INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




HelpIt | Aplicação Mobile Interativa para Apoio Social

ANDRÉ MIGUEL GUIMARÃES DE SOUSA
Outubro de 2019

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Aplicação Mobile Interativa para Apoio Social

"HELP IT"

André Miguel Guimarães de Sousa

1130346

Tese para obtenção do Grau de Mestre em

Engenharia Informática, Área de Especialização em

Sistemas Gráficos e Multimédia

Orientador: Filipe de Faria Pacheco (ffp@isep.ipp.pt)

Porto, Fevereiro de 2019

The moment when you want to quit, is the moment when you need to keep pushing.

Resumo

Este documento relata todo o processo de estudo, desenvolvimento e avaliação do projeto "HelpIt - Aplicação móvel interativa para apoio social", uma aplicação móvel que visa auxiliar as instituições e organizações sociais na coleta de donativos, através de um simples jogo de atribuição de donativos, desafios semanais e colecionamento de crachás.

Numa fase inicial, são identificadas as dificuldades que as instituições e organizações de solidariedade social sentem, nos dias de hoje, ao angariar fundos de apoio. É ainda feita uma descrição daquilo que se pretende da aplicação, bem como das motivações e oportunidades que conduziram à escolha do tema. Finaliza-se o capítulo inicial com estrutura detalhada do presente documento.

Num segundo capítulo, é feita uma análise ao estado da arte atual. Inicia-se a análise com uma pesquisa acerca de algumas componentes da psicologia humana, onde são abordados temas como o perfil do colecionador, a necessidade básica de integração e a partilha de feitos e conquistas. Mais à frente é descrito o estudo efetuado ao atual processo de doação, mencionando os meios de doação disponibilizados por algumas das mais conhecidas organizações, Cruz Vermelha, Banco Alimentar e Unicef. A descrição deste processo vê-se finalizada com uma breve redação acerca dos benefícios, atribuídos pelo Fisco, aos eventuais doadores. Em seguida é realizada uma extensa análise de valor, onde se apresenta a proposta de valor deste projeto, tendo em conta os modelos de negócio Peter Koen, Canvas, Verna Allee, cadeia de valor de Porter e um exemplo prático do método AHP.

Concluído o retrato do estado da arte, surge o capítulo de design. Nesta secção são enumeradas e explicadas todas as tecnologias e soluções possíveis para a concretização do produto final, descrevendo uma análise ao cumprimento dos requisitos funcionais e não funcionais. Posteriormente veem-se introduzidos os casos de uso que constituem as funcionalidades base do produto, diagramas arquiteturais e aplicacionais, e ainda um

subcapítulo de avaliação e escolha das tecnologias que irão envolver a fase de desenvolvimento.

Em consequência, é possível encontrar o capítulo de desenvolvimento e implementação da aplicação. Neste capítulo é minuciosamente relatado todo o processo de desenvolvimento aplicacional, abordando os problemas encontrados, mudanças de plano e reformulações estruturais e/ou tecnológicas.

Para finalizar é descrito o capítulo de avaliação da solução, secção essa onde se veem retratados os métodos de avaliação utilizados, bem como os resultados pretendidos e obtidos.

Abstract

This document describes the entire process behind the research, implementation and evaluation of the project HelpIt, a mobile application that aims to help social institutions and organizations in collecting donations, through a simple game of donation attribution, weekly challenges and badge collecting.

At an early stage, the difficulties that charities and organizations are experiencing today in raising support funds are identified. A description is also given of what the application is intended to be, as well as the motivations and opportunities that led to the choice of theme. This concludes the opening chapter with a detailed structure of this document.

In a second chapter, an analysis is made of the current state of the art. The analysis begins with research on some components of human psychology, where topics such as the collector's profile, the basic need for integration and the sharing of achievements and achievements are addressed. Further on is the study of the current donation process and the means of donation provided by some best-known organizations, the Red Cross, the Food Bank and UNICEF. The description of this process is consolidated with a brief wording about the benefits attributed by the tax authorities to potential donors. Then an extensive value analysis is performed, presenting the value proposition of this project, considering the business models Peter Koen, Canvas, Verna Allee, Porter value chain and a practical example of the AHP method.

Once the state-of-the-art portrait is completed, the design chapter appears. This section lists and explains all possible technologies and solutions for final product implementation, describing an analysis of compliance with functional and non-functional requirements. Subsequently, the use cases that constitute the basic functionalities of the product, architectural and application diagrams, and a sub chapter of evaluation and choice of technologies that will involve the development phase are introduced.

As a result, you can find the chapter on application development and implementation. In this chapter, the entire process of application development is covered, addressing the problems encountered, changes of plan and structural and/or technological reformulations.

Finally, the solution evaluation chapter is described, where the chosen evaluation methods, as well as the intended and obtained results are portrayed.

Agradecimentos

Com este projeto, quero agradecer a todos aqueles que me motivaram a continuar e não desistir, àqueles que ajudaram durante a fase criativa do produto, ao professor orientador desta tese pelo apoio e *feedback* prestados, e sobretudo ao grupo de utilizadores responsáveis pelos testes de avaliação da aplicação na sua fase final de desenvolvimento.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

HTML	*Hypertext Markup Language*
CSS	*Cascading Style Sheets*
PWA	*Progressive Web Apps*
API	*Application Programming Interface*
OCD	*Obsessive Compulsive Disorder*
SQL	*Structured Query Language*
NoSQL	*Non-SQL*
AHP	*Analytic Hierarchy Process*
IoT	*Internet of Things*
SSL	*Secure Socket Layer*
JSON	*JavaScript Object Notation*
IPSS	*Instituições Particulares Solidariedade Social*
HTTP	*Hypertext Transfer Protocol*
HTTPS	*Hypertext Transfer Protocol Secure*
URL	*Uniform Resource Locator*
CORS	*Cross-Origin Resource Sharing*
JWT	*JSON Web Token*
APK	*Android Package*
SDK	*Software Development Kit*

1 Introdução

Neste capítulo são identificados o problema que se pretende solucionar com este projeto, as motivações para o seu desenvolvimento e os objetivos planeados a serem atingidos.

Inicialmente é descrita a crescente necessidade do ser humano se expressar e consumir conteúdo através da multimédia e internet, seguindo-se uma explicação para a possível aliança ao fator "apoio social". Por fim veem-se identificados os objetivos que se pretendem alcançar, justificando com algumas das motivações que levaram ao planeamento e implementação deste produto.

1.1 Identificação e contextualização do problema

Numa tentativa de aliar uma atividade lúdica e interativa ao apoio social, surgiu o conceito de uma aplicação móvel baseada em donativos, instituições/associações e pontuações sociais.

Os movimentos de apoio social veem ainda hoje dificuldade em angariar fundos suficientes para a concretização dos seus objetivos, em parte pela falta de conhecimento por parte dos possíveis doadores no que toca à existência destes movimentos, mas sobretudo pela desinformação quanto ao processo de doação.

A base da aplicação consiste num jogo sério onde todas as semanas são escolhidas três organizações alvo. Cada utilizador/jogador poderá escolher as instituições às quais pretende fazer um donativo. O valor doado será da total responsabilidade do jogador.

Por se tratar de uma aplicação com movimentações monetárias, está destinada somente ao público adulto. Aquando do registo de nova conta, é solicitado ao utilizador

a confirmação da idade superior a 18 anos. Apenas com esta validação a conta será criada com sucesso.

Esta aplicação, acompanhada de um aspeto visual estimulante e lúdico, apresentará as organizações da semana, em forma de crachá, com o valor doado até então mais o valor restante para atingir o objetivo semanal. Caso o utilizador tenha contribuído para uma instituição (independentemente do seu valor), receberá aquando do fecho da doação, o crachá virtual respetivo, o qual poderá ser partilhado nos seus perfis das diversas redes sociais. O utilizador irá poder consultar dentro da aplicação uma lista, em forma de caderneta de conquistas, com os crachás ganhos e por desbloquear.

Um exemplo de crachá seria o "Defensor dos Animais" o qual seria ganho quando contribuísse para uma instituição de alguma forma ligada ao mundo animal. Caso o utilizador já tenha obtido um dado crachá, é incrementado um nível, transformando-se assim o crachá anterior em "Defensor dos Animais – Nível 2".

O sistema de pontuação/ranking não está dependente do valor doado pelo jogador, mas sim pela sua entrada ou não nessa mesma doação. O utilizador poderá assim contribuir e subir de nível independentemente das suas possibilidades económicas.

1.2 Objetivos e contributos

Com este projeto pretende-se aumentar o número de apoios sociais a organizações e instituições necessitadas através da aliança entre a tecnologia, a estimulação visual e o crescente impacto das redes sociais e estatuto virtual.

Um exemplo de um projeto distinto, mas com objetivos semelhantes, é o projeto Humble Bundle (Humble Bundle, 2019).

O conceito baseia-se também no aumento dos apoios sociais, mas através da compra de *bundles* de videojogos. Os preços destes pacotes de jogos começam em valores tão

baixos como 1$ e permitem ao cliente decidir que percentagem e para que movimento querem que o seu dinheiro seja distribuído.

Este projeto teve bastante adesão, como podemos de facto confirmar pelos valores apresentados no artigo (Conditt, 2014).

Uma questão que se poderá colocar logo à partida é se a recompensa dentro do jogo é significativa para levar o utilizador a aderir a este sistema de doações. Uma dúvida válida e que poderá ser respondida através de vários exemplos de sucesso.

O colecionamento de itens é um sucesso desde o início dos tempos, não apenas dentro do mundo dos videojogos, mas também nos jogos de cartas, cadernetas de cromos, posters, moedas e muito mais, tema que será abordado mais abaixo.

No projeto em causa, este fator aplica-se aos crachás virtuais que o utilizador tem possibilidade de adquirir ao longo das semanas de jogo. Paralelamente, juntamos ainda a necessidade de integração e a partilha de conquistas. Todos estes fatores elevam a capacidade de adesão e retenção dos jogadores.

1.3 Motivações e oportunidades

Este projeto surgiu do desconhecimento e desconfiança acerca de alguns movimentos sociais, com os quais muitas vezes nos deparamos na rua, em lojas e supermercados. A ideia de podermos ter a certeza de como fazer um donativo de forma simples e rápida, e de concretamente sabermos para onde esse dinheiro vai realmente, foi o que realmente instigou o desenvolvimento deste projeto.

A aplicação consiste num simples jogo de colecionamento de itens, particularmente de crachás representativos de cada movimento. Esta simplicidade permite, contudo, gerar uma excelente retenção dos jogadores, pois estes tentam a todo o custo completar a sua caderneta de crachás e evoluir os já capturados. Quanto maior o número de jogadores, maior o apoio que poderá ser gerado para as instituições sociais.

Este projeto poderá despoletar um grande movimento de solidariedade social, incentivando os utilizadores a doar às organizações com que mais se identificam e possibilitando o lançamento de outros projetos do mesmo âmbito. Tal como o Humble Bundle serviu de inspiração para esta aplicação, muitos outros programadores, gestores de marketing digital e/ou autodidatas se poderão inspirar e criar projetos com objetivos comuns.

1.4 Estrutura do presente documento

Este documento está dividido em sete capítulos, contendo cada um deles múltiplos subcapítulos, separadores de diferentes temas e conceitos:

- Capítulo 1 – Identificação do problema que se pretende resolver, contextualizando com o mundo real. São também referidos os objetivos e contributos que se pretendem atingir com o desenvolvimento do projeto, terminando com uma menção às motivações e oportunidades que despoletaram o conceito escolhido;

- Capítulo 2 – Análise ao estado da arte atual, explicando breve estudo a algumas componentes psicológicas humanas interessantes. É ainda descrito o processo de doação a instituições, terminando com a análise de valor do projeto em causa;

- Capítulo 3 – Descreve-se a arquitetura e grafismos propostos e enunciam-se as soluções possíveis existentes, fazendo uma análise aos diferentes casos de uso e ao seu grau de adaptabilidade com as potenciais tecnologias;

- Capítulo 4 – Capítulo de desenvolvimento da solução. É descrito todo o processo de implementação aplicacional até ao produto final;

- Capítulo 5 – Avaliação da solução, enunciam-se os processos de avaliação utilizados, concluindo com uma apreciação da solução implementada;

- Capítulo 6 – Reflexão sobre o conhecimento adquirido ao longo do projeto, bem como menção às falhas, sucessos e oportunidades futuras;

- Capítulo 7 – Lista completa das referências utilizadas na preparação e redação deste documento.

2 Estado da Arte

Neste capítulo é abordada a componente psicológica responsável pelo gosto do ser humano em colecionar, se integrar em grupos e partilhar os seus feitos e conquistas. Após a contextualização com o mundo real, é detalhado o estudo do funcionamento do ato de doar a instituições e movimentos sociais, falando ainda sobre os benefícios fiscais que daí advêm. Por fim é provada a utilidade deste projeto recorrendo a uma análise de valor.

2.1 Descrição da psicologia envolvente

Para o desenvolvimento deste projeto, não só é importante toda a parte tecnológica como arquitetura da solução, ferramentas e linguagem de programação, persistência de dados e segurança, mas também é essencial o estudo da componente psicológica que irá reter ou não reter os utilizadores no produto.

Analisando um pouco a fração da psicologia humana importante para este caso de estudo, podemos distinguir três fatores interessantes: a síndrome do colecionador, a necessidade básica de integração e a satisfação na partilha de "troféus" pessoais e conquistas.

2.1.1 Perfil de colecionador

O perfil de colecionador é uma componente do ser humano que o leva a recolher e colecionar, todo o tipo de produtos e objetos. Estes itens, físicos ou virtuais, como autocolantes, moedas ou carros em miniatura, transmitem aos seus colecionadores sensações relacionadas sobretudo com paz, nostalgia e tranquilidade (Wikipedia | Psychology of collecting, 2019).

Desde o início dos tempos que este perfil constitui uma necessidade básica do ser humano. A escassez de alimentos, roupa e armamento conduziram o Homem a adquirir a ânsia pela recolha de tudo o que pudesse vir a ser útil num futuro próximo. Este fator sofreu algumas transformações ao longo do tempo, especialmente nos países mais desenvolvidos onde a escassez de alimento e vestuário reduziu quase integralmente. Apesar de extinta a necessidade básica, o ser humano continuou em busca de preencher esta "falta", colecionando agora artigos mais fúteis (sem grande importância para a sobrevivência da espécie) (Jarrett, 2014).

As indústrias foram aproveitando este mercado, lançando cadernetas de autocolantes, coleções de carros miniatura e jogos de cartas (Joschik, 2016) (Animation Collectibles Market 2019 Industry Size, Trends Evaluation, Global Growth, Recent Developments and Latest Technology, Future Forecast Research Report 2025, 2019). Mais tarde o mercado atingiu o patamar tecnológico com a coleta de itens virtuais. Várias empresas de videojogos, por exemplo a Steam, lançaram objetos colecionáveis dentro dos seus jogos que poderiam ser trocados ou comprados/vendidos entre utilizadores (S, 2018). Apesar de virtuais estes itens têm muitas vezes valor monetário. Outros produtos disponibilizaram o mesmo sistema onde o valor era não económico. O interesse do utilizador era apenas completar a sua coleção para deleite pessoal, como é o caso do mais recente sucesso Pokemon GO.

Para além do fator sobrevivência, outro forte motivo que conduz o ser humano a colecionar itens, é o controlo da ansiedade e a sensação nostálgica que lhes transmite. O ato de adquirir objetos e agrupá-los em categorias induz o colecionador a um estado de calma e tranquilidade momentâneas. Da mesma forma, revisitar coleções despoleta nostalgia nos colecionadores, retornando-lhes memórias e sensações (M.Farouk Radwan, Why do people collect things?, 2017).

Em casos mais extremos, esta ação pode tornar-se uma dependência psicológica, denominada de *OCD (Obsessive Compulsive Disorder)*. Esta doença do foro psicológico pode estar associada a diversos tipos de obsessões, como arrumação excessiva, repetição de ações como forma de ritual e aquisição compulsiva de objetos (When

Collecting Things Becomes a Problem, 2013). Este grupo é aqui mencionado, no entanto não será utilizado como estratégia de retenção de jogadores por motivos sociais e morais.

2.1.2 Necessidade básica de integração

Paralelamente podemos identificar uma busca infindável por integração e aceitação social. A sensação de pertença ajuda na elevação da autoestima e bem-estar, o que leva de facto o Homem a perseguir incessantemente este sentimento em todas as oportunidades. De um pequeno grupo de amigos a uma grande organização, como um partido político, todos eles têm a habilidade de conferir esta necessidade aos seus membros (The Reasons Of People Joining Groups, 2013).

Existem várias razões pelas quais o ser humano procura esta integração. O Homem é um ser social, como tal, faz parte da sua natureza fugir à solidão procurando grupos de amigos. Ao mesmo tempo, pertencer a um novo grupo poderá simbolizar uma tentativa de afirmação e identificação social. São exemplo grupos de fãs, escuteiros, clubes de futebol e partidos políticos.

Outro motivo é a sensação de pertença a um propósito maior. Um dos maiores dilemas da vida humana é a procura pelo sentido da própria vida. Encontrar um conjunto de pessoas com as mesmas ideologias e valores, facilita bastante este processo. Por fim, outra razão é a possibilidade de conhecer pessoas novas, angariar contactos e revelar oportunidades de crescimento profissional (M.Farouk Radwan, Why do people join groups?, 2017).

Neste projeto em particular, o jogador poderá pertencer a um movimento com um objetivo maior e fazer parte de um grupo de jogadores com os quais poderá partilhar as suas conquistas.

2.1.3 Partilha de conquistas

Juntamente com as componentes psicológicas mencionadas acima, identificamos o gosto pela partilha de feitos e conquistas. Antigamente feita através de conversas e do "passa a palavra", e mais recentemente através das redes sociais. O ser humano, na sua generalidade, sente um orgulho ainda maior, não pela sua conquista, mas pela partilha do seu sucesso com o outro (Canfield, 2019). Seja ele amigo, família, conhecido ou desconhecido, o Homem sente uma forte sensação de concretização ao partilhar com o mundo os seus mais recentes feitos (Why Sharing Your Progress Makes You More Likely To Accomplish Your Goals, 2015). Uma simples passagem pelo *feed* de qualquer rede social comprova este argumento (Sutradhar, 2017). A empresa Facebook soube identificar na perfeição esta característica humana, implementando o sistema de partilha de *posts* que todos nós podemos ver ao aceder à plataforma.

Em épocas mais primitivas, esta partilha era feita sobretudo baseada em conquistas de território e vitórias de guerra. Alguns líderes mais extremistas, como é o caso de "Vlad, o Terrível", usaram este mecanismo como meio de transmissão de medo e pânico ao inimigo, fazendo dos mortos em batalha exemplo para que todos pudessem ver (A história do verdadeiro Drácula: Vlad, o Empalador, s.d.).

Na atualidade, tendo em conta a evolução natural do ser humano, este instinto atingiu um patamar mais superficial, onde a partilha está somente relacionada com orgulho pessoal, transmissão de informação e/ou simples ato de ostentação (Rogers, 2018).

2.1.4 Contextualização

Mas de facto, de que forma está tudo isto relacionado com o projeto? Esta componente psicológica é sem dúvida essencial para desenhar uma aplicação/jogo capaz de atrair e reter os jogadores. É importante para delinear as funcionalidades mais desejadas pelo utilizador, um aspeto gráfico chamativo e os mecanismos principais que sustentam este tipo de jogo de colecionação.

Compreender a mente humana é um ato complexo e demorado, mas quando realizado com sucesso, traz inúmeras vantagens a este tipo de projetos. Uma vez descritos alguns dos comportamentos mais importantes do Homem, é possível elaborar um plano de ação com todas as funcionalidades que o jogador poderá vir a querer.

Parte fundamental do sucesso de uma aplicação ou videojogo é a retenção do jogador. Apesar de fortemente importante uma boa adesão inicial ao produto, se as dinâmicas e mecânicas não conseguirem reter a atenção e interesse do jogador, o projeto rapidamente se torna num fracasso. Por este motivo, é cada vez mais forte a necessidade de estudar a fundo o consumidor final antes do planeamento e execução do projeto.

2.2 Processo de doação

O conhecimento do processo de doação a instituições e associações sociais não está ao alcance de todos. Muitas destas organizações não geram um maior número de apoios, maioritariamente por três motivos, falta de interessados ou incapacitados em doar, desconhecimento dos meios disponíveis e desconfiança quanto à veracidade da organização em causa. Segundo o artigo disponibilizado no site da DECO Proteste (Donativos em dinheiro para instituições reconhecidas, 2019), é possível ao doador confirmar a autenticidade do possível recebedor através de uma pesquisa na lista oficial de instituições particulares de solidariedade social – IPSS (Seg Social - Instituições Particulares de Solidariedade Social, 2019).

Uma vez validada a instituição alvo, o futuro doador poderá realizar o seu donativo de diferentes formas, dependendo dos meios disponibilizados pela organização. Entre as vias mais comuns estão a transferência bancária, transferência por PayPal, campanha de angariação de fundos no Facebook, multibanco, correio e Payshop.

Como análise concreta à redação do presente documento, foram estudados os meios de apenas três organizações, a Cruz Vermelha, o Banco Alimentar e a Unicef. Esta escolha foi concretizada baseando-se apenas no conhecimento e motivações pessoais.

Começando pela Cruz Vermelha, existem várias formas de doar dinheiro. Esta organização permite donativos através do site oficial, transferência por PayPal, campanha de angariação de fundos no Facebook (Como posso fazer donativos a uma angariação de fundos ou organização de beneficência no Facebook?, 2019), transferência bancária, multibanco ou *netbanking*, por correio através de cheque, pagamento Payshop, por telefone, pela campanha "Ser Solidário" nos multibancos, por consignação de 0.5% do IRS e ainda por herança ou legado (Cruz Vemelha | Formas de Doar Dinheiro, 2019). Toda a informação detalhada, para cada um dos métodos, poderá ser encontrada no website oficial da organização.

Abordando a segunda organização estudada, o Banco Alimentar apenas fornece informação para a realização de donativos através do site oficial. Para fazer um donativo a esta instituição, o doador deverá preencher um formulário de registo onde indica os seus dados pessoais, o montante desejado e a sede de destino (Abrantes, Porto, Aveiro, etc.), sendo contactado posteriormente para confirmar a ação e receber os dados de pagamento (Banco Alimentar | Quero Fazer Um Donativo, 2019).

A terceira e última organização, a Unicef, disponibiliza novamente um número de meios superior ao Banco Alimentar, no entanto, com menos algumas alternativas em comparação com a Cruz Vermelha. A Unicef tem um papel fundamental na vida e futuro das crianças, colocando o seu foco na vacinação, cuidados de saúde, educação, nutrição, proteção contra abusos, violência e guerra. Esta organização permite assim a realização de donativos por transferência bancária, por pagamento ou movimento "Ser Solidário" nas diversas caixas de multibanco, por cheque ou vale via correio, por MBWay e ainda através da linha telefónica de valor acrescentado, onde 0.50€ revertem para os fundos de apoio da instituição (Unicef | Outras formas de fazer o seu donativo, 2019).

2.2.1 Benefícios fiscais

Mais uma vez, segundo o artigo publicado pela DECO Proteste em Fevereiro de 2019, todos aqueles que efetuarem donativos a organizações fidedignas, registadas na lista de consignação fiscal, como a UNICEF Portugal e a Amnistia Internacional, terão acesso a alguns benefícios fiscais. O Fisco concede benefícios fiscais a quem apoia entidades públicas ou privadas em áreas como ciência, desporto, educação e cultura. Nos casos de donativos a instituições internacionais sem número fiscal português e donativos sem recibo/comprovativo de pagamento, não verão aplicados estes benefícios.

De forma a receber o apoio fiscal, o montante doado, até Dezembro do ano anterior, deverá estar devidamente registado na declaração de IRS, acompanhado do(s) recibo(s) que comprovam o movimento. O valor concreto do benefício estará dependente do escalão de rendimentos do contribuinte (Donativos em dinheiro para instituições reconhecidas, 2019).

2.3 Análise de valor

Em consequência da análise realizada neste projeto, foi possível levantar vários pontos vantajosos e de valor. O público alvo positivamente afetado por este produto está dividido em dois grupos distintos, jogadores e associações/instituições.

Os jogadores terão a possibilidade de se juntar a um movimento de ajuda social, no qual, enquanto jogam, estão a contribuir para o aumento de apoios financeiros às instituições e associações sociais pelas quais têm mais apreço e ligação. Ao mesmo tempo, é-lhes oferecida uma experiência de jogo agradável e atrativa, e o mais importante, um sentido de pertença e integração.

Paralelamente temos as instituições e associações sociais que mais favorecem com este projeto. Os objetivos traçados para cada desafio, quando atingidos, aumentarão numa pequena parte os fundos financeiros de cada organização.

2.3.1 Modelo de Peter Koen

De acordo com o modelo de Peter Koen, a fase inicial de um projeto de negócio e inovação com este deverá ser dividido em 5 etapas distintas: criação e enriquecimento da ideia, análise das oportunidades, identificação das oportunidades, seleção da ideia e definição de conceito (Martikainen, 2017). Estas cinco etapas poderão ser abordadas múltiplas vezes em diferentes alturas da fase inicial do projeto, de forma a assim construir um produto consolidado. A criação da ideia inicial surgiu com a vontade de implementar um produto que pudesse de alguma forma beneficiar a sociedade. Foram analisadas oportunidades e identificou-se a já referida necessidade do ser humano na partilha, conquista e ajuda. Posteriormente, e por consequência, surgiu o conceito deste jogo/aplicação móvel, que juntou o modernismo tecnológico à popularidade e apoios sociais.

2.3.2 Modelo de negócio Canvas

Após identificação concreta do conceito pretendido, foi importante desenvolver um modelo de negócio Canvas. Neste modelo podemos encontrar informação sobre as parcerias, recursos e atividades chave, proposta de valor, segmentação de mercado e canais de distribuição (O que é o Business Model Canvas, 2016).

O primeiro passo para iniciar a construção de um modelo Canvas é a identificação dos diferentes segmentos de mercado. Neste projeto teremos apenas dois segmentos distintos, os jogadores e as instituições/organizações.

O passo seguinte é a definição da proposta de valor, o que temos para oferecer aos diferentes segmentos de mercado. Este projeto, tal como já mencionado algumas vezes anteriormente, irá oferecer aos jogadores uma aplicação interativa e lúdica que tem como objetivo evoluir e completar uma caderneta de crachás, ao mesmo tempo que estarão a ajudar num movimento solidário destinado ao apoio do segundo segmento de mercado, as instituições.

Em seguida é essencial enumerar os diferentes canais de distribuição, os meios utilizados pela organização do projeto para fazer chegar a proposta de valor ao cliente final. Como se trata de uma aplicação móvel, os principais meios de distribuição serão as lojas de aplicações móveis, como a PlayStore (Android) e a AppStore (iOS). Paralelamente serão ainda usadas as redes sociais no plano de marketing para divulgação da aplicação. Outros meios publicitários interessantes na atualidade são o Youtube, anúncios pagos na Google e o Spotify.

Uma vez tendo a ligação entre proposta de valor e cliente é importante fortalecer a relação através de estratégias de retenção. Um dos pontos fortes será o vasto entendimento do funcionamento das redes sociais e proximidade etária com grande parte dos futuros jogadores. É essencial compreender o cliente, saber o que pensam e como pensam, daí o estudo psicológico descrito no capítulo 2.1. Analisando o modelo de Verna Allee, compreendemos que as "pessoas" deverão estar no centro da ação, sem elas um projeto não se poderá iniciar nem manter (Verna Allee describes Value Networks, 2009). Este é sem dúvida o principal fator que move este produto, pois tudo será desenvolvido em volta de movimentos sociais, utilizadores e interajuda. Todos os papeis são importantes numa história, uns mais do que outros em fases distintas, mas todos contribuem para a construção de algo consolidado e coeso.

Como se trata de um projeto de ação social, este produto não terá o seu foco nas fontes de rendimento. No entanto, quando escalado, um produto enfrenta sempre custos de manutenção. Para este cenário, e apenas para tornar a aplicação num projeto autossustentável, podemos identificar algumas fontes interessantes. Em primeiro lugar, um método não invasivo para o jogador, será a criação de um movimento online para apoio da manutenção da aplicação, são exemplos sites como o Patreon e o Kickstarter. Nestes websites, os criadores de conteúdo e/ou desenvolvedores de produtos têm a capacidade de pedir donativos aos seus utilizadores, de forma a lhes ser possível iniciar ou continuar o desenvolvimento dos seus projetos. Um segundo método, mais tradicional e um pouco mais invasivo, será o uso de anúncios dentro do jogo. Este método é por vezes mal-aceite por alguns utilizadores.

De forma a manter este produto em funcionamento serão necessários recursos. O principal recurso será de facto a infraestrutura da aplicação, base de dados e API. Existem muitas soluções disponíveis a custo zero para os tempos primordiais de um projeto. No entanto, o natural crescimento traz em consequência custos adicionais. Outro recurso serão os programadores, relações públicas e/ou responsáveis de marketing e especialistas legais (ex. advogado).

As atividades chave, como podemos facilmente concluir, estarão em torno da manutenção do serviço, divulgação e retenção e utilizadores e ainda contacto constante com as diferentes organizações e movimentos sociais.

As parcerias principais envolvidas neste projeto serão as instituições e organizações alvo de donativos, as quais irão ajudar na divulgação da aplicação.

Finalmente chegamos ao último patamar deste modelo, a estrutura de custos. Mais uma vez, grande parte dos custos estarão relacionados com a manutenção da infraestrutura do serviço, mas serão encontrados muitos outros ao longo do tempo de vida do projeto. Custos como divulgações pagas nas redes sociais, através de anúncios no Facebook e/ou influenciadores (perfis sociais com grande atração que se disponibilizam a divulgar um produto).

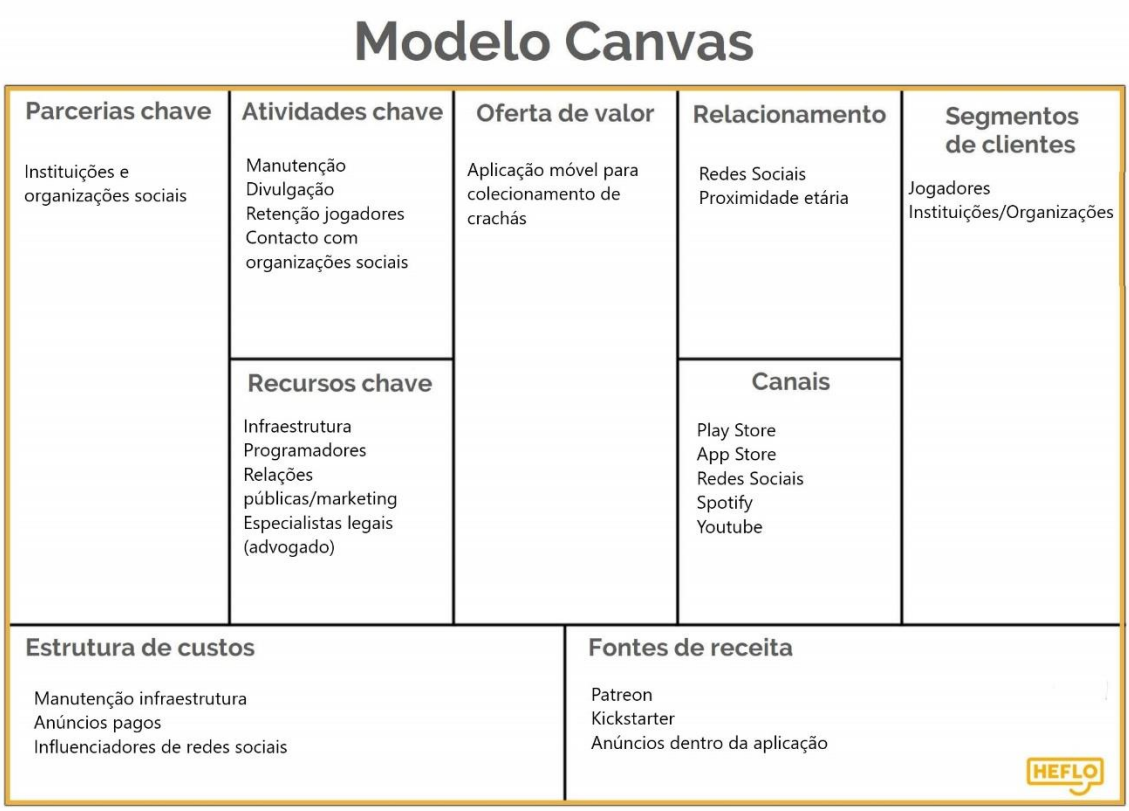


Figura 1 - Modelo de negócio Canvas

2.3.3 Modelo de Porter

Pegando agora como exemplo o modelo da cadeia de valor de Porter, é possível agrupar recursos e atividades em pequenos setores de ação, atividades primárias e atividades de suporte (Porter's Value Chain - Understanding How Value Is Created Within Organizations, 2016).

As atividades principais deste serviço estão divididas em logística de importação ou receção, operações, logística de exportação ou distribuição, marketing e vendas e manutenção do serviço. Todas estas seções deverão ser ainda subdivididas em subsecções de atividades diretas (são exemplo atividades como partilha de um post publicitário nos grupos de Facebook da faculdade ISEP), atividades indiretas (em seguimento do exemplo anterior, analisar os comentários à publicação e responder aos

diversos utilizadores) e ainda *quality assurance*, ou garantia de qualidade, que garantem o cumprimento dos padrões e ideais definidos.

As atividades de suporte são seccionadas em processo de aquisição (lista de ações que a organização executa de forma a adquirir os recursos necessários para o desenvolvimento e manutenção do produto), gestão de recursos humanos, desenvolvimento tecnológico e gestão da infraestrutura. Estas subactividades deverão também ser subdivididas em atividades diretas, indiretas e garantia de qualidade.

2.3.4 Método de análise AHP

De forma a concluir esta análise de valor iremos abordar superficialmente o método de análise *Analytic Hierarchy Process* (AHP). AHP é uma ferramenta de tomada de decisões introduzida em 1980 por Thomas Saaty. Este processo tem como objetivo, ajudar a organização de um projeto na tomada de decisões complexas, tendo por base um sistema de pesos e pontuações. Uma vez definidos os requisitos essenciais e as opções disponíveis para o problema em questão, podemos atribuir pesos ou graus de importância. Todos os requisitos deverão ser analisados contra cada uma das opções identificadas, gerando uma pontuação. Somando a pontuação de cada opção nos diferentes requisitos, multiplicada pelos pesos, obtemos em teoria a opção com maior pontuação e consequencialmente o caminho mais favorável a seguir (The Analytic Hierarchy Process).

Para efeitos de exemplo iremos testar esta ferramenta para decidir qual o meio publicitário mais eficaz para a nossa aplicação. Entre os meios publicitários já referidos, encontramos o Facebook/Instagram, o sistema de anúncios da Google, o Youtube e o Spotify. Como se trata de um projeto em fase embrionária, será dado um maior grau de importância ao fator monetário, como tal, distinguimos no topo das opções o Youtube e Facebook/Instagram pela gratuidade do serviço. Como segundo fator ou requisito marcamos o raio de ação ou afetação como o mais importante. Entre Facebook/Instagram e Youtube é-nos difícil chegar a uma conclusão, pelo facto de obterem uma pontuação similar, daí a necessidade de introdução de um novo fator. O

terceiro e último fator é o tempo necessário até que a promoção/publicação atinja o público alvo. Neste requisito, damos a maior pontuação ao Facebook/Instagram pela facilidade na partilha e divulgação da informação. O Youtube requer neste campo um período de gestação maior até atingir o mesmo número de alvos. Desta forma podemos teoricamente prever que o uso da plataforma Facebook/Instagram trará o melhor dos resultados tendo em conta as necessidades do projeto.

3 Design da solução

Este capítulo apresenta o processo de design da aplicação. É inicialmente realizada uma análise das possíveis soluções, fazendo uma passagem pelos diferentes casos de uso, que constituem as funcionalidades base do produto, uma descrição da arquitetura aplicacional através da apresentação de diagramas de modelo e componentes. Por fim são enunciadas as tecnologias escolhidas acompanhadas pelos principais motivos da seleção.

3.1 Soluções possíveis

Para este projeto iremos criar uma aplicação mobile, compatível com as plataformas Android e iOS, que deverá permitir ao utilizador ver o seu ranking, jogar nos desafios semanais, ver coleção de conquistas e partilhá-las nas redes sociais.

Uma vez que muitos destes dados necessitarão de ser persistidos, será imperativo definir uma estrutura de base de dados, bem como criar uma API que será usada pela aplicação para aceder aos dados.

3.1.1 Tecnologias para criação de aplicações mobile

Para implementar esta solução, poderíamos optar por diferentes *frameworks* que nos facilitariam o processo de compatibilidade multiplataforma. Entre elas encontram-se o Unity, Xamarin, Ionic, Power Apps e PWA.

3.1.1.1 Unity

A ferramenta de desenvolvimento de videojogos Unity é uma *framework* poderosa capaz de exportar rapidamente os projetos nela implementados para todo o tipo de plataformas, desde PC e consolas a dispositivos móveis (Unity | Home Page, 2019).

Na sua base, o Unity é um motor de jogo complexo desenhado para desenvolvedores de jogos 2D e 3D, amadores ou profissionais da área. Oferece um vasto conjunto de *assets* pré-definidos para facilitar parte do trabalho, capacidade de o utilizador implementar os seus próprios elementos e *scripts*, e ainda uma loja virtual oficial, composta por uma grande comunidade, que disponibiliza todo o tipo de recursos (gráficos, lógica, sonoros) pagos e gratuitos (exemplo da interface gráfica abaixo).

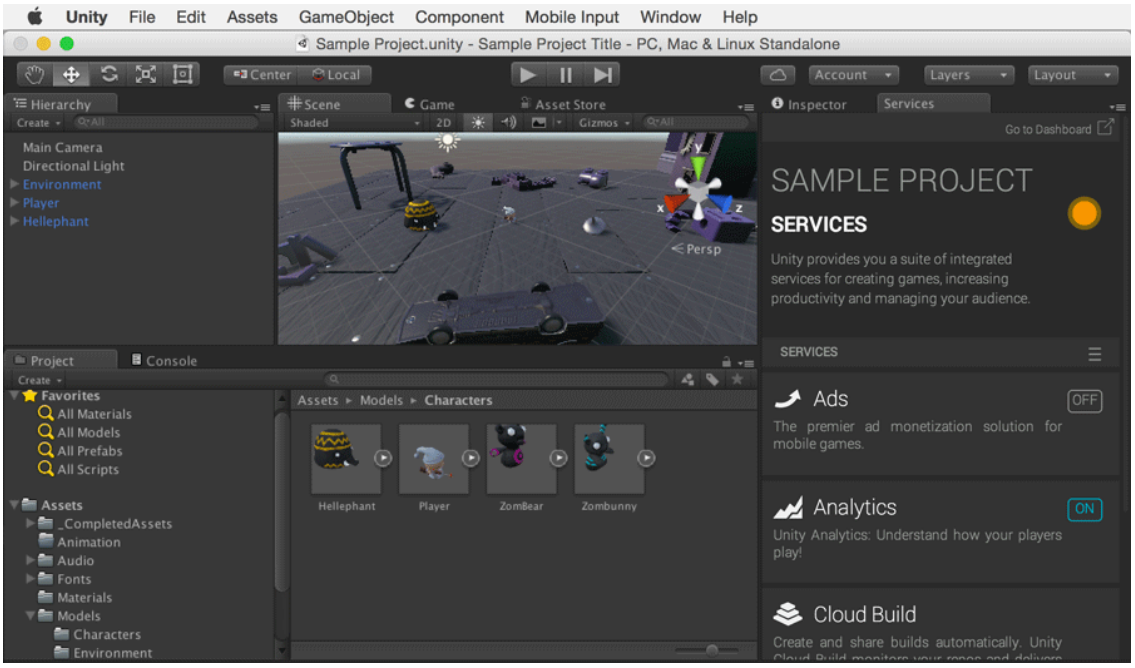


Figura 2 - Interface gráfica do editor Unity

3.1.1.2 Xamarin

Xamarin é um conjunto de produtos, comprados pela empresa Microsoft, que possibilitam o desenvolvimento de aplicações móveis nativas (Xamarin, 2019).

A linguagem de programação utilizada é o C#/.NET. Esta tecnologia permite ao programador implementar toda a sua lógica numa única linguagem que será partilhada para os diferentes sistemas (Android, iOS e Windows) (Quaiato, 2016). Uma aplicação em Xamarin vê a lógica de negócio separada da componente visual, justificado pela portabilidade que torna o sistema especial. Conforme é possível ver na imagem abaixo,

a arquitetura isola o *core* como código partilhado, que será consumido pelas diferentes interfaces gráficas.

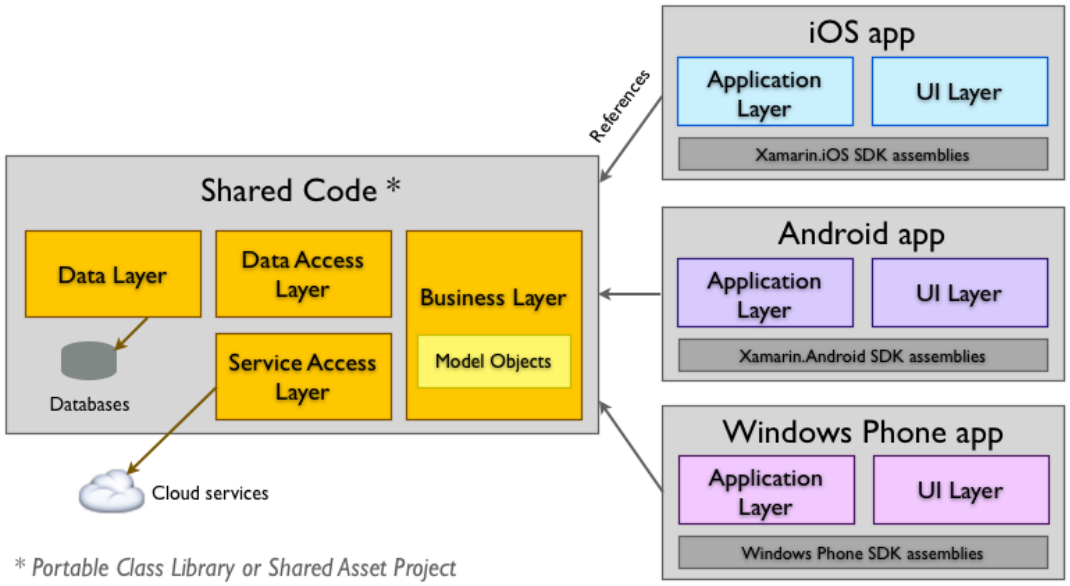


Figura 3 - Arquitetura de uma aplicação Xamarin

Com Xamarin a poupança de tempo no desenvolvimento de aplicações multiplataforma é substancial, pois numa só implementação conseguimos criar código nativo para todos os sistemas.

3.1.1.3 Ionic

A ferramenta Ionic, à semelhança da anteriormente mencionada Xamarin, é também uma poderosa framework de desenvolvimento de aplicações móveis multiplataforma.

Com esta framework, as aplicações são implementadas através do uso de HTML, CSS e Javascript. A grande diferença entre Xamarin e Ionic está na portabilidade das aplicações. Em Xamarin, como já referido, são geradas aplicações nativas nos dispositivos, enquanto que com Ionic, é criada uma aplicação web que é depois compilada para os diferentes sistemas (Documentation - Intro | What is Ionic Framework?, 2019).

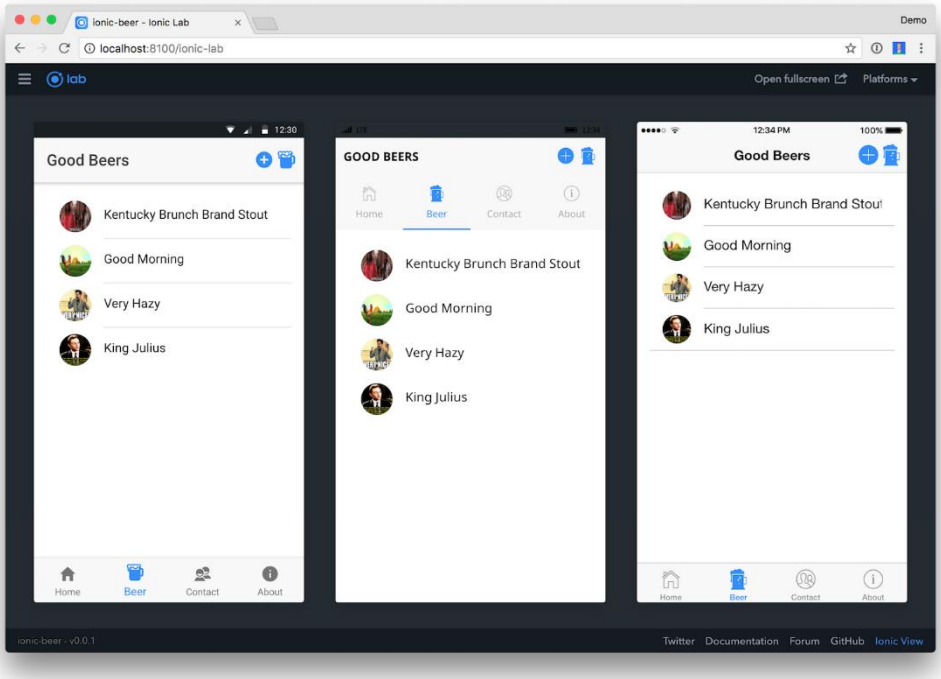


Figura 4 - Exemplo modo de teste de uma aplicação Ionic nos três tipos de plataformas (Android, iOS e Windows)

Apesar de não ser possível desenvolver aplicações nativas com Ionic, esta ferramenta disponibiliza inúmeros *plugins* que servirão de ponte entre a aplicação web e o dispositivo nativo (Ionic Framework | Documentation - Native, 2019).

Por exemplo, é possível utilizar o *plugin* leitor de códigos de barra para poder aceder à câmara do telemóvel ou tablet, ler a informação de um código e devolver um resultado para a aplicação processar.

3.1.1.4 Power Apps

Power Apps é um serviço disponibilizado pela Microsoft para a criação de aplicações móveis, especialmente de âmbito empresarial. Dispõe de um editor para desenhar a interface gráfica das aplicações, bem como de um vasto conjunto de conectores a diferentes serviços, como Sharepoint, Onedrive, Excel e Slack (Microsoft | Build PowerApps, 2019).

Uma das grandes vantagens na utilização deste serviço é a rápida integração com todo o tipo de produtos Microsoft. Os acessos a base de dados em Azure, dados de CRM no Sharepoint, autenticação com o Office 365, entre outros.

Um ponto negativo comparativamente às outras soluções, sobretudo por experiência a nível pessoal, é a limitação a nível gráfico. Apesar do editor de interfaces disponibilizado, nem sempre o "aspeto" visual pretendido é atingível. No entanto, para aplicações empresariais que possam prescindir de uma interface gráfica estonteante, é uma excelente opção.

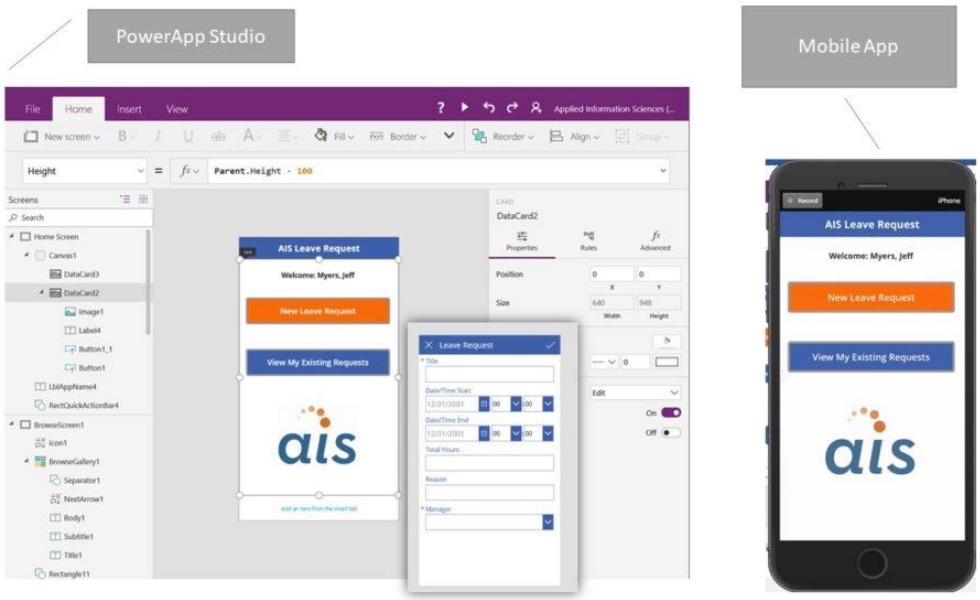


Figura 5 - Editor de interfaces gráficas PowerApps

3.1.1.5 Progressive Web Apps (PWA)

Para concluir a identificação das diferentes ferramentas de desenvolvimento móvel falamos de PWAs ou *Progressive Web Apps*. As PWAs são uma solução desenvolvida pela Google para responder aos problemas da implementação de aplicações multiplataforma. Este método, contrariamente a todas as outras *frameworks*, permite ao programador desenhar e implementar uma só aplicação que poderá ser vista em modo *desktop* e móvel, totalmente responsivo e compatível com todos os dispositivos.

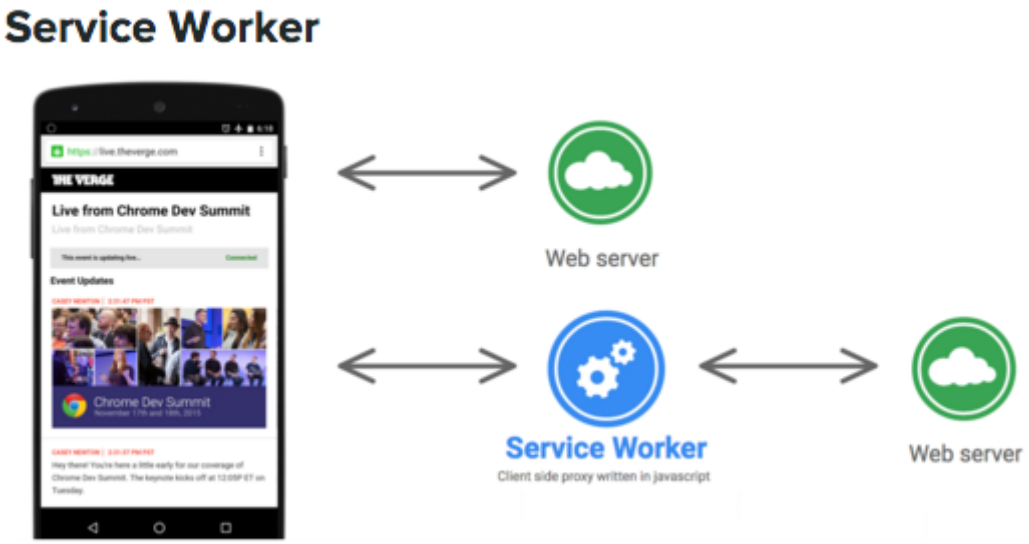


Figura 6 - Diagrama de arquitetura de uma aplicação PWA

Uma *Progressive Web App* é uma aplicação Web, construída com HTML, CSS e Javascript, que após compilada é lida pelo *browser* pré-definido do dispositivo como se fosse uma aplicação nativa. O processo por detrás desta "magia" está relacionado com o sistema de cache de páginas e dados que permitem à aplicação carregar o seu conteúdo, a partir do *browser*, independentemente do estado de conexão da internet.

O *service worker* da aplicação permite este armazenamento em cache e simulação de uma página web como aplicação nativa (LePage, 2019).

Possivelmente pela sua prematuridade, as PWAs ainda levantam algumas limitações quanto à comunicação entre a aplicação e comandos nativos dos dispositivos, tais como acessos à câmara, dados de contactos, notificações ou localização, mas que estão já a aguardar resolução.

3.1.2 Tecnologias para persistência e leitura de dados

De forma a construir toda uma estrutura *backend* apropriada aos requisitos deste projeto, poderiam ser abordadas diferentes estratégias. Uma delas, tendo em vista a necessidade de criação de uma base de dados, autenticação de utilizadores e API para

manipulação de dados, é o uso da ferramenta de manutenção de infraestruturas Firebase.

Firebase é uma ferramenta desenvolvida pela empresa Google e que tem como objetivo minimizar as "dores de cabeça" da implementação e gestão de infraestruturas em aplicações web e móveis. Disponibiliza um conjunto de serviços como autenticação, *push notifications*, base de dados e *hosting*, tudo numa só ferramenta (Firebase Google - Home, 2019). Muitos projetos aderiram a esta tendência pela sua simplicidade nos processos complexos. Apesar de bastante interessante, este serviço tem alguns contras. Quando se tratam de projetos de maior escala e com um grau de complexidade maior, sobretudo a nível de base de dados, esta ferramenta poderá não ser tão ágil e flexível como pretendido.

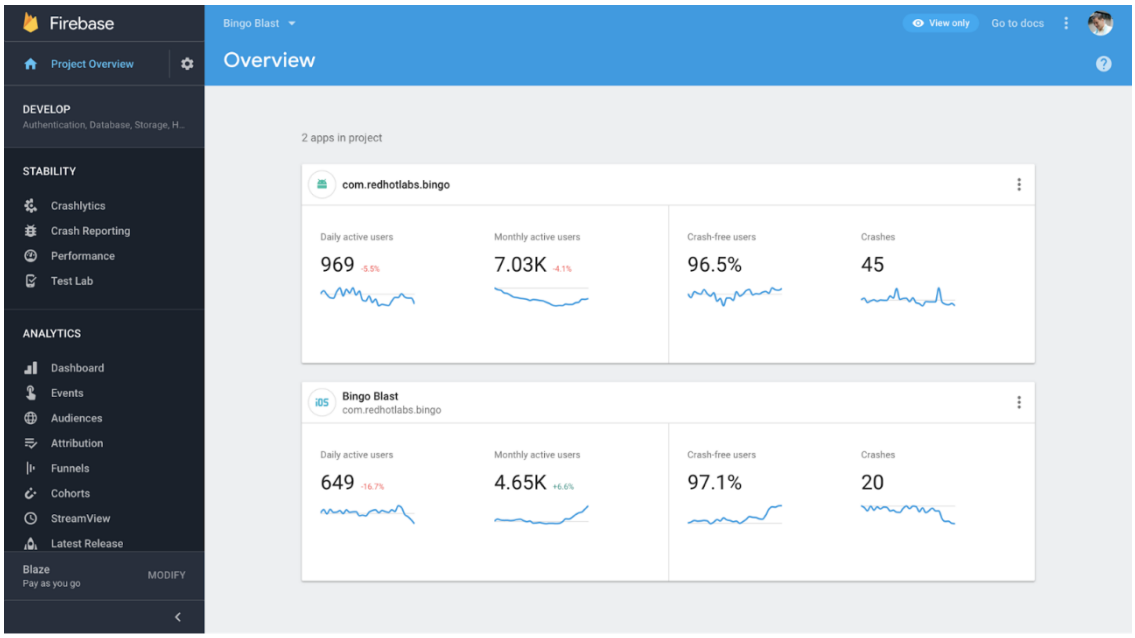


Figura 7 - Interface gráfica da ferramenta Firebase

Em alternativa a esta ferramenta, poderá ser implementada toda a estrutura de raiz, criando assim uma base mais sólida e de maior escalabilidade.

Para criar uma estrutura para este projeto seria então necessária uma base de dados e uma API para responder a pedidos da aplicação. A base de dados poderia ser desenhada

em várias linguagens, no entanto, a mais simples e barata na gestão e faturação seria uma base de dados não-relacional, NoSQL.

3.1.2.1 Base de dados NoSQL

Uma base de dados NoSQL é uma base de dados não-relacional, ou seja, onde não existem relações entre as diferentes tabelas, contrariamente às bases de dados relacionais SQL. Este formato não-relacional permite o uso de simples documentos de texto (JSON ou outros) como tabelas, o que possibilita inserções, edições e consultas de dados mais rápidas e eficientes (Dev Media | Introdução aos bancos de dados NoSQL, 2012). Num modelo de negócio mais complexo, com necessidade de relações entre entidades, as bases de dados NoSQL continuam a ter o seu lugar, pois é-nos possível inserir num dado documento, ou tabela, um *unique identifier* que servirá de ponte entre documentos. Um dos pontos negativos em comparação com bases de dados relacionais é a consistência e coerência dos dados. Como não existem relações entre tabelas ou documentos, a inserção, edição e remoção de linhas de um documento poderá implicar incoerências de dados noutro documento. Este problema é facilmente evitado através de uma boa arquitetura e desenvolvimentos aplicacionais. Através da figura abaixo podemos verificar que de facto as bases de dados não-relacionais sofrem no que toca à confiabilidade e consistência de dados, no entanto, veem o seu ponto forte no desempenho, disponibilidade e escalabilidade. Este tipo de base de dados é ainda a solução ideal para o armazenamento de grandes volumes de dados.

Feature	NoSQL Databases	Relational Databases
Performance	High	Low
Reliability	Poor	Good
Availability	Good	Good
Consistency	Poor	Good
Data Storage	Optimized for huge data	Medium sized to large
Scalability	High	High (but more expensive)

Figura 8 - Tabela de comparação entre bases de dados relacionais e NoSQL (NoSQL Databases: The Definitive Guide, 2017)

Uma ferramenta excelente para este tipo de base de dados, é o MongoDB, a qual oferece um serviço de *hosting* em *cloud*, gratuito até aos 512mb de armazenamento (MongoDB - Cloud Atlas Pricing, 2019).

Acerca da implementação da API responsável por servir de ponte entre base de dados e aplicação, poderão ser também utilizadas diversas tecnologias, entre as mais comuns, NodeJS e mais recentemente ASP.NET Core.

3.1.2.2 Tecnologia NodeJS

NodeJS é um ambiente virtual *open source* capaz de correr código Javascript fora dos browsers. Resumindo, com esta tecnologia temos a capacidade de implementar um servidor Javascript que corre localmente nas nossas máquinas e é capaz de responder a pedidos aplicacionais, ou seja, uma API local. O grande interesse no NodeJS advém da vasta biblioteca de extensões e módulos disponibilizados pela comunidade *open source*, que podemos juntar ao nosso projeto sem qualquer dificuldade acrescida, facilitando muito o processo de implementação (What exactly is Node.js?, 2018).

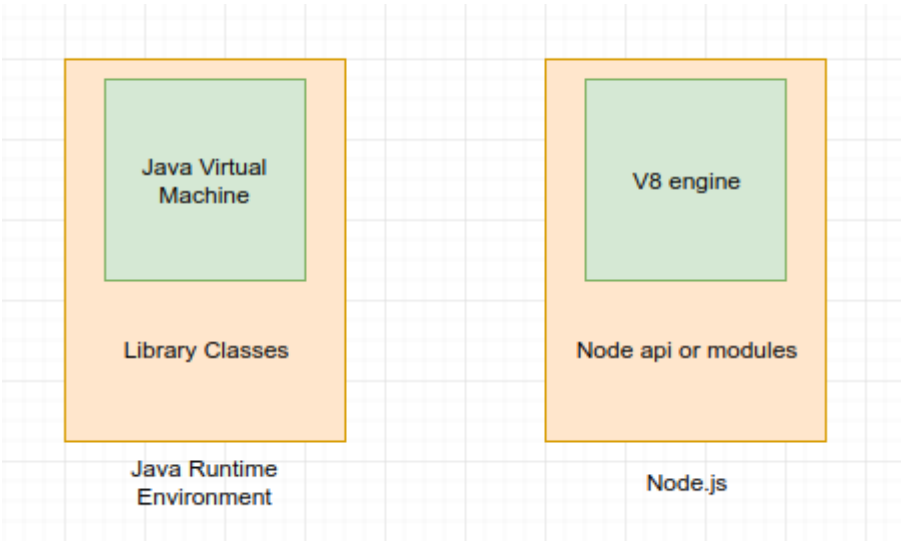


Figura 9 - Diagrama de analogia entre ambiente virtual Java e NodeJS (What exactly is Node.js?, 2018)

3.1.2.3 Tecnologia ASP.NET Core

Paralelamente encontramos o ASP.NET Core como solução alternativa. ASP.NET Core é uma *framework*, também ela *open source*, desenvolvida pela empresa Microsoft como atualização ao antigo ASP.NET. Esta ferramenta, um pouco mais robusta que NodeJS, permite criar aplicações e serviços web, aplicações baseadas no conceito *Internet of Things* (IoT) e *backends* aplicacionais (Roth, Anderson, & Luttin, 2019). A linguagem de programação utilizada para a componente do lado servidor é C#, enquanto que do lado do cliente é utilizo Razor, uma "mutação" do tradicional HTML que permite a inserção de tags C# para gerar conteúdo dinâmico nas páginas *web*.

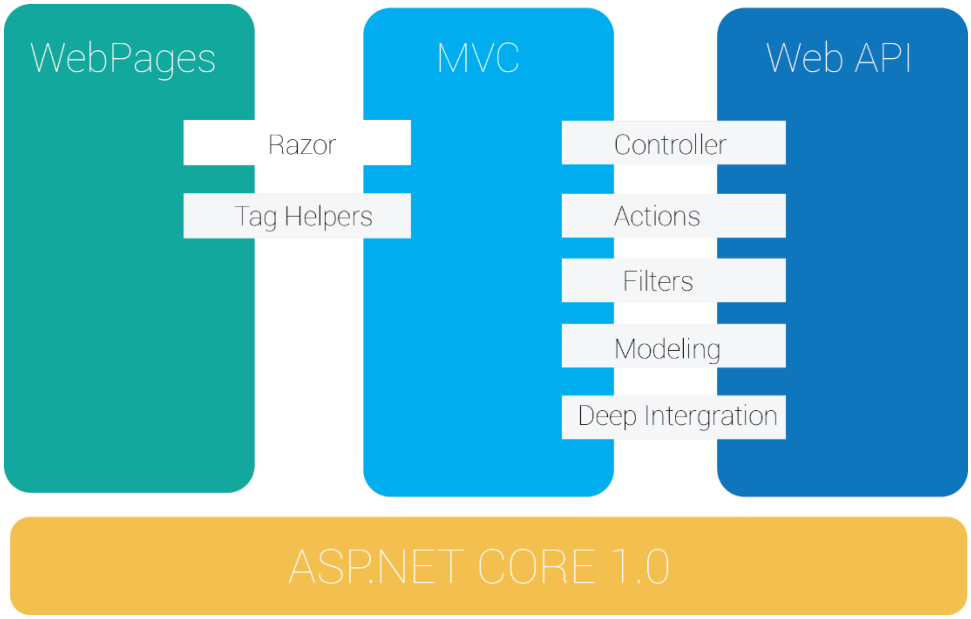


Figura 10 - Diagrama de arquitetura ASP.NET Core

Ambas as tecnologias têm o seu valor. A escolha de utilização irá cair maioritariamente na preferência da linguagem de programação por parte da equipa de arquitetura e implementação.

3.1.2.4 Serviço de hospedagem aplicacional

Uma vez criados os serviços *backend* locais, é altura de os tornar acessíveis a partir de qualquer dispositivo móvel do planeta com acesso à internet. Para tal será necessário um serviço de *hosting cloud* onde este servidor irá permanecer em execução de forma a responder aos seus pedidos. Uma plataforma que disponibiliza este serviço é o Heroku, uma ferramenta *cloud* que suporta diversas linguagens de programação, entre as quais NodeJS e ASP.NET Core. Este serviço oferece um endereço *web* com certificado *Secure Socket Layer* (SSL) que será usado pelas aplicações quando pretendem aceder a informação armazenada na base de dados.

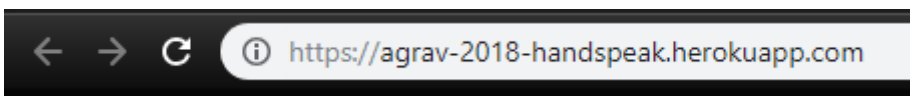


Figura 11 - Exemplo de endereço *web* disponibilizado pelo Heroku

3.2 Análise das abordagens existentes

De acordo com as abordagens identificadas acima, será feita uma avaliação integral tendo em conta o cumprimento ou não cumprimento dos requisitos funcionais e não funcionais.

Requisitos funcionais:

- Login e registo de conta;
- Apresentação das instituições da semana e coleção de crachás;
- Registo de pagamentos/donativos;
- *Push notifications* (alertar o utilizador de objetivos cumpridos);
- Partilha nas redes sociais.

Requisitos não funcionais:

- Portabilidade multiplataforma;
- Aspeto gráfico atrativo;
- Aplicação multi-idioma;
- Facilidade de comunicação com APIs (recolha e inserção de dados).

3.2.1 Requisitos funcionais

Nesta secção será analisado o cumprimento ou não cumprimento dos requisitos funcionais anteriormente definidos para cada uma das soluções identificadas.

3.2.1.1 Login e registo de conta

Para implementar um sistema de login e registo de utilizadores, todas as soluções identificadas são válidas. Para isto, apenas será necessária a interface gráfica e uma API capaz de responder aos pedidos do jogador.

No caso em particular das Power Apps, esta funcionalidade vê-se um pouco mais facilitada do que os restantes pelo facto da já existente integração com autenticação Office 365, no entanto, caso se pretenda implementar autenticação por email ou Facebook, o esforço volta a equiparar-se.

3.2.1.2 Apresentação das instituições da semana e coleção de crachás

Este requisito é também ele respondido positivamente por qualquer uma das opções, no entanto, em algumas delas poderemos encontrar algumas limitações e/ou dificuldades técnicas ao nível gráfico.

A ferramenta já mencionada Power Apps, apesar da sua robustez, é uma ferramenta limitada em implementações gráficas muito exigentes. Não permite ao programador implementar os ecrãs tal e qual conforme desenhados.

No caso do Unity, podemos enfrentar também dificuldades gráficas, desta vez relacionadas com a responsividade dos ecrãs nos diversos dispositivos móveis.

3.2.1.3 Registo de pagamentos/donativos

Este é um dos requisitos 100% independentes da solução a adotar, pois apenas será necessário um serviço de pagamentos que disponibilize uma API para comunicação.

Um exemplo de serviço de pagamentos online é o Stripe. Este serviço permite a criação de uma conta com diferentes métodos de pagamento associados, em seguida apenas será necessária autenticar e registar o pagamento através de uma chamada à API (Stripe - Home, 2019).

3.2.1.4 Push notifications

De forma a completar com eficácia este requisito, poderão ser escolhidas duas abordagens distintas, dependendo da tecnologia escolhida.

Um plano simples que resolveria o problema para qualquer uma das tecnologias seria a utilização do serviço Firebase. Por predefinição, o Firebase disponibiliza a funcionalidade de *push notifications* para qualquer dispositivo.

Em alternativa à utilização de Firebase, teríamos de dar uso a módulos oficiais ou implementados pela comunidade, criados especificamente para cada tecnologia. A *framework* Ionic dispõe de um *plugin* nativo chamado *Push*, com o qual será possível implementar o sistema de notificações dentro da própria aplicação sem qualquer ferramenta externa (Ionic Framework | Native Push Notifications, 2018). No caso de Xamarin e Power Apps, ambos produtos Microsoft, têm acesso aos *hubs* de notificações dentro do Portal Azure.

3.2.1.5 Partilha nas redes sociais

A partilha nas redes sociais é o outro requisito 100% independente da plataforma. De acordo com a rede social pretendida, a partilha de uma publicação não passará por ser mais do que um pedido *POST* a uma API REST disponibilizada pela plataforma. Qualquer uma das tecnologias mencionadas tem capacidade para efetuar pedidos e receber respostas de fontes externas, por esse motivo, o cumprimento do requisito vê-se assegurado.

3.2.2 Requisitos não funcionais

Nesta secção, tal como na anterior, será analisado o cumprimento ou não cumprimento de requisitos, desta vez dos requisitos não funcionais.

3.2.2.1 Portabilidade multiplataforma

A portabilidade entre plataformas é essencial quando falamos de poupança de tempo de implementação e manutenção. Num cenário ideal, uma implementação deveria servir para gerar um produto que corresse em qualquer plataforma. Das tecnologias mencionadas, todas elas permitem esta portabilidade, no entanto, umas mais facilmente do que outras.

Todas as opções tecnológicas têm o seu "calcanhar de Aquiles", neste caso em particular em relação ao processo de exportação multiplataforma. As tecnologias selecionadas permitem a portabilidade aplicacional, contudo, um maior grau de complexidade técnico conduz a mais problemas. Um cenário comum é a incompatibilidade de algumas funcionalidades dependendo da plataforma em questão, por exemplo, uma dada ação funcionar em Android, mas não funcionar, ou funcionar de forma distinta, em iOS.

Apesar dos eventuais atrasos, é possível concluir com sucesso esta exportação, apenas demorará o tempo necessário de aprendizagem da tecnologia.

3.2.2.2 Aspeto gráfico atrativo

Como já mencionado acima, algumas das ferramentas enfrentarão dificuldades na implementação visual dos ecrãs. Tecnologias como Power Apps, que veem limitada a criatividade gráfica, e Unity, que dificulta o trabalho ao desenvolvedor na criação de ecrãs responsivos. As restantes tecnologias, pelo motivo de funcionarem à base de linguagem de programação orientada à *web* conseguem cumprir na integra quase qualquer desafio imposto pelo programador.

3.2.2.3 Aplicação multi-idioma

Uma aplicação disponibilizada nas lojas virtuais de aplicações móveis de todo o mundo deverá ter em conta os vários idiomas dos utilizadores alvo. Entre os idiomas

fundamentais para este projeto encontram-se o Português, o Inglês, o Espanhol e o Francês.

De forma a criarmos um produto multilíngue ou multi-idioma, precisamos de abordar diferentes métodos oferecidos pelas tecnologias acima definidas.

Começando pela ferramenta Ionic, podemos facilmente encontrar artigos e tutoriais que nos ajudarão a cumprir com este requisito. A abordagem mais comum é a utilização do pacote NPM designado ngx-translate. Este módulo necessita de uma estrutura de pastas bem definida onde serão colocados ficheiros formato JSON que contêm as chaves e respetivas traduções. Um ficheiro JSON por pasta e uma pasta por idioma é a estrutura essencial ao funcionamento multilingue do projeto (HOW TO BUILD IONIC 3 MULTI-LANGUAGE APP, 2017).

Analisando em seguida Xamarin, descobrimos uma ferramenta bastante útil para facilitar o trabalho de tradução, o Multilingual App Toolkit. Este kit ajuda no processo de tradução pois oferece uma interface gráfica e comandos para gerar automaticamente as traduções da aplicação para os idiomas pretendidos. A estrutura de ficheiros é um pouco diferente da estrutura anterior. Os ficheiros de tradução são colocados todos na mesma pasta, mas com um código identificador de idioma no nome do ficheiro (Add Languages to Your Xamarin Apps with Multilingual App Toolkit, 2018).

Construir aplicações multi-idioma em Unity é também possível. O formato dos ficheiros que contêm as traduções são do mesmo tipo, JSON, no entanto, o processo de leitura é distinto. A leitura do ficheiro de traduções é feita aquando da execução da aplicação, guardado em memória num objeto do tipo lista e mais tarde utilizado para consulta. A lógica de leitura e consulta de dados é não nativa, pelo que deverá ser implementada pelo programador (Unity 3D | Localized Text Component, 2019).

Em seguida analisamos o cumprimento do requisito com base na ferramenta Microsoft Power Apps. As Power Apps têm também elas a capacidade de reconhecer e traduzir dinamicamente o conteúdo textual de uma aplicação. O formato de ficheiro utilizado é um simples ficheiro Excel que contém o código do idioma, a chave e respetiva tradução.

Este ficheiro é de fácil manutenção e poderá ser importado na ferramenta como uma nova *data source*. A aplicação irá ler em tempo real a tradução de que necessitar (Building multilingual apps in PowerApps, 2017).

Finalmente chegamos à quinta e última solução, as PWAs. As *Progressive Web Apps* são a tecnologia sobre a qual seria necessário um maior estudo e pesquisa, pois a documentação e informação tecnológica, em comparação com as restantes, é quase nula. Um artigo encontrado ao longo da fase de análise refere um método para implementar um sistema multilingue numa PWA. Este método, eficaz, mas pouco eficiente, envolve a criação de diferentes módulos ou páginas para cada idioma disponibilizado (CREATE A MULTILINGUAL APP, 2018). Um plano pouco escalável e que trará com o seu tempo dificuldades na manutenção do projeto. Por se tratarem de aplicações *web*, as PWAs irão certamente dispor futuramente de outras soluções de forma a cumprirem eficientemente com este requisito.

3.2.2.4 Facilidade de comunicação com APIs

Uma vez mais, e pegando de exemplo o subcapítulo 3.2.1.3, a comunicação com APIs externas é relativamente simples, a aplicação apenas deverá ser capaz de enviar pedidos e receber respostas de um link HTTP ou HTTPS.

3.3 Casos de uso

Uma vez feita a análise ao estado da arte e recolhida toda a informação necessária sobre os potenciais interesses dos utilizadores, é altura de definir quais as funcionalidades base que a aplicação irá disponibilizar. Das funcionalidades selecionadas encontramos o login e registo de nova conta, a possibilidade de o jogador ver a sua coleção de crachás, ver os desafios/instituições semanais, o registo de donativos nos desafios, a receção de notificações (*push notifications*) aquando da captura ou evolução de um crachá e por fim a capacidade de partilha de conquistas nas redes sociais.

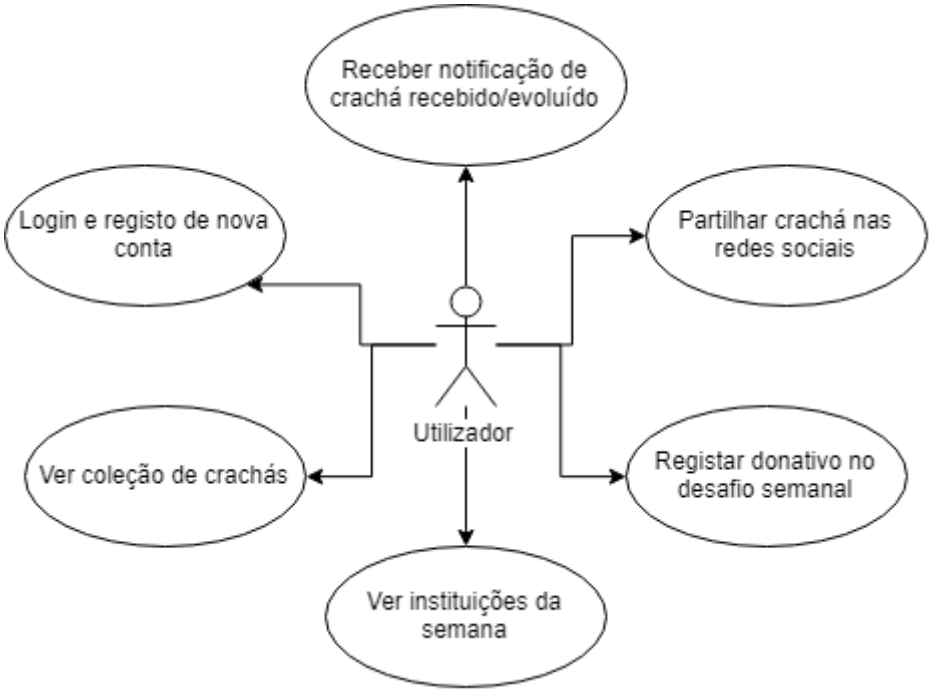


Figura 12 - Diagrama de casos de uso

3.3.1 Login e registo de nova conta

Tal como qualquer aplicação e/ou serviço web que pretenda a persistência de pontuações e perfis, irá ser necessária a implementação de um sistema de login e registo de utilizadores.

O jogador, após entrada na aplicação, verá dois botões distintos, “login” e “registar nova conta”.

A funcionalidade de login pedirá ao utilizador para introduzir os seus dados de acesso (email/username e password). Caso identificado com sucesso pelo sistema, o jogador será reencaminhado para a sua página *home* da aplicação onde poderá consultar o seu perfil, desafios semanais e coleção de crachás.

Caso o utilizador selecione a opção de novo registo, será reencaminhado para uma nova página onde lhe serão solicitados alguns dados, email, nome de utilizador, idade e palavra-chave. Uma vez preenchido o formulário de registo, será enviado um email de

confirmação para o endereço indicado. Após confirmação, o utilizador poderá aceder à aplicação através da funcionalidade de login.

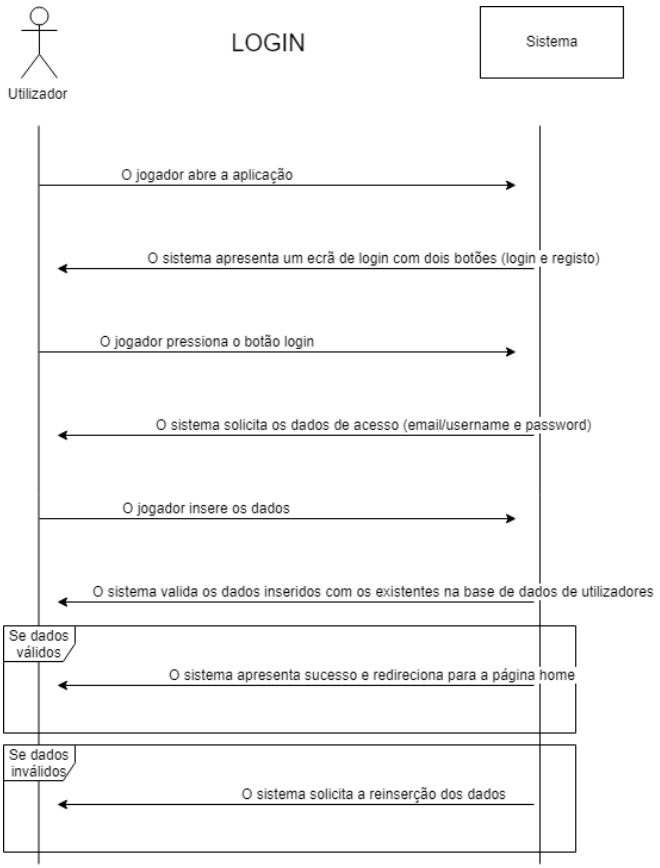


Figura 13 - Diagrama de caso de uso - Login

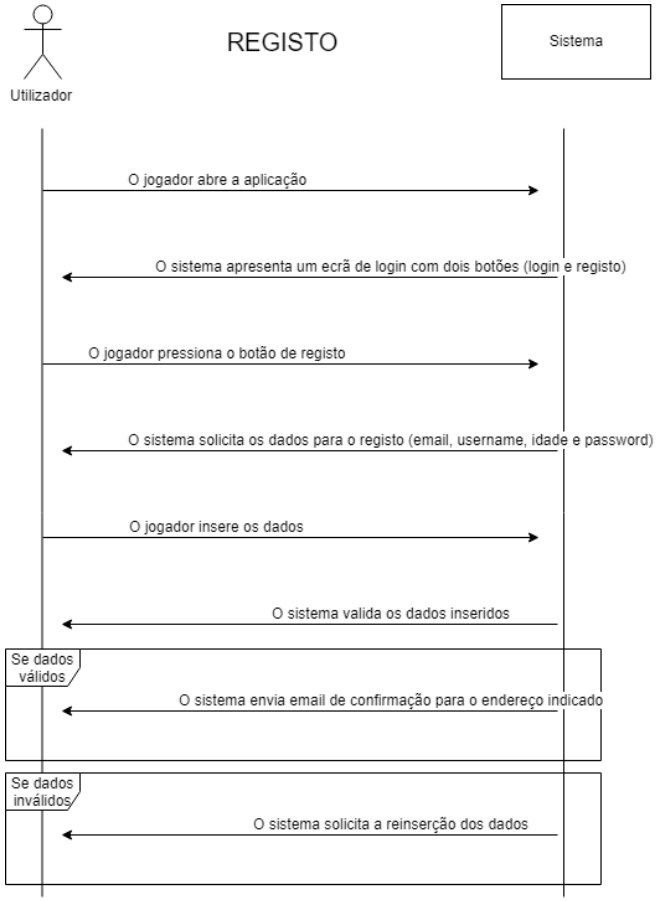


Figura 14 - Diagrama de caso de uso - Novo registo

3.3.2 Ver coleção de crachás

Uma vez autenticado, o jogador terá acesso à sua página de jogo, onde poderá consultar a coleção de crachás. O utilizador indica ao sistema que pretende aceder à sua coleção pessoal através de um clique na ação gráfica respetiva. A aplicação redireciona o jogador para uma nova página onde são apresentados, em forma de caderneta, os crachás capturados e por capturar, estilo inspirado no tradicional jogo de Pokemon (Pokémon GO! — Version 2.0 Concept, 2017).

Figura 15 - Exemplo interface gráfica Pokemon GO (Pokémon GO! — Version 2.0 Concept, 2017)

Dentro da página de listagem de crachás é ainda possível selecionar um item caso se pretenda ver mais detalhes. Nesta nova página o jogador tem a acesso a informação como o nome do crachá, nível atual e associação/instituição associada.

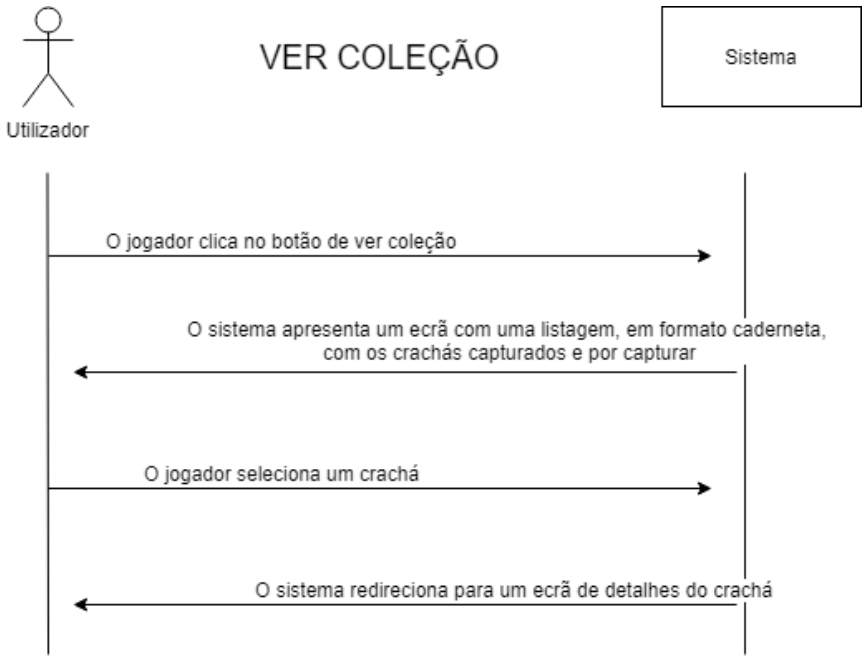


Figura 16 - Diagrama de caso de uso - Ver coleção

3.3.3 Ver instituições da semana

A página principal desta aplicação apresentará os desafios semanais. É importante, por questões de experiência de utilização do jogador, que a primeira página que surja ao utilizador, logo após o login, seja a página onde é possível ver e doar às instituições da semana. Serão apresentados três desafios a cada semana, cada um relacionado com uma organização, instituição ou movimento social. Ao fim de cada semana e caso atingido os objetivos semanais, serão lançados novos desafios. Os desafios que não acumularam doações suficientes, permanecerão em jogo na semana seguinte.

A apresentação será feita através de um sistema simples de navegação. Aleatoriamente, aquando da inicialização da aplicação, será selecionado um desafio para ser apresentado em primeiro lugar. O utilizador poderá navegar para os restantes desafios através de um deslize no ecrã tátil do dispositivo, tal como o sistema de navegação carrossel (Budiu, 2018).

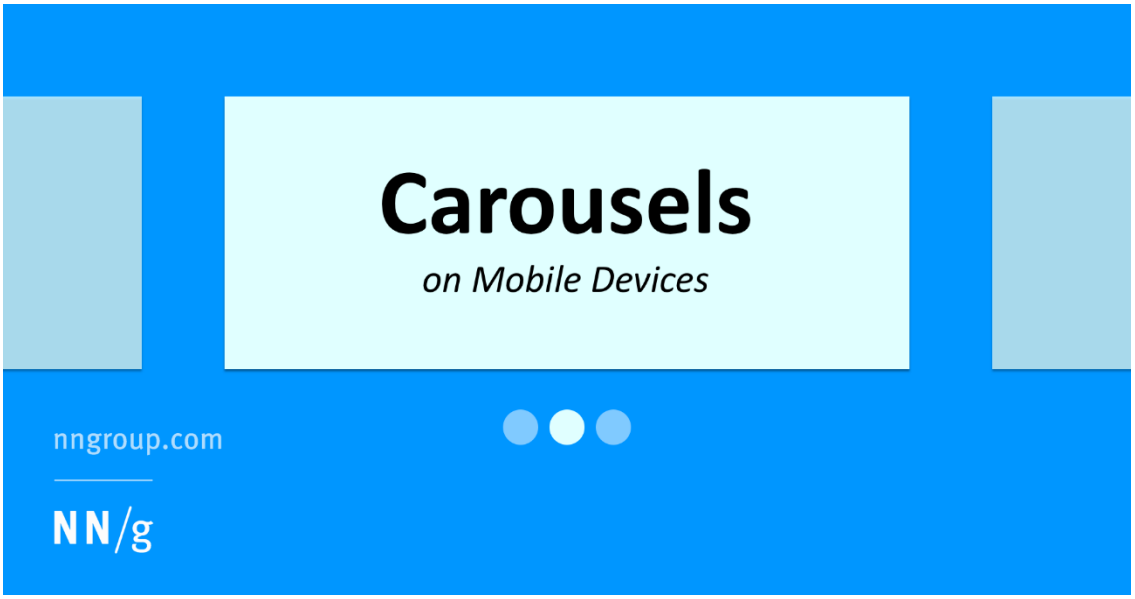


Figura 17 - Exemplo sistema de navegação carrossel (Budiu, 2018)

3.3.4 Registar donativo no desafio semanal

A funcionalidade mais importante desta aplicação, e que justifica a investigação e desenvolvimento deste projeto, é o registo de donativos nos desafios semanais.

Para executar esta ação, o jogador deverá aceder à página principal dos desafios semanais e escolher qual o desafio em que deseja participar. Uma vez selecionado, o utilizador poderá premir o botão "Doar" situado na margem inferior e ser-lhe-á solicitada a quantia pretendida. Após determinada e inserida a quantia o sistema confirma a doação ao utilizador e automaticamente lhe deverá apresentar os métodos de pagamento disponíveis (Paypal, cartão de crédito, multibanco e MBWay). O sistema responsável pela gestão dos pagamentos informará em seguida a API acerca do recebimento da doação, registando na base de dados como uma transação confirmada. O valor transferido é usado para recalcular o valor em falta para o desafio semanal, atualizando a interface gráfica e informando os jogadores em tempo real do processo a decorrer.

3.3.5 Receber notificação de crachá recebido/evoluído

Um requisito também ele importante para ajudar no aumento da retenção dos utilizadores, é o uso de *push notifications* para alertar o jogador aquando da conquista de um novo crachá.

Tal como é mencionado no estudo (Cronin, 2018), o apresentar regular de notificações ao utilizador, garante o não esquecimento e consequente aumento da taxa de retenção na aplicação. Um outro estudo, descrito no artigo (App Benchmarks Show Improving Engagement, Power of Push, 2018), revela que o número de sessões gastas pelos utilizadores difere significativamente entre aplicações com e sem *push notifications* ativas.

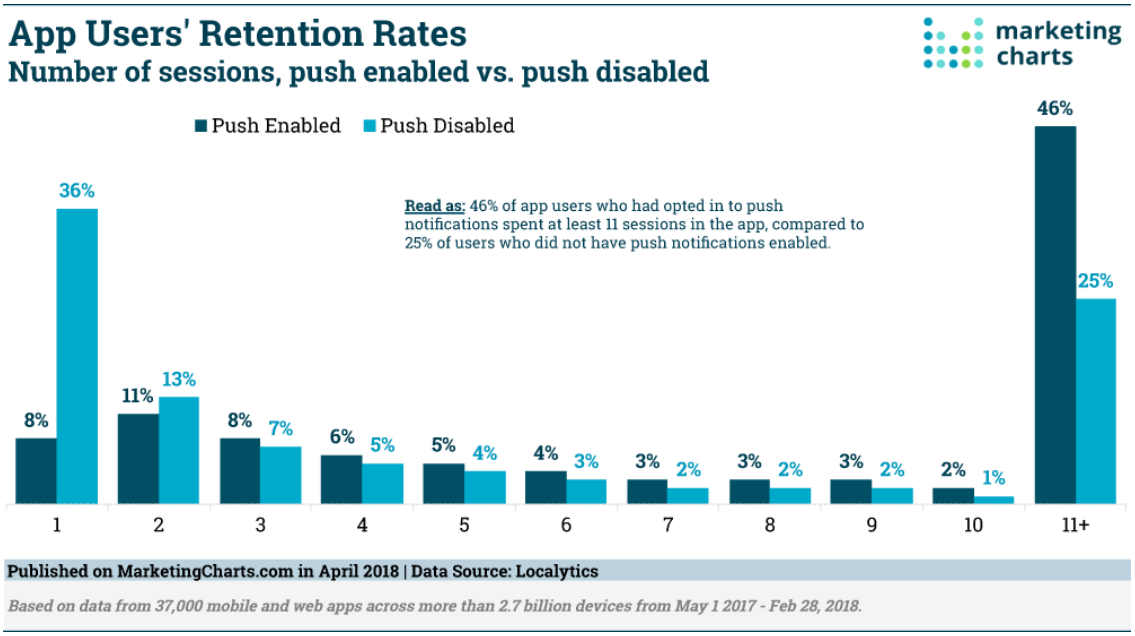


Figura 18 - Gráfico de comparação nº de sessões (notificações ativas e notificações desativas) (App Benchmarks Show Improving Engagement, Power of Push, 2018)

Sempre que um jogador obtiver um registo de alteração dos seus crachás na base de dados, a API deverá enviar uma mensagem à aplicação *frontend* que irá por sua vez alertar o jogador sob a forma de uma notificação de sistema.

3.3.6 Partilhar crachá nas redes sociais

A partilha de conquistas nas diversas redes sociais será realizada a partir da página de coleção ou página de detalhes de um crachá. Na página de coleção, um clique com a duração de um segundo deverá apresentar uma *tooltip* a informar acerca das diferentes ações possíveis de realizar, entre as quais, a partilha. Na página de detalhes de um crachá, esta ação estará disponível num botão "Partilhar", presente no canto superior direito.

3.4 Arquitetura da solução

Após análise dos requisitos e funcionalidades necessárias à implementação desta aplicação, foi desenhada uma arquitetura apropriada. Esta arquitetura envolve uma

componente *frontend* que estará instalada em cada dispositivo móvel dos jogadores, uma base de dados não relacional e uma API capaz de criar a ligação entre as duas, como é possível de observar na Figura 19 - Diagrama de componentes.

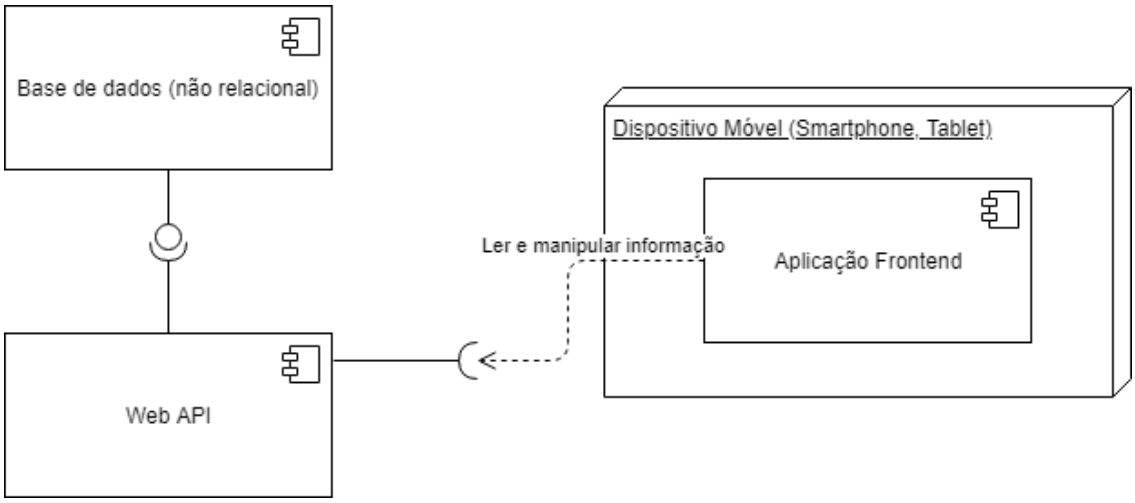


Figura 19 - Diagrama de componentes

Em relação à componente de negócio, este projeto engloba relacionamentos entre várias entidades. Tal como exemplificado na Figura 20 - Modelo de domínio, o modelo de domínio encontra-se dividido em utilizadores, desafios, crachás, doações e instituições, organizações ou movimentos sociais. Cada jogador tem um perfil com alguns dados pessoais associados, como nome, email e idade. Para cada desafio semanal deverá existir uma instituição ou organização diretamente associada (representada pela entidade "Target"), bem como um crachá para premiar os participantes. Como a entidade "crachá" irá estar associada tanto a desafios como a jogadores, será necessário criar uma entidade intermédia para lidar com esta relação de muitos para muitos. A última entidade é a responsável pelo registo das doações. Cada doação será composta por um utilizador, uma quantia e um desafio semanal.

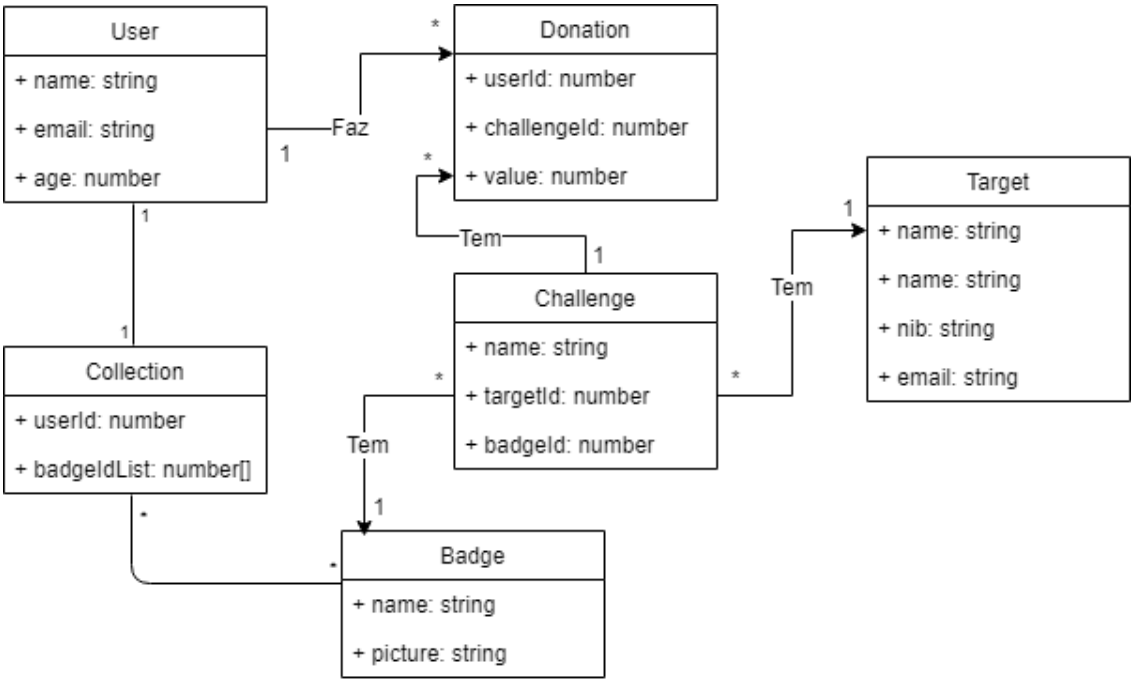


Figura 20 - Modelo de domínio

3.5 Tecnologias e serviços escolhidos

Após análise das soluções existentes, requisitos funcionais e não funcionais, e casos de uso, foi possível escolher quais os serviços e tecnologias a usar no desenvolvimento deste projeto.

Começando por abordar a componente gráfica da aplicação, identificámos a tecnologia Ionic como melhor solução para o problema descrito. O Ionic é uma tecnologia robusta e com um grande suporte por parte da comunidade informática, trazendo confiança ao desenvolvimento. Esta tecnologia disponibiliza nativamente inúmeros *plugins* para as mais variadas funcionalidades, entre as quais *push-notifications*, alertas, *popups* e terminais de pagamento, indispensáveis para o produto em causa (Ionic Native Community Edition, 2019). A escolha do versionamento do Ionic será abordada mais abaixo neste documento.

O segundo tema de discussão e ponderação foi a escolha do tipo de base de dados a usar. Conforme é analisado mais acima e resumido na Figura 8, o tipo de base de dados

mais indicado para esta aplicação seria o tipo não-relacional, tendo sido por isso escolhido o MongoDB como serviço a utilizar. O MongoDB oferece um conjunto de ferramentas importantes na implementação de bases de dados não-relacionais.

A primeira ferramenta é o MongoDB Atlas, um hospedeiro de base de dados baseado em *cloud* (What Is MongoDB?, 2019). Pelo plano gratuito deste serviço, é-nos oferecido um armazenamento de até 512 MB (MongoDB Atlas Pricing, 2019).

A segunda ferramenta, o MongoDB Compass, é um gestor de base de dados que permite a edição de dados, inserção de tabelas e execução de *queries*. Esta ferramenta não é essencial, pelo facto de os serviços por ela disponibilizados estarem também presentes no gestor online do MongoDB Atlas, no entanto, vê a sua utilidade num uso mais intensivo, com necessidades de *performance* acrescidas e na gestão de bases de dados locais, modo *offline* (MongoDB Compass, 2019).

A terceira e última componente, alvo de análise tecnológica, foi o servidor responsável pela ligação entre aplicação e base de dados. Conforme já mencionado acima, identificamos duas potenciais tecnologias, NodeJS e ASP.NET Core. Ambas as alternativas levariam a uma solução idêntica quanto à sua eficiência. A decisão foi baseada na preferência pessoal da linguagem de programação a usar, tendo sido selecionado o NodeJS.

A tecnologia NodeJS, permite de forma imediata e sem qualquer configuração adicional correr na máquina local uma nova instância de um servidor por nós criados. Este servidor, no caso particular do projeto HelpIt, disponibiliza uma WebAPI para receber instruções e enviar resultados entre a aplicação móvel e a base de dados MongoDB.

Aquando da época de desenvolvimentos e testes locais, o simples levantamento de uma nova instância na máquina é o suficiente, no entanto, quando se pretende testar a aplicação num dispositivo real, precisamos de garantir um acesso online a um endereço web. De maneira a gerar um endereço online para a nossa WebAPI, seria necessária a

escolha de um hospedeiro web compatível com NodeJS, levando assim à seleção da última ferramenta essencial ao desenvolvimento deste projeto.

O serviço de hospedagem escolhido foi o Heroku, serviço esse já referido anteriormente no capítulo 3.1.2 e que será detalhado abaixo ao longo do processo de desenvolvimento da aplicação.

Tabela 1 - Tabela de seleção de serviços e tecnologias

Serviços e tecnologias	
Tecnologia de criação de aplicações	Ionic
Tipo de base de dados	MongoDB
Serviço de hospedagem de base de dados	MongoDB Atlas
Serviço de manutenção de base de dados	MongoDB Compass
Tecnologia de criação de API's	NodeJS
Serviço de hospedagem de API's	Heroku

4 Desenvolvimento da solução

Neste capítulo de desenvolvimento da solução é descrito ao pormenor todo o processo de implementação da aplicação, desde a componente visual *frontend* ao servidor *backend* responsável pela conexão com a base de dados e envio de notificações.

Ao longo do capítulo são abordados temas como base de dados, servidores locais, hospedagem de aplicações web, autenticação de utilizadores, *push-notifications*, processo de pagamento e transferências monetárias e ainda distribuição de aplicações móveis para as diversas lojas aplicacionais (Play Store e App Store).

4.1 Criação aplicação base Ionic

Este projeto vê-se dividido em três componentes principais, a aplicação gráfica e visual (*frontend*), uma base de dados onde é armazenada toda a informação de negócio (*backend*) e um servidor responsável pela interação entre estes dois sistemas.

A primeira componente, conforme já mencionado no capítulo 3.5, foi desenvolvida sobre a ferramenta de criação de aplicações móveis Ionic. A versão da *framework* utilizada nesta implementação foi a Ionic 4 juntamente com Angular 7.

Antes de iniciar o desenvolvimento de qualquer aplicação Ionic, deve ser validada uma lista de requisitos tecnológicos, conforme mencionado na documentação oficial (Your First Ionic App: Angular, 2019). Esta *framework* requer quatro ferramentas mínimas iniciais. A primeira é o Git, responsável pelo controlo de versões e facilitador de programação em paralelo, por parte de diferentes programadores, ao longo do desenvolvimento do projeto. Em seguida, a instalação de NodeJS para interação com o ecossistema Ionic, a escolha de um editor de código e de um terminal para execução de comandos. Neste projeto em particular foi escolhido o editor Visual Studio Code, pela sua simplicidade, eficácia e disponibilização nativa de um terminal de comandos.

Após as validações tecnológicas iniciais, segue-se a instalação da plataforma Ionic e Cordova através da execução do seguinte comando:

```
$ npm install -g ionic cordova
```

Figura 21 - Comando instalação Ionic e Cordova

Cordova é um serviço *open-source*, utilizado pelo Ionic no desenvolvimento de aplicações móveis multiplataforma. Este serviço permite ao desenvolvedor programar em linguagem web – HTML5, CSS3 e Javascript como se desenvolvesse nativamente para cada uma das diferentes plataformas (Android, iOS e Windows) (Cordova Apache - Introduction | Overview, 2016).

O passo seguinte é a criação da aplicação base Ionic, seguindo uma estrutura aplicacional *template* com menu lateral, executando o comando:

```
$ ionic start helpit sidemenu
```

Figura 22 - Comando inicialização projeto Ionic

Este comando indica ao serviço Ionic para gerar um novo projeto do tipo *sidemenu* com o nome "helpit". Este tipo de projeto permite à aplicação ter um menu lateral que servirá de sistema de navegação entre páginas para os utilizadores. Por predefinição, o Ionic utiliza a *framework* Angular no desenvolvimento dos seus projetos, no entanto, possibilita o uso de *frameworks* alternativas, React e Vue.

O recém-criado projeto, contém já uma estrutura de pastas bem definida, separando a aplicação em modelos, serviços, componentes e páginas. Cada componente e página são compostos por um ficheiro HTML com o *template* gráfico, um ficheiro SCSS onde são definidos os estilos visuais, um ficheiro Typescript que contém a lógica de interação

com o utilizador e ainda um ficheiro direcionado para a implementação de testes unitários. Os modelos são simples ficheiros de classes representativos de entidades a usar ao longo do projeto, muitas vezes espelho de tabelas existentes em bases de dados. Por fim, os serviços veem-se descritos em ficheiros do tipo Typescript e são responsáveis por toda a lógica de negócio e conexão entre aplicação móvel e WebAPI, tema que será abordado mais à frente neste documento (Implementação servidor NodeJS).

Uma vez terminada a geração do novo projeto, é possível servir a aplicação num endereço local do tipo *localhost*. Para iniciar a aplicação deveremos aceder à pasta de projeto e indicar ao Ionic a nossa intenção, através da execução consecutiva dos comandos:

Figura 23 - Comando aceder a diretório

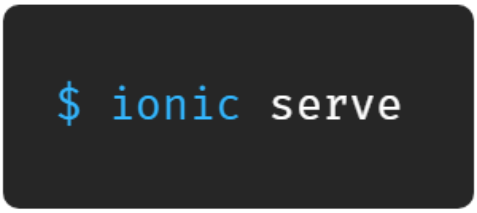


Figura 24 - Comando para correr projeto Ionic

A chamada ao comando "*ionic serve*" indica ao ecossistema Ionic, não só para servir a nossa aplicação, mas também para estar atento às eventuais alterações de código enquanto se encontra em execução. Esta funcionalidade facilita o processo de desenvolvimento, reduzindo os tempos de compilação da aplicação.

4.2 Base de dados MongoDB

A segunda componente principal, responsável pelo registo, atualização e manutenção de todos os dados que alimentam a aplicação é a base de dados.

Conforme detalhado no capítulo 3.5, o serviço de hospedagem de base de dados escolhido foi o MongoDB Atlas, acompanhado pelo MongoDB Compass como ferramenta de gestão e manutenção.

Para configurar a nova conta MongoDB Atlas dentro da ferramenta Compass, apenas foram necessários quatro passos sequenciais, os quais documentados nos tutoriais oficiais da tecnologia (MongoDB | Documentation - Connect via Compass, 2019). O primeiro passo foi aceder ao gestor online da nossa nova base de dados e selecionar o *cluster* desejado para a conexão. Em seguida, fundamental em questões de segurança, foi necessário registar o endereço IP da máquina de desenvolvimento como seguro na *whitelist* do *cluster*, permitindo assim a conexão e realização de pedidos à base de dados. O passo seguinte foi a criação de um novo utilizador para acessos de administrador. Este novo utilizador servirá como conta de *login* no *cluster* e base de dados. Em ordem de conclusão, o passo final foi a indicação ao gestor online da intenção de conexão com o MongoDB Compass. Uma vez instalada a ferramenta na máquina de desenvolvimento, o gestor online reencaminha o utilizador para o Compass, já com os dados de conexão preenchidos, apenas requisitando os dados de acesso da conta administradora. Este último passo pode ser observado na imagem abaixo.

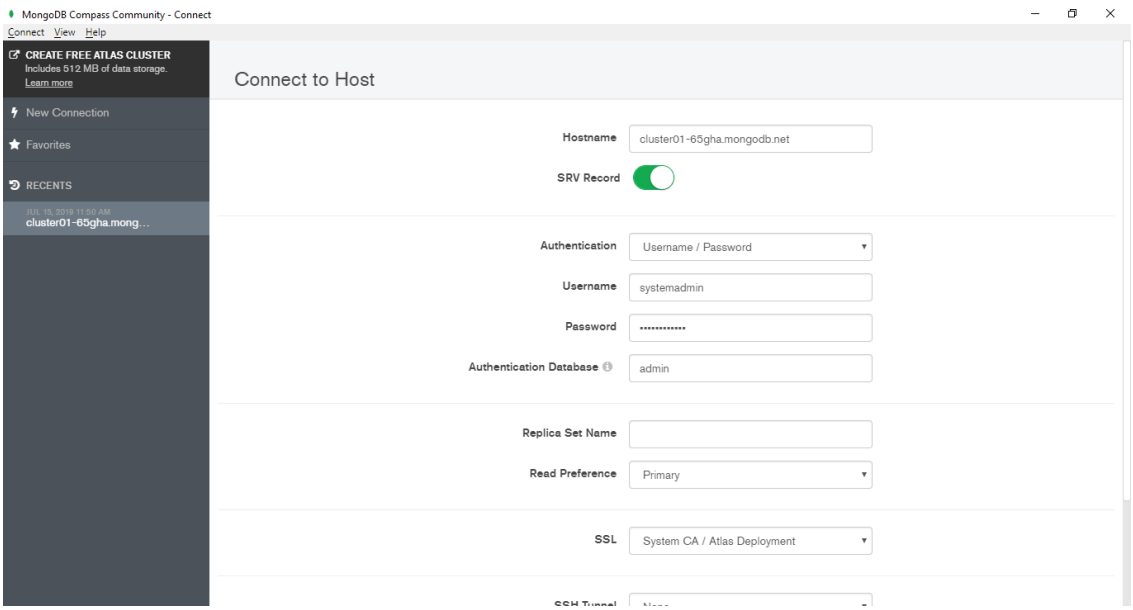


Figura 25 - Conexão com cluster MongoDB

Uma vez dentro da ferramenta de trabalho, seguiu-se a configuração das primeiras *collections*/tabelas da nova base de dados. As coleções criadas nesta fase, seguiram uma estrutura inicial definida anteriormente no subcapítulo de design 3.4. Mais à frente no projeto, foram efetuadas algumas alterações nesta configuração base.

Conforme visível na Figura 26, as primeiras tabelas a serem criadas foram as responsáveis pelos crachás (*badge*), desafios (*challenge*), registo de donativos (*donation*) e utilizadores (*user*).

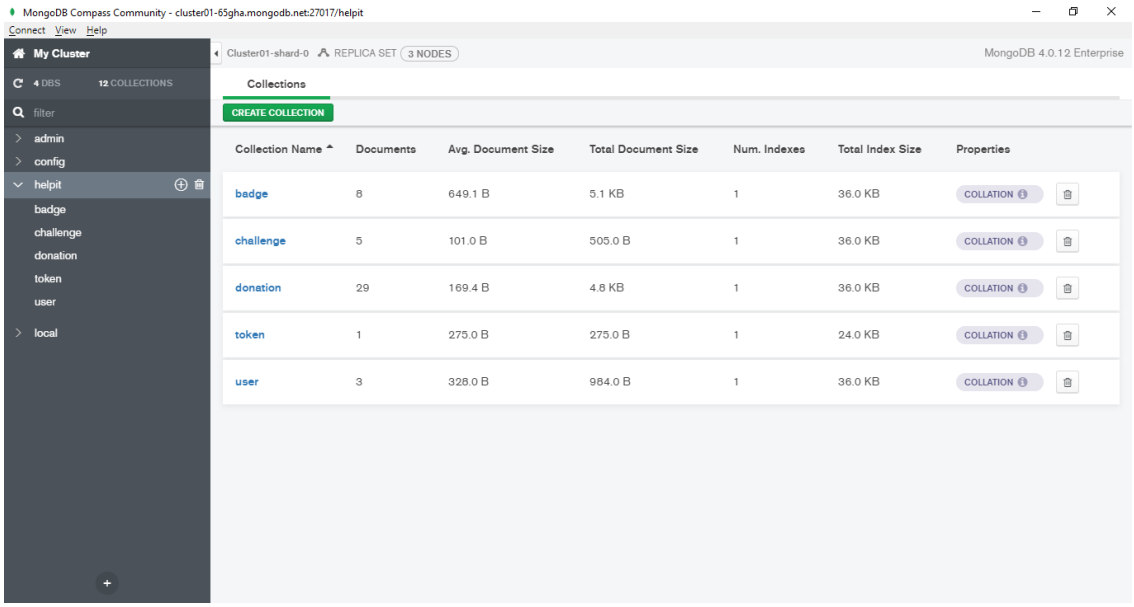


Figura 26- Listagem de tabelas da base de dados

4.3 Implementação servidor NodeJS

O terceiro componente e conetor entre a aplicação e base de dados é o servidor. Conforme mencionado anteriormente, a tecnologia escolhida para a implementação desta estrutura foi o NodeJS. Com este servidor, espera-se que seja capaz de receber instruções, processar informação e enviar uma resposta de regresso à aplicação. Para isso, o componente disponibiliza um conjunto de rotas, que servirão de endereços *web* para as chamadas da aplicação.

Antes de iniciar o desenvolvimento de um servidor NodeJS é necessária a instalação de três ferramentas, duas delas obrigatórias e uma opcional. As duas tecnologias obrigatórias são o NodeJS, que deverá ser instalado na máquina de desenvolvimento, e a conta de base de dados MongoDB, configurada no capítulo anterior 4.2. A terceira ferramenta é designada de Postman, uma aplicação *desktop* de realização de testes a WebAPI's.

O processo de implementação do servidor NodeJS seguiu o seu rumo com o auxílio de dois artigos principais. Do primeiro artigo " (Adesh, 2019)", utilizamos a estrutura base do servidor e organização dos métodos por ficheiros e rotas. Do segundo artigo " (Domes, 2017)", retiramos alguns dos comandos de consola para instalação de diversos *packages* necessários ao longo da implementação.

Uma vez terminada a configuração das ferramentas iniciais, foi possível iniciar a implementação do servidor. Para inicializar um novo projeto NodeJS, executamos o comando abaixo na nova pasta onde se pretende armazenar o código do servidor.

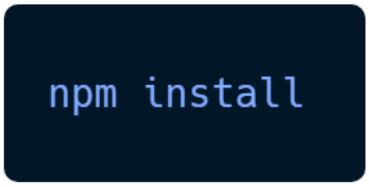


Figura 27 - Comando inicialização projeto NodeJS

Durante a execução do comando, é solicitado ao utilizador para definir o nome do novo projeto, neste caso "HelpIt". Após término da execução, vemos criado na pasta destino um ficheiro do tipo JSON, designado por package.json. Este ficheiro contém toda a informação acerca dos pacotes que se pretendem utilizar no projeto e *scripts* com instruções e operações a executar no servidor. O ficheiro gerado e posteriormente editado para o projeto relatado neste documento encontra-se representado na Figura 28.

```
{
  "name": "helpit",
  "version": "1.0.0",
  "description": "Help It Mobile Application Server",
  "main": "index.js",
  "scripts": {
    "start": "node server.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "body-parser": "^1.19.0",
    "cors": "^2.8.5",
    "express": "^4.16.4",
    "firebase-admin": "^8.3.0",
    "jsonwebtoken": "^8.5.1",
    "mongoose": "^5.5.7",
    "passport": "^0.4.0",
    "passport-jwt": "^4.0.0"
  }
}
```

Figura 28 - Ficheiro package.json

Conforme é possível de observar, dentro da tag *scripts* encontramos uma instrução designada “start”. Esta instrução despoleta a execução de um comando de consola responsável por ligar e correr o servidor. O nome de ficheiro server.js, é o nome do ficheiro Javascript que contém a lógica de inicialização base do servidor.

O passo seguinte foi a instalação de três pacotes essenciais, express, mongoose e body-parser. O ExpressJS é uma *framework* de NodeJS que disponibiliza um conjunto de funcionalidades no desenvolvimento web (ExpressJS | Home, 2017). O Mongoose é uma ferramenta simplificadora com implementações em MongoDB (MongooseJS | Home, 2011). O último pacote, body-parser, é um *package* NPM que vê o seu propósito na interpretação da leitura de pedidos HTTP e HTTPS (NpmJS | body-parser, 2019).

Para instalar estes pacotes, devemos indicar ao nosso gestor de *packages* (NPM) a respetiva instrução, através da execução do comando:

```
npm install express body-parser mongoose –save
```

Figura 29 - Comando instalação dos pacotes express e body-parser

O ficheiro package.json é automaticamente atualizado, refletindo de imediato a inserção dos novos pacotes.

Em seguida configuramos o ficheiro principal server.js com a inicialização de alguns *packages* e rotas base. O novo ficheiro, após configuração inicial, vê-se representado na Figura 30.

```
// get dependencies
const express = require('express');
const bodyParser = require('body-parser');

const app = express();

// parse requests
app.use(bodyParser.urlencoded({ extended: false }))
app.use(bodyParser.json())

// default route
app.get('/', (req, res) => {
    res.json({"message": "Welcome to HelpIt app"});
});

// listen on port 3000
app.listen(3000, () => {
    console.log("Server is listening on port 3000");
});
```

Figura 30 - Configuração inicial do ficheiro server.js

Conforme podemos observar na figura acima, iniciamos o serviço de express e indicamos ao pacote body-parser para ler os pedidos HTTP em formato JSON. Por fim, definimos uma rota *default* e colocamos o serviço express à escuta de pedidos.

A configuração indicada permite já correr o servidor na máquina local, usando o comando: `node server.js`.

4.3.1 Conexão com base de dados MongoDB

A etapa seguinte passou por conectar o servidor à base de dados MongoDB. Foi então necessária a criação de um novo ficheiro (config.js) com duas propriedades, o endereço URL de conexão com a base de dados e a porta sob a qual o servidor se tentará ligar na máquina hospedeira (exemplo na Figura 31).

```
module.exports = {
    url: 'mongodb://<dbUserName>:<dbUserPassword>@ds251002.mlab.com:51002/adeshtestdb',
    serverport: 3000
}
```

Figura 31 - Ficheiro config.js

Para consolidar o processo, foi necessário editar o ficheiro server.js, inicializando o serviço mongoose e conectando ao URL definido na configuração acima. Outra alteração foi a substituição da porta *hardcoded,* no código do servidor, pelo definido no ficheiro config.js. O código final, após configuração, ficou como representado na Figura 32.

```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();

var port = process.env.PORT || 3001;

// parse requests
app.use(bodyParser.urlencoded({ extended: false }))
app.use(bodyParser.json())

const mongoose = require('mongoose');

mongoose.Promise = global.Promise;

// Connecting to the database
mongoose.connect(db.url, { useNewUrlParser: true }).then(() => {
    console.log("Successfully connected to the database");
}).catch(err => {
    console.log('Could not connect to the database. Exiting now...', err);
    process.exit();
});

// default route
app.get('/', (req, res) => {
    res.json({ "message": "Welcome to HelpIt app" });
});

app.listen(port, () => {
    console.log("HelpIt Server is listening on port " + port);
});
```

Figura 32 - Ficheiro server.js após conexão com MongoDB

4.3.2 Definição de modelos, controladores e rotas

Uma outra etapa chave na implementação do servidor, foi a estruturação do código por pastas e ficheiros, fazendo separação do que eram modelos de dados, controladores e rotas de navegação.

Todos os modelos foram criados dentro de uma pasta *models*, os controladores numa pasta *controllers* e as rotas separadas em ficheiros por tipo de entidade associada (crachás, utilizadores, donativos, ...).

Os modelos, criados através do mongoose, são representações das entidades presentes na base de dados. Num ficheiro de modelo, definimos as propriedades que esperamos

receber por parte do MongoDB. Podemos observar um exemplo de um ficheiro modelo, representativo da entidade crachá ou *badge*, na Figura 33.

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const BadgeSchema = new Schema({
    name: String,
    picture: String,
    number: Number,
    progress: Number,
    description: String,
    collected: Boolean,
    level: Number,
    opened: Boolean
});

module.exports = mongoose.model('Badge', BadgeSchema, 'badge');
```

Figura 33 - Ficheiro badge.model.js

A implementação dos controladores seguiu a mesma lógica. Foram criados controladores por tipo de entidade associada, havendo assim uma separação temática de métodos e serviços. Um controlador de exemplo é o badge.controller.js (Figura 34), que agrupa todos os métodos relacionados com crachás, desafios semanais e coleções num único ficheiro.

```
const Badge = require('./../models/badge.model');
const User = require('./../models/user.model');
const Challenge = require('./../models/challenge.model');
const Donation = require('./../models/donation.model');
const Token = require('./../models/token.model');

// Retrieve a user's badge collection from the database.
exports.getCollection = (req, res) => {
    var collection = [];

    const userId = req.params.userId;
    const pageIndex = parseInt(req.params.pageIndex);
    const recordsPerPage = parseInt(req.params.recordsPerPage);

    var skip = (pageIndex * recordsPerPage);

    Badge.countDocuments().then((totalNumber) => {
        Badge.find().skip(skip).limit(recordsPerPage).then((badges) => {
            collection.push.apply(collection, badges);
            // collection.push.apply(collection, badges);
            // collection.push.apply(collection, badges);
            User.findById(userId).then(user => {
                if (!user) {
                    return res.status(404).send({
                        message: "User not found with id " + userId
                    });
                }
(...)
```

Figura 34 - Excerto do ficheiro badge.controller.js

O sistema de rotas não viu a sua segmentação feita por pastas, mas apenas por ficheiros. Cada ficheiro contém um conjunto de comandos que fazem a paridade entre rotas e métodos de controlador responsáveis por responder ao pedido aplicacional. Esta relação pode ser observada no ficheiro badge_routes.js (Figura 35).

```
module.exports = async function (app) {

    const badgeCtrl = require('./../controllers/badge.controller');

    app.get('/badge/collection/:userId/:pageIndex/:recordsPerPage', badgeCtrl.getCollection);

    app.get('/badge/details/:id/:userId', badgeCtrl.getDetails);

    app.get('/badge/weekchallenges/:userId', badgeCtrl.getWeekChallenges);

    app.get('/badge/toopenbadges/:userId', badgeCtrl.getToOpenBadges);

    app.post('/badge/addbadgetousers', badgeCtrl.addBadgeToUsers);

    app.post('/badge/openbadge', badgeCtrl.openBadge);
};
```

Figura 35 - Ficheiro badge_routes.js

4.3.3 CORS e finalização do servidor

Para concluir a configuração do servidor NodeJS, foi necessária a ativação de uma funcionalidade adicional. O CORS, ou Cross-Origin Resource Sharing, é um mecanismo HTTP que permite a um serviço que corre num determinado domínio, faça chamadas a endereços de domínio externo.

Posteriormente a todas as configurações iniciais, o ficheiro server.js vê-se conforme representado na Figura 36.

```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();
const cors = require('cors');

var port = process.env.PORT || 3001;

// parse requests
app.use(bodyParser.urlencoded({ extended: false }))
app.use(bodyParser.json())

app.use(cors());

const mongoose = require('mongoose');
require('./app/routes')(app, {});

mongoose.Promise = global.Promise;

// Connecting to the database
mongoose.connect(db.url, { useNewUrlParser: true }).then(() => {
    console.log("Successfully connected to the database");
}).catch(err => {
    console.log('Could not connect to the database. Exiting now...', err);
    process.exit();
});

// default route
app.get('/', (req, res) => {
    res.json({ "message": "Welcome to HelpIt app" });
});

app.listen(port, () => {
    console.log("HelpIt Server is listening on port " + port);
});
```

Figura 36 - Ficheiro server.js após configuração

4.4 Login e registo de utilizadores

Das funcionalidades base estruturais de uma aplicação, que visa a retenção e estudo dos dados dos seus utilizadores, é o sistema de registo e login.

Esta é uma das componentes mais importantes de um projeto *mobile*, e é sem dúvida das que requer mais atenção a nível de segurança, na medida de evitar o comprometimento dos dados. Existem diversas formas de implementar autenticação numa aplicação móvel, no entanto, a tecnologia escolhida para este projeto foi a autenticação JWT.

4.4.1 Autenticação JWT

O JWT, ou JSON Web Token, é um padrão *web* (RFC 7519) que define a transferência de informação de forma compacta e segura entre duas entidades, sob a forma de um objeto JSON. Os objetos JWT podem ser assinados de duas formas, através de um segredo, gerado a partir do algoritmo HMAC, ou da partilha de um par de chaves pública/privada, RSA ou ECDSA (Introduction to JSON Web Tokens, 2019).

Apesar de também usado na transmissão de informação entre entidades, o padrão JWT vê a sua utilidade na implementação de sistemas de autenticação. Aquando do login de um utilizador, é gerado um novo *token* JWT, que será usado para garantir ou não o acesso desse mesmo utilizador às diferentes páginas e rotas de uma aplicação.

Quando compactado, um *token* JWT contém três segmentos de informação, separados pelo delimitador ".".

O primeiro segmento, o *header*, é a conversão, em formato base64, de um objeto JSON que contém o tipo de *token* e algoritmo a usar (Figura 37).

Figura 37 - Exemplo *header* de um *token* JWT

O segundo segmento, designado por *payload*, contém, também sob a forma de objeto JSON, todas as *claims* (ou afirmações/informação) da entidade em questão, neste caso do utilizador. São possíveis de declarar três tipos de *claims* distintas, registadas, públicas e privadas. As *claims* registadas, são *claims* pré-definidas do padrão JWT e que são globalmente interpretadas pelos diferentes recetores. Por norma, não são informação obrigatória, mas são importantes na configuração de definições padrão. É exemplo, o tempo de expiração de um *token*. As *claims* públicas, podem conter qualquer tipo de informação que se pretenda que seja partilhada, apenas deverão obedecer a algumas regras de nomenclatura. Por fim, as *claims* privadas, são *claims* customizadas para o modelo de negócio do projeto em causa. abaixo fica um exemplo de um ficheiro *payload*.

Figura 38 - Exemplo *payload* de um *token* JWT

O terceiro e último segmento é conhecido como *signature*. Este segmento é construído através da codificação, com um segredo, do *header* e *payload*, em formato base64, usando o algoritmo definido no objeto JSON *header*. Aplicando como exemplo o uso do algoritmo HMAC SHA256, o segmento *signature* seria gerado conforme a Figura 39. Este último componente é importante pois permite validar a imutabilidade da mensagem durante a sua transmissão entre entidades.

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

Figura 39 - Exemplo *signature* de um *token* JWT

Um *token* JWT, após concatenação dos três segmentos, teria o aspeto seguinte:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

4.4.2 Implementação autenticação no servidor

Uma vez conscientes do tipo de autenticação a implementar, iniciou-se o desenvolvimento no lado do servidor. Esta implementação viu o apoio teórico e prático do artigo (Grimm, JWT Authentication with Ionic & Node.js – Part 1: The Auth Server, 2018).

Para implementar o sistema de autenticação NodeJS, baseado no padrão JWT, foram necessários quatros pacotes NPM adicionais, jsonwebtoken, passport, passport-jwt e bcrypt:

```
npm install jsonwebtoken passport passport-jwt bcrypt
```

Posteriormente, e seguindo um passo essencial da autenticação JWT, definimos o valor do segredo. Para futuro acesso a esta propriedade, adicionamos uma nova linha no ficheiro config.js, conforme demonstrado na Figura 40.

```
module.exports = {
    secret: '[redacted]',
    url: "mongodb+srv://systemadmin:[redacted]@cluster01-65gha.mongodb.net/helpit?
ssl=true&authSource=admin"
};
```

Figura 40 - Ficheiro config.js com adição de segredo JWT

Em seguida, surgiu a necessidade de criar a entidade utilizador na base de dados MongoDB, e espelhá-la num ficheiro modelo no lado do servidor. A entidade *user* encontra-se definida nas duas figuras abaixo.

Figura 41 - Entidade *user* MongoDB

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const UserSchema = new Schema({
    name: String,
    email: String,
    password: String,
    badges: [Object]
});

module.exports = mongoose.model('User', UserSchema, 'user');
```

Figura 42 - Entidade *user* Mongoose

A persistência de uma palavra-chave, nunca deverá ocorrer sob a forma de texto simples. Por questões de segurança, as *passwords* de login deverão ser encriptadas antes de persistidas na base de dados.

Vendo terminada a estrutura inicial dos dados, iniciou-se a implementação de um serviço *middleware*, responsável por, a cada pedido, verificar a autenticidade e acesso

de um determinado utilizador. Os pacotes utilizados neste desenvolvimento foram os já mencionados passport e passport-jwt. Este serviço interceta os pedidos feitos ao servidor NodeJS, desencripta o *token* JWT e devolve a presença ou ausência desse utilizador na base de dados (Figura 43).

```
var User = require('../models/user.model');
var JwtStrategy = require('passport-jwt').Strategy;
var ExtractJwt = require('passport-jwt').ExtractJwt;
var db = require('../../config/db');

var opts = {
    jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
    secretOrKey: db.secret
}

module.exports = new JwtStrategy(opts, function (jwt_payload, done) {
    User.findById(jwt_payload.id, function (err, user) {
        if (err) {
            return done(err, false);
        }
        if (user) {
            return done(null, user);
        } else {
            return done(null, false);
        }
    });
});
```

Figura 43 - Ficheiro *middleware* passport.js

Para concluir o desenvolvimento, restou criar o novo controlador responsável pelos métodos de registo e login de utilizadores.

O método de registo de novo utilizador, recebe por parâmetro um nome, email e *password*. Caso os parâmetros se encontrem inválidos ou o endereço de email já existir criado na base de dados, é retornada uma mensagem de aviso apropriada. A palavra-chave é encriptada através do uso do pacote bcrypt. Com os dados validados e encriptados, o novo utilizador é registado na base de dados com sucesso. O método retorna ainda um novo *token* JWT para poder ser armazenado na aplicação *frontend*. A Figura 44 reflete um excerto do recém implementado controlador.

```
const User = require('./../models/user.model');
const db = require('./../../config/db');
const jwt = require('jsonwebtoken');
const bcrypt = require('bcryptjs');

function createToken(user) {
    return jwt.sign({ id: user.id, email: user.email }, db.secret, {
        expiresIn: 86400 // 86400 expires in 24 hours
      });
}

// Register new User
exports.register = (req, res) => {
    (...)

    const name = req.body.name;
    const email = req.body.email;
    const password = bcrypt.hashSync(req.body.password);

    const newUser = new User({
        name: name,
        email: email,
        password: password
    });

    User.find({ email: email }).then(user => {
        if (user.email) {
            return res.status(409).send({
                message: "Email \"" + email + "\" already registered!"
            });
        }

        newUser.save().then(savedUser => {
            const expiresIn = 24 * 60 * 60;
            const accessToken = jwt.sign({ id: savedUser.id }, db.secret, {
                expiresIn: expiresIn
            });
            res.status(200).send({
                "user": savedUser, "access_token": accessToken, "expires_in": expiresIn
            });
        }).catch(err => {
            res.status(500).send({
                message: err.message || "Something wrong while registering the user."
            });
        });
    });
};
```

Figura 44 - Método registar utilizador no ficheiro auth.controller.js

O método de login de utilizador, recebe por parâmetro um endereço de email e uma palavra-chave encriptada. Após garantir que ambos os parâmetros contêm valor, o método desencripta a *password*, verifica se o utilizador existe na base de dados com o email fornecido e valida a autenticidade do par email/palavra-chave. Em caso de sucesso, um novo *token* JWT é gerado e retornado para a aplicação (Figura 45).

```
const User = require('./../models/user.model');
const db = require('./../../config/db');
const jwt = require('jsonwebtoken');
const bcrypt = require('bcryptjs');

(...)

// Login User
exports.login = (req, res) => {
    (...)

    const email = req.body.email;
    const password = req.body.password;

    User.findOne({ email: email }).then((user) => {
        if (!user) {
            return res.status(404).send('User not found!');
        }

        const result = bcrypt.compareSync(password, user.password);

        if (!result) {
            return res.status(401).send('Password not valid!');
        }

        return res.status(200).json({
            token: createToken(user)
        });
    (...)
};
```

Figura 45 - Método login de utilizador no ficheiro auth.controller.js

4.4.3 Implementação autenticação na aplicação

Terminada a implementação da autenticação no lado servidor, progredimos para a aplicação *frontend*. Este desenvolvimento apoiou-se no suporte técnico do artigo (Grimm, JWT Authentication with Ionic & Node.js – Part 2: The Ionic App, 2018).

Pelo motivo de utilizarmos o padrão de autenticação JWT, foi necessária a instalação de dois pacotes adicionais, ionic-storage e angular-jwt (Figura 46). O pacote ionic-storage viu o seu uso no armazenamento do *token* JWT proveniente do servidor. O pacote angular-jwt, ofereceu métodos e serviços para a leitura e escrita de objetos JSON Web Token.

```
npm i @ionic/storage
npm i @auth0/angular-jwt
```

Figura 46 - Instalação dos pacotes ionic-storage e angular-jwt

Por se tratar de uma aplicação Angular, é fornecido ao programador uma forma rápida de transitar entre testes locais e testes de produção. Esta transição é conseguida pela existência de dois ficheiros de *environment*, que contêm, tal como o nome sugere, variáveis específicas de cada ambiente. Como, numa fase ainda inicial, foi tomada a decisão de realizar o *deployment* do servidor NodeJS para o serviço Heroku (capítulo 0), configuramos juntamente os ficheiros de ambiente. O ficheiro *environment* de produção tem o aspeto sugerido pela figura abaixo.

```
export const environment = {
  production: true,
  baseUrl: 'https://helpit-api.herokuapp.com'
};
```

Figura 47 - Ficheiro *environment* de produção

Para a aplicação Ionic, foram criados dois ecrãs de autenticação, um de login e outro de registo de novo utilizador. De forma a responder aos pedidos de ambas as páginas e lidar com toda a lógica de autenticação, foi criado um serviço específico para o efeito. O serviço de autenticação é composto por uma variável de estado de autenticação, método para validar a expiração de um *token*, métodos de login, logout e registo e ainda um método público que devolve a validade de autenticação do utilizador atual da aplicação. Um excerto do código deste serviço pode ser observado na Figura 48.

```
(...)

import { Storage } from '@ionic/storage';
import { JwtHelperService } from '@auth0/angular-jwt';
import { User } from '../auth/user';
import { AuthResponse } from '../auth/auth-response';
import { environment } from '../../environments/environment';
import { Platform, AlertController } from '@ionic/angular';

@Injectable({
    providedIn: 'root'
})
export class AuthService {

    authenticationState = new BehaviorSubject(false);
    user = null;

    constructor(...) {
        this.platform.ready().then(() => {
            this.checkToken();
        });
    }

    checkToken() { (...) }

    login(user: User): Observable<any> { (...) }

    register(user: User): Observable<any> { (...) }

    logout() { (...) }

    isLoggedIn() { (...) }
}
```

Figura 48 - Excerto de código do ficheiro auth.service.ts

Tendo terminada a implementação do serviço, faltou um passo fulcral para a segurança da aplicação. Foi necessário o controlo de acesso, do utilizador *loggedin* na aplicação, às diversas páginas e rotas. Mais uma vez, o Ionic e Angular fornecem um apoio grande em funcionalidades deste tipo, através do uso das chamadas *guards* aplicacionais. As *guards* são mecanismos de interceção, que permitem controlar certos tipos de ações por parte dos utilizadores. São habitualmente utilizadas para validar a autenticidade e permissões de um utilizador para acesso a páginas da aplicação. Para este projeto apenas implementamos *guards* de autenticação. Foi então criado um novo ficheiro do tipo *guard*, com lógica de validação do estado de autenticação de um utilizador que tente aceder a uma página (Figura 49).

```
import { Injectable } from '@angular/core';
import { CanActivate } from '@angular/router';
import { AuthService } from './auth.service';

export class AuthGuardService implements CanActivate {

  constructor(public authService: AuthService) {}

  canActivate(): boolean {
    return this.authService.isLoggedIn();
  }
}
```

Figura 49 - Ficheiro *guard* auth.guard.ts

Para uma *guard* ser considerada pelo sistema como ativa, deve sempre ser inserida como parâmetro em todas as rotas onde seja necessária. No projeto documentado, inserimos esta validação em todas as rotas após página de *login* da aplicação. Esta ativação pode ser observada na Figura 50.

```
{
  path: 'home',
  loadChildren: './home/home.module#HomePageModule',
  canActivate: [AuthGuard]
},
```

Figura 50 - Exemplo ativação de uma *guard* numa rota aplicacional

De forma a concluir a implementação da autenticação na aplicação *frontend*, restou o desenvolvimento dos formulários de dados nas páginas login e registo de utilizador. Tendo o serviço e *guard* de autenticação prontos, as páginas apenas necessitaram de validar os campos email e *password*, no caso do login, e os campos nome, email, *password* e confirmar *password*, no caso de novo registo. Uma vez verificados os campos dos formulários, é chamado o serviço de autenticação e reencaminhado o utilizador para a página *home* em caso de sucesso. A aplicação, posteriormente a todos os desenvolvimentos gráficos, apresenta o grafismo representado na Figura 51.

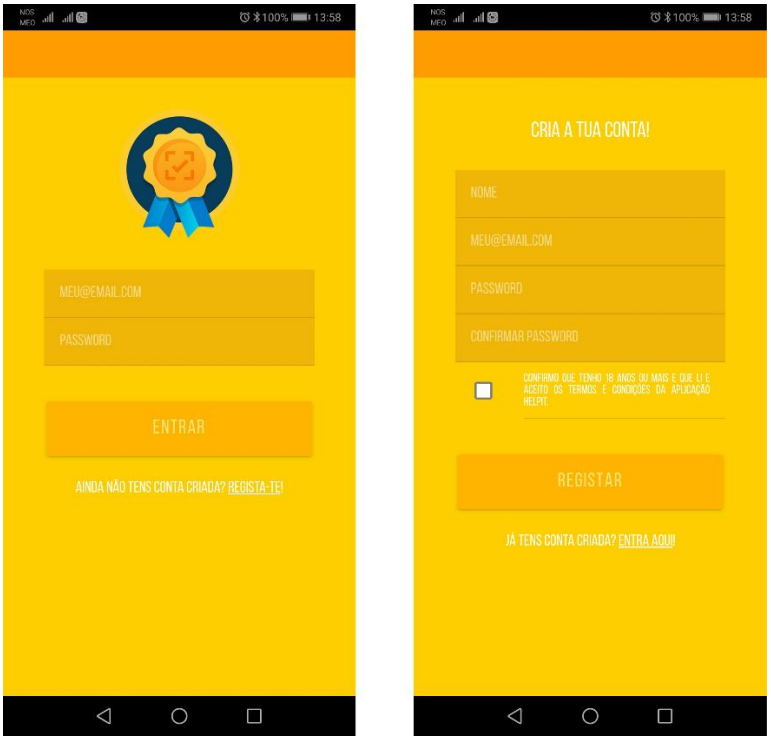


Figura 51 - Páginas de login e registo de novo utilizador

4.4.4 Deployment para o serviço Heroku

O *deployment* do servidor NodeJS para o serviço de hospedagem Heroku foi um passo essencial no desenvolvimento do projeto. O teste à ligação entre aplicação e serviço de hospedagem externo, viu a sua urgência o quanto antes durante a fase de implementação do servidor. Uma conexão com sucesso numa fase embrionária, permitiu o avanço mais rápido do desenvolvimento do projeto.

Em primeiro lugar, registou-se uma nova conta para o projeto HelpIt no website oficial da plataforma Heroku. Com esta conta foi possível a hospedagem de aplicações até um máximo de 500MB.

Dentro da nova conta, foi criada uma nova aplicação, definindo o nome e região de hospedagem (Europa ou Estados Unidos da América). A nova aplicação, para garantir a instalação de todas as dependências tecnológicas relacionadas com NodeJS, necessitou da configuração de um *buildpack*. Um *buildpack* é um conjunto de *scripts* que correm

de cada vez que uma aplicação é *deployed* para o serviço Heroku. O tipo de *buildpack* escolhido está diretamente relacionado com a linguagem de implementação do servidor, NodeJS.

Após configuração do serviço de hospedagem, foi possível o *deployment* da solução atual. Este processo é simples e requer apenas quatro passos. Inicia-se a tarefa com o login no serviço Heroku através da linha de comandos, seguindo-se a adição e *commit* das alterações atuais e finalizando com a instrução de *push*, enviando a solução para o hospedeiro (Figura 52). Se o processo ocorrer sem anomalias e o código da solução não apresentar erros de compilação, o servidor encontra-se *online*.

```
$ heroku login

$ git add .
$ git commit -am "my first app"
$ git push heroku master
```

Figura 52 - Instruções para *deployment* no serviço Heroku

4.5 Desafios semanais

A primeira página, introduzida ao utilizador após o login, é a página *home*. É nesta componente da aplicação que são apresentados os três desafios da semana. Cada desafio tem um objetivo monetário e crachá associados, e vão sendo alterados semana após semana até atingirem o valor proposto. Idealmente deveria ser da responsabilidade de um *job* automático do sistema a troca dos desafios entre semanas, no entanto, tal funcionalidade não foi implementada neste projeto.

A apresentação dos três desafios foi implementada sob a forma de *sliders* aplicacionais. Esta funcionalidade pertence ao conjunto de componentes disponibilizados nativamente pela ferramenta Ionic (Ionic Framework | Ion-slides, 2019). Os desafios semanais são requisitados, através de um pedido *web* ao servidor NodeJS, aquando do carregamento da página *home*. O servidor retorna os desafios com três estados

possíveis, novo desafio, desafio com crachá já adquirido pelo utilizador e desafio com objetivo monetário atingido com sucesso. Na Figura 53 é possível ter um vislumbre da página *home*, após conclusão da implementação.

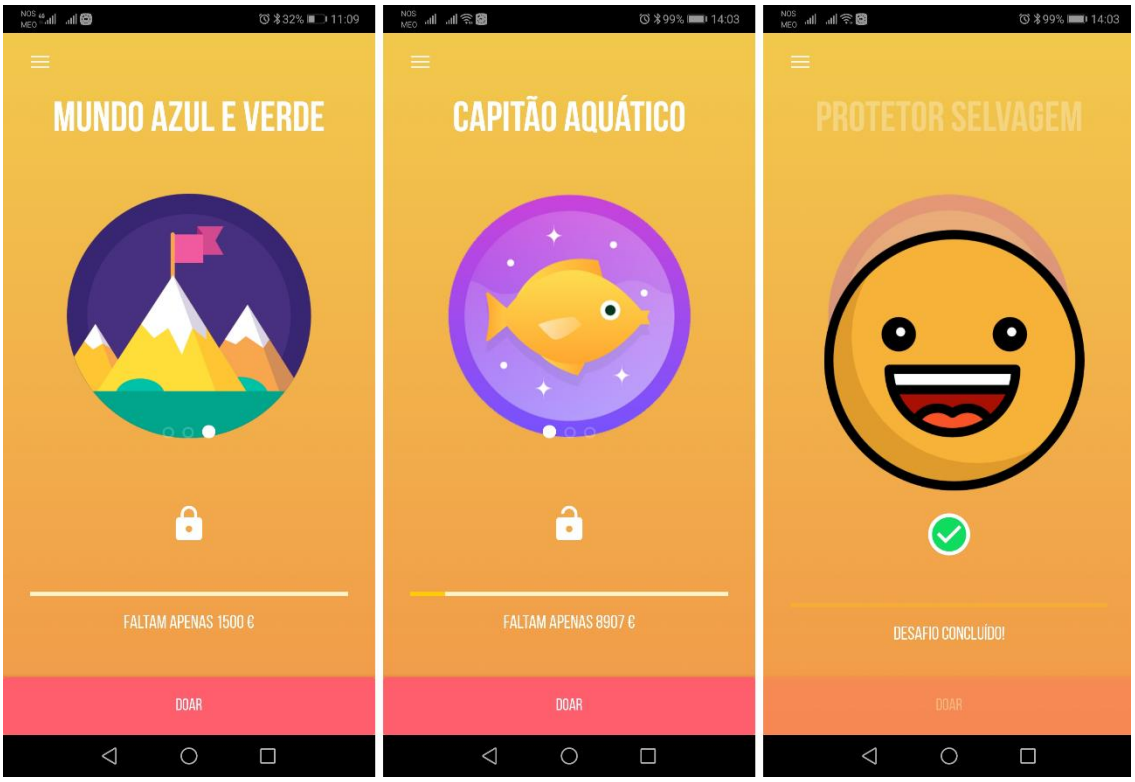


Figura 53 - Ecrã da página *home* com os três tipos de desafios

4.5.1 Registo de donativos

Durante a navegação na página principal da aplicação, o utilizador, para além de ver os desafios semanais, tem a possibilidade de registar um donativo para um desafio em particular, definindo a quantia e método de pagamento. Na implementação documentada neste relatório não foram desenvolvidas as funcionalidades de seleção de método nem registo oficial de pagamento, no entanto, esse processo é descrito teoricamente no capítulo 4.8.

Após encontrar o desafio pretendido, o utilizador pode selecionar a opção “Doar” e efetuar novo registo (Figura 54). A aplicação realiza novamente um pedido ao servidor NodeJS, que se encarregará de inserir na base de dados um novo donativo e atualizar o

montante em falta. Caso o desafio seja concluído, este mesmo serviço contactará a base de dados MongoDB, de forma a associar um novo crachá a todos os participantes na doação e enviar as respetivas notificações *push*.

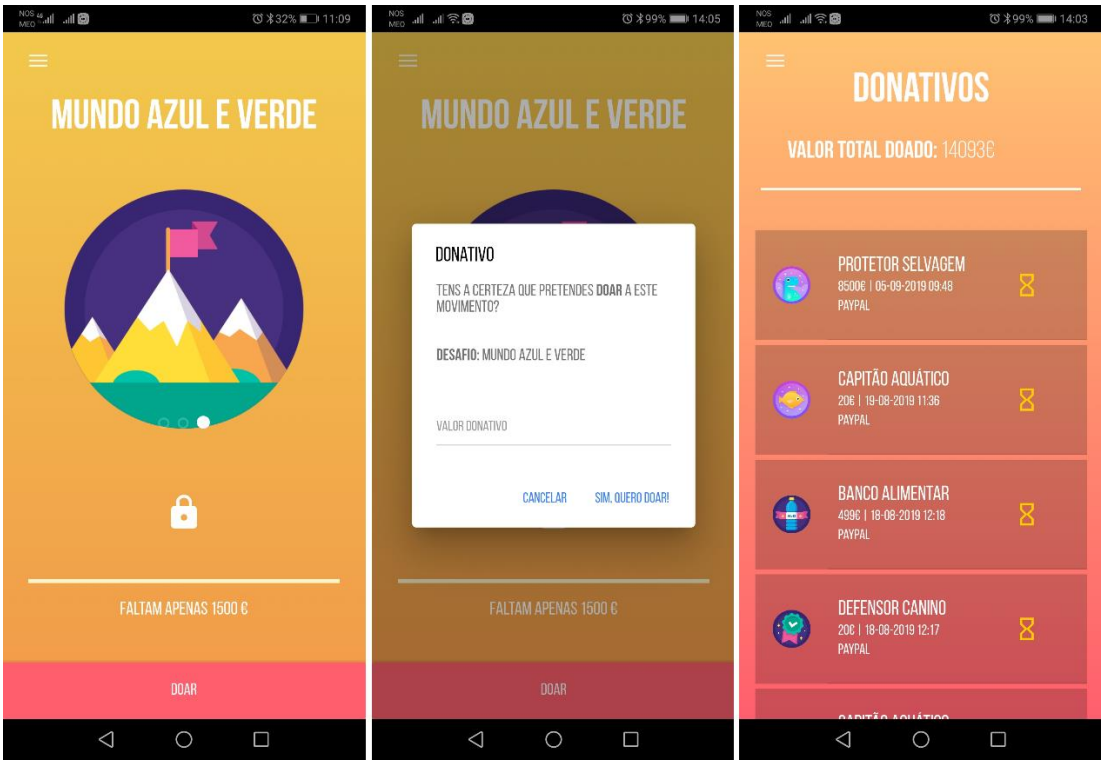


Figura 54 - Ecrã da página *home* - Registo de novo donativo

4.5.2 Notificações (Push)

O envio de notificações *push* para os utilizadores da aplicação, foi um dos processos mais complexos no desenvolvimento deste projeto. A funcionalidade viu-se dividida em duas implementações, lado aplicacional e lado servidor, acompanhadas pelo apoio de diversos artigos e tutoriais *web*.

Na maioria da documentação encontrada *online*, a implementação de *push notifications* passa pelo uso do serviço Firebase, já mencionado anteriormente no capítulo 3.2.1.4. Para este projeto, foi também escolhido o Firebase como ferramenta de envio de notificações. Após análise de diversos documentos e artigos, foi recolhida a informação mais importante acerca dos métodos mais eficientes neste tipo de implementação. No

lado aplicacional, foram selecionados três artigos, um útil na configuração inicial do ambiente Firebase (Junior, 2018), e outros dois essenciais na implementação do sistema de envio e receção de notificações na aplicação Ionic ((J., 2019) e (Rathore, 2019)). No lado servidor, foram também escolhidos três documentos fundamentais, tendo sido dois deles importantes na construção de um servidor NodeJS, capaz de comunicar com o serviço Firebase ((ANDRIANTO, 2018) e (Sending Firebase Cloud Messages from a Node.js Server, 2017)), e um outro, que surge da necessidade de um sistema de notificações independente (Rath, 2018).

Um sistema simples de notificações *push*, fazendo uso da ferramenta Firebase, passa pela leitura de um *token* único de um dispositivo móvel e subscrevê-lo a um determinado tópico. O Firebase, tendo em conta os *tokens* e tópicos associados, tem a capacidade de enviar notificações individuais a cada dispositivo. Tomando como exemplo uma aplicação de notícias desportivas, caso um utilizador subscreva o tópico "atletismo", sempre que lançada uma nova notícia nessa mesma área, uma notificação é enviada para o dispositivo móvel.

Para este projeto, foi pretendida a máxima independência e abstração possível. A existência de um servidor NodeJS e de uma base de dados MongoDB, funcionou como facilitador no desacoplamento. O sistema de subscrição de tópicos e o registo de *tokens* foi implementado na sua totalidade do lado *backend*, apenas utilizando o Firebase para o envio em si das notificações.

De modo a iniciar o desenvolvimento, surgiu a necessidade de criar uma tabela na base de dados, designada de "*token*". Esta tabela, representada na Figura 55, é responsável pela associação de um utilizador a um *token* e a uma série de tópicos de subscrição.

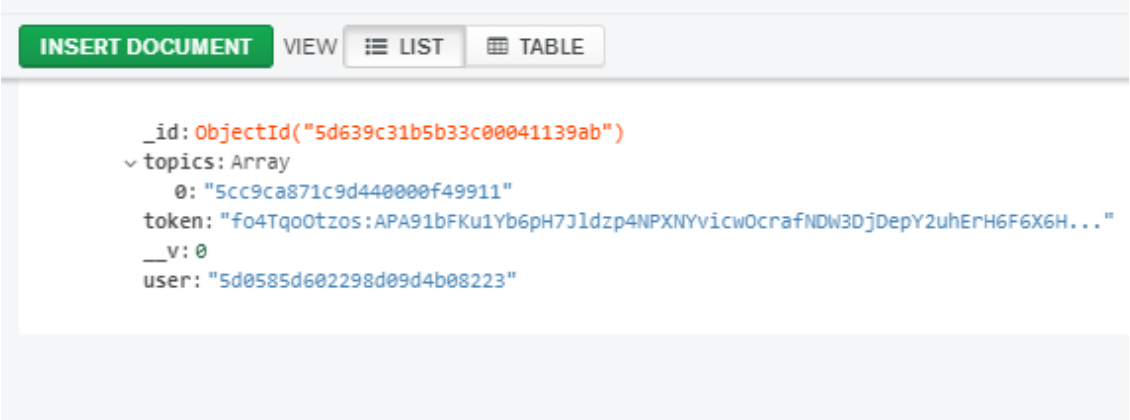


Figura 55 - Tabela *token* da base de dados MongoDB

Seguiu-se a configuração do projeto na plataforma Firebase. Dentro da plataforma *online*, é possível configurar aplicações de diversos tipos, entre as quais as essenciais Android e iOS. Caso se pretenda utilizar notificações *push* nos dois tipos de aplicações, deverão ser configuradas duas aplicações distintas. Para cada instância de aplicação, o Firebase fornece um ficheiro do tipo .json, designado google-services.json, que contém a informação obrigatória à conexão com sucesso entre a aplicação móvel e o serviço. No caso particular das aplicações iOS, é ainda exigido um conjunto de certificações Apple que desbloquearão o uso de notificações internas no dispositivo móvel. Para este projeto em particular, apenas foi necessário configurar uma aplicação Android. O ficheiro .json, fornecido pelo Firebase, foi colocado posteriormente na raiz do código fonte (Figura 56).

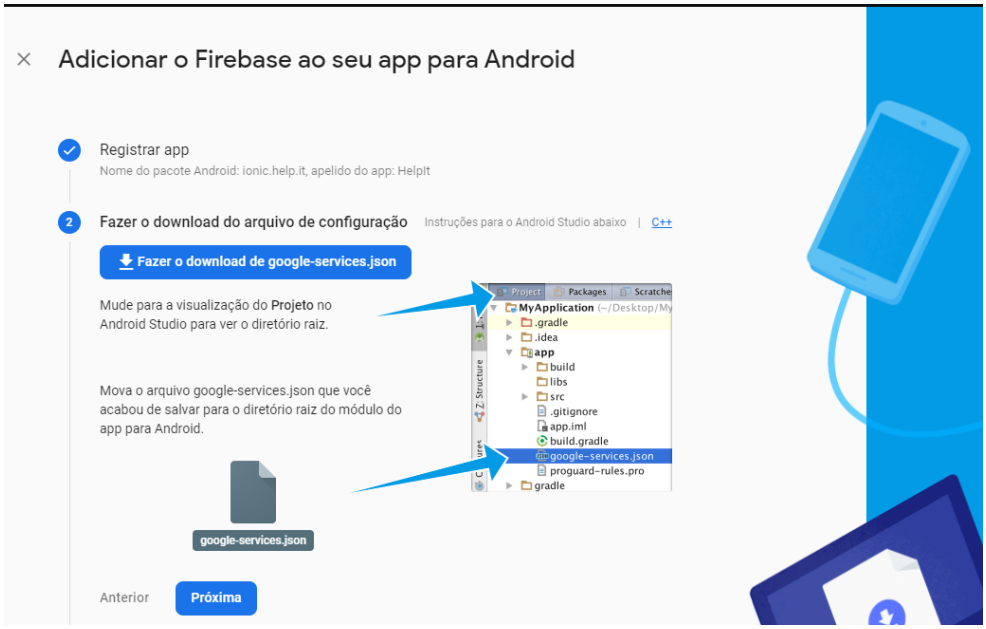


Figura 56 - Janela de configuração aplicação Firebase

Terminada a configuração no serviço de notificações, foi altura de iniciar o desenvolvimento da funcionalidade no lado aplicacional. A *framework* Ionic disponibiliza, de forma nativa, um pacote NPM com os métodos necessários à conexão com o Firebase. A instalação foi possível através da execução sequencial dos comandos representados abaixo.

```
$ ionic cordova plugin add cordova-plugin-fcm-with-dependecy-updated
$ npm install @ionic-native/fcm
```

Figura 57 - Comando de instalação do pacote FCM

Na aplicação Ionic, segui-se a implementação do sistema de leitura de *token*, receção de notificações por parte do Firebase e subscrição de tópicos. Este desenvolvimento foi codificado na classe base de qualquer aplicação Ionic, o ficheiro raiz app.component.ts. É neste ficheiro que é colocada a lógica que se pretende executar em qualquer página. Após a execução do método de registo ou *login*, a aplicação deteta o *token* do dispositivo, através de um método do pacote Firebase, e envia um pedido de registo ao servidor NodeJS. O servidor verifica a existência de um registo para o mesmo utilizador

na base de dados e, caso afirmativo, atualiza pelo novo *token*, caso contrário, insere uma nova linha nos registos. O *token* recebido é ainda guardado na aplicação Ionic para posterior uso (Figura 58).

```
initializeNotificationEvents() {
    this.fcm.getToken().then(newToken => {
      ...
    });

    this.fcm.onTokenRefresh().subscribe(newToken => {
      ...
    });

    this.fcm.onNotification().subscribe(data => {
      if (data.wasTapped) {
        console.log('Received in background');
        ...
      } else {
        console.log('Received in foreground');
        ...
      }
    });
  }

  subscribeNotifications(topic) {
    ...
  }

  unsubscribeNotifications(topic) {
    ...
  }

  sendNotifications(badgeId) {
    let url = environment.baseUrl + '/notification/sendNotification';
    let headers = this.authSvc.createHeaders();
    return this.http.post(url, { topic: badgeId });
  }
```

Figura 58 - Excerto de código do ficheiro app.component.ts

A subscrição de tópicos foi implementada sobre o método de registo de donativo. Sempre que um jogador conclui com sucesso o registo de um novo donativo, a aplicação *frontend* envia um pedido de subscrição de tópico ao servidor. O tópico a registar é um identificador único do desafio semanal escolhido.

A resposta do servidor indica à aplicação Ionic se o objetivo monetário do desafio semanal foi ou não atingido. No caso de resposta positiva, é enviado um novo pedido ao servidor NodeJS, para alertar os utilizadores subscritos ao tópico (ou seja, os

jogadores que atribuíram donativos ao desafio). O servidor solicita à base de dados MongoDB os *tokens* que subscreveram um tópico igual ao identificador único do desafio, e ordena ao serviço Firebase o envio das respetivas notificações *push*. Um excerto do código de envio pode ser observado na Figura 59.

```
await admin.messaging().sendToDevice(destinations, payload, options);
```

Figura 59 - Excerto de código para envio de notificações a partir do servidor NodeJS

Sob a forma de conclusão, cada dispositivo móvel dos doadores, receberá uma nova notificação por parte da aplicação HelpIt, alertando a conquista de um novo item. O jogador poderá abrir o crachá, através de clique na notificação, ou ignorar o aviso e desbloquear mais tarde (Figura 60).

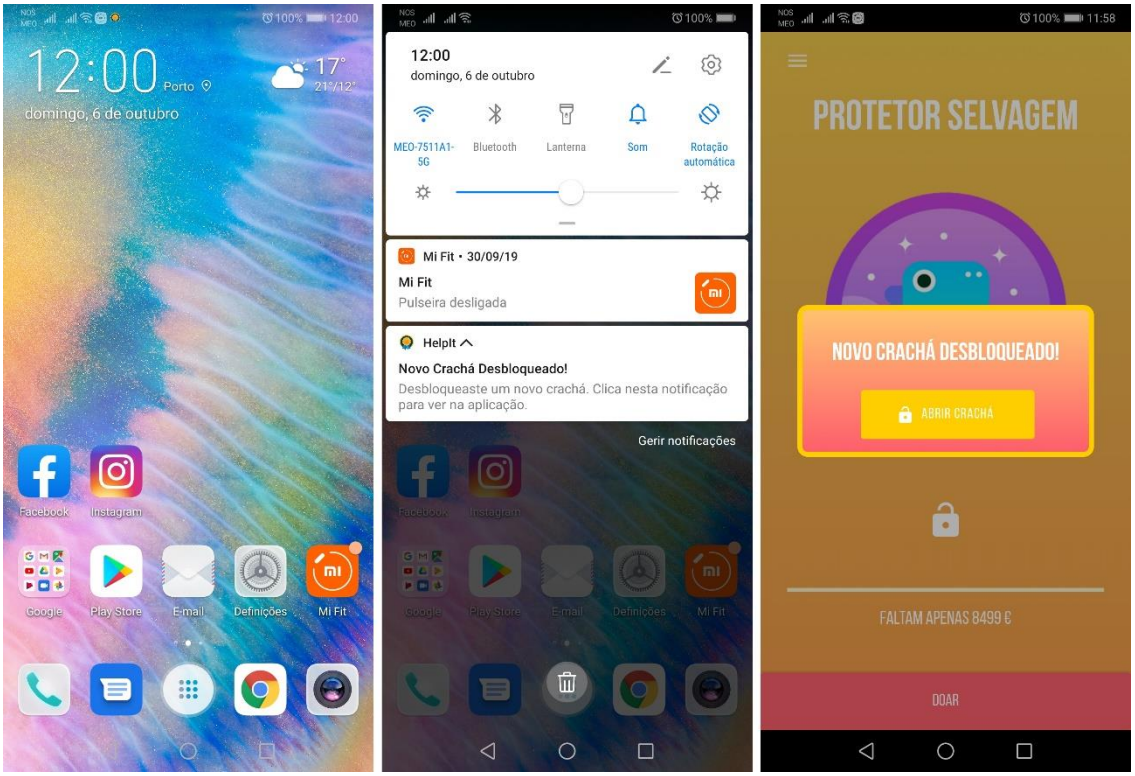


Figura 60 - Receção de nova *push-notification*

4.5.3 Abrir crachás

Após receção de novo crachá, o utilizador é marcado na base de dados com a indicação de que tem um item por abrir na sua coleção. De forma a elevar o grau de atração ao ato de conquistar um novo crachá, iniciou-se a implementação de um mecanismo de abertura de crachás. Em primeiro lugar foi desenhada a página "abrir crachás", na qual o jogador tem a indicação visual de quantos itens tem por abrir (Figura 61).

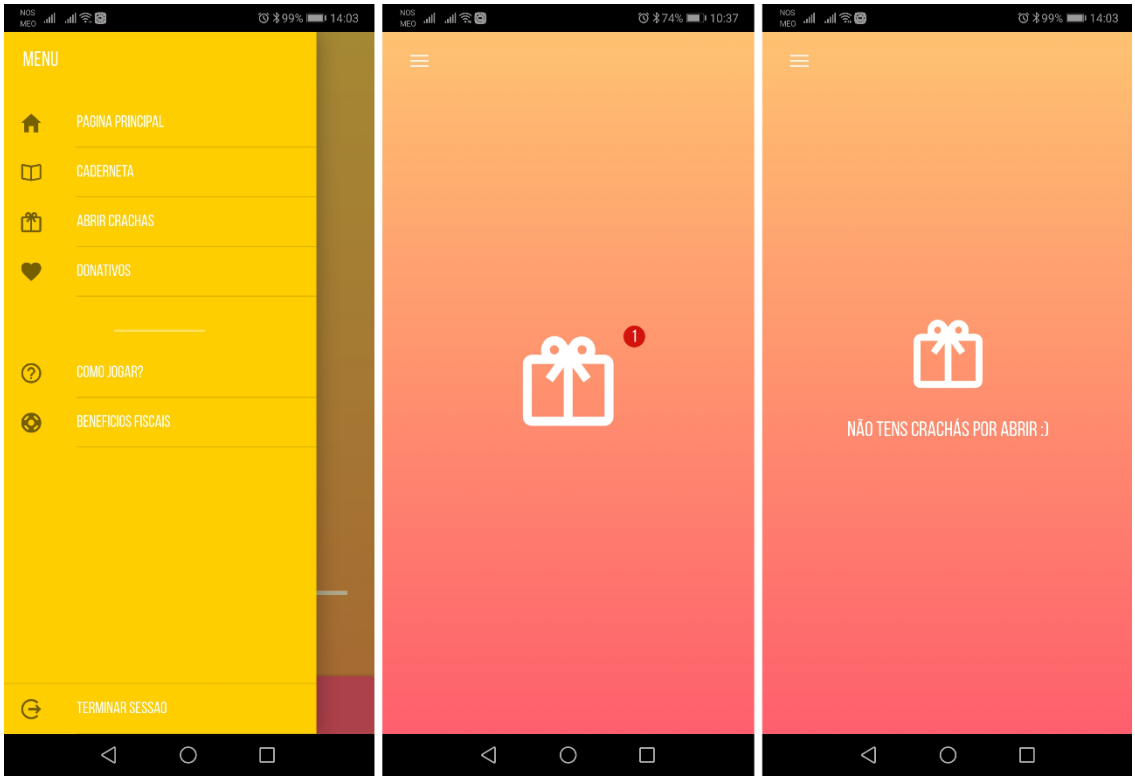


Figura 61 - Página de abertura de crachás

O clique no *icon* de novo item despoleta o mecanismo de abertura de crachá, presente de igual forma na receção de uma notificação *push*. A aplicação apresenta ao utilizador uma nova página, composta por um conjunto de animações e grafismo, que visam a captação da atenção do jogador e o aumento da sensação de satisfação na conquista (Figura 62). No término das animações, a aplicação envia um pedido ao servidor NodeJS para registar o novo crachá como item aberto, no perfil do utilizador, decrementando em uma unidade o valor numérico da página representada na Figura 61.

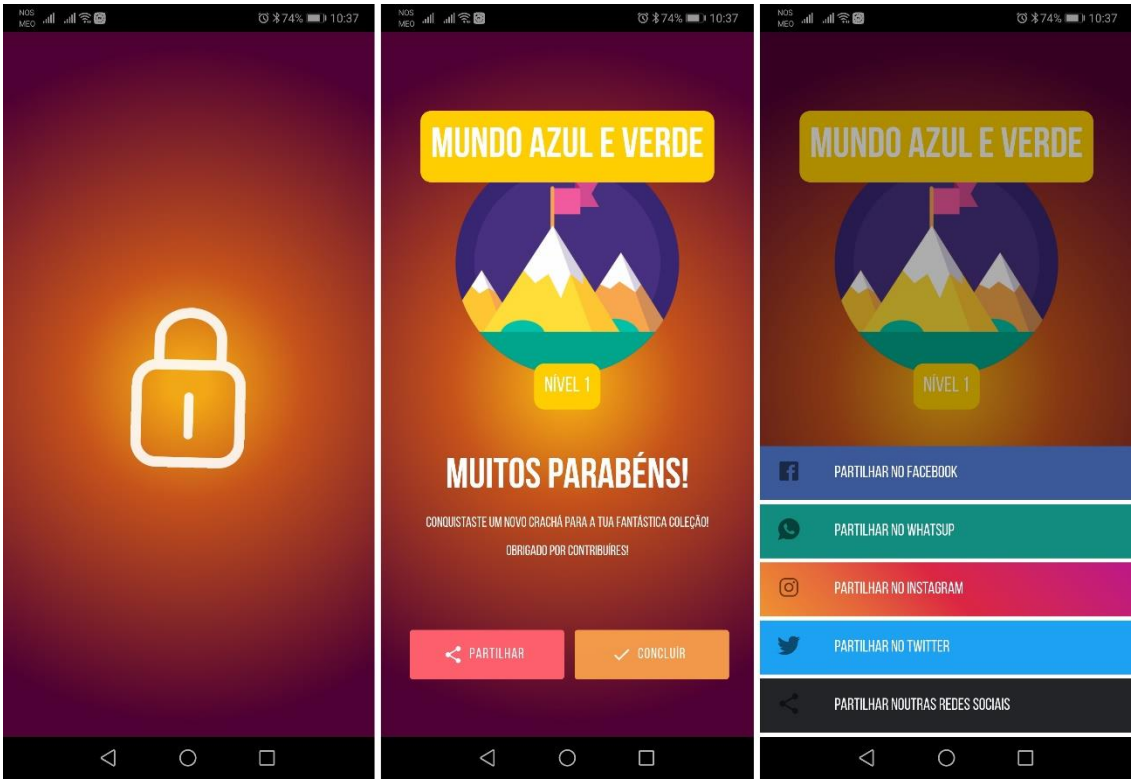


Figura 62 - Página de desbloqueio de crachá

4.6 Coleção de crachás

Uma outra funcionalidade, também ela essencial ao sucesso do projeto, é a permissão ao utilizador em visualizar, sob a forma de caderneta, os crachás conquistados. Esta implementação, apesar de relativamente simples, foi bastante importante na definição da dinâmica aplicacional.

O desenvolvimento teve início com a criação de uma nova página Ionic. Aquando da inicialização do componente, a aplicação *frontend* solicita ao servidor uma lista de todos os crachás. Este recolhe os dados da base de dados MongoDB, e retorna uma lista de registos com todos os crachás disponíveis na aplicação, bem como de uma *flag* indicativa da captura ou não captura do respetivo item por parte do jogador.

Uma vez na página Ionic e já com a lista de registos, a aplicação constrói o *layout* da caderneta, apresentando os crachás em filas de dois. Caso o utilizador ainda não tenha

capturado um determinado crachá, um símbolo interrogativo apropriado é apresentado. Se o jogador tiver conquistado o item, é-lhe apresentado no ecrã a imagem real do crachá.

Na visão de caderneta, o utilizador está habilitado a ver uma página de detalhes de um crachá através de um clique na imagem, caso o tenha conquistado anteriormente. No detalhe é possível de observar o título, imagem e breve descrição do movimento associado, bem como partilhar o feito através das redes sociais. A Figura 63 ilustra os respetivos ecrãs aplicacionais.

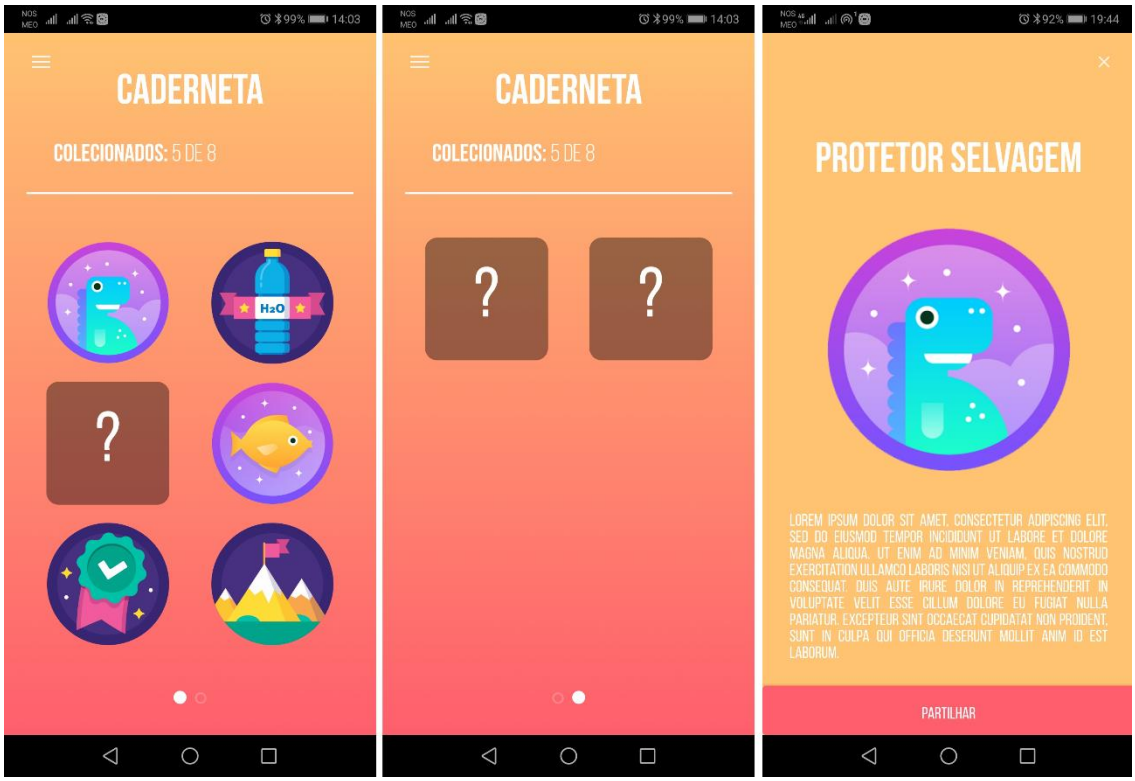


Figura 63 - Ecrã de coleção e detalhe de crachá

4.6.1 Partilha redes sociais

A necessidade de partilha de conquistas nas redes sociais foi algo definido na fase de estudo inicial da aplicação. Fornecer ao jogador a possibilidade de poder partilhar, numa rede social à escolha, uma determinada conquista, é tentar elevar o índice de retenção dos utilizadores.

Por se tratar de uma função partilhada entre páginas (detalhe e desbloqueamento de crachá), foi decidido implementar um componente partilhado. Em Ionic e Angular, a capacidade modular de criar e usar diferentes componentes dentro de qualquer página, facilitou o processo. A criação de um novo componente, bem como de um serviço especializado na partilha, foi realizada executando o comando abaixo.

```
$ ionic generate component components/share-badge

$ ionic g service services/share-badge
```

Figura 64 - Comando de geração de componente e serviço de partilha de crachá

A página parcial, apenas apresentará um botão de partilha. A ação de clique neste componente, despoleta a abertura de um menu do lado inferior da aplicação, com as opções de partilha nas diversas redes sociais. Após selecionar uma opção de partilha, o recém-criado serviço utiliza os métodos presentes num novo pacote NPM, para finalizar a ação. O pacote social-sharing foi instalado através do comando abaixo.

```
$ ionic cordova plugin add cordova-plugin-x-socialsharing

$ npm install --save @ionic-native/social-sharing
```

Figura 65 - Comando de instalação do pacote social-sharing

Após término da etapa de *design* aplicacional, a opção de partilha apresenta-se conforme ilustrada na Figura 66.

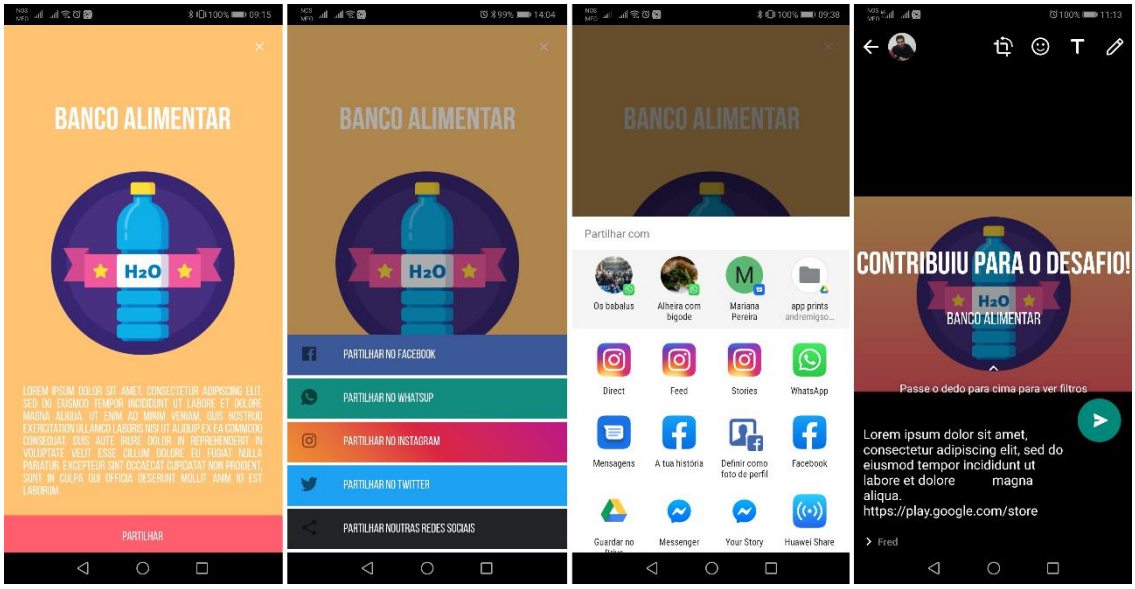


Figura 66 - Ecrã de partilha de crachá

4.7 Animações e grafismo

Componente chave na atração e retenção de utilizadores, em qualquer aplicação baseada no entretenimento e jogabilidade, é a interface gráfica (ou *user interface*). Como tal, neste projeto foi dedicado grande parte do tempo de desenvolvimento no planeamento e implementação de uma boa interação com o utilizador.

Por motivos de otimização e rentabilização de tempo, alguns recursos visuais foram adquiridos de páginas *web* como o Dribbble e o Pinterest. Os logótipos representativos dos crachás são da autoria de dois artistas gráficos registados no site Dribbble, referenciados em (Litvinov, 2017) e (Reen, 2017). O logótipo da aplicação foi obtido por outro artista gráfico, também ele registado no mesmo *website* e referenciado em (Warren, 2017). Todos estes recursos visuais, da autoria dos diferentes artistas, foram utilizados como parte do meio de demonstração de uma boa interface gráfica. Ainda com o auxílio destes *assets*, foi desenhado, usando a ferramenta Photoshop, o chamado ecrã de *splash screen*, representado visualmente na Figura 67.

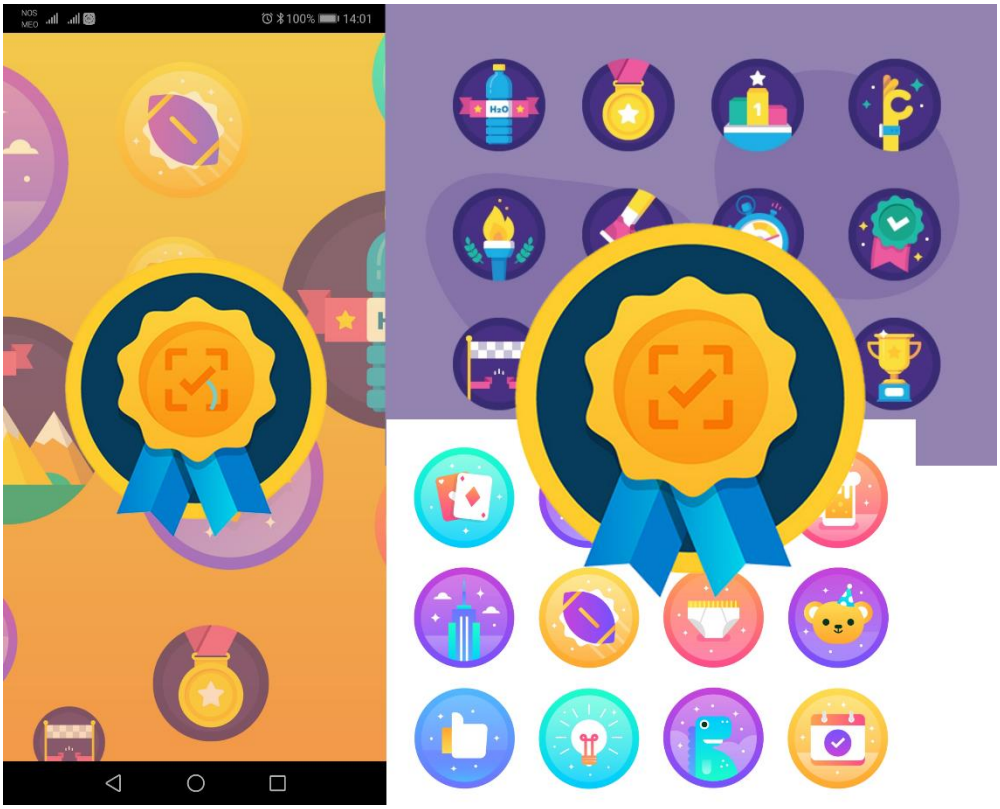


Figura 67 - Ecrã *splash screen* e recursos gráficos da aplicação

Após seleção e criação de todos os recursos gráficos necessários ao projeto, surgiu a implementação da interatividade aplicacional. Uma boa aplicação que visa o entretenimento dos seus utilizadores, aloca grande importância às animações nos diferentes ecrãs. Para a aplicação HelpIt não foi diferente, tendo sido desenvolvido todo um conjunto de animações e interações com o utilizador, de forma a elevar o nível de atração e retenção. Este processo foi concluído com o apoio de um novo pacote NPM *open-source*, designado por Animate.css (Eden, 2019). Para além de fornecer um vasto conjunto de animações pré-definidas, este *plugin* disponibiliza ainda mecanismos de definição de velocidade e tempo de início de execução das animações, permitindo desta forma a construção de animações sequenciais (Figura 68).

```
<h1 class="animated infinite bounce delay-2s">Example</h1>
```

Figura 68 - Exemplo de uso do *plugin* animate.css

Em forma de conclusão, esta etapa englobou ainda uma fase de decisão acerca do local de armazenamento dos recursos gráficos aplicacionais. Para evitar o exponencial aumento do tamanho da aplicação, foi decidido remover os logótipos e imagens do código fonte, e inseri-los num serviço de hospedagem *online*. O serviço escolhido foi o Flickr, no qual foi possível criar um novo álbum de recursos gráficos. Este álbum, apesar de privado, pode ser acedido através de um *link* direto a um recurso em particular. O código aplicacional, viu substituídas as referências a *assets* locais pelo respetivo endereço *web* do serviço hospedeiro.

4.8 Processo de pagamento

Tal como já mencionado em capítulos anteriores, o passo final no processo de registo de donativos não foi implementado neste projeto, no entanto, foi realizado um estudo teórico. Por motivos de segurança de dados e por se tratar de uma aplicação em fase de testes, optou-se por simulador o registo com sucesso de um donativo.

O desenvolvimento do processo de pagamento às instituições e associações alvo, poderia seguir diferentes estratégias. A estratégia mais simples e mais correta moral e eticamente, seria a transferência de um pagamento da aplicação diretamente para a conta bancária da entidade. Este seria o processo que levantaria menos ceticismo quanto à veracidade do projeto. Apesar das vantagens, o método levantaria um problema maior, visto as diferentes organizações disponibilizarem diferentes meios de pagamento. A disponibilização de meios de pagamento distintos entre desafios, levaria a uma quebra na igual probabilidade de escolha do jogador. Por exemplo, um desafio semanal que não permitisse o pagamento através de MBWay, poderia ser rejeitado pelo possível doador.

Alternativamente, outro processo que melhor se adaptaria às necessidades descritas, seria o registo dos donativos numa conta bancária oficial da aplicação HelpIt e posterior transferência para a conta institucional. Este desenvolvimento obrigaria à criação de uma conta bancária, dedicada apenas à receção e transferência de donativos. De forma

a evitar despesas adicionais e erros fiscais, a empresa HelpIt deveria ser devidamente registada nas Finanças. Este processo capacitaria a aplicação na disponibilização dos mesmos meios de pagamento a todos os desafios. Seria ainda necessário um mecanismo, manual ou automático, de transferência semanal do valor angariado para cada desafio.

4.8.1 Paypal

O primeiro método, e talvez o mais simples de implementar, é o pagamento por Paypal. A *framework* Ionic, disponibiliza mais uma vez um pacote nativo para integração do serviço na aplicação. O pacote ionic-native/paypal é instalado através da execução do comando abaixo (Ionic Framework | PayPal, 2019).

```
$ ionic cordova plugin add com.paypal.cordova.mobilesdk

$ npm install @ionic-native/paypal
```

Figura 69 - Comando de instalação do pacote ionic-native/paypal

Após instalação do novo pacote e criação de uma conta Paypal, o registo de pagamentos com sucesso poderá ser obtido através do uso do código exemplificado na Figura 70. A recém-criada conta Paypal deverá estar associada à conta bancária do projeto HelpIt.

```
this.payPal.init({
  PayPalEnvironmentProduction: 'YOUR_PRODUCTION_CLIENT_ID',
  PayPalEnvironmentSandbox: 'YOUR_SANDBOX_CLIENT_ID'
}).then(() => {
  // Environments: PayPalEnvironmentNoNetwork, PayPalEnvironmentSandbox, PayPalEnvironmentProduction
  this.payPal.prepareToRender('PayPalEnvironmentSandbox', new PayPalConfiguration({
    // Only needed if you get an "Internal Service Error" after PayPal login!
    //payPalShippingAddressOption: 2 // PayPalShippingAddressOptionPayPal
  })).then(() => {
    let payment = new PayPalPayment('3.33', 'USD', 'Description', 'sale');
    this.payPal.renderSinglePaymentUI(payment).then(() => {
      // Successfully paid
      (...)
}, () => {
  // Error in initialization, maybe PayPal isn't supported or something else
});
```

Figura 70 - Excerto do código de implementação de pagamentos Paypal

4.8.2 Stripe

O segundo meio de pagamento passaria pelo uso de cartões de débito/crédito ou referências multibanco. Existe atualmente um serviço, paralelo ao Paypal, que permite a configuração de pagamentos por cartão ou multibanco, designado por Stripe.

Tal como no caso anterior, o Ionic fornece um *plugin* de apoio à integração com as suas aplicações, o qual se vê instalado executando o comando da Figura 71 (Ionic Framework | Stripe, 2017).

```
$ ionic cordova plugin add cordova-plugin-stripe

$ npm install @ionic-native/stripe
```

Figura 71 - Comando de instalação do pacote ionic-native/stripe

Após instalação, o desenvolvimento dos métodos de pagamento poderá iniciar. Uma nova conta deverá ser criada e configurada na plataforma Stripe, associando os dados fiscais e conta bancária de destino. O tipo de pagamento por cartão de débito/crédito, encontra-se disponível nativamente, no entanto, para o caso das referências multibanco, deverá ser ativada uma opção especial durante a configuração.

Na Figura 72 é possível visualizar um excerto do código para criação do tipo de pagamento cartão de débito/crédito, dentro da *framework* Ionic.

```
this.stripe.setPublishableKey('my_publishable_key');

let card = {
 number: '4242424242424242',
 expMonth: 12,
 expYear: 2020,
 cvc: '220'
}

this.stripe.createCardToken(card)
   .then(token => console.log(token.id))
   .catch(error => console.error(error));
```

Figura 72 - Excerto de código para pagamento com cartão de débito/crédito

Para configurar a geração de referências multibanco, o código de implementação seria ligeiramente diferente (Stripe | Multibanco Payments with Sources, 2019). A Figura 73 representa a implementação deste meio de pagamento.

```
stripe.createSource({
  type: 'multibanco',
  amount: 1099,
  currency: 'eur',
  owner: {
    name: 'Jenny Rosen',
    email: 'jenny.rosen@example.com',
  },
  redirect: {
    return_url: 'https://shop.example.com/crtA6B28E1',
  },
}).then(function(result) {
  // handle result.error or result.source
});
```

Figura 73 - Excerto de código para pagamento com multibanco

4.8.3 MBWay

O terceiro e último método a implementar neste projeto, seria o pagamento por MBWay. Este seria o desenvolvimento mais complexo, uma vez não existir nenhum *plugin* nativo, da ferramenta Ionic, para auxiliar na integração. O processo de implementação passaria pela criação de uma conta de desenvolvedor, na plataforma MBWay, e seguir as instruções de configuração para aplicações móveis. A dificuldade

vê-se acrescida pelo facto de os diferentes sistemas operativos exigirem diferentes configurações. Para configurar o sistema MBWay numa aplicação Android, é necessária a inclusão de um ficheiro do tipo .jar, enquanto que para iOS, é necessária a inserção de uma nova pasta, na raiz do projeto, com o SDK (Software Development Kit) da MBWay (MBWay - Developers | SOFTWARE DEVELOPMENT KIT PARA OPERATIVA IN-APP, 2019).

Esta distinção atrasaria o processo de implementação de uma aplicação, pela remoção de uma das maiores vantagens do Ionic, o desenvolvimento abstraído do sistema operativo.

4.9 Deployment e distribuição

A etapa final na implementação de qualquer aplicação móvel é a fase de publicação e distribuição. Atualmente, no mundo das aplicações *mobile*, existem duas plataformas de distribuição oficiais, uma destinada ao sistema operativo Android e outra direcionada aos desenvolvedores de aplicações iOS. Foi lançada em tempos uma outra plataforma, focada nas aplicações Windows, mas que viu o seu uso decair com o passar do tempo.

4.9.1 Play store (Android)

A plataforma Play store, é serviço de publicação e distribuição de aplicações desenvolvidas para o sistema operativo Android. É um serviço que oferece uma grande liberdade de implementação, comparativamente com a rival App store.

Para publicar uma aplicação na loja Android é necessária a criação de uma conta de desenvolvimento. Esta conta permite a distribuição de inúmeras aplicações, apenas requer o pagamento de $25 para ativação de uma licença vitalícia.

O primeiro passo, partilhado com a distribuição de aplicações iOS, é a geração de uma *build* de produção ou lançamento (*release*). A execução do código representado na

FIGURA X, compila o código fonte da aplicação, remove todo o "lixo", como comentários e linhas em branco, e cria um novo ficheiro do tipo APK (Ionic Framework | Publishing - Android Play Store, 2019).

```
$ ionic cordova build android --prod --release
```

Figura 74 - Comando de compilação e geração de *build* de produção

O recém-criado ficheiro, apesar de compilado e otimizado, não se encontra ainda apto para publicação, faltando uma assinatura de autenticidade aplicacional. O ficheiro do tipo .keystore é a assinatura de uma aplicação, que verifica a integridade e permite a sua posterior atualização na Play store. Este ficheiro, pela sua importância, deverá ser guardado num local seguro, fora do repositório *online* onde está armazenado o código fonte do projeto.

Para gerar uma nova assinatura, executamos o código abaixo.

```
$ keytool -genkey -v -keystore my-release-key.keystore -alias alias_name -keyalg RSA -keysize 2048 -
validity 10000
```

Figura 75 - Comando de geração de nova assinatura

O passo seguinte foi assinar o ficheiro *build*, gerado anteriormente, com a nova assinatura. Este passo transforma o ficheiro compilado num ficheiro aplicacional compilado e autenticado (Figura 76).

```
$ jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore my-release-key.keystore helpit-
release-unsigned.apk alias_name
```

Figura 76 - Comando de assinatura da aplicação

O passo final, na geração de uma aplicação para publicação e distribuição, é a execução do comando abaixo, responsável pelo otimização e compilação finais, e criação de um novo ficheiro APK pronto a publicar na loja virtual.

```
$ zipalign -v 4 helpit-release-unsigned.apk HelpIt.apk
```

Figura 77 - Comando de otimização final na geração da aplicação de produção

Uma vez obtido o ficheiro final, é possível submeter a aplicação na Play store. Após *login* na conta de desenvolvimento, basta selecionar a opção "Nova Aplicação" e preencher os campos de título e descrição, e carregar alguns *screenshots* representativos das funcionalidades disponíveis na aplicação. Se tudo se encontrar em conformidade, o projeto ficará oficialmente disponível para *download*.

4.9.2 App store (Apple)

Tal como na submissão para a loja virtual Android, a publicação de uma aplicação na plataforma App store exige a validação de alguns requisitos (Ionic Framework | Publishing - Publishing to app store, 2019).

É necessária a criação de uma conta de desenvolvimento oficial Apple que, ao contrário da plataforma Android, requer o pagamento de $99 anuais. A entidade Apple apenas disponibiliza os acessos e permissões às *builds* aos seus dispositivos oficiais, como Macbook, iPhone e iPad.

O primeiro passo, efetuado a partir de uma máquina Apple, é a execução do comando de compilação e geração de uma *build* de produção da aplicação (Figura 78).

```
$ ionic cordova platform add ios

$ ionic cordova build ios --prod
```

Figura 78 - Comando de compilação e geração de nova *build*

Um novo ficheiro do tipo .xcworkspace é criado dentro da pasta de *builds* do projeto. Este ficheiro é lido pelo editor Xcode, presente em qualquer dispositivo Macbook. A ferramenta tem a capacidade de lidar automaticamente com os certificados de

desenvolvimento oficiais da Apple, bem como de compilar e submeter uma versão final da aplicação para a loja virtual.

O universo de certificações Apple é algo complexo, que apenas foi abordado superficialmente para este projeto. Na sua base, existem dois tipos de certificados, um de desenvolvimento e outro de distribuição. Uma aplicação em fase de desenvolvimento que tenha em vista a sua presença prematura, para testes, na loja oficial App store, necessitaria de uma certificação do tipo desenvolvimento. Já uma aplicação finalizada, com o objetivo de ser comercializada, precisaria de um certificado do tipo distribuição. Dentro de cada um destes certificados, aplicam-se ainda algumas nuances, quanto a permissões adicionais para dados biométricos, acesso à câmara do dispositivo e possibilidade de realizar compras virtuais.

Após gerar corretamente os certificados Apple, e dentro do editor Xcode, selecionamos a opção `Product > Archive` para compilar e otimizar uma nova *build* da aplicação. Para terminar a submissão, o programador deve inserir um título, descrição e *screenshots* da aplicação. O novo projeto apenas se torna disponível para *download*, assim que obtiver uma validação oficial da Apple em como a aplicação obedece a todas as regras exigidas.

5 Avaliação da solução

O método de avaliação do projeto sofreu alterações desde a fase de estudo inicial ao término do desenvolvimento da solução. A modificação no tipo de avaliação surgiu pela não implementação do registo oficial de pagamentos.

A primeira abordagem era composta por dois testes de avaliação, um teste à quantia acumulada no primeiro mês, após lançamento da aplicação, e um questionário de satisfação quanto ao conceito e usabilidade aplicacionais. No final do primeiro mês, seria angariada toda a informação registada quanto a novos utilizadores, número de desafios concluídos e quantia acumulada nas quatro semanas de estudo. Após análise dos resultados, e num cenário de continuidade, seria ainda redigido um plano de promoção e *marketing*, com vista no aumento dos participantes e quantia de donativos acumulados em cada desafio.

Como foi optada, na fase de desenvolvimento, pela não implementação do registo de pagamento, o cenário de testes acima não possível de concretizar. No entanto, foi reaproveitado o questionário de satisfação da aplicação.

O questionário conta com quinze perguntas, todas elas de resposta opcional (ver em anexo). Entre as diversas questões, foi possível agrupar algumas das mais importantes para análise inicial do sucesso aplicacional, e outras menos relevantes numa primeira fase. Para responder ao teste de avaliação, foram selecionados dez participantes, os quais receberam acesso *beta* à aplicação HelpIt. Os utilizadores escolhidos responderam a todas as perguntas, com exceção do endereço de email e uma data de nascimento. Todos os participantes avaliaram a aplicação no mês de Setembro, pelo que a análise do primeiro gráfico se torna irrelevante (Figura 79).

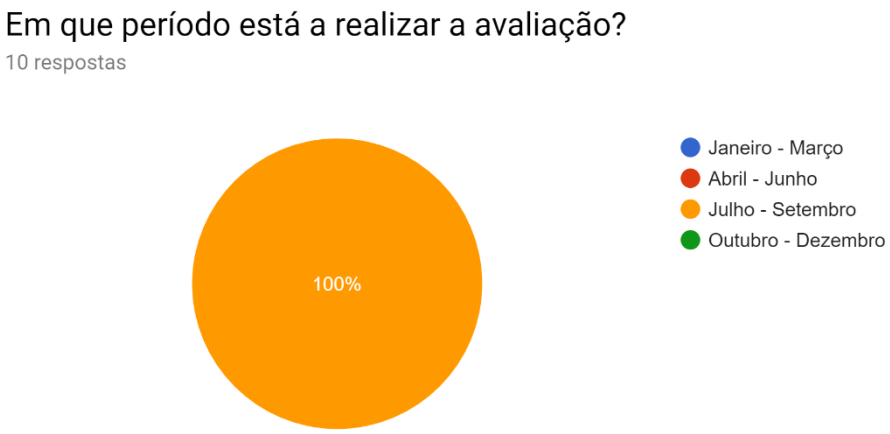


Figura 79 - Gráfico trimestre de avaliação

A primeira análise relevante para a avaliação da solução teve lugar com a questão da fácil compreensão acerca do objetivo do projeto. Numa escala de 1 a 10, os participantes avaliaram maioritariamente no nível 6 e 7 (Figura 80). A análise dos resultados indica que, sem explicação prévia do conceito do projeto, os participantes demoraram mais algum tempo a entender o objetivo. Uma solução possível seria a implementação de uma página inicial de *tutorial*, após registo de nova conta.

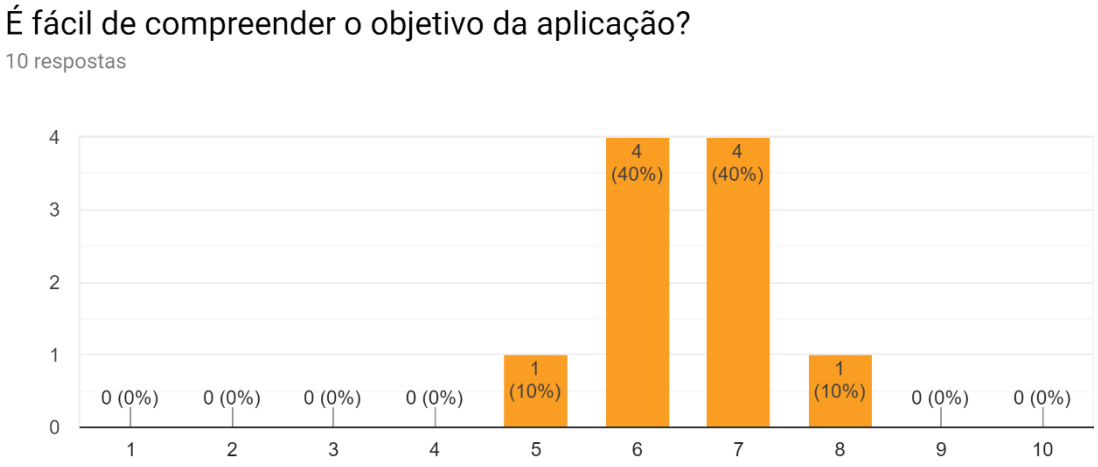


Figura 80 - Gráfico de facilidade de compreensão do objetivo aplicacional

Após compreensão do objetivo, os utilizadores de teste avaliaram com pontuação alta o interesse quanto ao conceito do projeto, indicando ainda na questão seguinte o alto grau de probabilidade de se tornarem jogadores numa versão final da aplicação. Os gráficos avaliados encontram-se representados na Figura 81 e Figura 82 respetivamente.

Figura 81 - Gráfico de avaliação do interesse conceitual

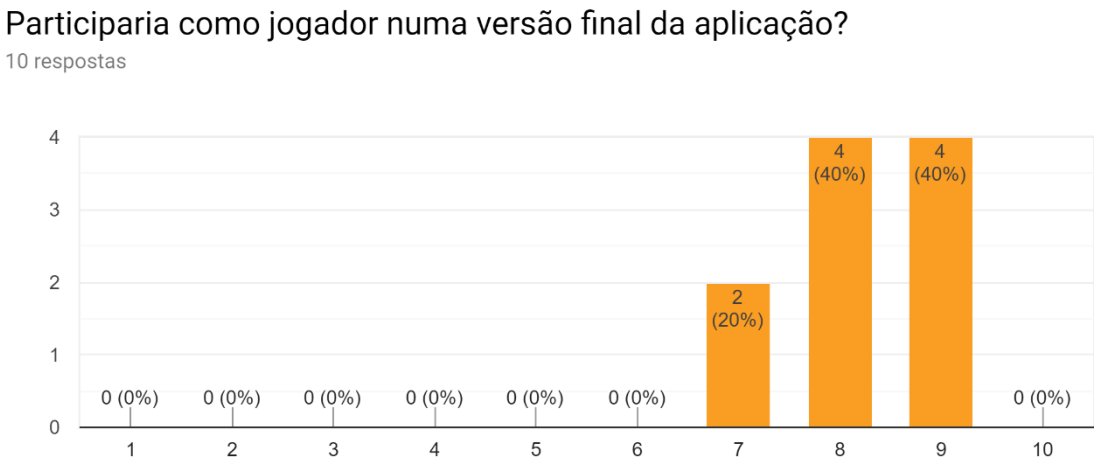


Figura 82 - Gráfico de avaliação da participação em versão final

As perguntas que se seguiram estão ligadas à usabilidade e aspeto gráfico da aplicação, algo a que foi atribuído grande importância aquando do desenvolvimento do projeto. Os participantes concederam pontuação alta ao aspeto gráfico estimulante e simplicidade na usabilidade da aplicação, tendo obtido valores entre 9 e 10 na escala já mencionada (Figura 83 e Figura 84).

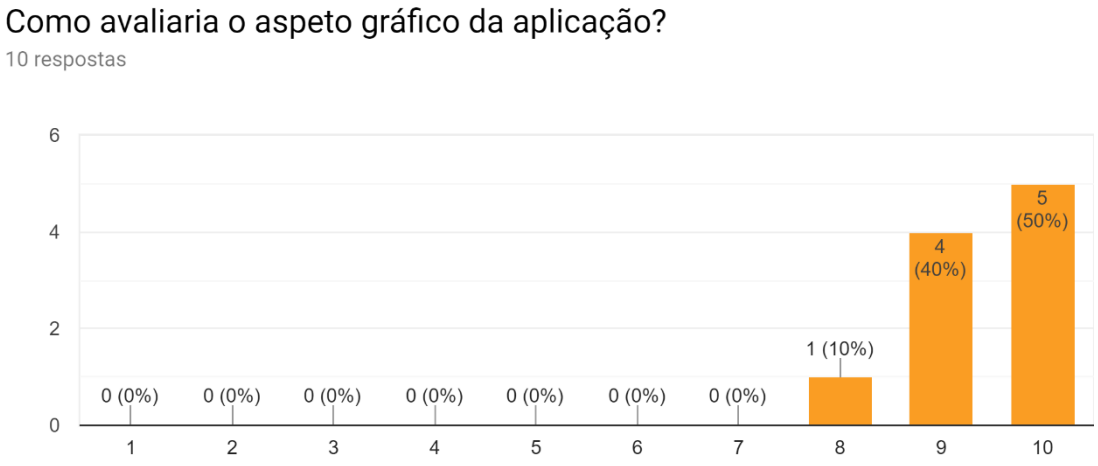


Figura 83 - Gráfico de avaliação do aspeto gráfico da aplicação

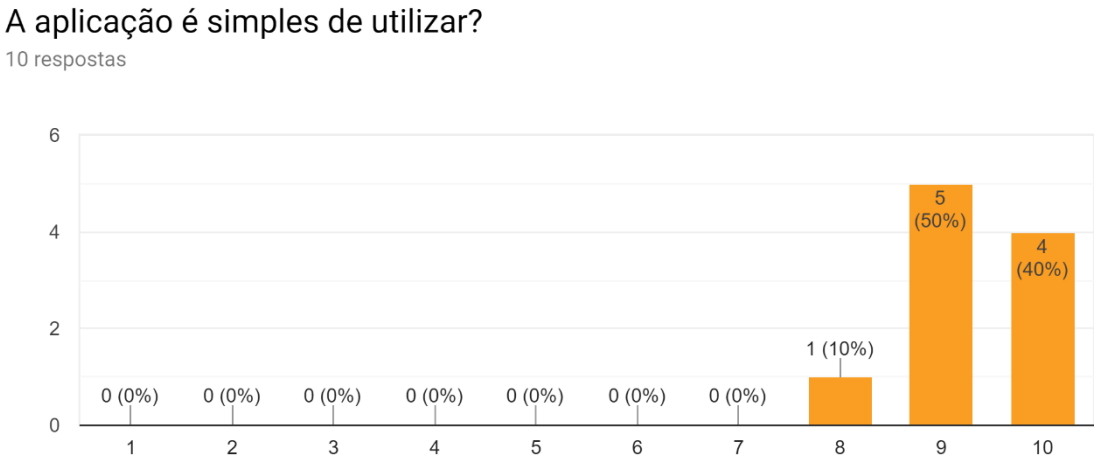


Figura 84 - Gráfico de avaliação da usabilidade aplicacional

Duas das questões seguintes foram apenas colocadas como forma de avaliação da seriedade dos participantes no preenchimento do questionário. As perguntas questionavam quanto à facilidade em encontrar a aplicação no mercado e quanto à compatibilidade com utilizadores com daltonismo. Pelo facto de a aplicação não ter sido disponibilizada no mercado aplicacional e da ausência de participantes daltónicos na avaliação, os resultados são os esperados. As respostas à primeira questão tiveram cotação de 1 valor ("Não encontrei"), e as respostas quanto à compatibilidade com daltonismo obtiveram resposta "Não sei". Os gráficos, apesar de não relevantes, estão representados nas figuras abaixo.

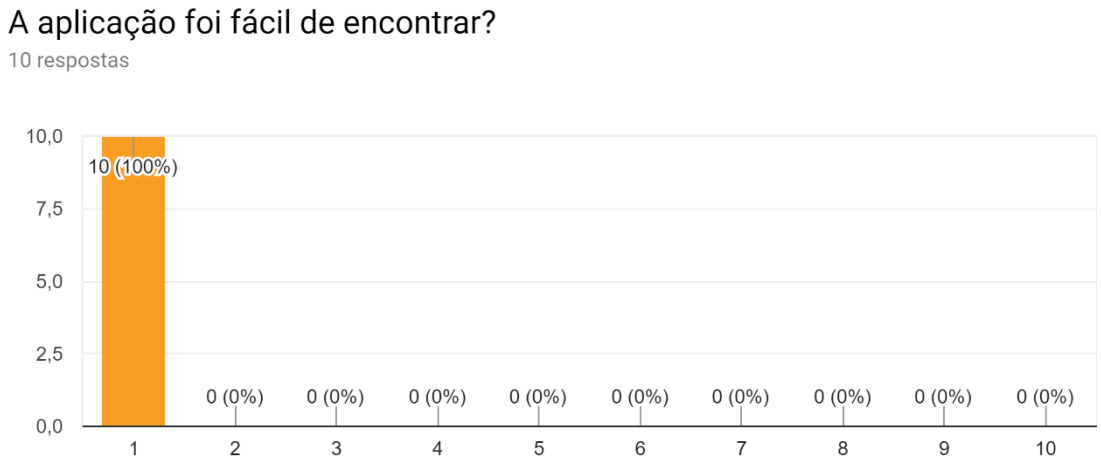


Figura 85 - Gráfico de filtragem de questionários inválidos (facilidade de obtenção da aplicação)

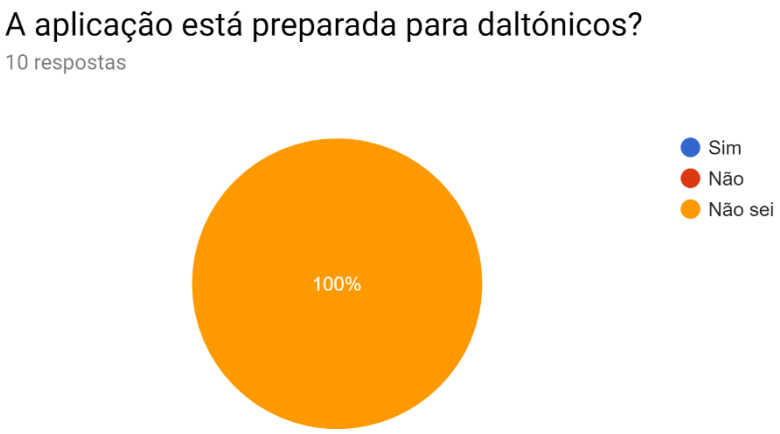


Figura 86 - Gráfico de filtragem de questionários inválidos (compatibilidade com daltonismo)

O questionário retorna ao aspeto visual da aplicação, numa tentativa de consolidar a sua importância na retenção dos utilizadores. A essencialidade das animações e grafismo aplicacionais na usabilidade do projeto, obteve uma pontuação de 9 valores pela maioria dos participantes (Figura 87).

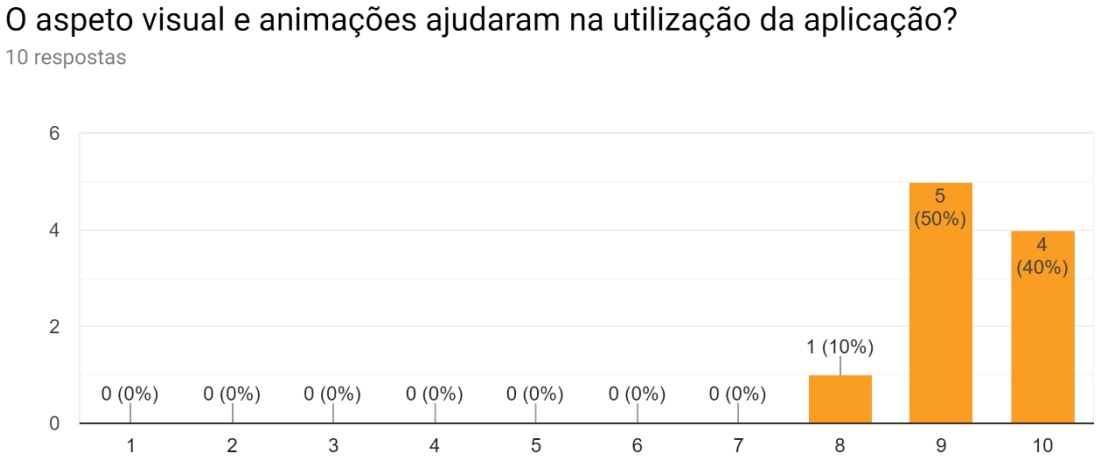


Figura 87 - Gráfico de avaliação da importância do grafismo na usabilidade

Seguiu-se uma questão essencial em qualquer projeto que visa interação social, o respeito moral e ético quanto à componente racial, política e religiosa. A pergunta obteve maioritariamente respostas positivas (60%), no entanto, cerca de 40% dos

participantes afirmaram não ter informação suficiente para responder à questão. Este tipo de resposta deveu-se ao facto de os inquiridos tentarem responder pela globalidade em vez de perspetiva pessoal.

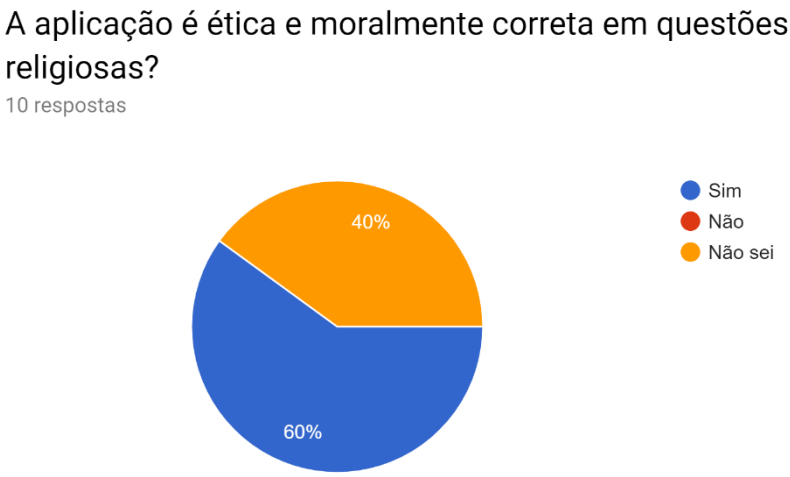


Figura 88 - Gráfico de avaliação à componente racial, política e religiosa

A décima quinta e última pergunta, serviu de consolidação quanto ao apresso geral dos inquiridos pelo projeto. Tal como em questões anteriores, a aplicação obteve uma cotação de 8 valores na sua maioria (Figura 89).

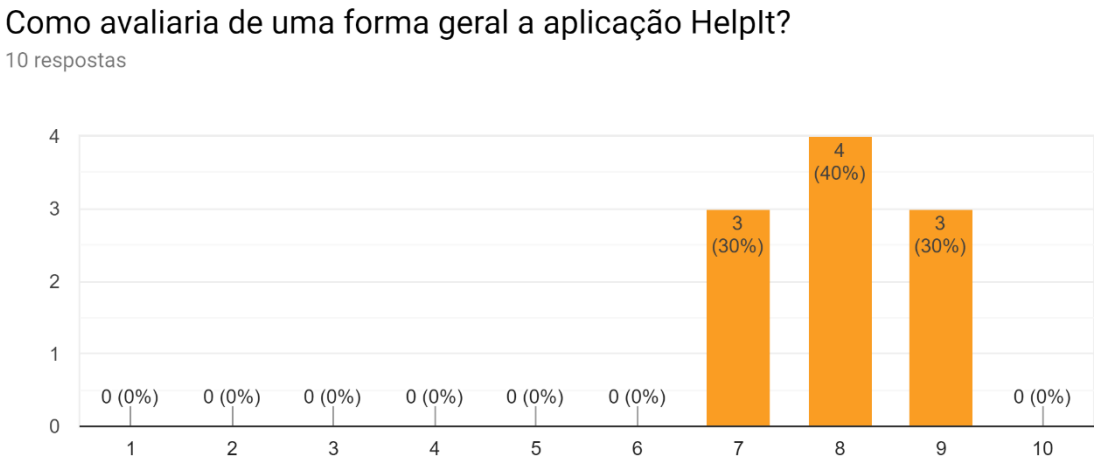


Figura 89 - Gráfico de avaliação geral da aplicação

Apesar de um grupo bastante restrito e reduzido, foi possível de apurar os pontos fortes e fracos da implementação atual do projeto, bem como de delinear um plano estratégico para futuros desenvolvimentos, de forma a atrair e reter mais utilizadores a este movimento de apoio social.

6 Conclusão

O desenvolvimento desta aplicação teve, como todos os projetos, alguns altos e baixos, contudo, a grande maioria dos objetivos propostos foram cumpridos com sucesso. O conceito inicialmente definido manteve-se constante, apenas sofreram alterações as tecnologias e arquitetura planeadas na fase de *design* da solução. Após o término do desenvolvimento do projeto documentado, foi possível apurar os desafios e problemas encontrados, bem como um conjunto de desenvolvimentos futuros, essenciais numa versão finalizada da aplicação HelpIt.

6.1 Desafios e problemas encontrados

Começando pelos desafios e problemas encontrados, a lista não é extensa, aliás, a implementação do projeto ocorreu na sua maioria dentro dos prazos e sem grandes percalços no seu percurso.

O problema mais evidente foi a decisão da não implementação do registo oficial de pagamentos. Esta funcionalidade viu-se impossibilitada, nesta versão da aplicação, por motivos de segurança de dados e falta de licenças e autorizações. Uma vez não sendo uma versão final, a aplicação não poderia nunca permitir o registo de transferências monetárias “reais”. Para implementar este sistema, teriam de ser realizados pedidos de autorização às diversas organizações e instituições alvo, e ainda um pedido de licença à entidade Finanças.

O segundo ponto chave nas mudanças de percurso, já mencionado algumas vezes, esteve relacionado com a não publicação da aplicação nas lojas Android e iOS, conforme inicialmente planeado. Por se tratar de um projeto em fase de testes e com implicações sérias a nível de segurança (pagamentos), optou-se pela distribuição manual da aplicação para os diferentes *beta testers* selecionados. Após obter uma versão estável, o HelpIt poderia ser disponibilizado na loja oficial Android, no entanto, para o sistema

operativo iOS, algumas validações e desenvolvimentos extra teriam de ser tidos em conta.

6.2 Desenvolvimentos futuros

Utilizando o conhecimento adquirido com os diversos desafios encontrados, foi possível desenhar uma estratégia para uma futura versão do projeto HelpIt. Esta estratégia tem por objetivo a geração de uma aplicação finalizada e publicada nas lojas de aplicações móveis.

Conforme já mencionado, parte essencial no sucesso do projeto, é a implementação do registo de pagamento às organizações alvo nos desafios semanais. Optando pela segunda via de desenvolvimento, documentada no capítulo 4.8, seria necessária a criação de uma conta bancária destinada à coleta e distribuição de fundos. Em seguida, e como forma de conclusão da funcionalidade, viria a integração da aplicação com os serviços Paypal, Stripe e MBWay.

Após fecho da função primordial da aplicação, seria possível utilizar os requisitos levantados na fase de avaliação da solução de forma a melhorar a atração e retenção dos jogadores. Um ponto relevante levantado na análise, vê-se relacionado com a facilidade de interpretação do objetivo da aplicação. De forma a simplificar o processo, seria importante a criação de uma página de *tutorial* explicativa, logo após o registo de nova conta. Informação como conceito, objetivos, meios de pagamento e benefícios fiscais deverão ser apresentados nesta página.

Para terminar, duas funcionalidades interessantes seriam a automação da seleção dos desafios semanais, através do uso de um serviço recorrente, e a implementação de um sistema de pontuação, já planeado na fase inicial de estudo da aplicação, com tabela de pontuação por jogador, por desafio e globalidade de utilizadores.

6.3 Pontos finais

Em forma de conclusão, a implementação do projeto HelpIt foi essencial na consolidação dos conhecimentos adquiridos ao longo dos seis anos de ensino superior.

O período de aprendizagem durante a licenciatura e mestrado em engenharia informática, juntamente com as habilidades técnicas adquiridas no mundo do trabalho, foram os pilares fulcrais na concretização dos objetivos propostos. Com este projeto, foi possível apreender novas tecnologias e aplicar padrões de desenvolvimento.

Apesar de não ter sido desenvolvida uma versão final e publicada nas lojas aplicacionais, o HelpIt revelou-se um projeto não só de interesse académico, mas também social. Poderá ser dada continuidade ao produto após conclusão da tese descrita neste documento, deixando em aberto uma via conceptual de elevado potencial.

Referências

A história do verdadeiro Drácula: Vlad, o Empalador. (s.d.). Obtido em 3 de Fevereiro de 2019, de Climatologia Geográfica: https://climatologiageografica.com/historia-do-verdadeiro-dracula-vlad-o-empalador/

Add Languages to Your Xamarin Apps with Multilingual App Toolkit. (8 de Fevereiro de 2018). Obtido em 9 de Fevereiro de 2019, de Blog Xamarin: https://blog.xamarin.com/add-languages-to-your-apps-with-xamarin-and-multilingual-app-toolkit/

Adesh. (6 de Janeiro de 2019). *How to create Restful CRUD API with Node.js MongoDB and Express.js*. Obtido em 10 de Setembro de 2019, de ZeptoBook: https://www.zeptobook.com/how-to-create-restful-crud-api-with-node-js-mongodb-and-express-js/

ANDRIANTO, I. (9 de Dezembro de 2018). *Node.js - Send Push Notification Message to Firebase*. Obtido em Agosto de 2019, de Woolha: https://www.woolha.com/tutorials/node-js-send-push-notification-message-to-firebase

Animation Collectibles Market 2019 Industry Size, Trends Evaluation, Global Growth, Recent Developments and Latest Technology, Future Forecast Research Report 2025. (16 de Agosto de 2019). Obtido em Agosto de 2019, de MarketWatch: https://www.marketwatch.com/press-release/animation-collectibles-market-2019-industry-size-trends-evaluation-global-growth-recent-developments-and-latest-technology-future-forecast-research-report-2025-2019-08-16

App Benchmarks Show Improving Engagement, Power of Push. (23 de Abril de 2018). Obtido em 17 de Fevereiro de 2019, de Marketing Charts: https://www.marketingcharts.com/digital/mobile-phone-83123

Banco Alimentar | Quero Fazer Um Donativo. (2019). Obtido em 7 de Setembro de 2019, de Banco Alimentar: https://www.bancoalimentar.pt/faca-um-donativo/

Budiu, R. (19 de Agosto de 2018). *Carousels on Mobile Devices*. Obtido em 16 de Fevereiro de 2019, de NNGroup: https://www.nngroup.com/articles/mobile-carousels/

Building multilingual apps in PowerApps. (9 de Outubro de 2017). Obtido em 9 de Fevereiro de 2019, de Power Apps Microsoft: https://powerapps.microsoft.com/en-us/blog/building-multilingual-apps-in-powerapps/

Canfield, J. (2019). *The Power of Sharing Your Goals With Others*. Obtido em Agosto de 2019, de JackCanfield: https://www.jackcanfield.com/blog/goal-sharing/

Como posso fazer donativos a uma angariação de fundos ou organização de beneficência no Facebook? (2019). Obtido em 7 de Setembro de 2019, de Facebook: https://www.facebook.com/help/395936720538262?helpref=uf_permalink

Conditt, J. (16 de Dezembro de 2014). *Engadget - Humble Bundle Milestones 50M to Charity*. Obtido em 29 de Setembro de 2018, de https://www.engadget.com/2014/12/16/humble-bundle-milestones-50m-to-charity-100m-to-devs/?guccounter=1

Cordova Apache - Introduction | Overview. (22 de Abril de 2016). Obtido em 31 de Agosto de 2019, de Cordova Apache: https://cordova.apache.org/docs/en/latest/guide/overview/index.html

CREATE A MULTILINGUAL APP. (12 de Junho de 2018). Obtido em 9 de Fevereiro de 2019, de PWA Builder: https://pwa-builder.io/knowledge-base/create-a-multilingual-app/

Cronin, K. (14 de Junho de 2018). *Using push notifications for mobile app retention*. Obtido em 17 de Fevereiro de 2019, de Marketing Land: https://marketingland.com/using-push-notifications-for-mobile-app-retention-242258

Cruz Vemelha | Formas de Doar Dinheiro. (2019). Obtido em 7 de Setembro de 2019, de Cruz Vermelha: https://www.cruzvermelha.pt/formas-de-doar-dinheiro.html

Dev Media | Introdução aos bancos de dados NoSQL. (2012). Obtido em 9 de Fevereiro de 2019, de Dev Media: https://www.devmedia.com.br/introducao-aos-bancos-de-dados-nosql/26044

Documentation - Intro | What is Ionic Framework? (23 de Janeiro de 2019). Obtido em 26 de Janeiro de 2019, de Ionic Framework: https://ionicframework.com/docs/intro

Domes, S. (10 de Janeiro de 2017). *Build a Node.js API in Under 30 Minutes*. Obtido em 10 de Setembro de 2019, de FreeCodeCamp: https://www.freecodecamp.org/news/building-a-simple-node-js-api-in-under-30-minutes-a07ea9e390d2/

Donativos em dinheiro para instituições reconhecidas. (28 de Fevereiro de 2019). Obtido em 7 de Setembro de 2019, de DECO Proteste: https://www.deco.proteste.pt/dinheiro/impostos/dicas/donativos-em-dinheiro-para-instituicoes-reconhecidas#

Eden, D. (3 de Julho de 2019). *Github | Animate.css*. Obtido em Agosto de 2019, de Github: https://github.com/daneden/animate.css

ExpressJS | Home. (2017). Obtido em 11 de Setembro de 2019, de ExpressJS: https://expressjs.com/

Firebase Google - Home. (2019). Obtido em 8 de Fevereiro de 2019, de Firebase Google: https://firebase.google.com/

Google | Progressive Web Apps. (s.d.). Obtido em 26 de Janeiro de 2019, de Developers Google: https://developers.google.com/web/progressive-web-apps/

Grimm, S. (13 de Novembro de 2018). *JWT Authentication with Ionic & Node.js – Part 1: The Auth Server*. Obtido em Junho de 2019, de Devdactic: https://devdactic.com/jwt-authentication-ionic-node/

Grimm, S. (27 de Novembro de 2018). *JWT Authentication with Ionic & Node.js – Part 2: The Ionic App*. Obtido em Junho de 2019, de Devdactic: https://devdactic.com/jwt-authentication-ionic/

Heroku - Home. (s.d.). Obtido em 8 de Fevereiro de 2019, de Heroku: https://www.heroku.com/

HOW TO BUILD IONIC 3 MULTI-LANGUAGE APP. (14 de Novembro de 2017). Obtido em 9 de Fevereiro de 2019, de Medium: https://medium.com/@salonimalhotra1ind/how-to-build-ionic-3-multi-language-app-b5a34d105b9

Humble Bundle. (2019). Obtido em 29 de Setembro de 2018, de https://www.humblebundle.com/about

Introduction to JSON Web Tokens. (2019). Obtido em 14 de Setembro de 2019, de JWT.io: https://jwt.io/introduction/

Ionic Framework | Documentation - Native. (19 de Junho de 2019). Obtido em 26 de Janeiro de 2019, de Ionic Framework: https://ionicframework.com/docs/native/

Ionic Framework | Ion-slides. (2019). Obtido em Julho de 2019, de Ionic Framework: https://ionicframework.com/docs/api/slides

Ionic Framework | Native Push Notifications. (9 de Dezembro de 2018). Obtido em 9 de Fevereiro de 2019, de Ionic Framework: https://ionicframework.com/docs/native/push/

Ionic Framework | PayPal. (15 de Janeiro de 2019). Obtido em Setembro de 2019, de Ionic Framework: https://ionicframework.com/docs/native/paypal

Ionic Framework | Publishing - Android Play Store. (29 de Junho de 2019). Obtido em Agosto de 2019, de Ionic Framework: https://ionicframework.com/docs/publishing/play-store

Ionic Framework | Publishing - Publishing to app store. (7 de Junho de 2019). Obtido em Agosto de 2019, de Ionic Framework: https://ionicframework.com/docs/publishing/app-store

Ionic Framework | Stripe. (17 de Setembro de 2017). Obtido em Setembro de 2019, de Ionic Framework: https://ionicframework.com/docs/native/stripe

Ionic Native Community Edition. (23 de Janeiro de 2019). Obtido em 2 de Setembro de 2019, de Ionic Framework: https://ionicframework.com/docs/native/overview

J., D. (20 de Fevereiro de 2019). *Push Notification using Ionic 4 and Firebase Cloud Messaging*. Obtido em Agosto de 2019, de Djamware: https://www.djamware.com/post/5c6ccd1f80aca754f7a9d1ec/push-notification-using-ionic-4-and-firebase-cloud-messaging#ch4

Jarrett, C. (9 de Novembro de 2014). *The Guardian | Why do we collect things? Love, anxiety or desire*. Obtido em Junho de 2019, de The Guardian: https://www.theguardian.com/lifeandstyle/2014/nov/09/why-do-we-collect-things-love-anxiety-or-desire

Joschik. (16 de Abril de 2016). *How Big are the Collectible Markets? Are we really spending $200 billion every year on them?* Obtido em Junho de 2019, de HobbyDB: https://blog.hobbydb.com/2016/04/16/how-big-are-the-collectible-markets/

Junior, S. A. (Fevereiro de 2018). *Medium | Push notifications no Android com Ionic 4 e Firebase*. Obtido em Agosto de 2019, de Medium:

https://medium.com/codengage/push-notifications-no-android-com-ionic-4-e-firebase-e5e92f73f08

LePage, P. (29 de Maio de 2019). *Google | Fundamentals - Your First PWA App*. Obtido em 26 de Janeiro de 2019, de Developers Google: https://developers.google.com/web/fundamentals/codelabs/your-first-pwapp/

Litvinov, D. (9 de Outubro de 2017). *Dribbble | Fitness Badges*. Obtido em Junho de 2019, de Dribbble: https://dribbble.com/shots/3859476-Fitness-Badges?utm_source=Pinterest_Shot&utm_campaign=dmitrilitvinov&utm_content=Fitness%20Badges&utm_medium=Social_Share

M.Farouk Radwan, M. (2017). *Why do people collect things?* Obtido em 22 de Janeiro de 2019, de 2 Know Myself: https://www.2knowmyself.com/Why_do_people_collect_things

M.Farouk Radwan, M. (2017). *Why do people join groups?* Obtido em 4 de Fevereiro de 2019, de 2 Know Myself: https://www.2knowmyself.com/Why_do_people_join_groups

Martikainen, A. (Novembro de 2017). *The New Concept Development (NCD)-model*. Obtido em 8 de Fevereiro de 2019, de Research Gate: https://www.researchgate.net/figure/The-New-Concept-Development-NCD-model-Koen-et-al-2001_fig12_324208591

MBWay - Developers | SOFTWARE DEVELOPMENT KIT PARA OPERATIVA IN-APP. (2019). Obtido em Setembro de 2019, de MBWay - Developers: https://www.mbway.pt/developers/implementacao/#software-development-kit-para-operativa-in-app

Microsoft | Build PowerApps. (2019). Obtido em 26 de Janeiro de 2019, de Microsoft PowerApps: https://powerapps.microsoft.com/pt-pt/build-powerapps/

MongoDB - Cloud Atlas Pricing. (2019). Obtido em 8 de Fevereiro de 2019, de MongoDB: https://www.mongodb.com/cloud/atlas/pricing

MongoDB - Home. (s.d.). Obtido em 8 de Fevereiro de 2019, de MongoDB: https://www.mongodb.com/

MongoDB | Documentation - Connect via Compass. (2019). Obtido em 8 de Setembro de 2019, de MongoDB | Documentation: https://docs.atlas.mongodb.com/compass-connection/

MongoDB Atlas Pricing. (2019). Obtido de MongoDB: https://www.mongodb.com/cloud/atlas/pricing

MongoDB Compass. (2019). Obtido de MongoDB: https://www.mongodb.com/products/compass

MongooseJS | Home. (2011). Obtido em 11 de Setembro de 2019, de MongooseJS: https://mongoosejs.com/

NodeJS - About. (s.d.). Obtido em 8 de Fevereiro de 2019, de NodeJS: https://nodejs.org/en/about/

NoSQL Databases: The Definitive Guide. (20 de Abril de 2017). Obtido em 9 de Fevereiro de 2019, de Pandora FMS: https://blog.pandorafms.org/nosql-databases-the-definitive-guide/

NpmJS | body-parser. (2019). Obtido em 11 de Setembro de 2019, de NpmJS: https://www.npmjs.com/package/body-parser

O que é o Business Model Canvas. (8 de Julho de 2016). Obtido em 9 de Fevereiro de 2019, de Analista Modelos de Negócios: https://analistamodelosdenegocios.com.br/o-que-e-o-business-model-canvas/

Pokémon GO! — Version 2.0 Concept. (11 de Janeiro de 2017). Obtido em 10 de Fevereiro de 2019, de Medium: https://medium.com/@AlbertChoi92/pok%C3%A9mon-go-version-2-0-369948adb47d

Porter's Value Chain - Understanding How Value Is Created Within Organizations. (15 de Novembro de 2016). Obtido em 9 de Fevereiro de 2019, de Mind Tools: https://www.mindtools.com/pages/article/newSTR_66.htm

Quaiato, V. (10 de Outubro de 2016). *O que é Xamarin?* Obtido em 21 de Janeiro de 2019, de Lambda3: https://www.lambda3.com.br/2016/10/o-que-e-xamarin/

Rath, S. (4 de Julho de 2018). *Medium | Independent Notification service using Ionic, Node & FCM*. Obtido em Agosto de 2019, de Medium: https://medium.com/hirewithparam/independent-notification-service-using-ionic-node-fcm-5cdde219480a

Rathore, A. (2019). *ENAPPD | Firebase Push notifications in Ionic 4*. Obtido em Agosto de 2019, de ENAPPD: https://enappd.com/blog/implement-ionic-4-firebase-push/34/

Reen, L. (28 de Março de 2017). *Dribbble | Freebies: Sport Badges*. Obtido em Junho de 2019, de Dribbble: https://dribbble.com/shots/3394444-Freebies-Sport-

Badges?utm_source=Pinterest_Shot&utm_campaign=laurareen&utm_content=Freebies:%20Sport%20Badges&utm_medium=Social_Share

Rogers, N. (27 de Setembro de 2018). *Psychology of Social Media: The Science Behind Why People Share Online*. Obtido em Agosto de 2019, de Meltwater: https://www.meltwater.com/blog/psychology-of-social-media-the-science-behind-why-people-share-online/

Roth, D., Anderson, R., & Luttin, S. (4 de Julho de 2019). *Introduction to ASP.NET Core*. Obtido em 8 de Fevereiro de 2019, de Microsoft Docs: https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-2.2

S, A. (1 de Outubro de 2018). *Medium | New Industry Trend: Digital Collectibles*. Obtido em Agosto de 2019, de Medium: https://medium.com/theblock1/new-industry-trend-digital-collectibles-bba221ea4ac8

Seg Social - Instituições Particulares de Solidariedade Social. (5 de Setembro de 2019). Obtido em 7 de Setembro de 2019, de Seg Social: http://www.seg-social.pt/documents/10152/13140219/Listagem_ipss/8371faa4-dea5-4c03-a47f-3446f1f4c6c3

Sending Firebase Cloud Messages from a Node.js Server. (30 de Agosto de 2017). Obtido em Agosto de 2019, de Techotopia: https://www.techotopia.com/index.php/Sending_Firebase_Cloud_Messages_from_a_Node.js_Server

Stripe - Home. (2019). Obtido em 9 de Fevereiro de 2019, de Stripe: https://stripe.com/pt

Stripe | Multibanco Payments with Sources. (2019). Obtido em Setembro de 2019, de Stripe: https://stripe.com/docs/sources/multibanco

Sutradhar, P. (29 de Março de 2017). *Psychology of Sharing: Why Do People Share On Social Media?* Obtido em Agosto de 2019, de LinkedIn: https://www.linkedin.com/pulse/psychology-sharing-why-do-people-share-social-media-prakash-sutradhar

The Analytic Hierarchy Process. (s.d.). Obtido em 9 de Fevereiro de 2019, de Dipartimento di Ingegneria dell'informazione e scienze matematiche: http://www.dii.unisi.it/~mocenni/Note_AHP.pdf

The Reasons Of People Joining Groups. (24 de Janeiro de 2013). Obtido em Agosto de 2019, de BMS: https://www.bms.co.in/the-reasons-of-people-joining-groups/

Unicef | Outras formas de fazer o seu donativo. (2019). Obtido em 7 de Setembro de 2019, de Unicef: https://www.unicef.pt/como-ajudar/outras-formas-de-fazer-o-seu-donativo/

Unity | Home Page. (2019). Obtido em 19 de Janeiro de 2019, de Unity: https://unity3d.com/pt

Unity 3D | Localized Text Component. (2019). Obtido em 9 de Fevereiro de 2019, de Unity 3D: https://unity3d.com/pt/learn/tutorials/topics/scripting/localized-text-component

Verna Allee describes Value Networks. (17 de Dezembro de 2009). Obtido em 9 de Fevereiro de 2019, de Youtube: https://www.youtube.com/watch?v=VC7W8cMiVFo

Warren, J. (4 de Janeiro de 2017). *Dribbble | Mission:pic badges*. Obtido em Junho de 2019, de Dribbble: https://dribbble.com/shots/3193478-Mission-pic-badges

What exactly is Node.js? (18 de Abril de 2018). Obtido em 9 de Fevereiro de 2019, de Medium | Free Code Camp: https://medium.freecodecamp.org/what-exactly-is-node-js-ae36e97449f5

What Is MongoDB? (2019). Obtido de MongoDB: https://www.mongodb.com/what-is-mongodb

When Collecting Things Becomes a Problem. (1 de Março de 2013). Obtido em Agosto de 2019, de LearningEnglish: https://learningenglish.voanews.com/a/when-collecting-things-becomes-a-problem/1613736.html

Why do humans join groups? (s.d.). Obtido em 4 de Fevereiro de 2019, de Speeli: http://www.speeli.com/articles/view/Why-do-humans-join-groups

Why do people feel the need to belong to groups or clubs of some kind? (s.d.). Obtido em 4 de Fevereiro de 2019, de Quora: https://www.quora.com/Why-do-people-feel-the-need-to-belong-to-groups-or-clubs-of-some-kind

Why do people join groups? (s.d.). Obtido em 4 de Fevereiro de 2019, de LLP Engage: http://llpengage.eu/en/home/training-resources/module-2-engagement-intervention-strategies/2-why-do-people-join-groups/

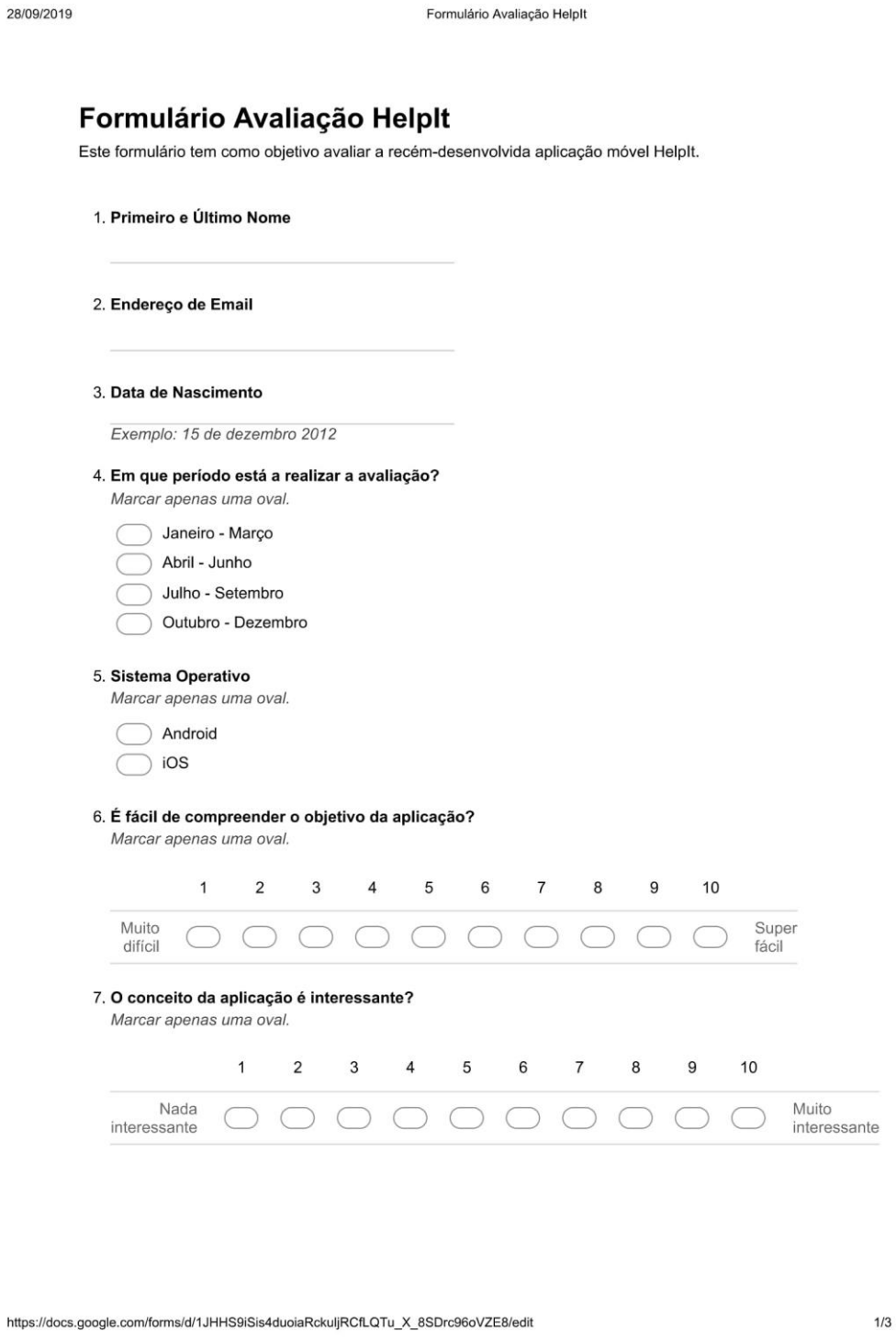
Why Sharing Your Progress Makes You More Likely To Accomplish Your Goals. (19 de Junho de 2015). Obtido em Agosto de 2019, de FastCompany: https://www.fastcompany.com/3047432/why-sharing-your-progress-makes-you-more-likely-to-accomplish-your-goals

Wikipedia | Psychology of collecting. (5 de Outubro de 2019). Obtido em 12 de Outubro de 2019, de Wikipedia: https://en.wikipedia.org/wiki/Psychology_of_collecting

Xamarin. (2019). Obtido em 21 de Janeiro de 2019, de Visual Studio Microsoft: https://visualstudio.microsoft.com/pt-br/xamarin/?rr=https%3A%2F%2Fwww.google.com%2F

Your First Ionic App: Angular. (1 de Julho de 2019). Obtido em 31 de Agosto de 2019, de Ionic Framework: https://ionicframework.com/docs/angular/your-first-app

Anexos



28/09/2019 Formulário Avaliação HelpIt

Formulário Avaliação HelpIt

Este formulário tem como objetivo avaliar a recém-desenvolvida aplicação móvel HelpIt.

1. **Primeiro e Último Nome**

2. **Endereço de Email**

3. **Data de Nascimento**

Exemplo: 15 de dezembro 2012

4. **Em que período está a realizar a avaliação?**

Marcar apenas uma oval.

- Janeiro - Março
- Abril - Junho
- Julho - Setembro
- Outubro - Dezembro

5. **Sistema Operativo**

Marcar apenas uma oval.

- Android
- iOS

6. **É fácil de compreender o objetivo da aplicação?**

Marcar apenas uma oval.

	1	2	3	4	5	6	7	8	9	10	
Muito difícil	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Super fácil

7. **O conceito da aplicação é interessante?**

Marcar apenas uma oval.

	1	2	3	4	5	6	7	8	9	10	
Nada interessante	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Muito interessante

https://docs.google.com/forms/d/1JHHS9iSis4duoiaRckuljRCfLQTu_X_8SDrc96oVZE8/edit 1/3

Figura 90 - Formulário de avaliação (página 1)

8. **Participaria como jogador numa versão final da aplicação?**
Marcar apenas uma oval.

	1	2	3	4	5	6	7	8	9	10	
Definitivamente não	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Sim, sem dúvida

9. **Como avaliaria o aspeto gráfico da aplicação?**
Marcar apenas uma oval.

	1	2	3	4	5	6	7	8	9	10	
Horrível	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Espetacular

10. **A aplicação é simples de utilizar?**
Marcar apenas uma oval.

	1	2	3	4	5	6	7	8	9	10	
Extremamente difícil	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Super simples

11. **A aplicação foi fácil de encontrar?**
Marcar apenas uma oval.

	1	2	3	4	5	6	7	8	9	10	
Não encontrei	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Muito fácil

12. **O aspeto visual e animações ajudaram na utilização da aplicação?**
Marcar apenas uma oval.

	1	2	3	4	5	6	7	8	9	10	
Não, nada	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Foram fundamentais

13. **A aplicação está preparada para daltónicos?**
Marcar apenas uma oval.

- ◯ Sim
- ◯ Não
- ◯ Não sei

14. **A aplicação é ética e moralmente correta em questões raciais, políticas e religiosas?**
Marcar apenas uma oval.

- ◯ Sim
- ◯ Não
- ◯ Não sei

Figura 91 - Formulário de avaliação (página 2)

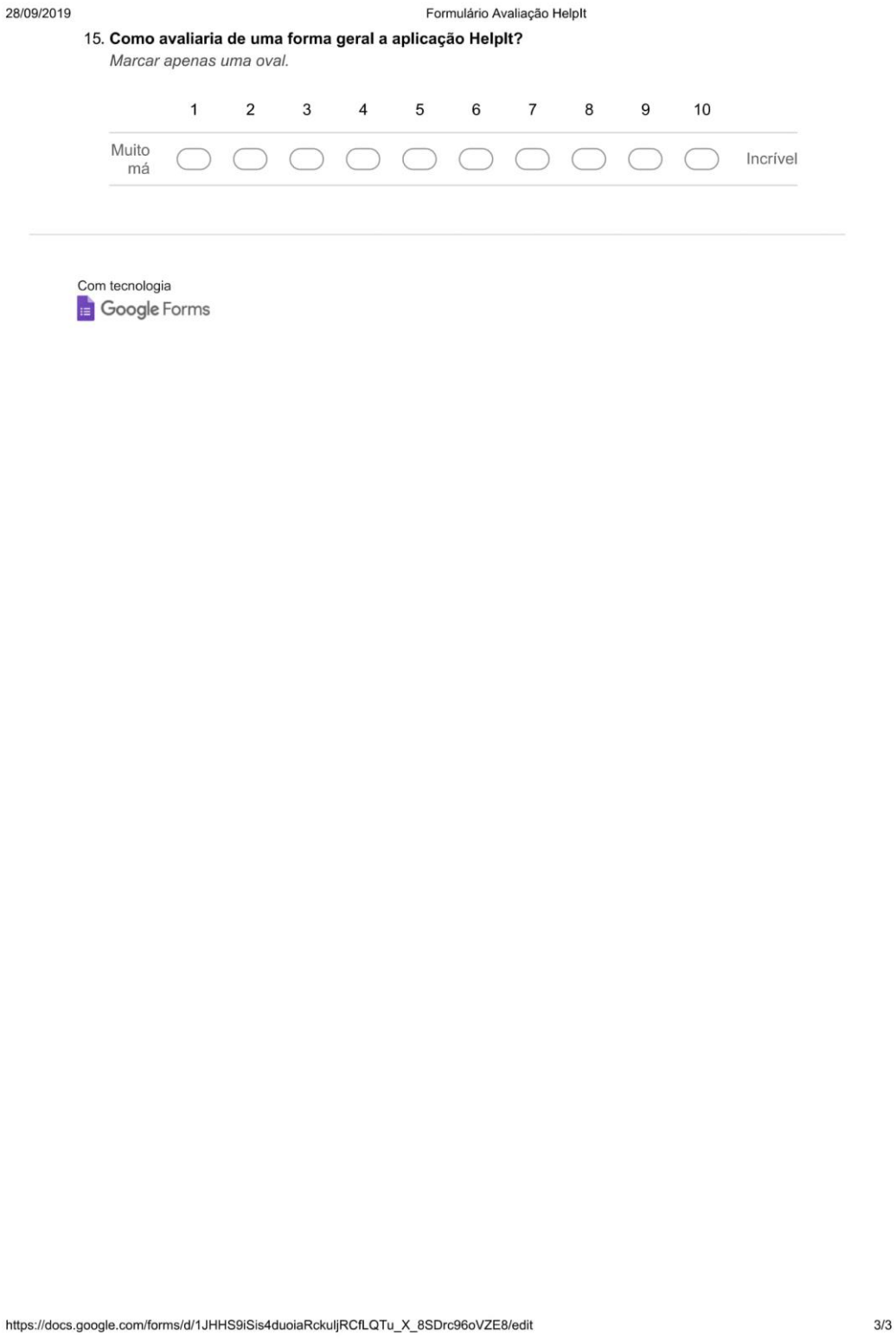


Figura 92 - Formulário de avaliação (página 3)