INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Plataforma de Fidelização de Clientes (Multiempresa)

FÁBIO RODRIGO TEIXEIRA DE MENDONÇA
Outubro de 2016

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Plataforma de Fidelização de Clientes (Multiempresa)

Adão Silva Santos

Fábio Rodrigo Teixeira de Mendonça

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Sistemas Computacionais

Orientador: Dr. Jorge Manuel Neves Coelho

Porto, Outubro de 2016

Resumo

O projeto idealizado para a realização da presente tese de mestrado tem como finalidade o desenvolvimento de um protótipo de plataforma de fidelização de clientes a ser disponibilizada a empresas como um serviço (PaaS). Este protótipo funcionará como um meio de validar a viabilidade do projeto quer a nível tecnológico como a nível comercial.

A realização desta plataforma foi dividida em 3 fases. Na primeira fase foram estudadas e testadas soluções existentes no mercado que endereçam problema identificado e recolhida informação que permitiu detalhar o contexto do problema e, base nessa informação, foram também definidos os intervenientes para a plataforma e os respetivos requisitos. A segunda fase centrou-se na design conceptual de toda a solução tendo sido definidas as funcionalidades esperadas para o sistema e, após uma análise de tecnologia relevante, planificada uma arquitetura para a solução. Na terceira e última fase são descritas para cada componente da plataforma (Portal, API e Aplicações Móveis) o detalhe da planificação e respetiva implementação.

Palavras-chave: PaaS, Fidelização de Clientes, Cartão Desconto, Smartphone

Abstract

The project conceived for the realization of this thesis aims to develop a prototype of a customer loyalty platform intended to be delivered to companies as a service (PaaS). This prototype will work as a validator for the feasibility of the project both on in technological and commercial level.

The development of this platform was divided in three phases. In the first phase consisted on testing and studying existing solutions that already address the identified problem and to gather information allowing to detail the problem context and defining the platform actors and associated requirements. The second phase focused on the conceptual design of the solution and the definition of the expected system features. After an analysis on several technologies relevant for the project an architecture was planned. On the third and final phase it's described for each component of the platform (Portal, API and Mobile Applications) the detail of the planning made and respective implementation.

Keywords: PaaS, Customer Loyalty, Discount Card, Smartphone

Agradecimentos

Aos nossos familiares por todo o apoio, paciência, confiança e motivação que nos transmitiram nesta fase final do percurso académico.

Ao nosso orientador Dr. Jorge Coelho, por todo o apoio e disponibilidade, aliado à elevada capacidade de trabalho e orientação demonstradas.

A todos os que, ainda que não tenham contribuído diretamente, o fizeram de forma indireta apoiando incondicionalmente o decorrer desta tese.

A todos o nosso muito obrigado!

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos e Símbolos

Lista de Acrónimos

AGILE	Metodologia de desenvolvimento de software
AHP	Analytic Hierarchy Process
ANDROID	Sistema operativo desenvolvido pela Google para sistemas móveis
API	Application Programming Interface
AS	Análise Semântica
ASP.NET MVC	Framework desenvolvida pela Microsoft
Bizspark	Programa da Microsoft de apoio às Startup.
CSS3	Folha de estilos de uma página web (Cascade style sheets)
ERP	Enterprise Resource Planning
GetBack	Plataforma de fidelização de clientes desenvolvida no âmbito da tese
HTML5	Hypertext Markup Language, versão 5
IDE	Ambiente de Desenvolvimento Integrado *(Integrated Development Environment*)
IA	Inteligência Artificial
IIS	Internet Information Services
IOS	Sistema operativo desenvolvido pela Apple
JAVASCRIPT	Linguagem de programação interpretada
JS	Linguagem de programação interpretada
LTS	Suporte de uma aplicação a longo termo (Long Term Support)
MOBILE	Sistemas móveis
MVC	Model-View-Controller
NIF	Número de Identificação Fiscal
PaaS	Platform as a Service
PHP	Linguagem de programação para desenvolvimento de aplicações web

PME	Pequena e Média Empresa
SGBD	Sistema de Gestão de Base de Dados
SO	Sistema Operativo
TDD	Desenvolvimento guiado por testes (Test Driven Development)
VP	Proposta de valor (Value proposition)
VSM	*Vector Space Model*
WEB	World wide web
WEBVIEW	Componente do sistema operativo Android para apresentação de conteúdo

1 Introdução

1.1 Contexto

No âmbito da Tese de Mestrado em Engenharia Informática, ramo Sistemas Computacionais, propõe-se a criação de um protótipo de uma plataforma para fidelização de clientes dirigida a pequenas e médias empresas. Um serviço com funcionalidades diferenciadoras na interação que promove entre a empresa e os seus clientes e, ao mesmo tempo, com uma abordagem diferente no conceito e formato do cartão de fidelização disponibilizado.

No presente trabalho será analisado o problema identificado, sendo testadas algumas soluções já existentes no mercado com o fim de encontrar funcionalidades base neste tipo de plataforma e identificar aspetos que possam ser diferenciadores.

A plataforma a desenvolver, batizada de ***GetBack***, pretende a diferenciação das soluções existentes no mercado pelos seguintes aspetos:

- Investimento inicial reduzido ou inexistente para a empresa e utilizador.
- Desmaterialização dos cartões de fidelização.
- Funcionamento em regime de PaaS (Platform as a Service).
- Implementação de ferramentas que promovam o feedback do cliente, para auxiliar as empresas na melhoria dos seus serviços ou produtos.

Será também descrito o processo de planeamento, design da solução e arquitetura da plataforma tendo como ponto de partida os requisitos identificados no estudo do problema bem como detalhes do desenvolvimento das diversas componentes que, no seu todo, constituirão a plataforma projetada.

1.2 Análise de Valor

Nos últimos anos ocorreram profundas alterações nas condições envolventes às atividades económicas, nomeadamente com a revolução tecnológica e a crescente responsabilidade social e ambiental. Estas alterações, fazem com que o mercado económico se encontre em constante mutação e a concorrência se tenha tornado mais agressiva. Consequentemente, as empresas têm necessidade de se adaptar rapidamente e a inovação torna-se fator crucial para a sua sobrevivência.

O problema identificado reside na indisponibilidade (tecnológica e/ou financeira) das Empresas adotarem soluções de fidelização de clientes que lhes permitam alavancar o seu negócio e manterem-se um passo à frente dos seus concorrentes. Pretende-se implementar uma solução que possibilite disponibilizar esta plataforma como um serviço a essas empresas ultrapassando as necessidades de investimento em tecnologia dentro de portas e assegurando um custo reduzido de

investimento inicial. Em simultâneo, com a disponibilização gratuita de uma aplicação móvel para os utilizadores, será possível aumentar a exposição das empresas ao mercado. Esta aplicação móvel trará como retorno aos utilizadores um sistema que lhes permite gerir todos os seus cartões de fidelização numa única aplicação e mesmo tempo obter de recompensas (pontos para troca, descontos, etc.) com base na sua utilização.

1.3 Abordagem Preconizada

Com o estudo efetuado ao nível funcional pretende-se desenhar uma solução que assente numa arquitetura PaaS – *Platform as a Service.* Esta abordagem permitirá retirar proveito do paradigma Cloud e da sua capacidade de acesso resiliente, alta disponibilidade e escalabilidade além de possibilitar um controlo de custos mais rigoroso durante o tempo de vida da plataforma.

A ideia inicial do projeto consiste na criação de um Portal onde tanto utilizadores como empresas terão acesso para gerir as suas contas e usufruir dos diversos serviços disponibilizados. A par com o portal existirá uma API que possibilitará a interação com a plataforma, quer seja por intermédio de aplicações móveis ou de integração com ERP e *softwares* de faturação e POS.

Por fim serão desenvolvidas aplicações móveis destinadas aos utilizadores e empresas que permitirão a interação destes com a plataforma no dia-a-dia.

1.4 Estrutura da dissertação

Este documento encontra-se organizado em cinco capítulos.

No primeiro é efetuada uma introdução. Esta contém uma visão generalizada das motivações que levaram à realização do trabalho apresentado.

O segundo capítulo apresenta o estado da arte das soluções existentes, considerando pontos fortes, pontos fracos e quais as funcionalidades que devemos ter como referência. É, também, neste capítulo efetuada uma descrição detalhada do contexto do problema e análise de valor.

No terceiro capitulo é efetuada a análise concetual da solução a implementar, o estudo da tecnologia relevante e, finalmente, a arquitetura da solução.

O quarto capítulo descreve todo o processo de desenvolvimento, agrupado por cada um dos módulos que compõe a solução final. Para cada um dos módulos são apresentados diversos diagramas que descrevem o seu funcionamento, excertos de código relevantes, os testes unitários e restrições existentes.

Por último, no quinto capítulo, são apresentadas as conclusões do trabalho realizado, uma recapitulação dos resultados obtidos e descrito trabalho futuro.

2 Contexto

2.1 Estado da Arte de Soluções Existentes

Após uma análise geral do problema foram identificadas algumas soluções já existentes no mercado às quais foi decidido efetuar testes de utilização. Essas aplicações são a MEO Cardmobili (Cardmobili.SA, s.d.), H3 (H3, 2016), (La FourDhette, SAS, 2016) e Perfumes e Companhia (Perfumes e Companhia, 2016).

Em primeiro lugar foi analisada a empresa MEO Cardmobili, que iniciou a sua atividade com uma abordagem de *Virtual Wallet* onde o utilizador poder descarregar uma aplicação mobile, efetuar o registo e associar cartões que possua no mundo real. O crescimento desta empresa e a sua notoriedade levaram a que um leque massivo de organizações que já possuíam sistemas de fidelização de clientes com base em cartões aderisse a esta plataforma uma vez que em nada interferia com o funcionamento normal das suas próprias plataformas. Além de cartões de fidelização a MEO Cardmobili conseguiu ainda adicionar cartões de pagamento o que impulsionou ainda mais a sua presença no mundo digital.

Neste momento desenvolvem também plataformas de fidelização para clientes (M|Loyalty (Cardmobili.SA, s.d.)) que incluem a criação de cartões virtuais com gestão de pontos e vouchers. Estes cartões permitem também a identificação dos consumidores através de códigos de barras e, em casos onde os POS estejam preparados, por tecnologia NFC.

Tal como descrito no site da MEO Cardmobili (Cardmobili.SA, s.d.) são disponibilizados outros serviços que alavancam a componente de marketing digital:

- **M|Communication**: solução de comunicação destinada a marketing One-2-One que disponibiliza funcionalidades de alertas proactivos, notificações push, touch-to-call e feedback.

- **M|Campaigns**: solução destinado à criação e envio de campanhas através da própria aplicação. Destacam-se a distribuição de cupões virtuais com capacidade de leitura por diversos meios (p.e. *Barcode*, NFC, etc).

- **M|Shopping**: este produto visa a utilização do equipamento móvel para adicionar produtos a listas de compras (p.e. por *scanning* de códigos de barras de produtos). Permite gerir e monitorizar o *wish list* e seguir o percurso da encomenda (*tracking*)

- **M|Proxymity**: solução de marketing direto que envia notificações de campanhas com base na proximidade dos consumidores com pontos de referências *(beacons*).

Esta empresa possui diversos mecanismos que são efetivamente uma mais valia na área de fidelização de clientes conseguindo obter um base de sistemas de fidelização alargado (com a inclusão de muitos sistemas pré-existentes), contudo não efetua qualquer estudo analítico dos dados nem disponibiliza essa informação de forma clara aos seus clientes. É entendido como estudo analítico dos dados a elaboração relatórios de tendências, hábitos de consumo e perfil de utilizadores com base nos dados recolhidos, relatórios estes que podem ser uma mais valia pois poderão ajudar as organizações a prepararem as suas ações de marketing e a definir forma com estão atualmente no mercado.

Numa análise mais prática à aplicação disponibilizada pela Cardmobili (agora MEO Cardmobili dado que foi adquirida pela operadora portuguesa de comunicações) é de destacar a simplicidade do processo de registo e também a forma como é possível adicionar à carteira uma grande quantidade de cartões. Tal como já referido esta aplicação permite ao utilizador associar informação de cartões de fidelização que já possua por forma a desmaterializa-los. A adições da informação de cada cartão pode ser feita manualmente ou por digitalização do código de barras (caso exista).

Outro pormenor existente nesta aplicação é a capacidade de receção de notificações e eventos referentes aos cartões que estão associados à nossa conta. Na Figura 1 é possível ver o aspeto da aplicação com o Menu de interação com o utilizador, o detalhe de dois dos cartões que foram criados e o código de barras de um desses cartões que serve de identificador junto da respetiva loja. A informação disponível no detalhe de cada cartão corresponde às funcionalidades que estão disponibilizadas por cada um. No caso do cartão IKEA FAMILY (Inter IKEA Systems B.V., 2016) podem ser consultados os cupões e visualizar no mapa as lojas da cadeia IKEA. No caso do cartão CONTINENTE (MODELO CONTINENTE HIPERMERCADOS, S.A., 2015) só é possível ver o detalhe dos dados do cartão não existindo informação adicional.

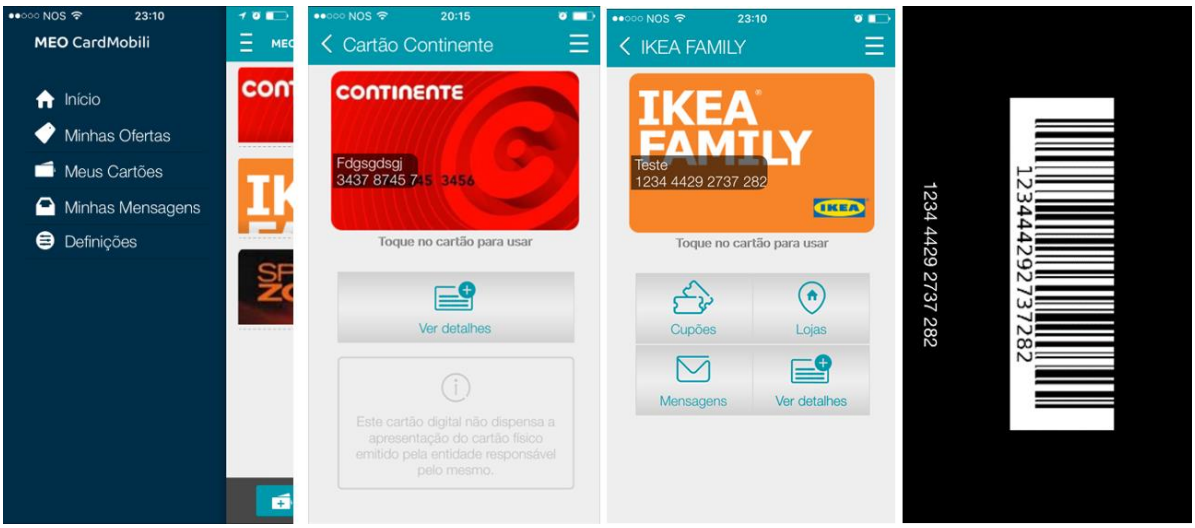


Figura 1 - Ecrãs da aplicação MEO CardMobili

A Tabela 1 sumariza a análise desta solução com os pontos fortes e pontos fracos encontrados.

Pontos Fortes	**Pontos Fracos**
- Processo de registo é simples e rápido de completar;	- Não efetua qualquer ligação com as empresas detentoras dos cartões;
- Permite criar cartão/ introduzir informação no cartão virtual a partir de leitura de código de barras do cartão físico;	- Não permite criar registos novos. Apenas permite criar cartões que o cliente já possui fisicamente;
- Inclui funcionalidades de notificações e eventos	- Não possui capacidade de indicar os pontos acumulados em cada cartão;

Tabela 1 - Pontos Fortes vs Pontos Fracos de aplicação MEO Cardmobili

Uma outra solução a ter em conta é a yLoyalty (YLoyalty , 2016). Trata-se de é uma plataforma criada à medida do cliente (Empresa) que tem como objetivo a criação de um programa de fidelização. Os principais serviços disponibilizados por esta plataforma a quem a adquire é, além da própria plataforma, a capacidade de segmentar os clientes tendo por base a informação recolhida, ou seja, a essência de um programa de fidelização. As organizações que implementem esta solução podem fazer o acompanhamento dos resultados de forma detalhada para procederem a uma abordagem de gestão integrada e BI – *Business Intelligence.*

A plataforma inclui um site institucional feito à imagem do cliente (Empresa) com um *backoffice* que lhe permite efetuar toda a gestão de consumidores, lojas, regras e conteúdos e aplicações mobile de forma a endereçar o crescente mercado de equipamentos móveis.

Outros serviços associados a esta plataforma são também disponibilizados, como por exemplo Marketing e Ações de Comunicação (Facebook (FACEBOOK, s.d.), Twitter (Twitter, 2016) e Email Marketing), criação de catálogos e loja online, desenvolvimento de material de suporte à publicitação da plataforma adquirida e consultoria e formação ás equipas dos clientes. De uma forma geral esta plataforma é vendida para endereçar as necessidades de uma única organização não existindo interação entre clientes empresarias. As análise e segmentação são feitas unicamente com o conhecimento dos próprios clientes não havendo uma visão mais alargada do mercado. Para testar a solução deste fabricante foi testada a aplicação de um dos seus clientes, a cadeia de restaurantes H3.

Tal como a aplicação MEO Cardmobili o processo de registo é extremamente intuitivo, contudo nesta aplicação da yLoyalty é possível efetuar o registo diretamente nos sistemas do cliente, ou seja, novos clientes não precisam de fazer o registo nos restaurantes e receber cartão físico, se bem que essa opção também seja disponibilizada pelos restaurantes H3. A navegação na aplicação é bastante fluida e é disponibilizada muita informação sobre os restaurantes sendo também divulgada informação sobre desafios que dão prémios aos utilizadores da aplicação.

A nível de operação, esta aplicação permite apresentar um QR Code a ser usado na identificação junto dos leitores existentes nos restaurantes facilitando o processo e evitando erros de perceção que por vezes ocorrem quando há necessidade de ditar identificador aos funcionários dos estabelecimentos. É ainda possível visualizar de imediato quantos pontos estão acumulados e quantas refeições faltam para ganhar uma refeição gratuita, ou seja, toda a informação relevante para o utilizador encontra-se facilmente acessível.

Nas definições é possível alterar os dados do utilizador e incluir o NIF de forma a que saia automaticamente a fatura com essa informação. Esta aplicação integra também com a aplicação MBWay (SIBS, 2015) para permitir o pagamento utilizando esse método.

Duas outras funcionalidades a destacar são a capacidade de encontrar lojas desta cadeia por localização e a possibilidade de submeter um *review* sobre uma visita à loja (chamado de cliente mistério na aplicação). Estas funcionalidades podem ser identificadas na Figura 2.

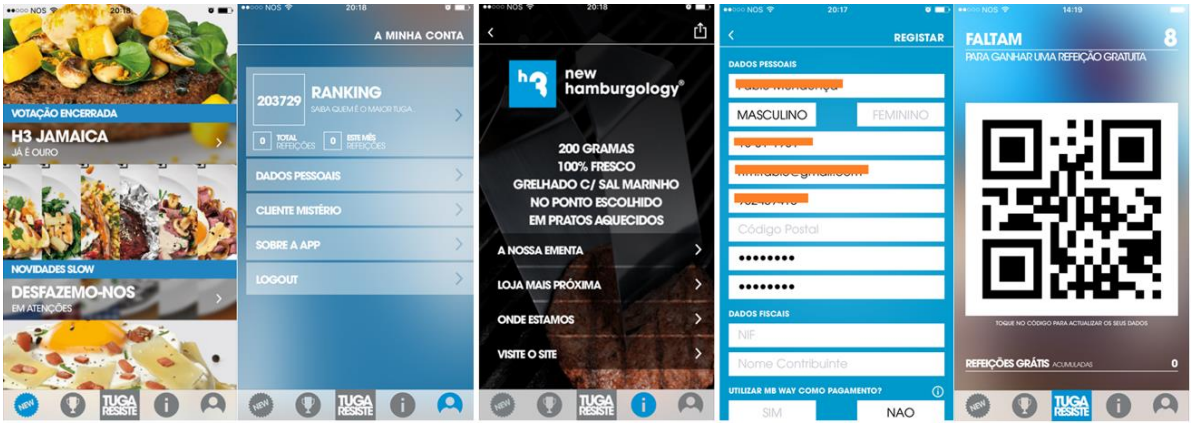


Figura 2 - Ecrãs da aplicação H3

Foram agrupados os pontos mais significativos da análise efetuada originando a Tabela 2.

Pontos Fortes	**Pontos Fracos**
- Processo de registo é simples e rápido de completar;	- Trata-se de uma aplicação exclusiva a esta empresa;
- Permite registo inicial a partir da aplicação	
- Inclui criação de QRCode para identificação nos restaurantes	
- Informação detalhada sobre o restaurante e os produtos vendidos	
- Integração com MBWay para efetuar pagamentos diretamente a partir do smartphone;	

Tabela 2 - Pontos Fortes vs Pontos Fracos de aplicação H3

É relevante para o projeto uma analise a soluções que permitam efetuar pesquisas por localização e possuam um sistema de pontos associada a clientes/empresas distintas, pelo que foi testada a aplicação da plataforma TheFork e a aplicação da cadeia de lojas de cosmética e perfumaria Perfumes e Companhia.

Quanto à TheFork, esta solução foi desenvolvida pela empresa francesa La Fourchette (La Fourchette , s.d.) e permite que os utilizadores efetuem pesquisa de restaurantes na sua base de clientes associados. Esta pesquisa pode ser feita por localização utilizando as coordenadas GPS dos smartphones ou por critério de pesquisa (p.e. localidade ou nome do restaurante). As empresas associadas a esta plataforma podem gerir as reservas nos seus restaurantes e criar menus de promoção tendo em vista angariar mais clientes.

A TheFork possui ainda um componente de pontos (chamados de "*yums"*) que são atribuídos aos utilizadores com base nas suas reservas e nos *reviews* que estes podem efetuar após usufruírem dos serviços do restaurante escolhido. Estes pontos podem ser utilizados em visitas a restaurantes que os aceitem como forma de pagamento. Nos testes efetuados a esta aplicação foi possível verificar a fiabilidade na pesquisa por proximidade bem como a simplicidade do sistema de reservas e posterior preenchimento de *reviews*. Conforme a Figura 3, o aspeto gráfico da aplicação é simplista, mas permite identificar toda a informação de forma rápida e uma experiência de utilização agradável e intuitiva. Aliada à aplicação móvel está também um motor que efetua a confirmação das reservas via SMS e que é extremamente rápido tendo as confirmações demorado menos de 1 minuto a serem validadas e notificadas.

De uma forma geral, nos restaurantes que foram reservados com esta aplicação, o funcionamento foi simples tendo sido inclusive utilizado *"yums"* (pontos da plataforma TheFork) como meio de pagamento.

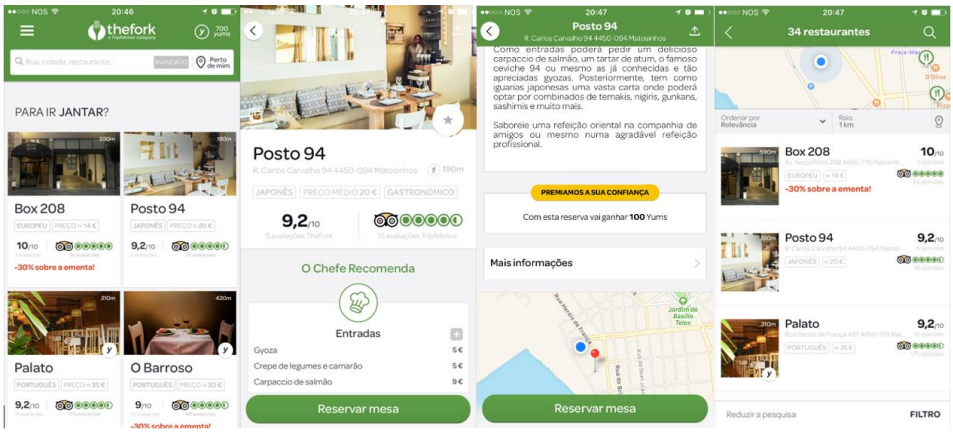


Figura 3 - Ecrãs da aplicação TheFork

Tal como já efetuado anteriormente para as outras soluções analisadas foram reunidos os pontos fortes na Tabela 3 tendo sido identificado com único ponto fraco a exclusividade do setor endereçado por esta aplicação.

Pontos Fortes	Pontos Fracos
- Processo de registo é simples e rápido de completar;	- Dedicado exclusivamente a restaurantes
- Permite registo inicial a partir da aplicação	
- Pesquisa por localização rápida	
- Sistema de pontos bem estruturado;	
- Integração com motor central que efetua a gestão de todas as reservas, reviews e pontos	

Tabela 3 - Pontos Fortes vs Pontos Fracos de aplicação TheFork

A última aplicação testada foi a da empresa Perfumes & Companhia intitulada *PC LikeMe* (Perfumes e companhia, s.d.). Trata-se de uma solução proprietária desta companhia de lojas de retalho de produtos de cosmética e tem como objetivo conhecer os hábitos específicos dos seus utilizadores disponibilizando acesso a informação no mundo da cosmética e perfumaria. O utilizador pode utilizar esta aplicação para efetuar o registo no programa de fidelização e a partir de aí acumular pontos que poderão ser trocados por descontos em compras. Através de uma página central é possível visualizar os pontos acumulados e qual o desconto já disponível para consumir além de um QRCode que pode ser utilizado nas lojas como identificador.

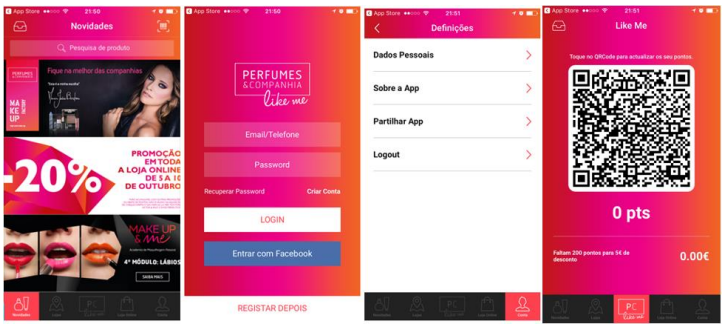


Figura 4 - Ecrãs da aplicação PC *LikeMe* (Pagina Novidades, Registo, Definições)

Como pode ser visto nas Figura 4 e 5 o layout da aplicação é simples e intuitivo sendo a navegação efetuada por uma página com menu com as funcionalidades disponíveis. A capacidade de obter informação sobre os produtos está disponível sob duas formas: em primeiro lugar pode ser utlizado o scanner para ler o código de barras do produto e obter informação sobre o mesmo. A segunda forma é a de pesquisar os produtos por critério (p.e. nome).

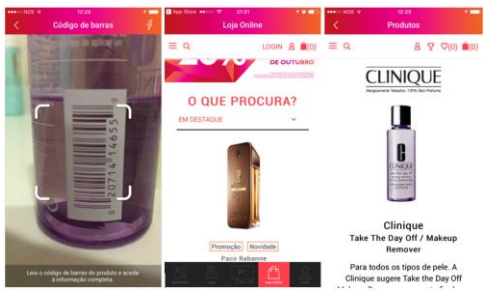


Figura 5 - Ecrãs da aplicação PC *LikeMe* (Pesquisa de informação)

A aplicação possui um localizador de lojas indicando no mapa ou em lista quais as que ficam mais próximas do utilizador, conforme pode ser verificado na Figura 6.

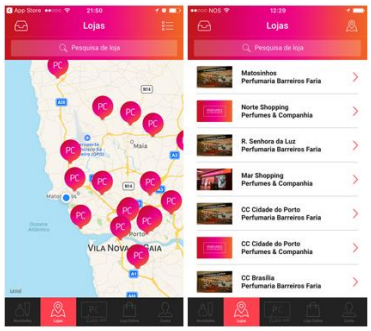


Figura 6 - Ecrãs da aplicação PC *LikeMe* (Localização)

Durante a utilização desta aplicação nunca foi solicitada a apresentação da mesma ou do QRCode para identificação tendo sido somente confirmado o nº de contacto (telemóvel) associado ao perfil de cliente.

Pontos Fortes	**Pontos Fracos**
- Processo de registo é simples e rápido de completar;	- Dedicado exclusivamente à empresa Perfumes&Companhia
- Permite registo inicial a partir da aplicação	- Sem possibilidade de fazer reviews
- Pesquisa por localização	
- Sistema de pontos	
- Pesquisa de produtos por digitalização de código de barras	

Tabela 4 - Pontos Fortes vs Pontos Fracos de aplicação PC *LikeMe*

Com base nas quatro aplicações testadas, foram identificadas algumas características consideradas uma mais valia na solução que a implementar. Em primeiro lugar a capacidade de disponibilizar informação de forma rápida requer o planeamento de uma API com elevada capacidade de resposta aos pedidos executados pelas aplicações móveis. Desta forma será possível obter uma pesquisa por localização com rapidez e precisão e também descarregar rapidamente a informação associada quer ao utilizador como às empresas encontradas e/ou associadas.

A nível de aplicação móvel foram identificados nas 4 aplicações testadas alguns pontos cuja implementação no projeto a desenvolver é essencial:

Pontos a considerar na implementação
- Interface simples, rápido e intuitivo
- Pesquisa por proximidade geográfica e critérios adicionais (p.e. categorias)
- Disponibilizar resultados de pesquisas quer em mapa como em lista
- Mostrar pontos acumulados em cada uma das empresas associadas
- Capacidade de efetuar *reviews* às empresas associadas

Tabela 5 - Pontos a considerar na implementação

Apesar de não terem sido testadas as aplicações do lado das empresas, com a interação com os diferentes sistemas analisados foram identificadas as seguintes necessidades que serão aplicáveis quer a uma aplicação móvel ou interação de ERP existentes com a plataforma central:

- Pesquisar utilizador por identificador (p.e. QRCODE ou nº telefone);
- Conseguir associar um utilizador (Caso este não este ainda associado);
- Adicionar/subtrair pontos à conta do utilizador;

2.2 Detalhes sobre o Contexto e Problema

Vivemos numa sociedade onde o consumidor está, cada vez mais, exposto a diversos artefactos de marketing e publicidade. Ter meios para chegar ao consumidor de forma acutilante é algo apenas disponível a organizações com meios (financeiros, técnicos e humanos). Esta capacidade de explorar ferramentas de marketing é limitada em muitas das empresas existentes uma vez que o foco da sua atividade está na procura da eficácia e eficiência das operações diretamente ligadas à sua atividade.

Empresas possuem, por norma, poucos recursos destinados a ações de marketing e o custo de infraestruturas tecnológicas capazes de suportar este tipo de soluções é demasiado elevado quando comparado com outros investimentos mais prementes para a organização.

Dado que vivemos numa sociedade onde a partilha de dados está em crescimento derivado da evolução tecnológica que tem vindo a ocorrer as empresas procuram disponibilizar aos seus clientes programas de fidelização que, em troca de descontos, acumulação de pontos ou campanhas, conseguem obter informação que lhes permite endereçar melhor as necessidades dos clientes incrementando assim as suas vendas.

Uma das grandes tendências do comercio atual é o *"customer centricity"* (Clerck, 2015) que na sua essência significa colocar o cliente no centro de tudo. A utilização de ferramentas que permitam obter um conhecimento mais aprofundado sobre os hábitos de consumo dos consumidores é assim algo muito valorizável e importante.

A forma mais comum de se obter este tipo de conhecimento é a criação de cartões de fidelização que registam todas as operações efetuadas pelo consumidor em troca de pontos ou descontos.

Em (Meyer-Waarden, 2008) foram definidos os programas de cartões de fidelização com um sistema integrado de ações de marketing com objetivo de aumentar a lealdade dos consumidores através da construção de relações personalizadas. Com o conhecimento sobre os hábitos do consumidor é possível adotar estratégias de *marketing-mix* que assentam na interatividade e individualização acompanhadas de técnicas de marketing direto.

Um outro fator relevante para a utilização deste tipo de artefacto é a segmentação dos consumidores identificando quais os mais rentáveis e leais garantindo assim a recompensa dos clientes certos.

Tal como indicado por (Yi & Jeon, 2003), um programa de fidelização para ser bem-sucedido deve segmentar os seus clientes de acordo com o valor que estes representam para organização, recompensando os mais valiosos e desencorajando os que são sinónimo de prejuízo.

Há que ter em conta que um programa de fidelização (baseado em cartões p.e.) para ser bem-sucedido deve ter uma elevada cobertura de utilizadores permitindo maximizar a exposição dos produtos e poder criar uma base de dado representativa relevante para estudo e segmentação. (Demoulin & Zidda, 2008).

2.2.1 Análise do Problema

O custo de implementação e manutenção de solução de fidelização de clientes não está no âmbito das necessidades imediatas das empresas e por isso verifica-se que apenas empresas com uma estrutura mais robusta possui este tipo de soluções. Falamos de cadeias de lojas, redes franchisadas ou grandes superfícies.

Pequenas empresas não dispõem normalmente de meios tecnológicos e financeiros para implementar soluções digitais deste tipo. Muitas destas empresas utilizam sistemas mais arcaicos como por exemplo cartões físicos com identificação ou ainda meros cartões descartáveis onde assinalam manualmente as visitas ou consumos. Estes métodos tradicionais são pouco apelativos para o consumidor final pois obrigam a que estes andem com esses mesmos cartões no dia a dia ocupando espaço nas suas carteiras e sendo possível o seu extravio. Por outro lado, a tendência dos consumidores é transportar para os smartphones todos os seus hábitos que anteriormente estevam em meios físicos (p.e. lista de tarefas, lista de contactos, etc.).

A utilização dos smartphones como porta cartões virtual não pode ser considerada uma novidade pois existem já soluções que endereçam essa função, contudo essas soluções ou são proprietárias e obrigam à instalação de uma aplicação móvel por cada cartão virtual/empresa ou são aplicações limitadas ao conceito de *virtual wallet* onde é apenas possível identificar o utilizador junto da empresa *in loco* sem capacidade de aceder a informação do utilizador (pontos acumulados, visitas, alteração de definições, reviews, etc.).

Assim sendo, faz falta no mercado uma solução que permita às empresas implementar sistemas de fidelização de clientes com custos controlados, sem investimento inicial em recursos tecnológicos (p.e. servidores, licenciamento de software, etc.) e com potencial para expansão suas estratégias de marketing. Por outro lado, dar a possibilidade aos utilizadores de, numa só aplicação, centralizarem todos os seus cartões (de empresas associadas) podendo, para cada uma delas, aceder à sua informação pessoal.

Aliando as necessidades descritas torna-se necessário o desenvolvimento de uma plataforma que possibilite que os utilizadores possam encontrar facilmente as empresas associadas baseando a pesquisa em categorias e geolocalização o que permite também uma exposição dessas mesmas empresas a um maior número de potenciais clientes.

2.2.2 Intervenientes da plataforma

Os intervenientes da plataforma podem ser agrupados da seguinte forma:

- **Administrador –** ator definido por omissão, tem acesso a todas as funcionalidades e vistas existentes.
- **Empresa -** clientes que aderiram ao serviço disponibilizado pela plataforma. Tem acesso a todas as funcionalidades e vistas disponíveis para as empresas associadas.
- **Utilizador –** pertence ao grupo de utilizadores com vistas e funcionalidades definidas pelo administrador do sistema.

2.2.3 Definição de requisitos

No âmbito da engenharia, um requisito consiste na definição documentada de uma propriedade ou comportamento que um produto ou serviço particular deve atender. O conceito de requisito é, também, utilizado formalmente na engenharia de software e engenharia de sistemas, referindo-se à definição de uma característica, atributo ou funcionalidade que um sistema deve implementar para atingir os objetivos propostos.

2.2.3.1 Requisitos do utilizador

A Tabela 6 apresenta os requisitos propostos na implementação da solução, estes requisitos estão agrupados por interveniente.

Interveniente	Requisitos
Administrador	- Configurar sistema - Gestão das categorias e subcategorias - Gestão das empresas associadas - Gestão de utilizadores - Consultar relatórios do sistema - Gestão dos administradores - Gestão dos perfis
Empresa	- Gestão do perfil - Gestão de cartões (utilizadores) associados - Subscrever produto - Gerir reviews - Registar visitas - Atribuir pontos
Utilizador	- Gerir perfil - Efetuar pesquisas - Gerir cartões virtuais - Consultar visitas e pontos - Consultar e responder a *reviews* e comentários

Tabela 6 - Listagem de requisitos por interveniente

2.2.3.2 Requisitos de sistema

- Requisitos funcionais
  - ✓ **Auditoria**

    - Associar a todos os registos informação sobre o utilizador que criou ou alterou a informação, bem como a hora a que esse evento ocorreu.

    - Para os dados mais importantes, utilização da técnica soft-delete (registo permanece guardado na base de dados, mas não está visível para o utilizador) durante um período de tempo definido pelo administrador. No final desse período, caso não seja revertida a eliminação, o registo é removido por completo da base de dados.

  - ✓ **Autenticação**

    - Implementação de um sistema de autenticação baseado em perfis.

    - Cada utilizador pode ter um ou mais perfis associados, cada um desses perfis define quais as funcionalidades a que tem acesso.

    - O acesso ao sistema está protegido por formulário de validação.

- Requisitos de usabilidade

  Os requisitos de usabilidade são essenciais para o sucesso da aplicação, pelo que no desenho e implementação do novo sistema, serão tidos em atenção os seguintes pontos:
  - ✓ Facilidade de instalação e utilização da aplicação
  - ✓ Navegação fluida
  - ✓ Fiabilidade da informação apresentada/sugerida

- Requisitos de fiabilidade

  Os requisitos de fiabilidade definem como o sistema reage quando está submetido a situações problemática e de stress.
  - ✓ **Disponibilidade**

    - O sistema deverá estar disponível 100% do tempo.
  - ✓ **Frequência e gravidade dos erros**

    - Erros serão classificados como: menores, significativos e críticos.

    - Quando um erro menor acontece o sistema avisa ao utilizador do erro e continua a execução normalmente.

    - Ao acontecer um erro significativo o sistema avisa ao utilizador e volta ao ponto em que estava antes do erro acontecer.

    - Quando se trata de um erro crítico o sistema deve reiniciar e se existiu alguma alteração na base de dados, esta operação tem de ser desfeita.

2.3 Análise de Valor

2.3.1 Conceito

Nos últimos anos ocorreram profundas alterações nas condições envolventes às atividades económicas, nomeadamente com a revolução tecnológica e a crescente responsabilidade social e ambiental. Estas alterações, fazem com que o mercado económico se encontre em constante mutação e a concorrência se tenha tornado mais agressiva. Consequentemente, as empresas têm necessidade de se adaptar rapidamente e a inovação torna-se fator crucial para a sua sobrevivência.

Por definição, proposta de valor (VP) é um método de competitividade organizado e criativo, visando a satisfação da necessidade do cliente, baseado num processo específico de conceção, simultaneamente funcional, económico e multidisciplinar. A Proposta de valor é a razão pela qual os clientes escolhem uma empresa em detrimento de outra.

O valor percebido de um produto ou serviço pode ser definido como a utilidade do ponto de vista do consumidor baseada na perceção do que é recebido e do que é dado, ou seja, dos benefícios e sacrifícios associados. Todos os fatores que influenciam o valor percebido e satisfação do cliente podem ser consultados na Figura 7.

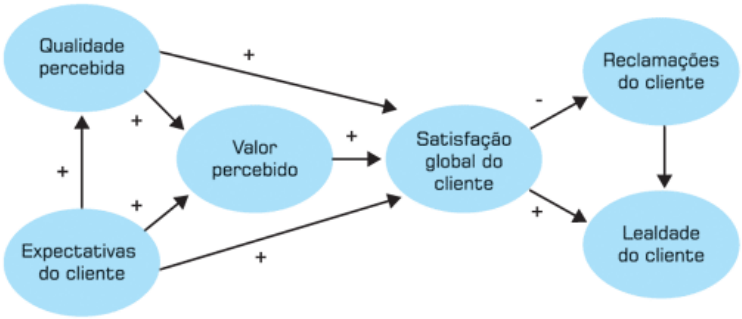


Figura 7 – Satisfação do cliente (Maria Auxiliadora Cannarozzo Tinoco, s.d.)

2.3.2 Proposta de valor – Utilizador

A viabilidade da plataforma depende, em larga escala, da adesão dos Utilizadores à plataforma. O reconhecimento de benefícios na sua utilização torna-se crucial. Nesse sentido, a estratégia inicial passou por recolher dados para perceber quais as funcionalidades que deveriam ser disponibilizadas aos utilizadores, permitindo a diferenciação das soluções já existentes no mercado.

Para melhor compreender as tendências dos utilizadores e quais as funcionalidades que estes mais valorizam ou estariam dispostos a testar, foi criado um inquérito online (Santos & Mendonça, 2016) utilizando a plataforma Google (Google , 2016). Posteriormente foi divulgado em diversas redes sociais (p.e Facebook) com o objetivo de atingir o maior número de participantes. Um inquérito é

um instrumento de investigação que utiliza processos de recolha sistemática de dados, com vista a dar resposta a um determinado problema. Baseia-se numa série de perguntas com respostas fechadas, permitindo quantificar os resultados e, consequentemente, efetuar uma análise estatística.

O inquérito foi estruturado com os seguintes objetivos:

- Obter a percentagem de utilizadores que possuem cartões de fidelização e qual a frequência de utilização;
- Expetativas em relação aos benefícios que encontram na sua utilização;
- Qual o grau de participação previsto dos utilizadores em formulários de satisfação
- Descobrir qual o formato dos cartões que os utilizadores mais valorizam
- Grau de aceitação dos utilizadores à desmaterialização dos cartões de fidelização e utilização do telemóvel para esse efeito.

O inquérito, composto por 7 questões com respostas fechadas, obteve 173 respostas.

Terminada a recolha de dados, procedeu-se ao tratamento da informação para dar respostas aos objetivos definidos. Os resultados formam animadores, a taxa de utilização de cartões é elevada, a percentagem foi de 79.8% (conforme se pode verificar na Figura 8) e o número de utilizadores que mostram interesse em utilizar o telemóvel como gestor de cartões de fidelização obteve 55% dos votos (conforme se pode verificar Figura 9). Quanto à possibilidade de poder gerir e consultar toda a informação de todos os cartões numa única aplicação de telemóvel e, ao mesmo tempo, libertar espaço na carteira, os números são ainda mais expressivos e atingem 75.7% (Figura 10).

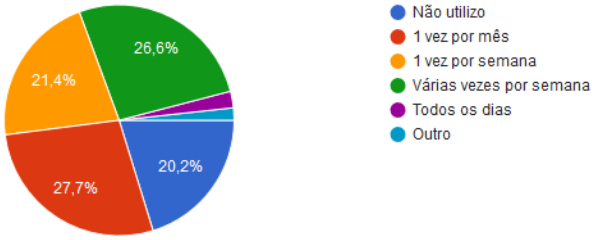


Figura 8 – Distribuição de respostas por utilização de cartões de fidelização

Figura 9 – Distribuição de respostas por formato do cartão preferido

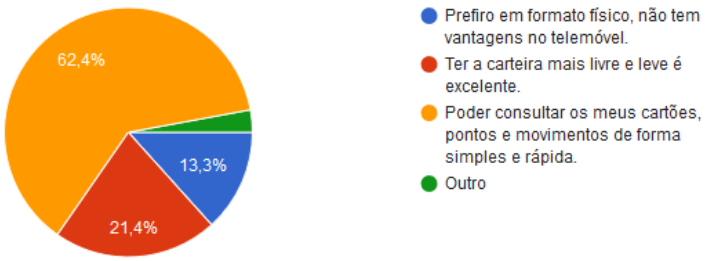


Figura 10 – Distribuição de respostas por vantagens associadas ao cartão virtual

Os resultados permitem validar os nossos objetivos iniciais e proposta de valor definida para os utilizadores. A aceitação das novas funcionalidades é elevada, os utilizadores mostraram muito interesse na solução proposta.

	Serviço	**Relação**
Benefícios	- Descontos diretos - Simplicidade utilização - Recompensas por participação em inquéritos de satisfação - Desmaterialização dos cartões	- Confiança
Sacrifícios	- Custo - Tempo	- Tempo

Tabela 7 - Tabela de sacrifícios e benefícios para o utilizador da plataforma

A proposta de valor para os utilizadores assenta na disponibilização de uma plataforma gratuita onde podem pesquisar empresas aderentes perto da sua localização atual, obter recompensas por responder a inquéritos de satisfação, centralizar numa única aplicação todos os cartões de fidelização e permitir consultar todos os detalhes relacionados com cada um dos cartões.

2.3.3 Proposta de valor – Empresas

Ao aderir à plataforma, a empresa passa a disponibilizar um sistema de fidelização e recompensa aos seus clientes. Mas, ao contrário dos utilizadores, esta adesão tem um custo associado. Por norma, as empresas apenas adquirem um produto ou serviço se este for considerado uma mais valia para o seu negócio. Torna-se necessário ir ao encontro das necessidades dessas empresas e transmitir confiança e valor da plataforma GetBack.

Para estudar a satisfação das empresas às soluções existentes no mercado e, ao mesmo tempo, perceber quais as expetativas e funcionalidades que seriam uma mais valia, foi criado outro inquérito (Santos & Mendonça, 2016), usando a plataforma Google. Contrariamente ao inquérito disponibilizado aos utilizadores, este estava focado para as empresas e foi apenas divulgado em grupos fechados para empresários e plataformas onde se concentram empresas e gestores de empresas.

O inquérito, composto por 5 perguntas, foi estruturado com os seguintes objetivos:

- Obter percentagem de empresas que implementam uma solução de fidelização com os clientes, grau de satisfação e nível de aceitação para testar um produto.
- Funcionalidades que, no ponto de vista das empresas, geram mais valor para os clientes.
- Funcionalidades que, no ponto de vista das empresas, geram mais valor para as próprias empresas.
- Dimensão média da carteira de clientes por empresa.
- Como estruturar o custo do produto perante as empresas aderentes.

O inquérito, composto por 5 questões com respostas fechadas, obteve 52 respostas.

Terminada a recolha de dados, procedeu-se ao tratamento da informação para dar respostas aos objetivos definidos. Os números são expressivos, 50% das empresas não têm, mas consideram a hipótese de implementar um sistema de fidelização (Figura 11).

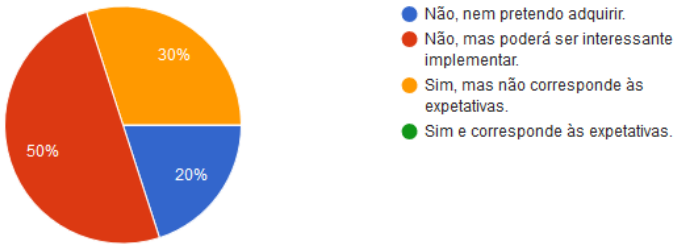


Figura 11 – Nível de implementação de sistemas de fidelização nas empresas

Pela mesma ordem de ideias, as empresas valorizam a possibilidade de possuir um meio para interagir com os seus clientes, nomeadamente a possibilidade de:

- Enviar mensagem de parabéns pelo aniversário – 70%
- Oferecer sistema de recompensa baseada em pontos – 60%
- Disponibilizar inquéritos de satisfação online – 55%

Por ultimo, foram colocadas questões para perceber qual o valor que as empresas consideravam justo pagar por um serviço com as caraterísticas descritas, bem como o número médio de clientes por empresa. A combinação destes dois parâmetros permite analisar e definir quais os planos de serviços a prestar às empresas que pretendam aderir à plataforma. Este estudo vai ao encontro do valor percebido das empresas em relação à plataforma *Getback*.

A proposta de valor para as empresas assenta na disponibilização de um serviço que lhes permite fidelizar os clientes, recompensando-os pela participação em inquéritos de satisfação e atribuindo pontos por cada compra efetuada, pontos que podem ser descontados em futuras compras. Sendo uma plataforma centrada no telemóvel, a empresa dispõe de ferramentas que lhe permitem interagir com os seus clientes, através do envio de mensagens escritas.

2.3.4 Plataforma GetBack

2.3.4.1 Modelo de negócios

Para melhor perceber e integrar a proposta de valor, deve-se gerar e compreender o modelo de negócios. Este modelo define a forma como uma empresa cria valor para si ao criar valor para os seus clientes.

Uma proposta de valor bem definida é a base do Modelo de Negócio, pois permite-nos entender o negócio que pretendemos implementar, determinar o método de gerar dinheiro, quais os parceiros chave, como atrair e satisfazer os clientes de forma a fideliza-los.

Após uma análise cuidada descrita nos pontos seguinte, foi gerado o modelo de negócios para a plataforma que se pretende implementar, o qual encontra-se em anexo.

2.3.4.2 Cenários de negócio

Negociação é o processo de alcançar objetivos por meio de um acordo nas situações em que existam interesses comuns, complementares e opostos, isto é, conflitos, divergências e antagonismos de interesses, ideias e posições (Wanderley, s.d.).

A abordagem à negociação entre o vendedor e o eventual comprador pode ser classificada em dois tipos:

- Integrativa – Todas as partes envolvidas ganham (Ganha-Ganha).
- Distributiva - Pelo menos uma das partes perde (Perde-Perde, Perde Ganha, Ganha-Perde)

Esta plataforma pretende disponibilizar às empresas um serviço, que seja entendido como um beneficio, uma mais valia e não um custo. Que fornece ferramentas que lhe permitam melhorar processos e estreitar a relação de com os seus clientes. Nesse sentido, a única negociação que faz sentido adotar é a negociação integrativa, em que ambas as partes ganham com o negócio.

Fases da Negociação Integrativa	
Recolha de dados	- Recolher dados necessários e importantes para a negociação
Análise dos dados	- Analisar os dados obtidos
Planeamento	- Delinear a melhor estratégia e solução para o cliente
Negociação	- Apresentar a proposta previamente preparada e estudada - Transmitir benefícios - Esclarecer dúvidas - Negociar
Fechar negócio	- Atingir um acordo favorável a ambas as partes - Implementação da solução

Tabela 8 - Fases da Negociação Integrativa

2.3.4.3 Criação de valor

Um estudo recente sobre a Criação de Valor gerada pelas empresas, revela que as organizações que têm melhor performance económica a médio prazo são as que cultivam, em primeiro lugar, a criação de valor para o cliente, por contrapartida daquelas para quem os números e balanços são a única "bíblia" de gestão e que desvalorizam muitas vezes o que acontece no mercado e nos clientes (Martin, 2010).

O sucesso deste projeto só poderá existir se as empresas sentirem que o serviço adquirido criou valor para o seu negócio. A análise para criação de valor deve, por isso, focar-se nas empresas, ir ao encontro das suas necessidades. Para melhor compreender e conhecer as suas principais necessidades foram definidas as seguintes estratégias:

- Promover interatividade com as empresas, fortalecendo a relação e promovendo o feedback
- Investir em pesquisa e desenvolvimento de novas soluções
- Estudar hábitos dos consumidores para auxiliar as PME nas estratégias a implementar
- Serviços personalizados para as empresas

Com base na informação recolhida, tendo como principal objetivo criar valor para o cliente e para a empresa, foi aplicado o modelo AHP (Analytic hierarchy process) para analisar alternativas para problemas encontrados e definir estratégias. O AHP permite, com base num problema bem definido, estruturar num diagrama hierárquico as alternativas e critérios, definindo para cada uma delas um nível de importância na decisão final. Os resultados são somados para a obtenção da prioridade global de cada alternativa.

A análise das alternativas e funcionalidades foi baseada nas respostas das empresas e dos utilizadores que participaram nos inquéritos. Estes inquéritos permitiram, também, perceber a melhor abordagem para gerar valor para a plataforma Getback.

3 Design da Solução

Neste capitulo pretende-se pegar na proposta original, incorporar as mais valias detetadas na analise das soluções existentes e, com base nisso, planificar a solução a desenvolver. Serão também definidos quais os requisitos funcionais para os diversos componentes e, através da analise à tecnologia disponível, eleger uma arquitetura final para a plataforma.

3.1 Design Conceptual

Na Figura 12 encontra-se o Modelo ER (Entidade Relação) idealizado ao momento onde é possível visualizar as relações de cada um dos intervenientes no sistema que será criado. De salientar que ainda está em análise se as entidades "Utilizador" e "Cartão" serão consideradas em separado ou como uma única entidade, contudo atualmente ainda não foi tomada qualquer decisão a esse respeito.

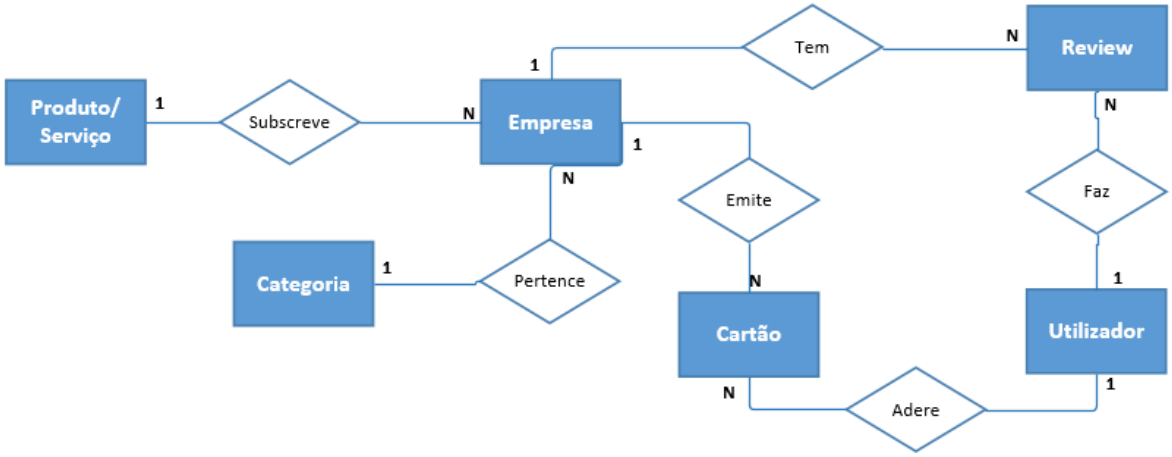


Figura 12 - Diagrama ER da solução

A ideia projetada por este diagrama assenta na existência de uma entidade "Empresa" que pertence a uma determinada "Categoria" permitindo assim catalogar as empresas. A associação das Empresas com os Produtos/Serviços reflete a interoperabilidade das empresas clientes com a plataforma.

Do lado do utilizador existem diversas associações que lhe permitem usufruir da plataforma, nomeadamente a associação à Empresas por intermédio de um cartão (virtual). O mesmo se aplica à caracterização do relacionamento entre Utilizador e os Reviews efetuados a uma determinada empresa onde é possível ter vários Reviews para várias empresas nas quais um dos utilizadores está associado.

Na Figura 13 esta representado o modelo relacional da base de dados, resulta do mapeamento do modelo Entidade-Relação.

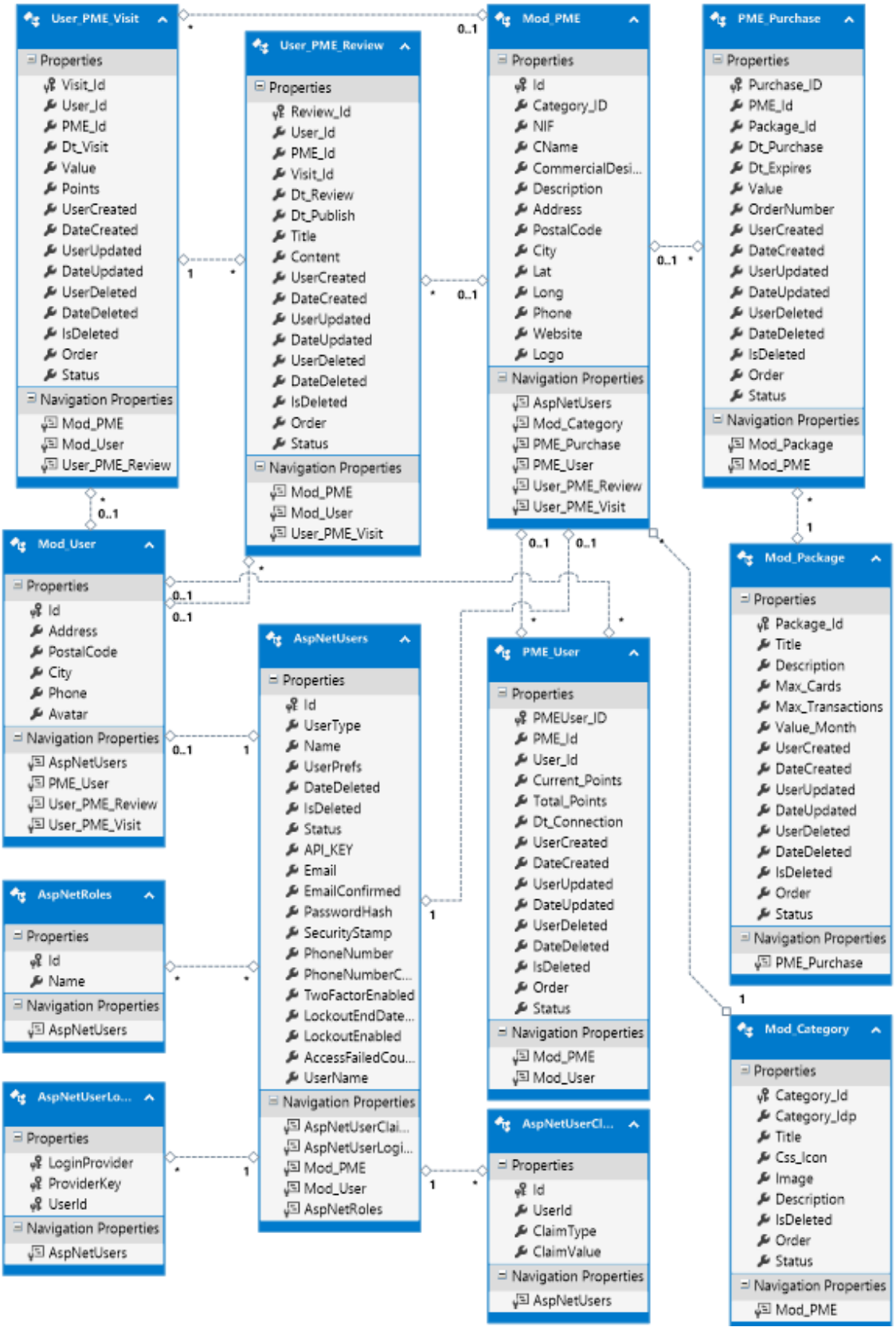


Figura 13 – Diagrama relacional da base de dados

3.2 Funcionalidades do sistema

No processo de desenvolvimento de sistemas, funcionalidade é definida como um comportamento ou uma ação para a qual possa ser visualizado um início e um fim, ou seja, algo passível de execução. As execuções de uma funcionalidade podem ser identificadas em termos de entrada e saída de entidades específicas ou de atributos pertencentes a entidades específicas. Por exemplo, a execução simples de uma funcionalidade chamada "Listar Empresas" lida com a entrada de uma certa informação sobre um particular documento e resulta na criação de uma instância da entidade "Empresa". Funcionalidades podem ser levantadas pela análise do ciclo de vida do negócio e do ciclo de vida das entidades, dentro do contexto do desenvolvimento de um projeto, identificando, então, as atividades necessárias para a criação e gestão do negócio e as entidades manipuladas por estes. A descrição de uma funcionalidade deve definir uma única execução sua, exprimindo o que ela faz e como ela o faz. Quem a executa, quando ela é executada ou como ela é executada não são questões fundamentais para a sua existência. É recomendável também apresentar na descrição o conjunto de pré-condições para uma execução da funcionalidade e as pós-condições que podem surgir dessa execução.

3.2.1 Portal

Dada a dimensão de funcionalidades associadas a cada um dos atores, a plataforma foi dividida em vários blocos lógicos, permitindo uma melhor perceção e organização do seu funcionamento. Cada bloco está relacionado com um dos perfis existentes no portal e atribuído a cada utilizador da plataforma.

3.2.1.1 Utilizador

Este perfil é atribuído a todos os registos efetuados através do portal e que tenham validado os dados introduzidos. A Figura 14 representa todas as funcionalidades propostas para o perfil Utilizador.

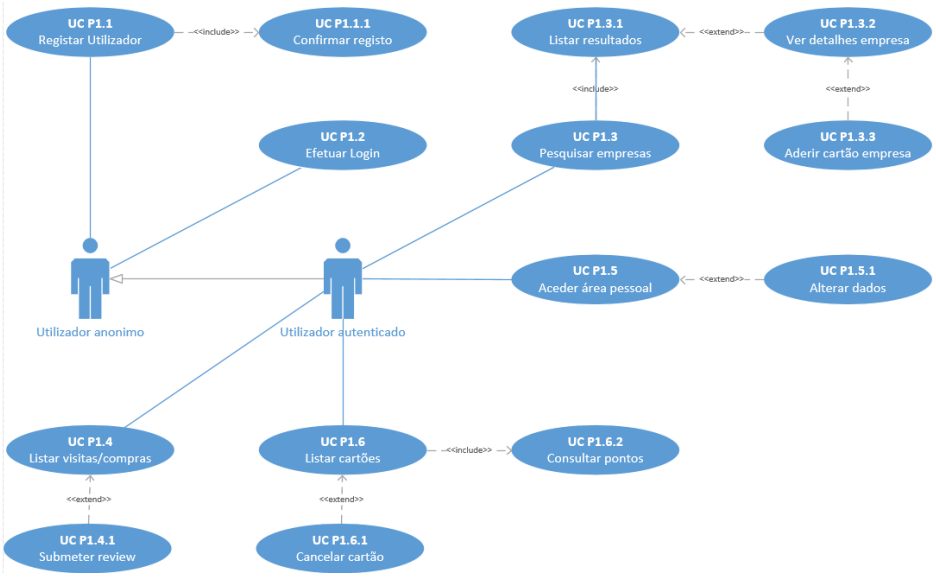


Figura 14 – Diagrama de casos de uso para o perfil Utilizador

Identificação: UC P1.1	
Nome: Registar Utilizador	
Atores: Utilizador não registado	
Tipo: primário	
Pré-Condições: Utilizador não deve estar validado no sistema	
Pós-Condições: deve ser enviado código ao utilizador para validar dados inseridos	
Sequência típica de eventos	
Ator	**Sistema**
1. Preenche informação no formulário	3. Valida informação preenchida e envia código para confirmação dos dados
2. Clica no botão para registar	
Sequência Alternativa	
3a. Número de telemóvel ou mail já registados	
1. Mensagem de erro e redireccionamento para página de login	
3b. Formulário contém erros	
1. Volta ao formulário de registo e apresenta os erros gerados	

Tabela 9 - Descrição do caso de uso “Registar utilizador”

Identificação: UC P1.1.1	
Nome: confirmar registo	
Atores: Utilizador não registado	
Tipo: primário	
Pré-Condições: Utilizador não deve estar validado no sistema	
Pós-Condições: Utilizador fica validado e autenticado no portal	
Sequência típica de eventos	
Ator	**Sistema**
1. Insere código de validação	3. Valida informação preenchida e valida o utilizador no sistema
2. Clica no botão para confirmar	
Sequência Alternativa	
3a. Código de validação inválido	
1. Redireccionamento para pagina de login e apresentação do erro gerado	

Tabela 10 - Descrição do caso de uso “Confirma registo”

Identificação: UC P1.3	
Nome: pesquisar empresas	
Atores: Utilizador não registado, Utilizador registado, empresa	
Tipo: primário	
Pré-Condições: nenhuma	
Pós-Condições: resultado da pesquisa deve ser retornado	
Sequência típica de eventos	
Ator	**Sistema**
1. Insere critérios de pesquisa	3. Valida informação preenchida e retorna resultados
2. Clica no botão para pesquisar	
Sequência Alternativa	
3a. Critérios inválidos	

1. É apresentado formulário de pesquisa com a mensagem de erro gerada

Tabela 11 - Descrição do caso de uso “Pesquisar empresas”

Identificação: UC P1.4.1	
Nome: submeter review	
Atores: Utilizador registado	
Tipo: primário	
Pré-Condições: Utilizador deve estar validado no sistema	
Pós-Condições: Review fica armazenado na base de dados a aguardar validação da empresa	
Sequência típica de eventos	
Ator	**Sistema**
1. Utilizador clica em listar visitas/compras	5. Sistema valida dados introduzidos
2. Seleciona visita a avaliar	
3. Preenche formulário	6. Sistema grava review e volta à listagem
4. Clica no botão para submeter review	
Sequência Alternativa	
5a. formulário contém erros	
1. É apresentado formulário com os erros gerados	
6a. ocorreu uma falha ao gravar dados 1. Sistema informa utilizador do erro e volta à listagem de reviews	

Tabela 12 - Descrição do caso de uso “Submeter Review”

Identificação: UC P1.6	
Nome: listar cartões	
Atores: Utilizador registado	
Tipo: primário	
Pré-Condições: Utilizador deve estar validado no sistema	
Pós-Condições: Todos os cartões do utilizador devem ser exibidos	
Sequência típica de eventos	
Ator	**Sistema**
1. Insere código de validação	3. Valida informação preenchida e valida o utilizador no sistema
2. Clica no botão para confirmar	
Sequência Alternativa	
3a. Código de validação inválido	
1. Mensagem de erro e redireccionamento para página de login	

Tabela 13 - Descrição do caso de uso “Listar cartões”

Identificação: UC P1.5	
Nome: aceder área pessoal	
Atores: Utilizador Não Registado	
Tipo: primário	
Pré-Condições: Utilizador deve receber código de confirmação	
Pós-Condições: Utilizador fica autenticado no portal	
Sequência típica de eventos	
Ator	**Sistema**
1. Clica no link A minha conta	3. Apresenta todos os dados referentes ao utilizador, incluindo pontos e dados pessoais
2.	
Sequência Alternativa	
3a. Ocorreu um erro ao obter os dados do utilizador	
1. Mensagem de erro e redireccionamento para página principal	

Tabela 14 - Descrição do caso de uso "Aceder área pessoal"

Nome	Descrição
UC P1.3.1 Listar resultados	Apresentação dos resultados relacionados com a pesquisa efetuada e os critérios definidos.
UC P1.3.2 Ver detalhes empresa	- Utilizador clica num resultado da pesquisa e obtém todos os detalhes da empresa em questão. - Se utilizador não estiver fidelizado à empresa então o sistema gera link que permite ao utilizador subscrever cartão de fidelização da empresa.
UC P1.3.3 Aderir cartão empresa	- Utilizador adere ao cartão de fidelização da empresa. - Pedido fica pendente até que a empresa valide a adesão.
UC P1.5.1 Alterar dados	- Utilizador ao aceder à área privada tem a possibilidade de alterar os seus dados pessoais.
UC P1.6.1 Cancelar cartão	- Utilizador cancela cartão de fidelização de uma determinada empresa.
UC P1.6.2 Consultar pontos	- Permite ao utilizador visualizar pontos associados às empresas onde possui cartão de fidelização.

Tabela 15 - Descrição dos restantes casos de uso para o perfil Utilizador

3.2.1.2 Empresa

Perfil atribuído a todos as empresas que subscrevem a plataforma GetBack. Este perfil herda todas as propriedades e funcionalidades do utilizador, além dessas são disponibilizadas funcionalidades que permitem à empresa gerir os cartões atribuídos, gerir subscrição e gerir reviews atribuídos. A listagem completa de funcionalidades pode ser consultada na Figura 15.

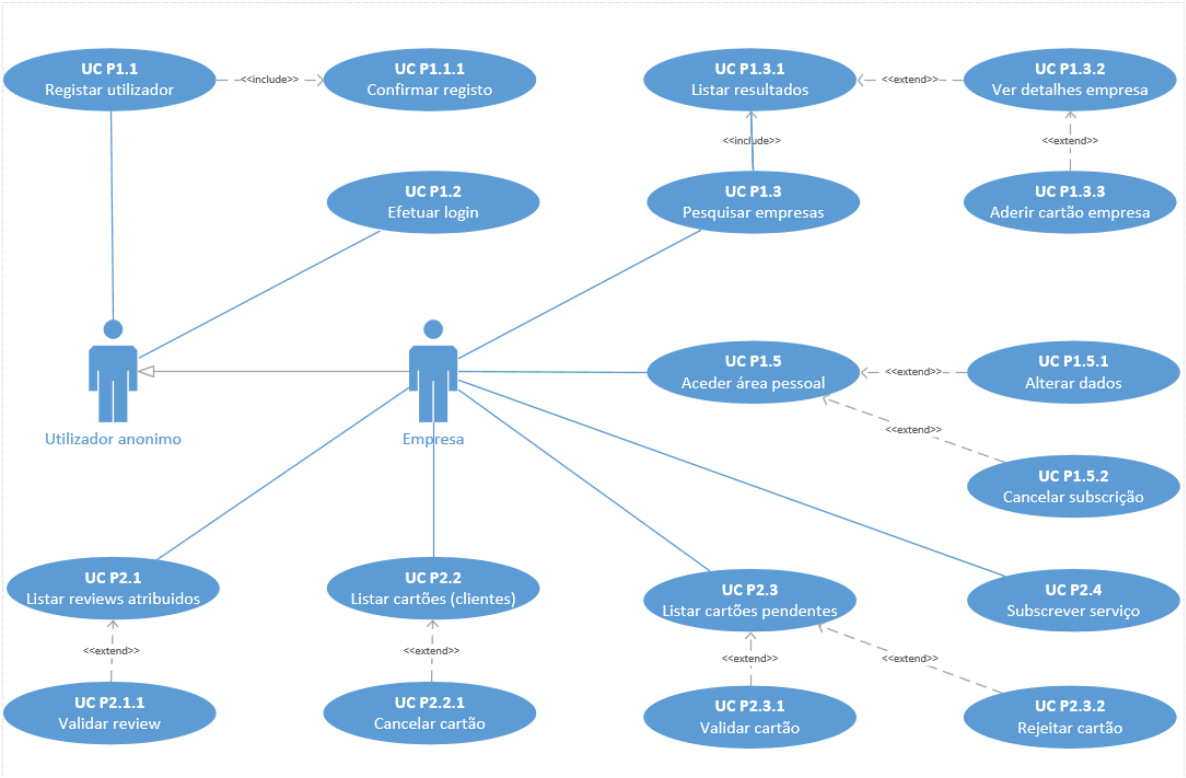


Figura 15 - Diagrama de casos de uso para o perfil Empresa

Identificação: UC P2.1.1	
Nome: validar review	
Atores: Utilizador registado	
Tipo: primário	
Pré-Condições: Utilizador deve estar validado no sistema e com perfil de empresa	
Pós-Condições: Review fica armazenado na base de dados a aguardar validação da empresa	
Sequência típica de eventos	
Ator	**Sistema**
1. Utilizador clica em listar visitas/compras	5. Sistema valida dados introduzidos
2. Seleciona visita a avaliar	
3. Preenche formulário	6. Sistema grava review e volta à listagem
4. Clica no botão para submeter review	
Sequência Alternativa	
5a. Formulário contém erros	
2. É apresentado formulário com os erros gerados	
6a. Ocorreu uma falha ao gravar dados 2. Sistema informa utilizador do erro e volta à listagem de reviews	

Tabela 16 - Descrição do caso de uso “Validar review”

Identificação: UC P2.2.1	
Nome: cancelar cartão	
Atores: Empresa	
Tipo: primário	
Pré-Condições: Utilizador deve estar validado e com permissões de empresa	
Pós-Condições: cartão foi removido do sistema	
Sequência típica de eventos	
Ator	**Sistema**
1. Clica em Listar cartões	4. Sistema elimina o cartão da base de dados
2. Seleciona cartão que pretende remover	
3. Empresa confirma cancelamento do cartão	
Sequência Alternativa	
4a. ocorreu um erro ao cancelar o cartão	
1. Mensagem de erro e redireccionamento para listagem dos cartões emitidos	

Tabela 17 - Descrição do caso de uso “Cancelar cartão”

Identificação: UC P2.4	
Nome: subscrever serviço	
Atores: Empresa	
Tipo: primário	
Pré-Condições: **1.** Utilizador deve estar validado e com permissões de empresa **2.** Empresa não subscreveu o serviço	
Pós-Condições: **1.** Subscrição fica associada à empresa **2.** Empresa recebe link e chave para download da aplicação mobile para empresas	
Sequência típica de eventos	
Ator	**Sistema**
1. Clica em Subscrever serviço	4. Sistema valida dados do formulário
2. Preenche formulário de adesão	
3. Clica em submeter formulário	5. Sistema gera referência para pagamento
Sequência Alternativa	
4a. ocorreu um erro ao cancelar o cartão	
1. Redireccionamento para formulário de adesão e apresentação do erro gerado	
5a. ocorreu uma falha ao gerar referência	
1. Redireccionamento para formulário de adesão e apresentação do erro gerado	

Tabela 18 - Descrição do caso de uso “Subscrever serviço”

Nome	Descrição
UC P1.5.2 Cancelar subscrição	- Empresa cancela serviço; Todos os cartões emitidos ficam bloqueados, mas permanecem no sistema havendo a possibilidade de os recuperar caso a empresa decida voltar a subscrever o serviço.
UC P2.1 Listar reviews atribuídos	- Todos os reviews atribuídos pelos clientes são listados; todos os reviews que ainda estão pendentes são marcados e é dada a possibilidade de validar esse registo.
UC P2.2 Listar cartões (clientes)	- Todos os Cartões validados pela empresa são listados; para cada um dos cartões é associado um link que permite cancelar esse cartão.
UC P2.3 Listar cartões pendentes	- Todos os Pedidos de cartão pendentes são listados; para cada um dos cartões são associados dois links, um que permite validar o pedido (UC P2.3.1) e outro que permite rejeitar o cartão (UC P2.3.2).
UC P2.3.1 Validar cartão	- Empresa aceita pedido de emissão de cartão de fidelização requerida pelo cliente.
UC P2.3.2 Rejeitar cartão	- Empresa rejeita pedido de emissão de cartão de fidelização requerida pelo cliente.

Tabela 19 - Descrição dos restantes casos de uso para o perfil Empresa

3.2.1.3 Administrador

Perfil atribuído aos administradores da plataforma, herda todas as funcionalidades atribuídas aos utilizadores e empresas, além dessas, são disponibilizadas funcionalidades que permitem gerir parâmetros, categorias, empresas e utilizadores (Figura 16).

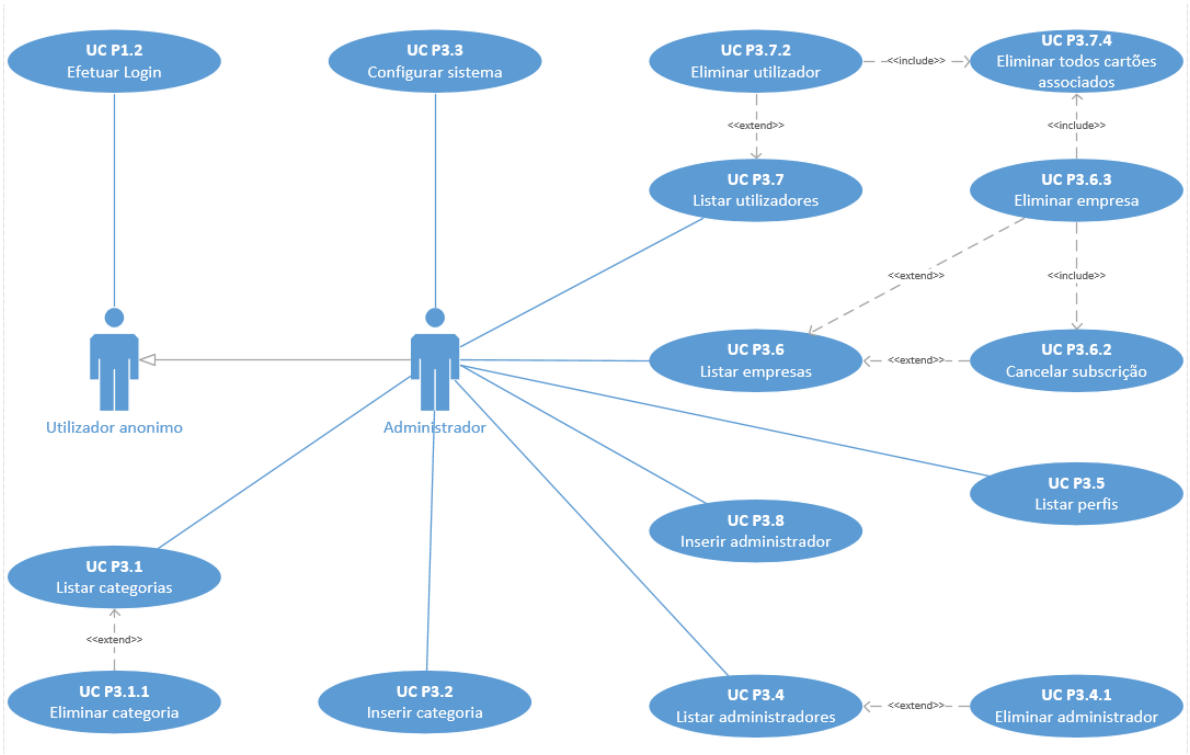


Figura 16 – Diagrama de casos de uso para o perfil Administrador

Identificação: UC P3.2	
Nome: inserir categoria	
Atores: Utilizador registado	
Tipo: primário	
Pré-Condições: Utilizador deve estar validado no sistema e com perfil de administrador	
Pós-Condições: Categoria é inserida na base de dados	
Sequência típica de eventos	
Ator	**Sistema**
1. Clica em inserir categoria	5. Sistema valida dados introduzidos
2. Preenche formulário	
3. Clica no botão para inserir categoria	6. Sistema grava dados e volta à listagem
Sequência Alternativa	
5a. Formulário contém erros	
1. Redireccionamento para o formulário e exibida mensagem de erro gerada	
5b. Nome da categoria já existe	
1. Redireccionamento para o formulário e exibida mensagem de erro gerada	
6a. ocorreu uma falha ao gravar dados 1. Sistema informa utilizador do erro e volta à listagem das categorias	

Tabela 20 - Descrição do caso de uso “Inserir categoria”

Identificação: UC P3.8

Nome: inserir administrador	
Atores: Utilizador registado	
Tipo: primário	
Pré-Condições: Utilizador deve estar validado no sistema e com perfil de administrador	
Pós-Condições: Administrador é inserido na base de dados	
Sequência típica de eventos	
Ator	**Sistema**
1. Clica em inserir administrador	5. Sistema valida dados introduzidos
2. Preenche formulário	
3. Clica no botão para inserir administrador	6. Sistema grava dados e volta à listagem
Sequência Alternativa	
5a. Formulário contém erros	
2. Redireccionamento para o formulário e exibida mensagem de erro gerada	
5b. Nome da categoria já existe	
2. Redireccionamento para o formulário e exibida mensagem de erro gerada	
6a. ocorreu uma falha ao gravar dados 2. Sistema informa utilizador do erro e volta à listagem das categorias	

Tabela 21 - Descrição do caso de uso “Inserir administrador”

Identificação: UC P3.6.3	
Nome: Eliminar empresa	
Atores: Utilizador registado	
Tipo: primário	
Pré-Condições: Utilizador deve estar validado no sistema e com perfil de administrador	
Pós-Condições: Empresa é removida da base de dados, cartões associados e subscrição ficam bloqueados.	
Sequência típica de eventos	
Ator	**Sistema**
1. Clica em listar empresas	5. Subscrição e cartões associados ficam bloqueados.
2. Clica no link eliminar referente à empresa pretendida	
3. Confirma intenção de eliminação	6. Empresa fica marcada com eliminada. Todas as funcionalidades relacionadas ficam bloqueadas.
Sequência Alternativa	
5a. Empresa não possui subscrição ativa	
1. Redireccionamento para a listagem de empresas e exibida mensagem de erro	
5b. ocorreu um erro ao bloquear subscrição	
1. Redireccionamento para a listagem de empresas e exibida mensagem de erro	
6a. ocorreu uma falha ao gravar dados 1. Redireccionamento para a listagem de empresas e exibida mensagem de erro	

Tabela 22 - Descrição do caso de uso “Eliminar empresa”

Nome	Descrição
UC P3.1 Listar categorias	- É apresentada uma listagem hierárquica com todas as categorias e subcategorias disponíveis; para cada registo é associado um link para eliminar a categoria (UC P3.1.1).
UC P3.1.1 Eliminar categoria	- A categoria é removida da base de dados.
UC P3.3 Configurar sistema	- Permite configurar parâmetros da aplicação.
UC P3.4 Listar administradores	- Listagem de todos os administradores; para cada registo é associado um link que permite eliminar o administrador (UC P3.4.1)
UC P3.4.1 Eliminar administrador	- Elimina o administrador
UC P3.5 Listar perfis	- Listagem de todos os perfis existentes com estatísticas sobre cada perfil (número de registos por perfil)
UC P3.6 Listar empresas	- Listagem de todas as empresas que têm ou tiveram uma subscrição na plataforma; para cada registo são associados dois links, um para cancelar a subscrição (UC P3.6.2) e outro para eliminar a empresa (UC P3.6.3).
UC P3.7 Listar utilizadores	- Listagem de todos os utilizadores que utilizam a plataforma; para cada registo é apresentado um link que permite eliminar o utilizador, é também apresentada a informação de quantos cartões o cliente possui.
UC P3.7.2 Eliminar utilizador	- Remove o utilizador; todos os cartões que o utilizador possuía são bloqueados.
UC P2.3.2 Rejeitar cartão	- Empresa rejeita pedido de emissão de cartão de fidelização requerida pelo cliente.

Tabela 23 - Descrição dos restantes casos de uso para o perfil Administrador

3.2.2 Utilizador (Aplicação Móvel)

A Figura 17 representa o diagrama de casos de uso para a aplicação móvel, sendo cada caso de uso detalhado nas respetivas tabelas subsequentes.

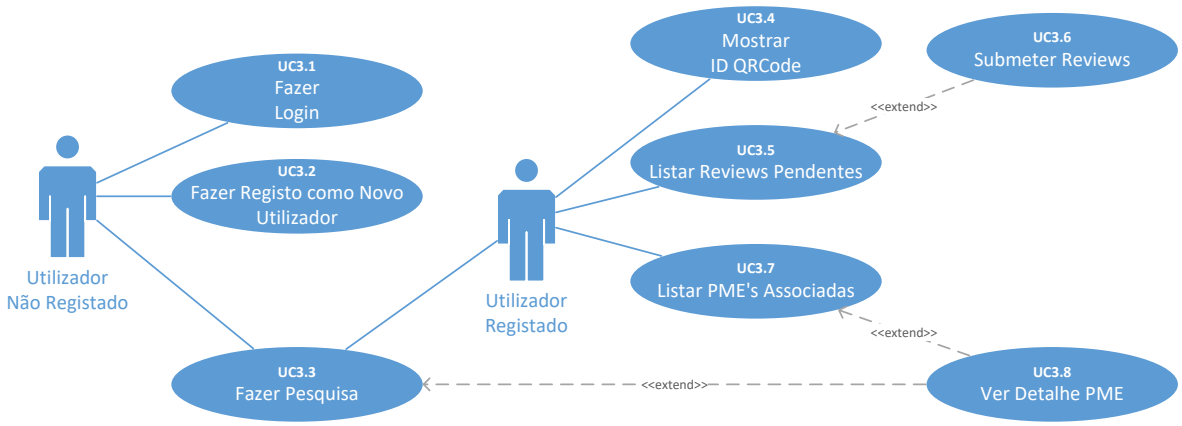


Figura 17 - Diagrama UseCase para aplicação móvel (utilizador)

Neste primeiro caso de uso é descrita a forma de autenticação do utilizador na aplicação.

Identificação: UC3.1	
Nome: Fazer Login	
Atores: Utilizador Não Registado	
Tipo: primário	
Pré-Condições: nenhuma	
Pós-Condições: Utilizador fica autenticado na aplicação	
Sequência típica de eventos	
Ator	**Sistema**
1. Preenche informação no formulário	3. Valida informação preenchida via API redirecionando utilizador para página de utilizador registado
2. Clica no botão para entrar autenticar	
Sequência Alternativa	
3a. Credenciais de utilizador inválidas	
2. Mensagem de erro e redireccionamento para página de login	
3b. Timeout no acesso a API	
2. Mensagem de erro e redireccionamento para página de login	

Tabela 24 - Descrição do caso de uso “Utilizador Não Registado”

No caso do utilizador não se encontrar ainda registado na aplicação deverá ser-lhe dada a possibilidade de efetuar registo. Esse processo é descrito no caso de uso presente na Tabela 25.

Identificação: UC3.2	
Nome: Fazer registo como novo utilizador	
Atores: Utilizador Não Registado	
Tipo: Primário	
Pré-Condições: nenhuma	
Pós-Condições: Pedido de registo de utilizador efetuado	
Sequência típica de eventos	
Ator	**Sistema**
1. Clica no botão para registar novo utilizador e é direcionado para formulário.	4. Valida informação preenchida no formulário

2. Insere dados no formulário	5. Efetua registo de utilizador na API, mensagem de sucesso e redireciona utilizador para view principal
3. Clica no botão para entrar autenticar	
Sequência Alternativa	
4a. Informação preenchida incorretamente	
1. Mensagem de erro e mantém utilizador no formulário	
5a Timeout no acesso a API	
1. Mensagem de erro e mantém utilizador no formulário	
5b. Utilizador com dados já registador	
1. Mensagem de erro e mantém utilizador no formulário	

Tabela 25 - Diagrama de Caso de Uso UC3.2

Um dos pontos mais importantes da aplicação é a sua capacidade de efetuar pesquisas sendo o nível de registo do utilizador uma variável indiferente. O processo de pesquisa encontra-se especificado no caso de uso presente na Tabela 26.

Identificação: UC3.3	
Nome: Fazer Pesquisa	
Atores: Utilizador Não Registado ou Utilizador Registado	
Tipo: Primário	
Pré-Condições: nenhuma	
Pós-Condições: Listagem de resultados obtidos	
Sequência típica de eventos	
Ator	**Sistema**
1. Clica no botão pesquisa	2. Efetua pedido de categorias disponíveis à API e direciona para página de seleção de categorias
3. Seleciona Categoria e é direcionado para página de especificação de critérios de pesquisa	5. Valida critérios de pesquisa e verifica posição GPS do utilizador
4. Especifica critérios de pesquisa	6. Efetua chamada de pesquisa à API e direciona utilizador para página com lista de resultados
7. Utilizador escolhe item da lista	8. É efetuado pedido de informação sobre item selecionado à API e redirecionar para página de detalhe (UC3.8)
Sequência Alternativa	
2a. Erro na API	
1. Mensagem de erro e mantém utilizador na página atual	
5a Informação preenchida incongruente	
2. Mensagem de erro e mantém utilizador no formulário	
5b Sem posicionamento GPS	
1. Mensagem de erro e mantém utilizador no formulário	
6a. Pesquisa sem resultados	
1. Mensagem de erro e mantém utilizador no formulário	
6b. Erro na API	
1. Mensagem de erro e mantém utilizador no formulário	
7a. Utilizador clica em “visualizar em mapa”	

1. Items aparecem em Pins sobre mapa da localização do utilizador
2. Executa passo 8
8a. Erro na API
3. Mensagem de erro e mantém utilizador na listagem de resultados

Tabela 26 – Diagrama de Caso de Uso “Fazer Pesquisa”

Precavendo a necessidade de alguns sistemas de terceiros efetuarem a identificação do utilizador via leitura de QRCode o caso de uso presente na Tabela 27 especifica a necessidade dessa implementação.

Identificação: UC3.4	
Nome: Mostra ID QRCode	
Atores: Utilizador Registado	
Tipo: Primário	
Pré-Condições: utilizador autenticado e no menu principal	
Pós-Condições: mostra página com identificador	
Sequência típica de eventos	
Ator	**Sistema**
1. Clica no botão para mostrar identificador	2. Gera Imagem QRCode com base no identificador o utilizador e redireciona para página com detalhe

Tabela 27 - Diagrama de Caso de Uso “Mostra ID QRCode”

Uma outra necessidade que foi identificada é a capacidade dos utilizadores efetuarem reviews aos estabelecimentos que visitam por isso é necessário que estes recebam na aplicação indicação de quais os reviews que ainda não foram preenchidos (Tabela 28) e submetidos (Tabela 29).

Identificação: UC3.5	
Nome: Lista Reviews Pendentes	
Atores: Utilizador Registado	
Tipo: Primário	
Pré-Condições: nenhuma	
Pós-Condições: direcionado para formulário de submissão de review	
Sequência típica de eventos	
Ator	**Sistema**
1. Clica no botão para listar reviews pendentes	2. Efetua chamada a API para receber Reviews pendentes de resposta e redireciona para listagem de reviews
3. Clica em review a preencher	4. Efetua chamada a API para receber informação das questões a preencher e redireciona para formulário de submissão de review (UC3.6)
Sequência Alternativa	
2a Timeout no acesso a API	
1. Mensagem de erro e mantém utilizador no menu principal	
2b Sem reviews pendentes	

1. Mensagem de informação e mantém utilizador no menu principal
4a Timeout no acesso a API
1. Mensagem de erro e mantém utilizador na listagem de reviews pendentes

Tabela 28 - Diagrama de Caso de Uso UC3.5

Identificação: UC3.6	
Nome: Submeter Review	
Atores: Utilizador Registado	
Tipo: primário	
Pré-Condições: tem reviews pendentes	
Pós-Condições: Pedido de registo de utilizador efetuado	
Sequência típica de eventos	
Ator	**Sistema**
1. Preenche formulário e clica em botão de submissão	2. Efetua submissão review na API messagem de sucesso e redireciona para listagem de reviews pendentes
Sequência Alternativa	
5a Timeout no acesso a API	
1. Mensagem de erro e mantém utilizador na listagem de reviews pendentes	

Tabela 29 - Diagrama de Caso de Uso UC3.6

A aplicação não faria sentido se o utilizador registado não conseguisse rapidamente aceder aos seus "cartões virtuais", ou seja, às empresas às quais se encontra associado (Tabela 30) e visualizar o detalhe essa mesma empresa (Tabela 31).

Identificação: UC3.7	
Nome: listar empresas associadas	
Atores: Utilizador Registado	
Tipo: primário	
Pré-Condições: Utilizador registado	
Pós-Condições: Listagem de resultados obtidos	
Sequência típica de eventos	
Ator	**Sistema**
1. Clica em botão "As Minhas Empresas"	2. Efetua chamada a API redireciona para listagem de empresas associadas
3. Utilizador escolhe item da lista	4. É efetuado pedido de informação sobre item selecionado à API e redirecionar para página de detalhe (UC3.8)
Sequência Alternativa	
2a Erro no acesso a API	
1. Mensagem de erro e mantém utilizador no menu principal	
3a utilizador pesquisa por texto	
1. Listagem é atualizada com base no critério de pesquisa	
4a Erro no acesso a API	

2. Mensagem de erro e mantém utilizador na lista de empresas associadas

Tabela 30 - Diagrama de Caso de Uso “Listar empresas associadas”

Identificação: UC3.8	
Nome: Ver detalhe empresa	
Atores: Utilizador Registado e Utilizador Não registado	
Tipo: Primário	
Pré-Condições: Originar em Lista de Pesquisa ou em Lista de PMEs Associadas	
Pós-Condições: Página com Detalhe da empresa	
Sequência típica de eventos	
Ator	**Sistema**
	1. Recebe informação da PME
	2. Apresenta página com detalhes de PME
3. Visualiza página com informação sobre PME	
Sequência Alternativa	
2a Página para Utilizador Registado	
1. Inclui bloco na página a apresentar com informação exclusiva de utilizador registado	
3a Clica em detalhe de fotos	
1. Visualiza foto escolhida	
3b Clica em detalhe de Review	
1. Visualiza review escolhido	

Tabela 31 - Diagrama de Caso de Uso “Ver detalhe empresa”

3.2.3 Empresas (Aplicação Móvel)

As empresas também necessitam de interagir com a plataforma de uma forma rápida e ágil no seu dia a dia e, apesar de toda a solução estar preparada para receber interações por intermédio de software de 3ºs através da sua API, julgamos essencial que as empresas não fiquem dependentes dessa interação. Assim sendo torna-se necessária uma plataforma móvel que possibilite esta interação através de dispositivos de baixo custo. O diagrama presente na Figura 18 ilustra as funcionalidades básicas esperadas para essa aplicação.

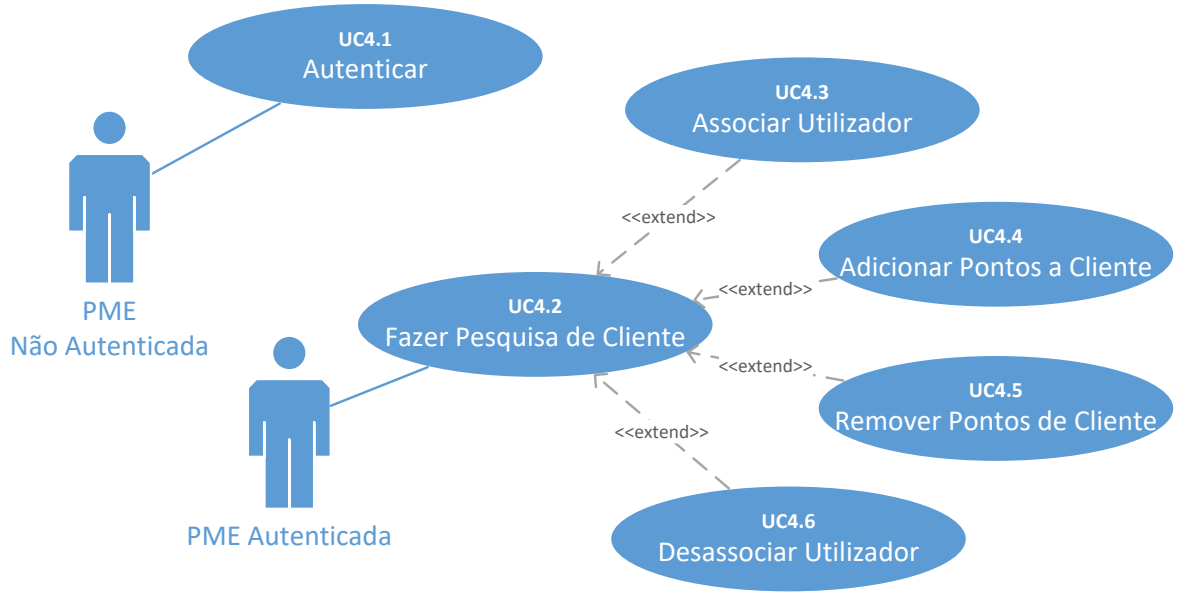


Figura 18 - Diagrama UseCase para aplicação móvel (utilizador)

O caso de uso presente na Tabela 32 escreve a forma de autenticação na aplicação.

Identificação: UC4.1	
Nome: Autenticar	
Atores: PME Não Autenticada	
Tipo: Primário	
Pré-Condições: nenhuma	
Pós-Condições: Utilizador fica autenticado na aplicação	
Sequência típica de eventos	
Ator	**Sistema**
1. Preenche informação no formulário	3. Valida informação preenchida via API redirecionando para página com menu principal
2. Clica no botão para entrar autenticar	
Sequência Alternativa	
3a. Credenciais inválidas	
1. Mensagem de erro e redireccionamento para página de login	
3b. Timeout no acesso a API	
1. Mensagem de erro e redireccionamento para página de login	

Tabela 32 - Diagrama de Caso de Uso “PME Não Autenticada”

Identificação: UC4.2
Nome: Fazer Pesquisa de Cliente

Atores: PME Autenticada	
Tipo: primário	
Pré-Condições: PME tem que estar autenticada	
Pós-Condições: encontra utilizador e redireciona para página com opções	
Sequência típica de eventos	
Ator	**Sistema**
1. Preenche informação no formulário	3. Efetua chamada a API redireciona para página com menu de opções
2. Clica no botão "Procurar Utilizador"	
Sequência Alternativa	
3a. Utilizador não encontrado	
1. Mensagem de erro e mantém em página com formulário de pesquisa	
3b. Utilizador não associado	
1. UseCase UC.4.3	
3c. Erro no acesso a API	
1. Mensagem de erro e redireccionamento para página de login	

Tabela 33 - Diagrama de Caso de Uso "Fazer Pesquisa de Cliente"

Identificação: UC4.3	
Nome: Associar Utilizador	
Atores: PME Autenticada	
Tipo: primário	
Pré-Condições: PME tem que estar autenticada	
Pós-Condições: associa utilizador à empresa	
Sequência típica de eventos	
Ator	**Sistema**
	1. Mostra Mensagem com questão se pretende associar utilizador.
2. Clica no botão "Sim"	3. Associa utilizador à empresa redireciona para menu de utilizador
Sequência Alternativa	
2a. Clica no botão "Não"	
2. Redireciona para a página com formulário de pesquisa	

Tabela 34 - Diagrama de Caso de Uso "Associar Utilizador"

Identificação: UC4.4
Nome: Adicionar Pontos a Utilizador

Atores: PME Autenticada	
Tipo: primário	
Pré-Condições: PME tem que estar autenticada; Utilizador tem que estar associado	
Pós-Condições: Adiciona pontos a Utilizador	
Sequência típica de eventos	
Ator	**Sistema**
1. Clica em “Adicionar Pontos”	2. Redireciona para página de adição de pontos
3. Insere informação no formulário	5. Executa Envio para API dos Pontos, mostra mensagem de sucesso e volta à página com menu de utilizador.
4. Clica em “Adicionar Pontos a Utilizador”	
Sequência Alternativa	
5a. Erro no acesso a API	
1. Mensagem de erro e redireccionamento para página de login	

Tabela 35 - Diagrama de Caso de Uso “Adicionar Pontos a Utilizador”

Identificação: UC4.5	
Nome: Subtrair Pontos a Utilizador	
Atores: PME Autenticada	
Tipo: primário	
Pré-Condições: PME tem que estar autenticada; Utilizador tem que estar associado	
Pós-Condições: subtrai pontos a Utilizador	
Sequência típica de eventos	
Ator	**Sistema**
1. Clica em “Subtrair Pontos”	2. Redireciona para página de subtração de pontos
3. Insere informação no formulário	5. Verifica se existem pontos suficientes a serem subtraídos
4. Clica em “Adicionar Pontos a Utilizador”	6. Executa Envio para API dos Pontos a remover, mostra mensagem de sucesso e volta à página com menu de utilizador.
Sequência Alternativa	
5a. Não existem pontos suficientes	
1. Mensagem de erro e redireccionamento para página de subtração de pontos	
6a. Erro no acesso a API	
2. Mensagem de erro e redireccionamento para página de login	

Tabela 36 - Diagrama de Caso de Uso “Subtrair Pontos a Utilizador”

3.3 Tecnologia Relevante

Depois de definidas as funcionalidades de utilizador e do sistema, foi efetuada uma análise ao estado da arte em tecnologia relevante para escolher aquela que reúne melhores condições para o desenvolvimento da solução proposta.

Para simplificar a análise, agrupamos os requisitos em quatro grandes áreas:

- WEB
- MOBILE
- API
- SGBD

3.3.1 WEB

A world wide web, também conhecida como web ou www, é um sistema de documentos em hipermédia que são interligados e estão disponíveis na net através de um endereço especifico. Inicialmente estas páginas eram estáticas e baseadas na linguagem de marcação HTML, CSS e Javascript. Posteriormente, com a necessidade de criação de páginas mais complexas e dinâmicas surgiram diversas linguagens de programação server-side (geração do conteúdo é efetuado pelo servidor e retornado para o utilizador o resultado). Sendo a plataforma Getback totalmente dinâmica, efetuou-se uma análise às principais tecnologias existentes para escolher qual a ser usada no desenvolvimento da solução.

3.3.1.1 ASP.NET MVC (C#)

O ASP.NET MVC (ASP.NET MVC, 2016) é uma framework web gratuita, desenvolvida pela Microsoft (Microsoft, 2016) que implementa o padrão MVC. Recentemente houve uma fusão do ASP.NET MVC, ASP.NET Web API e ASP.NET Web Pages, inicialmente com o nome ASP.NET Vnext e, depois, batizada de ASP.NET Core (ASP.NET core, 2016). Além de ser disponibilizada à comunidade como código-aberto, esta nova versão poder ser instalada em servidores Linux (The Linux Foundation, 2016).

O padrão MVC (Model-View-Controller) permite a separação da aplicação web em três componentes principais: modelo (model), vista (view) e controlador (controller). A estrutura ASP.NET MVC oferece uma alternativa ao padrão Web Forms do ASP.NET para criar aplicações web. A estrutura ASP.NET MVC é uma estrutura de apresentação leve e altamente testável que (à semelhança das aplicações baseados em Web Forms) é integrada aos recursos ASP.NET existentes. A estrutura MVC é definida na libraria System.Web.Mvc. Na Figura 19 está representado o ciclo de vida de um pedido efetuado ao servidor, através da framework ASP.NET MVC.

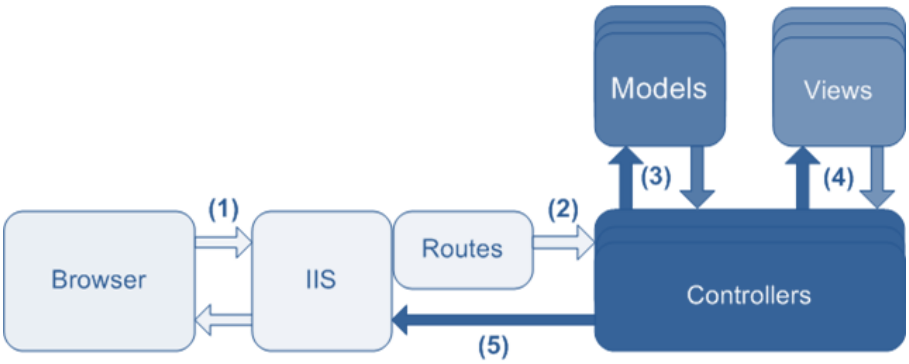


Figura 19 – Ciclo de vida de um pedido em ASP.NET MVC (ilyaigpetrov, 2007)

VANTAGENS	DESVANTAGENS
- Separação de conceitos simples e intuitiva	- Curva de aprendizagem considerável
- IDE (VS Studio 2015) de desenvolvimento completo e potente	- Upgrade de versão gratuita para versão paga do IDE é dispendiosa
- Extensível	
- Documentação completa com exemplos práticos	
- Facilmente testável (TDD)	
- Código compilado	
- Focado na segurança e estabilidade	

Tabela 37 - Vantagens e desvantagens da framework ASP.NET MVC

3.3.1.2 PHP

Em alternativa ao ASP.NET MVC, foi analisado o PHP (The PHP Group, 2016) , uma linguagem de programação muito popular e amplamente utilizada no desenvolvimento de aplicações web dinâmicas. De referir que alguns dos maiores sites mundiais utilizam o PHP no desenvolvimento das suas plataformas, por exemplo, Facebook e Google.

PHP é uma linguagem de programação gratuita, desenvolvida inicialmente por Rasmus Lerdorf em 1995. Tem uma grande comunidade de programadores que apoiam e contribuem para a constante evolução da linguagem. Apesar de ser possível o desenvolvimento de grandes projetos em PHP nativo, é comum a utilização de uma ou várias frameworks para implementar as tarefas mais comuns. No entanto, para projetos mais complexos o tempo de desenvolvimento tende a ser superior e há maior risco de existir erros.

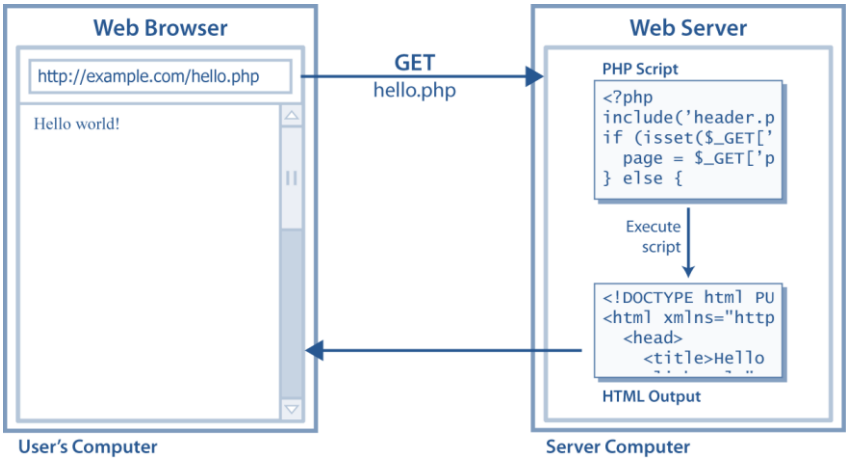


Figura 20 – Ciclo de vida de um pedido em PHP (Stepp & Miller, 2009).

A Figura 20 representa um pedido efetuado ao servidor de um script PHP, o qual retorna o HTML gerado após execução local do ficheiro. Desenvolver aplicações web pode ser extremamente complexo e demorado, para facilitar este processo surgiram diversas frameworks.

Ao utilizar uma framework estamos a simplificar e a acelerar o desenvolvimento (reutilização de componentes genéricos e módulos) e a uniformizar o desenvolvimento, respeitando as regras definidas pela própria biblioteca. Baseadas em PHP, a oferta de bibliotecas é extensa, no entanto o gráfico seguinte enumera as mais utilizadas no desenvolvimento. A Figura 21 apresenta a lista das frameworks mais populares.

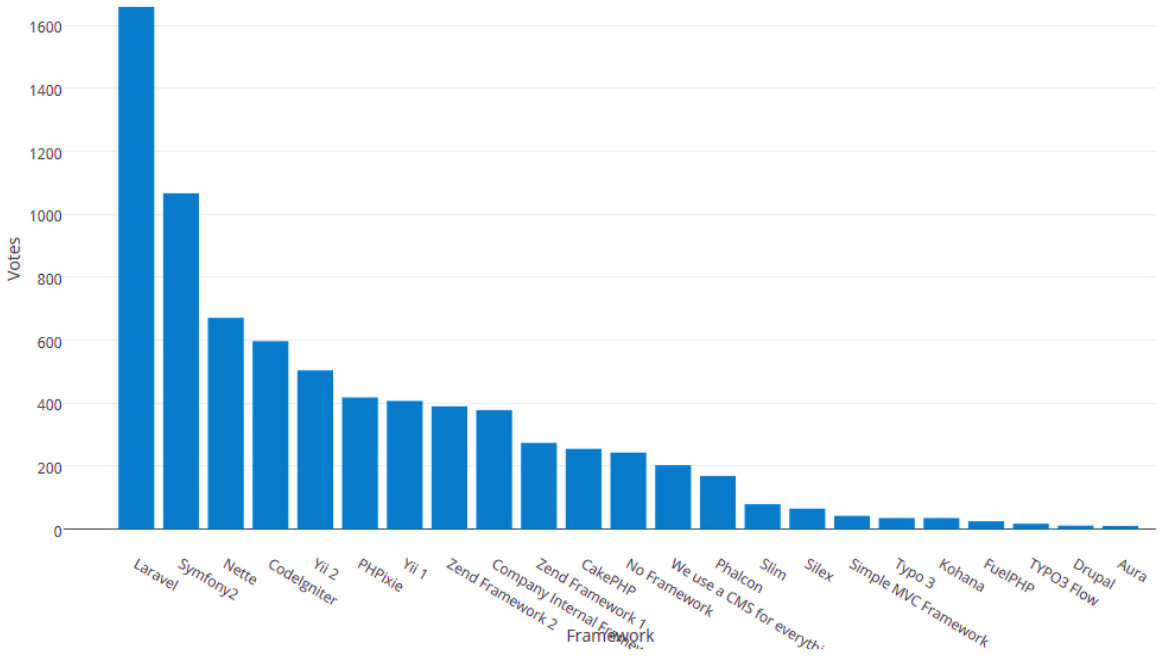


Figura 21 – Distribuição de preferências de bibliotecas baseadas em PHP (Swader, 2015)

VANTAGENS	DESVANTAGENS
- Controlo total sobre o código desenvolvido	- Maior tempo e esforço no desenvolvimento
- Desenvolvimento feito à medida e otimizado para as nossas necessidades	- Custos superiores, todo o código é desenvolvido de raiz e à medida
- Apenas são desenvolvidos módulos necessários	

Tabela 38 - Vantagens e desvantagens no desenvolvimento de código PHP sem framework

VANTAGENS	DESVANTAGENS
- Curva de aprendizagem simples	- Sintaxe das funções inconsistente
- Comunidade de desenvolvimento alargada e muito dinâmica	- IDE gratuitos são limitados, os que contêm funcionalidades mais interessantes são pagos (*syntax highlighting*, *code completion*)
- Grande variedade de frameworks gratuitas	- Código nativo não promove boas práticas
- Documentação completa com exemplos práticos	- É recomendada a utilização de uma framework, obriga a aprendizagem inicial.

Tabela 39 - Vantagens e desvantagens na utilização do PHP

Com exceção das empresas que pretendem ter controlo total sobre o código que desenvolvem e/ou necessitam de funcionalidades especificas, é recomendada a utilização de uma framework para agilizar o desenvolvimento. A grande desvantagem é a necessidade de despender tempo aprender a arquitetura da framework, como está organizada, etc… Para alem desta desvantagem, cada framework é implementada de forma diferente.

3.3.2 Aplicações móveis (utilizador e empresa)

O principal ponto de debate nesta área envolve linguagens nativas e HTML5, já que a base de programação de uma aplicação móvel poderá ser inteiramente suportada pela linguagem nativa do dispositivo (Android, iOS e Windows Phone possuem linguagens nativas diferentes entre si) ou por HTML5, linguagem transversal a todos os dispositivos móveis quando disponibilizada num browser. Esta última opção denomina-se aplicação web (web app) pois tal como um website engloba de forma generalizada HTML5, apoiado por CSS e JavaScript. Há ainda a possibilidade de fazer uso das duas opções na mesma aplicação, ou seja, criar aplicações híbridas (hybrid apps).

Como é óbvio, construir uma aplicação que faz uso das mesmas linguagens do sistema operativo que a acolhe (linguagens de programação nativas) pressupõe um maior poder de acesso aos recursos do sistema, eficiência e independência. Entende-se, portanto, que quando as aplicações móveis (mobile app) começaram a difundir-se em larga escala, fazer uso das linguagens nativas para o seu desenvolvimento era a escolha por excelência. Porém cedo se percebeu que em determinados contextos um website devidamente otimizado para dispositivos móveis poderia obter o mesmo nível de experiência de utilização quando comparado a uma aplicação nativa. O surgimento e evolução do

HTML5 foi um grande impulso para agilizar as web apps como alternativa às aplicações nativas (native apps).

3.3.2.1 Apache Cordova/ Phonegap

Apache Cordova (The Apache Software Foundation, 2015) é uma plataforma gratuita para desenvolvimento de aplicações móveis. Facilita o desenvolvimento de aplicações móveis com recurso às tecnologias web HTML5, CSS3 e Javascript e permite o acesso a funções nativas móveis (acelerômetro, câmara, geolocalização, etc..) dos aparelhos em que será executada. As aplicações resultantes são híbridas, no sentido de que não são aplicações móveis nativas, mas também não são aplicações web. Em vez de utilizar a UI nativa de cada plataforma, a geração do layout é feita por webviews e têm acesso à API nativa de cada dispositivo via Javascript.

Phonegap (Adobe Systems Inc., 2016) é um projeto baseado no Apache Cordova, contém as mesmas funcionalidades. A principal diferença é a capacidade de compilar código remoto, enquanto a versão original apenas permite que o mesmo seja compilado localmente.

VANTAGENS	DESVANTAGENS
- Código único, múltiplos sistemas operativos	- Performance reduzida
- Comunidade alargada e bastante ativa	- Documentação limitada
- Baseado em tecnologias atuais HTML5/CSS/JS	
- Gratuito	

Tabela 40 - Vantagens e desvantagens do Apache Cordova/Phonegap

3.3.2.2 Titanium

O desenvolvimento de aplicações em Titanium (Appcelerator Inc. , 2016) é através da linguagem de programação javascript, porém também requer XML para desenvolver interfaces personalizados e a API fornecida pela Appcelerator. Há relatos de utilizadores sobre demoras pontuais no carregamento das bibliotecas necessárias pelo Titanium. Para além de ser necessário dominar Javascript, esta plataforma requer a aprendizagem da API Titanium. O desenvolvimento de aplicações mais complexas torna-se um processo árduo, com grande possibilidade de surgirem erros desconhecidos e comportamentos estranhos por parte da aplicação (Clarity Ventures, Inc., 2016).

VANTAGENS	DESVANTAGENS
- Reutilização entre 60% e 90% do código	- Serviço é pago
- Acesso direto à API via Hyperloop	- Requer aprendizagem de API especifica
- Gera aplicações nativas	- Só suporta IOS e Android
- Performance	

Tabela 41 - Vantagens e desvantagens da plataforma Titanium

3.3.2.3 Xamarin

A plataforma Xamarin (Xamarin Inc., 2016) permite o desenvolvimento de aplicações nativas para as principais plataformas móveis, através da linguagem de programação C#. Disponibiliza ferramentas que permitem reduzir a duplicação de código, tornando possível desenvolver aplicações móveis para Android e iOS usando a mesma linguagem de programação. No entanto, não elimina completamente a necessidade de esforço duplicado: para o desenvolvimento da interface gráfica, requer código específico para cada sistema operativo. É, porém, possível partilhar todo o código referente a regras de negócio, comunicação com o servidor e outros elementos que não dependam de interação com o utilizador.

VANTAGENS	DESVANTAGENS
- Reutilização entre 60% e 100% do código	- Requer conhecimentos de c#
- Excelente documentação e exemplos práticos	- Incoerência na informação disponibilizada
- Gera aplicações nativas	- UI requer código especifico por plataforma
- Performance	
- Integração com o Visual Studio 2015	
- Código partilhado entre projetos (API/WEB)	

Tabela 42 - Vantagens e desvantagens da plataforma Xamarin

3.3.3 API

Interface de Programação de Aplicações, cujo acrónimo API provém do Inglês Application Programming Interface, é um conjunto de rotinas e padrões estabelecidos por um software para permitir o acesso de outras aplicações às suas funcionalidades. De modo geral, a API é composta por uma série de funções disponíveis por programação e que permitem acesso a caraterísticas da aplicação, invisíveis para o utilizador.

A API assume um papel importante na solução a desenvolver, é crucial garantir alta disponibilidade, performance e que seja escalável. Para a sua implementação foram analisadas duas soluções distintas, ambas utilizadas a nível académico e profissional, Node.js (Node.js Foundation, 2016) com Express framework (StrongLoop, 2016) e ASP.NET Web API 2 (Microsoft, 2016). Serão referidas outras alternativas existentes para conhecimento.

3.3.3.1 Node.js + Express Framework

O Node.js é um interpretador de código JavaScript que funciona do lado do servidor. Esta plataforma permite aos programadores o desenvolvimento de aplicações em rede, em tempo real e de **alta escalabilidade**, de uma forma simples e rápida. O Node.js é baseado no interpretador V8 da Google (Google, 2016). As operações de I/O são realizadas de forma assíncrona conforme descreve a Figura 22.

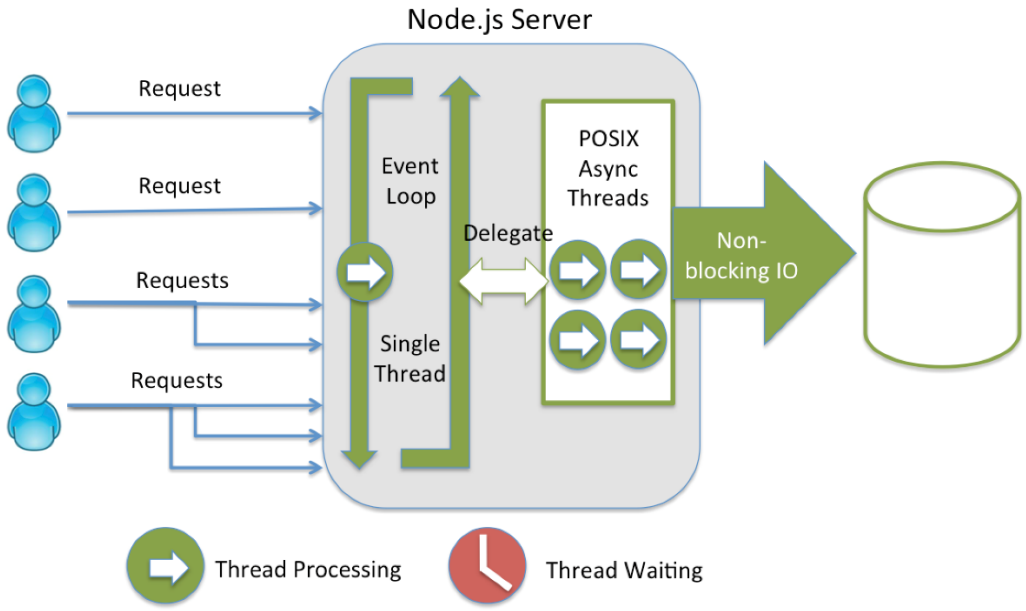


Figura 22 – Processamento de pedidos em Node.js (ilyaigpetrov, 2007)

Express framework é um pacote para Node.js e permite o desenvolvimento de API robustas e de alto desempenho num curto espaço de tempo. A Figura 23 define o diagrama de processamento de um pedido em Express.

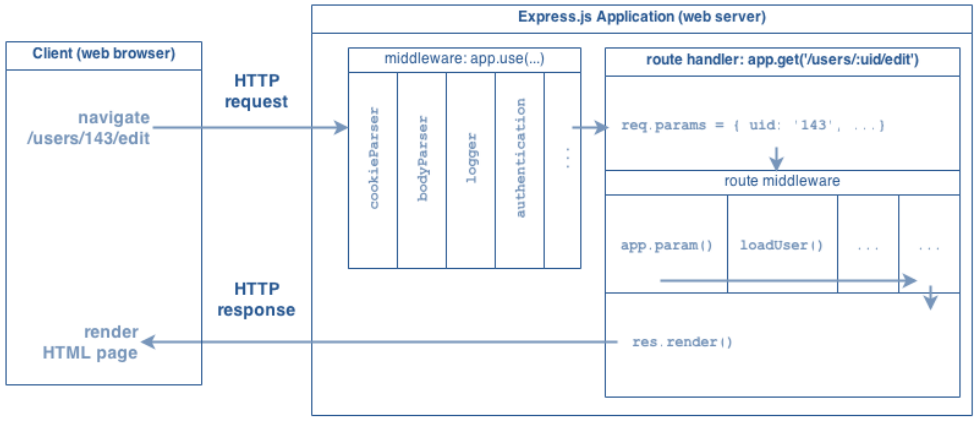


Figura 23 - Sequência de processamento de um pedido em Express (Mejia, 2016)

VANTAGENS	DESVANTAGENS
- Otimizado para operações de I/O	- Bloqueia em tarefas intensivas de CPU
- Operações de I/O assíncronas	- Não providencia escalabilidade
- Robustez E performance	- Acesso a base de dados é complicado
- Grande variedade de pacotes disponíveis	- Apenas uma thread para receber pedidos

Tabela 43 - Vantagens e desvantagens da plataforma Node.JS

3.3.3.2 ASP.NET Web API 2

Desenvolvida pela Microsoft, ASP.NET Web API é uma framework que simplifica o desenvolvimento de serviços expostos na web (API), usando o protocolo HTTP, para serem utilizados por uma gama alargada de clientes, incluindo aplicações web e dispositivos móveis. As operações de I/O são bloqueantes conforme representado na Figura 24.

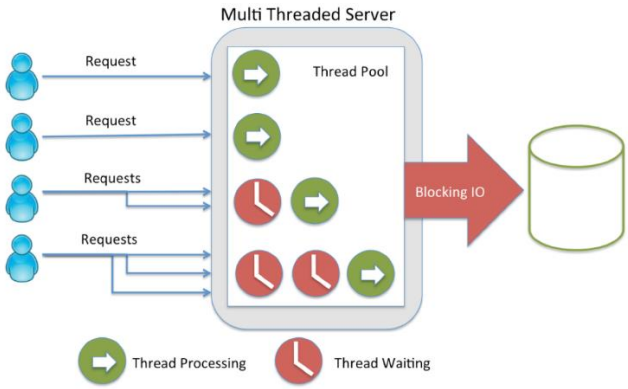


Figura 24 – Processamento de pedidos em ASP.NET WEB API (ilyaigpetrov, 2007)

VANTAGENS	DESVANTAGENS
- Documentação E exemplos funcionais	- Performance
- Escalabilidade	
- Integração com o Visual Studio 2015	
- Possível integrar com outros tipos de projetos (mobile/web)	

Tabela 44 - Vantagens e desvantagens da framework ASP.NET WEB API

Na Figura 25 são apresentados os resultados dos testes de saturação efetuados para verificar a performance das duas frameworks analisadas.

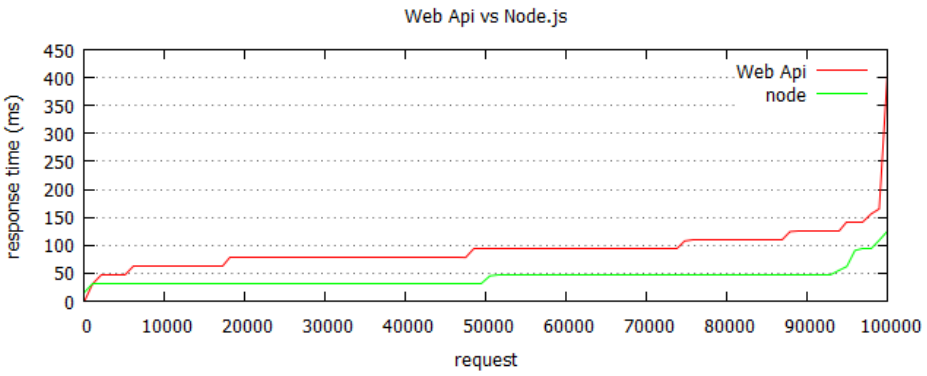


Figura 25 - Comparação de performance entre Node.js + Express e Web API (Mikael, 2012)

3.3.4 SGBD – Sistema de Gestão de Base de Dados

Um Sistema de Gestão de Bases de Dados (SGBD) é o software que gere o armazenamento, manipulação e pesquisa dos dados existentes na base de dados, funcionando como uma interface entre as aplicações e os dados necessários para a execução dessas aplicações.

Para garantir uma resposta rápida e eficaz em qualquer situação, a plataforma é, toda ela, baseada em serviços na *nuvem*. A nível de base de dados comparamos três alternativas com provas dadas ao longo dos últimos anos.

3.3.4.1 Mysql

O **MySQL** (Oracle Corporation , 2016) é um sistema de gestão de base de dados (SGBD), que utiliza a linguagem SQL (*Structured Query Language*) como interface. Foi desenvolvido na Suécia na década de 1980. É, atualmente, um dos sistemas mais populares para gestão de base de dados. O seu sucesso deve-se em grande medida à fácil integração com a linguagem de programação PHP, o que permite a criação de aplicações dinâmicas com relativa simplicidade.

Praticamente está disponível para todas as plataformas atuais (Linux, Windows, etc..) e disponibiliza diversos drivers para integração com as mais variadas linguagens de programação, tais como Java, C/C++, C#, Ruby, entre outras.

VANTAGENS	DESVANTAGENS
- Portabilidade	- Serviço para nuvem não é nativo
- Compatibilidade	- Pouco escalável
- Excelente desempenho e estabilidade	
- Diversos motores de armazenamento	
- Gestão simples	
- Replicação fácil de implementar	
- Gratuito	

Tabela 45 - Principais vantagens e desvantagens do SGBD MySQL

3.3.4.2 Microsoft SQL Server Azure

O Microsoft Azure SQL DB (Microsoft, 2016) (também conhecido como SQL Azure) é um serviço de base de dados na *cloud,* e integra a plataforma de serviços Azure (Microsoft, 2016). Este SGDB é baseado numa versão especial do SQL Server. Esta versão não implementa toda as funcionalidades do SQL Server (Microsoft, 2016), incluindo diagramas da base de dados.

Tem como principais vantagens a facilidade de gestão e redução de custos na administração dos sistemas, conforme se pode analisar na Figura 26.

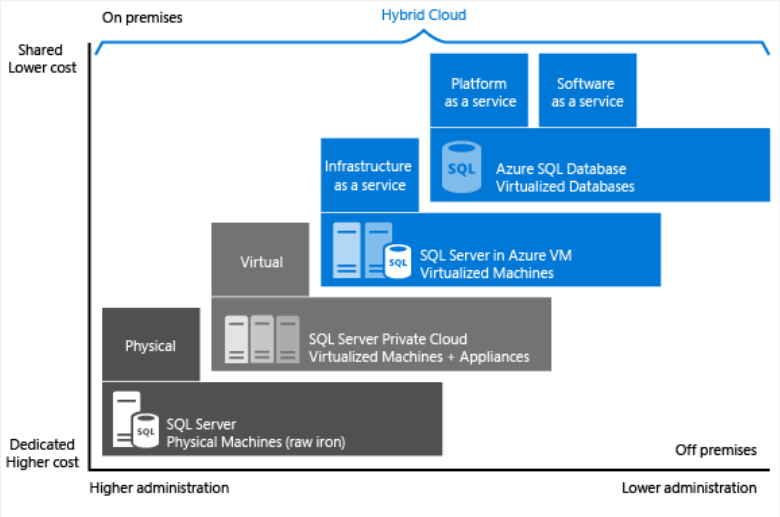


Figura 26 – Relação custo/administração em versões SQL SERVER (Microsoft, 2016)

VANTAGENS	DESVANTAGENS
- Suporte nativo para cloud	- Funcionalidades limitadas em relação ao SQL SERVER
- Desempenho E estabilidade	- Produto pago
- Redução nos custos de administração	
- Integração com Visual Studio 2015	

Tabela 46 – Vantagens e desvantagens da utilização do Microsoft SQL Server Azure

3.3.5 Outras tecnologias

Para complementar o desenvolvimento da plataforma, com a intenção de agilizar e simplificar o processo, foram utilizadas as seguintes frameworks:

Bootstrap (mdo & fat, 2016)	É uma framework web, desenvolvida pelo twitter, que facilita o desenvolvimento de interfaces das aplicações web. Inclui, por defeito, vários estilos para os componentes mais usados nas aplicações web tais como botões, hiperligações, formulários, etc…
Autofac (Autofac, 2013)	Implementa o padrão de inversão de controlo (IoC) na framework .NET. Inversão de controlo é um padrão de desenvolvimento onde a sequência (controle) de chamadas dos métodos é invertida em relação à programação tradicional, ou seja, ela não é determinada diretamente pelo programador.
Entity Framework (Microsoft, 2015)	Entity Framework (EF) implementa o mapeamento objeto/relacional que permite ao programador .NET trabalhar com um modelo relacional de dados através de objetos de dominio. Mapeamento objeto/relacional é o processo de conversão bidirecional entre objetos e tabelas de um modelo relacional.
FLOT (IOLA & Laursen, 2007)	Plugin para a biblioteca jQuery que permite criar diversos gráficos simples e interativos.

3.4 Arquitetura

Uma vez que a arquitetura da solução está intrinsecamente ligada à escolha de tecnologia procedeu-se a uma análise das diversas tecnologias relevantes descritas anteriormente definiu-se como condições os seguintes fatores:

a. Facilidade de integração dos diversos componentes que compõe a aplicação
b. Ferramentas de desenvolvimento avançadas e funcionais
c. Performance e escalabilidade
d. Segurança
e. Suporte continuo (LTS) da versão a ser usada
f. Documentação organizada e com exemplos práticos

A Microsoft, com os seus diversos produtos, permite endereçar todos os critérios de uma forma realmente consistente quando comparada com as restantes tecnologias estudadas, além de disponibilizar um IDE capaz de integrar todas as componentes necessárias ao projeto e inclusive a integração direta com a Azure.

Um outro fator que fez pender a escolha da tecnologia para o mundo Microsoft foi o facto de se poder usufruir do programa Microsoft BizSpark (Microsoft, 2016) dedicado a projetos inovadores e startups.

Este programa permite o usufruto da plataforma Azure durante 3 anos de forma gratuita com plafond para utilização de recursos e com acesso às ferramentas de desenvolvimento da Microsoft, por exemplo, o Visual Studio 2015 Enterprise (Microsoft, 2016) .

Por fim, o facto de ambos os autores terem tido contacto com tecnologia Microsoft ao longo dos seus percursos académicos e profissionais fechou a decisão tecnológica da solução.

Por todos os fatores acima descritos a escolha para desenvolvimento foi a seguinte:

- ASP.NET MVC C# (Portal)
- XAMARIN (Aplicações Móveis)
- WEB API 2.0 (RESTfull API)
- SQL SERVER Azure (Base de dados)
- Azure App Services (Alojamento)

A nível arquitetural a solução preconizada irá assentar sobre as diversas tecnologias que servirão para desenvolver as componentes necessárias à plataforma e garantindo um elevado grau de fiabilidade, performance, escalabilidade e resiliência.

Utilizando a plataforma Microsoft Azure será criada a infraestrutura virtual que disponibilizará os serviços necessários ao funcionamento da solução.

Teremos uma base de dados Microsoft *Azure* Cloud *SQL Database* que albergará todos os dados da plataforma, ou seja, todas as tabelas essenciais ao funcionamento da solução que serão manipuladas pelo site e pela API.

A utilização da solução Microsoft *Azure* Cloud *SQL Database* permitirá posteriormente recorrer a ferramentas de *Data Mining* e *Business Intelligence* e também a possibilidade de escalar a solução criando uma topologia de base de dados distribuída caso o crescimento o justifique.

De forma a garantir a segurança e integridade da solução o acesso à base de dados não ficará exposta ao exterior sendo unicamente acedido através de pedidos feitos à API que estará embutida no Portal. Assim sendo ambas estas componentes, o Portal e a API, serão publicadas em App Services no Microsoft Azure ficando disponíveis aos utilizadores através do URL registado.

A Figura 27 representa a arquitetura da solução proposta, verifica-se uma separação física dos serviços em duas camadas (2-tier).

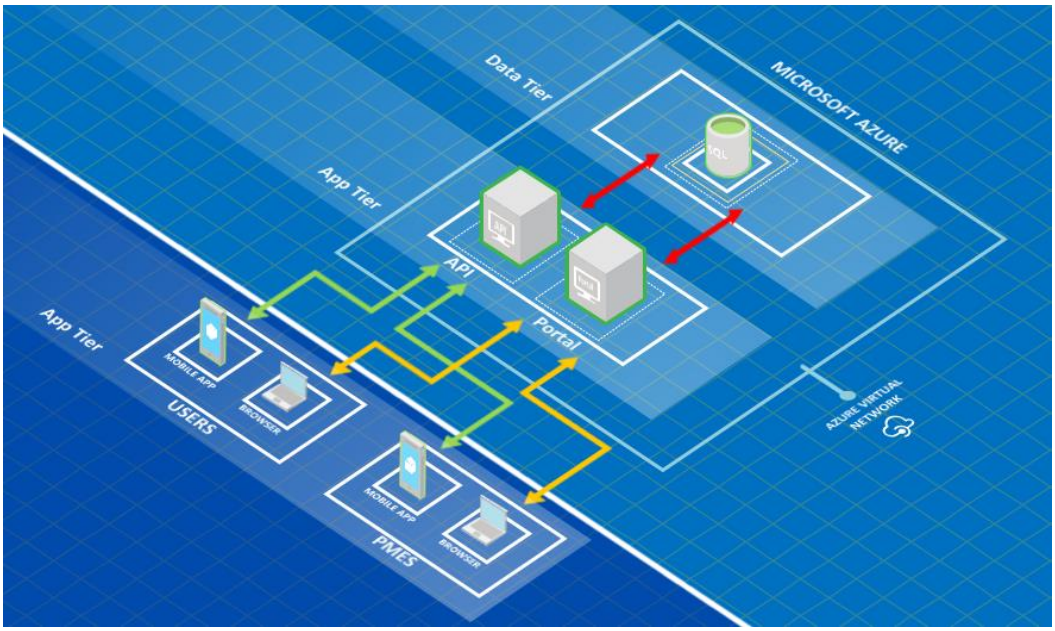


Figura 27 - Layout Arquitetura

Todo o desenvolvimento do website será efetuado em ASP.NET MVC e a API será desenvolvida em conjunto utilizando ASP.NET WebAPI 2.0.

A componente de aplicações móveis, quer para utilizadores como para Empresas será desenvolvida utilizando Xamarin.Forms em programação Cross-Platform, que possibilitará a compilação para diferentes sistemas (IOS, Android, Universal Windows Platform) permitido chegar a um maior número de utilizadores e equipamentos.

4 Desenvolvimento

Após um trabalho conjunto na análise e planificação das funcionalidades a implementar e da arquitetura a desenvolver desenrolou-se o processo de implementação das diversa componentes. De seguida encontra-se detalhado o planeamento e a construção de cada uma das áreas que constituem a plataforma idealizada para este projeto.

4.1 Portal

O portal assume papel essencial na ligação de todos os componentes que compõe a plataforma GetBack. Todos os serviços disponíveis para clientes móveis e persistência de dados foram implementados neste projeto.

Foram implementadas as seguintes funcionalidades:

- Painel de administração disponível aos administradores do sistema para gestão de clientes, utilizadores e dados relacionados com o sistema.
- Painel de administração disponível às empresas associadas para gestão dos cartões e dados da empresa.
- Área privada para os utilizadores da plataforma poderem consultar cartões, pontos e gerir dados pessoais.
- Servidor de autenticação por token para clientes móveis.
- Serviço de API para clientes móveis.

Neste capitulo é descrito todo o processo adotado para implementação do portal. Inclui o planeamento efetuado, o desenvolvimento, testes unitários e as restrições definidas.

4.1.1 Planeamento

A escolha da arquitetura adequada para o desenvolvimento de aplicações, principalmente em projetos mais complexos, é de extrema importância. O visual Studio 2015 traz, por defeito, um template para criação de projetos ASP.NET MVC que facilita e acelera a criação de formulários para gestão de dados, listagens, páginas, etc. Esta funcionalidade até se pode revelar útil em projetos de pequena dimensão, no entanto é preciso avaliar com precaução a escolha da arquitetura a adotar em projetos mais abrangentes. Manter no mesmo projeto regras de negocio, acesso aos dados e a camada de apresentação pode afetar a escalabilidade, usabilidade e testes da aplicação.

Para o desenvolvimento do portal, o sistema será dividido em três camadas, ou seja, arquitetura de três camadas (3-layer). As camadas a considerar serão as seguintes:

- **Serviço de Apresentação (UI layer/Presentation layer)**
  Responsável pela interação entre o utilizador (ou sistema externo) e a aplicação.
- **Regras de Negócio (Business layer)**
  Núcleo da aplicação em termos de processamento. Implementa todas as validações de dados necessárias e apenas se preocupa com a implementação das regras de negócio associadas ao problema e às entidades de negócio.
- **Serviço de Dados (Data access layer)**
  Implementa serviços de persistência. Encapsula, de um ponto de vista de operações de negócio, o acesso aos dados, isolando a camada de lógica de negócio.

Vantagens	Desvantagens
- Facilita a manutenção do código	- Aumenta complexidade e interpretação
- Separação lógica	- Requer maior tempo de desenvolvimento
- Potencia separação de responsabilidades	
- Baixa dependência entre camadas (low coupling) e alta coesão (high coesion)	
- Cada camada depende apenas da camada seguinte	

Tabela 47 - Análise à arquitetura de 3 camadas (3-layer)

Além da divisão da solução em camadas lógicas (3-layer), também será efetuada a divisão da solução em camadas físicas (2-tier). Enquanto a camada lógica refere-se à organização do código e dos dados, a camada física refere-se à distribuição do código e dos dados.

Para auxiliar o desenvolvimento desta arquitetura, serão adotados os seguintes padrões:

- **Padrão Repositório (Repository)**
- **Padrão Unidade de trabalho (UoW - Unit of Work)**
  Estes dois padrões funcionam em conjunto. É responsabilidade do Repositório representar a camada de dados na aplicação, enquanto que o padrão Unidade de Trabalho mantém uma lista de objetos afetados por uma transação, coordena a gravação das alterações e a resolução de problemas de concorrência.
- **Padrão Injeção de dependências (DI - Dependency Injection)**
  Padrão de desenvolvimento utilizado quando é necessário manter baixo o nível de acoplamento entre diferentes módulos de um sistema. Nesta solução as dependências entre os módulos não são definidas programaticamente, mas sim pela configuração de uma infraestrutura de software (container) que é responsável por "injetar" em cada componente suas dependências declaradas.

A arquitetura proposta é descrita na Figura 28, são apresentadas as várias camadas que compõe o projeto e, para cada uma das camadas, são listadas as componentes incluídas. De referir que as camadas designadas como *UI Layer* e *Presentation Layer* estão incluídos no Serviço de Apresentação.

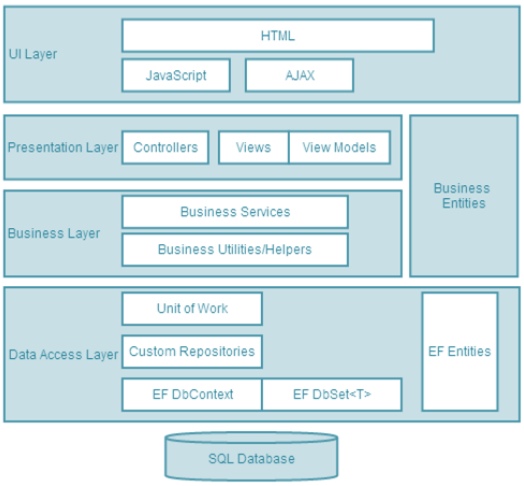


Figura 28 – Estrutura das camadas a implementar no portal (Redd, 2014)

4.1.2 Desenvolvimento

A solução original baseou-se no template gerado pelo Visual Studio 2015 e era composta pelos projetos: *GetBack.Web* e *GetBack.Tests*. Para implementar a arquitetura por camadas, foram criados outros projetos, sendo, posteriormente, atribuído a cada um deles responsabilidades especificas. O resultado final, com a lista de todos os projetos incluídos na solução está representado na Figura 29.

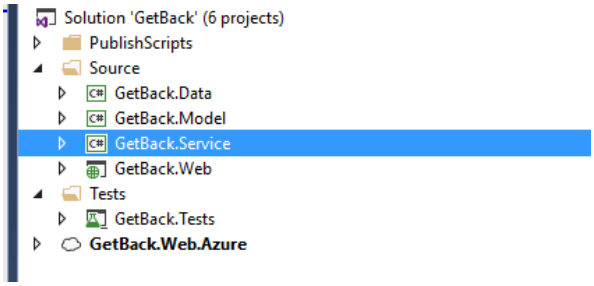


Figura 29 – Lista de projetos incluídos na solução GetBack

Com exceção do projeto *GetBack.Model*, cada projeto representa uma camada da arquitetura que se pretende implementar.

4.1.2.1 GetBack.Model

Este projeto armazena todos os objetos de domínio. A Entity Framework utiliza estes objetos para gerar ou modificar, posteriormente, a base de dados associada. Para auxiliar este processo, cada classe e atributo existente neste projeto pode conter anotações (*DataAnnotations*). Deve ser importada a libraria **System.ComponentModel.DataAnnotations**, pois esta disponibiliza atributos de classes (usados para definir meta dados) e métodos que podemos usar nas classes para alterar as convenções padrão e definir um comportamento personalizado que pode ser usado em vários

cenários. Em exemplo prático de anotações é apresentado na Figura 30, referente ao atributo *Category_Id* .

```
[Key]
[DatabaseGeneratedAttribute(DatabaseGeneratedOption.Identity)]
[ScaffoldColumn(false)]
4 references | 0 exceptions
public int Category_Id { get; set; }
```

Figura 30 – Exemplo de anotações associadas ao atributo *Category_Id*

Para implementar auditoria, definida como requisito funcional, foi gerada a classe *BaseModel* (Figura 31) que implementa todos os atributos e métodos necessários para satisfazer o requisito.

```
9   namespace GetBack.Model
10  {
        14 references
11      public abstract class BaseModel
12      {
13          [ScaffoldColumn(false)]
            2 references | 0 exceptions
14          public int? UserCreated { get; set; }
15          [ScaffoldColumn(false)]
            2 references | 0 exceptions
16          public DateTime? DateCreated { get; set; }
17          [ScaffoldColumn(false)]
            3 references | 0 exceptions
18          public int? UserUpdated { get; set; }
19          [ScaffoldColumn(false)]
            4 references | 0 exceptions
20          public DateTime? DateUpdated { get; set; }
21          [ScaffoldColumn(false)]
            3 references | 0 exceptions
22          public int? UserDeleted { get; set; }
23          [ScaffoldColumn(false)]
            3 references | 0 exceptions
24          public DateTime? DateDeleted { get; set; }
25          [ScaffoldColumn(false)]
            4 references | 0 exceptions
26          public Boolean? IsDeleted { get; set; }
27
28          [Display(Name = "Ordem de apresentação")]
29          [Range(1, 100, ErrorMessage = "A ordem deve ser um número intero entre 1 e 100")]
            2 references | 0 exceptions
30          public Int32 Order { get; set; }
31          [ScaffoldColumn(false)]
            2 references | 0 exceptions
32          public Int32? Status { get; set; }
33
            0 references | 0 exceptions
34          public BaseModel()...
48
            1 reference | 0 exceptions
49          public void OnCreate(int? uid)...
65
            0 references | 0 exceptions
66          public void OnUpdate(int? uid)...
78
            0 references | 0 exceptions
79          public void OnSoftDelete(int? uid)...
92      }
93  }
```

Figura 31 – Implementação da classe *BaseModel*

Além dos atributos que permitem registar todos os detalhes relacionados com inserção, alteração e eliminação de registos, esta classe implementa também métodos para gerir estas operações.

Todos os objetos de domínio implementados herdam as propriedades e métodos desta classe (Herança de objetos). A Figura 32 representa a classe *User_PME_Visit* deriva da classe *BaseModel*.

```
5 references
public class User_PME_Visit : BaseModel
{
    [Key]
    [DatabaseGeneratedAttribute(DatabaseGeneratedOption.Identity)]
    [ScaffoldColumn(false)]
    0 references | 0 exceptions
    public int Visit_Id { get; set; }
```

Figura 32 – Exemplo de herança da classe BaseModel pela classe User_PME_Visit

4.1.2.2 GetBack.Data

Este projeto implementa todos os mecanismos que permitem a persistência de dados. É o único com acesso direto à base de dados. As restantes camadas para aceder aos dados devem faze-lo através dos repositórios definidos neste projeto.

A ligação com a base de dados é da responsabilidade da classe *DBEntities* (Figura 33), é nesta classe que se definem todos os objetos de domínio (gerados no projeto *GetBack.Model*) que pretendemos criar persistência. Por defeito, o nome do modelo corresponde ao nome da tabela existente na base de dados, da mesma forma que cada um dos atributos definidos no modelo corresponde a um campo na tabela.

```
27 references
public class DBEntities : IdentityDbContext<ApplicationUser>
{
    7 references | 0 exceptions
    public DBEntities() : base("Name=DefaultConnection", throwIfV1Schema: false) {}
    // System
    0 references | 0 exceptions
    public DbSet<Sys_Config> SysConfig { get; set; }
    // Modules
    0 references | 0 exceptions
    public DbSet<Mod_Package> ModPackage { get; set; }
    18 references | 0 exceptions
    public DbSet<Mod_Category> ModCategory { get; set; }
    8 references | 0 exceptions
    public DbSet<Mod_PME> ModPme { get; set; }
    4 references | 0 exceptions
    public DbSet<Mod_User> ModUser { get; set; }
    // PME
    0 references | 0 exceptions
    public DbSet<PME_Media> pmeMedia { get; set; }
    0 references | 0 exceptions
    public DbSet<PME_User> pmeUser { get; set; }
    0 references | 0 exceptions
    public DbSet<PME_Config> pmeConfig { get; set; }
    0 references | 0 exceptions
    public DbSet<PME_Purchase> pmePurchase { get; set; }
    0 references | 0 exceptions
    public DbSet<PME_WebPart> pmeWebpart { get; set; }
    // USER
    0 references | 0 exceptions
    public DbSet<User_Config> userConfig { get; set; }
    0 references | 0 exceptions
    public DbSet<User_PME_Review> userPmeReview { get; set; }
    0 references | 0 exceptions
    public DbSet<User_PME_Visit> userPmeVisit { get; set; }
    0 references | 0 exceptions
    public DbSet<User_PME_Discount> userPmeDiscount { get; set; }
    // CMS
    9 references | 0 exceptions
    public DbSet<Sys_Content> sysContent { get; set; }
```

Figura 33 – Classe responsável pela ligação com a base de dados

Todas as interfaces e componentes necessários para a correta implementação do padrão Repositório e Unidade de trabalho estão definidas na pasta ***Infrastructure*****.** A lista completa de classes que compõe esta pasta pode ser consultada na Figura 34.

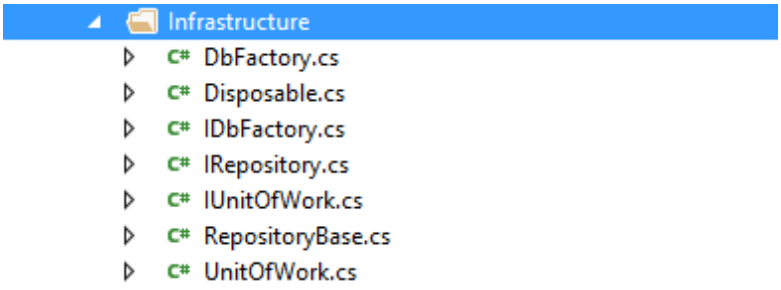


Figura 34 – Listagem das classes e interfaces para implementação do padrão repositório

A interface *IDbFactory* é responsável por inicializar as instancias da classe *DBEntities*. Estas interfaces serão, também, usadas para injetar instancias das classes que as implementam noutras camadas da solução. Na Figura 35 é apresentada a classe *DbFactory*, que implementa a interface referido.

```
namespace GetBack.Data.Infrastructure
{
    1 reference
    public class DbFactory : Disposable, IDbFactory
    {
        DBEntities dbContext;

        3 references | 0 exceptions
        public DBEntities Init()
        {
            return dbContext ?? (dbContext = new DBEntities());
        }

        2 references | 0 exceptions
        protected override void DisposeCore()
        {
            if (dbContext != null)
                dbContext.Dispose();
        }
    }
}
```

Figura 35 – Implementação da interface *IDbFactory*

Com a implementação da interface *IDbFactory* através da classe *DBFactory*, estão criados os mecanismos necessários para expor acesso aos dados a todos os repositórios, através do padrão injeção de dependência (DI).

Todos os repositórios devem herdar um conjunto de operações genéricas. Para o efeito foi definida a interface ***IRepository*** (Figura 36), que define quais os métodos genéricos a serem implementados e que serão, depois, herdados.

```
namespace GetBack.Data.Infrastructure
{
    3 references
    public interface IRepository<T> where T : class
    {
        // Insere novo objeto (NEW)
        3 references | 0 exceptions
        void Add(T entity);
        // Modifica objeto (MODIFIED)
        7 references | 0 exceptions
        void Update(T entity);
        // Elimina Objecto (DELETED)
        1 reference | 0 exceptions
        void Delete(T entity);
        1 reference | 0 exceptions
        void Delete(Expression<Func<T, bool>> where);
        // Obter objeto pelo ID
        3 references | 0 exceptions
        T GetById(int id);
        // Obter objeto através de critérios (DELEGATE)
        1 reference | 0 exceptions
        T Get(Expression<Func<T, bool>> where);
        // Todos os registos
        5 references | 0 exceptions
        IEnumerable<T> GetAll();
        // Obter objeto(s) através de critérios (DELEGATE)
        1 reference | 0 exceptions
        IEnumerable<T> GetMany(Expression<Func<T, bool>> where);

    }
}
```

Figura 36 – Interface IRepository, define os métodos genéricos comuns aos repositórios.

De referir que todos os métodos expostos para criar, alterar ou eliminar registos (CRUD) nos repositórios se limitam a marcar o objeto, nenhum comando é enviado para a base de dados. É da responsabilidade da camada que instancia o repositório (por norma um serviço), enviar a ordem para que as operações se tornem efetivas (**Commit**). Todas as tarefas que envolvam alterações à base de dados implementam o padrão unidade de trabalho (Unit of Work).

Na Figura 37 é disponibilizada a definição da interface *IUnitOfWork*. De realçar a simplicidade desta interface, apenas um método é definido. Como referido, o padrão unidade de trabalho tem como funções principais gerir a persistência dos dados e concorrência. A implementação desta interface pode ser consultada na Figura 38, através da classe *UnitOfWork*.

```
namespace GetBack.Data.Infrastructure
{
    8 references
    public interface IUnitOfWork
    {
        5 references | 0 exceptions
        void Commit();
    }
}
```

Figura 37 – Interface IUnitOfWork

```
namespace GetBack.Data.Infrastructure
{
    2 references
    public class UnitOfWork : IUnitOfWork
    {
        private readonly IDbFactory dbFactory;
        private DBEntities dbContext;

        0 references | 0 exceptions
        public UnitOfWork(IDbFactory dbFactory)
        {
            this.dbFactory = dbFactory;
        }
        1 reference | 0 exceptions
        public DBEntities DbContext
        {
            get { return dbContext ?? (dbContext = dbFactory.Init()); }
        }
        5 references | 0 exceptions
        public void Commit()
        {
            DbContext.Commit();
        }
    }
```

Figura 38 – Implementação da interface IUnitOfWork

A instancia necessária para chamar o método **Commit** é injetada no serviço, usando o padrão de injeção de dependência, neste caso é criada uma instancia do UnitOfWork através da sua interface IUnitOfWork.

Depois da definição de todos os interfaces e a implementação dos mesmos, fica garantida a implementação de todos os padrões propostos para este projeto. Resta a implementação de todos os repositórios necessários para suportar a plataforma e que serão usados pela camada que define as regras de negócio. Para cada repositório é definida uma interface (Figura 39 representa interface do repositório *UserRepository*) que contém os métodos a serem implementados (Figura 40 representa a implementação do *UserRepository*).

```
    3 references
    public interface IUserRepository : IRepository<ApplicationUser>
    {
        2 references | 0 exceptions
        Mod_User GetByEmail(string email);

        4 references | 0 exceptions
        Mod_User GetByToken(string token);

        2 references | 0 exceptions
        Mod_User GetByHashKey(string hashkey);

        2 references | 0 exceptions
        Mod_User GetByEmailOrPhone(string email, string phone);
    }
```

Figura 39 – Interface que define os métodos a implementar no repositório dos utilizadores

```
namespace GetBack.Data.Repositories
{
    /// <summary>
    ///
    /// </summary>
    1 reference
    public class UserRepository : RepositoryBase<ApplicationUser>, IUserRepository
    {
        /// <summary>
        ///
        /// </summary>
        /// <param name="dbFactory"></param>
        0 references | 0 exceptions
        public UserRepository(IDbFactory dbFactory)
            : base(dbFactory) { }

        /// <summary>
        ///
        /// </summary>
        /// <param name="login"></param>
        /// <returns></returns>
        2 references | 0 exceptions
        public Mod_User GetByEmail(string login)
        {

            var user = DbContext.ModUser.Where(l => l.Email == login).FirstOrDefault();
            return user;
        }
```

Figura 40 – Excerto da implementação do repositório dos utilizadores (*UserRepository*)

4.1.2.3 GetBack.Service

Todas as operações expostas aos controladores e API são implementadas neste projeto. É, também, neste projeto que são definidas todas as regras de negócio de toda a solução. Tem como dependências os dois projetos definidos anteriormente, ou seja, o *GetBack.Model* e o *GetBack.Data*.

Por defeito, por cada repositório existente é criado um serviço. E, tal como acontece nos repositórios, para cada serviço é criada uma interface que define os metodos que o serviço deve implementar. A Figura 41 representa um excerto da interface *IUserService*, parte da sua implementação está disponivel na Figura 42.

```
namespace GetBack.Service
{
    // operations you want to expose
    3 references
    public interface IUserService
    {

        2 references | 0 exceptions
        Boolean ValidateToken(string token);

        2 references | 0 exceptions
        String GetUserToken(string email);

        2 references | 0 exceptions
        ApiUserDetailsResponse GetUserDetails(string token);
```

Figura 41 – Interface *IUserService*, define os métodos para o serviço dos utilizadores

```
1 reference
public class UserService : IUserService
{
    private readonly IUserRepository UserRepository;
    private readonly IUnitOfWork unitOfWork;

    0 references | 0 exceptions
    public UserService(IUserRepository userRepository, IUnitOfWork unitOfWork)
    {
        this.UserRepository = userRepository;
        this.unitOfWork = unitOfWork;
    }
```

Figura 42 – Excerto da implementação da interface *IUserService*

Conforme se pode analisar na Figura 42, a injeção de dependências é feita de forma transparente e automática.

Neste projeto, para além da utilização dos repositórios, define todas as regras de negócio que se pretende implementar. A Figura 43 contém excerto de código com regras de negócio referentes à gestão dos utilizadores através da API.

```
public ApiUserResponse ActivateAccountByHashKey(string hashkey)
{
    ApiUserResponse _r = new ApiUserResponse();
    var user = UserRepository.GetByHashKey(hashkey);
    if (user == null)
    {
        _r.Result = ApiResultResponse.FAIL.ToString();
    }
    else
    {
        try
        {
            user.SecurityStamp = DateTime.Now.ToUniversalTime().ToString();
            user.PhoneNumberConfirmed = true;
            user.EmailConfirmed = true;
            user.LockoutEnabled = false;
            user.Status = 1;
            user.API_KEY = Guid.NewGuid().ToString();
            UserRepository.Update(user);
            unitOfWork.Commit();
            _r.Result = ApiResultResponse.OK.ToString();
        }
        catch (Exception)
        {
            _r.Result = ApiResultResponse.FAIL.ToString();
        }
    }

    return _r;
}
```

Figura 43 – Excerto de regra de negócio implementada pelo serviço dos utilizadores.

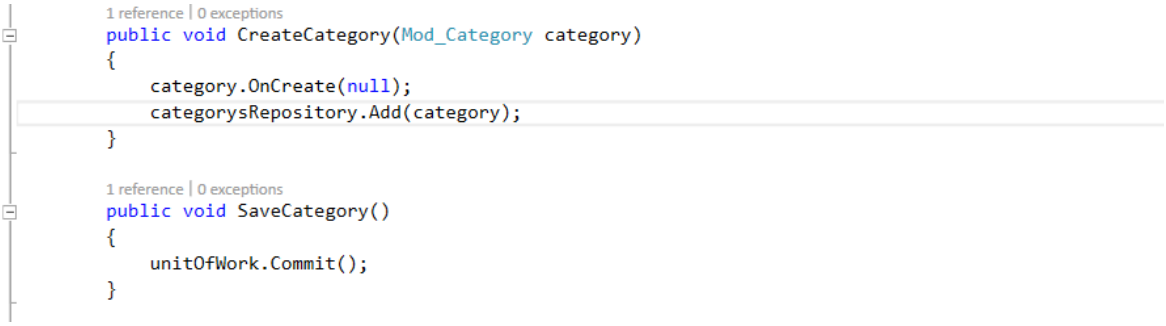


Figura 44 – Chamada ao método *Commit* dentro do serviço

Conforme especificado no projeto *GetBack.Data*, os repositórios não implementam rotinas com acesso à base de dados, limitam-se a marcar os registos afetados. É responsabilidade do serviço chamar o método *Commit* existente na implementação da classe *UnitOfWork* (Figura 44).

4.1.2.4 GetBack.Web

Este projeto define todos os aspetos relacionados com a apresentação, mas não só. Toda a informação relacionada acesso à base de dados (dados de ligação), a definição das dependências da aplicação nomeadamente repositórios e serviços implementados são tratados neste projeto.

É neste projeto que reside o ficheiro *web.config*, onde se pode definir os dados para acesso à base de dados. A Figura 45 contém excerto do ficheiro *web.config* onde é possível consultar o valor da *ConnectionString*, variável assumida por defeito para conter dados de ligação à base de dados.

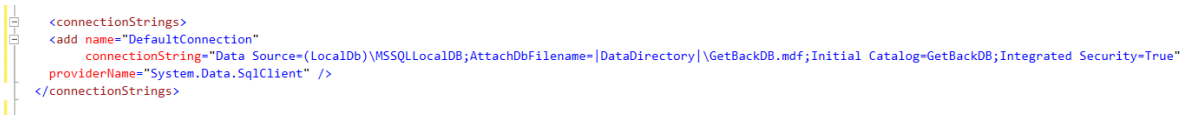


Figura 45 – Definição dos dados de acesso à base de dados

Para a implementação do padrão injeção de dependência, utilizado nos projetos anteriores, é necessário a instalação de um pacote (disponível via NuGet) chamado Autofac.

A classe ***Startup*** (Figura 46)**,** existente na raiz do projeto, tem a responsabilidade de instanciar todos os módulos da framework, registar dependências, configurar serviços e funcionalidade disponíveis pela aplicação.

Figura 46 – Excerto de código da classe Startup

O excerto apresentado, é de extrema importância pois permite interligar todos os projetos existentes na solução proposta para implementação do portal.

#	Descrição
1	Definição das classes que implementam as interfaces e que serão injetadas nos serviços, em concreto estas duas linhas definem a classe que tem acesso à base de dados (*DbFatory*) e a classe responsável pela gravação de dados e concorrência **(***UnitOfWork***).**
2	Registo automático de todos os repositórios presentes no projeto ***GetBack.Data***. Todas as classes que implementem um repositório devem obrigatoriamente terminar com o texto ***Repository*.**
3	Registo automático de todos os serviços presentes no projeto ***GetBack.Service***. Todas as classes que implementem um serviço devem obrigatoriamente terminar com o texto ***Service*.**

Tabela 48 - Descrição de métodos da classe ***Startup***

Após configuração de toda a solução, falta implementar mecanismo de autenticação por formulário (Figura 47) para os utilizadores registados, o método é comum a todos os perfis (utilizador, empresa ou administrador). Este requisito é implementado por defeito em ASP.NET MVC através do pacote ASP.NET Identity (Microsot, 2016).

```
<authentication mode="Forms">
  <forms loginUrl="Account/Auth" />
</authentication>
```

Figura 47 – Configuração de autenticação via Forms.

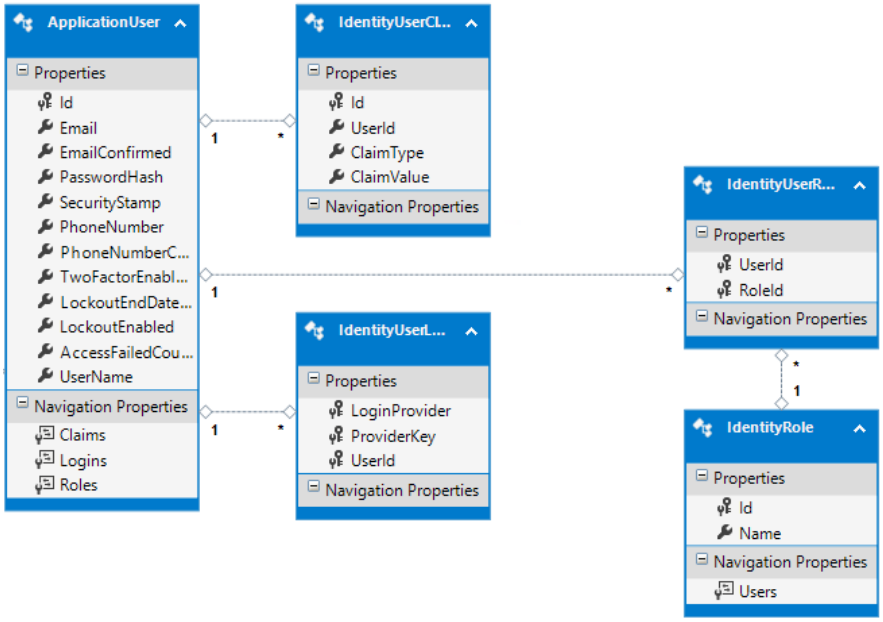


Figura 48 – Diagrama de classes por defeito do pacote Asp.NET Identity

Para implementar os diversos perfis existentes no portal, houve a necessidade de criar duas novas classes que derivam da classe ***ApplicationUser*** (Figura 48)**.** Em cada uma das classes foram definidas as propriedades e métodos necessários a cada um dos perfis (Figura 49). A classe ***Mod_User*** define todas as propriedades necessárias para suportar os utilizadores do sistema enquanto que a classe ***Mod_PME*** implementa todas as propriedades necessárias para suportar o perfil empresa. Quanto aos administradores do sistema, não há necessidade de implementar campos especiais.

As permissões associadas a cada um dos perfis está definida na classe ***IdentityUserRole***.

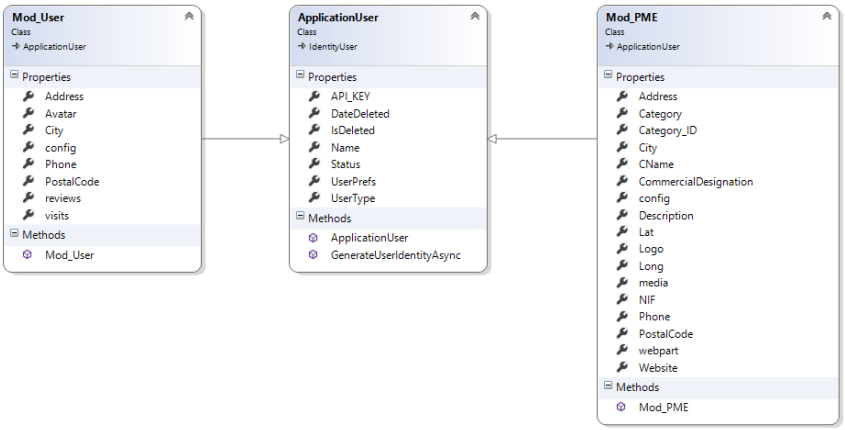


Figura 49 – Diagrama das classes que suportam os diferentes perfis.

Com exceção do projeto **GetBack.Web**, os restantes projetos da solução estão desenvolvidos e devidamente interligados. Neste projeto falta implementar a principal função que lhe está destinada,

a apresentação de informação e interação com os utilizadores do sistema. Para o efeito é necessário gerar controladores (Figura 50), que por sua vez recorrem aos serviços implementados.

```
1 reference
public class CategoryController : Controller
{
    private readonly ICategoryService categoryService;

    0 references | 0 exceptions
    public CategoryController(ICategoryService categoryService)
    {
        this.categoryService = categoryService;
    }

    // GET: Category
    0 references | 0 requests | 0 exceptions
    public ActionResult Index()
    {
        IEnumerable<Mod_Category> categories;
        categories = categoryService.GetCategories("").ToList();
        IEnumerable<CategoryViewModel> vmcats = Mapper.Map<IEnumerable<Mod_Category>, IEnumerable<CategoryViewModel>>(categories);
        return View(vmcats);
    }
}
```

Figura 50 – Exemplo de controlador para apresentação de dados

Seguindo os padrões definidos, também nos controladores os serviços são injetados automaticamente.

Toda a informação passada do controlador para cada vista (Figura 51) é mapeada do modelo para o viewmodel respetivo. Este padrão, conhecido como MVVM, é descrito em pormenor no planeamento da aplicação móvel.

```
@using GetBack.Model.ViewModel
@model LoginViewModel

<div class="row">
    <div class="col-md-6 col-lg-4 col-sm-12 col-xs-12 col-lg-offset-4 col-md-offset-3">
        <section id="loginForm">
            @using (Html.BeginForm("Login", "Account", new { ReturnUrl = ViewBag.ReturnUrl }, FormMethod.Post, new { @class = "form-ho
            {
                @Html.AntiForgeryToken()
                <h4 class="tblue"><i class="fa fa-user tblue"></i> Introduza os seus dados de acesso.</h4>
                <hr />
                @Html.ValidationSummary(true, "", new { @class = "text-danger" })
                <div class="form-group">
                    @Html.LabelFor(m => m.Email, new { @class = "col-md-2 control-label" })
                    <div class="col-md-10">
                        @Html.TextBoxFor(m => m.Email, new { @class = "form-control" })
                        @Html.ValidationMessageFor(m => m.Email, "", new { @class = "text-danger" })
                    </div>
                </div>
                <div class="form-group">
                    @Html.LabelFor(m => m.Password, new { @class = "col-md-2 control-label" })
                    <div class="col-md-10">
                        @Html.PasswordFor(m => m.Password, new { @class = "form-control" })
                        @Html.ValidationMessageFor(m => m.Password, "", new { @class = "text-danger" })
                    </div>
                </div>
```

Figura 51 – Excerto da vista para apresentação do formulário de login

O portal é composto por três áreas, cada uma personalizada para o perfil a que está destinado. A Figura 52 apresenta a página principal do portal e oferece ligações para todas as funcionalidades, com especial relevo para as consideradas principais: acesso à área privada (Figura 53), registo de novo utilizador e pesquisa.

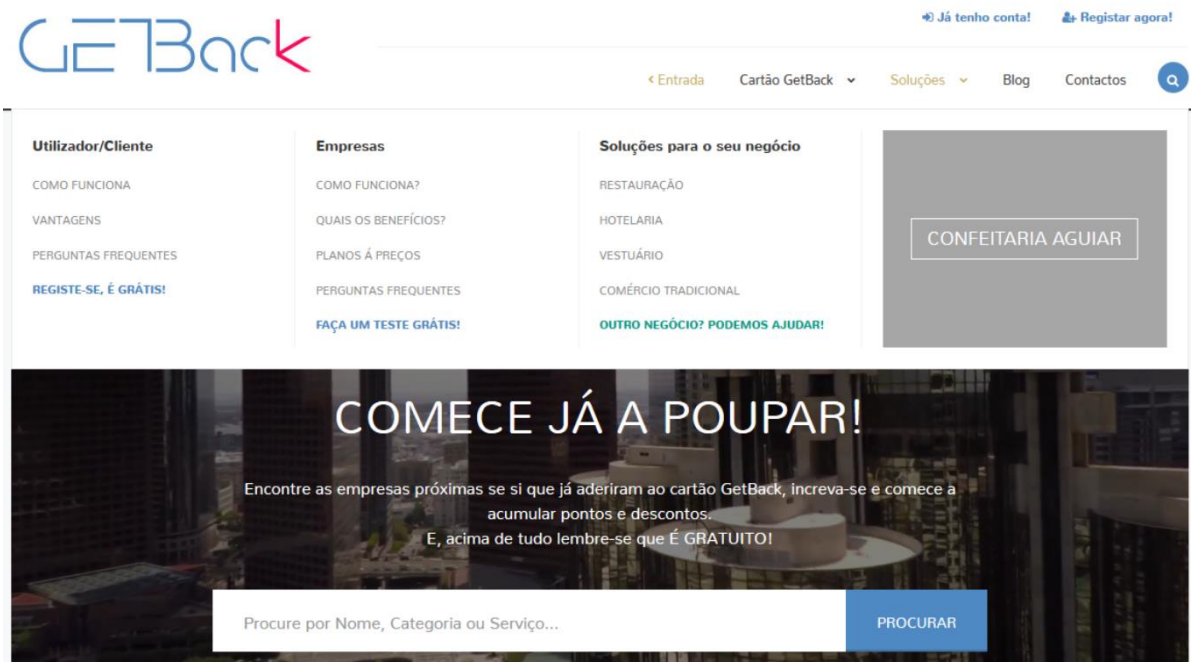


Figura 52 – Página principal do portal.

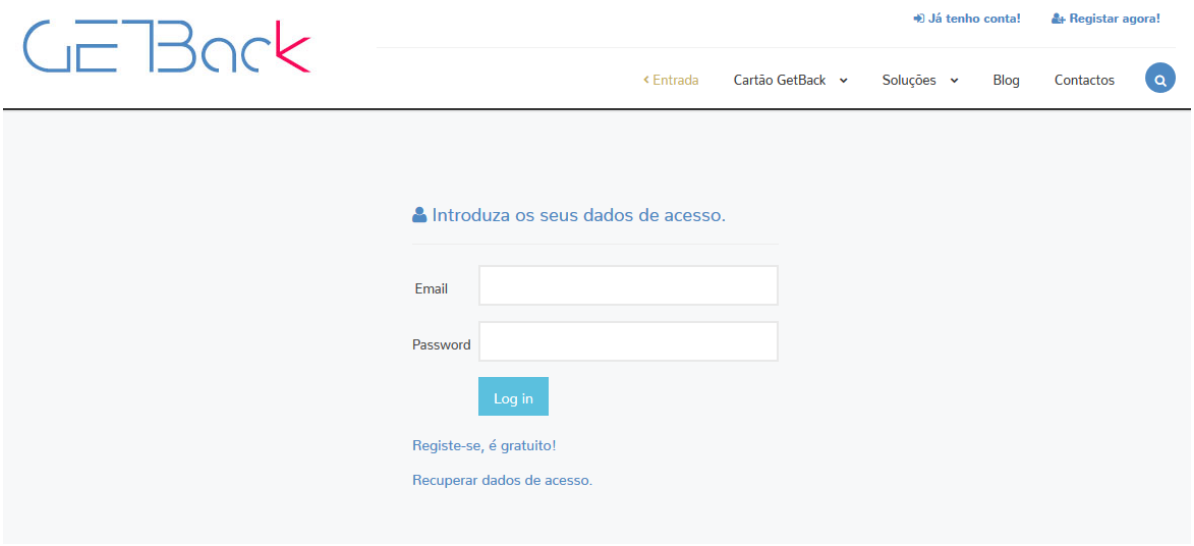


Figura 53 – Formulário de login para acesso à área privada.

Os administradores e empresas têm ao dispor uma página, apresentada após validação no sistema, onde são apresentadas informações relevantes. No caso das Empresas são apresentadas informações detalhadas sobre o seu negócio, nomeadamente total de cartões, total de compras efetuadas com utilização do cartão, entre outros dados conforme é visível na Figura 54. O painel disponível para os Administradores fornece informações sobre o sistema e gestão de informação relevante. A Figura 55 apresenta lista de todas as categorias disponíveis no sistema.

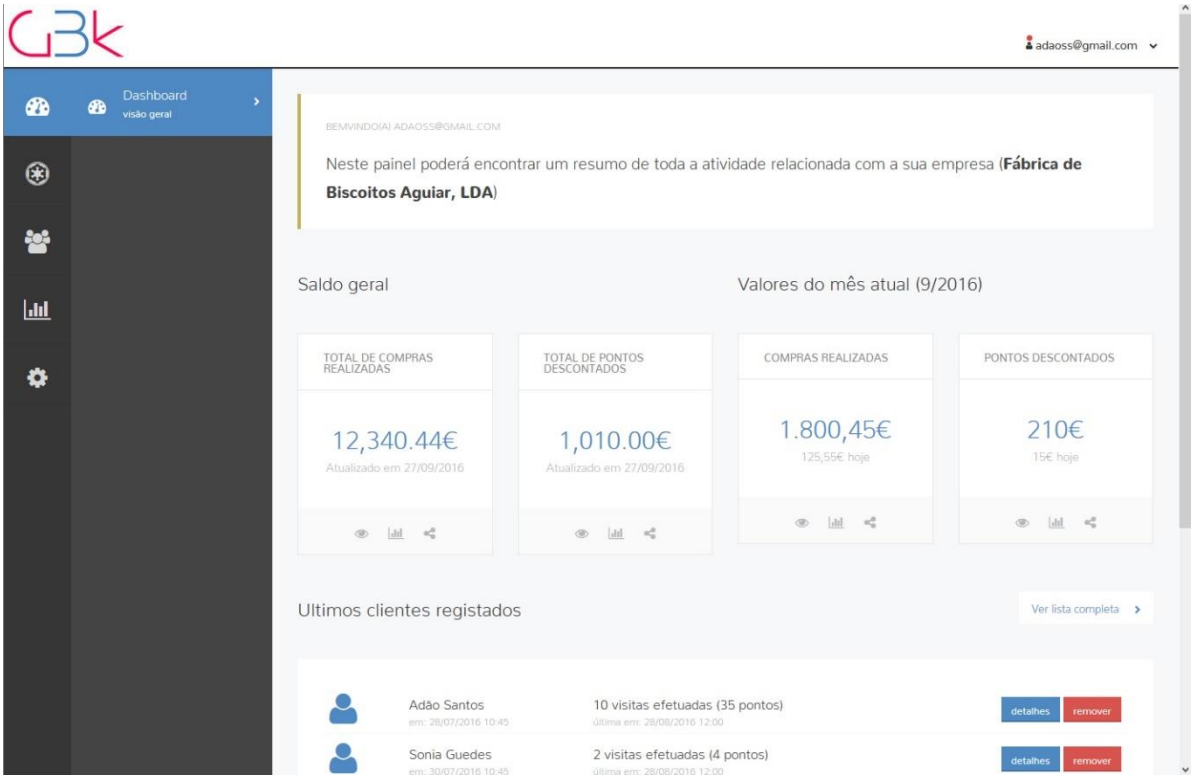


Figura 54 – Área privada reservada ao perfil empresa.

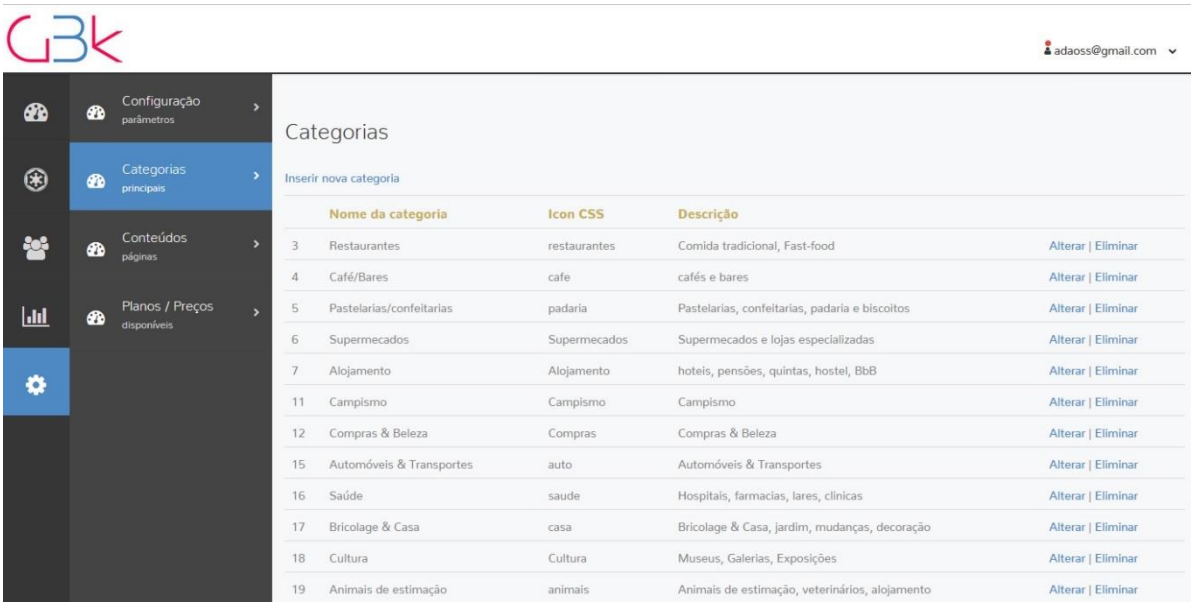


Figura 55 – Área privada reservada ao perfil Administrador.

4.1.3 Testes

Testes de software refere-se à analise do software com a intenção de fornecer informações sobre a sua qualidade em relação ao contexto em que ele deve operar. Inclui o processo de utilizar o produto para encontrar seus defeitos.

Também conhecida como testes unitários ou teste de módulo, é a fase em que se testam as menores unidades de software desenvolvidas (pequenas partes ou unidades do sistema). O universo alvo deste tipo de teste são as sub-rotinas, métodos, classes ou mesmo pequenos excertos de código. Assim, o objetivo é o de encontrar falhas de funcionamento dentro de uma pequena parte do sistema funcionando independentemente do todo.

4.1.3.1 Testes unitários

A adoção dos padrões descritos no planeamento do portal revelou-se uma mais valia quando se começaram a desenvolver os testes unitários.

O principal requisito para esta componente foi a criação de uma nova classe que implementasse a interface ***IUnitOfWork*** e outra classe que implementasse a interface ***IRepositoryBase***. A principal diferença, na implementação destas duas classes, em relação às classes iniciais prende-se com a exclusão do ***DBEntities*** **(ou *DbContext*)**, classe que efetua a ligação à base de dados. A Figura 56 representa a diferença nas diversas camadas no projeto de testes relativamente à aplicação.

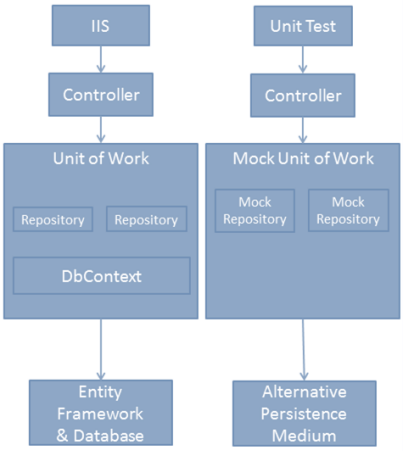


Figura 56 – Diferença das camadas entre aplicação e testes unitários (Microsoft, 2012)

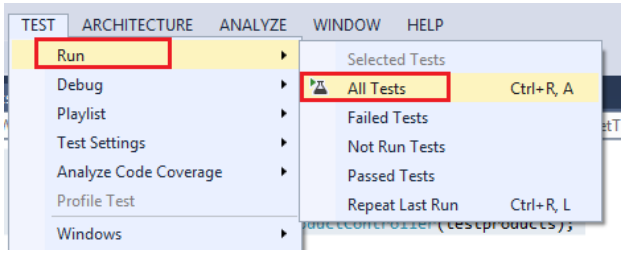


Figura 57 – Procedimento para execução dos testes unitários implementados.

A execução dos testes unitários é iniciada seguindo os passos descritos na Figura 57, a execução dos testes definidos gera um resultado semelhante ao apresentado pela Figura 58.

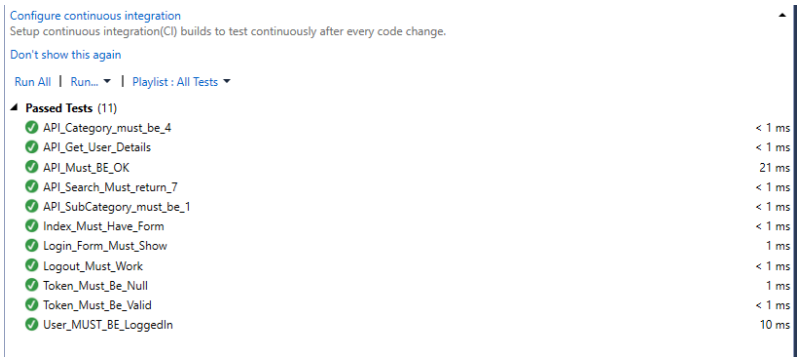
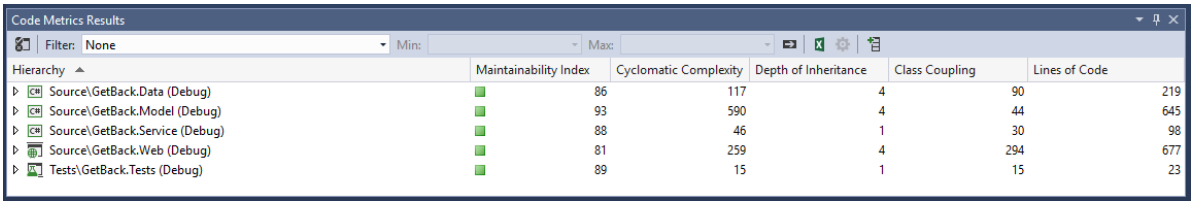


Figura 58 – Output da execução dos testes unitários

4.1.3.2 Métrica de software (Software Measurements)

Para além dos testes unitários, foi analisado todo o projeto para obter as métricas do software desenvolvido (Figura 59). Estes valores são calculados pelo Visual Studio 2015.



Hierarchy	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Code
Source\GetBack.Data (Debug)	86	117	4	90	219
Source\GetBack.Model (Debug)	93	590	4	44	645
Source\GetBack.Service (Debug)	88	46	1	30	98
Source\GetBack.Web (Debug)	81	259	4	294	677
Tests\GetBack.Tests (Debug)	89	15	1	15	23

Figura 59 – Métricas para a solução proposta, gerada pelo Visual Studio 2015

As métricas de software são compostas pelos seguintes critérios:

- **Manutenção (Maintainability Index)** – Valor entre 0 e 100, representa a facilidade de manutenção do código. É apresentado com dois parâmetros, o valor e a cor.
  - Quando a cor é vermelha e o valor está entre 0 e 9, o código é de difícil manutenção.
  - Quando a cor é amarela e o valor entre 10 e 19, o código esta num ponto intermedio de manutenção;
  - Quanto a cor é verde e o valor entre 20 e 100, indica boa manutenção.
- **Complexidade (Cyclomatic Complexity)** – Mede a complexidade estrutural do código. O valor é obtido através do cálculo do número de caminhos diferentes no fluxo do programa.
- **Profundidade de herança (Depth of Inheritance)** – Indica o número de classes definidas que estendem até à raiz da hierarquia de classes. Quanto maior o valor, mais difícil se torna compreender e encontrar métodos e propriedades.
- **Acoplamento de classes (Class coupling)** – Mede o acoplamento de classes exclusivas através de parâmetros, variáveis locais, tipos de retorno, chamadas de método, classes base, implementações de interface ou campos definidos em tipos externos. Uma arquitetura de software bem definida determina que tipos e métodos devem ter alta coesão e baixo acoplamento.

- **Linhas de código (Lines of Code**) – Indica o valor aproximado de linhas de código do projeto. Este valor é baseado na linguagem intermedia (IL – Intermediate Language) e não nas linhas de código que o projeto tem. Valores demasiado elevados podem indicar que sobrecarga dos métodos implementados no projeto.

Aplicando estas métricas à solução proposta pode-se afirmar que mantém valores excelentes no que diz respeito à manutenção (nenhum dos projetos tem valor abaixo de 80). Nos restantes critérios

4.1.4 Restrições

No desenvolvimento da solução proposta foram definidas as restrições presentes na Tabela 49.

Perfil	Restrição
Utilizador	- Email do utilizador deve ser único no sistema.
	- Número de telemóvel deve ser único no sistema.
	- Utilizador apenas pode aceder à conta depois de confirmar dados
	- A adesão ao cartão de fidelização de uma determinada empresa está condicionado à aceitação por parte da empresa.
Empresa	- Dispõe de 5 dias, depois de finalizado o prazo para pagamento da subscrição para regularizar a situação. Depois desse tempo o serviço é bloqueado.
	- Agrupamento diversas lojas e/ou filiais não é possível.
Sistema	- Senhas de acesso devem ter no mínimo 8 carateres com números, letras maiúsculas e minúsculas.

Tabela 49 - Restrições definidas na implementação do portal

4.2 API

A API tem a responsabilidade de expor recursos para serem consumidos pelas aplicações móveis disponíveis para os utilizadores e empresas associadas.

4.2.1 Planeamento

O planeamento da API foi, praticamente, executado em conjunto com o planeamento do portal. Uma das principais alterações está no mecanismo a implementar para garantir a segurança da informação e validação dos utilizadores. Enquanto no portal foi usada a framework ASP.NET Identity, na API será usada a framework de autenticação OAuth (OAuth0, 2016).

Um dos mecanismos mais usados para a autenticação de clientes sobre HTTP, é através do uso de um token. No contexto de autenticação, pode-se definir token como um identificador que permite validar o cliente perante o servidor sem necessidade de enviar, em cada pedido, detalhes sobre o utilizador e a password. No processo de atribuição de um token válido, o cliente envia ao servidor um pedido inicial de validação que contém o utilizador e password e, no caso dos dados de acesso

fornecidos serem validos, é retornado um identificador que será usado, posteriormente, para validar os pedidos do cliente (Figura 60).

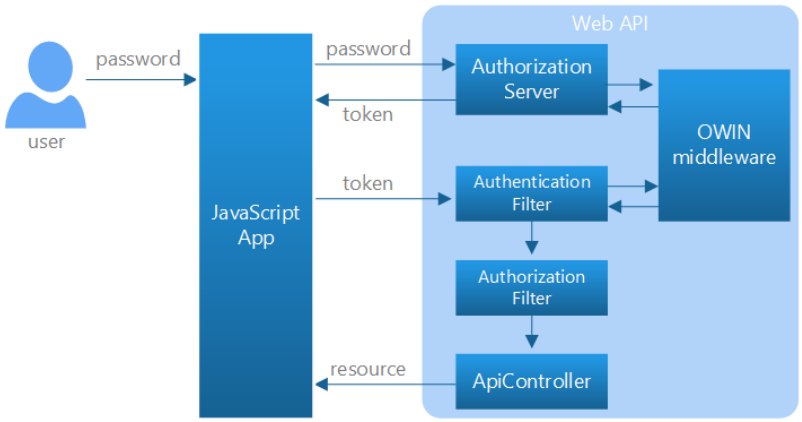


Figura 60 - Diagrama de fluxo de dados para autenticação e uso do token (Wasson, 2014)

4.2.2 Desenvolvimento

Todos os componentes necessários para a implementação da API estão disponíveis na estrutura base do portal, ou seja, no projeto *GetBack.Web*.

Ao gerar o projeto inicial, que serviu como base para desenvolvimento do portal, o assistente permite definir quais os componentes que pretendemos instalar por defeito, um dos componentes disponíveis é a framework para desenvolver a API.

Optou-se por instalar apenas os componentes (pacotes) necessários para o portal e, numa fase posterior, acrescentar os pacotes necessários para a API. A framework Microsoft ASP.NET WEB API pode ser instalada através da consola do gestor de pacotes (NuGet) ou, em alternativa, através do Visual Studio.

```
PM> Install-Package Microsoft.AspNet.WebApi
```

Figura 61 – Instalação do pacote *Microsoft.AspNet.WebApi* via consola

Finalizada a instalação do pacote, é necessário configurar a WEB API no projeto *GetBack.Web.* Em primeiro lugar vão ser definidas as rotas (routes) para poder aceder aos recursos disponibilizados. Para o efeito, acrescentou-se o ficheiro *WebApiConfig.cs* (Figura 62) a pasta *App_Start*.

```
1 reference
public static class WebApiConfig
{
    1 reference | 0 exceptions
    public static void Register(HttpConfiguration config)
    {
        // Web API configuration and services
        // Configure Web API to use only bearer token authentication.
        config.SuppressDefaultHostAuthentication();
        config.Filters.Add(new HostAuthenticationFilter(OAuthDefaults.AuthenticationType));

        // Web API routes
        config.MapHttpAttributeRoutes();

        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );

        config.Formatters.JsonFormatter.SupportedMediaTypes.Add(new MediaTypeHeaderValue("text/html"));
        config.Formatters.JsonFormatter.SerializerSettings.ContractResolver
            = new CamelCasePropertyNamesContractResolver();
        config.Formatters.JsonFormatter.UseDataContractJsonSerializer = false;

        // Enforce HTTPS
        config.Filters.Add(new RequireHttpsAttribute());
    }
}
```

Figura 62 – Configuração da WEB API disponível na classe *WebApiConfig*

Esta configuração só se torna oficial e é executa quando procedermos ao registo da mesma na classe responsável pelo carregamento e configuração da aplicação conforme especificado na Figura 63.

```
0 references
public class MvcApplication : System.Web.HttpApplication
{
    0 references | 0 exceptions
    protected void Application_Start()
    {
        AreaRegistration.RegisterAllAreas();
        GlobalConfiguration.Configure(WebApiConfig.Register);
        FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
        RouteConfig.RegisterRoutes(RouteTable.Routes);
        BundleConfig.RegisterBundles(BundleTable.Bundles);

        //
        System.Data.Entity.Database.SetInitializer(new GetBackSeedData());

        HttpConfiguration config = GlobalConfiguration.Configuration;
        config.Formatters.JsonFormatter.SerializerSettings.ContractResolver = new CamelCasePropertyNamesContrac
        config.Formatters.JsonFormatter.UseDataContractJsonSerializer = false;

    }
}
```

Figura 63 – *Global.asax.cs* modificado para incorporar a WEB API

Neste momento, o projeto já se encontra configurado para responder às chamadas através da API, apesar de ainda não existirem controladores. Antes de criar controladores, é necessário efetuar a instalação e configuração dos mecanismos de autenticação e validação da API (Figura 64).

```
Install-Package Microsoft.AspNet.WebApi.Owin -Version 5.1.2
Install-Package Microsoft.Owin.Host.SystemWeb -Version 2.1.0
```

Figura 64 – Instalação dos pacotes para implementar autenticação via token na API

```
// TOKEN
PublicClientId = "self";
OAuthOptions = new OAuthAuthorizationServerOptions
{
    TokenEndpointPath = new PathString("/Token"),
    Provider = new ApplicationOAuthProvider(PublicClientId),
    AuthorizeEndpointPath = new PathString("/api/Account/ExternalLogin"),
    AccessTokenExpireTimeSpan = TimeSpan.FromDays(14),
    AllowInsecureHttp = true
};

app.UseOAuthBearerTokens(OAuthOptions);


// REGISTER WITH OWIN
app.UseAutofacMiddleware(container);
app.UseAutofacMvc();

// Auth
ConfigureAuth(app);
```

Figura 65 – Configuração do OAuth2.0 na classe Startup

A classe *Startup* é carregada automaticamente pelo servidor no seu arranque. Todos os serviços e configurações que pretendemos implementar devem ser registados nesta classe conforme especificado na Figura 65. A camada de apresentação de dados disponibiliza agora uma API com autenticação e validação por token paralelamente às páginas web (Figura 66).

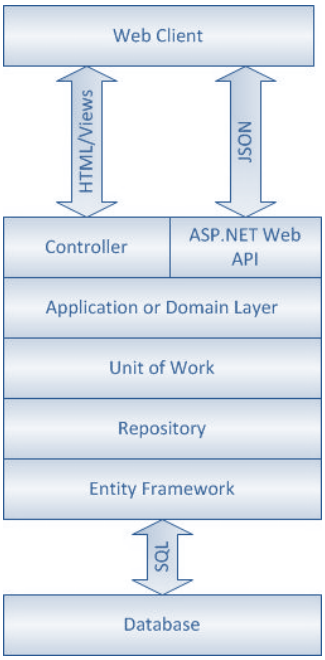


Figura 66 – Nova arquitetura após implementação da API

Todos os controladores da API derivam do controlador *BaseApiController*, neste são implementadas rotinas que facilitam o acesso aos dados do utilizador que está a efetuar o pedido à API (Request). A Figura 67 contém excerto da classe que serve de base a todos os controladores da API.

```
namespace GetBack.Web.Api
{
    1 reference
    public class BaseApiController : ApiController
    {
        private ApplicationUserManager _AppUserManager = null;
        private ApplicationSignInManager _AppSignInManager = null;

        /// <summary>
        ///
        /// </summary>
        4 references | 0 exceptions
        protected ApplicationUserManager AppUserManager
        {
            get...
            private set...
        }
```

Figura 67 – Excerto de código do controlador base da API

Ao analisar os controladores que compõe a API, as semelhanças com os controladores do portal são imensas, isto deve-se à adoção da arquitetura 3 camadas (3-layer). Todos os serviços e repositórios implementados para responder aos pedidos do portal, são agora usados para responder a pedidos da API (Figura 68).

```
    1 reference
    public class CategoryController : BaseApiController
    {
        // default access to database
        // private DBEntities db = new DBEntities();

        /// <summary>
        /// Ibjected service on controller
        /// </summary>
        private readonly ICategoryService categoryService;

        /// <summary>
        /// Dependency injection of ICategoryService
        /// </summary>
        /// <param name="categoryService"></param>
        0 references | 0 exceptions
        public CategoryController(ICategoryService categoryService)
        {
            this.categoryService = categoryService;
        }
        // GET: api/Category
        0 references | 0 requests | 0 exceptions
        public ApiListResult<CategoryViewModel> GetCategories()
        {
            var cats = categoryService.GetCategories().ToList();
            var res = MiniMap.Map<Mod_Category, CategoryViewModel>(cats).ToList();

            ApiListResult<CategoryViewModel> T = new ApiListResult<CategoryViewModel>();
            T.Result = ConstantsConfiguration.ApiResultResponse.OK.ToString();
            T.Results = res;
            return T;
        }
```

Figura 68 –Excerto do controlador da API *CategoryController*, deriva do *BaseAPiController*

4.2.3 Testes

Os testes unitários implementados para testar a API estão incluídos com os testes unitários do Portal. Além desses, foram efetuados testes à API, principalmente para testar a autenticação, usando o programa PostMan (Postman, 2016).

A figura 69 representa o pedido efetuado ao servidor para obter uma token baseada nos dados passados, a resposta do servidor pode ser consultada na Figura 70.

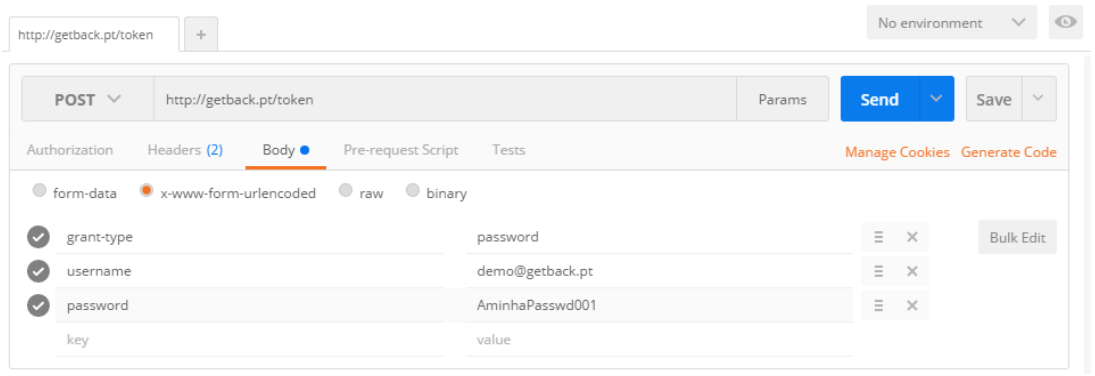


Figura 69 – Teste para verificar funcionamento da validação por token


```
{
    access_token:
    "A4xGkbfXnIGryZ6zYRjL5WZPWQ0fyl8zYz9_aHhb9w7Qgx7KccYWcgm2vM5qFRwNyTHwYVNrgsdJU0U5RmPDyivWUI0NBRBgHOEhE
    PIJRPyuqr__LEGGlkjbm5fLEMBo_f0-GZUqq0dfYRpXTTSwggD9i_BQah0ddxbP_W-
    vvpar61CPaTN89WDKJD3IpHxDbksNRsMqZ3J1KauvoaPs0s7j5Mw2PY",
    token_type: "bearer",
    expires_in: 86399
}
```


Figura 70 – Resposta ao pedido de autenticação com retorno de token

4.3 Aplicação Móvel Para Utilizadores

O objetivo no desenvolvimento desta aplicação é que o seu código seja de fácil manutenção e que possibilite a implementação em várias plataformas. Neste subcapítulo é descrito o planeamento efetuado para a construção desta aplicação e explicitada a estrutura utilizada e alguns pormenores da implementação.

4.3.1 Planeamento

A aplicação móvel para os utilizadores começou a ser delineada desde a análise do problema e levantamento dos requisitos funcionais. Com a escolha da tecnologia Xamarin para desenvolver esta plataforma móvel foi adotado padrão MVVM (Figura 71) com algumas alterações. Este padrão assenta em três camadas: Model, View Model e View, cada uma com uma função especifica.

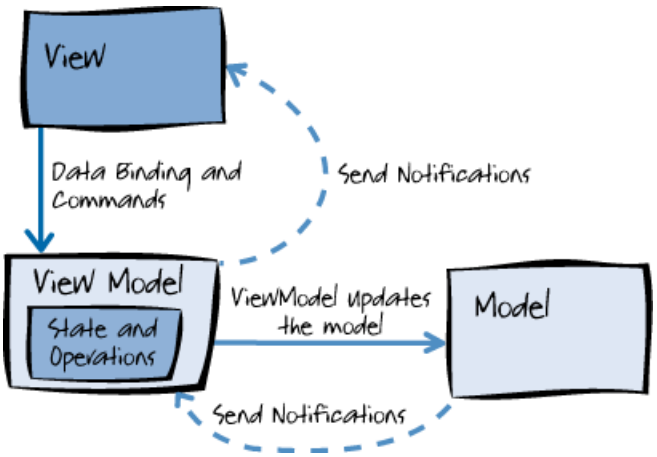


Figura 71 – Padrão MVVM

Enquanto a camada View representa a parte visível ao utilizador, ou seja, a aparência da informação que será apresentada ao utilizador, a camada ViewModel contém a informação a ser passada às Views através de DataBinding.

Uma vez que não está prevista a persistência de dados (salvo informação de utilizador) a camada de Model será utilizada essencialmente como apoio, sendo as transações/updates de dados efetuada através de uma classe de serviço que executará as chamadas à API anteriormente descrita.

Apesar da reduzida necessidade de persistência de dados será utilizado o SQLite (SQLite, 2016) para guardar informação sobre o utilizador apenas no caso deste querer manter-se autenticado na aplicação.

Outro fator a ter em conta no planeamento da aplicação foi o de aproveitamento de livrarias existentes para desempenhar funcionalidades definidas para a aplicação.

A Geolocalização, conversão de JSON's e a criação de QRCode são exemplos de funcionalidades que foram endereçadas com recurso a livrarias de terceiros nomeadamente Geolocator (jamesmontemagno, 2016) (by JamesMonteMagno), Newtonsoft (Newtonsoft, 2016) e ZXING (Redth, 2016) respetivamente.

Para esta aplicação foi definida uma estrutura composta por 5 camadas principais (Figura 72). Em primeiro lugar teremos toda a componente de apresentação que será desenvolvida maioritariamente em código existindo casos específicos onde será utilizada a codificação em XAML. A camada de apresentação está ligada a ViewModels que possibilitarão estabelecer regras e ligar dados aos componentes existentes nas diversas Views.

Ao contrário do que é normal no Padrão MVVM, a camada Model não possuirá funcionalidades que providenciem a persistência dos dados, funcionando somente como modelos de definição de tipo de dados para serem utilizados quer nos diversos ViewModels quer na camada de serviços, mais concretamente na classe que tratará de todo o contacto com a WebAPI. Essa classe estará na camada Service juntamente com outras classes que executem trabalhos que não interfiram diretamente na lógia de negócio nem na lógica de apresentação. Outro exemplo será a classe que calcula o posicionamento GPS do utilizador.

Por fim será necessária uma base de dados para garantir a única persistência de dados que julgamos necessária e que estará associada ao facto de o utilizador querer (ou não) manter-se autenticado na aplicação. Esta base de dados será utilizada para guardar temporariamente o token enviado pela API para os utilizadores autenticados.

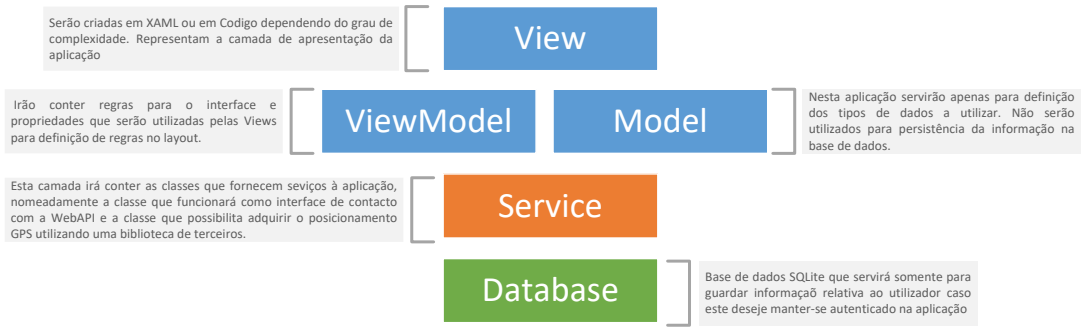


Figura 72 – Estrutura aplicação móvel

Com base nesta topologia passou-se a uma análise mais detalhada dos casos de uso criando os diagramas de sequência que endereção cada uma das especificações e tendo em vista simplificar o processo de desenvolvimento da aplicação.

O primeiro caso de uso que iremos descrever é o que reflete a autenticação do utilizador (Figura3). Quando iniciada a aplicação o utilizador deverá ter a possibilidade de efetuar o Login inserindo as suas credências (email ou nº telemóvel e password) e submetendo para aprovação.

O sistema deverá então validar junto da WebAPI criada se se trata de um utilizador válido e em caso afirmativo irá guardar o token recebido na Base de Dados local. De seguida efetuará um novo pedido à WebAPI de forma a receber informação do utilizador guardando também essa informação Base de Dados Local.

Com o sucesso das operações na interação com a API a aplicação deverá então atualizar as definições do ViewModel associado permitindo mostrar ao utilizador o Menu mais adequado.

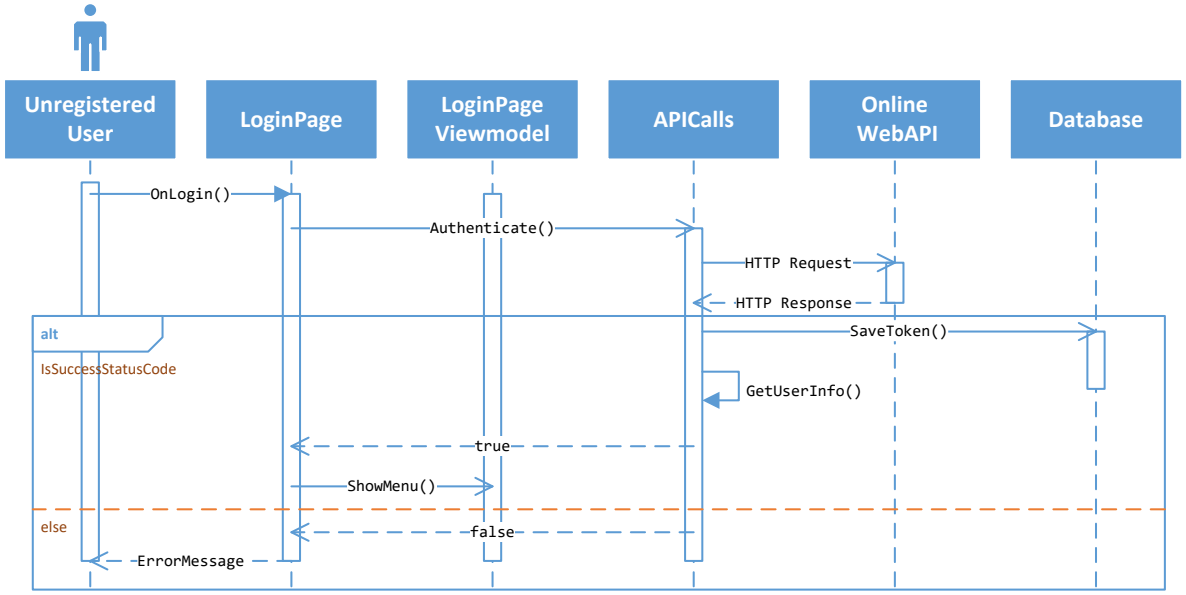


Figura 73 - Diagrama de Sequência para Autenticação de utilizador

Uma outra funcionalidade importante é a capacidade de registo na plataforma a partir da aplicação móvel. Desta forma o diagrama de sequência para o caso de uso apresentado na Tabela 25 encontra-se ilustrado na Figura 74.

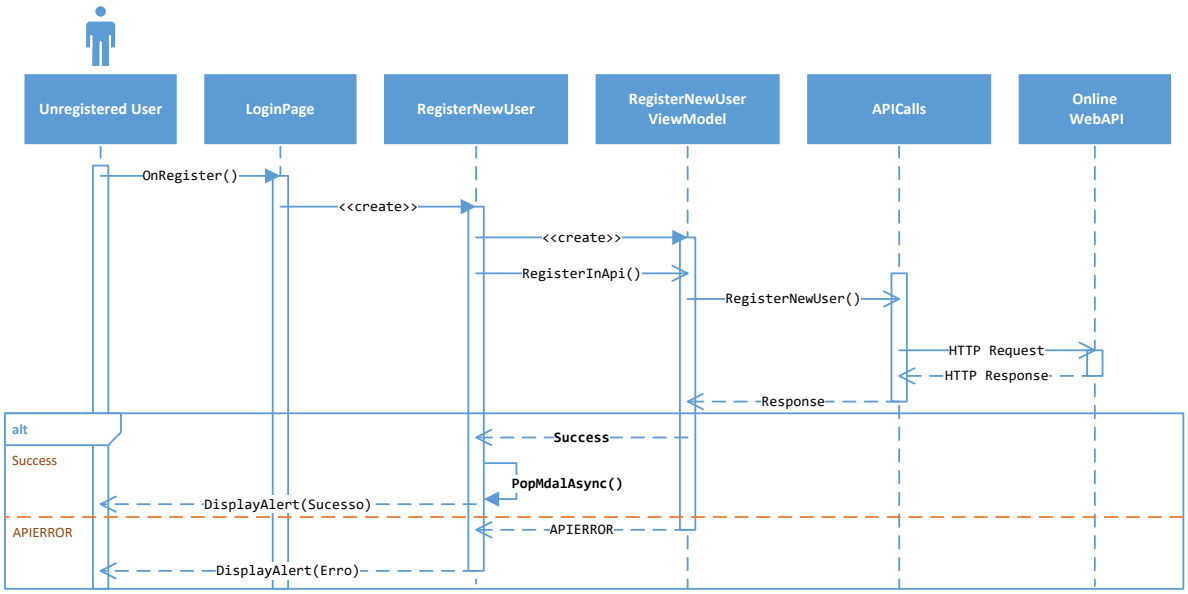


Figura 74 - Diagrama de Sequência para registo de novo utilizador

Como podemos observar o registo de novo utilizador inicia-se na página principal da aplicação irá criar uma nova View e subsequentemente um View Model associado. Com a inserção de dados no formulário constante na View e respetiva submissão será então efetuado um registo na WebAPI

utilizando mais uma vez a classe de serviço que irá presentear o utilizador com uma mensagem indicativa do sucesso caso este ocorra. Na incapacidade de efetuar o registo na API o sistema deverá mostrar um alerta indicando o erro ocorrido.

Uma outra funcionalidade considera como fundamental e porventura das mais importantes na aplicação móvel é a pesquisa. No diagrama de sequência representado na Figura75 é possível verificar o planeamento desta função.

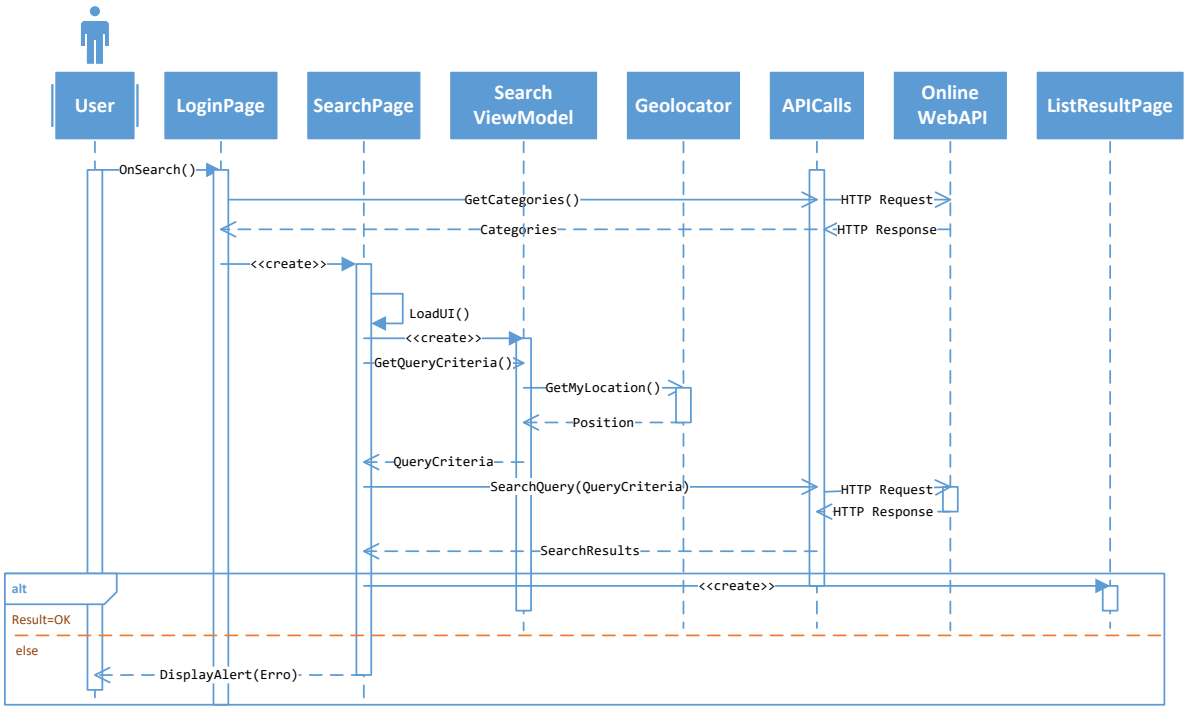


Figura 75 - Diagrama de Sequência de funcionalidade Pesquisa

O primeiro passo a ser executado quando o utilizador escolhe a opção “Pesquisa” é a chamada à API para receber uma lista de categorias disponíveis para pesquisa. Este passo permite que a aplicação esteja sempre atualizada em relação ao portal sem necessidade de novas releases. Ao receber as categorias estas serão incorporadas na classe ViewModel que apoiará a View com o formulário de detalhe de pesquisa dando a possibilidade ao utilizador de aplicar critérios e então submeter a sua pesquisa à API.

Os critérios são carregados através do ViewModel que necessita executar uma chamada à classe de serviços auxiliar que calcula a posição do utilizador através do GPS do dispositivo.Por definição, a API retorna uma lista com um número fixo de resultados indicando também o total de resultados obtidos na pesquisa. Desta forma será possível adicionar ao layout botões para solicitar os resultados seguintes, voltando a submeter o pedido à API com indicação de página seguinte.

Ao ser redirecionado para uma página com os resultados o utilizador poderá visualizar a informação em lista (visualização por defeito) ou em mapa, sendo carregados os pins correspondentes a cada uma das empresas que constem nos resultados de pesquisa.

A partir deste ponto o utilizador deverá conseguir navegar para o detalhe de cada uma das empresas através de um simples clique, quer seja no item listado ou no pino no mapa.

O diagrama representado na Figura 76 ilustra o funcionamento dos botões que possibilitam obter mais resultados na visualização em modo lista e também o funcionamento da escolha de items/pinos.

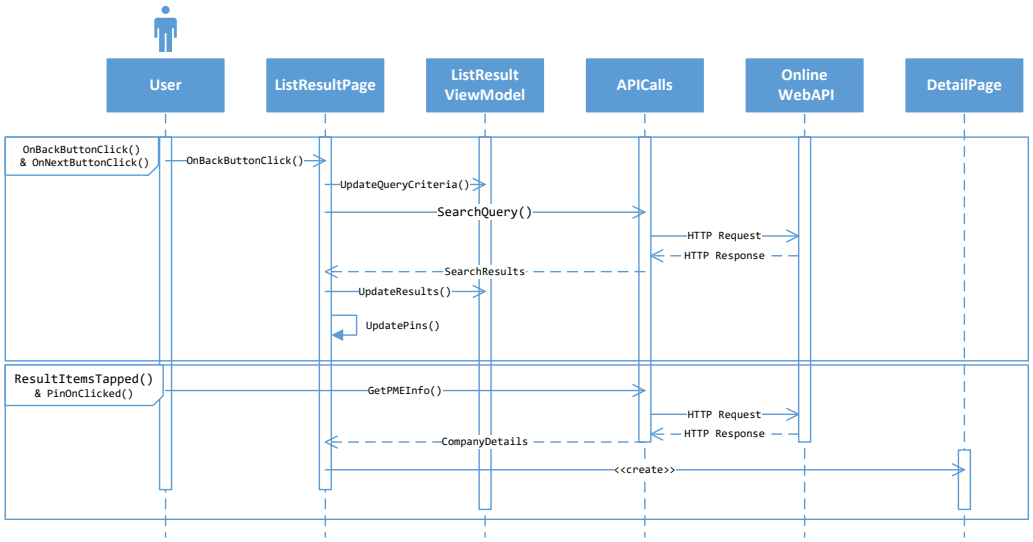


Figura 76 - Diagrama de Sequência de ações em Página ListResult

Focando nas ações *ResultItemsTapped()* e *PinOnClicked()* que representam a intensão de visualizar o detalhe de uma determinada empresa podes verificar que é efetuado um pedido de informação sobre da empresa a escolhida na API e só então é que é feita a criação e redireccionamento para a página de detalhe.

Ao inicializar o ViewModel associado a esta View é também necessário recolher mais alguma informação, nomeadamente a lista de imagens e a lista de reviews de forma a alimentar o layout. Esta informação é recolhida através de dois pedidos à API ilustrados da Figura 77.

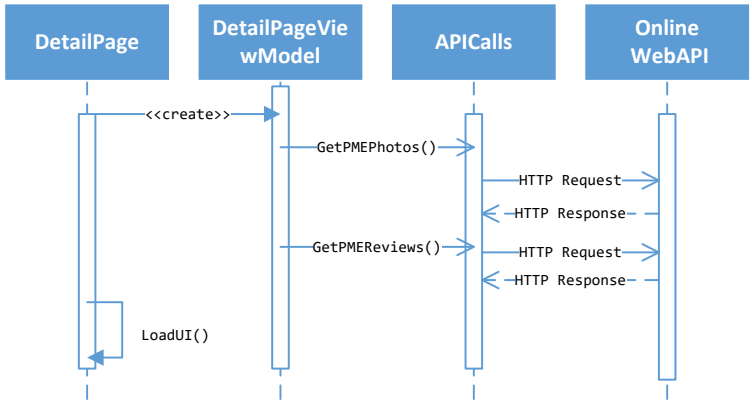


Figura 77 - Diagrama Sequência da escolha de informação para página de detalhe

A página de detalhe descrita anteriormente poderá ser o culminar da navegação de uma pesquisa, mas pode também ser atingida a partir de uma outra opção no menu do utilizador registado: Listar Empresas Associadas.

Esta funcionalidade descrita na Tabela 30 será implementada seguindo o diagrama de sequência constante na Figura 78.

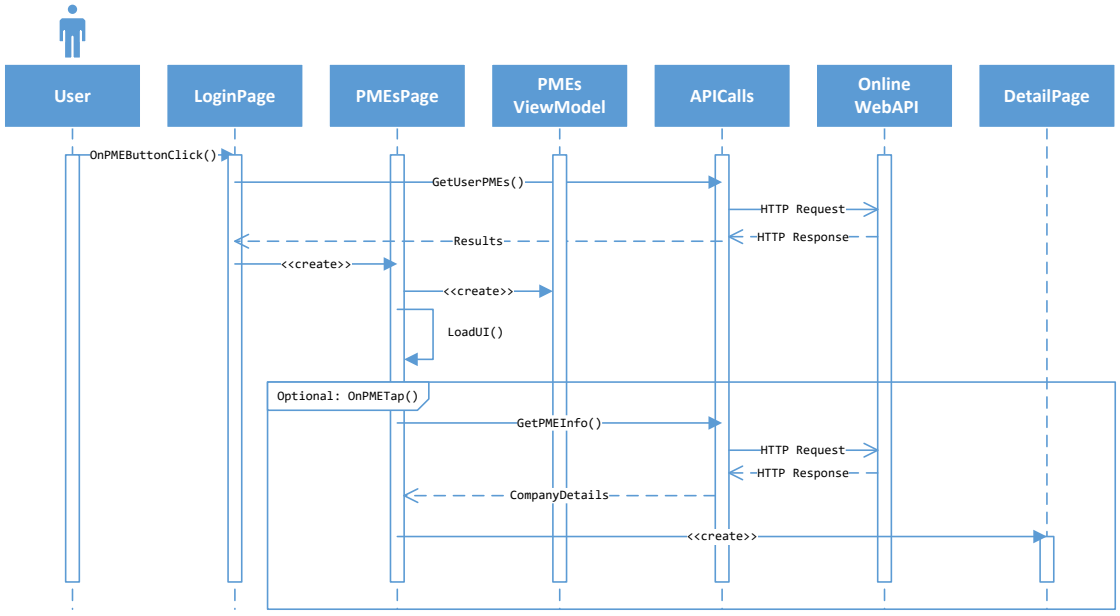


Figura 78 - Diagrama de Sequência: Listar Empresas Associadas

Ao selecionar esta opção no menu principal o utilizador dará indicação ao sistema para solicitar na API quais as empresas que lhe estão associadas e receberá uma lista. Será então redirecionado para uma página onde estarão listados os items recebidos e que, caso o utilizador pretenda, poderão ser acedidos em detalhe procedendo o sistema a uma nova chamada à API para carregar a informação e seguindo para a página de detalhe de empresa já descrita anteriormente. Para construir uma solução multiplataforma em Xamarin.Forms será necessário que todo o código comum seja desenvolvido dentro da solução Portable e que especificações de cada uma das plataformas sejam implementadas nas soluções respetivas (IOS, Android e/ou UWP – Windows Universal Platform.

4.3.2 Desenvolvimento

Após a definição dos diagramas de sequência acima referidos e que endereçam as necessidades dos diversos casos de uso passou-se ao desenvolvimento da aplicação.

Foram parametrizadas as diversas classes e organizadas numa estrutura que facilita a compreensão e a atualização do código.

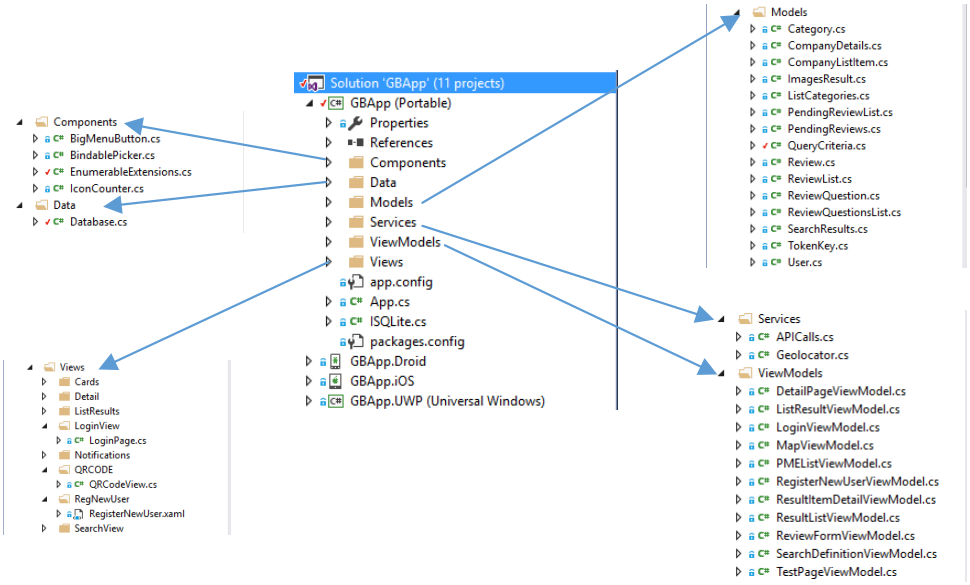


Figura 79 - Estrutura da Aplicação

Como podemos observar na Figura 79, a estrutura e todas as classes foram criadas na solução Portable permitindo assim a compilação em qualquer uma das plataformas especificadas. Ao longo desta subcapítulo serão dados exemplos de implementações que necessitaram de desenvolvimento especifico em cada plataforma.

Components: nesta pasta foram criadas as classes que correspondem a elementos visuais e que foram utilizados em alguns vires. Destacamos p.e. o *BindablePick* que possibilitou incorporar na página de pesquisa um *Picker* com capacidade de ter como lista de opções informação recolhida da API. Ou o botão existente no menu principal da aplicação (*BigMenuButton*) que inclui além do texto uma imagem personalizada. Na Figura 80 seguinte é possível ver a implementação do *BindablePicker* no layout desenvolvido em XAML e na Figura 81 a incorporação em código da inclusão do BigMenuButton no menu principal.

```
<Label Text="SubCategoria *" />
<local:BindablePicker x:Name="subcategorias" IsEnabled="{Binding EnableComponents}"
                      ItemsSource="{Binding Lista_sub_categorias, Mode=TwoWay}" />


<Label Text="Category Adicional" IsVisible="{Binding EnableSubSubCatPicker}"/>
<local:BindablePicker x:Name="subsubcategorias" IsEnabled="{Binding EnableComponents}"
                      IsVisible="{Binding EnableSubSubCatPicker}" ItemsSource="{Binding Lista_sub_sub_categorias, Mode=TwoWay}" />
```

Figura 80 - Detalhe incorporação de *BindablePicker* e XAML

```
var QRCODEBBT = new BigMenuButton("images\\qrcode.png", "QR Code ID", 0);
QRCODEBBT.SetBinding(BigMenuButton.IsVisibleProperty, "Registered");
QRCODEBBT.SetBinding(BigMenuButton.IsEnabledProperty, "EnableComponents");
var QRCODEBBTGestureRecognizer = new TapGestureRecognizer();
QRCODEBBTGestureRecognizer.Tapped += (s, e) =>
{
    OnQRCODEBBTButtonClick(s, e);
};
QRCODEBBT.GestureRecognizers.Add(QRCODEBBTGestureRecognizer);
```

Figura 81 - Detalhe incorporação de *BigMenuButton* em código.

Data: contém somente a classe que faz a conexão com a base de dados que assume a persistência temporária de informação de utilizador registado. A Figura 82 contém o código dessa classe.

Figura 82 - Classe Database e respetivos métodos

Uma vez que o desenvolvimento é feito em Xamarin.Forms Cross-Platform a utilização de SQLite teve que ser implementada como uma Interface na solução Portable e feito o respetivo código em cada uma das soluções relativas às diversas tecnologias.

A interface (Figura 83) foi criada com a assinatura de um único método que possibilita retorno da ligação à base de dados:

Figura 83 - Interface iSQLite.cs

Depois foi necessário implementar em cada um dos projetos o código especifico para cada uma das plataformas.

Figura 84 - Classe SQLite_iOS

```
public class SQLite_Android : ISQLite
{
    0 references | Fábio Mendonça, 105 days ago | 1 author, 1 change
    public SQLite_Android()
    {
    }

    #region ISQLite implementation
    2 references | Fábio Mendonça, 105 days ago | 1 author, 1 change
    public SQLite.SQLiteConnection GetConnection()
    {
        var sqliteFilename = "DBSQLite.db3";
        string documentsPath = System.Environment.GetFolderPath(System.Environment.SpecialFolder.Personal);
        var path = Path.Combine(documentsPath, sqliteFilename);
        Console.WriteLine(path);
        if (!File.Exists(path))
        {
           var s = Forms.Context.Resources.OpenRawResource(Resource.Raw.DBSQLite);  //Nome do Recurso criado
            FileStream writeStream = new FileStream(path, FileMode.OpenOrCreate, FileAccess.Write);
            ReadWriteStream(s, writeStream);
        }
        var conn = new SQLite.SQLiteConnection(path);
        return conn;
    }
    #endregion

    1 reference | Fábio Mendonça, 105 days ago | 1 author, 1 change
    void ReadWriteStream(Stream readStream, Stream writeStream)...
}
```

Figura 85 - Classe SQLite_Android

```
public class SQLite_UWP : ISQLite
{
    0 references | Fábio Mendonça, 105 days ago | 1 author, 1 change
    public SQLite_UWP()
    {
    }

    #region ISQLite implementation
    2 references | Fábio Mendonça, 105 days ago | 1 author, 1 change
    public SQLite.SQLiteConnection GetConnection()
    {
        var sqliteFilename = "DBSQLite.db3";
        string path = Path.Combine(ApplicationData.Current.LocalFolder.Path, sqliteFilename);
        var conn = new SQLite.SQLiteConnection(path);
        return conn;
    }
    #endregion
}
```

Figura 86 - Classe SQLIte_UWP

Como é possível observar nas Figuras 84, 85 e 86 a implementação em cada uma das plataformas é efetuada de forma distinta. Isto prendesse essencialmente com a forma com os recursos são acedidos.

Models: esta diretório contem todos os modelos que foram criados para servirem de apoio à aplicação. Tal como já descrito anteriormente não foi utilizada uma metodologia MVVM na usa plenitude logo os modelos criados não interagem com a persistência local. Todos os modelos criados endereçam as necessidades de envio ou recolha de dados pela API descrita anteriormente.

Na Figura 87 é possível visualizar, com exemplo desta abordagem, os modelos utilizados no processo de pesquisa quer no envio como na receção de informação na interação com a API.Em primeiro lugar temos a classe que serve de modelo ao pedido que será efetuado à API. Trata-se da *QueryCriteria.cs* e contém os campos necessários para solicitar a informação.

```
public class QueryCriteria
{
    – references | Fábio Mendonça, 75 days ago | 1 author, 1 change
    public Int32 Category { get; set; }
    – references | Fábio Mendonça, 75 days ago | 1 author, 1 change
    public String TextSearch { get; set; }
    – references | Fábio Mendonça, 102 days ago | 1 author, 1 change
    public Double radius { get; set; }
    – references | Fábio Mendonça, 75 days ago | 1 author, 1 change
    public Position MyPos { get; set; }
    – references | Fábio Mendonça, 90 days ago | 1 author, 1 change
    public String Sorting { get; set; }
    – references | Fábio Mendonça, 102 days ago | 1 author, 1 change
    public Int32 CurrentPage { get; set; }

    – references | Fábio Mendonça, 75 days ago | 1 author, 2 changes
    public QueryCriteria(Int32 category, String textSearch, Double radius,
        Position myPos, String Sorting, Int32 CurrentPage)...

    – references | Fábio Mendonça, 75 days ago | 1 author, 6 changes
    public override String ToString()
    {
        return "{\"Token\": \"" + App.DB.GetToken(1).Token + "\" ,\"Category\": \""
            + Category + "\",\"TextSearch\": \"" + TextSearch + "\" ,\"Radius\": \""
            + radius + "\" ,\"myLat\": \"" + MyPos.Latitude + "\" ,\"myLong\": \""
            + MyPos.Longitude + "\" ,\"Sorting\": \"" + Sorting + "\",\"CurrentPage\": \""
            + CurrentPage + "\"}";
    }
}
```

Figura 87 - Classe QueryCriteria

O método *ToString()* permite obter uma String com o JSON que servirá de conteúdo no pedido à API.

Por outro lado, na receção dos dados é utilizada uma classe modelo (*SearchResults*) que permite albergar quer o estado da resposta (que categoriza de o pedido foi transacionado com sucesso ou não), o nº de resultados encontrados e o primeiro lote de items (Figura 88).

```
public class SearchResults
{
    – references | Fábio Mendonça, 112 days ago | 1 author, 1 change
    public String Result { get; set; }
    – references | Fábio Mendonça, 112 days ago | 1 author, 1 change
    public Int32 TotalResults { get; set; }
    – references | Fábio Mendonça, 112 days ago | 1 author, 1 change
    public ObservableCollection<CompanyListItem> Results { get; set; }

    – references | Fábio Mendonça, 81 days ago | 1 author, 1 change
    public List<CompanyListItem> GetResults()
    {
        List<CompanyListItem> lista = new List<CompanyListItem>();
        foreach (var CLI in Results)
        {
            lista.Add(CLI);
        }
        return lista;
    }
}
```

Figura 88 - Classe SearchResult

Como é possível observar na Figura 89 o conteúdo recebido é alojado numa lista cujo modelo é a classe CompanyListItem. Esta classe serve de modelo à informação que será posteriormente passada ao layout via ViewModel.

```
public class CompanyListItem
{
    – references | Fábio Mendonça, 122 days ago | 1 author, 1 change
    public Int32 ID { get; set; }
    – references | Fábio Mendonça, 122 days ago | 1 author, 1 change
    public String Title { get; set; }
    – references | Fábio Mendonça, 122 days ago | 1 author, 1 change
    public String Logo { get; set; }
    – references | Fábio Mendonça, 122 days ago | 1 author, 1 change
    public Double Long { get; set; }
    – references | Fábio Mendonça, 122 days ago | 1 author, 1 change
    public Double Lat { get; set; }
    – references | Fábio Mendonça, 122 days ago | 1 author, 1 change
    public Double Distance { get; set; }
    – references | Fábio Mendonça, 122 days ago | 1 author, 1 change
    public Boolean Registered { get; set; }
    – references | Fábio Mendonça, 105 days ago | 1 author, 2 changes
    public Boolean IsOpen{ get; set; }
    – references | Fábio Mendonça, 81 days ago | 1 author, 1 change
    public Boolean IsClosed { get { return !IsOpen; } set { } }
    – references | Fábio Mendonça, 75 days ago | 1 author, 3 changes
    public Double Rating { get; set; }
    – references | Fábio Mendonça, 78 days ago | 1 author, 1 change
    public Int32 Points { get; set; }
    – references | Fábio Mendonça, 78 days ago | 1 author, 1 change
    public DateTime LastVisit { get; set; }
    – references | Fábio Mendonça, 78 days ago | 1 author, 1 change
    public Boolean ReviewsToDo { get; set; }

}
```

Figura 89 - Classe CompanyListItem

As três classes que acabam de ser mencionadas são um exemplo de como foram estruturadas as relações entre Model e Services. Ou seja, na ausência de persistência local não há necessidade de dotar classes da camada Model de métodos que garantam a interação com a camada de dados (BD).

Viewmodels: com a utilização do padrão MVVM as classes Viewmodel têm a função muito especifica de dotar as Views às quais estão associadas de regras de layout e, através do Binding, passagem de informação.

Como exemplo temos a classe *PMEListViewModel* que endereça regras de layout para View que mostrará ao utilizador registado as empresas que lhe estão associadas. Em primeiro lugar esta classe implementará uma interface que possibilita receber notificações de alterações nas propriedades que nele sejam definidas tornando-se necessário inicializar uma EventHandler (Figura 90) e o método que permite configurar o que acontece aquando da receção de notificações desse evento.

```
– references | Fábio Mendonça, 75 days ago | 1 author, 4 changes
public class PMEListViewModel : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    – references | Fábio Mendonça, 90 days ago | 1 author, 1 change
    private void OnPropertyChanged(string propertyName)
    {
        if (PropertyChanged != null)
            PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
    }
```

Figura 90 - Classe PMELIsViewModel - Implementação de EventHandler

De seguida foram criadas as propriedades que serão ligadas através e Binds com a view. Estas Propriedades serão definidas como privadas tendo sido também criados os métodos que possibilitam os GET e SET da informação nelas contidas.

Neste exemplo temos as duas propriedades que receberão a informação a mostrar na View, nomeadamente uma lista com as diversas empresas às quais o utilizador está associado e o número total de resultados obtidos.

As outras duas propriedades (Figura 91) servirão para definir regras de layout, mais concretamente o *isBusy* será posteriormente ligado a um elemento do tipo *ActivityIndicator* nas propriedades *IsVisible* e *IsRunning* criando assim icon visual para quando estiver a carregar dados na View (no género do tradicional "Loading"). Já a propriedade *EnableComponents* estará ligada às propriedades dos elementos que quisermos bloquear o acesso por parte do utilizador aquando do carregamento de dados.

```
private ObservableCollection<CompanyListItem> companies = new ObservableCollection<CompanyListItem>();
private Boolean isBusy;
private Boolean enableComponents;
private Int32 totalResults;
```

Figura 91 - Classe PMEListViewModel: definiçãode variáveis/propriedades

```
public ObservableCollection<CompanyListItem> Companies
{
    get
    {
        return companies;
    }
    set
    {
        companies = value;
        OnPropertyChanged("Companies");
    }
}
```

```
public bool EnableComponents
{
    get
    {
        return enableComponents;
    }
    set
    {
        enableComponents = value;
        OnPropertyChanged("EnableComponents");
    }
}
```

```
0 references | Fábio Mendonça, 90 days ago | 1 author, 1 change
public bool IsBusy
{
    get
    {
        return isBusy;
    }
            set
    {
        isBusy = value;
        OnPropertyChanged("IsBusy");
    }
}
```

```
0 references | Fábio Mendonça, 90 days ago | 1 author, 1 change
public int TotalResults
{
    get
    {
        return totalResults;
    }
    set
    {
        totalResults = value;
        OnPropertyChanged("TotalResults");
    }
}
```

Figura 92 - Classe PMEListViewModel: Gets e Sets de Propriedades

A inclusão do método *OnPropertyChanged("nomedapropriedade")* na parametrização dos SET de cada propriedade (Figura 92) permite que, cada vez que é alterado o valor dessa propriedade é forçada a sua atualização o que significa que na View correspondente o conteúdo do elemento gráfico ao qual esta propriedade está ligada será também atualizado.

Apesar de não ser utilizado no exemplo acima, existem casos onde o Set de uma propriedade desencadeia a atualização de várias propriedades. Na imagem seguinte temos o exemplo de um outro Viewmodel (*ListResultViewModel*) onde a atualização da numeração da página (propriedade *CurrentPage* conforme Figura 93) desencadeia atualizações em duas outras propriedades (*NextButton* e *BackButton*) que estão associadas à propriedade *IsEnable* de dois botões existentes na respetiva View.

```
private Boolean backButton;
private Boolean nextButton;
private Int32 currentPage;

//GETS & SETS
– references | Fábio Mendonça, 75 days ago | 1 author, 2 changes
public int CurrentPage
{
    get
    {
        return currentPage;
    }

    set
    {
        currentPage = value;
        if (value == 1)
        {
            BackButton = false;
        }
        else if (CurrentPage == ((TotalResults + 15 - 1) / 15))
        {
            NextButton = false;
        }
        else
        {
            BackButton = true;
            NextButton = true;
        }
        OnPropertyChanged("CurrentPage");
        OnPropertyChanged("NextButton");
        OnPropertyChanged("BackButton");
    }
}
```

Figura 93 - Classe ListResultViewModel: Detalhe de propriedade CurrentPage

Views: constituído a componente de apresentação para o utilizador as vires criadas foram maioritariamente desenvolvidas em código existindo somente uma View desenvolvida em XAML.

Antes de descrevermos a forma como foram construídas as Views foi necessário adicionar aos projetos elementos que viriam a ser utilizados nesta camada. Estamos a falar da utilização de mapas e de QRCode.

Para uso de mapas foram instalados em todos os projetos os pacote NuGET de *Xamarin.Forms.Maps* e *Xamarin.GooglePlayServices.Maps* (apenas para o projeto Android). Para responder às necessidades de utilização de QRCode foi também instalado pacote *ZXing.Net.Mobile*

A inclusão destas duas componentes no projeto IOS foi efetuada de uma forma muito simples adicionando as respetivas inicializações na classe *AppDelegate* (Figura 94):

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{

    global::Xamarin.Forms.Forms.Init();
    global::Xamarin.FormsMaps.Init();
    ZXing.Net.Mobile.Forms.iOS.Platform.Init();
    LoadApplication(new App());
    return base.FinishedLaunching(app, options);
}
```

Figura 94 - IOS AppDelegate.cs Inicialização de Mapas e QRCode

No projeto Android foram também introduzidas as referências aos pacotes no *MainActivity* (Figura 95) imediatamente antes da chamada da aplicação e um método que possibilita ultrapassar o pedido de autorização ao utilizador por parte da biblioteca ZXING.

```
public class MainActivity : global::Xamarin.Forms.Platform.Android.FormsApplicationActivity
{
    2 references | Fábio Mendonça, 17 days ago | 1 author, 4 changes
    protected override void OnCreate(Bundle bundle)
    {
        base.OnCreate(bundle);
        global::Xamarin.Forms.Forms.Init(this, bundle);
        global::Xamarin.FormsMaps.Init(this, bundle);
        ZXing.Net.Mobile.Forms.Android.Platform.Init();
        LoadApplication(new App());
    }

    0 references | Fábio Mendonça, 17 days ago | 1 author, 1 change
    public override void OnRequestPermissionsResult(int requestCode, string[] permissions, Permission[] grantResults)
    {
        global::ZXing.Net.Mobile.Forms.Android.PermissionsHandler.OnRequestPermissionsResult(requestCode, permissions, grantResults);
    }
}
```

Figura 95 - MainActivity.cs: Inicialização de mMapas e QRCode

Ainda no projeto Android foi necessário editar o AndroidManifest para inserir uma Key válida (Figura 96) para acesso ao GoggleMapsAPI (Google, 2016)

```
<meta-data android:name="com.google.android.maps.v2.API_KEY" android:value="AIzaSyBsuCE4QtifB8tdbXq6OIWPcfIN67668AXXXX" />
```

Figura 96 - Detalhe AndroidManifest.xml

Para o projeto Windows (UWP) foram adicionadas ao code-behind do *Mainpage.xaml* (Figura 97) as inicializações de ambos os componentes sendo que no Xamarin.Forms.Maps foi necessária a inserção de uma key de acesso ao BingMapsPortal (Microsoft, 2016).

```
public MainPage()
{
    this.InitializeComponent();
    Xamarin.FormsMaps.Init("LOmPY44FSv9k8LqwVMgn~2ALxOiC9ORe7GDe3oglWGQ~AqRrwn6eJqORDNnhS7JS-JvbQKpv_ZMOcmm1_tZeygAzXevvb2nPI_9JT257DLIvzszsxs");
    ZXing.Net.Mobile.Forms.WindowsUniversal.ZXingScannerViewRenderer.Init();
    LoadApplication(new GBApp.App());
}
```

Figura 97 – Classe *MainPage.xaml*: Inicialização de Mapas e QRCode

Com estas componentes devidamente parametrizadas em todos os projetos foram então desenvolvidas as Views que possibilitarão ao utilizador navegar na aplicação.

Um dos exemplos de Views elaborado em código é a classe *MyPMEsPage* (Figura 98) que implementa o layout associado ao Viewmodel *PMEListViewModel* explicado anteriormente.

```
public class MyPMEsPage : ContentPage
{
    private PMEListViewModel viewmodel = new PMEListViewModel();

    SearchBar searchBar;
    Button viewAll;
    StackLayout cardstack;

    – references | 0 changes | 0 authors, 0 changes
    public MyPMEsPage(SearchResults pmesData)
    {
        viewmodel.Companies = pmesData.Results;
        viewmodel.TotalResults = pmesData.TotalResults;
        BindingContext = viewmodel;
        LoadUI();
    }
```

Figura 98 - Classe PMEListVIewModel

De inicio é efetuada a inicialização do ViewModel que ficará associado e conterá as regras e informação e no construtor da classe *MyPMEsPage* que recebe a informação sobre os resultados obtidos via API são passados os valores para as propriedades da Viewmodel (*Companies* e *TotalResults*) e efetuada a definição de qual o *BindingContext* associado.

Nesta classe foi criado um método *LoadUI()* (Figura 99) onde foram introduzidos os diversos componentes que constituição a página. Isto torna mais simples a manutenção do código tendo todos os componentes visuais inseridos neste método e permitindo, caso se justifique, que sejam feitas chamadas assíncronas a outros serviços ou métodos algo que não pode ser efetuado no construtor da classe. Para a construção do layout procedeu-se à criação dos vários elementos e à definição de Bindings e de ações sempre que aplicável. Neste caso temos a criação de um *ActivityIndicator* que estará ligado à propriedade *IsBusy* do ViewModel associado tal como já descrito anteriormente. Outro elemento é uma barra de pesquisa (*SearchBar*) que tem como ação executar o método *OnSearch*() passando como argumento o valor contido na sua propriedade "Text".

```
var loadingSymbol = new ActivityIndicator {Color = Color.Blue};
loadingSymbol.SetBinding(ActivityIndicator.IsVisibleProperty, "IsBusy");
loadingSymbol.SetBinding(ActivityIndicator.IsRunningProperty, "IsBusy");

searchBar = new SearchBar
{
    HorizontalOptions = LayoutOptions.FillAndExpand,
    Placeholder = "Procurar por Nome ",
    BackgroundColor = Color.White,
    CancelButtonColor = Color.Gray,
    SearchCommand = new Command(() =>
    {
        OnSearch(searchBar.Text);
    })
};

viewAll = new Button
{
    BackgroundColor = Color.Gray,
    Text = "Ver Todos",
    IsVisible = false
};
viewAll.Clicked += ViewAll_Clicked;

var searchLayout = new StackLayout
{
    Orientation = StackOrientation.Horizontal,
    HorizontalOptions = LayoutOptions.FillAndExpand,
    Children = { searchBar, viewAll }
};
```

Figura 99 - Classe MyPMEsPage: Detalhe método LoadUI()

Para alimentar a lista de items foi criado um ciclo carrega num StackLayout as várias empresas contantes na propriedade "*Companies*" do Viewmodel e adiciona a cada um dos elementos visuais a capacidade de reagir ao toque do utilizador referenciando para a ação *OnPMETAP*(s,e) (Figura 100).

Por fim todos os elementos são adicionados a um *StackLayout* final que é encapsulado num *Scrollview* e apresentado como conteúdo da página. Ao utilizarmos *StackLayout* e *ScrollView* conseguimos obter uma página organização e com capacidade de deslocamento vertical facilitando a navegação ao utilizador. Após o carregamento das componentes gráficas desenvolveram-se os métodos associados às ações desta página.

```
public async void OnPMETap(object s,EventArgs e)
{
    var item = s as CardView;
    CompanyDetails pme = await App.API.GetPMEInfo(item.PME.ID);
    if (pme != null)
    {
        await Navigation.PushModalAsync(new DetailPage(pme, true));
    }
    else
    {
        await DisplayAlert("Alert!","Erro ao receber Informação", "OK");
    }
}
```

Figura 100 - Classe MyPMEsPage: método OnPMETap()

O método *OnPMETap*() permite redirecionar o utilizador para a página de detalhe da respetiva PME que é definida pelo objeto enviado como argumento na chamada deste método. Antes de redirecionar o utilizador é necessário proceder à recolha da informação a mostrar na próxima página (neste caso a *DetailPage*), pelo que é utilizada a classe de serviços *APICalls* para o fazer.

Outro método relevante nesta página é a pesquisa e atualização da lista de empresas com base num critério de texto. O método *OnSearch*() (Figura 101) ativado a partir da barra de pesquisa efetua um varrimento à lista "*Companies*" comparando a String passada como argumento com o título das empresas contidas na lista. De seguida, caso existam resultados, limpa o *StackLayout* que contém os items e adiciona a nova lista filtrada.

```
public async void OnSearch(string searchText)
{
    var count = 0;
    ObservableCollection<CompanyListItem> UpdateList = new ObservableCollection<CompanyListItem>();
    foreach (var pme in viewmodel.Companies)
    {
        if (pme.Title.IndexOf(searchText, StringComparison.CurrentCultureIgnoreCase) >= 0)
        {
            count++;
            UpdateList.Add(pme);
        }
    }
    if (count > 0)
    {
        cardstack.Children.Clear();
        foreach (var pme in UpdateList)
        {
            var tapGestureRecognizer = new TapGestureRecognizer();
            tapGestureRecognizer.Tapped += (s, e) =>
            {
                OnPMETap(s,e);
            };
            var c = new CardView(pme);
            c.GestureRecognizers.Add(tapGestureRecognizer);
            cardstack.Children.Add(c);
        }
        viewAll.IsVisible = true;
    }
    else
    {
        await DisplayAlert("Alert!", "Não foram encontrados Registos", "OK");
    }
```

Figura 101 - Classe MyPMEsPage.cs: Método OnSearch()

Um outro exemplo que jugamos relevante referir é a implementação da componente Mapa nas Views. Neste caso o mapa foi inserido na página de Listagem de resultados e pesquisa com recurso ao seguinte código (Figura 102).

```
    MyMap = new Xamarin.Forms.Maps.Map(MapSpan.FromCenterAndRadius(
new Position(viewmodel.Query.MyPos.Latitude, viewmodel.Query.MyPos.Longitude), Distance.FromKilometers(3)))
    {
        HasZoomEnabled = true,
        Margin = new Thickness(5),
        IsShowingUser = true,
        MapType = MapType.Street
    };
    MyMap.SetBinding(Map.IsVisibleProperty, "VisibleMap");
```

Figura 102 - Exerto de código de implementação de Mapa

De seguida foi executado um ciclo que adiciona Pinos no mapa (Figura 103) de acordo com o conteúdo da lista de companhias existentes no ViewModel associado (neste caso ListResultViewModel) com a respetiva localização e evento para ligar com o toque do utilizador.

```
public void UpdatePins(Map _map)
{
    _map.Pins.Clear();
    foreach (var PME in viewmodel.Companies)
    {
        var pinPME = new Pin
        {
            Type = PinType.SearchResult,
            Position = new Position(PME.Lat, PME.Long),
            Label = PME.Title,
            Address = viewmodel.Companies.IndexOf(PME).ToString()
        };
        pinPME.Clicked += PinOnClicked;
        MyMap.Pins.Add(pinPME);
    }
}
```

Figura 103 - Exerto de código para adição de pinos no mapa

Como já referido, uma das páginas não foi desenvolvida em código, mas sim em XAML. Trata-se da página para registo de novo utilizador (*RegisterNewUser)* (Figura 104) que tem como contexto de ligação a classe *RegisterNewUserViewModel* e com o respetivo Code-Behind.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="GBApp.Views.RegNewUser.RegisterNewUser"
             Title="Registar Novo Utilizador"
             Padding="10,40,10,10">
  <StackLayout>
    <Image Source="gblogo.jpg" HorizontalOptions="Center"/>
    <ActivityIndicator Color="Blue" IsRunning="{Binding IsBusy}" IsVisible="{Binding IsBusy}"/>

    <Label Text="Nome" IsEnabled="{Binding EnableComponents}"/>
    <Entry Keyboard="Text" IsEnabled="{Binding EnableComponents}" Text="{Binding Nome, Mode=TwoWay}"></Entry>

    <Label Text="Email"/>
    <Entry Keyboard="Email" IsEnabled="{Binding EnableComponents}" Text="{Binding Email, Mode=TwoWay}"></Entry>

    <Label Text="Telemóvel" IsEnabled="{Binding EnableComponents}"/>
    <Entry Keyboard="Telephone" IsEnabled="{Binding EnableComponents}" Text="{Binding Telefone, Mode=TwoWay}"></Entry>

    <Label Text="Password" IsEnabled="{Binding EnableComponents}"/>
    <Entry Keyboard="Text" IsPassword="True" IsEnabled="{Binding EnableComponents}" Text="{Binding Password, Mode=TwoWay}"></Entry>

    <Label Text="Repita a Password"/>
    <Entry Keyboard="Text" IsPassword="True" IsEnabled="{Binding EnableComponents}" Text="{Binding Password1, Mode=TwoWay}"></Entry>

    <Label Text="{Binding Message, Mode=TwoWay}" IsVisible="{Binding Warning}" TextColor="Red" />
    <Button Clicked="OnRegisterUser" IsEnabled="{Binding EnableComponents}" CommandParameter="{Binding .}" Text="Register"/>
  </StackLayout>
</ContentPage>
```

Figura 104 - Classe RegisterNewUser

A inclusão de elementos em formato XAML é também feita de forma simples, sendo o Binding efetuado com o nome das propriedades e tendo por base o contexto definido no Code-Behind. Neste caso temos somente a inclusão de uma série de *Labels* e *Entry* que constituem um formulário de registo sendo cada *Entry* associada uma componente especifica do Viewmodel. No Code-Behind temos somente o construtor sem argumentos que inicializa a componente gráfica através da instrução *InitializarComponent*() e efetua a ligação com a classe ViewModel respetiva (Figura 105).

```
public partial class RegisterNewUser : ContentPage
{
    private RegisterNewUserViewModel viewmodel = new RegisterNewUserViewModel();
    – references | Fábio Mendonça, 75 days ago | 1 author, 1 change
    public RegisterNewUser()
    {
        InitializeComponent();
        BindingContext = viewmodel;
        IsEnabled = true;
    }

    – references | Fábio Mendonça, 75 days ago | 1 author, 1 change
    public async void OnRegisterUser(object sender, EventArgs e)
    {
        var resp = await viewmodel.RegisterInAPI();
        if (resp =="Success")
        {
           await DisplayAlert("GBApp","O seu registo foi efetuado com sucesso.", "OK");
           await Navigation.PopModalAsync();
        }
        else if (resp =="APIERROR")
        {
            await DisplayAlert("GBApp", "O registo falhou. \n Motivo:"+resp, "OK");
        }
    }
}
```

Figura 105 - RegisterNewUser (Code-Beind)

Ainda neste excerto de código é possível analisar o método utilizado para submeter o registo do utilizador que utiliza um método *RegisterInAPI*() disponível no ViewModel e validando a resposta de retorno (Figura 106).

```
public async Task<String> RegisterInAPI()
{
    Loading(true);
    Warning = false;
    if (Password != Password1)
    {
        Warning = true;
        Message = "Passwords Don't Match";
        Loading(false);
        return "Error!Password";
    }
    else if (Name !="" || Phone !="" || Email!="")
    {
        Warning = true;
        Message = "Please fill all fields";
        return "Erro!NullFields";
    }
    else
    {
        var content = "{\"Token\":\""+ App.DB.GetToken(1).Token + "\"Name\": \""
            + Name + "\" ,\"Address\": \"\",\"City\": \"\",\"PostalCode\": \"\",\"Email\": \""
            + Email + "\",\"Phone\":\"" + Phone + "\",\"Password\": \"" + Password + "\"}\"}";

        var RegNewUserRequest = await API.RegisterNewUser(content);

        if (!RegNewUserRequest.IsSuccessStatusCode)
        {
            Loading(false);
            return RegNewUserRequest.StatusCode.ToString();
        }
        else
        {
            dynamic RegNewUserRequestResponse = JsonConvert.DeserializeObject(RegNewUserRequest.Content.ToString());
            Loading(false);
            return (String)RegNewUserRequestResponse.Result;
        }
    }
}
```

Figura 106 - Viewmodel RegisterNewUservViewModel.cs : Metodo RegisterInAPI()

Este método valida a informação contida nas propriedades (Atualizadas através do binding com a View) e caso não seja detetado nenhum erro é executado o pedido de utilizador junto da API recorrendo à classe de serviços.

Services: este diretório contém as classes que disponibilizam serviços à aplicação, mais concretamente a Geolocalização (*Geolocartor*) e o Comunicação com a API (*APICalls*).

Ambos as classes foram construídas utilizando o padrão Singleton de forma a garantir que existe apenas uma instância dessas classes e mantendo assim um ponto de acesso único e global às mesmas.

No caso da Geolocalização a implementação foi efetuada utilizando a biblioteca *Geolocator* já mencionada anteriormente e criado um método que possibilita o retorno das coordenadas geográficas do utilizador utilizando um formato disponível no Xamarin.Forms.Maps (Figura 107) , mantendo assim a consistência do tido de dados quando aplicado sobre a componente mapa.

```
public async Task<Xamarin.Forms.Maps.Position> GetMyLocation()
{
    var locator = CrossGeolocator.Current;
    var geoposition = await locator.GetPositionAsync(timeoutMilliseconds: 10000);
    return new Xamarin.Forms.Maps.Position(geoposition.Latitude, geoposition.Longitude);
}
```

Figura 107 - Classe Geolocator: Método GetMyLocation()

Para otimizar a manutenção da aplicação criou-se uma classe para efetuar toda a comunicação com a API. Esta classe foi implementada seguindo o padrão Singelton e incorpora os métodos utilizadores nos mais diversos pedidos à API da plataforma.

Foram definidos alguns parâmetros transversais nesta classe que estão relacionados com o URI para onde os pedidos devem ser direcionados (Figura 108).

```
public class APICalls
{
    private static APICalls instance = new APICalls();
    private String uri = "http://advft-ws.azurewebsites.net/";
    private HttpClient client = new HttpClient();

    0 references | Fábio Mendonça, 127 days ago | 1 author, 1 change
    private APICalls() { }

    0 references | Fábio Mendonça, 127 days ago | 1 author, 1 change
    public static APICalls getInstance()
    {
        return instance;
    }
```

Figura 108 - Classs APICalls

Nas Figuras 109 e 110 encontram-se algumas implementados feitas nesta classe e que refletem funções já descritas ao longo deste subcapítulo.

```
public async Task<Boolean> Authenticate(String Login, String Password)
{
   var content = "{\"Login\": \"" + Login + "\" ,\"Password\": \"" + Password + "\" }";
   HttpContent contentPost = new StringContent(content, Encoding.UTF8, "application/json");
   HttpResponseMessage TokenRequest = await client.PostAsync(uri + "/api/user", contentPost);
    if (TokenRequest.IsSuccessStatusCode) //Se recebe resposta da API
    {
        var token = JsonConvert.DeserializeObject<TokenKey>(TokenRequest.Content.ReadAsStringAsync().Result);
        token.ID = 1;
        if (token.Result.Equals("FAIL")) //Se a resposta é um erro da aplicação
        {
            return false;
        }
        else //Se recebe um token Valido
        {
            App.DB.SaveToken(token);
            var UserDataRequest = await GetUserInfo(token.Token);

            if (UserDataRequest.IsSuccessStatusCode) //se recebe uma resposta da API
            {
                var user = JsonConvert.DeserializeObject<User>(UserDataRequest.Content.ReadAsStringAsync().Result);
                user.ID = 1;
                App.DB.UpdateUser(user);
                return true;
            }
            else return false;
        }
    }
    else return false;
}
```

Figura 109 - APICalls: Método Aunthenticate()

```
public async Task<HttpResponseMessage> GetUserInfo(String token)
{
    try
    {
        return await client.GetAsync(uri + "/api/user/"+token);
    }
    catch
    {
        return new HttpResponseMessage(System.Net.HttpStatusCode.Unauthorized);
    }
}
```

Figura 110 - APICAlls: Método GetUserinfo

Os dois métodos acima são utilizados no processo de autenticação do utilizador. Podemos verificar que é construída uma String com o conteúdo a ser passado à API por POST. Caso o retorno seja feito convenientemente é efetuada a conversão da resposta para uma variável definida com base no modelo TokenKey e se a nível aplicacional o retorno seja uma resposta positiva relativamente à autenticação do utilizador é executado método *GetUserInfo*(). Este método usa um GET à API para recolher a informação do utilizador, informação essa que é depois convertida com base na classe modelo User, atualizando a informação da base de dados e retornando à página que solicitou esta autenticação. Nessa altura é efetuada a atualização do Viewmodel que procede ao *enforcing* de regras sobre o respetivo View.

```
//****  Metodo para fazer registo/update de utilizador na API.
– references | Fábio Mendonça, 75 days ago | 1 author, 3 changes
public async Task<HttpResponseMessage> RegisterNewUser(String content)
{
    try
    {
        HttpContent contentPost = new StringContent(content, Encoding.UTF8, "application/json");
        return await client.PutAsync(uri, contentPost);
    }
    catch
    {
        return new HttpResponseMessage(System.Net.HttpStatusCode.BadRequest);
    }
}
```

Figura 111 - APICAlls: Método RegisterNewUser()

Na Figura 111 pode ser visualizada a utilização do verbo PUT para inserir na plataforma um novo utilizador. Como podemos ver o conteúdo é recebido como argumento e submetido o pedido de forma assíncrona.

Em alguns casos é necessário efetuar a conversão de uma determinada classe recebida como argumento par formato JSON para então submeter o pedido à API. Na Figura 112 é possível identificar a conversão de um objeto do tipo *QueryCriteria* em JSON utilizando um método da biblioteca Newtonsoft.Json.

```
//****  Metodo para fazer pesquisa na  API.
– references | Fábio Mendonça, 48 days ago | 1 author, 4 changes
public async Task<SearchResults> SearchQuery(QueryCriteria q)
{   try{
        var content = await Task.Run(() => JsonConvert.SerializeObject(q));
        HttpContent contentPost = new StringContent(content, Encoding.UTF8, "application/json");
        var QueryRequest = await client.PostAsync(uri + "/api/search", contentPost);
        return JsonConvert.DeserializeObject<SearchResults>(QueryRequest.Content.ReadAsStringAsync().Result);
    }
    catch
    {
        return new HttpResponseMessage(System.Net.HttpStatusCode.BadRequest);
    }
}
```

Figura 112 - APICAlls: Método SearchQuery();

Neste caso o retorno da função é também uma conversão do conteúdo da reposta da API num objeto do tipo *SearchResults.*

Com a implementação de toda a estrutura anteriormente descrita e seguindo este padrão de desenvolvimento foi possível criar as diversas Views associadas a cada uma das funcionalidades definidas nos casos de uso. Cada View teve associada uma classe de ViewModel de forma a gerir o layout com implementação de *Eventhandlers* para alteração no estado das propriedades e com uma recolha (ou envio) de informação através de uma camada de serviços que assegura totalmente a comunicação com a API. Esta estruturação possibilita a compilação dos projetos nas diversas plataformas, ou seja, é possível publicar a aplicação em IOS, Android e Windows UWP o que aumenta o número de dispositivos capazes de correr esta aplicação sem implicar o desenvolvimento em separado das várias plataformas.

No que respeita ao aspeto gráfico da aplicação não foi dado muita importância nesta fase dado que o que era pretendido era uma prova de conceito e uma aplicação funcional principalmente com a capacidade de geolocalização fiável e integração com a API de forma rápida e robusta. Contudo, mesmo numa versão protótipo tentou-se que o aspeto fosse de uma aplicação User Friendly e com uma capacidade de navegabilidade simplificada.

Na Figura 113 podemos observar além dos ecrãs da aplicação testada em Windows Phone (emulador) o fluxo de navegação entre ecrãs da aplicação demonstrando a navegabilidade esperada.

Figura 113 - Ecrãs da aplicação

4.3.3 Testes

Durante o desenvolvimento da aplicação foram efetuados vários testes de utilização quer em ambiente Windows UWP como em ambiente Android. Devido a limitações técnicas, mais concretamente a ausência de equipamentos Mac, não foi possível testar a compilação do projeto em IOS e por conseguinte não foi feito qualquer teste de utilização nessa plataforma.

Nas plataformas UWP e Android os testes efetuados foram essencialmente de navegação na aplicação e verificação do correto funcionamento da classe de serviços APICalls.cs não tendo sido feitos testes unitários a todo o desenvolvimento.

Dado que durante o processo de desenvolvimento a API não estava concluída foi desenvolvida uma API de testes e alojada na Cloud Azure, tendo sido parametrizados os controllers adequados à simulação dos pedidos à API e ao retorno de informação esperada.

Na vertente de navegação foram testadas todas as funcionalidades desenvolvidas utilizando emuladores disponibilizados sobre Hyper-V:

- **Mobile Emulator 10.0.10586.0 720p 5 inch 1GB**: que corresponde a um dispositivo com Windows 10 com ecrã de 5" e definição de 720p. Este emulador foi parametrizado com 1 GB de memória RAM e processador x86, emulação de ligação à Internet utilizando um switch virtual com bridge à placa de rede do portátil onde estava a ser feito o desenvolvimento e módulo GPS com simulação de localização. (Figura 114)

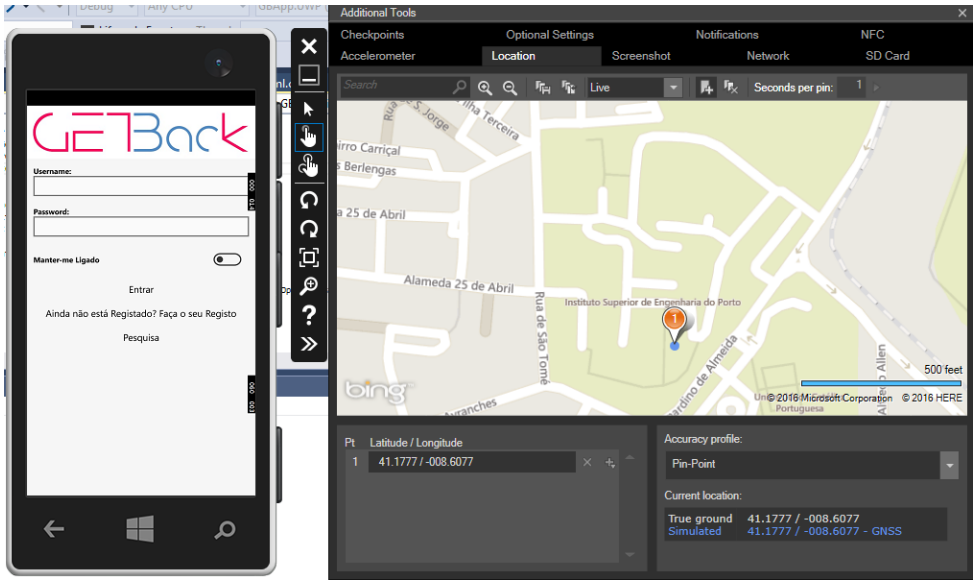


Figura 114 - Emulador Windows 10

- **5"KitKat (4.4) XXHDPI Phone (Android 4.4 – API19):** correspondendo a um dispositivo com Android Kitkat, ecrã de 5" e definição de 960X1600. Este emulador foi também parametrizado com emulação de ligação à Internet utilizando um switch virtual com bridge à placa de rede do portátil onde estava a ser feito o desenvolvimento e módulo GPS com simulação de localização. (Figura 115)

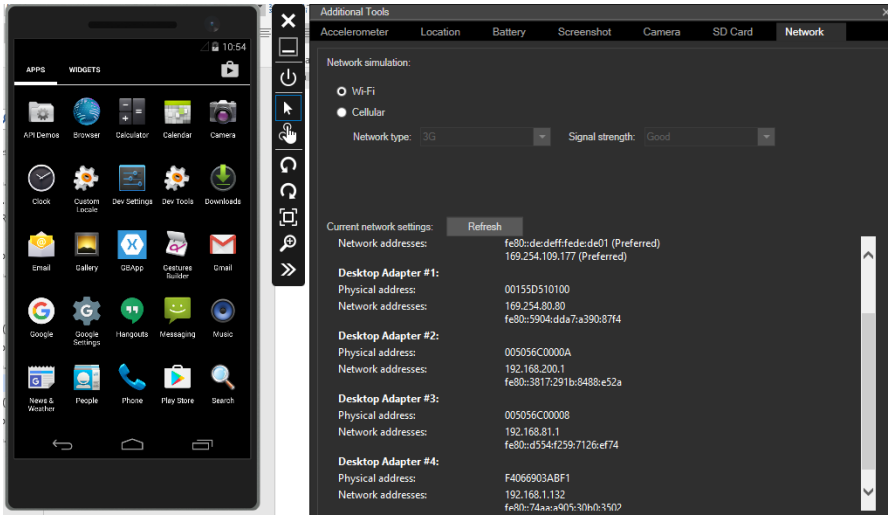


Figura 115 - Emulador Android 4.4

Dos testes efetuados não foram retiradas métricas de utilização da aplicação nem tempos de resposta da API, contudo verificou-se uma lentidão no primeiro acesso à API sendo essa latência diminuída nos pedidos subsequentes.

4.4 Aplicação Móvel para Empresas

Como referido anteriormente, esta aplicação visa dotar as empresas de um meio simples para atribuição e consumo de pontos não tendo sido idealizada como uma solução que possibilite uma gestão avançada por parte das empresas. Com a API em funcionamento é possível a integração de ferramentas de terceiros (por exemplo soluções ERP ou POS) com a plataforma, dispensando esta aplicação, contudo caso tal não exista é necessário dotar as empresas de uma forma simples de concretizar as operações básicas. De seguida está explicado o planeamento efetuado e alguns pormenores na implementação desta aplicação.

4.4.1 Planeamento

Tal como na aplicação móvel para os utilizadores, o padrão no qual foi baseada a construção desta aplicação foi o MVVM. Desta vez não está planeado qualquer tipo de persistência de dados pelo que todas as operações CRUD serão executadas através da API. A Figura 116 ilustra as camadas definidas.

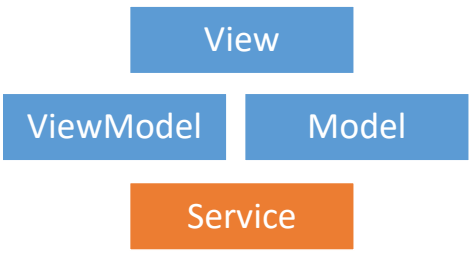


Figura 116 - Definição de Camadas

Apesar desta aplicação ser muito minimalista foram delineados os diagramas de sequência que endereçam as funcionalidades descritas nos casos de uso estabelecidos.

Para a autenticação a sequência de ações deverá passar pelo envio de das credenciais à API que validará e em caso de sucesso retornará o Token e o URL para o Logotipo da empresa criando a página principal da aplicação e colocando-a no topo do stack de visualização. (Figura 117)

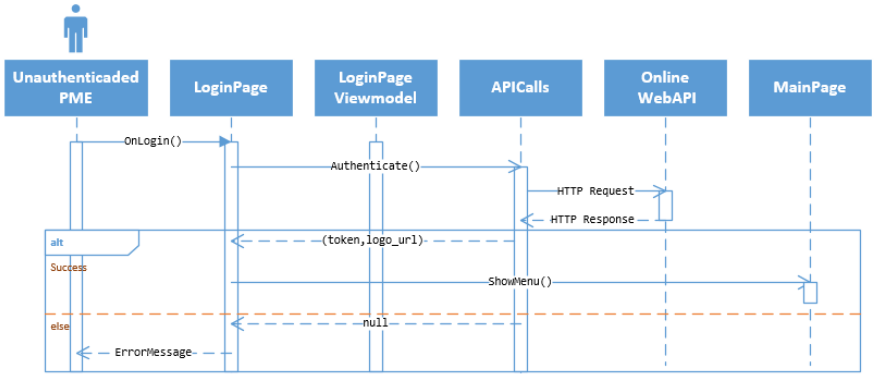


Figura 117 - Diagrama de Sequêcnia de Autenticação de PME

Para a componente de pesquisa de utilizador o diagrama de sequência é o idealizado encontra-se na Figura 118.

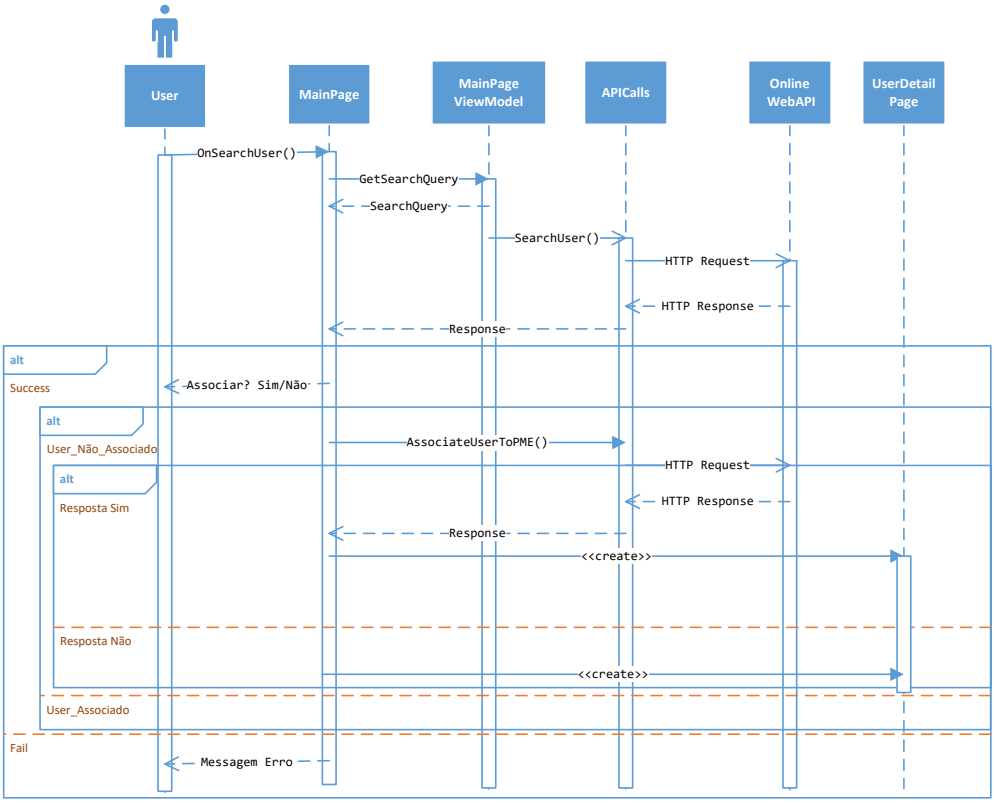


Figura 118 - Diagrama de Sequência Pesquisa de Utilizador

Aquando de uma pesquisa por um utilizador, através do email ou número de telemóvel, o pedido de pesquisa é efetuado á API e caso retorne um utilizador registado na plataforma valida se este já está associado à empresa. Caso o utilizador ainda não esteja associado é oferecida a possibilidade de efetuar essa associação passando de seguida à página de detalhe de utilizador onde se encontram as opções de adição/subtração de pontos e desassociação de utilizador.

A página de detalhe de utilizador deverá disponibilizar a leitura rápida do nome do utilizador e da quantidade de pontos acumulados. A partir desta página será possível avançar para uma das 3 opções:

- Adicionar Pontos
- Remover Pontos
- Desassociar utilizador

A adição e remoção de pontos deverá obedecer aos diagramas ilustrados pelas Figuras 119 e 120.

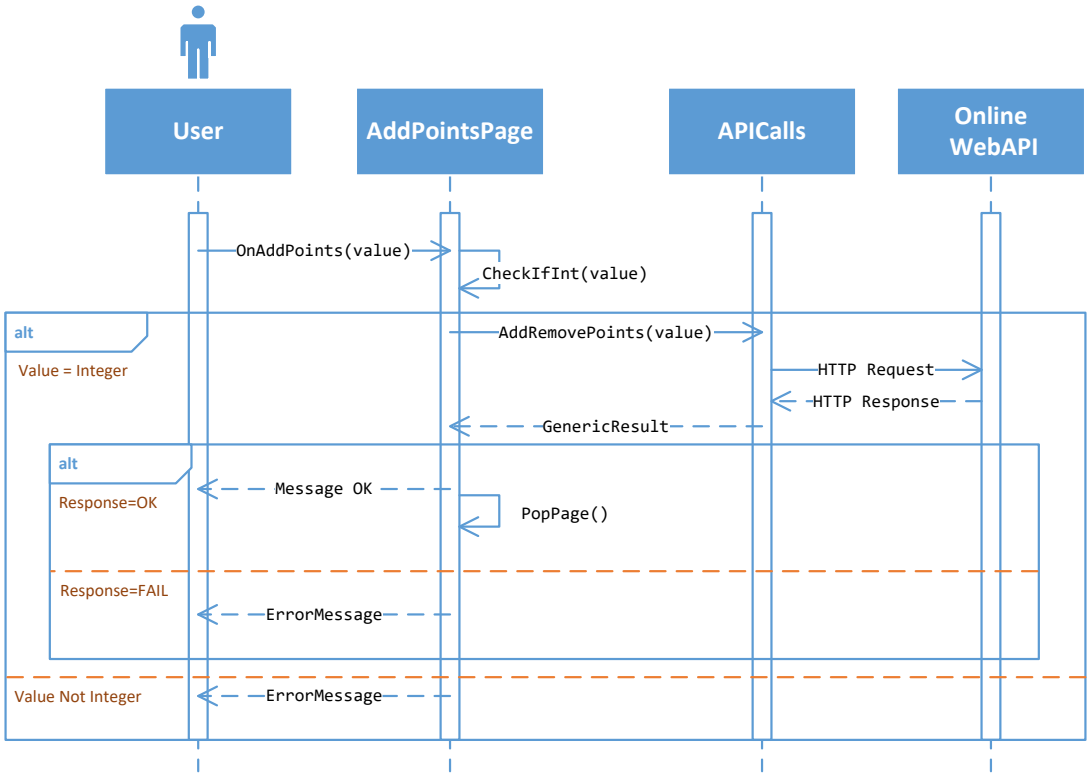


Figura 119 - Diagrama de Sequência Adição de Pontos a Utilizador

O diagrama de subtração de pontos da conta do utilizador tem algumas diferenças quando comparado com a adição. Neste caso torna-se necessário validar se existem pontos suficientes a serem utilizados e apena em caso afirmativo é que poderá ser processado o pedido de remoção de pontos.

Uma vez que está previsto utilizar a mesma chamada à API para adição/remoção de pontos é imperativo converter o valor de pontos inseridos em negativo isto porque foi convencionado que adições de valores negativos correspondem a remoções de pontos para simplificar o histórico de operações.

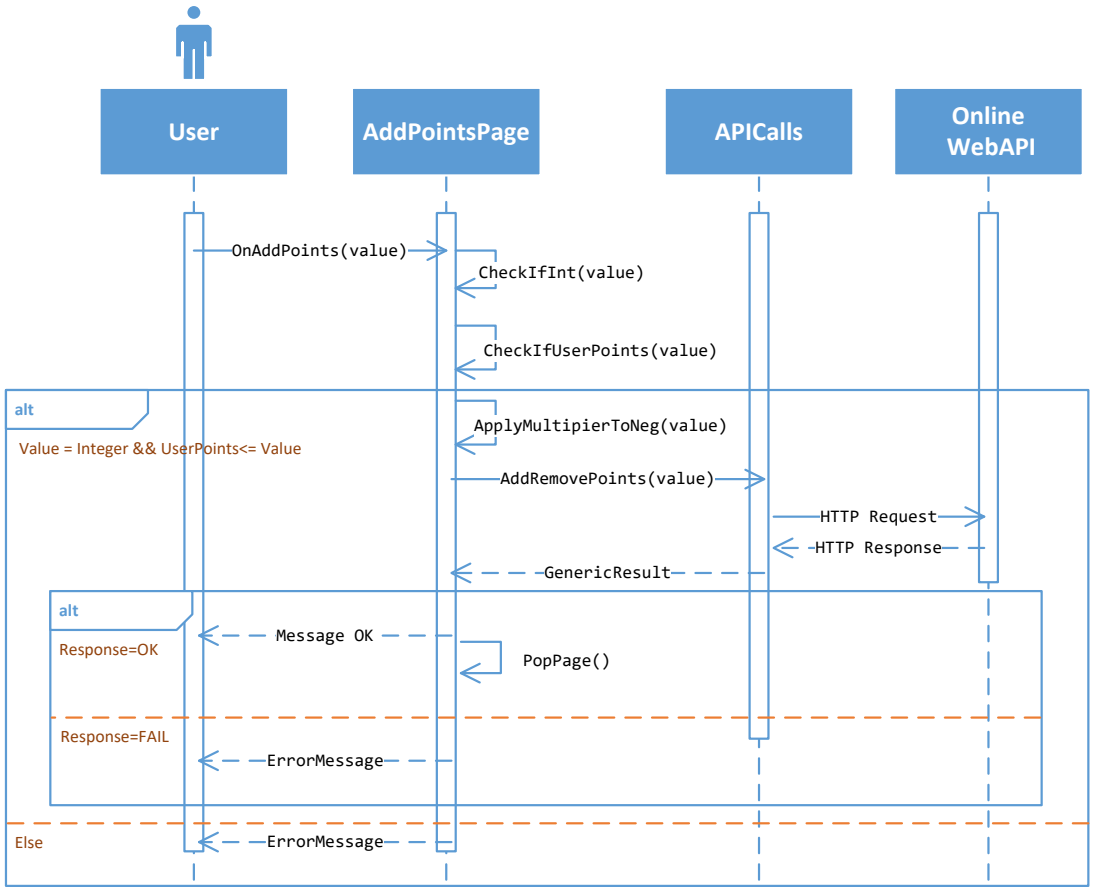


Figura 120 - Diagrama de Sequência Adição de Pontos a Utilizador

4.4.2 Desenvolvimento

No trabalho de desenvolvimento desta aplicação utilizou-se a mesma metodologia que para a aplicação móvel par utilizadores organizando o projeto por pastas que referenciam as várias camadas Model, ViewModel, Services e Views.

Como pode ser observado na Figura 121 esta aplicação é muito mais simples que a anteriormente descrita possuindo apenas 5 Views correspondentes aos diversos ecrãs necessários para operação sendo a View de Adição de pontos particamente igual à subtração.

Figura 121 - Screenshots da aplicação

Nesta estrutura foram criados alguns modelos que permitirão auxiliar a passagem de dados entre as Views e ViewModels bem como no envio e receção de informação junto da API. Não foi utilizada qualquer componente de persistência de dados locais e não houve necessidade de sobrepor código nos projetos das plataformas IOS, Android e UWP. Foi somente necessário incorporar via NuGet a biblioteca Newtonsoft.Json para auxiliar na conversão de dados de e para formato JSON.

Destacando as componentes de código mais relevantes para esta aplicação temos o processo de pesquisa de utilizador que foi implementado no evento de clique (Figura 123) no botão de submissão na *MainPage* (Figura 122).

```
Button SearchUserButton = new Button
{
    Text = "Pesquisar Por Utilizador",
};
SearchUserButton.Clicked += OnSearchUserButtonClick;
SearchUserButton.SetBinding(Button.IsEnabledProperty, "NotBusy");
```

Figura 122 - Excerto de código de implementaçãode botão SearchUser

```
private async void OnSearchUserButtonClick(object sender, EventArgs e)
{
    viewmodel.IsBusy = true;
    var response = await App.API.SearchUser(Auth.Token, viewmodel.Query);
    if (response != null && response.Result.Equals("OK"))
    {
        if (!response.Linked)//utilizador não está associado
        {
            var answer = await DisplayAlert("Questão?", "O utilizador não está associado à sua empresa.\nDeseja associar?", "Sim", "Não");

            if (answer==true)
            {
                var resp = await App.API.AssociateUserToPME(Auth.Token,response.UserID);
               if (resp.Result.Equals("OK"))
                {
                    await DisplayAlert("PME App", "Utilizador Associado com Sucesso!", "OK");
                    await Navigation.PushModalAsync(new Views.UserDetailPage(Auth,response));
                }
                else { await DisplayAlert("PME App", "Erro ao Associar Utilizador.\nPor Favor tente mais tarde!", "OK"); }
            }
        }else //utilizador está associado
        {
            await Navigation.PushModalAsync(new Views.UserDetailPage(Auth, response));
        }
    }
    else await DisplayAlert("PME App", "Utilizador Não Encontrado", "OK");
    viewmodel.IsBusy = false;
}
```

Figura 123 - Código associado ao evento clique no botão de pesquisa de utilizador

Endereçando o diagrama de sequência planeado este bloco de código permite obter através da API informação de um utilizador através de um critério de pesquisa (neste caso o número de telefone) e, caso o utilizador seja encontrado, mas não esteja associado à empresa, é da a possibilidade de proceder-se a essa associação (Figura 124)

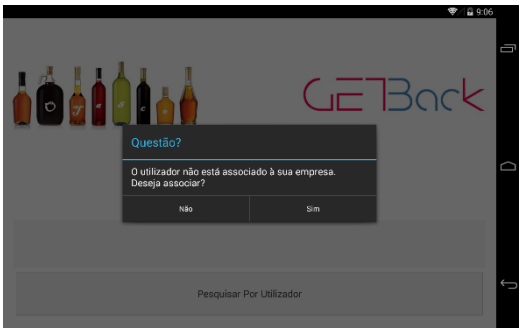


Figura 124 - Detalhe de Alerta para associação de utilizador

Para as componentes de adição (Figura 125) e subtração (Figura 126) de pontos foram desenvolvidos os métodos que processam os pedidos referentes a estas operações.

```
private async void OnSubmitButtonClick(object sender, EventArgs e)
{
    if (Int32.Parse(addPointsEntry.Text )% 2 == 0)
    {
        var resp = await App.API.AddRemovePointsToUser(Auth.Token, addPointsEntry.Text, User.UserID);
        if (resp.Result.Equals("OK"))
        {
            await DisplayAlert("PME App", addPointsEntry.Text+" Pontos Adicionados com Sucesso!", "OK");
            await Navigation.PopModalAsync();
        }
        else
        {
            await DisplayAlert("PME App", "Erro ao Adicionar Pontos.\nPor Favor tente mais tarde!", "OK");
        }
    }else
    {
        await DisplayAlert("PME App", "Erro! Deve inserir um valor inteiro!", "OK");
    }
}
```

Figura 125 - Método para Adição de pontos na conta do utilizador

No caso subtração, tal como planeado, é necessário proceder a mais validações e à conversão par número negativo do valor a subtrair.

```
private async void OnSubmitButtonClick(object sender, EventArgs e)
{
    if (Int32.Parse(subPointsEntry.Text) % 2 == 0 && User.Points > Int32.Parse(subPointsEntry.Text))
    {
        var aux = Int32.Parse(subPointsEntry.Text) * -1;
        var resp = await App.API.AddRemovePointsToUser(Auth.Token, aux.ToString(), User.UserID);

        if (resp.Result.Equals("OK"))
        {
            await DisplayAlert("PME App", subPointsEntry.Text+" pontos subtraidos com Sucesso!", "OK");
            await Navigation.PopModalAsync();
        }
        else
        {
            await DisplayAlert("PME App", "Erro ao Subtrair Pontos.\nPor Favor tente mais tarde!", "OK");
        }
    }
    else
    {
        await DisplayAlert("PME App", "Erro! Verifique a quantidade de pontos e volte a tentar novamente!", "OK");
    }
}
```

Figura 126 - Método para Subtração de pontos na conta do utilizador

De uma forma muito semelhante à já referida para a aplicação móvel de utilizadores foi criada uma classe de serviços *APICalls* utilizando o padrão *Singleton* e onde forma incorporados todos os métodos de acesso à API (Figura 127).

```
public class APICalls
{
    private static APICalls instance = new APICalls();
    private String uri = "http://advft-ws.azurewebsites.net/";
    private HttpClient client = new HttpClient();

    1 reference | 0 changes | 0 authors, 0 changes
    private APICalls() { }

    1 reference | 0 changes | 0 authors, 0 changes
    public static APICalls getInstance()
    {
        return instance;
    }

    //****  Metodo para autenticar PME na API.
    0 references | 0 changes | 0 authors, 0 changes
    public async Task<Model.PMEAuth> Authenticate(String Login, String Password)
    {
        var content = "{\"Login\": \"" + Login + "\" ,\"Password\": \"" + Password + "\" }";
        HttpContent contentPost = new StringContent(content, Encoding.UTF8, "application/json");
        HttpResponseMessage TokenRequest = await client.PostAsync(uri + "/api/pme", contentPost);

        if (TokenRequest.IsSuccessStatusCode) //Se recebe resposta da API
        {
            return  JsonConvert.DeserializeObject<Model.PMEAuth>(TokenRequest.Content.ReadAsStringAsync().Result);
        }
        else return null;
    }
    //****  Metodo para pesquisar utilizador PME na API.
    0 references | 0 changes | 0 authors, 0 changes
    public async Task<Model.UserInfo> SearchUser(String token, String querystring)
    {
        var content = "{\"Token\": \"" + token + "\",\"User\":\""+ querystring +"\"}";
        HttpContent contentPost = new StringContent(content, Encoding.UTF8, "application/json");
        var PMEsRequest = await client.PostAsync(uri + "/api/pme/user", contentPost);
        return JsonConvert.DeserializeObject<Model.UserInfo>(PMEsRequest.Content.ReadAsStringAsync().Result);
    }
    //****  Registar Utilizador {id} na PME.
    0 references | 0 changes | 0 authors, 0 changes
    public async Task<Model.GenericResult> AssociateUserToPME(String token, Int32 userID)
    {
        var content = "{\"Token\": \"" + token + "\"}";
        HttpContent contentPost = new StringContent(content, Encoding.UTF8, "application/json");
        var PMEsRequest = await client.PostAsync(uri + "/api/pme/register/"+userID.ToString(), contentPost);
        return JsonConvert.DeserializeObject<Model.GenericResult>(PMEsRequest.Content.ReadAsStringAsync().Result);
    }

    //****  Desassociar Utilizador {id} na PME  .
    0 references | 0 changes | 0 authors, 0 changes
    public async Task<Model.GenericResult> RemoveUserAssocitationToPME(String token, Int32 userID)
    {
        var content = "{\"Token\": \"" + token + "\"}";
        HttpContent contentPost = new StringContent(content, Encoding.UTF8, "application/json");
        var PMEsRequest = await client.PostAsync(uri + "/api/pme/unregister/" + userID.ToString(), contentPost);
        return JsonConvert.DeserializeObject<Model.GenericResult>(PMEsRequest.Content.ReadAsStringAsync().Result);
    }

    //****  Metodo para Adicionar/Remover pontos a User na API. (Remover = adicionar a negativo)
    0 references | 0 changes | 0 authors, 0 changes
    public async Task<Model.GenericResult> AddRemovePointsToUser(String token, String points,Int32 userid)
    {
        var content = "{\"Token\": \"" + token + "\",\"Value\":\"" + points + "\"}";
        HttpContent contentPost = new StringContent(content, Encoding.UTF8, "application/json");
        var PMEsRequest = await client.PutAsync(uri + "/api/pme/points/"+userid.ToString(), contentPost);
        return JsonConvert.DeserializeObject<Model.GenericResult>(PMEsRequest.Content.ReadAsStringAsync().Result);
    }
}
```

Figura 127 - Classe APICalls.

4.4.3 Testes

Durante o desenvolvimento da aplicação foram sendo feitos alguns testes de utilização em ambiente Android e Windows UWP

Tal como na aplicação móvel para utilizadores mantiveram-se as limitações técnicas ao nível de equipamentos IOS não tendo feitos testes de compilação e utilização em IOS.

Na API criada para testar a aplicação móvel foram incluídos Controllers para responderam aos pedidos desta nova aplicação móvel tendo sido executados testes contra esta plataforma.

Na vertente de navegação foram testadas todas as funcionalidades desenvolvidas utilizando emuladores disponibilizados sobre Hyper-V:

- **7" KitKat (4.4) XHDPI PTablet (Android 4.4 – API19):** correspondendo a um dispositivo Tablet Android Kitkat, ecrã de 7" e definição de 640X960 e configurado com acesso à internet (Figura 128).

Figura 128 - Emulador Android 7"

- **LocalMachine**: para Windows 10 (Figura 129) foi utlizado o próprio laptop onde foi desenvolvido o projeto correndo a aplicação e usufruindo dos recursos da própria máquina física (nomeadamente acesso á internet)

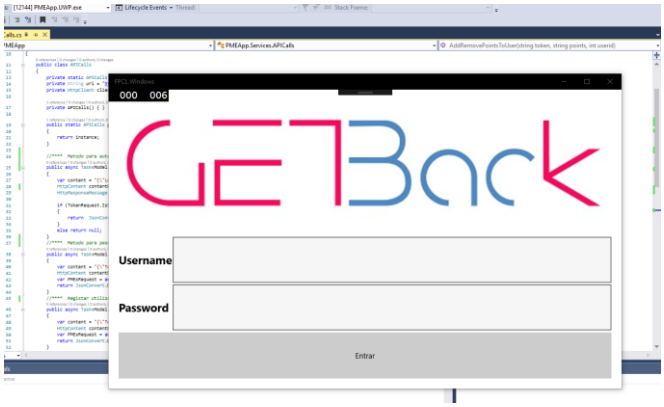


Figura 129 - Aplicação a ser executada sobre Windows 10

Os testes efetuados serviram para validar a passagem de dados entre Views (Figuras 130 e 131), bem como conferir as diversas variantes possíveis na pesquisa de utilizador, associação/desassociação de utilizador na empresa e adição/remoção de pontos.

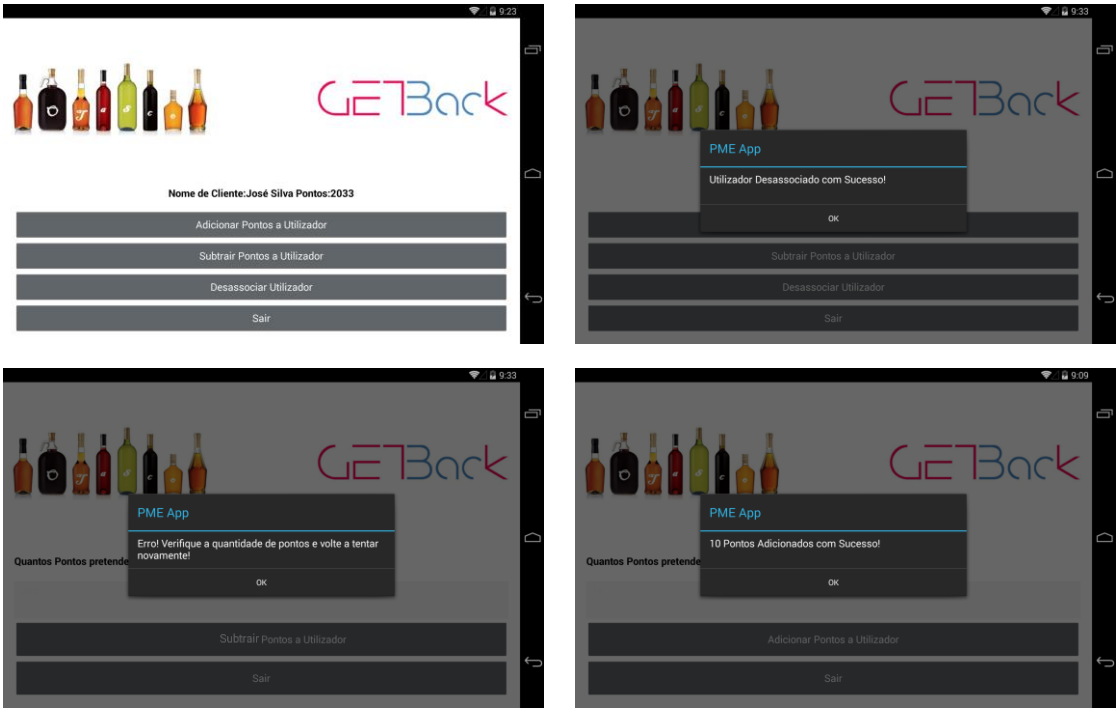


Figura 130 - Ecrãs da aplicação sobre Android

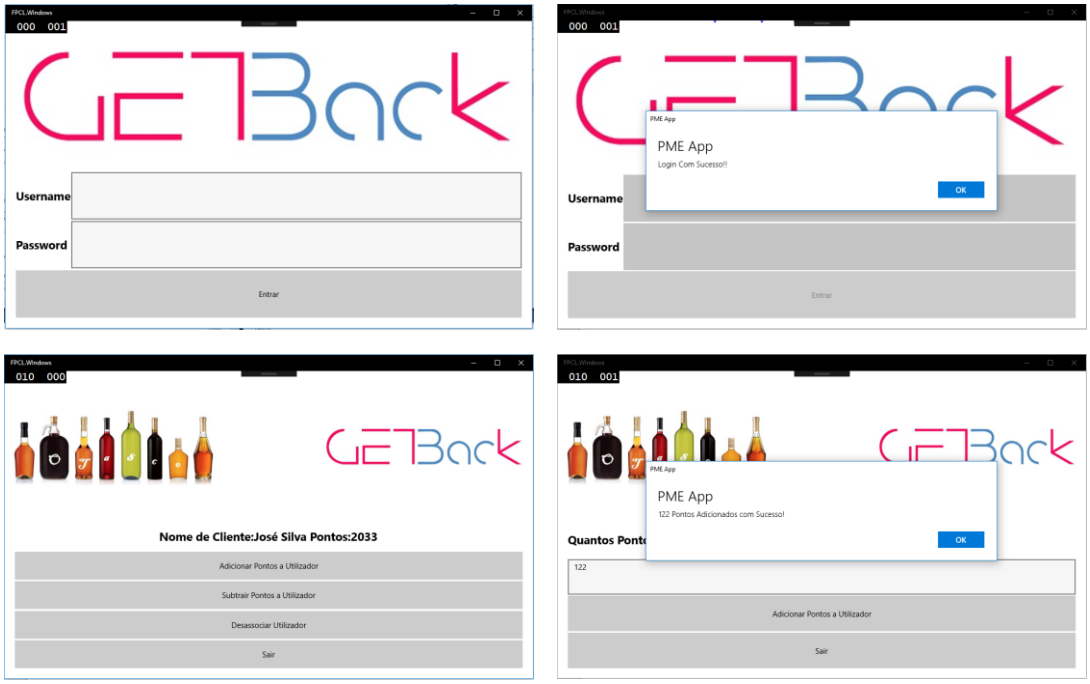


Figura 131 – Ecrãs da aplicação sobre Android

5 Conclusão

Neste projeto pretendia-se contruir uma plataforma que respondesse a um conjunto de funcionalidades idealizadas. Foi efetuada uma analise cuidada quer a soluções já existentes tendo em vista identificar pontos fortes e fracos dessas soluções. Em simultâneo, foram desenvolvidos inquéritos para recolher opiniões e necessidades tanto de utilizadores como de empresas. Estes dados permitiram definir claramente requisitos imperativos à nossa aplicação bem como explorar novas funcionalidades para diferenciar esta solução.

Com base em toda a informação recolhida, foi efetuado todo o planeamento da solução quer a nível negócio como a nível conceptual. A nível de negócio foram encontradas as abordagens que julgamos ser mais convenientes para promover a solução. A nível conceptual, foi efetuada uma analise tecnológica para escolher quais as tecnologias que melhor se enquadravam quer tecnicamente quer financeiramente.

Na componente de desenvolvimento foram construídas as diversas componentes que compõe a plataforma proposta, respeitando os requisitos e planeamento definidos, sendo utilizada a metodologia *agile* para reformular, sempre que necessário, o desenho da solução.

Tratando-se se uma tese elaborada por dois elementos, todas as fases de analise, estudo e planificação geral foram efetuadas em conjunto. Houve uma divisão de tarefas, por área, no desenvolvimento. A lista completa de tarefas efetuadas por cada elemento pode ser consultada no documento em anexo.

Encarando este projeto como um protótipo ou prova de conceito, podemos concluir que as ideias iniciais foram corroboradas pela plataforma desenvolvida, havendo viabilidade para a implementação comercial. Contudo, foram também identificados ao longo do desenvolvimento do projeto pontos a serem melhorados com a evolução do projeto. Por exemplo a uniformização a nível de design de toda a plataforma e definição dos produtos a serem comercializados.

6 Bibliografia

Adobe Systems Inc., 2016. *Phonegap.* [Online]
Available at: http://phonegap.com/
[Acesso em 15 10 2016].

Clarity Ventures, Inc., 2016. *Xamarin Vs. Titanium Vs. PhoneGap Vs. Cordova: A Comparison.* [Online]
Available at: https://www.clarity-ventures.com/resources/xamarin/xamarin-vs-titanium-vs-phonegap-vs-cordova-a-comparison
[Acesso em 20 10 2016].

The jQuery Foundation, 2016. *JQuery.* [Online]
Available at: https://jquery.com/
[Acesso em 20 10 2016].

Appcelerator Inc. , 2016. *Appcelerator: Mobile App Development Platform & MBaaS.* [Online]
Available at: http://www.appcelerator.com/
[Acesso em 18 10 2016].

ASP.NET core, 2016. *ASP.NET core.* [Online]
Available at: https://www.asp.net/core
[Acesso em 20 10 2016].

ASP.NET MVC, 2016. *ASP.NET MVC.* [Online]
Available at: https://www.asp.net/mvc
[Acesso em 20 10 2016].

Autofac, 2013. *Autofac.* [Online]
Available at: https://autofac.org/
[Acesso em 20 10 2016].

Cardmobili.SA, s.d. [Online]
Available at: http://corporate.cardmobili.com/en/homepage
[Acesso em 06 10 2016].

Clerck, J. D., 2015. *http://www.i-scoop.eu/.* [Online]
Available at: http://www.i-scoop.eu/customer-centricity/
[Acesso em 22 01 2016].

Demoulin, N. T. & Zidda, P., 2008. On the Impact of Loyalty Cards on Store Loyalty: Does the Customers' Satisfaction with the Reward Scheme Matter?. *Journal of Retailing and Consumer Services.*

FACEBOOK, s.d. [Online]
Available at: htp://www.facebook.com
[Acesso em 07 10 2016].

Google , 2016. *Google.* [Online]
Available at: http://www.google.com
[Acesso em 20 10 2016].

Google, 2016. *Chrome V8.* [Online]
Available at: https://developers.google.com/v8/
[Acesso em 20 10 2016].

Google, 2016. *The best of Google Maps for every Android app.* [Online]
Available at: https://developers.google.com/maps/documentation/android/
[Acesso em 20 10 2016].

H3, 2016. *H3.* [Online]
Available at: HTTP://H3.COM
[Acesso em 06 10 2016].

ilyaigpetrov, 2007. *Comparison of ASP.NET and Node.js for Backend Programming.* [Online]
Available at: https://gist.github.com/ilyaigpetrov/f6df3e6f825ae1b5c7e2
[Acesso em 20 10 2016].

Inter IKEA Systems B.V., 2016. *IKEA FAMILY.* [Online]
Available at: http://www.ikea.com/ms/pt_PT/family/
[Acesso em 07 10 2016].

IOLA & Laursen, O., 2007. *Resultados da procura.* [Online]
Available at: http://www.flotcharts.org/
[Acesso em 20 10 2016].

jamesmontemagno, 2016. *Xamarin.Plugins.* [Online]
Available at: https://github.com/jamesmontemagno/xamarin.plugins
[Acesso em 20 10 2016].

La Fourchette , s.d. *La Fourchette.* [Online]
Available at: https://www.lafourchette.com/
[Acesso em 20 10 2016].

La FourDhette, SAS, 2016. *TheFork.pt - TheFork - Porto.* [Online]
Available at: https://www.thefork.pt/cidade/porto
[Acesso em 06 10 2016].

Maria Auxiliadora Cannarozzo Tinoco, s.d. *A new approach for modeling client satisfaction determinants relationships in services.* [Online]
Available at: http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0103-65132007000300005
[Acesso em 01 10 2016].

Martin, R. L., 2010. *The Age of Customer Capitalism.* [Online]
Available at: https://hbr.org/2010/01/the-age-of-customer-capitalism
[Acesso em 20 February 2016].

mdo & fat, 2016. *Bootstrap.* [Online]
Available at: http://getbootstrap.com/
[Acesso em 20 10 2016].

Mejia, A., 2016. *Creating RESTful APIs with NodeJS.* [Online]
Available at: http://adrianmejia.com/blog/2014/10/01/creating-a-restful-api-tutorial-with-nodejs-and-mongodb/
[Acesso em 21 10 2016].

Meyer-Waarden, L., 2008. The influence of loyalty programme membership on customer purchase behaviour. *European Journal of Marketing*.

Microsoft, 2012. *Implementing the Repository and Unit of Work Patterns in an ASP.NET MVC.* [Online]
Available at: https://www.asp.net/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application
[Acesso em 21 10 2016].

Microsoft, 2015. *Entity Framework - MSDN - Microsoft.* [Online]
Available at: https://msdn.microsoft.com/en-us/data/ef(v=vs.113).aspx
[Acesso em 20 10 2016].

Microsoft, 2016. *ASP.NET Web API | The ASP.NET Site.* [Online]
Available at: https://www.asp.net/web-api
[Acesso em 20 10 2016].

Microsoft, 2016. *Bing Maps.* [Online]
Available at: https://www.bingmapsportal.com/
[Acesso em 20 10 2016].

Microsoft, 2016. *Choose a cloud SQL Server option: Azure SQL (PaaS) Database or SQL Server on Azure VMs (IaaS).* [Online]
Available at: https://azure.microsoft.com/en-us/documentation/articles/sql-database-paas-vs-sql-server-iaas/
[Acesso em 20 10 2016].

Microsoft, 2016. *Home - BizSpark - Microsoft.* [Online]
Available at: https://bizspark.microsoft.com//
[Acesso em 20 10 2016].

Microsoft, 2016. *Microsoft.* [Online]
Available at: https://www.microsoft.com/pt-pt/
[Acesso em 20 10 2016].

Microsoft, 2016. *Microsoft Azure: Plataforma de Informática em Nuvem e Serviços.* [Online]
Available at: https://azure.microsoft.com/pt-pt/
[Acesso em 20 10 2016].

Microsoft, 2016. *SQL Database – Cloud Database as a Service | Microsoft Azure.* [Online]
Available at: https://azure.microsoft.com/en-us/services/sql-database/
[Acesso em 20 10 2016].

Microsoft, 2016. *SQL SERVER 2016.* [Online]
Available at: https://www.microsoft.com/pt-pt/server-cloud/products/sql-server/overview.aspx
[Acesso em 20 10 2016].

Microsoft, 2016. *Visual Studio IDE.* [Online]
Available at: https://www.visualstudio.com/vs/
[Acesso em 20 10 2016].

Microsot, 2016. *ASP.NET Identity.* [Online]
Available at: https://www.asp.net/identity
[Acesso em 20 10 2016].

Mikael, 2012. *ASP.NET Web Api vs Node.js Benchmark.* [Online]
Available at: http://mikaelkoskinen.net/post/asp-net-web-api-vs-node-js-benchmark
[Acesso em 20 10 2016].

MODELO CONTINENTE HIPERMERCADOS, S.A., 2015. *MODELO CONTINENTE HIPERMERCADOS, S.A..* [Online]
Available at: htttp://continente.pt
[Acesso em 08 10 2016].

Newtonsoft, 2016. *Json.NET.* [Online]
Available at: http://www.newtonsoft.com/json
[Acesso em 20 10 2016].

Node.js Foundation, 2016 . *Node.js.* [Online]
Available at: https://nodejs.org/en/
[Acesso em 20 10 2016].

OAuth0, 2016. *OAuth 2.0.* [Online]
Available at: https://oauth.net/
[Acesso em 20 10 2016].

Oracle Corporation , 2016. *MySQL.* [Online]
Available at: http://www.mysql.com/
[Acesso em 20 10 2016].

Osterwalder, A., 2004. *The business model ontology: A proposition in a design science approach,* Lausanne, Switzerland: s.n.

Perfumes e Companhia, 2016. *Perfumes e Companhia.* [Online]
Available at: https://www.perfumesecompanhia.pt/pt/
[Acesso em 06 10 2016].

Perfumes e companhia, s.d. *PC like me.* [Online]
Available at: https://www.perfumesecompanhia.pt/pt/like-me/

Pesce, B., 2012. *A menina do vale - Como o empreendedorismo pode mudar a sua vida.* [Online]
Available at: http://www.ameninadovale.com
[Acesso em 20 February 2016].

Postman, 2016. *Modern software is built on APIs.* [Online]
Available at: https://www.getpostman.com/
[Acesso em 10 10 2016].

Redd, S. M., 2014. *Entity Framework: It's not a stack of pancakes!.* [Online]
Available at: http://www.reddnet.net/entity-framework-its-not-a-stack-of-pancakes/
[Acesso em 21 10 2016].

Redth, 2016. *ZXing.Net.Mobile.* [Online]
Available at: https://github.com/Redth/ZXing.Net.Mobile
[Acesso em 20 10 2016].

Santos, A. & Mendonça, F., 2016. *Formulário sobre cartões de fidelização clientes.* [Online]
Available at: https://goo.gl/VK2Vpv .
[Acesso em 21 10 2016].

Santos, A. & Mendonça, F., 2016. *Questionário para PME.* [Online]
Available at: https://goo.gl/OrqSBB
[Acesso em 20 10 2016].

SIBS, 2015. *MB WAY.* [Online]
Available at: http://www.mbway.pt
[Acesso em 20 10 2016].

SQLite, 2016. *SQLite.* [Online]
Available at: https://sqlite.org/
[Acesso em 20 10 2016].

Stepp, M. & Miller, J., 2009. *PHP for Server-Side Programming.* [Online]
Available at: http://www.webstepbook.com/supplements/slides/ch05-php.shtml
[Acesso em 20 10 2016].

StrongLoop, 2016. *Express - Node.js web application framework.* [Online]
Available at: http://expressjs.com/
[Acesso em 20 10 2016].

Swader, 2015. *Most popular php framrwork at work.* [Online]
Available at: https://plot.ly/~swader/8/most-popular-php-framework-at-work-sitepoint-2015/
[Acesso em 20 10 2016].

The Apache Software Foundation, 2015. *Apache Cordova.* [Online]
Available at: https://cordova.apache.org/
[Acesso em 10 10 2016].

The Linux Foundation, 2016. *The Linux Foundation.* [Online]
Available at: https://www.linuxfoundation.org/
[Acesso em 20 10 2016].

The PHP Group, 2016. *PHP: Hypertext Preprocessor.* [Online]
Available at: https://secure.php.net/
[Acesso em 20 10 2016].

Twitter, 2016. *Twitter.* [Online]
Available at: http://www.twitter.com
[Acesso em 10 10 2016].

Wanderley, J. A., s.d. *Negociação Tota.* s.l.:Gente.

Wasson, M., 2014. *Secure a Web API with Individual Accounts and Local Login in ASP.NET Web API 2.2.* [Online]
Available at: https://www.asp.net/web-api/overview/security/individual-accounts-in-web-api
[Acesso em 15 09 2016].

Wikipedia, 2016. *Análise do valor.* [Online]
Available at: https://pt.wikipedia.org/wiki/An%C3%A1lise_do_valor
[Acesso em 18 02 2016].

Wolfgang , U. & Andreas , E., 2006. Value-Based Differentiation in Business Relationships: Gaining and Sustaining Key Supplier Status. *Journal of Marketing*, January, pp. 119-136.

Xamarin Inc., 2016. *Xamarin: Mobile Application Development to Build Apps in C#.* [Online]
Available at: https://www.xamarin.com/
[Acesso em 20 10 2016].

Yi, Y. & Jeon, H., 2003. Effects of Loyalty Programs on Value Perception, Program Loyalty, and Brand Loyalty. *Journal of the Academy of Marketing.*

YLoyalty , 2016. *YLoyalty - O seu programa de fidelização - YLoyalty.* [Online]
Available at: https://www.yloyalty.com/pt/
[Acesso em 06 10 2016].

Divisão de Trabalho

Capítulo	Adão Santos	Fábio Mendonça
1. Introdução	✔	✔
2. Contexto		
2.1 Estado da Arte	✔	✔
2.2 Detalhes sobre o Contexto e Problema	✔	✔
2.3 Análise de Valor	✔	✔
3. Design da Solução		
3.1 Design Conceptual	✔	✔
3.2 Funcionalidades do Sistema		
3.2.1 Portal	✔	
3.2.2 Aplicação Móvel para Utilizadores		✔
3.2.2 Aplicação Móvel para Empresas		✔
3.3 Técnologia Relevante	✔	✔
3.4 Arquitetura	✔	✔
4. Desenvolvimento		
4.1 Portal	✔	
4.2 API	✔	✔
4.3 Aplicação Móvel para Utilizadores		✔
4.4 Aplicação Móvel para Empresas		✔
5. Conclusão	✔	✔

MODELO DE NEGÓCIO

<table>
<tr>
<td rowspan="2">Key Partners
Empresas</td>
<td>Key Activities
Plataforma de fidelização</td>
<td rowspan="2">Value Propositions
Fidelizar clientes

Aumentar vendas</td>
<td>Customer Relationships
Internet

Smartphone</td>
<td rowspan="2">Customer Segments
Empresas de comércio/serviços

Utilizadores de smartphone</td>
</tr>
<tr>
<td>Key Resources
Software</td>
<td>Channels
Portal

Smartphone

API</td>
</tr>
<tr>
<td colspan="3">Cost Structure

Custo desenvolvimento

Manutenção da plataforma</td>
<td colspan="2">Revenue Streams
Avança mensal

Publicidade</td>
</tr>
</table>

API

Verbo	URL	Descrição	Body Envio	Response OK	Response Fail
POST	/api/user	Permite a autenticação de um utilizador junto da plataforma.	{Login: "String", Password: "String"}	{Result:"OK" Token:"String"}	{Result: "FAIL"}
GET	/api/user/{token}	Recebe informação de um utilizador já autenticado	-	{Result:"OK", Name:"String", Address:"String", City:"String", PostalCode:"String", Phone:"String", Email:"String" }	{Result: "FAIL"}
PUT	/api/user/	Altera dados de utilizador ou inser e utilizador na plataforma. Nota: É validado pela API se existe utilizador com o mail/phone enviados.	{ Token:"TOKEN", Name:"String", Address:"String", City:"String", PostalCode:"String", Phone:"String", Email:"String", Pasword:"String" }	{Result:"OK" }	{Result: "FAIL"}
GET	/api/user/activate/hashkey	Ativa utilizador. Isto será utilizado através do email enviado para o utilizador para ativar a sua conta.	-	{Result:"OK"}	{Result: "FAIL"}
POST	/api/user/register/{idPME}	Associa User à PME	{Token:"String"}	{Result:"OK"}	{Result: "FAIL"}
POST	/api/user/unregister/{idPME}	Desassocia User à PME	{Token:"String"}	{Result:"OK"}	{Result: "FAIL"}

POST	/api/user/companies	Solicta e recebe lista de Emoresas às quais o utilizador autenticado está associado.	{Token="String"}	{Result:"OK", TotalResults:"Int32", Results:[{ID:"Int32", Title:"String", Logo:"String", Registered:"Boolean", IsOpen:"Boolean", Points:"Int232", Rating: "Double", LastVisit:"DateTime" }]}	{Result: "FAIL"}
POST	/api/user/reviews	Solicta e recebe Reviews pendentes de preenchimento associados ao utilizador	{Token:"String"}	{Result:"OK", Results:[{ReviewID:"Int32", CategoryID:"Int32", CompanyName:"String", VisitDate:"DateTime"},{ ReviewID:"Int32", CategoryID:"Int32", CompanyName:"String", VisitDate:"DateTime"}]}	{Result: "FAIL"}
POST	/api/user/reviews/{ReviewID}	Submete a resposta a um Review	{Token:"String", Review: {Q1:"Int32", Q2:"Int32", Q3:"Int32", Q4:"Int32", Q5:"Int32", Opinion:"String" }}	{Result:"OK"}	{Result: "FAIL"}

GET	/api/category	Retorna lista de todas as categorias	-	{ "Result": "OK", "Results": [{ "ID": "Int32", "IDP": "Int32", "Title": "String", "Icon": "String" }]}	{Result: "FAIL"}

GET	/api/category/{id]	Solicta e recebe Questões associadas a uma determinada categoria.		{Result:"OK",
				Results:{ "Q1":{
				QText:"String",
				QIcon:"String"
				},"Q2":{
				QText:"String",
				QIcon:"String"
				},"Q3":{
				QText:"String",
				QIcon:"String"
				},"Q4":{
				QText:"String",
				QIcon:"String"
					{Result: "FAIL"}

POST	/api/search	Solicta e recebe resultados de Pesquisa	{Token:"String", Category:"Int32", TextSearch:"String", Radius:"Double", myLat:"Double", myLong:"Double", Sorting:"String", CurrentPage:"Int32"}	{Result:"OK", TotalResults:"Int32", Results:[{ID:"Int32", Title:"String", Logo:"String", Long:"Double", Lat:"Double", Distance:"Double", Registered:"Boolean", IsOpen:"Boolean", Rating:"Double"}]}	{Result: "FAIL"}

POST	/api/pme/{id}	Solicta e recebe detalhes de uma empresa	{Token:"String"}	{Result:"OK", ID:"Int32", Title:"String", Description:"String", Logo:"String", Phone:"String", Mobile:"String", Address:"String", PostalCode:"String", City:"String", Long:"Double", Lat:"Double", Registered:"Boolean", Points:"Int32", IsOpen:"Boolean", Rating:"Double", Lastvisited:"DateTime"}	{Result: "FAIL"}

GET	/api/pme/{id}/images	Retorna lista de imagens da Empresa	-	{
				"Result": "OK",
				"Results": [
				"Image1String",
				"Image2String"
				]
				}	

GET	/api/pme/{id}/reviews	Solicta e recebe Lista de reviews (randoms) de uma determinada empresa	-	{
				"Result": "OK",
				"Results": [{Questions:
				[
				{
				QIcon:"String",
				QValue:"Int32",
				QText:"String"
				}, {
				QIcon:"String",
				QValue:"Int32",
				QText:"String"
				}, {
				QIcon:"String",
				QValue:"Int32",
				QText:"String"
				}, {
				QIcon:"String",
				QValue:"Int32",
				QText:"String"
				}, {
				QIcon:"String",
				QValue:"Int32",
				QText:"String"
				}
				],Opinion:"String"}]
				}	

POST	/api/pme	Autentica PME	{Login: "String", Password: "String"}	{Result:"OK" Token:"String", Logo:"Stirng"}	{Result: "FAIL"}

POST	/api/pme/user	Pesquisa por utilizador	{Token:"String", User:"String TELEF"}	{Result:"OK", userID = "int32", username:"string, Ponits:"value", linked="trie/false}	{Result: "FAIL"}
POST	/api/pme/register/{iduser}	Associa utiluzador à empresa autenticada	{Token:"String"}	{Result:"OK"}	{Result: "FAIL"}
PUT	/api/pme/points/{iduser}	Adiciona/Removo pontos de utlizador na empresa autenticada	{Token:"String",Value:"Int32"}	{Result:"OK"}	{Result: "FAIL"}
POST	/api/pme/unregister/{iduser}	Desassocia utilizador de uma da Empresa autenticada.	{Token:"String"}	{Result:"OK"}	{Result: "FAIL"}