UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

Origination – Crowdsourcing na Resolução de Desafios de Negócio

Pedro Miguel Dâmaso Lourenço

Mestrado em Engenharia Informática
Especialização em Engenharia de Software

Trabalho de Projeto orientado por:
Prof. Doutor André Nuno Carvalho Souto
e por Homero Machado Dos Santos Fonseca De Figueiredo

2018

Agradecimentos

Agradeço, aos meus pais por terem sido o principal suporte de todas as etapas ultrapassadas ao longo do meu percurso académico possibilitando sempre todas as condições para que me sentisse bem e feliz e para que me pudesse sempre focar no essencial sem nunca me faltar nada.

Agradeço, ao meu irmão pelo companheirismo e amizade que sempre nos uniu.

Agradeço, aos meus avós pelo carinho e dedicação que sempre me facultaram.

Agradeço, aos meus padrinhos, primos e restante família por todo o apoio e auxílio que me disponibilizaram sempre que foi necessário.

Agradeço, à minha namorada por me ter acompanhado ao longo de todo o percurso académico no que ao ensino secundário e universitário diz respeito e a parte do ensino básico, por ter sido um grande apoio e estar presente em várias fases importantes da minha vida.

Agradeço, aos meus orientadores por toda a disponibilidade, ajuda, orientação e conhecimento que me transmitiram durante o projeto.

Agradeço, aos meus amigos de infância, em especial ao Duarte Rei, pela a amizade que sempre nos uniu, pelos bons momentos ao longo de todos estes anos e por me terem sempre acompanhado.

Agradeço, ao Francisco Ferreira pela ajuda e inserção no início do meu percurso académico.

Agradeço, ao Miguel Correia pela a amizade e companhia durante o ensino universitário.

Agradeço, aos meus amigos da faculdade, em especial ao Rodrigo Rodrigues, Nuno Ramanlal, Miguel Saragoça, André Carocha, Micael Franco, João Duarte e Gustavo Silva, pela a amizade que criámos ao longo destes anos, pela ajuda mútua nos vários trabalhos realizados e pelos restantes momentos que partilhamos.

Agradeço, a todos os que contribuíram direta ou indiretamente para a realização deste trabalho.

Aos meus pais e irmão.

Resumo

Este relatório descreve o trabalho realizado no âmbito do estágio curricular na empresa Accenture. O objetivo deste Projeto de Engenharia Informática é o desenvolvimento de uma aplicação móvel que permita a geração de ideias espontâneas por parte de colaboradores da empresa e, também, a resolução de desafios de negócio, quer sejam estes existentes ou futuros, recorrendo a ideias e comentários publicados pelos mesmos usando esta aplicação.

Sabendo da inquestionável importância na atualidade dos *smartphones* e, inevitavelmente, das aplicações móveis surgiu a ideia e a necessidade de desenvolver esta aplicação fazendo uso de uma solução *cross-platform*. Este tipo de solução permite às empresas uma diminuição de custos e de alocação de recursos pois em vez de se desenvolver de forma nativa para as diversas plataformas, poder-se-á usufruir de um desenvolvimento único.

O trabalho descrito ao longo do documento foi composto por várias fases: (i) levantamento do estado da arte e análise de aplicações semelhantes, (ii) análise de requisitos através da elaboração de *user stories*, (iii) desenho da arquitetura do sistema, (iv) desenvolvimento das funcionalidades da aplicação, (v) avaliação das funcionalidades desenvolvidas através de vários tipos de testes.

A aplicação desenvolvida atingiu os objetivos previamente definidos.

Palavras-chave: *Smartphones* Aplicação Móvel *Cross-Platform* Desenvolvimento único

Abstract

This report describes the work done within the curriculum internship at Accenture. The objective of this Computer Engineering Project is to develop a mobile application that allows spontaneous ideas to be generated by the company's employees, as well as solving business challenges, whether existing or future, using ideas and comments published by them using this application.

Knowing the unquestionable importance in the present moment of smartphones and, inevitably, of the mobile applications came the idea and the need to develop this application making use of a cross-platform solution. This type of solution allows companies to reduce costs and allocate resources because instead of developing natively for several platforms, a single development can be used.

The work described throughout the document was composed of several phases: (i) survey of the state of the art and analysis of similar applications, (ii) analysis of requirements through the elaboration of user stories, (iii) design of the system architecture, iv) development of the functionalities of the application, (v) evaluation of the functionalities developed through several types of tests.

The application developed reached the objectives previously defined.

Keywords: Smartphones Mobile Application Cross-Platform Single development

Conteúdo

Lista de Figuras

Lista de Tabelas

Glossário

Termo	Designação
ACID	*Atomicity, Consistency, Isolation, Durability*
API	*Application Programming Interface*
Badge	Prémio (crachá) ganho quando um utilizador atinge determinadas metas
BASE	*Basically available, soft state, eventual consistency*
BCNF	*Boyce-Codd Normal Form* usado para aumentar a integridade e flexibilidade dos dados através da organização das colunas e tabelas assegurando a normalização da base de dados
Boolean	Tipo de dados que toma apenas dois valores, verdadeiro ou falso
Bottom Bar	Barra localizada na parte inferior do ecrã, menu da aplicação
Bugs	Nome comummente dado a um erro, falha ou falta no *software*
Byte Array	Tipo de dados que representa uma coleção de elementos do tipo de dados "byte"
Card View	Componente de UI utilizado para a apresentação de cada item de uma lista. Apresenta como conteúdo, normalmente, uma imagem e texto com *background* a branco assemelhando-se a um "cartão"
Checkbox	Componente que permite ao utilizador selecionar um ou mais elementos de uma lista, consoante o contexto
Constraint	Estrangeirismo usado para designar um constrangimento ou uma restrição
CSS	*Cascading Style Sheets*
Debug	Processo para detetar e remover falhas e erros do *software*
DELETE	Comando utilizado para apagar uma entidade identificada pelo URI
Dependency Injection	Quando se cria um objeto de uma interface é criado um objeto da classe que a implementa
DevOps	Metodologia que unifica o processo de desenvolvimento (Dev) com operação (Ops) de *software*. Aposta na automação, manutenção e monitorização constante das fases de construção de *software* e aumenta a quantidade, qualidade e a frequência das entregas
Double	Tipo de dados que representa um número fracionário

Editor	Elemento de UI que permite que o utilizador insira texto em múltiplas linhas
Entry	Designação dada ao elemento de UI que permite ao utilizador introduzir texto numa única linha
FCM	*Firebase Cloud Messaging*
For Each	Procedimento que consiste num ciclo executado algumas vezes sobre uma coleção de elementos
Foreign Key	Coluna de uma tabela que se liga aos dados da *Primary Key* de uma outra tabela
Frame	Componente de *design* que simula uma espécie de "moldura" aos elementos que estarão dentro desta
Framework	Estrutura que tem o intuito de servir como suporte ou guia para a construção de algo. São bibliotecas de *software* devido a consistirem em abstrações reutilizáveis de código com APIs definidas
GET	Comando que permite recuperar informação na forma de uma entidade identificada pelo URI
Grid	Designação de uma forma de colocação dos elementos dentro de uma grelha de linhas e colunas. Cada elemento é colocado numa determinada linha e coluna
HEAD	Comando idêntico ao *GET* mas, no entanto, o servidor não retorna um *message-body* na resposta
Hint	Designação de *placeholder* na biblioteca do Android
HTML	*Hypertext Markup Language*
HTTP	*Hypertext Transfer Protocol*
IDE	*Integrated Development Environment*
ImageSource	Elemento que tem o objetivo de mostrar ao utilizador uma imagem proveniente de um ficheiro, de uma *stream* ou da *web*
Insert	Operação utilizada para inserir uma ou mais linhas numa tabela. Cada linha inserida terá valores, indicados na operação, para as várias colunas da tabela
JSON	*JavaScript Object Notation*
List View	Componente de UI que mostra ao utilizador uma coleção de elementos
Mockup	Estrangeirismo para maquete ou esboço usado para simular um ecrã da aplicação
MVC	*Model-View-Controller*
MVP	*Model-View-Presenter*
MVVM	*Model-View-ViewModel*
NOSQL	Bases de dados não relacionais
Package	Estrangeirismo para designar pacote
PCL	*Portable Class Library*
PEI	Projeto de Engenharia Informática

Placeholder	Texto que aparece ao utilizador quando este ainda não tenha introduzido nada, por exemplo, numa *entry* ou *editor*
POST	Comando que indica ao servidor para aceitar a entidade incluída no pedido identificado pelo URI; utilizado para criar novas entidades
Primary Key	Termo que identifica univocamente cada entrada de uma tabela
Push Notification	Termo usado para designar uma mensagem entregue ao utilizador sem que exista um pedido específico por parte deste, podendo o dispositivo estar bloqueado e com a aplicação fechada
PUT	Comando que solicita que a entidade incluída seja armazenada conforme o pedido identificado pelo URI; utilizado para criar ou atualizar a entidade
Query	Estrangeirismo usado para designar pedidos de acesso a dados a uma ou várias tabelas da base de dados
Relative Layout	Forma de organização dos elementos em posições relativas. Um elemento é posicionado relativamente a outro(s) elemento(s) ou à área ocupada pelo *relative layout*
Resize	Estrangeirismo que indica a alteração do tamanho de uma imagem
REST	*REpresentative State Transfer*
ROI	*Return on Investment*
RPC	*Remote Procedure Call*
SDK	*Software development Kit*
Search Bar	Designação da barra onde um utilizador poderá inserir o texto que deseja pesquisar
Select	Operação utilizada para selecionar dados das tabelas da base de dados
SMTP	*Simple Mail Transfer Protocol*
SOA	*Service Oriented Architecture*
SOAP	*Simple Object Access Protocol*
SQL	*Structured Query Language*
SSMS	*Microsoft SQL Server Management Studio*
Stack Layout	Forma de organização dos elementos da página linearmente, como numa pilha, horizontal ou verticalmente
Stream	Sequência de dados. Pode ser usado para ler ou escrever dados de/para um ficheiro, por exemplo
String	Tipo de dados utilizado para representar texto
T-SQL	Transact-SQL
Tabbed Page (tab)	Permite ao utilizador selecionar o contexto que pretende visualizar mediante o título da *tab* selecionada. Por exemplo, se selecionar a *tab* de Comentários de um desafio apenas irá ver os comentários ao desafio em questão

Target	Versão alvo
Template	Modelo; padrão
Thread	Um programa pode ser dividido em tarefas que sejam executadas paralelamente. Cada tarefa é executada numa *thread* que corresponde a um fluxo sequencial único dentro do programa
Token	Cadeia de caracteres que identifica univocamente a instância da aplicação num dispositivo
Toolbar	Barra situada na parte superior do ecrã composta por ícones clicáveis
UI	*User Interface*
URI	*Uniform Resource Identifier*
URL	*Uniform Resource Locator*
Web Service	Sistema que envolve um servidor e um cliente. O servidor envia respostas ao cliente mediante o intuito do serviço invocado por este último
XAML	*eXtensible Application Markup Language*
XML	*eXtensible Markup Language*

Capítulo 1

Introdução

O *smartphone* é uma ferramenta, cada vez mais, indispensável no quotidiano. Estes dispositivos possuem mais funcionalidades que os telemóveis tradicionais, pois além de permitirem ao utilizador fazer chamadas e enviar mensagens, também permitem a utilização de aplicações móveis.
Em qualquer lugar público que se frequente é usual notar a utilização dos *smartphones* e de acordo com [8] há mais vendas destes dispositivos verificando-se também a tendência para a utilização *mobile* em detrimento dos *desktops* [9].
Com os estudos efetuados em [10] e [11] é fácil constatar a dependência que as pessoas têm dos *smartphones* e das aplicações móveis que podem ser instaladas nestes dispositivos. Uma das conclusões apresentadas em [11] é que o *smartphone* "domina" o dia-a-dia das pessoas.

O mundo empresarial, tomando partido do que acontece na sociedade relativamente às dependências indicadas acima, faz uso das tecnologias como veículo de comunicação e transmissão de informação fundamentais ao cumprimento da missão de uma organização. Neste Projeto de Engenharia Informática o objetivo passa pelo desenvolvimento de uma aplicação *mobile* capaz de ser executada pelos sistemas operativos Android e iOS (os sistemas operativos com maior utilização no que ao mercado *mobile* diz respeito [12]).

Esta aplicação terá o objetivo da resolução de desafios de negócio através das diversas ideias e comentários partilhados pelos colaboradores Accenture. Existe ainda uma componente de inovação onde é permitido aos utilizadores a publicação das suas ideias mais inspiradoras que poderão vir a ser desenvolvidas.
Para o desenvolvimento desta aplicação tivemos como base de inspiração aplicações descritas na secção 2.2 do Capítulo 2.

O aluno Gustavo Silva também irá desenvolver esta aplicação no âmbito do seu Projeto de Engenharia Informática.

1.1 Instituição de Acolhimento

A Accenture, tal como dito em [13], é uma empresa global de consultoria de gestão, tecnologias de informação e *outsourcing* que serve clientes em 120 países. É uma empresa com experiência em operar em mais de 40 indústrias, como por exemplo a indústria automóvel e industrial, na área da saúde ou na administração pública, e em todas as funções de negócio, estando organizada em 5 áreas: *Strategy*, *Consulting*, *Digital*, *Technology* e *Operations*.
Os valores fundamentais da Accenture passam por servir os seus clientes e por contribuírem para a comunidade como cidadãos corporativos responsáveis e assegurando que tudo é feito com integridade, tal como referido em [14].

1.2 Motivação

As empresas beneficiam quando tornam os seus funcionários como parte integrante das suas decisões. Isto promove não só o bem-estar na vida profissional destes como o aumento da sua produtividade e, desta maneira, levar à evolução da empresa devido à inevitável aproximação dos vários níveis desta.
Para além de ideias espontâneas que os funcionários possam ter é importante também que estes estejam ocorrentes dos desafios de negócio da sua empresa e que lhes seja dado a permissão de partilharem os seus pensamentos que por vezes possam levar a uma interpretação diferente do problema e a uma resolução mais rápida e/ou mais simples.

A principal motivação deste PEI passa pelo desenvolvimento de uma aplicação *mobile* para a resolução de desafios de negócio, quer sejam estes problemas já existentes ou futuros, através da publicação de ideias ou de comentários por parte dos vários intervenientes da Accenture de modo a que seja possível aproveitar o potencial de uma multidão (*crowd*), neste caso os funcionários da Accenture, de forma a promover um debate de ideias e de vários pontos de vista, daí o título deste PEI ser "*Origination – crowdsourcing* na resolução de desafios de negócio". Outra motivação passa pela publicação de ideias espontâneas que os utilizadores da aplicação tenham nos seus momentos de maior inspiração.

Inicialmente pretende-se uma utilização interna desta aplicação mas visando futuramente a integração desta aplicação nos clientes da empresa e, assim, fomentar a co-inovação , a implementação e o desenvolvimento de ideias de forma a antecipar tendências e marcar a diferença no mercado.

1.3 Objectivos

Os objetivos para este Projeto de Engenharia Informática passam pelo desenvolvimento de uma aplicação móvel e de todos os processos ligados ao seu desenvolvimento como análise de requisitos, *design*, implementação e fase de testes.

Esta aplicação móvel pretende que a Accenture e, posteriormente os seus clientes, possam resolver desafios de negócio através da ajuda dos seus colaboradores aproveitando-se igualmente para uma aproximação entre os diferentes cargos ocupados por estes colaboradores dentro da empresa.

Foi proposto que o desenvolvimento desta aplicação recorresse a uma solução *cross platform* que permita a geração de executáveis para os sistemas operativos Android e iOS através de um único código fonte. Outro dos objetivos deste projeto é a integração na dinâmica de um projeto *agile* em contexto profissional.

1.4 Contribuições

A minha participação neste projeto, desenvolvido com a metodologia *agile*, acompanhou todas as atividades do ciclo de vida do desenvolvimento, nomeadamente análise de requisitos, *design*, implementação e testes que avaliem o funcionamento correto da aplicação e a sua usabilidade por possíveis utilizadores.

No que toca à metodologia de projeto, participei nos diferentes atos associados a projetos geridos com recurso a metodologias ágeis, como a definição de *user stories*, no processo de *grooming* ou, também, em testes de usabilidade.

Do ponto de vista de execução do projeto, as minhas contribuições focaram-se em duas vertentes principais:

- Contribuições de âmbito funcional, nomeadamente as componentes:
  - Desafios;
  - Comentários;
  - Notificações;
  - Pesquisa;
  - Prémios;
  - Seguir/Não Seguir;
  - Mensagens Alerta.
- Contribuições de âmbito técnico, nomeadamente:
  - Decisões de desenho da arquitetura do sistema;

- Decisões relativas a tecnologias utilizadas;
- Desenvolvimento de uma camada de serviços que tinha o objetivo de intermediação entre a aplicação móvel e a base de dados;
- Integração do Firebase para armazenamento de imagens e envio de notificações para os dispositivos móveis.

De uma forma sucinta, o âmbito funcional para o qual contribuí na sua implementação passou pelos desafios de negócio, publicados por administradores, e que são resolvidos através da publicação de ideias e comentários por parte dos restantes utilizadores.

Para além destas funcionalidades *core* indicadas, a aplicação possui uma componente de gamificação, de criação de grupos para discussão de ideias num contexto privado, colocação de "gosto" em ideias e "follow" de grupos do qual o utilizador faça parte e de desafios e ideias que este considere interessantes.
A funcionalidade "Seguir/Não Seguir", corresponde à componente de "follow"e, está ligada à funcionalidade das "Notificações" tendo o intuito do utilizador, posteriormente, receber notificações caso exista algum evento num desafio, ideia ou grupo que siga. Por sua vez a funcionalidade "Mensagens Alerta" tem o intuito de informar o utilizador de que ocorreu algum evento como, por exemplo, ganhou pontos com alguma ação que tenha executado na aplicação. As funcionalidades "Seguir/Não Seguir" e "Mensagens Alerta" não faziam inicialmente parte do meu plano de trabalhos mas seguindo a metodologia *agile*, e à medida que fomos enriquecendo o *backlog* durante a execução do projecto, acabou por se prioritizar estas funcionalidades pelo que procedi à sua implementação.

Ao longo da execução do projeto procurei recolher *feedback* de diferentes colaboradores da empresa, tendo sido igualmente realizada uma fase de testes de usabilidade com a qual pretendemos medir a facilidade de utilização da aplicação desenvolvida.

A ideia subjacente a este projeto foi apresentada num fórum internacional da empresa, tendo sido muito bem recebida, e premiada com financiamento para evolução futura.

1.5 Estrutura do documento

O documento encontra-se organizado em 7 capítulos:

- Capítulo 1: apresentação do conteúdo do documento, onde se faz uma breve introdução ao projeto, explicando a motivação deste, a instituição de acolhimento onde foi desenvolvido, as contribuições e os objetivos deste PEI;
- Capítulo 2: consiste no levantamento do estado da arte onde são abordadas algumas tecnologias relacionadas com o projeto bem como os casos de estudo onde foram vistas algumas aplicações que serviram como inspiração para o desenvolvimento da aplicação móvel;

- Capítulo 3: apresenta a equipa do projeto, a aplicação, os possíveis perfis de utilizadores da aplicação, *user stories* e *mockups* de alguns ecrãs da aplicação, o modelo entidade-relação e a calendarização do projeto;
- Capítulo 4: arquitetura e estilo arquitetural do sistema, decisões técnicas, vista arquitetural do sistema e tecnologias utilizadas;
- Capítulo 5: descrição do processo relativo à implementação das funcionalidades;
- Capítulo 6: testes efetuados à aplicação móvel;
- Capítulo 7: resumo do projeto, problemas encontrados no desenvolvimento e trabalho futuro.

Capítulo 2

Trabalho relacionado

2.1 Estado da Arte

2.1.1 Abordagens de desenvolvimento

Na literatura é possível encontrar várias abordagens ao desenvolvimento de aplicações móveis. Neste trabalho dá-se particular ênfase ao desenvolvimento nativo e ao desenvolvimento *cross-platform*.

- Desenvolvimento nativo
  Segundo membros da RapidValue (ver referência [15]), as aplicações desenvolvidas de forma nativa visam ser utilizadas em dispositivos específicos que correm um determinado sistema operativo, como iOS ou Android, e que pretendem tirar o máximo partido das suas características [16].
  As aplicações nativas têm APIs mais "ricas" e são capazes de usar todas as potencialidades do dispositivo para as quais são desenvolvidas incluindo o uso de sensores ou o sistema de ficheiros [17]. Desta forma estas aplicações proporcionam ao utilizador uma melhor *user experience*.
  De acordo com L. Delia *et* al [18], as grandes vantagens deste tipo de desenvolvimento é que as aplicações resultantes não requerem continuamente conexão à internet, podem correr em *background* e enviar notificações ao utilizador. Por norma estas aplicações são disponibilizadas nas *app stores* apropriadas de cada plataforma.
  Para além das vantagens apresentadas acima e tal como referido em [15] e [16], as principais desvantagens do desenvolvimento de aplicações nativas passam pelo tempo despendido no desenvolvimento de código em diferentes linguagens, sendo que, para tal, é necessário que os profissionais tenham experiência em diferentes linguagens de programação, como Objective C no caso do iOS e em Java ou Kotlin (ver referência [19]) no caso do Android. Desta forma, uma solução comummente utilizada pelas empresas é ter duas equipas de desenvolvimento, uma por cada plataforma. Esta solução traduz-se em mais custos monetários para a empresa.

- Desenvolvimento *cross-platform*
  Segundo Raj e Tolety [1], o desenvolvimento por *cross-platform* pode ser subdividido em *web, hybrid, interpreted* e *cross compiled*. Esta última também se designa por *generated*, como referido em [6]. A descrição detalhada das abordagens que se apresenta a seguir foram escritas com base em [1].

  – Abordagem *Web*
    Uma aplicação desenvolvida com este tipo de abordagem não necessita de ser instalada no dispositivo móvel, sendo apenas necessário para a sua execução que o dispositivo possua um *web browser* de forma a executar a aplicação. Desta forma podemos afirmar que o produto resultante deste tipo de desenvolvimento é independente do sistema operativo do dispositivo móvel onde irá ser utilizado. O desenvolvimento destas aplicações é feito recorrendo a tecnologias *web* como HTML, CSS e JavaScript.

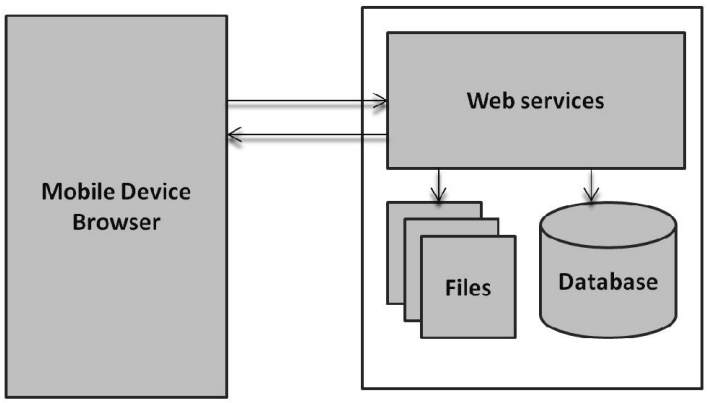


Figura 2.1: Arquitetura da Abordagem *Web*, retirado de [1]

Na Figura 2.1, o "*Mobile Device Browser*" representa o lado do cliente que executa pedidos ao servidor, representado no lado direito da figura, onde é processada toda a lógica de negócio da aplicação e é feito o acesso aos dados. As vantagens desta abordagem de desenvolvimento passam por não necessitarem da instalação da aplicação no dispositivo e portanto não serem necessários *updates* da aplicação, bem como a padronização dos *web browsers* que permitem a reutilização das interfaces em diversas plataformas.
Por sua vez nas desvantagens destaca-se o facto das aplicações desenvolvidas desta forma não poderem ser colocadas numa *app store* pois o seu acesso ocorre através de um *Uniform Resource Locator* (URL). Isto revela-se de facto como uma grande desvantagem pois os utilizadores, por hábito, procuram numa *app store* por qualquer tipo de aplicação que necessitem, diminuindo

o potencial uso da aplicação. Aliado a isto, o facto de ser exigido sempre conexão à internet revela-se também como uma desvantagem. Por fim este tipo de aplicações não pode aceder ao *hardware* e *software* dos dispositivos e portanto não pode tirar partido destas caraterísticas. Quem desenvolve este tipo de aplicações tem de ter em atenção as diferentes resoluções de ecrã.

– Abordagem Híbrida
De acordo com membros da Adobe PhoneGap [20], este tipo de abordagem é uma espécie de mistura entre a abordagem nativa e a abordagem *web* pois caracteriza-se por recorrer a tecnologias *web*, como HTML, JavaScript e CSS, e executa-se dentro de um *container* nativo no dispositivo móvel.
Este tipo de abordagem utiliza o mecanismo de renderização do *web browser* para mostrar o conteúdo HTML. Tal como pode ser visto, na Figura 2.2, as funcionalidades do dispositivo são expostas à aplicação através das APIs JavaScript. As aplicações híbridas ao invés das aplicações *web* necessitam de ser instaladas no dispositivo, e portanto são disponibilizadas numa *app store* mitigando uma das desvantagens da abordagem anterior.

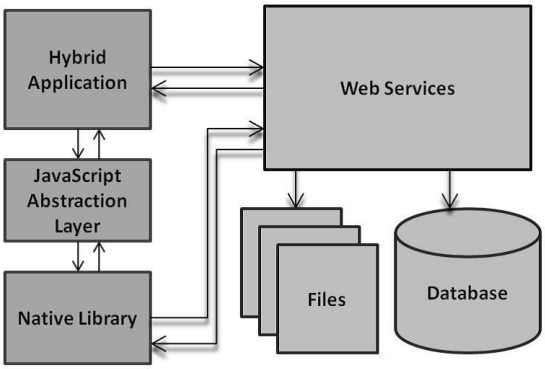


Figura 2.2: Arquitetura da Abordagem Híbrida, retirado de [1]

Assim, a principal vantagem desta abordagem passa pela reutilização da interface do utilizador entre diversas plataformas recorrendo a características específicas da plataforma nativa.
As desvantagens deste tipo de metodologia passam pelo menor desempenho quando comparado com as aplicações nativas pois a execução recorre ao *web browser*. Como a aplicação híbrida utiliza a camada de abstração JavaScript poderão existir vulnerabilidades. Apesar da interface do utilizador poder ser reutilizada nas diversas plataformas poderá não proporcionar a melhor *user experience*.

– Abordagem *Interpreted*
Este tipo de abordagem corresponde a aplicações que são instaladas no dis-

positivo sendo que existe um interpretador que executa o código apenas em tempo de execução, podendo levantar problemas ao nível de desempenho. Tal como na abordagem anterior as funcionalidades nativas do dispositivo são expostas à aplicação recorrendo a uma camada de abstração.

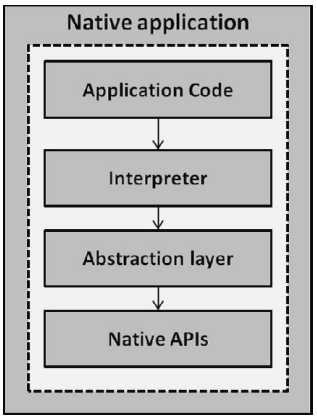


Figura 2.3: Arquitetura da Abordagem *Interpreted*, retirado de [1]

As principais vantagens desta abordagem passam pelo facto das interfaces serem nativas e pelo facto da lógica de negócio poder ser utilizada entre as várias plataformas. As características de *hardware* e *software* são manuseadas de acordo com a API da *framework*. A outra vantagem é que as aplicações resultantes deste tipo de metodologia são colocadas numa *app store*.
No que diz respeito às desvantagens destaca-se a dependência em relação ao nível de abstração da *framework* utilizada para a reutilização da interface e a dependência do desenvolvimento do conjunto de recursos fornecidos pela *framework* escolhida.

– Abordagem *Cross Compiled/Generated*
Este tipo de metodologia caracteriza-se pela presença de um *cross compiler* que é responsável pela geração do código executável nativo para a plataforma específica. Esta abordagem recorre a um só código fonte construído num único tipo de linguagem [21].
Nesta metodologia as vantagens passam pelo desenvolvimento nativo com acesso a todos os recursos do dispositivo, pois além do acesso às funcionalidades de *hardware* e *software* poderão ser acedidas todas as componentes da interface com o utilizador. Destaca-se ainda o bom desempenho das aplicações resultantes.
No que diz respeito às desvantagens evidencia-se a não reutilização da interface com o utilizador, a impossibilidade de utilização das características específicas de cada sistema operativo, como acesso à câmara ou ao GPS, pois o acesso a estas características difere de plataforma para plataforma. Este tipo

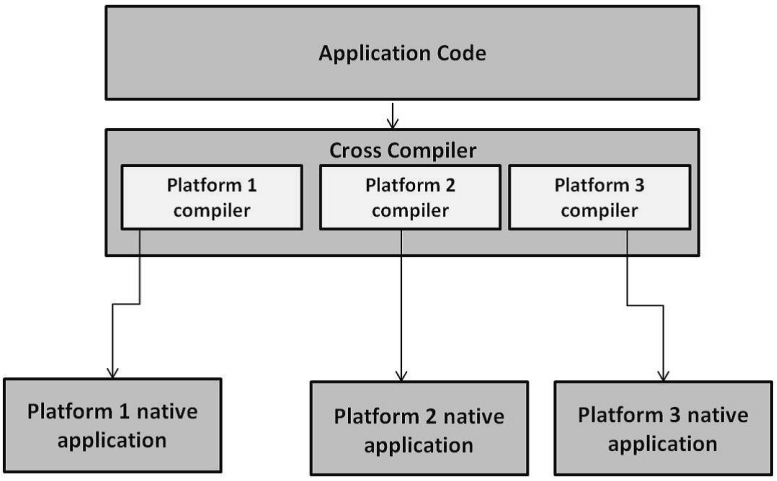


Figura 2.4: Arquitetura da Abordagem *Cross Compiled/Generated*, retirado de [1]

de metodologia é apropriada para aplicações simples, enquanto que aplicações mais sofisticadas deverão ser desenvolvidas através da abordagem nativa.

No que se segue, e seguindo o estudo feito em [6], apresenta-se uma comparação sucinta das diferentes abordagens de *cross platforms*. A análise feita tem em linha de conta os seguintes fatores:

- *Marketplace deployment* – que avalia a facilidade de colocar as aplicações nas *app stores* como *Google Play* ou *iTunes*;
- *Widespread technologies* – que avalia se as aplicações podem ser criadas recorrendo a tecnologias *widespread* como o JavaScript;
- *Hardware and data access* – que avalia o nível de acesso classificado em: sem acesso, acesso limitado ou acesso total aos dados e *hardware* do dispositivo onde a aplicação está instalada;
- *User interface and look & feel* – que avalia se a metodologia suporta os componentes nativos da interface, bem como a aparência das interfaces em comparação às interfaces nativas;
- *User-perceived performance* – que avalia o desempenho da aplicação (baixo, médio ou alto) de acordo com a perceção do utilizador final relativamente ao tempo de carregamento e à velocidade de execução comparativamente a uma aplicação nativa. [1]

[1] Em [6] é referido que este critério é uma estimativa empírica baseada na experiência prática feita pelos autores de [6].

	Web	***Hybrid***	***Interpreted***	***Generated***
Marketplace deployment	Não	Sim, mas não garantido*	Sim**	Sim**
Widespread technologies	Sim	Sim	Sim	Não
Hardware and data access	Limitado	Limitado	Limitado	Acesso Total
User interface and look & feel	Simulado	Simulado	Nativo	Nativo
User-perceived performance	Baixo	Médio	Médio	Alto

Tabela 2.1: Análise às diferentes abordagens, retirado de [6]

* Por exemplo a Apple poderá rejeitar conteúdo *web* ou alguns *links*
** As aplicações podem ser distribuídas sem dificuldade mas a experiência que a aplicação fornece deve obedecer às diretrizes de desenvolvimento da *app store*

Podemos concluir que não existe uma abordagem que possa ser considerada melhor que as outras. No entanto, a abordagem mais promissora é a *Cross Compiled/Generated* [6]. Obviamente, a seleção da abordagem a escolher depende de vários fatores como os mencionados acima.

Native vs Cross Platforms Solutions Segundo [17], o contexto da aplicação móvel indica qual a melhor abordagem a utilizar para o desenvolvimento. Caso a aplicação móvel seja maioritariamente utilizada para a interação com serviços e conteúdos online deverá ser evitada a abordagem nativa. Por outro lado, se a aplicação tiver o intuito de ser utilizada maioritariamente *offline*, a abordagem nativa oferece uma melhor *user experience*.

2.1.2 *Frameworks* para desenvolvimento *mobile*

Nesta secção apresentarei *frameworks* que têm o propósito de simplificar o desenvolvimento de aplicações móveis para múltiplas plataformas.

- *Xamarin.Forms*
  Segundo Procedi (ver referência [22]), o Xamarin é uma plataforma que permite criar aplicações nativas através de um código único na linguagem C# (*C-Sharp*), sendo apontada esta razão como uma vantagem. É uma plataforma que aborda o desenvolvimento *cross-platform Cross Compiled/Generated*, segundo L. Delia *et* al [18]. Contudo esta abordagem não garante necessariamente que o desenvolvimento não se torne repetido pois na camada da interface do utilizador é necessário desenvolver código específico para cada plataforma, o que não acontece no que diz respeito à lógica de negócio, acesso a dados ou à comunicação com o servidor pois nessas componentes é garantido que o código é desenvolvido apenas uma vez, de

acordo com Radi [23].
A plataforma Xamarin possui a *framework* Xamarin.Forms onde é permitido criar uma única interface com o utilizador através da disponibilização de uma biblioteca comum que torna as diferenças entre as plataformas transparentes a quem desenvolve o código. Para garantir essa transparência recorre-se a uma metalinguagem denominada XAML (*Extensible Markup Language*). A biblioteca comum disponibilizada é mais limitada relativamente às bibliotecas específicas do Android ou do iOS.
Assim, caso não exista necessidade de recorrer a componentes das bibliotecas específicas poderá todo o código tornar-se reutilizável.
Aplicações que utilizam o Xamarin.Forms conseguem reaproveitar entre 70-90% do código, segundo [24].

- PhoneGap
  Segundo Palmieri, Singh e Cicchetti [25], PhoneGap é uma *framework open-source* para desenvolvimento *cross-platform* de aplicações *mobile* híbridas. Utiliza linguagens como HTML, HTML5, CSS, CSS3, e JavaScript e a funcionalidade dos SDKs ao invés de linguagens menos conhecidas como Objective C [26]. Basicamente, PhoneGap é um "*wrapper*" que permite aos programadores desenvolverem aplicações nativas recorrendo a linguagens conhecidas, como as mencionadas acima.
  Ainda segundo [25], é referido que esta *framework* não fornece um *Integrated Development Environment* (IDE) para o desenvolvimento de aplicações. Estas aplicações deverão ter o código fonte escrito num IDE e aceder a esse código fonte noutros IDEs consoante a plataforma, como Eclipse para Android, Xcode para iOS. Esta abordagem não fornece um ambiente de desenvolvimento centralizado, referindo [25] que o esforço requerido para compilar o código fonte e produzir uma aplicação executável é alto.

- Ionic
  De acordo com Ravulavaru [27], Ionic é uma *framework open source* criada para o desenvolvimento de aplicações *mobile* híbridas. Recorre a HTML5, HTML, CSS e JavaScript bem como a ferramentas que proporcionam aplicações interativas. Utiliza AngularJS como a sua *framework* JavaScript. Ionic é *cross-platform* que permite criar aplicações móveis nativas, recorrendo a um único código base e poderão ser colocadas nas *app stores* da plataforma específica [28]. Para além disso tem uma boa integração com a API Cordova permitindo o acesso a esta API usando uma biblioteca como ngCordova integrando isto com as componentes da interface do utilizador do Ionic. Segundo membros da Ionic ([28] e [29]), Cordova é um meio de empacotar HTML, CSS e JavaScript em aplicações *mobile* ou num dispositivo

desktop fornecendo uma arquitetura *plugin* que acede a funcionalidades nativas que vão mais além do JavaScript e que é executado pelo navegador *web*. Desta forma permite-se o *deploy* nativo das aplicações móveis.

2.1.3 Padrões Arquiteturais

É crítico que exista uma boa estruturação quando se desenvolve *software* e, para que tal aconteça, é inevitável que se recorra a padrões arquiteturais. O processo de escolha do padrão arquitetural para a organização do código deverá ocorrer antes do início da fase de desenvolvimento [4].
Nesta secção abordam-se os padrões arquiteturais MVVM, MVC e MVP.

- Padrão MVC
  Este padrão foi um dos padrões iniciais a ser concebido e é composto por três componentes: *View*, *Controller* e *Model*. Este padrão de acordo com [2], é fundamental para a separação entre a lógica associada à interface do utilizador com a lógica de negócio.
  O *Model* gere os dados e a lógica de negócio, e segundo [2], o *Model* responde a pedidos de informação sobre o seu estado geralmente por parte da *View* e responde a instruções para a mudança de estado por parte do *Controller*.
  A *View* tem como principal objetivo apresentar os dados ao utilizador, enquanto que o *Controller* recebe os pedidos por parte do utilizador e informa o *Model* e/ou a *View* para as mudanças necessárias.

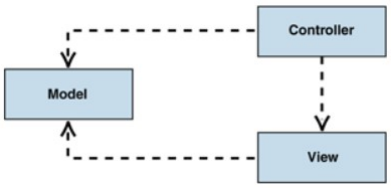


Figura 2.5: Estrutura do padrão MVC, retirada de [2]

Como pode ser visto na Figura 2.5, o *Controller* e a *View* dependem ambos do *Model*, sendo este independente das outras componentes. Este facto é bastante importante pois permite a testabilidade e a construção do *Model* independentemente da interface com o utilizador. Segundo [2] a separação entre a *View* e o *Controller* é secundária e existem muitas *frameworks* que implementam os objetivos dessas componentes como um único objeto. Contudo, nas aplicações *web* a separação entre a *View* e o *Controller* encontra-se bem definida.

- Padrão MVVM (ver referência [30])
  Este padrão é utilizado em plataformas de aplicações XAML. Este padrão é um

refinamento do padrão MVC e tem como objetivo separar a interface com o utilizador da sua lógica. Existem três componentes principais que compõem este padrão, a *View*, *View Model* e o *Model*.

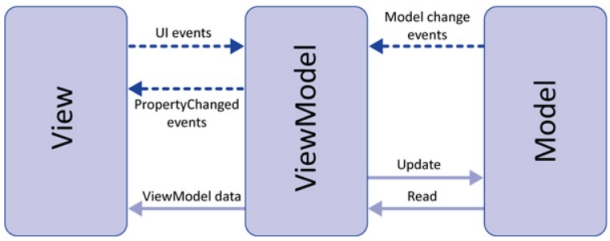


Figura 2.6: Estrutura do padrão MVVM, retirada de [3]

Estas componentes poderão ser desenvolvidas e testadas independentemente umas das outras.
A *View* é responsável por definir a aparência do que o utilizador vê no ecrã e é, normalmente, definida recorrendo a XAML. No entanto poderá conter código por trás onde não está relacionada lógica de negócio. Cada *View* poderá ter o seu próprio *View Model* sendo que a *View* obtém os dados a partir do *View Model* através do recurso de *bindings* ou à invocação de métodos implementados no *View Model*. A *View* poderá ser alterada em tempo de execução quando os controlos da interface do utilizador respondem às propriedades do *View Model* ocorrendo isto devido a eventos de notificação para mudança.
O *Model* é uma implementação do modelo de domínio da aplicação que inclui os dados com lógica de validação e lógica de negócio.
O *View Model* atua como um intermediário entre o *Model* e a *View* e tem como responsabilidade gerir a lógica da apresentação da informação ao utilizador, ou seja, a lógica da visualização. Esta componente interage com a componente *Model* através da invocação de métodos definidos nas classes dessa mesma componente. Ao recuperar os dados do *Model* a componente *View Model* fornece os dados à *View* para que sejam apresentados ao utilizador. Esta componente é também responsável por responder a ações executadas na *View* por parte dos utilizadores, como o *click* num botão.

- Padrão MVP
  Este padrão, segundo [4], é uma variação do padrão MVC. Este padrão foi projetado para automação de testes com o objetivo de aumentar a quantidade de código que pode ser testado. As componentes deste padrão são a *View*, o *Presenter* e o *Model*.
  A *View* é a componente responsável pela exibição da informação dos dados do *Model* e encaminha os pedidos do utilizador para o *Presenter*.
  O *Presenter* tem como função a ligação entre a *View* e o *Model* respondendo aos

pedidos da *View* e lendo ou alterando os dados no *Model*, de modo a atualizar a *View* consoante o contexto do pedido.
O *Model* representa os dados que serão mostrados na *View*.

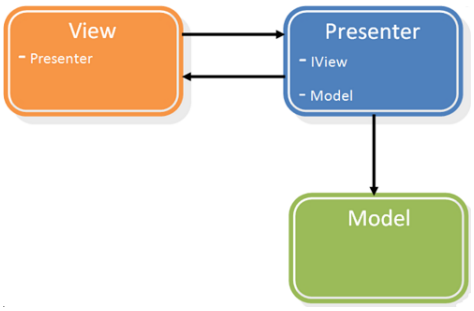


Figura 2.7: Estrutura do padrão MVP, retirada de [4]

Este padrão é complexo para ser aplicado em soluções avançadas mas também poderá não ser a escolha ideal para soluções simples.

2.1.4 REST e SOAP

Aplicações cliente-servidor necessitam que existam protocolos que estabeleçam a comunicação entre ambas as partes. Por exemplo o cliente necessita de aceder aos *web services* para obter os dados de que precisa e o servidor de "devolver" esses mesmos dados. Para tal existem duas formas que respondem a esta necessidade: a arquitetura REST e o protocolo SOAP.

- REST
  Segundo Christensen [31], REST é um acrónimo de *REpresentative State Transfer*.
  Os *web services* baseados neste estilo arquitetural são fáceis de serem consumidos em plataformas móveis porque o cliente e o servidor apenas necessitam de um único protocolo que executa a invocação e a resposta.
  Assim os principais aspetos que tornam a arquitetura REST desejável em aplicações móveis passam por estas serem *stateless* minimizando possíveis problemas que advenham da volatilidade da conexão. É baseado em *URLs* o que torna fácil a invocação de *web services*, sendo as respostas baseadas em HTTP e a entrega poder ser feita de forma muito sucinta, ideal para ambientes com memória restrita.
  Os *web services* baseados em REST geralmente utilizam o protocolo HTTP e os correspondentes verbos deste protocolo, como *HEAD, POST, PUT, GET e DELETE*.
  Christensen afirma que REST e HTTP são ideais para redes instáveis.

- SOAP
  SOAP é um acrónimo de *Simple Object Access Protocol* e é um protocolo que se

baseia na linguagem XML [32]. Jorge em [33] afirma que a linguagem XML serve para estruturar e formatar mensagens permitindo a troca de mensagens entre plataformas descentralizadas e distribuídas.
De acordo com [34], SOAP define o conceito de envelope utilizado para a troca de mensagens. O envelope é dividido num *header*, onde constam informações relativas à autenticação, codificação dos dados e à maneira como o recetor da mensagem a deverá processar, e num *body* que é composto pela mensagem.
De acordo com [32] e [33], SOAP é caracterizado por ser independente da plataforma e do sistema operativo, da linguagem utilizada e do protocolo utilizado para a troca de mensagens como, o protocolo HTTP, SMTP ou RPC.

2.1.5 Bases de dados relacionais e não relacionais

Segundo Ramakrishnan e Gehrke [35], as bases de dados relacionais são uma coleção de uma ou mais relações em que cada uma delas corresponde a uma tabela com linhas e colunas. As bases de dados relacionais utilizam SQL (*Structured Query Language*) como linguagem padrão, possuem um esquema pré-definido e utilizam o sistema ACID (*atomicity, consistency, isolation* e *durability*) para transações ([35] e [36]). A relação entre as tabelas pode ser classificada em: (ver [35])

- Um para um: um registo na tabela relaciona-se com outro registo de uma outra tabela;
- Um para muitos: um registo na tabela relaciona-se com muitos registos numa outra tabela;
- Muitos para muitos: mais do que um registo na tabela relaciona mais do que um registo numa outra tabela.

Segundo Nayak, Poriya e Poojary em [37], as bases de dados não relacionais (NOSQL) não garantem as propriedades ACID mas garantem as propriedades BASE (*basically available, soft state, eventual consistency*) dando primazia à disponibilidade e escalabilidade, deixando a consistência para segundo plano.
Este tipo de base de dados pode ser dividido em 5 tipos: *Key-Value Store Databases, Column-Oriented Databases, Document Store Databases, Graph Databases* e *Object Oriented Databases* (ver referência [37]).

Ainda seguindo os autores da referência [37], as vantagens de NOSQL relativamente ao modelo relacional passam por fornecer um amplo domínio de modelos de dados, ser facilmente escalável, não ser necessário administradores de base de dados, as bases de dados relativas ao modelo não relacional são programadas para gerir falhas relativas a *hardware*, serem mais rápidas, mais eficientes e flexíveis e evoluírem a um grande ritmo.

No contraponto, as desvantagens de NOSQL em comparação com bases de dados relacionais são a inexistência de *query language* padrão, serem imaturas, não garantirem as propriedades ACID e a sua manutenção ser mais difícil.

2.1.6 A metodologia *Agile* e as metodologias tradicionais no desenvolvimento de *software*

O ciclo de vida do desenvolvimento de *software* é o processo de construção e manutenção dos sistemas de *software* [38]. Este ciclo inclui as fases que ocorrem antes do desenvolvimento, análise de requisitos, até às fases posteriores ao desenvolvimento como o caso da fase de testes.

Existem duas metodologias no que diz respeito ao ciclo de vida do desenvolvimento de *software*: as metodologias tradicionais e a metodologia *Agile* [7].

- Metodologias Tradicionais
  As metodologias *Waterfall* e *V-Model* são exemplos deste tipo de metodologias e caracterizam-se por quatro fases:

  1. Requisitos do projeto e determinação do tempo que será necessário para implementar todas as fases de desenvolvimento;
  2. Fase de *design* e planeamento da arquitetura;
  3. Fase de desenvolvimento, onde o objetivo é atingir os requisitos anteriormente levantados. Esta fase é dividida em pequenas tarefas distribuídas pelas várias equipas encarregues do desenvolvimento;
  4. Depois do projeto estar de acordo com os seus requisitos existe a inclusão do cliente na fase de testes.

  Cada uma das fases só se inicia depois do término da anterior sendo que o produto é entregue ao cliente após verificar na fase de testes de que não existem problemas. Este tipo de metodologias dependem de processos pré-determinados e de documentação relativa aos processos efetuados à medida que o projeto avança e que se projeta o que será tratado nas fases seguintes [39].
  Poderão existir problemas quando se pretende efetuar alterações na fase de desenvolvimento [7]. Outra desvantagem deste tipo de metodologia é o facto da entrega do sistema ser em "*one-shot*" e que caso aconteça algum problema na fase de testes poderá ter que ser alterado, no pior dos casos, todo o módulo onde aconteceu o problema.
  Os utilizadores depois do produto ser entregue poderão querer fazer alterações no produto que impliquem lidar com problemas de compatibilidade e integridade do *software*.

No entanto recorrendo a este tipo de metodologias torna-se fácil determinar os custos, a calendarização e a alocação de recursos para o projeto [40].

- Metodologia *Agile*
  Este tipo de metodologia baseia-se no desenvolvimento iterativo e incremental em que cada fase do ciclo de vida de desenvolvimento de *software* é revisitada várias vezes. Segundo Szalvay [41], o *software* é melhorado de acordo com o *feedback* dado pelo cliente. Neste tipo de desenvolvimento existem quatro fatores determinantes:

  1. O envolvimento do cliente logo nas fases iniciais do projeto;
  2. Desenvolvimento iterativo;
  3. Equipas auto-organizadas;
  4. Adaptação à mudança.

  De acordo com os autores de [42] existem seis métodos ágeis: *Scrum*, metodologias *Crystal*, *dynamic software development method*, *feature-driven development*, *lean software development*, e *extreme programming*.
  Em [43] refere-se que os métodos ágeis colocam no retorno de investimento (ROI) do negócio a sua principal prioridade. Este tipo de métodos, de acordo com Petersen e Wohlin [44], também dão uma grande importância à comunicação e ao envolvimento do cliente.
  Recorrendo a este tipo de metodologia, o levantamento de requisitos é efetuado de uma outra forma em que se recorre às frequentes demonstrações e entregas de *software* para que os clientes consigam dar *feedback*. Esta opinião é usada para o refinamento dos requisitos levantados antes da versão demonstrada do *software* aquando desse *feedback*. A abordagem *Agile* permite o início do desenvolvimento antes que todos os requisitos sejam conhecidos. Estas frequentes demonstrações também são utilizadas para que sejam apresentadas novas funcionalidades desenvolvidas ao cliente. Assim o cliente pode indicar novas funcionalidades que necessite para a aplicação ainda durante a fase de desenvolvimento. Utilizando o refinamento dos requisitos, o desenvolvimento *Agile* entrega um produto que melhor serve as necessidades do cliente.
  Segundo [7], o facto deste tipo de desenvolvimento ser aberto a mudanças incrementais abre duas questões relativas ao *design*, como a rigidez e a mobilidade. A rigidez refere-se a uma mudança no sistema implicar outras mudanças noutros módulos, enquanto que a mobilidade refere a incapacidade do sistema em encapsular componentes que podem ser reutilizados, porque isso implica esforço e risco.
  Uma das desvantagens deste tipo de abordagem segundo Vijayasarathy e Dan [45]

é a pouca quantidade de documentação afirmando que o código deve atuar como documento o que causa muitos comentários nos ficheiros respetivos por parte dos programadores. Isto poderá levantar problemas relativamente à dificuldade que novos membros de uma equipa de desenvolvimento tenham em perceber o projeto e, consequentemente, a um atraso na iteração, podendo aumentar o custo de desenvolvimento.

A tabela seguinte sumaria a comparação entre as duas abordagens.

	Agile	**Tradicional**
Requisitos do Utilizador	Aquisição iterativa	Requisitos detalhados do utilizador bem definidos antes da implementação do código
Custo de alterações	Baixa	Alto
Direção do desenvolvimento	Facilidade de mudança	Fixo
Testing	Em todas as iterações	Após a fase de desenvolvimento do código
Envolvimento do cliente	Alto	Baixo
Qualidade extra requerida para os programadores	*Skills* interpessoais e conhecimento básico do negócio	Nada em particular
Escala adequada do projeto	Baixo a média-escala	Alta-escala

Tabela 2.2: Comparação das duas abordagens, retirado de [7]

2.2 Casos de Estudo

- Crowdicity
  A Crowdicity é um *software* de gestão de ideias para organizações hospedado na *cloud*. É uma plataforma *web-based* mas pode ser acedido de forma *mobile* por iOS ou Android, pois segundo [46] o *design* responsivo torna a Crowdicity compatível com os dispositivos móveis.
  O seu objetivo é através das suas funcionalidades permitirem a inovação dos seus clientes, aproveitando as ideias das pessoas afetas ao cliente ou de qualquer outra pessoa que possa ajudar a empresa a melhorar.
  A Crowdicity é composta por várias funcionalidades como poderá ser verificado em [47] sendo que as funcionalidades que se evidenciam são:

  – Permitir o lançamento de desafios aos quais os utilizadores respondem com ideias. Os desafios poderão ser divididos por categorias ou temas e poderão ser privados ou públicos, têm uma data de início e fim e são divididos por etapas. É permitido aos utilizadores que marquem os desafios como favoritos;

  – O utilizador pode partilhar ideias para a comunidade ou para equipas específicas. As ideias poderão receber *feedback* de diferentes formas como através de “gosto”, “não gosto”, por uma avaliação quantitativa baseada em

"estrelas" (1-5) ou através de comentários. As ideias poderão ser fechadas para votos e comentários;

- Abrangem o conceito de comunidade fechada onde apenas utilizadores autorizados podem ver o seu conteúdo. Numa comunidade aberta é permitido a qualquer utilizador juntar-se e contribuir com ideias;
- Existe um "*feed* de notícias" onde é oferecido ao utilizador em tempo real a atividade da comunidade, como por exemplo a indicação de que alguém votou numa ideia ou que existiram atualizações num desafio que o utilizador em questão segue;
- Cada utilizador tem um perfil e é permitido pesquisar pelos perfis de outros utilizadores e visualizar a sua atividade, ideias e contribuições. Para além disto poderá-se conectar a outros utilizadores;
- É permitido pesquisar ideias por filtros, como ver as ideias com mais votos ou as ideias mais comentadas;
- Para promoverem a participação nos desafios através do contributo de ideias, a Crowdicity atribui pontos ao contributo dos utilizadores e elaboram um *ranking* com os utilizadores, podendo estes pontos serem traduzidos em prémios para os utilizadores;
- Têm um mecanismo de notificações, nomeadamente através do envio de emails para manter a comunidade envolvida e informada permitindo a um utilizador receber alertas quando um utilizador, por exemplo, responde a um comentário seu;
- Uma das funcionalidades mais populares da Crowdicity é um mecanismo integrado de *blog* onde os utilizadores poderão criar o seu *blog* e colocar vídeos, imagens ou anunciar novos desafios;
- Esta plataforma permite a customização aos utilizadores, podendo estes adicionarem páginas à sua comunidade, como por exemplo uma página "Conheça a Equipa";
- Existem administradores e moderadores da comunidade. Possuem ferramentas de moderação que permitem editar ou remover ideias, alterar o estado de um desafio e supervisionar todos os aspetos da execução de um desafio;
- É possível construir uma base de dados de ideias onde se pode arquivar ideias ou desafios;
- É permitido entrar na plataforma fazendo uso das contas que o utilizador tenha nas redes sociais.

- OpenideaL
  Tal como dito em [48], OpenideaL é um sistema de gestão de ideias adaptado para

empresas e organizações interessadas em partilharem decisões estratégicas e um planeamento futuro de produtos e serviços com os seus clientes. É uma ferramenta que permite a quem toma decisões poder ter uma nova visão do problema através do contributo de empregados da organização ou de outras pessoas que poderão não ter qualquer ligação à organização.
Adapta-se às necessidades específicas de uma organização, comunicando com aplicações externas como o Facebook ou com aplicações internas de uma organização e permite o acesso por múltiplas plataformas devido a ser responsivo.
Esta aplicação tem como principal foco as ideias e a discussão destas, deixando para segundo plano os desafios [49], permitindo apenas que um desafio esteja ativo em cada momento. Contudo, as possíveis respostas a um determinado desafio são ideias.
Para ver todas as funcionalidades desta plataforma poderá consultar [49] sendo que estas são as principais funcionalidades:

  - Adicionar ideias de acordo com um *template* composto por título, descrição e categoria;
  - Página da ideia onde aparece os detalhes da ideia;
  - Editar uma ideia onde o administrador pode editar uma ideia ou removê-la;
  - Visualização de comentários a ideias;
  - Adição de *tags* a uma lista predeterminada. O administrador poderá editar esta lista;
  - Possui um mecanismo que recorre a um algoritmo para verificar as ideias com maior relevância tomando em conta o número de comentários, votos e o tempo passado desde a última atividade;
  - O utilizador poderá receber notificações sobre as suas ideias preferidas ou dos últimos *updates* que aconteceram nas suas ideias;
  - Permissão para o utilizador votar "gosto" ou "não gosto" em cada ideia. Cada membro pode votar uma única vez em cada ideia;
  - O utilizador recebe pontos por cada adição de ideias, comentários ou votos que faça.

- OpenIDEO Challenges
  De acordo com [50], a OpenIDEO é uma plataforma onde é possível construir comunidades que trabalhem conjuntamente de forma a construírem soluções para os grandes *challenges* mundiais. Esta plataforma associa-se com organizações líderes que impulsionam a colaboração, inovação e impacto sobre os mais difíceis problemas existentes no mundo através do lançamento de desafios, programas e outras

experiências personalizadas.

Os desafios ocorrem normalmente durante três a cinco meses e caracteriza-se por ser um processo colaborativo que foca um determinado assunto criando um espaço para os membros da comunidade poderem contribuir, refinar e prototipar soluções através da partilha de ideias, segundo [51].

Seguindo [51], as melhores ideias dos desafios são escolhidas pelo patrocinador do desafio e pelos membros da equipa do OpenIDEO tomando em atenção o tema do desafio e os seus critérios, estando estes critérios presentes no resumo do desafio.

Quando um desafio é terminado algumas ideias são identificadas e poderão ter o apoio dos patrocinadores do desafio para que sejam implementadas, estando desta maneira implícito o conceito de *gamification* nesta plataforma.

Capítulo 3

Análise

3.1 A Equipa e a metodologia escolhida no desenvolvimento de *software*

A equipa que está afeta ao projeto é composta por mim, pelo Gustavo Silva e pelo Homero Figueiredo. A equipa decidiu que o desenvolvimento de *software* deveria estar de acordo com a abordagem *Agile*, pois como se perspetiva a necessidade de mudanças, as alterações tornam-se mais fáceis e com custos menores. Para além destes motivos, concordamos que o desenvolvimento iterativo/incremental seria o mais adequado pois, desse modo, existirão demonstrações frequentes e entregas de *software* que ocorrerão no fim de cada *sprint*.

Inicialmente, foi decidido pelo gestor do projeto, Homero Figueiredo, a escolha da metodologia *Scrum* relativa à abordagem *Agile* em que existem demonstrações da aplicação no fim de cada *sprint* sendo que geralmente os *sprints* têm duração de cerca de duas semanas. Num cenário típico *Agile*, o papel de *Product Owner* cabe ao cliente, enquanto que o *Scrum Master* corresponde ao integrador. No entanto, como se trata de um projeto inicialmente para uso interno foi decidido a correspondência dos dois papéis ao gestor do projeto. Ele foi o responsável pela tomada de decisões relativamente à escolha de *frameworks* a serem utilizadas, à escolha de atividades a serem realizadas em cada *sprint* e também por desenvolver trabalho em prol da superação de qualquer impedimento que a equipa possa ter. A equipa *Scrum* foi composta por mim e pelo Gustavo Silva. Estamos ligados a todas as fases do projeto, como o levantamento de requisitos ou o desenvolvimento de software.

Posteriormente com o decorrer do projeto tendemos para a metodologia *Kanban* da abordagem *Agile*. Devido à equipa ser composta apenas por dois elementos e através da constante comunicação entre estes eram selecionadas as *user stories* mais prioritárias em cada momento do desenvolvimento do *software*. Após a implementação das *user stories* escolhidas, selecionávamos, de novo, as mais prioritárias e procedíamos à sua implementação ou voltávamos a *user stories* anteriormente implementadas que necessita-

vam de alguma alteração. Todas as *user stories* encontravam-se num ficheiro de *backlog* que fomos mantendo atualizado e que era utilizado para a seleção das *user stories*.

Para o controlo de versões foi utilizada a plataforma "Git" de modo a que a equipa *Scrum* possa desenvolver atividades separadamente e, posteriormente, existir a junção do código elaborado.

3.1.1 Experiência Accenture com metodologia *Agile*

No contexto de projetos com clientes, a Accenture tem experiência na aplicação da metodologia *Agile*, considerando a aplicação de *DevOps*.

Entre os principais fatores de sucesso desta metodologia destacam-se a comunicação e colaboração frequente entre as equipas de negócio durante a implementação, disponibilização de uma solução modular funcional, qualidade nas entregas do *software* devido à aplicação da metodologia *scrum* ou a disponibilização de *user stories* que são elementos críticos com base ao desenvolvimento.

3.2 A Aplicação

A aplicação tem como objetivo a centralização de ideias que visam a resolução de desafios de negócio fomentando um debate de diversos pontos de vista dos colaboradores da empresa. Estes pontos de vista serão concretizados através da criação de ideias ou de comentários.

Nesta aplicação, os utilizadores poderão publicar e partilhar ideias espontâneas (ainda que não aliadas à resolução de desafios) que possam ter nos seus momentos de maior inspiração.

Para além das funcionalidades indicadas e que são o foco da aplicação existem ainda componentes de gamificação através de prémios que os utilizadores poderão obter devido à sua interação com a aplicação, uma componente de grupos onde os utilizadores se poderão "juntar"para o debate de ideias que prentendam ser mais privadas ou uma componente de notificações que indicam ao utilizador de que algum evento ocorreu num determinado desafio, ideia ou grupo que este siga.

As funcionalidades desta aplicação são:

- *Login*/Autenticação e autorização;
- Desafios;
- Ideias;
- Comentários;
- Grupos;

- Perfil;
- Notificações;
- Mensagens Alerta;
- Pesquisa;
- *Ratings*;
- Seguir/Não Seguir;
- Prémios.

3.3 Levantamento de Requisitos

Todas as atividades realizadas nesta fase do Projeto foram elaboradas com o Gustavo Silva.

3.3.1 *Personas*

Para iniciar o projeto foi pedido que identificássemos os perfis de possíveis utilizadores da aplicação. Para tal, foram criados 5 potenciais perfis, todos estes com base em funcionários de diferentes áreas e com necessidades e objetivos de negócio distintos.

Estas *personas* foram "marcados" com algo que os diferencia, como por exemplo "*Ideator*" devido a ser um utilizador mais afeto à criação de ideias ou "*Challenger*" devido à organização de desafios e à criação destes.

A título ilustrativo apresento 2 dos perfis criados:

Rui Silva – Consultor
"Ideator"
Rui é uma pessoa muito entusiasta que tem sempre muitas ideias. Ele coloca uma ideia todas as semanas, vota e comenta as ideias das outras pessoas. Pode-se dizer que ele é um grande dinamizador da empresa.

Necessidades do Utilizador:

- Criar ideias;
- Gerir ideias (Progresso, votos, comentários);
- Ver detalhes de uma ideia;
- Ver perfil dos outros utilizadores;

- Ver novas ideias.

Objetivos de Negócio:

- Gerir as expetativas dos utilizadores;
- Providenciar uma forma de desenvolver/criar ideias;
- Fazer com que as pessoas conheçam as ideias das outras pessoas e poderem interagir e colaborar no progresso dessas ideias.

João Reis – IT
"Challenger"
O João organiza os desafios e é responsável por criar novos desafios e associar prémios aos desafios.

Necessidades do Utilizador:

- Criar um novo desafio;
- Colocar prémios no desafio;
- Convidar pessoas para um desafio.

Objetivos de Negócio:

- Promover a ajuda nos problemas das outras pessoas;
- Motivar as pessoas a participar nos desafios através de incentivos, como prémios;
- Promover a competição para melhorar a vida das pessoas;
- Gerar ideias/soluções sobre um tema de forma a responder às necessidades de forma rápida.

3.3.2 *User Stories*

Para o levantamento de requisitos foram efetuadas cerca de 65 *user stories*, considerando todos os possíveis exemplos da utilização da aplicação e de que forma deveria o sistema responder ao utilizador, bem como os requisitos funcionais e não funcionais existentes em cada uma destas. Estas *user stories* foram elaboradas para as *personas* identificadas acima. De seguida apresento três *user stories* elaboradas a título ilustrativo.

- *User Story*: US0019
  Como *Challenger* eu quero criar um novo desafio para melhorar a vida das outras pessoas.

- Critérios de Aceitação:
  - Existir uma secção no menu denominada ”Desafios”;
  - Na página onde aparece a lista de desafios existirá um botão de criação de um novo desafio;
  - O utilizador ao clicar no botão abre um *popup* onde existirão os campos necessários para criar um novo desafio, como Título, Descrição, Prémios, uma entrada para o utilizador indicar a data de fim, um botão onde poderá aceder à galeria para selecionar uma imagem da sua galeria que ilustre o desafio e um botão de submissão;
  - Após carregar no botão de submissão será verificado se preencheu todos os campos obrigatórios e em caso afirmativo o utilizador recebe um *popup* de como o desafio foi criado com sucesso; Caso não se verifique o preenchimento de todos os campos obrigatórios, o desafio não é criado e o utilizador recebe um *popup* indicando que não preencheu todos os campos obrigatórios;
  - Se o desafio for corretamente persistido na base de dados todos os utilizadores da aplicação recebem uma notificação (interna/*push*) de que existe um novo desafio;
  - Será aberta de novo a janela com a lista de desafios onde se poderá verificar a presença do novo desafio criado.
- Requisitos não-funcionais:
  - O processo entre carregar no botão de submissão e o aparecimento da lista de desafios não deverá exceder os 5 segundos.
- Requisitos funcionais:
  - Utilizador *Challenger* é o único tipo de utilizador a poder criar um novo desafio.

Campo	Tipo	Nº de caracteres	Papel de Preenchimento
Título	*String*	100	Obrigatório
Descrição	*String*	1000	Obrigatório
Prémios	*String*	500	Opcional
Data de Fim	*Datetime*		Opcional

Tabela 3.1: Descrição do tipo de dados e suas características para a criação de desafios

- *User Story*: US0028
  Como um *Enthusiast* eu quero ver as notificações sobre actualizações que possam ter ocorrido nas ideias/desafios/grupos que sigo, bem como respostas a comentários que tenha publicado.

  - Critérios de Aceitação:
    - Existe uma opção no menu denominada ”Notificações”.
    - Ao clicar nas notificações será aberta uma página onde são mostradas os updates relativos a ideias/desafios/grupos que o utilizador segue.
  - Requisitos não-funcionais:
    - As páginas das notificações deverá ser aberta em menos de 2 segundos.

- *User Story*: US0031
  Como um *Enthusiast* eu quero poder comentar um desafio que estou a seguir para que as outras pessoas possam receber a minha opinião.
  - Critérios de Aceitação:
    - Existe uma opção no menu denominada “Desafios”;
    - Ao clicar num desafio é aberto a página de detalhe desse desafio, onde existe uma *tab* de “Comentários” onde se podem visualizar todos os comentários a esse desafio;
    - Existe uma caixa de texto para adicionar um comentário ao desafio;
    - Ao clicar na caixa de texto aparece o teclado para o utilizador inserir o seu comentário;
    - No contexto da caixa de texto existe um botão para submeter o comentário;
    - Caso o utilizador não tenha escrito nada ou tenha apenas introduzido “espaços” deverá aparecer um *popup* indicando a impossibilidade de introduzir comentários vazios;
    - Contrariamente, caso o utilizador tenha introduzido um comentário que não seja vazio deverá ser indicado se o comentário foi introduzido ou não na base de dados através de um *popup*;
    - Se o comentário foi corretamente persistido na base de dados todos os utilizadores que seguem a ideia recebem uma notificação (interna/*push*) de que existe um novo comentário;
    - O utilizador que submeteu o comentário recebe um número pré-determinado de pontos devido à sua acção e recebe um *popup* indicando ao utilizador a recepção desses pontos.

Campo	Tipo	Nº de caracteres	Papel de Preenchimento
Comentário	*String*	500	Obrigatório

Tabela 3.2: Descrição do tipo de dados que compõe um comentário

3.3.3 *Mockups*

Foram elaborados *Mockups* com o intuito de aproximar ao *design* das diferentes páginas da aplicação. Para tal recorremos à ferramenta Justinmind (ver referência [52]) para criar os *mockups* das interfaces. Procurou-se implementar interfaces simplistas para que a aplicação seja de fácil utilização. Algumas das páginas da aplicação foram inspiradas em exemplos de [53] e [54], como o caso do *template* de cada desafio.

De seguida são apresentados alguns dos *mockups* criados.

Figura 3.1: *Mockup* da página *Home Page*

Figura 3.2: *Mockup* da página *Notificacoes*

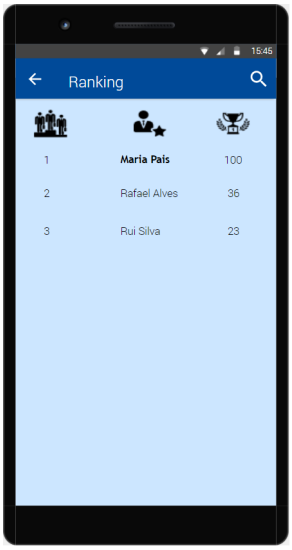


Figura 3.3: *Mockup* da página *Ranking*

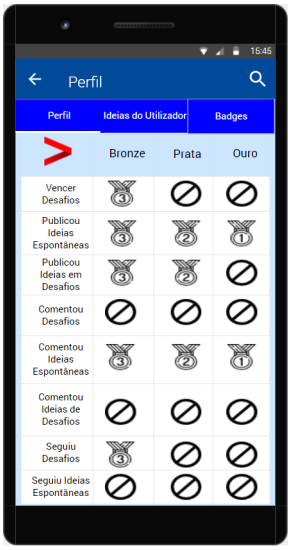


Figura 3.4: *Mockup* da página *Badges*

3.3.4 Modelo Entidade-Relação

Antes de iniciarmos a implementação foi solicitado que levantássemos as entidades que seriam necessárias criar na base de dados e que elaborássemos um modelo Entidade-Relação com as entidades e a relação entre estas. No entanto, ao longo do projeto, foi necessário ir atualizando o modelo pois foram necessárias criar mais tabelas do que as inicialmente previstas ou adicionar campos às já existentes.

Foi pedido que todas as tabelas criadas na base de dados tivessem quatro campos adicionais devido a uma questão de auditoria e de gestão das mesmas. Em cada tabela do modelo foram criados os seguintes quatro campos: "Utilizador_Criação", "Data_Criação", "Utilizador_Modificação" e "Data_Modificação". Todos estes têm de ser *not null*. Os campos "Utilizador_Criação" e "Utilizador_Modificação" são duas *foreign keys* para a tabela "Utilizador".

No (**Anexo A**) encontra-se representado este modelo e a explicação do que cada tabela representa.

3.4 Calendarização do Projeto

O plano de trabalho, abaixo, indica a calendarização prevista para as várias fases do projeto.

- 16 de Outubro de 2017 – 27 de Outubro de 2017:
  - Levantamento de requisitos
  - Análise funcional, técnica e arquitetural
  - Definição do *backlog* inicial
  - *Setup* da infra-estrutura de projeto
- 16 de Outubro de 2017 – 16 de Dezembro de 2017:
  - Elaboração do relatório preliminar
- 30 de Outubro de 2017 – 21 de Dezembro de 2017:
  - Desenvolvimento do *Minimum Viable Product*
- 22 de Dezembro de 2017 – 15 de Junho de 2018:
  - Levantamento de *feedback* através de *focus groups* e *hallway testing*
  - Incorporação de *feedback* e desenvolvimento da aplicação final
  - Realização de testes integrados

  - Acompanhamento de testes de aceitação pela equipa de originação da Accenture Technology em Portugal
  - Escrita da tese de mestrado

- 18 de Junho de 2018 – 30 de Junho de 2018:
  - Finalização da escrita da tese de mestrado
  - Preparação da defesa da tese de mestrado

Capítulo 4

Desenho

4.1 Arquitetura

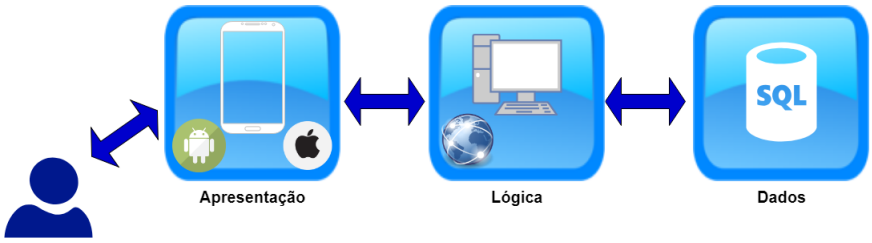


Figura 4.1: Arquitetura Inicial do Sistema

Racional Arquitetural:
O sistema divide-se em 3 camadas: (i) a camada de apresentação; (ii) a camada de lógica e a (iii) camada de acesso aos dados. A camada de apresentação é a camada relativa à aplicação móvel, e é com esta camada que o utilizador interage.

A camada de lógica corresponde à camada de serviços. Esta camada de serviços foi uma decisão que coube ao *Scrum Master* pois, futuramente, poderá ser do interesse da Accenture o desenvolvimento desta aplicação para a plataforma *web*, podendo fazer uso de um potencial reaproveitamento das APIs desenvolvidas. Serve de intermediário entre a aplicação móvel e a base de dados pois é responsável por fornecer os dados à aplicação móvel através de *queries* feitas à base de dados, de modo à aplicação móvel conseguir apresentar ao utilizador os dados pretendidos, e por alterar os dados existentes na base de dados de acordo com as operações executadas pelo utilizador, para além da execução de lógica de negócio que não faça sentido ser executada na aplicação *mobile*, como por exemplo a avaliação para o envio de notificações.

A camada de acesso aos dados corresponde à base de dados onde serão armazenados os registos das entidades descritas no modelo Entidade-Relação. Cada entidade referida no modelo corresponde a pelo menos uma tabela, tendo procurado seguir-se um modelo

normalizado *Boyce-Codd normal form* (BCNF).

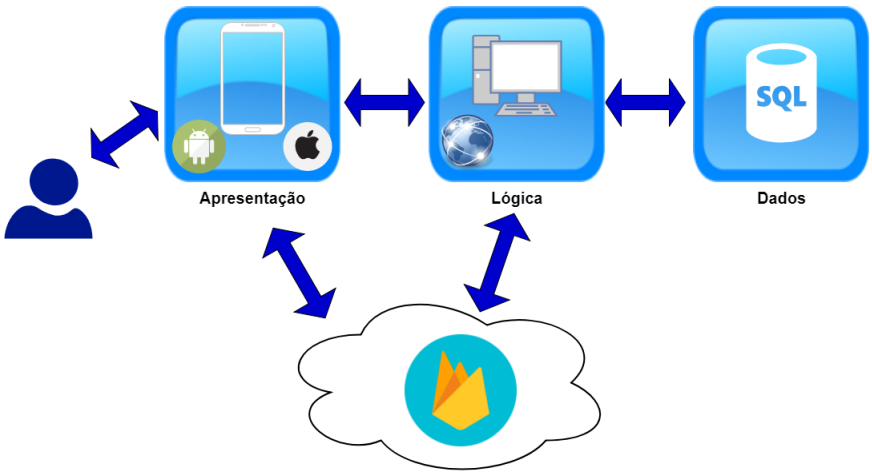


Figura 4.2: Arquitetura Final do Sistema

A Figura 4.2 representa a arquitetura final do sistema. Ao longo da implementação e devido a alguns problemas encontrados (explicados em 7.1) e, também, devido à necessidade da implementação de *push notifications* recorreu-se aos serviços fornecidos pelo Firebase e tornou-se necessário alterar a arquitetura inicial do sistema.

A camada de lógica comunica com o Firebase para o envio de uma nova *push notification*, para a criação de uma chave ou para saber uma determinada chave de um grupo de dispositivos de um utilizador ou para a adição e remoção de um dispositivo num grupo.

A camada de apresentação comunica com o Firebase para o *subscribe* e *unsubscribe* de tópicos, para a receção das *push notifications*, para a persistência de uma imagem aquando da criação de um desafio e para o retorno de uma imagem através do seu URL persistido na camada de acesso aos dados.

4.1.1 Estilo Arquitetural

O estilo arquitetural deste sistema é *Service Oriented Architecture* (SOA). Segundo [55] este estilo projeta um sistema de *software* que fornece serviços a aplicações de utilizador final através de interfaces públicas. No caso do sistema a ser desenvolvido, a aplicação móvel corresponde à aplicação utilizada pelo utilizador final que se conecta aos *web services* da camada de serviços que executam a lógica de negócio associada ao *web service* invocado.

4.2 Decisões

4.2.1 Decisões *Frameworks* e Base de dados

Antes de iniciar a fase de implementação das funcionalidades relativas à aplicação foi necessário definir quais as *frameworks* a serem utilizadas nas várias componentes do sistema.

Como o intuito da aplicação móvel é que seja utilizada em diversas plataformas, nomeadamente iOS e Android, é necessário recorrer a um desenvolvimento *cross-platform*. No contexto deste projeto é mais eficiente desenvolver um único código fonte pois o desenvolvimento nativo exigiria duplicação de esforço devido à necessidade do desenvolvimento das funcionalidades para cada plataforma específica. Mediante isto, e conforme o referido no Capítulo 2, existem diversas *frameworks* como o Xamarin.Forms, o PhoneGap ou o Ionic que respondem às necessidades do projeto. A plataforma Xamarin é *Cross-Compiled/Generated* e, segundo [6], esta é a abordagem mais promissora no que diz respeito a aplicações *cross-platform*. Para além disso, utiliza a linguagem C# (orientada a objetos) para o desenvolvimento do código e a *framework* Xamarin.Forms permite criar uma interface única para as diversas plataformas. Também foi pedida a opinião a elementos da Accenture Technology relativamente à *framework* a utilizar para o desenvolvimento da aplicação *mobile* entre as *frameworks* referidas acima sendo as respostas favoráveis à utilização da plataforma Xamarin.

Relativamente à camada de serviços foi definido pelo *Scrum Master* a utilização de ASP.NET Web API para a "construção" desta componente, e da Entity Framework, para trabalhar com classes que correspondem às tabelas da base de dados. ASP.NET Web API permite a construção de aplicações RESTful com serviços HTTP que poderão ser acedidos por dispositivos móveis [56]. Esta *framework* recorre ao padrão arquitetural MVC para estruturar o código pois nas suas configurações *default* existe uma dependência de "System.Web.Mvc". No âmbito deste estágio a componente de visualização (*view*) não será desenvolvida, sendo utilizada apenas a camada de serviços. A Entity Framework permite não escrever código de base de dados como "SELECT"ou "INSERT"caso não exista necessidade para tal [57].

As duas *frameworks* da componente da camada de serviços foram decididas pelo *Scrum Master* com o objetivo de ficar tudo em ambiente Microsoft pois é, também, utilizado o Visual Studio como IDE.

A base de dados, que foi disponibilizada pela Accenture, é uma base de dados relacional SQL Server.

4.2.2 Decisão Padrão Arquitetural Aplicação Móvel

É necessário estruturar o código relativo à aplicação móvel recorrendo a um padrão arquitetural de forma a existir uma separação entre a interface com o utilizador da sua lógica.

Para tal foi escolhido o padrão MVVM, pois este padrão é utilizado em plataformas de aplicações XAML, tal como é o caso do Xamarin.Forms que utiliza esta linguagem para o desenvolvimento das interfaces com o utilizador, sendo este o padrão que melhor se enquadra neste contexto.

4.2.3 Decisão relativa à comunicação entre a Aplicação Móvel e a camada de Serviços

Devido à escolha da *framework* ASP.NET Web API para o desenvolvimento da camada de serviços e sabendo que esta *framework* desenvolve aplicações RESTful é utilizada a arquitetura REST. A aplicação móvel necessita que exista um cliente REST para aceder aos *web services*, sendo utilizado o protocolo HTTP para estabelecer a ligação entre as aplicações.

4.3 Vista Arquitetural do Sistema

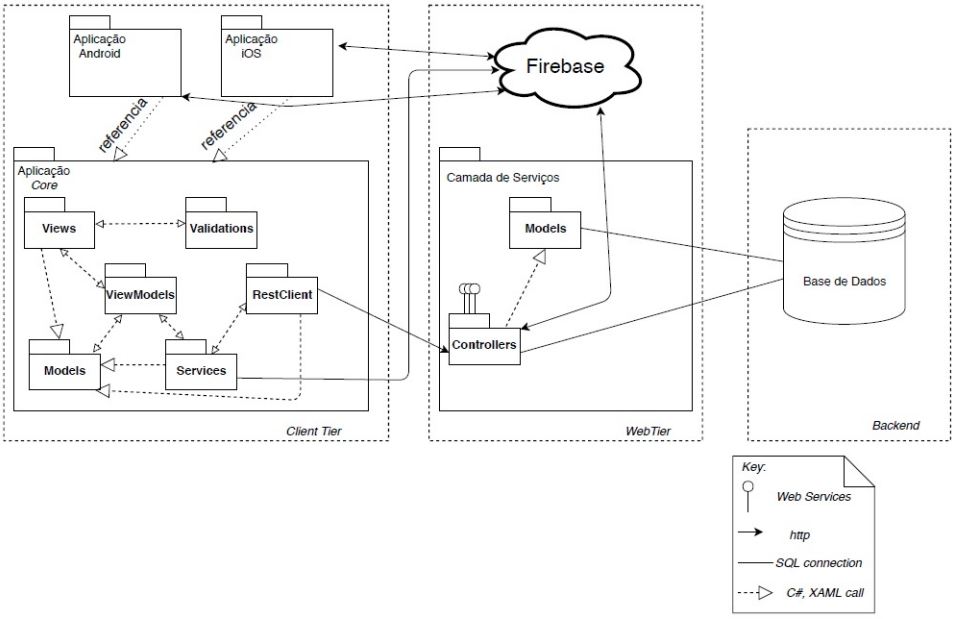


Figura 4.3: Vista *Multi-tier* de Componentes e Conectores do Sistema

Racional da Vista apresentada na Figura 4.3 e explicações técnicas:
A figura indicada exprime o funcionamento do sistema em *run-time*. O *package Views* é responsável pela interface com o utilizador. A interface é criada maioritariamente recorrendo à linguagem XAML e, por vezes a código C#. Cada *view* é composta por dois ficheiros, um com extensão ".xaml" onde é desenvolvido código XAML e outro ficheiro ".xaml.cs" onde é desenvolvido o código C#.

O utilizador vê no ecrã o que é disponibilizado pela *view* correspondente e executa operações que serão transmitidas para o *view model* que está ligado a essa *view*. Para existir esta associação é necessário indicar na *view* que o *Binding Context* corresponde ao ficheiro pretendido do *package view model*, como pode ser visto na figura seguinte.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="App1.Views.Desafios.ListaDesafios"
             Title="Lista de Desafios"
             xmlns:challengeVM="clr-namespace:App1.ViewModels.Desafios">

    <ContentPage.BindingContext>
        <challengeVM:ListaDesafiosViewModel/>
    </ContentPage.BindingContext>
```

Figura 4.4: Código XAML com *BindingContext*

Na Figura 4.4 verifica-se que a *view* "ListaDesafios" associa-se ao *view model* "ListaDesafiosViewModel".

Os *view models* têm o propósito de executar a lógica de apresentação da(s) *view(s)*. Estas duas componentes ligam-se através de *Data Bindings* e de *Commands*.

```
<Entry x:Name="DescricaoEntry" Text="{Binding Comentario.Descricao, Mode=TwoWay}" Placeholder="Insira aqui o seu comentário">
    <Entry.Behaviors>
        <behaviors:EntriesBehaviors x:Name="ComentarioBehavior"/>
    </Entry.Behaviors>
</Entry>
```

Figura 4.5: Código XAML com *DataBinding*

Como pode ser visto na Figura 4.5 existe um *Data Binding* de "Comentario.Descricao". Este Comentário é um *model* que é uma propriedade do *view model*. O *view model* implementa a interface *INotifyPropertyChanged* que tem o intuito de notificar a *view* e ser notificada por esta, quando alguma das suas propriedades é alterada, daí poder ser visualizado na figura o "Mode=TwoWay".

A ligação que pode ser vista na Figura 4.3 entre o *package View* e o *package Model* ocorre devido a serem feitas chamadas a atributos dos *models*, como por exemplo na Figura 4.5 através da chamada ao atributo "Descricao" do *model* Comentário. No entanto, isto permite o desenvolvimento independente de *models* e *views*.

```
<Button x:Name="SubmeterComentario" Text="Submeter Comentário!" Command="{Binding Preencher_Campos}" IsEnabled="False">
    <Button.Triggers>
        <DataTrigger TargetType="Button" Binding="{Binding Source={x:Reference ComentarioBehavior}, Path=IsValid}" Value="true">
            <Setter Property="IsEnabled" Value="True"/>
            <Setter Property="Command" Value="{Binding Submeter_Comentario}"/>
        </DataTrigger>
    </Button.Triggers>
</Button>
```

Figura 4.6: Código XAML com *Command*

Na Figura 4.6 encontra-se a expressão "Command=Binding Preencher_Campos" que executa o método "Command Preencher_Campos" do *view model* quando o utilizador

clica no botão e não preencheu a *entry* que possui o *behavior* "ComentarioBehavior". Caso o utilizador tenha preenchido a *entry* é executado o método "Command Submeter_Comentario" do *view model*.

O *behavior* mencionado acima encontra-se implementado no *package Validations* onde foi desenvolvido código para verificar se uma determinada *entry* está ou não vazia sendo isto necessário para quando um utilizador pretende criar, por exemplo, um desafio ou um comentário e se possa esquecer de algum campo obrigatório.

Os *view models* ligam-se ao *package Services* quando é necessário criar um objeto que se pretende colocar na base de dados ou quando é necessário enviar informação para a *view* de alguma informação contida na base de dados. Por sua vez os *services* criam uma instância do RestClient para que seja enviado para os *web services* criados na camada de serviços uma mensagem HTTP com o conteúdo necessário.

Os *controllers* da camada de serviços aplicam *queries* ou *updates* à base de dados utilizando os *models* dessa mesma aplicação.
Os *controllers* da camada de serviço encontram-se com uma ligação para o Firebase devido a existir um *controller* que envia a mensagem que deverá ser convertida em *push notification* pelo Firebase através de um pedido HTTP. O Firebase, posteriormente, envia a *push notification* para os dispositivos Android e iOS daí a sua ligação para os *packages* desses mesmos sistemas operativos. A existência de uma ligação entre essas duas componentes no sentido inverso deve-se a ser efetuado nos *packages* desses sistemas operativos o *subscribe* e *unsubscribe* de tópicos e o pedido de *token* que identifique a instância da aplicação no dispositivo em que esta é executada.

Também existe uma ligação do Firebase para os *controllers* devido às notificações diretas pois é necessário saber a chave do grupo de dispositivos de um utilizador armazenada no Firebase.

Um *service* da aplicação envia através de um pedido HTTP a imagem de cada desafio para o Firebase armazenar, daí a ligação entre estas duas componentes.

4.4 Tecnologias Utilizadas

Racional Diagrama 4.7:
O *Integrated Development Environment* (IDE) utilizado para o desenvolvimento da aplicação *mobile* e da camada de serviços foi o Microsoft Visual Studio (1). Dentro deste IDE foram utilizadas as *frameworks* do Xamarin (Xamarin.Forms (2), abordado na secção 2.1.2), ASP.NET Web API (3) (abordado na secção 4.2.1) e as linguagens de programação C# (4), XAML (5) e SQL (6).

A base de dados SQL Server foi gerida utilizando o Microsoft SQL Server Management Studio (8) onde foram criadas as tabelas e executadas as operações necessárias recorrendo à linguagem SQL.

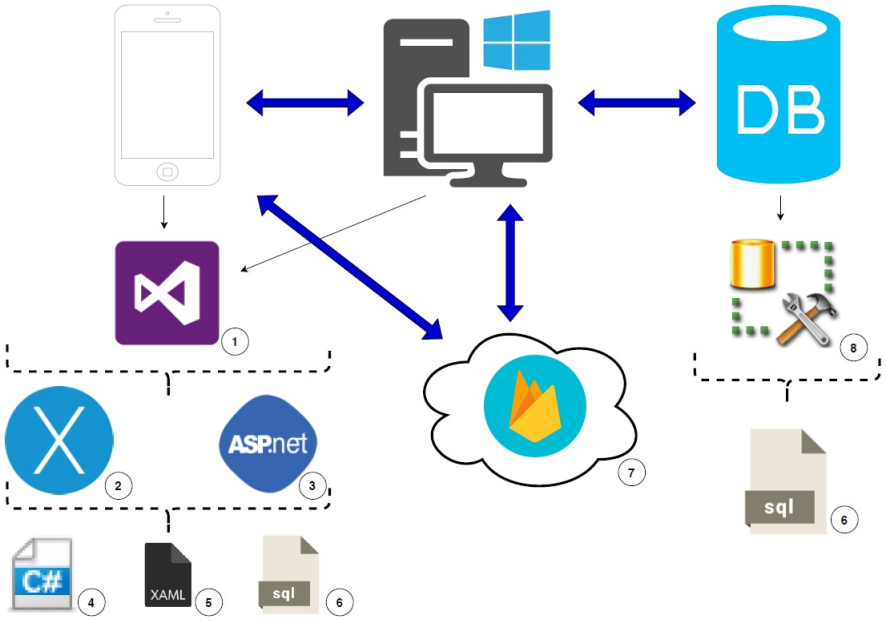


Figura 4.7: Diagrama com as tecnologias utilizadas

Tal como referido na secção 5.2.2 e 5.2.6, o Firebase (7) foi utilizado para o armazenamento de imagens e para as *push notifications* que refletem o despoletamento de eventos.

- Microsoft Visual Studio (1)
  Segundo [58], este IDE permite a criação de aplicações móveis (nativas ou híbridas) focadas em Android, iOS e Windows, aplicações *web*, jogos ou desenvolver bases de dados SQL Server. O Microsoft Visual Studio permite trabalhar com as linguagens C#, C++, Python ou JavaScript, por exemplo. Permite além do desenvolvimento, o teste do código através de ferramentas de teste, a depuração de forma a encontrar *bugs* no código rapidamente e permite também o controlo de versões com a integração do Git. Este IDE tem versões para Windows e Mac.

- C# (4)
  Segundo [59], C# é uma linguagem orientada a objetos e que possui muitas similaridades relativamente a C, C++ e Java. Esta linguagem permite, além de outras operações, expressões *lambda* e acesso direto a memória e oferece suporte a conceitos de encapsulamento, herança e polimorfismo. De acordo com [60], um dos conceitos importantes desta linguagem reflete-se nos métodos assíncronos que permite que a aplicação não bloqueie, como no acesso a recursos da web, e que continue a executar enquanto aguarda por uma resposta.

- XAML (5)
  De acordo com Buri e Lalonde em [61], XAML é uma linguagem com base em XML e tem o propósito de implementar as interfaces que o utilizador vê no ecrã. É

uma linguagem declarativa e é, normalmente, utilizada com outra linguagem imperativa como C# ou C++. É referido que esta linguagem usufruindo dos seus recursos como, *data bindings*, *routed events* e *attached properties*, permite a separação entre o *design* das interfaces com o utilizador da lógica da camada de apresentação.

- SQL (6)
  A linguagem SQL, de acordo com Melton em [62], é uma linguagem relacionada com as bases de dados relacionais. O conceito fundamental desta linguagem são as tabelas da base de dados com as suas várias colunas, em que cada uma tem um determinado tipo de dados. Uma coluna poderá ser identificada como chave primária de uma tabela indicando que os dados colocados naquela coluna são únicos e "not null". Para além disso uma coluna poderá ser marcada com uma chave estrangeira identificando uma relação entre duas tabelas.
  Recorrendo aos vários comandos característicos desta linguagem, como "Select" ou "Insert", é possível armazenar, manipular e verificar os dados inseridos nas várias tabelas de uma base de dados.

- Firebase (7)
  Segundo [63], o Firebase é a plataforma da Google direcionada a dispositivos móveis permitindo aos seus utilizadores criar aplicações e aumentar o número de utilizadores destas bem como adicionar funcionalidades a uma aplicação já existente. Esta plataforma é compatível com Android, iOS, Web, Unity e C++ mas, no entanto, nem todos os seus produtos estão disponíveis para todas as plataformas indicadas.
  Os produtos do Firebase dividem-se em três grandes áreas, de acordo com [64]: "Build better apps", "Improve app quality" e "Grow your business".

- Microsoft SQL Server Management Studio (SSMS) (8)
  Segundo [65], é um ambiente integrado que permite gerir uma infraestrutura SQL. Este programa permite configurar ou gerir, entre outras funcionalidades, as componentes de SQL Server, Azure SQL Database, ou SQL Data Warehouse. De acordo com [66] é possível através do SSMS consultar a base de dados quer esteja esta alocada ao computador local ou na *cloud*.

- Microsoft SQL Server
  De acordo com [67], Microsoft SQL Server é um sistema de gestão de bases de dados relacionais desenvolvido pela Microsoft e tem como principais funções as de armazenar e selecionar dados. As *queries* efetuadas à base de dados são executadas com recurso a T-SQL (Transact-SQL) que expõe palavras-chave para as operações que poderão ser executadas no SQL Server, como inserir e editar dados da base de dados.

- Justinmind
  Esta ferramenta permite prototipar *websites* ou aplicações móveis através do mecanismo de *drag and drop* sem a necessidade de desenvolvimento de código. Permite fazer navegação entre as diferentes páginas da aplicação simulando a futura lógica de navegação que a aplicação venha a ter. Para mais detalhes, visitar [52].

- Git
  Sistema que permite o controlo de versões de ficheiros. É uma ferramenta que possibilita a uma equipa de desenvolvimento a divisão dos seus elementos em funcionalidades distintas e quando necessário possam juntar o trabalho realizado. Para além disto permite um histórico das alterações nos vários ficheiros afetos às alterações e que a equipa possa reverter alguma alteração efetuada para um estado anterior de um desses ficheiros. Para mais detalhes, visitar [68].

- Git Extensions
  Interface gráfica que permite ao utilizador efetuar as operações necessárias no Git sem a necessidade do acesso à linha de comandos. Existe um *plugin* para o Visual Studio que permite a integração desta ferramenta, de acordo com [69]. Para mais detalhes, visitar [70].

Capítulo 5

Implementação

5.1 *Setup*

No IDE utilizado para o desenvolvimento da aplicação foi necessário criar um projeto *cross platform* Xamarin, utilizando a *framework* Xamarin.Forms.

Para a partilha do código entre as diferentes plataformas optou-se pela estratégia do *Portable Class Library* (PCL) que como o próprio nome indica, é uma biblioteca portátil referenciada pelos projetos que representam os sistemas operativos, ao invés de *Shared Project* que copia o código comum para cada um destes sistemas [71].

Após a criação do projeto existem três projetos, um projeto onde está o código comum da lógica de negócio e de implementação das *views* da aplicação, um projeto relativo ao sistema operativo Android e outro relativo a iOS.

5.2 Funcionalidades Desenvolvidas

Para auxiliar na implementação da aplicação, nomeadamente em questões de *design*, na ligação entre a aplicação móvel e a camada de serviços ou para serializar e desserializar dados no formato JSON devido a, por exemplo, pedidos enviados para o Firebase foram utilizadas bibliotecas externas. Estas bibliotecas externas encontram-se num gerenciador de pacotes NuGet que, segundo [72], é um dos recursos mais importantes das plataformas utilizadoras de .NET pois permite produzir, partilhar e utilizar pacotes de código com vista ao aceleramento do processo de desenvolvimento.

As principais bibliotecas externas utilizadas na implementação foram:

- *RestClient* - utilizada para conectar a aplicação móvel à camada de serviços;
- *Popups* - utilizada para permitir ao utilizador criar/editar desafios e para mostrar informação ao utilizador relativamente a algum evento ocorrido na aplicação;
- *Firebase Storage* - permite armazenar imagens no Firebase;

- *Top Tabbed Page* - biblioteca que permite que as *tabbed pages* fiquem na parte superior do ecrã tanto num dispositivo Android como iOS pois, por defeito, nos dispositivos iOS as *tabbed pages* surgem na parte inferior do ecrã;
- *Connectivity* - para verificar se o utilizador está com conexão à Internet;
- *Media* - utilizada para aceder à galeria de imagens do dispositivo;
- *User Dialogs* - utilizada para indicar através de um ecrã de espera (ex: "*Loading...*") que alguma ação está a ser processada;
- *Google Play Services* - utilizada para verificar se os serviços do google *play* eram suportados pelo dispositivo e se estavam disponíveis;
- *Firebase Messaging* - utilizada para a implementação das *push notifications* nomeadamente para o envio de notificações ou para o *subscribe* e *unsubscribe* de tópicos;
- *Newtonsoft Json* - utilizada para a camada de serviços serializar e desserializar dados nos pedidos HTTP.

5.2.1 Funcionalidade Mensagens Alerta

Esta funcionalidade foi desenvolvida para dar *feedback* ao utilizador relativamente às operações que este executa na aplicação.

Foi desenvolvida uma *view* que corresponde a um *popup* (biblioteca externa). Esta *view* é estruturada numa *grid* em que na primeira coluna encontra-se o logótipo da empresa e na segunda coluna encontra-se o texto que corresponde à mensagem que será mostrada ao utilizador. A *view* é composta ainda por um botão para fechar o *popup*.

Em cada método *Command* dos *view model* em que exista a necessidade de dar *feedback* ao utilizador faz-se a navegação para esta *view* que é aberta por cima da página atual em que o utilizador se encontra, com informação do texto que irá ser mostrado no *popup*.

Os eventos que permitem ao utilizador receberem um *popup* com uma mensagem de *feedback* são:

- Não tem *wifi*;
- Necessita de fazer *login*;
- Criação/Edição de um desafio/ideia;
- Criação/Resposta de um comentário;
- Seguir/Deixar de seguir um desafio/ideia/grupo;
- Gostar/Não gostar de uma ideia;

- Não encontrar resultado para a pesquisa inserida;
- Prémio atribuído.

Nas restantes funcionalidades, sempre que for evidenciado a abertura de um *popup* estarei a referir esta funcionalidade.

As Figuras 5.1, 5.2, 5.3 e 5.4 mostram alguns dos contextos em que é utilizada esta funcionalidade.

Figura 5.1: O utilizador tentou submeter um comentário vazio

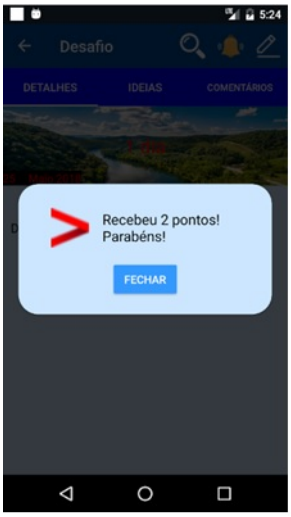


Figura 5.2: O utilizador recebeu pontos

Figura 5.3: O utilizador editou uma ideia

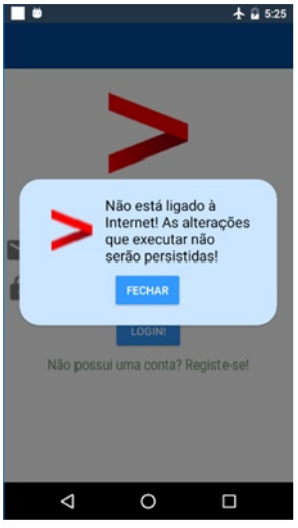


Figura 5.4: O utilizador está sem ligação à Internet

5.2.2 Funcionalidade Desafios

A funcionalidade dos Desafios é composta por várias componentes:

- Visualizar todos os desafios existentes
  O objetivo é mostrar ao utilizador todos os desafios que estejam a ocorrer no momento da invocação. Para isso foi necessário criar uma *view* composta por uma *list view*. Esta *list view* faz *binding* de uma propriedade *list* existente no *view model* que é preenchida por um método da classe *service* correspondente aos desafios. Este método retorna uma lista composta pelos dados provenientes da base de dados, pois executa o RestClient que faz o pedido dos desafios ao *controller* correspondente da camada de serviços. O controller executa uma *query* à base de dados para que sejam retornados todos os desafios armazenados que ainda não tenham terminado.

- Visualizar os desafios que o utilizador segue
  Esta componente pertence igualmente à funcionalidade "Seguir" pois são apresentados os desafios que um utilizador segue. Para tal foi criada uma *view* composta por uma *list view* que, do mesmo modo da componente acima, faz *binding* com uma propriedade *list* pertencente ao *view model*. A list é preenchida pela lista retornada pelo método que é chamado da classe respetiva do *package* "Services". Essa classe *service* faz um pedido HTTP para a camada de serviços. No método do *controller* invocado foi necessário desenvolver uma *query* SQL na tabela correspondente aos Desafios e na tabela "Seguir" para selecionar os desafios que têm o seu ID na tabela "Seguir" e em que esses mesmos registos possuem o ID do utilizador que faz o pedido.

- Visualizar todos os desafios concluídos
  Esta componente tem o objetivo de mostrar ao utilizador todos os desafios que já tenham terminado. Do mesmo modo que as componentes acima explicadas executa-se o preenchimento da *list view* da *view* responsável pela visualização dos desafios concluídos.
  No *controller* da camada de serviços responsável pelo *select* dos desafios concluídos é necessário comparar a data de fim de todos os desafios armazenados com a data atual em que o *select* é executado.

- *Template* de cada desafio nas listas de Desafios
  Cada desafio nas várias listas da aplicação tem o seu *template* definido numa *view*. Devido às três listas indicadas nas componentes acima e à lista de desafios resultante da pesquisa de um utilizador foi decidido que teria de definir o seu *template* num ficheiro que seria invocado pelas restantes *views*, evitando assim repetição de código e que caso fosse necessário alterar algo na estrutura do *template* tivesse que o fazer apenas num local.

Cada elemento da lista foi estruturado com recurso a uma *frame* para que parecesse um *card view* com margem relativamente aos limites do ecrã e ao desafio imediatamente abaixo na lista. Dentro da *frame* os elementos encontram-se estruturados com recurso a um *stack layout* com dois elementos: a imagem do desafio e por baixo uma *grid*. Esta *grid* é composta por duas colunas e duas linhas. Na primeira coluna e primeira linha aparece o dia de fim do desafio e na segunda linha aparece o mês e ano de fim do desafio. Na segunda coluna e primeira linha aparece o título e por baixo, na segunda linha, aparece a sua descrição.

As Figuras 5.5, 5.6 e 5.7 mostram as diferentes listas de desafios. Todas estas listas possuem o mesmo *template* para a representação de cada desafio, como indicado acima.

Figura 5.5: Lista de Desafios

Figura 5.6: Desafios a Seguir

Figura 5.7: Desafios Concluídos

- Criação de Desafios
  Esta atividade apenas está disponível para utilizadores do tipo "Challenger" (diferença nos botões das Figuras 5.8 e 5.9). Para criar um desafio é necessário que o utilizador atribua obrigatoriamente um título e uma descrição ao desafio. Caso o utilizador não tenha preenchido algum destes campos o utilizador recebe um *popup* com uma mensagem de alerta a indicar que terão que ser preenchidos os campos obrigatórios. A verificação de que o utilizador não preencheu os campos é feita recorrendo à classe desenvolvida no *package* "Validations". É facultativo que o desafio tenha prémios e caso o utilizador não tenha indicado uma data de fim do desafio é assumido por defeito que este acaba 30 dias após a data de criação. O utilizador poderá também ilustrar o desafio com uma imagem da galeria do seu dispositivo.

Quando o utilizador carregar no botão de submissão, e caso todos os campos obrigatórios estejam preenchidos, é acionado um método *command* que está implementado no *view model* da *view*. Neste método são preenchidos os restantes campos como a data de criação e de modificação com o valor da data atual da criação. Após este processo, o objeto "Desafio" é passado como parâmetro para a classe *service* dos desafios. Essa classe, por sua vez, cria uma instância do RestClient e invoca o método necessário na camada de serviços, através do URL, que recebe no conteúdo da mensagem HTTP o objeto "Desafio". Por fim, o *controller* trata de fazer o *commit* na base de dados.
O utilizador no fim de ter carregado no botão de submeter o desafio recebe um *popup* com uma mensagem indicando se o desafio foi guardado na base de dados.

Figura 5.8: Um utilizador não *Challenger* não pode criar desafios

Figura 5.9: Um utilizador *Challenger* pode criar desafios

- Edição de um desafio
  Esta atividade, tal como a de criação de um desafio, só pode ser executada por utilizadores que sejam administradores e ocorre dentro da página que mostra os detalhes do desafio (diferença nos botões das Figuras 5.10 e 5.11). Ao premir o ícone existente na *toolbar* para editar o desafio é invocado um método *command* onde é feita a navegação para o *popup* da edição. Este *popup* é composto pelos mesmos campos da página de criação de um desafio. No entanto é necessário que a *view* relativa aos detalhes passe como parâmetro desse método *command* o desafio, para que os campos preenchidos aquando da criação ou da última edição sejam mostrados ao utilizador. Após o utilizador carregar no botão de submissão da edição, é necessário verificar se os campos obrigatórios foram todos preenchidos. Após a edição correta do desafio executa-se o mesmo processo para persistir o objeto "Desafio" na base

de dados com as alterações efetuadas.
Da mesma forma ao ocorrido na criação de um desafio caso o objeto tenha sido guardado com as alterações na base de dados recebe um *popup* com uma mensagem indicando o sucesso da operação. Caso contrário, é indicado que a edição não ocorreu.

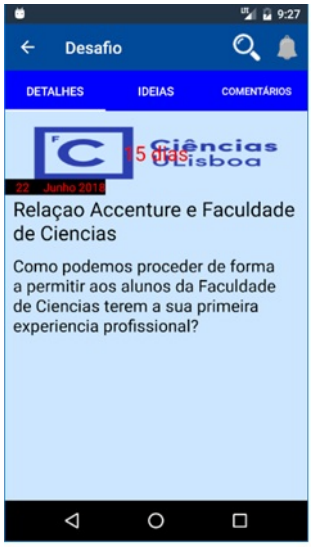


Figura 5.10: Um utilizador não *Challenger* não pode editar desafios

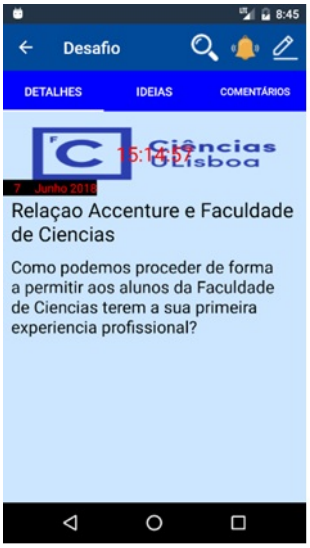


Figura 5.11: Um utilizador *Challenger* pode editar desafios

- Selecionar imagem para o desafio e armazenamento no Firebase
  Esta componente da funcionalidade atribui uma descrição visual do desafio quando o utilizador cria o desafio ou quando, na edição do desafio, o utilizador pretenda alterar a imagem atual do desafio.
  No *view model* correspondente à criação e edição de um desafio existe um método que é chamado quando o utilizador pressiona o botão para escolher uma imagem da sua galeria. Depois de escolhida a imagem é divulgada ao utilizador no *popup* para que este a possa visualizar enquanto termina a criação ou edição.
  Ao carregar no botão de submissão invoca o *service* correspondente ao Firebase. O *service* mencionado tem a responsabilidade de armazenar a imagem escolhida, através de um *stream*, no Firebase. Todas as imagens persistidas são alvo de um *resize* (explicado na secção 7.1) para que todos os desafios tenham as imagens do mesmo tamanho. O método que coloca a imagem no Firebase retorna um URL que indica a localização da imagem no sistema. No fim de persistido o desafio na base de dados é criado um objeto "Imagem" que possui o ID do desafio e o URL da imagem deste, para além de outros campos, que é persistido na base de dados através da ligação entre o *service* responsável pelas imagens e o *controller* da camada de serviços que executa a persistência na tabela "Imagem".

- *Popup* de criação/edição de um desafio
  Para a criação e edição do desafio é utilizado o mesmo *popup* estando a sua estrutura definida numa única *view* (Figuras 5.12 e 5.13).
  Este *popup* é constituído por uma *frame* em que os seus elementos estão organizados com recurso ao *stack layout*. O primeiro elemento corresponde à imagem da empresa e o segundo a uma *grid* com os elementos necessários para criar o desafio, como mostram as figuras indicadas acima.

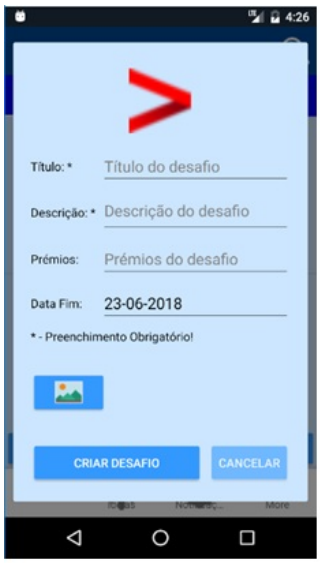


Figura 5.12: *Popup* de criação de desafios

Figura 5.13: *Popup* de edição de desafios

- Verificar os detalhes de um desafio
  Esta componente da funcionalidade ocorre quando um utilizador seleciona um desafio da lista de desafios existentes, da lista de desafios que segue, da lista de desafios concluídos ou da lista de desafios resultante da pesquisa por um desafio. Cada uma dessas listas está preenchida por objetos "Desafio" e encontram-se caracterizadas, como indicado anteriormente, através de *list views*. Esta componente de UI possui uma propriedade denominada por "SelectedItem", que tem o intuito de identificar o elemento da lista selecionado pelo utilizador, e que se encontra ligada a uma propriedade do tipo "Desafio" criada no *view model* ligado à *view*. Quando o utilizador seleciona um desafio, o *view model* é notificado e executa a navegação para a *view* responsável pela demonstração dos detalhes do desafio. O desafio selecionado é passado pelo *view model* para a *view* que é responsável pela apresentação destes detalhes.
  As ideias e os comentários são mostrados em duas *list views* onde o processo para a visualização é o mesmo que ocorre para mostrar as várias listas de desafios. No

caso da *list view* das ideias foi necessário recorrer de igual modo à propriedade "SelectedItem" pois o utilizador poderá querer visualizar a ideia em mais detalhe tendo sido realizado o mesmo procedimento indicado acima.
É necessário chamar os *services* relativos aos comentários e às ideias pois são estas classes que se conectam com os *web services* correspondentes e que executam os *selects* necessários à base de dados de forma a preencher as listas indicadas.
A integração da lista de ideias nesta componente foi feita em conjunto com o Gustavo Silva pois a funcionalidade "Ideias" é da sua responsabilidade.

- Estrutura/*Look and feel* das páginas de detalhes do desafio
  As páginas relativas às ideias e comentários do desafio têm apenas definidas as duas *list views* que irão mostrar ao utilizador, sendo necessário invocar os *templates* da ideia e comentário implementados noutras *views*, como poderá ser visto nas Figuras 5.17 e 5.18. A página relativa às ideias tem ainda um botão para permitir ao utilizador sugerir uma ideia para o desafio e a página relativa aos comentários possui uma entrada para o utilizador comentar o desafio, como explicado na funcionalidade dos "Comentários". A inclusão do *template* das ideias foi feita conjuntamente com o Gustavo uma vez que esta funcionalidade é da sua responsabilidade.
  O *look and feel* da página de detalhes do desafio onde aparece a imagem, um contador refletindo o tempo que falta para o término do desafio, o título, a descrição e os prémios do desafio tornou-se bastante desafiante de implementar. Na sua implementação recorri ao ficheiro da *view* no qual se desenvolve código C# devido a não conseguir implementar algumas particularidades necessárias em código XAML.
  A *view* foi estruturada com o recurso ao *relative layout* devido à necessidade de colocar restrições para colocar o contador no meio da imagem ilustrativa do desafio bem como a data de fim do desafio no canto inferior esquerdo da imagem.

```
Func<Image, double> getContadorWidth = (parent) => Contador.Measure(parent.Width, parent.Height).Request.Width;
Func<Image, double> getContadorHeight = (parent) => Contador.Measure(parent.Width, parent.Height).Request.Height;
```

Figura 5.14: Funções que calculam a altura e largura mínima do contador relativamente à imagem

As funções ilustradas na Figura 5.14 retornam um *double* da largura e altura mínimos do contador relativamente à imagem. Posteriormente adiciona-se o contador ao *relative layout* indicando que terá que ter em conta duas *constraints*, relativamente ao eixo dos xx e yy, resultantes da subtração da altura e largura da imagem por cada *double* calculado, respetivamente para a altura e largura necessárias para o contador. Cada um dos valores utilizados nas subtrações terá de ser dividido por 2 de

modo a colocar o contador no meio da imagem.
Do mesmo modo se procede para a data de fim do desafio. No entanto, neste caso apenas tem de se ter em conta a altura da imagem devido à não existência de uma *constraint* relativamente ao eixo dos xx pois a data de fim será colocada no canto inferior esquerdo da imagem, ou seja no início do ecrã do dispositivo. Relativamente à *constraint* realizada no eixo dos yy para a adição da data de fim ao *relative layout* resulta da subtração da altura da imagem pela altura necessária da data de fim.
Os restantes elementos que compõem a *view* como o título, a descrição e os prémios também tiveram que ser adicionados ao *relative layout*. O título foi adicionado imediatamente abaixo da imagem sendo que a *constraint* relativamente ao eixo dos yy consiste na altura da imagem. Os restantes elementos, tal como o título, também só tiveram apenas restrições no eixo dos yy resultantes da soma da altura da imagem com todos os elementos já colocados.
O contador encontra-se ligado a uma propriedade do *view model* responsável pela lógica de apresentação desta *view*. Esta propriedade reflete uma *string* que poderá ter os seguintes valores, conforme o tempo para o fim do desafio:

- Superior a 31 dias: "(...) meses";
- Inferior ou igual a 31 dias e superior a 1 dia: "(...) dias";
- Igual a 1 dia: "1 dia";
- A data de fim do desafio é no dia atual: "(...):(...):(...)";
- Desafio concluído: "Vencedor: (...)".

Quando a data de fim do desafio é no dia atual criei uma *thread* que de segundo a segundo executa uma função criada para atualizar o contador e fazer decrescer a contagem de "23:59:59" até "00:00:00". Esta *thread* apenas é executada quando o utilizador está na página de detalhes do desafio.
As Figuras 5.15 e 5.16 mostram a página de detalhes de um desafio em dias diferentes como poderá ser visto no contador colocado no meio da imagem do desafio.

5.2.3 Funcionalidade Comentários

- Racional da Ação de Comentar
  A funcionalidade de Comentários tem como objetivo permitir ao utilizador responder a uma ideia, ou a um desafio ou a um comentário. Além disto é necessário que os utilizadores visualizem esses mesmos comentários.
  Para permitir ao utilizador responder a um desafio ou a uma ideia foi invocado na página "Comentários", da página de detalhe de cada uma das funcionalidades,

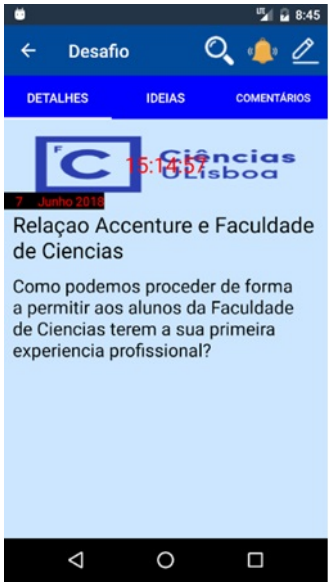


Figura 5.15: Detalhes de um desafio no último dia que está ativo

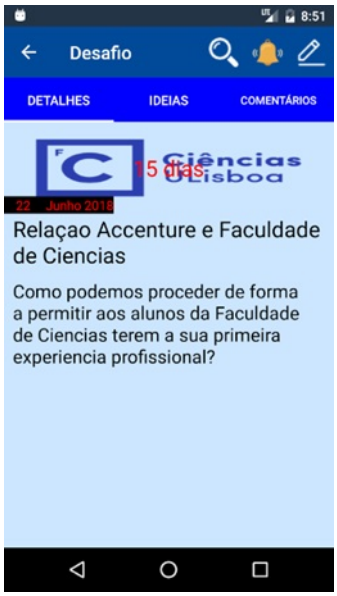


Figura 5.16: Detalhes de um desafio a 15 dias do seu término

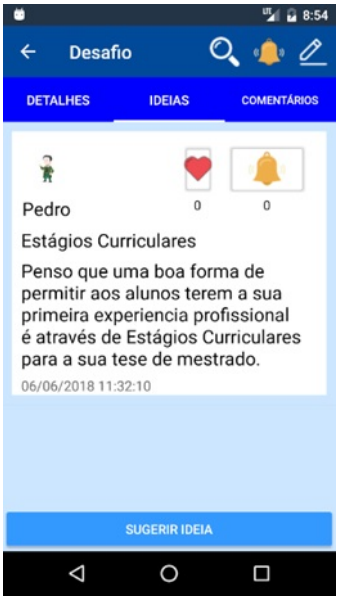


Figura 5.17: Página de ideias de um desafio

Figura 5.18: Página de comentários de um desafio

uma *view* onde se encontra implementado uma entrada em que o utilizador pode escrever o texto que pretende comentar e um botão (ícone de um comentário) que executa a ação de publicação de um comentário. Esta *view* foi desenvolvida com o intuito de evitar a repetição de código associada à utilização em contextos diferentes, como por exemplo para os comentários a desafios e ideias ou para respostas

a comentários. Cada *view* que invoca esta *view* passa uma *key* para que se saiba qual a classe *view model* que terá de ser chamada (do desafio ou da ideia) quando o utilizador escreve o texto e pressiona o ícone de comentar.
Na classe *view model* (do desafio ou da ideia) é invocado um método *command* que, por sua vez, executará um método *command* da classe *view model* afeta aos comentários onde são passados como parâmetros uma nova chave que identifica se é um comentário raíz a um desafio ou ideia e o ID da entidade alvo da resposta. Esse mesmo método responsável pela criação do comentário invoca a classe *services* correspondente aos comentários que, por sua vez, se conecta ao *web service* que persiste o comentário com a sua descrição e ID do desafio ou da ideia a que corresponde.
É também possível que o utilizador possa responder a um comentário sendo que para tal acontecer, no *template* de cada comentário da lista de comentários existe um botão (ícone de um comentário) que indica ao utilizador a permissão para responder ao comentário. O *template* de cada comentário utilizado nas listas de comentários encontra-se definido numa única *view* para evitar a repetição de código, como no caso da resposta a um comentário, e para que caso existisse a necessidade de alterar a estrutura do *template* fosse apenas necessário alterar num único ficheiro. Quando um comentário é uma resposta a um comentário é dada uma maior indentação relativamente à margem esquerda do ecrã de modo que o utilizador entenda que é uma resposta ao comentário acima.
Ao ser premido o botão de um comentário abre-se um *popup* composto pelo comentário (com o mesmo *template* dos comentários incluídos nas listas de comentários) que será alvo da resposta (para que o utilizador possa ter sempre a visualização desse comentário), por uma entrada onde o utilizador pode escrever o texto e um botão de submissão da resposta. A entrada e o botão encontram-se definidos na *view* indicada para resposta, como indicado na *tabbed page* "Comentários" dos desafios e ideias, e aparecem devido à invocação desta *view*. O processo ocorre do mesmo modo que o efetuado acima com a diferença que para além do comentário adicionado na base de dados possuir o ID do desafio ou ideia, possui, também, o ID do "Comentário Pai".
No caso do utilizador premir o botão e não ter adicionado texto ou apenas tenha digitado "espaços", o *view model* em que se encontra o contexto abre um *popup* indicando que não é possível publicar comentários vazios.
Tal como feito na criação e edição de um desafio é, também, enviado um *popup* com uma mensagem de alerta ao utilizador que indica se o comentário foi ou não adicionado à base de dados.
A explicação acima encontra-se esquematizada no diagrama da Figura 5.19:

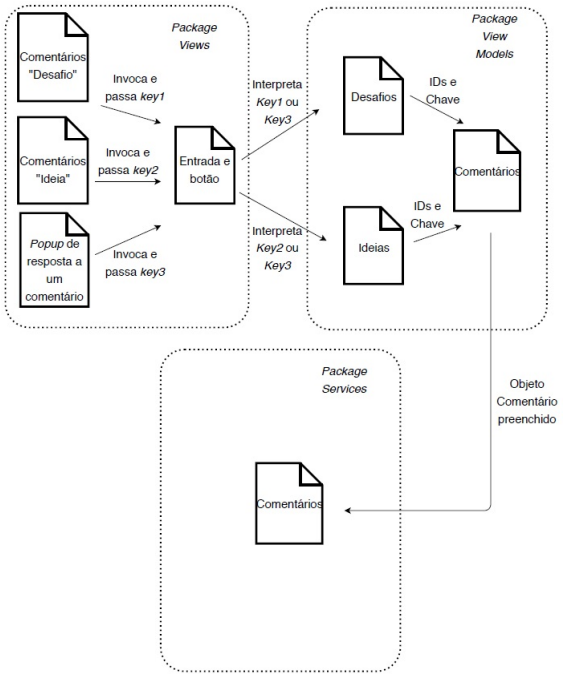


Figura 5.19: Diagrama de explicação do racional

A integração desta funcionalidade com a funcionalidade "Ideia" foi feita com o consentimento do Gustavo Silva.

- Racional da lista de comentários de cada entidade
  Na página de detalhe de uma ideia e de um desafio aparece a lista de comentários à entidade em questão. O processo para preencher esta lista ocorre da mesma forma que o processo para o preenchimento da lista de desafios. No entanto, a lógica do preenchimento desta lista ocorre no *controller* da camada de serviços através do seguinte algoritmo:

  1. Verificar qual o tipo da entidade que está a pedir os seus comentários: se uma ideia ou se um desafio;
  2. Após esta verificação, é feita uma *query* na tabela correspondente utilizando o ID dessa entidade;
  3. Desse modo obtém-se a lista de comentários "raiz", ou seja, comentários que não são respostas a outros comentários;
  4. De seguida é invocado um método recursivamente que faz uma *query* na tabela "Comentário" para verificar os comentários que são resposta a cada comentário;
  5. Cada comentário, quando procuradas as suas respostas, é colocado numa lista de comentários procurados para garantir a paragem da recursão;

6. A lista de comentários de cada comentário é colocada logo após o comentário pai para que a lista se encontre ordenada;
7. Quando já não existirem respostas a qualquer dos comentários verifica-se que a lista de procurados é igual à lista ordenada, pára-se a recursão e retorna-se a lista de comentários ordenada.

As Figuras 5.20, 5.21 e 5.22 representam as páginas referentes a esta funcionalidade.

Figura 5.20: Lista de comentários de um desafio

Figura 5.21: *Popup* para responder a um comentário

Figura 5.22: Lista de comentários a uma ideia

5.2.4 Funcionalidade Seguir

Esta funcionalidade não estava planeada inicialmente como parte do plano do projeto e como se trata de uma funcionalidade relacionada com as que me foram atribuídas acabei por ser eu a implementá-la.

Esta funcionalidade tem o propósito de facultar ao utilizador a possibilidade de seguir um desafio, uma ideia ou um grupo do qual faça parte. Posteriormente, servirá para o utilizador receber notificações relativas aos desafios, ideias e grupos que segue. Na restante explicação da funcionalidade sempre que referir "entidade" pretendo indicar um desafio, uma ideia ou um grupo.

Na página de detalhe de cada entidade existe um ícone, que corresponde a uma campainha, na *toolbar* que permite ao utilizador seguir essa entidade ou, caso o utilizador já o faça, em que poderá deixar de a seguir, como pode ser visto nas Figuras 5.23 e 5.24. Quando o utilizador não está a seguir a entidade a campainha é cinzenta de modo a indicar

ao utilizador que este não segue a entidade em questão. No entanto quando o utilizador passa a seguir uma das entidades a campainha passa a ser amarela.

Existem dois métodos *Command* no *view model* que corresponde à *view* de detalhe. O *Command* relativo à ação de seguir a entidade invoca um método da classe *services* relacionada com a funcionalidade ”Seguir”onde é passado um objeto “Seguir” com o ID do desafio, da ideia ou do grupo, com o ID do utilizador que executou a ação e com os outros campos devidamente preenchidos. A classe do *package* “Services” cria uma instância do RestClient onde é enviado o objeto “Seguir” que será persistido na base de dados pelo *controller* da camada de serviços. O *Command* relativo a deixar de seguir a entidade segue o mesmo procedimento só que o *controller* da camada de serviços remove o registo da base de dados relativa ao utilizador e à entidade que este seguia.

Em qualquer destas operações é aberto um *popup* com uma mensagem indicando ao utilizador se as operações indicadas foram devidamente executadas (Figuras 5.25 e 5.26).

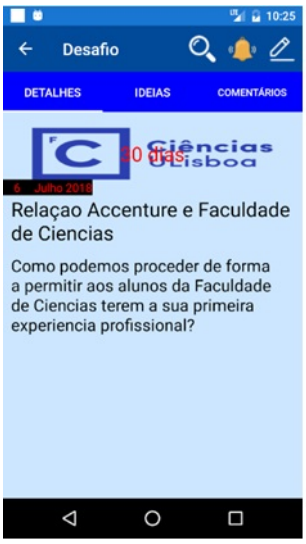


Figura 5.23: Desafio seguido pelo utilizador

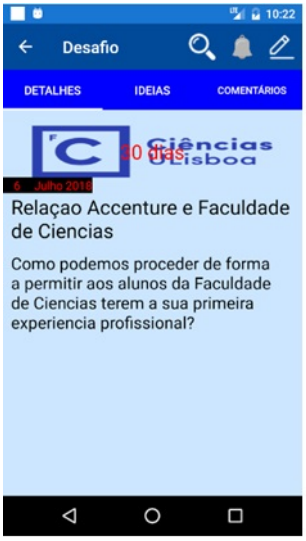


Figura 5.24: Desafio não seguido pelo utilizador

5.2.5 Funcionalidade Pesquisa

Esta funcionalidade permite ao utilizador da aplicação uma pesquisa em três contextos diferentes: desafios, ideias ou pelo perfil de outros utilizadores; Caracteriza-se por uma página composta por três secções diferentes permitindo que o utilizador possa pesquisar no contexto pretendido. Cada uma destas secções é composta por uma *list view* correspondente que será preenchida com o resultado proveniente da camada de serviços.

Cada *list view* encontra-se em ficheiros separados e será preenchida quando o utilizador efetuar a sua pesquisa. Cada uma destas *list view* têm o seu *item template* definido

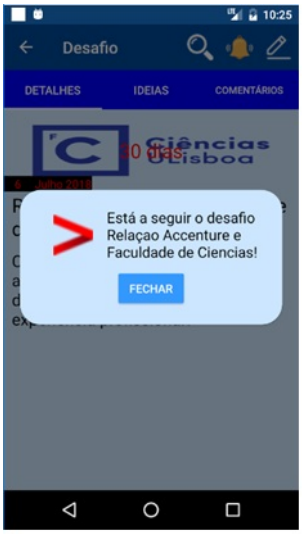


Figura 5.25: *Popup* que indica ao utilizador que está a seguir o desafio

Figura 5.26: *Popup* que indica ao utilizador que deixou de seguir o desafio

em ficheiros à parte sendo apenas necessário invocar cada um dos ficheiros para que, por exemplo, a lista de desafios pesquisados apresente o mesmo *template* quando comparada com a lista de desafios concluídos. No topo de cada página existe uma "SearchBar" onde o utilizador insere o texto que pretende pesquisar. Quando o utilizador submete a sua pesquisa é invocado um método *command* desenvolvido no *view model* que é o responsável pela lógica de apresentação de cada uma das três *views*.
Este método *command* recebe um parâmetro que indica o contexto no qual o utilizador se encontra a pesquisar e pode tomar os seguintes valores: "Desafios", "Ideias" ou "Utilizadores". Este parâmetro funciona como chave e conforme o seu valor é chamado o método responsável do *service* correspondente ao contexto. Este método recebe o texto que o utilizador pesquisou e invoca o RestClient que envia um pedido HTTP para a camada de serviços.

Na camada de serviços o *controller* correspondente efetua um *select* responsável pelo retorno do resultado da pesquisa do utilizador e que é enviado de seguida para a aplicação *mobile* para ser apresentado ao utilizador.
No *select* efetuado no "DesafiosController" retornam-se todos os desafios que possuam no seu título ou descrição o texto pesquisado. Do mesmo modo executou-se um *select* no ficheiro "IdeiasController" mas teve-se o cuidado de não colocar no resultado da pesquisa as ideias que pertençam a grupos do qual o utilizador não faça parte e as ideias que ainda são "rascunhos". O *select* executado no ficheiro "PerfilController" retorna todos os utilizadores que o seu nome ou sobrenome corresponda ao texto pesquisado.

Com o conhecimento do Gustavo efetuei os dois métodos necessários nos ficheiros "IdeiasController" e "PerfilController" devido a estes ficheiros fazerem parte da imple-

mentação das funcionalidades “Ideias” e “Perfil de Utilizadores” relativa ao seu plano de trabalhos.

Recuando à aplicação móvel, o utilizador ao interagir com a pesquisa implementada na aplicação poderá visualizar o resultado desta (Figuras 5.27, 5.28 e 5.29) e navegar para a página correspondente ao elemento selecionado e no contexto selecionado. Com isto pretende-se indicar que o utilizador pode selecionar um desafio, ideia ou o perfil de um outro utilizador e visualizar com mais detalhe a entidade selecionada. No caso de não existirem resultados para a pesquisa do utilizador a aplicação móvel abre um *popup* indicando ao utilizador que não há resultados para a sua pesquisa, como pode ser visto na Figura 5.30.

Figura 5.27: Pesquisa de Desafios: Faculdade de Ciências

Figura 5.28: Pesquisa de Ideias: Estágio

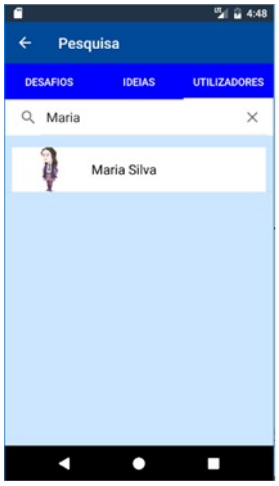


Figura 5.29: Pesquisa de Utilizadores: Maria

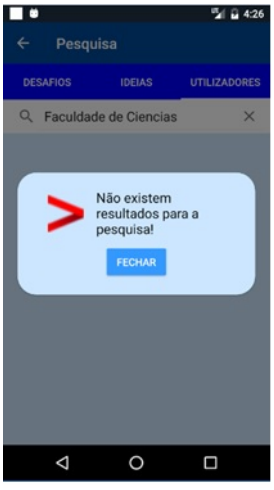


Figura 5.30: Sem resultados para a pesquisa

5.2.6 Funcionalidade Notificações

Foi decidido pela equipa que deveriam ser disponibilizadas notificações *push* e notificações internas aos utilizadores, sendo que este último tipo de notificações serão vistas numa secção da aplicação.

O intuito das notificações é manter o utilizador informado sobre eventos que ocorram e que possam ser do seu interesse, como por exemplo desafios que sejam criados, comentários a ideias que o utilizador segue ou ideias criadas nos grupos dos quais o utilizador é membro e sobre os quais pretende ser notificado.

As notificações podem ser distintas de alvo e classificadas em 4 níveis:

1. Todos os utilizadores da aplicação;
2. Para um conjunto de utilizadores;
3. Para o criador do desafio, da ideia ou para o administrador de um grupo;
4. Para o utilizador que foi aceite num grupo.

As notificações são de nível 1 quando for criado um novo desafio na aplicação.

As notificações são de nível 2 quando um utilizador segue um determinado desafio, ideia ou é membro de um grupo e pretende receber notificações desse grupo. Um utilizador recebe este tipo de notificações após os seguintes eventos:

- Escolha de vencedor para um desafio;
- Edição de um desafio;
- Criação de um comentário a um desafio ou adição de uma resposta a um comentário;
- Criação de uma ideia no desafio;
- Edição de uma ideia;
- Criação de um comentário a uma ideia ou adição de uma resposta a um comentário;
- Criação de uma ideia num grupo;
- Existência de um novo membro num grupo.

O utilizador recebe as notificações acima caso siga as entidades (desafio, ideia e grupo). Assim, caso algum destes eventos ocorra e o utilizador não siga a entidade em questão, este não será notificado.

As notificações são de nível 3 quando são direcionadas para o criador de um desafio, ideia ou para o administrador de um grupo. Estas podem ser de um dos seguintes tipos:

- Um utilizador passou a seguir um desafio ou uma ideia da sua autoria;

- Um utilizador colocou "gosto" numa ideia da sua autoria;

- Um utilizador pediu para aderir a um grupo do qual é administrador.

Apenas existe uma notificação de nível 4 e é direcionada a um utilizador que tenha pedido para aderir a um grupo. Esta notificação tem o intuito de indicar ao utilizador que foi aceite pelo administrador no grupo ao qual fez o pedido de adesão. A decisão pela utilização deste tipo de notificações deveu-se com a necessidade de diferenciar entre o utilizador que fez o pedido e o utilizador administrador do grupo.

Notificações internas – Detalhes da Implementação

Foi necessário a criação de duas tabelas relativas a esta funcionalidade da aplicação para que na tabela "Notificação" sejam registadas as notificações e todos os detalhes que as caracterizam, nomeadamente a descrição e o tipo de notificação que representam. A tabela "Notificações_Utilizador" guarda o ID da notificação registada na tabela "Notificação" e cria as entradas necessárias conforme os utilizadores que necessitem de ser notificados. Cada entrada possui o ID do utilizador correspondente.

- Explicação da visualização das notificações
  Foi adicionada uma secção na aplicação onde o utilizador poderá visualizar as suas notificações. De forma a permitir que o utilizador consiga distinguir entre as notificações que já leu e as que ainda não leu colocou-se diferentes cores de fundo e de texto para cada um dos casos.
  As notificações quando são abertas e mediante as notificações sejam relativas a um desafio, ideia ou a um grupo, é feita a navegação para o contexto indicado de forma a que o utilizador consiga verificar o evento que despoletou a notificação. Esta lista de notificações é ordenada de forma a que as primeiras notificações que aparecem sejam as notificações mais recentes. Portanto, existe uma *list view* na *view* relativa a esta secção da aplicação que está ligada a uma lista no *view model* correspondente. O procedimento efetuado ocorre da mesma maneira que nas funcionalidades anteriormente explicadas, sendo que o método invocado do *controller* da camada de serviços faz um *select* na tabela "Notificação" e "Notificações_Utilizador" para que sejam retornadas todas as notificações do utilizador atual. A Figura 5.31 representa a lista de notificações de um utilizador.

- Explicação da criação das notificações
  Para todos os casos de notificações indicados acima existem métodos implementados nos *view models* correspondentes que executam a criação/edição de um desafio ou um comentário a uma ideia tal como explicado nas funcionalidades anteriores. Para cada um desses casos foi criado um objeto que implementa uma interface que

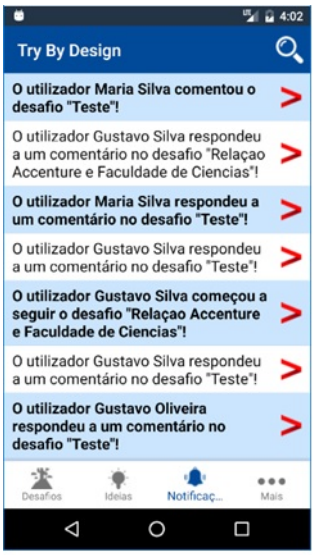


Figura 5.31: Lista de notificações de um utilizador

invoca o método pretendido e que ilustrará corretamente qual a notificação adequada no contexto dado. Na classe, que implementa a interface, existe um método chamado por todos os outros métodos e que tem o intuito de tratar do preenchimento dos restantes campos do objeto ”Notificação”que não foram preenchidos nos métodos que o invocam, nomeadamente a data de criação e de modificação. Após o preenchimento destes campos, o objeto “Notificação”, através da invocação do *service* das notificações, é enviado através de um pedido HTTP, tal como nas outras funcionalidades, para que o *controller* da camada de serviços o persista na base de dados.

A descrição de cada notificação, para além dos tipos de notificação que estas representam, é a parte mais importante desta funcionalidade porque é necessário que o utilizador perceba qual o evento que cada notificação representa. A identificação unívoca foi conseguida existindo um método declarado na interface para cada um dos eventos, excepto no caso dos comentários. Neste caso foi feito apenas um método que é invocado quando o utilizador comenta um desafio, uma ideia de um desafio, uma ideia espontânea, uma ideia de um grupo ou responde aos comentários “raízes” de todos os casos indicados anteriormente. Assim, para que se conseguisse na classe que implementa a interface indicar a descrição correta de cada caso de comentário foi necessário receber uma *string* que funciona como chave de identificação do tipo de comentário efetuado. Esta *string* é passada como parâmetro do método quando este é invocado no *view model* relativo aos comentários.

A camada de serviços quando recebe a ordem para persistir o objeto ”Notificação”, verifica quais os utilizadores que seguem a entidade onde ocorreu o evento da notificação e cria um objeto ”Notificações_Utilizador” para cada utilizador persis-

tido pelo *controller* responsável pela persistência desses objetos na tabela "Notificações_Utilizador".

Notificações *Push* – Detalhes da Implementação

Este tipo de notificações foi implementado para que os utilizadores, mesmo que não tenham a aplicação aberta, possam ser informados pelo despoletar de um evento. A sua utilização permite também aumentar o envolvimento e o uso da aplicação. Quando uma notificação é recebida, o utilizador ao clicar sobre esta fará com que a aplicação seja aberta e possa verificar a alteração mencionada na notificação.

O principal desafio deste tipo de notificações é a necessidade existente de criar um serviço que "escuta" pelas notificações que serão enviadas pelo Firebase Cloud Messaging (FCM), serviço que não poderá ser criado no código comum devido às diferenças das plataformas de Android e iOS.

Foi necessário criar um projeto no Firebase para que possam ser enviadas e recebidas as notificações e adicionar os pacotes relativos ao Google Play Services e ao Firebase Messaging para que a aplicação possa contactar com os serviços do FCM. Cada dispositivo ao instalar a aplicação recebe um *token* gerado pelo Firebase para a instância da aplicação instalada no dispositivo em questão, para que esta possa ser identificada e para que seja possível ao FCM enviar notificações para esse dispositivo. O FCM atualiza os *tokens* nas seguintes situações, de acordo com [73]:

- Quando a aplicação é instalada ou desinstalada;
- Quando o utilizador apaga os dados da aplicação;
- Quando a aplicação apaga a "Instance ID";
- Quando a segurança do *token* é comprometida.

Para mais detalhes relativos à implementação para a aplicação Android ver [73]. Quando um utilizador se regista na aplicação, é criado uma entrada na tabela "Utilizador" onde é armazenado o *token* que lhe corresponde bem como a chave correspondente à criação do grupo de dispositivos do utilizador. Cada utilizador no seu dispositivo poderá subscrever um tópico e receber *push notifications* relativos aos tópicos:

- Desafios;
- Ideias;
- Grupos.

Um utilizador quando passa a seguir um desafio, automaticamente passa a subscrever o tópico relativo a esse desafio. A *string* que identifica univocamente o tópico de um desafio

é "Desafio_X", em que X é o ID do desafio na base de dados. O mesmo procedimento é efetuado quando um utilizador segue uma ideia ("Ideia_X") e quando é membro de um grupo e passa a seguir esse mesmo grupo ("Grupo_X"), de forma a receber as notificações relativas à publicação de ideias ou à junção de novos membros ao grupo. No entanto, quando um utilizador deixa de seguir um tópico é executado o *unsubscribe* do tópico, passando a *string* que o identifica. Para o *subscribe* e *unsubscribe* de tópicos recorreu-se aos respetivos métodos da API do FCM que recebem como parâmetro uma *string* que identifica o tópico. Para os dispositivos receberem as notificações é necessário que estas sejam enviadas. Para tal, foi decidido pelo gestor do Projeto que seria a camada de serviços a componente do sistema responsável pela submissão das notificações para o Firebase. O Firebase, posteriormente, direciona, como indicado acima, para os dispositivos que deverão receber cada notificação.

Para os eventos identificados nas páginas 62 e 63 e sobre os quais são geradas notificações internas, foram também geradas *push notifications* para cada um desses eventos.

Dependendo do tipo de notificação, é interpretado se deverão ser enviadas *push notifications* para um conjunto de utilizadores ou para um utilizador em específico.
Caso o tipo de notificação seja do tipo 2, será enviada uma *push notification* para o tópico relativo ao desafio, ideia ou grupo, conforme o contexto. Todos os dispositivos utilizadores da aplicação subscrevem, de forma obrigatória, o tópico "Try_By_Design" que é o tópico onde são publicadas notificações relativas à criação de novos desafios.

O FCM identifica os dispositivos que subscreveram o tópico e encaminha a notificação para cada dispositivo indicando o evento que foi despoletado, representando isto um mecanismo de *Publish-Subscribe*.

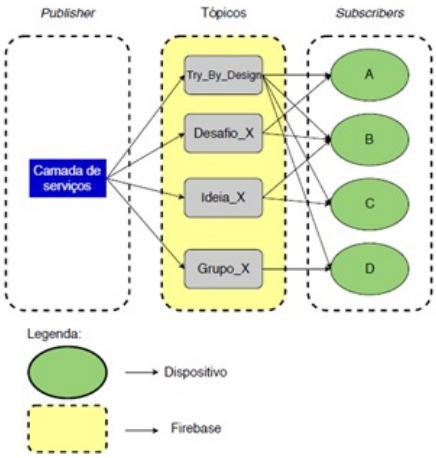


Figura 5.32: Mecanismo *Publish-Subscribe*

A Figura 5.32 representa o racional explicado acima referente às push notifications afeta aos tópicos. A camada de serviços envia as notificações para o Firebase que por sua

vez encaminha para cada dispositivo subscritor do tópico.

Os restantes tipos de notificações são notificações diretas ou seja, em vez do destinatário da notificação ser quem subscreveu um determinado tópico, o destinatário será um determinado utilizador, casos dos tipos de notificações 3 e 4.

Figura 5.33: Funcionamento do Firebase Cloud Messaging, retirado de [5]

A Figura 5.33 mostra o funcionamento do FCM. A consola do Firebase ou um servidor aplicacional envia as notificações para o FCM e depois estas são encaminhadas para os dispositivos.

A Figura 5.34 mostra *push notifications* recebidas pelo utilizador.

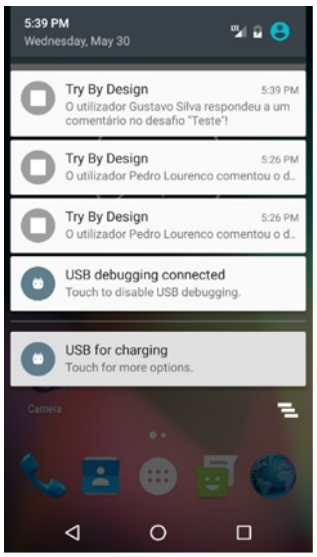


Figura 5.34: Lista de *push notifications* recebidas

O FCM permite criar grupos de dispositivos. Neste caso optou-se por criar um grupo de dispositivos afetos a cada utilizador para que este tipo de notificações fossem enviadas para todos os dispositivos utilizados por um utilizador. Para tal é necessário que

na camada de serviços, no método referente ao envio das *push notifications* se invoque um método responsável por retornar a chave que identifica o grupo de dispositivos do utilizador para o qual se pretende enviar a notificação.

Na finalização do registo de um utilizador é enviado um pedido HTTP por parte da aplicação móvel para que a camada de serviços envie, por sua vez, um pedido ao Firebase com o intuito de gerar uma chave para o grupo de dispositivos de um utilizador. Na criação do grupo é necessário que no pedido ao Firebase para a criação da chave exista um nome que identifique univocamente esse mesmo grupo (usou-se o ID do utilizador) e o *token* do dispositivo onde está a ser feito o registo do utilizador. A camada de serviços ao receber a *key* identificadora do grupo de dispositivos do utilizador faz a persistência desta chave na coluna "GrupoDispositivosKey" referente à entrada do utilizador na tabela "Utilizador" da base de dados.

Cada utilizador, já registado, ao fazer *login* num dispositivo diferente é executado um pedido pela aplicação móvel para a camada de serviços indicando que é necessário adicionar o *token* do dispositivo ao grupo de dispositivos do utilizador em questão para que este enquanto tem a sessão iniciada possa receber as suas notificações diretas para além das notificações de tópicos. No método da camada de serviços onde se processa o pedido para o Firebase adicionar o *token* ao grupo de dispositivos é necessário fazer uma chamada a um outro método que pede ao Firebase a chave que identifique o grupo de dispositivos do utilizador, pois a chave referente ao grupo está constantemente a ser alterada e assim garante-se que o pedido é efetuado para a chave mais recente.

O mesmo procedimento é efetuado para a remoção de um *token* do grupo mediante o utilizador faça *logout* na aplicação ou volte a um dispositivo que não seja o dispositivo mais recente onde fez *login*, apenas diferindo no método da camada de serviços a que a aplicação móvel executa o pedido. Por exemplo, um utilizador com *login* feito no seu telemóvel e que depois instalou a aplicação no seu *tablet* e, neste dispositivo, passou a seguir um novo desafio ou uma nova ideia. Quando este volta a tentar entrar na aplicação instalada no telemóvel é forçado a fazer de novo *login* para que o novo *token*, gerado pelo Firebase para o telemóvel, passe a subscrever os tópicos que o utilizador tenha subscrito no *tablet*, garantindo assim a sincronização para a receção das *push notifications* relativas a tópicos. No caso do utilizador não ter voltado a entrar na aplicação no telemóvel este continua sempre a receber todas as notificações, exceto as notificações relativas aos novos tópicos subscritos no *tablet*. Para além da remoção do *token* do grupo de dispositivos é também necessário que no *logout* se indique ao Firebase que apague o *token* do dispositivo (apagar a "Instance ID") e gerar um novo. Assim caso um outro utilizador faça *login* num telemóvel de um outro utilizador recebe apenas as suas notificações enquanto tiver a sessão iniciada.

O diagrama 5.35 ilustra o processo explicado acima:

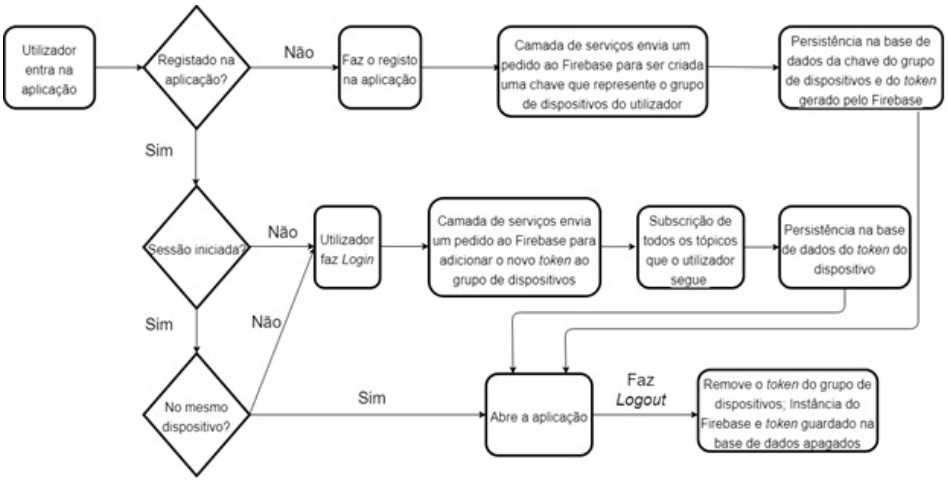


Figura 5.35: Diagrama explicativo dos grupos de dispositivos de um utilizador

5.2.7 Funcionalidade Prémios

De modo a incentivar o uso e a interação com a aplicação uma componente essencial do seu planeamento é a gamificação, isto é, mecanismos que promovam a aplicação como um jogo. Para tal foram desenvolvidas três componentes de gamificação na aplicação, nomeadamente, os prémios derivados da conquista de um desafio, um *ranking* que reflete os pontos ganhos conforme as ações efetuadas pelos utilizadores e os *badges* que cada utilizador conquista mediante o alcance de uma determinada meta.

Ranking e Atribuição de pontos

Quando um utilizador se regista na aplicação é criada uma entrada para o utilizador na tabela "Ranking"iniciada com 0 pontos.

Os utilizadores recebem pontos pela sua interação com a aplicação, sempre que executar uma das seguintes ações:

- Publicação de ideias espontâneas/ideias num desafio ou grupo;
- Comentar um desafio ou uma ideia;
- Vencer um desafio;
- Criar um grupo;
- Pertencer a um grupo;
- Seguir um desafio ou uma ideia;
- Colocar "gosto" numa ideia.

Cada caso de atividade indicado acima representa um número de pontos diferente sendo que também se optou por não atribuir pontos à criação de desafios pois essa funcionalidade é restrita a administradores da aplicação. Os pontos ganhos, por exemplo, na colocação de gosto numa ideia, por pertencer a um grupo ou por seguir um desafio são retirados quando o utilizador faz a ação inversa, de modo a não permitir aos utilizadores a angariação de pontos apenas para fins competitivos. As duas ações identificadas (soma e subtração de pontos) ocorrem na classe *view model* afeta ao Ranking. Cada um dos métodos invoca o *service* responsável por enviar um pedido para a camada de serviços que retorna o número de pontos do utilizador. O *view model* para saber o número de pontos a somar ou subtrair necessita de invocar o *service* responsável por retornar os pontos pré-definidos, para o caso de atividade, na tabela "Pontos". Posteriormente, executa-se uma das duas ações identificadas e envia-se para a camada de serviços que é responsável pela a atualização do número de pontos do utilizador na tabela "Ranking".

Esta componente da funcionalidade é representada por uma lista onde constam os utilizadores da aplicação ordenados pelos pontos conquistados através da sua interação com a aplicação. Esta lista aparece na *view* e é preenchida pelo *view model* correspondente a essa mesma *view*. Tal como nas funcionalidades anteriores existe o contacto entre a aplicação móvel e a camada de serviços onde é efetuado um *select* na tabela "Ranking" que retorna todas as entradas desta tabela ordenadas de forma descendente de pontuação.

Cada elemento da *list view* que representa o *ranking* foi estruturado numa *grid* com três colunas sendo que a primeira coluna indica a posição do utilizador, a segunda coluna indica o nome e sobrenome do utilizador e a última coluna indica o número de pontos do utilizador, como pode ser visto na Figura 5.36.

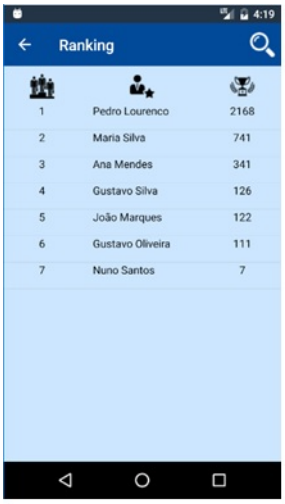


Figura 5.36: *Ranking*

Vencedor do Desafio

Quando um desafio é concluído cabe a um utilizador do tipo "Challenger" escolher o seu vencedor. Para executar essa ação existe um ícone que representa uma taça na página de detalhes de um desafio que ao ser clicado abre um *popup* com todos os utilizadores que participaram no desafio através da publicação de ideias ou de comentários ao desafio. No caso de nenhum utilizador ter participado no desafio aparece uma mensagem no *popup* indicando que ninguém participou no desafio, como poderá ser visto na Figura 5.38.

O primeiro *popup* indicado está estruturado no ficheiro xaml através de duas *grids*. A primeira *grid* possui duas colunas em que na primeira coluna aparece o logótipo da empresa e a segunda coluna aparece a *list view* com o nome dos utilizadores. A segunda *grid* possui, de igual modo, duas colunas com dois botões em que um deles tem o intuito de escolher o vencedor e o outro o de fechar o *popup*, ficando cada um em cada uma das colunas.

O *popup* relativo à indicação que nenhum utilizador participou no desafio encontra-se explicado na funcionalidade "Mensagens Alerta".

Os utilizadores participantes no desafio são obtidos através de um pedido HTTP que a aplicação móvel, no *service* responsável pelos pedidos relacionados com os desafios, envia à camada de serviços passando o ID do desafio como parâmetro do pedido. A camada de serviços retorna uma lista com todos os utilizadores criadores de ideias e comentários no desafio e que ao ser obtida pelo *view model*, responsável pela lógica de apresentação do desafio, preenche a *list view* do *popup* (Figura 5.39). Um utilizador administrador da aplicação seleciona um utilizador da lista e submete o vencedor através de pressionar o botão "Ok" existente no *popup* que invoca um método *command* do *view model*. Este método cria um objeto "Vencedor_Desafio" com o ID do utilizador vencedor e com o ID do desafio, para além de outros campos. De seguida invoca-se o *service* dos desafios que envia um pedido para a camada de serviços que cria uma entrada na tabela "Vencedor_Desafio" com o objeto "Vencedor_Desafio".

A página de detalhe do desafio passa a indicar o vencedor no meio da imagem do desafio (Figura 5.37).

Badges

A componente de *badges* foi implementada para premiar os utilizadores quando atingem uma determinada meta, quer seja na criação de ideias, na conquista de desafios ou em qualquer outra interação com a aplicação identificadas na secção "Ranking".

Quando um utilizador se regista na aplicação são criadas entradas para o utilizador na tabela "Badges_Utilizador" conforme o número de *badges* existentes (que se encontra definido na tabela "Tipo_Badge").

Esta componente encontra-se visível na aplicação numa secção do perfil de cada utilizador e imita uma caderneta onde se evidenciam as medalhas conquistadas. As medalhas

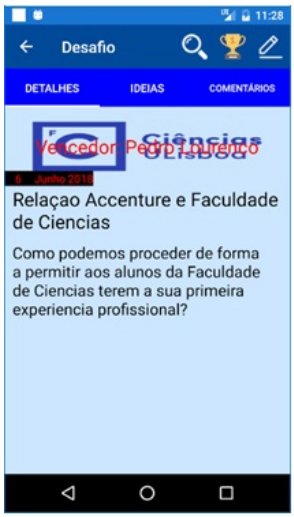


Figura 5.37: Vencedor do desafio no meio da imagem e ícone (taça) para escolher o vencedor

Figura 5.38: *Popup* indicando que ninguém participou no desafio

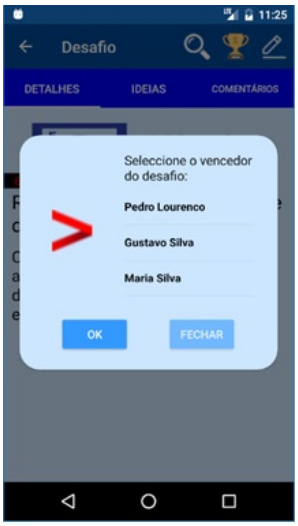


Figura 5.39: *Popup* para escolher o vencedor

poderão ser de bronze, prata ou ouro. Por exemplo, se vencer um desafio o utilizador recebe uma medalha de bronze, se vencer cinco desafios recebe uma medalha de prata e caso vença dez desafios recebe uma medalha de ouro. Esta componente está englobada na funcionalidade de Perfil, que pertence ao plano de trabalhos do Gustavo, e foram adicionados os *badges* no perfil dos utilizadores com o seu conhecimento.

A camada de serviços é a componente do sistema responsável pela atribuição das medalhas aos utilizadores. Quando ocorre uma ação suscetível de atribuir uma medalha é efetuado um *select* na tabela do contexto (caso seja a publicação de uma ideia espontânea tem que se verificar quantas ideias espontâneas existem da autoria do utilizador em questão) que retorna o número de vezes em que um utilizador aparece nas condições pretendidas. Por exemplo, o *select* seguinte retorna o número de presenças em grupos de um utilizador:
”SELECT COUNT(Utilizador_Criacao) FROM Membros_Grupo WHERE (Utilizador_Criacao = @p0 AND E_Aprovado = 1)”;
Usando o resultado obtido em cada *select* efetuado para cada entidade, é invocado um método que verifica o tipo de *badge* que está em questão, e compara-se com os valores pretendidos, para atingir cada tipo de medalha, da tabela “Tipo_Badge”. Caso o retorno do *select* seja superior ou igual a algum dos valores das medalhas, é alterada a entrada da tabela “Badges_Utilizador” que faz a interligação entre o ID do utilizador e o ID correspondente ao tipo de *badge* indicado para que o valor booleano da medalha em questão passe a ser *true*, evidenciando assim que o utilizador passou a possuir aquela medalha para a atividade indicada.

Como indicado mais acima, esta componente da funcionalidade reflete uma caderneta de medalhas. A implementação desta caderneta é composta por uma *list view*. Esta lista é preenchida pelo *view model* que está ligada à *view* que, como nas funcionalidades anteriores, chama o *service* que por um pedido REST à API da camada de serviços obtém todos os *badges* do utilizador específico. Após a obtenção da lista de *badges*, no *view model* é criada uma lista de "Badge_Caderneta". Este objeto "Badge_Caderneta" pertence aos *models* e possui três propriedades *ImageSource*, cada uma correspondente aos diferentes tipos de medalhas. Existem ainda duas propriedades que referem a descrição e a informação, provenientes da tabela "Tipo_Badge" que indicam as condições necessárias para o utilizador vencer cada um dos tipos de medalha, como se verifica na Figura 5.41. É esta lista de "Badge_Caderneta" que será depois visualizada pelo utilizador por estar ligada à *list view* incluída na *view* responsável pela visualização da caderneta. Cada elemento desta *list view* está estruturado dentro de uma *grid* dividida em quatro colunas, cada uma delas representando a descrição do *badge*, a medalha de bronze, prata e ouro, como ilustra a Figura 5.40.

Para que o utilizador saiba que se selecionar um *badge* abre um *popup* indicando as condições para vencer as medalhas do *badge* selecionado foi colocado um *popup* que aparece sempre que o utilizador navega para a página de *badges* do perfil. Este *popup* indica ao utilizador que caso este pretenda verificar as condições de cada *badge* terá que selecioná-lo, como se verifica na Figura 5.42.

Figura 5.40: Caderneta de *badges* de um utilizador

Figura 5.41: *Popup* indicando as condições para vencer as medalhas do *badge*

Figura 5.42: *Popup* indicando ao utilizador como poderá ver as condições para vencer cada *badge*

Capítulo 6

Testes

Todo o *software* tem *bugs* e, como tal, é muito importante testar todo o código desenvolvido de forma a verificar que todas as funcionalidades desenvolvidas se comportam da maneira esperada.

Tal como referido por Ahamed em [74], testar o *software* é uma importante fase do ciclo de vida do desenvolvimento de *software* e um elemento crítico para assegurar a sua própria qualidade. O processo de teste é composto por um conjunto de atividades que podem ser planeadas antecipadamente e é baseado nos conceitos de verificação e validação. Verificação refere-se a um conjunto de atividades que asseguram que o *software* implementa corretamente uma função específica e validação refere-se às atividades que asseguram que o *software* foi construído de forma a cobrir os requisitos pedidos pelo cliente.

Para testar o *software* desenvolvido foram executados testes às *user stories* escritas, testes de usabilidade e de compatibilidade explicados nas secções seguintes. À medida que foi efetuado o desenvolvimento das várias funcionalidades foram sempre feitos testes unitários aos métodos escritos de modo a verificar que o retorno destes era o esperado para os vários casos de teste. No entanto não documentei este tipo de testes devido à elevada quantidade de métodos implementados.

A versão Windows do IDE utilizado para o desenvolvimento não permite a compilação de código para dispositivos iOS e devido a essa razão os testes foram apenas executados para dispositivos Android. Para testar a aplicação para os dispositivos portadores das diferentes versões do sistema operativo iOS seria necessário compilar o código num computador com o sistema operativo macOS da Apple. Como não foi disponibilizado nenhum computador com o sistema operativo indicado não foi possível testar a aplicação em ambiente iOS.

6.1 Testes às *User Stories*

No contexto deste projeto foi indicado pelo gestor do Projeto que deveriam ser executados testes às *user stories* elaboradas ao longo do desenvolvimento de *software*, para verificar que o *software* desenvolvido respeitava os critérios de aceitação. Fiquei responsabilizado de executar testes a todas as *user stories* (cerca de 36) que incluíssem as funcionalidades relativas ao meu plano de trabalhos, sendo que as restantes *user stories* ficaram atribuídas ao Gustavo.

Identificou-se os possíveis casos de teste para cada critério de aceitação de cada *user story* e colocou-se os resultados esperados e obtidos numa tabela que identifica a(s) *user story* testada(s).

O Anexo B mostra as tabelas relativas aos testes das *user stories* indicadas na Secção 3.3.2 do Capítulo 3.

Testes às *User Stories* das *push notifications*

Para testar as *push notifications* tive que optar por um método diferente. Para verificar que dois dispositivos recebiam as mesmas *push notifications* foi necessário instalar a aplicação em dois dispositivos diferentes de modo a que estes pertencessem ao grupo de dispositivos de um utilizador. Com o utilizador que tinha *login* feito nos dois dispositivos passei a seguir um desafio, um grupo, uma ideia espontânea, uma ideia de um desafio e uma ideia de um grupo.

Após a instalação, executei a aplicação com recurso ao emulador do IDE na conta de um outro utilizador da aplicação e fiz os casos de teste que abrangessem os vários eventos identificados na funcionalidade "Notificações" (páginas 62 e 63). Para verificar que o utilizador com *login* efetuado nos dispositivos físicos não receberia notificações de tópicos que não tinha subscrito comentei um desafio que este não seguia, criei uma ideia num grupo do qual não fazia parte e passei a seguir um desafio que não tinha sido criado por esse mesmo utilizador.

Os resultados obtidos foram ao encontro dos resultados esperados pois os dispositivos receberam as *push notifications* de eventos que ocorreram nos tópicos que o utilizador seguia e não receberam qualquer notificação relativa aos tópicos que não seguia.

Testei também no caso de fazer *logout* no último dispositivo físico em que tinha feito *login*, pois era o *token* deste último que estava persistido na base de dados. Essa ação tinha o intuito de removê-lo do grupo de dispositivos e verificar se continuaria a receber notificações. Com o emulador comentei um desafio seguido pelo o utilizador dos dispositivos físicos e apenas o dispositivo mantido no grupo de dispositivos recebeu a *push notification*. De novo, fiz *login* no dispositivo anteriormente removido do grupo de dispositivos para que fosse adicionado o novo *token* deste dispositivo ao grupo do utilizador. Com o emulador voltei a comentar um desafio seguido tendo os dois dispositivos recebido

a *push notification*.

6.2 Testes de Usabilidade

Testar a usabilidade é uma condição *sine qua non* para qualquer tipo de *software* desenvolvido pois permite entender de que forma potenciais utilizadores interagem com a aplicação móvel desenvolvida.

Existem diversas definições de usabilidade e a norma ISO 9241 ([75]) estabeleceu a sua definição no modo como um produto pode ser utilizado por utilizadores específicos a completarem determinadas tarefas com eficácia, eficiência e satisfação num determinado contexto de utilização.

Nayebi, Desharnais e Abran em [76] consideram que é importante considerar três aspetos na usabilidade de todos os tipos de *software*:

- Eficiência de utilização: "levar pouco tempo a completar uma tarefa";
- Facilidade de aprendizagem: "as operações podem ser aprendidas observando o objeto";
- Maior satisfação do utilizador: "atende às expectativas do utilizador".

Considerando os testes de usabilidade para aplicações móveis, Zhang e Adipat em [77] identificaram diversos desafios a ter em conta, nomeadamente o contexto móvel, a conexão da rede, o pequeno tamanho dos ecrãs e as suas diferentes resoluções ou a limitada capacidade de processamento.

Os autores de [76] consideram a existência de 3 metodologias para estudos de usabilidade em contexto *mobile*:

- Experiências de laboratório: utilizadores que realizam tarefas específicas utilizando uma aplicação móvel num ambiente controlado, com o uso de um emulador ou de um dispositivo móvel físico, de acordo com [77];
- Estudos de campo: utilizadores recebem a aplicação móvel num dispositivo móvel físico e são questionados relativamente à experiência de utilização;
- Medições práticas: existem métodos de avaliação quantitativa para avaliar a usabilidade de uma aplicação.

Execução dos testes

No contexto do projeto foram questionados 12 utilizadores para executarem um conjunto de 18 tarefas indicadas na Tabela 6.1. Na Tabela 6.2 estão indicados os resultados esperados para cada uma das tarefas.

Optei que as tarefas que visam a funcionalidade de "Comentários" e de "Seguir" apenas ocorressem nos desafios para que os utilizadores não necessitassem de efetuar trabalho repetido nos restantes casos onde se procede do mesmo modo, como comentar ideias espontâneas, de desafios ou de grupos e seguir ideias e grupos.

Dos 12 utilizadores, 10 utilizam dispositivos portadores do sistema operativo Android e os restantes utilizam dispositivos iOS. As idades das pessoas inquiridas variou entre os 19 e os 51 anos e todos os utilizadores usam aplicações móveis diariamente.

Tarefa	Descrição
1	Criar um desafio
2	Comentar um desafio
3	Responder a um comentário
4	Pesquisar por um utilizador
5	Pesquisar por uma ideia
6	Pesquisar por um desafio
7	Editar um desafio/Selecionar uma imagem diferente da atual
8	Visualizar o *ranking*
9	Visualizar os *badges*
10	Ver as condições para vencer as medalhas de um *badge*
11	Seguir um desafio
12	Deixar de seguir um desafio
13	Ver os desafios que está a seguir
14	Ver a lista de desafios
15	Ver os desafios concluídos
16	Visualizar as notificações
17	Escolher o vencedor para um desafio concluído
18	Distinguir entre notificações lidas e não lidas

Tabela 6.1: Tarefas de teste de usabilidade

Tarefa	Resultados Esperados
1	Na página da aplicação com os desafios existentes o utilizador carrega no botão "Criar Desafio". Abre um *popup* e preenche o título, a descrição, prémios, data de fim e seleciona uma imagem.
2	Na página de "Comentários" de um desafio o utilizador escreve o que pretende comentar na entrada localizada no fim da página e, após a escrita, submete o comentário clicando no ícone do comentário.
3	Na página de "Comentários" de um desafio o utilizador carrega no ícone de comentar num comentário já existente. Abre um *popup* com o comentário alvo da resposta e com uma entrada para o utilizador introduzir o texto. Após a escrita, submete o comentário clicando no ícone do comentário.
4	O utilizador seleciona o ícone relativo à pesquisa. Navega até à página "Utilizadores" e escreve na *search bar*.
5	O utilizador seleciona o ícone relativo à pesquisa. Navega até à página "Ideias" e escreve na *search bar*.

Tabela 6.2 continua a partir da página anterior

6	O utilizador seleciona o ícone relativo à pesquisa. Navega até à página "Desafios" e escreve na *search bar*.
7	O utilizador ao navegar para a página de um desafio seleciona o ícone relativo à edição do desafio na *toolbar*. Abre um *popup* com a informação atual do desafio. O utilizador seleciona o botão para selecionar uma nova imagem da galeria do dispositivo e após a escolha submete as alterações ao pressionar o botão "Editar Desafio".
8	O utilizador seleciona a secção "Mais" na *bottom bar* e seleciona "Ranking".
9	O utilizador seleciona a secção "Mais" na *bottom bar* e seleciona "Perfil". No "Perfil" navega para a página "Badges".
10	Na página "Badges" do perfil do utilizador, o utilizador seleciona um *badge*. Abre o *popup* com as condições necessárias para vencer as medalhas do *badge*.
11	O utilizador seleciona um desafio existente. Aberta a página de detalhes, o utilizador seleciona o ícone da campainha cinzenta na *toolbar*. A campainha passa a ser amarela.
12	O utilizador seleciona um desafio existente. Aberta a página de detalhes, o utilizador seleciona o ícone da campainha cinzenta na *toolbar*. A campainha passa a ser amarela.
13	Na secção relativa aos desafios o utilizador navega para a página "Desafios a Seguir".
14	Na secção relativa aos desafios o utilizador navega para a página "Lista de Desafios".
15	Na secção relativa aos desafios o utilizador navega para a página "Desafios Concluídos".
16	O utilizador seleciona a secção "Notificações" na *bottom bar*.
17	Na secção relativa aos desafios o utilizador navega para a página "Desafios Concluídos". Carrega num desafio e seleciona o ícone da taça na *toolbar*. Abre um *popup* com a lista de utilizadores que participaram no desafio. O utilizador seleciona um utilizador e submete o vencedor ao pressionar o botão "Ok".
18	O utilizador indica quais as notificações lidas e não lidas.

Tabela 6.2: Resultados esperados para as tarefas de usabilidade

Para cada tarefa indicada na Tabela 6.1 verificou-se se o utilizador conseguiu terminar a tarefa e avaliou-se a dificuldade de cada utilizador na execução destas, tendo sido utilizada uma escala de avaliação: 1 (muito fácil), 2 (fácil), 3 (médio), 4 (difícil) e 5 (muito difícil).

Todos os utilizadores conseguiram realizar todas as tarefas na íntegra mas avaliaram as tarefas com diferentes níveis de dificuldades. A avaliação de cada tarefa encontra-se representada no gráfico da Figura 6.1 e reflete a percentagem da escolha de cada nível, da escala de avaliação, para cada tarefa.

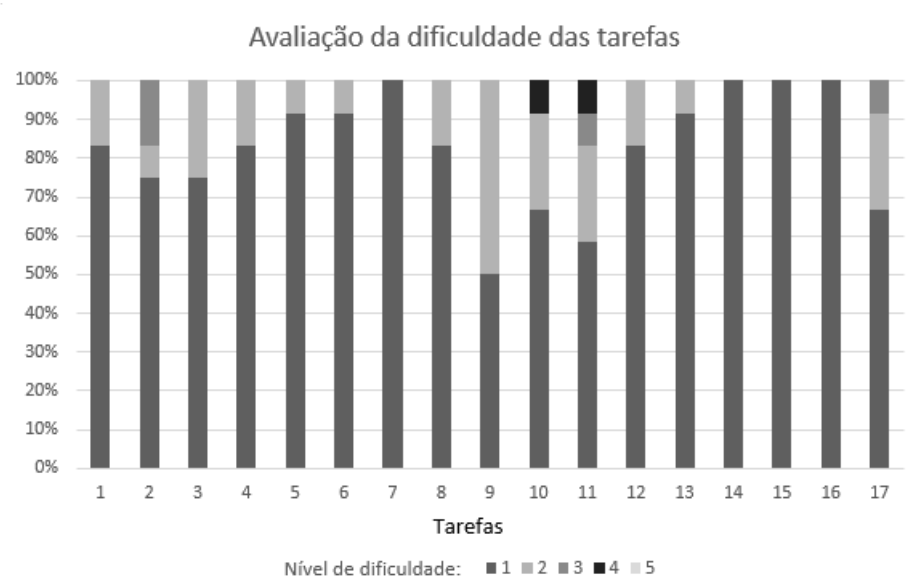


Figura 6.1: Percentagem dos níveis escolhidos para cada tarefa

A tarefa 18 não se encontra no gráfico pois foi avaliada de forma diferente das restantes. Nesta tarefa apenas se avaliava se o utilizador identificava de forma correta as notificações lidas e não lidas, sendo 1 (identificado corretamente) e 0 (identificação errada).

- Análise dos resultados indicados no gráfico da Figura 6.1:

No geral prevaleceram os níveis de dificuldade 1 e 2 na avaliação de cada tarefa indicando que os utilizadores não tiveram dificuldade na realização de cada uma.

As tarefas 2, 10, 11 e 17 foram as tarefas que causaram uma maior dificuldade aos utilizadores pois em cada uma destas pelo menos um utilizador indicou um nível superior a 2 na avaliação da dificuldade. Na tarefa 2, dois utilizadores indicaram o nível 3 para a tarefa pois não perceberam de forma imediata que teriam de escrever o seu comentário na entrada relativa a "Adicionar um comentário..." e submeter o comentário através do clique no ícone de comentar. Na tarefa 10, um utilizador necessitou de navegar três vezes para a *tab* de *badges* do perfil para ler o *popup* que informa o utilizador de que ao carregar sobre um *badge* poderá visualizar de forma detalhada as condições para vencer as medalhas do *badge* selecionado. O mesmo utilizador identificou a tarefa com o nível 4 de dificuldade pois necessitou de algum tempo para perceber como poderia completar a tarefa. Na tarefa 11, dois utilizadores indicaram o nível de dificuldade 4 e 3 para a tarefa pois tiveram alguma dificuldade em interpretar que ao pressionar o ícone "campainha" na *toolbar* passariam a seguir o desafio. A tarefa 17 suscitou também alguma dificuldade num utilizador que indicou o nível 3 para a tarefa indicando que poderia existir uma forma mais simples de selecionar o vencedor para um desafio.

As restantes tarefas não criaram dificuldades aos utilizadores tendo existido alternância em avaliar as tarefas com o nível 1 e 2.

Apesar da tarefa 18 não estar indicada no gráfico, foi atingida uma percentagem de 100% no que remete à identificação correta das notificações lidas e não lidas.

No geral os utilizadores indicaram que o *design* estava intuitivo e de fácil aprendizagem para quem viesse a utilizar a aplicação.

- Sugestões dadas pelos utilizadores
  Apesar de não ter existido grandes dificuldades na realização das tarefas os utilizadores deixaram algumas sugestões de melhoria, nomeadamente:

  – Tarefa 1: alterar a cor do botão que permite criar desafio para um azul mais escuro para focar mais a atenção do utilizador;

  – Tarefa 3: alterar o ícone de comentar num comentário que permite publicar uma resposta ao comentário para um botão que indique "Responder";

  – Tarefa 4: permitir uma única pesquisa que mostre todos os resultados da pesquisa ao invés da pesquisa por "Desafios", "Ideias" e "Utilizadores";

  – Tarefa 11: alterar o ícone "campainha" por um ícone mais ilustrativo do que é a ação de "Seguir", talvez um ícone com uma "estrela";

  – Tarefa 16: colocar para cada notificação a data respetiva, na lista de notificações;

  – Tarefa 17: selecionar o vencedor do desafio através de uma *checkbox*.

6.3 Testes de Compatibilidade

Este tipo de testes foram executados com o objetivo de verificar o desempenho da aplicação em diferentes dispositivos com distintos sistemas operativos, tamanhos e resoluções de ecrã.

É fundamental verificar alguns dos múltiplos ambientes existentes no mercado para verificar que o *software* desenvolvido está apto para as várias plataformas utilizadas pelos potenciais utilizadores.

A aplicação móvel tem como mínimo *target* o Android 4.2 e foram realizados apenas testes a dispositivos portadores deste e de sistemas operativos mais recentes. A aplicação móvel foi testada nos seguintes dispositivos:

- Emulador (Nexus 5)
- One Plus One
- LeNovo k5

- Huawei P Smart
- Huawei Y5II
- Samsung Galaxy J5
- Vodafone Smart N8
- Sony Xperia E3
- Samsung Galaxy Tab 3 Lite (*Tablet*)

Os sistemas operativos utilizados para os testes foram:

- Android 8.0
- Android 7.1
- Android 7.0
- Android 6.0
- Android 5.1
- Android 4.4
- Android 4.2

A aplicação executou sem problemas nos vários dispositivos e sistemas operativos. Ao executar a aplicação no Android 8.0 do Huawei P Smart verificou-se um ligeiro aumento no desempenho da aplicação no que remete à navegação entre as diferentes interfaces da aplicação.

Capítulo 7

Conclusões e Trabalho Futuro

Neste Projeto de Engenharia Informática procedeu-se à implementação de uma aplicação móvel através do paradigma do desenvolvimento *cross-platform*. Este paradigma permite o desenvolvimento de um só código fonte para aplicações evitando a duplicação de esforço permitindo gerar executáveis para os sistemas operativos mais comuns (Android e iOS) utilizados em dispositivos móveis.

A aplicação móvel desenvolvida tem o objetivo da resolução de desafios de negócio por parte dos colaboradores da Accenture através da publicação de ideias e comentários. É composta ainda por outras funcionalidades identificadas na Secção 3.2.

Para a implementação da aplicação móvel recorreu-se à plataforma Xamarin, mais concretamente à *framework* Xamarin.Forms. Para além de um único código relativo à lógica de negócio da aplicação permite de igual modo, a reutilização de código entre as diversas plataformas, implementando interfaces únicas com recurso a uma biblioteca comum. Os elementos de UI desta biblioteca são mapeados em *run time* nos elementos nativos das plataformas.

A utilização do padrão arquitetural MVVM permite uma separação entre as várias componentes em que se divide a aplicação móvel e possibilita que futuramente se possa acrescentar novas funcionalidades ou que se alterem as existentes sem grande esforço. A linguagem C# utilizada para o desenvolvimento é bastante importante pois devido às características assíncronas que possui permite que as aplicações continuem em execução e não fiquem a aguardar uma resposta quando fazem um pedido, por exemplo, à base de dados.

Com os resultados e análises dos testes de usabilidade verifica-se que o *design* adotado está intuitivo para os utilizadores. Verifica-se que a utilização da *framework* Xamarin.Forms foi suficiente para criar interfaces intuitivas que permitem ao utilizador uma rápida aprendizagem das várias funcionalidades da aplicação e uma boa experiência de utilização.

Como poderá ser verificado em [78], a plataforma Xamarin é uma excelente ferramenta na resposta ao desenvolvimento *cross platform* pois permite criar aplicações nati-

vas. O desempenho das aplicações desenvolvidas é, também, nativo pois são compiladas para aproveitar o aceleramento do *hardware* específico da plataforma em que a aplicação está a ser executada ao invés de soluções que interpretam código em tempo de execução.

Apesar dos problemas identificados (limitações da biblioteca comum e *resize*) na secção seguinte, na minha opinião terei que tomar em maior consideração todas as vantagens que já apresentei relativamente à plataforma e à *framework* Xamarin.Forms. Os benefícios que poderão permitir às empresas, nomeadamente diminuir restrições temporais e poupar recursos monetários e humanos aliado ao IDE Visual Studio que permite uma implementação cómoda com recurso a uma linguagem orientada a objetos (C#) e, principalmente, à qualidade das aplicações desenvolvidas, assegura um futuro auspicioso para a plataforma no que remete ao mercado de desenvolvimento *cross platform mobile*.

7.1 Problemas encontrados durante o desenvolvimento e soluções adoptadas na sua resolução

Os principais problemas encontrados no desenvolvimento da aplicação foram:

- Limitações da biblioteca comum do Xamarin.Forms
  A biblioteca comum possui algumas limitações e, por vezes, torna-se necessário recorrer a propriedades dos elementos ou a elementos de UI que não se encontram presentes nesta levando à necessidade do desenvolvimento de código específico para as diferentes plataformas.
  Um dos exemplos que verifiquei relativamente a essa questão prendeu-se com o uso de um elemento de *design* "Editor" que tem como função permitir ao utilizador escrever em múltiplas linhas. Este elemento foi usado para o utilizador escrever a descrição de um desafio aquando da sua criação ou edição. No entanto este elemento não possuía uma propriedade "Placeholder" necessária para indicar ao utilizador qual o conteúdo que deverá inserir. O texto relativo a esta propriedade (Figura 5.12: "Descrição do desafio") desaparece quando o utilizador insere o texto da descrição. Na Figura 5.13 não aparece o texto do "Placeholder" pois o utilizador já tem inserida a descrição do desafio.
  Para ultrapassar esta limitação foi necessário criar um *Editor* customizável adaptado às minhas necessidades, que estende as propriedades do elemento original com o acrescento da propriedade "Placeholder". No entanto como não existe esta propriedade na biblioteca, teve que ser mapeada na propriedade das bibliotecas de iOS e Android. Na implementação das interfaces em vez do uso do *Editor* da biblioteca comum utilizou-se o *Editor* customizável.
  Criou-se uma classe em cada um dos projetos (Android e iOS) e quando na implementação da interface (código comum) é usada a propriedade "Placeholder" é

invocada a classe referente ao sistema operativo do dispositivo que mapeia para a propriedade nativa, como por exemplo "Hint" no Android.

- *Resize* das imagens dos desafios
  Para além do exemplo acima que levou à necessidade de escrita de código específico para cada plataforma existiu também a necessidade relativamente ao *resize* das imagens dos desafios para que todas tivessem as mesmas medidas. Para tal, criou-se uma interface que continha um método que recebia o conteúdo da imagem num *byte array*, e a altura e largura para que tinha de ser alterada. Este método foi implementado em cada uma das classes dos projetos Android e iOS. Estas classes são invocadas através de *dependency injection* pois ao criar um objeto da interface é criado, por sua vez, um objeto da classe respetiva que a implementa, conforme o sistema operativo do dispositivo que executa a aplicação.
  Em cada um dos métodos das classes executa-se o código necessário ao *resize* da imagem com recurso a elementos nativos, como *UIImage* para iOS e *Bitmap* para Android. O método retorna um *byte array* onde se encontra a imagem com as alterações pretendidas e é este *byte array* que é colocado pelo código comum através de uma *stream* no Firebase, como indicado na Secção 5.2.2 (Selecionar imagem para o desafio e armazenamento no Firebase).

- Armazenamento/apresentação das imagens aos utilizadores
  Tal como indicado na funcionalidade dos desafios foi solicitado que estes fossem ilustrados com uma imagem que poderia ser escolhida pelo utilizador que criasse o desafio ou, posteriormente, alterada para uma outra imagem.
  No entanto foram verificados problemas nomeadamente a nível do tempo em que era divulgada a lista de desafios existentes. Esta lista apresenta a imagem de cada um dos desafios e, caso fossem carregados todos os desafios sem a solicitação das suas imagens esta lista era apresentada quase instantaneamente não se verificando o mesmo quando havia necessidade do carregamento de todas as imagens. O tempo de processamento atingia cerca de 20 segundos até ao aparecimento de todos os desafios existentes e correspondentes imagens.
  Após a constatação deste problema, que levava a uma má experiência de utilização por parte dos utilizadores da aplicação, foram várias as tentativas de resolução, explicadas de seguida:

  1. Inicialmente, cada imagem era guardada na tabela "Imagem"da base de dados como um *byte array*. Posteriormente, para que os utilizadores pudessem visualizar a lista de desafios existentes fazia-se um *select* na tabela "Desafio"com o intuito de ir buscar todos os desafios existentes e de seguida um *for each* sobre cada desafio para selecionar a imagem que o ilustrasse mas este processamento, tal como referido acima, demorava cerca de 20 segundos. Nesta fase

da resolução do problema pensei que o problema se devia à conversão do *byte array* para a *ImageSource*, conversão esta que é necessária para que a imagem possa ser visualizada no ecrã;

2. Utilizei o Firebase para guardar a imagem e em vez de guardar o *byte array* na tabela "Imagem"passou-se a guardar apenas o URL do Firebase onde a imagem estava armazenada. No entanto, o problema persistiu e através da técnica de *debug* percebi que o problema da *performance* se devia a um *for each* que ocorria na camada de serviços, o *for each* indicado no passo anterior;
3. Para resolver este problema integrei o *select* de todos os desafios com o *select* da imagem de cada desafio num único *select* para evitar o processamento feito no *for each*. Então pensei que o problema ficaria resolvido com este novo *select* que integrava os outros dois *selects* e que poderia, de novo, guardar o *byte array* da imagem, ao invés do URL, e, desse modo, não necessitava da utilização do Firebase;
4. O tempo de visualização diminui um pouco mas não o suficiente para dar uma boa experiência de utilização aos utilizadores da aplicação. Após isto comparei os tempos quando a lista de desafios não possuia nenhum desafio com imagem e quando a lista de desafios tinha apenas um desafio com imagem e os outros sem imagem para verificar se existiria diferenças na demonstração da lista ao utilizador. Para o primeiro caso a visualização ocorria de forma eficiente enquanto que no segundo caso a visualização tornava-se lenta. Através de pedidos diretos à camada de serviços, com recurso ao URL da API desenvolvida na camada de serviços e que invoca o *controller* pretendido, verifiquei os tamanhos das transferências que ocorriam entre a camada de serviços e a aplicação móvel, para os dois casos indicados acima.
   Tamanhos das transferências:
   - Lista de desafios em que nenhum tem imagem: 25 kb;
   - Lista de desafios em que apenas 1 tem imagem: 499 kb.

   Apesar de ambas as transferências não serem muito grandes existe uma certa diferença entre estas suficiente para interferir na experiência de utilização;
5. De modo a que a transferência fosse mais eficiente voltou-se a fazer uso do Firebase para guardar as imagens, de modo a que fosse apenas guardado na tabela "Imagem"da base de dados o URL, como explicado no passo 2, e não o *byte array*.

- *Push Notifications*
  A implementação das *push notifications* passou por duas etapas antes da solução final e que é indicada na funcionalidade "Notificações".

Na primeira etapa da implementação era guardado apenas o *token* do primeiro dispositivo que o utilizador utilizava para interagir com a aplicação. No entanto esta solução tinha as seguintes limitações:

- O utilizador inicia a aplicação noutro dispositivo que não o primeiro utilizado e não receberia qualquer notificação, exceto no caso de passar a seguir um novo tópico nesse dispositivo. Neste caso o Firebase "registava" que o *token* deste novo dispositivo utilizado passava a seguir um novo tópico;
- Caso um outro utilizador entrasse na aplicação num dispositivo de um outro utilizador iria receber as notificações do utilizador portador do dispositivo e não as suas;
- Caso acontecesse alguma das situações indicadas na funcionalidade "Notificações" (Notificações *Push* - Detalhes da Implementação) como por exemplo o Firebase detetar que o *token* tinha sido comprometido, o utilizador não voltaria a receber notificações diretas pois o *token* guardado na base de dados seria o *token* comprometido. De igual modo, não receberia notificações dos tópicos seguidos com esse *token*.

A segunda etapa da implementação apenas diferia da implementação final no que diz respeito à abrangência dos grupos de dispositivos. A limitação envolvida consistia que apenas o dispositivo mais recentemente utilizado receberia as *push notifications* diretas, pois na base de dados era o *token* mais recente que estava persistido. Na versão final da implementação desta funcionalidade todas as limitações detetadas nos parágrafos anteriores foram ultrapassadas.

7.2 Trabalho Futuro

Após a realização do trabalho poderão existir vertentes que ao serem adicionadas tornam a aplicação mais rica e completa. As principais componentes que poderiam ser adicionadas são:

- Sincronização *offline*: permitir ao utilizador que as alterações que efetue na aplicação em modo *offline* sejam persistidas quando voltar a estar conectado à internet;
- Pesquisa com *autocomplete*: auxiliar o utilizador quando pesquisa. Isto permitiria indicar ao utilizador algumas palavras que poderia pesquisar mediante o que este for colocando na *search bar*;
- *Analytics*: o gestor do projeto indicou que seria interessante numa perspetiva futura implementar uma componente de análise à aplicação. Com esta componente seria

possível analisar, por exemplo, as funcionalidades mais utilizadas, quantos utilizadores interagem com a aplicação ou, no caso da utilização no seio da empresa, fazer comparações entre os utilizadores dos vários departamentos. À margem do efetuado para o armazenamento das imagens dos desafios e da implementação das *push notifications* poderia ser utilizado uma ferramenta do Firebase (ver referência [79]) que auxilia o processo pretendido;

- *Push notifications* para iOS: para os dispositivos iOS da aplicação poderem receber *push notifications* seria necessário ter uma conta como Apple Developer. Devido a não ter acesso à conta necessária e a não ter acesso a um dispositivo iOS para testar esta componente da funcionalidade "Notificações", no futuro deverá ser implementada esta parte;

- Plataforma *web*: reaproveitamento da API desenvolvida na camada de serviços para o desenvolvimento da aplicação para a plataforma *web*;

- Integração de testes com o Xamarin Test Cloud: permite testar a aplicação recorrendo a testes unitários, de UI e de integração. Pode ser um complemento aos testes efetuados e indicados no capítulo anterior de modo a dar um maior suporte à avaliação e testagem da aplicação;

- Múltiplos idiomas: com a visão de integrar a aplicação a nível global, quer na empresa ou em clientes, seria importante permitir ao utilizador escolher o idioma em que pretende visualizar o seu conteúdo.

Bibliografia

[1] CP Rahul Raj and Seshu Babu Tolety. A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach. In *India Conference (INDICON), 2012 Annual IEEE*, pages 625–629. IEEE, 2012.

[2] Model-view-controller. `https://msdn.microsoft.com/en-us/library/ff649643.aspx`. Último acesso em: 15.Dezembro.2017.

[3] Implementing the model-view-viewmodel pattern. `https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff798384(v=pandp.10)`. Último acesso em: 18.Junho.2018.

[4] Difference between mvvm and mvp. `http://www.differencebetween.net/technology/difference-between-mvvm-and-mvp/`. Último acesso em: 15.Dezembro.2017.

[5] Firebase cloud messaging. `https://docs.microsoft.com/en-us/xamarin/android/data-cloud/google-messaging/remote-notifications-with-fcm?tabs=vswin`. Último acesso em: 13.Junho.2018.

[6] Spyros Xanthopoulos and Stelios Xinogalos. A comparative analysis of cross-platform development approaches for mobile applications. In *Proceedings of the 6th Balkan Conference in Informatics*, pages 213–220. ACM, 2013.

[7] Yu Beng Leau, Wooi Khong Loo, Wai Yip Tham, and Soo Fun Tan. Software development life cycle agile vs traditional approaches. In *International Conference on Information and Network Technology*, volume 37, pages 162–167, 2012.

[8] Gartner says worldwide sales of smartphones grew 7 percent in the fourth quarter of 2016. `https://www.gartner.com/newsroom/id/3609817`. Último acesso em: 15.Dezembro.2017.

[9] Desktop vs mobile vs tablet market share worldwide. `http://gs.statcounter.com/platform-market-share/desktop-mobile-tablet`. Último acesso em: 15.Dezembro.2017.

[10] Suliman S Aljomaa, Mohammad F Al Qudah, Ismael S Albursan, Salaheldin F Bakhiet, and Adel S Abduljabbar. Smartphone addiction among university students in the light of some variables. *Computers in Human Behavior*, 61:155–164, 2016.

[11] Christian Montag, Konrad Błaszkiewicz, Rayna Sariyska, Bernd Lachmann, Ionut Andone, Boris Trendafilov, Mark Eibes, and Alexander Markowetz. Smartphone usage in the 21st century: who is active on whatsapp? *BMC research notes*, 8(1):331, 2015.

[12] Mobile operating system market share worldwide. `http://gs.statcounter.com/os-market-share/mobile/worldwide`. Último acesso em: 15.Dezembro.2017.

[13] Sobre a accenture. `https://www.accenture.com/pt-pt/company`. Último acesso em: 16.Dezembro.2017.

[14] Accenture - cultura e valores. `https://www.accenture.com/us-en/careers/team-culture-values`. Último acesso em: 16.Dezembro.2017.

[15] How to choose the right architecture for your mobile application. `https://www.rapidvaluesolutions.com/wp-content/uploads/2013/04/How-to-Choose-the-Right-Technology-Architecture-for-Your-Mobile-Application.pdf`. Último acesso em: 14.Dezembro.2017.

[16] Nicolás Serrano, Josune Hernantes, and Gorka Gallardo. Mobile web apps. *IEEE software*, 30(5):22–27, 2013.

[17] Pavel Smutnỳ. Mobile development tools and cross-platform solutions. In *Carpathian Control Conference (ICCC), 2012 13th International*, pages 653–656. IEEE, 2012.

[18] Lisandro Delia, Nicolas Galdamez, Pablo Thomas, Leonardo Corbalan, and Patricia Pesado. Multi-platform mobile application development analysis. In *Research Challenges in Information Science (RCIS), 2015 IEEE 9th International Conference on*, pages 181–186. IEEE, 2015.

[19] Kotlin and android. `https://developer.android.com/kotlin/`. Último acesso em: 18.Junho.2018.

[20] Phonegap. `http://phonegap.com/about/`. Último acesso em: 14.Dezembro.2017.

[21] Xamarin. `https://www.xamarin.com/`. Último acesso em: 15.Dezembro.2017.

[22] Lisandro Procedi. Avaliação do framework xamarin. forms para desenvolvimento de aplicativos móveis multiplataforma, criando uma aplicação real. 2016.

[23] Amer A Radi. Evaluation of xamarin forms for multiplatform mobile application development. 2016.

[24] Desenvolvimento em xamarin: Nossa experiência. `http://www.opus-software.com.br/desenvolvimento-em-xamarin`. Último acesso em: 15.Dezembro.2017.

[25] Manuel Palmieri, Inderjeet Singh, and Antonio Cicchetti. Comparison of cross-platform mobile development tools. In *Intelligence in Next Generation Networks (ICIN), 2012 16th International Conference on*, pages 179–186. IEEE, 2012.

[26] Jose Fermoso. Phonegap seeks to bridge the gap between mobile app platforms. `https://gigaom.com/2009/04/05/phonegap-seeks-to-bridge-the-gap-between-mobile-app-platforms/`. Último acesso em: 15.Dezembro.2017.

[27] Arvind Ravulavaru. *Learning Ionic*. Packt Publishing Ltd, 2015.

[28] Ionic. `https://ionicframework.com/`. Último acesso em: 15.Dezembro.2017.

[29] Ionic concepts. `https://ionicframework.com/docs/v1/concepts/structure.html`. Último acesso em: 15.Dezembro.2017.

[30] The mvvm pattern. `https://msdn.microsoft.com/en-us/library/hh848246.aspx`. Último acesso em: 15.Dezembro.2017.

[31] Jason H. Christensen. Using restful web-services and cloud computing to create next generation mobile applications. In *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications*, OOPSLA '09, pages 627–634, New York, NY, USA, 2009. ACM.

[32] Simple object access protocol overview. `https://docs.oracle.com/cd/A97335_02/integrate.102/a90297/overview.htm`. Último acesso em: 15.Dezembro.2017.

[33] André Jorge. Desenvolvimento de plataforma para monitorização de consumos em edifícios (frontend). `https://estudogeral.sib.uc.pt/bitstream/10316/35605/1/Desenvolvimento%20de%20plataforma%20para%20monitorizacao%20de%20consumos%20em%20edificios_Frontend.pdf`. Último acesso em: 15.Dezembro.2017.

[34] Service architecture - soap. https://www.service-architecture.com/articles/web-services/soap.html. Último acesso em: 15.Dezembro.2017.

[35] Raghu Ramakrishnan and Johannes Gehrke. *Database management systems*. McGraw Hill, 2000.

[36] Rick Cattell. Scalable sql and nosql data stores. *Acm Sigmod Record*, 39(4):12–27, 2011.

[37] Ameya Nayak, Anil Poriya, and Dikshay Poojary. Type of nosql databases and its comparison with relational databases. *International Journal of Applied Information Systems*, 5(4):16–19, 2013.

[38] Systems development life cycle: Objectives and requirements. http://www.benderrbt.com/Bender-SDLC.pdf. Último acesso em: 15.Dezembro.2017.

[39] Oksana Ņikiforova, Uldis Sukovskis, and Vladimirs Ņikuļšins. Integration of mda framework into the model of traditional software development. *publication. editionName*, pages 229–239, 2008.

[40] Davor Gornik. Ibm rational unified process: Best practices for software development teams. ftp.software.ibm.com/software/rational/web/whitepapers/2003/rup_bestpractices.pdf. Último acesso em: 15.Dezembro.2017.

[41] Victor Szalvay. An introduction to agile software development. *Danube technologies*, pages 1–9, 2004.

[42] Tore Dybå and Torgeir Dingsøyr. Empirical studies of agile software development: A systematic review. *Information and software technology*, 50(9):833–859, 2008.

[43] David F Rico and CSM PMP. What is the roi of agile vs. traditional methods? *TickIT International*, 10(4):9–18, 2008.

[44] Kai Petersen and Claes Wohlin. A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case. *Journal of systems and software*, 82(9):1479–1490, 2009.

[45] LEOR Vijayasarathy and Dan Turk. Agile software development: A survey of early adopters. *Journal of Information Technology Management*, 19(2):1–8, 2008.

[46] Crowdicity - mobile. http://crowdicity.com/en/product/accessing-crowdicity/. Último acesso em: 16.Dezembro.2017.

[47] Crowdicity. http://crowdicity.com/en/product/. Último acesso em: 16.Dezembro.2017.

[48] Openideal - ideation management platform. http://www.openidealapp.com/public-innovation/. Último acesso em: 16.Dezembro.2017.

[49] Openideal - full features list. https://www.openidealapp.com/public-innovation/openideal-full-features-list/. Último acesso em: 16.Dezembro.2017.

[50] Openideo. https://openideo.com/how-open-ideo-works. Último acesso em: 16.Dezembro.2017.

[51] Openideo - questions. https://challenges.openideo.com/faq. Último acesso em: 16.Dezembro.2017.

[52] Justinmind. https://www.justinmind.com/. Último acesso em: 13.Junho.2018.

[53] 10 mobile app designs for user experience inspiration. https://1stwebdesigner.com/mobile-apps-designs/. Último acesso em: 16.Junho.2018.

[54] The best mobile app ui designs of 2016. https://blog.proto.io/best-mobile-app-ui-designs-2016/. Último acesso em: 16.Junho.2018.

[55] Alan Brown, Simon Johnston, and Kevin Kelly. Using service-oriented architecture and component-based development to build web service applications. *Rational Software Corporation*, 6, 2002.

[56] Asp.net web api. https://msdn.microsoft.com/en-us/library/hh833994(v=vs.108).aspx. Último acesso em: 16.Dezembro.2017.

[57] Fundamentos do entity framework 4. https://msdn.microsoft.com/pt-br/library/jj128157.aspx. Último acesso em: 16.Dezembro.2017.

[58] Visual studio. https://www.visualstudio.com/pt-br/vs/. Último acesso em: 13.Junho.2018.

[59] Introdução ao c#. https://docs.microsoft.com/pt-br/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework. Último acesso em: 13.Junho.2018.

[60] Programação assíncrona com c#. https://msdn.microsoft.com/pt-br/library/hh191443(v=vs.120).aspx. Último acesso em: 13.Junho.2018.

[61] James Buri and Lori Lalonde. *Pro XAML with C# - From Design to Deployment on WPF, Windows Store, and Windows Phone*. Apress, 2015.

[62] Jim Melton. Sql language summary. *ACM Comput. Surv.*, 1996.

[63] Firebase. `https://firebase.google.com/firebase-and-gcp/?authuser=0`. Último acesso em: 13.Junho.2018.

[64] Firebase - produtos. `https://msdn.microsoft.com/pt-br/library/hh191443(v=vs.120).aspx`. Último acesso em: 13.Junho.2018.

[65] Microsoft sql server management studio 1. `https://docs.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-2017`. Último acesso em: 13.Junho.2018.

[66] Microsoft sql server management studio 2. `https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-2017`. Último acesso em: 13.Junho.2018.

[67] Microsoft sql server. `https://en.wikipedia.org/wiki/Microsoft_SQL_Server`. Último acesso em: 13.Junho.2018.

[68] Scott Chacon and Ben Straub. *Pro Git*. Apress, Berkely, CA, USA, 2nd edition, 2014.

[69] Gitextensions - visual studio. `https://marketplace.visualstudio.com/items?itemName=HenkWesthuis.GitExtensions`. Último acesso em: 29.Junho.2018.

[70] Gitextensions. `http://gitextensions.github.io/`. Último acesso em: 29.Junho.2018.

[71] Sharing code overview. `https://docs.microsoft.com/pt-pt/xamarin/cross-platform/app-fundamentals/code-sharing`. Último acesso em: 16.Junho.2018.

[72] Isaac Pontes. Xamarin e o desenvolvimento de aplicações mobile multiplataforma: um estudo de caso com geofence. `http://bd.centro.iff.edu.br/bitstream/123456789/2036/1/Texto.pdf`. Último acesso em: 16.Junho.2018.

[73] Xamarin android - remote notifications with firebase cloud messaging. `https://docs.microsoft.com/en-us/xamarin/android/data-cloud/google-messaging/remote-notifications-with-fcm?tabs=vswin`. Último acesso em: 13.Junho.2018.

[74] SS Ahamed. Studying the feasibility and importance of software testing: An analysis. *CoRR*, 2010.

[75] ISO. ISO 9241-11:1998 Ergonomic requirements for office work with visual display terminals (VDTs) – Part 11: Guidance on usability. Technical report, International Organization for Standardization, 1998.

[76] F. Nayebi, J. M. Desharnais, and A. Abran. The state of the art of mobile application usability evaluation. In *2012 25th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2012.

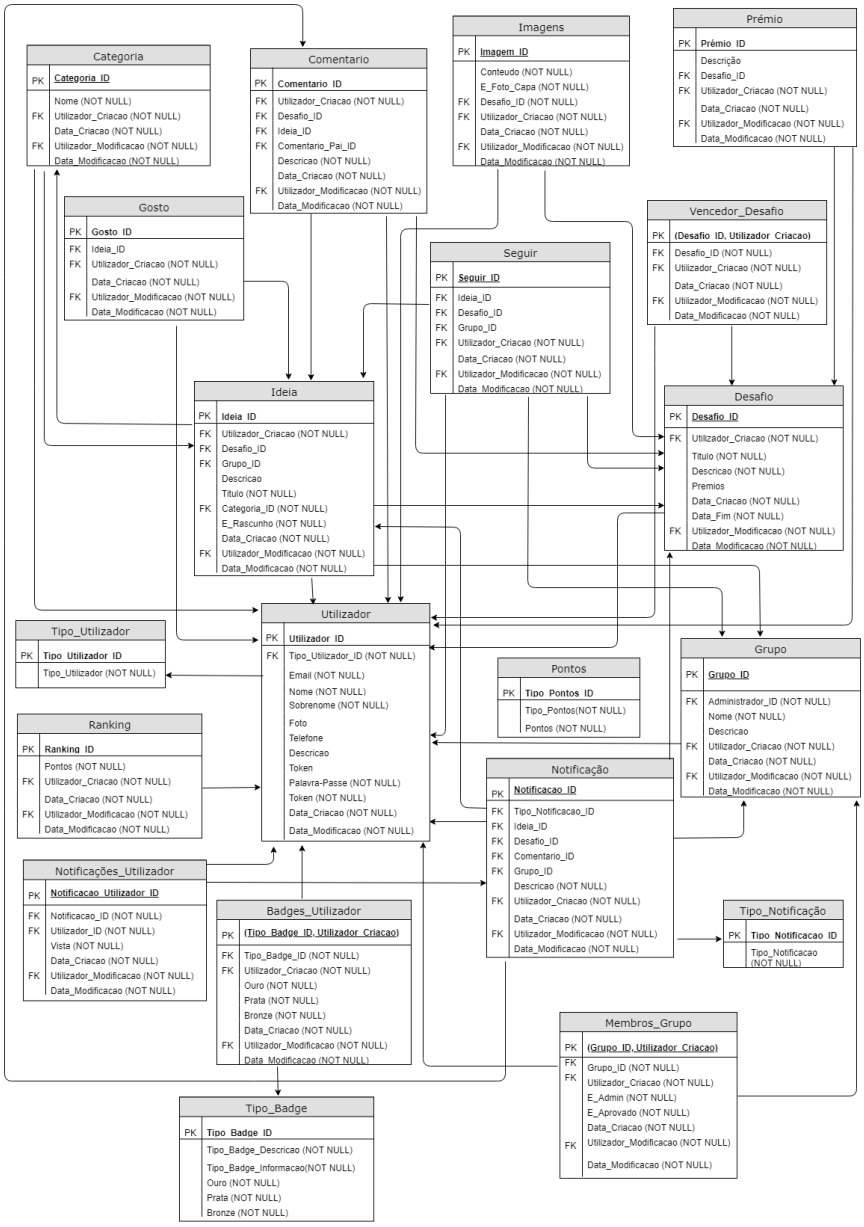
[77] Dongsong Zhang and Boonlit Adipat. Challenges, methodologies, and issues in the usability testing of mobile applications. *International Journal of Human–Computer Interaction*, 2005.

[78] Ferramentas do visual studio para xamarin. `https://www.visualstudio.com/pt-br/xamarin/?rr=https%3A%2F%2Fwww.xamarin.com%2F`. Último acesso em: 17.Junho.2018.

[79] Google analytics para firebase. `https://firebase.google.com/docs/analytics/`. Último acesso em: 18.Junho.2018.

Apêndice A

Modelo Entidade Relação



Racional das tabelas:

- Categoria - esta tabela tem o objetivo de representar as categorias das ideias. É composta pelo campo "Nome" que representa o nome da categoria e pelo seu ID que representa a sua *Primary Key*.

- Ideia - representa as ideias que um utilizador publique. A ideia é composta por um ID único, por uma descrição onde o utilizador irá indicar a sua ideia com maior detalhe, por um título que representa o assunto da ideia, por uma categoria e por um campo *boolean* que representa um valor onde o utilizador indica na criação da ideia se esta será apenas inicialmente um rascunho (que apenas este conseguirá ler) ou se publicará para que os outros utilizadores possam ler o seu ponto de vista. Para além destes campos existem dois campos adicionais pois as ideias poderão ser espontâneas ou poderão ser ideias afetas a um desafio ou a um grupo. Estes dois campos podem ser *null* e representam duas *Foreign Keys* para as tabelas Desafio e Grupo, respetivamente.

- Comentário – composto por uma *Primary Key* que identifica com um ID cada objeto da tabela. Esta tabela é composta por uma descrição, onde o utilizador escreve o texto que deseja que os outros utilizadores leiam. Possui ainda três *Foreign Keys* uma vez que um comentário pode ser feito no contexto de um desafio, de uma ideia ou de um outro comentário.

- Desafio – composto pelo título que refere o tema do desafio, pela descrição que indica com maior detalhe o desafio e pelos possíveis prémios que um desafio possa ter. Este último campo pode ser *null* pois não é obrigatório que um desafio tenha prémios. Esta tabela é ainda composta pela sua chave-primária que identifica de forma única cada objeto da tabela.

- Grupo – composto pelo nome que indica o nome do grupo, pelo "Administrador_ID", que é um campo *not null* e uma *Foreign Key* para a tabela Utilizador, pois todos os grupos terão que ter um administrador de forma a permitirem a entrada de outros utilizadores para o grupo e por uma descrição onde se fará uma breve introdução ao contexto do grupo.

- Membros_Grupo – esta tabela tem uma dupla chave primária composta pelo ID do grupo e pelo utilizador que é membro desse grupo, pois para não adicionarmos um novo campo para o ID do utilizador que faz parte do grupo utilizamos o campo de "Utilizador_Criacao" para fazer essa mesma função. Para além desses campos existem ainda dois campos *booleans* de que dão informação se um utilizador é administrador do grupo ou, no que diz respeito ao outro campo, se está aprovado como membro deste.

- Tipo_Utilizador – esta tabela tem o objetivo de representar os tipos de utilizador da aplicação, nomeadamente se é um administrador da aplicação ou um utilizador normal. É composta pelo campo "Nome" que representa o nome da categoria e pelo seu ID que representa a *Primary Key*.

- Utilizador - um utilizador é composto por um e-mail, palavra-passe, nome e sobrenome, que são campos obrigatórios. Cada utilizador tem também um tipo associado que corresponde a uma *Foreign Key* para a tabela "Tipo_Utilizador". Existem campos que poderão ser *null* por não serem obrigatórios como o caso da descrição, da foto e do número de telefone. O utilizador possui ainda um campo "Token" que indica o *token*, mais recente, do dispositivo que este utilizou, e um campo "GrupoDispositivosKey" que serve para verificar se o utilizador já tem uma chave que agrega todos os *tokens* de dispositivos onde este já utilizou a aplicação.

- Seguir - para além da sua chave primária que identifica com um ID cada objeto desta tabela existem ainda três campos que correspondem a *Foreign Keys* para as tabelas "Ideia", "Desafio" e "Grupo". Fizemos uso do campo obrigatório "Utilizador_Criacao" para preencher com o ID do utilizador que segue uma das entidades referidas.

- Gosto – tem o mesmo racional da tabela "Seguir".

- Imagens – esta tabela corresponde às imagens dos desafios. Cada objeto desta tabela possui, para além dos quatro campos obrigatórios, um campo "URL" onde será colocado o URL da imagem no Firebase e um campo *boolean* "E_Foto_Capa" que indica se a imagem é a imagem principal do desafio.

- Pontos - esta tabela funciona como uma tabela apenas para referência. Nesta tabela encontram-se persistidos os pontos que o utilizador pode ganhar mediante a ação que este executa. A coluna "Tipo_Pontos" indica a descrição da ação e a coluna "Pontos" indica o valor numérico desta.

- *Ranking* - esta tabela reflete o número de pontos que cada utilizador tem devido à interação com a aplicação e os seus pontos são alterados conforme a ação executada e o valor númerico desta, definidos na tabela "Pontos". A coluna "Pontos" indica os pontos do utilizador.

- Tipo_Notificação - esta tabela indica os tipos de notificação possíveis na aplicação. A coluna "Tipo_Notificacao" descreve o tipo de notificação.

- Notificação - cada objeto desta tabela indica uma notificação ocorrida na aplicação. Para além dos quatro campos obrigatórios possui uma coluna "Descricao" que descreve o evento ocorrido, uma coluna "Tipo_Notificacao_ID" que é uma chave de

ligação (*foreign key*) com a tabela "Tipo_Notificacao" para saber de que tipo é a notificação em questão. Possui ainda quatro colunas "Ideia_ID", "Desafio_ID", "Grupo_ID" e "Comentario_ID" que são preenchidos com os IDs da entidade em questão conforme o evento ocorrido.

- Notificações_Utilizador - funciona como uma tabela de ligação entre um utilizador (coluna "Utilizador_ID") e uma notificação (coluna "Notificacao_ID") sendo que para cada notificação são criadas as entradas na tabela conforme o número de utilizadores que tenham que ser notificados. Cada entrada possui uma coluna "Vista" que representa um único valor conforme a notificação tenha sido ou não vista pelo o utilizador dessa mesma entrada.

- Tipo_Badge - tabela que tem o mesmo intuito das tabelas "Tipo_Utilizador" e "Tipo_Notificacao". Reflete os *badges* existentes indicando na coluna "Tipo_Badge_Descricao" a descrição do *badge*, a coluna "Tipo_Badge_Informacao" indica uma descrição textual com as condições para vencer uma medalha do *badge*, e as colunas "Ouro", "Prata" e "Bronze" onde estão refletidos os valores numéricos predefinidos para o utilizador vencer a medalha.

- Badges_Utilizador - cada utilizador tem tantas entradas nesta tabela quanto os números de *badges* existentes na tabela "Tipo_Badge" e funciona como uma tabela de ligação entre essa mesma tabela e a tabela "Utilizador". Possui ainda três campos "Ouro", "Prata" e "Bronze" que indicam se o utilizador possui para o *badge*, dessa entrada, as respetivas medalhas.

- Vencedor_Desafio - cada entrada desta tabela indica um vencedor de um desafio. A coluna "Utilizador_Criacao" indica o ID do utilizador vencedor e a coluna "Desafio_ID" indica o desafio que este venceu.

Apêndice B

Tabelas - Testes *User Stories*

US0019: Criar desafios
Engloba a US0040 (receção de notificação) e a US0061 (escolher imagem para o desafio)

Caso de Teste	**Parâmetros de Teste**	**Resultados Esperados**	**Resultados Obtidos**	**Observações**
Permissão para criação de desafios	Utilizador do tipo “Challenger”	Na página dos desafios existe um botão para criar um novo desafio; Ao ser clicado abre um *popup* onde poderá criar um novo desafio		
	Utilizador de um outro tipo	Não existe um botão para criação de um novo desafio		
Entrada para escrever título do desafio		O *popup* para criação do desafio tem uma caixa de texto para permitir ao utilizador inserir o título do desafio		

Tabela B.1 continua a partir da página anterior

Entrada para escrever descrição do desafio		O *popup* para criação do desafio tem uma caixa de texto para permitir ao utilizador inserir a descrição do desafio		
Entrada para inserir prémios do desafio		O *popup* para criação do desafio tem uma caixa de texto para permitir ao utilizador inserir os prémios do desafio		
Entrada para escolher data de fim do desafio		O *popup* tem um "date picker" para escolher a data de fim do desafio		
Escolher imagem do desafio		Ao carregar no botão abre a galeria do dispositivo do utilizador para este selecionar a imagem que melhor ilustre o desafio; Após a seleção, no *popup* poderá ver a imagem que selecionou enquanto acaba de criar o desafio		
	Título *null*	Aparece no ecrã um *popup* indicando que é necessário preencher os campos obrigatórios		

Tabela B.1 continua a partir da página anterior

Título vazio	Aparece no ecrã um *popup* indicando que é necessário preencher os campos obrigatórios		
Título com mais de 100 caracteres	Aparece no ecrã um *popup* indicando que o desafio não foi criado		
Descrição *null*	Aparece no ecrã um *popup* indicando que é necessário preencher os campos obrigatórios		
Descrição vazia	Aparece no ecrã um *popup* indicando que é necessário preencher os campos obrigatórios		
Descrição com mais de 1000 caracteres	Aparece no ecrã um *popup* indicando que o desafio não foi criado		
Prémios com mais de 500 caracteres	Aparece no ecrã um *popup* indicando que o desafio não foi criado		
Data de Fim < Data Atual	Não permitir que o utilizador escolha uma data anterior à data atual		

Tabela B.1 continua a partir da página anterior

Submissão do desafio	(Título != *null* && Título != vazio && Título <100 caracteres) && (Descrição != *null* && Descrição != vazio && Descrição <1000 caracteres) && (Prémios <500 caracteres || Prémios == *null* || Prémios == vazio) && (Data de Fim >= Data Atual)	Aparece no ecrã um *popup* indicando se o desafio foi corretamente adicionado à base de dados		
Envio de notificação interna		Todos os utilizadores da aplicação receberam a notificação		
Lista de desafios		Abre a página com a lista de desafios onde aparece o desafio criado		

Data de Teste: 18/05/2018

US0028: Ver notificações internas
Engloba a US0045 (distinguir as notificações lidas das não lidas) e US0054 (navegar para o contexto da notificação)

Caso de Teste	**Parâmetros de Teste**	**Resultados Esperados**	**Resultados Obtidos**	**Observações**	**Re-Teste**
Secção "Notificações"		Na secção "Notificações" do menu existe uma lista que reflete as notificações recebidas devido ao despoletar de alguns eventos			

Tabela B.2 continua a partir da página anterior

Diferença nas notifi-cações lidas e não lidas		Na lista de notificações é percetível quais as notificações lidas e não lidas; As notificações não lidas têm o texto a *bold* e um *background* diferente das notificações lidas			
Clicar numa notifi-cação	Notificação relativa a um desafio	Abre a página de detalhes do desafio			
	Notificação relativa a uma ideia espontânea	Abre a página de detalhes da ideia espontânea		Deu erro devido a não conter o novo código feito pelo Gustavo pois a funcionalidade “Ideias” faz parte do seu plano de trabalhos	
	Notificação relativa a um grupo	Abre a página de detalhes do grupo			
Notifi-cação clicada		A notificação clicada passa a ser uma notificação lida; As restantes notificações ficam de acordo com o estado que tinham anteriormente			

Data de Teste e Re-Teste: 09/05/2018

US0031: Comentários raízes a desafios
Engloba a US0044 (receber uma notificação quando alguém comenta um desafio que sigo) e a US0059 (receção de pontos devido a comentar um desafio)

Caso de Teste	**Parâmetros de Teste**	**Resultados Esperados**	**Resultados Obtidos**	**Observações**
Clicar num desafio		Ao clicar num desafio deverá abrir a página deste com os seus detalhes, ideias e comentários		
Entrada para escrever comentário		Na *tab* dos comentários na página do desafio existe uma entrada (caixa de texto) que permite ao utilizador comentar o desafio		
Submissão de comentário	Comentário null	Aparece no ecrã um *popup* indicando que não é possível submeter comentários vazios		
	Comentário vazio	Aparece no ecrã um *popup* indicando que não é possível submeter comentários vazios		
	Comentário com menos ou com 500 caracteres	Aparece no ecrã um *popup* indicando se o comentário foi corretamente adicionado à base de dados		
	Comentário com mais de 500 caracteres	Aparece no ecrã um *popup* indicando que o comentário não foi adicionado à base de dados		
Envio de notificação interna		Todos os utilizadores que seguem o desafio receberam a notificação		
		Um utilizador que não siga o desafio não recebeu a notificação		

Tabela B.3 continua a partir da página anterior

Atribuição de prémio		O utilizador que submeteu o comentário recebe um *popup* evidenciando os pontos que ganhou; o *ranking* reflete a soma do número de pontos anteriores com os pontos recebidos deste evento		
		Um utilizador que não tenha submetido o comentário não recebe pontos		

Data de Teste: 17/05/2018