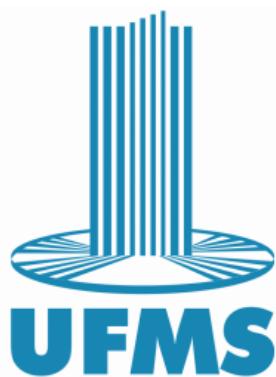


NOTIFIQUE-ME: MENSAGEIRO INSTANTÂNEO UTILIZANDO GEOLOCALIZAÇÃO

Higor de Oliveira Chaves

Bacharelado em Sistemas de Informação

Orientação: Prof. Me. Kleber Kruger



Campus de Coxim
Universidade Federal de Mato Grosso do Sul
11 de dezembro de 2018

Notifique-me: Mensageiro Instantâneo Utilizando Geolocalização

11 de dezembro de 2018

Banca Examinadora:

- Prof. Me. Kleber Kruger (CPCX/UFMS) - Orientador
- Prof. Dr. Gedson Faria (CPCX/UFMS)
- Prof. Bel. Cleiton Gonçalves de Almeida (CPCX/UFMS)

Agradecimentos

A Deus pela vida, saúde e forças para superar as dificuldades.

A meus pais, pelo incentivo e apoio incondicional.

A meus amigos, que mostraram a importância do companheirismo e contribuíram diretamente para minha formação.

A todo o corpo docente do curso de Sistemas de Informação, que além de excelentes professores, mostraram-se grandes amigos.

A todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

Resumo

Por ser importante para o Homem desde os tempos antigos, tanto a comunicação quanto as formas para se obtê-la sempre estiveram em constante progresso. Como resultado deste avanço, atualmente mensageiros instantâneos tem o poder de unir indivíduos de diversas partes do globo, e permitir que a comunicação entre eles seja praticamente em tempo-real. No entanto, para que esta comunicação seja estabelecida, um vínculo deve ser realizado, e para isso uma das partes comunicantes deve possuir alguma informação pessoal do indivíduo o qual deseja se comunicar, como um número de telefone celular ou e-mail. O mesmo conceito ainda se aplica a pessoas fisicamente próximas, pois mesmo estando em um mesmo local, dois indivíduos não podem estabelecer um vínculo se forem desconhecidos entre si.

Pensando nisto, neste trabalho é proposta uma forma diferente de vínculo, onde o único limitador para se iniciar um ambiente de conversa é a proximidade física. Utilizando principalmente as *frameworks Ionic* e *Firebase*, foi implementado o Notifique-me, um aplicativo que permite criar salas de bate-papo em uma localização geográfica específica, tornando possível a qualquer usuário - estando próximo a sala criada - adentrá-la e se comunicar. Sendo o Notifique-me apenas uma forma de unir digitalmente indivíduos que estejam fisicamente próximos, acredita-se que vários nichos podem ser beneficiados por ele. Para este trabalho no entanto, foi priorizado apenas um: eventos corporativos.

O Notifique-me foi disponibilizado para *download* e passou por um teste em um evento real, e mesmo apresentando certa imprecisão, cumpriu com o que lhe foi proposto. Após ter sido implementada, a ideia proposta nesta monografia passou por uma etapa de validação, que consistiu em uma pesquisa de mercado. Por meio da pesquisa pôde-se concluir que o público entrevistado apresentou interesse para com a ideia proposta, mas que também possuem outros problemas que poderiam ter sido solucionados no aplicativo desenvolvido. Assim, além de apresentar a modelagem e construção do aplicativo aqui implementado, esta monografia documentou as dificuldades que uma validação tardia pode acarretar.

Palavras-chave: *Ionic*, *Firebase*, Geolocalização.

Abstract

Being important to man since ancient times, both the communication and the ways to obtain it have always been in constant progress. As a result of this advancement, currently instant messengers have the power to unite individuals from various parts of the globe, and to allow communication between them to be practically real-time. However, for this communication to be established, a bond must be made, and for that one of the communicating parties must possess some personal information of the individual who wishes to communicate, such as a cell phone number or e-mail. The same concept still applies to physically close people, because even though they are in the same place, two individuals cannot establish a bond if they are unknown to each other.

Thinking about this, in this work is proposed a different form of bond, where the only limiter to start a conversation environment is physical proximity. Using mainly the frameworks Ionic and firebase, has been implemented Notifique-me, an application that allows to create chat rooms in a specific geographic location, making possible to any user-being close The room created–enter it and communicate. Being the Notifique-me only one way to digitally unite individuals who are physically close, it is believed that several niches can benefit from it. For this work however, only one was prioritized: corporate events.

The Notifique-me was made available to download and passed a test in a real event, and even showing some inaccuracy, fulfilled what was proposed to him. After being implemented, the idea proposed in this monograph went through a validation step, which consisted of a market research. Through the research it was possible to conclude that the interviewed public presented interest to the proposed idea, but also have other problems that could have been solved in the application developed. Thus, in addition to presenting the modeling and construction of the application implemented here, this monograph documented the difficulties that a delayed validation can entail.

Keywords: Ionic, Firebase, Geolocation.

Conteúdo

1	Introdução e Justificativa	1
1.1	Objetivos	1
1.1.1	Objetivo Geral	1
1.1.2	Objetivos Específicos	1
1.2	Organização do Texto	1
2	Fundamentação Teórica	3
2.1	Aplicativos Web vs Aplicativos Nativos vs Aplicativos Híbridos	3
2.2	Sistemas de navegação por satélite	5
2.2.1	Triangulação	5
2.2.2	Coordenadas Geográficas	6
2.3	Bancos de Dados	9
2.3.1	Modelo Relacional	9
2.3.2	NoSQL	10
2.4	Protocolos de comunicação	13
2.4.1	HTTP	13
2.4.2	<i>WebSocket</i>	14
2.5	UML	15
3	Metodologia	17
3.1	<i>Firebase</i>	17
3.1.1	<i>Authentication</i>	18
3.1.2	<i>Realtime Database</i>	18
3.1.3	<i>Cloud Storage</i>	20

3.2	Angular	20
3.3	<i>Apache Cordova</i>	22
3.4	<i>Ionic</i>	23
4	Desenvolvimento	26
4.1	Visão Geral do Sistema	26
4.2	Levantamento de Requisitos	27
4.2.1	Requisitos não funcionais	27
4.2.2	Requisitos funcionais	27
4.3	Casos de Uso	29
4.3.1	Diagramas de Casos de Uso	29
4.3.2	Casos de Uso Expandidos	31
4.4	Diagrama de Classe	38
4.5	Modelagem dos Bancos de Dados	39
4.5.1	<i>Realtime Database</i>	39
4.5.2	<i>Cloud Storage</i>	46
4.6	Prototipação	48
5	Avaliação Final	57
5.1	Teste do aplicativo	57
5.2	Validação	59
5.2.1	Lean Canvas	59
5.2.2	Pesquisa de Aceitação	60
5.2.3	Análise da Pesquisa	66
6	Conclusão	67
6.1	Contribuições deste Trabalho	68
6.2	Dificuldades Encontradas	68
6.3	Trabalhos Futuros	68

Lista de Tabelas

4.1	Caso de Uso - Fazer Cadastro	31
4.2	Caso de Uso - Fazer Login	31
4.3	Caso de Uso - Fazer Logoff	32
4.4	Caso de Uso - Alterar Apelido	32
4.5	Caso de Uso - Alterar Foto de Perfil	32
4.6	Caso de Uso - Criar Sala de Bate-papo	33
4.7	Caso de Uso - Entrar em sala de bate-papo	33
4.8	Caso de Uso - Enviar convite de amizade	34
4.9	Caso de Uso - Excluir amigo	34
4.10	Caso de Uso - Aceitar convite de amizade	34
4.11	Caso de Uso - Recusar convite de amizade	35
4.12	Caso de Uso - Sair de sala de bate-papo	35
4.13	Caso de Uso - Enviar mensagem em uma sala de bate-papo	35
4.14	Caso de Uso - Enviar foto em uma sala de bate-papo	36
4.15	Caso de Uso - Enviar convite de amizade em uma sala de bate-papo.	36
4.16	Caso de Uso - Excluir a sala	36
4.17	Caso de Uso - Alterar o nome da sala	37
4.18	Caso de Uso - Alterar a descrição da sala	37
4.19	Caso de Uso - Alterar a foto da sala	37
4.20	Caso de Uso - Bloquear usuário da sala de bate-papo	38
4.21	Caso de Uso - Desbloquear usuário da sala de bate-papo	38

Listas de Figuras

2.1	Principais diferenças entre aplicativos <i>web</i> , nativos e híbridos. Retirado de [57].	4
2.2	Exemplo de utilização de 4 satélites para determinar uma posição geográfica. Retirado de [49].	6
2.3	Linhas imaginárias no Globo terrestre. Retirado de [49].	7
2.4	Exemplo de tabelas em um Banco de Dados Relacional. Retirado de [48].	10
2.5	Com duas interrupções nas linhas de comunicação, a rede divide-se em dois grupos. Retirado de [56].	11
2.6	Exemplo de armazenamento de dados em um Banco NoSQL - Chave/Valor.	12
2.7	Exemplo de uso do método GET, seguido de uma resposta a ele. Retirado de [46].	14
2.8	Representação do funcionamento do protocolo <i>WebSocket</i>	15
3.1	Exemplo de exceção ao tentar criar uma conta com um <i>e-mail</i> inválido.	18
3.2	Notificação visual disparada pelo <i>Realtime Database</i> após ocorrer uma adição no banco.	19
3.3	Arquitetura do Angular. Retirado de [2].	20
3.4	Exemplo de metadados utilizados pelo Angular, retirado de [18].	21
3.5	Exemplo de uso da diretiva <i>ngModel</i> , retirado de [18].	21
3.6	Arquitetura do <i>Apache Cordova</i> . Retirado de [22].	22
3.7	Interface do <i>Ionic Deploy</i>	23
3.8	Exemplo de uso do método responsável por trazer as informações sobre a localização. Retirado de [14].	24
3.9	Exemplo de criação de uma <i>geofence</i> . Retirado de [13].	25
4.1	Exemplo de funcionamento do aplicativo.	27

4.2	Caso de Uso de um usuário deslogado.	29
4.3	Caso de uso de um usuário logado.	30
4.4	Caso de uso de um usuário e de um administrador de uma sala.	30
4.5	Diagrama de classe do sistema.	39
4.6	Nível mais alto do banco <i>Realtime Database</i> .	40
4.7	Salas criadas na aplicação, salvas no banco <i>Realtime Database</i> .	40
4.8	Detalhes de uma sala, salva no banco <i>Realtime Database</i> .	41
4.9	Mensagens de uma sala, salvas no banco <i>Realtime Database</i> .	42
4.10	Exemplo de mensagens sem e com imagem, enviadas a uma sala, salvas no banco <i>Realtime Database</i> .	43
4.11	Usuários de uma sala, salvos no banco <i>Realtime Database</i> .	43
4.12	Exemplo de como são salvos usuários não-bloqueados e bloqueados no banco <i>Realtime Database</i> .	44
4.13	Lista de usuários da aplicação, salvos no banco <i>Realtime Database</i> .	45
4.14	Conteúdo de um usuário específico, salvo no banco <i>Realtime Database</i> .	45
4.15	Lista de amigos e convites de um usuário, salvos no banco <i>Realtime Database</i> .	46
4.16	Nível mais alto do banco <i>Cloud Storage</i> .	46
4.17	Lista de salas no banco <i>Cloud Storage</i> .	47
4.18	Conteúdo de uma sala específica, salva no banco <i>Cloud Storage</i> .	47
4.19	Lista de fotos enviadas a uma sala, salvas no banco <i>Cloud Storage</i> .	47
4.20	Fotos de perfil dos usuários da aplicação, salvas no banco <i>Cloud Storage</i> .	48
4.21	Tela de <i>login</i> da aplicação.	48
4.22	Tela de registro da aplicação.	49
4.23	Tela principal da aplicação, que exibe todas as salas existentes.	49
4.24	Código simplificado para capturar a posição com menor acurácia.	51
4.25	Salas de bate-papo dispostas em um mapa.	52
4.26	Tela de criação de sala.	52
4.27	Sala de bate-papo comum e sala de leitura.	53
4.28	Configurações de Sala exibida a um usuário comum.	54
4.29	Configurações de Sala exibida ao criador da sala.	55
4.30	Tela de convites pendentes e amigos.	56

4.31	Tela de configurações de conta.	56
5.1	Sala criada no evento <i>Startup Weekend Coxim</i> .	58
5.2	Contas criadas no dia 10/11/2018.	58
5.3	Contas criadas no dia 10/11/2018 - Visualizadas no Firebase <i>Authentication</i> e <i>Realtime Database</i> .	59
5.4	Modelo <i>Lean Canvas</i> da ideia proposta.	60
5.5	Frequência de participação em eventos.	61
5.6	Frequência de dúvidas sobre a programação de um evento.	62
5.7	Forma ideal para se receber informações durante um evento.	62
5.8	Aprovação da ideia proposta em escala.	63
5.9	Aprovação da ideia proposta.	63

Capítulo 1

Introdução e Justificativa

Ao ultrapassar a marca de 4 bilhões de usuários em 2018 [7], a rede mundial de computadores corrobora que o uso da tecnologia tem se tornado cada vez mais intrínseco na sociedade. Com tal progresso é natural que o dia-a-dia social também avance, ou no mínimo se transforme. Devido a popularização das redes sociais e mensageiros instantâneos por exemplo, até mesmo o simples e natural ato de se estabelecer um diálogo ganhou novos meios. Em números, são aproximadamente 3 bilhões de usuários ativos em redes sociais, e 128 a quantidade de países onde o aplicativo *WhatsApp* domina entre os mensageiros instantâneos [7]. Perante estes dados é notável a inclinação humana para o contexto social. O grande número de pessoas interagindo entre si e a vasta quantidade de tecnologias disponíveis para este fim mantém atual uma frase escrita por Aristóteles: “o Homem é um animal político, sendo de sua natureza a necessidade de interagir com outros de sua espécie” [31].

A popularidade de aplicativos sociais é natural, pois ao transportar o diálogo humano para o universo digital, criou-se um novo meio de comunicação, beneficiando assim diversas áreas. Na educação por exemplo, o aplicativo *WhatsApp* já foi utilizado em escolas a fim de tornar o estudo contínuo e de fácil acesso para os alunos [33]. Como resultado, além de o aprendizado ter se tornado mais eficaz foi criada uma atmosfera amigável na relação aluno-aluno e aluno-professor, e mais bem desenvolvida a sensação de “pertencer ao grupo”.

Nota-se que um dos objetivos primários dos aplicativos de comunicação é permitir a interação de dois ou mais indivíduos que se conheçam, direta ou indiretamente (podendo ou não estar distantes fisicamente) [37, 39]. Todavia, se esses indivíduos são desconhecidos entre si, mesmo estando próximos fisicamente, a maioria destes aplicativos não tem utilidade, pois sem o conhecimento de alguma informação pessoal (como um número de telefone celular ou e-mail), o vínculo entre as duas partes não pode ser estabelecido. Além disso, ainda que os indivíduos que desejam se comunicar estejam próximos e se conheçam, o processo tradicional para que esta comunicação se torne possível, dependendo do número de integrantes é lento, tendo em vista o simples objetivo final.

Em uma sala de aula por exemplo, se um professor desejar passar uma informação para seus alunos utilizando o aplicativo *WhatsApp*, ele deverá criar um grupo e adicionar nele todos os alunos que deverão receber a informação. Se no exemplo dado a sala contiver

poucos alunos, o processo é eficaz e relativamente rápido. No entanto, se na sala de aula houver uma grande quantidade de alunos, o tempo que será levado para o término do processo pode torná-lo inviável. Para este problema, o *WhatsApp* oferece como solução *links* compartilháveis e *QR codes*. Dessa forma, para entrar em um grupo de conversa (mesmo não conhecendo nenhum dos integrantes) basta possuir um *link* de entrada ou realizar a leitura de um código *QR* [28]. Todavia percebe-se nesta solução que para se estabelecer um vínculo ainda é necessário uma informação diretamente relacionada ao grupo.

O *Tinder* e alguns outros aplicativos de mesmo nicho resolvem este problema utilizando outra abordagem. Ao se aproveitarem do conceito de geolocalização e de proximidade física, tornam desnecessário a utilização de alguma informação pessoal para se estabelecer um vínculo. Com isso, pessoas fisicamente próximas - ainda que desconhecidas - podem interagir e se comunicar. Porém mesmo não sendo a única [58], a principal finalidade desta gama de aplicativos é realizar encontros, sendo assim específico para determinado grupo de usuários e exclusivo para apenas duas pessoas.

O conceito de comunicação por localidade - utilizado em aplicativos como o *Tinder* - já foi empregado em outras áreas. Tantothai por exemplo propôs um aplicativo que permite que indivíduos próximos possam se comunicar, a fim de facilitar o processo de busca por estabelecimentos como restaurantes e hospedarias [61]. Assim, entende-se que mais áreas poderiam se aproveitar do mesmo conceito. Locais como: lojas, restaurantes, hotéis, escolas e faculdades são alguns dos exemplos que normalmente possuem diversas pessoas desconhecidas entre si que poderiam se beneficiar de uma “comunicação local”. Restaurantes por exemplo, poderiam utilizar este conceito para prover uma comunicação rápida entre atendimento e clientela, podendo compartilhar o cardápio do dia e receber os pedidos de forma centralizada. Lojas e mercados poderiam informar aos seus compradores as promoções do dia e compartilhar cupons de desconto. Hotéis poderiam fornecer um ambiente rápido para solucionar as dúvidas de seus hóspedes, e assim por diante.

Por isso, e a fim de beneficiar nichos pouco explorados, é proposto nesta monografia um aplicativo social que une a interação e comunicação de diversos indivíduos - presente em aplicativos como o *WhatsApp*, e a proximidade física - funcionalidade vista em aplicativos como o *Tinder*. Como dito, acredita-se que este conceito pode ser utilizado em diversas áreas, mas para que se pudesse trabalhar com um público-alvo menos abrangente, neste documento direcionou-se o aplicativo especificamente para eventos corporativos (e.g., congressos, simpósios, *workshops*, oficinas). A escolha realizada, dá-se ao fato de que uma vez sendo acadêmico, este é o público o qual possui mais afinidade e proximidade, e por isso houve um interesse pessoal em priorizar a comunidade a qual faço parte.

1.1 Objetivos

1.1.1 Objetivo Geral

Neste trabalho tem-se como objetivo a implementação de um aplicativo *mobile* no contexto social e comunicativo. O aplicativo utilizará o conceito de geolocalização para permitir que indivíduos fisicamente próximos, mesmo que desconhecidos entre si, possam interagir e se comunicar. Dessa forma, a proximidade física será o único limitador para se estabelecer uma comunicação com um ou mais indivíduos.

1.1.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

- Realizar um levantamento de requisitos para modelar a aplicação que será desenvolvida.
- Estudar a *framework Ionic*, seu uso e como é realizado sua interação com os recursos nativos.
- Estudar uma forma viável para implementar um *chat* bidirecional multiusuário.
- Implementar uma lógica que utilize a localização atual de dispositivos, a fim de validar sua proximidade.
- Testar o aplicativo criado com usuários reais.
- Realizar uma pesquisa de mercado para obter uma avaliação final do aplicativo, a fim de recolher o *feedback* dos usuários sobre a ideia proposta.

1.2 Organização do Texto

Os capítulos subsequentes desta proposta estão organizados da seguinte forma:

No Capítulo 2 apresenta-se os conceitos importantes para o desenvolvimento da aplicação. Nele é explicado a diferença de aplicativos *web*, nativos e híbridos e por que o modelo híbrido foi utilizado nesta proposta. São também explicados alguns conceitos de geolocalização e como pode-se calcular a distância entre dois pontos geográficos. É abordado o conceito de bancos de dados relacionais e NoSQL e explicado o funcionamento dos protocolos HTTP e *WebSocket*. Por fim, há um breve resumo sobre a UML.

As tecnologias utilizadas na implementação da aplicação estão descritas no Capítulo 3, onde é apresentada a *Framework Firebase* bem como os módulos que foram utilizados. Ademais, neste capítulo é explanado o *Ionic* e sua interação com a *framework Angular* e a estrutura *Apache Cordova*.

O processo de documentação, desde o levantamento de requisitos até o produto final, é descrito no Capítulo 4. No Capítulo 5 é apresentado um teste feito com o aplicativo final e explanado como foi realizado o processo de validação.

Finalmente, no Capítulo 6 conclui-se este trabalho, apresentando as contribuições realizadas, as dificuldades encontradas e sendo sugerido um trabalho futuro.

Capítulo 2

Fundamentação Teórica

Neste capítulo são explanados os principais conceitos que serão utilizados no processo de criação e implementação da aplicação proposta. Na Seção 2.1 são abordadas as principais diferenças entre aplicativos *web*, nativos e híbridos e por que o modelo híbrido foi escolhido nesta proposta. Na Seção 2.2 é apresentado um breve histórico de tecnologias de navegação, abordado alguns conceitos de geolocalização, e explicado como se pode calcular a distância entre dois pontos geográficos. Na Seção 2.3 é pontualmente explicado algumas diferenças de bancos relacionais e bancos NoSQL. Na Seção 2.4 é resumido o conceito de protocolos de comunicação e apresentado os protocolos utilizados indiretamente na aplicação. Finalmente, na Seção 2.5 é explicado a UML e apresentado os conceitos utilizados para a etapa de análise de *software* deste trabalho.

2.1 Aplicativos Web vs Aplicativos Nativos vs Aplicativos Híbridos

Ao se desenvolver um aplicativo *mobile* é possível escolher entre 3 opções principais qual será o modelo de implementação [36]. O uso de cada um destes modelos tem suas vantagens e desvantagens, e por isso não é regra utilizar apenas um modelo para todos os casos. Os três modos principais para desenvolvimento *mobile* são:

Aplicativos Web são páginas na internet que podem ser acessados como qualquer outro site através de um navegador e são desenvolvidos geralmente em HTML5 (*Hypertext Markup Language*). Por utilizarem as principais tecnologias *web*, estes aplicativos tem se tornado cada vez mais poderosos e bonitos visualmente. Chen demonstra como exemplo que o aplicativo *web* da plataforma *Youtube* é muito parecida com seu aplicativo nativo [36].

Aplicativo Nativo “consiste essencialmente em um aplicativo desenvolvido para um dispositivo móvel específico e instalado diretamente sobre o próprio dispositivo com interpretação direta pelo sistema operacional” [57, p. 2]. Por serem implementados para um SO (Sistema Operacional) específico, aplicativos nativos são os que mais

bem utilizam os recursos disponíveis (como câmera e *bluetooth*), além disso são os mais velozes desta lista e são facilmente distribuídos nas lojas de aplicativos *mobile* [36].

Aplicativos Híbridos estão no meio termo entre aplicativos *web* e aplicativos nativos. Assim como os aplicativos *web*, os aplicativos híbridos são escritos utilizando tecnologias *web*, como HTML5 e *JavaScript*, porém são executados em *webviews*, navegadores invisíveis aos olhos dos usuários que tem apenas o básico para renderizarem uma página *web* [57], e por conta disto são quase indistinguíveis de aplicativos nativos [36]. Assim, aplicativos híbridos são essencialmente aplicativos nativos, podendo ser obtidos diretamente de lojas *mobile* oficiais e podendo fazer acesso aos mesmos recursos que aplicativos nativos fazem [50].

A Figura 2.1 exibe as principais diferenças entre esses 3 tipos de implementação.

	Principais Vantagens	Principais Desvantagens
WebApps	- Executados pelo <i>browser</i> , proporcionando o uso em outras plataformas; Atualização e distribuição rápida e abrangente, não precisam ser baixados ou atualizados; Acesso rápido e fácil, os usuários têm acesso imediato pelo <i>Smartphone</i> .	- Pouca ou quase nenhuma integração com o <i>hardware</i> do dispositivo em que está sendo executado; Mais lentos, dependendo da conexão com a <i>Internet</i> ; Interação entre o usuário e o aplicativo menos rica em funcionalidades.
Nativo	Interação entre o usuário e o aplicativo mais rica em funcionalidades e recursos; Velocidade na execução. Independente da <i>Internet</i> .	Uma nova aplicação escrita para cada plataforma diferente Distribuição e atualização dependentes de lojas on-line; (<i>Apple Store</i> , <i>Google play</i>).
Híbrido	Compartilhamento de boa parte do código entre plataformas; Possibilidade do uso de recursos da plataforma com código nativo; Pode ser distribuído lojas on-line; (<i>Apple Store</i> , <i>Google play</i>) Interoperabilidade.	Performance. Limitação de design.

Figura 2.1: Principais diferenças entre aplicativos *web*, nativos e híbridos. Retirado de [57].

Assim pode-se concluir que dependendo do objetivo a ser alcançado uma das formas de implementação será mais propícia e benéfica que as demais. Neste trabalho, como grande parte da aplicação aqui desenvolvida necessita de acesso constante a recursos nativos como o GPS (*Global Positioning System*), a opção de implementação como aplicativo *web* foi descartada. Restando então apenas duas opções, foi escolhido o estilo híbrido pois mesmo não apresentando o mesmo desempenho que aplicativos nativos [55], o uso de linguagens *web* certamente facilitará e agilizará o processo de desenvolvimento.

2.2 Sistemas de navegação por satélite

Desde a antiguidade a humanidade percebeu que localizar-se geograficamente é um importante passo a ser conquistado. A navegação por exemplo, tinha muito a ganhar com este avanço, e por meio de sinais de fumaça e estrelas as antigas sociedades alcançaram tal objetivo [49]. Desde então viu-se importante utilidade no uso da localização, e por isso as formas para consegui-la começaram a se desenvolver.

Como resultado desse desenvolvimento foi criado pela defesa dos Estados Unidos da América em 1972 o *Global Positioning System* ou GPS e com ele o termo Sistema de Navegação por Satélite [34]. Desde então várias tecnologias foram criadas baseadas no GPS, o GLONASS e o GALILEO são os dois que juntamente com o próprio GPS mais se popularizaram. O GPS europeu GALILEO por exemplo, apenas 1 ano após entrar em operação, atingiu aproximadamente 100 milhões de usuários [12], o que prova que mesmo sendo uma tecnologia antiga - criada na década de 70 [30] - ainda é muito necessária.

2.2.1 Triangulação

Os satélites de GPS transmitem de forma contínua informações que dispositivos receptores podem analisar, como dados sobre os satélites que estão em órbita, a própria órbita e/ou trajeto do satélite emissor, o tempo de transmissão, entre outros [49]. Com tais dados, o receptor pode calcular o tempo necessário para receber mensagens de diferentes satélites, e assim, determinar sua própria posição geográfica. Este processo é comumente chamado de triangulação.

Holdener explica que, para que o processo de triangulação seja iniciado, deve-se saber primariamente a localização dos satélites que serão utilizados e que, dentre as informações que um satélite emissor envia a um receptor, as 3 principais são:

- *Timestamp* - Indicando quando a mensagem foi enviada.
- Efemérides - Responsável por prover informações orbitais e correções do relógio para o satélite específico. As informações contidas no Efeméride são muito precisas e são válidas por até cerca de 30 minutos.
- Almanaque - Fornece assim como o Efemérides informações orbitais e de correção de relógio, porém para uma matriz de satélites. As informações contidas no Almanaque não são muito precisas e podem ser válidas por até quatro meses.

Com a utilização de três satélites (triangulação) é possível calcular uma posição geográfica aproximada, mas tendo em vista que os satélites emissores estão em órbita e que pode-se levar alguns segundos para que uma informação chegue de um satélite a um receptor, o fator tempo deve ser considerado. Portanto, para fugir de erros de imprecisão geográfica, um quarto satélite deve ser utilizado [49].

São exemplificados em 4 passos por Holdener como funciona o cálculo de uma posição utilizando 4 satélites:

1. Primeiramente, mensura-se a distância do receptor até um satélite A, criando uma esfera centralizada no próprio satélite. O ponto de localização do receptor estará em algum dos pontos na superfície da esfera.
2. Mensura-se a distância do receptor até um satélite B, criando assim, uma segunda esfera. O ponto de localização do receptor estará em algum dos pontos na área de intersecção criado pelas duas esferas.
3. Mensura-se a distância do receptor até um satélite C, criando uma terceira esfera. O ponto de localização do receptor está em um dos dois únicos pontos criados pela intersecção das três esferas.
4. Mensura-se a distância até um satélite D, criando uma quarta esfera. O ponto é determinado pela intersecção desta esfera com um dos dois pontos descritos no passo anterior.

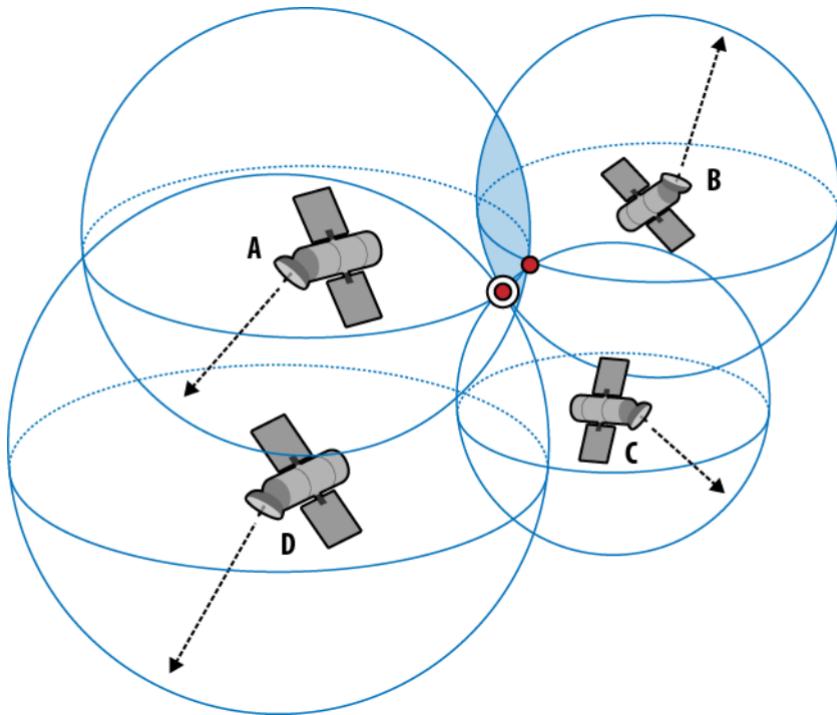


Figura 2.2: Exemplo de utilização de 4 satélites para determinar uma posição geográfica. Retirado de [49].

2.2.2 Coordenadas Geográficas

Para que seja possível representar a posição de um ponto em uma superfície qualquer, como um plano, uma esfera ou até um elipsoide, é necessário a utilização de algum sistema de coordenadas [41]. Em planos por exemplo, é usualmente aplicado as coordenadas cartesianas, dadas por X e Y . Na geolocalização, para que se possa expressar qualquer posição

na superfície do planeta, é utilizado o sistema de coordenadas geográficas, expressadas em latitude e longitude [49].

Latitude e Longitude

Demonstrado na Figura 2.3, as coordenadas geográficas se baseiam em linhas imaginárias traçadas sobre o globo terrestre. A Linha do Equador divide o globo em duas metades: Hemisfério Norte e Sul, e está situada a uma igual distância dos polos. As linhas paralelas a Linha do Equador, determinam a latitude do sistema de coordenadas geográficas [41]. Desta forma, Latitude é a distância de qualquer ponto da superfície terrestre até a Linha do Equador, iniciando em 0° na própria Linha do Equador, e atingindo até 90° para o Norte ou para o Sul [41].

De forma semelhante a Linha do Equador, o Meridiano de Greenwich divide o globo em outras duas metades: os Hemisférios Leste e Oeste. As linhas paralelas a Linha de Greenwich determinam a longitude do sistema de coordenadas geográficas. Longitude é portanto a distância de qualquer ponto da superfície terrestre até a Linha de Greenwich, a contagem se inicia em 0° no próprio Meridiano, e varia até 180° para Leste e 180° para Oeste [41].

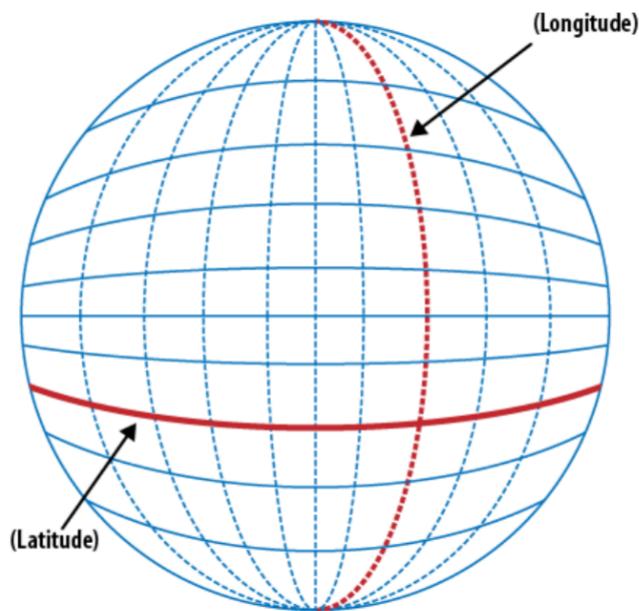


Figura 2.3: Linhas imaginárias no Globo terrestre. Retirado de [49].

Notação

Tendo em vista que 1 grau é equivalente a 60 minutos, e 1 minutos é equivalente a 60 segundos, para se ter uma melhor precisão ao localizar um ponto no globo, os graus de latitude e longitude podem ser divididos em graus($^\circ$), minutos($'$) e segundos($''$) [49]. As

coordenadas de localização do Cristo Redentor por exemplo são $22^{\circ} 57' 07''$ S, $43^{\circ} 12' 37''$ W, que pode ser lido como 22 graus, 57 minutos e 7 segundos ao Sul do Equador e 43 graus, 12 minutos e 37 segundos a Oeste de *Greenwich*. Esta notação é chamada de Graus, Minutos e Segundos ou GMS.

A mesma posição geográfica pode também ser escrita como -22.951944, -43.210278. Esta notação é chamada de Graus decimais, e nela não é necessário utilizar as iniciais de *North*, *South*, *East* e *West* para se indicar a direção do ponto no globo, ao invés disso apenas utilizam-se números decimais positivos e negativos. O decimal só será positivo para a latitude e longitude quando a posição estiver a Norte do Equador, e a Leste de *Greenwich*, respectivamente. Para se converter um GMS para Graus Decimais, segue-se os seguintes passos:

Conversão da coordenada $22^{\circ} 57' 07''$ S (latitude) para Graus Decimais:

1. Obter o total de segundos da coordenada (desconsiderando o grau);

$$\begin{aligned} 57 \text{ (min da coordenada)} * 60 \text{ (total de segundos em um minuto)} &= 3420 \text{ segundos} \\ 3420 + 7 \text{ (segundos da coordenada)} &= 3427 \text{ segundos} \end{aligned}$$

2. Dividir o total de segundos por 3.600 (número de segundos em um grau);

$$\frac{3427}{3600} = 0.951944444$$

3. Adicionar o resultado da divisão no número total de graus;

$$22 + 0.951944444 = 22.951944444$$

4. Se a coordenada é Sul como latitude, ou Leste como longitude, nega-se o resultado final.

$$-22.951944444$$

Distância entre dois pontos

Apesar de ser elipsoidal, por ser demasiadamente grande a Terra pode ser considerada plana para cálculos entre dois pontos que são muito próximos [8]. Porém os erros começam a se mostrar perceptíveis quando a distância entre os pontos é de mais de 20 quilômetros e por isso a geometria da Terra é normalmente levada em consideração. Dito isto, a fim de se obter uma alta precisão, utilizou-se¹ neste trabalho a Fórmula de *Haversine*.

A Fórmula de *Haversine* é uma equação importante na navegação, uma vez que considera o raio da Terra e calcula a distância entre dois pontos a partir de suas longitudes

¹A utilização da *API Distance Matrix* do *Google Maps* foi considerada e até mesmo utilizada inicialmente, mas deixada de lado uma vez que o cálculo realizado é baseado em rotas, e para a implementação deste trabalho fez-se necessário a distância em linha reta entre os dois pontos.

e latitudes [16]. A fórmula pode ser observada na Equação 2.1, onde ϕ é a latitude, λ é a longitude, R é o raio da Terra (aproximadamente 6,371km) e d é a distância entre os dois pontos.

$$a = \sin^2\left(\frac{\Delta\phi}{2}\right) + \cos\phi_1 \cdot \cos\phi_2 \cdot \sin^2\left(\frac{\Delta\lambda}{2}\right) \quad (2.1)$$

$$c = 2 \cdot \text{atan}2(\sqrt{a}, \sqrt{(1-a)})$$

$$d = R \cdot c$$

2.3 Bancos de Dados

Uma vez visto sua utilidade e vasta aplicação, os bancos de dados tornaram-se essenciais na sociedade moderna [43]. Sua utilização tornou-se tão intrínseca na vida cotidiana, que sua presença pode até ser imperceptível a usuários comuns. Atividade simples como realizar um *post* em uma rede social, fazer uma transferência bancária, redigir um texto em um editor e ver um vídeo em um *player* são alguns dos muitos exemplos de processos que utilizam assiduamente um Banco de Dados. Banco de Dados é, portanto, uma coleção de dados logicamente relacionados que tenham algum significado, ainda que implícito [59].

2.3.1 Modelo Relacional

Após alguns anos de vida, o mundo computacional teve diversos avanços: de linguagens, de arquiteturas, de plataforma, e etc. A utilização de bancos de dados relacionais porém, é algo que por suas características aparenta ser constante [56]. Sadalage e Fowler brincam ao dizer que arquitetos iniciando um novo projeto não discutem sobre qual banco de dados será empregado, e sim qual dos bancos relacionais irão utilizar [56]. Tal estabilidade no mercado tem fundamento, os próprios autores argumentam que bancos de dados relacionais são uma tecnologia bem-sucedida há mais de duas décadas, fornecendo persistência, controle de concorrência e um mecanismo de integração.

“A estrutura fundamental do modelo relacional é a relação (tabela). Uma relação é constituída por um ou mais atributos (campos) que traduzem os tipos de dados a armazenar. Cada instância do esquema (linha) é chamada de tupla (registro)” [59, p. 8]. Este modelo tem como base a teoria dos conjuntos e álgebra relacional e foi resultado de um estudo teórico realizado por Codd [38]. Na Figura 2.4 é mostrado um exemplo de tabelas em um banco de dados relacional.

TipoDeProduto	
CodTipoProd	DescrTipoProd
1	Computador
2	Impressora

Produto			
CodProd	DescrProd	PrecoProd	CodTipoProd
1	PC desktop modelo X	2.500	1
2	PC notebook ABC	3.500	1
3	Impressora jato de tinta	600	2
4	Impressora laser	800	2

Figura 2.4: Exemplo de tabelas em um Banco de Dados Relacional. Retirado de [48].

Quando o volume de dados cresce demasiadamente porém, o modelo relacional tende a mostrar suas limitações. Com a expansão natural da internet, este é um problema cada vez mais comum a ser considerado, pois as aplicações tendem a processar e armazenar cada vez mais dados. O *Facebook* por exemplo, no início de 2018 contava com cerca de 2,17 bilhões de usuários, um total de 15% a mais que no ano anterior [7]. Assim, apesar de suas qualidades, “as bases de dados relacionais não foram feitas para lidar com um enorme volume de informação num curto espaço de tempo, nem preveem o aumento das suas capacidades de processamento de um dia para o outro, sem ser necessário uma reconfiguração” [35, p. 5]. Para aumentar o desempenho de uma base de dados relacional, existem duas possíveis opções [38]:

1. Escalar verticalmente - Substituindo o *hardware* atual por um mais robusto;
2. Escalar horizontalmente - Distribuindo a base de dados por múltiplas instâncias.

Todavia, dependendo da quantidade de dados com que se deseja lidar, ambas as soluções mostram ser pouco viáveis. Cardoso explana que atualmente, para processar informações na ordem dos Petabytes, é necessário um *hardware* muito potente, que por consequência será também muito dispendioso [35], e sendo assim, a primeira opção já não se mostra tão factível. A segunda opção também apresenta problemas, pois a medida que a base de dados é distribuída, a complexidade dos *joins* com tabelas que não estejam na mesma máquina aumenta. Além disso, conforme se aumenta o número de instâncias da base de dados, torna-se cada vez mais difícil garantir as propriedades ACID (*Atomicity, Consistency, Isolation, Durability*) [35].

2.3.2 NoSQL

Escalar horizontalmente, distribuindo o processamento em diversas máquinas (*clusters*) é uma boa solução, porém bancos de dados relacionais não foram projetados para serem executados dessa forma [56]. A solução óbvia foi tomada, e criou-se então um conceito

(ou movimento [56]) de bases de dados que fossem fáceis de distribuir e escalar, e a esta definição dá-se o nome de Bancos NoSQL [35].

Por ser idealizado para executar de forma elástica, Bancos NoSQL não garantem as propriedades ACID, ao invés disso, utilizam o modelo de consistência eventual BASE (*Basically Available, Eventual, Consistency*) - normalmente associado a sistemas distribuídos [35] - para o controle de consistência, o que consequentemente traz uma sensível diminuição no custo computacional para a garantia de consistência dos dados em relação a SGBD tradicionais [64]. Neste modelo, se estipula que se durante determinado tempo não existirem atualizações, todas as atualizações pendentes serão propagadas por todos os nós do sistema, tornando-o assim, eventualmente consistente.

O paradigma BASE advém do teorema CAP, o qual afirma que existem três propriedades úteis em SGBD:

C - Consistency (Consistência) Tem como objetivo permitir que transações distribuídas em vários nós sejam realizadas como “tudo ou nada”, e que as réplicas (quando existirem) estejam sempre em um estado consistente.

A - Availability (Disponibilidade) Tem como objetivo manter o sistema sempre disponível. Em caso de falha, o sistema deve continuar funcionando com alguma réplica dos recursos indisponíveis.

P - Partition tolerance (Tolerância a partições) Tem como objetivo manter o sistema operando mesmo no caso de falhas de rede. A este fim é dividido o processamento dos nós em grupos que não se comunicam (os subgrupos continuam o processamento independentemente). Esta situação é conhecida como *split brain* ou divisão cerebral e pode-ser observada na Figura 2.5.

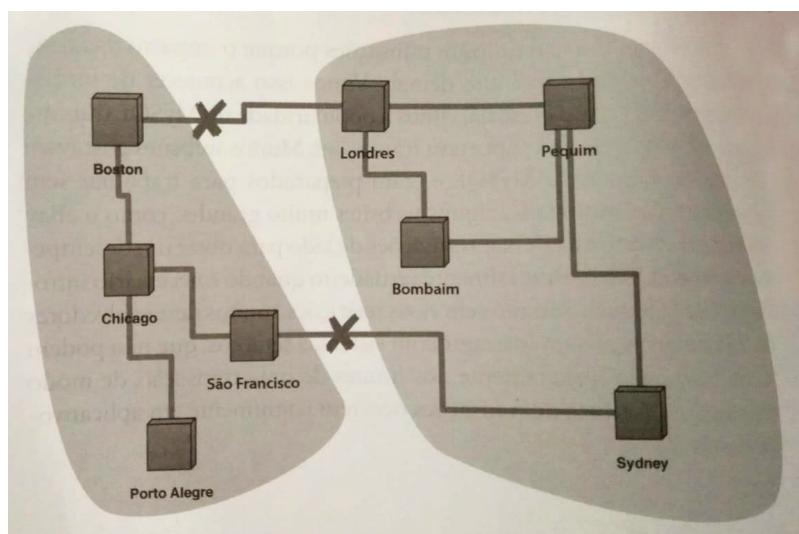


Figura 2.5: Com duas interrupções nas linhas de comunicação, a rede divide-se em dois grupos. Retirado de [56].

No entanto, uma das propriedades (C, A ou P) deve ser desprezada, pois não é possível garantir todas as três simultaneamente [64, 56]. Sadalage e Fowler citam como exemplo um sistema com um único servidor que possui Consistência e Disponibilidade (C e A) mas não Tolerância a partições (P), pois como só há um servidor, se este estiver funcionando, o sistema estará disponível, não sendo necessário então a preocupação com a Tolerância a partições [56]. Em sistemas distribuídos, a proposta então é de enfraquecer o requisito de Consistência (C), e dar prioridade a Disponibilidade (A) e tolerância a partições (P), assim nascendo o conceito BASE [62].

Segundo Vieira, bases de dados NoSQL possuem várias características em comum, mas se diferenciam quanto ao modelo de dados utilizados [64]. Os modelos principais são baseados em: Chave-Valor, Grafos, Coluna e Documentos, sendo o primeiro destes o mais importante, pois dele originam-se alguns outros [64].

O modelo Chave-Valor consiste em conjuntos de pares chave/valor, isto é, para cada chave única e exclusiva, têm-se um valor atribuído a ela. Bancos chave/valor são altamente particionáveis e permitem escalabilidade horizontal que outros tipos de bancos de dados não conseguem alcançar [3]. Cardoso cita que a maior vantagem deste modelo é a sua simplicidade de utilização e implementação, tendo em vista que é apenas uma coleção de chaves com um único valor [35]. Além disso, programadores demoram menos tempo a aprendê-lo pois já estão habituados aos conceitos de Dicionários e *Hashes*. É possível observar na Figura 2.6 como são armazenados alguns dados em uma base de dados chave/valor.



Figura 2.6: Exemplo de armazenamento de dados em um Banco NoSQL - Chave/Valor.

2.4 Protocolos de comunicação

De forma semelhante ao diálogo humano, para que duas ou mais máquinas possam se comunicar, elas também devem conhecer e “conversar no mesmo idioma”. E assim como são vários os idiomas humanos, são também numerosos os idiomas das máquinas. A estes idiomas que permitem a conversa entre computadores dá-se o nome de Protocolos.

Segundo Tanenbaum, protocolo é um acordo entre as partes que se comunicam, que estabelece como se dará a comunicação [60]. Os protocolos são portanto um aglomerado de regras e padrões e são eles os responsáveis por padronizar e regrer de que forma diferentes computadores se comunicarão.

2.4.1 HTTP

Responsável por especificar as mensagens que os clientes podem enviar aos servidores e as respostas que eles receberão, o HTTP (*HyperText Transfer Protocol*) é um protocolo de transferência utilizado em toda a *World Wide Web* [60]. Embora tenha sido projetado para a utilização na *web*, o HTTP foi criado de modo mais genérico que o necessário, visando futuras aplicações orientadas a objetos. Por isso, além da solicitação de páginas *web*, são também aceitas operações chamadas métodos [60].

Segundo Tanenbaum, cada solicitação realizada consiste em uma ou mais linhas de texto ASCII², sendo a primeira palavra da primeira linha o nome do método solicitado. Os métodos mais comuns são:

GET - Solicita um recurso ao servidor.

POST - Envia dados ao servidor.

PUT - Requisita que um recurso seja armazenado na URI³ fornecida. Se a URI se refere a um recurso que já existe, ele será modificado, se não, o servidor tem permissão para criar o recurso com a URI especificada.

DELETE - Exclui um recurso especificado.

Após receber uma requisição, o servidor a processa e retorna uma mensagem de resposta, composta por uma linha de *status*, algumas linhas de cabeçalho e um corpo [46]. Na linha de *status* está contida a versão do protocolo HTTP, o código de processamento da requisição e uma mensagem textual. Se processada com sucesso, é estabelecida então uma conexão TCP⁴, normalmente na porta 80, entre o cliente e o servidor. Um exemplo de requisição e resposta podem ser observados na Figura 2.7.

²*American Standard Code for Information Interchange* - É um padrão amplamente usado para codificar documentos de texto em computadores [27].

³*Uniform Resource Identifier* - É o identificador de um recurso, como uma imagem ou uma página. Exemplo de URI: <https://exemplo.com/imagem.png>.

⁴*Transmission Control Protocol* - É um protocolo que provê um fluxo fim-a-fim confiável de dados [46, 60].

```
C: GET /newsgen/index.html HTTP/1.1
    Host: www.rnp.br
    Connection: close
    User-Agent: Mozilla/4.0
    Accept-Language: sgn-BR
    <CR><LF>

S: HTTP/1.1 200 OK
    Connection: close
    Date: Sun, 01 Aug 2004 15:00:00 GMT
    Server: Apache/1.3.0 (Unix)
    Last-Modified: Mon, 22 Jun 2004 10:04:17 GMT
    Content-Length: 1500
    Content-Type: text/html
    <CR>LF>
    ...

```

Figura 2.7: Exemplo de uso do método GET, seguido de uma resposta a ele. Retirado de [46].

2.4.2 *WebSocket*

No passado, quando se fazia necessário uma comunicação bidirecional entre um cliente e um servidor, exigia-se um abuso no uso do protocolo HTTP para pesquisar atualizações no servidor [29], entretanto, este método resulta em alguns problemas:

- O servidor é forçado a usar várias conexões TCP subjacentes diferentes para cada cliente: uma para enviar informações para o cliente e uma nova para cada mensagem recebida.
- O protocolo de rede tem uma alta sobrecarga, pois cada mensagem cliente-servidor possui um cabeçalho HTTP.
- O cliente é forçado a manter um mapeamento das conexões de saída para a conexão de entrada para que seja possível rastrear respostas.

Para contornar estes problemas, o protocolo *WebSocket* utiliza uma única conexão TCP para o tráfego bidirecional [29]. Dessa forma, cada lado da conexão pode enviar dados a qualquer momento, diferentemente do HTTP, “onde o servidor não consegue enviar dados para o cliente no momento em que desejar, precisando aguardar que o cliente faça uma requisição para que só então ele consiga transmitir qualquer informação” [63, p. 4]. Assim, o ciclo de vida do protocolo *WebSocket* consiste basicamente em:

1. Um *handshake*⁵ inicial;
2. A transferência de dados propriamente dita, sendo esta bidirecional: tanto o cliente quanto o servidor podem enviar informações sem ser necessário uma requisição para isso;
3. O pedido de uma das partes para que se encerre a conexão.

É mostrado na Figura 2.8 uma representação do ciclo de vida do protocolo *WebSocket*.



Figura 2.8: Representação do funcionamento do protocolo *WebSocket*.

2.5 UML

De acordo com Cortez e Gonçalves, assim como na engenharia é necessário projetar uma casa antes de construí-la, na computação é necessário projetar um sistema antes de desenvolvê-lo [44]. Esse projeto é obtido por meio de uma modelagem, i.e., representações gráficas que tem o intuito de facilitar o processo de desenvolvimento, uma vez que cada representação expressa um conjunto de características que a aplicação a ser desenvolvida deve possuir.

Por ser a fusão de várias linguagens de modelagem, a *Unified Modeling Language* (Linguagem de Modelagem Unificada) é uma tentativa de padronizar os artefatos de análise e projetos, como diagramas, modelos semânticos e sintaxes de notação [44]. Os diagramas da UML são classificados em três categorias:

Diagramas Estruturais - Modelam o aspecto estrutural do sistema. Engloba os diagramas de: Classes, Objetos, Componentes, Estrutura Composta, Pacotes e Implementação.

⁵Processo inicial em que as máquinas comunicantes se reconhecem antes de realizar a comunicação propriamente dita [60].

Diagramas Comportamentais - Modelam o aspecto dinâmico do sistema, sendo usados para representar as partes que sofrem alterações - como a interação entre usuários e o sistema. Engloba os diagramas de: Casos de Uso, Atividade e Máquina de Estados.

Diagramas de Interação - Também modelam o aspecto dinâmico do sistema, estes porém são utilizados para representar a comunicação entre entidades e a sequência de operações que serão invocadas. Engloba os diagramas de: Sequência, Comunicação, Tempo e Interatividade.

Os conceitos e diagramas da UML utilizados para o desenvolvimento deste trabalho foram:

Levantamento de Requisitos - É a etapa de compreensão do problema aplicada ao desenvolvimento de *software* [32], e tem como objetivo estabelecer quais as funcionalidades um sistema deve ter [44]. As principais seções de um documento de requisito são:

Requisitos funcionais - Definem as funcionalidades do sistema;

Requisitos não-funcionais - Declararam as características de qualidade que o sistema deve possuir, e que estão diretamente relacionadas às suas funcionalidades.

Casos de Uso - É um modelo de análise que representa um refinamento dos requisitos funcionais de um sistema em desenvolvimento. “O modelo de casos de uso é uma representação das funcionalidades externamente observáveis do sistema e dos elementos externo ao sistema que interagem com ele” [32, p. 54]. Ou seja, este modelo representa os possíveis usos de um sistema, na visão de um observador externo. Cada caso de uso representa um relato de uso de determinada funcionalidade do sistema em questão, sem ligação direta com a estrutura e comportamento direto desse sistema [32].

Diagrama de Classe - Segundo Cortés e Gonçalves, o Diagrama de Classe é possivelmente o diagrama mais utilizado na UML [44]. Este é um diagrama estrutural, que tem como objetivo mostrar a composição interna das entidades envolvidas. Neste modelo, é descrito o problema representado pelo sistema a ser desenvolvido sem levar em consideração as características da solução que será utilizada. Ou seja, apenas descreve-se a solução em um alto nível de abstração [32].

Capítulo 3

Metodologia

Neste capítulo são introduzidas as tecnologias que foram utilizadas na implementação do Notifique-me. Na Seção 3.1 é resumido o que é a *framework Firebase* e explicado os módulos que foram utilizados durante o desenvolvimento. Nas Seções 3.2 e 3.3 são explanadas a *framework Angular* e a estrutura *Apache Cordova*, respectivamente. Por fim, na Seção 3.4 é resumido e apresentado as funcionalidades usadas da *framework Ionic*.

3.1 *Firebase*

Firebase é uma plataforma BaaS (*Backend as a Service*) mantida pela *Google*, a qual visa abstrair a infraestrutura do lado do servidor, permitindo consequentemente ao desenvolvedor um ganho na velocidade de codificação [51], uma vez que não precisará mais dar enfoque a programação comumente associada ao *backend* (e.g., serviço REST¹). Sganzerla e Lummertz [51] citam algumas vantagens em utilizar o *Firebase* como serviço *BaaS*:

- A isenção de responsabilidade, tendo em vista que a segurança dos dados é incumbência da *Google*;
- Alta disponibilidade, uma vez que os servidores da *Google* tem boa confiabilidade;
- Redução de custos, pois não é necessário manter uma estrutura física com servidores;
- Economia de tempo, uma vez que são disponibilizadas diversas *API's* que facilitam a implementação de etapas comuns no ciclo de vida de uma aplicação (e.g., sistema de autenticação).

De acordo com Francis Ma, a missão da plataforma é tornar mais fácil o desenvolvimento de aplicativos, oferecendo soluções para os principais desafios no ciclo de vida de

¹REpresentational State Transfer - Conjunto de princípios e regras que permitem que a interface de um projeto criado seja bem definida [42]. No contexto utilizado, diz respeito mais especificamente a implementação de métodos para responder requisições HTTP, como um *GET* ou *POST*.

uma aplicação [10]. Dentre os diversos produtos providos pela plataforma, os utilizados no desenvolvimento deste trabalho serão abordados nas próximas subseções.

3.1.1 *Authentication*

Visa facilitar o desenvolvimento de um sistema de autenticação seguro e melhorar a experiência de *login* para os usuários finais. O *Firebase Authentication* oferece suporte à autenticação por meio de senhas, números de telefone e provedores de identidade federados como *Google*, *Facebook*, *Twitter* e outros [9]. Dentre as opções disponíveis, a autenticação por senha foi a utilizada na implementação deste trabalho.

Para criar uma conta utilizando a autenticação com senha, é necessário apenas um e-mail e a própria senha. Nesta forma de autenticação o provedor de *e-mail* não é validado, e por consequência é possível criar contas como “teste@teste.com” ou “a@a.com” sem problema algum, porém o formato “nome@dominio.com” deve ser respeitado. Esta e outras validações² são feitas pelo próprio *Firebase*, sendo necessário ao programador apenas tratar as exceções enviadas por ele. Um exemplo de exceção enviado pelo *Firebase* pode ser visto na Figura 3.1.

```
✖ ▶ ▷ L {code: "auth/invalid-email", message: "The email address is badly formatted."} ⓘ
  code: "auth/invalid-email"
  message: "The email address is badly formatted."
▶ __proto__: Error
```

Figura 3.1: Exemplo de exceção ao tentar criar uma conta com um *e-mail* inválido.

Quando uma conta é criada com sucesso, o *Firebase* retorna ao desenvolvedor entre outras informações o *UID* - um *token* único para cada usuário. Este *token* foi então utilizado como “chave primária” na modelagem de arquitetura dos bancos *Realtime Database* e *Cloud Storage*, que será mais bem detalhado na Subseção 4.5.1 e Subseção 4.5.2.

3.1.2 *Realtime Database*

É um banco de dados *NoSQL* hospedado na nuvem. Os dados são armazenados como uma espécie de árvore em *JSON* (*JavaScript Object Notation*) e sincronizados entre todos os clientes conectados em tempo real [11]. Assim, viu-se no *Realtime Database* duas características benéficas para a sua utilização no presente trabalho:

1. Simplicidade - Uma vez que a modelagem do banco nada mais é que a estruturação de uma árvore;
2. Atualização dos dados - Pois tendo em vista que parte fundamental deste trabalho é a criação de um *chat*, um banco de dados que promete a atualização em tempo-real dos dados para todos os usuários aparentou-se muito promissor.

²Tamanho da senha, existência ou singularidade do *e-mail* e falhas de conexão.

Sincronização

A maioria dos bancos de dados tradicionais trabalha com um modelo de solicitação e resposta baseado em *pull* [15, 23], e por isso, se for necessário saber se determinado dado foi atualizado, deve-se consultá-lo, fazendo uma chamada ao banco que por sua vez retornará uma resposta. O *Realtime Database* no entanto trabalha de forma inversa. Se o desenvolvedor necessitar observar as alterações em um determinado local da árvore, basta “pedir” isto ao banco, e ele avisará quando alguma mudança ocorrer. Para que as atualizações sejam recebidas em tempo-real pelos clientes, o *Realtime Database* não utiliza requisições HTTP típicas, e sim o protocolo *WebSocket* [17, 24].

Por conta disso, a implementação do *chat* - que aparentava ser a maior dificuldade e empecilho para a realização deste trabalho - tornou-se simples. Em alto nível, o *chat* é apenas um nó no banco de dados, o qual é “escutado” por todos os usuários da aplicação. Assim, sempre que um dos usuários enviar uma mensagem, será criado um nó filho no *chat*, e tendo em vista que uma adição na árvore é uma alteração, todos os demais usuários irão escutá-la e consequentemente permanecerão atualizados. Na Figura 3.2 é possível observar um exemplo de notificação visual utilizada pelo *Realtime Database* quando há alguma alteração no banco.



Figura 3.2: Notificação visual disparada pelo *Realtime Database* após ocorrer uma adição no banco.

3.1.3 Cloud Storage

É um banco de dados para se armazenar e compartilhar arquivos como: fotos, músicas e documentos [4]. Dentre os formatos aceitos pelo *Cloud Storage* para o *upload* de arquivos, nesta implementação foi utilizado apenas o *Base64*. Para realizar o *upload* de determinado arquivo, são necessários dois parâmetros: um *Base64* e um *path* (caminho que o arquivo será salvo no banco). Portanto, mesmo não sendo documentado oficialmente, subentende-se que o *Cloud Storage* também trabalha como uma árvore de arquivos. Uma vez salvo, o arquivo torna-se disponível para *download*, sendo necessário apenas o seu *path* para realizá-lo. A modelagem deste banco será aprofundada na Subseção 4.5.2.

3.2 Angular

Angular é um projeto de código aberto mantido primariamente pela Google. Desde seu lançamento em 2009, tem sido um dos mais populares *frameworks* para se desenvolver páginas *web* [47]. Angular é por definição, uma estrutura *JavaScript* que visa ajudar os desenvolvedores a criar aplicações, fornecendo vários recursos que tornam trivial a implementação de requisitos complexos de aplicativos modernos (e.g., roteamento e animações) [26]. Os pontos chaves da *framework* são mostrados na Figura 3.3 e brevemente comentados na sequência.

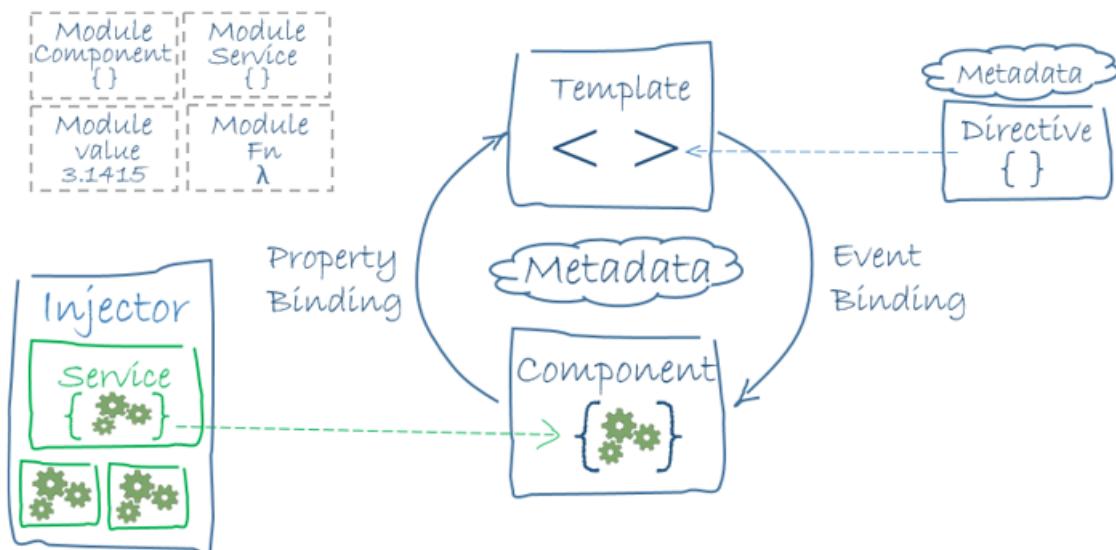


Figura 3.3: Arquitetura do Angular. Retirado de [2].

Módulo - Módulos são contêineres que armazenam um grupo de componentes, diretivas, serviços e outros. É uma biblioteca de componentes e serviços que implementa determinada funcionalidade de uma aplicação [45]. Utilizar módulos permite tirar proveito do conceito de *lazy-loading* - módulos são carregados sob demanda - o que minimiza a quantidade de código que precisa ser carregado durante a execução da

aplicação. Todos as aplicações Angular tem ao menos um módulo raiz e normalmente tem vários outros módulos de recursos [2].

Componente - Um componente é qualquer parte visual da aplicação Angular [40]. Cada componente consiste de duas partes: uma *view*, que diz respeito a interface do usuário e uma classe que implementa sua lógica [45]. Assim como comentado sobre o Módulo, aplicações Angular também possuem ao menos um Componente raiz.

Template - Os *Templates* combinam HTML com marcações Angular, que podem modificar elementos HTML antes destes serem mostrados ao usuário, gerando assim conteúdo dinâmico. Os *templates* fornecem lógica de programação e uma ligação de dados entre *view* e o DOM³ (conhecido como *bind*) [2].

Metadados - São usados para fornecer informações a uma classe. Explicitam ao Angular onde obter os principais blocos de construção necessários para criar e apresentar o componente e sua visualização. Na Figura 3.4 é mostrado o uso de três metadados.

```
@Component({
  selector: 'app-hero-list',
  templateUrl: './hero-list.component.html',
  providers: [ HeroService ]
})
```

Figura 3.4: Exemplo de metadados utilizados pelo Angular, retirado de [18].

Serviço - O serviço é uma categoria abrangente, que pode incluir valores, funções ou recursos de que a aplicação precisa. É utilizado normalmente com um propósito pequeno e bem definido. Ou seja, deve fazer algo específico, e fazê-lo com eficiência [19].

Diretivas - Como dito, *templates* são dinâmicos. Quando eles são renderizados, o DOM pode ser transformado de acordo com as instruções passadas pelas diretivas. As diretivas podem alterar o *layout* adicionado - adicionando, removendo e substituindo elementos no DOM, e alterar a aparência ou o comportamento de um elemento existente [2]. Na Figura 3.5 é mostrado um exemplo da diretiva *ngModel*, que modifica o comportamento de um elemento existente definindo sua propriedade de valor de exibição e respondendo a quaisquer eventos de alteração.

```
<input [(ngModel)]="hero.name">
```

Figura 3.5: Exemplo de uso da diretiva *ngModel*, retirado de [18].

³Interface de programação que fornece uma representação estruturada do documento como uma árvore. O DOM define métodos que permitem acesso à arvore, para que seja possível a alteração de estrutura, estilo e conteúdo do documento [21].

Injeção de dependência - É utilizado para fornecer aos componentes os serviços e outras coisas que eles necessitam. Componentes consomem serviços, dessa forma pode-se por exemplo injetar um serviço em um componente, o que fará possível ao componente acessar o serviço requisitado [19].

3.3 Apache Cordova

O *Apache Cordova* é uma estrutura de desenvolvimento móvel *opensource* que permite a utilização de tecnologias *web* para desenvolvimento em várias plataformas [22]. Também é ele o responsável por permitir que aplicações não nativas tenham acesso a funcionalidades nativas do dispositivo, como sensores, contatos e câmera [52]. A Figura 3.6 mostra uma visão de alto nível da arquitetura *Cordova*.

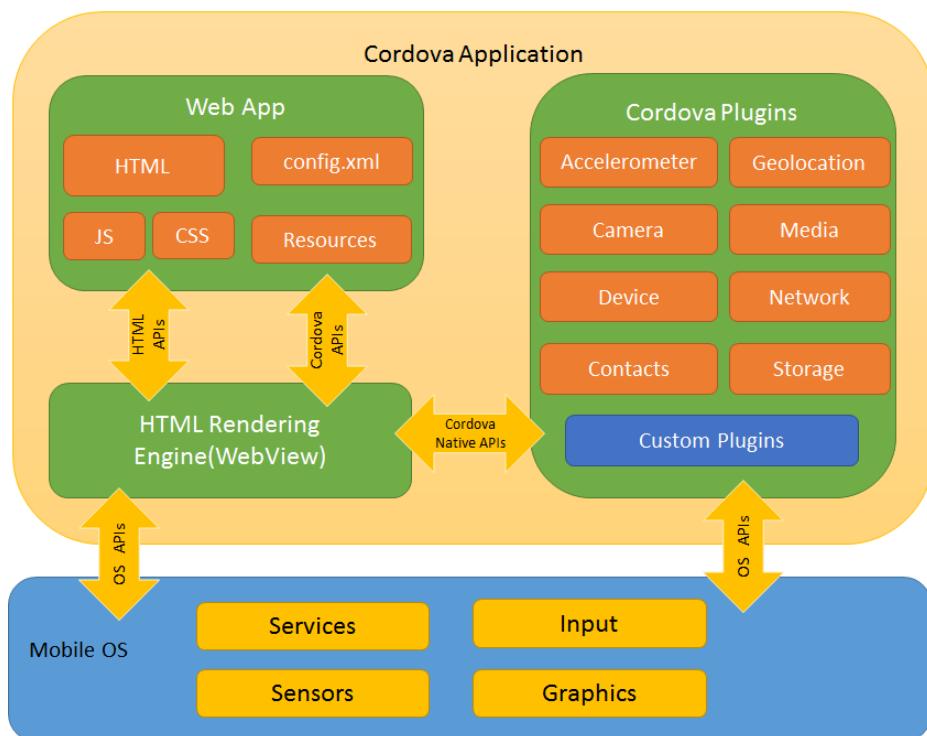


Figura 3.6: Arquitetura do *Apache Cordova*. Retirado de [22].

Assim, o *Cordova* possui apenas a tarefa de fazer com que aplicações criadas com tecnologias *web* possam executar como se fossem nativas, não sendo sua responsabilidade imitar a usabilidade e aparência de aplicativos nativos [52]. Segundo Matos este problema é resolvido com o uso de algumas *frameworks* que contêm bibliotecas de *HTML* e *CSS* [52], tal como o *Ionic*.

3.4 Ionic

O Ionic é uma *framework open-source* que utiliza tecnologias *web* (*HTML*, *CSS* e *JavaScript*) [52] para criar aplicativos híbridos. Foi lançado com base no *AngularJS* e no *Apache Cordova*, mas sua versão mais recente migrou do *AngularJS* para o *Angular* [36]. Em essência, um aplicativo que utiliza *Ionic* é apenas uma página *web* que executa em uma *webview*, deste modo é possível utilizar nele grande parte das tecnologias *web* existentes [1].

Para auxiliar o ciclo de desenvolvimento de uma aplicação *Ionic*, foi criado o *Ionic Pro* - um conjunto de serviços e recursos que visam agilizar o processo de prototipação [25]. Dentre os serviços existentes, o *Deploy* foi o mais utilizado na etapa de desenvolvimento. Este recurso permite a atualização remota da interface do usuário e a lógica de negócios de um aplicativo em tempo real, o que torna possível enviar atualizações direto para os usuários, sem passar pela loja de aplicativos [5].

Com a utilização deste recurso, sempre que necessário testar em um dispositivo real uma nova funcionalidade que foi implementada, basta salvar o novo código nos servidores do *Ionic* e logo após implantá-la. Na Figura 3.7 é mostrada a interface do *Ionic Deploy* com alguns *commits*, sendo o último deles o que foi implantado.

Deploy builds

	e93216 Teste utilização geofence 2			
	hchaves committed 2 months ago from pmaster			
	6c7085 Teste utilização geofence			
	hchaves committed 2 months ago from pmaster			
	a54714 Melhoria na atualização de páginas			
	hchaves committed 2 months ago from pmaster			
	9306a1 Aviso tem amigo na sala, exclusão-própria da sala works			
	hchaves committed 2 months ago from pmaster			
	36ab1e Adição e remoção de amigos works			
	hchaves committed 2 months ago from pmaster			
	60ae96 Correção 'compilação' convites			
	hchaves committed 2 months ago from pmaster			
	66f3dd Envio de convites, Bloqueio de usuários works			
	hchaves committed 2 months ago from pmaster			

Figura 3.7: Interface do *Ionic Deploy*.

Depois de implantada, é possível executar a nova versão em dispositivos reais utilizando o aplicativo *Ionic View*. Todavia, este processo apenas emula a execução do aplicativo [6], e por consequência muitos *plugins* não funcionam como o esperado. Assim, durante a parte inicial do desenvolvimento da aplicação, o *Ionic Deploy* foi assiduamente utilizado, mas a partir do momento que necessitou-se testar funcionalidades nativas (e.g., câmera, gps), seu uso não era mais uma opção. Deste momento em diante, utilizou-se então o

deploy real, que através de um processo de compilação⁴, gera um binário (e.g., APK para plataformas *Android*) instalável em dispositivos.

Na seção *Native* da documentação do *Ionic* são encontrados os *plugins* que fazem acesso aos recursos nativos do dispositivo, como *bluetooth*, câmera, leitor de digital e vários outros [20]. Os principais *plugins* utilizados no desenvolvimento deste trabalho foram:

- *Geolocation* - Utilizando o GPS e sinais de rede como *IP*, *WiFi* e endereços *MAC Bluetooth* este *plugin* provê informações sobre a localização do dispositivo como latitude, longitude, altitude e velocidade [14].

Uma vez instalado e importado, o *plugin* é utilizado chamando uma função de *callback*, que ao retornar traz consigo um objeto com as informações sobre a atual localização do dispositivo, conforme mostrado na Figura 3.8. São duas as funções disponíveis para obter a localização do dispositivo: *getCurrentPosition* e *watchPosition*. A primeira destas faz uma requisição única, e quando houver uma resposta, o método se dará por encerrado. A segunda função mencionada - *watchPosition* - trabalha de forma diferente. Ela permanece ativa monitorando a movimentação do dispositivo, e a qualquer sinal de mudança invoca novamente a função de *callback*.

```
this.geolocation.getCurrentPosition().then((resp) => {
  // resp.coords.latitude
  // resp.coords.longitude
}).catch((error) => {
  console.log('Error getting location', error);
});
```

Figura 3.8: Exemplo de uso do método responsável por trazer as informações sobre a localização. Retirado de [14].

- *Geofence* - Este *plugin* monitora uma área circular, dado inicialmente por uma latitude e longitude, e envia uma notificação quando o usuário entra ou sai dos limites declarados [13]. Segundo sua documentação, as *Geofences* são persistidas mesmo após a reinicialização do dispositivo, e seu monitoramento continua mesmo se o aplicativo não estiver em execução.

Após instalado e importado, para adicionar uma *geofence* deve-se chamar um método e informar alguns atributos sobre a cerca, como raio e posição. A Figura 3.9 demonstra a criação de uma cerca.

⁴Para este processo, é necessário ter previamente instalado o ambiente de desenvolvimento *Android Studio* [6].

```
private addGeofence() {
    //options describing geofence
    let fence = {
        id: '69ca1b88-6fbe-4e80-a4d4-ff4d3748acdb', //any unique ID
        latitude: 37.285951, //center of geofence radius
        longitude: -121.936650,
        radius: 100, //radius to edge of geofence in meters
        transitionType: 3, //see 'Transition Types' below
        notification: { //notification settings
            id: 1, //any unique ID
            title: 'You crossed a fence', //notification title
            text: 'You just arrived to Gliwice city center.', //notification body
            openAppOnClick: true //open app when notification is tapped
        }
    }

    this.geofence.addOrUpdate(fence).then(
        () => console.log('Geofence added'),
        (err) => console.log('Geofence failed to add')
    );
}
```

Figura 3.9: Exemplo de criação de uma *geofence*. Retirado de [13].

Capítulo 4

Desenvolvimento

Neste capítulo apresenta-se a modelagem da aplicação, utilizando os conceitos da UML - Levantamento de Requisitos, Casos de Uso e Diagrama de Classes. Também tem-se aqui explicada a modelagem dos dois bancos de dados utilizados: *Realtime Database* e *Cloud Storage*. E ao final do capítulo, encontra-se a apresentação do resultado final: um aplicativo sendo executado na plataforma Android.

4.1 Visão Geral do Sistema

O Notifique-me é um aplicativo de comunicação pensado para eventos corporativos, que utiliza como critério fundamental para se iniciar um ambiente de conversa a proximidade física, tornando assim desnecessário os vínculos tradicionais, como salvar um novo contato na agenda telefônica ou pedir uma amizade em uma rede social. Dessa forma, o objetivo do aplicativo é tornar mais fácil e prática a interação de indivíduos em locais como: congressos, simpósios, conferências, *workshops*, cursos, oficinas e outros.

A comunicação no aplicativo funcionará por meio de salas de bate-papo, que serão criadas em determinadas localizações geográficas - especificada pelos próprios usuários. Dessa forma, todo e qualquer indivíduo - estando fisicamente próximo a sala criada - poderá adentrá-la e se comunicar.

Relembrando o exemplo dado na Introdução e Justificativa deste trabalho (1), se um professor desejar transmitir uma informação para sua sala de aula, agora é dele apenas a responsabilidade de criar uma sala no aplicativo, cabendo aos alunos a tarefa de entrarem na sala criada. Assim, ainda que sejam centenas de alunos que desejam entrar na sala, e mesmo sendo desconhecidos uns aos outros, por estarem fisicamente próximos, todos poderão entrar na sala criada pelo professor. Na Figura 4.1 é demonstrado o funcionamento do aplicativo.

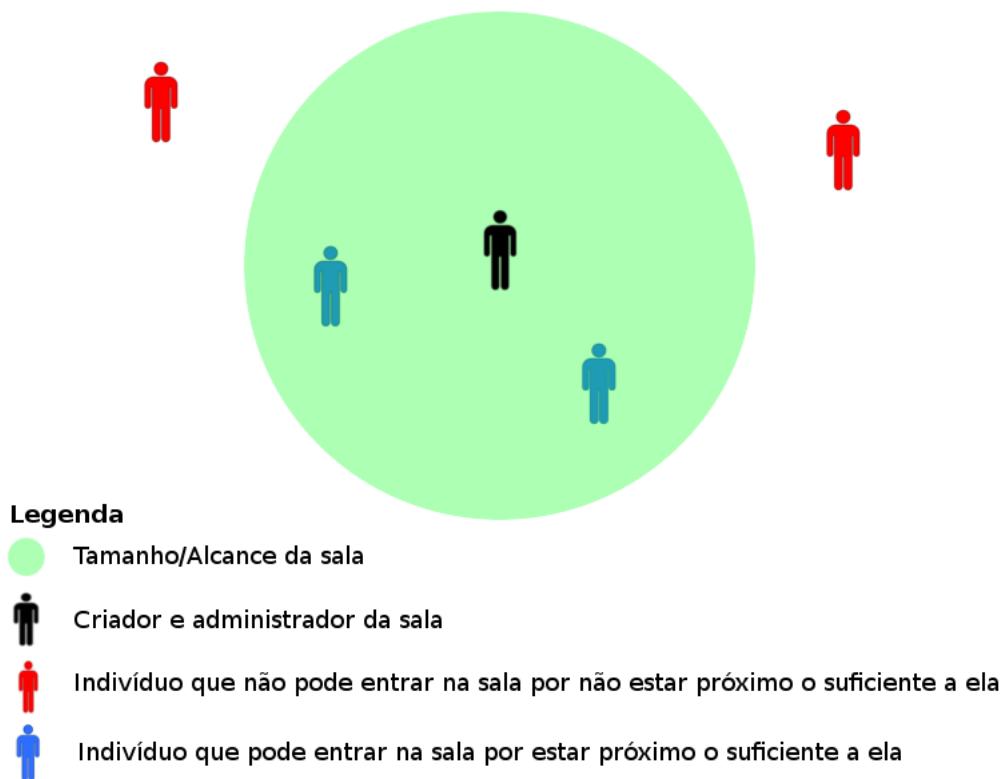


Figura 4.1: Exemplo de funcionamento do aplicativo.

4.2 Levantamento de Requisitos

A seguir estão documentados os requisitos funcionais e não-funcionais da aplicação.

4.2.1 Requisitos não funcionais

1. O aplicativo deve ser compatível com a plataforma Android.
2. O aplicativo deve ser compatível com a plataforma iOS.
3. O aplicativo deve ter uma margem de erro de até 50 metros quanto a localização.
4. O envio de mensagens no aplicativo não deve levar mais que 1 segundo.

4.2.2 Requisitos funcionais

1. O aplicativo deve permitir que um usuário se cadastre no sistema.
2. O aplicativo deve permitir que um usuário cadastrado faça *login* no sistema.

3. Com exceção do cadastro e *login*, as funcionalidades do aplicativo só devem ser acessíveis a usuários logados.
4. O aplicativo deve permitir que um usuário faça *logoff*.
5. O aplicativo deve permitir que um usuário altere sua foto de perfil.
6. O aplicativo deve permitir que um usuário altere seu apelido.
7. O aplicativo deve permitir que usuários adicionem uns aos outros como amigos.
8. O aplicativo deve permitir que um usuário crie salas de bate-papo com base em sua localização atual.
9. O aplicativo deve permitir que um usuário configure uma sala como privada - o que fará necessário uma senha para acessá-la - no momento de sua criação.
10. O aplicativo deve permitir que um usuário configure uma sala como de leitura - o que fará com que apenas o administrador tenha privilégio de envio de mensagens - no momento de sua criação.
11. O aplicativo deve permitir que um usuário delimite em metros o tamanho de uma sala no momento de sua criação.
12. O aplicativo deve enviar notificações ao usuário quando este entrar ou sair dos limites de uma sala existente.
13. O aplicativo deve permitir que um usuário configure uma mensagem de boas vindas para uma sala - que será enviada aos demais usuários quando estes entrarem nos limites da sala - no momento de sua criação.
14. O aplicativo deve permitir a escolha de um nome e foto de perfil para uma sala, no momento de sua criação.
15. O aplicativo deve permitir que o administrador de uma sala a exclua, ainda que nela hajam outros usuários.
16. O aplicativo deve permitir que o administrador de uma sala bloqueeie outros usuários.
17. O aplicativo não deve permitir que um usuário entre em uma sala de bate-papo se este estiver bloqueado.
18. O aplicativo não deve permitir que um usuário entre em uma sala de bate-papo se este estiver mais distante que o permitido nas configurações da sala.
19. O aplicativo deve exibir uma listagem de salas, com todas as salas criadas.
20. O aplicativo deve separar na listagem de salas, as próximas (o usuário poderá entrar) das distantes (o usuário não poderá entrar).
21. O aplicativo deve informar na listagem de salas, quais delas possuem ao menos um amigo presente.

22. O aplicativo deve informar na listagem de salas, quais delas são privadas.
23. Uma vez dentro de uma sala, os usuários devem ter a opção de enviar mensagens, que serão encaminhadas para todos os outros usuários presentes.
24. Uma vez dentro de uma sala, os usuários devem ter a opção de enviar fotos, que serão encaminhadas para todos os outros usuários presentes.
25. Uma vez dentro de uma sala, os usuários devem ter a opção de visualizar os outros usuários presentes.
26. Uma vez dentro de uma sala, os usuários devem ter a opção de enviar um convite de amizade aos outros usuários presentes.

4.3 Casos de Uso

A seguir estão documentados os casos de uso do sistema a ser desenvolvido, visto resumidamente em diagramas e detalhadamente em tabelas.

4.3.1 Diagramas de Casos de Uso

Na Figura 4.2 é mostrado os Casos de Uso de um usuário deslogado. Na Figura 4.3 é mostrado os Casos de Uso de um usuário logado. Por fim, na Figura 4.4 é mostrado os Casos de Uso de um usuário e de um administrador dentro de uma sala de bate-papo.

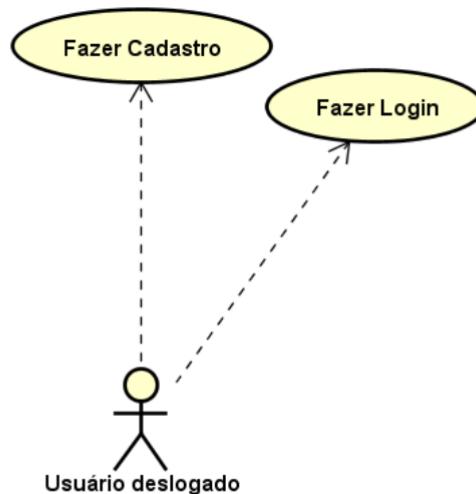


Figura 4.2: Caso de Uso de um usuário deslogado.

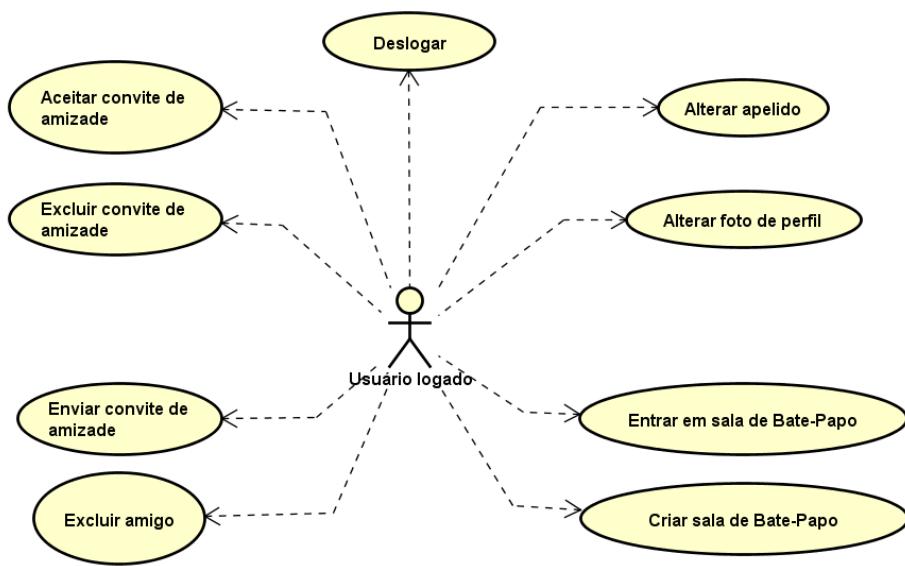


Figura 4.3: Caso de uso de um usuário logado.

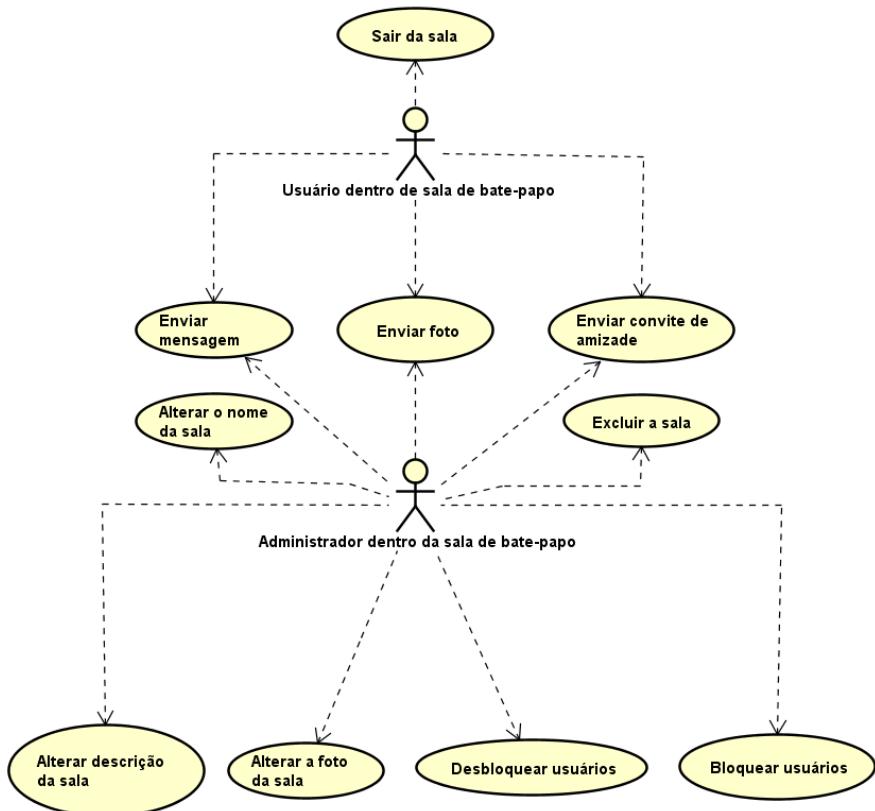


Figura 4.4: Caso de uso de um usuário e de um administrador de uma sala.

4.3.2 Casos de Uso Expandidos

As Tabelas subsequentes detalham o passo-a-passo dos Casos de Uso presentes na Figura 4.2.

Tabela 4.1: Caso de Uso - Fazer Cadastro

Nome do caso de uso	Fazer Cadastro.
Autor principal	Usuário.
Pré-condições	O usuário deve estar deslogado do sistema.
Pós-condições	O sistema criará uma conta com os dados.
Fluxo principal	<ol style="list-style-type: none"> 1. [IN] O usuário informa um e-mail e uma senha; 2. [OUT] O sistema valida as informações inseridas; 3. [OUT] O sistema cria uma nova conta e faz o login automático do usuário; 4. Fim do caso de uso.
Fluxos alternativos	<ol style="list-style-type: none"> 2a. O e-mail inserido é inválido. <ol style="list-style-type: none"> 1. O sistema exibe uma mensagem informando ao usuário que o e-mail é inválido; 2. Retorna para o passo 1 do fluxo principal. 2b. A senha inserida é menor que 6 dígitos; <ol style="list-style-type: none"> 1. O sistema exibe uma mensagem informando que uma senha deve conter ao menos 6 dígitos. 2. Retorna para o passo 1 do fluxo principal. 2c. Uma conta já existe com o e-mail informado. <ol style="list-style-type: none"> 1. O sistema exibe uma mensagem que uma conta já existe com o e-mail informado; 2. Retorna para o passo 1 do fluxo principal.

Tabela 4.2: Caso de Uso - Fazer Login

Nome do caso de uso	Fazer Login.
Autor principal	Usuário.
Pré-condições	O usuário deve estar deslogado do sistema.
Pós-condições	O usuário será logado no sistema.
Fluxo principal	<ol style="list-style-type: none"> 1. [IN] O usuário informa um e-mail e uma senha; 2. [OUT] O sistema valida as informações inseridas; 3. [OUT] O sistema faz o login do usuário; 4. Fim do caso de uso.
Fluxos alternativos	<ol style="list-style-type: none"> 2a. O e-mail inserido é inválido. <ol style="list-style-type: none"> 1. O sistema exibe uma mensagem informando ao usuário que o e-mail é inválido; 2. Retorna para o passo 1 do fluxo principal. 2b. Não existe uma conta com o e-mail informado. <ol style="list-style-type: none"> 1. O sistema exibe uma mensagem informando que não existe uma conta com o e-mail informado; 2. Retorna para o passo 1 do fluxo principal. 2c. A senha inserida está incorreta. <ol style="list-style-type: none"> 1. O sistema exibe uma mensagem informando que a senha inserida está incorreta; 2. Retorna para o passo 1 do fluxo principal.

As Tabelas subsequentes detalham o passo-a-passo dos Casos de Uso presentes na Figura 4.3.

Tabela 4.3: Caso de Uso - Fazer Logoff

Nome do caso de uso	Fazer Logoff.
Autor principal	Usuário.
Pré-condições	O usuário deve estar logado no sistema.
Pós-condições	O usuário será deslogado
Fluxo principal	<ol style="list-style-type: none"> 1. [IN] O usuário informa ao sistema que deseja realizar logoff; 2. [OUT] O sistema faz o logoff do usuário; 3. Fim do caso de uso.
Fluxos alternativos	Não há.

Tabela 4.4: Caso de Uso - Alterar Apelido

Nome do caso de uso	Alterar Apelido.
Autor principal	Usuário.
Pré-condições	O usuário deve estar logado no sistema.
Pós-condições	O usuário terá seu apelido alterado.
Fluxo principal	<ol style="list-style-type: none"> 1. [IN] O usuário informa ao sistema um novo apelido; 2. [OUT] O sistema informa que o novo apelido foi salvo; 3. Fim do caso de uso
Fluxos alternativos	Não há.

Tabela 4.5: Caso de Uso - Alterar Foto de Perfil

Nome do caso de uso	Alterar Foto de Perfil.
Autor principal	Usuário.
Pré-condições	O usuário deve estar logado no sistema.
Pós-condições	O usuário terá sua foto de perfil alterada.
Fluxo principal	<ol style="list-style-type: none"> 1. [IN] O usuário informa ao sistema sua intenção de alterar a foto de perfil; 2. [OUT] O sistema abre um aplicativo de fotografia no dispositivo do usuário; 3. [IN] O usuário tira uma fotografia; 4. [IN] O usuário avalia e aceita a fotografia tirada; 5. [OUT] O sistema informa que a foto de perfil foi salva; 6. Fim do caso de uso.
Fluxos alternativos	4a. O usuário não aceita a foto tirada. 1. O sistema cancela toda a operação.

Tabela 4.6: Caso de Uso - Criar Sala de Bate-papo

Nome do caso de uso	Criar Sala de Bate-papo.
Ator principal	Usuário.
Pré-condições	O usuário deve estar logado no sistema.
Pós-condições	O sistema criará uma nova sala de bate-papo com as informações inseridas.
Fluxo principal	<ol style="list-style-type: none"> 1. [IN] O usuário informa ao sistema o nome, a descrição, e o tamanho da sala; 2. [IN] O usuário informa ao sistema se a sala é de leitura; 3. [IN] O usuário informa ao sistema se a sala é privada; 4. [IN] O usuário informa ao sistema sua intenção de alterar a foto de perfil da sala; 5. [OUT] O sistema abre um aplicativo de fotografia no dispositivo do usuário; 6. [IN] O usuário tira uma fotografia; 7. [IN] O usuário avalia e aceita a fotografia tirada; 8. [OUT] O sistema informa que uma nova sala foi criada; 9. Fim do caso de uso.
Fluxos alternativos	<ol style="list-style-type: none"> 1a. O usuário não informa o nome da sala. <ol style="list-style-type: none"> 1. O sistema não permite a criação de uma nova sala. 1b. O usuário informa que a sala é privada mas não informa uma senha. <ol style="list-style-type: none"> 1. O sistema não permite a criação de uma nova sala. 7a. O usuário não aceita a foto tirada. <ol style="list-style-type: none"> 1. Retorna para o passo 4 do fluxo principal.

Tabela 4.7: Caso de Uso - Entrar em sala de bate-papo

Nome do caso de uso	Entrar em sala de bate-papo.
Ator principal	Usuário.
Pré-condições	O usuário deve estar logado no sistema.
Pós-condições	O usuário será direcionado para a sala requerida.
Fluxo principal	<ol style="list-style-type: none"> 1. [IN] O usuário informa ao sistema a intenção de entrar em uma sala específica; 2. [OUT] O sistema adiciona o usuário na lista de usuários presentes da sala; 3. [OUT] O sistema direciona o usuário para a sala requerida; 4. Fim do caso de uso.
Fluxos alternativos	<ol style="list-style-type: none"> 1a. A sala requerida é privada. <ol style="list-style-type: none"> 1. O sistema verifica se o usuário já é participante da sala, se sim segue para o passo 3 do fluxo principal; 2. Não sendo participante da sala, o usuário informa uma senha; 3. Estando a senha correta, segue para o passo 2 do fluxo principal; 4. Estando a senha incorreta, retorna ao passo 1 do fluxo principal. 2a. A sala requerida é de leitura. <ol style="list-style-type: none"> 1. O sistema adiciona o usuário na lista de usuários presentes da sala; 2. O sistema direciona o usuário para a sala requerida sem privilégio para o envio de mensagens.

Tabela 4.8: Caso de Uso - Enviar convite de amizade

Nome do caso de uso	Enviar convite de amizade.
Autor principal	Usuário.
Pré-condições	O usuário deve estar logado.
Pós-condições	Um convite de amizade será enviado a determinado usuário.
Fluxo principal	<ol style="list-style-type: none"> 1. [IN] O usuário informa ao sistema o e-mail do usuário o qual deseja pedir amizade; 2. [OUT] O sistema envia ao usuário do e-mail informado um pedido de vínculo de amizade; 3. Fim do caso de uso.
Fluxos alternativos	<ol style="list-style-type: none"> 1a. O e-mail informado é inválido <ol style="list-style-type: none"> 1. O sistema informa que o e-mail informado é inválido; 2. Retorna ao passo 1 do fluxo principal. 1b. Não existe um usuário cadastrado com o e-mail informado. <ol style="list-style-type: none"> 1. O sistema informa que o e-mail informado é inválido; 2. Retorna ao passo 1 do fluxo principal. 1c. O e-mail informado já possui um convite de amizade do usuário requerente. <ol style="list-style-type: none"> 1. O sistema informa que já foi enviado um convite de amizade ao e-mail indicado; 2. Retorna ao passo 1 do fluxo principal. 1d. O usuário do e-mail informado já é amigo do usuário atual. <ol style="list-style-type: none"> 1. O sistema informa que o usuário do e-mail informado já é amigo do usuário atual. 2. Retorna ao passo 1 do fluxo principal.

Tabela 4.9: Caso de Uso - Excluir amigo

Nome do caso de uso	Excluir amigo.
Autor principal	Usuário.
Pré-condições	<ol style="list-style-type: none"> 1. O usuário deve estar logado; 2. O usuário deve ter ao menos um (1) amigo.
Pós-condições	Um vínculo de amizade será desfeito.
Fluxo principal	<ol style="list-style-type: none"> 1. [IN] O usuário informa ao sistema o amigo o qual quer desfazer o vínculo de amizade; 2. [OUT] O sistema realiza a desvinculação; 3. [OUT] O sistema atualiza a lista de amigos do usuário; 4. Fim do caso de uso.
Fluxos alternativos	Não há.

Tabela 4.10: Caso de Uso - Aceitar convite de amizade

Nome do caso de uso	Aceitar convite de amizade.
Autor principal	Usuário.
Pré-condições	<ol style="list-style-type: none"> 1. O usuário deve estar logado; 2. O usuário deve possuir ao menos um (1) convite de amizade.
Pós-condições	Um vínculo de amizade será realizado.
Fluxo principal	<ol style="list-style-type: none"> 1. [OUT] O sistema vincula o usuário atual e o requerente como amigos. 2. [OUT] O sistema exclui o convite e atualiza a lista de amigos do usuário; 3. Fim do caso de uso.
Fluxos alternativos	Não há.

Tabela 4.11: Caso de Uso - Recusar convite de amizade

Nome do caso de uso	Recusar convite de amizade.
Autor principal	Usuário.
Pré-condições	1. O usuário deve estar logado; 2. O usuário deve possuir ao menos um (1) convite de amizade.
Pós-condições	Um convite de amizade será excluído.
Fluxo principal	1. [OUT] O sistema exclui o convite de amizade; 2. [OUT] O sistema atualiza a lista de amigos do usuário; 3. Fim do caso de uso.
Fluxos alternativos	Não há.

As Tabelas subsequentes detalham o passo-a-passo dos Casos de Uso presentes na Figura 4.4.

Tabela 4.12: Caso de Uso - Sair de sala de bate-papo

Nome do caso de uso	Sair de sala de bate-papo.
Autor principal	Usuário.
Pré-condições	1. O usuário deve estar logado; 2. O usuário deve estar em uma sala de bate-papo.
Pós-condições	O usuário será removido da lista de usuários presentes da sala.
Fluxo principal	1. [OUT] O sistema remove o usuário da lista de usuários presentes da sala; 2. [OUT] O sistema redireciona o usuário para a lista de salas; 3. Fim do caso de uso.
Fluxos alternativos	Não há.

Tabela 4.13: Caso de Uso - Enviar mensagem em uma sala de bate-papo

Nome do caso de uso	Enviar mensagem em uma sala de bate-papo.
Autor principal	Usuário.
Pré-condições	1. O usuário deve estar logado; 2. O usuário deve estar em uma sala de bate-papo.
Pós-condições	Será enviada uma mensagem na sala de bate-papo.
Fluxo principal	1. [IN] O usuário informa a mensagem que deseja enviar; 2. [OUT] O sistema salva a mensagem na sala atual; 3. [OUT] O sistema atualiza as mensagens enviadas da sala; 4. Fim do caso de uso.
Fluxos alternativos	Não há.

Tabela 4.14: Caso de Uso - Enviar foto em uma sala de bate-papo

Nome do caso de uso	Enviar foto em uma sala de bate-papo.
Autor principal	Usuário.
Pré-condições	1. O usuário deve estar logado; 2. O usuário deve estar em uma sala de bate-papo.
Pós-condições	Será enviada uma foto na sala de bate-papo
Fluxo principal	1. [OUT] O sistema abre um aplicativo de fotografia no dispositivo do usuário; 2. [IN] O usuário tira uma fotografia; 3. [IN] O usuário avalia e aceita a fotografia tirada; 4. [OUT] O sistema salva a foto na sala atual; 5. [OUT] O sistema atualiza as mensagens enviadas da sala; 6. Fim do caso de uso.
Fluxos alternativos	3a. O usuário não aceita a foto tirada. 1. O sistema cancela toda a operação.

Tabela 4.15: Caso de Uso - Enviar convite de amizade em uma sala de bate-papo.

Nome do caso de uso	Enviar convite de amizade em uma sala de bate-papo
Autor principal	Usuário.
Pré-condições	1. O usuário deve estar logado; 2. O usuário deve estar em uma sala de bate-papo; 3. Na sala de bate-papo deve existir ao menos um (1) outro usuário.
Pós-condições	Será enviado um convite de amizade para o usuário selecionado.
Fluxo principal	1. [IN] O usuário seleciona um indivíduo na lista de usuários presentes na sala. 2. [OUT] O sistema envia ao usuário do e-mail informado um pedido de vínculo de amizade; 3. Fim do caso de uso.
Fluxos alternativos	1a. O indivíduo selecionado já é amigo do usuário. 1. O sistema informa que o indivíduo selecionado já é amigo do usuário. 1b. O indivíduo selecionado já possui um convite de amizade do usuário requerente. 1. O sistema informa que já foi enviado um convite de amizade ao indivíduo selecionado.

Tabela 4.16: Caso de Uso - Excluir a sala

Nome do caso de uso	Excluir a sala.
Autor principal	Usuário.
Pré-condições	1. O usuário deve estar logado; 2. O usuário deve estar em uma sala de bate-papo; 3. O usuário deve ser o administrador da sala de bate-papo.
Pós-condições	A sala será removida da lista de salas.
Fluxo principal	1. [OUT] O sistema remove todos os usuários presentes na sala; 2. [OUT] O sistema remove a sala da lista de salas; 3. [OUT] O sistema informa que a sala foi excluída; 4. [OUT] O sistema atualiza a lista de salas; 5. Fim do caso de uso.
Fluxos alternativos	Não há.

Tabela 4.17: Caso de Uso - Alterar o nome da sala

Nome do caso de uso	Alterar o nome da sala.
Autor principal	Usuário.
Pré-condições	<ol style="list-style-type: none"> 1. O usuário deve estar logado; 2. O usuário deve estar em uma sala de bate-papo; 3. O usuário deve ser o administrador da sala de bate-papo.
Pós-condições	O nome da sala será alterado.
Fluxo principal	<ol style="list-style-type: none"> 1. [IN] O usuário informa o novo nome da sala; 2. [OUT] O sistema informa que o nome da sala foi alterado; 3. Fim do caso de uso.
Fluxos alternativos	<ol style="list-style-type: none"> 1a. O nome informado é vazio. <ol style="list-style-type: none"> 1. O sistema informa que o nome da sala não pode ser vazio; 2. Retorna ao passo 1 do fluxo principal.

Tabela 4.18: Caso de Uso - Alterar a descrição da sala

Nome do caso de uso	Alterar a descrição da sala.
Autor principal	Usuário.
Pré-condições	<ol style="list-style-type: none"> 1. O usuário deve estar logado; 2. O usuário deve estar em uma sala de bate-papo; 3. O usuário deve ser o administrador da sala de bate-papo.
Pós-condições	A descrição da sala será alterada.
Fluxo principal	<ol style="list-style-type: none"> 1. [IN] O usuário informa a nova descrição da sala; 2. [OUT] O sistema informa que a descrição da sala foi alterada; 3. Fim do caso de uso.
Fluxos alternativos	Não há.

Tabela 4.19: Caso de Uso - Alterar a foto da sala

Nome do caso de uso	Alterar a foto da sala.
Autor principal	Usuário.
Pré-condições	<ol style="list-style-type: none"> 1. O usuário deve estar logado; 2. O usuário deve estar em uma sala de bate-papo; 3. O usuário deve ser o administrador da sala de bate-papo.
Pós-condições	A foto de perfil da sala será alterada.
Fluxo principal	<ol style="list-style-type: none"> 1. [OUT] O sistema abre um aplicativo de fotografia no dispositivo do usuário; 2. [IN] O usuário tira uma fotografia; 3. [IN] O usuário avalia e aceita a fotografia tirada; 4. [OUT] O sistema informa que a foto de perfil da sala foi salva; 5. Fim do caso de uso.
Fluxos alternativos	<ol style="list-style-type: none"> 3a. O usuário não aceita a foto tirada. <ol style="list-style-type: none"> 1. O sistema cancela toda a operação.

Tabela 4.20: Caso de Uso - Bloquear usuário da sala de bate-papo

Nome do caso de uso	Bloquear usuário da sala de bate-papo
Autor principal	Usuário.
Pré-condições	<ol style="list-style-type: none"> 1. O usuário deve estar logado; 2. O usuário deve estar em uma sala de bate-papo; 3. O usuário deve ser o administrador da sala de bate-papo; 4. Na sala deve haver ao menos um (1) outro usuário desbloqueado.
Pós-condições	Um usuário será bloqueado da sala.
Fluxo principal	<ol style="list-style-type: none"> 1. [IN] O usuário seleciona o indivíduo que deseja bloquear; 2. [OUT] O sistema remove da sala o indivíduo selecionado; 3. Fim do caso de uso.
Fluxos alternativos	Não há.

Tabela 4.21: Caso de Uso - Desbloquear usuário da sala de bate-papo

Nome do caso de uso	Desbloquear usuário da sala de bate-papo.
Autor principal	Usuário.
Pré-condições	<ol style="list-style-type: none"> 1. O usuário deve estar logado; 2. O usuário deve estar em uma sala de bate-papo; 3. O usuário deve ser o administrador da sala de bate-papo; 4. Na sala deve haver ao menos um (1) outro usuário bloqueado.
Pós-condições	Um usuário será desbloqueado da sala.
Fluxo principal	<ol style="list-style-type: none"> 1. [IN] O usuário seleciona o indivíduo que deseja desbloquear; 2. [OUT] O sistema remove da lista de usuários bloqueados da sala o indivíduo selecionado; 3. Fim do caso de uso.
Fluxos alternativos	Não há.

4.4 Diagrama de Classe

Na Figura 4.5 é possível observar o Diagrama de Classe do sistema. Neste diagrama pode-se entender que:

- Um usuário pode possuir nenhum ou n amigos, e que cada amigo é nada mais que um usuário;
- Um usuário pode possuir nenhum ou n convites de amizade, e que cada convite é nada mais que um usuário;
- Uma sala deve possuir apenas um criador (usuário), mas este criador pode possuir outras n salas.
- Uma sala pode conter nenhum ou n participantes (usuários), e cada participante pode estar em nenhuma ou em outras n salas;
- Uma sala pode conter nenhuma ou n mensagens, mas cada mensagem pertence a somente uma sala;

- Um usuário pode ter escrito nenhuma ou n mensagens, mas cada mensagem pertence a somente um usuário;

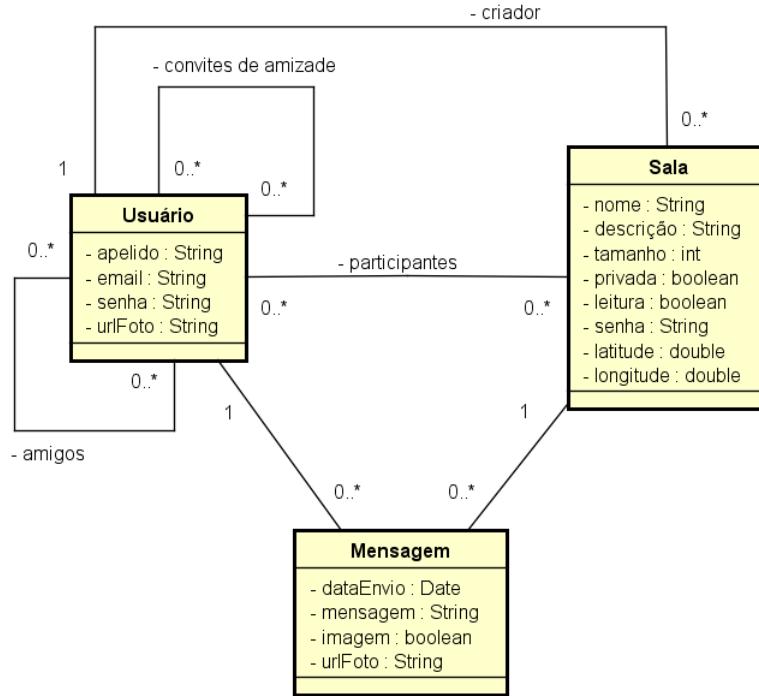


Figura 4.5: Diagrama de classe do sistema.

4.5 Modelagem dos Bancos de Dados

Nesta seção, são expostas as modelagens dos dois bancos utilizados na implementação do Notifique-me, sendo eles o *Realtime Database* e *Cloud Storage*.

4.5.1 *Realtime Database*

Neste tópico será apresentado e detalhado a forma como foi modelado o banco de dados principal da aplicação. Como comentado na Seção 3.1, o *Realtime Database* estrutura os dados como uma única grande árvore. Na Figura 4.6 é possível observar que o banco de dados, em seu nível mais alto, possui apenas 2 nós filhos: **salas** e **usuários**.



app-tcc-793bf

- + salas
- + usuarios

Figura 4.6: Nível mais alto do banco *Realtime Database*.

Salas

Na Figura 4.7 é possível observar que no nó **salas** são armazenadas todas as salas criadas pela aplicação, identificadas por uma chave criada pelo próprio *Firebase*.

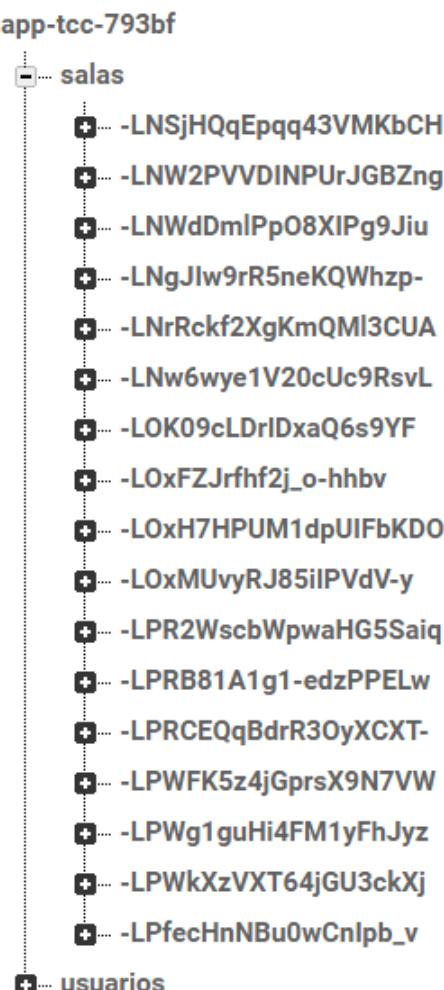


Figura 4.7: Salas criadas na aplicação, salvas no banco *Realtime Database*.

Cada sala criada, além de sua chave, armazena as seguintes informações:

apenasLeitura - Boolean que armazena se a sala é de leitura;

coordenadas - Objeto que armazena a latitude e longitude (em Graus Decimais), do local de criação da sala.

criador - O email do criador da sala;

descricao - A descrição/mensagem de boas-vindas da sala;

mensagens - Objeto que armazena todas as mensagens enviadas na sala;

nome - Nome da sala;

privada - Boolean que armazena se a sala é privada;

raio - Raio da sala, em metros;

senha - Senha para acessar a sala, caso esta for privada.

usuarios - Objeto que armazena todos os usuários presentes na sala.

Na Figura 4.8 é mostrado um exemplo de sala salva no banco.

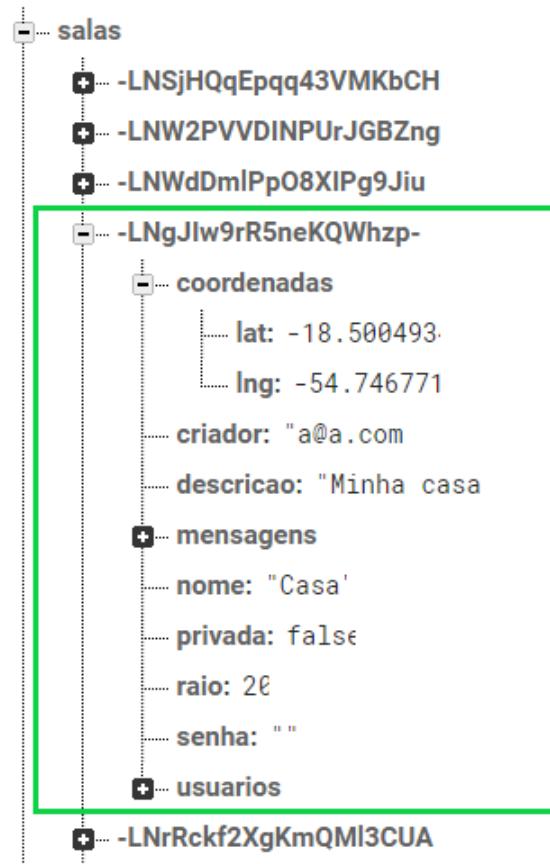


Figura 4.8: Detalhes de uma sala, salva no banco *Realtime Database*.

Na Figura 4.9 é possível observar que no objeto **mensagens**, são armazenados como outros objetos todas as mensagens enviadas nesta sala. Na Figura 4.10 é mostrado os detalhes de dois objetos **mensagem**. Cada objeto é composto por:

apelidoAutor - Apelido do autor da mensagem;

dataEnvio - Hora de envio da mensagem;

emailAutor - Email do autor da mensagem;

mensagem - Texto da mensagem enviada;

imagem - Boolean que armazena se a mensagem é uma foto ou não;

urlFoto - Url de *download*, caso a mensagem seja uma foto;

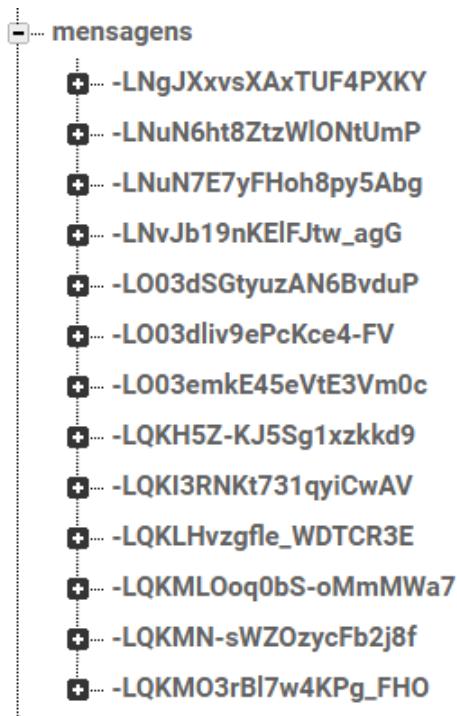


Figura 4.9: Mensagens de uma sala, salvas no banco *Realtime Database*.



Figura 4.10: Exemplo de mensagens sem e com imagem, enviadas a uma sala, salvas no banco *Realtime Database*.

Na Figura 4.11 é possível observar que no objeto **usuarios**, são armazenados como outros objetos todos os usuários presentes na sala. Como mencionado na Seção 3.1.1, é válido salientar que todas as **chaves** - se tratando de usuários - utilizadas no *Realtime Database* são a variável *uid*, advindas do *Firebase Authentication* no momento de criação de uma conta. Na Figura 4.12 são expostos dois usuários, onde é possível notar que cada objeto armazena apenas um booleano, indicando se este usuário está bloqueado.

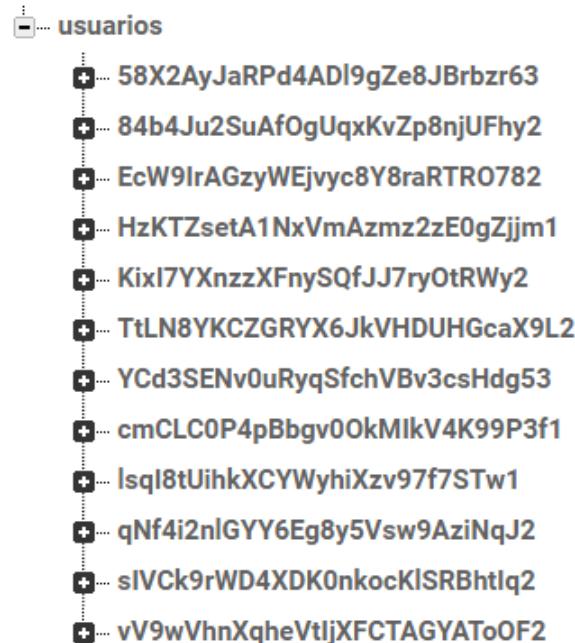


Figura 4.11: Usuários de uma sala, salvos no banco *Realtime Database*.



Figura 4.12: Exemplo de como são salvos usuários não-bloqueados e bloqueados no banco *Realtime Database*.

Usuários

O objeto **usuarios** (segundo nó raiz do banco) armazena todos os usuários da aplicação. Cada usuário armazena consigo:

amigos - Lista de amigos deste usuário;

convites - Lista de convites enviados a este usuário;

email - Email deste usuário;

nome - Nome/apelido deste usuário.

Na Figura 4.13 é mostrado alguns usuários salvos na aplicação, e na Figura 4.14 é mostrado com mais detalhes um usuário específico.

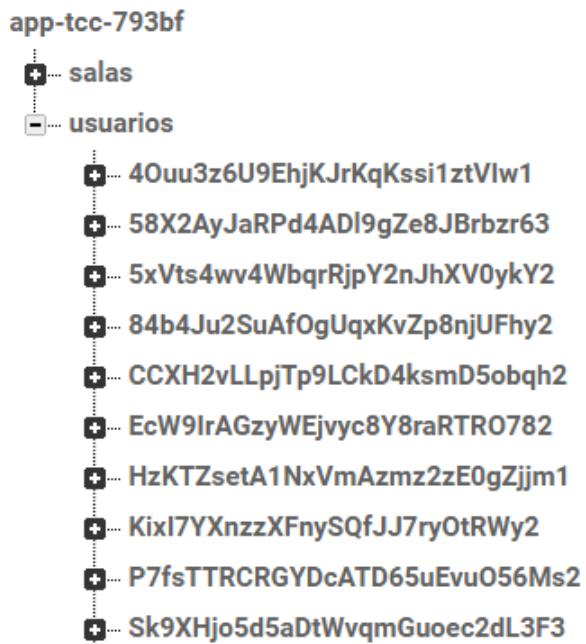


Figura 4.13: Lista de usuários da aplicação, salvos no banco *Realtime Database*.



Figura 4.14: Conteúdo de um usuário específico, salvo no banco *Realtime Database*.

Na Figura 4.15 é possível observar que as listas de **amigos** e **convites** armazenam apenas a chave de um outro usuário.

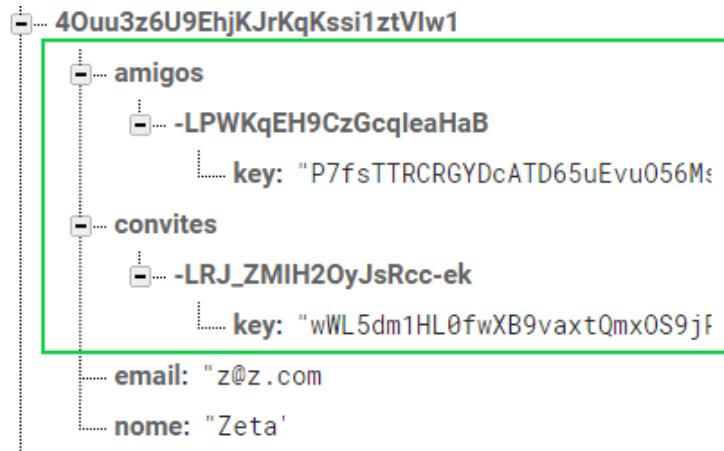


Figura 4.15: Lista de amigos e convites de um usuário, salvos no banco *Realtime Database*.

4.5.2 *Cloud Storage*

Assim como na modelagem do *Realtime Database*, o *Cloud Storage* também possui apenas 2 filhos em seu nível mais alto: **salas** e **usuarios** (Figura 4.16).

	Nome	Tamanho	Tipo	Última modificação
<input type="checkbox"/>	salas/	—	Pasta	—
<input type="checkbox"/>	usuarios/	—	Pasta	—

Figura 4.16: Nível mais alto do banco *Cloud Storage*.

Salas

Conforme mostrado na Figura 4.17, dentro de **salas** armazenam-se todas as salas criadas na aplicação, identificadas pela mesma chave criada pelo *Realtime Database*.

	Nome	Tamanho	Tipo	Última modificação
□	└ -LNgJlw9rR5neKQWhzp-/	—	Pasta	—
□	└ -LnRckf2XgKmQMI3CUA/	—	Pasta	—
□	└ -Lnw7Oepes0-SVRqcQUw/	—	Pasta	—
□	└ -LNWdDmlPpO8XIPg9Jiu/	—	Pasta	—
□	└ -LOK09cLDrlDxaQ6s9YF/	—	Pasta	—
□	└ -LoxMUvyRJ85lIPVdV-y/	—	Pasta	—
□	└ -LPRB81A1g1-edzPPELw/	—	Pasta	—
□	└ -LPRCEEqqBdrR3OyXCXT-/	—	Pasta	—

Figura 4.17: Lista de salas no banco *Cloud Storage*.

Na Figura 4.18 é possível verificar que dentro de cada sala, haverão no máximo 2 filhos, sendo eles:

1. A foto de perfil da sala;
2. Uma pasta, contendo as fotos enviadas como mensagens.

□	fotoSala	243,45 KB	image/jpeg	9 de nov de 2018
□	└ mensagens/	—	Pasta	—

Figura 4.18: Conteúdo de uma sala específica, salva no banco *Cloud Storage*.

Dentro da pasta **mensagens**, estão - identificadas com a mesma chave criada pelo *Realtime Database* - todas as imagens enviadas a esta sala (Figura 4.19).

	Nome	Tamanho	Tipo	Última modificação
□	└ -LQKH72e9FyXJBIn2jRL	248,87 KB	image/jpeg	2 de nov de 2018
□	└ -LQKKVXOAcfcTCXUyG5K	242,84 KB	image/jpeg	2 de nov de 2018
□	└ -LQKLHvzgfile_WDTCR3E	192,62 KB	image/jpeg	2 de nov de 2018
□	└ -LQKMN-sWZ0zyFcB2j8f	222,89 KB	image/jpeg	2 de nov de 2018

Figura 4.19: Lista de fotos enviadas a uma sala, salvadas no banco *Cloud Storage*.

Usuários

Conforme mostrado na Figura 4.20, na pasta/nó **usuarios** encontram-se as fotos de perfil de todos os usuários da aplicação - identificadas com a mesma chave criada pelo *Firebase Authentication*.

	Nome	Tamanho	Tipo	Última modificação
<input type="checkbox"/>	4Ouu3z6U9EhjKJrKqKssi1ztVlw1	240,62 KB	image/jpeg	23 de out de 2018
<input type="checkbox"/>	sIVCk9rWD4XDK0nkocKlSRBhtlq2	176,74 KB	image/jpeg	22 de out de 2018
<input type="checkbox"/>	Sk9XHjo5d5aDtWvqmGuoe2dL3F3	280,2 KB	image/jpeg	15 de out de 2018
<input type="checkbox"/>	TtLN8YKCZGRYX6JkVHDUHGcaX9L2	177,35 KB	image/jpeg	9 de nov de 2018
<input type="checkbox"/>	wWL5dm1HL0fwXB9vaxtQmxOS9jP2	214,02 KB	image/jpeg	14 de nov de 2018

Figura 4.20: Fotos de perfil dos usuários da aplicação, salvas no banco *Cloud Storage*.

4.6 Prototipação

Assim, utilizando todas as tecnologias mencionadas anteriormente neste documento, criou-se o Notifique-me, um aplicativo *mobile* capaz de responder aos requisitos anteriormente citados. Nesta Seção serão expostas fotos do resultado final - o aplicativo executando em uma plataforma Android - e comentado alguns detalhes da implementação. A tela inicial do aplicativo, responsável pelo *login* de um usuário, pode ser observada na Figura 4.22.

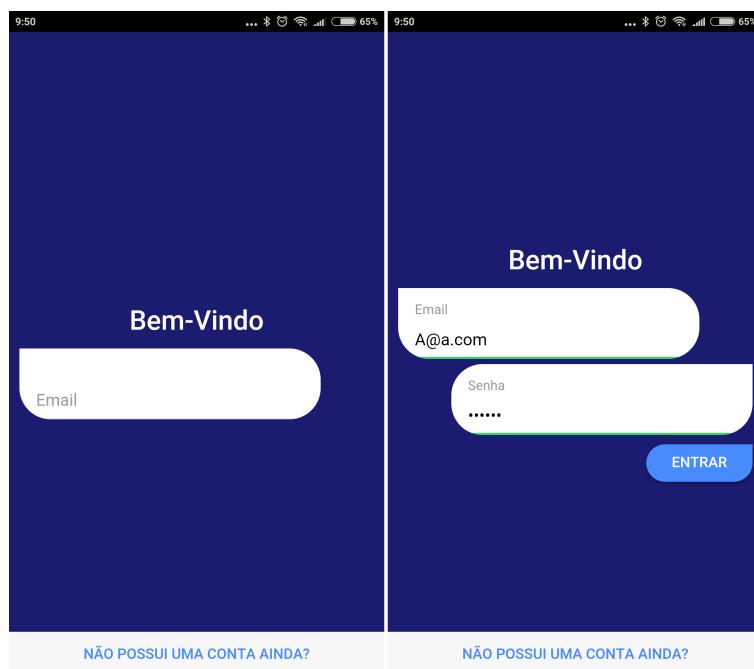


Figura 4.21: Tela de *login* da aplicação.

Na Figura 4.22 pode ser observada a tela de criação de nova conta da aplicação.

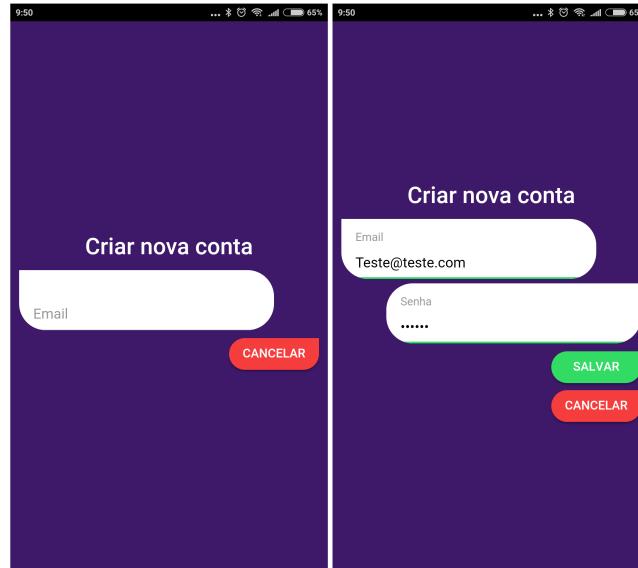


Figura 4.22: Tela de registro da aplicação.

Uma vez realizado o *login*, o usuário pode acessar as 3 telas principais da aplicação: Salas, Amigos e Configurações.

Salas

Conforme mostrado na Figura 4.23, após realizar o *login* o usuário é redirecionado diretamente para a tela Salas, que exibe todas as salas de bate-papo da aplicação.

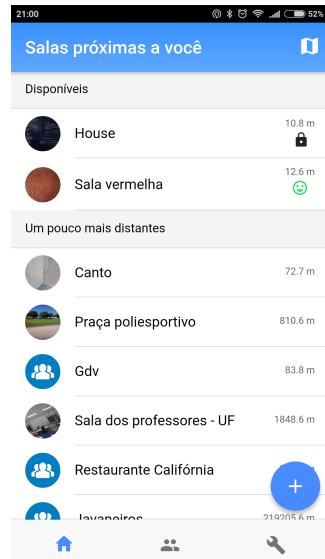


Figura 4.23: Tela principal da aplicação, que exibe todas as salas existentes.

Nesta tela, é informado ao usuário algumas informações de cada sala, sendo elas:

1. A distância do usuário até a sala;
2. Um pequeno indicador para quando há algum amigo do usuário na sala;
3. Um pequeno indicador para quando a sala é privada.

Nesta tela, as salas são divididas em dois grupos: *Disponíveis* e *Um pouco mais distantes*. O primeiro destes contém as salas próximas o suficiente para permitir a entrada do usuário. A segunda armazena as demais salas existentes. Para que as salas sejam divididas nestes grupos, os seguintes passos são tomados:

1. Utiliza-se a função *watchPosition* para se monitorar a localização atual do dispositivo;
2. Faz-se uma requisição ao *Firebase Realtime Database* para se obter todas as salas criadas;
3. Para cada sala obtida, calcula-se a fórmula de *Haversine* a fim de descobrir a distância em linha reta entre dispositivo e sala. Os parâmetros passados para função são: a latitude e longitude do dispositivo (advindos da função *watchPosition*) e a latitude e longitude da sala (salvos nas configurações da própria sala).
4. As salas em que seus respectivos cálculos obtiverem resultado menor ou igual ao tamanho salvo nas configurações da sala serão postas no grupo *Disponíveis*, as demais serão postas no grupo *Um pouco mais distantes*.

Como mencionado na Seção 3.4, a função *watchPosition* monitora continuamente a localização do dispositivo, e a qualquer sinal de mudança, executa uma função de *callback*. Assim a implementação óvia seria executar os passos anteriormente mencionados no *callback* da função *watchPosition*, pois desta forma sempre que ocorre-se uma movimentação do dispositivo, as salas seriam novamente carregadas. Todavia, ao seguir esta estratégia de implementação notou-se um problema: mesmo com o dispositivo fisicamente imóvel, a função *watchPosition* continua sendo chamada inúmeras vezes em curtos períodos de tempo, com resultados de localização diferentes (que em muitos casos se mostraram imprecisos). Com isso, a aplicação tornou-se inutilizável, pois uma vez na tela inicial, mal terminava-se uma requisição outra já era iniciada.

Para contornar o problema, foi necessário utilizar outra variável vinda no retorno da função *watchPosition* - a acurácia¹. A estratégia utilizada então foi utilizar a posição com a menor acurácia possível. Para isso, seguiu-se os seguintes passos:

1. Uma variável *aux* é setada com um grande número (e.g., 9999) antes de se iniciar a função *watchPosition*;

¹Precisão da latitude e longitude, dada em metros [14]

2. Inicia-se a função `watchPosition`;
3. Quando a função `watchPosition` é resolvida, verifica-se se a acurácia vinda em seu retorno é menor do que a presente na variável `aux`;
 - Se sim, a nova acurácia é setada na variável `aux`, e nada mais é feito até o próximo retorno da função `watchPosition`;
 - Se não, para-se de “escutar” a função `watchPosition`, e o cálculo de distância das salas é feito com base nesta última localização obtida.

Na Figura 4.24 é mostrado estes passos em um código simplificado.

```
function atualizarSalas() {
  let aux = 9999;

  watchPosition(function (posicao) {
    if (posicao.acuracia < aux) {
      // A posição vinda é mais precisa que a anterior
      aux = posicao.acuracia
    } else{
      // Cancelar o watch
      // Realizar o cálculo de distância
    }
  })
}
```

Figura 4.24: Código simplificado para capturar a posição com menor acurácia.

Infelizmente porém, notou-se que a estratégia implementada não é perfeita. Em alguns casos testados, mesmo quando a acurácia tem valores baixos, como 2 ou 3, a localização não é suficientemente precisa, estando as vezes a cerca de 50m do local real. Mas mesmo não sendo perfeita, esta implementação foi mantida por obter resultados superiores as demais.

Conforme mostrado na Figura 4.25, além da visualização por lista as salas podem ser visualizadas em um mapa com seus respectivos tamanhos, o que visa mostrar ao usuário se este está próximo de cruzar os limites de determinada sala.

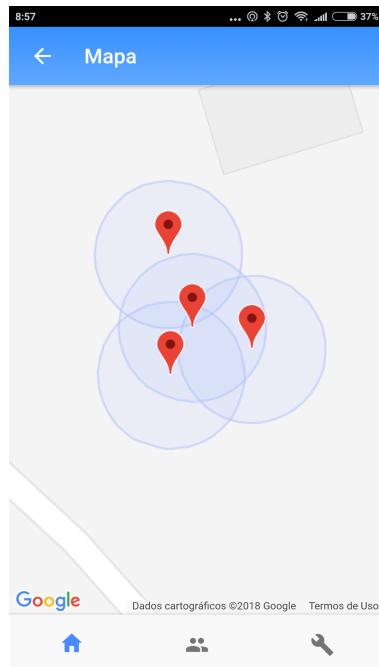


Figura 4.25: Salas de bate-papo dispostas em um mapa.

Na tela principal, tocando no botão flutuante no canto inferior direito, apresenta-se a tela de criação de nova sala, que pode ser observada na Figura 4.26. Quando esta tela é chamada, é enviada a ela por parâmetro a última localização obtida pela função *watchPosition*, pois se uma nova localização for requisitada, há um real risco de ser retornado uma posição diferente do local tido como atual no momento de abertura da nova página.

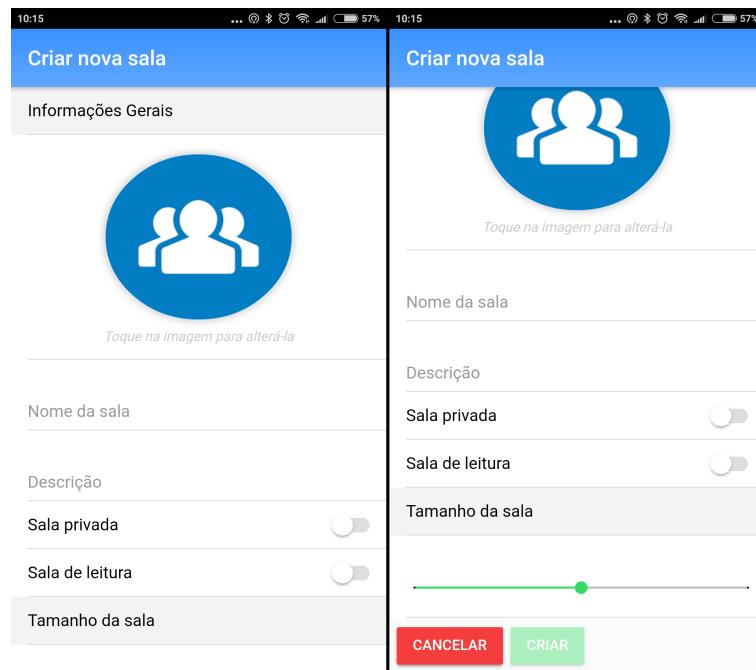


Figura 4.26: Tela de criação de sala.

No momento de criação da sala, o usuário pode escolher a opção “Sala de leitura”, o que fará com que somente ele (criador) tenha permissão para envio de mensagens e fotos. Na Figura 4.27 é mostrado um exemplo de sala comum e uma sala de leitura.

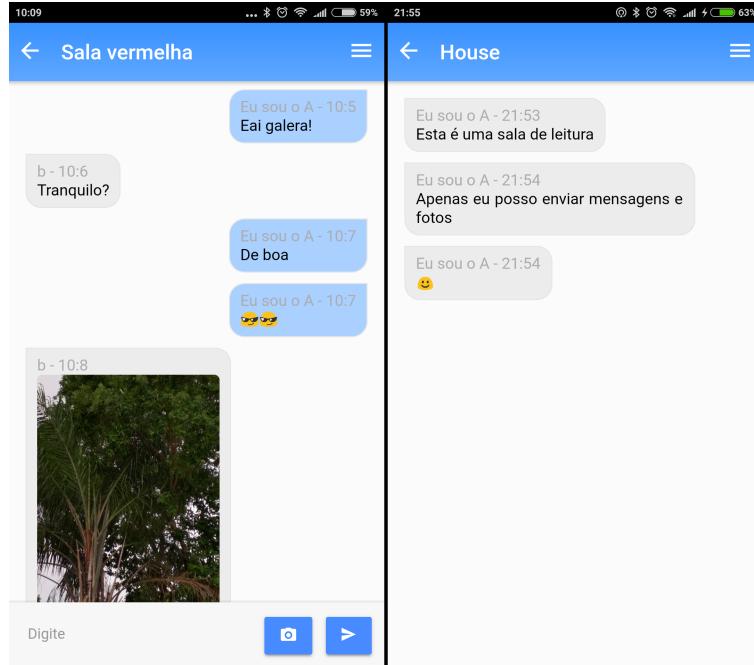


Figura 4.27: Sala de bate-papo comum e sala de leitura.

Nesta tela, as conversas são obtidas e exibidas basicamente “escutando” o nó mensagens da sala em questão no banco *Realtime Database*. Isto é, todo usuário ao adentrar na sala passa a escutar o nó mensagens no banco de dados. Se ele ou outro usuário enviar uma mensagem, o banco enviará para todos aqueles que o estão escutando esta nova alteração. Dessa forma, basta exibir toda nova alteração obtida do banco como novas mensagens.

Uma vez dentro de uma sala, um usuário pode navegar até as Configurações de Sala, exibida após tocar no ícone no canto superior direito. Pode ser observado na Figura 4.28 que nesta tela, além da opção de deixar a sala, é exibido ao usuário todos os demais usuários presentes nela e uma opção caso deseje enviar convites de amizades para qualquer um deles.

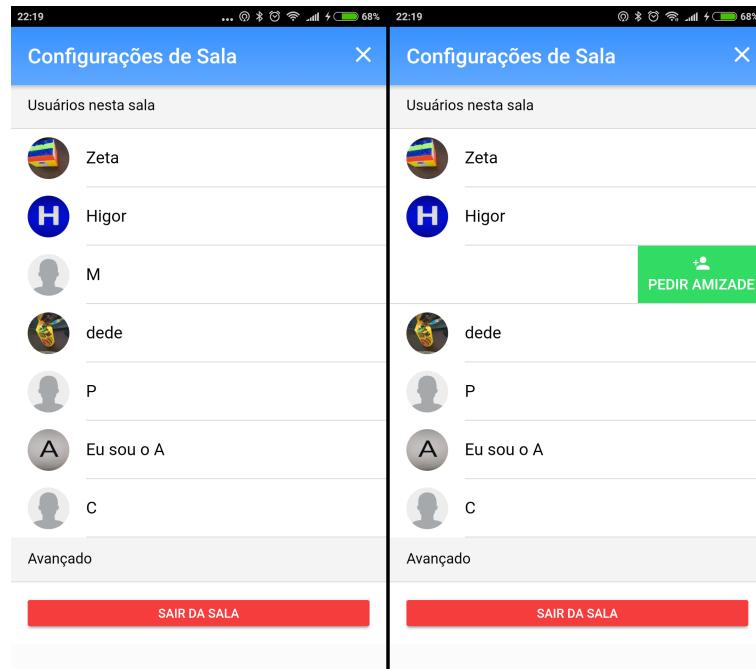


Figura 4.28: Configurações de Sala exibida a um usuário comum.

Se o usuário é o criador da sala porém, mais opções mostram-se disponíveis nas Configurações de Sala. Na Figura 4.29 pode-se observar que além das funções disponíveis a usuários comuns, o criador/administrador da sala tem acesso a:

1. Troca de foto de perfil, nome e descrição da sala;
2. Bloqueio ou desbloqueio de usuários;
3. Exclusão da sala.

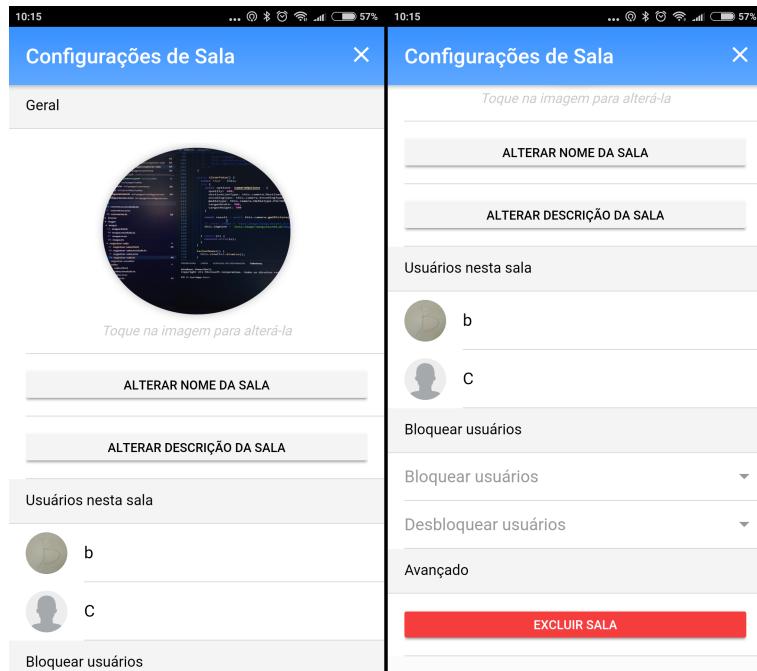


Figura 4.29: Configurações de Sala exibida ao criador da sala.

Durante a implementação do método Excluir Sala, percebeu-se que não se pode simplesmente deletar o nó da sala específica no banco de dados. Isto dá-se ao fato de que se for pedido para o *Realtime Database* adicionar um novo nó em um caminho inexistente, ele criará o caminho, e adicionará o novo nó sem apresentar nenhum problema. Exemplificando: supondo que exista uma sala *A*, e nela está presente o usuário *u*, quando o usuário *u* envia uma mensagem, o caminho enviado ao banco para salvar esta mensagem é algo como: *salas/u/mensagens*. Se a sala for deletada, e o usuário tentar enviar uma mensagem, o banco “verá” que o caminho *salas/u/mensagens* não existe, e tratará de criá-lo. Desse modo, a atitude feita pelo administrador, pode ser desfeita por um usuário comum.

Para não passar por este cenário propício a erros, antes de excluir a sala bastou-se excluir todos os usuários presentes. Para isto foi empregada a mesma função utilizada para bloquear usuários, e só após todos terem sido bloqueados, a sala é devidamente excluída.

Amigos

Conforme mostrado na Figura 4.30, na tela Amigos encontram-se todos os convites pendentes e amigos do usuário. Nesta tela pode-se aceitar ou recusar os convites pendentes, excluir os laços de amizade e enviar um convite a determinado usuário.

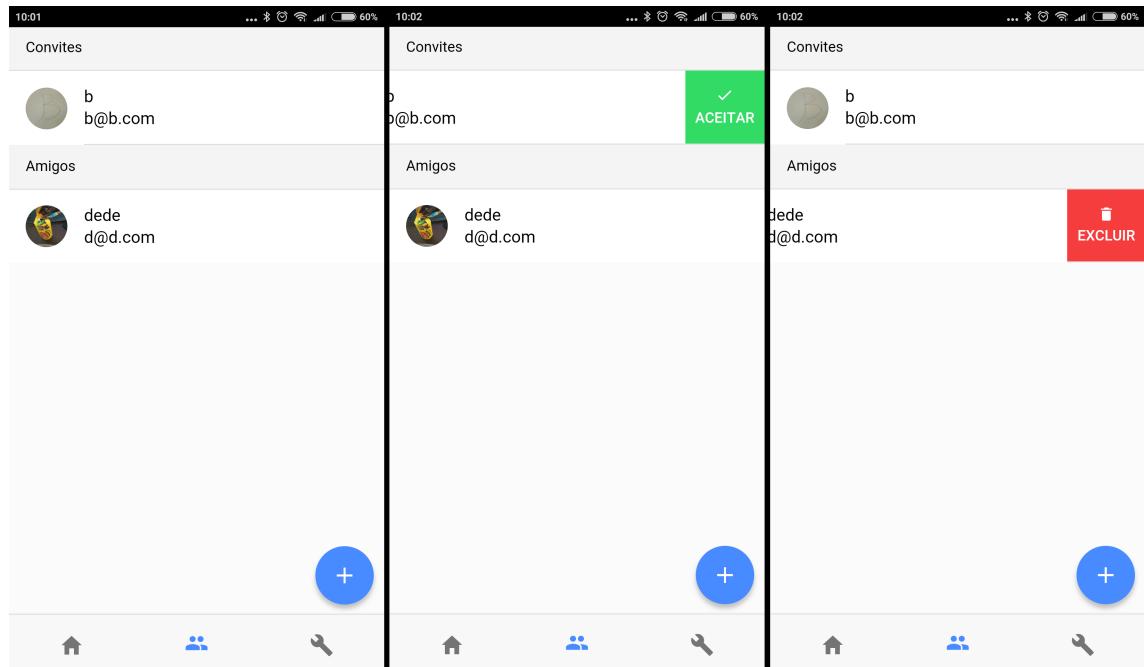


Figura 4.30: Tela de convites pendentes e amigos.

Configurações

Por fim, é mostrado na Figura 4.31 que na tela de Configurações, o usuário tem acesso apenas a troca de foto de perfil e nome de usuário e ao *logout* de conta.



Figura 4.31: Tela de configurações de conta.

Capítulo 5

Avaliação Final

Neste Capítulo estão documentadas as avaliações realizadas nesta monografia, sendo elas um teste real da aplicação e uma pesquisa de mercado.

5.1 Teste do aplicativo

Para testar se o princípio fundamental do Notifique-me funcionava de maneira correta - permitir que um indivíduo entre em uma sala estando próximo a ela - foi criada uma sala de leitura no Sebrae/Coxim com um raio de 200 metros (suficiente para englobar todo o local e suas proximidades). Este local foi escolhido pois foi sede de um evento real - o *Startup Weekend Coxim* - que ocorreu nos dias 09, 10 e 11 de Novembro de 2018. Na Figura 5.1 pode ser visto a sala criada visualizada em um mapa, representada pela maior circunferência, na cor preta.

Após a sala ser criada, o *download* do aplicativo foi disponibilizado para alguns indivíduos. Utilizando os dados salvos no *Firebase Authentication* e *Realtime Database* pôde-se analisar quantos usuários se cadastraram na aplicação e quantos entraram na sala criada. Na Figura 5.2 pode ser observado que houveram 10 contas criadas no *Firebase Authentication* no dia 10 de Novembro de 2018.

Analizando as contas criadas no *Firebase Authentication* e os usuários salvos na sala criada para o evento - registrados no *Realtime Database*, pôde-se concluir que 8 dos 10 usuários adentraram na sala criada (Figura 5.3). Desta forma, conclui-se que a funcionalidade primordial do aplicativo - mesmo com os erros de precisão comentados na Seção 4.6 - funcionou da forma esperada.

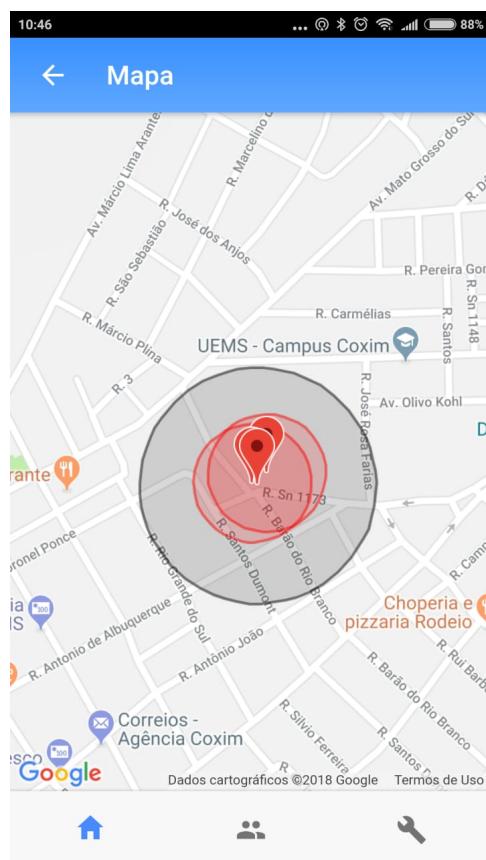


Figura 5.1: Sala criada no evento *Startup Weekend Coxim*.

Pesquise por endereço de e-mail, número de telefone ou UID do usuário				
Identificador	Provedores	Criado em	Conectado	UID do usuário
rafaeljg19@gmail.com	✉	10 de nov de ...	11 de nov de ...	fvwtSz8GsPSotp90m1Qsyh5roPl1
gabrielroches@outlook.com	✉	10 de nov de ...	17 de nov de ...	HzKTZsetA1NxVmAzmz2zE0gZjjm1
adilhurani@gmail.com	✉	10 de nov de ...	10 de nov de ...	qNf4l2nIGYY6Eg8y5Vsw9AziNqJ2
avencormento@outlook.com	✉	10 de nov de ...	10 de nov de ...	EcW9lIrAGzyWEjvyc8Y8raRTRO782
williampernambucapaulo@gmail.com	✉	10 de nov de ...	17 de nov de ...	V99wVhnXqheVtijXFCTAGYAToOF2
clintonlima@hotmail.com	✉	10 de nov de ...	12 de nov de ...	84b4Ju2SuAfOgUqxKvZp8njJFhy2
alec.alvarenga@gmail.com	✉	10 de nov de ...	13 de nov de ...	YCd3SEnv0uRyqSfchVBv3csHdg53
matheuslmc@gmail.com	✉	10 de nov de ...	10 de nov de ...	lsq8tUihkXYWyhixzv97f7STw1
maia_ma_braga@hotmail.com	✉	10 de nov de ...	10 de nov de ...	58X2AyJaRPd4ADl9gZe8JBrbzr63
marioesqueleto@gmail.com	✉	10 de nov de ...	10 de nov de ...	CCXH2vLLpjTp9LckD4ksmD5obqh2

Figura 5.2: Contas criadas no dia 10/11/2018.

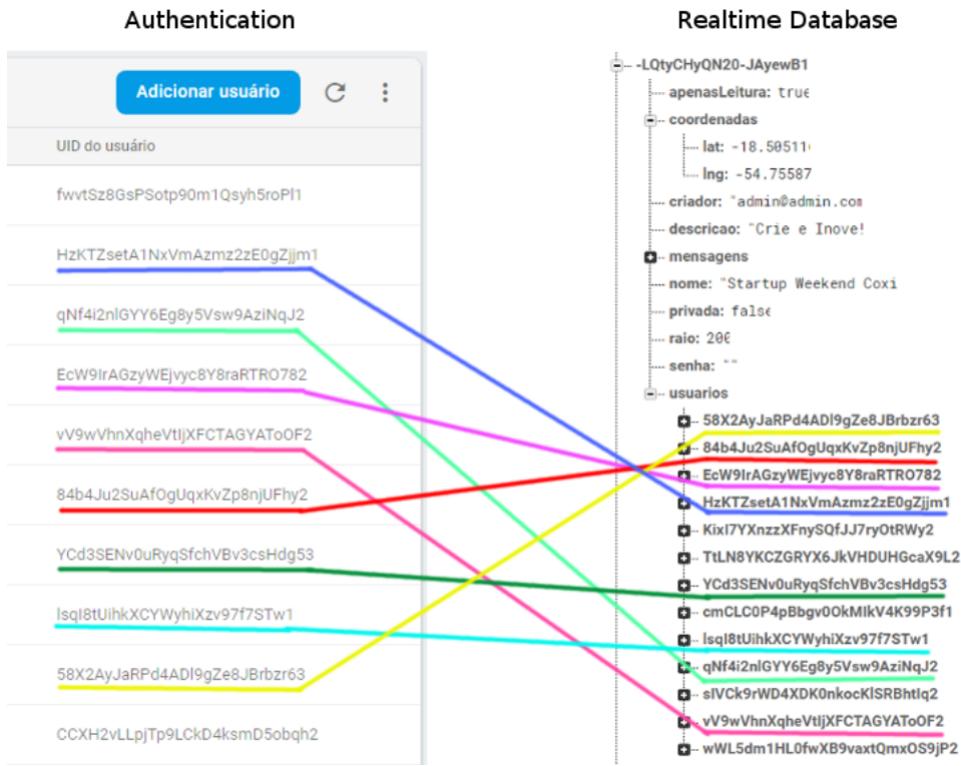


Figura 5.3: Contas criadas no dia 10/11/2018 - Visualizadas no Firebase *Authentication* e *Realtime Database*.

5.2 Validação

Segundo Maurya, o primeiro passo a se tomar após documentar uma hipótese, é validá-la sistematicamente [53]. No entanto, nesta monografia esta etapa apresentou alguns problemas. No começo deste projeto, a ideia proposta foi indiretamente validada, mas tendo sido apresentada a poucos participantes de um mesmo grupo social, entende-se que os resultados obtidos são irrelevantes e possuem pouca representatividade.

Assim, a proposta dada foi validada corretamente somente ao final de todo o processo, quando a própria ideia já havia sido implementada. Por conta disso, será visto no decorrer desta seção que alguns outros problemas reais da comunidade não foram solucionados nesta implementação, pois sua existência não era sabida no início deste trabalho. Ainda assim, mesmo tendo ciência do erro, decidiu-se registrar e documentar aqui os dados obtidos pois viu-se neles real relevância.

5.2.1 Lean Canvas

O *Business Model Canvas* ou simplesmente *Canvas*, é uma ferramenta de planejamento estratégico criada por Alex Osterwalder e Yves Pigneur, que permite planejar e visualizar

as principais funções de um negócio e suas relações [54]. O *Lean Canvas* é uma ferramenta criada por Ash Maurya com base no *Canvas* que tem maior ênfase em hipóteses que precisam ser validadas precocemente em uma *startup* [53]. Assim, tendo em vista que a ideia desta proposta tem grandes semelhanças com uma *startup*, o *Lean Canvas* foi o modelo escolhido para representá-la. Na Figura 5.4 é mostrado a representação da ideia proposta nesta monografia utilizando o modelo *Lean Canvas*.



Figura 5.4: Modelo *Lean Canvas* da ideia proposta.

5.2.2 Pesquisa de Aceitação

Para avaliar a aceitação do público quanto a ideia proposta, uma pesquisa de mercado foi realizada durante o mês de novembro de 2018. No público-alvo da aplicação - eventos corporativos, observou-se dois subgrupos distintos: participantes e organizadores, e por isso fez-se necessário questionar os dois “lados da moeda”.

Participantes de Eventos

Para questionar os participantes de eventos foi realizada uma pesquisa por meio da plataforma Formulários - *Google*. Esta pesquisa obteve 61 respostas de indivíduos de diferentes idades, as quais serão apresentadas e comentadas na sequência.

Na primeira parte do formulário foi perguntado aos entrevistados com que frequência participavam de eventos. Se a resposta obtida fosse: “Nunca participei”, o formulário dava-se por encerrado. Caso contrário, se o participante assinala-se uma das outras opções, as próximas questões eram liberadas. Para esta pergunta 8,2% responderam nunca ter participado de um evento, e 63,9% responderam ter participado de menos de 5 eventos

neste ano. A pergunta completa e o gráfico com as respostas obtidas a esta pergunta podem ser visualizados na Figura 5.5.

Com que frequência você participa de eventos?

61 respostas

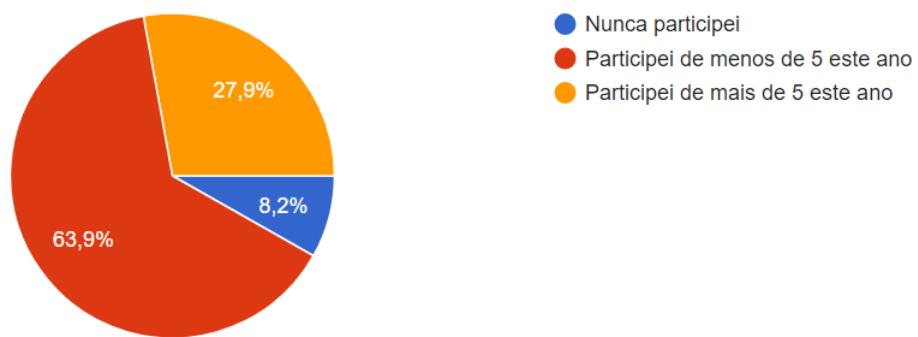


Figura 5.5: Frequência de participação em eventos.

Perguntou-se então quais problemas os participantes já tiveram que enfrentar ao frequentar eventos. Não sendo esta questão de múltipla escolha, o participante deveria descrever o(s) problema(s) enfrentado(s). Todavia, apenas 15 indivíduos responderam ter algum tipo de problema ao frequentar eventos, sendo que mais da metade (34) responderam não ter qualquer tipo de problema. Alguns dos problemas apontados foram:

- Falta de divulgação;
- Cancelamento e mudança de horários;
- Programação confusa;
- Comunicação com a organização;
- A localização correta da palestra de abertura do evento;
- Falta de informação sobre o que fazer em seguida.

Foi perguntado então aos entrevistados se durante os eventos que já participaram foi frequente o surgimento de dúvidas sobre a programação - como horários/locais de palestras e alterações na própria programação. Os entrevistados poderiam responder em uma escala de 1 a 10 com que frequência este problema acontecia, sendo 1 pouca ou nenhuma frequência e 10 muita frequência. Como pode ser observado na Figura 5.6, os resultados obtidos a esta pergunta foram mais平衡ados. Entende-se portanto que a maioria dos entrevistados passa, vez ou outra, pelo problema citado. Todavia, notou-se que a opção mais escolhida da escala foi o número 1 (23,2%), que indica o mais baixo nível de frequência. Sendo assim, entende-se que boa parte dos entrevistados não passam

pelo problema citado. A pergunta completa e o gráfico com as respostas obtidas a esta pergunta podem ser visualizados na Figura 5.6.

"Em eventos acadêmicos que já participei houveram situações em que eu não estava totalmente ciente da programação que acontecia no momento. Por desatenção, dúvidas com os horários/locais das palestras e informações de última hora (como alterações na programação) mostraram-se frequentes." Ao participar de eventos, com que frequência você passa por isso?

56 respostas

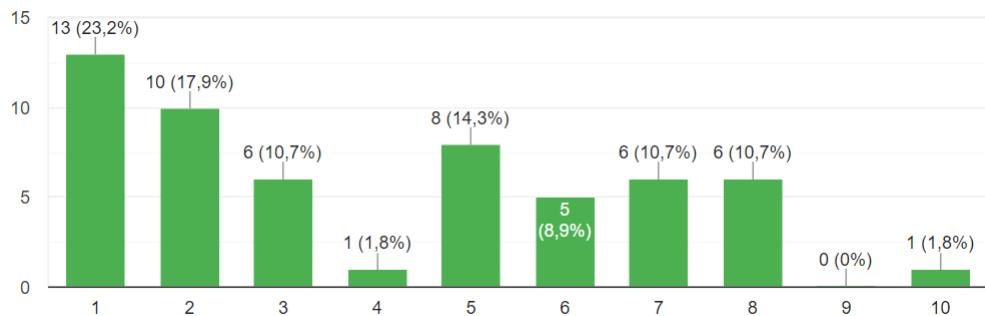


Figura 5.6: Frequência de dúvidas sobre a programação de um evento.

Na sequência, perguntou-se aos entrevistados qual seria a forma ideal para se receber e/ou consultar as informações de um evento, durante o seu acontecimento. Sendo esta questão de múltipla escolha, a opção mais escolhida foi "Através de notificações no celular" (50%). A pergunta completa e o gráfico com as respostas obtidas a esta pergunta podem ser visualizados na Figura 5.7.

Na sua opinião, qual seria a forma ideal para receber/consultar as informações de um evento, DURANTE o seu acontecimento?

56 respostas

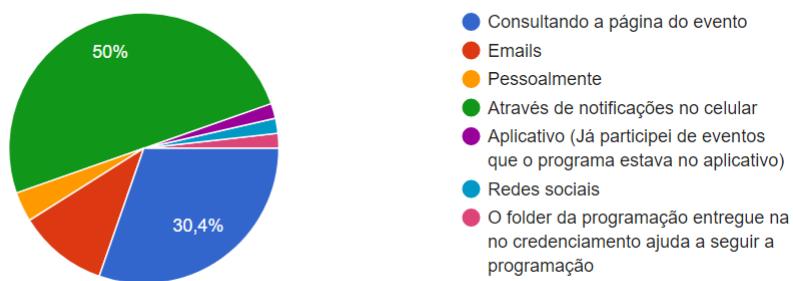


Figura 5.7: Forma ideal para se receber informações durante um evento.

Após questionar os entrevistados sobre o problema, foi oferecida uma solução e per-

guntado em uma escala de 1 a 10 se o participante achava a ideia apresentada útil, sendo 1 pouca ou nenhuma utilidade e 10 muita utilidade. A opção mais escolhida foi o número 10 (62,5%), representando portanto que a maioria dos indivíduos vê muita utilidade na ideia proposta. A pergunta completa e o gráfico com as respostas obtidas a esta pergunta podem ser visualizados na Figura 5.8.

A ideia aqui proposta é: criar um aplicativo que permita que indivíduos fisicamente próximos possam se comunicar, mesmo que não se conheçam. Este aplicativo poderia ser utilizado em eventos para centralizar e facilitar a comunicação entre a organização e os participantes e até mesmo entre os próprios participantes.

Você vê utilidade na ideia proposta e no exemplo citado?

56 respostas

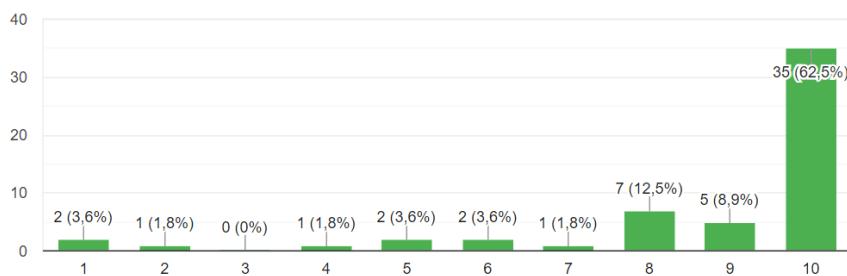


Figura 5.8: Aprovação da ideia proposta em escala.

Perguntou-se então aos entrevistados se estes utilizariam o aplicativo proposto em eventos futuros. A esta pergunta 92,9% das respostas foram positivas. A pergunta completa e o gráfico com as respostas obtidas a esta pergunta podem ser visualizados na Figura 5.9.

Você utilizaria este aplicativo nos próximos eventos que participasse?

56 respostas

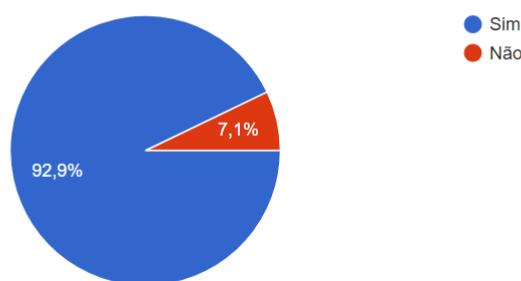


Figura 5.9: Aprovação da ideia proposta.

Como última pergunta, disponível apenas aos indivíduos que responderam “Sim” na questão anterior, foi perguntado que funcionalidade os entrevistados gostariam de possuir neste aplicativo. Algumas das respostas obtidas foram:

- Resposta claras e objetivas sobre o evento, data hora local, regras e deveres para participantes e futuros eventos que sejam relacionados ao que estou participando;
- Caso valesse horas tivesse uma forma direta para computar as horas na grade do aluno;
- Lista de amigos pós evento para manter contato;
- Manuseio simples e pontual;
- Tópicos e informações relacionadas aos debate ou projetos;
- Alertas sobre mudanças de horários, cancelamento de cursos/palestras, disponibilidade de vagas;
- Horário, síntese do que irá conter no evento, local (pátio, anfiteatro, local fora da faculdade);
- Transferência de arquivos;
- Programação atualizada mapa/guia do ambiente (caso seja um espaço muito grande);
- Informações em tempo real sobre o evento;
- Maior comunicação entre os organizadores e os participantes;
- Informações sobre o conteúdo sobre os palestrantes, acessibilidade;

Organizadores de Eventos

Após obter um resultado parcial da pesquisa feita aos participantes, foi realizada uma entrevista com duas organizadoras de eventos. A primeira entrevistada foi Luzicarla Souza Softov, Gerente do Sebrae - Coxim/MS. As perguntas feitas bem como os resultados obtidos serão descritos e comentados na sequência.

1. Alguns eventos que teve a oportunidade de participar como organizadora.
Startup Weekend, Empretec, Rodada de negócios e cursos no geral.
2. Nos eventos que participou como organizadora, quais meios foram utilizados para estabelecer a comunicação entre organização/participantes?

A esta pergunta a entrevistada respondeu que é comum a utilização de grupos no aplicativo *WhatsApp*, mas que pessoalmente é a forma de comunicação mais utilizada (sendo para ela a ideal) pois a maioria dos eventos citados são de pequeno porte. Ela ainda comentou que em um dos eventos de médio-grande porte que participou da organização, foi utilizado um aplicativo para tentar facilitar a comunicação entre os participantes, mas que este apresentou alguns problemas e falhas.

3. Perguntou-se então se já ocorreu algum problema de última hora (como alterações em locais de palestras e cancelamentos) que fez-se necessário uma comunicação rápida com os participantes, e como este problema foi resolvido.

A entrevistada respondeu que o tipo de problema citado já ocorreu, e que normalmente a comunicação pessoal, juntamente com o auxílio de mensageiros instantâneos e redes sociais, sempre foi o suficiente pra resolver a situação.

4. Antes de realizar esta pergunta, foi apresentado a entrevistada os dados obtidos com a primeira pesquisa, ou seja, foi mostrado que alguns indivíduos já enfrentaram problemas de comunicação ao participar de eventos, e que alguns gostariam de receber as informações do evento por notificações no celular. Foi então apresentado a ideia proposta: um aplicativo em que pessoas fisicamente próximas, mesmo que sendo desconhecidas, podem interagir e se comunicar. Perguntou-se a ela então, se esta ideia era viável para utilizar em próximos eventos que organizasse.

A esta pergunta a entrevistada explanou que não considera viável a utilização do aplicativo, uma vez que, como comentado anteriormente, a maioria dos eventos organizados por ela são de pequeno porte, sendo desnecessário a utilização de um aplicativo para suprir a demanda de comunicação.

A segunda entrevistada foi Ana Lina Farias, responsável pelo planejamento e marketing no *Living Lab - Campo Grande/MS*. As perguntas feitas bem como os resultados obtidos serão descritos e comentados na sequência.

1. Alguns eventos que teve a oportunidade de participar como organizadora.
Startup Day, Feira do Empreendedor, eventos de *talks* e oficinas.
2. Nos eventos que participou como organizadora, quais meios foram utilizados para estabelecer a comunicação entre organização/participantes?
A esta pergunta a entrevistada respondeu que pelos eventos citados serem de médio/grande porte, utiliza assiduamente e-mails, grupos no aplicativo *WhatsApp* e publicações nas redes sociais *Facebook* e *Instagram*.
3. Perguntou-se então se já ocorreu algum problema de última hora (como alterações em locais de palestras e cancelamentos) que fez-se necessário uma comunicação rápida com os participantes, e como este problema foi resolvido.

A entrevistada respondeu que este tipo de problema é bastante recorrente, e que normalmente lida com este problema através de e-mails e redes sociais, mas que já houveram ocasiões em que teve que recorrer a comunicação pessoal.

4. Antes de realizar esta pergunta, foi apresentado a entrevistada os dados obtidos com a primeira pesquisa, ou seja, foi mostrado que alguns indivíduos já enfrentaram problemas de comunicação ao participar de eventos, e que alguns gostariam de receber as informações do evento por notificações no celular. Foi então apresentado a ideia proposta: um aplicativo em que pessoas fisicamente próximas, mesmo que

sendo desconhecidas, podem interagir e se comunicar. Perguntou-se a ela então, se esta ideia era viável para utilizar em próximos eventos que organizasse.

A entrevistada respondeu que considera a proposta útil e viável principalmente em eventos de grande porte.

5. Visto o interesse da entrevistada na ideia proposta, perguntou-se o que ela gostaria de ter como funcionalidade no aplicativo.

A entrevistada respondeu que gostaria que fosse disponibilizado à organização principalmente métricas e relatórios, pois com isso poderiam calibrar e melhorar os futuros eventos de acordo com o gosto dos participantes. Citou que os usuários poderiam ter perfis, que registram além de informações pessoais, seus interesses, eventos que já participaram, e o que gostariam de ver em eventos futuros. Além disso seria importante o aplicativo possibilitar uma forma de manter a comunicação pós-evento.

5.2.3 Análise da Pesquisa

Mesmo a pesquisa tendo poucos entrevistados (63 no total), algumas conclusões podem ser tiradas a partir dos dados obtidos. Os entrevistados - em sua maioria - afirmaram que vez ou outra enfrentam problemas de comunicação com a organização de eventos, e que estariam dispostos a utilizar um aplicativo que resolvesse este problema nos próximos eventos que participassem. No entanto, é interessante ressaltar a quantidade de respostas obtidas quando perguntado que funcionalidades seriam úteis no aplicativo proposto. Esta questão não era de caráter obrigatório, e ainda assim obteve 48 respostas. Ou seja, aproximadamente 77% dos participantes entrevistados (incluindo uma organizadora de eventos) descreveram alguma funcionalidade que gostariam de possuir. Assim, entende-se que este é um nicho real, que pode ser mais bem explorado e melhor atendido.

Capítulo 6

Conclusão

Aplicativos que utilizam a geolocalização não são nenhuma novidade. O *Tinder* por exemplo, desde 2012 permite a comunicação de indivíduos fisicamente próximos, a fim de estabelecer algum tipo de relação amorosa [58]. No entanto, o conceito de interação por localidade não é exclusivo a somente esta gama de aplicativos, e tem sido explorado por diversas áreas. Tantothai por exemplo, propôs uma plataforma de *chat* local para auxiliar indivíduos na procura por locais próximos, como restaurantes e acomodações [61].

Utilizando o mesmo conceito citado anteriormente, e a fim de beneficiar a comunidade participante de eventos corporativos, neste trabalho foi desenvolvido um aplicativo *mobile* capaz de permitir que indivíduos fisicamente próximos possam se comunicar, ainda que sejam desconhecidos entre si. A *framework Firebase* foi utilizada assiduamente para suprir algumas demandas do aplicativo: o módulo *Authentication* foi usado para gerenciar a criação e *login* de usuários; o banco de dados *Cloud Storage* foi utilizado para armazenar fotografias; e por fim, o banco de dados *Realtime Database* foi empregado principalmente para a criação do bate-papo bidirecional multiusuário. O aplicativo foi desenvolvido utilizando a *framework Ionic*, e por isto tem caráter multiplataforma. Assim, mesmo tendo sido executado e testado apenas na plataforma *Android*, o mesmo código pode ser utilizado para criar a aplicação na plataforma *iOS*, bastando apenas algumas configurações adicionais.

Após ser implementada, a ideia proposta pelo aplicativo passou por uma pesquisa de mercado, que mesmo sendo realizada tarde, obteve resultados relevantes. Foi concluído por meio da pesquisa que o público despertou curiosidade e interesse para com a ideia apresentada, mas que gostaria de mais funcionalidades específicas para os eventos que participam. Dessa forma, comprehende-se que o aplicativo aqui desenvolvido atingiu os objetivos propostos no início desta monografia, mas pode ser aprimorado a fim de abranger um maior conjunto de problemas, descritos pela própria comunidade. Além disso, a demora para executar a validação de forma correta ainda permitiu documentar os tipos de dificuldades e reviravoltas que uma validação tardia pode ocasionar, sendo válida como aprendizado e experiência para trabalhos futuros.

6.1 Contribuições deste Trabalho

- Utilizando o *Firebase*, foi encontrado um meio para criar salas de bate-papo bidimensionais multiusuário.
- Foi desenvolvido um aplicativo *mobile* multiplataforma capaz de permitir que indivíduos fisicamente próximos possam se comunicar, sem a necessidade de vínculo direto.
- Foi realizada uma pesquisa de mercado com dados não utilizados diretamente na implementação do aplicativo aqui proposto, que podem ser relevantes para um possível trabalho futuro.

6.2 Dificuldades Encontradas

Sendo um problema recorrente na comunidade, o principal problema encontrado durante o desenvolvimento deste trabalho, foi a dificuldade para se obter uma localização precisa, uma vez que ela é fundamental para o bom funcionamento do aplicativo. Entende-se que o problema não está nas funções disponibilizadas pelo *Ionic*, pois elas apenas utilizam as funções nativas do próprio *JavaScript* [14]. Além disso, este problema também não ocorre simplesmente por estar sendo executado em uma plataforma móvel, pois testou-se as mesmas funções em navegadores para computadores e os mesmos erros se mostraram aparentes. Uma técnica para tentar melhorar a precisão foi utilizada: utilizou-se a localização com a melhor acurácia, mas notou-se que a própria acurácia vez ou outra apresenta imprecisão.

Outro problema deste trabalho foi a validação realizada, ou mais especificamente sua execução tardia. Uma validação inicial foi feita indiretamente, mas por conter poucos participantes de um mesmo grupo social, os resultados obtidos foram irrelevantes. Assim, a validação ocorreu de forma correta somente ao final de todo o processo, quando o aplicativo proposto já havia sido desenvolvido e por isso, não houve tempo hábil para se implementar as sugestões dadas como *feedback*. Contudo, mesmo a validação final tendo sido executada de maneira mais correta, ainda vê-se nela falhas que podem ser melhoradas. Tendo a pesquisa final sido divulgada em redes sociais pessoais, boa parte do público alcançado provavelmente ainda pertence a um mesmo grupo social, compartilhando por exemplo dos mesmos gostos, o que resultará em possíveis respostas iguais e direcionadas.

6.3 Trabalhos Futuros

Por meio de uma pesquisa de mercado, mostrou-se anteriormente que a ideia proposta neste trabalho despertou curiosidade e interesse dos entrevistados. Contudo, na mesma pesquisa viu-se que participantes e organizadores de eventos gostariam de funcionalidades que também poderiam ser implementadas no aplicativo desenvolvido. Assim, como trabalho futuro, pode-se:

1. Realizar uma nova pesquisa de mercado a fim de comprovar que este é um real nicho a ser atingido;
2. Corrigir/melhorar os erros na precisão da localização. Para isso sugere-se atualizar a *framework Ionic* e a estrutura *Cordova* para suas respectivas versões mais recente, a fim de checar se o problema com as funções foi resolvido, ou se uma outra abordagem foi implementada.
3. Criar um aplicativo direcionado para o público atuante em eventos, com mais funcionalidades específicas para resolver os problemas deste nicho.

Assim, a ideia base desta proposta - de unir indivíduos fisicamente próximos - é somente o início, apenas uma forma diferente de se interagir com outros usuários. O aplicativo deste momento em diante pode se comportar como uma rede social, fornecendo perfis de usuários, interesses, bate-papo, eventos futuros e etc. Com isso, aos organizadores pode ser oferecido relatórios de uso: quantos participantes estiveram em determinado evento, de quais palestras/cursos participaram, o que gostariam de ver no futuro e assim por diante.

Referências Bibliográficas

- [1] All about ionic. <https://ionicframework.com/docs/v1/guide/preface.html>. Accessed: 2018-06-13.
- [2] Architecture overview. <https://angular.io/guide/architecture>. Accessed: 2018-11-25.
- [3] Banco de dados de chave-valor. <https://aws.amazon.com/pt/nosql/key-value/>. Accessed: 2018-10-30.
- [4] Cloud storage. <https://firebase.google.com/docs/storage/>. Accessed: 2018-11-06.
- [5] Deploy: push app updates in real-time. <https://ionicframework.com/docs/pro/deploy/>. Accessed: 2018-06-13.
- [6] Deploying to a device. <https://ionicframework.com/docs/intro/deploying/>. Accessed: 2018-11-12.
- [7] DIGITAL IN 2018: WORLD'S INTERNET USERS PASS THE 4 BILLION MARK kernel description. <https://wearesocial.com/blog/2018/01/global-digital-report-2018>. Accessed: 2018-05-03.
- [8] Distance on a sphere: The haversine formula. <https://community.esri.com/groups/coordinate-reference-systems/blog/2017/10/05/haversine-formula>. Accessed: 2018-11-07.
- [9] Firebase authentication. <https://firebase.google.com/docs/auth/>. Accessed: 2018-10-31.
- [10] The firebase blog. <https://firebase.googleblog.com/>. Accessed: 2018-10-31.
- [11] Firebase realtime database. <https://firebase.google.com/docs/database/>. Accessed: 2018-11-01.
- [12] French Space Agency claims 100 million users for Galileo navigation system kernel description. <https://www.geospatialworld.net/news/europe-claims-100-million-users-galileo-navigation-system/>. Accessed: 2018-05-17.

- [13] Geofence. <https://ionicframework.com/docs/native/geofence/>. Accessed: 2018-06-25.
- [14] Geolocation. <https://ionicframework.com/docs/native/geolocation/>. Accessed: 2018-06-25.
- [15] Have you met the realtime database? <https://firebase.googleblog.com/2016/07/have-you-met-realtime-database.html>. Accessed: 2018-11-01.
- [16] Haversine formula. https://rosettacode.org/wiki/Haversine_formula. Accessed: 2018-11-07.
- [17] If firebase doesn't use http requests, what does it use? <https://www.quora.com/If-Firebase-doesnt-use-HTTP-requests-what-does-it-use>. Accessed: 2018-11-02.
- [18] Introduction to components. <https://angular.io/guide/architecture-components>. Accessed: 2018-11-25.
- [19] Introduction to services. <https://angular.io/guide/architecture-services>. Accessed: 2018-11-25.
- [20] Ionic native. <https://ionicframework.com/docs/native/>. Accessed: 2018-06-25.
- [21] Modelo de objeto de documento (dom). https://developer.mozilla.org/pt-BR/docs/DOM/Referencia_do_DOM. Accessed: 2018-11-25.
- [22] Overview. <https://cordova.apache.org/docs/en/latest/guide/overview/>. Accessed: 2018-06-14.
- [23] A real-time database survey: The architecture of meteor, rethinkdb, parse and firebase. <https://goo.gl/hYJK97>. Accessed: 2018-11-02.
- [24] Using firebase to provide real-time notifications. <https://medium.com/swlh/using-firebase-to-provide-real-time-notifications-2f86b2f7d499>. Accessed: 2018-11-02.
- [25] Welcome to ionic pro. <https://ionicframework.com/docs/pro/>. Accessed: 2018-06-13.
- [26] What is angular. <https://developer.telerik.com/topics/web-development/what-is-angular/>. Accessed: 2018-11-25.
- [27] What is ascii. <https://kb.iu.edu/d/afht>. Accessed: 2018-11-02.
- [28] Whatsapp: Criando grupos e convidando participantes. https://faq.whatsapp.com/pt_br/android/26000123/?category=5245251. Accessed: 2018-11-8.
- [29] The WebSocket Protocol. RFC 6455, 2011.

- [30] RM Alkan, H Karaman, e M Sahin. Gps, galileo and glonass satellite navigation systems & gps modernization. In *Recent Advances in Space Technologies, 2005. RAST 2005. Proceedings of 2nd International Conference on*, páginas 390–394. IEEE, 2005.
- [31] Aristóteles. *Política*.
- [32] Eduardo Bezerra. *Princípios de Análise e Projeto de Sistema com UML*, volume 3. Elsevier Brasil, 2015.
- [33] Dan Bouhnik e Mor Deshen. Whatsapp goes to school: Mobile instant messaging between teachers and students. *Journal of Information Technology Education: Research*, 13(1):217–231, 2014.
- [34] Karla Schuch Brunet e Juan Freire. Cultura digital e geolocalização: a arte ante o contexto técnico-político. *Ene cult, Salvador*, 1(6):1–14, 2010.
- [35] Ricardo Manuel Fonseca Cardoso. *Bases de Dados NoSQL*. Tese de Doutoramento, 2012.
- [36] Zhi Chen et al. Html5 hybrid mobile application: Building mobile applications using web technologies with ionic. 2018.
- [37] David Wai Kee Chu, Kwan Keung Ng, Ivan KW Lai, e Paul Wai Ming Lam. Analysis of student behaviors in using wechat/whatsapp for language learning at diploma level in hong kong: A pilot test. In *Educational Technology (ISET), 2015 International Symposium on*, páginas 104–108. IEEE, 2015.
- [38] Edgar F Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.
- [39] Alex Ferreira Damasceno. A interação entre estranhos no omegle. com: sociabilidade, relacionamento e identidade. In *E-Compós*, volume 16. 2013.
- [40] Daniel Pedrinha Georgii Daniel Schmitz. *Angular 2 na prática*. 2016.
- [41] Edilson Alves de Carvalho e Paulo César de Araújo. Localização: coordenadas geográficas. 2008.
- [42] Emílio Dias. *Desmistificando Rest*. algaworks, 2016.
- [43] Ramez Elmasri, Shamkant B Navathe, Marília Guimarães Pinheiro, et al. Sistemas de banco de dados. 2005.
- [44] Mariela Inés Cortés Enyo José Tavares Gonçalves. *Computação Computação Análise e Projeto de Sistemas*. EdUECE, 2015.
- [45] Yakov Fain e Anton Moiseev. *Angular 2 Development with TypeScript*. Manning Publications Co., 2016.
- [46] Luiz Carlos Lobato Glêdson Elias. *Arquitetura e Protocolos de Rede TCP-IP*. 2013.

- [47] Chris Griffith. *Mobile App Development with Ionic, Revised Edition: Cross-Platform Apps with Ionic, Angular, and Cordova.* "O'Reilly Media, Inc.", 2017.
- [48] Carlos Alberto Heuser. *Projeto de banco de dados: Volume 4 da Série Livros didáticos informática UFRGS.* Bookman Editora, 2009.
- [49] Anthony T Holdener. *HTML5 geolocation.* "O'Reilly Media, Inc.", 2011.
- [50] Ionic. Hibrid vs native. 2018.
- [51] Ramon Santos Lummertz e Antoni Sganzerla. Direto ao ponto—app colaborativo do transporte coletivo usando o firebase. *Conversas Interdisciplinares*, 14(1), 2018.
- [52] Beatriz Rezener Dourado Matos e João Gabriel de Britto. Estudo comparativo entre o desenvolvimento de aplicativos móveis utilizando plataformas nativas e multiplataformas, 2017.
- [53] Ash Maurya. *Running lean: iterate from plan A to a plan that works.* "O'Reilly Media, Inc.", 2012.
- [54] Alexander Osterwalder e Yves Pigneur. Business model canvas. *Self published. Last*, 2010.
- [55] Ezequiel Douglas Prezotto e Bruno Batista Boniati. Estudo de frameworks multiplataforma para desenvolvimento de aplicações mobile híbridas, 2017.
- [56] Pramod J Sadalage e Martin Fowler. *NoSQL Essencial: Um guia conciso para o mundo emergente da persistência poliglota.* Novatec Editora, 2013.
- [57] LLB SILVA, Daniel Facciolo Pires, e Silvio Carvalho Neto. Desenvolvimento de aplicações para dispositivos móveis: tipos e exemplo de aplicação na plataforma ios. *Franca/SP*, 2015.
- [58] Thálita Teles Silva. O amor em suas mãos: Um estudo sobre a sociabilidade entre os indivíduos no aplicativo tinder. *Panorama*, 6(2):99–102, 2016.
- [59] Osvaldo Kotaro Takai, Isabel Cristina Italiano, e João Eduardo Ferreira. Introdução a banco de dados. *Departamento de Ciências da Computação. Instituto de Matemática e Estatística. Universidade de São Paulo. São Paulo*, 2005.
- [60] Andrew S Tanenbaum e David Wetherall. *Computer networks.* Prentice hall, 1996.
- [61] Pawin Tantothai, Chanatan Srisittimongkol, Wachirawit Rukijkanpanich, e Suppawong Tuarob. mipmap: A mobile application for proximate social network communication. In *Student Project Conference (ICT-ISPC), 2017 6th ICT International*, páginas 1–4. IEEE, 2017.
- [62] Renato Molina Toth. Abordagem nosql—uma real alternativa. *Sorocaba, São Paulo, Brasil: Abril*, 13, 2011.

- [63] T Varela e L Stanley. Implementação e análise da utilização de websockets em sistemas computacionais. *Universidade Luterana do Brasil (ULBRA), Rio Grande do Sul*, 2012.
- [64] Marcos Rodrigues Vieira, Josiel Maimone de FIGUEIREDO, Gustavo Liberatti, e Alvaro Fellipe Mendes Viebrantz. Bancos de dados nosql: conceitos, ferramentas, linguagens e estudos de casos no contexto de big data. *Simpósio Brasileiro de Bancos de Dados*, 2012.