Departamento de Eletrónica,
Universidade de Aveiro Telecomunicações e Informática
2018

Mariana da Silva Costa Pereira

Serviços de *Backend* para o Desenvolvimento de Aplicações Móveis

Mobile Backend as a Service

Universidade de Aveiro
2018

Departamento de Eletrónica,
Telecomunicações e Informática

Mariana da Silva Costa Pereira

Serviços de *Backend* para o Desenvolvimento de Aplicações Móveis

Mobile Backend as a Service

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Informática, realizada sob a orientação científica do Professor Doutor Ilídio Castro Oliveira, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, em contexto de estágio em empresa na Ubiwhere S.A, com a supervisão do Mestre João Pedro Pedrosa.

o júri

presidente

Professor Doutor José Luis Guimarães Oliveira
Professor Associado C/ Agregação, Universidade de Aveiro

vogais

Professor Doutor Ricardo Alexandre Peixoto Queirós
Professor Adjunto do Instituto Politécnico do Porto

Professor Doutor Ilídio Fernando de Castro Oliveira
Professor Auxiliar da Universidade de Aveiro

agradecimentos

Em primeiro lugar gostaria de agradecer à minha família, em especial ao meu pai, à minha mãe e à minha irmã por todo o apoio que me deram não só durante o desenvolvimento desta dissertação, como também ao longo de todo o meu percurso académico.

De seguida gostaria de agradecer ao meu orientador, Professor Doutor Ilídio Oliveira, por todo o apoio e disponibilidade manifestados ao longo da realização deste trabalho.

Gostaria também de agradecer à Ubiwhere, não só por me ter dado a oportunidade de realizar este projeto, mas também pelo acolhimento que me proporcionou e por me facultar todos os recursos necessários para a realização deste trabalho. Um agradecimento especial ao João Pedrosa e ao Hugo Fonseca, que se mostraram sempre disponíveis para me ajudar em qualquer dificuldade e que foram fundamentais para a conclusão deste trabalho.

Agradeço ainda a todos os meus colegas e amigos que me aconselharam e apoiaram ao longo destes anos, sem os quais teria sido difícil ultrapassar os desafios encontrados durante o meu percurso académico.

palavras-chave aplicações móveis, plataformas de servidor, Parse Server.

resumo

A crescente importância das aplicações móveis na oferta das empresas de software coloca desafios quanto à sua arquitetura. Neste contexto, é necessário garantir um nível de prontidão e sofisticação do desenvolvimento apoiado na utilização de plataformas de servidor (*backends)* para garantir um conjunto de serviços comuns, tais como a gestão de utilizadores, seguimento dos padrões de utilização ou distribuição de mensagens em larga escala.
Neste projeto de estágio, tomou-se como ponto de partida a solução de código aberto do Parse Server, que foi adaptado e configurado para poder ser usado como *backend* de referência na Ubiwhere, empresa de acolhimento deste trabalho.
A solução desenvolvida envolve a integração do Parse Server com o Parse Dashboard, bem como o desenvolvimento de novas funcionalidades para estes dois componentes e a integração dos módulos para o envio de notificações e para a automatização do processo de envio de *emails*.
Para validar as funcionalidades desenvolvidas e a correta integração dos componentes foram ainda integradas duas aplicações cliente.
Além disso, o processo de configuração e instalação dos componentes foi automatizado através da criação de *containers*.
A solução desenvolvida, que se encontra em produção, permite à empresa utilizar um *backend* próprio para o desenvolvimento de aplicações, especialmente para as plataformas móveis.

keywords mobile applications, server platforms, Parse Server.

abstract

The increasing importance of mobile applications in the portfolio of software companies puts challenges towards their architecture. In this context, the required levels of readiness and sophistication in the development is supported by the use of server platforms (backends) to ensure a set of common services, such as user management, monitoring of usage levels or the distribution of messages in large-scale.
In this internship project, the Parse Server Open source solution was selected as a starting point, then adapted and configured to be used as a reference backend at Ubiwhere, the host company of this work.
The developed solution included the integration of parse Server with the Parse Dashboard, the development of new features for these two components, and the integration of modules for sending notifications and for automating the process of sending emails.
Two client applications were integrated with the backend to validate the developed features and the correct integration of the components. In addition, the process of configuring and installing the components has been automated by using containers.
The developed solution, already used in production, allows the company to use its own backend instance for the development of applications, especially for mobile platforms.

CONTEÚDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

ACRÓNIMOS

CMS – Content Management System

IDE – Integrated Development Environment

SDK – Software Development Kit

UI – User Interface

API – Application Programming Interface

DOM – Document Object Model

MVC – Model View Controller

MBaaS – Mobile Backend as a Service

BaaS – Backend as a Service

CRUD – Create, Read, Update and Delete

BLOB – Binary Large Object

AWS – Amazon Web Services

RTOS – Real-Time Operating System

IoT – Internet of Things

MVP – Model View Presenter

URI – Uniform Resource Identifier

1 INTRODUÇÃO

Com o crescimento incremental da utilização das aplicações móveis, surge uma procura e uma exigência cada vez maiores no que diz respeito a este tipo de aplicações, o que obriga a que o seu desenvolvimento e distribuição seja agilizado [1]. Assim, estas aplicações tornaram-se progressivamente mais complexas, quer em termos de custos de desenvolvimento quer em relação ao número de funcionalidades que englobam, sendo necessário recorrer a ferramentas que permitam suportar essas mesmas funcionalidades.

Tendo isto em conta, para construir uma aplicação nos dias de hoje, é fundamental que exista um serviço de *backend* – que consiste na lógica de negócio da aplicação e no processamento e gestão de dados da mesma – que suporte as funcionalidades que a aplicação engloba [2]. No entanto, este tipo de serviços implica custos de desenvolvimento elevados, tanto a nível monetário como em tempo gasto para o desenvolvimento do serviço. Desta forma, cada vez mais são procuradas soluções que permitam dirigir o foco do desenvolvimento para o design da aplicação e para a *user experience* que a mesma proporciona, ou seja, soluções que aliviem o esforço depositado no desenvolvimento relativo ao *backend* da aplicação.

De forma a responder a esta necessidade, uma abordagem possível passa por ignorar o desenvolvimento do *backend* e por focar o esforço do desenvolvimento no *frontend* das aplicações, surgindo assim o conceito de *Mobile Backend as a Service* ou apenas *Backend as a Service.* Consequentemente, surgiu este projeto proposto pela Ubiwhere[1], a empresa na qual decorreu o estágio que conduziu à elaboração do projeto no contexto do Mestrado em Engenharia Informática.

1.1 Enquadramento e motivação para o projeto de estágio

Uma vez que as aplicações móveis estão cada vez mais presentes no dia-a-dia da maior parte da população, a introdução de novas aplicações no mercado torna-se relevante não só para os seus consumidores como também para quem as desenvolve, pois constituem um negócio para estes últimos. Desta forma, é relevante agilizar o desenvolvimento deste tipo de aplicações, pois permite que estas sejam introduzidas no mercado mais rapidamente e que atribuam uma vantagem competitiva ao negócio no qual se inserem.

Este trabalho foi proposto no contexto empresarial, por constituir uma alternativa *open-source* a outras plataformas semelhantes que se encontram em utilização, com vista a ser utilizado por aplicações já existentes ou por novas aplicações da empresa.

Pessoalmente considero que este é um projeto de grande interesse, visto que permite utilizar tecnologias atuais e trabalhar nas diferentes áreas que dizem respeito a um projeto. Para além disso, está relacionado com uma plataforma atualmente *open-source,* mas que já foi da responsabilidade do Facebook[2].

[1] https://www.ubiwhere.com/

[2] https://www.facebook.com/

1.2 Objetivos do trabalho

Com este projeto pretende-se tirar partido da plataforma Parse, agora *open source*, para que a mesma possa ser utilizada em aplicações já existentes ou no desenvolvimento de novas aplicações. Desta forma, este projeto pretende atingir os seguintes objetivos:

- Instalação de uma instância do Parse Server;
- Integração do Parse Server com o Parse Dashboard;
- Integração de módulos do Parse (como o Parse Mailgun Adapter);
- Implementação de protótipos funcionais que integrem um ou mais SDKs do Parse.

1.3 Sumário das contribuições

Com a realização deste trabalho, foi possível alcançar um conjunto de resultados que vão ao encontro dos objetivos estabelecidos para o mesmo.

Foi possível obter uma plataforma de *backend* funcional, com uma interface gráfica que facilita não só a perceção das operações que ocorrem no contexto desta plataforma, como também a interação com a mesma.

Para além disso foram integrados módulos do Parse com esta plataforma, relativos ao envio de notificações e de *emails* automáticos (Parse Mailgun Adapter).

No âmbito do Parse Dashboard, foram desenvolvidas novas funcionalidades, nomeadamente para a criação de novas aplicações, bem como para a visualização e gestão das *keys* das aplicações e dos parâmetros relativos ao envio de notificações e *emails* automáticos.

Foram ainda implementados protótipos funcionais, uma aplicação Android e uma aplicação *web*, que permitiram validar as funcionalidades desenvolvidas na plataforma e garantir que estas últimas se encontravam a funcionar corretamente.

Por último, o processo de criação das instâncias do Parse Server, Parse Dashboard e da base de dados, foi automatizado com a utilização do Docker[3].

[3] https://www.docker.com/

2 REVISÃO DE TECNOLOGIAS SELECIONADAS

No decorrer do presente projeto foram exploradas diversas tecnologias necessárias para a sua concretização e desenvolvimento.

No que diz respeito ao *backend*, as principais tecnologias aplicadas foram o Node.js, aplicado do lado do Parse Server e o MongoDB, aplicado para a tarefa de gestão e armazenamento dos dados.

Para o *frontend* – constituído pela aplicação Android, pelo Parse Dashboard e pelo CMS – foi utilizado maioritariamente React para o desenvolvimento do Parse Dashboard e do CMS.

Por fim, relativamente à gestão e ao *deployment* dos componentes constituintes do projeto – instâncias do Parse Server, Parse Dashboard e MongoDB – foi utilizado Docker.

2.1 Desenvolvimento de aplicações móveis

Ao planear e desenvolver uma aplicação móvel, existe a possibilidade de escolha entre o desenvolvimento de uma aplicação móvel nativa ou de uma aplicação móvel híbrida. No decorrer do processo de decisão entre uma aplicação móvel nativa e uma aplicação móvel híbrida, é necessário ter em conta um conjunto de fatores, não existindo a melhor solução, mas apenas a solução mais adequada consoante a situação enfrentada.

Tendo em consideração os objetivos de negócio de uma empresa, as metas que esta pretende atingir e o seu propósito, a decisão acerca do tipo de aplicação móvel que irá ser desenvolvida poderá ser crítica. Precedentemente à etapa de desenvolvimento, devem ser considerados fatores que permitam escolher qual o tipo de aplicação mais adequado, como por exemplo a complexidade que a aplicação irá exigir e o orçamento disponível [3].

As aplicações móveis nativas são construídas para plataformas específicas e, como tal, são escritas nas linguagens de programação aceites por essas mesmas plataformas (as aplicações para Android[4] são escritas em Java, as aplicações para iOS[5] são escritas em Swift[6] ou em Objective-C[7], etc.). Estas aplicações são construídas através da utilização de um IDE específico, consoante o sistema operativo em que a aplicação se encontra e podem ser utilizadas ferramentas de desenvolvimento e SDKs próprios, assim como os próprios elementos para a interface do utilizador.

Consequentemente, este tipo de aplicações deve ser evitado quando se tenha em mente o desenvolvimento de aplicações simples, uma vez que se pode tornar dispendioso e que requer uma equipa experiente, visto que a curva de aprendizagem das linguagens implicadas é acentuada.

[4] https://developer.android.com/studio/

[5] https://www.apple.com/pt/ios/ios-11/

[6] https://developer.apple.com/swift/

[7] https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/

Contudo, as aplicações móveis nativas apresentam um amplo conjunto de vantagens em comparação com outros tipos de aplicações existentes, como por exemplo:

- Rapidez e responsividade, uma vez que este tipo de aplicações é idealizado para ser executado numa plataforma específica;
- Disponibilização em plataformas de distribuição digitais como a App Store[8] para aplicações iOS e o Google Play[9] para aplicações Android.
- Mais interativas e intuitivas;
- Proporcionam uma melhor *user experience*, uma vez que o fluxo é mais natural ao existirem diferentes padrões da UI para cada plataforma.

Quanto às aplicações móveis híbridas, pode dizer-se que são aplicações que funcionam em várias plataformas e que se comportam como aplicações nativas, sendo essencialmente uma combinação das funcionalidades de aplicações nativas e de aplicações web.

Estas aplicações são, no entanto, mais lentas e menos interativas quando comparadas com as aplicações nativas, para além de que, muitas das vezes, sacrificam a *user experience* por não serem customizadas consoante a plataforma. Ainda assim, algumas das vantagens proporcionadas por este tipo de aplicações são:

- Fácil construção, pela utilização de tecnologias web como HTML, CSS e JavaScript;
- Menor custo comparativamente às aplicações nativas;
- Menor tempo de desenvolvimento, uma vez que apresenta uma única base de código;
- Acesso às API internas dos dispositivos (como o armazenamento, a câmara, etc.);
- Não necessita de um *browser* para ser executada, por oposição às aplicações web.

Em suma, o desenvolvimento de aplicações móveis nativas pode apresentar um custo inicial mais elevado, no entanto permite poupar tempo e dinheiro a longo prazo e oferece uma experiência mais personalizada aos seus utilizadores.

[8] https://www.apple.com/pt/ios/app-store/

[9] https://play.google.com/store?hl=pt_PT

2.2 Desenvolvimento de aplicações web com React

O React[10] é uma biblioteca JavaScript, desenvolvida pelo Facebook, bastante utilizada para a construção de UIs e que aplica uma sintaxe denominada JSX, que lhe permite combinar HTML com JavaScript e, desta forma, facilitar a escrita de componentes que resultam num código mais simples e limpo.

Também através da utilização de componentes, verifica-se uma diminuição da complexidade da UI, ao tornar possível a convergência de secções de código relacionadas, ou que tenham o mesmo propósito, em componentes individuais. Cada componente tem ainda a sua própria lógica, controla a forma como faz o seu *render* e pode ser reutilizado conforme seja necessário, sendo que essa reutilização ajuda a tornar as aplicações consistentes, fáceis de desenvolver e fáceis de manter [4].

Uma característica do React é a possibilidade de transmitir dados a componentes, quer seja para o *parent component* ou para os seus *children components*. Para transmitir dados de um *parent* para ele próprio ou para um child, são utilizadas as *properties* ou *props*, que correspondem a dados imutáveis. Para dados mutáveis existe o *state*, utilizado para as comunicações que se realizem internamente a um componente.

Outra característica principal é a existência de um *virtual DOM*, um conceito de programação segundo o qual é mantida uma representação virtual da UI em memória, que por sua vez é sincronizada com o *real DOM* sempre que o conteúdo sofra alterações. O *real DOM*, que corresponde à representação real da UI, é atualizado sempre que seja necessário ou que as alterações sofridas o justifiquem [5]. Esta característica é de elevada importância, sobretudo para aplicações que envolvam elevadas *user interactions* e que sofram atualizações constantes dos seus dados, pois pode ter um elevado impacto no desempenho da aplicação.

Embora seja geralmente referido como tal, o objetivo do React não é o de oferecer uma *framework* completa – uma vez que possui uma implementação simples e estritamente direcionada para a construção de UIs, constituindo uma solução muito focada, correspondente ao V no MVC [4] – existe a possibilidade de que este não inclua muitas das ferramentas necessárias para determinadas aplicações que precisem de implementações mais robustas. No entanto, oferece a liberdade de que sejam integradas outras bibliotecas, sendo frequentemente utilizado em conjunto com o Redux[11].

Em suma, podem ser enumeradas as seguintes vantagens com a utilização do React:

- Capacidade de transmitir dados e manipular componentes sem ter de recarregar a página;
- Eficiência e rapidez, conseguida através da criação do *virtual DOM* que calcula todas as alterações necessárias antes de as aplicar ao *real DOM*, aumentando o desempenho e fazendo um *render* mais rápido.

[10] https://reactjs.org/

[11] https://redux.js.org/

Alguns exemplos de empresas que adotaram o React nas suas aplicações são o Airbnb, Instagram, Netflix, PayPal e Walmart [6].

2.3 Microserviços e utilização de *containers*

Um microserviço é uma pequena aplicação que pode ser escalada e testada de forma independente e cujo *deploy* também pode ser realizado independentemente. Este tipo de aplicações segue um princípio de responsabilidade única, no sentido de que faz apenas uma tarefa individualmente e de que pode ser facilmente entendida [7].

A utilização de *containers* pode ser descrita como uma forma de virtualização ao nível do sistema operativo, pois permite isolar o sistema operativo *host* dos *containers*. Ao contrário das máquinas virtuais, com a utilização de *containers* não é necessário um sistema operativo completo para funcionar. Enquanto que com as máquinas virtuais tem de existir um sistema operativo *host* e um sistema operativo *guest*, sendo necessária ainda a utilização de um *hypervisor* que traduza as instruções entre ambos os sistemas, com a utilização de *containers* os processos correm sobre o sistema operativo *host* de forma isolada (Figura 1). O Docker é um exemplo de tecnologia de virtualização baseada em *containers*. Esta tecnologia permite automatizar processos de configuração e de execução repetitivos.

Em suma, a utilização de *containers* torna possível usar menos recursos, uma vez que os *containers* partilham o sistema operativo com a máquina onde estão a ser executados. Para além disso, como não necessitam de um *hypervisor* para a tradução das instruções entre os sistemas, as aplicações que correm nos *containers* vão apresentar um desempenho idêntico ao desempenho apresentado pela execução das mesmas no sistema operativo *host* [8].

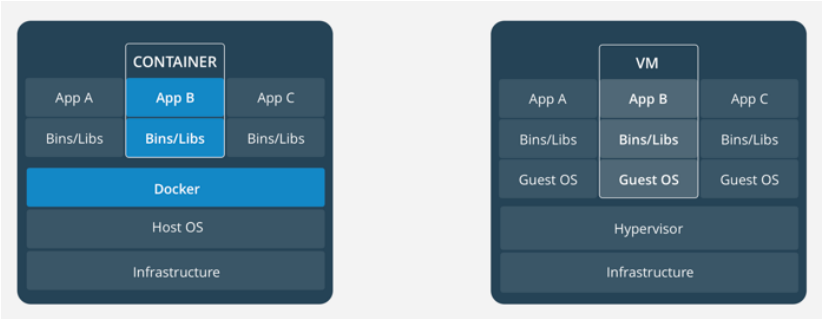


Figura 1 – Containers vs Máquinas Virtuais

Referência: https://www.docker.com/what-container

3 REQUISITOS DE UMA PLATAFORMA DE *BACKEND* PARA APLICAÇÕES MÓVEIS

O termo *Mobile Backend as a Service* pode ser utilizado para descrever um modelo que atua como *middleware* e que permite conectar as aplicações a serviços *cloud* através de APIs e SDKs [9]. Para além disso, ajuda a mitigar a complexidade de integração de funcionalidades de valor agregado, ao utilizar os SDKs e as APIs para fornecer determinados recursos como notificações, possibilidade de integração com redes sociais, serviços de localização, armazenamento de ficheiros, *analytics* e gestão de dados e utilizadores [10].

Deste modo, ao assumirem a complexidade do *backend*, as plataformas de *Mobile Backend as a Service* permitem agilizar o desenvolvimento de aplicações móveis, podendo ver o tempo de desenvolvimento reduzido em três quartos (Figura 2) do tempo que é necessário para desenvolver uma aplicação que não utilize MBaaS [11][2].

Este tipo de plataformas é, cada vez mais, uma tendência de desenvolvimento que irá continuar a apresentar um aumento de popularidade nos próximos anos, impulsionada sobretudo pelo crescimento das economias orientadas para as aplicações, e pela subsequente indispensabilidade das aplicações móveis na criação de valor económico [12][10]. Para além disso, como já foi referido anteriormente, observou-se ao longo do tempo uma crescente complexidade de gerir as funcionalidades de *backend* e a integração das mesmas no desenvolvimento de aplicações, o que proporcionou que as plataformas de MBaaS recebessem um maior destaque.

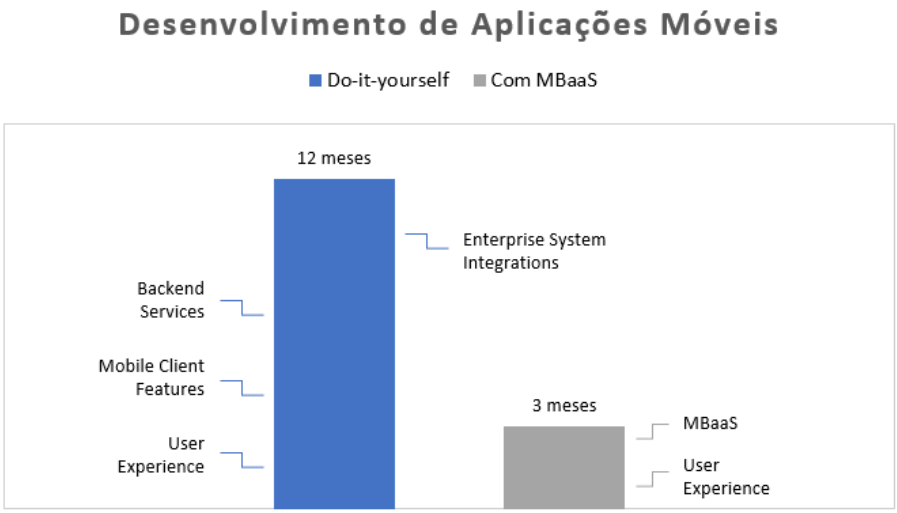


Figura 2 – Comparação do tempo de desenvolvimento de aplicações DIY e MBaaS [13]

Para além de permitirem desenvolver aplicações com um menor sacrifício no que diz respeito ao desenvolvimento do lado do servidor e ao tempo de configuração, o modelo *Mobile Backend as a Service* apresenta muitos benefícios importantes a ter em consideração, tais como [1][12]:

- Menor tempo de desenvolvimento – ao fornecer muitas das necessidades subjacentes, permite que seja construído apenas o necessário em cima da estrutura já existente, em vez de se ter de começar sempre do zero;
- Preço – a maior parte dos fornecedores oferece um modelo *freemium*;
- Facilmente configurável – a configuração é geralmente fácil, no entanto, para BaaS *open source* pode tornar-se um pouco mais complicado, uma vez que é necessário configurar os servidores e instalar dependências;
- Grande diversidade de plataformas disponíveis;
- Permite uma maior acessibilidade – se cada aplicação possuir a mesma base subjacente, tem o potencial de conectar facilmente as aplicações entre as várias plataformas;
- Escalabilidade – as plataformas BaaS são construídas de forma a estarem preparadas para servir qualquer aplicação, portanto devem ser flexíveis;
- Funcionalidades adicionais – possibilita acrescentar elementos complementares como os SDK.

Apesar de apresentar diversos benefícios, o modelo *Mobile Backend as a Server* também apresenta algumas desvantagens enumeráveis. Uma das maiores desvantagens deste tipo de plataformas, à qual se deve dar especial importância, é a probabilidade de encerramento do serviço [1][2]:

- Preço – no processo de escolha deve ser tido em conta o cenário de crescimento da aplicação e a existência dos modelos premium e subscription;
- Probabilidade de encerramento do serviço – o que resultará na necessidade de migrar todos os dados para outro sistema de backend.

3.1 Serviços comuns para suportar aplicações móveis

A escolha da plataforma de *Mobile Backend as a Service* mais adequada pode tornar-se uma tarefa difícil uma vez que, para além da grande disponibilidade de plataformas existente, esta escolha depende maioritariamente das necessidades e do objetivo das aplicações às quais a plataforma se destina.

Atualmente, grande maioria dos fornecedores deste tipo de soluções dispõe de várias funcionalidades comuns, tais como [14][1]:

- Operações CRUD sobre os dados;
- Autorização (integração com fornecedores de OAuth já existentes);
- Notificações;
- Gestão de utilizadores;
- Armazenamento de ficheiros;
- Geolocalização.

Desta forma, estas soluções englobam a lógica de negócio das aplicações e o processamento e gestão de dados (Figura 3). *"In the context of enterprise application development, the concept of the services in the "Data Processing/Management" section is defined through integrations with existing systems. For instance, the user service which is responsible for user registrations, logins, user permissions or user role management may need to work with an external user identity management system."* [15].

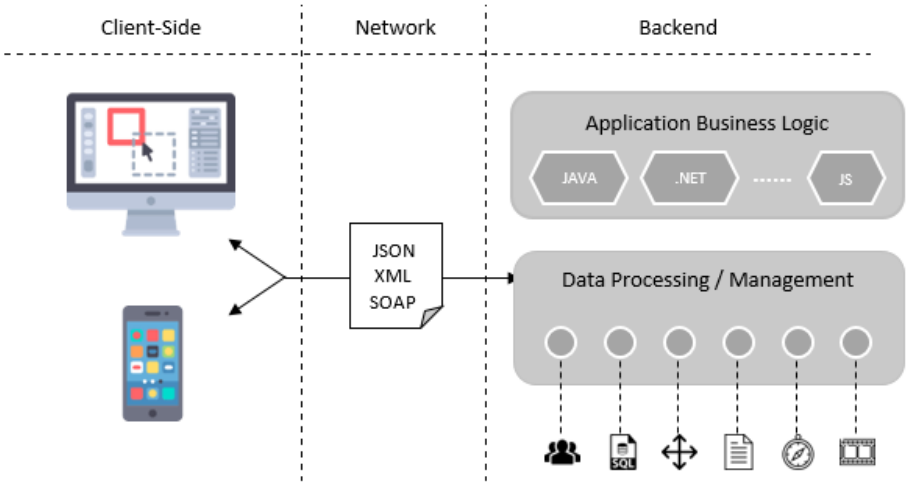


Figura 3 – Esquema de um *backend* e das funcionalidades que este engloba [16]

Para além das funcionalidades que a solução engloba, devem também ser tidos em conta alguns tópicos igualmente importantes no momento da escolha, nomeadamente o preço e a escalabilidade. Embora seja difícil estimar a demanda que uma aplicação móvel terá antes do seu lançamento, a escolha de um *backend* que escale eficazmente à medida que o público da aplicação cresce é muito importante. Alguns fornecedores de BaaS oferecem um modelo *freemium* e preços baseados na utilização e aumento de escalabilidade, permitindo que se consiga gerir mais eficazmente o risco comercial [14].

3.2 Análise de plataformas existentes

Após o anúncio de encerramento do Parse, deu-se um processo de transição em que houve a necessidade de migrar os dados do serviço do Parse para outros serviços semelhantes. Como tal, existem inúmeras plataformas e serviços alternativos ao Parse que apresentam diversas funcionalidades que o atual Parse Server não inclui e que lhe poderiam ser úteis. Exemplos dessas funcionalidades são o envio de notificações e mensagens instantâneas, o módulo de *analytics* e o envio automático de emails de confirmação de conta ou de recuperação de *password*, disponibilizados de forma integrada nas soluções que competem diretamente com o Parse Server.

No entanto, apesar destas plataformas constituírem fortes alternativas ao Parse, são soluções que diferem do mesmo – e também entre elas próprias – sobretudo na questão do preço. Ainda que consigam ser mais simples e práticas que o Parse Server em termos de implementação e manutenção, uma grande maioria destas alternativas requer um pagamento recorrente consoante os serviços que engloba. Este motivo pode constituir uma desvantagem para estas soluções quando comparadas com o Parse Server, dado que este é *open-source* e, ainda que suas alternativas possuam planos gratuitos de utilização, esses planos apresentam limitações nos serviços que oferecem. Para ultrapassar essas limitações, é necessário pagar por um plano que confira uma maior liberdade de utilização dos seus serviços, o que pode vir a tornar-se um problema futuro devido ao cenário de crescimento da aplicação e à demanda que a mesma poderá vir a exigir. Como foi analisado anteriormente, de forma a prevenir possíveis inconvenientes resultantes do crescimento inesperado da demanda das aplicações, no processo de escolha da plataforma, deve ser escolhida uma solução que assegure a escalabilidade das aplicações num amplo espaço de tempo.

Outra possível desvantagem de optar por qualquer uma destas alternativas ao invés de optar pelo Parse Server é a possibilidade de encerramento do serviço. Não existe garantia de que a situação – de encerramento da plataforma – que ocorreu com o Parse não seja possível de se vir a repetir com qualquer outra destas plataformas. Tendo isto em conta, o simples facto de o Parse Server ser uma plataforma *open source*, mantida por uma comunidade de programadores que assumiu o projeto após o distanciamento do Facebook do mesmo, constitui uma vantagem em comparação com outras alternativas de código fechado que se encontrem sob o domínio de entidades empresariais com fins lucrativos. Embora o facto de ser *open source* não lhe confira uma superioridade inequívoca – a comunidade pode sempre abandonar o projeto, deixando de contribuir para o mesmo – irá ser seguramente mais doloroso presenciar o encerramento de um serviço de código fechado (ainda mais se esse serviço for pago) pois o mesmo irá provocar um processo de transição e de migração dos dados para um outro serviço alternativo, que também não estará livre do mesmo destino.

Firebase e Flamelink CMS

A Firebase[12] é uma plataforma de *Backend as a Service*, fundada em 2011 e adquirida pela Google em 2014 [17][18]. Para além de disponibilizar funcionalidades para desenvolver e testar aplicações, esta plataforma possibilita ainda a integração com outros serviços da Google, tais como:

- **Google Analytics** – fornece uma visão acerca da forma como os utilizadores interagem com as aplicações, o que permite entender o comportamento dos mesmos e adaptar a estratégia aplicada conforme as informações recolhidas;
- **App Indexing** – permite direcionar o tráfego de pesquisa, através da exibição de *links* para a aplicação nas páginas de resultados de pesquisas do Google. Possibilita ainda melhorar o *ranking* da aplicação, para que a mesma tenha maior destaque nas páginas de resultados das pesquisas e para que a aplicação seja mais facilmente descoberta por novos utilizadores [19];
- **AdMob** – permite que as aplicações móveis gerem receita através de publicidade segmentada. Esta plataforma de publicidade móvel utiliza o SDK de anúncios da Google, que permite adquirir informações acerca dos utilizadores da aplicação, impulsionar mais compras na aplicação e maximizar a receita advinda dos anúncios através da recolha de dados e informações de dispositivos [20];
- **Google AdWords** – colabora no impulsionamento do número de instalações das aplicações e proporciona a exibição de campanhas de anúncios segmentados, dirigidos aos públicos-alvo do Google Analytics [21].

A Firebase apresenta ainda algumas características de maior destaque e interesse, sobretudo no contexto do presente trabalho, de salientar:

- **Cloud Storage** – facilita o rápido armazenamento e disponibilidade do conteúdo;
- **Realtime Database** – armazenamento e sincronização de dados em tempo real, o que facilita o acesso aos dados em qualquer lugar e independentemente do dispositivo. Permite desenvolver aplicações sem a existência de um servidor e encontra-se otimizada para utilização *offline*, uma vez que os seus SDKs utilizam a cache local do dispositivo para aplicar e armazenar as alterações e quando o mesmo volta a ficar *online* os dados são sincronizados automaticamente;
- **Autenticação** – possibilita que os utilizadores façam login rapidamente com a conta do Facebook, Twitter, Google ou GitHub[13].

[12] https://firebase.google.com/

[13] https://github.com/

Em termos de preço, esta solução dispõe do Plano Spark, um plano gratuito, mas com algumas limitações nas funcionalidades e produtos que inclui. Para além do plano gratuito, a Firebase disponibiliza o Plano Flame a partir de 20€ por mês e o Plano Blaze, um plano de pagamento dependente do grau de utilização do serviço.

Uma funcionalidade que a Firebase não inclui é a disponibilização de um CMS, isto é, de uma interface que permita adicionar e gerir conteúdo de uma forma relativamente simples. Para colmatar esta necessidade surgiu o Flamelink[14], um CMS fundado em outubro de 2017 especificamente para ser integrado com a Firebase. Esta solução dispõe de um SDK JavaScript – útil para o desenvolvimento web – e compreende as funcionalidades [22][23]:

- **Tipos de conteúdo flexíveis** – permite criar os próprios tipos de conteúdo e dispõe de um criador de conteúdo *drag-and-drop*;
- **Campos de dados relacionais** – para referenciar os mesmos dados a partir de diferentes tipos de conteúdo;
- **Gestão de utilizadores** – permite adicionar e gerir utilizadores, bem como controlar as permissões dos mesmos, de forma integrada com os serviços de autenticação da Firebase;
- **Gestão de multimédia** – possibilita o *upload* de qualquer tipo de ficheiros, o redimensionamento automático de imagens e a criação de diretórios, sendo tudo armazenado na base de dados da Firebase;
- **Interface de simples navegação** – facilita a criação de múltiplos menus, classes customizadas e hierarquias ordenáveis.

Para além das funcionalidades já disponibilizadas, encontra-se idealizado um conjunto de funcionalidades para serem lançadas mais tarde, tais como:

- **Backups** – atualmente é uma funcionalidade paga da Firebase, exclusiva para assinantes do Plano Blaze [23][24];
- **Multi-ambiente** – permite testar as alterações efetuadas num ambiente de produção antes de as tornar públicas;
- **Suporte multilinguagem**;
- **Configurações de controlo de utilizador** (como tamanhos de imagem, etc.);

Em termos de preço, esta solução dispõe de um plano gratuito, de planos mensais a partir de 7€ por mês, de planos anuais a partir de 68€ por ano e de planos customizáveis. O que distingue os vários planos é o número de utilizadores permitido e o suporte prestado.

[14] https://flamelink.io/

SashiDo

A SashiDo[15] é uma plataforma de *Mobile Backend as a Service* que oferece uma solução baseada na junção do Parse Server com MongoDB[16]. Esta solução também disponibiliza um conjunto próprio de funcionalidades, como por exemplo [25]:

- **Cloud Code integrado com o GitHub** – é possível ter acesso a esta funcionalidade a partir de uma secção presente na *dashboard* da SashiDo, pela associação de uma conta do GitHub, sendo fornecido um repositório privado por cada aplicação, com liberdade para adicionar colaboradores, criar *branches*, etc.;
- **LiveQueries** – incluídas na *dashboard*, permitem que a aplicação atue em tempo real. Resumidamente, as LiveQueries permitem subscrever uma determinada *query* e sempre que um objeto que satisfaz essa *query* sofra alterações, o servidor irá enviar imediatamente para o cliente os novos dados daquele objeto;
- **Notificações** – com a possibilidade de escolha da plataforma móvel, do público alvo e de obtenção de uma pré-visualização antes do envio concreto;
- **Background Jobs** – oferece a hipótese de incluir tarefas de longa execução no Cloud Code e de configurar com que frequências as mesmas irão ser executadas;
- **WebHooks** – possibilita a integração de *frameworks* e tecnologias não suportadas pelo Cloud Code e a programação em outras linguagens para além do JavaScript. Oferecem uma comunicação *server-to-server* simples e eficaz sem a necessidade de conexões de longa duração.

Para além deste conjunto de funcionalidades, a SashiDo fornece uma cobertura de múltiplas regiões e uma forma segura e automatizada de escalar as aplicações, permitindo lidar com tráfego inesperado. É ainda expectável que esta plataforma lance mais funcionalidades em breve, tais como [26]:

- Serviços de localização;
- Teste A/B de notificações;
- Envio automático de mensagens;
- Introspeção acerca da forma como a aplicação é utilizada, ou *analytics.*

Quanto aos preços estipulados, a SashiDo oferece um período de teste de 14 dias, no entanto não dispõe de um plano de utilização gratuito. Esta solução apresenta um preço inicial de 4€ por mês, que por sua vez varia consoante a região, contudo a SashiDo garante aplicar um preço é flexível, baseado na utilização mensal da plataforma.

[15] https://www.sashido.io/

[16] https://www.mongodb.com/

Backendless

O Backendless[17] é uma plataforma de *Mobile Backend as a Service*, cujo *core* dos serviços oferecidos se encontra nas suas APIs, uma vez que cada uma das APIs fornecidas por esta plataforma pode ser ampliada com código customizado, consoante a necessidade das aplicações [26]. Para além disso, ainda disponibiliza todos os serviços necessários para a construção de aplicações móveis, entre eles [27]:

- **Autenticação e gestão de utilizadores** – inclui os processos de registo, gestão de *passwords* e edição de perfis de utilizadores, bem como de autenticação com redes sociais como o Facebook, Twitter ou Google;
- **Armazenar dados relacionais** – os objetos armazenados na base de dados desta plataforma podem referenciar outros objetos relacionados, sendo admitidas relações entre objetos de *one-to-one* e de *one-to-many*;
- **Notificações** – abrange desde notificações simples de texto e atualizações nos contadores dos *badges* das aplicações, a notificações sonoras; A *dashboard* do Backendless, denominada Backendless Console, contém uma secção para gerir dispositivos registados e para compor e enviar notificações. Desta forma, as notificações podem ser agendadas ou enviadas instantaneamente, para dispositivos com os sistemas operativos Android, iOS ou Windows Phone[18];
- **Cloud Code** – simplifica a execução de código customizado do lado do servidor, permitindo centralizar a lógica da aplicação e estender ou sobrepor o comportamento por omissão dos serviços;
- **Geolocalização** – o conceito de localização é adicionado a qualquer objeto sobre a qual a aplicação opere, quer seja um utilizador, ficheiro ou local de interesse;
- **Analytics** – recolhe informação pertinente e fornece uma visão acerca da utilização das aplicações. Podem ser analisadas situações como a dinâmica de crescimento dos utilizadores e a atividade de envio de mensagens, por exemplo;
- **SDKs *open source*** – dispõe de um conjunto de SDKs, todos *open-source* e hospedados no GitHub, para as plataformas Android, iOS, JavaScript e .NET.

Em termos de preço, esta solução dispõe de um plano inicial gratuito, mas com algumas limitações expectáveis. Consoante o crescimento da aplicação, existem ainda planos pagos à escolha, desde o pano *developer* a partir de cerca de 6€ por mês à possibilidade de pagar para aumentar os limites do plano escolhido, conforme a necessidade.

[17] https://backendless.com/

[18] https://support.microsoft.com/pt-br/help/10650/get-started-with-windows-phone

Socket.IO

O Socket.IO[19] é uma biblioteca que permite realizar comunicações bidirecionais, em tempo real e baseadas em eventos. Consiste num servidor Node.js e numa biblioteca JavaScript para a aplicação cliente (também com implementações em outras linguagens como Java, C++ e Swift). Esta *framework*, para além de ser *open-source*, compreende essencialmente as seguintes funcionalidades:

- **Confiabilidade** – as conexões são estabelecidas mesmo na presença de *proxies*, *firewalls* pessoais e antivírus. Para esse efeito depende do Engine.IO[20], que implementa uma comunicação bidirecional *cross-browser / cross-device*;
- **Deteção de desconexões** – é implementado um mecanismo de *heartbeat*, que consiste no envio de um sinal periódico em intervalos de tempo regulares como indicativo de um normal funcionamento daquele componente, e que permite ao servidor saber quando algum cliente deixou de estar disponível e vice-versa;
- **Reconexões automáticas** – sempre que ocorrer uma desconexão, salvo indicação contrária, o cliente irá tentar conectar-se eternamente até o servidor voltar a estar disponível;
- **Suporte para multiplexação** – separação de conceitos, podendo ser uma separação por módulos ou baseada em permissões, através da criação de Namespaces, que irão atuar como canais de comunicação distintos, mas que partilham a mesma conexão subjacente;
- **Estabelecimento de canais** – no contexto de cada Namespace, podem ser definidos canais arbitrários, denominados Rooms, aos quais os *sockets* se podem conectar ou desconectar. Esta funcionalidade tem a utilidade de permitir o envio de notificações para um grupo particular de utilizadores, ao realizar a transmissão para uma Room específica que por sua vez afetaria todos os *sockets* ligados à mesma;
- **Analytics em tempo real** – envio de dados para os clientes, dados esses representados como contadores em tempo real, gráficos ou *logs*;
- **Binary *streaming*** – possibilita o envio de BLOBs como imagens, áudio e outros objetos multimédia;

[19] https://socket.io/

[20] https://github.com/socketio/engine.io

CloudBoost

O CloudBoost[21] é uma plataforma *cloud* que oferece uma solução baseada na combinação de vários serviços oferecidos por outras plataformas existentes: "*Think of CloudBoost as Parse + Firebase + Algolia + Iron.io all combined into one*" [28]. Deste modo, esta plataforma abrange os seguintes serviços:

- **Autenticação de utilizadores;**
- **Notificações** – suporta dois tipos de notificações em tempo real: Custom Realtime Notifications e CloudObject Notifications. O primeiro tipo é útil por exemplo numa aplicação de *chat* e funciona com base em canais e na publicação de mensagens para esses mesmo canais. Por outro lado, as CloudObject Notifications dizem respeito a notificações de eventos que ocorrem na base de dados, como por exemplo na publicação de um comentário numa rede social, que levaria à atualização automática da página;
- **Armazenamento de dados** – o risco e o custo associado à escalabilidade é eliminado e há a garantia de que os dados estão sempre disponíveis em qualquer situação;
- **Aplicações em tempo real** – quando os dados são alterados, as aplicações são instantaneamente atualizadas para abrangerem essas alterações, em todos os dispositivos;
- **Pesquisa** – os dados são indexados e os utilizadores têm à sua disposição a funcionalidade de CloudSearch, para realizar pesquisas sobre esses dados e receber informação relevante e precisa;
- **Cache** – armazenamento chave/valor indicado para aplicações que necessitam de partilhar o seu estado, transmitir informação e coordenar as atividades entre processos e dispositivos;
- **Queues** – permite a comunicação entre componentes de forma segura e garante uma alta disponibilidade.

Em termos de preço, esta solução dispõe de um plano inicial gratuito com algumas limitações esperadas, como por exemplo o facto de não permitir o acesso nativo à base de dados MongoDB, o que impossibilita ter o controlo completo da mesma e impede a utilização das suas APIs nativas. Existem ainda planos pagos, desde cerca de 40€ por mês e a possibilidade de escolher um plano que se adapte consoante a necessidade das aplicações ou de escolher um plano que permita customizar os serviços que o mesmo engloba.

[21] https://www.cloudboost.io/

Syncano

O Syncano[22] é uma plataforma *serverless* que permite fazer *deploy* do backend para a *cloud* instantaneamente, proporcionando o desenvolvimento de aplicações de uma forma mais eficiente. Esta solução apresenta as seguintes particularidades:

- **Comunidade** – o Syncano possui uma comunidade dedicada ao projeto, para além da equipa principal. A equipa responsável por esta plataforma afirma ter planos para a criação de um Socket Marketplace, que irá permitir que os seus utilizadores especifiquem o Socket de que precisam e obtenham ajuda de outos utilizadores para o construir. Para além disso, é também um objetivo da equipa tornar o Syncano Cloud OS *open-source*.
- **SDK de automação** – permite acelerar o processo de desenvolvimento de aplicações. Através deste SDK podem ainda ser agregadas funções para a customização do backend, sendo que esse processo ocorre localmente e o SDK sincroniza as alterações de forma dinâmica e contínua com o Syncano Cloud OS;
- **Syncano Cloud OS** – fundamental para a plataforma Syncano, hospeda e executa todos os Syncano Sockets que constituem o backend dedicado, assegurando que o backend escala de forma flexível para responder às necessidades do *frontend*. Atualmente é executado no AWS, mas também pode ser executado em qualquer outra infraestrutura privada, uma vez que é agnóstico em termos de infraestrutura;
- **Syncano Socket Registry** – o Syncano permite o acesso a vários componentes criados pela comunidade, que podem ser utilizados para customizar o backend dedicado. Os "blocos padronizados" utilizados para a construção do backend, um dos conceitos fundamentais da plataforma, são denominados Syncano Sockets. Cada Socket tem um propósito único e claro, quer seja o de enviar um email, traduzir um ficheiro de texto ou analisar uma página web. Um Socket pode conectar-se a outro Socket e a combinação de Sockets permite eliminar o sacrifício de desenvolver código – pré-existente – por conta própria;

Em termos de preço, esta solução oferece um período de teste de 30 dias, no entanto não dispõe de um plano de utilização gratuito. Além disso, apresenta um plano base de 20€ por mês, com acesso completo a todas as funcionalidades e com armazenamento e número de utilizadores ilimitado. Conforme a necessidade, é ainda possível ajustar o plano de acordo com o crescimento das aplicações ou com a exigência das mesmas.

[22] https://syncano.io/

Back4App

O Back4App[23] é uma plataforma que permite criar, hospedar e gerir aplicações de forma simples e rápida. Esta plataforma encontra-se hospedada em servidores AWS e permite o desenvolvimento de aplicações multiplataforma. Para além disso, oferece um vasto conjunto de recursos, tais como:

- **Auto escalável** – a plataforma foi idealizada com o problema da escalabilidade em mente, tendo em conta tanto as grandes aplicações como as aplicações de menores dimensões, que confiam os seus dados e informações à plataforma;
- **Notificações** – possibilita a criação de canais de forma a que um certo grupo de utilizadores consiga subscrever determinados canais, permitindo enviar notificações especificamente para certos canais e, consequentemente, para certos grupos de utilizadores. O público-alvo pode ser segmentado através da *dashboard*, com base em alguma característica conhecida, facilitando o envio de conteúdo relevante para segmentos de utilizadores mais apropriados;
- **LiveQueries** – permitem subscrever uma *query* específica, armazenar e sincronizar dados em tempo real. O servidor notifica os clientes que subscreveram uma determinada *query*, quando algum objeto que corresponda a essa *query* for criado, alterado ou apagado;
- **Smart Indexing** – criação automática de índices de acordo com o fluxo de dados recebido, permitindo uma gestão mais eficaz de instâncias Mongo;
- **Rotina de backups** – os servidores estão configurados para efetuar um backup aos dados e ao código de hora em hora;
- **Redundância e disponibilidade** – o MongoDB utiliza conjuntos de réplicas para garantir tanto a redundância da informação como a sua alta disponibilidade. Os conjuntos de réplicas correspondem a grupos de instâncias que hospedam o mesmo conjunto de dados;
- **MongoRocks** – implementação de MongoDB com RocksDB, para tirar partido do desempenho acrescido e das capacidades de backup da RocksDB [29].

No que diz respeito ao preço, esta plataforma dispõe de um plano gratuito com limitações no armazenamento que viabiliza e no número de pedidos que permite realizar. De outra forma, possui planos pagos consoante os recursos pretendidos e proporciona a aquisição de recursos extra (obtenção de mais espaço de armazenamento, possibilidade de incorporar um maior número de pedidos, etc.).

[23] https://www.back4app.com/

3.3 Caracterização do Parse e Parse Server

O Parse é uma plataforma de *Mobile Backend as a Service* que surgiu em junho de 2011, em São Francisco, fundada por Tikhon Bernstam, James Yu, Ilya Sukhar e Kevin Lacker. O código fonte era fechado e constituía um serviço pago. A empresa continuou a crescer graças a investimentos e *fundraising*, tendo recebido 1.5 milhões de dólares, ainda em 2011, financiada pelo Y Combinator[24] e por outros investidores.

Mais tarde, em finais de 2011, a empresa conseguiu angariar 5.5 milhões de dólares para o desenvolvimento do Parse, a partir de uma sessão de financiamento criada pelos Ignition Partners [2][30][31].

Em 2013, o Facebook comprou o Parse por 85 milhões de dólares e continuou a investir no produto, tornando-o mais robusto e adicionando funcionalidades à plataforma [32][33].

Em janeiro de 2017 a plataforma foi encerrada, após o anúncio do seu encerramento eminente a 28 de janeiro de 2016, uma vez que com o passar do tempo o Parse afastara-se do propósito da empresa e dos produtos principais da mesma.

No entanto, em vez de apenas fechar a plataforma, o Facebook decidiu torná-la *open source* e compensar os seus utilizadores ao lançar um SDK *open source* e vários tutoriais de forma a explicar como migrar dos dados para outras plataformas [34].

Atualmente, a comunidade *open source* tem acesso a uma plataforma de *Backend as a Service* completa e *open-source* – resultante de um investimento prévio de inúmeros milhões de dólares – conseguindo assim obter o máximo controlo sobre a sua infraestrutura, código e dados.

O Parse Server é a versão *open-source* do Parse e encontra-se disponível no repositório da comunidade *open-source* [35]. No entanto, existem grandes diferenças entre as duas plataformas (Tabela 1). Apesar de o Parse Server já suportar muitas das funcionalidades do Parse, algumas delas ainda não se encontram implementadas, como é o caso das analytics, das notificações e dos e-mails do sistema [34][36].

[24] https://www.ycombinator.com/

Tabela 1 – Diferença de funcionalidades entre as plataformas Parse e Parse Server

Funcionalidades	Parse	Parse Server
Analytics	✓	✗
Autenticação	✓	✓
Notificações	✓	✗
E-mails automatizados	✓	✗
Logs	✓	✓
Dashboard	✓	✓
Queries	✓	✓
Utilizadores	✓	✓
Roles	✓	✓
Ficheiros	✓	✓

O Parse dispunha de três funcionalidades principais: Parse Core, Parse Analytics e Parse Push (Tabela 2). O Parse Core estava relacionado com a gestão de dados e utilizadores, integração com as redes sociais e criação da lógica do lado do servidor. O Parse Analytics responsabilizava-se por localizar qualquer ponto de informação numa aplicação em tempo real. Por fim, o Parse Push dizia respeito à funcionalidade de notificações, que auxiliava a criação de mensagens ou notificações e tratava do envio das mesmas para os utilizadores [2].

Tabela 2 – Principais funcionalidades do Parse

Principais Funcionalidades do Parse		
Parse Core	**Parse Analytics**	**Parse Push**
Gestão de dados	Localização de qualquer ponto de informação numa aplicação em tempo real	Criação de mensagens
Integração com redes sociais		Criação de notificações
Gestão de utilizadores		Envio das mensagens ou notificações para os utilizadores
Lógica do lado do servidor		

Embora não possua uma *dashboard* dedicada ou outras características que o Parse possuía, o Parse Server suporta muitas das funcionalidades do Parse e ainda implementa algumas que este último não incluía, como por exemplo as consultas em tempo real *(live queries)* [37].

Como foi observado anteriormente, existem certas funcionalidades que o Parse Server ainda não suporta e que são de particular interesse [38][36]:

- ***Analytics*** – uma característica de grande interesse e importância, pois fornecia uma visão acerca das interações dos utilizadores. Apesar de o Parse Server não suportar esta funcionalidade, existem algumas opções pagas que a substituem.
- **Notificações** – apesar de estarem disponíveis no Parse Server, requerem esforço de implementação. O *Push Adapter* permite que sejam enviadas notificações através da utilização de qualquer fornecedor de notificações *push*.
- **Envio automático de e-mails** – o Parse oferecia uma forma fácil de enviar e-mails de boas vindas, recuperação de *password*, verificação e outro tipo de e-mails automatizados. Apesar de não estar disponível no Parse Server, existe a alternativa de enviar e-mails aos utilizadores através da utilização de um fornecedor de e-mail.

3.3.1 Estrutura de Ficheiros do Parse Server

O Parse Server agrupa funcionalidades ou funções comuns em diretórios, sendo possível identificar alguns diretórios principais: Adapters, Controllers e Routers (Figura 4).

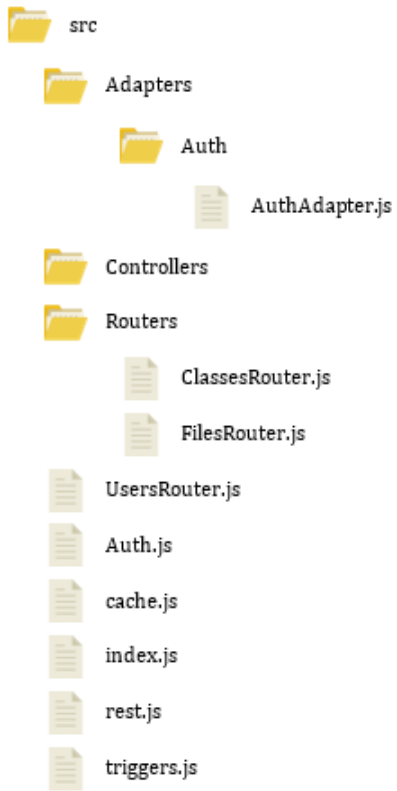


Figura 4 – Estrutura simplificada do Parse Server

O diretório Adapters engloba um conjunto de subdiretórios, entre eles o diretório Auth, que reúne um conjunto de ficheiros que são responsáveis pela gestão de toda a parte de autenticação da plataforma (redes sociais inclusive). No ficheiro AuthAdapter.js é criado um objeto Auth que contém informação acerca de quem realizou o pedido de autenticação e se foi utilizada uma *master key*. Este ficheiro também contém métodos auxiliares para conseguir um objeto Auth, ou a partir de um *token* de sessão [36].

O diretório Routers agrupa todos os ficheiros que lidam com rotas, sendo que o ficheiro ClassesRouter.js lida com as rotas /classes, o ficheiro FilesRouter.js lida com as rotas /files e o mesmo se verifica para a grande maioria dos restantes ficheiros pertencentes a este diretório. No entanto existem exceções, como é o caso do ficheiro AnalyticsRouter.js que lida com as rotas /eventos e que, por enquanto, apenas ignora o *input* e devolve uma resposta bem-sucedida ao cliente.

3.3.2 Parse Server Modules

Para colmatar a ausência de determinadas funcionalidades que não se encontram implementadas nativamente no Parse *open-source*, encontra-se disponível um conjunto de módulos e *adapters* que podem ser incorporados como uma forma de substituição das funcionalidades em falta, como por exemplo [39]:

- **applinks-metatag** – utiliza Express JS para injetar ligações referentes a conteúdo de aplicações móveis já existentes. Quando os utilizadores interagem com essas ligações, o conteúdo é carregado na aplicação móvel correspondente, caso o utilizador possua essa mesma aplicação [40][41];
- **SocketRocket** – um *fork* do repositório original pertencente ao Facebook [42] que disponibiliza uma biblioteca de WebSockets para iOS, macOS e tvOS [43];
- **parse-files-utils** – contém ferramentas para listar e migrar ficheiros do Parse, mas a funcionalidade de migração de ficheiros já não se encontra disponível, uma vez que o serviço do Parse foi encerrado [44].
- **parse-server-simple-mailgun-adapter** – permite enviar emails de reposição de *password* e de verificação de email através do Mailgun;
- **parse-server-amazon-ses-adapter** – permite enviar emails de reposição de *password* e de verificação de email com o Amazon SES[25];
- **parse-server-s3-adapter** – possibilita a utilização do Amazon S3[26] para o armazenamento de objetos;

[25] https://aws.amazon.com/pt/ses/

[26] https://aws.amazon.com/pt/s3/

- **parse-server-urban-airship** – *adapter* que permite utilizar o serviço de notificações da Urban Airship[27];
- **parse-server-onesignal-push-adapter** – *adapter* que permite utilizar o serviço da OneSignal[28] para permitir a entrega de um grande volume de notificações, independentemente da plataforma;
- **parse-server-gcloud-pubsub** – *adapter* que permite a utilização do serviço Google Cloud Pub/Sub, um serviço de mensagens em tempo real que permite o envio e a receção de mensagens entre aplicações independentes, para Live Queries [45][46].

Mailgun

Um dos *adapters* de maior relevância para o projeto e que merece especial destaque é o **parse-server-simple-mailgun-adapter**. Este *adapter* utiliza o Mailgun[29], um serviço que oferece um conjunto de APIs que permitem, entre outras particularidades:

- Enviar e receber emails de reposição de *password* e de verificação de email;
- Personalizar o conteúdo;
- Efetuar validações – possibilita a prevenção de erros de escrita ao verificar cada endereço de email que se encontra na lista. Para além de ser baseada em biliões de emails enviados e nas regras formais existentes para endereços de email, esta verificação integra regras de gramática personalizadas consoante o serviço de email, uma vez que cada fornecedor tem as suas próprias regras personalizadas para a verificação de emails;
- Realizar análises – proporciona a obtenção de informações avançadas acerca do desempenho dos emails consoante o serviço de email, o país ou o dispositivo em que o email foi recebido. Dispõe ainda de métricas que possibilitam prever os momentos mais adequados para o envio dos emails.

Esta solução permite enviar 10.000 mensagens e 100 validações gratuitamente todos os meses, possibilitando ainda a aquisição de recursos adicionais, conforme a necessidade.

[27] https://www.urbanairship.com/

[28] https://onesignal.com/

[29] https://www.mailgun.com/

Como alternativa, existe o **parse-server-amazon-ses-adapter**, que utiliza o serviço de email da Amazon (Amazon SES) e permite receber 1.000 mensagens e enviar 62.000 mensagens por mês, de forma gratuita, para qualquer destinatário [47]. Existem ainda outros *adapters* interessantes, que não são mantidos pela mesma comunidade responsável pelo Parse Server, como por exemplo:

- **parse-server-mailjet-adapter** – utiliza o serviço do Mailjet, que permite o envio de 6.000 emails por mês de forma gratuita [48];
- **parse-server-sendgrid-adapter** – utiliza o SendGrid, que permite enviar 40.000 emails por um período experimental de 30 dias e 100 emails por dia (cerca de 3.000 emails por mês) permanentemente [49];
- **parse-server-genericemail-adapter** – *plugin* que permite encaminhar todos os emails enviados do Parse Server através de inúmeros fornecedores de email como o Gmail, Mailgun, Mailjet e SendGrid [50].

Notificações

Uma das funcionalidades mais importantes e que mais falta faz aos utilizadores do Parse Server é o envio de notificações. Esta funcionalidade, que se encontrava originalmente disponível no Parse, não se encontra implementada nativamente no Parse Server. Assim sendo, a comunidade responsável pelo projeto oferece algumas soluções para lidar com este problema:

- **parse-server-push-adapter** – possibilita o envio de notificações com a integração do serviço de qualquer fornecedor das mesmas. Ao abstrair a forma como as notificações são enviadas, pode ser facilmente conectado a qualquer serviço que exponha uma API para envio [51];
- **parse-server-fcm-push-adapter** – projeto cujo objetivo seria integrar o serviço Firebase Cloud Messaging (FCM) ou antigo Google Cloud Messaging (GCM) mas que se encontra estagnado, sem desenvolvimento;
- **parse-server-onesignal-push-adapter** – permite utilizar o serviço gratuito da OneSignal, que proporciona um número ilimitado de notificações e não apresenta restrições na plataforma à qual as mesmas se destinam;

- **parse-server-urban-airship** – utiliza o serviço da Urban Airship que, para além do envio de notificações, disponibiliza um vasto conjunto de funcionalidades como um centro de mensagens *user friendly* e uma tecnologia de localização que inclui *beacons*, geo-fencing e um histórico de localizações [52]. Apesar de disponibilizar uma solução com interface do utilizador, também possui uma edição alternativa sem a interface que garante o acesso às mesmas funcionalidades através de várias APIs [53].

Analytics

A funcionalidade de *Analytics* é particularmente útil e de grande interesse, uma vez que permite adquirir conhecimento através da análise das interações dos utilizadores, o que pode tornar-se vantajoso para qualquer negócio.

Como foi referido anteriormente, ainda que esta funcionalidade tenha estado disponível no Parse, o mesmo não se verifica no Parse Server. No entanto, existem algumas opções desacopladas do Parse Server que podem ser integradas para suprimir a ausência desta funcionalidade, como o Google Analytics[30], o Mixpanel[31] e o CleverTap[32].

O Google Analytics é uma solução gratuita que permite compreender a forma como as aplicações são utilizadas. Esta ferramenta proporciona a geração de relatórios, um maior auxílio no processo de tomada de decisão e uma melhor perceção acerca do comportamento dos seus utilizadores [54]; Esta solução dispõe ainda de uma interface do utilizador avançada e de bibliotecas e APIs organizadas em 4 elementos principais, sendo eles:

- **Recolha** – reúne os dados referentes a interações dos utilizadores;
- **Configuração** – permite controlar a forma como os dados são processados;
- **Processamento** – processa os dados das interações dos utilizadores com os dados de configuração;
- **Relatórios** – fornece acesso a todos os dados processados.

[30] https://developers.google.com/analytics/

[31] https://mixpanel.com/

[32] https://clevertap.com/

Da mesma forma, o Mixpanel fornece um *Product Analytics* que permite analisar de que forma os clientes interagem com determinado produto e estimular a evolução incremental para um produto melhor. *"Building a product people love is about measurement, not intuition. To know how people are using your product, you need a tool that can easily surface those insights."* [55]. Esta solução permite ainda perceber quais os clientes mais fiéis, qual a melhor forma de cativar novos clientes, como manter os clientes atuais por mais tempo e como lucrar mais com esses clientes. Em termos de preços, o Mixpanel dispõe de um plano gratuito, de um plano mensal (cerca de 121€ / mês) ou anual (cerca de 808€ / ano) e de um plano customizado.

O CleverTap, à semelhança do Mixpanel, possibilita analisar o comportamento dos utilizadores para interagir com estes da melhor forma possível, permitindo aplicar campanhas com abordagens mais pessoais e personalizadas. Identicamente ao Mixpanel, esta solução disponibiliza um plano gratuito, um plano mensal com contrato anual (cerca de 808€ /mês) e um plano customizado.

3.3.3 Parse Dashboard

O Parse Dashboard[33] começou por ser um componente da plataforma Parse, que com a sua transição para *open-source* tornou o Parse Dashboard num projeto *standalone*. Para o seu desenvolvimento é utilizado React, uma biblioteca JavaScript para a construção de interfaces do utilizador [56] e SCSS, uma sintaxe introduzida com o Sass 3 e que é compatível com CSS enquanto suporta todo o poder do Sass[34].

Como pode ser observado pela Figura 5, o Parse Dashboard é um projeto que permite gerir as aplicações que estejam a correr nas instâncias do Parse Server, oferecendo uma melhor visualização dos eventos que decorrem no contexto das mesmas. Ao examinar a Figura 5, podem ser identificadas três aplicações de teste – *Aplicação*, *MyApp* e *Teste* – cada uma a correr na sua própria instância do Parse Server. É ainda possível obter informações acerca do total de utilizadores e do número total de instalações que cada aplicação possui.

[33] https://github.com/parse-community/parse-dashboard/

[34] https://sass-lang.com/documentation/

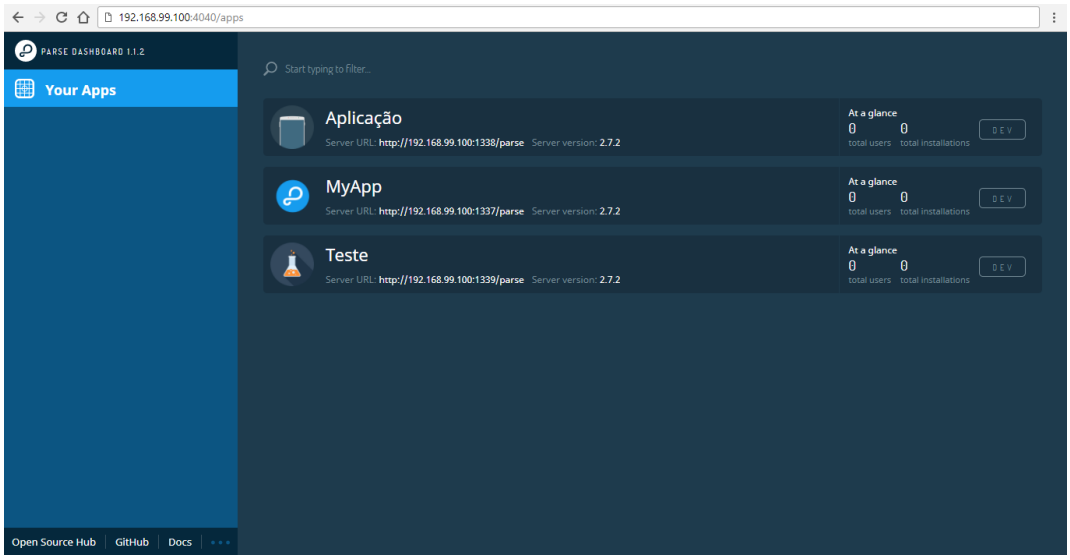


Figura 5 – Vista inicial do Parse Dashboard

Este projeto pode ser comparado a um painel intuitivo que permite visualizar, consultar e gerir todos os dados de diversas instâncias do Parse Server, o que pode ser observado na Figura 6, que representa a visualização dos dados e eventos associados à aplicação *MyApp*. Através da análise da Figura 6, podem ser observadas três classes existentes no contexto da aplicação *MyApp*, sendo elas as classes *Role*, *User* e *GameScore*. É ainda possível visualizar os registos das classes – a classe GameScore, por exemplo, apresenta quatro registos – e adicionar, editar e eliminar registos existentes, bem como adicionar novas classes à aplicação.

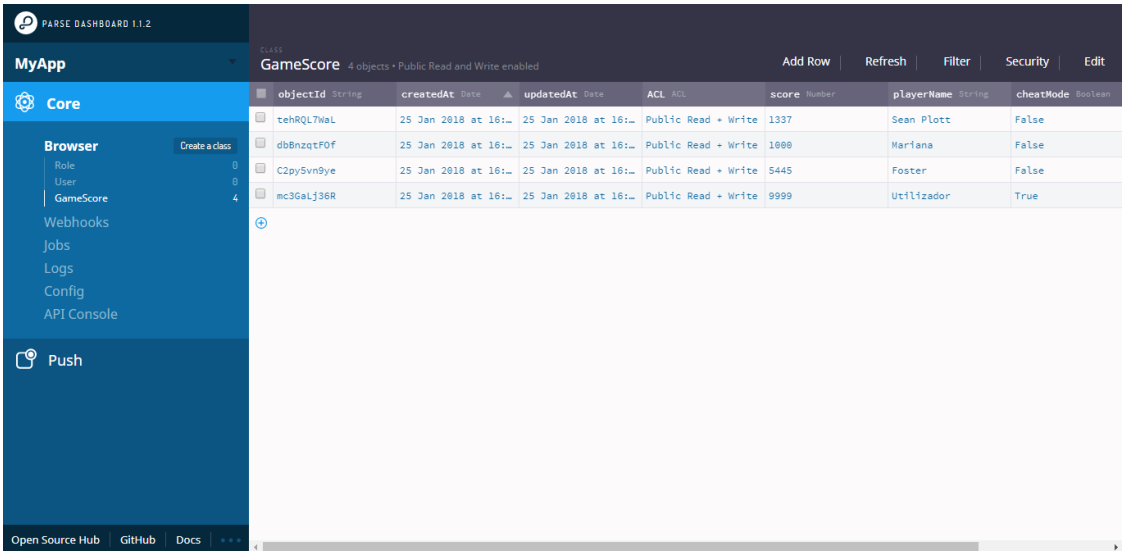


Figura 6 – Vista do Parse Dashboard no contexto de uma aplicação

Precedentemente ao encerramento do Parse e à respetiva migração para *open-source*, uma opção que se encontrava presente no menu lateral da *dashboard* era a da funcionalidade de *Analytics* que, como é possível observar pela figura anterior, não se encontra disponível no Parse Dashboard *open-source* (Figura 7). De acordo com a informação presente no ficheiro *DashboardView.react.js*, algumas páginas de *analytics* poderão nunca mais voltar a integrar o Parse Server novamente [57].

```
//These analytics pages may never make it into parse server
/*
if (...) {
  analyticsSidebarSections.push({
    name: 'Overview',
    link: '/analytics/overview'
  });
}

if (...) {
  analyticsSidebarSections.push({
    name: 'Explorer',
    link: '/analytics/explorer'
  });
}*/

//These ones might, but require some endpoints to added to Parse Server
/*
if (features.analytics && features.analytics.retentionAnalysis) {
  analyticsSidebarSections.push({
    name: 'Retention',
    link: '/analytics/retention'
  });
}
```

Figura 7 – Excerto do ficheiro *DashboardView.react.js* [57]

3.3.4 Estrutura de ficheiros do Parse Dashboard

De forma idêntica ao Parse Server, também o Parse Dashboard apresenta uma estrutura de ficheiros agrupada em diretórios, exibindo uma organização de ficheiros relacionados consoante as funcionalidades que os mesmos englobam ou implementam. Para além do diretório PIG, que diz respeito à Parse Interface Guide, podem ser destacados os diretórios: Parse-Dashboard, dashboard e *stylesheets* (Figura 8).

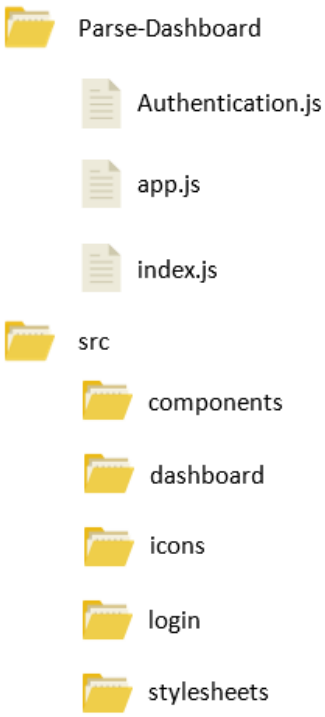


Figura 8 – Estrutura simplificada do Parse Dashboard

O diretório Parse-Dashboard possui o ficheiro Authentication.js, com os métodos necessários para suportar o processo de autenticação do lado do *frontend*. Os diretórios dashboard, login e stylesheets possuem os ficheiros responsáveis pela interface do utilizador, quer no que diz respeito à sua construção e visualização, quer no que se refere às interações que esta interface possibilita, aos resultados que são desencadeadas pela execução dessas interações e o processamento destes últimos.

3.3.5 SDKs e Bibliotecas

O Parse Server dispõe de algumas bibliotecas e SDKs[35] que permitem o acesso à plataforma Parse através de aplicações externas. Do conjunto de bibliotecas e SDKs disponíveis, salientam-se três de maior importância para o presente trabalho: o SDK para Android, o SDK para iOS/OS X/watchOS/tvOS e o SDK para JavaScript.

[35] http://parseplatform.org/#sdks

O SDK para Android é uma biblioteca que permite ter acesso ao Parse Server a partir de aplicações Android, podendo ser integrado com outros projetos do Parse para Android, como por exemplo o Parse UI[36], o Parse LiveQuery Client[37], o Parse Facebook Utils[38] e o Parse Twitter Utils[39]. O Parse UI contém duas bibliotecas, uma delas – ParseUI-Widget – disponibiliza *widgets* úteis para a interface do utilizador e a outra – ParseUI-Login – permite configurar o *look and feel* dos ecrãs de início de sessão das aplicações. Outro projeto passível de ser integrado com o SDK para Android é o Parse LiveQuery Client, ao disponibilizar uma ferramenta que permite subscrever uma ParseQuery e sempre que um ParseObject, que corresponda a essa ParseQuery, seja criado ou atualizado, o servidor irá notificar os clientes em tempo real. Esta ferramenta vem resolver o problema inerente ao ParseQuery, uma vez que este último é baseado num modelo *pull* e não tem suporte para *real-time*. Por fim, os projetos Parse Facebook Utils e Parse Twitter Utils permitem associar ParseUsers às suas contas de Facebook e Twitter, respetivamente.

O SDK para iOS/OS X/watchOS/tvOS pode ser traduzido como uma biblioteca que possibilita o acesso à plataforma Parse Server a partir de aplicações iOS, OS X, watchOS e tvOS. À semelhança do SDK para Android, este SDK pode ser integrado com outros projetos do Parse para iOS ou OS X, no entanto, em janeiro de 2018, grande parte desses projetos foram incorporados neste SDK. Desta forma, este SDK engloba agora projetos como o Parse Facebook Utils para iOS, o Parse Twitter Utils para iOS e o Parse UI para iOS. Por fim, da mesma forma que existe o Parse LiveQuery Client para Android, existe também o Parse LiveQuery Client para iOS e OS X[40]. Este projeto disponibiliza uma ferramenta que permite subscrever uma PFQuery e sempre que um PFObject, que corresponda a essa mesma PFQuery seja criado ou atualizado, o servidor irá notificar os clientes em tempo real. Esta ferramenta vem resolver o problema inerente ao PFQuery, uma vez que este último é baseado num modelo *pull* e não tem suporte para *real-time*.

O SDK para JavaScript diz respeito à biblioteca que permite o acesso à plataforma Parse Server através de aplicações JavaScript.

Para além dos SDKs mencionados, existem ainda outros SDKs como o SDK para Unity[41], .NET[42] e Xamarin[43]; o SDK para PHP[44], que possibilita o acesso à plataforma Parse Server a partir de aplicações ou scripts PHP; o SDK para Arduino[45], que fornece suporte para Arduino Yún e permite a construção de aplicações de IoT; e o SDK para Embedded C, que fornece suporte para Arduino Yún, Unix e plataformas RTOS C e permite construir aplicações de IoT.

36 https://github.com/parse-community/ParseUI-Android

37 https://github.com/parse-community/ParseLiveQuery-Android

38 https://github.com/parse-community/ParseFacebookUtils-Android

39 https://github.com/parse-community/ParseTwitterUtils-Android

40 https://github.com/parse-community/ParseLiveQuery-iOS-OSX

41 https://unity3d.com/pt

42 https://www.microsoft.com/net/

43 https://www.xamarin.com/

44 https://php.net/

45 https://www.arduino.cc/

4 PROJETO DE APLICAÇÃO (CASO DE ESTUDO)

Para demonstrar as funcionalidades implementadas e para testar a integração do Parse Server com uma aplicação Android, foi escolhido um cenário de aplicação real através da utilização da aplicação móvel Thumbeo. Esta aplicação faz parte do Thumbeo Corporate, um *software* de mobilidade (*ridesharing*) que pretende oferecer uma nova forma de fomentar a partilha de boleia entre viaturas de uma frota e/ou carros pessoais de pessoas que pertençam a uma determinada entidade (empresas, concelhos, associações, eventos, etc.) criando assim um sistema de gestão de viagens e reduzindo a pegada ecológica.

O *software* Thumbeo Corporate é constituído por 3 módulos principais: o CMS, o Core Processor e o Backend. O CMS possibilita que um administrador faça a gestão da informação relacionada com os utilizadores, frotas de viaturas e trajetos que irão ser efetuados. Este módulo tem o propósito de funcionar como uma plataforma de gestão e inserção de dados que posteriormente poderão ser "consumidos" do lado das aplicações móveis. O Core Processor tem o objetivo de funcionar como *middleware* e de estar responsável por receber os dados vindos do CMS, da análise de dados ou futuramente de outros *endpoints*, por processá-los e enviá-los estruturados para o Backend. Desta forma, é garantido que os dados irão chegar ao Backend sempre no mesmo formato. O Backend será onde os dados vindos do Core Processor e das aplicações irão ser armazenados, sendo preferencialmente um MBaaS para facilitar a integração com os vários serviços mobile necessário e também para ter suporte a *spatial queries*.

Para além destes 3 módulos principais, existe ainda um modulo responsável por analisar os dados que vão sendo criados (ou analisar base de dados de dados passados) de modo a ser possível prever trajetos futuros automaticamente. Por fim, as aplicações móveis constituem a interface que os utilizadores usarão para interagir entre si e o *backend*.

4.1 Enquadramento do projeto nas necessidades da empresa

Com o presente projeto, pretendia-se tirar partido da plataforma Parse *open-source* para que a mesma pudesse ser utilizada em aplicações já existentes na empresa ou para a elaboração de novas aplicações. Esta plataforma já contém todas as ferramentas básicas para o desenvolvimento *mobile*, dado que permite criar um *backend* escalável em poucos minutos e ainda fornece vários SDKs para Android, iOS, Javascript, entre outros.

Uma das principais vantagens do Parse é o facto de permitir a realização de *queries* por proximidade (*spatial queries*) ou dentro de um determinado raio, através dos Geo Points e das Geo Queries, o que acaba por ser bastante útil para aplicações que sejam *location aware*. Em comparação com a Firebase, por exemplo, o Parse já tem este tipo de *queries* integradas, ao contrário da Firebase que fornece uma biblioteca para efetuar esse tipo de *queries* aos seus dados, o GeoFire[46].

[46] https://github.com/firebase/geofire-java

4.2 Descrição funcional da aplicação Thumbeo

A aplicação Thumbeo irá permitir realizar um conjunto de ações, destacando-se entre elas a autenticação de utilizadores, a criação de pedidos de boleias, a criação de boleias, a gestão de disponibilidade de utilizadores e a *gamification*.

No Thumbeo, a autenticação e o acesso dos utilizadores à aplicação poderá ser feitas de duas maneiras distintas: caso a empresa ou entidade à qual a aplicação vá ser aplicada já possua um sistema de autenticação de utilizadores gerido pela própria empresa, esse sistema poderá ser integrado na aplicação e usado para validar o acesso dos utilizadores à aplicação (sendo imprescindível que os utilizadores acabem o seu registo através do preenchimento dos restantes dados necessários), a outra forma é fazendo o registo através da própria aplicação e aguardar a validação da conta por parte de um administrador (através do CMS).

Quanto à criação de pedidos de boleias, um utilizador registado poderá efetuar um pedido de boleia de modo a conseguir encontrar uma boleia que se enquadre nos seus parâmetros. Para efetuar este pedido, o utilizador terá de inserir o ponto de origem e destino, um intervalo de data, um intervalo de horas e o tipo de boleia (se é uma boleia de uma pessoa, de uma carga ou ambas). Após efetuar este pedido aparecerão os resultados que se enquadram na pesquisa feita pelo utilizador, sendo que neste momento o utilizador pode enviar um pedido para todos os resultados, enviar um pedido para um condutor específico ou telefonar diretamente a um condutor. Caso não apareça nenhum resultado para o pedido feito, o utilizador pode publicar essa boleia na secção do "mural" de modo a que, mais tarde, algum condutor envie um pedido de disponibilidade para realizar a boleia. Poderá ainda ser integrado *geofencing* e desta forma, se existir algum condutor que se aproxime daquela zona na data em que o pedido de boleia é suposto acontecer, o utilizador é notificado.

No que diz respeito à criação de boleias, para que um utilizador consiga criar uma boleia através da aplicação, necessita de ter pelo menos uma viatura registada ou de ser responsável por uma viatura. Em termos de informação, o utilizador necessita de inserir a origem e o destino da boleia, a data, hora, o carro que irá ser utilizada e, por fim, o número de lugares livres que terá na boleia. Para além disso, os dados das boleias poderão também ser carregados de outras formas: através de um formulário que existirá no CMS, através de um documento Excel estruturado e que poderá ser posteriormente carregado para o CMS e, por fim, através da análise dos dados de boleias passadas, sendo feitas estimativas de possíveis boleias que irão acontecer num dado período.

A funcionalidade de gestão de disponibilidade, para um condutor que tenha registado um veículo pessoal na aplicação, por exemplo, refere-se à possibilidade que esse utilizador tem de poder gerir a sua disponibilidade para efetuar uma boleia ou para receber pedidos de boleias, por não querer ser incomodado quando se encontra de férias ou fora do horário de expediente. Neste sentido, o utilizador poderá definir a sua disponibilidade como disponível ou indisponível, ou criar um conjunto de regras, ou seja, um conjunto de datas ou horários em que se encontrará disponível.

Por último, a introdução de *gamification* (Figura 9) é fundamental para incentivar os utilizadores e para os manter motivados em utilizar a aplicação. Neste sentido, será mostrado ao utilizador um conjunto de métricas (quilómetros percorridos, níveis de CO_2 não emitidos, dinheiro poupado) que serão calculadas através dos quilómetros feitos ou dados através de boleias e que irão gerar um conjunto de pontos que serão atribuídos ao utilizador. Consoante os pontos que o utilizador for acumulando, pode atingir diferente níveis. Esses pontos e níveis serão posteriormente comparado e apresentados segundo um *ranking* interno onde os utilizadores poderão ver a sua posição relativamente às restantes pessoas a utilizar a aplicação. Com a integração da *gamification* e com a criação do *ranking* entre utilizadores, a empresa ou entidade pode criar um modelo de recompensas para as pessoas que tenham atingido mais pontos ao final de um trimestre ou de um ano, o que iria incentivar ainda mais as pessoas a utilizar a aplicação.

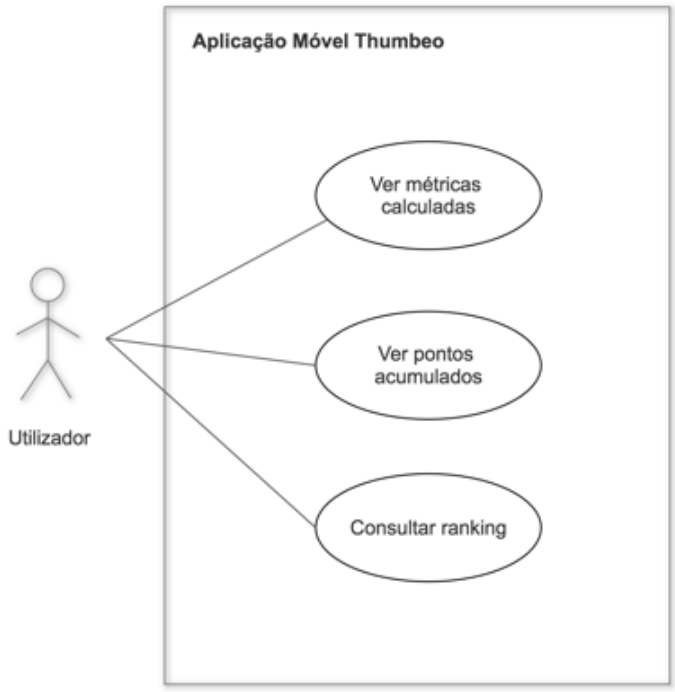


Figura 9 – Diagrama de casos de utilização para *gamification*

4.3 Requisitos não funcionais e restrições técnicas

Nesta secção são enumerados os requisitos não funcionais que são representativos de necessidades em aplicações móveis da empresa, nomeadamente na aplicação Thumbeo.

Em primeiro lugar, é apropriado referir que a aplicação Thumbeo corresponde a um projeto que já se encontrava em desenvolvimento, sendo necessário respeitar o UI/UX previamente planeado e desenvolvido. Tanto no caso do Thumbeo como em outros projetos da empresa, antecedentemente à etapa de criação das interfaces das aplicações são elaborados os *mockups* representativos dessas mesmas interfaces. Para esse efeito é utilizado o InVision[47] (Figura 10), uma ferramenta de *design* de soluções cuja principal função é garantir a melhor *user experience* e que permite criar *mockups* interativos de uma forma rápida e simples. Os *mockups* criados podem depois ser partilhados com o resto da equipa ou com os clientes e podem ser discutidos pelos mesmos através de comentários associados especificamente aos pontos que estão a ser discutidos [58]. Ao mimetizar uma *user experience* o mais aproximada possível da realidade, esta ferramenta é bastante benéfica pois permite que os clientes percebam facilmente o modo de funcionamento da solução que lhes será entregue e, por sua vez, possibilita que seja obtido *feedback* de maior qualidade.

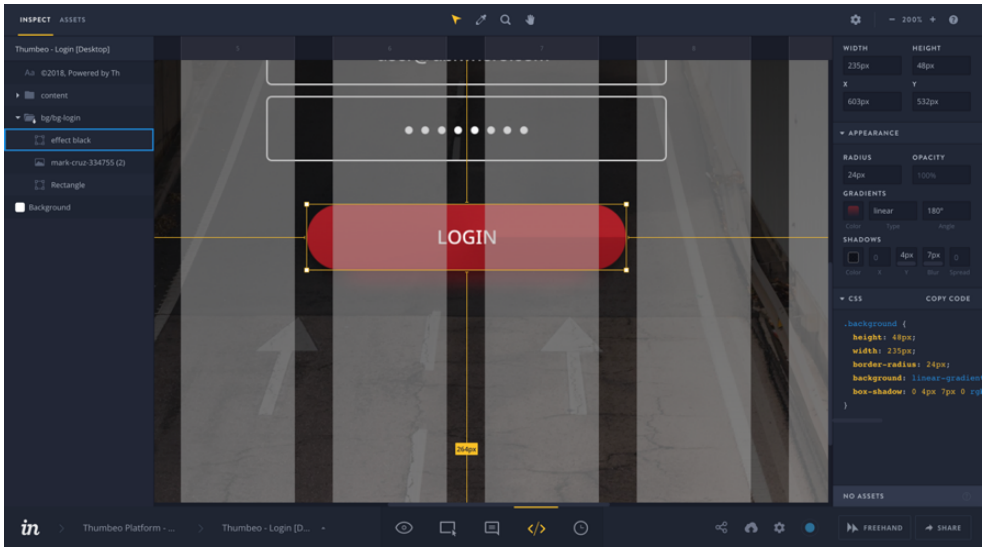


Figura 10 – Exemplo de utilização da ferramenta InVision

No que diz respeito à usabilidade, durante os vários processos que as aplicações englobam é sempre fornecido *feedback* ao utilizador relativo a todas as ações que o mesmo executa, isto é, fornece-se *feedback* ao utilizador quer seja na validação dos dados inseridos – através de mensagens de erro caso o utilizador tenha tentado submeter os dados sem ter inserido qualquer informação, caso os dados se encontrem no formato errado, etc. – quer seja no resultado de determinadas ações executadas, como por exemplo na criação, edição ou eliminação de um veículo. Deste modo, quer as ações de criação, edição e eliminação de um veículo tenham um resultado positivo, conforme o esperado, quer tenham um resultado negativo, pela ocorrência de situações de erro, é sempre apresentado *feedback* ao utilizador.

[47] https://www.invisionapp.com/

De forma a garantir o desempenho das aplicações, no caso concreto da aplicação Thumbeo, é utilizado o método **saveInBackground**, pertencente à classe **ParseObject** e que integra o SDK do Parse para Android. Pela utilização de um **ParseObject** – que deverá ter associado o nome da respetiva tabela à qual corresponde – é possível persistir informação no *backend* do Parse sem que seja necessário especificar previamente quais as chaves que o mesmo define [59]. Ainda no que diz respeito ao método **saveInBackground**, este procede à persistência de um **ParseObject** através da criação de uma *background thread*, contribuindo desta forma para que a *thread* principal não seja sobrecarregada, o que por sua vez se revela importante para um bom desempenho da aplicação. Este método recebe ainda um *callback* como argumento, executado no devido momento e enquanto os restantes processos decorrem normalmente e sem esperar pela resposta deste método.

Por último, também no que diz respeito ao desempenho e escalabilidade da aplicação, foi seguido o padrão de desenvolvimento MVP, que já se encontrava implementado desde o início do desenvolvimento da aplicação. Este padrão permite obter um projeto com um código mais organizado, escalável e testável, para além de oferecer algumas vantagens em comparação com outros padrões nomeadamente ao aplicar uma separação bem definida dos conceitos, uma preocupação bastante importante com o ganho de complexidade dos projetos.

O padrão MVP pode ser decomposto em 3 *layers* distintas: o Model, o Presenter e a View, como se encontra representado na Figura 11. Neste padrão, a *layer* Model corresponde estritamente a um modelo de domínio, sendo responsável pelos dados que são exibidos na UI e por qualquer lógica relacionada com a manipulação e o acesso aos dados. A *layer* Presenter, por sua vez, atua como intermediário entre as *layers* View e Model, sendo responsável não só por recolher os dados da *layer* Model e por retorná-los à *layer* View, como também por decidir o que ocorre quando o utilizador interage com a UI. Finalmente, neste padrão a *layer* View é responsável pela UI e pela manipulação dos eventos resultantes da interação com a mesma, o que lhe permite obter apresenta uma separação ainda maior da *layer* Model e, desta forma, uma separação bem definida dos conceitos [60].

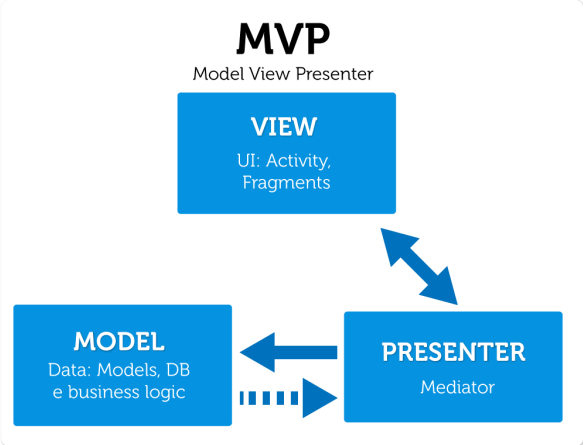


Figura 11 – Relações entre as *layers* do padrão MVP

Referência: https://goo.gl/aupa8u

5 ARQUITETURA PROPOSTA

Uma das tarefas de maior importância para qualquer projeto é a definição da sua arquitetura. Ao definir e planear determinadas características do projeto, como os componentes que o irão constituir e a forma como estes componentes irão interagir entre si, é possível identificar e corrigir determinados problemas com a devida antecedência e diminuir a probabilidade de vir a encontrar certos problemas no futuro.

5.1 Visão geral da arquitetura e novos desenvolvimentos

Neste subcapítulo é apresentada uma visão geral da arquitetura proposta, o que inclui todos os módulos e SDKs necessários para o correto funcionamento da solução. Uma vez que fazem parte do projeto duas aplicações distintas – a aplicação Android e o CMS – a informação apresentada e utilizada em cada um destes módulos é a mesma, uma vez que ambos partilham o mesmo serviço de *backend* (Figura 12).

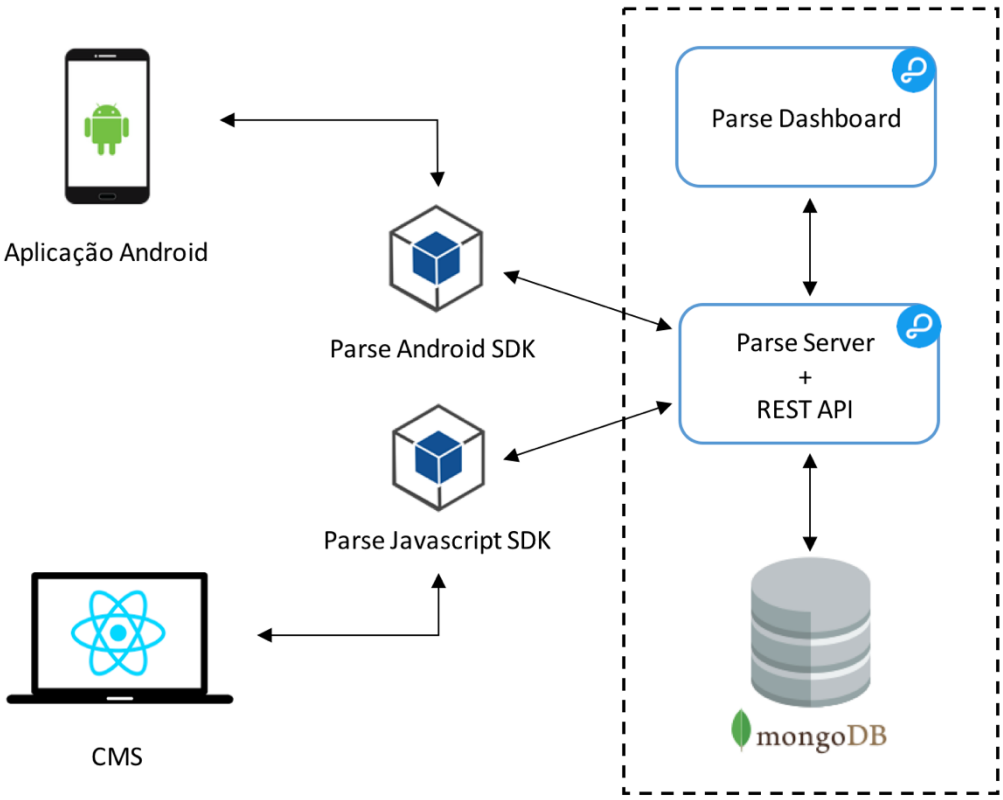


Figura 12 – Arquitetura geral proposta para a solução

5.2 Módulo Parse Server

O Parse Server pode ser descrito como o componente central da solução, uma vez que lida com operações de grande importância – nomeadamente com questões de segurança, autenticação e autorização – e que trata da comunicação entre outros componentes.

Através da REST API, cujo acesso é fornecido por meio do domínio para a instância do Parse Server, os componentes da solução podem interagir com o Parse Server pela realização de pedidos HTTP. Esta API permite que os componentes tenham acesso aos dados persistidos na base de dados, independentemente da linguagem de programação que utilizem na sua implementação. Para além da representação geral dos serviços que esta API fornece, ilustrados na Figura 13, alguns exemplos de serviços efetivos disponibilizados são:

- **login** – serviço responsável pelo *login* de um utilizador, através do seu *username* e *password* e que devolve um objeto JSON com todos os campos fornecidos pelo mesmo no momento do seu registo, com exceção da *password*;
- **logout** – elimina o objeto de sessão que por sua vez revoga o *token* de sessão e faz com o utilizador faça *logout* do dispositivo que está a utilizar aquele *token* de sessão;
- **installations** – disponibiliza os métodos GET e POST, responsáveis por devolver e por atualizar a informação relativa à classe **_Installations**;
- **installations/<objectId>** – disponibiliza os métodos GET, PUT e DELETE e através do objectId que recebe e do método em questão pode devolver, atualizar ou eliminar a informação do objeto Installations a que diz respeito.

Objects API

URL	HTTP Verb	Functionality
/parse/classes/<className>	POST	Creating Objects
/parse/classes/<className>/<objectId>	GET	Retrieving Objects
/parse/classes/<className>/<objectId>	PUT	Updating Objects
/parse/classes/<className>	GET	Queries
/parse/classes/<className>/<objectId>	DELETE	Deleting Objects

Figura 13 – Objects API [61]

5.3 Módulo Parse Dashboard

Este componente tem o propósito de facultar uma interface gráfica que permita visualizar as aplicações que se encontrem a correr nas instâncias do Parse Server, assim como analisar e gerir o conteúdo das mesmas. Desta forma, se forem efetuadas alterações aos dados presentes na base de dados, quer sejam feitas diretamente, através da aplicação Android ou do módulo CMS, este componente irá ser capaz de exibir essas mesmas modificações no contexto da aplicação em que foram feitas.

Para que consiga apresentar e alterar as informações da base de dados, este componente utiliza a REST API para fazer pedidos ao Parse Server, que responde afirmativamente aos pedidos caso o Parse Dashboard possua as permissões necessárias para aceder à informação requisitada, caso contrário nega o acesso ao conteúdo solicitado.

5.4 Módulo Android

O intuito da aplicação Android seria o de testar a sua integração com o *backend* do Parse, de forma a demonstrar a receção e o envio de dados da base de dados, a receção e o envio de notificações e o envio automático de *emails.*

Em vez de utilizar a REST API para as operações CRUD sobre os dados presentes na base de dados, este módulo utiliza o Parse SDK para Android, conforme recomendado [62]. Desta forma, a aplicação Android comunica com o Parse SDK, que por sua vez comunica com o Parse.

5.5 Módulo CMS

À semelhança do módulo Android, o objetivo do módulo CMS também seria o de testar a integração com o *backend* do Parse, para demonstrar a modificação e a receção dos dados que se encontram na base de dados do mesmo.

Para aceder e realizar modificações aos dados, este módulo utiliza o Parse SDK para JavaScript. Desta forma, este componente comunica com o Parse SDK correspondente, que por sua vez comunica com o Parse para realizar as operações desejadas.

6 IMPLEMENTAÇÃO E INSTALAÇÃO DA PLATAFORMA

Neste capítulo vão ser apresentados os aspetos relativos ao desenvolvimento de funcionalidades para plataforma – que compreende o Parse Server, o Parse Dashboard, a aplicação Android e o módulo CMS – e à instalação dos serviços da mesma em *containers*. Enquanto que no capítulo anterior foi introduzida a estrutura do projeto e de que forma os componentes do mesmo se encontram organizados, neste capítulo vão ser retratados todos os pormenores relativos ao processo de desenvolvimento da solução, nomeadamente os passos necessários para alcançar o resultado final.

6.1 Desenvolvimento do Parse Server

Para que algumas das alterações efetuadas ao Parse Dashboard pudessem ser executadas, foi necessário realizar também algumas modificações ao Parse Server, nomeadamente a respeito da persistência da informação referente às configurações das aplicações, que até então era apenas feita nos ficheiros *parse-dashboard-config.json* e *parse-server-config.json*. Desta forma, foi criada a tabela **_Configurations** na base de dados de cada aplicação, para que cada aplicação pudesse ter as suas configurações persistidas na sua própria base de dados. Para que isto fosse possível, foi necessário alterar o ficheiro **SchemaController –** que lida com a validação, persistência e modificação do *schema* da base de dados – onde foi definida a *system class* **_Configurations** e as colunas que a constituem por omissão, conforme se encontra representado na Figura 14. Cada coluna constituinte da classe **_Configurations** corresponde a um parâmetro dos ficheiros de configurações, sendo que uma grande maioria dos parâmetros adota o tipo de dados *String*, com exceção dos parâmetros **push** e **emailAdapter** que assumem o tipo de dados *Object* e do parâmetro **verifyUserEmails** que assume o tipo de dados *Boolean.*

```
const defaultColumns: {[string]: SchemaFields} = Object.freeze({
  // Contain the default columns for every parse object type (except _Join collection)
  _Default: { ···
  },
  // The additional default columns for the _Configurations collection (in addition to DefaultCols)
  _Configurations: {
    "appId":            {type:'String'},
    "appName":          {type:'String'},
    "masterKey":        {type:'String'},
    "clientKey":        {type:'String'},
    "javascriptKey":    {type:'String'},
    "restKey":          {type:'String'},
    "webhookKey":       {type:'String'},
    "windowsKey":       {type:'String'},
    "serverURL":        {type:'String'},
    "databaseURI":      {type:'String'},
    "push":             {type:'Object'},
    "publicServerURL":  {type:'String'},
    "verifyUserEmails": {type: 'Boolean'},
    "emailAdapter":     {type:'Object'}
  }
});

const systemClasses = Object.freeze(['_User', '_Installation', '_Role', '_Session', '_Configurations']);
```

Figura 14 – Integração da classe _Configurations no SchemaController

Com a classe **_Configurations** definida, o comportamento do Parse Server muda um pouco, pelo menos no que se refere à sua inicialização. Enquanto que anteriormente as configurações da instância do Parse Server eram carregadas na sua inicialização, a partir do ficheiro de configuração, foi alterado esse comportamento para que, caso existam configurações persistidas na base de dados, as mesmas sejam carregadas em vez das configurações que se encontram definidas no ficheiro de configuração. Por conseguinte, para conseguir as configurações mais recentes é utilizada a função **getParseServerConfigurations**, representada na Figura 15, que caso devolva resultados substitui os parâmetros carregados a partir do ficheiro pelos resultados obtidos da base de dados. Caso não tenham sido devolvidos resultados, são assumidos os parâmetros por omissão para a inicialização da instância, como pode ser observado na Figura 16.

```
dbo.collection("_Configurations").find().toArray(function(err, result) {
  if (err) {
    reject(err);
  } else {
    resolve(result);
    db.close();
  }
});
```

Figura 15 – Aquisição das configurações da base de dados

```
static start(options: ParseServerOptions, callback: ?()=>void) {
  configs.getParseServerConfigurations(options)
  .then(function(result) {
    if (result.length != 0) {
      let mostRecent = result.length - 1;
      Object.keys(result[mostRecent]).forEach((key) => {
        if (options[key] != result[mostRecent][key] && !key.startsWith('_')) {
          options[key] = result[mostRecent][key];
        }
      });
    }
    updateTemplateFiles(options);
    const parseServer = new ParseServer(options);
    return parseServer.start(options, callback);
  }, function(err) {
    console.error('The promise was rejected', err);
  }).catch(function(e) {
    console.log(e);
  });
}
```

Figura 16 – Função de inicialização do Parse Server

Para que as configurações definidas na base de dados pudessem ser atualizadas, foi necessário criar um novo Router, denominado **ConfigurationsRouter** e que estende o Router **ClassesRouter**. Como foi mencionado anteriormente, os ficheiros pertencentes ao diretório Routers definem as rotas que são disponibilizadas, sendo que neste caso, o ficheiro **ClassesRouter** define as rotas relacionadas com as classes presentes na base dados. Uma vez que as configurações são um dos parâmetros persistidos na base de dados, tendo sido criada a tabela **_Configurations** para esse efeito, a classe **ConfigurationsRouter** precisa de herdar o comportamento da classe **ClassesRouter**, para que seja possível realizar todas as operações definidas pela metodologia CRUD (create, read, update, delete) através do Parse Dashboard.

Um dos serviços que o **ConfigurationsRouter** define é o serviço **configurations**, que disponibiliza os métodos GET, POST E PUT (Figura 17) e que é responsável pelo envio e pela persistência da informação relacionada com os parâmetros de configuração do servidor, dos quais fazem parte as chaves da aplicação. O comportamento de todos os métodos disponibilizados por esta classe, para os serviços definidos pela mesma, é herdado do **ClassesRouter** com exceção do método PUT para o serviço **configurations**, que é manipulado pela reimplementação da função **handleUpdate** na classe **ConfigurationsRouter**. Esta função, para além de atualizar o registo de configurações mais recente na base de dados, atualiza também as variáveis **config** e **cache**, para que as modificações efetuadas às configurações da aplicação produzam efeito imediato, conforme demonstra a Figura 18.

```
mountRoutes() {
  this.route('GET','/configurations', req => { return this.handleFind(req); });
  this.route('GET','/configurations/:objectId', req => { return this.handleGet(req); });
  this.route('POST','/configurations', req => { return this.handleCreate(req); });
  this.route('PUT', '/configurations', ConfigurationsRouter.handleUpdate);
  this.route('PUT','/configurations/:objectId', req => { return this.handleUpdate(req); });
  this.route('DELETE','/configurations/:objectId', req => { return this.handleDelete(req); });
}
```

Figura 17 – Rotas do Router ConfigurationsRouter

```
for (let f in req.body) {
  config[f] = req.body[f];
  cache[f] = req.body[f];
  successes[f] = req.body[f];
}

rest.find(req.config, req.auth, '_Configurations', {}, {}, req.info.clientSDK).then((response) => {
  if (response.results.length == 0) {
    rest.create(req.config, req.auth, '_Configurations', successes, req.info.clientSDK);
  } else {
    let mostRecent = response.results[response.results.length - 1];
    rest.update(req.config, req.auth, '_Configurations', { objectId: mostRecent.objectId },
      successes, req.info.clientSDK);
  }
});

return {
  response: {
    successes
  }
};
```

Figura 18 – Função handleUpdate da classe ConfigurationsRouter

6.2 Desenvolvimento do Parse Dashboard

Relativamente ao Parse Dashboard, um projeto que oferece uma UI intuitiva e que torna possível a visualização e a gestão das aplicações que correm nas instâncias do Parse Serve, foram acrescentadas funcionalidades e realizadas algumas alterações.

O objetivo das funcionalidades acrescentadas é, sobretudo, o de simplificar determinadas tarefas que dispunham de uma complexidade desnecessária. Um exemplo de uma tarefa cuja complexidade era dispensável é a criação de novas aplicações que, para que as alterações efetuadas se refletissem na Dashboard, obrigava a que fossem efetuadas alteração no ficheiro de configuração *parse-dashboard-config.json* e precisava que o serviço fosse terminado e iniciado novamente, para que as configurações fossem novamente carregadas e as alterações pudessem ser refletidas na Dashboard. O mesmo acontecia com a alteração de *keys*, o que resultava numa discrepância entre as *keys* definidas no ficheiro de configuração da Dashboard e as *keys* definidas no ficheiro de configuração do Server e que podia constituir um problema maior se a *key* alterada fosse a *master key*, uma vez que iria originar obstáculos relacionados com acesso e autorização.

Quanto às alterações que foram efetuadas, os tópicos que lhes dizem respeito incluem sobretudo a configuração dos parâmetros necessários para o envio de notificações e a configuração do módulo e das opções para o envio de *emails*.

6.2.1 Criação de novas aplicações

Uma das funcionalidades solicitadas foi a possibilidade de criar novas aplicações diretamente a partir da Dashboard visto que, até então, só era possível adicionar novas aplicações à Dashboard através da alteração do ficheiro de configuração da mesma, carregado no momento de inicialização do serviço. Como tal, a Sidebar da vista inicial da Dashboard foi modificada de forma a incluir um novo SidebarSubItem, constituído por uma SidebarAction denominada “Create a new App” e que pode ser observada na Figura 19.

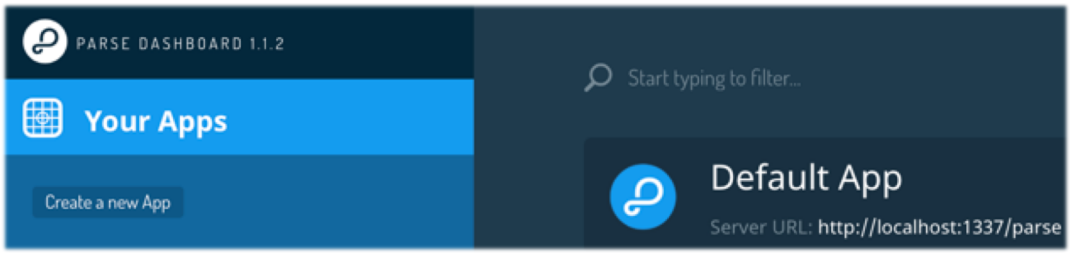


Figura 19 – SidebarAction "Create a new App"

Ao ser despoletada a SidebarAction compreendida pelo SidebarSubItem, é apresentado um Modal, representado na Figura 20, que abrange os campos necessários para a criação de uma aplicação, tais como o seu *application name*, *application id*, *master key*, *icon name*, *server URL* e outras *keys* que poderão ser futuramente úteis à aplicação. Para além disso, são aplicadas validações aos campos para que não seja possível criar uma aplicação cujos campos obrigatórios não tenham sido preenchidos, cujos campos *icon name* ou *server URL* se encontrem no formato errado ou caso os valores introduzidos para os campos *application name* e *application id* já estejam a ser utilizados por outra aplicação.

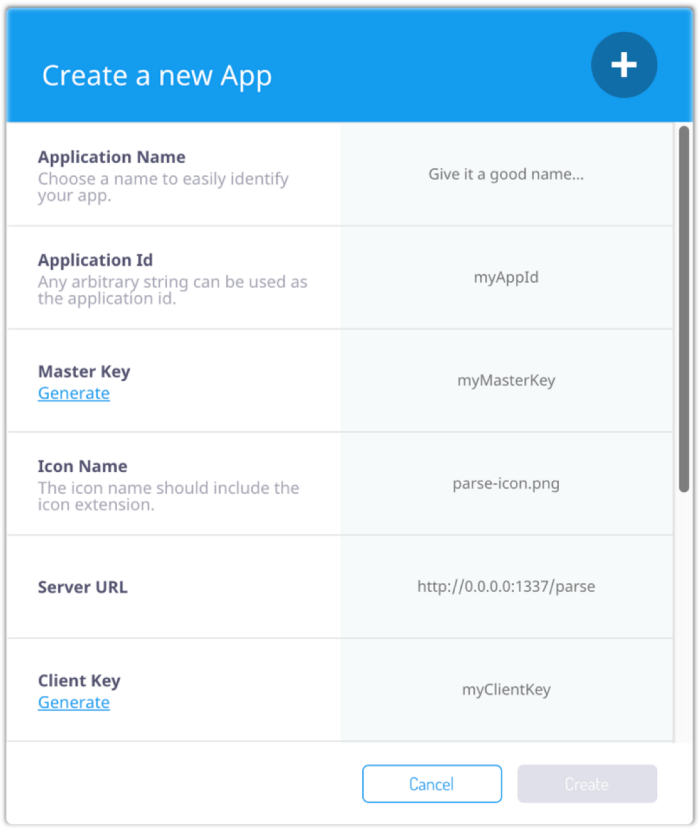


Figura 20 – Modal apresentado na criação de uma nova aplicação

Se todos os campos forem válidos, é feito um pedido *post* para atualizar a informação relativa à configuração da Dashboard. Para além de ser atualizada a configuração que se encontra em utilização, definida pela variável *config*, é ainda atualizado o ficheiro *json* que contém a configuração desatualizada, para que caso o serviço termine possa ser iniciado novamente com a configuração mais recente, que contém a nova aplicação criada. Caso o processo de criação de uma nova aplicação decorra sem erros, é apresentado um Modal de sucesso, caso contrário é apresentado um Modal indicativo de que decorreu um erro durante o processo.

6.2.2 Visualização e modificação de *keys*

Analogamente à criação de novas aplicações, só era possível visualizar e modificar as *keys* de cada aplicação através do ficheiro de configuração da Dashboard, carregado no momento de inicialização da mesma. Dado que a visualização e modificação de *keys* era uma funcionalidade que se encontrava disponível na Dashboard do antigo Parse e que deixou de fazer parte do Parse Dashboard quando este se tornou *open-source*, foi necessário acrescentar esta funcionalidade ao Parse Dashboard. Assim sendo, foram feitas alterações ao ficheiro *DashboardView.react.js* de maneira a que fosse possível ter acesso às *Settings* da aplicação a partir da Sidebar da Dashboard. Para além disso, foi criado o ficheiro *CustomSecuritySettings.react.js* baseado no ficheiro *SecuritySettings.react.js* e que engloba toda a estrutura necessária para que as *keys* da aplicação atual sejam enumeradas e para que seja possível alterar as *keys* desejadas, como pode ser observado na Figura 21.

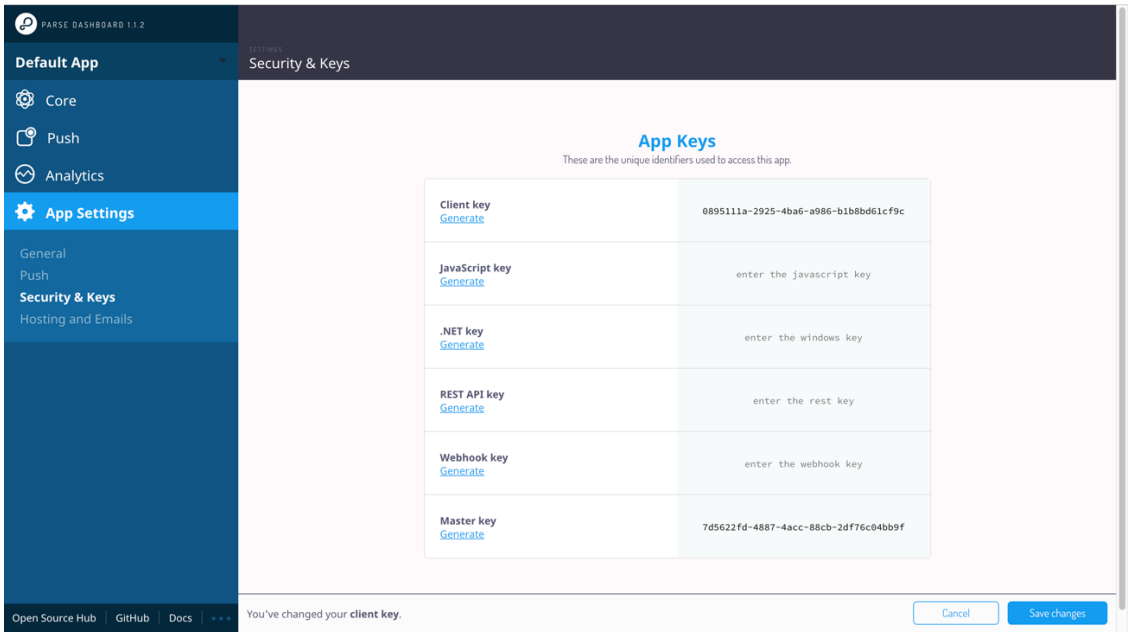


Figura 21 – Fieldset que enumera as keys de uma aplicação

Para além de ser constituída por um Fieldset composto por vários Fields, cada Field com a sua *key* correspondente, a estrutura do ficheiro *CustomSecuritySettings.react.js* inclui ainda a possibilidade de gerar chaves automaticamente através da opção "Generate" e compreende ainda as validações necessárias para não que seja possível submeter uma *master key* vazia, sendo esta a única chave de carácter obrigatório. Caso decorra a tentativa de submissão de uma *master key* vazia, é apresentada uma mensagem de erro e o utilizador fica impedido de submeter as alterações efetuadas (Figura 22).

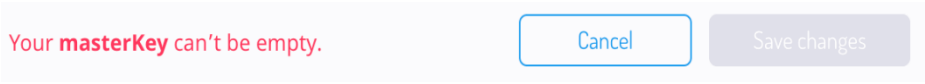


Figura 22 – Mensagem de erro apresentada para uma *master key* vazia

Como foi referido anteriormente, ao realizar alterações às *keys* passa a existir uma discrepância entre as *keys* ilustradas no Parse Dashboard e as *keys* definidas no Parse Server, o que pode constituir um problema se a *key* alterada for a *master key*. Ao existir uma discordância entre a *master key* definida no Parse Dashboard e a *master key* definida no Parse Server, deixa de ser possível aceder à aplicação a partir da Dashboard. Consequentemente, para resolver este problema, para além de serem atualizadas as chaves no ficheiro de configuração da Dashboard, também são atualizadas as chaves definidas do lado do servidor. Para tal é utilizada a função **apiRequest**, que tem a função de efetuar pedidos à API do Parse, para um determinado serviço e consoante o método explicitado. Esta função e a função **setParseKeys** podem ser visualizadas na Figura 23.

```
setParseKeys() {
  Parse.serverURL = this.serverURL;
  Parse._initialize(this.applicationId, this.javascriptKey, this.masterKey);
}

apiRequest(method, path, params, options) {
  this.setParseKeys();
  return Parse._request(method, path, params, options);
}
```

Figura 23 – Funções apiRequest e setParseKeys

Com a definição do serviço **configurations** no **ConfigurationsRouter**, referida anteriormente, torna-se possível utilizar a função **apiRequest** com o método PUT para atualizar as chaves do lado do servidor, como demonstra a Figura 24. Desta forma torna-se possível não só atualizar as chaves da aplicação, como atualizar a **master key** sem que se deixe de ter acesso à aplicação através da Dashboard.

```
  let promise = this.apiRequest(
    'PUT',
    'configurations',
    appFields,
    { useMasterKey: true }
  );
  promise.then(({ successes }) => {
    for (let f in fields) {
      if(this.hasOwnProperty(f) && this[f] != fields[f]) {
        this[f] = successes[f];
      }
    }
  });
  return promise;
}
```

Figura 24 – Pedido à Parse API para a atualização de chaves

6.2.3 Envio de notificações

Para o envio de notificações, à semelhança das funcionalidades apresentadas anteriormente, foi necessário alterar o ficheiro **DashboardView.react.js**, como demonstra a Figura 25, de maneira a que fosse possível ter acesso à opção **Push** a partir da Dashboard. Com esta opção disponível, torna-se possível ter acesso a três subopções: o envio de notificações, a visualização das notificações enviadas e a gestão de audiências.

```
let pushSubsections = [];

if (features.push && features.push.immediatePush) {
  pushSubsections.push({
    name: 'Send New Push',
    link: '/push/new'
  });
}

if (features.push && features.push.storedPushData) {
  pushSubsections.push({
    name: 'Past Pushes',
    link: '/push/activity'
  });
}

if (features.push && features.push.pushAudiences) {
  pushSubsections.push({
    name: 'Audiences',
    link: '/push/audiences'
  });
}
```

Figura 25 – Funcionalidade push no ficheiro DashboardView

Por omissão, o *adapter* utilizado para o envio de notificações é o **ParsePushAdapter** do módulo **parse-server-push-adapter** que, como foi referido anteriormente, abstrai a forma como as notificações são enviadas e pode ser facilmente conectado a qualquer serviço que exponha uma API para o envio de notificações. Este *adapter* utiliza o Apple Push Notifications Service para o envio de notificações para iOS e o Google Cloud Messaging para o envio de notificações para Android, no entanto, como o GCM foi descontinuado pela Google em Abril de 2018, as aplicações que utilizavam GCM tiverem de migrar para o Firebase Cloud Messaging, que herda a infraestrutura do GCM e lhe acrescenta novas funcionalidades [63].

A comunidade responsável pelo projeto *open-source* do Parse teve em conta a descontinuação do GCM e os pedidos de vários utilizadores para o desenvolvimento de um *adapter* para o FCM e decidiu acrescentar suporte para o FCM no Parse Server e no SDK do Parse para Android, passando este último a conter ambas as configurações dos serviços GCM e FCM [64]. Esta mudança implica que a configuração do lado do Parse Server passe a incluir uma chave FCM que corresponda ao **sender id** e **API key** para Android já existentes, o que irá garantir compatibilidade total com o FCM.

Para além disso, foi criado o ficheiro **CustomPushSettings.react.js**, baseado no ficheiro **PushSettings.react.js** e que abrange toda a estrutura necessária para que seja possível visualizar as credenciais relacionadas com o envio de notificações, definidas nos ficheiros de configuração. Como tal, na página definida por este ficheiro, são apresentados os parâmetros **sender id**, **API key** e um Toggle que indica se o envio de notificações do lado do cliente é permitido.

O envio de notificações do lado do cliente era uma funcionalidade que estava presente no Parse, no entanto deixou de fazer parte do projeto quando este se tornou *open source*, sobretudo por ser considerada uma funcionalidade pouco segura [65][66]. Para introduzir esta funcionalidade novamente na solução foi necessário, em primeiro lugar, defini-la no ficheiro de configuração do Parse Server. Consequentemente, o objeto **push** definido no ficheiro de configuração passou a conter o *boolean* **enableClientPush**, ilustrado na Figura 26, que indica se o envio de notificações do lado do cliente se encontra ativado.

```
"push": {
  "android": {
    "senderId": "83998201603",
    "apiKey": "AAAAE46u1wM:APA91bF-HYFtQh74Opjo0VXf4XIpNxXv_3GrrZ9W8",
    "enableClientPush": false
  }
}
```

Figura 26 – Parâmetro enableClientPush no ficheiro de configuração

Esta funcionalidade é considerada insegura dado que, para enviar notificações do lado do cliente, não é utilizada a **master key** da aplicação, mas sim a **cliente key**, uma chave pública. Tendo isso em conta, esta funcionalidade não iria funcionar sem que fossem alteradas determinadas verificações e validações aplicadas. Dado que não vai ser utilizada a **master key** e que são sempre feitas verificações de autorização para o envio de notificações – é sempre verificado que está a ser utilizada uma **master key** para a realização dessa tarefa – foi criado o *prototype* **canPush** no contexto do objeto **Auth**, que devolve *true* caso esteja a ser utilizada uma **master key** ou caso o envio de notificações do lado do cliente se encontre ativado.

Tendo em conta que o envio de notificações é efetuado para cada registo da tabela **Installations**, é necessário ter acesso à informação desta tabela para poder proceder ao envio das notificações. O acesso à tabela **Installations** encontrava-se restrito, visto que podia apenas ser acedido com a utilização de uma **master key**, no entanto, com a criação do método **canPush**, tornou-se possível garantir o acesso a esta tabela, exclusivamente para consultas, caso a opção do envio de notificações do lado do cliente se encontre ativada. Como pode ser observado pela Figura 27, o acesso à tabela **Installations** apenas é negado caso não esteja a ser utilizada uma **master key** para o método *delete* ou caso não esteja a ser utilizada uma **master key** para o método *find* com a opção do envio de notificações do lado do cliente desativada.

```
const classesWithMasterOnlyAccess = ['_JobStatus', '_PushStatus', '_Hooks', '_GlobalConfig', '_JobSchedule'];
// Disallowing access to the _Role collection except by master key
function enforceRoleSecurity(method, className, auth) {
  if (className === '_Installation' && !auth.isMaster) {
    if (method === 'delete' || (method === 'find' && !auth.canPush())) {
      const error = `Clients aren't allowed to perform the ${method} operation on the installation collection.`
      throw new Parse.Error(Parse.Error.OPERATION_FORBIDDEN, error);
    }
  }
```

Figura 27 – Acesso à tabela Installations no ficheiro rest.js

Para além do acesso à informação da tabela **Installations**, é também necessário ter em consideração o acesso ao serviço /push, que também se encontra condicionado pela utilização de uma **master key**. Tendo em conta que para as aplicações cliente é utilizada a **client key** e não a **master key**, para ter acesso a este serviço do lado do cliente foi alterada a função **promiseEnforceMasterKey**, que verifica a autenticação com **master key**, para que fosse possível autenticar com **client key** no caso de se tratar de um pedido de envio de notificações vindo do lado do cliente (Figura 28).

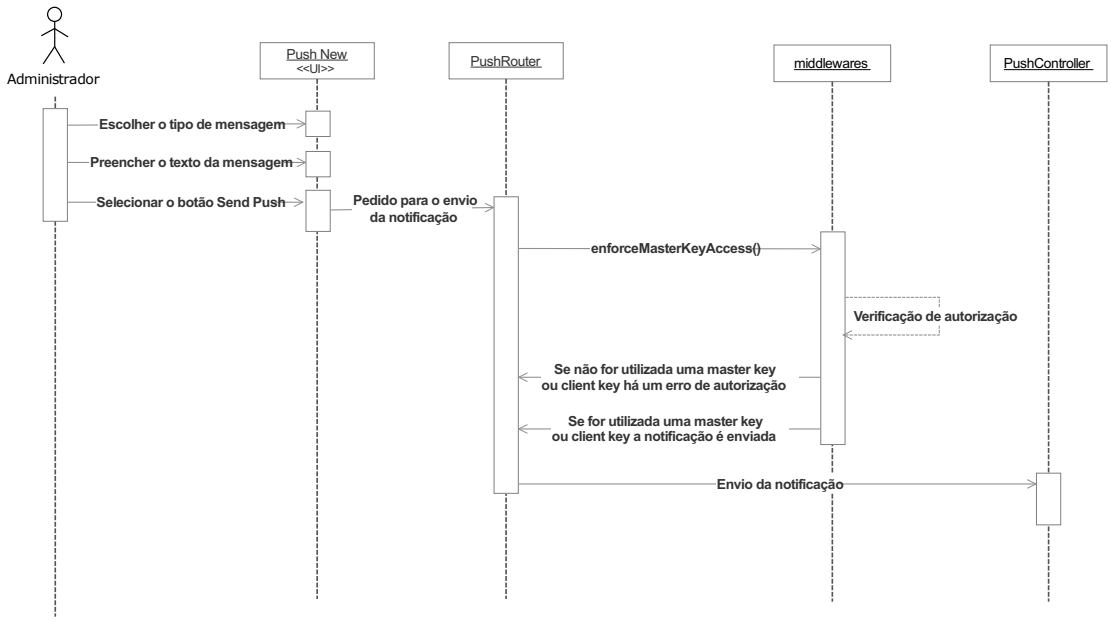


Figura 28 – Diagrama de sequência para o envio de notificações a partir da Dashboard

6.2.4 Envio de *emails*

Analogamente às funcionalidades já apresentadas, foi necessário alterar o ficheiro **DashboardView.react.js** para que fosse possível ter acesso à subopção **Hosting and Emails** a partir da Dashboard. Esta subopção é apresentada no contexto da opção **App Settings** e disponibiliza uma página que permite gerir os parâmetros e opções relativos ao envio de *emails* segundo o serviço Mailgun, definida pelo ficheiro **CustomHostingSettings.react.js**. Parte da representação dessa página pode ser observada na Figura 29, que exibe alguns parâmetros relativos à configuração do envio automático de *emails* através do Mailgun, como por exemplo:

- **Reply-to address** – endereço de *email* utilizado pela aplicação para o envio dos *emails* automatizados;
- **Domain** – domínio definido no Mailgun;
- **API key** – chave da API do Mailgun;
- **Verify user emails** – define se vai ser enviado um *email* de verificação após o registo de um utilizador e pode resultar na alteração do campo **emailVerified**.

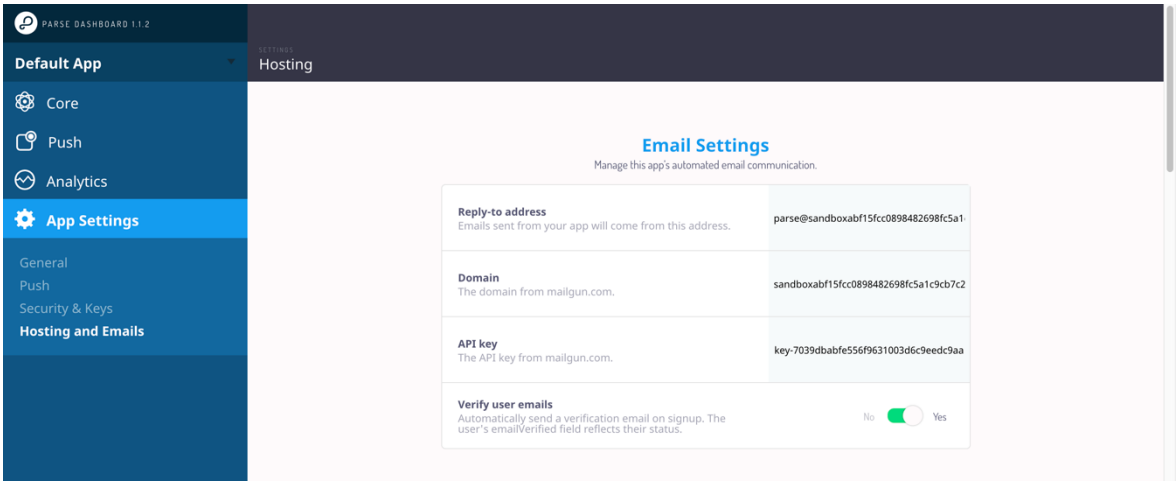


Figura 29 – Email Settings da subopção Hosting and Emails

Para proceder ao envio de *emails* não foi utilizado o **simple-mailgun-adapter,** mas sim o módulo **parse-server-mailgun-adapter-template** [67], por permitir a definição do conteúdo HTML dos *emails* através da Dashboard. Este módulo disponibiliza um conjunto de parâmetros e variáveis bastante úteis e que se aproximam da estrutura definida no ficheiro **HostingSettings.react.js** – no qual o **CustomHostingSettings.react.js** é baseado – e à abordagem seguida pelo antigo Parse.

O módulo **parse-server-mailgun-adapter-template** permite personalizar diversos campos relativos aos *emails*, como por exemplo o assunto e o corpo da mensagem, podendo o corpo da mensagem ser definido como texto simples ou como HTML. Caso o parâmetro relativo ao conteúdo HTML do corpo da mensagem tenha sido definido, é este que é assumo por omissão para o envio dos *emails*, caso contrário é assumido o parâmetro relativo ao conteúdo de texto simples para o corpo da mensagem. Estes parâmetros de configuração são apresentados e aplicados para os *emails* que dizem respeito à verificação de *email*, representados na Figura 30 e à reposição de *password*. Para além destes campos, o módulo permite ainda a utilização de variáveis pré-definidas, aplicadas ao assunto ou ao corpo dos *emails*, designadamente:

- **%username%** – o nome do utilizador ao qual *email* se destina;
- **%email%** – o *email* do utilizador ao qual o *email* se destina;
- **%appname%** – o nome da aplicação que é apresentado;
- **%link%** – o *link* para a verificação de *email* ou reposição de *password*.

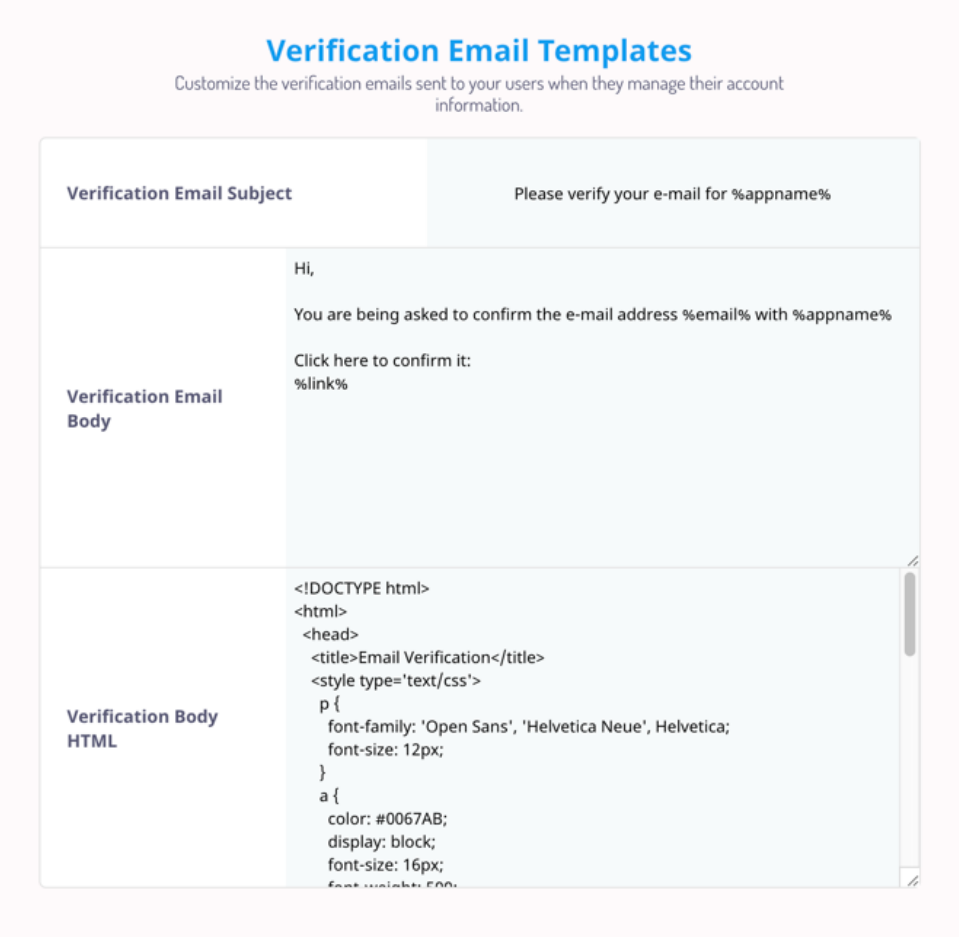


Figura 30 – Parâmetros relativos às mensagens de verificação de *email*

Para além da customização do conteúdo dos *emails*, é ainda possível personalizar outras páginas apresentadas ao utilizador, nomeadamente para a verificação de *email* e reposição de *password*. Assim sendo, a subopção **Hosting and Emails** disponibiliza ainda os campos necessários para a customização de três páginas HTML: a página para a escolha da nova *password* (quando o utilizador clica no *link* de reposição de *password*), a página apresentada quando a *password* foi redefinida com sucesso e a página apresentada no decorrer da verificação do *email*. Estes três campos podem ser visualizados na Figura 31.

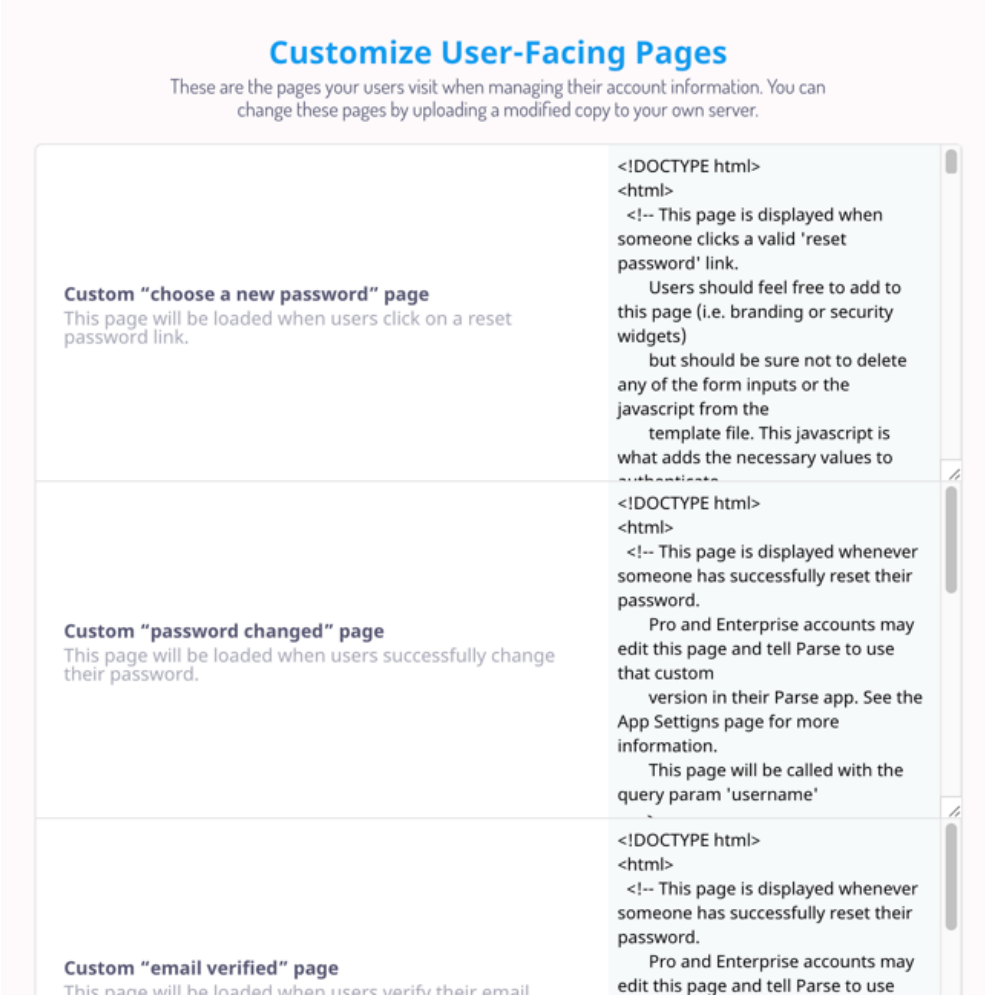


Figura 31 – Páginas apresentadas aos utilizadores no contexto dos *emails* que foram enviados

Toda a informação relativa a páginas HTML que a subopção **Hosting and Emails** disponibiliza encontra-se persistida em ficheiros do Parse Server, que podem ser encontrados nas diretorias **public_html** e **views**. Estes ficheiros são utilizados por omissão para representar as páginas a que correspondem, a menos que o seu conteúdo tenha sido alterado através da Dashboard e que a base de dados possua o conteúdo mais recente. Neste caso, a informação que vai ser apresentada na Dashboard, nos *emails* e nas páginas exibidas ao utilizador vai ser a informação que se encontra na base de dados.

Tendo em consideração o *adapter* escolhido, foi necessário criar uma conta para o Mailgun. Este serviço permite a configuração de um domínio próprio ou a utilização de um domínio *sandbox*, exclusivamente para testes. Se for utilizado um domínio próprio, para além de ser imprescindível utilizar um *DNS provider*, têm de ser configurados alguns registos – disponíveis no painel de controlo – no contexto do mesmo, para que seja possível verificar a propriedade do domínio indicado. Por outro lado, os domínios *sandbox* são restritos e para os utilizar é necessário adicionar os endereços de *email* dos destinatários a uma lista de destinatários autorizados, que permite indicar no máximo 5 destinatários e protegê-los de *spam*. Ao adicionar o primeiro destinatário à lista é criado um domínio *sandbox*, com parâmetros que vão ser úteis para fazer a ligação com o Parse Server, nomeadamente o parâmetro **API Key**, exibido na Figura 32.

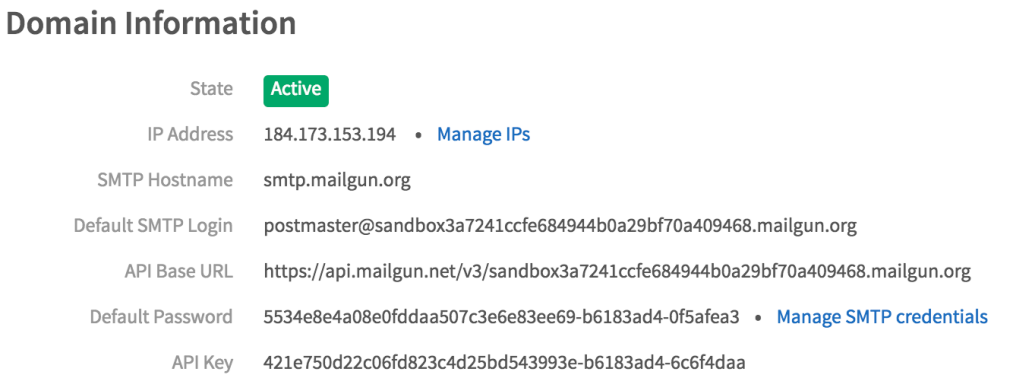


Figura 32 – Informações do domínio de sandbox criado

Para utilizar este serviço é necessário especificar o módulo que é utilizado e alguns parâmetros relativos ao Mailgun no ficheiro de configuração do Parse Server, como se encontra representado na Figura 33. Para além disso, é indispensável indicar os parâmetros **verifyUserEmails** e **publicServerURL**, caso contrário o envio não funcionaria.

```
"emailAdapter": {
  "module": "parse-server-mailgun-adapter-template",
  "options": {
    "fromAddress": "parse@sandboxabf15fcc0898482698fc5a1c9cb7c2ab.mailgun.org",
    "domain": "sandboxabf15fcc0898482698fc5a1c9cb7c2ab.mailgun.org",
    "apiKey": "key-7039dbabfe556f9631003d6c9eedc9aa"
  }
},
"verifyUserEmails": true
}
```

Figura 33 – Parâmetro emailAdapter no ficheiro de configuração do Parse Server

O resultado do envio dos *emails* de verificação pode ser visualizado na Figura 34, que demonstra uma circunstância na qual o parâmetro relativo ao conteúdo HTML do corpo da mensagem se encontra definido e na Figura 35, que ilustra uma circunstância em que foi utilizado o texto por omissão para o envio do *email* apresentado.

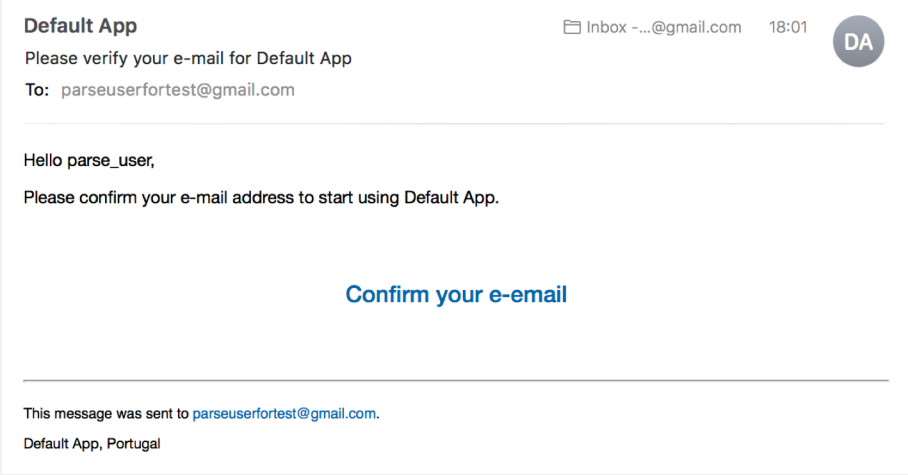


Figura 34 – Exemplo de um *email* definido pelo parâmetro Verification Body HTML

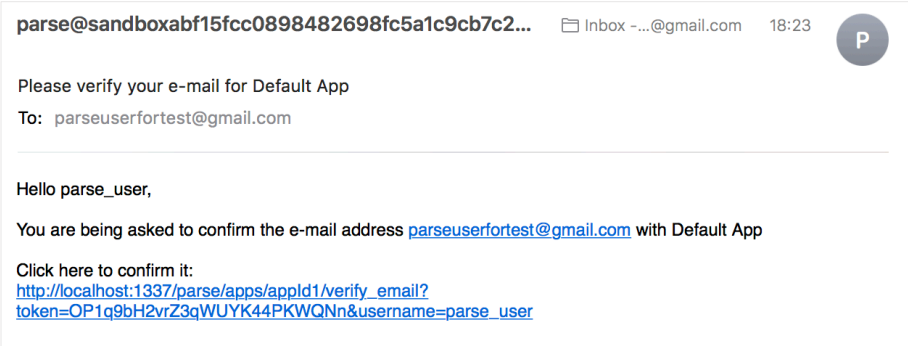


Figura 35 – Exemplo de um *email* definido pelo parâmetro Verification Email Body

Ao clicar no *link* de confirmação indicado no *email* podem ocorrer duas situações: o *email* é verificado com sucesso ou o *link* de verificação é considerado invalido e é apresentada a opção de reenviar um *link* de confirmação. Estas duas hipóteses encontram-se representadas na Figura 36.

Figura 36 – *Email* verificado com sucesso e *link* de verificação inválido

O Mailgun dispõe de uma *dashboard* que permite visualizar as informações relativas à entrega de *emails* e que se encontra representada na Figura 37. Para além disso e como foi referido anteriormente no capítulo relativo aos

Parse Server Modules, o serviço fornecido pelo Mailgun proporciona ainda a aquisição de informações avançadas acerca do desempenho dos *emails* e consoante determinados critérios, como o país ou o tipo de dispositivo no qual o *email* foi recebido. Desta forma, torna possível fazer certas previsões e proceder ao envio de *emails* nos momentos mais proveitosos.

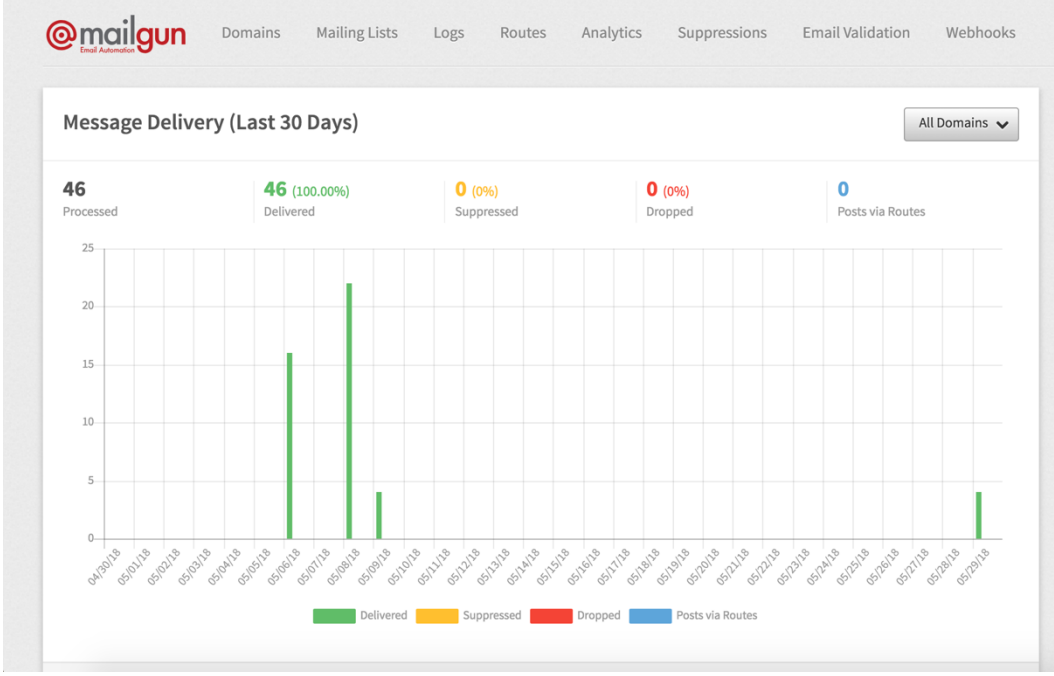


Figura 37 – Dashboard do Mailgun que exibe a entrega de mensagens nos últimos 30 dias

6.2.5 Analytics

Uma funcionalidade de grande utilidade para qualquer projeto no qual se procure compreender a forma como os utilizadores interagem com determinada aplicação é a funcionalidade das *analytics.* Esta funcionalidade possibilita a obtenção de um melhor entendimento acerca do comportamento dos utilizadores, o que por sua vez permite planear estratégias específicas conforme as conclusões retiradas. Apesar de ter estado integrada no antigo Parse, a comunidade atualmente responsável pelo projeto afirma que algumas páginas relativas às *analytics* poderão nunca mais voltar a integrar o mesmo, como foi explicado na secção 3.3.3 – Parse Dashboard.

Para colmatar a ausência desta funcionalidade, foi integrado o serviço *open source* da Metabase[48] na Dashboard. Consecutivamente, o ficheiro **DashboardView.react.js** foi alterado para que fosse possível ter acesso à opção **Analytics** e foi criado o ficheiro **CustomAnalytics.react.js**, que incorpora a *dashboard* criada a customizada na Metabase com o Parse Dashboard. Através da especificação da base de dados da aplicação nas propriedades da Metabase, este serviço tem acesso aos dados da aplicação e possibilita aplicar *queries* aos mesmos. Consoante o tipo de dados e as *queries* aplicadas, é possível gerar gráficos adequados ao contexto e, desta forma, obter informações pertinentes acerca de determinada questão ou assunto.

A Figura 38 demonstra os gráficos possíveis de gerar com a Metabase, consoante a informação presente na tabela **Installations** de uma aplicação que se encontra a correr no Parse Server. A partir dos gráficos gerados consegue-se perceber quais as regiões e os dispositivos nos quais a aplicação foi mais vezes instalada e o número de utilizadores disponíveis em comparação com os que já não se encontram disponíveis.

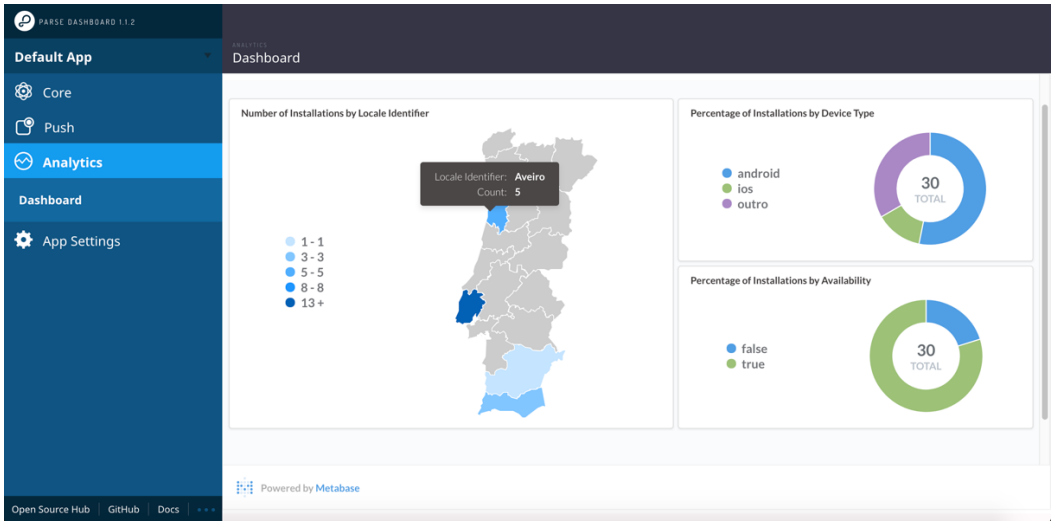


Figura 38 – Analytics segundo o serviço disponibilizado pela Metabase

6.3 Desenvolvimento da aplicação Android

Quanto ao desenvolvimento de funcionalidades para a aplicação Android, apresentada na secção 4, foi utilizado o Android Studio[49] que por sua vez utiliza o Gradle[50], uma ferramenta de automação responsável pela gestão de versões e das dependências do projeto.

[48] https://www.metabase.com/

[49] https://developer.android.com/studio/

[50] https://gradle.org/

Em primeiro lugar, começou-se por integrar o Parse Android SDK 1.16.3 com a aplicação Thumbeo – tendo em conta que este SDK funciona apenas para o Android 2.3 ou versões superiores – e foram desenvolvidas as funcionalidades de criação, edição e remoção de veículos. Para que fosse possível ter a aplicação a comunicar com o Parse Server, primeiro foi necessário garantir que o projeto permite adicionar dependências de repositórios Maven[51], como indica o bloco **repositories** do ficheiro **build.gradle**, representado na Figura 39. Seguidamente foi adicionada a dependência do Parse Android SDK ao bloco **dependencies**, também presente no ficheiro **build.gradle**, como se encontra exposto na Figura 40.

```
repositories {
    google()
    jcenter()
    mavenCentral()
    maven { url 'https://maven.google.com' }
}
```

Figura 39 – Bloco repositories do ficheiro build.gradle

```
dependencies {

    //Parse
    compile 'com.parse:parse-android:1.16.3'
```

Figura 40 – Junção do Parse Android SDK à lista de dependências

Para além disso, foi ainda necessário garantir que a aplicação tinha permissões de acesso à internet, definidas no ficheiro **AndroidManifest.xml** e exibidas na Figura 41. A este ficheiro, foram ainda acrescentadas as credenciais necessárias para a inicialização do Parse e para a conexão com o Parse Server, tendo sido adicionadas à *tag* <**application**> recorrendo a componentes do tipo <**meta-data**>, cujos valores foram depois definidos no ficheiro **strings.xml**. Desta forma, o ficheiro **strings.xml** passou a incluir os parâmetros relativos ao URL do servidor, ao id da aplicação e à **client key**. Os valores definidos nas *tags* <**meta-data**> são reunidos num único objeto **Bundle** e são posteriormente disponibilizados.

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
```

Figura 41 – Permissões para a aplicação aceder à internet

As credenciais definidas no ficheiro **AndroidManifest.xml**, em componentes do tipo <**meta-data**>, podem ser observadas na Figura 42 e a definição dos valores das mesmas, no ficheiro **strings.xml**, pode ser visualizada na Figura 43.

[51] https://maven.apache.org/

```
<meta-data
    android:name="com.parse.SERVER_URL"
    android:value="http://10.0.2.2:1337/parse" />
<meta-data
    android:name="com.parse.APPLICATION_ID"
    android:value="0rF4sRxn9WUDGrAV9nziFygodJrpHehe7ouwkQBZ" />
<meta-data
    android:name="com.parse.CLIENT_KEY"
    android:value="JRvWNxUcn7C1u9IuE6aWr6dtLmjYADYZSd4X99hP" />
```

Figura 42 – Componentes <meta-data> definidos no AndroidManifest.xml

```
<!--Parse-->
<string name="parse_server_url" translatable="false">http://10.0.2.2:1337/parse</string>
<string name="parse_app_id" translatable="false">appId1</string>
<string name="parse_client_key" translatable="false">c577dc77-876a-420a-9f69-e83e87f3ae22</string>
```

Figura 43 – Credenciais do Parse definidas no ficheiro strings.xml

Após todas as configurações feitas, torna-se possível inicializar o Parse na aplicação, existindo a classe ParseManager para o efeito. Para além disso, esta classe ainda lida ainda com o registo das tabelas constituintes da aplicação, com a persistência da instalação atual na tabela **_Installations**, com o *refresh* de todos os dados relacionados com determinado utilizador e com a gestão das *settings* da aplicação e integração das mesmas com o Parse. Para proceder à inicialização do Parse é necessário fornecer à função **initialize()** um objeto **Context**, que por sua vez irá ser utilizado na inicialização de um objeto **Builder**. Este objeto **Builder** acede aos parâmetros definidos nas *tags* **<meta-data>** do ficheiro **AndroidManifest.xml** através de um objeto **Bundle**, que reúne os valores definidos nas *tags*, como pode ser observado na Figura 44.

```
private static final String PARSE_SERVER_URL = "com.parse.SERVER_URL";
private static final String PARSE_APPLICATION_ID = "com.parse.APPLICATION_ID";
private static final String PARSE_CLIENT_KEY = "com.parse.CLIENT_KEY";

public Builder(Context context) {
  this.context = context;

  // Yes, our public API states we cannot be null. But for unit tests, it's easier just to
  // support null here.
  if (context != null) {
    Context applicationContext = context.getApplicationContext();
    Bundle metaData = ManifestInfo.getApplicationMetadata(applicationContext);
    if (metaData != null) {
      server(metaData.getString(PARSE_SERVER_URL));
      applicationId = metaData.getString(PARSE_APPLICATION_ID);
      clientKey = metaData.getString(PARSE_CLIENT_KEY);
    }
  }
}
```

Figura 44 – Acesso às configurações definidas no Manifest a partir do Parse

Para testar a integração do Parse com a aplicação Thumbeo, foram desenvolvidas as funcionalidades de criação, edição e remoção de veículos no contexto da mesma.

Para a criação de um novo veículo foi criada a função **addVehicle**, que recebe um objeto **Vehicle** e o adiciona à lista local de veículos, para depois persistir o objeto na base de dados do Parse através da função **saveInBackground()**, conforme representado na Figura 45.

```
public void addVehicle(Vehicle vehicle, ISaveHandler<Vehicle> callback) {
    listCarsUser.add(vehicle);

    vehicle.saveInBackground(e -> {
        if (e == null) {
            callback.onSuccess(vehicle);
            return;
        }
        callback.onError(e.getLocalizedMessage());
    });
}
```

Figura 45 – Função de adicionar um veículo

No que diz respeito à edição de um veículo foi criada a função **editVehicle**, que recebe um objeto **Vehicle** ao qual foram efetuadas alterações e percorre a lista local de veículos até encontrar um veículo com um **objectId** que corresponda ao **objectId** do veículo que foi modificado, sendo esse veículo substituindo na lista pelo veículo atualizado. O mesmo é aplicado no Parse, sendo procurado na base de dados um veículo com um **objectId** que corresponda ao **objectId** do veículo que foi editado, através de uma ParseQuery e da função **getInBackground()** e caso seja encontrada uma correspondência, o novo veículo é persistido através da função **saveInBackground()**, conforme representado na Figura 46.

```
public void editVehicle(Vehicle vehicle, ISaveHandler<Vehicle> callback) {
    Iterator<Vehicle> iterator = listCarsUser.iterator();
    while (iterator.hasNext()) {
        Vehicle v = iterator.next();

        if(v.getObjectId().equals(vehicle.getObjectId())) {
            listCarsUser.set(listCarsUser.indexOf(v), vehicle);
            break;
        }
    }

    ParseQuery<Vehicle> vehicleParseQuery = Vehicle.getQuery();
    vehicleParseQuery.getInBackground(vehicle.getObjectId(), (object, e) -> {
        if (e == null) {
            vehicle.saveInBackground();
            callback.onSuccess(vehicle);
            return;
        }
        callback.onError(e.getLocalizedMessage());
    });
}
```

Figura 46 – Função de editar um veículo

Para remover um veículo existente foi criada a função **removeVehicle**, que recebe um objeto correspondente ao **Vehicle** que deve ser eliminado e remove-o da lista local de veículos. Após ter sido removido da lista local de veículos, o veículo é eliminado da base de dados do Parse, através da função **deleteInBackground()**, como pode ser observado na Figura 47.

```java
/**
 * Delete a vehicle from the backoffice
 *
 * @param vehicle  vehicle object
 * @param callback callback
 */
public void removeVehicle(Vehicle vehicle, ISaveHandler<Vehicle> callback) {
    listCarsUser.remove(vehicle);

    vehicle.deleteInBackground(e -> {
        if (e == null) {
            callback.onSuccess(vehicle);
            return;
        }
        callback.onError(e.getLocalizedMessage());
    });
}
```

Figura 47 – Função de remover um veículo

6.4 Desenvolvimento do Módulo CMS

No que diz respeito ao *Content Management System* foi utilizado React e Redux, previamente introduzidos na secção 2.2 – Desenvolvimento de aplicações web com React.

Em primeiro lugar, foi necessário integrar o SDK do Parse para JavaScript com o CMS, para que fosse possível ter a aplicação a comunicar com o Parse Server. Como tal, foi adicionada a respetiva dependência ao ficheiro **package.json**, como pode ser observado na Figura 48.

```
},
"dependencies": {
  "element-react": "^1.4.19",
  "element-theme-default": "^1.4.13",
  "history": "^4.7.2",
  "immutable": "^3.8.2",
  "kea": "^0.26.6",
  "kea-saga": "^0.2.1",
  "lodash": "^4.17.4",
  "moment": "^2.19.2",
  "object-assign": "4.1.1",
  "object-to-formdata": "^1.0.13",
  "opensans-npm-webfont": "^1.0.0",
  "parse": "1.11.0",
  "path": "^0.12.7",
  "path-to-regexp": "^2.1.0",
```

Figura 48 – Ficheiro package.json presente no CMS

Visto que foi desenvolvida a funcionalidade de *login*, o mais adequado para o desenvolvimento desta funcionalidade seria utilizar os módulos direcionado para aplicações *server-side* e, como tal, foi utilizado o *package* **parse/node**.

Para esta funcionalidade, foi criada uma página em React, ilustrada na Figura 49, que através dos campos do **email** e **password** procede ao *login* do utilizador, caso a informação introduzida para estes dois campos corresponda à informação de algum utilizador registado na base de dados do Parse, com as permissões necessárias para aceder ao CMS.

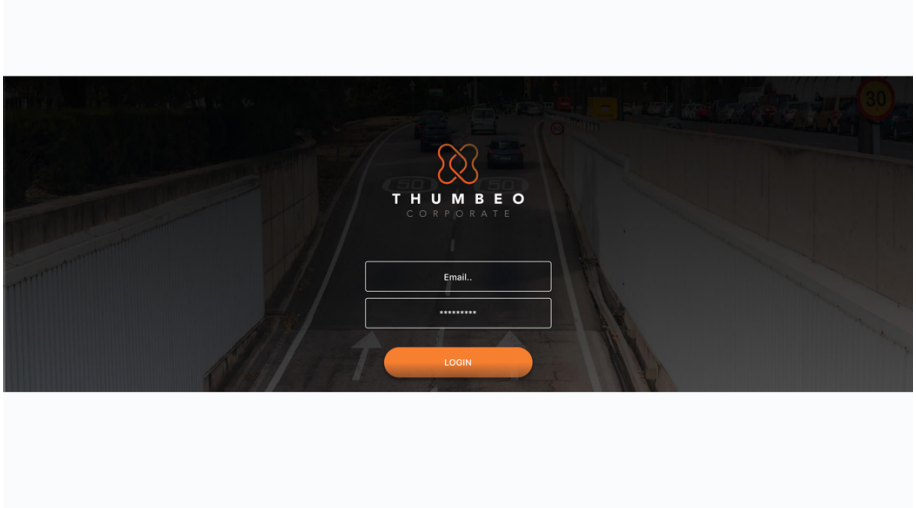


Figura 49 – Página de login do CMS

No ficheiro que contém a lógica referente ao processo de *login*, foi criado um *worker* que para cada tentativa de submissão das credenciais de acesso verifica que os campos introduzidos são válidos (não se encontram vazios e o campo relativo ao **email** apresenta a estrutura correta) e procura uma correspondência na base de dados para os dados inseridos, como pode ser observado na Figura 50. Caso seja encontrada uma correspondência o utilizador é redirecionado para a Dashboard do CMS, caso contrário seria apresentada uma mensagem de erro.

```
workers: {

  * submit (action) {
    const { navigate, reset } = this.actions
    const form = yield this.get('form')

    // Check validations
    const validation = Check.checkValidation(form, VALIDATIONS)

    if (validation.invalid) {
      yield put(reset())
      return
    }

    // Transform object and remove uneeded state values
    let params = mapValues(form, ({ value }) => value)

    try {
      yield delay(1000)
      let result = yield findUser('email', params.email)
      yield login(result, params.password)
      yield put(navigate('/'))
      yield put(reset())
    } catch (error) {
      console.log(JSON.stringify(error))
      yield put(reset())
    }
  }
}
```

Figura 50 – *Worker* criado para cada ação de *submit*

Para realizar a procura de uma correspondência na base de dados do Parse, foi criada a função **findUser** no ficheiro **utils**, que também inclui as funções **initializeParse** e **login**, conforme representado na Figura 51. Como pode ser deduzido, o ficheiro **utils** reúne todas as funções relacionadas com o Parse e que são aplicadas no contexto do processo de *login*.

```
import Parse from 'parse/node'

export const initializeParse = () => {
  Parse.initialize('0rF4sRxn9WUDGrAV9nziFygodJrpHehe7ouwkQBZ', 'd0PVm1WbprwGtv7yjbxigRiO8R6wGR8sJxKTB062')
  Parse.serverURL = 'http://localhost:1337/parse'
  return Parse
}

export const findUser = (field: string, value: string) => new Promise((resolve, reject) => {
  let query = new Parse.Query(Parse.User)
  query.useMasterKey = true
  query.equalTo(field, value)
  query.first({
    success: function (result) {
      result ? resolve(result) : reject(result)
    },
    error: error => reject(error)
  })
})

export const login = (result: any, password: string) => new Promise((resolve, reject) => {
  let user = JSON.parse(JSON.stringify(result))
  Parse.User.logIn(user.username, password, {
    success: user => resolve(user),
    error: error => reject(error)
  })
})
```

Figura 51 – Funções relacionadas com o Parse presentes no ficheiro utils.js

6.5 Instalação dos serviços em *containers*

Como foi referido anteriormente, para simplificar e automatizar o processo de criação de instâncias foi utilizado o Docker. A utilização desta tecnologia tornou possível a criação de um ou mais *container*s por cada componente da solução, através da aplicação de ferramentas próprias como o Dockerfile e o Docker Compose.

6.5.1 Dockerfiles

Para criar um *container* é necessária uma imagem – um *template* com as instruções essenciais para a criação de um *container* – existindo a possibilidade de utilizar imagens pré-existentes, que se encontram disponíveis no Docker Hub[52] ou de criar as próprias imagens. Desta forma, para construir as próprias imagens automaticamente, são utilizados os ficheiros Dockerfile – documentos de texto que contêm todas as instruções necessárias para criar uma imagem e as configurações que têm de ser aplicadas ao *container* – e o comando *docker build*. Para a instalação dos serviços Parse Server e Parse Dashboard foram adaptados os Dockerfiles pertencentes a estes dois projetos, que incluem instruções como a cópia dos ficheiros relativos a cada projeto e a definição do serviço que irá ser executado no momento de iniciação do respetivo *container*, como pode ser observado nas figuras Figura 52 e Figura 53, respetivamente. Para os restantes serviços foram utilizadas as imagens por omissão, presentes no Docker Hub.

[52] https://hub.docker.com/

```
FROM node:boron

RUN mkdir -p /parse-server
COPY . /parse-server/

RUN mkdir -p /parse-server/config
VOLUME /parse-server/config

RUN mkdir -p /parse-server/cloud
VOLUME /parse-server/cloud

WORKDIR /parse-server

RUN npm install && npm run build
EXPOSE 1337

ENTRYPOINT npm start ./parse-server-config.json
```

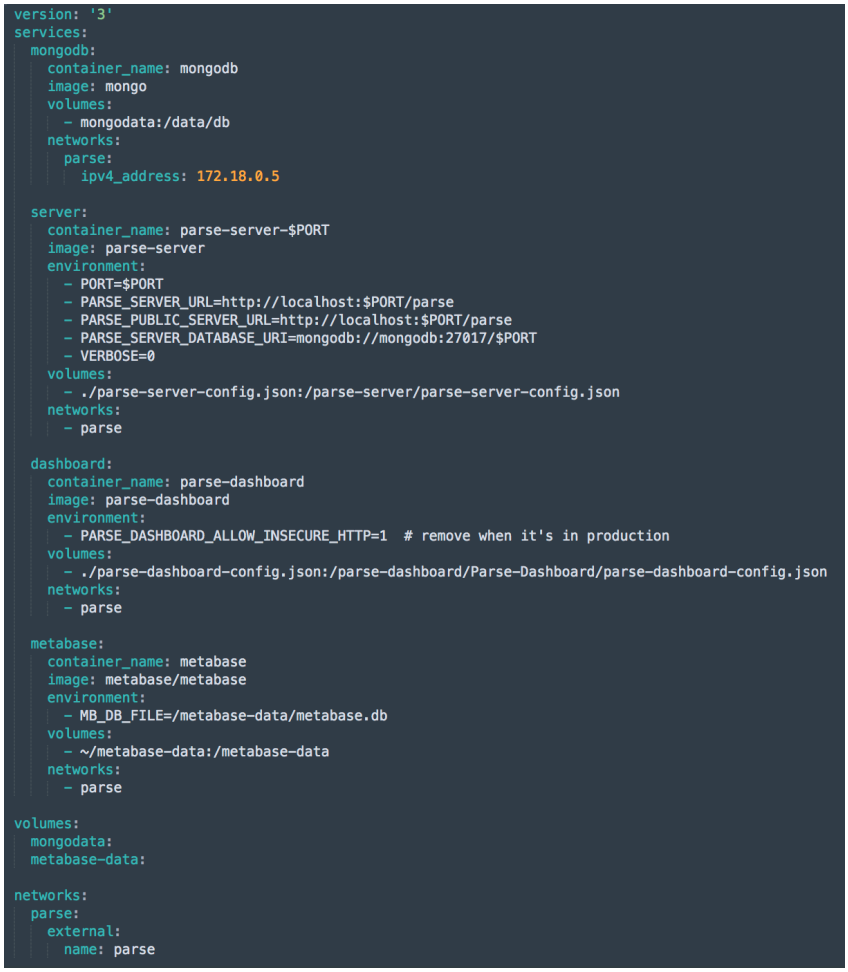
Figura 52 – Dockerfile do serviço Parse Server

```
FROM node:9-alpine

ENV NPM_CONFIG_LOGLEVEL error

RUN mkdir -p /parse-dashboard
COPY . /parse-dashboard/
WORKDIR /parse-dashboard

RUN npm install && npm run build && npm cache clear --force && rm -rf ~/.npm && rm -rf /var/lib/apt/lists/*
EXPOSE 4040

ENTRYPOINT npm start
```

Figura 53 – Dockerfile do serviço Parse Dashboard

6.5.2 Docker Compose

Visto que os serviços constituintes da solução exigem um ambiente *multi-container* e que, como tal, requerem um maior número de comandos e configurações mais complexas, a ferramenta de orquestração Docker Compose permite diminuir essa complexidade. Esta ferramenta utiliza um ficheiro YAML, ilustrado na Figura 54, para configurar todos os serviços em simultâneo, sendo esses mesmos serviços criados e iniciados de seguida, a partir da configuração feita, com um único comando.



```yaml
version: '3'
services:
  mongodb:
    container_name: mongodb
    image: mongo
    volumes:
      - mongodata:/data/db
    networks:
      parse:
        ipv4_address: 172.18.0.5

  server:
    container_name: parse-server-$PORT
    image: parse-server
    environment:
      - PORT=$PORT
      - PARSE_SERVER_URL=http://localhost:$PORT/parse
      - PARSE_PUBLIC_SERVER_URL=http://localhost:$PORT/parse
      - PARSE_SERVER_DATABASE_URI=mongodb://mongodb:27017/$PORT
      - VERBOSE=0
    volumes:
      - ./parse-server-config.json:/parse-server/parse-server-config.json
    networks:
      - parse

  dashboard:
    container_name: parse-dashboard
    image: parse-dashboard
    environment:
      - PARSE_DASHBOARD_ALLOW_INSECURE_HTTP=1  # remove when it's in production
    volumes:
      - ./parse-dashboard-config.json:/parse-dashboard/Parse-Dashboard/parse-dashboard-config.json
    networks:
      - parse

  metabase:
    container_name: metabase
    image: metabase/metabase
    environment:
      - MB_DB_FILE=/metabase-data/metabase.db
    volumes:
      - ~/metabase-data:/metabase-data
    networks:
      - parse

volumes:
  mongodata:
  metabase-data:

networks:
  parse:
    external:
      name: parse
```

Figura 54 – Ficheiro docker-compose.yml

Como é possível observar, o ficheiro YAML possibilita a especificação de várias opções de configuração, nomeadamente:

- *build* – permite indicar o caminho para o Dockerfile que vai der utilizado para construir a imagem do *container*;
- *image* – especifica a imagem (do Docker Hub) utilizada na criação do *container*;
- *ports* – realiza o mapeamento entre os portos (internos) do *container* e os portos do *host* (publicamente expostos);
- *environment* – permite enumerar as variáveis de ambiente do *container*;
- *volumes* – utilizados para persistir os dados gerados e/ou usados pelo *container*, são armazenados numa parte do sistema de ficheiros do *host* que é gerida pelo Docker;
- *networks* – especifica a *network* à qual o serviço se irá conectar, permitindo a criação de *networks* customizadas.

Ao executar o comando *PORT=1338 docker-compose up*, são criados todos os serviços enumerados no ficheiro YAML e é feito o mapeamento entre configurações definidas nos ficheiros parse-server-config.json e parse-dashboard-config.json, presentes no sistema de ficheiros do *host* e os ficheiros correspondentes, presentes nos respetivos *containers*. Ao especificar a variável de ambiente *PORT* na execução do comando *docker-compose*, é possível configurar todos os serviços que utilizem essa variável em simultâneo e facilita a criação de novas instâncias do serviço Parse Server, uma vez que cada porto é único para cada instância. Na Figura 55 podem ser observados os *containers* criados após a execução do *docker-compose*. Para facilitar esta visualização foi utilizado o Portainer[53], uma solução que disponibiliza uma UI *open-source* e que permite visualizar e gerir Docker *containers* facilmente.

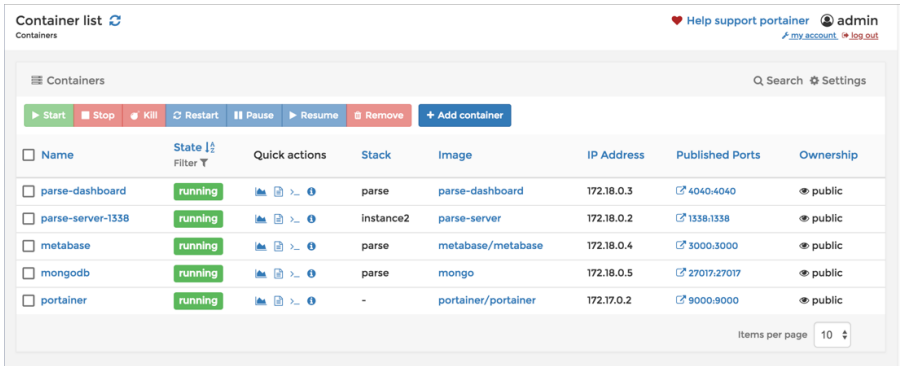


Figura 55 – Portainer para a visualização e gestão dos Docker containers

6.5.3 Integração do Parse Dashboard com o Parse Server

Como foi referido anteriormente, o Parse Dashboard encontrava-se diretamente integrado com o Parse Server, mas com o encerramento da plataforma e com a disponibilização do projeto à comunidade, o Parse Dashboard tornou-se um módulo independente. Para realizar a integração do Parse Dashboard com o Parse Server, é necessário que a configuração inicial com a qual a Dashboard é inicializada corresponda à mesma configuração que foi utilizada para inicializar o Parse Server. Em primeiro lugar e para que tal seja possível, ao inicializar uma instância do Parse Server é necessário explicitar os parâmetros da aplicação à qual essa instância se destina (Figura 56), sendo os parâmetros obrigatórios e sem os quais não é possível ter a instância a correr os seguintes:

- **appId** – o identificador único da aplicação;
- **masterKey** – chave que sobrepõe todas as permissões e que deverá ser secreta.

[53] https://portainer.io/

Para além dos parâmetros supramencionados, convém configurar outros parâmetros adicionais, seja por serem importantes para a correta configuração da instância ou para que os mesmos não assumam valores configurados por omissão, nomeadamente:

- **appName** – a identificação nominal da aplicação;
- **port** – o porto no qual a instância se encontra a correr (porto 1337 por omissão);
- **databaseURI** –o URI da base de dados que será utilizada pela aplicação;
- **push** – objeto que contém as configurações relativas ao envio de notificações;
- **emailAdapter** – objeto com as configurações referentes ao envio de emails.

Figura 56 – Estrutura do ficheiro de configuração de uma instância do Parse Server

Para que seja possível integrar o Parse Dashboard com o Parse Server, à semelhança dos parâmetros configurados para a instância do Parse Server, é necessário configurar os parâmetros *appId*, *masterKey* e *serverURL* no Parse Dashboard, para que este consiga comunicar com o Parse Server. É também recomendado definir o parâmetro *appName* para uma visualização mais intuitiva das aplicações na Dashboard (Figura 57).

Figura 57 – Estrutura do ficheiro de configuração da Parse Dashboard

Uma configuração incorreta destes parâmetros, ou configurações distintas entre o Parse Server e o Parse Dashboard, iria resultar numa visualização incorreta da aplicação na Dashboard. Uma discrepância entre o parâmetro *serverURL* iria originar o erro *unable to connect to server*, representado na Figura 58 e discrepâncias entre os parâmetros *appId* ou *masterKey* iriam devolver o erro *unauthorized*, representado na Figura 59.

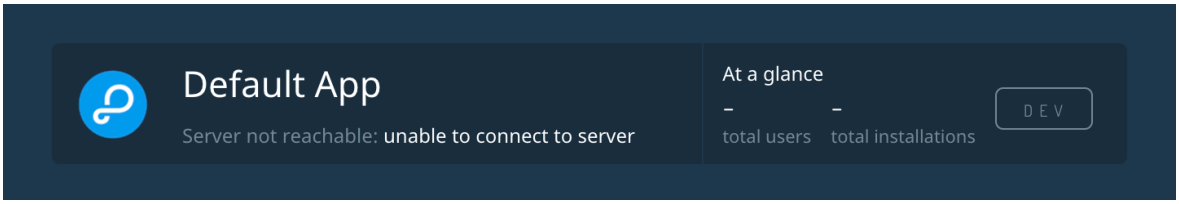


Figura 58 – Representação da aplicação *unable to connect to server* no Parse Dashboard

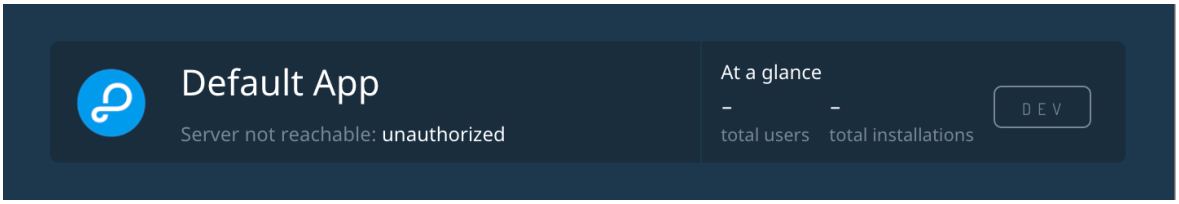


Figura 59 – Representação da aplicação *unauthorized* no Parse Dashboard

7 VALIDAÇÃO DA PLATAFORMA

Nesta secção, para demonstrar a utilização e o funcionamento de algumas *features* que dependem do MBaaS, vão ser apresentados diferentes *workflows* que demonstram algumas funcionalidades da aplicação e o respetivo efeito que a execução dessas funcionalidades provocou do lado do *backend*. Serão ainda apresentados os resultados dos testes de carga efetuados.

7.1 Cenários demonstrativos

Para validar a integração do Parse com a aplicação móvel, foram considerados os cenários de criação, modificação e eliminação de um veículo.

7.1.1 Criação de um veículo

Como pode ser observado na Figura 60, onde se encontra representada a informação que estava presente na base de dados antes da criação do novo veículo, a Dashboard apresenta apenas a informação de um veículo registado por outro utilizador que não aquele que irá proceder ao registo de um novo veículo.

Figura 60 – Veículos presentes na base de dados antes da criação do novo veículo

A Figura 61 representa o processo de criação de um novo veículo na aplicação Thumbeo. No ecrã referente ao perfil do utilizador é possível observar o número de carros registados e o número de carros que se encontram de serviço, para aquele utilizador. Através da seleção do botão relativo aos veículos, são apresentadas informações mais detalhadas de todos os veículos associados àquele utilizador, ou a informação de que o utilizador ainda não tem veículos associados. Ao escolher a opção de adicionar um novo veículo e ao preencher as informações necessárias, se todo o processo correr conforme o esperado o utilizador é redirecionado novamente para o seu perfil, que já mostrará a informação dos veículos atualizada.

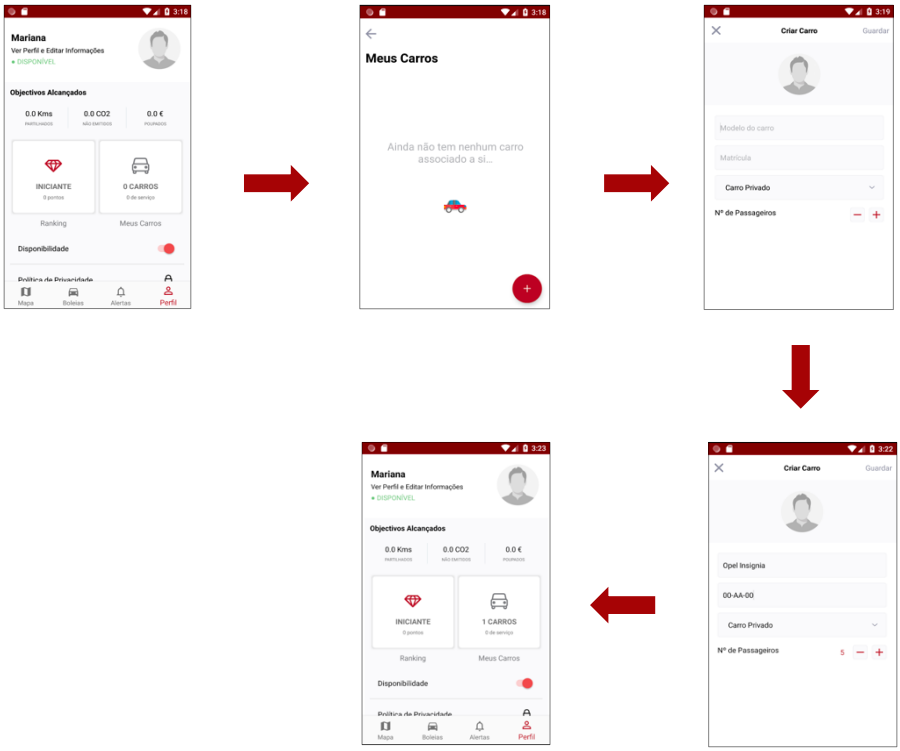


Figura 61 – *Workflow* da criação de um novo veículo

Após o registo do novo veículo, a base de dados foi atualizada com a nova informação referente ao veículo adicionado e essa atualização é também refletida no Parse Dashboard, como revela a Figura 62.

Figura 62 – Veículos presentes na base de dados após a criação do novo veículo

7.1.2 Modificação de um veículo

A Figura 63 representa a informação que estava presente na base de dados antes da modificação do veículo com o modelo “Open Insignia”.

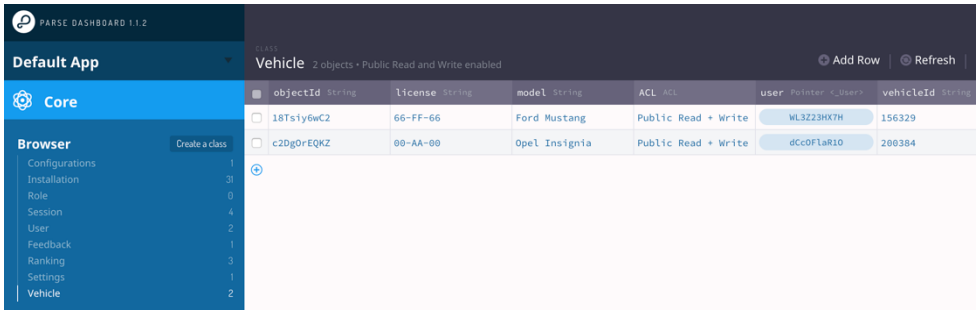


Figura 63 – Veículos presentes na base de dados antes da edição

Para editar um veículo, cujo processo se encontra representado na Figura 64, é necessário selecionar o veículo que se pretende editar a partir do ecrã que lista os veículos associados ao utilizador atual. Após selecionar o veículo pretendido é apresentado o ecrã referente à edição do mesmo, sendo alterada a informação relativa ao modelo do carro, à matrícula e sendo escolhida a opção “Frota da Empresa”. Depois de modificadas as informações, é selecionado o botão superior direito para guardar as alterações efetuadas e, caso o processo corra conforme o esperado, o utilizador é redirecionado para a sua lista de veículos, que já mostrará a informação dos veículos atualizada. Ao voltar para o ecrã do perfil, é possível observar que dos carros registados um deles se encontra de serviço.

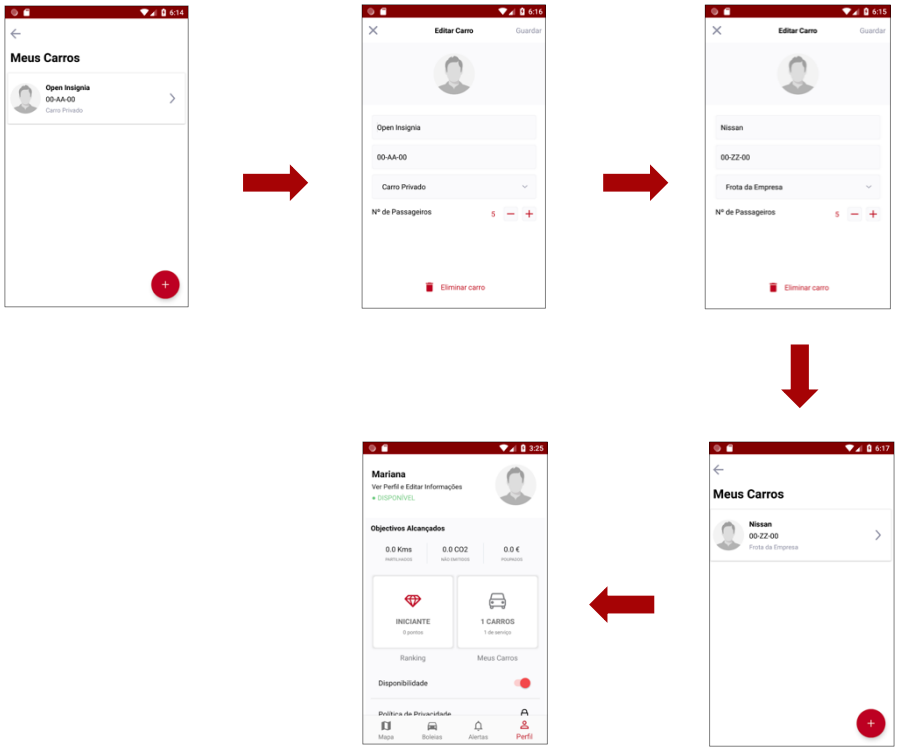


Figura 64 – *Workflow* da edição de um veículo existente

Após efetuar as modificações ao veículo escolhido, a base de dados é atualizada com a nova informação, sendo essa atualização também refletida no Parse Dashboard, como revela a Figura 65.

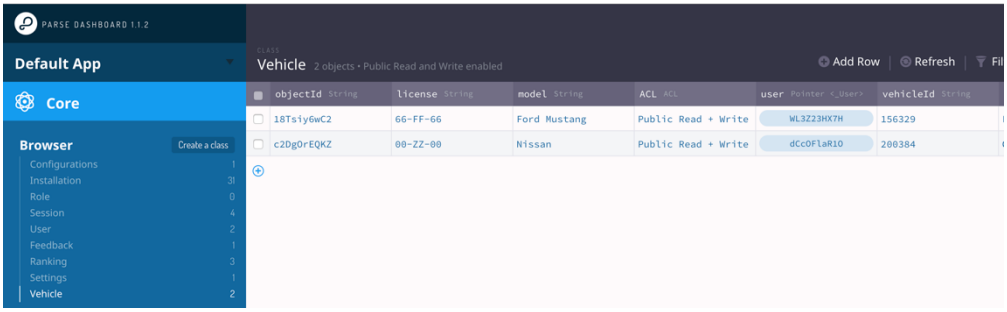


Figura 65 – Veículos presentes na base de dados após a edição

7.1.3 Eliminação de um veículo

Para remover um veículo existente, de forma semelhante ao processo de edição de um veículo, é necessário selecionar o veículo que se pretende eliminar a partir do ecrã que lista os veículos associados ao utilizador atual. Após selecionar o veículo pretendido é apresentado o ecrã referente à edição do mesmo, que inclui a opção de eliminação do veículo atual presente no final do ecrã. Ao selecionar a opção de eliminação do veículo atual, é apresentada uma mensagem de confirmação, que redireciona o utilizador para o ecrã que lista os veículos associados ao mesmo. Caso o processo corra conforme o esperado, o ecrã referente aos veículos do utilizador irá apresentar todos os veículos associados ao mesmo, à exceção do veículo eliminado e ao retroceder para o perfil do utilizador é possível visualizar também a informação dos veículos atualizada.

A informação presente no Parse Dashboard e refletida na base de dados após a eliminação do veículo pode ser observada na Figura 66. Também o *workflow* do processo descrito pode ser observado na Figura 67.

Figura 66 – Veículos presentes na base de dados após a eliminação

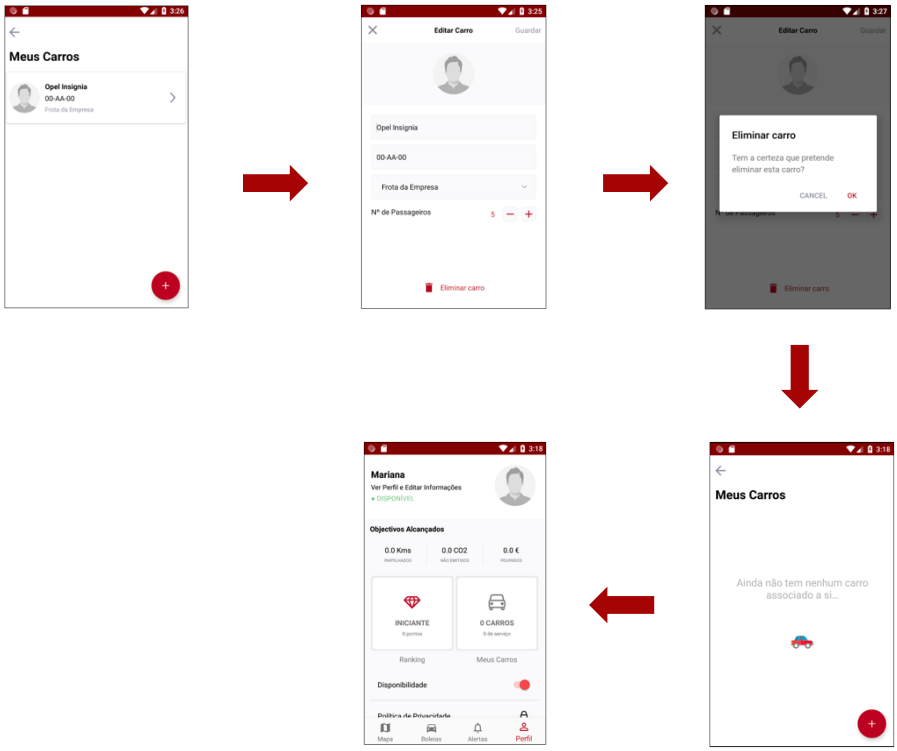


Figura 67 – *Workflow* da eliminação de um veículo existente

7.2 Testes de desempenho

Para testar o desempenho da solução e perceber até que ponto é que esta pode ou não ser escalada, foi realizado um conjunto de testes de carga. Para que fosse possível executar estes testes e analisar os resultados produzidos pelos mesmos, foi criado um *droplet*, um servidor virtual, para o Parse Server no DigitalOcean[54] (Figura 68), tendo sido utilizada uma máquina CentOS[55] para o efeito. Neste *droplet*, tiveram ainda de ser assegurados alguns recursos imprescindíveis para o funcionamento do Parse Server, como o Node.js e o MongoDB.

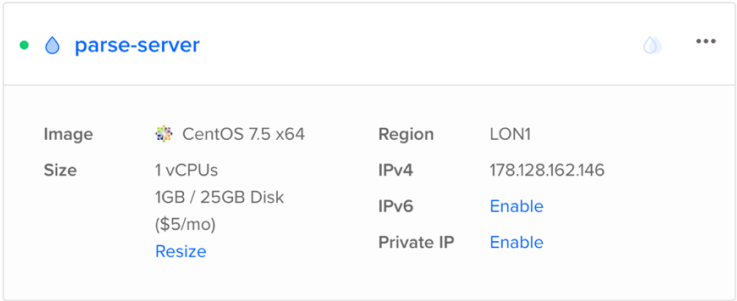


Figura 68 – *Droplet* criado no DigitalOcean para o Parse Server

[54] https://www.digitalocean.com/

[55] https://www.centos.org/

A base de dados MongoDB poderia ser incluída no *droplet* criado, no entanto, tendo em consideração possíveis cenário em que possam existir múltiplas instâncias do Parse Server, seria mais prático se a base de dados fosse disponibilizada num *droplet* exclusivamente para o efeito, de modo a ser partilhada pelas várias instâncias. De outra forma, cada instância iria possuir a sua própria base de dados e seria necessário garantir a coerência da informação entre as várias bases de dados. Como tal, foi criado um *droplet* exclusivamente para MongoDB.

Uma vez que o *droplet* criado para a base de dados é independente do *droplet* criado para o Parse Server, para que o Parse Server tivesse acesso à base de dados foi necessário configurar endereços de IP privados para ambos os *droplets*, tornando-se apenas possível aceder à base de dados dentro da rede privada criada. Uma solução incorreta seria a de expor as interfaces da base de dados publicamente, para que o Parse Server lhe pudesse ter acesso, o que iria sacrificar a segurança da mesma ao torná-la facilmente acessível por qualquer pessoa.

Feitas estas configurações, foi então possível instalar e executar o Parse Server, tendo sido ainda necessário modificar o ficheiro de configuração do mesmo, nomeadamente os parâmetros **serverURL** e **publicServerURL** – para que refletissem a informação do *droplet* – e o parâmetro **databaseURI**, com o URI do *droplet* criado para a base de dados.

Este cenário inicial incluía, portanto, um *droplet* para o Parse Server e outro *droplet* para a base de dados MongoDB (Figura 69). Após terem sido efetuadas todas as configurações necessárias e estando todo o cenário operacional, foram efetuados os testes de carga a *endpoints* da API do Parse Server, de forma a verificar os limites suportados pelo serviço.

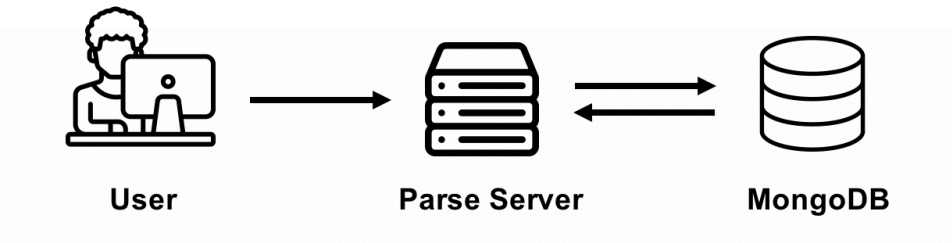


Figura 69 – Esquema representativo do cenário inicial [68]

Para a realização destes testes foi utilizado o serviço gratuito do Loader.io[56], tendo sido escolhido o tipo de teste "clientes por segundo". Este tipo de teste permite especificar o número de clientes que se conectam ao servidor por segundo, durante um determinado período de tempo. Assim, um teste com 1000 clientes por segundo com a duração de 1 minuto, significaria que a cada segundo iriam ser feitos 1000 pedidos ao servidor, o que iria resultar num total de 6000 pedidos realizados no espaço de 1 minuto [69]. Para além do número de clientes e da duração dos testes, é ainda possível especificar outro tipo de parâmetros, como por exemplo um valor limite para a taxa de erro – que resultaria na suspensão do teste caso fosse atingido – e as *headers* necessárias para a realização dos pedidos aos *endpoints* da API.

[56] https://loader.io/

Os testes realizados a este cenário inicial, em que existia apenas um *droplet* direcionado ao Parse Server, tinham a intenção de perceber de que forma a solução se comportava para o cenário mais simples, com apenas um servidor. Para a execução dos testes foi definido um conjunto de valores fixo para o número clientes por segundo e a duração de 1 minuto por cada teste, sendo ainda considerado um limite de 100% para o valor da taxa de erro. Estes testes foram efetuados aos *endpoints* **/classes/User** – que devolve os registos relativos a cerca de 3.600 utilizadores – e **/classes/Vehicle/GDfeo6RNgi**, que devolve a informação do veículo com o objectId GDfeo6RNgi.

Quanto aos resultados obtidos (Tabela 3), relativamente ao *endpoint* **/classes/User** foi possível testar até 6000 clientes por segundo, embora com uma taxa de erro bastante elevada e para o número mínimo de clientes/seg testado, foi observada uma taxa de erro de 27.4%. Tendo em conta o conjunto de valores testado, não foi possível obter resultados para valores de 8000 e 10.000 clientes/seg, uma vez que os testes foram suspendidos por alcançarem o limite de erro definido. No que diz respeito ao *endpoint* **/classes/Vehicle/GDfeo6RNgi**, foi possível testar até 10.000 clientes/seg, com uma taxa de erro de 75.6% e para o número mínimo de clientes/seg testado, foi obtida uma taxa de erro de 0.0%.

Tabela 3 – Resultados dos testes efetuados aos *endpoints* para 1 servidor

	/classes/User		**/classes/Vehicle/GDfeo6RNgi**	
Clientes/seg	Média (ms)	Taxa de Erro (%)	Média (ms)	Taxa de Erro (%)
500	1732	27.4	99	0.0
1000	3339	42.9	1669	26.5
2000	5216	59.0	3828	42.9
4000	6210	70.6	5023	57.1
6000	6288	78.2	5242	65.3
8000	-	-	5593	71.5
10000	-	-	5251	75.6

Tendo em conta os gráficos gerados pelo Loader.io para os testes executados com o valor de 500 clientes/seg, por exemplo, é possível observar que no gráfico relativo ao *endpoint* **/classes/User** a representação do tempo médio de resposta é bastante irregular e que o número de clientes por segundo se mantém constante (Figura 70). Embora o número de clientes por segundo ilustrado no gráfico não corresponda ao valor de 500 clientes/seg definido previamente, sendo que no gráfico se encontra representado o dobro do valor definido para este parâmetro, esta situação deve-se ao facto de os clientes se manterem ativos por mais de 1 segundo, sendo que 500 clientes/seg significam realmente 500 clientes por segundo e não 500 clientes ativos a cada segundo [70].

Figura 70 – Teste de carga ao *endpoint* /classes/User com 500 clientes/seg

Por outro lado, para o *endpoint* **/classes/Vehicle/GDfeo6RNgi** o número de clientes por segundo não se revela tão constante ao longo do tempo como para o *endpoint* **/classes/User** e, para além disso, aproxima-se mais do valor real definido para este parâmetro. Quanto ao tempo médio de resposta, o gráfico gerado (Figura 71) ilustra uma representação menos irregular deste parâmetro, que apresenta um declive mais acentuado nos primeiros segundos de execução do teste, o que significa que o recurso requisitado não se encontrava em cache e, portanto, a sua aquisição foi mais demorada. Nos segundos seguintes, o tempo médio de resposta apresenta uma representação mais constante, uma vez que o recurso já se encontra disponível em cache.

Finalmente, pode-se concluir que foram obtidos melhores resultados para o *endpoint* **/classes/Vehicle/GDfeo6RNgi**, sobretudo devido ao facto de a tabela **Vehicle** possuir um número de registos substancialmente menor do que a tabela **User**, o que contribuiu para o aumento do tempo médio de resposta.

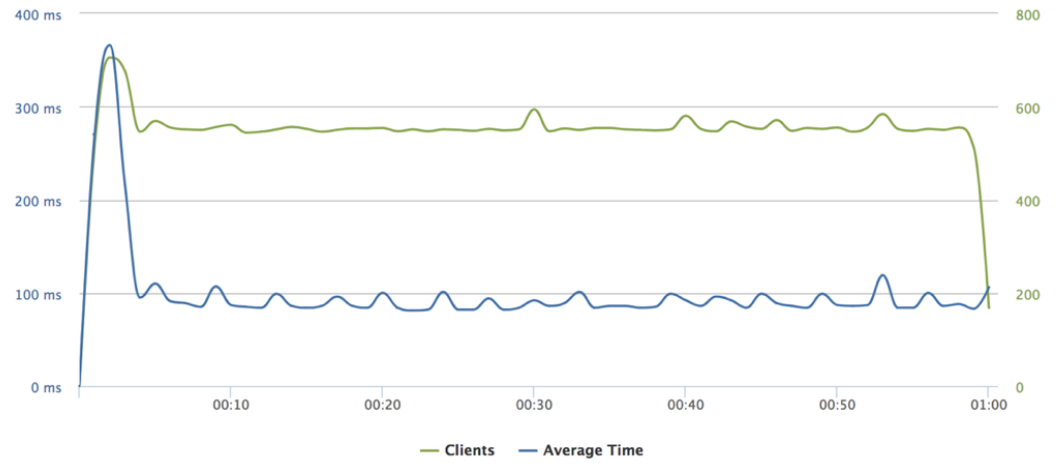


Figura 71 – Teste de carga ao *endpoint* /classes/Vehicle/GDfeo6RNgi com 500 clientes/seg

Com este projeto, mostrou-se ser possível criar várias instâncias do Parse Server, ainda que a intenção inicial fosse a de ter múltiplas aplicações a utilizar o *backend* do Parse Server simultaneamente, sendo necessário uma instância por cada aplicação distinta, uma vez que o Parse Server não permite ter mais do que uma aplicação por instância.

Tendo isto em conta, a criação de múltiplas instâncias pode ser igualmente útil para escalar aplicações, ao permitir ter várias instâncias destinadas a uma única aplicação. Para um cenário deste género, a utilização de um *load balancer* iria ser bastante útil, uma vez que iria permitir obter uma maior disponibilidade e um melhor desempenho [68]. Com a sua utilização, se um dado servidor deixar de estar operacional, existirá pelo menos um servidor adicional, que disponibilize o mesmo conteúdo, para tomar o seu lugar. Caso um utilizador faça um pedido direcionado ao *load balancer*, este irá encaminhar o pedido para um dos servidores disponíveis, que por sua vez irá tratar desse mesmo pedido (Figura 72).

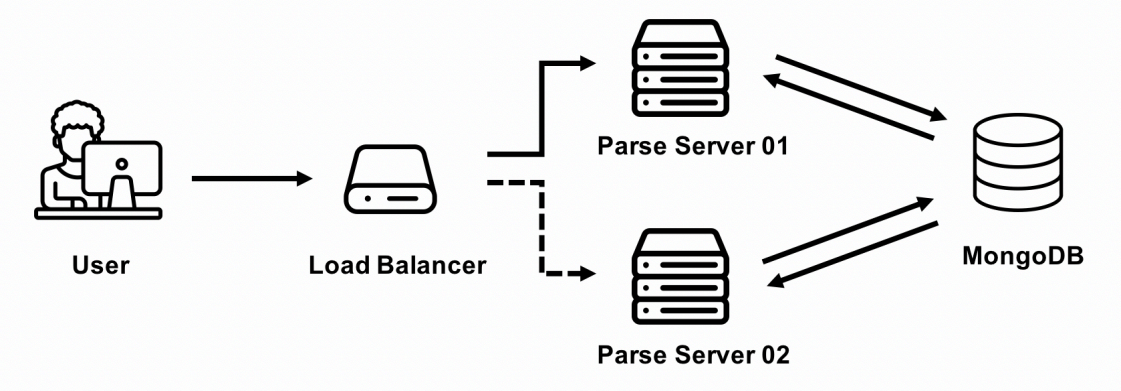


Figura 72 – Esquema representativo de um cenário com *load balancing* [68]

O DigitalOcean possibilita a criação de um *load balancer*, sendo apenas necessário especificar o conjunto de servidores com os quais este irá comunicar e configurar as suas *forwarding rules*, isto é, as regras que definem a forma como o tráfego é encaminhado do *load balancer* para os *droplets*. É ainda possível escolher o algoritmo utilizado pelo *load balancer*, podendo ser utilizado o algoritmo *Round Robin*, em que os pedidos são distribuídos igualmente pelos servidores disponíveis, ou o algoritmo *Least Connections*, em que cada pedido é encaminhado para o servidor com o menor número de conexões ativas [68].

Posto isto, foi criado um *load balancer* e 5 réplicas do *droplet* relativo ao Parse Server. Consequentemente, o novo cenário passou a ser constituído por 6 *droplets* com o Parse Server, por 1 *droplet* com MongoDB e por 1 *load balancer* (Figura 73).

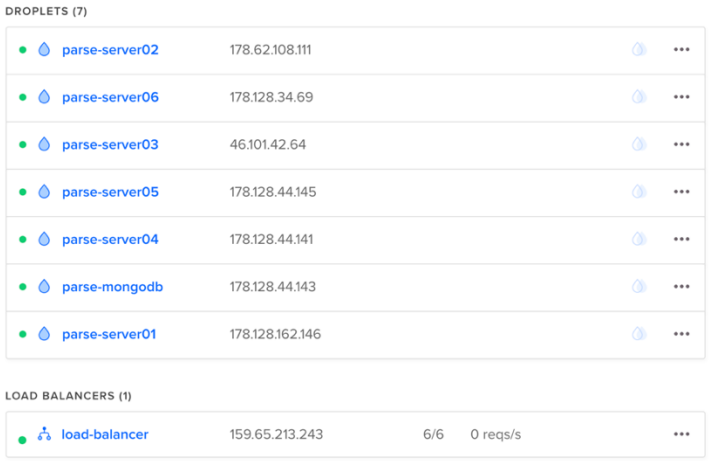


Figura 73 – *Droplets* criados no DigitalOcean para o cenário com *load balancing*

Após o novo cenário se encontrar operacional, os testes realizados anteriormente foram repetidos, aumentando-se gradualmente o número de servidores disponíveis para os mesmos. Desta forma, foram realizados testes aos *endpoints* para 2, 4 e 6 servidores (Tabela 4).

Tabela 4 – Resultados dos testes efetuados com a utilização de um *load balancer*

		/classes/User		**/classes/Vehicle/GDfeo6RNgi**	
	Clientes/seg	Média (ms)	Taxa de Erro (%)	Média (ms)	Taxa de Erro (%)
2 servidores	500	145	0.0	90	0.0
	1000	1712	27.9	106	0.0
	2000	3386	40.3	2135	27.3
	4000	4765	53.6	3125	39.8
	6000	5546	62.5	4190	50.2
	8000	5713	68.5	4277	55.1
	10000	5889	72.5	4392	60.8
4 servidores	500	90	0.0	83	0.0
	1000	147	0.0	88	0.0
	2000	2698	15.0	113	0.0
	4000	3477	31.3	2169	22.4
	6000	4240	42.6	2512	35.2
	8000	4271	52.1	2683	41.1
	10000	4477	57.8	3121	48.8

		/classes/User		/classes/Vehicle/GDfeo6RNgi	
	Clientes/seg	Média (ms)	Taxa de Erro (%)	Média (ms)	Taxa de Erro (%)
6 servidores	500	90	0.0	82	0.0
	1000	111	0.0	84	0.0
	2000	1669	8.2	91	0.0
	4000	2130	25.7	257	0.1
	6000	2341	35.4	525	29.6
	8000	2421	44.1	559	40.2
	10000	2899	50.9	534	46.6

Pela análise dos resultados obtidos, percebe-se que a solução pode ser escalada e que a utilização de um *load balancer* é benéfica para o desemprenho da mesma. No geral, foram obtidos melhores resultados com o aumento do número de servidores e com a distribuição equilibrada dos pedidos por esses mesmos servidores. Embora com a repetição dos testes os resultados obtidos oscilassem um pouco, a diferença entre o cenário inicial e o cenário com *load balancing* é notória.

8 CONCLUSÕES

A tendência evolutiva da tecnologia – nomeadamente a dos *smartphones* – e a dependência cada vez maior das pessoas pela mesma, tem implicações diretas nas aplicações móveis. Estas aplicações irão cada vez mais desempenhar um papel fulcral nas vidas dos seus utilizadores, sendo necessário que estas se adaptem às necessidades dos mesmos. Desta forma, as aplicações irão necessitar de incluir um número cada vez maior de funcionalidades que lhes permitam obter uma vantagem competitiva em relação às aplicações concorrentes e satisfazer as necessidades dos seus utilizadores, de forma a conquistar a fidelização dos mesmos.

Apesar de já existirem diversas plataformas de Backend as a Service, que disponibilizam um conjunto de funcionalidades semelhante e cuja finalidade é essencialmente a mesma, a possibilidade de descontinuação do serviço é uma das principais ameaças à utilização destas plataformas e pode pesar bastante no momento da escolha de uma plataforma de Backend as a Service. Consequentemente, as plataformas *open-source* podem ser vantajosas em relação às plataformas concorrentes pagas, uma vez que as plataformas *open-source* permitem a obtenção de um maior controlo sobre a plataforma e, desta forma, atribuem-lhe uma maior confiança. Embora ainda seja necessário instalar e configurar a plataforma, ao contrário de um serviço pago que normalmente já assume esse esforço, as plataformas *open-source* são mais confiáveis a longo prazo no que diz respeito à probabilidade de encerramento do serviço uma vez que, ao contrário das plataformas de código fechado, permitem ter total acesso à implementação da plataforma. Ao possuir acesso total à implementação, significa que será menos provável ter de transitar para um outro serviço de Backend as a Service, caso o serviço atual seja descontinuado, uma vez que com acesso à implementação da plataforma é possível mantê-la operacional com algum esforço de implementação.

8.1 Trabalho desenvolvido no estágio

Os objetivos principais deste projeto relacionavam-se com a instalação, desenvolvimento e utilização da plataforma Parse para suportar determinadas funcionalidades desejadas nas aplicações móveis. Com a exploração deste projeto, foi possível constatar que esta plataforma dispõe uma base sólida e de grande utilidade para este tipo de aplicações, apresentando ainda a vantagem de ser uma plataforma *open-source*. Com a utilização desta plataforma e com o desenvolvimento de novas funcionalidades para a mesma, é possível obter um serviço de *backend* bastante completo e que poderá permitir agilizar o desenvolvimento de novas aplicações.

Por ter sido desenvolvido em contexto empresarial, foi possível obter um maior apoio no que diz respeito ao desenvolvimento deste projeto e às dificuldades encontradas no decorrer do mesmo. Para além de ter sido útil no esclarecimento de dúvidas, uma vez que havia sempre alguém disponível e com conhecimentos para auxiliar, o contexto empresarial referido anteriormente permitiu ainda o contacto com novas tecnologias e a oportunidade de integração nos processos de trabalho da empresa.

8.2 Integração nos processos de trabalho da empresa

Para a execução deste projeto foram utilizadas as mesmas ferramentas que são empregues em projetos reais da empresa, o que permitiu obter uma melhor perceção do trabalho num âmbito empresarial através da integração nos processos de trabalho da empresa.

Relativamente ao armazenamento do código do Parse Server, Parse Dashboard e das respetivas aplicações associadas, foi utilizado o GitLab[57] como repositório e para o controlo de versões foi utilizado o Sourcetree[58], representado na Figura 74.

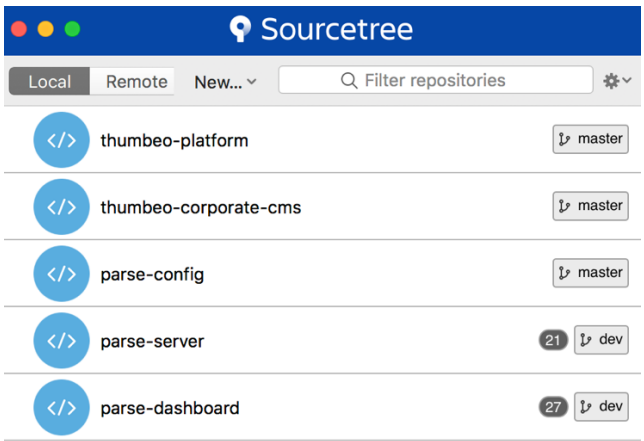


Figura 74 – Controlo de versões com o Sourcetree

No que diz respeito à gestão do projeto, para o planeamento e para a organização das tarefas relativas ao mesmo foi utilizado o Redmine[59] e os quadros presentes nos repositórios do GitLab, como pode ser observado na Figura 75.

Figura 75 – *Board* de tarefas para o Parse Server presente no GitLab

[57] https://about.gitlab.com/
[58] https://www.sourcetreeapp.com/
[59] https://www.redmine.org/

8.3 Lições aprendidas

Com a realização deste estágio, foi possível adquirir uma melhor perceção sobre como será trabalhar no âmbito empresarial, que difere do contexto académico em diversos aspetos.

Enquanto que no contexto universitário é importante demonstrar responsabilidade para obter bons resultados, no âmbito empresarial o tipo de responsabilidade que deve ser demonstrada é um pouco diferente, uma vez que a falta de responsabilidade não terá só impacto nos resultados individuais, mas também nos resultados de toda a equipa, podendo ainda afetar o cliente e a imagem com que este irá ficar de toda empresa.

Assim como no meio académico, saber trabalhar em equipa é essencial e, para além disso, é importante saber aceitar as críticas e aprender com as mesmas. Também é indispensável ter autonomia e espírito crítico, para que seja possível proceder à divisão do trabalho pelos elementos da equipa de forma equilibrada.

As capacidades de comunicação são também bastante importantes, não só no que se refere ao trabalho em equipa, contribuindo para a criação de um bom ambiente entre colegas e para que seja possível alcançar melhores resultados, mas também no que diz respeito à comunicação com os clientes. Uma boa comunicação permite que sejam evitados erros no levantamento de requisitos dos projetos, por exemplo, ou outro tipo de erros resultantes de uma má articulação dos pensamentos ou de uma incorreta interpretação dos mesmos.

Para além disso, é vantajoso saber organizar o tempo e as tarefas, sobretudo quando se trabalha em vários projetos em simultâneo, para que se consiga dar prioridade às tarefas mais importantes, tendo em conta os prazos estipulados para a entrega das mesmas aos clientes.

8.4 Oportunidade para evolução e trabalho futuro

Para além das funcionalidades desenvolvidas, seria proveitoso desenvolver outras funcionalidades que deixaram de estar disponíveis no Parse após a descontinuação do projeto, como por exemplo a funcionalidade relacionada com os utilizadores do Parse Dashboard.

Seria também bastante útil explorar os restantes módulos do Parse e ver aqueles que seriam benéficos para integrar no projeto.

Como trabalho futuro, seria ainda interessante integrar a plataforma em aplicações com sistemas operativos diferentes dos que foram mencionados, uma vez que o Parse disponibiliza um vasto conjunto de SDKs e seria proveitoso ver esta plataforma aplicada a diferentes ambientes. Seria também possível analisar de que forma esta integração é mais ou menos difícil consoante o ambiente em que a aplicação é executada e as principais diferenças em termos da integração de funcionalidades ou da implementação das mesmas.

8.5 Dificuldades Sentidas

No decorrer do projeto surgiram diversas dificuldades que tiveram de ser resolvidas e ultrapassadas para a correta concretização do mesmo.

A principal dificuldade diz respeito à documentação disponível para a plataforma Parse e respetivos projetos associados. Tendo passado por um processo de transição e, como tal, sofrido modificações no decorrer desse processo, seria de esperar que a documentação referente a esta plataforma refletisse essas mesmas alterações. No entanto, grande parte da documentação encontra-se desatualizada ou em atualização, o que constitui um obstáculo à compreensão do projeto e à execução do mesmo. Para a instalação do Parse Server e do Parse Dashboard, algumas variáveis de configuração e de ambiente referidas na documentação destes projetos já não se encontram em uso ou foram alteradas, o que dificultou a configuração e instalação dos mesmos. Também a forma como é feita a inicialização do Parse Server, por exemplo, sofreu alterações na transição do projeto para *open-source*, havendo documentação desatualizada relativamente a certas funcionalidades que a plataforma englobava e à estrutura do projeto e respetivos ficheiros, o que dificulta o entendimento do projeto e respetivo desenvolvimento.

Ainda relacionado com o facto de alguma da informação disponibilizada se encontrar desatualizada, outro obstáculo encontrado diz respeito aos Dockerfiles. Tanto o Parse Server como o Parse Dashboard têm o seu ficheiro Dockerfile para possibilitar a execução de cada um destes projetos em Docker *containers*, no entanto, ao executar estes ficheiros, as imagens dos respetivos *containers* não são construídas com sucesso, o poderá ser um indício de que os mesmos também se encontram desatualizados. Ao consultar o repositório do Parse Dashboard, é possível verificar que foram reportados *issues* relativos a este problema e ao adaptar os Dockerfiles (nomeadamente no que diz respeito às versões do *node* utilizadas pelos mesmos) foi possível finalmente construir as imagens dos *containers* com sucesso [71].

Outra das dificuldades encontradas relaciona-se com a possibilidade de ter um cenário em que existam múltiplas aplicações a ser executadas numa única instância. Na documentação presente no repositório do Parse Dashboard, encontra-se referida a possibilidade de definir várias aplicações no ficheiro de configuração relativo a este módulo, o que pode induzir em erro quanto a ser possível executar mais do que uma aplicação por cada instância do Parse Server. No então, após consultar as *issues* do repositório do Parse Server, é possível concluir que esse projeto apenas permite a definição de uma única aplicação por cada instância do mesmo [72][73]. Desta forma, a criação de um Docker container por cada instância do Parse Server serviu para contornar este obstáculo e permitiu o acesso e a gestão de várias aplicações – cada uma com o seu contexto de execução individual e base de dados própria – a partir da Dashboard.

Por fim, outra dificuldade sentida refere-se às notificações do lado do cliente, uma vez que se encontravam presentes na plataforma antes da transição. Como tal, foi necessário perceber de que forma se poderia voltar a integrar esta funcionalidade [74].

9 REFERÊNCIAS

[1] S. Plangi, “Overview of Backend as a Service platforms,” Tartu, 2016.

[2] P. Nguyen, “Mobile Backend as a Service: The Pros and Cons of Parse,” Lahti, 2016.

[3] “A Guide to Mobile App Development: Web vs. Native vs. Hybrid | Clearbridge Mobile.” [Online]. Available: https://clearbridgemobile.com/mobile-app-development-native-vs-web-vs-hybrid/#Web_Apps. [Accessed: 14-May-2018].

[4] “Benefits of using React for Web Applications.” [Online]. Available: https://xigen.co.uk/news-article/react-for-web-applications. [Accessed: 17-May-2018].

[5] “Virtual DOM and Internals - React.” [Online]. Available: https://reactjs.org/docs/faq-internals.html#what-is-the-virtual-dom. [Accessed: 17-May-2018].

[6] “Sites Using React.” [Online]. Available: https://github.com/facebook/react/wiki/sites-using-react. [Accessed: 16-May-2018].

[7] J. Thönes, “Microservices,” *IEEE Softw.*, vol. 32, no. 1, 2015.

[8] “What is a Container | Docker.” [Online]. Available: https://www.docker.com/what-container. [Accessed: 08-Jun-2018].

[9] “What is Backend as a Service (BaaS)?” [Online]. Available: https://www.techopedia.com/definition/29428/backend-as-a-service-baas. [Accessed: 10-Oct-2017].

[10] “Backend-as-a-Service - A Global Strategic Business Report,” 2016. [Online]. Available: http://www.strategyr.com/pressMCP-7959.asp. [Accessed: 26-Oct-2017].

[11] F. Carranza-Garcia, C. Rodriguez-Dominguez, J. L. Garrido, and G. Guerrero-Contreras, “BaaS-4US: A Framework to Develop Standard Backends as a Service for Ubiquitous Applications,” *Proc. - 2016 15th Int. Conf. Ubiquitous Comput. Commun. 2016 8th Int. Symp. Cybersp. Secur. IUCC-CSS 2016*, pp. 23–30, 2017.

[12] J. Weber, “What is Backend as a Service and What Does It Mean for Devs?,” 2016. [Online]. Available: https://www.ctl.io/developers/blog/post/what-is-backend-as-a-service. [Accessed: 26-Oct-2017].

[13] “DIY App Dev vs App Dev with mBaaS.” [Online]. Available: http://web.archive.org/web/20160407155909im_/http://www.kinvey.com/images/why/app-dev-graphic.png.

[14] “How To Choose The Right Backend as a Service (BaaS) Platform,” 2016. [Online]. Available: http://waracle.net/how-to-choose-the-right-backend-as-a-service-baas-platform/. [Accessed: 26-Oct-2017].

[15] “Enterprise BaaS.” [Online]. Available: https://backendless.com/what-is-backend-as-a-service/enterprise-baas/. [Accessed: 28-Oct-2017].

[16] “Backend as a Service.” [Online]. Available: https://backendless.com/wp-content/uploads/2014/01/client-server-diagram.png. [Accessed: 10-Oct-2017].

[17] “Firebase Opens Its Real-Time App Infrastructure To All Developers.” [Online]. Available: https://techcrunch.com/2013/02/13/firebase-public-beta/. [Accessed: 26-Oct-2017].

[18] “Firebase is Joining Google!” [Online]. Available: https://firebase.googleblog.com/2014/10/firebase-is-joining-google.html. [Accessed: 26-Oct-2017].

[19] “App Indexing for Firebase.” [Online]. Available: https://firebase.google.com/products/app-

indexing/. [Accessed: 26-Oct-2017].
[20] “AdMob for Firebase.” [Online]. Available: https://firebase.google.com/docs/admob/. [Accessed: 26-Oct-2017].
[21] “Google AdWords for Firebase.” [Online]. Available: https://firebase.google.com/docs/adwords/. [Accessed: 29-Jan-2018].
[22] “Flamelink CMS.” [Online]. Available: https://flamelink.io/. [Accessed: 16-Nov-2017].
[23] J. Mill, “Flamelink, the CMS for Firebase you’ve been searching for, is here.” [Online]. Available: https://hackernoon.com/flamelink-the-cms-for-firebase-youve-been-searching-for-is-here-dac5bfccdd4a. [Accessed: 16-Nov-2017].
[24] “Backups automatizados | Firebase.” [Online]. Available: https://firebase.google.com/docs/database/web/backups. [Accessed: 16-Nov-2017].
[25] “Becoming the Real Parse Alternative.” [Online]. Available: https://hackernoon.com/becoming-the-real-parse-alternative-f2a5474c830f. [Accessed: 27-Oct-2017].
[26] K. Zolotareva, “5 Solid Parse Alternatives 2017,” 2017. [Online]. Available: https://thetool.io/2017/parse-alternatives. [Accessed: 26-Oct-2017].
[27] “Backendless Features.” [Online]. Available: https://backendless.com/platform/backend-as-a-service/#7274. [Accessed: 27-Oct-2017].
[28] “CloudBoost repository.” [Online]. Available: https://github.com/CloudBoost/cloudboost. [Accessed: 04-Jan-2018].
[29] “Parse Server Hosting | Back4App.” [Online]. Available: https://www.back4app.com/product/parse-server-hosting. [Accessed: 16-Jan-2018].
[30] “Parse, The ‘Heroku For Mobile’, Raises $5.5 Million Series A.” [Online]. Available: https://techcrunch.com/2011/11/09/parse-the-heroku-for-mobile-raises-5-5-million-series-a/. [Accessed: 16-Nov-2017].
[31] “Parse raises $5.5M to give any mobile app a home in the cloud.” [Online]. Available: https://gigaom.com/2011/11/09/parse-funding/. [Accessed: 16-Nov-2017].
[32] “Facebook Buys Parse To Offer Mobile Development Tools As Its First Paid B2B Service.” [Online]. Available: https://techcrunch.com/2013/04/25/facebook-parse/. [Accessed: 10-Nov-2017].
[33] “Facebook Buys Mobile App Platform Parse.” [Online]. Available: https://www.forbes.com/sites/matthickey/2013/04/25/facebook-buys-mobile-app-platform-parse/#7df68e3e6ad4. [Accessed: 10-Nov-2017].
[34] “Parse Shut Down Reasons.” [Online]. Available: https://blog.back4app.com/2016/10/30/parse-server-dossier/. [Accessed: 10-Nov-2017].
[35] “Parse Server repository.” [Online]. Available: https://github.com/parse-community/parse-server.
[36] “Parse Server Guide.” [Online]. Available: http://docs.parseplatform.org/parse-server/guide/#development-guide. [Accessed: 15-Nov-2017].
[37] “Parse x Parse Server.” [Online]. Available: https://blog.back4app.com/2016/11/08/parse-com-vs-parse-server/. [Accessed: 17-Nov-2017].
[38] “Parse open-source is moving fast.” [Online]. Available: https://blog.back4app.com/2016/03/30/parse-open-source-is-moving-fast-but-still-has-some-features-to-be-opened-up-see-which/. [Accessed: 17-Nov-2017].
[39] “Parse Server Modules.” [Online]. Available: https://github.com/parse-server-modules. [Accessed: 06-Dec-2017].

[40] “Parse Server applinks-metatag.” [Online]. Available: https://github.com/parse-server-modules/applinks-metatag. [Accessed: 06-Dec-2017].
[41] “App Links.” [Online]. Available: https://developers.facebook.com/docs/applinks. [Accessed: 06-Dec-2017].
[42] “Facebook SocketRocket.” [Online]. Available: https://github.com/facebook/SocketRocket. [Accessed: 06-Dec-2017].
[43] “Introducing SocketRocket: A WebSocket library for Objective-C.” [Online]. Available: https://medium.com/square-corner-blog/introducing-socketrocket-a-websocket-library-for-objective-c-e8b386442805. [Accessed: 06-Dec-2017].
[44] “Parse Files Utils.” [Online]. Available: https://github.com/parse-server-modules/parse-files-utils. [Accessed: 06-Dec-2017].
[45] “O que é o Google Cloud Pub/Sub?” [Online]. Available: https://cloud.google.com/pubsub/docs/overview. [Accessed: 06-Dec-2017].
[46] “Documentação do Google Cloud Pub/Sub.” [Online]. Available: https://cloud.google.com/pubsub/docs/. [Accessed: 06-Dec-2017].
[47] “Nível gratuito da AWS.” [Online]. Available: https://aws.amazon.com/pt/free/?awsf.undefined=categories%23alwaysfree. [Accessed: 17-Jan-2018].
[48] “Mailjet.” [Online]. Available: https://www.mailjet.com/pricing/#developers. [Accessed: 17-Jan-2018].
[49] “SendGrid Pricing and Plans.” [Online]. Available: https://sendgrid.com/pricing/. [Accessed: 17-Jan-2018].
[50] “Parse Server Generic Email Adapter.” [Online]. Available: https://www.npmjs.com/package/parse-server-genericemail-adapter. [Accessed: 17-Jan-2018].
[51] “Here Are the Alternatives to Push Notifications via Parse – Adweek.” [Online]. Available: http://www.adweek.com/digital/alternatives-push-notifications-parse/. [Accessed: 17-Jan-2018].
[52] “Best alternative for Parse Push Notification.” [Online]. Available: https://www.kumulos.com/2016/02/04/best-alternative-for-parse-push-notification/. [Accessed: 17-Jan-2018].
[53] “Alternatives to Parse Push Notifications.” [Online]. Available: https://www.urbanairship.com/blog/alternatives-to-parse-push-notifications-faqs. [Accessed: 17-Jan-2018].
[54] “Google Analytics for Firebase.” [Online]. Available: https://firebase.google.com/docs/analytics/. [Accessed: 26-Oct-2017].
[55] “User Engagement | Product Analytics | Mixpanel.” [Online]. Available: https://mixpanel.com/engagement/. [Accessed: 24-Jan-2018].
[56] “React - A JavaScript library for building user interfaces.” [Online]. Available: https://reactjs.org/. [Accessed: 18-Jan-2018].
[57] “Ficheiro DashboardView.react.js.” [Online]. Available: https://github.com/parse-community/parse-dashboard/blob/master/src/dashboard/DashboardView.react.js. [Accessed: 25-Jan-2018].
[58] “7 Reasons Why I Use InVision for Rapid Prototyping – Jeremy Wells – Medium.” [Online]. Available: https://medium.com/@mrjeremywells/7-reasons-why-i-use-invision-for-rapid-prototyping-ed1c33d5b86. [Accessed: 04-Jun-2018].

[59] “Android Developers Guide | The ParseObject.” [Online]. Available: http://docs.parseplatform.org/android/guide/#the-parseobject. [Accessed: 02-Jun-2018].
[60] “Model View Presenter in Android – Medium.” [Online]. Available: https://medium.com/@tinmegali/model-view-presenter-mvp-in-android-part-1-441bfd7998fe. [Accessed: 01-Jun-2018].
[61] “REST API Guide | Parse.” [Online]. Available: http://docs.parseplatform.org/rest/guide/#objects-api. [Accessed: 14-May-2018].
[62] “REST API Guide | Parse.” [Online]. Available: http://docs.parseplatform.org/rest/guide/#calling-from-client-apps. [Accessed: 14-May-2018].
[63] “Google Cloud Messaging | Google Developers.” [Online]. Available: https://developers.google.com/cloud-messaging/. [Accessed: 14-May-2018].
[64] “Parse Server FCM push adapter.” [Online]. Available: https://github.com/parse-community/parse-server-push-adapter/issues/32. [Accessed: 14-May-2018].
[65] “Client push unavailable.” [Online]. Available: https://github.com/parse-community/parse-server/issues/396. [Accessed: 14-May-2018].
[66] “Parse Server - Client Push Integration.” [Online]. Available: https://github.com/parse-community/parse-server/issues/1525. [Accessed: 16-May-2018].
[67] “bcomeau/parse-server-mailgun-adapter-template.” [Online]. Available: https://github.com/bcomeau/parse-server-mailgun-adapter-template. [Accessed: 16-May-2018].
[68] “What is Load Balancing? | DigitalOcean.” [Online]. Available: https://www.digitalocean.com/community/tutorials/what-is-load-balancing. [Accessed: 12-Jun-2018].
[69] “What does 1,000 clients per second mean?” [Online]. Available: https://support.loader.io/article/83-what-does-1-000-clients-per-second-mean. [Accessed: 14-Jun-2018].
[70] “Why would my test use twice the number of clients as I specified?” [Online]. Available: https://support.loader.io/article/187-why-would-my-test-use-twice-the-number-of-clients-as-i-specified. [Accessed: 14-Jun-2018].
[71] “Issue #830 - Docker hasn’t been building.” .
[72] “Issue #51 - Do I need one Instances for each App?” .
[73] “Issue #116 - Multiple apps on one server.” [Online]. Available: https://github.com/parse-community/parse-server/issues/116.
[74] “Issue #1525 - Client Push Integration.”