado para interpretar o código escrito em *runtime*. Quando uma aplicação é iniciada é criado um ambiente de execução de JavaScript em código nativo e o código da aplicação é interpretado. É neste ambiente que também são colocados os objetos "proxy", ou seja, um objeto JavaScript que tem um objeto correspondente em código nativo. Por exemplo, quando é criado um botão em JavaScript, é chamado um método em código nativo que tratará de criar

um botão na UI nativa, passando a existir um objeto em JavaScript ("proxy") que expõe as propriedades e os métodos deste objeto nativo.

Resumidamente, apesar do Titanium™ também usar JavaScript, toda a aplicação é nativa não existindo uma web *View* (a não ser que utilizador queira). Assim sendo, toda a interface de utilizador é criada com os componentes nativos do sistema operativo.

Relativamente a *plug-ins*, também é possível a utilização destes sendo para tal necessária a criação de uma "proxy", detalhes desta já mencionados em cima.

Já nos serviços extra, fornecidos pelo Appcelerator®, incluem-se: MBaaS que permite criar APIs otimizadas para consumo através de aplicações móveis; Analytics para ter uma visão completa sobre o ciclo de vida da aplicação assim como KPIs e também informação sobre a performance da aplicação e possíveis erros ou exceções que possam acontecer, e um Marketplace onde se encontram módulos desenvolvidos por utilizadores para a própria *framework*.

3.2.5 Xamarin™

O Xamarin™ [16] é a única solução encontrada baseada em C# (C Sharp) e permite criar aplicações para iOS™, Android™, Windows®, Windows Phone® e Mac OS™ (no entanto, para Windows® e Windows Phone®, já que o C# é a linguagem nativa, é recomendável utilizar apenas as funcionalidades de partilha de código). Outra característica diferenciadora é o facto de disponibilizar, através da sua API, todas as funcionalidades existentes nos SDKs nativos dos diferentes sistemas operativos. No entanto também permite chamadas a código escrito nas linguagens nativas respetivas (Objective-C para iOS™ e Java para Android™).

Tanto a solução da Xamarin™ para iOS™ como a de Android™ são construídas recorrendo ao Mono que é a implementação *open source* da *framework* .NET. No caso do iOS™, o compilador do Xamarin™ transforma o código C# da aplicação em código nativo ARM Assembly. Quanto a Android™, o compilador do Xamarin™ transforma o código original numa linguagem intermédia que posteriormente é compilada JIT para *assembly* nativo quando a aplicação é iniciada. As aplicações criadas com esta solução resultam assim em aplicações nativas que executam código nativo sem necessidade de intervenientes intermédios que tratam de passar objetos e instruções de um ambiente para outro.

Relativamente aos ambientes de programação, é possível utilizar o Visual Studio® ou Xamarin™ Studio em Windows® e Xamarin™ Studio no Mac OS™. O programador também está limitado a compilar e testar a solução para os diferentes sistemas operativos móveis em função do sistema operativo da sua máquina (é necessário Mac OS™ para compilar para iOS™ e Windows® 8 para compilar para Windows Phone®; para Android™ é possível compilar a solução em qualquer um dos dois).

É então possível partilhar grande parte da lógica da aplicação e chamadas a serviços, ficando depois dependente de cada plataforma a interação com a interface de utilizador desta e resposta a eventos. A partilha de código pode ser feita através de projetos partilhados (onde é colocado o código partilhado com anotações para o compilador #IF) ou através de uma PCL (dado que é o mesmo código para todas as plataformas, é mais limitado no que é possível partilhar).

Na criação da interface de utilizador das aplicações também existem duas soluções. Recentemente foi introduzido o Xamarin.Forms que permite criar controlos visuais nativos para as três plataformas (iOS™, Android™ e Windows Phone®) a partir do mesmo código C#. Esta solução é limitada pelo número reduzido de controlos disponíveis, pois torna-se complicado criar uma solução geral para vários sistemas operativos com controlos visuais tão distintos. A outra solução, e a mais comum, passa por desenvolver a interface da aplicação especificamente para cada sistema operativo. No caso do Android™ é possível criar a interface de utilizador através de XML ou através do editor visual existente no Xamarin™ Studio. No caso do iOS™ é possível criar os controlos através de C# ou com o editor visual do XCode®.

À semelhança de outras soluções, o Xamarin™ também oferece serviços adicionais que visam ajudar os programadores na criação ou manutenção das suas aplicações: o Xamarin™ Studio vem integrado com o Nuget que é um gestor de pacotes para a plataforma de desenvolvimento da Microsoft® que facilita a descoberta e instalação de pacotes criados por outros (faz download dos ficheiros necessários e adiciona-os aos projetos assim como atualiza as configurações e referências do projeto); a Test Cloud permite efetuar testes ao código e interface de utilizador da aplicação num largo número de dispositivos reais na *cloud* (permite receber relatórios detalhados com métricas e *screenshots*); o serviço Insights permite monitorizar em tempo real qualquer problema com a aplicação, assim como a utilização que levou a esse problema e o número de utilizadores que está a afetar.

3.2.6 Codename One

O Codename One [17] é uma solução *open source* que permite desenvolver aplicações móveis em Java para os seguintes sistemas operativos: iOS™, Android™, BlackBerry®, Windows Mobile® e Windows Phone® (ainda com suporte limitado). Sendo baseada em Java, permite ao programador utilizar as ferramentas a que este está habituado, nomeadamente NetBeans™, Eclipse™ ou Intellij IDEA. À semelhança de outras soluções também poupa ao utilizador ou empresa a necessidade de ter hardware específico para desenvolver para certas plataformas. Esta solução disponibiliza um sistema de compilação das soluções na *cloud*, podendo posteriormente o utilizador apenas fazer download da aplicação pronta a instalar.

Quando a solução é compilada para iOS™, é usada uma máquina virtual que transforma o código Java em código Objective-C, correndo a aplicação de forma nativa no iOS™. Para

os sistemas operativos que correm Java nativamente o processo é semelhante ao do Android™. No caso do Windows Phone® é utilizada uma ferramenta chamada XMLVM [18] que converte o código em instruções *byte code*.

Relativamente à interface de utilizador, esta solução utiliza uma abordagem diferente, no sentido de que todos os componentes visuais são criados da mesma maneira em todos as plataformas apresentando uma estrutura consistente. A criação da interface de utilizador também é facilitada pela existência de um GUI Builder que permite criar os componentes visuais da aplicação através de "*drag and drop*".

Quanto à existência de serviços extra associados à *framework*, o Codename One também possui alguns, que se enumeram de seguida: Analytics API que permite integrar a aplicação com o familiar Google Analytics™ e o CN1Libs que representa um conjunto reduzido de *plug-ins* e extensões que podem ser integrados em projetos da *framework*.

3.2.7 RhoMobile

Inicialmente, a RhoMobile [19] apenas permitia criar aplicações usando HTML e CSS para a parte visual e Ruby para a parte lógica. Mais recentemente abraçou o JavaScript, sendo que agora dá ao programador a escolha da linguagem de programação que pretende usar. No entanto, o acesso aos vários métodos e API nativos ainda não se encontra totalmente transitado para JavaScript. Atualmente as aplicações construídas com esta *framework* podem correr nos seguintes sistemas operativos: iOS™, Android™, Windows Mobile®, Windows Phone® e Windows®.

Para desenvolver e compilar para os diferentes sistemas operativos é necessário software e hardware específico. Para iOS™ é necessário um computador Apple® e os SDKs respetivos. Para Android™ é possível usar qualquer sistema operativo para desenvolvimento. Para Windows®, Windows Mobile® e Windows Phone® é necessária a utilização de um computador com o sistema operativo Windows®. Relativamente aos IDEs suportados, consoante a versão da aplicação que se está a desenvolver é possível utilizar o Rho Studio, XCode® ou Visual Studio®.

Sendo que para a interface de utilizador são utilizados HTML e CSS para criar elementos visuais, a *framework* acaba por comportar-se como algumas das mencionadas anteriormente, dado que também faz uso de uma web *View* dentro da aplicação. No caso dos controladores, Ruby é compilado para Ruby *byte code* e executado numa máquina virtual de Ruby específica para cada sistema operativo que é incluída na aplicação. Neste caso, são os controladores em Ruby que conseguem interagir com os controlos nativos do SDK e a comunicação entre estes e a parte visual é feita através de *tags* específicas no HTML.

Analogamente, esta *framework* também oferece um conjunto de outros programas e serviços que se propõem a auxiliar o programador: o Rho Elements, além das funcionalidades necessárias para construir aplicações, também oferece um criador de aplicações que facilita

o início do processo; o Rho Connect é uma solução que trata da gestão e sincronização de dados, independentemente de a aplicação ter conectividade ou não; e o Rho Gallery é uma solução de gestão das aplicações de uma empresa.

3.2.8 MoSync

O MoSync [20] é uma *freamework open source* que oferece uma solução semelhante ao Appcelerator® Titanium™, sendo que esta *framework* também permite criar aplicações nativas ou híbridas, onde as nativas são desenvolvidas em C/C++ e as híbridas recorrendo à adição de código em HTML, CSS e JavaScript usando a web *View* do sistema operativo. Com o MoSync é possível criar aplicações para diversos sistemas operativos: Windows Phone®, Windows Mobile®, Android™, iOS™, Symbian™OS, entre outros. Relativamente a ambientes de desenvolvimento, é recomendado utilizar o MoSync IDE (baseado em Eclipse™) pois contém muitas funcionalidades adicionais que facilitam o desenvolvimento com este SDK.

Assim como as outras soluções que apenas disponibilizam a compilação de forma local, é necessário hardware e software específico consoante a plataforma para que se pretende compilar (computador Apple® e XCode® para iOS™ e Windows® para Windows Phone®).

À semelhança da solução baseada em Ruby (Rho Mobile), esta também faz uso de compiladores para linguagens intermédias. Usando uma versão modificada do conhecido GCC, é gerado código *assembly* para a plataforma virtual MoSync. Posteriormente, e consoante a plataforma de destino pretendida, é gerado código Java (que correrá em dispositivos baseados neste, tal como o Android™) ou código MoSync IL (para dispositivos baseados em Windows® e Symbian™) que posteriormente é transformado em código binário para, em última instância, ser convertido em código nativo da plataforma pretendida.

Já na parte da interface de utilizador, é possível utilizar diferentes linguagens e tecnologias: JavaScript (com HTML, CSS e/ou Wormhole NativeUI) ou então C++ com diferentes bibliotecas (NativeUI, MAUI e Widget C). A escolha sobre uma destas, além da preferência ou familiaridade com a linguagem, também dependerá dos objetivos de implementação da interface de utilizador, dado que algumas bibliotecas são recomendadas para dispositivos mais antigos e outras para mais recentes, assim como umas produzem o mesmo especto em todos os sistemas operativos enquanto outras apresentam o especto nativo em cada sistema operativo.

Como serviços adicionais à solução esta apenas fornece o MoSync Reload que é uma aplicação de suporte ao desenvolvimento das aplicações (para as versões híbridas apenas) que permite fazer edição e ver o resultado em tempo real nos dispositivos ligados.

3.2.9 AppGyver™ Supersonic

O Supersonic [21], baseado na *framework* Ionic, é semelhante a outras *frameworks* baseadas em HTML, CSS e JavaScript com a vantagem de ter fácil integração com controlos nativos da interface de utilizador do sistema operativo. Esta *framework* apenas está disponível para iOS™ e Android™ e pode ser utilizada em qualquer IDE de desenvolvimento web.

Como as restantes *frameworks* híbridas, a aplicação é executada dentro da aplicação *wrapper* numa web *View*. No caso do Android™, possibilita ainda a escolha entre o *wrapper* por omissão ou um mais optimizador e com características adicionais (chamado Crosswalk). O acesso às funcionalidades nativas do sistema operativo é disponibilizado através de uma API em JavaScript que pode ser chamada a partir da web *View*. Uma das vantagens deste *wrapper* é a flexibilidade, permitindo ao programador executar aplicações criadas por outras *frameworks* tais como PhoneGap™ ou Ionic.

Relativamente à compilação da aplicação, esta é sempre feita na *cloud* da AppGyver™, não sendo necessário hardware ou software específico. Quanto à instalação das aplicações nos dispositivos móveis, existem várias hipóteses consoante o tipo de aplicação: é possível obter uma versão *standalone* (autossuficiente) da aplicação e outra que pode ser descarregada através de um código QR.

Ao criar a interface de utilizador, além de tudo o que é possível fazer com HTML, CSS e JavaScript, também é possível chamar controlos nativos e posicionar a web *View* relativamente a estes de modo a obter o especto desejado.

Quanto a serviços extra associados a esta solução, a AppGyver™ disponibiliza os seguintes: Supersonic Data que, juntamente com o serviço de compilação na *cloud* mencionado anteriormente, permite ligar a aplicação a um conjunto diverso de APIs pré-existentes ou definidas pelo utilizador, assim como bases de dados para o programador utilizar; o Steroids, que é uma ferramenta de CLI que oferece várias funções que visam ajudar o programador na sua tarefa, tais como *debug* remoto das aplicações assim como visualizar as alterações à aplicação em tempo real; e por fim, também é possível usar grande parte dos *plug-ins* criados para PhoneGap™ numa aplicação Supersonic, tirando algum trabalho ao programador.

3.2.10 Adobe® Flex®

Como o PhoneGap™, o Flex® [22] é uma *framework open source* que tanto existe na vertente livre (apoiada pela fundação Apache™), como existe adotada por uma empresa e à qual foram adicionados alguns serviços extra. Além de aplicações móveis para alguns dispositivos (Android™, iOS™ e BlackBerry®), o Flex® também permite criar aplicações web e aplicações para desktop que correm sobre o Adobe® Air®.

Nos dispositivos móveis, a aplicação criada e a maneira como esta é executada pode variar de dispositivo para dispositivo, nomeadamente entre Android™ e iOS™. Relativamente ao primeiro sistema operativo, dado que este suporta Flash (assim como o BlackBerry®), a aplicação consegue correr no *browser* à semelhança do PhoneGap™, por exemplo; quanto a iOS™, dada a sua famosa rejeição do Flash, a aplicação tem de ser empacotada juntamente com o seu próprio *runtime*, neste caso o Adobe® Air®.

Passando agora ao ambiente de programação, é recomendável utilizar o Adobe® Flash Builder (baseado no Eclipse™) dado que está preparado para trabalhar com ActionScript, tanto na parte da programação como na parte do *debug*. Já na parte da compilação, muitas das tarefas e configurações podem ser acedidas através da CLI do Flex®.

Quanto às linguagens de programação, são usadas duas: MXML para a interface de utilizador e ActionScript para a lógica. No caso de a aplicação correr no *browser* (Android™, entre outros), estas duas linguagens são facilmente compiladas para HTML e JavaScript.

3.2.11 Sencha Touch

A Sencha Touch é mais uma *framework* que cria um *wrapper* nativo para o sistema operativo, onde depois este trata de correr a aplicação numa web *View* e fornecer uma API para as funções nativas do telemóvel. A principal diferença desta *framework* para as outras é o facto de esta utilizar uma outra *framework* proprietária da empresa para escrever o código e também a parte visual da aplicação, a ExtJS [23], que é baseada em JavaScript. Esta solução é capaz de correr em iOS™, Android™, BlackBerry® e Windows Phone®.

Como as outras *frameworks* baseadas em JavaScript, o utilizador tem um grande leque de escolha no que toca a IDE, desde os mais focados em desenvolvimento web a outros mais complexos, como é o caso do Eclipse™, NetBeans™ ou Visual Studio®.

Relativamente à compilação e criação da aplicação final, é possível escolher entre três opções: PhoneGap™, Cordova™ ou *wrapper* proprietário. A escolha depende de o utilizador querer ou não usar algumas das funcionalidades dessas *frameworks*, como é o caso do PhoneGap™ Build.

Na parte da interface de utilizador, como mencionado anteriormente, os controlos visuais são criados também através da *framework* ExtJS que contém um número elevado de controlos para escolha e já vêm disponíveis numa série de estilos otimizados para dispositivos *touch*. No entanto, é possível alterar o aspeto destes recorrendo a uma versão de mais alto nível do CSS, o Sass.

Relativamente a serviços extra, a Sencha oferece os seguintes: Space, que permite um controlo granular sobre as aplicações e dispositivos onde estas correm; Architect, um editor visual de aplicações que facilita a alteração de estilos e disposição dos componentes; e a Cmd, a CLI que permite criar projetos e compilá-los, entre outras funções mais específicas.

3.2.12 Qt®

A Qt® [24] é outra framework baseada em C++. Esta também é *open source* mas, à semelhança de outras soluções, também possui uma versão paga fornecida por uma empresa, juntamente com vários produtos extra. Além de suporte para iOS™, Android™ e outros sistemas operativos para desktop, na versão atual o Windows Phone® ainda se encontra em fase beta.

Relativamente a ferramentas de desenvolvimento, tudo depende da plataforma em que se pretende correr a aplicação. Para desenvolver em Android™ é possível utilizar Windows®, Mac OS™ ou Linux, para iOS™ é apenas possível usar Mac OS™ e para Windows Phone® só se pode usar Windows® 8. Além dos IDEs respetivos já mencionados para cada plataforma, existe também a ferramenta da empresa, o QT® Creator, onde também é possível desenvolver e compilar a aplicação.

Passando para a parte da interface de utilizador, atualmente existem duas opções de utilização: ou se constrói usando as APIs disponíveis em C++, ficando tanto a lógica da aplicação como a interface de utilizador na mesma linguagem, ou se constrói em QML [25] (linguagem criada pelo grupo responsável pelo Qt®) que é uma linguagem muito simples e com um nível de abstração muito maior. Se em alguma altura for necessário criar ou estender alguma funcionalidade que não se encontra disponível através de QML, é sempre possível estender funcionalidades e controlos desta através de C++.

Como serviços extra, a versão comercial do Qt® disponibiliza o Qt® Cloud Services que consiste num conjunto de serviços que visam facilitar a integração de vários serviços com a aplicação, tais como: gestão de identidades, integração com bases de dados e comunicação através de *websockets* entre vários clientes e servidores.

3.2.13 Ionic

O Ionic é outra *framework* baseada em Cordova™ mas que ainda se encontra em fase beta, suportando de momento apenas Android™ e iOS™. Assim sendo, também usa como linguagens de programação o HTML, CSS e JavaScript. As poucas diferenças que tem relativamente a *frameworks* semelhantes centram-se no foco na parte visual da aplicação e na grande facilidade de integração com o AngularJS.

Como ambiente de programação é possível usar os normais IDEs de desenvolvimento web, sendo que para a parte da compilação e distribuição utiliza a já referida CLI do Cordova™. No caso do iOS™, para compilar e fazer distribuição/*debug* é necessária a utilização de um computador com Mac OS™. Como as outras aplicações criadas através do Cordova™, esta *framework* também origina aplicações híbridas que correm numa web *View* dentro da aplicação nativa.

Relativamente à interface de utilizador, além da grande variedade de controlos que é possível criar com HTML e CSS, o Ionic também disponibiliza uma série de controlos seus baseados em funcionalidades do AngularJS que visam facilitar a construção e utilização de alguns controlos.

Quanto a serviços adicionais, devido à fase prematura da *framework*, ainda não existem nenhuns disponíveis.

3.3 Soluções

Tendo em conta os Objetivos do Estágio, descrição do produto, Cenários de Uso e Requisitos, já é possível analisar de forma concreta quais as *frameworks*, das estudadas no Estado da Arte, que respeitam as necessidades do projeto.

De seguida é apresentada uma tabela que visa sumariar as características principais de cada *framework* analisada no Estado da Arte. O símbolo "visto" representa a concordância com o indicado no cabeçalho da coluna, a "cruz" representa a discordância com este e o "círculo" representa uma concordância limitada, que será posteriormente explicitada na análise à tabela.

Relativamente à análise das bibliotecas gráficas para cada *framework*, foi utilizado um sistema de numeração na escala de zero a três, sendo que o valor mais alto representa a existência de várias bibliotecas à escolha e o valor mais baixo representa poucas alternativas neste campo. Para os casos em que tal análise não é aplicável, foi utilizado o hífen. Os números foram definidos após pesquisa de soluções e leitura da documentação relativa a este ponto.

	iOS™	Android™	Windows Phone®	Nativa	Híbrida	Linguagem	Gratuita	Bibliotecas Gráficas	
								Nativas	Híbridas
Apache™ Cordova™	✓	✓	✓	✗	✓	HTML, JavaScript e CSS	✓	-	3
Adobe® PhoneGap™	✓	✓	✓	✗	✓	HTML, JavaScript e CSS	✗	-	3
Telerik® AppBuilder®	✓	✓	✓	✗	✓	HTML, JavaScript e CSS	✗	-	3
Appcelerator® Titanium™	✓	✓	○	✓	✗	JavaScript, HTML e XML	✗	3	-
Xamarin™	✓	✓	✓	✓	✗	C#, XAML e XML	✗	3	-
Codename One	✓	✓	○	✓	✗	Java	✗	1	-
RhoMobile	✓	✓	✓	✗	✓	HTML, CSS, Ruby e JavaScript	✗	-	3
Mosync	✓	✓	✓	✓	✓	HTML, CSS, JavaScript e C++	✗	2	2
AppGyver™ Supersonic	✓	✓	✗	✗	✓	HTML, CSS e JavaScript	✗	-	3
Adobe® Flex®	✓	✓	✗	✓	✓	MXML e ActionScript	✗	1	1
Sencha Touch	✓	✓	✓	✗	✓	ExtJS, CSS, HTML	✗	-	3
Qt®	✓	✓	○	✓	✗	C++ e QML	✗	1	-
Ionic	✓	✓	✗	✗	✓	HTML, CSS e JavaScript	✓	-	3

TABELA 1 – CARACTERÍSTICAS PRINCIPAIS DAS FRAMEWORKS ANALISADAS

Tendo em vista as características necessárias, requisitos, cenários de uso e desafios mencionados, podemos observar que nem todas as *frameworks* inicialmente estudadas respeitam as limitações impostas. Assim sendo, através de um estudo aprofundado destas e de vários testes realizados, foi possível eliminar várias *frameworks* da lista inicial.

Começando pela obrigatoriedade de a aplicação ter de correr nos três sistemas operativos móveis principais, têm de ser eliminadas as seguintes *frameworks*: Codename One, AppGyver™ Supersonic, Adobe® Flex®, Qt®, Ionic. Começando pela *framework* Codename One, o facto de ainda se encontrar em fase beta no que respeita ao suporte de

Windows Phone®, cria algumas dúvidas quanto à possibilidade de execução da própria aplicação: será que a *framework* não possui *bugs*? A *framework* já suporta todas as funcionalidades do sistema operativo Windows Phone®? Outro fator eliminatório desta *framework* é a própria linguagem de programação, Java, que mais a frente seria também causa para eliminação, devido ao SignalR já mencionado anteriormente não suportar esta linguagem (apenas suporta C# e JavaScript). Relativamente ao AppGyver™, apesar de ter um funcionamento e arquitetura semelhante a outras *frameworks* híbridas que correm em vários sistemas operativos móveis, ainda não indica suporte para o Windows Phone® sendo, portanto, eliminada. Passando para a Adobe® Flex®, esta também não oferece suporte para Windows Phone® e assim também não é necessário efetuar quaisquer testes posteriores sobre esta. Quanto à *framework* Qt®, a situação é semelhante à do Codename One. Além do suporte para a plataforma Windows Phone® ainda se encontrar em fase beta e levantar as mesmas questões que a *framework* mencionada, também se apresenta com uma linguagem de programação que não seria suportada por uma das características mais importantes de toda a solução, pelo que também foi necessário eliminar esta *framework*. Chegando à última *framework* deste conjunto, a Ionic, podemos ver que esta também não suporta a plataforma Windows Phone®, possivelmente por ser a *framework* com menos maturidade da lista e, portanto, também é eliminada de futuros testes mais específicos.

Outra característica que as *frameworks* necessitam de respeitar é o suporte ao SignalR. Esta biblioteca de ASP.NET apenas tem suporte para clientes e servidores desenvolvidos em JavaScript ou C#, pelo que *frameworks* que façam uso de outras linguagens de programação terão necessariamente de ser eliminadas. Assim sendo, começamos por ter de eliminar a *framework* RhoMobile, que faz uso da linguagem Ruby para implementar a lógica das aplicações desenvolvidas com recurso a esta *framework*. Já a framework MoSync faz uso da linguagem C++ para implementar a lógica das suas aplicações e, pelo mesmo impedimento do uso do SignalR, também terá de ser eliminada. Por fim, a framework Sencha Touch, que usa uma variação da linguagem JavaScript, a ExtJS, também terá de ser eliminada.

Como mencionado na secção de desafios, quer por questões de agilidade de desenvolvimento, quer por questões de facilidade da manutenção de código, é de extrema importância a partilha da maior quantidade possível de código entre as três versões da aplicação. Por esta mesma razão é que se realizou um período de investigação sobre qual a melhor *framework* a utilizar para a construção desta aplicação. Para responder a esta necessidade de negócio, torna-se necessário eliminar a *framework* Appcelerator® Titanium™ pelo facto de não ser possível desenvolver o mesmo tipo de aplicação para as três plataformas móveis. Atualmente, esta *framework* suporta o desenvolvimento de aplicações nativas para iOS™ e Android™, mas apenas a criação de aplicações híbridas para Windows Phone®, o que leva à impossibilidade de partilhar grande parte do código desta com as outras versões.

Assim sendo, da lista inicial de treze *frameworks* apresentadas no Estado da Arte, apenas quatro respeitam as necessidades do projeto.

3.4 Análise do Fluxo de Trabalho das Frameworks

Agora com um número mais reduzido de *frameworks* (Cordova™, PhoneGap™, Xamarin™ e AppBuilder®) que cumprem as necessidades discutidas anteriormente, foi possível realizar testes funcionais sobre cada uma destas.

Antes da execução dos testes, procedeu-se à instalação das *frameworks*. Este processo de instalação e primeira utilização já confere alguma informação sobre o fluxo de trabalho que cada uma suporta e, portanto, representa informação importante para o processo de escolha.

É importante referir que, devido à elevada semelhança entre o Cordova™ e PhoneGap™, tanto a nível da estruturação do código como do próprio funcionamento da *framework*, a análise detalhada seguinte, assim como os testes exaustivos explicitados mais adiante neste relatório, foram realizados sobre a plataforma PhoneGap™ assumindo-se resultados semelhantes para a *framework* Cordova™.

3.4.1 PhoneGap™

Começando pela solução baseada em PhoneGap™, a instalação desta solicitou a instalação de várias ferramentas adicionais, como é o caso do Node.js™ [27] e Git™ [28]. Além destas, no ambiente de desenvolvimento Windows®, também são necessários os SDKs de Windows Phone® [29] e Android™ [30], assim como as respetivas imagens para emulação.

Quanto ao fluxo de desenvolvimento, a programação é feita numa pasta "geral" cujo conteúdo, quando pretendido pelo programador (e através da execução de um simples comando na consola), pode ser propagado para as pastas respetivas de cada plataforma. Por um lado, esta funcionalidade facilita a partilha de código entre as várias plataformas, assim como agiliza qualquer alteração necessária. Por outro lado, e dependendo do padrão arquitetural de desenvolvimento aplicado no projeto, podem ser alterados ficheiros nas outras plataformas que não deviam, originando erros ou produzindo efeitos indesejados.

Relativamente ao *debug* de aplicações, o processo revelou-se fácil para umas plataformas e mais complicado para outras. No caso do Android™, a tarefa de fazer *debug* a uma aplicação revelou-se a mais simples: foi apenas necessário usar a ferramenta Chrome™ DevTools do *browser* Google Chrome™ para poder utilizar a ferramenta de *debugging* remoto em dispositivos que corram este sistema operativo. As funcionalidades de *debug* disponíveis são semelhantes às ferramentas disponíveis no Google Chrome™ para *debug* de páginas / aplicações web (dado que a aplicação a correr no telemóvel acaba por ser uma aplicação web): é possível inspecionar o código HTML da aplicação assim como os estilos em CSS aplicados aos vários elementos do código, também é possível ver os pedidos efetuados pela aplicação

que façam uso de recursos na internet (download de ficheiros e pedidos a outros websites ou serviços), assim como ver e fazer *debug* ao código JavaScrip da nossa aplicação, entre outras funções. À semelhança do *debug* nesta plataforma, o *debug* de aplicações a correr em dispositivos iOS™ faz uso das Developer Tools do *browser* Safari®. Esta solução também oferece as funcionalidades já referenciadas no *debug* da plataforma anterior: observar código HTML e CSS, assim como código JavaScript, observar pedidos feitos a serviços externos, entre outros. Por fim, a plataforma que apresentou a pior solução de *debug* foi o Windows Phone®. Devido a tal, acabaram por ser testados dois processos diferentes de *debug*: através de uma ferramenta chamada WEINRE [31] (Web INspecter REmote) e através da consola. A utilização do WEINRE passa por este criar um servidor intermédio onde irá ocorrer o *debug* da aplicação com funcionalidades semelhantes às ferramentas de *debug* apresentadas anteriormente, sendo que posteriormente o programador acede através do *browser* ao servidor onde é efetuado o *debug*. Para ligar a aplicação ao servidor de *debug* é necessária a inclusão de um ficheiro JavaScript da ferramenta. Ao falhar por diversas vezes em conseguir utilizar esta ferramenta, foi testado o outro método disponível. Ao utilizar a consola de *debug* do Visual Studio®, é possível aceder a mensagens de *debug* que sejam enviadas para a consola através de uma instrução em JavaScript. No entanto, este método é severamente limitado quando comparado aos outros métodos demonstrados.

3.4.2 Xamarin™

Passando para o Xamarin™, além do próprio SDK e IDE deste, é apenas necessário instalar os SDKs das plataformas respetivas (as várias *frameworks* .NET já se encontravam instaladas na máquina de desenvolvimento).

Já no fluxo de desenvolvimento usando Xamarin™, como mencionado no subcapítulo 3.3, é possível programar em qualquer um dos IDEs (Xamarin™ Studio ou Visual Studio®), consoante a plataforma móvel para que se pretende compilar. Também, como referido anteriormente, a partilha de código pode ser feita de duas maneiras: projeto partilhado ou através de *Portable Class Libraries*. Qualquer alteração feita ao código partilhado entre os projetos fica então disponível para ser utilizada em cada plataforma específica.

Quanto ao processo de *debug* das aplicações, ente também é diferente dependendo da plataforma móvel que se está a analisar. No caso do Windows Phone®, devem ser usadas as ferramentas presentes no Visual Studio®, que é a ferramenta de desenvolvimento especializada para esta plataforma móvel. Já no caso do iOS™, é necessária a presença de um computador com Mac OS™ e tanto pode ser utilizado o XCode® como o Xamarin™ Studio. Quanto a realizar *debug* no Android™, este é semelhante ao caso do Windows Phone®, dado que não é necessário nenhum hardware específico e é possível utilizar qualquer um dos IDEs disponíveis.

3.4.3 Telerik®AppBuilder®

Relativamente ao AppBuilder®, para a instalação das ferramentas da CLI, volta a ser preciso utilizar a ferramenta Node.js™. Quanto aos ambientes de desenvolvimento, como mencionado no Estado da Arte, a escolha é abundante (Visual Studio®, editor web, cliente *desktop* do AppBuilder®, entre outros).

O *debug* ocorre de forma semelhante às *frameworks* híbridas mencionadas acima, recorrendo ao *debugger* web nativo de cada uma. A principal diferença é encontrada na estrutura dos ficheiros e pastas dentro do projeto. Apesar de apresentar algumas diferenças, a estrutura continua a ser útil para a separação de conceitos e é fácil a troca de código entre as outras *frameworks* híbridas (PhoneGap™ e Cordova™).

Relativamente à execução das aplicações, independentemente do editor que se use, é possível entre escolher instalar a aplicação num dispositivo ou no emulador.

3.5 Testes Realizados

Partindo de um grupo mais reduzido de *frameworks* disponíveis, já é possível realizar sobre estas alguns testes que visam auxiliar a decisão sobre qual a *framework* mais indicada para o projeto. Das várias informações que foi possível recolher durante a elaboração dos cenários de uso, vamos focar-nos, neste ponto do relatório, na satisfação dos requisitos mais importantes deste projeto: o suporte ao SignalR, a existência de uma boa solução no que respeita à apresentação de informação sobre a forma de gráficos e à persistência de dados no dispositivo móvel.

Nesta altura é importante mencionar que parte dos testes funcionais detalhados de seguida também foram realizados sobre a *framework* AppBuilder®. Este número reduzido de testes serviu apenas para confirmar que de facto o núcleo da *framework* é partilhado com o PhoneGap™/Cordova™. Assim sendo, qualquer um dos testes realizados por nós funcionará sempre nas três plataformas.

3.5.1 Servidor SignalR

Como dito anteriormente, o SignalR é uma biblioteca que fornece comunicação bidirecional entre servidor e cliente. Esta biblioteca utiliza *web sockets*, quando disponíveis para transportar as mensagens entre os intervenientes. Através da função de *push*, é possível enviar dados aos clientes sem estes terem de os requisitar, ou seja, quando existem dados novos o servidor toma a iniciativa de os enviar. A possibilidade da criação de grupos também é uma funcionalidade importante pois, através da subscrição de vários clientes a um grupo, é então possível filtrar as mensagens para um determinado conjunto de clientes.

Para se poder testar o suporte a esta biblioteca nas *frameworks* de teste, foi criado um servidor que implementa a biblioteca SignalR e cuja única função é o envio de um número aleatório de forma regular, num intervalo de tempo estipulado por nós.

3.5.1.1 Cliente SignalR em PhoneGap™

Para a criação deste teste recorreu-se à implementação da biblioteca SignalR em JavaScript. O objetivo deste teste era a verificação se, de facto, a framework Cordova™ suporta esta biblioteca. Inicialmente existiram algumas dificuldades em estabelecer a ligação da aplicação cliente ao servidor. Estas dificuldades foram causadas pela diferença de versões entre a biblioteca cliente e a biblioteca servidor (o número usado para versionamento no nome da biblioteca não é o mesmo que é impresso na mensagem de erro). Também foi necessário criar *callbacks* para as mudanças de estado da aplicação (conectado, desconectado, ligação retomada, erro de ligação, entre outros) pois sem estes não seria possível detetar o que se passa de errado. Ultrapassadas estas dificuldades iniciais, o resto do teste foi bem-sucedido e foi possível receber as mensagens oriundas do servidor.

3.5.1.2 Cliente SignalR em Xamarin™

Na vertente Xamarin™, a criação da aplicação de teste à biblioteca SignalR sofreu dos mesmos problemas iniciais de diferença de versões mas, devido ao sistema de gestão de pacotes existente tanto no Xamarin™ Studio como no Visual Studio®, é apenas necessário especificar qual a versão que se pretende instalar e a ferramenta trata de substituir os ficheiros e alterar as referências necessárias. Fora este problema inicial, o resto do teste foi concluído rapidamente e sem problemas.

3.5.2 Bibliotecas Gráficas

Concluídos os testes ao suporte da biblioteca SignalR, ainda com base no servidor e aplicação cliente existente, foram criados testes para cada uma das bibliotecas de representação gráfica para esta plataforma. Cada teste consistiu na tentativa de utilizar a biblioteca em teste para criar um gráfico de barras que era atualizado com novos valores vindos do servidor de SignalR.

3.5.2.1 PhoneGap™

Neste caso, dado que o PhoneGap™ suporta código JavaScript, existe desde logo uma grande oferta de soluções para este desafio. À semelhança do estudo realizado no Estado da Arte, o estudo de soluções existentes tentou abordar de forma geral as principais ofertas, tentando comparar bibliotecas pagas e gratuitas, assim como diferentes níveis de funcionalidades disponibilizados.

3.5.2.1.1 RazorFlow

O RazorFlow [32] é uma biblioteca focada na construção de painéis de instrumentos, com amplo suporte para dispositivos móveis, que pode ser utilizada tanto em PHP como JavaScript. Para este teste focou-se apenas nas funcionalidades gráficas através da API em JavaScript.

A implementação do código para teste da biblioteca foi muito simples de efetuar. As configurações disponíveis para alteração de cores no gráfico, assim como o formato dos números nos eixos deste, estão disponíveis através de atributos em JavaSCript para o programador. No entanto, se se pretenderem efetuar alterações adicionais ou com maior granularidade, será necessário alterar a configuração CSS da própria biblioteca. A par da customização, outra limitação observada no teste desta biblioteca, prende-se com o tipo de gráficos disponibilizados, sendo apenas possível criar gráficos do tipo barra, barra dupla, coluna, linha e pie. A ação de atualizar o gráfico também se revelou um pouco complexa, sendo necessário obter um *lock* sobre o gráfico e, finda a atualização este, libertar o *lock*.

Outra limitação observada durante os testes foi a resposta da biblioteca às atualizações frequentes. Correndo a mesma aplicação em dois dispositivos diferentes, foi possível observar diferenças no estado atual dos gráficos, existindo um atraso de alguns segundos entre os dois dispositivos. Outros pontos negativos observados foram: existência de algumas paragens no funcionamento da biblioteca durante alguns segundos; interações *touch* fazem com que o gráfico pare de se atualizar (possivelmente serão necessárias configurações adicionais para desligar as interações ao toque); apesar de ser uma biblioteca muito utilizada, o facto de ser paga leva a que exista pouca informação/exemplos sobre a configuração de gráficos.

3.5.2.1.2 Flot

O Flot [33] é uma biblioteca gratuita e *open source* escrita para o jQuery. Esta pretende oferecer um uso simplificado aliado a um aspeto cuidado.

A criação do código de teste foi muito simples, tanto devido aos exemplos fornecidos pela página web da própria biblioteca, como devido à extensa comunidade e exemplos existentes na internet. Comparativamente ao RazorFlow, esta biblioteca tem uma performance bastante superior, não apresentando atrasos nem adiantamentos face à mesma aplicação a correr noutro dispositivo simultaneamente, mesmo saindo e entrando na aplicação, ou então através de *tombstoning*.

Quanto às opções de configuração, esta biblioteca apresenta tantas ou mais do que a RazorFlow. No entanto, algumas destas acabam por ser difíceis de implementar pois a biblioteca tenta apresentar a informação gráfica da maneira que acha melhor, por vezes sobrepondo-se a algumas configurações do utilizador. É então necessário realizar algumas tentativas e um pouco de *debug* para obter o resultado pretendido.

Quanto a pontos negativos há apenas um a apontar que é o aspeto pouco polido que os gráficos gerados apresentam, sendo necessária experiência em CSS para alterar e refinar o aspeto destes.

3.5.2.1.3 Flotr2

Esta [34] biblioteca é muito semelhante ao Flot, também *open-source* e gratuita, tendo muitas funcionalidades semelhantes, mas sendo distinguida principalmente pela não dependência do jQuery. Outra das vantagens é que já traz de raiz algumas funcionalidades extra, como é o caso do *tooltip* de um certo ponto no gráfico.

A criação da aplicação de teste foi muito simples, notando-se a semelhança com o Flot. Apresentou-se com os mesmos pontos fortes e fracos (principalmente o aspeto pouco polido dos gráficos), com especial ênfase no suporte a *plug-ins* e funcionalidades extra desenvolvidas pelo utilizador.

3.5.2.1.4 jqPlot

O jqPlot [35] é muito semelhante às duas bibliotecas mencionadas em cima (também é gratuita e *open-source*) mas com algumas funcionalidades mais específicas em falta. Esta biblioteca também depende de jQuery, no entanto, toda a biblioteca foi construída em cima de e preparada para funcionar com *plug-ins*. Um pouco à semelhança da biblioteca Flotr2 mas com impacto ainda maior.

Mais uma vez, a implementação da aplicação de teste correu sem problemas devido à grande simplicidade do código necessário. No entanto, são necessárias algumas linhas extra devido à definição de *plug-ins* para os vários componentes do gráfico, como mencionado em cima.

Relativamente a pontos negativos, voltamos novamente ao aspeto dos gráficos. São necessárias algumas alterações ao código JavaScript e CSS para obtermos o aspeto pretendido.

3.5.2.1.5 amCharts

A amCharts [36] é uma biblioteca paga com um elevado número de funcionalidades e diferentes opções de configuração.

A criação da aplicação de teste já foi mais trabalhosa do que as versões que utilizaram as bibliotecas em cima devido ao elevado número de opções de configuração existentes. Esta aplicação de teste acabou por ser a que mais código necessitou para funcionar (cerca de duas a três vezes mais) e ter o aspeto pretendido.

Ainda assim, esta necessidade de código adicional não foi interpretada como um ponto negativo pois permite um elevado controlo sobre toda a funcionalidade e aspeto do gráfico, sem a necessidade de alterar ficheiros CSS. Outros pontos fortes passam pela existência de uma galeria de tutoriais, além da detalhada documentação oferecida pela empresa

responsável, assim como um editor de gráficos online e a disponibilidade de temas pré-definidos que podem ser aplicados aos gráficos.

Outro ponto forte que nenhuma outra biblioteca apresentou foi a adaptação automática à orientação do telemóvel, consoante se passa de modo retrato para modo panorâmico. Outras bibliotecas podem suportar esta funcionalidade mas apenas esta biblioteca a suporta por defeito.

3.5.2.1.6 HighCharts™

Esta [37] biblioteca também é paga e revelou-se muito semelhante à amCharts no que toca a funcionalidades assim como toda a parte da documentação e exemplos (também disponibiliza um editor de gráficos).

Relativamente à implementação, também foi necessário escrever algum código (novamente devido à elevada oferta de opções de configuração). No entanto, à primeira vista, parece causar alguma perda de performance na aplicação, sendo que se existirem mais gráficos na mesma página desta ou se for necessário realizar alterações à interface de utilizador, poderá causar alguma falha de performance percetível. Este revelou-se o único ponto negativo detetado no teste.

3.5.2.1.7 FusionCharts

Assim como as outras bibliotecas pagas, esta [38] também oferece documentação detalhada e um editor online onde o utilizador pode observar o código necessário para criar um gráfico assim como testar alterações à configuração deste em tempo real.

Passando para a criação do teste, esta biblioteca foi a que exigiu a menor quantidade de código na criação do gráfico. Já no método de atualização do gráfico foi uma das que necessitou da maior quantidade de código, devido ao formato de dados que requer (objetos JavaScript com um certo formato).

Quanto a pontos negativos, é necessário referir que a criação de um gráfico, com a configuração por defeito, traz um número elevado de configurações desnecessárias tais como estilos e cores, que posteriormente é necessário descativar. Outro ponto negativo centra-se no ato de redesenhar o gráfico sempre que é feito *scroll* na página da aplicação.

3.5.2.1.8 xCharts

A biblioteca xCharts [39] também é um produto *open source* baseado na conhecida biblioteca de visualização gráfica D3.js. A documentação desta biblioteca é mais simples que as outras aqui mencionadas mas este também é o seu objetivo, o de abstrair um pouco a complexidade e o controlo de baixo nível presente na biblioteca D3.js.

A implementação da aplicação de teste também foi muito simples, graças à configuração de alto nível disponível. Tanto ao nível da criação do gráfico como ao nível da atualização regular deste, apenas foram precisas algumas linhas de código para tudo funcionar.

Quanto a pontos fortes e fracos desta biblioteca, apenas foram encontradas características positivas. O aspeto polido dos gráficos assim que são criados, sem necessidade de configuração adicional, assim como as transições suaves quando este é atualizado, contribuíram para uma aparência e fluidez que nem todas as soluções que são pagas oferecem.

3.5.2.2 Bibliotecas de Gráficos para Xamarin™

Relativamente a soluções de representação de gráficos em Xamarin™, devido à natureza da partilha de código entre as aplicações para cada plataforma distinta, é possível obter dois conjuntos de bibliotecas: específicas para cada sistema operativo e bibliotecas gerais que funcionam em mais do que um sistema operativo móvel.

Devido à enorme disponibilidade de bibliotecas de representação de gráficos para cada sistema operativo móvel em específico (iOS™, Android™ e Windows Phone®) e seguindo o cunho característico da *framework* Xamarin™ que se centra na partilha de código entre as três plataformas, centraram-se os estudos e testes nas bibliotecas que suportem os três sistemas operativos.

3.5.2.2.1 Telerik® UI for Xamarin™

Além da plataforma para criar aplicações híbridas, a Telerik® também oferece uma lista extensa de vários controlos [40] para os diversos sistemas operativos. Recentemente, também introduziu uma linha de controlos para o Xamarin™, onde começou por disponibilizar uma solução para a representação de gráficos. Estes controlos visam estender as funcionalidades oferecidas pelo Xamarin.Forms, que já é um conjunto de controlos visuais simples (como blocos de texto, caixas de texto, imagens, listas, entre outros) que podem ser implementados nos vários sistemas operativos a partir de uma única base de código C#.

A criação da aplicação de teste foi simples de realizar, utilizando apenas poucas linhas de código para obter o efeito desejado. Os gráficos apresentados já possuem um aspeto polido sem necessidade de configurações adicionais. Comparativamente às outras bibliotecas referenciadas anteriormente, devido à natureza da adaptação às várias plataformas, esta solução apresenta menos funcionalidades do que as outras. No entanto, as funcionalidades básicas e alguns extras estão disponíveis já na versão atual da biblioteca.

3.5.2.2.2 OxyPlot

O OxyPlot [41] é uma biblioteca *open source* para .NET com suporte para Xamarin™ e Xamarin.Forms. À semelhança da Xamarin™ UI, a OxyPlot também pode ser utilizada em aplicações Xamarin a partir de uma única base de código em C#.

A aplicação de teste também foi fácil de criar e muito semelhante à da biblioteca anterior. Também se obteve um resultado com aspeto polido sem necessidade de recorrer a configurações adicionais.

Como pontos fracos devem ser apontados a pouca documentação existente (tendo, no entanto, muitos exemplos que podem ser consultados) e poucas opções de configuração.

3.5.2.2.3 Syncfusion™ Essential Studio for Xamarin™

À semelhança da solução oferecida pela Telerik®, a Syncfusion™ também oferece um conjunto de controlos [42] pagos para o Xamarin.Forms, entre eles um controlo de gráficos que oferece elevado grau de funcionalidade e configuração.

A implementação da aplicação de testes também foi semelhante a ambas as anteriores. Esta opção também necessita de pouco código, tanto para a criação e configuração como atualizações do gráfico. Também como os seus antecessores, produziu uma solução com aspeto polido nas várias plataformas de teste.

Como ponto negativo, esta biblioteca também oferece um conjunto reduzido de documentação, com poucos exemplos (possivelmente por este conjunto de controlos ter sido disponibilizado recentemente). No entanto, as características da linguagem C# acabam por ajudar com algumas destas falhas.

3.5.2.2.4 Steema TeeChart™ for Xamarin.Forms

A TeeChart™ [43] é outra solução paga que oferece uma série de controlos específicos para gráficos com suporte para a plataforma Xamarin™. Sendo uma solução recente, como todas as outras, ainda oferece pouca informação no seu website e a documentação é quase inexistente. No entanto, possui algum código de exemplo que é suficiente para conseguir testar esta biblioteca.

Seguindo o exemplo referenciado em cima, foi possível criar uma representação gráfica de teste semelhante às anteriores, tanto ao nível do código necessário, como a nível de aspeto final. Também, analogamente às bibliotecas testadas anteriormente, não foi encontrado qualquer problema ou dificuldade na implementação do gráfico da TeeChart™.

Como pontos negativos temos a já mencionada falta de documentação e exemplos, que considero ser uma das ferramentas mais úteis na introdução a uma nova ferramenta, assim como um número muito limitado de recursos sobre esta ferramenta espalhados pela internet (resultados de pesquisa, perguntas no *StackOverflow*).

3.5.2.2.5 Infragistics™ Xamarin.Forms

A Infragistics™ [44] foi outra empresa que recentemente lançou um conjunto de controlos pagos para o Xamarin.Forms. Tal como a biblioteca mencionada anteriormente, a documentação sobre esta ainda é inexistente e grande parte dos controlos de gráficos ainda se encontram em fase beta ou CTP, podendo alguns ter funcionalidade limitada ou até *bugs*.

Mesmo assim, recorrendo às aplicações de exemplo disponibilizadas juntamente com a biblioteca, foi criada uma aplicação de teste. Durante a criação desta aplicação foram encontrados vários problemas causados principalmente pela falta de documentação. No

entanto, foi possível obter o resultado pretendido após alguma experimentação com as configurações disponibilizadas pela biblioteca. A complexidade e tamanho do código foi semelhante às bibliotecas já indicadas.

Resumindo, como pontos fortes temos a familiaridade do código e o elevado grau de configuração da biblioteca, à semelhança das outras, enquanto como pontos fracos temos a inexistência de documentação e o estado prematuro da biblioteca (apesar de não terem sido encontrados nenhuns problemas durante a elaboração da aplicação de teste).

3.5.3 Persistência de Dados

Como foi mencionado na secção de desafios, será necessário encontrar uma solução que disponibilize a persistência de alguns dados no dispositivo móvel, podendo estes ser respeitantes ao utilizador (tais como definições, dados de autenticação, entre outros) ou respeitantes aos próprios dados que a aplicação pretende mostrar ao utilizador.

À semelhança de características e funcionalidades específicas que cada sistema operativo móvel suporta, também cada um disponibiliza diferentes meios para implementar um sistema de persistência de dados na aplicação. No caso do Windows Phone®, existem três sistemas diferentes [45]: *isolated storage*, que é a memória interna e persistente da aplicação no telemóvel onde é possível salvaguardar diversas estruturas de dados e ficheiros; SQL Server® Compact, que se apresenta como uma base de dados embebida para aplicações; e o SQLite™ que também é uma base de dados relacional embebida. Quanto ao iOS™, este suporta vários tipos de persistência de dados [46]: *file* API, que permite leitura e escrita de ficheiros dentro do sistema de ficheiros da aplicação (outras funcionalidades de persistência de dados para o iOS™ têm esta classe como base); Core Data, que é um modelo de dados de uso geral para persistência de informação, otimizado para trabalhar com *Models* da arquitetura MVC (baseado em SQLite™); e, por fim, o SQLite™ já descrito anteriormente. Relativamente ao Android™, existem as seguintes opções [47]: conjuntos chave-valor (dicionários) nas SharedPreferences, que apenas suportam tipos de dados primitivos (*booleans*, *floats*, *ints*, *longs* e *strings*); *file* API, com opções de escrita e leitura de ficheiros para o espaço de memória da aplicação; e, novamente, o SQLite™.

Como podemos ver por esta breve análise aos sistemas de persistência de dados dos dispositivos móveis, a solução baseada em SQLite™ [48] é a única nativa e comum aos três. Deve ser considerada então como escolha preferencial para este problema. Com algum estudo seria certamente possível encontrar outras soluções que tivessem uma utilização comum aos três sistemas operativos, no entanto, tais soluções teriam de obrigatoriamente ser baseadas nas opções nativas acima enumeradas. Sendo também um *standard* do armazenamento de dados em dispositivos móveis e já existindo há vários anos, é possível considerar o SQLite™ uma solução robusta e estável.

Feita esta escolha, procedeu-se então à criação de aplicações simples com o intuito de testar o suporte ao SQLite™ pelas plataformas restantes.

3.5.3.1 SQLite™ em PhoneGap™

Para a aplicação de teste ao suporte de SQLite™ foi utilizado o *plug-in* mais popular da biblioteca de *plug-ins* do PhoneGap™, o SQLitePlugin [49] (este *plug-in* suporta os três sistemas operativos pretendidos).

A aplicação de teste criada consiste numa interface simples com alguns botões e que permite realizar várias opções, tais como: criar tabela, eliminar tabela, adicionar registo, eliminar registo, entre outras funções. Para a instalação do *plug-in* foi apenas necessária a execução de um comando na consola e os ficheiros necessários e as respetivas referências foram adicionados ao projeto. Relativamente ao código necessário, este foi muito simples, sendo apenas necessária a escrita das instruções de SQL a executar, sendo que o *plug-in* tratava do resto.

Recorrendo a informações de *debug*, foi possível testar a aplicação nos vários sistemas operativos e registar o correto funcionamento/suporte desta. Está então confirmado o suporte do PhoneGap™ ao SQLite™.

3.5.3.2 SQLite™ em Xamarin™

A implementação da solução em Xamarin™ revelou-se um pouco mais trabalhosa. Como referido anteriormente, os três sistemas operativos suportam o uso do SQLite™ mas apenas o Android™ e iOS™ trazem o SQLite™ embutido. No caso do Windows Phone®, é necessário introduzir a biblioteca para este na aplicação (foi utilizada a biblioteca oficial [50] criada pela equipa do SQLite™), assim como o código [51] necessário para aceder às funções de manipulação da base de dados (no caso do Android™ e iOS™ apenas esta última biblioteca é necessária). Adicionalmente, no código partilhado entre as três plataformas, é necessário usar instruções condicionais de compilação pois, no caso do Windows Phone®, a localização do ficheiro da base de dados é definida de maneira diferente.

Ultrapassadas estas dificuldades iniciais, foi então possível testar com sucesso o suporte do SQLite™ nos vários dispositivos móveis do Xamarin™. A aplicação também se apresenta com uma interface onde mostra vários botões para realizar operações sobre a base de dados, à semelhança da aplicação acima descrita para o PhoneGap™.

3.6 Seleção da Framework a Utilizar

Como é possível observar, a escolha da melhor ferramenta a utilizar para desenvolver um produto ao nível empresarial é um procedimento complexo dividido em várias etapas.

Após a definição da ideia e requisitos que a solução criada terá de respeitar, inicia-se o processo de escolha da melhor ferramenta para o problema, que terá de tentar obter um equilíbrio entre os vários intervenientes no projeto.

De um estudo inicial de soluções existentes para o problema em concreto resultou um número grande de potenciais candidatos. No entanto, aplicando sobre esta lista os requisitos previamente definidos, nem todas as soluções iniciais foram capazes de satisfazer estes, obtendo-se no final deste processo uma lista mais reduzida.

Dando ênfase especial a três requisitos específicos que a solução a desenvolver terá de respeitar (representações gráficas, suporte à biblioteca SignalR e a SQLite™), foram realizadas uma série de provas de conceito e testes de modo a observar, dentro da lista já reduzida de *frameworks*, quais destas apresentam maior potencial e se adequam melhor ao problema a resolver, pois não é só por respeitarem os requisitos impostos que se deve partir para uma escolha. Toda a análise posterior efetuada em termos do fluxo de desenvolvimento das *frameworks*, quantidades de bibliotecas gráficas disponíveis, entre outos critérios de comparação mencionados anteriormente serviu para, das *frameworks* restantes, conseguir classificar e comparar estas.

Durante este processo de testes e criação de provas de conceito, foi possível conhecer de forma mais profunda e verdadeiramente funcional as *frameworks* restantes, informação esta que não é possível alcançar apenas através da leitura sobre estas e as suas funcionalidades.

Apenas então findo o processo de testes foi possível avaliar as várias *frameworks* sobre os critérios apresentados na seguinte tabela. Os pontos atribuídos pertencem a uma escala de 1 a 5, representando o 1 a pior avaliação e o 5 a melhor avaliação.

	Cordova™	PhoneGap™	Xamarin™	AppBuilder®
Preço	5	4	2	3
Apoio/Suporte/Documentação	4	5	4	4
Características específicas	1	2	3	5
Adaptação a vários tamanhos de ecrã	2	2	5	3
Velocidade de desenvolvimento	4	4	4	4
Curva de aprendizagem	3	3	4	3
Acesso a APIs nativas	5	5	5	5
Performance	3	3	5	3
Suporte para aspeto nativo	3	3	5	4
Debug	3	3	5	3

TABELA 2 – ANÁLISE E TESTES ÀS FRAMEWORKS SELECIONADAS

Começando pela avaliação de custo monetário de cada *framework*, a que obteve melhor nota foi a *framework* Cordova™ pois é a única completamente gratuita. A PhoneGap™ ainda oferece algumas funcionalidades num modo gratuito, mas a necessidade de funcionalidades extra, como aplicações e *plug-ins* privados, leva a que seja necessária uma subscrição mensal, sendo que ainda oferece posteriormente um plano empresarial. De seguida temos o AppBuilder® que só oferece subscrições ao seu serviço pagas, tendo no entanto já a maioria das funcionalidades necessárias nos planos mais baixos. Por último, o Xamarin™ apresenta-se com os planos mais caros, sendo necessário escolher acima dos planos mais baixos para ter funcionalidades de nível empresarial.

Os critérios de apoio, suporte e documentação foram agrupados pois considerou-se que respondem à mesma pergunta: "Quão fácil é esclarecer dúvidas sobre esta *framework*?". Neste caso, todas as soluções acabaram por ter uma nota semelhante, mas por diferentes razões. Agrupando a Cordova™, PhoneGap™ e AppBuilder®, tendo todas por base o mesmo *core* e, principalmente, sendo baseadas nas linguagens HTML, CSS e JavaScript, pode-se dizer que têm de facto a melhor documentação, pois partilham grande parte dos temas de desenvolvimento de páginas web (este tema tem a maior quantidade de recursos na internet). O PhoneGap™ apenas se destaca face aos seus dois parceiros devido a ser, dos três, a *framework* mais utilizada e, consequentemente, aquela com mais informação disponível, quer em termos de tutoriais, de dúvidas esclarecidas e outros recursos (livros, blogues, entre outros). O Xamarin™ também obteve uma nota elevada devido às linguagens que utiliza: C#, que é um padrão para aplicações Windows®, possuindo toda documentação e suporte variado que daí advém, assim como as linguagens nativas, tanto para a parte lógica, como para a componente visual de cada plataforma móvel (é possível, assim, utilizar as enormes quantidades de informação disponíveis para cada plataforma específica).

Relativamente às características específicas que cada *framework* oferece, estas já foram mencionadas no subcapítulo 3.3. Sendo o Cordova™ a base do PhoneGap™ e AppBuilder® e mantendo-se *open source*, acaba por não disponibilizar nenhum serviço adicional, não oferecendo nenhumas utilidades extra. O PhoneGap™ e AppBuilder® foram construídos em cima do Cordova™ com o objetivo de se integrarem com outras soluções que estão disponíveis via subscrição. Por estas razões o Cordova™ é classificado como a pior *framework* relativamente a este critério. O PhoneGap™ apresenta uma pequena melhoria de pontuação pois já oferece alguns serviços extra, como o sistema de compilação na *cloud* ou a sua galeria de *plug-ins*. O Xamarin™ aparece em segundo lugar pois oferece alguns serviços a mais que os seus antecessores. Por último, a AppBuilder® apresenta-se com a melhor pontuação neste critério devido ao conjunto alargado de funcionalidades e serviços extra que disponibiliza aos seus utilizadores (serviços estes enumerados no subcapítulo 3.3).

No critério de adaptação aos vários tamanhos de ecrã, o Cordova™ e PhoneGap™ partilham a pontuação mais baixa pois, devido ao seu funcionamento híbrido, o ajuste às dimensões do ecrã é feito com recurso a CSS e *media tags*. O AppBuilder® apresenta uma ligeira melhoria dado que já oferece algum suporte para diferentes disposições da interface de utilizador conforme o tamanho do dispositivo móvel. Já o Xamarin™ apresenta a melhor pontuação pois é possível definir, de forma nativa, quais as disposições de conteúdo a usar consoante as dimensões do ecrã.

No caso do critério de velocidade de desenvolvimento, temos pontuações iguais para todas as plataformas. Aquelas baseadas numa solução híbrida (Cordova™, PhoneGap™ e AppBuilder®), por razões mencionadas anteriormente, possuem a facilidade e rapidez de desenvolvimento devido à semelhança com o desenvolvimento de páginas e aplicações web, tendo o programador apenas de se adaptar às chamadas das APIs nativas do sistema operativo móvel. Relativamente ao Xamarin™, com a funcionalidade de Xamarin.Forms, também esta na linguagem C# e igual ao código de lógica, é possível programar quase toda a aplicação (dependendo do caso) numa só linguagem.

Relativamente à curva de aprendizagem, este é um critério apenas influenciado pela equipa deste projeto. Como mencionado no capítulo 3, os intervenientes neste projeto e a restante equipa da empresa têm fortes conhecimentos em tecnologias Microsoft®, usando na maioria dos seus projetos a linguagem C# (assim como o estagiário). Apenas um número mais limitado de colaboradores da empresa possui sólidos conhecimentos em linguagens de programação web (HTML, CSS e JavaScript). Por estas razões, o Xamarin™ apresenta a melhor classificação neste ponto.

Já no suporte de chamadas a APIs nativas, todas as *frameworks* apresentam pontuação máxima, já que todas disponibilizam as interfaces e APIs necessárias para fazer as chamadas às funcionalidades específicas de cada sistema operativo móvel. Esta é uma das razões que torna estas *frameworks* nas mais escolhidas pelos utilizadores quando se fala em aplicações nativas para vários dispositivos móveis.

Como era de esperar, as aplicações nativas apresentam sempre melhor performance que as aplicações híbridas. Esta performance pode ainda ser pior no caso de aplicações com muitos componentes visuais com atualizações frequentes (como é o caso da aplicação que se pretende desenvolver). O Xamarin™ aparece com a melhor classificação pois produz aplicações completamente nativas, lógica e interface de utilizador, com um impacto mínimo na performance.

Quanto ao suporte para aspeto nativo, as *frameworks* utilizam diferentes abordagens. Cordova™ e PhoneGap™ dão preferência ao mesmo aspeto em todas as plataformas, precisando o utilizador de usar *plug-ins* externos ou definir código CSS diferente para cada tema, o que se pode tornar pouco prático ou de difícil manutenção, acabando por nunca obter um aspeto verdadeiramente nativo. No caso do AppBuilder®, um dos outros produtos da Telerik® é o Kendo Ui® que, para aplicações móveis, facilita a criação do aspeto nativo da plataforma em que a aplicação está a correr. Graças a este produto extra da mesma empresa, o AppBuilder® apresenta melhor pontuação que as outras soluções híbridas. O Xamarin™ volta a apresentar a melhor classificação, desta vez devido ao simples facto de a parte visual da aplicação ser criada na linguagem nativa de cada plataforma, obtendo assim o aspeto nativo de cada.

No critério de *debug*, a situação já foi descrita em detalhe no subcapítulo 3.6. Para as aplicações híbridas é preciso ferramentas específicas para cada sistema operativo móvel, nem sempre podendo aceder a todas as funcionalidades pretendidas. Por esta razão é que se encontram com a pior classificação. O Xamarin™ volta a oferecer a melhor solução neste campo devido à aplicação resultante do uso desta *framework*. Sendo totalmente nativa, é possível recorrer ao *debug* de cada IDE para cada plataforma em específico.

Tendo em conta todos os requisitos, estudos detalhados, aplicações de teste criadas, assim como esta apreciação final das *frameworks*, foi escolhida a *framework* Xamarin™ como a melhor solução para este projeto. Além de todos os pontos fortes e fracos referidos anteriormente, também se deve fazer referência à possibilidade da partilha de código da camada de negócio entre o *core* da solução e a aplicação móvel, sendo possível propagar qualquer alteração aos objetos de negócio na aplicação principal para a aplicação móvel, com extrema facilidade.

O próprio desenvolvimento da aplicação, através da linguagem C#, torna-se mais fácil (devido as várias vantagens desta, algumas já referidas) e menos propício a erros, dado que não é uma linguagem de *scripting* como o JavaScript. Dado que toda a lógica da aplicação principal está desenvolvida também em C#, a partilha de código com a aplicação móvel também é um enorme ponto a favor.

O conjunto de bibliotecas de criação de gráficos, apesar de serem e menor número (no caso das soluções para as três plataformas) do que para as soluções híbridas, não é um fator com grande peso devido à qualidade e performance que é possível obter com estes controlos visuais nativos.

Devido ao facto do produto principal ainda se encontrar em desenvolvimento e ainda não ter sido definido o módulo que fará uso extensivo de representações gráficas, ainda não se pode indicar qual a biblioteca de representação gráfica que será escolhida a par do Xamarin™. No entanto, qualquer uma das analisadas assim como das indicadas para disponibilidade futura, terá certamente as características necessárias.

4 Desenvolvimento

Na etapa de planeamento, imediatamente antes de entrar para o desenvolvimento propriamente dito da aplicação, tiveram lugar vários outros testes e estudos que serviram para responder a várias questões que surgem neste ponto: qual o *design pattern* ou padrão de programação a utilizar? Como estruturar o código da aplicação? Existe facilidade/possibilidade de partilha de código com a aplicação web? Partilha ou não de código da IU entre as diferentes plataformas? Que outras ferramentas ou paradigmas devem ser utilizados neste caso em específico? Através de vários testes realizados e pequenas provas de conceito foi possível chegar a uma solução de desenvolvimento adotada como a melhor para este projeto.

Começando pelo *design pattern* a utilizar, os mais populares e mais utilizados são: Model-View-Controller (MVC), Model-View-Presenter (MVP) e Model-View-ViewModel (MVVM), sendo que todos os três possuem características em comum, das quais se destacam o Model (representa as entidades e é completamente agnóstico aos outros elementos do paradigma) e a View (representa a parte visual da aplicação). O fator diferenciador e aquele que normalmente influencia a escolha prende-se com as necessidades que o programador precisa de satisfazer com o Controller/Presenter/ViewModel. Neste caso em específico, a possibilidade de fazer uso extensivo de *bindings* e de tornar as Views completamente agnósticas (facilitando em grande parte a criação de testes para a aplicação) fez com que a escolha recaísse sobre o MVVM. No caso específico da aplicação a desenvolver, em C# e utilizando Xamarin™, apenas se destacam duas *frameworks* que já possuem algumas funcionalidades: MVVMLight [52] e MVVMCross [53]. Neste caso, a escolha acabou por recair sobre o MVVMLight por várias razões, entre elas: menos verboso que o MVVMCross, conjunto de funcionalidades necessárias é mais específico e portanto é preferível utilizar uma *framework* que se aproxime da funcionalidade desejada, assim como a familiaridade da equipa de desenvolvimento com esta.

Relativamente à estrutura do código da solução, a figura seguinte representa a disposição do código do projeto partilhado.

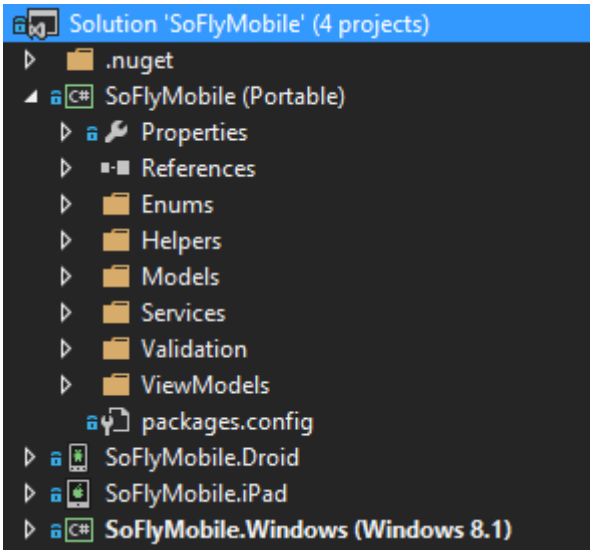


FIGURA 3 – ESTRUTURA DO CÓDIGO DA SOLUÇÃO CRIADA

Como se pode observar na Figura 2, a necessidade de ter um conjunto de serviços *core* comum às três plataformas, levou a que tivesse de ser utilizado um dos métodos de partilha de código mencionado no subcapítulo 3.3.5. Neste caso o método escolhido foi a utilização de uma PCL pela facilidade resultante do uso desta: é gerado apenas um DLL que posteriormente deve ser incluído nas diversas aplicações; a não existência de diretivas #IF leva a que operações de *refactoring* sobre o código sejam sempre aplicadas a todas as suas referências (caso mais difícil recorrendo à outra solução de partilha) e a integração fácil com uma situação futura onde os objetos de negócio da aplicação estejam contidos num DLL que seja partilhado tanto pela aplicação móvel como pelo servidor. Na mesma imagem pode-se observar a principal vantagem da organização desacoplada escolhida previamente (MVVM) num projeto multiplataforma como este, já que é possível partilhar logo de início mais de dois terços do código da aplicação, estando contidos na PCL os Models e ViewModels, assim como todo o código relativo aos diferentes serviços. A aplicação desta arquitetura leva a que o código específico a cada plataforma, salvo a utilização de alguma API própria do dispositivo, se limite à parte visual da aplicação, neste caso as Views. A utilização desta estrutura de pastas na PCL e dentro de cada projeto, é coincidente com a definição dos diferentes *namespaces* da solução.

Quanto à partilha de código da IU entre as várias aplicações, após algumas provas de conceito iniciais, as preocupações mencionadas no subcapítulo 3.3.5 também foram confirmadas, notando-se uma oferta limitada de controlos para construir *layouts* complexos (que vários módulos da aplicação possuem) e a inexistência de algumas propriedades básicas dos controlos para o seu posicionamento e aspeto (como é o caso da margem e moldura). Assim sendo decidiu-se recorrer às linguagens nativas de cada plataforma para a criação das respetivas interfaces de utilizador.

Relativamente a outros padrões de programação utilizados, recorreu-se extensivamente a *inversion of control* e *dependency injection* por forma a facilitar a separação de código assim como a utilização de código específico consoante a plataforma, dos quais serão mencionados exemplos mais à frente.

Para finalizar o processo inicial de desenvolvimento é importante mencionar os SDKs escolhidos para as respetivas plataformas. Começando pelo Android™, a empresa escolheu focar-se na versão mais recente deste, definindo o *target* da aplicação para a versão Lolipop (API *level* 22). No caso do iOS™, por imposição da própria loja de aplicações da Apple, a versão do SDK utilizada foi a 8.6. Relativamente à versão para Windows, foi decidido suportar a versão Windows 8 RT®, sendo que devido à partilha deste sistema operativo entre tablets e desktops/portáteis e telemóveis Windows Phone 8, a aplicação deverá também estar adaptada ao controlo por rato (este requisito resultou apenas em mudanças visuais conforme se esteja a usar um rato ou uma ação *touch*).

4.1 Código Partilhado da Aplicação

Começando pelo trabalho realizado na PCL, dar-se-á ênfase aos três grupos mais trabalhosos e que contêm a maior parte do código: *Models*, *Services* e *ViewModels*.

A pasta *Models* possui todas as representações dos objetos de negócio que a aplicação necessita para funcionar, armazenar e apresentar a informação corretamente. Alguns destes objetos apenas existem em páginas específicas (objeto *Plugin* nas páginas de gestão de *Plugins*, por exemplo) enquanto outros existem durante todo o tempo de execução da aplicação (como é o caso do objeto *User*, *Subscription* e *Project*). A maior parte destes objetos possui uma representação visual na aplicação, sendo que qualquer alteração que ocorra a uma propriedade destes é necessário que seja refletida visualmente. Assim sendo, foi necessário implementar na maior parte dos objetos a interface *INotifyPropertyChanged*.

```
public class Person : INotifyPropertyChanged
{
    private string customerName;

    public string CustomerName
    {
        get
        {
            return this.customerName;
        }
        set
        {
            if(this.customerName != value)
            {
                this.customerName = value;
                NotifyPropertyChanged();
            }
        }
    }

    private void NotifyPropertyChanged([CallerMemberName] String propertyName = "")
    {
        if(PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(propertyName);
        }
    }
}
```

FIGURA 4 – EXEMPLO DE IMPLEMENTAÇÃO DA INTERFACE INOTIFYPROPERTYCHANGED

Esta interface expõe um único evento chamado *PropertyChanged* que notifica qualquer cliente que esteja à escuta que o valor da propriedade mudou. Existem diferentes metodologias de implementação desta interface mas costumam representar todas o mesmo comportamento, sendo que só é lançada a notificação quando o valor passado é diferente do atual. Esta interface é extremamente útil quando combinada com a utilização de *bindings* em *Data Binding*. Um *binding* representa uma ligação entre a interface de utilizador e o objeto que representa e permite a troca de informação entre estes. Normalmente esta relação é definida na linguagem visual quando possível ou em C# quando tal não é possível ou por razões de legibilidade.

```
<TextBlock Text="{Binding person.CustomerName, Mode=TwoWay,
 UpdateSourceTrigger=PropertyChanged, Converter={StaticResource DisplayStringConverter}}" />
```

FIGURA 5 – EXEMPLO DE UTILIZAÇÃO DE BINDING EM LINGUAGEM VISUAL XAML NO WINDOWS 8 RT

```
CustomerNameBinding = this.SetBinding(
    () => this.person.CustomerName,
    () => this.CustomerNameTextBlock.Text,
    BindingMode.TwoWay)
    .ConvertSourceToTarget(DisplayStringConverter)
    .UpdateTargetTrigger(UpdateTriggerMode.PropertyChanged);
```

FIGURA 6 – EXEMPLO DE UTILIZAÇÃO DE BINDING EM CÓDIGO C# NO XAMARIN ANDROID

Tanto na Figura 5 como Figura 6 temos a definição do mesmo *binding*, sendo que este é criado entre a propriedade CustomerName (string) da instância referenciada pela variável `person` do *Data Context* actual e a propriedade `Text` da caixa de texto referenciada pela variável CustomerNameTextBlock. O modo de *binding* declarado é *TwoWay* levando a que alterações numa propriedade se reflitam na outra, como por exemplo, ao preencher o nome na caixa de texto a variável irá ser atualizada em correspondência, assim como o caso de uma atualização à propriedade CustomerName resultará na atualização da propriedade visual. De entre várias opções extra temos a possibilidade de definir uma função que irá converter os dados passados entre as propriedades caso estes sejam de tipo diferentes ou necessitem de alguma formatação prévia, assim como oportunidade de dizer quando deve ocorrer a sincronização dos valores (neste caso é sempre que a propriedade é alterada, mas poderia ser, por exemplo, quando a caixa de texto perde o *focus*). A utilização de *bindings* permite tornar o código mais legível e reduzir a quantidade de código suplementar (deixa de ser necessário criar funções para ficar à escuta de eventos para atualizar as propriedades) e por estas razões as expressões de *binding* são utilizadas extensivamente nas três plataformas.

Passando para o tópico dos *ViewModels*, nesta solução foi decidido que deveria ser implementado um *ViewModel* para cada *View*. Esta decisão foi baseada nas seguintes razões: correta separação de conceitos devido a esta relação de um para um entre a *View* e o *ViewModel*; todo o código relacionado com obtenção de dados através dos vários serviços acima mencionados assim como preparação destes para apresentação ao utilizador é realizada no *ViewModel* correspondente à *View*; dado que é apresentada a mesma informação independentemente da plataforma onde a aplicação está a correr, é apenas necessária uma instância de cada *ViewModel* no código partilhado; ao passar toda a lógica comum para os *ViewModels*, deixando apenas no ficheiro associado às páginas (normalmente designado por *code-behind*) código relativo a alterações visuais, é possível criar testes que alcancem a maior parte do código da aplicação, dado que a criação de testes sobre *code-behind* é um processo complicado e por vezes impraticável.

Para cara *ViewModel* são passados por *dependency* injection os diferentes serviços que cada um necessita, sejam estes serviços de comunicação com a API respetiva ou o serviço de navegação. Como mencionado anteriormente, todos os serviços à exceção do serviço de navegação, disponibilizam quase na totalidade apenas métodos que executam chamadas à API no servidor. A execução de pedidos web assim como outras ações bloqueantes ou

demoradas podem impedir a IU de funcionar corretamente devido a estarem a executar as suas tarefas na *thread* principal. Outra vantagem da utilização do Xamarin™ e da linguagem C# para programar para os vários sistemas operativos é a possibilidade da utilização do padrão *async/await* que esta disponibiliza. Este padrão permite a utilização e criação de métodos não bloqueantes de uma maneira muito mais fácil sem a necessidade de criação de *threads*. Em vez de *threads*, a execução deste tipo de métodos faz uso do contexto de sincronização actual, apenas fazendo uso da *thread* atual quando a função se encontra ativa. Neste caso particular, os vários métodos de interação com a API (*getAsync(), postAsync()*, *putAsync()* e *deleteAsync()*) encontram-se todos implementado recorrendo a métodos assíncronos disponibilizados pela framework .NET, levando a que a IU nunca fique bloqueada à custa da realização de um pedido a um recurso web. Outras operações associadas a estas ações, tais como leituras de conteúdo dos pedidos ou criação e leitura de *streams*, também se encontram implementadas com recurso aos métodos assíncronos disponíveis.

A pasta *Services* possui as classes que representam os diferentes *endpoints* e métodos da API do servidor que vão ser chamados pela aplicação. Além destes também inclui interfaces de outros serviços cuja implementação é específica a cada sistema operativo.

Da definição dos requisitos veio a necessidade de criar serviços para comunicar com os seguintes *endpoints* da API: AccountsService, PluginsService, PluginThemesService, SocialMessagesService e StreamsService. De seguida será apresentada uma visão global sobre cada um destes serviços:

AccountsService - apenas dispõe de métodos que realizam a autenticação do utilizador junto do servidor (a gestão de utilizadores, por enquanto, ainda é feita de forma manual no servidor), assim como outros métodos que vão buscar informação essencial ao funcionamento da aplicação tais como a subscrição atual pertencente ao utilizador assim como os projetos aos quais este se encontra associado. Esta informação (*User*, *Subscription* e *Project*) é normalmente usada em conjunto para realizar a maior parte dos pedidos à API.

PluginsService - contém os métodos que realizam as várias operações CRUD sobre os Plugins, assim como métodos auxiliares ao tratamento deste tipo de dados e ainda pedidos a outros *endpoints* mas que foram colocados nesta classe, pois estão diretamente relacionados com este objeto e não são utilizados em mais nenhum lugar.

PluginThemeService - contém os métodos que realizam as várias operações CRUD sobre os PluginThemes, assim como métodos auxiliares ao tratamento deste tipo de dados.

SocialMessagesService - contém os métodos que realizam as várias operações CRUD sobre as SocialMessages (representam conteúdo criado numa Stream), assim como métodos auxiliares ao tratamento deste tipo de dados.

StreamsService - contém os métodos que realizam as várias operações CRUD sobre as Streams, assim como métodos auxiliares ao tratamento deste tipo de dados.

Esta separação dos *endpoints* a consumir em diversas classes, além de resultar numa clara separação de conceitos e funcionalidades que cada uma, também apresenta vantagens na altura da instanciação dos ViewModels quando utilizada em conjunto com *dependency injection*.

A pasta *Services* também contém a interface que define o sistema de navegação personalizado a utilizar pelas várias aplicações, dado que o processo de navegação entre páginas é executado de maneira bastante diferente em cada sistema operativo.

Considerou-se necessária a criação de um sistema de navegação personalizado devido à análise dos *mockups* da IU iniciais: ambos os menus laterais assim como o *header* e *footer* da aplicação mantêm-se sempre visíveis e sujeitos a alterações e interações em todas as páginas da aplicação. Dado que estão sempre visíveis e maioritariamente com o mesmo aspeto, torna-se desnecessário estar a carrega-los/redesenha-los sempre que se navega para uma página, pois a criação da interface assim como qualquer interação com o utilizador (nos três sistemas operativos) ocorre sempre na mesma *thread* (UI *thread* ou *Main thread*). Parar tornar este processo mais rápido e manter esta *thread* desimpedida mais tempo, foi decidido que a melhor opção seria implementar um sistema de páginas *Master-Detail*, sendo que a página *Master* é a que se encontra sempre visível e contém os itens acima mencionados, enquanto a página *Detail* é aquela que vai variando consoante as ações do utilizador.

Foi por isso necessária a implementação desta interface em cada plataforma de maneira a fazer uso das componentes específicas de cada uma. A vantagem desta metodologia é tangível, como dito anteriormente, na partilha de código entre as aplicações, dado que nos ViewModels apenas está definido que farão uso de uma classe que implementa esta interface, sendo que na inicialização de cada aplicação é passado por *dependency injection* a implementação correta e específica da plataforma. De seguida será apresentada rapidamente este serviço e posteriormente as suas implementações.

Interface IMasterNavigationService

Esta interface implementa alguns métodos que são responsáveis por executar a navegação, variando apenas o número de parâmetros (definindo página *Master* e/ou página *Detail* assim como a passagem de parâmetros para uma, ambas ou nenhuma).

Assim sendo, quando esta classe é utilizada dentro de um ViewModel para navegar, apenas é necessário chamar o método pretendido com os respetivos parâmetros que se pretenda passar.

4.2 Código Específico a Cada Plataforma

Implementação IMasterNavigationService em Windows 8 RT®

A implementação para Windows® foi uma das mais simples, já que existe apenas o conceito de *Frame* (página da aplicação) e cada uma pode ter outras *Frames* dentro de si (Anexo 4, Figura 1 – consoante o separador escolhido, a *Frame* com a lista de Plugins é trocada pela *Frame* com a lista de PluginThemes). A implementação foi baseada numa solução muito simples apresentada por Loek van den Ouweland [54].

Implementação IMasterNavigationService em Android™

Em Android™, a solução implementada consistiu na criação de *Fragments* (páginas com lógica independente que representam páginas *Detail*) incluídos dentro de *Activities* (páginas principais que representam páginas *Master*), sendo que também foi possível incluir *Fragments* dentro de *Fragments*. Além da implementação da interface do serviço de navegação, também foi necessário criar classes auxiliares para a criação e exibição dos *Fragments* assim como a gestão dos parâmetros passados. Assim sendo, das classes auxiliares criadas *CustomBaseActivity* e *CustomBaseFragment* descendem todas as outras *Activies* e *Fragments* criados na aplicação.

Começando pela classe mais simples, a *CustomBaseFragment* apenas necessita de uma propriedade onde guardar o parâmetro passado para esta, já que qualquer referência necessária à sua *Activity* responsável pode ser acedida através de uma propriedade da classe base *Fragment*.

A classe *CustomBaseActivity* é mais complexa pois é responsável pela obtenção dos parâmetros passados tanto para a *Activity* como para o *Fragment*, assim como a criação e exibição do *Fragment* associado. A passagem de parâmetros entre *Activities* é realizada através de *Intents*, neste caso para navegação de uma *Activity* para outra, sendo que é mantido no sistema de navegação um dicionário estático onde são guardados os objetos (parâmetros) até serem recebidos pela página de destino.

Em qualquer uma das três implementações, se alguma página de destino, *Master* ou *Detail* for uma das que já se encontra exibida, o redesenho não é executado. No caso de ser apenas necessário alterar a página *Detail*, é feito uso de um dos métodos da classe *CustomBaseActivity* para apenas substituir o seu *Fragment*, passando o parâmetro correspondente para o substituto.

Implementação IMasterNavigationService em iOS™

No caso do iOS™, a lógica da implementação foi muito semelhante ao Android™, devido ao facto de também existirem dois tipos distintos de páginas, assim como uma hierarquia definida entre estes. As páginas *Master* são definidas através de *UIViewControllers* enquanto as páginas *Detail* são definidas através de *UIViews*. Nesta implementação também

se recorreu à criação de duas classes base de onde as várias páginas do projeto se estendem: *BaseUIViewController* e *BaseUIView*.

A classe *BaseUIView* além de guardar o parâmetro que lhe é passado, também é responsável por guardar a referência e preparar a representação visual a que está associada (neste caso ficheiros .xib). Quanto à classe *BaseUIViewController*, à semelhança da classe homóloga em Android™, também é responsável pela correta passagem dos parâmetros para as páginas assim como instanciação e apresentação das *BaseUIViews* que lhe são passadas.

Interface IFacebookService

Inicialmente, para a obtenção de SocialMessages sobre uma Stream do Facebook, assim como informação sobre esta, era utilizado o SDK em C# oficial do Facebook. No entanto, para as versões de Xamarin.Android e Xamarin.iOS era necessária uma biblioteca adicional desenvolvida pela equipa do Xamarin™ de forma a ser possível fazer uso da biblioteca oficial. Foi então necessária a criação de outra interface, denominada IFacebookService, que definia os vários métodos que eram necessários para obter esta informação, tendo sido posteriormente implementada em Android™ e iOS™. Contudo, em reunião com a equipa responsável pela API, foram levantados pelo estagiário alguns problemas com esta implementação, nomeadamente a observação de frequentes alterações à API do Facebook sem aviso prévio aos utilizadores desta, que resultam em alterações em triplicado neste caso (versão Web, versão Windows® e versões Android™ e iOS™), assim como a consequente manutenção adicional de duas implementações diferentes. Foi então decidido passar a recolha de dados sobre Streams e SocialMessages para a API do servidor sendo que depois todas as aplicações cliente passam a consumir a mesma informação.

Barras da Aplicação

Além da necessidade de criação das Views especificamente para cada plataforma (baseadas em *mockups* cedidos pela empresa), também foi necessária a criação de código adicional para se poder observar o mesmo comportamento e aspeto nas várias plataformas (diferentes eventos de *life cycle* da criação de páginas, utilização de imagens nos diferentes sistemas operativos, entre outros). Como referido anteriormente, a página Master da aplicação possui uma barra a todo o comprimento do ecrã em baixo, onde são apresentados ao utilizador alguns botões com ações que pode executar consoante a página da aplicação onde se encontra. Além desta e em situações específicas, como o caso da remoção de um item ou de feedback de uma ação, existe outra barra imediatamente acima da anterior, que trata de mostrar, respetivamente, mensagens de confirmação com ações que o utilizador pode executar e também mensagens de notificações.

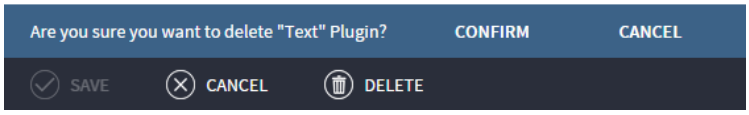


FIGURA 7 – BARRA DE AÇÃO E MENSAGEM DE CONFIRMAÇÃO

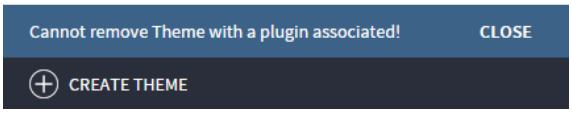


FIGURA 8 – BARRA DE AÇÃO E MENSAGEM DE FEEDBACK

Mais uma vez, foram utilizadas as vantagens do padrão MVVM e a clara separação de conceitos que este disponibiliza. A View *Master* é a única responsável pela apresentação destas barras, assim como pela execução das ações correspondentes a cada botão. Independentemente da plataforma onde a aplicação se encontra a ser executada, cada página apresenta ao utilizador as mesmas opções e comportamento, ou seja, cada ViewModel apenas tem de notificar a IU atual sobre o que pretende que a barra de ação ou de notificações apresente ao utilizador. Cabe à implementação da View *Master* de cada sistema operativo a receção destas notificações e atuação sobre as mesmas. Para esta comunicação assíncrona entre os vários ViewModels e uma View em específico, fez-se uso do sistema de mensagens da biblioteca MVVMLight. Esta classe *helper* disponibiliza ao programador diferentes métodos de subscrição e envio de mensagens, por tipo ou por chaves.

Neste caso, foram criadas algumas classes adicionais que representam os diferentes tipos de mensagens que podem ser passados. A classe AppBarContent representa o conteúdo contido na barra de ação. Esta classe tem apenas duas propriedades que representam listas de botões (é possível criar botões alinhados à esquerda e à direita). Cada botão é representado pela classe ApplicationBarButton que contém as seguintes propriedades: *Text*, texto do botão; *Image*, imagem a apresentar no botão; *Enabled*, se o botão se deve encontrar ativo (disponível para clique) ou não; e *Action* que representa o método a executar. Assim sendo, o ViewModel de uma página que pretenda alterar o conteúdo da barra de ação, apenas tem de criar uma mensagem do tipo acima mencionado com o conteúdo pretendido e enviá-la.

```
public void CreateAppBar()
{
    Messenger.Default.Send(new AppBarContent
    {
        LeftAppBarButtons = new List<ApplicationBarButton>
        {
            new ApplicationBarButton
            {
                Text = "CREATE FOO",
                Enabled = true,
                Image = "bar.png",
                Action = () =>
                {
                    FooBar();
                }
            }
        }
    });
}
```

FIGURA 9 – CRIAÇÃO E ENVIO DE MENSAGEM DO TIPO APPBARCONTENT

Foi escolhido definir a função a executar como um *delegate* para que do lado recetor seja necessário apenas fazer *Action.Invoke()* sem a necessidade de conhecer qualquer contexto de execução (*models*, funções ou expressões *using*). Do outro lado, na criação da View *Master*, é definida a função a executar aquando a receção de uma mensagem do tipo AppBarContent.

```
Messenger.Default.Register<AppBarContent>(this, UpdateAppBar);
```

FIGURA 10 – DEFINIÇÃO DA FUNÇÃO QUE DEVE TRATAR DAS MENSAGENS DO TIPO APPBARCONTENT

O código da Figura 10 faz com que a View atual, ao receber uma mensagem do tipo AppBarContent, chame a função UpdateAppBar passando a mensagem recebida por parâmetro. Assim sendo, volta-se a observar mais uma vantagem da separação de conceitos atingida com o padrão MVVM - do lado da lógica de negócio temos o programador que apenas pretende que apareça uma barra de ação com determinados botões, recebendo o *designer* no *code-behind* da View a mensagem com toda a informação necessária para a construção e exibição desta, ficando a seu cargo a utilização dos controlos nativos de cada plataforma para obter o resultado desejado. De notar apenas que na implementação deste método em cada plataforma, é necessária a correta tradução da *string* que representa a imagem a apresentar para o tipo de dados utilizado pela plataforma correspondente. No caso do Windows® e iOS™, é possível aceder às imagens da aplicação usando um sistema de diretório com o caminho completo para a imagem; no caso do Android™, as imagens são do tipo *drawable* e devem ser acedidas através de um Id atribuído pelo sistema operativo,

sendo que foi necessária fazer esta conversão após a receção da mensagem. A lógica para a criação e exibição da barra de notificação é semelhante a esta.

Controlos Personalizados

Quando se pretende criar um controlo nativo com funcionalidades específicas, existem várias opções à escolha do programador, cada uma com vantagens e desvantagens específicas: pode já existir uma biblioteca criada por terceiros que nos disponibiliza o resultado pretendido ou então será necessário criar o controlo, juntando vários controlos existentes ou criando um novo de raiz.

Na aplicação desenvolvida, existem vários controlos que, tanto pelo facto de requererem aspeto e funcionalidades específicas, como pelo facto de terem de ser reutilizáveis em várias páginas, tiveram de ser convertidos ou criados, consoante o caso, em controlos personalizados. Nestes vários casos, a criação do controlo foi só considerada/executada após extensa pesquisa por soluções equivalentes já existentes. Os principais e mais trabalhosos controlos personalizados criados nas várias plataformas foram o Plugin e respetivo PluginTheme, assim como as paletes de cor que permitem alterar a cor em tempo real.

Como se pode ver nas imagens no Anexo 2, o tema do PluginTheme é composto por várias zonas e vários controlos individuais, tais como imagens e blocos de texto, cuja implementação reutilizável em cada plataforma foi uma tarefa que apresentou vários desafios devido às diferentes limitações de cada, tais como a não possibilidade de fazer *clipping* ao conteúdo quando aplicada uma *border* redonda, assim como o não suporte do iOS™ e Android™ a *borders* com diferentes larguras consoante o lado. Em alguns dos casos foi possível alterar a estrutura visual do PluginTheme de modo a simular o efeito pretendido, sendo que noutros casos foi mesmo impossível de obter um aspeto semelhante ao reproduzido em HTML e JavaScript no final. Outra funcionalidade que necessitou de especial atenção foi a lista de comentários do Plugin. Tanto na página de criação/edição do tema como na página de criação do Plugin, apenas são mostrados ao utilizador comentários fictícios para ser fácil perceber o aspeto final destes com o tema aplicado. Já na página de edição do Plugin, são carregados comentários reais da Stream correspondente. Para suportar estas duas situações, na implementação do Plugin em cada sistema operativo, a lista é criada dinamicamente consoante a situação em que é apresentada, tratando os dados recebidos de igual maneira.

Para a alteração do tema em tempo real, foi feito uso extensivo dos *bindings* acima mencionados, propagando as alterações aplicadas às paletes de cor tanto ao modelo do próprio tema (para edição ou criação deste) como ao controlo personalizável para efeitos de visualização.

Quanto às paletes de cor, consoante o sistema operativo, foram utilizadas combinações de controlos personalizados assim como bibliotecas criadas por terceiros. No caso da plataforma Windows®, a palete de escolha de cor utilizada foi criada por terceiros enquanto

o retângulo representativo da cor escolhida foi um simples controlo criado para o efeito. Exclusivo a este sistema operativo são as propriedades dos controlos visuais: todas estas são *Dependency Properties*, podendo ser definidas através de *bindings*, estilos visuais ou outras ações, sendo que todas também criam notificações no caso de serem alteradas. Assim sendo, quando a cor é definida na palete e o retângulo representativo da cor selecionada é alterado, esta alteração é apanhada pelo *binding* entre a cor deste a propriedade que representa esta no modelo PluginTheme. No caso do Android™, também foi utilizada uma biblioteca externa que além de disponibilizar a palete de cor, também permitia usar o retângulo representativo da cor escolhida. Infelizmente, ao contrário da plataforma Windows®, não era possível saber quando a propriedade deste, neste caso a cor, era alterada, tendo sido necessário criar um controlo personalizado que implementa a interface *INotifyPropertyChanged* mencionada anteriormente. Foi então possível obter um comportamento semelhante ao da plataforma Windows®. Esta biblioteca também incluía algumas opções visuais extras que tiveram de ser removidas de modo a alcançar um aspeto homogéneo nas várias plataformas. No caso do iOS™ foi novamente utilizada uma biblioteca desenvolvida por terceiros sendo que para o retângulo representativo da cor escolhida é utilizada o controlo visual mais simples do iOS™, a UIView. Neste caso, à semelhança do Android™, este controlo não consegue notificar quando alguma propriedade deste é alterada. Foi então necessário recorrer a uma funcionalidade específica da biblioteca MVVMLight para forçar a reavaliação da expressão de *binding* quando a propriedade de destino é alterada.

Principais Desafios Encontrados

Além dos desafios mencionados ao longo deste relatório, considera-se importante mencionar os seguintes:

- Uma aplicação para Android™ é gravemente limitada em termos de memória disponível. Adicionalmente, durante a execução desta, temos um Garbage Collector de Java e outro de C# que nem sempre andam sincronizados no seu trabalho. Durante o desenvolvimento da aplicação esta limitação fez-se notar principalmente quando se faz uso de imagens grandes, originando erros do tipo *OutOfMemoryError*. Felizmente existem várias indicações de como resolver este problema desde *caching* a carregamento das imagens numa resolução mais pequena. Consoante a necessidade, recorreu-se tanto a funções para apenas carregar para memória a imagem na resolução de destino, reduzindo o espaço ocupado por esta, como a *caching* e *lazy loading* em páginas em que são mostradas muitas imagens (Anexo 4, Figura 9). A implementação de estas diferentes soluções ficou a cargo de uma classe exclusiva do Android™ chamada ImageManager. Para não ocupar muito tempo de processamento na *thread* da IU, estes métodos são executados numa *thread* diferente ou de forma assíncrona, consoante a necessidade.

- As imagens de avatar dos utilizadores estão guardadas e comprimidas em *blobs* no Azure (*cloud* Microsoft). Por defeito, o Android™ não suporta realizar pedidos com a *flag* GZip, sendo que foi necessário criar uma simples implementação da classe WebClient (uma das várias classes em C# que permite realizar pedidos web) para este caso, de modo a conseguir obter as imagens no seu formato correto.

- A utilização de *bindings* no Android™ e iOS™ foi conseguida apenas após várias semanas de tentativas e erros, dado que o suporte pela biblioteca MVVMLight a estas plataformas ainda é precoce e não existe documentação nem exemplos complexos para estas duas.

- No iOS™, para manter a aplicação adaptável a vários tamanhos de ecrã, fez-se uso das funcionalidades de *auto-layout* através do uso de *constraints*. Contrariamente às outras plataformas, foi necessário alterar valores destas últimas durante a execução da aplicação por forma a obter o comportamento e aspeto visual desejado.

- Também existiram muitos desafios na criação das interfaces devido às limitações de cada sistema operativo, ou por não suportarem alguma funcionalidade, ou por ser necessário código em C# no *code-behind* para obter o efeito pretendido.

- Como mencionado anteriormente, o desenvolvimento da aplicação encontra-se um pouco atrás do desenvolvimento do core/API do produto. Ainda assim, a API do produto sofreu várias alterações ao longo do tempo (algumas mais profundas que outras) que necessitaram de ser propagadas às várias classes de serviço da aplicação móvel assim como aos objetos de negócio e lógica desta (os Models e ViewModels).

- Inicialmente, existia muito código repetido no *code-behind* para uma mesma página implementada nas diferentes plataformas. Após uma análise profunda deste e do código já presente em cada ViewModel, foi possível passar quase todo para este último, ficando apenas no *code-behind* código relativo a interações com a IU. Este feito foi conseguido tentando manter um código generalizado sem nenhuma ligação à parte visual da aplicação.

- Durante o desenvolvimento da aplicação nas várias plataformas foram encontradas outras limitações que não puderam ser ultrapassadas. Estas limitações, que impediram a aplicação de ter o mesmo aspeto em todas as plataformas, resumiram-se aos controlos visuais disponíveis nas várias plataformas, dentro os quais se destacam: listas com paginação em vez de *scrolling* (não disponível em iOS™ e Android™), lista *dropdown* (não disponível em iOS™) e *checkbox* (não disponível em iOS™).

- O principal desafio encontrado durante o decorrer do estágio foi a necessidade de aprender três linguagens de IU novas e completamente distintas, assim como os respetivos ciclos de vida das aplicações para cada sistema operativo. Tendo a oportunidade de dedicar mais tempo ao projeto, certamente seria possível apresentar uma solução mais polida visualmente.

5 Conclusão

Findo o período correspondente ao estágio curricular, é possível fazer uma análise ao trabalho realizado ao longo dos vários meses.

Olhando para a primeira fase do estágio, é possível perceber que foi realizada uma análise detalhada das soluções existentes e das funcionalidades de cada para discernir qual a mais adequada a este trabalho em específico. Apesar de todas as funcionalidades testadas e requeridas para a *framework* a escolher não terem sido ainda utilizadas nos módulos desenvolvidos, progride-se com a certeza de que, quando os módulos que fazem uso destas forem implementados, não haverá falta de suporte às funcionalidades pretendidas. A alteração do *target* da aplicação de telemóvel para *tablet* acabou por causar pouco impacto como aludido no capítulo 3. Apesar de inicialmente se ter planeado fazer uso da funcionalidade do Xamarin Forms para definir a IU, a *framework* Xamarin™ facilmente suportou a definição da IU para cada sistema operativo. Pode-se então concluir que o considerável investimento temporal realizado em estudos e planeamento efetuados durante este período inicial do estágio tiveram enorme relevância e impacto no desenvolvimento, qualidade e adaptabilidade da solução.

Relativamente à segunda fase do estágio, que corresponde a todo o período de desenvolvimento e implementação da solução, pode mencionar-se que tudo decorreu dentro dos parâmetros definidos pela empresa. A solução encontra-se funcional e estável, respondendo aos requisitos definidos no início do estágio. Os desafios que surgiram ao longo do trabalho deste semestre, tanto ao nível das especificações das várias plataformas, como ao nível de alterações da API por parte da equipa desta, foram ultrapassados.

No geral, é plausível afirmar que o balanço do estágio curricular é bastante positivo. Esta afirmação é fundamentada em vários pontos, dos quais se destacam: decisões arquiteturais e padrões utilizados no desenvolvimento da solução que criam uma arquitetura modular, com fácil integração de novos módulos e manutenção dos existentes; adoções de vários padrões (*MVVM*, *bindings*, *dependency injection*, entre outros) que visam melhorar a legibilidade do código e facilidade de programação e manutenção deste; criação de vários controlos personalizados e reutilizáveis nos diferentes sistemas operativos que trazem funcionalidades acrescidas à aplicação. Também é de notar a comunicação bidirecional entre o estagiário e a empresa, os conhecimentos transmitidos a este pela equipa sénior e não só, assim como todos os conhecimentos resultantes de todo o processo de investigação, estudo, planeamento e implementação associados à conceção e desenvolvimento do produto deste estágio.

Findo o período de estágio, a aplicação encontra-se funcional com os três módulos mencionados: Plugins, PluginThemes e Streams. Posteriormente e à medida que forem desenvolvidos novos módulos no core e API do produto, estes deverão ser passados para as aplicações móveis. Quer isto dizer que para os testes e requisitos que foram respeitados no

processo de escolha da *framework* e que ainda não foram aplicados, aquando a definição de novos módulos, a solução estará pronta e suportará a inclusão destes.

Pretende-se então fazer crescer a plataforma e torná-la numa referência na área. Além de novos módulos a ser adicionados, o suporte a novas redes sociais e adição de funcionalidades requeridas pelos clientes são opções que certamente serão consideradas num futuro próximo.

6 Referências

[1] Estágios @ Departamento de Engenharia Informática. Disponível em https://intra.estagios.dei.uc.pt/

[2] Scrum Guides. Disponível em http://www.scrumguides.org/

[3] Apache™ Cordova™. Disponível em http://cordova.apache.org/

[4] jQuery Mobile. Disponível em http://jquerymobile.com/

[5] The Dojo Toolkit. Disponível em http://dojotoolkit.org/

[6] Sencha Touch. Disponível em http://www.sencha.com/products/touch/

[7] PhoneGap™. Disponível em http://www.sencha.com/products/touch/

[8] AppBuilder® Cross-Platform Mobile Application Development IDE. Disponível em http://www.telerik.com/appbuilder

[9] Knockout. Disponivel em http://knockoutjs.com/

[10] AngularJS. Disponível em https://angularjs.org/

[11] Backbone.js. Disponível em http://backbonejs.org/

[12] Kendo Ui®. Disponível em http://www.telerik.com/kendo-ui

[13] Ionic. Disponível em http://ionicframework.com/

[14] Titanium™ Mobile Application Development. Disponível em http://www.appcelerator.com/titanium/

[15] Alloy. Disponível em http://www.appcelerator.com/platform/alloy/

[16] Xamarin™. Disponível em http://xamarin.com/

[17] Codename One. Disponível em http://www.codenameone.com/

[18] XMLVM. Disponível em http://xmlvm.org/overview/

[19] RhoMobile Suite. Disponível em http://rhomobile.com/

[20] MoSync. Disponível em http://www.mosync.com/

[21] AppGyver™. Disponível em http://www.appgyver.com/

[22] Adobe® Flex®. Disponível em http://www.adobe.com/pt/products/flex.html

[23] Sencha Ext JS. Disponível em http://www.sencha.com/products/extjs/

[24] Qt®. Disponível em http://www.qt.io/

[25] QML. Disponível em http://doc.qt.io/qt-5/qtqml-index.html

[26] SignalR. Disponível em http://www.asp.net/signalr

[27] Node.js™. Disponível em http://nodejs.org/

[28] Git™. Disponível em http://git-scm.com/

[29] Windows Phone® SDK. Disponível em https://dev.windows.com/en-us/develop/download-phone-sdk

[30] Android™ Studio and SDK. Disponível em http://developer.android.com/sdk/index.html

[31] Weinre. Disponível em http://people.apache.org/~pmuellr/weinre/docs/latest/

[32] RazorFlow. Disponível em https://razorflow.com/

[33] Flot. Disponível em http://www.flotcharts.org/

[34] flotr2. Disponível em http://www.humblesoftware.com/flotr2

[35] jqPlot Charts. Disponível em http://www.jqplot.com/

[36] amCharts. Disponível em http://www.amcharts.com/

[37] HighCharts™. Disponível em http://www.highcharts.com/

[38] FusionCharts. Disponível em http://www.fusioncharts.com/

[39] xCharts. Disponível em http://tenxer.github.io/xcharts/

[40] Telerik® Xamarin.Forms UI. Disponível em http://www.telerik.com/xamarin-ui

[41] OxyPlot. Disponível em http://oxyplot.org/

[42] Essential Studio for Xamarin™. Disponível em http://www.syncfusion.com/products/xamarin

[43] TeeChart™ for Xamarin.Forms. Disponível em http://www.steema.com/xamarinforms

[44] Infragistics™ Xamarin.Forms. Disponível em http://www.infragistics.com/infragistics-xamarin-forms

[45] Saving and Loading Data. Disponível em https://msdn.microsoft.com/en-us/library/gg680266(v=pandp.11).aspx

[46] Data Management. Disponível em https://developer.Apple.com/technologies/ios/data-management.html

[47] Data Storage Options. Disponível em http://developer.android.com/guide/topics/data/data-storage.html

[48] SQLite™. Disponível em http://www.sqlite.org/

[49] Cordova™/PhoneGap™ SQLitePlugin. Disponível em https://github.com/brodysoft/Cordova-SQLitePlugin

[50] SQLite™ for Windows Phone® 8.1. Disponível em https://visualstudiogallery.msdn.microsoft.com/5d97faf6-39e3-4048-a0bc-adde2af75d1b

[51] sqlite-net. Disponível em https://github.com/praeclarum/sqlite-net

[52] MVVMLight. Disponível em https://mvvmlight.codeplex.com/

[53] MVVMCross. Disponível em https://github.com/MvvmCross/MvvmCross

[54] Chosing a specific master page for each Windows Store App page. Disponível em http://loekvandenouweland.com/content/choosing-a-specific-master-page-for-each-windows-store-app-page

Mestrado em Engenharia Informática

Dissertação

Relatório Final

SoFly goes mobile

Anexos

Diogo Manuel Ferreira Mendonça

dmfm@student.dei.uc.pt

Orientador do departamento:

Fernando Barros

Orientador da empresa:

Virgílio Esteves

Data: 6 de Julho de 2015

Anexos ao relatório referente à disciplina de Estágio de Mestrado em Engenharia Informática, lecionado pelo Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Anexo 1

SoFly Overview

SoFly (HK) Ltd.

Outubro 2013

SOFLY ANALYTICS

Engage with your audience &
Get meaningful insights from social conversations

@SoFlyHQ | contact@sofly.tv | www.analytics.sofly.tv | SoFly (HK) Ltd.

#Major challenges of Social Television

With the rise of Social Media the importance of engaging your audience across these platforms is critical to a successful brand story. When it comes to television, the social aspect of the experience can dictate the success of your production. With all the activity surrounding televised broadcasts over these platforms, producing companies can have access to meaningful insights and crucial feedback. Relationships with your customers can be achieved and maintained over social platforms even though most of the users prefer to remain anonymous. This relationships are as important for your band or company as they are critical in today's digital world for your consumers and effectively drive audience's choices.

Reach your audience and drive conversations

Considering the information flow over Social Media, it's hard to get your audience to notice your message. An even harder job is to generate meaningful conversations that would propagate your brand over these platforms. Personal relationships are the key aspect to brand propagation and this can be achieved by actively interacting with your audience on a daily basis and constantly improving and adjusting your storytelling according to your audience taste.

Amplify your message and retain followers

By engaging with your audience over Social Media platforms you amplify their engagement directly with your production or broadcast, as well as, with the brands that are being advertised. This engagement alongside with the direct communication and personal relationship built over time are crucial factors to audience retention and the relationship between your social impact and your direct business results.

Create customer experience across all platforms

Currently over 50% of your audience is actively searching the World Wide Web for related content, which means that this behaviour makes it not only important but imperative to your business to track, measure and actively contribute to this process. Your audience is looking for an augmented and meaningful experience across every platform. Accounting for all this activity across various platforms and devices requires a simple and intuitive tool that could measure, analyse and help your team react in real-time to ongoing trends and events.

#SoFly Analytics

SoFly Analytics helps your team identify what topics are predominant in social conversations around your content and act upon this insights in order to reach new audience and enhance your engagement with existing viewers. Our platform empowers your team to collaborate and place their focus on aspects of social activity that require attention. The real-time analyses capability will provide an essential tool for adjusting your storytelling even during live broadcasts. The resulting improvement of your audience's viewing experience over diverse platforms and devices will benefit your general business results as well as provide you with new communication channels and advertisement opportunities.

Engage socially with a single solution

SoFly Analytics allows you to follow, measure, engage and analyse all of your social activity within one simple and intuitive tool letting your team focus their effort on what matters – your audience.

Listen and Publish

By tracking social conversations you'll stay up-to-date with current trends and topics that are being talked about your brand. Engaging with your audience is as important as understanding what drives them and for that matter SoFly Analytics provides you with all the required tools to schedule and publish your content, as well as understand when this content should be published in order to amplify your reach.

Collaborate and Moderate

Marketing is not a one man job and SoFly Analytics is ready to support your team by providing them a tool to communicate as well as collaborate and moderate the content that is being published. Scheduling and planning of upcoming broadcasts will save your marketing team precious time that can be used to engage with your audience during the broadcast.

Integrate social experience within your digital content

SoFly Analytics has a wide range of web plugins that can be used on your website to integrate social experience and provide your audience with augmented viewing experience across every platform and device.

Integrate social experience into your live broadcasts

SoFly Analytics allows you to exports moderated and filtered social feeds and integrate social experience into your live broadcasts providing your audience with real-time social experience while keeping the content profanity free and moderated.

Measure and analyse your impact

Knowing who your key influencers are and engaging with them will greatly contribute to overall propagation of your content over Social Media channels. SoFly Analytics provides you with easy and intuitive tools to achieve that in real-time. Alongside, we provide you with tools to analyse historical data in detail allowing you to drill down to specific metrics and post performances. Gained insights will greatly improve your understanding of your audience's taste and as a result allow you to adjust your storytelling and to stand out from the crowd.

Anexo 2

FactorX Case Study

SoFly (HK) Ltd.

Outubro 2013

SOFLY ANALYTICS

@Case Study

FACTOR X

Factor X engages social audience during live broadcasts and
Becomes the most trending show on Twitter in Portugal in 2013.

Case Study was developed in partnership with SIC – Sociedade Independente de Comunicação.

#Overview

SIC - Factor X

SOFLY ANALYTICS

Organization	SIC
Project	Factor X
Audience	TV viewers
Country	Portugal
Industry	Media
Sector	Live Broadcast
Products	SoFly Analytics
	SoFly Social Plugins

#KeyObjectives

Increase social activity around the real-time TV show in order to attract new fans, improve engagement and increase viewership.

#Strategy

Incorporate elements of real-time interaction of the audience with the show by using **SoFly Social Polls**; use of **SoFly Analytics** for Twitter stream analyses and data visualization.

#KeyResults

Factor X became the most trending Portuguese TV show on Twitter in 2013. Twitter activity around the show jumped 665% with the general increase in Twitter audience of 432%. Overall, potential impressions on Twitter had reached 5.3 million prints per broadcast (equivalent to 49% of the population of Portugal).

#Background

Gain social traction in emerging Twitter community

Social Television and the technological evolution seen around it during the past few years has changed and moulded in a whole new way the TV-viewing experience. Factor X, which aired in the fall of 2013, is a live TV Show focused on finding emerging talented singers around the country. Factor X production and marketing teams wanted to explore in-depth the social media opportunities in the emerging Twitter community in Portugal. As a result they wanted a platform capable to analyse, in real-time, Twitter interaction of the audience and empower communication and social integration during the show broadcast.

Involvement of SoFly Analytics and SoFly Social Plugins allowed to establish an easy and comprehensive communication channel between Twitter audience of the show and the production teams. Involvement of an interactive real-time Twitter content, active audience polls and social data visualization in the studio allowed the show to take the first place between most trended TV shows in Portugal on Twitter in 2013.

#Objectives

Establish a two-way communication channel with the audience

In order to capture the attention of the audience and to create an emerging experience, SoFly team had three goals in mind:

- Attract new fans and increase Twitter audience of the show in order to increase viewership.
- Create a real-time platform that would allow an easy understanding of social conversations.
- Empower social engagement over Twitter by integrating live data in the studio and on-air.

#Strategy

Comprehensive real-time social data delivered with Simplicity

In order to activate and engage the audience, production team encouraged TV-viewers to Tweet with a show-centric hashtag. This strategy was eagerly adopted once the social content was displayed on stage during a live broadcast.

SoFly Analytics provided the marketing team with real-time Twitter activity analyses, Twitter streams, audience activation metrics, most active and influent user lists, counters for contestant and judge mentions as well as respective sentimental analyses.

On-set, SoFly Social Plugins supplied the production team with the leader-boards of most active users on Twitter in order to display their photos on-air. An online poll was made to identify most popular judge and the results were displayed once the poll time was over.

Social integration and its adoption by the audience quickly made Factor X the most trending topic on Twitter in Portugal during the broadcasts.

#Results

The "Social Era" of live Television

During the period of time that social interaction was integrated into Factor X, Twitter activity jumped to a stunning 665% and the active audience has increased 432% once data visualisations were periodically shown on-air.

Average amount of tweets per broadcast rounded 37,000 tweets with a major spike of 1191 tweets per minute related to a key moment of a show. Number of potential impressions on Twitter also increased by 274% reaching 5,264,474 potential views (equivalent to 49% of population of Portugal).

The grows of interest and interaction could also be tracked by following the interval of time during the show that the amount of tweets per minute were larger than their average value and that systematically grew by 6.5 minutes per episode. This indicated a substantial and constant increase in time that the active audience remained engaged with the show.