UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

JOÃO CARLOS FERREIRA MARQUES

Desenvolvimento para plataformas móveis utilizando backend como serviço

Estudo de caso do aplicativo Minha UFG

Goiânia
2017

UNIVERSIDADE FEDERAL DE GOIÁS

INSTITUTO DE INFORMÁTICA

AUTORIZAÇÃO PARA PUBLICAÇÃO DE TRABALHO DE CONCLUSÃO DE CURSO EM FORMATO ELETRÔNICO

Na qualidade de titular dos direitos de autor, **AUTORIZO** o Instituto de Informática da Universidade Federal de Goiás – UFG a reproduzir, inclusive em outro formato ou mídia e através de armazenamento permanente ou temporário, bem como a publicar na rede mundial de computadores (*Internet*) e na biblioteca virtual da UFG, entendendo-se os termos "reproduzir" e "publicar" conforme definições dos incisos VI e I, respectivamente, do artigo 5º da Lei nº 9610/98 de 10/02/1998, a obra abaixo especificada, sem que me seja devido pagamento a título de direitos autorais, desde que a reprodução e/ou publicação tenham a finalidade exclusiva de uso por quem a consulta, e a título de divulgação da produção acadêmica gerada pela Universidade, a partir desta data.

Título: Desenvolvimento para plataformas móveis utilizando backend como serviço – Estudo de caso do aplicativo Minha UFG

Autor(a): João Carlos Ferreira Marques

Goiânia, 04 de Julho de 2017.

João Carlos Ferreira Marques – Autor

Marcelo Ricardo Quinta – Orientador

JOÃO CARLOS FERREIRA MARQUES

Desenvolvimento para plataformas móveis utilizando backend como serviço

Estudo de caso do aplicativo Minha UFG

Trabalho de Conclusão apresentado à Coordenação do Curso de Computação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Bacharel em Computação.

Área de concentração: Otimização.

Orientador: Prof. Marcelo Ricardo Quinta

Goiânia
2017

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Códigos de Programas

CAPÍTULO 1

Introdução

O desenvolvimento de aplicações mobile e web, tem se tornado cada vez mais popular, com diversos aplicativos surgindo no mercado diariamente. Com o mercado cada vez mais competitivo, para que uma aplicação continue sendo usada, ela deve se manter útil ao usuário. Uma forma de ser útil é sempre apresentar o conteúdo que usuário deseja de forma rápida.

De forma a atender as exigências dos usuários, se faz necessário a utilização de uma infraestrutura que cubra essas necessidades da aplicação. Porem, para pequenas empresas e desenvolvedores *solo*, criar e manter servidores é custoso quanto a tempo e mão de obra.

Com o objetivo de facilitar o desenvolvimento de aplicações mobile e web, surgem empresas que fornecem *Backend* como um serviço de nuvem. Eliminando a necessidade de criação de uma infraestrutura local, deixando desenvolvedores livre para focar em seus aplicativos.

Este trabalho está organizado da seguinte forma: o Capítulo 2 apresenta a fundamentação teórica, abordando os conceitos relacionados a computação em nuvem e seus provedores de serviço; o Capítulo 4 trás os principais desafios envolvidos no uso de serviços de *Backend* hospedados em ambiente de computação nuvem; o Capítulo 3 apresenta um estudo de caso por meio da implementação do aplicativo Minha UFG; e o Capítulo 5 apresenta as considerações finais.

CAPÍTULO 2

Fundamentação Teórica

2.1 Cloud Computing

Cloud Computing ou computação na nuvem, é um termo que pode ser usado para descrever uma plataforma que provê, configura, reconfigura e desprovê servidores a medida do necessário. Esses servidores podem ser máquinas físicas ou máquinas virtuais e podem ser utilizados para hospedar aplicações que são estendidas para serem acessadas através da internet. Essas máquinas são providas por data centers (centro de dados) que hospedam *web services* e *web applications*, tornando possível o acesso por meio de qualquer navegador web.

No mercado, alguns famosos provedores de computação na nuvem são a Amazon EC2 [15] e IBM Bluemix [12].

Neste capítulo destrincharemos o conceito, taxonomia e exemplos de computação na nuvem, o que servirá de base para o entendimento e discussão deste trabalho.

2.1.1 Taxonomia

Segundo [14], os provedores de serviços em nuvem podem ser separados em categorias, todas seguindo um padrão quanto ao modelo de pagamento, porem com suas devidas peculiaridades:

Software como serviço - SaaS Do inglês *Software as a Service*,Clientes de software como serviço alugam o uso de aplicações que rodam dentro da infraestrutura do provedor do serviço de computação na nuvem. Essas aplicações são geralmente ofertadas para os clientes através da internet e são gerenciadas totalmente pelo provedor do serviço. O maior benefício de SaaS é a manutenção de uma mesma versão do software para todos os cliente. Adicionalmente, novas funcionalidades podem ser integradas pelo provedor através de uma simples atualização centralizada de arquivos e assim disponível para todos os clientes.

Ao contrario do modelo tradicional de software comprado por licença para utilização do produto, em SaaSs, a precificação geralmente é realizada com base em assinaturas de conjunto de serviços.

Infraestrutura como serviço - IaaS Do inglês *Infrastructure as a Service*, entrega recursos de hardware tais como processador, espaço de disco ou componentes de rede como um serviço. Esses recursos são geralmente entregues como uma plataforma virtual pelo provedor do serviço e pode ser acessada através da Web pelo cliente. O cliente tem total controle sobre a maquina virtual e não é responsável por gerenciar a infraestrutura que mantêm essa maquina.

A cobrança nos IaaS é realizada de acordo com fatores como o numero de servidores virtuais, quantidade de dados trafegados, dados armazenados, horas de funcionamento do serviço e outros fatores que variam entre provedores.

Plataforma como serviço - PaaS Do inglês *Plataform as a Service*, é um modelo intermediário entre IaaS e SaaS, onde o desenvolvedor tem toda as vantagens de ter seu software em nuvem e de não precisar de uma infraestrutura local. Os provedores de PaaS oferecem a plataforma de uma aplicação como serviço. Assim os clientes podem disponibilizar software usando as ferramentas e linguagens de programação disponibilizadas pelo provedor e ter controle sobre suas aplicações via um painel de controle.

Armazenamento como serviço - STaaS Do inglês *Storage as a Service - STaaS*, é um modelo de negócio em que o fornecedor do serviço aluga parte da sua infraestrutura de armazenamento de dados para o cliente e é mantido com base em uma assinatura. Como o provedor possui uma estrutura escalável, ele é capaz de ofertar armazenamento a um preço mais compensador. Sem levar em conta o custo de pessoas e manutenção necessário para manter um servidor de armazenamento próprio. STaaS são geralmente usados quando se quer fazer um backup remoto. Um dos pontos fracos desse serviço é a necessidade de uma grande largura de banda para utilizar o armazenamento em nuvem.

Segurança como serviço - SECaaS Do inglês *Security as a Service*, é um modelo de negócio em que provedores integram seus serviços a infraestrutura de terceiros cujo objetivo é adicionar funcionalidades de segurança ao software em si, como *firewall*, verificação de anti-vírus ou tráfego de informação criptografada.

Dados como serviço - DaaS Do inglês *Data as a Service* é baseado no conceito de que o produto (dados) pode ser providenciado sob demanda pelo usuário, não importando localização geográfica ou separação organizacional entre provedor e consumidor.

Ambiente de testes como serviço - TEaaS Do inglês *Test Environment as a Service* e geralmente referenciado como “ambiente de teste sob demanda”, é um ambiente distribuído de testes em que software e os dados associados são hospedados e

executados na nuvem afim de realizar verificações, validações ou o teste de software em si. Em geral, este tipo de software é utilizado por equipes que no podem ter acesso a dispositivos ou contextos específicos do usuário final, incluindo testes de escalabilidade. Um exemplo de TEaaS é o Firebase Test Lab [9].

***Backend* como serviço - BaaS** Do inglês *Back-end as a Service* e também conhecido como *Mobile Back-end as a Service* (MBaaS), é um modelo que provê para desenvolvedores de plataformas web e móvel, uma maneira de ligar suas aplicações a uma plataforma de *backend* e armazenamento na nuvem, além de prover funcionalidades como gestão de usuários, notificações e integração com serviços de redes sociais. Esses serviços são providos via o uso de Software Development Kits (SDKs) e Application Programming Interfaces (APIs).

2.2 Aprofundando em BaaS

O conceito de BaaS é relativamente recente no mundo de computação em nuvem [13], sendo que no mercado a maioria das *startups* de BaaS são datadas do começo de 2011 ou depois. É um conceito que surgiu da frustração em implantar nas plataformas de IaaS e de PaaS que não oferecem o que é necessário para plataformas móveis. BaaS é sobre abstrair as complexidades de lançamento e gerenciamento da sua própria infraestrutura e trazer recursos que almejam exatamente o que desenvolvedores precisam para construir as próximas gerações de aplicativos moveis [13].

BaaS tem a mesma intenções do PaaS: acelerar o processo de desenvolvimento. Porem, BaaS se concentram em trazer um infraestrutura que escala automaticamente e otimiza uma série de recursos essenciais que desenvolvedores precisam, tais como ferramentas de conteúdo, dados, mensagens e APIs de terceiros tais como Facebook[3], Twitter[17] e Dropbox[2].

Em tempos não muito distantes ou até atuais, empresas hospedavam seus próprios servidores de infraestrutura que são responsáveis por todo o *backend*, assim ficando responsável pelos desafios relacionados a gerencia de TI tais como disponibilidade, escalabilidade e manutenção por exemplo. Esse cenário vem se modificando ao longo dos últimos anos. Empresas estão migrando para o modelo de Software as a Service(SaaS), onde serviços de infraestrutura são fornecidos por outra empresa, passando para o fornecedor a responsabilidade de manutenção do serviço.

Agilização no desenvolvimento de novas soluções, menor custo inicial, integridade e consistência dos dados são as principais vantagens do modelo de SaaS.

A maioria dos provedores de BaaS, fornece segurança, autorização, autenticação, escalabilidade, disponibilidade e manutenção a um custo inicial baixo, bastando apenas a configuração de pequenos arquivos para se ter *backend* funcionando. Porém, este

não é um serviço gratuito, o custo varia de acordo com a utilização e a mudança de um provedor para outro nem sempre é realizada de maneira fácil, devido a possíveis diferenças arquiteturais.

2.2.1 Definição

BaaS é um tipo de SaaS que prevê o suporte para que uma ou mais aplicações funcionem sem a necessidade do desenvolvimento *backend*. Este tipo de serviço, no geral, provê armazenamento

Este trabalho tem como objetivo a análise e discussão das funcionalidades e desafios referentes ao serviço Firebase. Visto que este é considerado o mais utilizado da atualidade. **??**

2.2.2 Funcionalidades / Oportunidades

As funcionalidades dos *back-ends* como serviços variam de acordo com a empresa que está provendo o serviço e com qual finalidade o serviço foi contratado, dentre os serviços mais comuns estão armazenamento de dados, gerenciamento de usuários, notificações assíncronas e integração com redes sociais [13], [16].

Gerenciamento de Usuários

Sistemas podem possuir usuários cadastrados, e atribuir permissões aos mesmos. Isso pode ser realizado por meio de Gerenciamento de Usuários. BaaS oferecem serviços diferentes de acordo com as necessidades do cliente. Firebase por exemplo, fornece o Firebase Authentication [5], que permite login anônimo, via e-mail e senha ou usando provedores populares de identidade federada tais como o Google, Facebook, Twitter entre outros. Firebase Authentication utiliza os padrões da industria tais como OAuth 2.0 e OpenID Connect, facilitando a integração com um *backend* personalizado caso necessário.

Um usuário logado no seu aplicativo é representado por um objeto do tipo Firebase User. Essa classe possui um conjunto de propriedades básicas - ID único, e-mail principal, nome e a URL de uma foto - que são armazenados no banco de dados da aplicação. Caso seja necessário, informações adicionais podem ser armazenadas dentro do Firebase Realtime Database [8]. Firebase User mantém informações dos diferentes provedores usados para autenticar, permitindo que seja possível atualizar propriedades que faltavam, usando informações de outros provedores.

Gerenciamento de Dados

Usuários estão sempre produzindo e consumindo conteúdos, que podem ser desde arquivos diversos como fotos, documentos, ou dados que podem ser do tipo chave valor, endereço - rua. Assim BaaS oferecem APIs que facilitam para o desenvolvedor o acesso e armazenamento desses dados [13].

No Firebase existem duas APIs que tratam de armazenamento de dados, Firebase Storage [6], que trata do armazenamento e arquivos e o Firebase Realtime Database que trata do armazenamento de conjuntos chave-valor.

O Firebase Realtime Database, é um banco de dados NoSQL oferecido pela plataforma do Firebase, onde todos os dados são armazenados no formato JSON e disponibilizados em tempo real para todos os clientes que estão conectados. A API trata de sincronização e armazenamento off-line, então se ocorrer uma atualização nos dados e o cliente não estava conectado, ele ainda poderá utilizar dos dados que estavam em cache e posteriormente, quando uma conexão for estabelecida, os dados serão sincronizados [8].

O Firebase Storage, é a plataforma utilizada para armazenamento de arquivos. Ela é fortemente integrada ao Google Cloud Platform, [6].

Gestão de Permissão

A funcionalidade de banco de dados como serviço é uma das mais utilizadas dentro dos BaaS [referencia para essa afirmação]. Dessa maneira, por questões de manutenção de integridade, privacidade e segurança, os sistemas do mercado oferecem diferentes maneiras para a a definição de regras de acesso e validação. O objetivo é deixar livre a tentativa de persistência de informação para todos os usuários e com qualquer tipo de dado. Porém, tudo o que for submetido a gravação passará por um controle restrições de acesso e limites internos.

No Firebase, esse controle de segurança e regras é implementado usando um conjunto de regras definidas pelo programador através de descritores JSON. Estas são aplicadas a cada operação de leitura ou escrita, fazendo com que uma requisição seja valida somente se essas regras permitirem. [11].

No exemplo 2.1 de código, todos os usuários podem ler o nó *foo*, mas ninguém pode escrever nele.

Código 2.1 Exemplo de Regras

```
{
  "rules": {
    "foo": {
      ".read": true,
      ".write": false
    }
  }
}
```

Objetos Customizáveis

Objetos do mundo real são complexos, e em muitas das vezes somente os tipos primitivos de dados não são suficientes para representar os atributos desses objetos. Assim necessitando de objetos que permitam acessar dados (DAO, do inglês Data Access Object) sejam criados justamente para converter essa estrutura complexa em uma que o banco de dados consegue entender.

No Firebase, os dados são armazenados como JSON assim, a lista de tipos primitivos é composta de tipos que podem ser convertidos para JSON diretamente, contendo String, Long, Double, Map<String,Object> e List<Object> isso falando de Java. Para usar objetos personalizados, a classe que o implementa deve de utilizar ter um construtor vazio e métodos públicos de acesso para os atributos dessa classe. [10]

API

Em alguns casos, é necessário implementar lógicas de negócio mais robustas e que exigem mais processamento do dispositivo do cliente, ou precisam ser ocultadas por questões de segurança. Assim, alguns provedores possibilitam que o desenvolvedor implemente sua própria API sobre o serviço contratado, estendendo ou personalizando o comportamento de uma funcionalidade já existente. A geração de *thumbnail* de imagens é um exemplo de operação que pode ser realizada no servidor para economizar processamento dos clientes.

Para resolver esse problema, o Firebase desenvolveu o Cloud Functions, que permite a criação de um serviço hospedado por eles onde operações custosas quanto ao processamento ou que não deveriam ser feitas no cliente, são realizadas nos servidores do próprio Firebase.

Escalabilidade e Disponibilidade

Quando se tem um servidor interno, uma das principais preocupações é com a escalabilidade e a disponibilidade. Ou seja, como que o seu serviço vai funcionar quando esse servidor estiver sobre grande carga, que pode ser gerada por vários usuários ativos ao mesmo tempo, várias operações de acesso ao banco de dados ocorrendo ao mesmo tempo ou até mesmo falta de recursos. Um sistema é dito escalável se permanece eficiente quando há um aumento significativo no número de recursos e no numero de usuários [1]. A disponibilidade de um sistema é a medida da proporção de tempo em que ele está pronto para uso [1].

BaaS investem em infraestrutura para mantêm o serviço escalável e disponível na medida que a demanda dos usuários vai crescendo, deixando o programador responsável por estruturar os dados e as consultas, de forma a minimizar a quantidade de requisições e a quantidade de nós a serem acessados para realizar uma consulta.

No Firebase, os problemas de usuários simultâneos e escalabilidade são resolvidos pela plataforma, porém existe um custo de acordo com o que o cliente precisa. Esse custo deve ser acertado entre as partes contratantes.

2.2.3 Precificação

Os serviços ofertados pela empresa também possuem um custo de manutenção, em geral, quantidade de dados transferidos e armazenados, e quantidade de requisições a API. Custo esse que é repassado para os contratantes do serviço.

O *backend* como serviço, assim como todo software que roda como serviço na nuvem, é mantido por uma empresa que obviamente vai cobrar pelo serviço, por tanto, deve haver uma preocupação com o valor a ser pago para o fornecedor dessa solução.

O modelo de precificação para provedores de BaaS ainda está evoluindo. Muito dos provedores, adotou o custo de armazenamento e chamadas de API como sendo o seu modelo, porem ainda existe muito o que debater sobre o que é modelo mais proveitoso para os desenvolvedores e que gera lucro para o provedor.

Atualmente alguns dos provedores de BaaS operam usando serviços de terceiros, tal como Amazon Web Services, o que faz com que eles cobrem pelos custos da plataforma. Os modelos mais utilizados para cobrança são chamadas de API e Armazenamento.

Os itens mais utilizados na precificação são:

Chamadas de API Cobrar do desenvolvedor pela quantidade de chamadas a APIs. O valor de um BaaS está nos recursos que as APIs fornecem, o modelo de negocio é construído sob o valor dessas APIs.

Armazenamento Armazenamento é o maior custo para os provedores de BaaS. Para os desenvolvedores, é comum o armazenamento de dados, arquivos e objetos usando o *storage* do provedor. Grande parte dos provedores BaaS rodam em nuvem, assim esse armazenamento é sempre uma solução em nuvem.

Apesar de esses serem os métodos mais comuns para cobrança, existem vários outros requisitos por onde o provedor de BaaS pode cobrar.

O foco de vários BaaS é o mundo de aplicativos móveis, assim, faz sentido cobrar dos desenvolvedores por quantidade de usuários finais.

O numero de chamadas da API e quantidade de usuários ativos, é uma forma de medir o sucesso do aplicativo, e assim o desenvolvedor está apto a pagar um pouco mais pelos serviços do provedor.

A cobrança pelo provedor BaaS, é um processo que não tem um padrão estabelecido. A questão do custo é um reflexo do modelo de negocio de IaaS utilizada pelo provedor. Mas existem outras abordagens comuns de cobrança entre BaaS:

Usuários ativos Para alguns, o numero de usurários ativos, é uma maneira sensível de cobrar pelos serviços dos BaaS. Cobrar desenvolvedores por cada usuário ativo é uma maneira de deixar os custos do *backend* de acordo com o modelo que foi escolhido pelo desenvolvedor. Cada provedor define diferente o que ele considera usuário ativo.

Analytics Muitos provedores fornecem ferramentas de análise grátis como parte do serviço e cobram por uma versão *premium* ou melhorada da ferramenta. Ter em mãos os dados das ferramentas de analise ajudam o desenvolvedor a localizar qual parte do seu aplicativo necessita de mais foco.

Número de aplicações Desenvolvedores criam seus aplicativos usando os serviços dos BaaS, assim, cobrar pelo número de aplicações faz sentido. Alguns provedores BaaS, fornecem uma chave para um aplicativo grátis e você deve pagar uma taxa por cada chave extra requisitada.

Largura de banda utilizada Largura de banda é mais um dos custos passados juntamente com o provedor de IaaS. É um custo que as vezes vem com uma sobre taxa

Beta Muitos provedores de BaaS ainda estão em fase de desenvolvimento, assim eles possuem um modelo de cobrança e não cobram pelos serviços oferecidos.

Campanhas Os provedores realizam campanhas de marketing e cobram por esse serviço.

Chats Comunicação por chat é provida por BaaS através de soluções de terceiros. Assim passando para os desenvolvedores uma parte do custo. A comunicação via chat, é uma função que agrega valor ao aplicativo, assim, compensa pagar um pouco por ela e ter usuários mais satisfeitos.

E-mails entre as tecnologias de comunicação, e-mail é a mais fácil de se medir, associar valor e cobrar por. Alguns provedores de BaaS estão terceirizando seus serviços de

e-mail para servidores dedicados para esse serviço passando para desenvolvedores parte dos custos.

Enterprise Contratos com empresas são a maior fonte de renda para provedores BaaS, assim criar planos empresarias se faz necessário, onde a contratante pagará pelos serviços que for mais utilizar eliminando ou aumentando os limites de utilização que certas APIs impõem em alguns serviços.

Acesso a ferramentas adicionais Alguns provedores alegam que custos de chamadas de API, armazenamento, entre outros, são insignificantes. Defendendo que desenvolvedores deveriam ser cobrados apenas pelas ferramentas(*features*). Assim, criando um modelo onde os desenvolvedores pagam pelas ferramentas que serão disponibilizadas pelo provedor, por terceiros ou atraves de uma loja virtual.

Push Notifications *Push notifications* além de ser um dos serviços mais fornecidos pelos provedores BaaS, cerca de 2/3 dos provedores possuem alguma forma de notificação *push*, é um dos mais custosos, assim os provedores passam parte do custo para os desenvolvedores além de que alguns podem cobrar uma taxa extra pelas notificações.

SSL Alguns provedores BaaS fornecem SSL em seus pacotes de preços. Os custos para implementação de SSL no provedor são altos, mas alguns veem como um recurso aceitável para se pagar por.

Support Muitos provedores fornecem suporte como um serviço a parte aonde o desenvolvedor deve pagar por.

Sincronizações Diversos provedores BaaS incluem a possibilidade de sincronizar conteúdo, dados e mensagens. Essas plataformas estruturam seu preço em torno da quantidade de sincronizações entre a plataforma e dispositivos. Sincronização é uma *feature* valorizada por desenvolvedores pois evita que o tempo gasto em desenvolver tal ferramenta além de proporcionar uma melhor experiência para o usuário.

Estudo de caso: Precificação do Firebase

O Firebase[4] é o provedor BaaS que escolhemos utilizar para o aplicativo Minha UFG. Ele divide o pagamento em forma de planos, em que todos possuem acesso às funcionalidades de Analytics, App Indexing, Authentication, Cloud Messaging, Crash Reporting, Dynamic Links, Invites, Notifications & Remote Config, porem limitados em Realtime Database, Storage, Cloud Functions, Hosting, Test Lab e Google Cloud Platform.

O plano mais básico é o Spark, gratuito, seguido pelo Flame(25 dólares mensais) e Blaze. O Blaze é oferecido sob demanda. Paga-se pelo que se usa. Os limites de cada plano, como disponibilizados em [7], estão representados na Tabela **??**:

		Planos		
Funcionalidade	Restrições	Spark	Flame	Blaze
Realtime Database	Conexões Simultâneas	100	Ilimitado	Ilimitado
	GB Armazenados	1 GB	2.5GB	($)5/GB/mês
	GB Baixados	10GB/mês	20GB/mês	($)1/GB
	Backups Automatizados	não	não	sim
Armazenamento	GB Armazenados	5 GB	50 GB	($)0.026/GB
	GB Baixados	1 GB/dia	50 GB/mês	($)0.12/GB
	Operações de Upload	20,000/dia	100,000/dia	($)0.10/mil
	Operações de Download	50,000/dia	250,000/dia	($)0.01/mil
Cloud Functions	Invocações	125,000/mês	2,000,000/mês	($)0.40/milhão
	GB-segundos	40,000/mês	400,000/mês	($)0.0025/mil
	CPU-segundos	40,000/mês	200,000/mês	($)0.01/mil
	Outbound Networking	google-only	5GB/mês	($)0.12/GB
Hosting	GB Armazenados	1 GB	10 GB	($)0.026/GB
	GB Transferidos	10GB/mês	50 GB/mês	($)0.15/GB
	Custom domain hosting & SSL	sim	sim	sim
Testing	GB Armazenados	15 testes/dia	15 testes/dia	horas/dispositivo
Google Cloud Platform	Usar BigQuery & outros IaaS	não	não	sim

Tabela 2.1: *Precificação Firebase*

CAPÍTULO 3

Utilização de técnicas em caso real: MinhaUFG

Este capítulo apresenta o estudo de caso do uso do *backend* como serviço em uma aplicação real, MinhaUFG.

3.1 Justificativa e escopo

Em um cenário que os dados de interesse de estudantes e servidores da UFG encontravam-se "dispersos" por portais das unidades da UFG, cartazes, impressos em geral, e-mails, telefonemas e contatos pessoais, a localização de uma informação relevante tornava-se impraticável para quem precisa acessá-la. Tal contexto, fomentado pela ausência de instrumento alternativo de interação foi contemplado pelos entregáveis do projeto "MuralUFG", posteriormente batizado e lançado como MinhaUFG.

O aplicativo Android MinhaUFG é uma alternativa para os métodos atualmente usados para a divulgação da informação, convergindo a informação de diferentes perspectivas em uma só aplicação. Essas informações são agrupadas e divididas em categorias de serviços, facilitando a busca das mesmas pelos interessados. Veja a Figura 3.1.

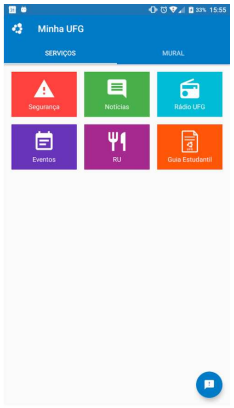

Figura 3.1: *Imagem da tela inicial do aplicativo MinhaUFG.*

Atualmente com seis funcionalidades e cerca de seiscentos usuários utilizando-a diariamente, a aplicação contém as seguintes funcionalidades:

Segurança : Contém lista de telefones de segurança e serviço de chamado de vigilante/envio de denúncia com foto e devido acompanhamento pela central. Este serviço foi criado especialmente para a aplicação.

Notícias : Disponibiliza a lista de notícias e o detalhamento de cada uma das matérias, divididas por Campus Universitário. A fonte das notícias é o sistema weby.

Rádio Universitária : Acesso a Rádio Universitária e sua programação diária. A fonte do *streaming* é o sistema da própria UFG, porém o sistema de visualização da programação foi criado especialmente para a aplicação.

Eventos : Listagem dos eventos cadastrados no sistema interno da UFG, podendo ser filtrados por dia.

Restaurantes Universitários : Listagem de restaurantes universitários por campus, além do cardápio semanal de cada um, incluindo jantar. Foi criado especialmente para a aplicação.

Guia estudantil : Guia da vida acadêmica para novos estudantes. Foi criado especialmente para a aplicação com a colaboração da ASCOM.

O aplicativo hoje tem mais de 5 mil *downloads* na Google Play Store, tendo nota acima de 4.5, considerada alta. Novas funcionalidades serão futuramente implementadas, de acordo com a integração de novos serviços criados pela universidade.

3.2 Arquitetura

Dentro da visão de implantação, o projeto MinhaUFG possui uma arquitetura descentralizada composta de uma agregação de vários servidores dos serviços da universidade somados a serviços de um *backend* como serviço criado especialmente para cobrir os sistemas não existentes. Logo, não há um servidor dedicado e centralizado para recuperação da informação. A comunicação é realizada de acordo com o serviço, sendo também complementada por serviços do próprio telefone. Veja um exemplo na figura 3.2.

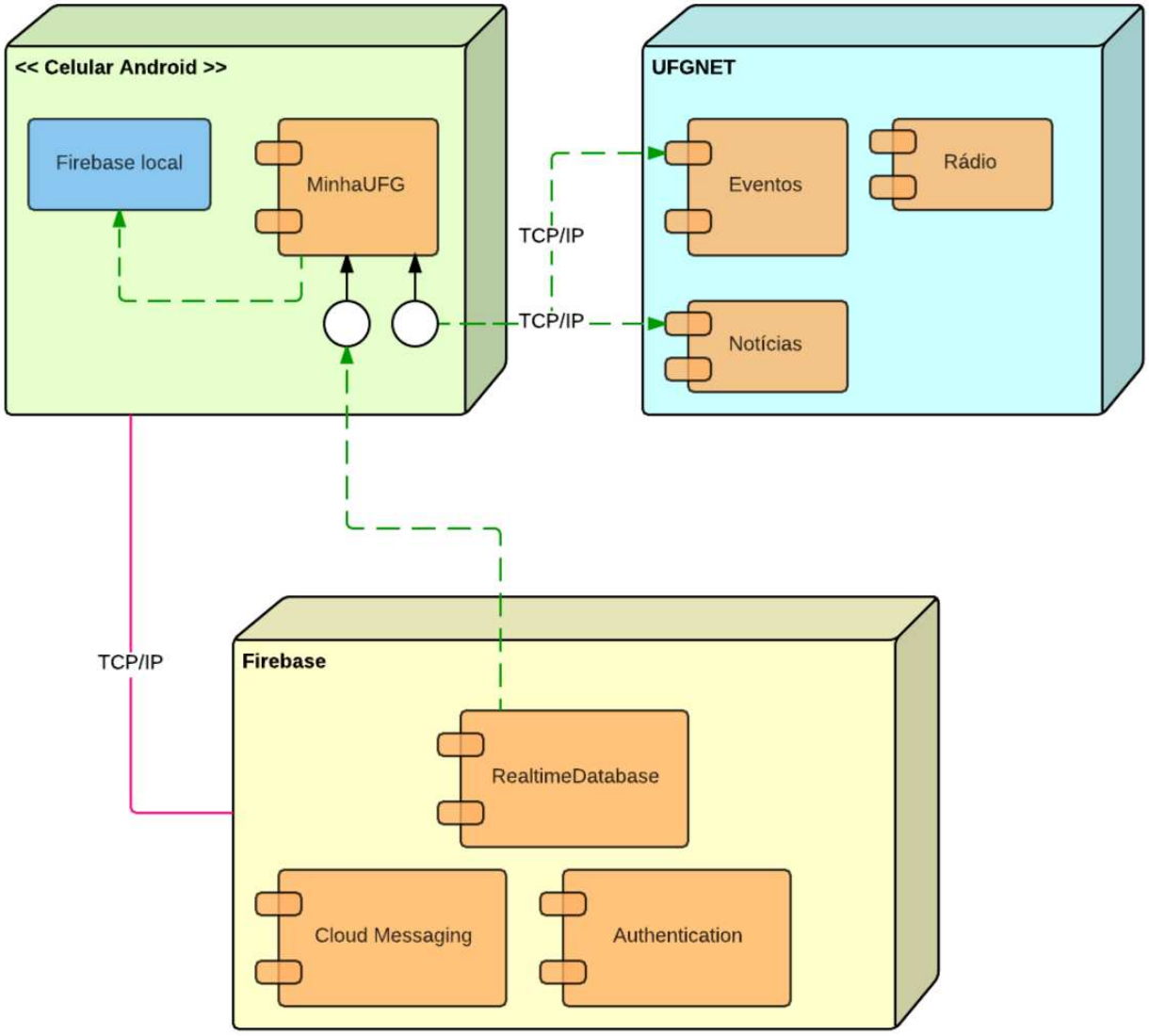

Figura 3.2: *Arquitetura de implantação do MinhaUFG.*

A arquitetura de pacotes aplicativo MinhaUFG é modular. Tal decisão tem o objetivo de diminuir o acoplamento de funcionalidades e facilitar a manutenção do código, principalmente no que se refere ao modelo do banco de dados. Cada módulo é responsável por gerenciar segmentos de dados específicos, havendo pouca intersecção entre as informações nos grupos de serviços. Para realizar tal implementação, o padrão arquitetural escolhido foi o MVP (Model-View-Presenter), cuja característica de desacoplamento entre controle de interface e lógica da aplicação.

A figura 3.3 ilustra este padrão arquitetural.

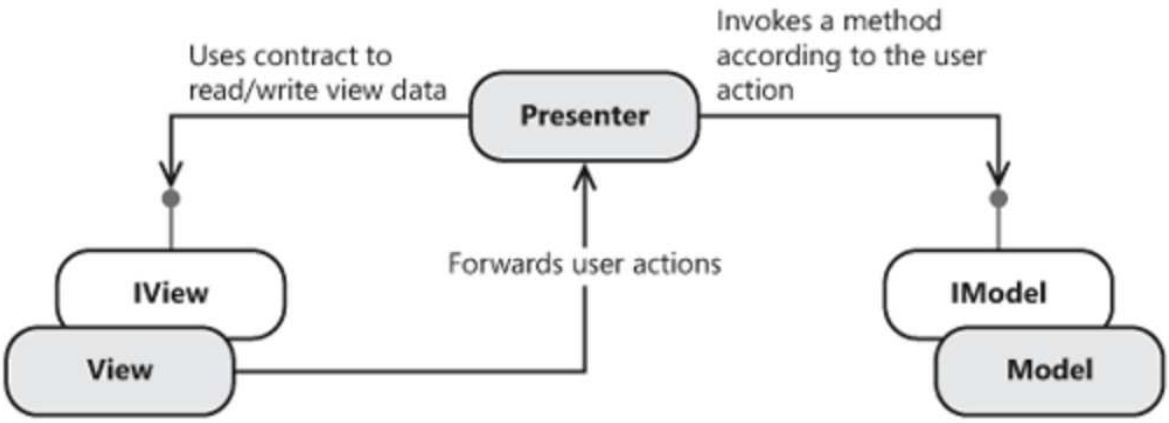

Figura 3.3: *Padrão arquitetural MVP e a relação de seus componentes.*

3.3 Implementação

Para armazenar os dados de ambos os módulos, estamos usando os serviços de Storage e de Realtime Database providos pelo Firebase, este que além de ser provedor de serviços na nuvem também oferece *backend* como sendo um serviço.

Para representar um restaurante, foi criada uma estrutura que continha o nome, uma breve descrição, uma URL referente a imagem, um campus e uma referência que indica qual seria o local ao qual o restaurante pertence. Essa referência é feita usando o identificador da "tabela" de locais, que contém informações sobre qual prédio, campus, endereço, e coordenadas de latitude e longitude.

3.4 Resultados

No estágio inicial do desenvolvimento, desenvolvemos os módulos "restaurantes" e "locais". O módulo "restaurantes" tem como objetivo mostrar qual é o cardápio dos restaurantes localizados nas proximidades dos campus da UFG. A princípio, apenas os restaurantes universitários, R.U. Campus Samambaia e R.U. Campus Colemar Natal e Silva, estão inclusos na atual lista de restaurantes.

O módulo "locais", apresenta uma "lista" que contém lugares, tais como um instituto, um prédio, uma sala. Locais podem existir dentro de outros locais, exemplo, a CAI,

CAPÍTULO 4

Desafios técnicos no uso de BaaS e suas soluções

Possuir um servidor escalável para receber todos os usuários de forma simultânea é um grande desafio, iniciando pela implantação ou pela do provedor do serviço. Aliado a este problema, existe a necessidade de minimizar o custo monetário do serviço contratado e o tempo de reposta do servidor.

Neste capítulo discutiremos alguns problemas e soluções relacionadas a manutenção de serviços que utilizam *backend* como serviço pagando o mínimo possível. Todos eles estão divididos por área.

4.1 Requisitos Não-Funcionais

Um requisito não-funcional, tem como objetivo atender a requisitos do sistema que não se referem a funcionalidades do negócio, mas que fazem parte do escopo do sistema.

Nessa seção será apresentado quatro requisitos não funcionais sobre BaaS: flexibilidade na troca de provedor, segurança, sincronização de dados e escalabilidade.

4.1.1 Flexibilidade na troca de provedor

Dentro da manutenção de software uma ação que pode ser realizada é a troca de fornecedor de algum serviço, por exemplo a troca do *backend* como serviço utilizado. A troca de provedor pode ser complicada. Os provedores podem utilizar diferentes estruturas de dados ou métodos de interação com as APIs. Caso isso ocorra, é necessário repensar os componentes responsáveis pela comunicação com o servidor e a criação dos objetos.

Uma solução para esse problema é utilizar o padrão de projeto *facade*, também conhecido como *façade*. O padrão cria uma interface intermediária, que modela a comunicação entre o servidor e o cliente, de forma que ao alterar o provedor, é necessário apenas reescrever a implementação dessa interface.

No Minha UFG, foi adotado o padrão de projeto *façade*, no presente momento, uma parte dos dados vem do Firebase e a outra de servidores internos, toda essa comuni-

cação externos é encapsulada seguindo o padrão *façade*. Em 4.1 é possivel ver o código da interface provida para a funcionalidade de notícias.

Código 4.1 Interface *facade* para Notícias

```
1  public interface NewsRepository {
2
3      /**
4       * pesquisa noticias
5       * @param region {Region} objeto que contem as
6       * URL's necessarias para a pesquisa
7       * @param callback {NewsCallback} callback quando
8       * existem ou não resultados
9       */
10     void queryNewsByRegionalUri(
11         Region region,
12         NewsCallback callback);
13
14     /**
15      * pesquisa por paginas de noticias
16      * @param region {Region} objeto que contem as
17      * URL's necessarias para a pesquisa
18      * @param pageNum {int} pagina a ser requisitada
19      * @param callback {NewsCallback} callback quando
20      * existem ou não resultados
21      */
22     void queryNewsByRegionalUriAndPageNumber(
23         Region region,
24         int pageNum,
25         NewsCallback callback);
26
27     interface NewsCallback {
28         void onSuccess(ArrayList<NewsStory> result);
29         void onError(FailedToGetResultException exception);
30     }
31
32 }
```

4.1.2 Segurança

Usuários estão sempre criando conteúdo, desde conteúdos simples como um uma foto, vídeo ou até mesmo escrevendo um texto de sua dissertação. Visto que o conteúdo

desses dados pode ser sensível, deve existir uma preocupação quanto ao armazenamento, o tráfego e o acesso desses dados. A segurança é uma das grandes preocupações que provedores de BaaS devem ter em mente quando projetar seu modelo de negócio.

Os BaaS em geral tratam de utilizar um canal seguro via TTS para transmissão de informações e portanto o cliente deve estar preparado para esse tipo de comunicação. Em relação aos dados, de modo geral, a segurança deve ser implementada pelos desenvolvedores, os provedores BaaS não lidam com o aspecto de criptografia. Portanto boa parte da lógica de segurança fica implementada na aplicação cliente. Deve se ter um cuidado de como é feita a implementação para não expor a todo o sistema pois o código está no cliente.

No Firebase a transmissão de dados é realizada utilizando uma conexão segura, todos os dados, antes de serem salvos ou recuperados do banco de dados, passam pelas regras de permissão, tal como exemplificado em 2.1. Assim previne-se que usuários sem autorização de acesso escrevam ou recuperem dados que eles não tem acesso. Lembrando que a definição dessas regras fica a cargo do programador.

4.1.3 Sincronização de Dados

A garantia de sincronização de dados em todos os dispositivos é muito importante pois trata de diferentes versões da informação em diferentes clientes.

Alguns serviços fazem uma camada de transparência onde os dados são atualizados em *background* sem a necessidade da intervenção do usuário. Uma abordagem comum é manter informações de estado, assim trafegando apenas as mudanças que ocorreram entre estados.

Para garantir a integridade das informações armazenadas ou transferidas, serviços geralmente salvam a informação que possui o *timestamp* mais recente, assim ficando para o desenvolvedor a tarefa de estruturar os dados para evitar a sobreposição desses dados caso esse seja necessário em sua aplicação.

Na API do Firebase Realtime Database [8], toda atualização de dados passa pelo servidor do Firebase, permanecendo a última modificação. Assim todos os clientes conectados que estiverem interessados nesses dados, receberão a atualização em questão de milissegundos. Caso o cliente perca a conexão com a internet, a API armazena em cache os dados prévios, e assim que o cliente recuperar a conexão, os novos dados são baixados de forma transparente.

4.1.4 Escalabilidade

Para oferecer um serviço que seja escalável e que esteja disponível na maior parte do tempo para o cliente, é necessário um grande investimento em infraestrutura por

parte dos BaaS e é importante ter em mente que para cada fornecedor existem máximos e mínimos com relação a estrutura fornecida.

O Firebase, por exemplo, consegue lidar com centenas de milhares de usuários conectados ao mesmo tempo tanto para acesso ao banco de dados quanto para o armazenamento de arquivos, mas caso um dos nós passa a ter mais de 10 milhões ou mais de filhos (digamos que você esteja escrevendo um aplicativo de mensagens e colocou todas as mensagens de um certo período em um mesmo ramo '*/root/messages*') o sistema passa a ter um decaimento da performance. A razão para essa limitação é que o Firebase é armazenado em memória "quente" então quando se tenta acessar um nó, todos os filhos são baixados e isso acaba lotando a memória causando uma queda no desempenho. Uma solução para esse problema seria estruturar seus dados de forma que impeça isso de acontecer (por exemplo, arquivando chats que são mais antigos que 1 dia, uma semana ou um mês).

Outros fatores que devem ser considerados para se ter um sistema escalável no Firebase são os padrões de acesso. Digamos que para gerar a lista de usuários online desse aplicativo de mensagens, existe a possibilidade de iterar sobre toda a lista de usuários verificando um atributo para decidir se o usuário está online, uma alternativa é denormalizar os dados e ter um nó separado '*online-users*'. Ao utilizar a segunda abordagem, visto que apenas os dados realmente necessários são recuperados, a quantidade de informações trafegada é reduzida. Um contra dessa abordagem é que ela requer mais espaço no banco. Fica para o desenvolvedor decidir qual tipo de abordagem seguir. Veja 4.2.2.

4.2 Relacionados aos custos

Quando um serviço de BaaS é contratado, existem varios fatores que influenciam diretamente no custo do serviço. Assim o desafio é minimizar o valor a pago.

Nessa seção será apresentado três requisitos relacionados ao custo dentro de BaaS: minimização de tráfego, minimização de armazenamento e minimização de consultas.

4.2.1 Minimizar Tráfego

Um dos fatores que impacta diretamente nas taxas pagas para o fornecedor do BaaS é a quantidade de dados trafegados. Quanto mais dados trafegados, maior a quantia a ser paga. Isso vem da limitação que provedores possuem quanto a sua infraestrutura, assim passando parte dos custos operacionais para o desenvolvedor.

No Firebase os dados trafegados são separados por duas APIs distintas, o Firebase Realtime Database [8], que é utilizada para trafegar objetos da sua a aplicação, e o Firebase Storage [6] que é usado para transferir arquivos tal como musicas, fotos, vídeos dentre outros. Ambas APIs possuem seus limites de acordo com o plano escolhido como pode ser observado na Tabela **??**.

Para amenizar o valor final da conta do mês, pode-se adotar medidas tal como a denormalização de dados, onde a estrutura de dados pode ser reescrita de maneira a criar uma árvore de consulta menos profunda, veja no exemplo 4.2 a estrutura de um chat não normalizado e em 4.3 essa mesma estrutura, porem normalizada. Assim, ao realizar uma consulta, apenas os dados necessários são baixados, minimizando banda transferida entre cliente e servidor, o que também ajudar a preservar dados moveis do cliente.

Outra opção, é contratar mais banda a medida que for necessário, seja mudando o plano ou alterando o contrato diretamente com o provedor.

Código 4.2 Exemplo de dados não normalizados.

```
{
  // Essa é uma arquitetura de dados mal aninhada,
  // porque iterando sobre os filhos de "chats"
  // para obter o titulo das conversas
  // requer uma transferência potencial de centenas de
  // megabytes de mensagens.
  "chats": {
    "one": {
      "title": "Historical Tech Pioneers",
      "messages": {
        "m1": {
            "sender": "ghopper",
            "message": "Relay malfunction found. Cause: moth." },
        "m2": { ... },
        // uma lista bem longa de mensagens
      }
    },
    "two": { ... }
  }
}
```

Código 4.3 Exemplo de dados normalizados.

```
{ // Conversar contêm apenas meta dados
  // sobre cada conversa armazenados
  // sobre o ID unido de cada "chat"
  "chats": {
    "one": {
      "title": "Historical Tech Pioneers",
      "lastMessage": "ghopper: Relay malfunction found. Cause: moth.",
      "timestamp": 1459361875666
    },
    "two": { ... },
    "three": { ... }
  },
  // Os membros da conversa são facilmente acessíveis
  // e armazenados sobre o ID da conversa.
  "members": {
    // mais sobre índices como este abaixo
    "one": {
      "ghopper": true,
      "alovelace": true,
      "eclarke": true
    },
    "two": { ... },
    "three": { ... }
  },
  // Mensagens são separadas de dados que podemos
  // querer para iterar rapidamente porem
  // ainda facilmente paginada, recuperada e
  // organizada pelo ID da conversa
  "messages": {
    "one": {
      "m1": {
        "name": "eclarke",
        "message": "The relay seems to be malfunctioning.",
        "timestamp": 1459361875337
      },
      "m2": { ... },
      "m3": { ... }
    },
    "two": { ... },
    "three": { ... }
  }
}
```

Vale levar em consideração que espaço de disco é barato nos dias atuais, já o tempo do usuário não. Então para o usuário final, vale a pena pensar em replicar dados com a finalidade de agilizar uma ou mais consultas.

4.2.2 Minimizar Armazenamento

BaaS em geral oferecem uma cota de espaço em disco limitada pelo serviço contratado, existem limites de transferência e limites de armazenamento. O armazenamento de arquivos e o tráfego dos mesmos impacta direto na quantia a ser paga no final do mês. Assim o principal desafio é tentar minimizar o tamanho dos arquivos trafegados.

Uma solução que pode ser empregada no armazenamento de arquivos é a compressão dos mesmos. Existem muitos algoritmos de compressão nos quais a perca de qualidade não chega a ser tão impactante para o usuário, assim economizando não apenas a banda usada na transferência dos mesmos, mas também a bateria do dispositivo do cliente. Outra abordagem seria utilizar-se de outro provedor com a única finalidade de armazenamento.

4.2.3 Minimizar Consultas

Outra métrica utilizada por provedores é a quantidade de chamadas de APIs, onde se tem um limite diário ou mensal. Ultrapassar esse limite pode resultar em: indisponibilidade de funcionalidades ou mudança de plano de cobrança.

Caso esse limite seja facilmente atingido, pode-se optar por contratar um plano que ofereça um limite maior, ou armazenar consultas prévias em cache. Outro exemplo que causa o aumento de chamadas, é fazer *pooling* no servidor em uma frequência muito alta. Assim pode-se utilizar de notificações *push* caso o provedor forneça esse serviço. Dessa maneira, existe transferência apenas quando novo conteúdo for disponibilizado.

CAPÍTULO 5

Conclusão

Nesse trabalho foram estudados os *backend* como serviço e as soluções que podem ser desenvolvidas para que a utilização dentro do projeto de um aplicativo móvel possa ser possível.

Evidenciado que os mesmos agilizam o processo de desenvolvimento, pois já implementam serviços complexos.

Mesmo com a documentação escassa e em inglês, curva de aprendizagem do desenvolvedor e a integração com o processo de desenvolvimento da empresa, as diversas soluções citadas se mostraram eficientes para diminuir a quantidade de conexões simultâneas, o armazenamento de dados, a quantidade média de requisição usuários, trafego de dados e as plataformas a serem atingidas pela aplicação em desenvolvimento. Tendo isso em mente, é garantido que o custo do desenvolvimento e manutenção do sistema como um todo irá diminuir, pois etapas como desenvolvimento e manutenção do serviço *backend*, tanto na sua parte lógica como na infraestrutura, saem do processo da organização/desenvolvedor.

Como estudo de caso, as técnicas estudadas foram implementadas dentro de um aplicativo real, Minha UFG, demonstrando que sem a utilização em um projeto da Universidade Federal de Goiás. Dessa maneira, conhecimento teórico e prático puderam se juntar.

Finalmente, devemos também destacar que softwares de *backend* como serviços não são a bala-de-prata. Em alguns casos eles não podem ser aplicados, além do quê a dependência do serviço de empresas que podem acabar com o serviço podem trazer riscos a negócios ou ideias.

Referências Bibliográficas

[1] COULOURIS, G. F.; DOLLIMORE, J.; KINDBERG, T. **Distributed systems: Concepts and Design**. Pearson Education, 2005.

[2] DROPBOX. **Dropbox Developers**. https://www.dropbox.com/developers, 2017. Ãšltimo Acesso: 01/07/2017.

[3] FACEBOOK. **Faceboook Developers**. https://developers.facebook.com/, 2017. Ãšltimo Acesso: 01/07/2017.

[4] FIREBASE. **Firebase**. https://firebase.google.com/, 2017. Ãšltimo Acesso: 01/07/2017.

[5] FIREBASE. **Firebase Authentication**. https://firebase.google.com/docs/auth/, 2017. Ãšltimo Acesso: 01/07/2017.

[6] FIREBASE. **Firebase Cloud Storage**. https://firebase.google.com/docs/storage/, 2017. Ãšltimo Acesso: 01/07/2017.

[7] FIREBASE. **Firebase Pricing**. https://firebase.google.com/pricing/, 2017. Ãšltimo Acesso: 20/05/2017.

[8] FIREBASE. **Firebase Realtime Database**. https://firebase.google.com/docs/database/, 2017. Ãšltimo Acesso: 01/07/2017.

[9] FIREBASE. **Firebase Test Lab for Android**. https://firebase.google.com/docs/test-lab/, 2017. Ãšltimo Acesso: 20/05/2017.

[10] FIREBASE. **Read and Write Data on Android**. https://firebase.google.com/docs/database/android/read-and-write, 2017. Ãšltimo Acesso: 01/07/2017.

[11] FIREBASE. **Understand Firebase Realtime Database Rules**. https://firebase.google.com/docs/database/security/, 2017. Ãšltimo Acesso: 01/07/2017.

[12] IBM. **Cloud Infrastructure, Storage, Security & More - IBM Bluemix**. https://www.ibm.com/cloud-computing/bluemix/, 2017. Ãšltimo Acesso: 01/07/2017.

[13] LANE, K. **Overview of the backend as a service (baas) space**, 2013.

[14] RICARDO, A. **O que Ã© SaaS, IaaS e PaaS em Cloud Computing? (Conceitos bÃ¡sicos)**. https://antonioricardo.org/2013/03/28/o-que-e-saas-iaas-e-paas-em-clou 2013. Ãšltimo Acesso: 01/07/2017.

[15] SERVICES, A. W. **Elastic compute cloud (EC2) - Cloud Server & Hosting - AWS**. https://aws.amazon.com/ec2/, 2017. Ãšltimo Acesso: 01/07/2017.

[16] TOMAZ, J. **Como o BaaS pode ajudar o dia a dia de um desenvolvedor**. https://imasters.com.br/desenvolvimento/como-o-baas-pode-ajudar-o-dia-dia-de-um 2014. Ãšltimo Acesso: 01/07/2017.

[17] TWITTER. **Twitter Developers**. https://dev.twitter.com/, 2017. Ãšltimo Acesso: 01/07/2017.