UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

TAGARELA: INTEGRAÇÃO E MELHORIAS NO APLICATIVO DE REDE DE COMUNICAÇÃO ALTERNATIVA

ANDRÉ FILIPE WIPPEL

BLUMENAU
2015

2015/02

ANDRÉ FILIPE WIPPEL

TAGARELA: INTEGRAÇÃO E MELHORIAS NO APLICATIVO DE REDE DE COMUNICAÇÃO ALTERNATIVA

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Ciência da Computação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Prof. Dalton Solano dos Reis, Mestre - Orientador

BLUMENAU
2015

2015/02

TAGARELA: INTEGRAÇÃO E MELHORIAS NO APLICATIVO DE REDE DE COMUNICAÇÃO ALTERNATIVA

Por

ANDRÉ FILIPE WIPPEL

Trabalho de Conclusão de Curso aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II pela banca examinadora formada por:

	_____
Presidente:	Prof. Dalton Solano dos Reis, M. Sc. – Orientador, FURB
	_____
Membro:	Prof. Alexander Roberto Valdameri, M. Sc. – FURB
	_____
Membro:	Prof. Aurélio Faustino Hoppe, M. Sc. – FURB

Blumenau, 07 de dezembro de 2015

Dedico este trabalho à família e a todos os meus amigos, especialmente meus pais e meu irmão que estiveram ao meu lado me apoiando sempre que necessário.

AGRADECIMENTOS

A meus pais e meu irmão pelo apoio e incentivo a mim dispostos.

A toda minha família que me apoiou muito ao longo destes anos.

Ao meu orientador Dalton Solano dos Reis pela confiança, dedicação e apoio a mim dispostos ao longo do desenvolvimento deste trabalho.

A todos os meus amigos e colegas de turma que trilharam este caminho comigo e me apoiaram constantemente.

A todos os professores da FURB que me concederam direta ou indiretamente conhecimento e capacidade para que eu pudesse completar este percurso de minha vida.

Só se pode alcançar um grande êxito quando nos mantemos fiéis a nós mesmos.

Friedrich Nietzsche

RESUMO

Este trabalho apresenta o desenvolvimento de uma aplicação multiplataforma que envolve a extensão de uma aplicação de comunicação alternativa. A aplicação estendida foi implementada através da ferramenta PhoneGap, que utiliza recursos de desenvolvimento *web* para disponibilizar a aplicação nos dispositivos móveis desejados. Além da plataforma *mobile*, a aplicação também foi disponibilizada na plataforma *web*. Para a implementação foram utilizadas as linguagens Javascript e *Hypertext Preprocessor* (PHP), em conjunto com o *framework* de estilos Materialize. Utilizou-se também diversos *plugins* que o PhoneGap disponibiliza. Para o armazenamento local foi utilizado WebSQL, que é uma das tecnologias sugeridas pelo PhoneGap para realizar este tipo de persistência de dados, porém no decorrer da implementação notaram-se certas limitações e foi identificado que esta tecnologia não é mais suportada pela W3C. De maneira geral, a aplicação de comunicação alternativa desenvolvida mostrou-se apta à medida que permite a criação e reutilização de pranchas de comunicação, bem como a interação dos usuários com estas pranchas. A aplicação também permite o cadastro do usuário com perfil de tutor ou paciente, conforme a necessidade do mesmo. Na análise comparativa de interface gráfica as versões testadas mostraram-se equivalentes e na análise de desempenho a aplicação também obteve bons resultados, apesar de possuir alguns pontos de atenção no consumo de memória do *plugin media* do PhoneGap.

Palavras-chave: PhoneGap. Comunicação alternativa. Multiplataforma. *Plugins*.

ABSTRACT

This work presents the development of a multiplatform application that involves the extension of an alternative communication app. The extended application was implemented through the PhoneGap tool, which uses web development resources to provide the application on the desired mobile devices. In addition to the mobile platform, the application is also available on the web platform. For the implementation, the Javascript and PHP languages have been used, in conjunction with the Materialize framework. It was also used a set of plugins that PhoneGap offers. For local storage was used WebSQL, which is a technology suggested by PhoneGap to perform this type of data persistence, but during the implementation were noted certain limitations and was identified that this technology is not supported anymore by the W3C. In general, the alternative communication application developed proved to be suitable as it allows for users to create and reuse communication boards as well as interact with these boards. The application also allows the user to register tutor or patient profiles, according to the need of the user. In the comparative analysis of GUI, the versions tested were shown to be equivalent and in the performance analysis the application also performed well, despite having some points of attention in memory consumption with the media plugin of PhoneGap.

Key-words: PhoneGap. Alternative communication. Multiplatform. Plugins.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AJAX – *Asynchronous Javascript and XML*

API – *Application Programming Interface*

CA – Comunicação Alternativa

CLI – *Command Line Interface*

CSS – *Cascading Style Sheets*

GUI – *Graphical User Interface*

HTML – *HyperText Markup Language*

JSON – *JavaScript Object Notation*

MER – Modelo de Entidade Relacionamento

OS – *Operacional System*

PCS – *Picture Communication Symbols*

PHP – *Hypertext Preprocessor*

RF – Requisito Funcional

RNF – Requisito Não-Funcional

SDK – *Software Development Kit*

SQL – *Structured Query Language*

UC – *Use Case*

UML – *Unified Modeling Language*

URI – *Uniform Resource Identifier*

URL – *Uniform Resource Locator*

W3C – *World Wide Web Consortium*

SUMÁRIO

1 INTRODUÇÃO

A comunicação é um mecanismo presente entre os humanos desde a época em que viviam em cavernas e sabe-se hoje em dia que existem diversas formas de se comunicar, dentre elas a comunicação oral é sem dúvida a mais utilizada (FOTON, 2008). Porém, muitas pessoas, devido a fatores físicos ou até mesmo psicológicos, não conseguem se expressar através da fala.

Desde quando crianças a fala já é algo muito importante, pois é ela que humaniza a criança, que a coloca como ser humano na sociedade atual, e que nomeia e define tudo que está ao seu redor (TEIXEIRA, 2013). Portanto, crianças que se mostram incapazes de se comunicar têm grandes chances de se sentirem excluídas socialmente.

Atualmente já existem diversas ferramentas e aplicações comerciais que servem como um auxílio para estas pessoas com deficiência ou dificuldade na fala, definidas como ferramentas de CA (Comunicação Alternativa). As ferramentas CAs, na sua maioria, utilizam o conceito de Pranchas de Comunicação, que são um conjunto de símbolos no qual o usuário utiliza para se comunicar com maior clareza e facilidade. Porém, nem todas estas ferramentas são de fácil uso, ou mesmo que sejam, muitas vezes se faz necessário o acompanhamento de um responsável para o auxilio na utilização dependendo do estado clínico do usuário.

Desta forma, é possível identificar que existe um grande avanço ainda a ser feito em relação às ferramentas de CA. Foi visto que o aplicativo Tagarela (TAGARELA, 2014), desenvolvido em um projeto acadêmico pela Universidade Regional de Blumenau, possui várias funcionalidades que auxiliam na comunicação de pessoas com limitações fonoarticulosas, mas procura principalmente propiciar um ambiente que permita armazenar e compartilhar as experiências relacionadas ao desenvolvimento do usuário. Estas experiências são de grande valia para os profissionais que auxiliam o usuário, seja o tutor, ou mesmo o fonoaudiólogo (especialista).

Assim como outras ferramentas de CA, o aplicativo Tagarela fornece uma alternativa e auxilia na insuficiência da fala de pessoas com alguma dificuldade ou deficiência. No entanto, é importante observar que aplicações deste tipo possuam uma interface acessível e que sejam simples de usar, para que os usuários se motivem e gradativamente possam adquirir a autonomia na sua utilização.

Diante do exposto, este trabalho apresenta o desenvolvimento de uma série de melhorias no projeto Tagarela, onde é fornecida uma interface mais acessível para os usuários, além de disponibilizar o acesso às funcionalidades de maneira distinta para cada

papel de usuário presente na aplicação (Tutor e Paciente). Todas as funcionalidades desenvolvidas, inclusive as já existentes, foram disponibilizadas nas versões atuais do aplicativo. Para isto foi utilizado o *framework* de desenvolvimento móvel PhoneGap.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é aprimorar o "Tagarela: Aplicativo de Comunicação Alternativa na Plataforma Android" (MARCO, 2014), integrando o desenvolvimento entre as versões Android, iOS e *web*.

Os objetivos específicos do trabalho são:

a) disponibilizar o acesso ao aplicativo através de perfis distintos (Tutor e Paciente);
b) disponibilizar uma interface mais acessível utilizando o conceito de pranchas de comunicação;
c) integrar o desenvolvimento das versões do Tagarela em um único *framework*.

1.2 ESTRUTURA

Este trabalho está estruturado em quatro capítulos, sendo que no primeiro é apresentada a introdução ao tema, bem como os objetivos e a estrutura deste trabalho.

O segundo capítulo aborda a fundamentação teórica necessária para o melhor entendimento deste trabalho.

O capítulo três contempla as etapas de desenvolvimento do aplicativo, onde são apresentados os requisitos, os diagramas para melhor entendimento, a implementação, demonstrando alguns quadros com código fonte, os resultados e discussões.

Por fim, o capítulo quatro trata das conclusões obtidas do presente trabalho e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

A seção 2.1 trata sobre Comunicação Alternativa e o conceito de Pranchas de Comunicação. A seção 2.2 descreve o projeto acadêmico Tagarela. A seção 2.3 apresenta a plataforma de desenvolvimento móvel PhoneGap em conjunto com o *framework* Materialize. Por fim, na seção 2.4, são apresentados os trabalhos correlatos.

2.1 COMUNICAÇÃO ALTERNATIVA E PRANCHAS DE COMUNICAÇÃO

Comunicação Alternativa (CA) é a área da tecnologia assistiva que se destina a ampliação das habilidades de comunicação de pessoas sem fala ou sem escrita funcional. Cada indivíduo fará uso dos recursos de comunicação com características distintas, que por sua vez devem atender suas necessidades específicas (SARTORETTO; BERSCH, 2014).

Atualmente os sistemas de CA são usados na área clínica para compensar temporária ou permanentemente as dificuldades de indivíduos com desordens de expressão. Antes de fazer uso funcional de qualquer ferramenta assistiva, os profissionais e especialistas da área devem tomar como preocupação a diversidade de aspectos envolvidos em cada usuário, para então indicar um recurso que otimize o uso dos sistemas de CA (GONÇALVES, 2008).

Dentro da CA um recurso, é o objeto ou ferramenta que será utilizado para transmitir as mensagens. Um dos mais utilizados atualmente são as pranchas de comunicação, que são aglomerados de símbolos, letras, sílabas, palavras, frases ou números. Cada prancha deve ser personalizada com seus símbolos de acordo com as possibilidades cognitivas, visuais e motoras de seu usuário (PELOSI, 2011).

Cada conjunto de símbolos está ligado diretamente a um sistema de símbolos gráficos, que são uma coleção de imagens que apresentam características comuns entre si e foram criadas para atender a diferentes necessidades de comunicação dos usuários. Existem diferentes sistemas simbólicos, sendo que o mais utilizado em todo o mundo é o *Picture Communication Symbols* (PCS), que possui como característica os desenhos simples e de fácil reconhecimento (SARTORETTO; BERSCH, 2014).

Utilizando todos estes conceitos em conjunto pode-se fornecer uma ferramenta muito eficaz para pessoas com dificuldades na fala, fazendo com que elas quebrem barreiras sociais e tenham uma melhor qualidade de vida.

2.2 TAGARELA

O Tagarela é um projeto da Universidade Regional de Blumenau que tem por objetivo desenvolver uma plataforma que auxilie no tratamento de usuários com necessidades

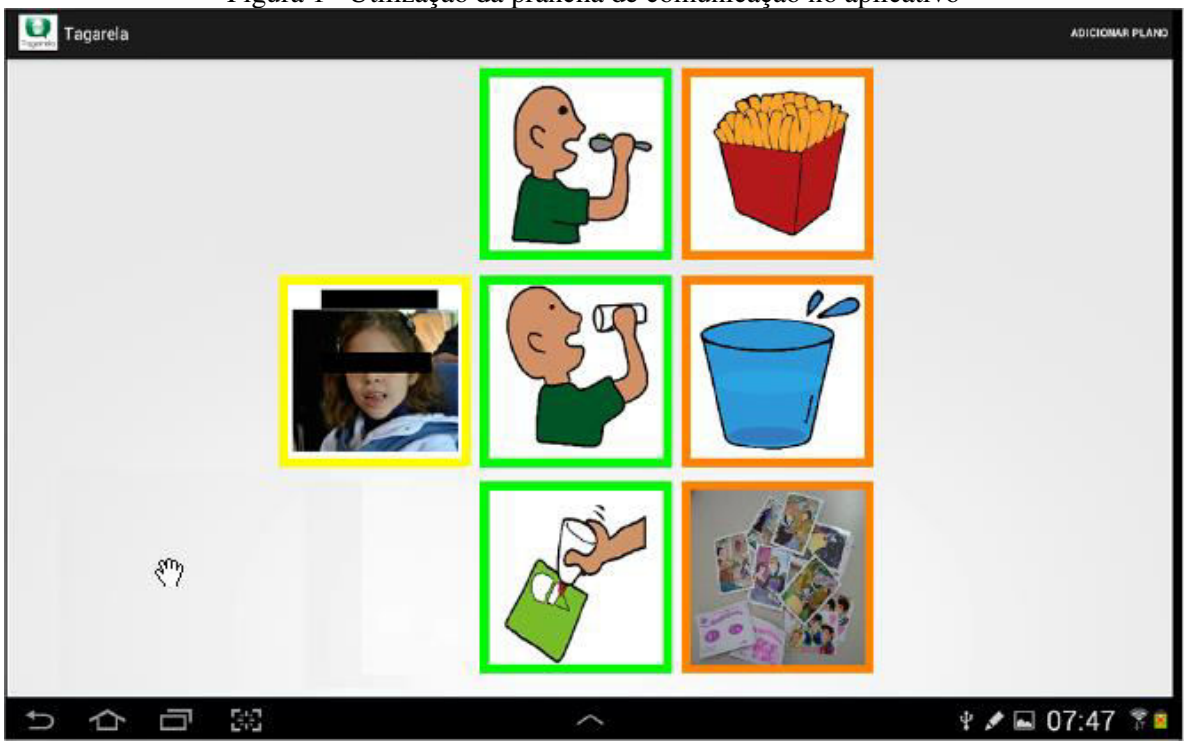
especiais, ou alguma limitação fonoarticulosa, fornecendo uma forma de Comunicação Alternativa. Além disso, o projeto busca facilitar a troca de experiências entre os indivíduos no decorrer do tratamento, que são denominados pelos perfis “Especialista”, “Tutor” e “Paciente” (TAGARELA, 2014).

Fabeni (2012) desenvolveu o primeiro produto deste projeto, um aplicativo para a plataforma iOS. Um dos principais objetivos deste aplicativo é disponibilizar um ambiente no qual o especialista, representado por um fonoaudiólogo, possa trocar experiências com o usuário em conjunto com seu tutor, geralmente um profissional da área da pedagogia, montar um plano de atividade para estimular a capacidade de comunicação do usuário (TAGARELA, 2014). Posteriormente este aplicativo foi desenvolvido também para a plataforma Android, através do trabalho acadêmico “Tagarela: Aplicativo de Comunicação Alternativa na Plataforma Android”, desenvolvido por Marco (2014).

O foco principal do trabalho desenvolvido por Marco (2014) é tornar a experiência do usuário interativa através da utilização de pranchas de comunicação, além de possibilitar a sincronização das informações entre dispositivos e permitir o uso da aplicação sem conexão com a internet. Porém, algumas funcionalidades não foram tratadas, como permitir a utilização de perfis para cada tipo de usuário (Especialista, Tutor e Paciente), possibilitar a reprodução de áudio encadeada nas pranchas de comunicação e também permitir que o usuário movimente os símbolos nas pranchas (MARCO, 2014).

Uma das principais atividades do aplicativo está relacionada à criação de um plano, pois é por meio desta atividade que o especialista e o tutor irão interagir e buscar resultados com seus usuários. Um plano contém as atividades do usuário e os símbolos que ele irá interagir, por meio de uma prancha de comunicação conforme a Figura 1 (MARCO, 2014).

Figura 1– Utilização da prancha de comunicação no aplicativo



Fonte: Marco (2014).

Está em construção também uma aplicação w*eb* que deve possuir as mesmas funcionalidades das versões atuais do Tagarela disponíveis para os dispositivos móveis com Android e iOS.

2.3 PHONEGAP E MATERIALIZE

PhoneGap é um *framework* de código desenvolvido pela Adobe Systems, tem licença de uso gratuita e permite a criação de aplicações móveis utilizando *Application Programming Interfaces* (API) *web* padronizadas. Em Outubro de 2011 o PhoneGap foi doado a Apache Software Foundation (ASF), porém permanece sob licença gratuita, fazendo parte agora do projeto Apache Cordova (ADOBE, 2014).

Existem diversas aplicações criadas através do PhoneGap, que já estão sendo distribuídas comercialmente (ADOBE, 2014). O *framework* PhoneGap permite o desenvolvimento de aplicativos para múltiplas plataformas de maneira simples, apenas escrevendo o código uma única vez, utilizando *HyperText Markup Language* (HTML), *Cascading Style Sheets* (CSS) e Javascript. E, após isso, é possível realizar o *deploy* para as plataformas móveis desejadas (ADOBE, 2014).

O PhoneGap possui diversos *plugins* que são instalados na aplicação para auxiliar no desenvolvimento. A Figura 2, mostra quais são os *plugins* suportados em relação a cada plataforma móvel, no desenvolvimento de um aplicativo utilizando o PhoneGap, onde a

primeira coluna mostra o *plugin* e as demais colunas mostram um comparativo entre as plataformas.

Figura 2– *Plugins* suportados pelo PhoneGap

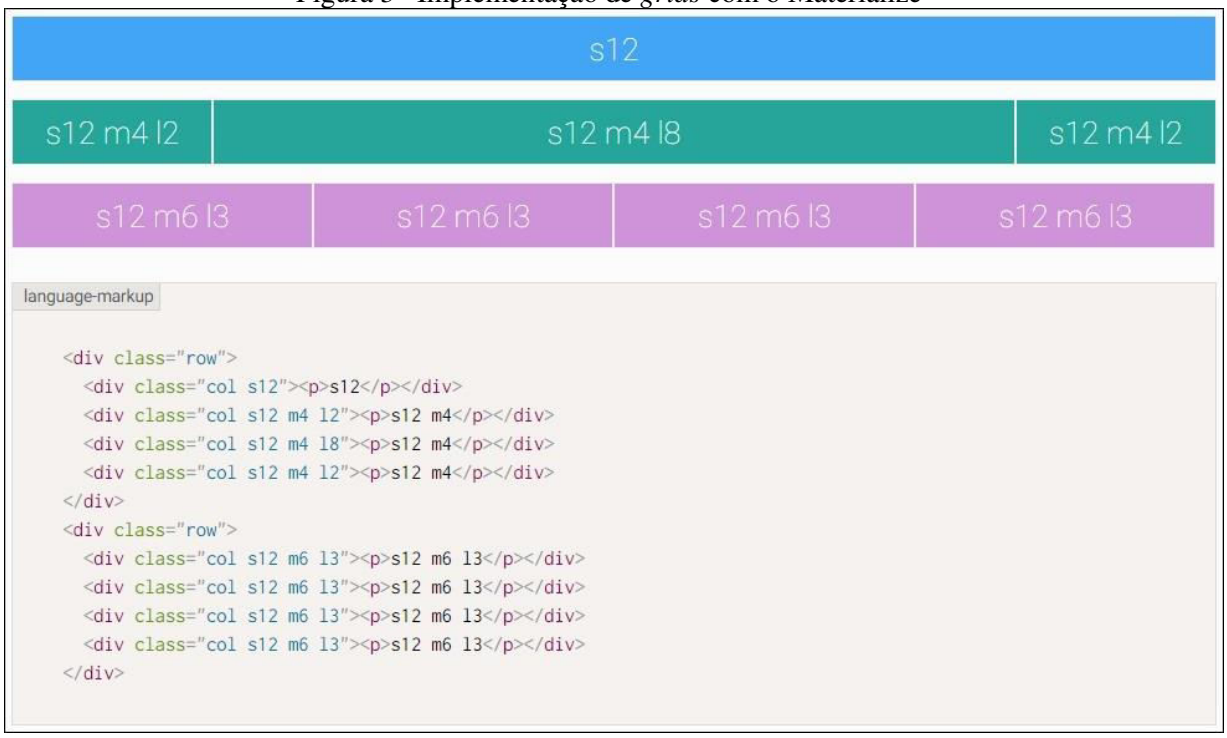
	iPhone / iPhone 3G	iPhone 3GS and newer	Android	Blackberry OS 6.0+	Blackberry 10	Windows Phone 8	Ubuntu	Firefox OS
Accelerometer	✓	✓	✓	✓	✓	✓	✓	✓
Camera	✓	✓	✓	✓	✓	✓	✓	✓
Compass	X	✓	✓	X	✓	✓	✓	✓
Contacts	✓	✓	✓	✓	✓	✓	✓	✓
File	✓	✓	✓	✓	✓	✓	✓	X
Geolocation	✓	✓	✓	✓	✓	✓	✓	✓
Media	✓	✓	✓	X	✓	✓	✓	X
Network	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Alert)	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Sound)	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Vibration)	✓	✓	✓	✓	✓	✓	✓	✓
Storage	✓	✓	✓	✓	✓	✓	✓	✓

Fonte: Adobe (2014).

Para a construção do estilo das telas no PhoneGap é utilizada a linguagem CSS, o que permite também a utilização de *frameworks* CSS, que surgem para facilitar este desenvolvimento de estilos. O Materialize é um *framework* CSS, baseado no Material Design da Google, que atende perfeitamente a necessidade de agilizar e facilitar este processo de desenvolvimento. Além de proporcionar um *design* limpo e intuitivo, ele possui também componentes padronizados, o que torna a experiência do usuário unificada (MATERIALIZE, 2015).

Através do Materialize é possível implementar estilos de páginas *web* utilizando o conceito de *grids*, que permite a disposição do conteúdo de forma vertical e horizontal, simulando linhas e colunas de uma tabela. Este conceito se encaixa perfeitamente quando o leiaute do *site* necessita mostrar conteúdo de imagens de forma dinâmica e responsiva. A Figura 3 apresenta a disposição de conteúdo e o respectivo trecho de código HTML, necessário para obter o resultado (MATERIALIZE, 2015).

Figura 3– Implementação de *grids* com o Materialize



Fonte: Materialize (2015).

2.4 TRABALHOS CORRELATOS

Foram selecionados dois trabalhos correlatos, ambos são aplicações de comunicação alternativa que utilizam o conceito de pranchas de comunicação. O item 2.4.1 descreve a ferramenta Tobii Sono Flex e o item 2.4.2 descreve a aplicação HelpTalk, ambas ferramentas comerciais disponíveis na plataforma Android.

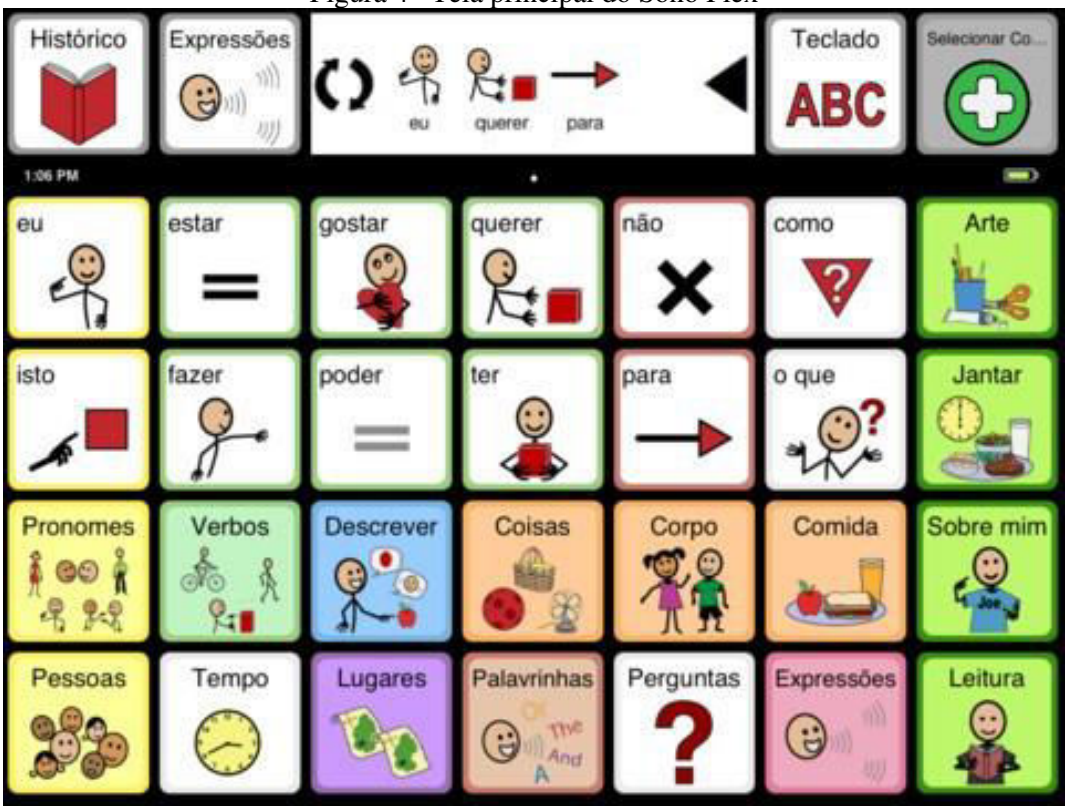
2.4.1 Sono Flex

Desenvolvido pela Comércio de Materiais Esportivos e Educativos Civiam Ltda EPP, é um aplicativo de comunicação assistiva e alternativa que oferece o recurso da linguagem aos usuários sem capacidade verbal ou alfabetização (CIVIAM, 2014). O Sono Flex é disponibilizado nas plataformas Android e iOS.

O Sono Flex oferece uma grande flexibilidade no que diz respeito a personalização do vocabulário, atendendo necessidades individuais e situacionais. Além de permitir a utilização da câmera e álbum de fotos do dispositivo para a criação de novos símbolos (CIVIAM, 2014).

A principal característica do aplicativo é sua fácil utilização, pois faz uso de uma interface baseada em símbolos, como visto na Figura 4, tornando a experiência do usuário muito mais agradável. Isto também permite que usuários sem alfabetização utilizem a aplicação sem maiores dificuldades.

Figura 4– Tela principal do Sono Flex



Fonte: Civiam (2014).

O aplicativo ainda permite ao usuário customizar esta disposição de símbolos na tela principal, para ficar de acordo com sua necessidade. Vale constatar ainda que o aplicativo é gratuito tanto no Android quanto no iOS.

2.4.2 HelpTalk

O HelpTalk, assim como outros aplicativos de Comunicação Alternativa, é voltado para pessoas incapazes de estabelecer comunicação fluente oral ou até mesmo escrita. Por meio de um conjunto de ações (símbolos) pré-definidas qualquer pessoa pode se comunicar, inclusive crianças e analfabetos, uma vez que são utilizados ícones juntamente com o texto (HELPTALK, 2014).

Uma das particularidades da aplicação é a possibilidade de agrupamento de símbolos por assunto ou situação. Estes agrupamentos podem ser mantidos de maneira privada ou pública, sendo que nesta última qualquer usuário poderá clonar o agrupamento e adequá-lo as suas necessidades. De maneira a facilitar o uso do aplicativo, os usuários podem também carregar seus agrupamentos para o dispositivo e utilizá-los de modo *offline*.

A Figura 5 mostra a tela principal do aplicativo que contém as ações disponíveis para um determinado agrupamento criado.

Figura 5– Visualização de um agrupamento e seus símbolos no HelpTalk

Fonte: HelpTalk (2014).

3 DESENVOLVIMENTO

Neste capítulo são apresentadas as etapas de desenvolvimento do aplicativo. A primeira seção apresenta os requisitos funcionais e não-funcionais. Em seguida, a segunda seção contém a especificação do aplicativo, utilizando diagramas da *Unified Modeling Language* (UML). A terceira seção detalha a implementação do aplicativo, apresentando os principais trechos de código e exemplos de uso das telas. Por fim, a quarta seção aborda os resultados obtidos deste trabalho.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O aplicativo descrito nesta proposta deverá:

a) permitir a criação de usuários com o perfil de tutor ou paciente (Requisito Funcional - RF);
b) permitir o vínculo entre usuários com o perfil de tutor e usuários com o perfil de paciente (RF);
c) permitir aos usuários a inserção e alteração de suas informações pessoais (RF);
d) permitir a criação e reutilização de pranchas de comunicação (RF);
e) permitir a interação do usuário com suas pranchas de comunicação (RF);
f) permitir que o usuário com o perfil de tutor possa criar e visualizar observações referentes a seus pacientes (RF);
g) permitir que o usuário possa visualizar seu histórico de uso das pranchas de comunicação (RF);
h) apresentar uma interface acessível, utilizando o conceito de símbolos de uma prancha nas telas da aplicação (Requisito Não-Funcional - RNF);
i) permitir uma fácil integração de novas funcionalidades entre as versões Android, iOS e *web* (RNF);
j) implementar todas as funções já existentes nas versões anteriores utilizando um único *framework* (RNF);
k) ser implementado utilizando o *framework* PhoneGap (RNF).

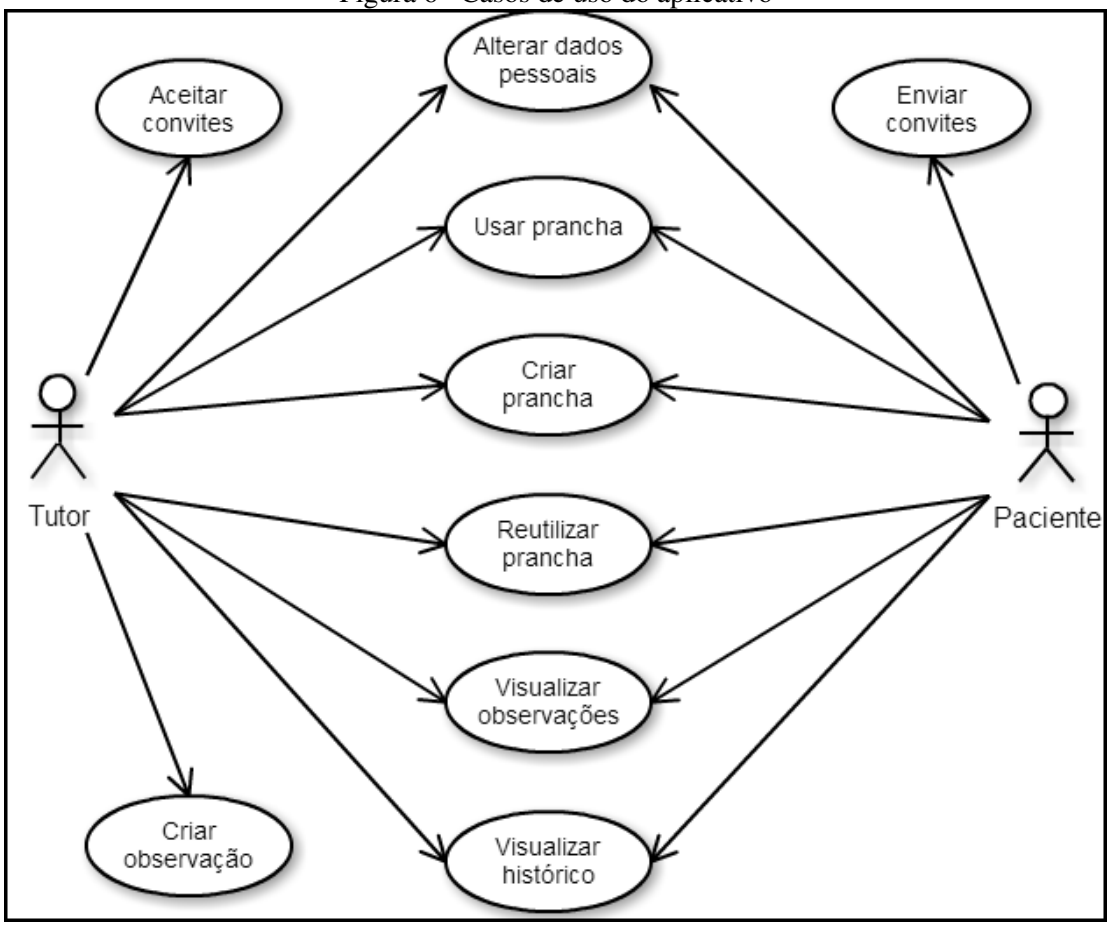
3.2 ESPECIFICAÇÃO

A especificação deste trabalho foi desenvolvida utilizando modelagem de diagrama de casos de uso, diagrama de atividades, modelo de entidade e relacionamento (MER) e diagrama de classes, todos da UML.

3.2.1 Casos de uso

Nesta seção são apresentados os casos de uso que descrevem as funcionalidades do aplicativo após os usuários estarem cadastrados. No aplicativo existem dois perfis de usuário, o `Tutor` e o `Paciente`. O `Tutor` tem como principal responsabilidade criar e gerenciar os planos e as pranchas de comunicação que serão utilizadas pelos pacientes. O `Paciente`, por sua vez, tem a responsabilidade de interagir com estes planos e pranchas, criados por seus tutores. Desta forma, foram identificados dois atores dentro do diagrama de casos de uso, conforme a Figura 6.

Figura 6– Casos de uso do aplicativo

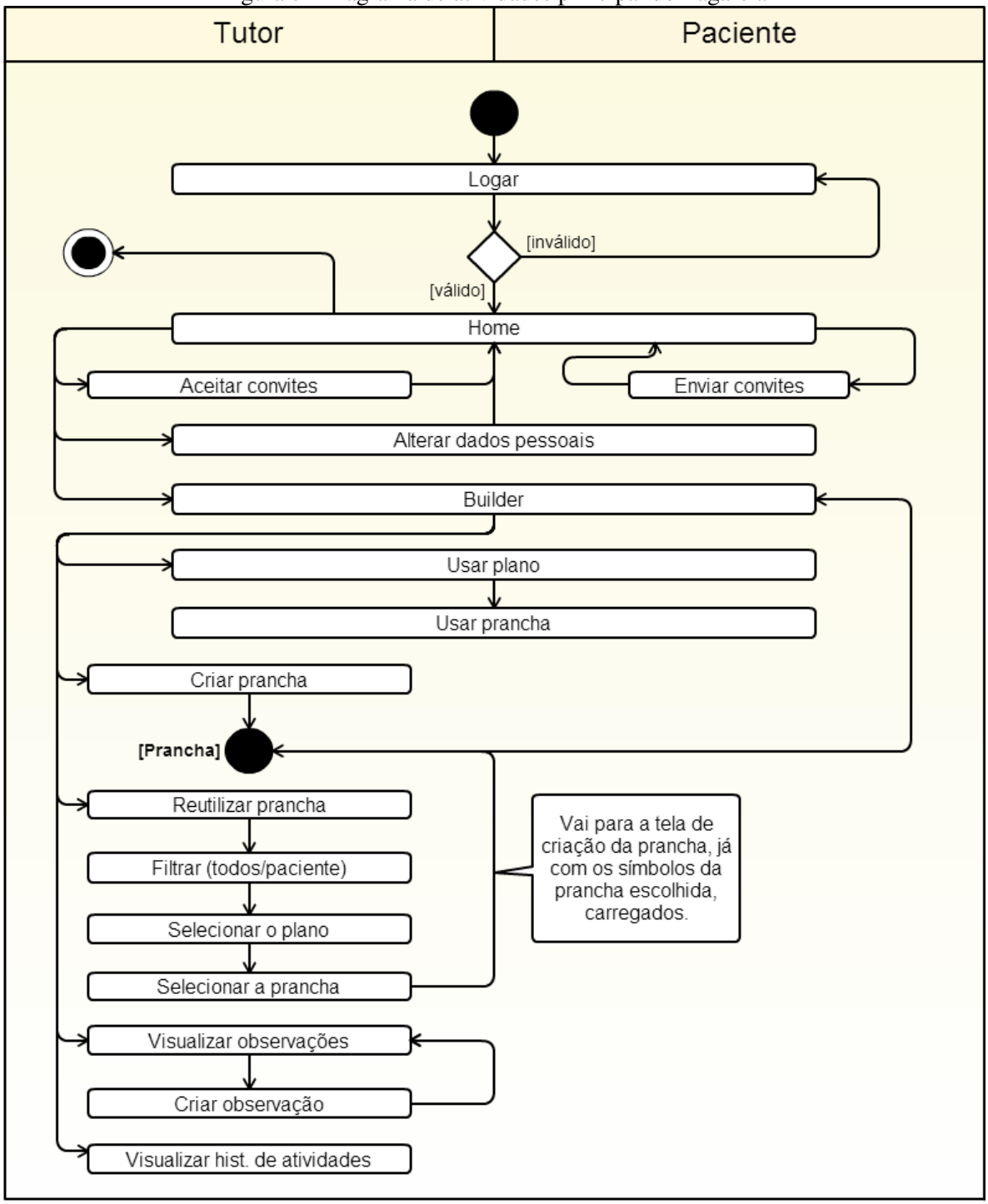


Os casos de uso acima foram desenvolvidos com base no padrão UML de especificação. Foi criado ao menos um cenário para cada caso de uso acima e cada um deles foi vinculado a um requisito funcional, defendendo sua existência. As descrições dos *Use Cases* (UC), condições, cenários e exceções destes casos de uso podem ser vistas no apêndice A.

3.2.2 Diagrama de atividades

O diagrama de atividades demonstra de forma geral a utilização de todos os recursos do Tagarela pelos dois atores envolvidos. Este diagrama divide-se em duas partes, a primeira contém as atividades principais e a segunda parte a criação de uma prancha, que é uma funcionalidade importante do Tagarela.

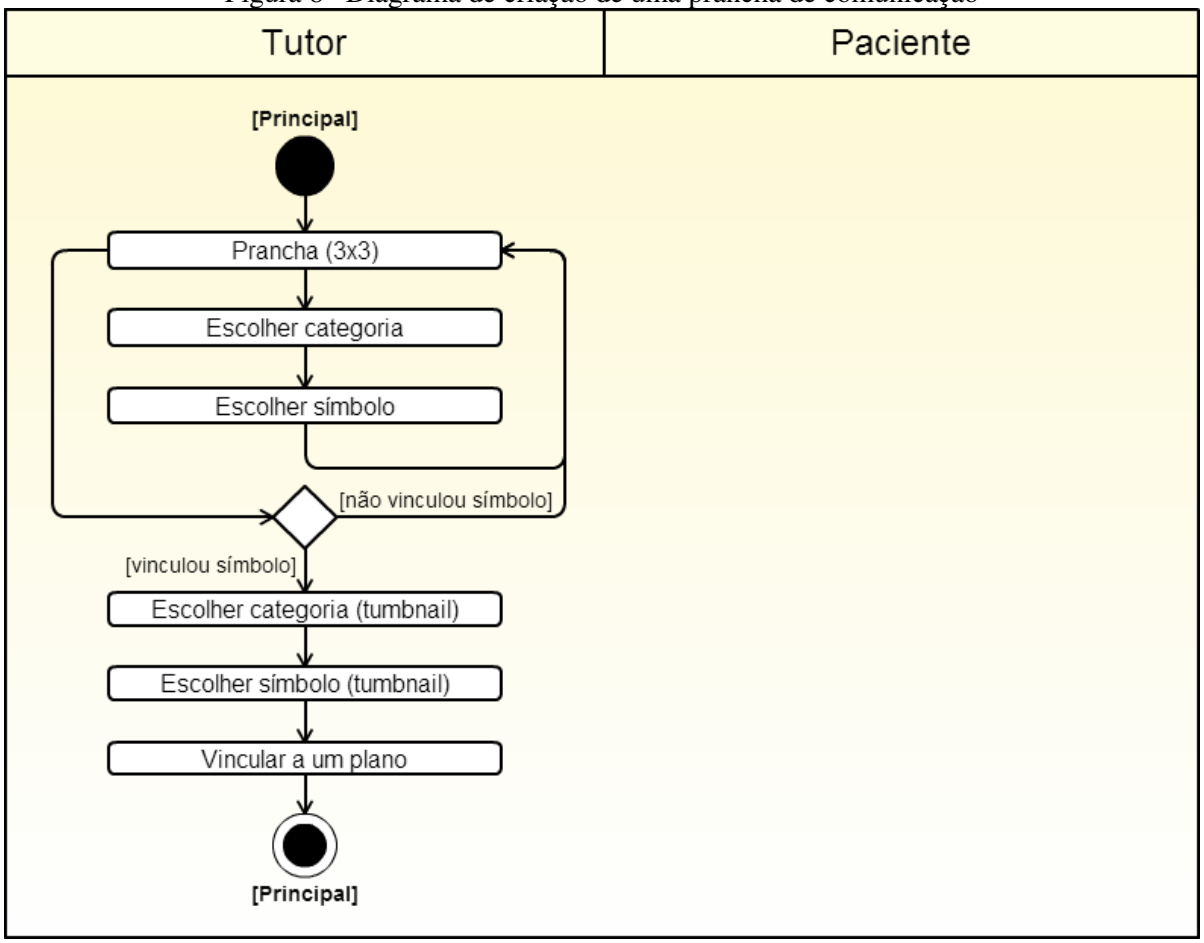
Figura 7– Diagrama de atividades principal do Tagarela



Conforme pode ser visto na Figura 7, as atividades no aplicativo iniciam-se com o *login* do usuário, caracterizado como tutor ou paciente. Caso o usuário seja um paciente, na tela principal (*home*) ele poderá enviar convites para outros usuários com perfil de tutor. Um tutor, por sua vez, poderá aceitar os convites que lhe foram enviados. Ambos os perfis podem alterar suas informações pessoais ainda nesta tela. Quando um convite é aceito, é criado um vínculo entre o tutor e o paciente. Este vínculo é chamado de construtor ou *builder*, dentro da aplicação, e irá aparecer na tela principal do usuário logado.

Após o usuário escolher o seu *builder* desejado, ele poderá utilizar as pranchas já criadas para ele, ou realizar as seguintes operações: visualizar e criar observações, visualizar o histórico de atividades e criar ou reutilizar uma prancha. A atividade de criação de uma prancha de comunicação é demonstrada através da Figura 8.

Figura 8– Diagrama de criação de uma prancha de comunicação



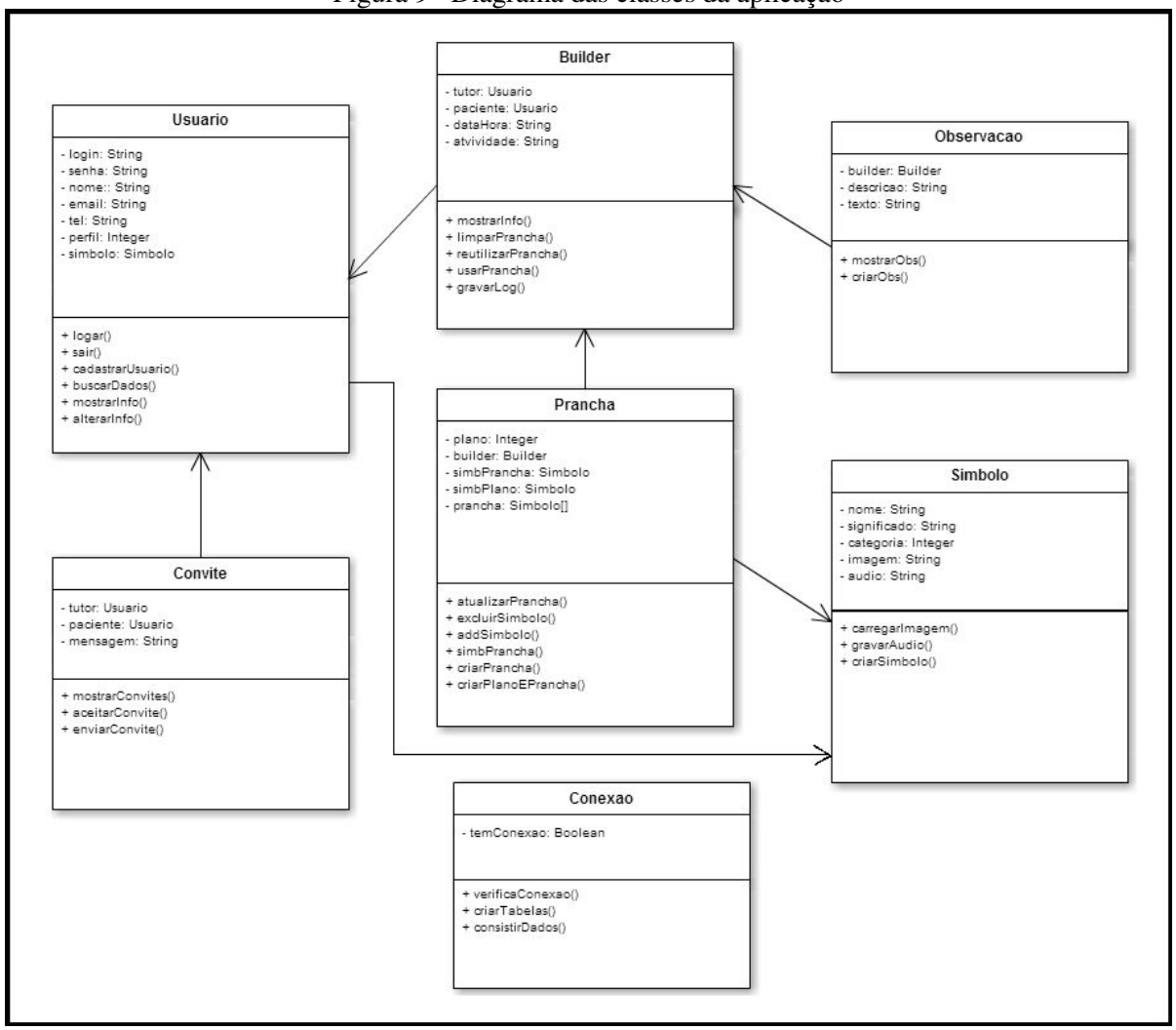
Ao criar uma prancha, o usuário deverá primeiramente escolher os símbolos desejados, que estarão separados entre quatro categorias de cores distintas, pessoas (amarelo), verbos (verde), substantivos (vermelho) e descritivos (azul). Caso optou-se pela reutilização de uma prancha já existente, os símbolos dela virão previamente carregados na nova prancha, com exceção daqueles que possuírem a categoria pessoas.

O usuário poderá então alterar os símbolos conforme desejar e confirmar a criação da prancha, escolhendo logo em seguida o plano no qual esta nova prancha ficará vinculada. Neste momento também poderá ser criado um novo plano para vincular a prancha.

3.2.3 Diagrama de classes

A seguir é apresentado o diagrama de classes que compõem o sistema e seus relacionamentos. Apesar da linguagem Javascript não possuir classes, apenas objetos, cada arquivo principal da aplicação foi simulado através de uma classe no diagrama, separados por funcionalidades ou assuntos, com o intuito de organizar a estrutura da aplicação e atender os requisitos especificados de maneira mais clara. A Figura 9 apresenta as classes da aplicação.

Figura 9– Diagrama das classes da aplicação



A primeira classe definida é a classe `Usuario`, que tem como responsabilidade o gerenciamento dos usuários da aplicação, possuindo métodos para acessar e sair do sistema, bem como criar um novo usuário escolhendo entre o perfil de tutor e paciente, informação que

será armazenada no atributo `perfil`. Além disso, através do método `alterarInfo` o usuário consegue modificar seus dados pessoais, como nome, email, telefone, função, etc.

A classe `Usuario` possui um relacionamento com a classe *`Convite`*, que através dos métodos `enviarConvites` e `aceitarConvites` cria um vínculo entre dois usuários com perfis distintos. Este vínculo será tratado pela classe `Builder`, que irá verificar qual o perfil do usuário logado na aplicação e definirá quem é o tutor e quem é o paciente, armazenando esta informação nos atributos `especialista` e `paciente`. O método `usarPrancha` permite ao usuário interagir com uma de suas pranchas de comunicação, neste método é implementado o `plugin media` do PhoneGap que é responsável por realizar a reprodução de áudio quando o usuário pressiona algum símbolo. Além disso, cada vez que este método é chamado cria um registro no histórico de atividades através do método `gravarLog`, informando que o usuário utilizou determinada prancha.

A classe `Builder` pode ser considerada a classe central da aplicação, pois além de ser responsável por tratar a interação dos usuários com suas pranchas, é através dela que são chamadas as demais classes responsáveis por efetuar as principais funcionalidades do sistema. Como a classe *`Observacao`*, que possui os métodos para visualizar e criar observações e também a classe *`Prancha`*, que é responsável por criar e compartilhar pranchas de comunicação, através dos métodos `criarPrancha` e `criarPlanoEPrancha`.

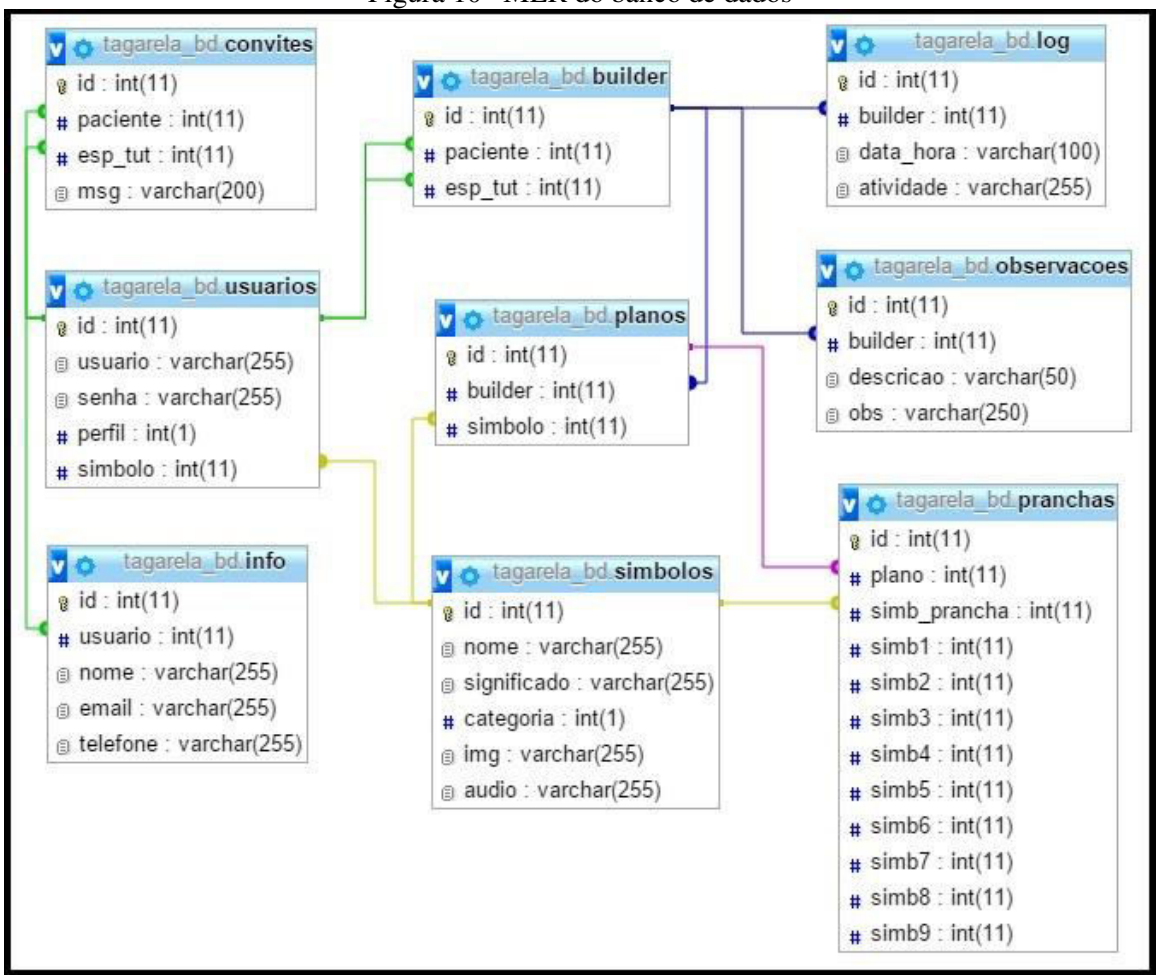
Uma prancha deve possuir um ou mais símbolos, este relacionamento é determinado através da classe `Simbolo`, que faz o gerenciamento dos símbolos cadastrados na aplicação. O método `carregarImagem` implementa o *plugin* `camera` para permitir ao usuário a seleção de uma imagem através do rolo de fotos de seu dispositivo, o método `gravarAudio` possui implementado o *plugin* `media-capture` que permite o usuário realizar a gravação de um áudio para vincular ao símbolo criado.

Por fim, a classe `Conexao` verifica a disponibilidade de conexão com a internet do dispositivo, através do método `verificarConexao`, e caso encontre conexão com a internet atribui o valor `true` para o atributo `temConexao`. Desta forma, caso o dispositivo móvel encontrar-se conectado a internet, o método `consistirDados` será chamado para realizar a atualização entre os dados locais e os dados o servidor.

3.2.4 Diagrama MER

As informações do sistema são persistidas em banco de dados, tanto no servidor quanto localmente na versão de dispositivos móveis. A Figura 10 apresenta o MER do banco de dados do aplicativo no servidor.

Figura 10– MER do banco de dados



Como pode ser visto na Figura 10 o banco de dados é composto por nove tabelas. As tabelas `usuarios` e `info` são responsáveis por armazenar os usuários cadastrados na aplicação e suas informações pessoais. Na tabela `convites` ficam armazenados todos os convites enviados pelos usuários e através da aceitação destes convites na aplicação é criado um novo registro na tabela `builder`. A tabela `log`, por sua vez, armazena todas as pranchas que foram utilizadas e a tabela `observacoes` possui todas as observações criadas, sendo que os registros de ambas as tabelas são gravados por construtor. Por fim, as tabelas `planos`, `simbolos` e `pranchas` armazenam respectivamente os planos, os símbolos e as pranchas cadastrados na aplicação, sendo que cada símbolo das pranchas possui um campo referente na tabela `pranchas`.

A estrutura das tabelas no banco de dados do servidor é a mesma do que as tabelas criadas nos dispositivos móveis, com exceção da existência da coluna `sinc` em todas as tabelas do banco de dados dos dispositivos. Esta coluna foi criada com a finalidade de atender a rotina de atualização dos dados entre o servidor e os dispositivos móveis.

3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas, e a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

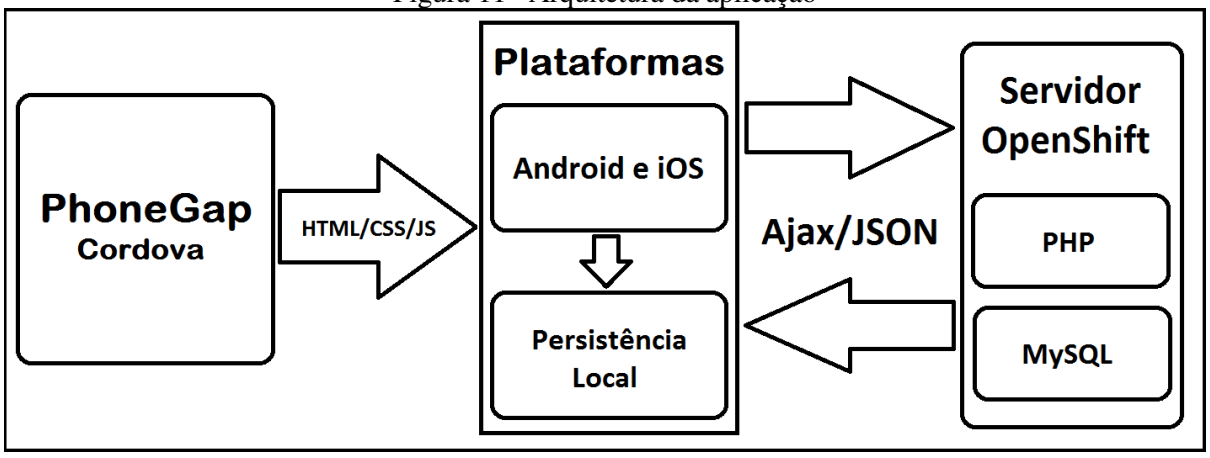
O desenvolvimento da aplicação foi realizado utilizando o *framework* PhoneGap. Desta forma, pôde-se desenvolver a mesma aplicação nas três versões desejadas (*web*, Android e iOS), utilizando os mesmos recursos de linguagem. Não foi possível manter o mesmo código fonte da versão *web* para a versão *mobile*, devido a necessidade de uso dos *plugins* do PhoneGap e da implementação de uma persistência local para os dispositivos móveis, no entanto isto poderia ser resolvido através do carregamento do código Javascript de forma dinâmica, verificando se a aplicação está sendo executada na plataforma *web* ou *mobile*.

O ambiente de desenvolvimento escolhido para a criação do trabalho foi a ferramenta Microsoft Expression Web 4, no qual foram utilizadas as linguagens HTML5, CSS 3.0, PHP 5.4 e Javascript. Para o desenvolvimento no Javascript foi usada a biblioteca jQuery v2.1.1 e para o CSS optou-se por fazer uso do Materialize (ver seção 2.3). Como ambiente de testes para a versão *web,* foram utilizados os navegadores Chrome versão 46.0.2490.80 m, Firefox versão 42.0 e Internet Explorer versão 11.0.9600.18053. Para os testes na plataforma *mobile* foi utilizado um dispositivo Samsung Galaxy Tab 2 modelo GT-P5110, com o sistema operacional Android na versão 4.1.2.

3.3.1.1 Arquitetura proposta

Nesta seção será descrita a arquitetura proposta para implementar o trabalho. A aplicação foi concebida sob uma arquitetura cliente/servidor, considerando também o *framework* PhoneGap, conforme mostra a Figura 11.

Figura 11– Arquitetura da aplicação



Utilizando o *framework* PhoneGap é realizado o *deploy* da aplicação *web* construída utilizando HTML, CSS e Javascript, para as plataformas Android e iOS. Nos dispositivos móveis é realizada ainda a persistência local, utilizando a tecnologia WebSQL, que fará a atualização dos seus dados com o banco de dados no servidor sempre que encontrar conexão com a internet. No servidor, as informações são gravadas em uma base de dados MySQL, onde estão também os *scripts* PHP, responsáveis por interagir com o código Javascript utilizando a metodologia *Asynchronous Javascript and XML* (AJAX) para resgatar os dados no dispositivo. Foi escolhido utilizar o servidor OpenShift, devido a ser uma ferramenta gratuita e, de forma geral, robusta.

3.3.1.2 Utilização do PhoneGap

O PhoneGap é uma ferramenta de desenvolvimento híbrido, porém é necessário efetuar algumas configurações no seu ambiente de trabalho antes de começar a utilizá-lo. Primeiramente deve ser decidido com quais plataformas deseja-se trabalhar, pois apesar das aplicações serem desenvolvidas utilizando recursos *web,* é necessário obter os *Software Development Kits* (SDK) específicos para cada plataforma. O Tagarela é disponibilizado nas plataformas Android e iOS, portanto para efetuar o *deploy* nas duas versões foi utilizado o sistema operacional Mac OS X, com o Android SDK instalado e o aplicativo Xcode em conjunto com a ferramenta Xcode *Command Line Tools*.

Após isso foi instalada a ferramenta `npm` *(NodeJS package manager)*, que é responsável por fazer o *download* e a instalação do PhoneGap. Para isso, é executado o comando `sudo npm install -g phonegap`, dentro do aplicativo `Terminal`, desta forma será efetuada a instalação do PhoneGap CLI (*Command Line Tools*) em seu ambiente de trabalho. A partir deste momento o PhoneGap está pronto para uso, todas as operações são realizadas através de comandos via `Terminal`, conforme mostra a Figura 12.

Figura 12– Comandos do PhoneGap CLI

```
1  $ phonegap create tagarela com.tagarela Tagarela
2
3  $ phonegap platform add android
4  $ phonegap platform add ios
5
6  $ phonegap plugin add org.apache.cordova.device
7  $ phonegap plugin add org.apache.cordova.camera
8  $ phonegap plugin add org.apache.cordova.media
9  $ phonegap plugin add org.apache.cordova.media-capture
10 $ phonegap plugin add org.apache.cordova.file-transfer
11
12 $ phonegap build android
13 $ phonegap build ios
14
15 $ phonegap emulate android
16 $ phonegap emulate ios
17 $ phonegap run android
18 $ phonegap run ios
```

Primeiramente executou-se o comando `phonegap create` passando três parâmetros, `tagarela`, `com.tagarela` e `Tagarela`. Este comando cria um novo projeto no diretório corrente, dentro da pasta `tagarela`, com identificador em estilo domínio reverso `com.tagarela`, de um aplicativo chamado `Tagarela`. Após isso, são copiados os arquivos da aplicação *web* criada para esta nova pasta. O aplicativo não pode ser executado ou simulado nos dispositivos desejados até que sejam adicionadas as plataformas. Isto é realizado através do comando `phonegap platform add`, onde se deve passar por parâmetro as plataformas desejadas, neste caso Android e iOS. É importante realçar que para executar a compilação na plataforma iOS estes comandos devem ser executados em um ambiente com MacOS.

Para executar o aplicativo existem dois passos necessários, que são mostrados da linha 12 até a linha 18. O primeiro comando, `phonegap build`, compila a aplicação para a plataforma desejada, gerando os arquivos específicos de cada plataforma dentro do subdiretório `platforms` do projeto. Após executar este comando, pode-se então rodar o aplicativo, tanto em um simulador quanto em um dispositivo móvel conectado ao computador, respectivamente através dos comandos `phonegap emulate` e `phonegap run`.

Entre as linhas 6 e 10 são adicionados diversos *plugins* que serão utilizados na aplicação. Os *plugins* `camera`, `media` e `media-capture`, por exemplo, são utilizados para acessar o rolo de fotos ou a câmera do dispositivo, reproduzir e gravar áudio, respectivamente. Já o *plugin* `file-transfer` é responsável por realizar o *download* ou *upload* de arquivos.

3.3.1.3 Persistência local e atualização dos dados

A persistência local foi implementada através do uso de WebSQL, onde os dados são persistidos em tabelas locais que possuem a mesma estrutura das tabelas do servidor. A especificação do WebSQL se baseia no uso de funções de *callback* que são executadas quando uma instrução *Structured Query Language* (SQL) é completada. Devido a esta natureza, é uma tecnologia assíncrona, o que significa que na implementação da persistência local foi necessário se preocupar para não modificar o comportamento do código Javascript da parte lógica da aplicação, devido à criação de eventos em tempo de execução. O Quadro 1 demonstra a implementação do método `criarObs` da classe `Observacao`, encontrado no código da parte *mobile*. Este método cria um registro de observação na base local dos dispositivos móveis.

Quadro 1 – Método `criarObs`

```
37 $(".cad-obs").click(function criarObs() {
38   db.transaction(transBuilderCriarObs, nokQuery);
39   function nokQuery(erro) {
40     alert("Erro ao realizar operação no banco de dados! Erro:"
41           +erro.code);
42   }
43   function transBuilderCriarObs(tx) {
44     tx.executeSql("SELECT id FROM builder WHERE paciente = ?
45                    AND esp_tut = ?", [localStorage.idBuilder,
46                    localStorage.idUser], insObs, nokQuery);
47   }
48   var builderCriObs;
49   function insObs(tx, results) {
50     var len = results.rows.length;
51     for (var i=0; i<len; i++) {
52       builderCriObs = results.rows.item(i).id;
53       tx.executeSql("INSERT INTO observacoes
54                      (builder,descricao,obs,sinc) VALUES (?,?,?,1)",
55                      [builderCriObs,$(".descricao").val(),
56                       $(".obs").val()]);
57     }
58     location.href = "../builder.html";
59   }
60 });
```

Como pode ser observado no código acima, o método primeiramente seleciona o *builder* atual. Isto é feito através da chamada da função `tx.executeSql` passando como parâmetro o comando SQL desejado, uma função de *callback*, chamada caso o comando for corretamente executado no banco e uma segunda função de *calback* que será chamada caso ocorra algum erro na execução do comando. O objeto `tx` é criado a partir da chamada da função `db.transaction`, onde o objeto `db` armazena a instancia do banco de dados que foi previamente criada. Após a realização do *select* na tabela `builders` a função de *callback* `insObs` é executada, trazendo todos os resultados retornados pelo *select* e também o objeto

`tx`, a partir disso então é possível executar o *insert* na tabela de observações, realizando uma nova chamada da função `tx.executeSql`.

A existência de um banco de dados local nos dispositivos móveis tomou necessária a criação de uma rotina que atualizasse estes dados locais com as informações persistidas no servidor, sempre que houver conexão com a internet. A lógica utilizada na consistência dos dados é a mesma independente da tabela que está sendo atualizada, com exceção da tabela de símbolos, onde foi implementada uma rotina mais elaborada considerando que seria necessário realizar também a transferência dos arquivos de imagem e áudio, o que se torna mais custoso do que consistir os dados armazenados nas tabelas do banco de dados, que são apenas campos do tipo `String` e `Integer`.

Em cada tabela do banco de dados local existe uma coluna chamada `sinc` que é responsável por armazenar o tipo de atualização que deve ser feita no servidor. Caso esta coluna possua o valor ‘0’ a rotina interpretará que não é necessário inserir ou atualizar o registro no servidor, caso a coluna possua o valor ‘1’ ou ‘2’ o registro deve ser respectivamente, inserido ou atualizado na tabela correspondente no servidor. O banco de dados é centralizado no servidor, portanto é considerado que as tabelas do mesmo sempre irão possuir as informações mais atualizadas entre os dispositivos que possuem conexão com a internet. Neste caso, após os registros da tabela local forem inseridos no servidor, através do valor existente na coluna `sinc` como explicado acima, todos os registros desta tabela são excluídos e é realizada a inserção dos registros existentes na tabela do servidor, mantendo assim as informações sempre atualizadas.

Quando está se falando de um fluxo pequeno de informações, como esse das tabelas no banco de dados, não há problema em realizar a consistência desta forma. Porém, para a transferência dos arquivos de imagem e áudio optou-se por realizar o *download* dos arquivos que se encontram no servidor apenas quando necessário. Desta forma, é verificado se a quantidade de registros na tabela de símbolos do servidor é maior do que a quantidade de símbolos existentes na tabela local, então é identificado quais são os símbolos faltantes no dispositivo móvel e só então é realizado o *download* dos arquivos de imagem e áudio referentes a estes símbolos. Após isso, a rotina identifica quais os registros da tabela de símbolos do banco local, que possuem a coluna `sinc` com valor ‘1’ e realiza o *upload* dos arquivos de imagem e áudio destes símbolos. O *download* e o *upload* são feitos pelo *plugin* `file-transfer` do PhoneGap, como demonstrado respectivamente no Quadro 2 e no Quadro 3.

Quadro 2 – Utilização do *plugin* `file-transfer` para realizar o *download* dos arquivos de imagem e áudio dos símbolos

```
var ft = new FileTransfer();
for (var i = (simbolosNome.length-1); i >= 0; i--) {
    var urlImg = "http://tagarela-afwippel.rhcloud.com/img/"
                 +simbolosImg[i];
    var localImg = "file:///storage/sdcard0/tagarela/img/"
                   +simbolosImg[i];
    var urlAud = "http://tagarela-afwippel.rhcloud.com/audio/"
                 +simbolosAudio[i];
    var localAud = "file:///storage/sdcard0/tagarela/audio/"
                   +simbolosAudio[i];
    tx.executeSql("INSERT INTO simbolos (nome,significado,categoria,
                   img,audio,sinc) VALUES (?,?,?,?,?,0)",
                  [simbolosNome[i],simbolosSign[i],simbolosCat[i],
                   localImg,localAud]);
    // Chamada do método download do plugin file-transfer
    ft.download(encodeURI(urlImg), localImg, successDownload,
                failDownload);
    ft.download(encodeURI(urlAud), localAud, successDownload,
                failDownload);
}
db.transaction(webSimbolos, nokTransSinc, okTransSinc);
```

Quadro 3 – Utilização do *plugin* `file-transfer` para realizar o *upload* dos arquivos

```
var opImg = new FileUploadOptions();
var ftImg = new FileTransfer();
for (var i = 0; i < simbsNomeLocal.length; i++) {
    var imgLocal = simbsImgLocal[i];
    var imgWeb = dirImg + simbsNomeLocal[i] + ".png";
    opImg.fileKey = "file";
    opImg.fileName = imgLocal.substr(imgLocal.lastIndexOf('/')+1);
    opImg.mimeType = "image/png";
    opImg.chunkedMode = false;
    var params = {};
    params.value1 = imgWeb;
    opImg.params = params;
    // Chamado do método upload para enviar a imagem ao servidor
    ftImg.upload(imgLocal,
encodeURI("http://tagarela-afwippel.rhcloud.com/scripts/upload.php"),
successUpload, failUpload, opImg);
}
```

As parte principal destes dois trechos de código baseia-se principalmente no uso dos métodos `ft.download` e `ftImg.upload`. O objeto `FileTransfer` deve ser criado previamente, através dele fica disponível o uso destes dois métodos, o primeiro parâmetro do método *download* é a *Uniform Resource Locator* (URL) onde se encontra o arquivo desejado, o segundo parâmetro é a *Uniform Resource Identifier* (URI) onde o arquivo será armazenado no dispositivo e o terceiro e quarto parâmetro são as funções de *callback* de sucesso e erro, respectivamente. O método *upload* espera como parâmetros a URI de onde o arquivo está armazenado no dispositivo, a URL especificando para onde este arquivo será transferido, que neste caso pode ser um *script* PHP conforme código acima, e por fim as duas funções de *callback*.

3.3.1.4 Criação e reutilização de pranchas

Nesta seção será detalhada a implementação das duas principais funcionalidades do Tagarela, que são a criação e a reutilização de uma prancha de comunicação. Estas duas funcionalidades compartilham os mesmos métodos, a diferença é que quando é escolhido reutilizar uma prancha, primeiramente deve ser efetuada a seleção da prancha que servirá como modelo para esta nova prancha. O Quadro 4 mostra um trecho do método `reutilizarPrancha` da classe `Builder`, implementado em Javascript e utilizando a metodologia AJAX, para atender esta funcionalidade na plataforma *web*.

Quadro 4 – Método `reutilizarPrancha`

```
138 $.ajax({
139   type     : "post",
140   url      : "http://tagarela-afwippel.rhcloud.com/scripts/
141              buscar-prancha.php",
142   data     : dados,
143   dataType : "json",
144   success  : function(ret) {
145     $("body").removeClass("loading");
146     if (ret.erro) {
147       alert(ret.msg);
148     }
149     else {
150       localStorage.simb1 = "../img/"+ret.simbolosImg[0];
151       localStorage.simb2 = "../img/"+ret.simbolosImg[1];
152       localStorage.simb3 = "../img/"+ret.simbolosImg[2];
153       localStorage.simb4 = "../img/"+ret.simbolosImg[3];
154       localStorage.simb5 = "../img/"+ret.simbolosImg[4];
155       localStorage.simb6 = "../img/"+ret.simbolosImg[5];
156       localStorage.simb7 = "../img/"+ret.simbolosImg[6];
157       localStorage.simb8 = "../img/"+ret.simbolosImg[7];
158       localStorage.simb9 = "../img/"+ret.simbolosImg[8];
159       localStorage.idSimb1 = ret.simbolosId[0];
160       localStorage.idSimb2 = ret.simbolosId[1];
161       localStorage.idSimb3 = ret.simbolosId[2];
162       localStorage.idSimb4 = ret.simbolosId[3];
163       localStorage.idSimb5 = ret.simbolosId[4];
164       localStorage.idSimb6 = ret.simbolosId[5];
165       localStorage.idSimb7 = ret.simbolosId[6];
166       localStorage.idSimb8 = ret.simbolosId[7];
167       localStorage.idSimb9 = ret.simbolosId[8];
168       localStorage.idSimbPrancha = 0;
169       localStorage.idPlanoPrancha = 0;
170       location.href = "../prancha/prancha.html";
171     }
172   }
```

Neste trecho de código é realizada a busca dos símbolos desta prancha no banco de dados. Foi escolhido utilizar a metodologia AJAX para realizar o acesso ao banco de dados do servidor, por ser uma metodologia simples e eficaz que atende bem a este tipo de requisição. A URL passada por parâmetro condiz com o *script* PHP que realizará este acesso ao banco, este *script* está localizado dentro do servidor OpenShift. Após a requisição retornar com

sucesso, as variáveis que armazenam os símbolos da nova prancha em memória são inicializadas com os símbolos da prancha modelo e a página HTML que permite o usuário alterar estes símbolos é chamada, seguindo o processo normal de criação de uma prancha.

Após o usuário selecionar todos os símbolos desejados para sua nova prancha, bem como o plano no qual ela será vinculada, o método `criarPrancha` da classe `Prancha` é chamado. O Quadro 5 apresenta um trecho da implementação deste método e posteriormente uma explicação mais detalhada.

Quadro 5 – Método `criarPrancha`

```
17 $.ajax({
18   type     : "post",
19   url      : "http://tagarela-afwippel.rhcloud.com/scripts/
20              gravar-prancha.php",
21   data     : dados,
22   dataType : "json",
23   success  : function(ret) {
24     $("body").removeClass("loading");
25     if (ret.erro) {
26       alert(ret.msg);
27     }
28     else {
29       location.href = "../builder.html";
30     }
31   }
```

A variável `dados` na linha 21 armazena as informações que serão enviadas ao *script* `gravar-prancha.php`, no formato *Array* JSON (*Javascript Object Notation*). Estas informações são variáveis e dizem respeito aos símbolos e ao plano que o próprio usuário selecionou a medida que realizava o processo de criação da prancha, nas telas anteriores. O trecho do *script* que realiza a gravação da prancha no banco de dados é demonstrado no Quadro 6, onde é feito uma operação *Insert* na tabela `pranchas` com as informações que foram enviadas pela requisição AJAX, no método `criarPrancha`.

Quadro 6 – *Script* de gravação da prancha na base de dados

```
1 $query = "INSERT INTO pranchas (plano, simb_prancha, simb1,
2           simb2, simb3, simb4, simb5, simb6, simb7, simb8,
3           simb9) " . "VALUES ($plano, '$simbPrancha',
4           '$simb1', '$simb2', '$simb3', '$simb4', '$simb5',
5           '$simb6', '$simb7', '$simb8', '$simb9')";
6 mysqli_query($con,$query);
```

3.3.1.5 Interação com as pranchas criadas

A utilização das pranchas pelos usuários é também implementada dentro da classe `Builder`, no método `usarPrancha`. No Quadro 7 é apresentado um trecho de código com a implementação deste método na aplicação *web*.

Quadro 7 – Reprodução de áudio no método `usarPrancha`

```
52 complete: function() {
53    $("body").removeClass("loading");
54
55    $(".img-simbolo").click(function() {
56        var src = "audio/"+$(this).attr("title");
57        audioElement.setAttribute("src",src);
58        audioElement.play();
59    });
60 }
```

Primeiramente, pode-se notar que no evento `complete` da requisição AJAX é realizada a reprodução do som vinculado ao símbolo quando o usuário pressioná-lo, operação que é realizada entre as linhas 55 e 59, na chamada do evento `$(".img-simbolo").click`. Para a implementação desta funcionalidade na plataforma *mobile*, foi utilizado o *plugin* `media` dentro deste evento de clique no símbolo, conforme apresentado no Quadro 8.

Quadro 8 – Uso do *plugin* `media`

```
63 $(".img-simbolo").click(function() {
64   srcAudio = $(this).attr("title");
65   var media = new Media(srcAudio, onSuccess, onError);
66   media.play();
67 });
```

Na linha 65 é instanciado um objeto `media`, implementado dentro do *plugin* `media` do PhoneGap, este objeto já é criado passando a URI do áudio que será reproduzido como parâmetro e também duas funções de *callback,* seguindo o padrão dos *plugins* do PhoneGap. A partir do objeto `media` pode ser chamada a função `play` que irá reproduzir o áudio carregado na memória.

Após os símbolos da prancha terem sido carregados e apresentados ao usuário no formato 3x3, é realizada a gravação de um registro no histórico de atividades, informando que o usuário utilizou esta prancha de comunicação. Este registro é gravado com a data atual do sistema e a prancha que o usuário selecionou, informações que são geradas entre as linhas 64 e 74.

3.3.1.6 Criação de símbolos e uso dos *plugins* `camera` e `media-capture`

A rotina de criação de um novo símbolo para as versões *mobile* faz uso de dois *plugins* do PhoneGap, o *plugin* `camera` e o *plugin* `media-capture`, a partir deles foi possível permitir ao usuário a seleção de uma imagem em seu dispositivo e também a gravação do áudio que será vinculado ao símbolo. O Quadro 9 apresenta a utilização do *plugin* `camera` na implementação da rotina que seleciona uma imagem do rolo de fotos do dispositivo.

Quadro 9 – Seleção de imagem e uso do *plugin* `camera`

```
function selImagem(event) {
  navigator.camera.getPicture(onPhotoURISuccess,onFail,
       {quality:50,
        destinationType:navigator.camera.DestinationType.FILE_URI,
        sourceType:navigator.camera.PictureSourceType.PHOTOLIBRARY});
}
var imgURI = null;
function onPhotoURISuccess(imageURI) {
  imgURI = imageURI;
  $(".img-simb").attr("src", imageURI);
  var corBorda;
  switch (localStorage.catSel) {
    case '1':
        corBorda = "yellow";
        break;
    case '2':
        corBorda = "red";
        break;
    case '3':
        corBorda = "green";
        break;
    case '4':
        corBorda = "blue";
        break;
    default:
        corBorda = "black";
        break;
    }
    $(".img-simb").css({"border":"7px solid "+corBorda});
}
```

Na função `selImagem` é executada a chamada da função `getPicture`, que recebe como parâmetro uma função de *callback* que será executada no caso de tudo ocorrer corretamente, uma função de *callback* de erro e um conjunto de opções não obrigatórias. Dentro destas opções pode ser especificado, por exemplo, a qualidade desejada da imagem retornada, o formato de retorno da imagem, podendo ser em base-64 ou a URI da imagem no dispositivo, etc. Caso não seja especificado nenhuma opção, será considerado o conjunto padrão de opções do *plugin*. Na função de *callback* de sucesso a imagem será retornada, conforme o formato escolhido, possibilitando a manipulação da mesma. O Quadro 10 apresenta o trecho de código onde é utilizado o *plugin* `media-capture`.

Quadro 10 – Gravação de áudio e uso do *plugin* `media-capture`

```
124 function selAudio(event) {
125   navigator.device.capture.captureAudio(captureSuccess, captureError,
126                                         {limit: 1});
127 }
128
129 var audURI = null;
130 function captureSuccess(mediaFiles) {
131   var i, len;
132   for (i = 0, len = mediaFiles.length; i < len; i += 1) {
133     audURI = dirGravacao + mediaFiles[i].name;
134     $(".img-simb").attr("title", audURI);
135   }
136 }
137 function captureError(error) {
138   alert("Erro ao gravar audio: " + error.code);
139 }
```

O *plugin* `media-capture` é assíncrono e permite a gravação de áudios múltiplos em uma única sessão. Ao ser executada a função *`captureAudio`* abrirá o aplicativo de gravação de áudio nativo do dispositivo. Seus parâmetros são duas funções de *callback,* uma de sucesso e outra de erro e um conjunto de opções, assim como o *plugin* `camera`. As opções suportadas são o limite de áudios que poderão ser gravados e a duração máxima de cada áudio gravado.

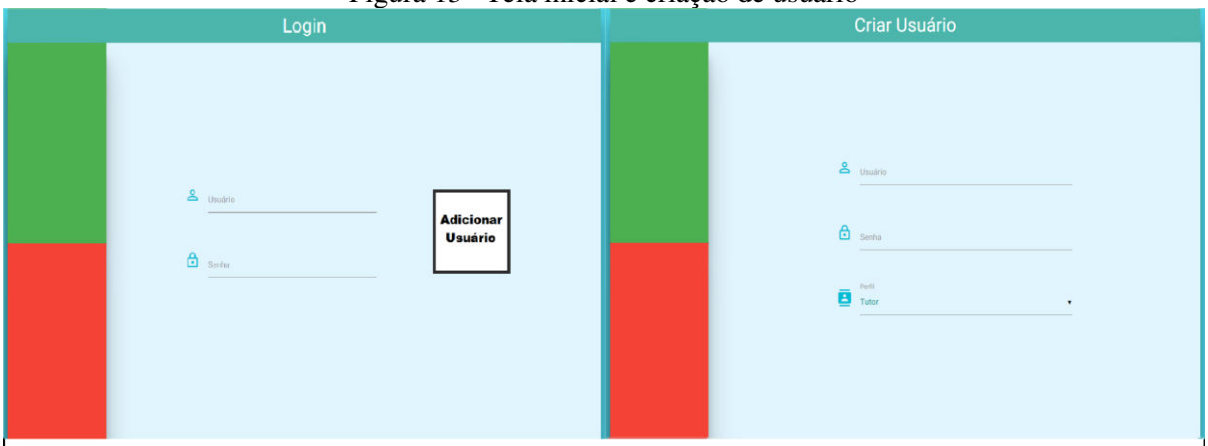
3.3.2 Operacionalidade da implementação

No Tagarela, a principal atividade do usuário é a interação com suas pranchas de comunicação. Porém, antes disso é necessário citar algumas etapas necessárias para que esta funcionalidade seja realizada, como a criação de usuários, o envio e a aceitação de convites e a criação de uma prancha de comunicação. Além destas, ainda existem outras funcionalidades secundárias, como a reutilização de uma prancha já existente, a criação de observações e a visualização do histórico de atividades do paciente. Esta seção apresenta de forma resumida a operacionalidade de cada uma das etapas supracitadas.

3.3.2.1 Criação de usuários

A tela inicial da aplicação possibilita o usuário realizar o *login* no Tagarela ou criar um novo usuário, conforme Figura 13a. Ao pressionar o símbolo `Adicionar Usuário`, será apresentada a tela de criação de usuários Figura 13b, onde deverá ser informado o usuário, a senha e o perfil desejado para o novo usuário, que poderá ser `Tutor` ou `Paciente`. Na aplicação desenvolvida a navegação entre as telas, assim a confirmação ou cancelamento das operações, é realizada através de dois botões laterais. O botão verde serve para realizar confirmação e o avanço das telas e botão vermelho serve para o cancelamento ou retorno a tela anterior.

Figura 13– Tela inicial e criação de usuário

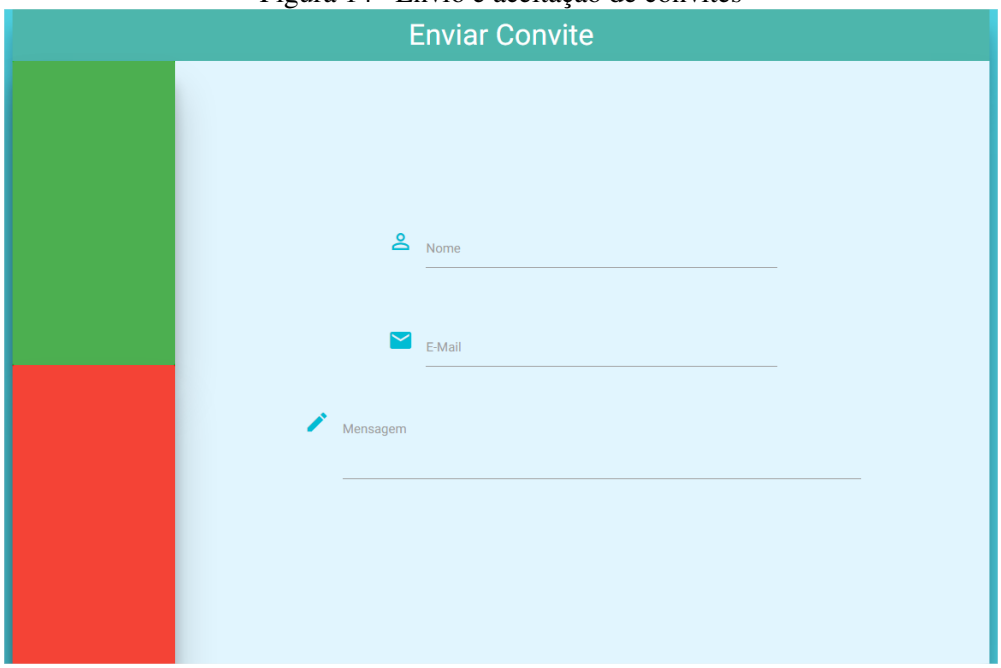


(a) Tela de *login* (b) Criação de usuário

3.3.2.2 Vínculo de usuários

Após criar os tutores e pacientes deve-se criar um vínculo entre eles, para concretizar o papel destes dois perfis dentro da aplicação. Um tutor, por exemplo, não consegue exercer seu papel se não estiver vinculado a um paciente, assim como um paciente também precisa estar vinculado a pelo menos um tutor. Este vínculo dentro da aplicação forma um `builder`. Sendo assim, todos os pacientes devem possuir um `builder` com o perfil tutor e todos os tutores devem possuir um `builder` com o perfil paciente, para que possam utilizar as funcionalidades da aplicação. A criação destes vínculos é feita através do envio e aceitação de convites, onde os pacientes podem apenas enviar convites para tutores e os tutores podem apenas aceitar os convites que lhe foram enviados. A Figura 14 abaixo mostra a tela de envio de convites.

Figura 14– Envio e aceitação de convites

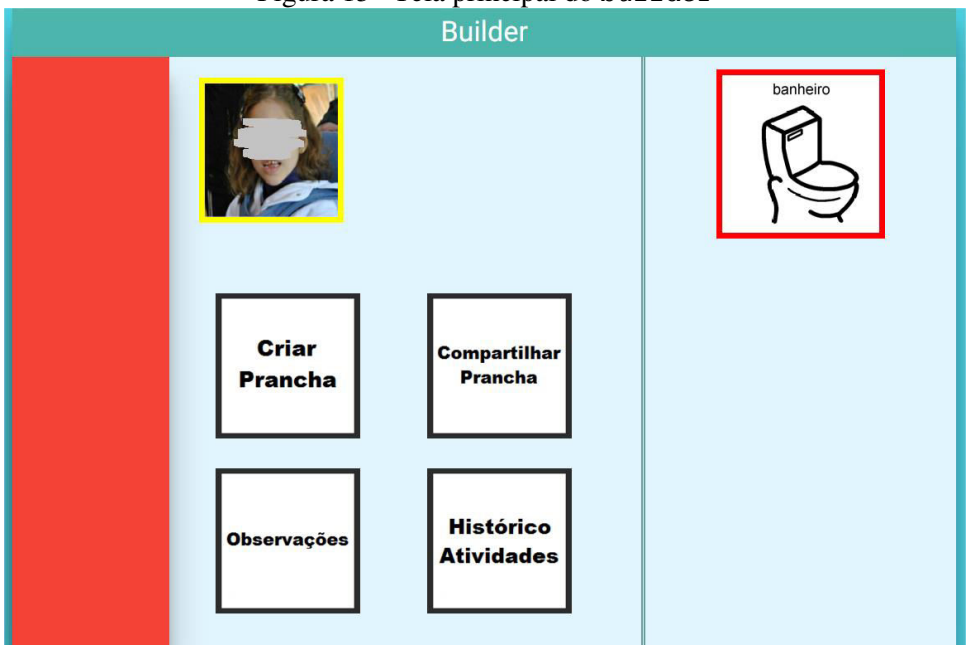


Ao criar um paciente, já é criado automaticamente um `builder` para ele. Este facilitador serve para os usuários que não necessitam estar vinculados a algum tutor, e também não queiram passar por este processo de vínculo de usuário para utilizar todas as funcionalidades da aplicação.

3.3.2.3 Criação de prancha de comunicação

O processo de criação de uma prancha de comunicação, assim como as demais funcionalidades da aplicação, é realizado a partir da tela do `builder`, conforme Figura 15. Nesta tela é possível também acessar, além da criação de pranchas, todas as funcionalidades que serão demonstradas nas próximas seções. Além disso, no lado direito são listados todos os planos deste `builder`.

Figura 15– Tela principal do `builder`



Ao pressionar o símbolo `Criar Prancha`, será mostrado ao usuário uma prancha 3x3 vazia, conforme a Figura 16. Ele então deve selecionar dentro de uma categoria, quais símbolos farão parte desta nova prancha pressionando o símbolo `Adicionar`. A seleção do símbolo é realizada a partir de quatro categorias específicas, verbos, substantivos, descritivos e pessoas, conforme demonstrado na Figura 17.

Figura 16– Escolha dos símbolos para a prancha

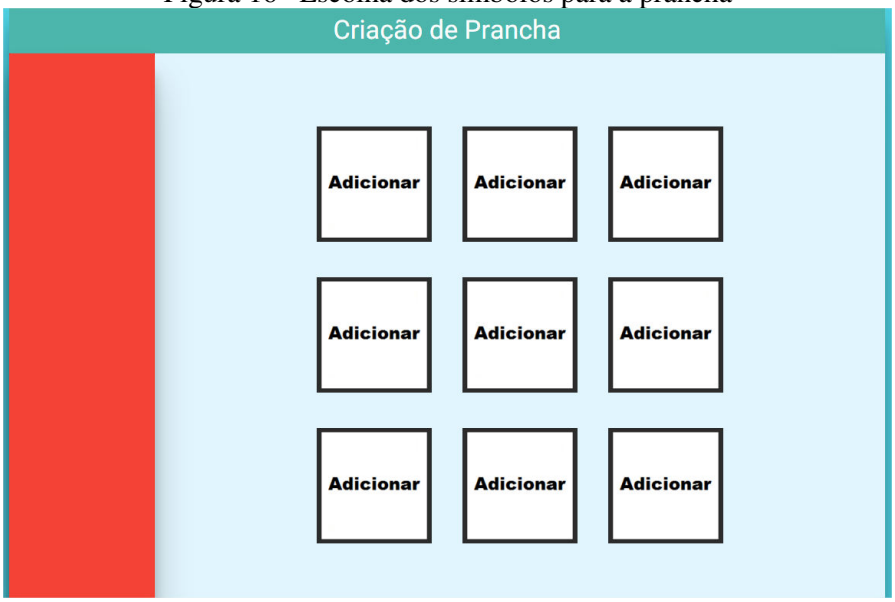
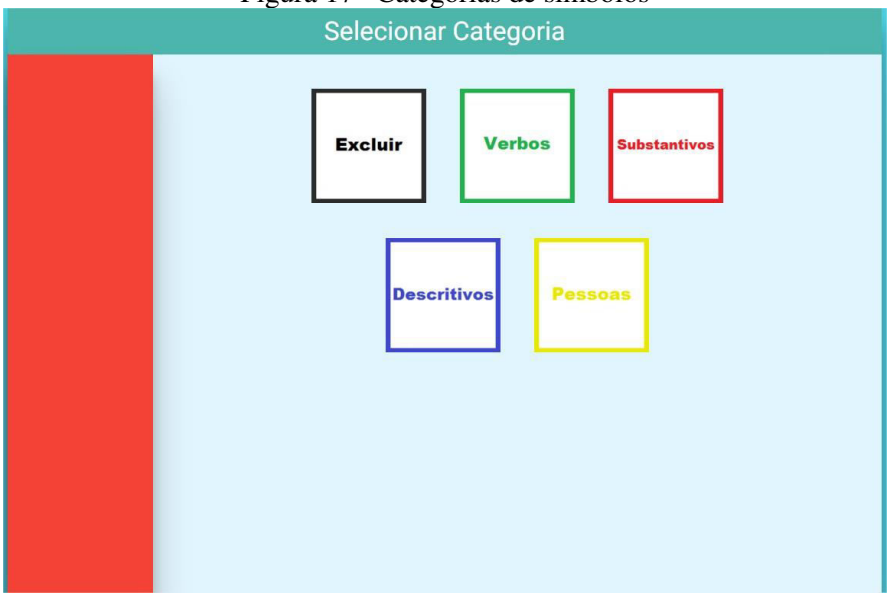


Figura 17– Categorias de símbolos



Após escolher os símbolos desejados o usuário deverá vincular a prancha a um plano existente, ou então criar um plano para incluir esta nova prancha de comunicação. A aplicação então retornará para a tela principal do `builder`, onde a nova prancha já estará disponível para ser utilizada.

3.3.2.4 Compartilhamento de pranchas

Além de criar uma prancha, há também a possibilidade de ser reutilizada uma prancha já existente. Este processo é similar a criação de uma prancha, tendo como única diferença a escolha da prancha que será utilizada como modelo. Neste caso, na tela onde seria mostrada uma prancha 3x3 vazia, conforme Figura 16, serão previamente carregados os símbolos da prancha escolhida como modelo.

A seleção da prancha modelo pode ser feita de duas formas, filtrando apenas as pranchas do próprio paciente selecionado, ou todas as pranchas dos demais pacientes cadastrados, conforme apresenta a Figura 18. Caso for escolhida esta segunda opção, não serão carregados os símbolos da categoria `Pessoa`, pois estes são considerados confidenciais para cada paciente. A Figura 19 mostra uma prancha, já criada para outro paciente, sendo reutilizada.

Figura 18– Seleção do filtro de compartilhamento de pranchas

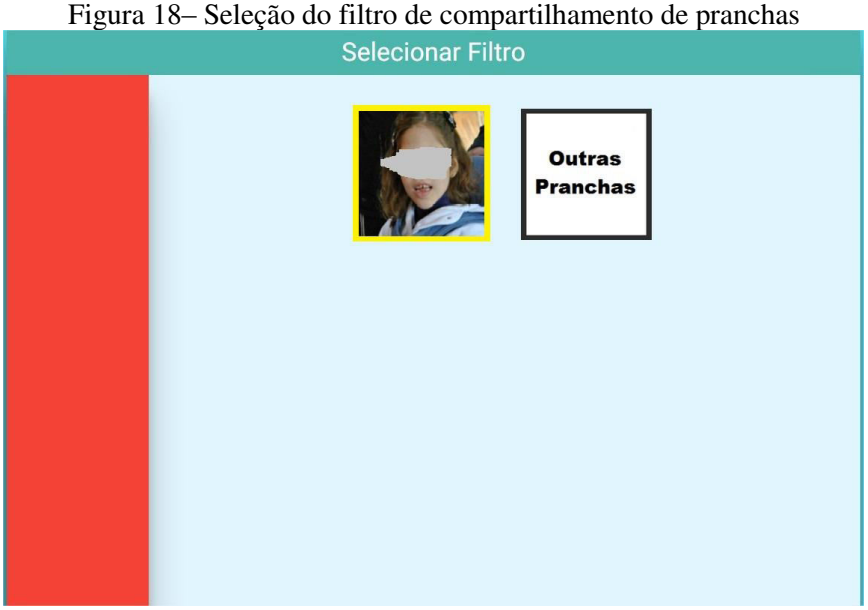
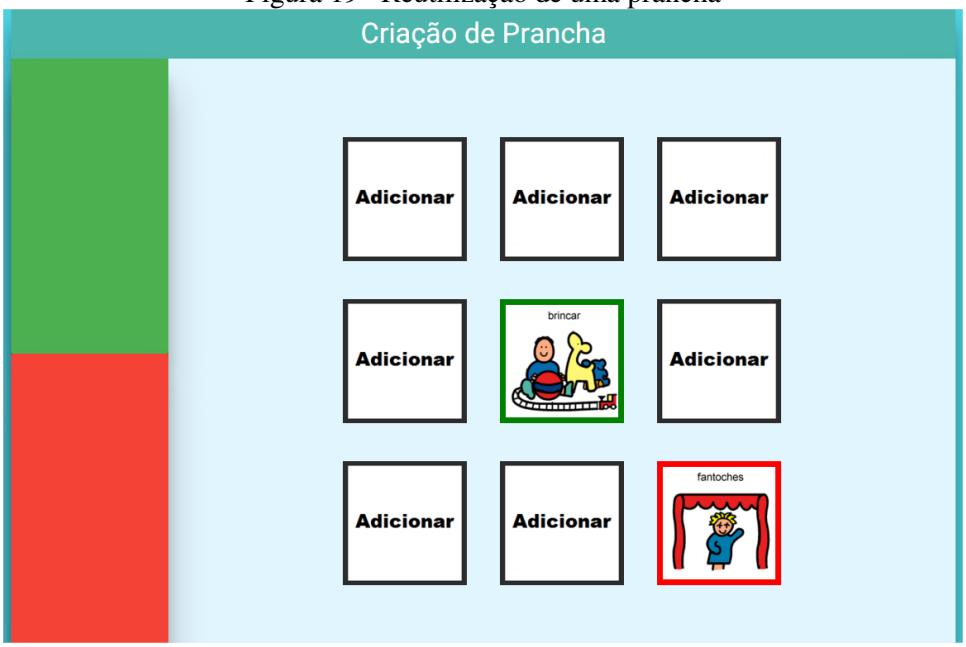


Figura 19– Reutilização de uma prancha

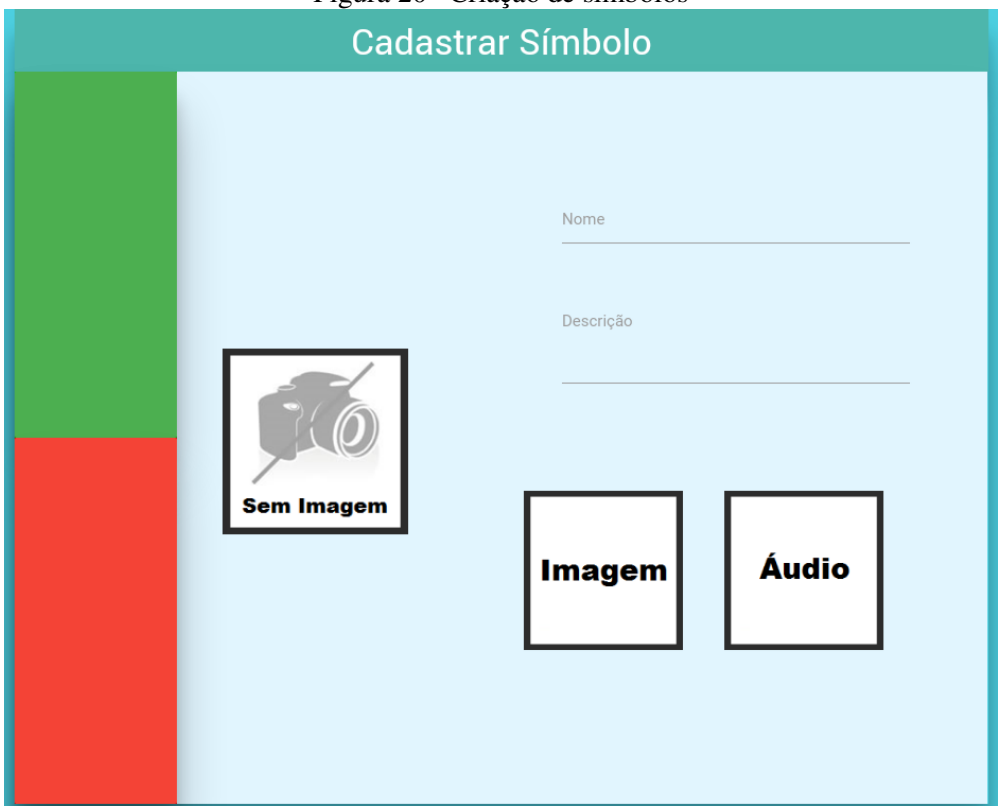


Após a escolha da prancha modelo o usuário poderá alterar os símbolos pré-carregados, ou apenas confirmar a criação da prancha e selecionar um plano para vinculá-la. Segue o mesmo processo de criação de uma prancha.

3.3.2.5 Criação de símbolos

Para criar uma nova prancha é necessário escolher ao menos um símbolo, porém o usuário pode optar também por criar um novo símbolo. Um símbolo contém as seguintes informações, nome, descrição, imagem e áudio, conforme pode ser visto na Figura 20.

Figura 20– Criação de símbolos



Nos dispositivos móveis o usuário pode selecionar a imagem que será vinculada ao símbolo através do álbum de fotos de seu dispositivo. Para testar o áudio selecionado pode ser pressionada a imagem e então o áudio será reproduzido.

3.3.2.6 Criação e visualização de observações

A criação de observações pode ser realizada apenas por usuários com o perfil `Tutor`, pois são direcionadas especificamente aos pacientes. Esta função também é acessada pela tela principal do `builder`, pressionando o símbolo `Observações`. Será apresentada uma tela onde poderão ser visualizadas as observações já cadastradas, conforme Figura 21, ou através do símbolo `Adicionar Obs`. será possível criar uma nova observação. Conforme mostra a Figura 22, a observação possui apenas duas informações, uma descrição e o texto da observação. Ela ficará registrada para o paciente que foi selecionado como `builder`.

Figura 21– Visualização das observações

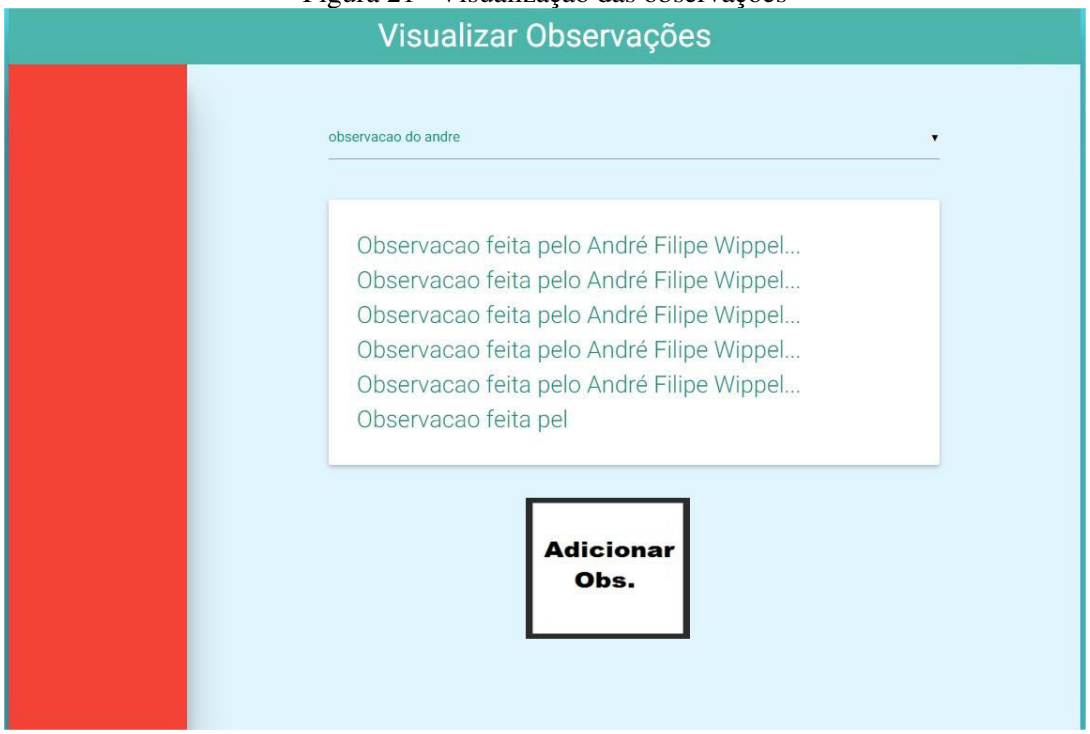
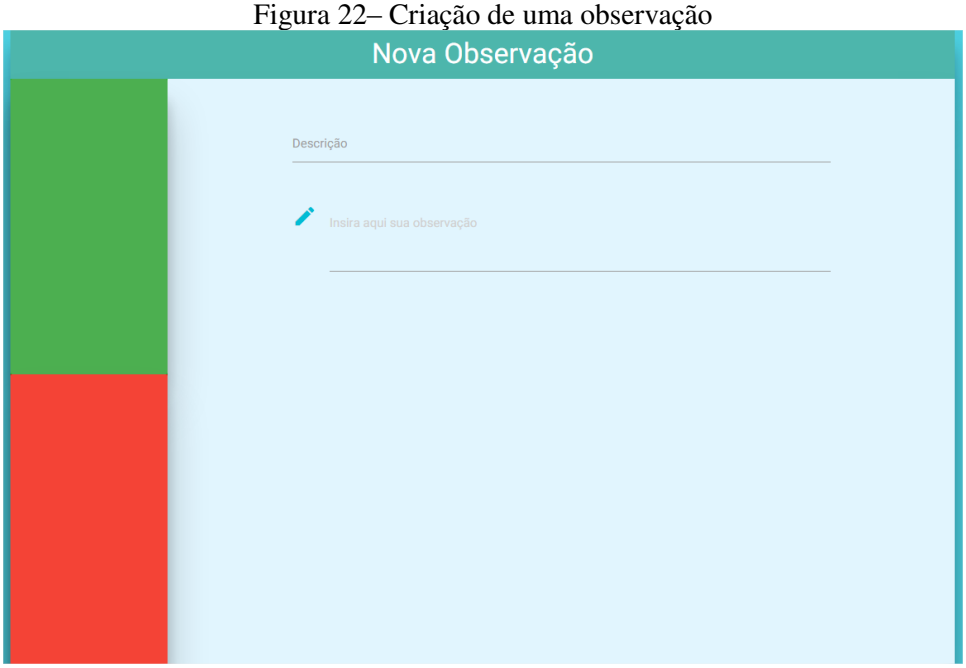


Figura 22– Criação de uma observação



3.3.2.7 Visualização dos históricos

Toda vez que um paciente utiliza uma prancha de comunicação, é gravada esta interação no seu histórico de atividades. Posteriormente, o usuário poderá consultar este histórico para fins didáticos. Esta consulta é realizada através do símbolo `Histórico Atividades`, na tela principal do `builder`, conforme a Figura 23.

Figura 23– Histórico de atividades

3.3.2.8 Utilização das pranchas

A principal atividade do Tagarela, a interação com as pranchas de comunicação, é realizada a partir da tela principal do `builder`. Primeiramente, deve-se selecionar um plano de atividades, objeto que agrupa pranchas de comunicação, para então selecionar a prancha desejada, conforme demonstrado na Figura 24. Será apresentada uma tela para o usuário, com os símbolos da prancha 3x3, conforme a Figura 25.

Figura 24– Seleção da prancha

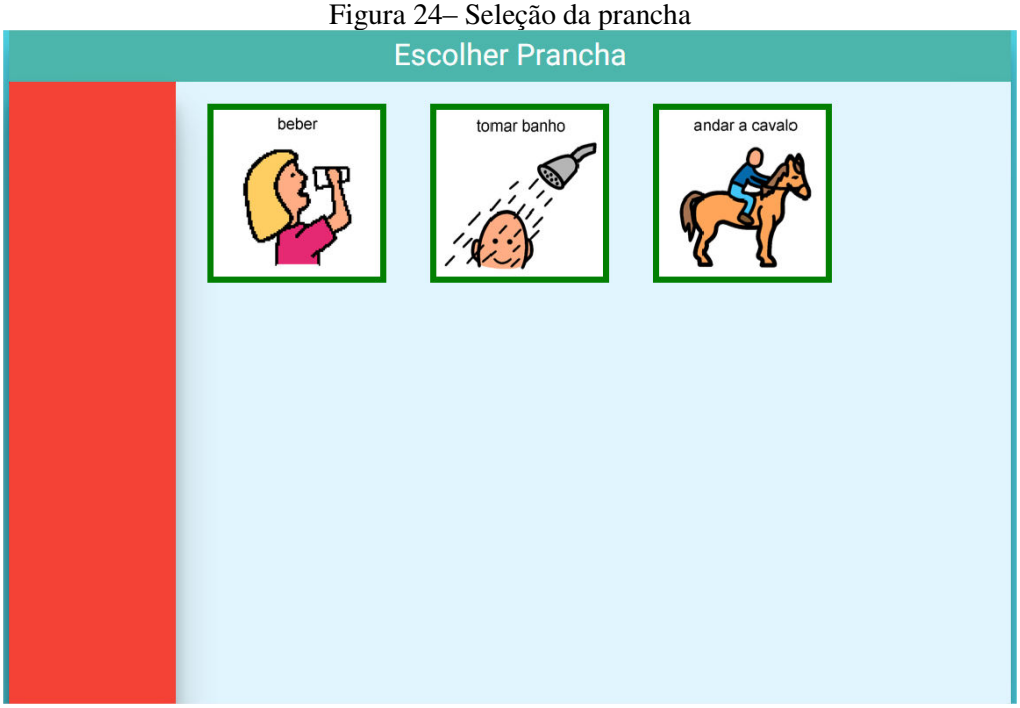
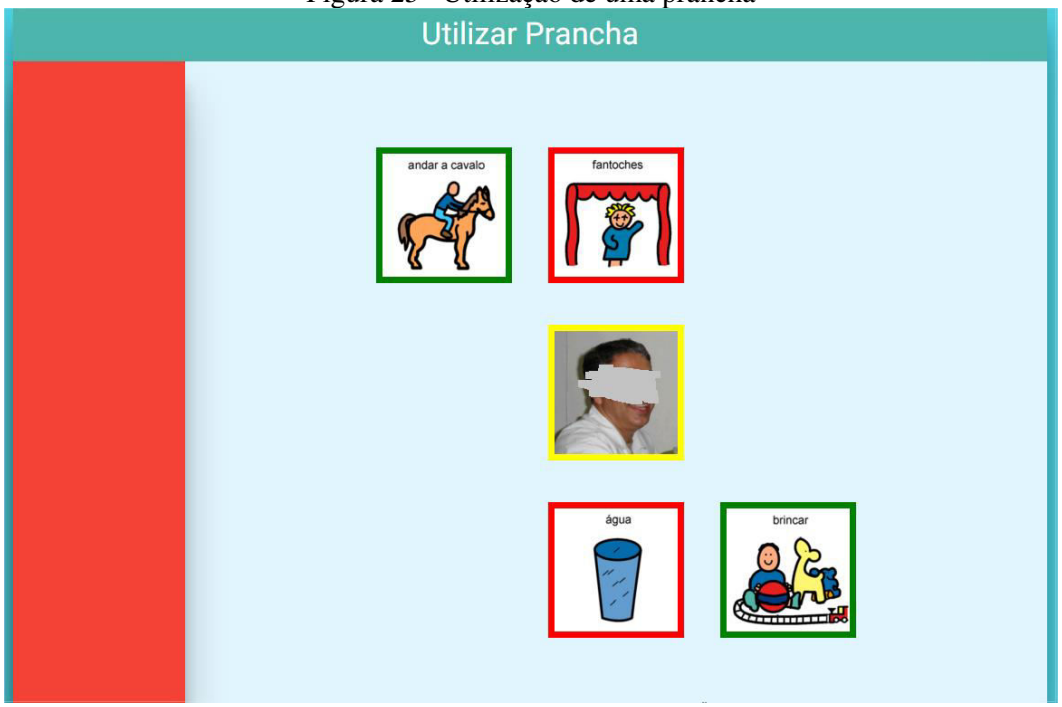


Figura 25– Utilização de uma prancha



Ao pressionar um símbolo, o áudio vinculado a ele será reproduzido. As posições da prancha que não possuírem símbolos ficarão em branco e não reproduzirão nenhum áudio ao serem pressionadas.

3.4 RESULTADOS E DISCUSSÕES

Os resultados obtidos pelo trabalho desenvolvido foram encontrados através de testes experimentais, baseando-se no desempenho da aplicação ao realizar as principais funcionalidades, principalmente no uso do *plugin* `media`, utilizado para reproduzir os áudios dentro da aplicação, e na criação de uma prancha. Foram realizados testes do aplicativo no dispositivo Android e também na versão *web* concluindo-se que houve alterações mínimas na interface gráfica entre estas duas versões, este teste tem por objetivo analisar a compatibilidade dos componentes gráficos utilizados pelo PhoneGap e também do *framework* Materialize utilizado na implementação.

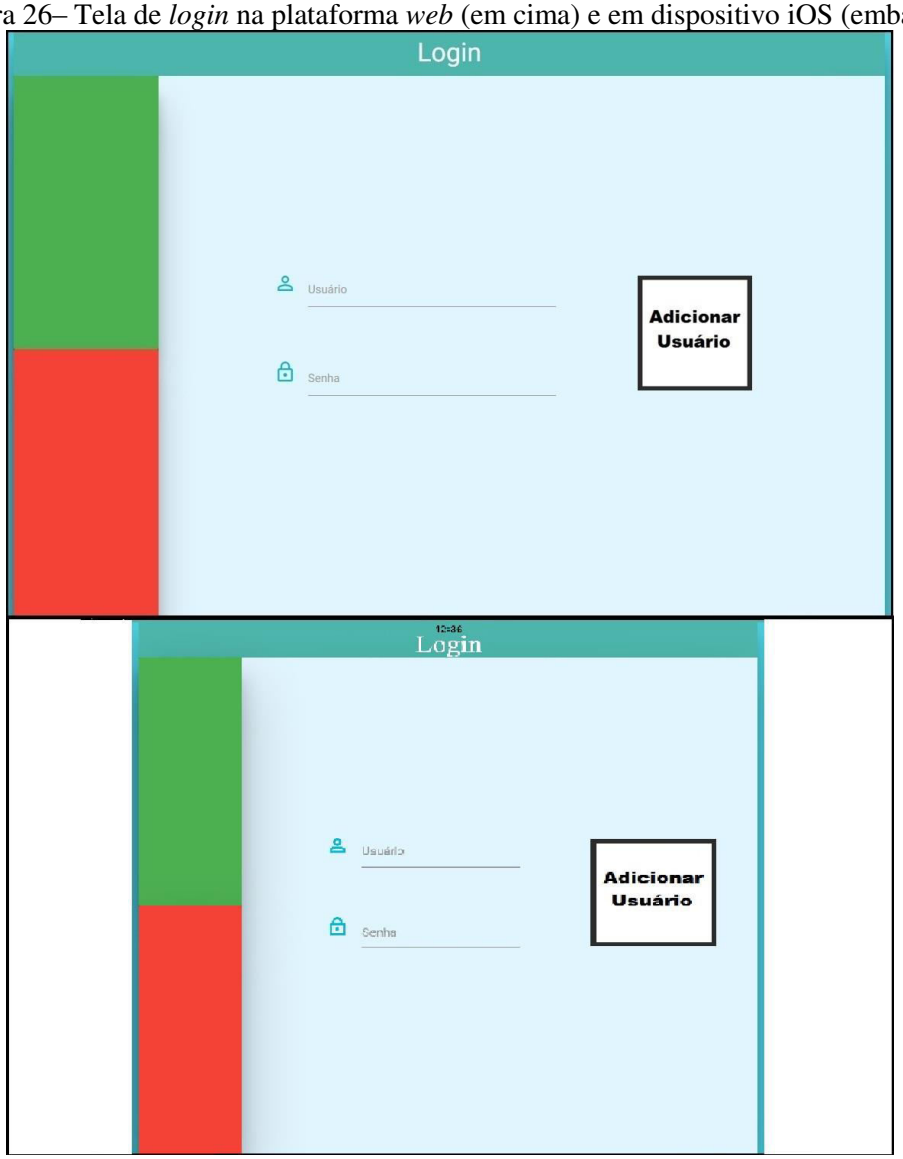
3.4.1 Interface gráfica

O desenvolvimento do aplicativo multiplataforma tinha por objetivo manter uma interface semelhante nas versões trabalhadas. A princípio as telas seguiram o mesmo padrão, tanto na versão *web* quanto na versão *mobile*, isto ocorreu principalmente devido ao uso do *framework* Materialize. Em relação a interação do usuário com a aplicação, também obteve-se semelhança entre as plataformas, com exceção apenas da tela de criação de símbolos, pois a

mesma na versão *mobile* utiliza os *plugins* `camera` e `media-capture` do PhoneGap, já na versão *web* é feito o uso do próprio componente nativo do Javascript para selecionar os arquivos de imagem e áudio.

É destacada nesta seção a comparação entre a interface gráfica da aplicação entre as plataformas *web* e *mobile*. A Figura 26 apresenta a comparação da tela de *login* da aplicação, entre a versão *web* (em cima) e a versão iOS (embaixo).

Figura 26– Tela de *login* na plataforma *web* (em cima) e em dispositivo iOS (embaixo)



Nos campos de preenchimento do usuário foi feito uso do recurso de ícones, que o próprio Materialize disponibiliza, porém na plataforma Android estes ícones não são carregados. Mesmo através de pesquisas realizadas não foi possível determinar a causa desta limitação. Os demais recursos utilizados pelo Materialize, como por exemplo o uso do conceito de *grids*, funcionam corretamente como pode ser visto na Figura 27, que apresenta a comparação da tela principal da aplicação, na qual o usuário possui acesso a diversas

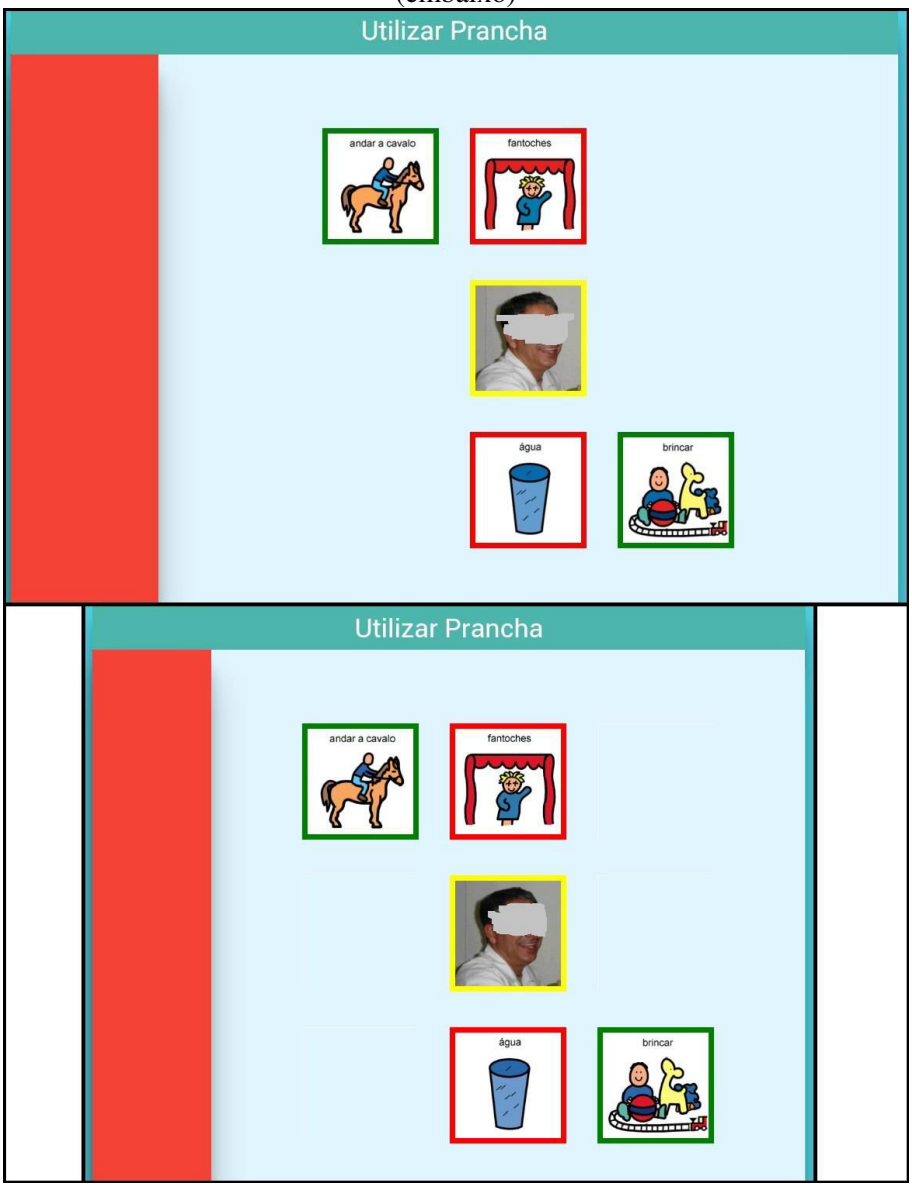
funcionalidades e também aos planos vinculados a ele, entre a versão *web* (em cima) e a versão Android 4.1.2 (embaixo).

Figura 27– Tela principal da aplicação na plataforma *web* (em cima) e em dispositivo Android 4.1.2 (embaixo)

A tela que permite a interação do usuário com uma prancha de comunicação também não apresentou diferenças em sua interface gráfica. No dispositivo móvel a reprodução do áudio foi realizada através do *plugin* `media` e também não apresentou comportamento diferente da aplicação na versão *web*, que utiliza o recurso de reprodução de áudio nativo do

Javascript. A Figura 28 apresenta a comparação desta tela entre a plataforma *web* (em cima) e a versão Android 4.1.2 (abaixo).

Figura 28– Tela de uso das pranchas na plataforma *web* (em cima) e em dispositivo Android 4.1.2 (embaixo)



Na Figura 29 é apresentada a comparação entre a tela de criação de símbolos na versão *web* (em cima) e na versão Android 4.1.2 (embaixo). A principal diferença entre as duas versões, é que na plataforma *web* o usuário seleciona o arquivo de áudio através do componente `<input type="file">`, nativo do HTML. Na versão de dispositivo móvel a escolha do áudio é realizada através da gravação de um novo áudio, recurso que foi implementado utilizando o *plugin* `media-capture` do PhoneGap. A figura Figura 30 demonstra a utilização deste recurso na versão Android 4.1.2.

Figura 29– Tela criação de símbolos na plataforma *web* (em cima) e em dispositivo Android 4.1.2 (embaixo)

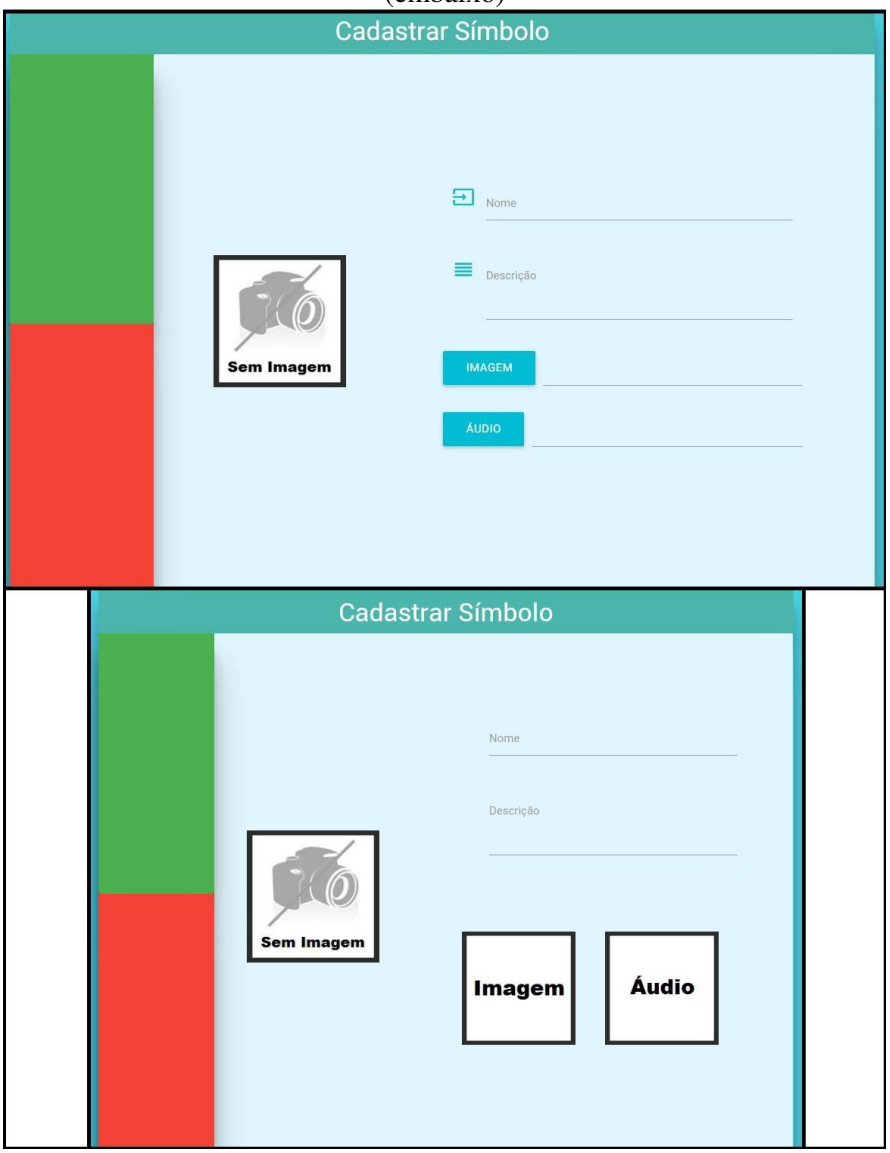
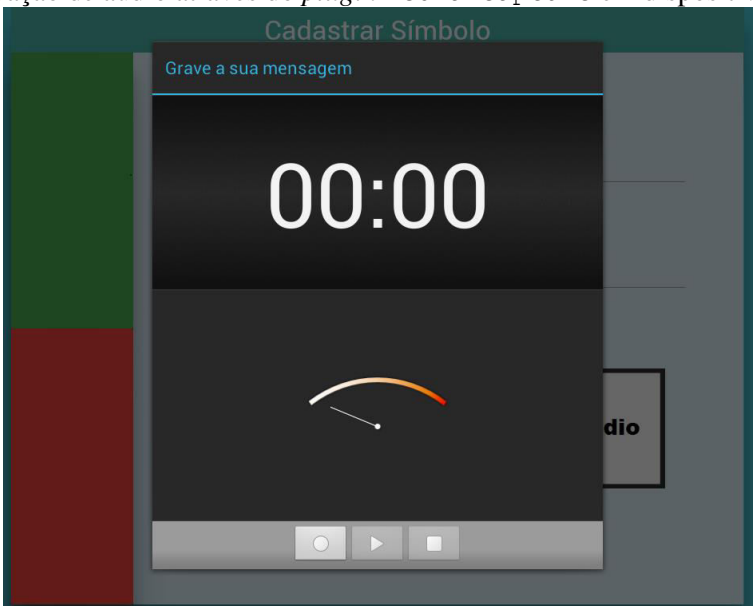


Figura 30– Gravação de áudio através do *plugin* `media-capture` em dispositivo Android 4.1.2



3.4.2 Teste de desempenho

Foi realizado um teste de consumo de memória da aplicação em dispositivo Android e também na plataforma *web*. Através deste teste procurou-se determinar o consumo de memória da aplicação, realizando um comparativo entre as duas versões. Os testes na versão Android foram realizados através do monitoramento da memória RAM no uso do aplicativo em um dispositivo Tablet Samsung Galaxy Tab 2 modelo GT-P5110 e o teste na versão *web* foi realizado no navegador Google Chrome, utilizando sua própria ferramenta Profiles, em um notebook VAIO modelo SVF14AA1QX. Os testes abordam o consumo de memória em *Megabytes* (MB) na abertura da aplicação, navegação entre as telas e reprodução de áudio na interação do usuário com suas pranchas.

Na abertura da aplicação o consumo de memória é aproximadamente igual nas duas plataformas, na versão Android a média de consumo inicial foi de 24.0 MB, enquanto na plataforma *web* a aplicação consumiu um total de 26.1 MB ao ser inicializada. Percebeu-se também na versão Android, que o consumo inicial de memória não varia de acordo com a conexão com a internet, porém com a base local vazia e ao ser realizada a carga inicial das informações com o servidor, o consumo de memória sobe momentaneamente para 35 MB e após isso se estabiliza novamente em 24 MB.

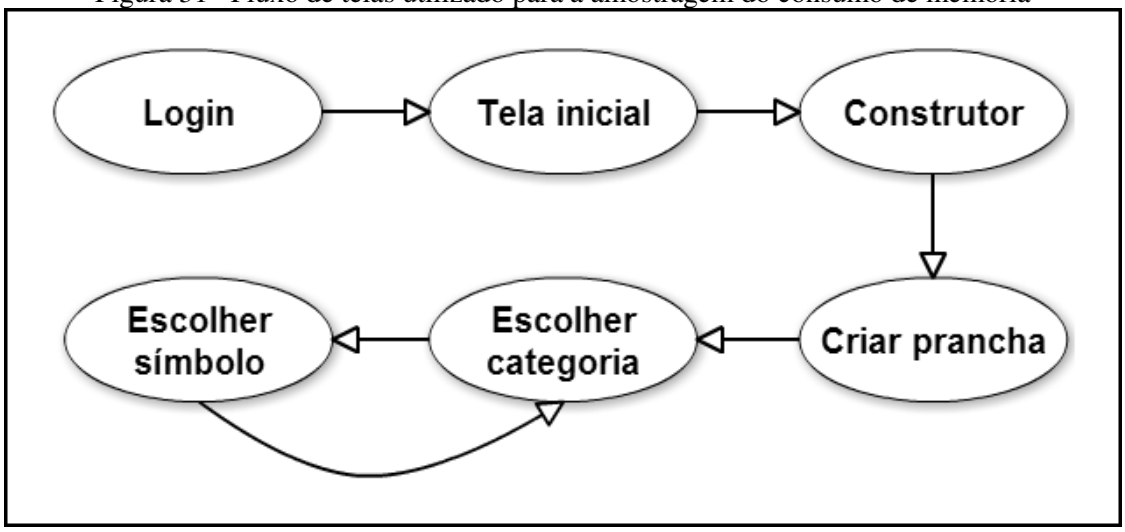
Foi percebido que ao navegar entre as telas da aplicação a memória é sempre liberada e consumida novamente na próxima tela. Percebeu-se que há um comportamento onde ao realizar esta transição de tela a memória consumida aumenta em torno de 1 MB para cada símbolo carregado, conforme demonstra a Tabela 1. Porém, logo após o consumo esta memória adicional é liberada, o que faz a aplicação consumir 24 MB na maioria das telas.

Tabela 1 – Quantidade de símbolos carregados e seus respectivos valores de consumo de memória

quantidade de símbolos na tela	memória consumida (MB)
2	26
4	29
6	31
8	33
10	36
12	40

Esta amostragem foi realizada seguindo um fluxo específico de telas, conforme mostrado na Figura 31, desde a tela de *login* da aplicação até a escolha de um símbolo para adicionar a uma nova prancha.

Figura 31– Fluxo de telas utilizado para a amostragem do consumo de memória



Em relação a tela de interação do usuário com as pranchas de comunicação pôde-se notar um aumento no consumo de memória ao reproduzir os áudios vinculados aos símbolos, sendo que enquanto o usuário não saísse daquela tela a memória consumida não era liberada. A tabela contendo os resultados do consumo de memória para cada reprodução de áudio é apresentada na Tabela 2.

Tabela 2 – Quantidade de áudios reproduzidos e seus respectivos valores de consumo de memória

quantidade de áudios reproduzidos	memória consumida (MB)
0	24
3	26
6	29
9	31
12	33
15	36

Considerando que ao entrar na tela de utilização da prancha a aplicação consome 24 MB de memória, e após a reprodução de 15 áudios a memória consumida aumentou em 12 MB. Divide-se esta quantidade por 15 e chega-se a uma estimativa de aumento de memória de 0.8 MB para cada áudio reproduzido. Através dessa conclusão obtém-se a fórmula Cm = 24.0 + Qa x 0.8, onde Cm representa o consumo de memória e Qa representa a quantidade de áudios que foram reproduzidos até o momento. Este mesmo teste foi realizado também na versão *web,* porém foi percebido que a memória consumida não ficava maior a cada reprodução de áudio. Chegou-se, portanto, a conclusão de que o *plugin* `media`, diferente do recurso de reprodução de áudio nativo do Javascript, não libera a memória utilizada para cada execução.

3.4.3 Resultados obtidos

O presente trabalho teve como objetivo inicial o desenvolvimento de uma aplicação de comunicação alternativa multiplataforma, visando migrar o projeto Tagarela para o *framework* PhoneGap, desta forma, suas futuras funcionalidades poderão ser implementadas utilizando apenas recursos de linguagem *web* e ser distribuídas para as versões Android e iOS sem a necessidade de conhecimento das linguagens Java e Objective-C.

Por se tratar de uma aplicação multiplataforma, houve a preocupação de permitir que o usuário utilizasse a aplicação em seu dispositivo móvel mesmo sem conexão com a internet, para isso todos os dados são persistidos localmente. Esta persistência local é realizada através da API WebSQL, da *World Wide Web Consortium* (W3C). Apesar de atender a necessidade da aplicação, percebeu-se durante o desenvolvimento deste trabalho uma certa dificuldade para trabalhar com esta API, pois a mesma utiliza transações assíncronas e possui algumas instabilidades. Em pesquisas realizadas posteriormente foi visto que isso se dá devido a esta tecnologia ter sido descontinuada, pois a maioria dos navegadores utiliza um único *backend*, baseado no SQLite e por este motivo o requisito de múltiplas implementações da W3C não estava sendo atendido, o que levou o consórcio a descontinuar o seu desenvolvimento.

Quando há conexão com a internet, é realizada a integração dos dados do servidor *web* com a base de dados local dos dispositivos móveis, esta integração foi realizada através de uma rotina que envia ao servidor os últimos registros que foram inseridos ou alterados na base local e que ainda não estão no banco de dados do servidor. Após isso é apagado todos os registros das tabelas locais e realizada a inclusão dos registros existentes no servidor. Esta rotina mostrou-se eficaz quando trata da inserção de novos dados, porém a alteração de dados já existentes pode ocasionar problemas de sincronismo de dados, o que talvez seria resolvido com a utilização de um *timestamp* para estes registros. Para realizar o *download* e *upload* dos arquivos de imagem e áudio que estão armazenados no servidor *web* e no dispositivo móvel, respectivamente, o *plugin* `file-transfer` que se mostrou bastante eficaz. Assim como os demais *plugins* do PhoneGap que foram utilizados no desenvolvimento desta aplicação.

Havia sido proposto que a aplicação permitisse a criação de usuários com perfil de especialista, tutor ou paciente, visando tornar a experiência do usuário prática e funcional. Porém, foi verificado que não se faz necessário o perfil de especialista, uma vez que o perfil de tutor dentro da aplicação contempla as mesmas funcionalidades e gerenciamento de informações que o especialista teria acesso. Manteve-se, no entanto, o perfil de paciente atendendo ao requisito da existência de perfis de usuários distintos dentro da aplicação.

3.4.4 Comparativo entre os trabalhos correlatos

O Quadro 11 apresenta de forma comparativa algumas características em relação aos trabalhos correlatos e o trabalho apresentado nesta proposta.

Quadro 11 – Características dos trabalhos correlatos e o presente

Características	Sono Flex (2014)	HelpTalk (2014)	Tabalho Presente
tratamento diferenciado para perfis distintos de usuário	não	não	sim
utiliza o conceito de pranchas de comunicação na apresentação das informações	sim	sim	sim
suporte a mais de duas plataformas	não	não	sim
reprodução de áudio encadeada	sim	sim	não
permite o uso do aplicativo no modo *offline*	não	sim	sim
Reprodução de áudio através de texto	sim	sim	não

Em relação ao trabalho presente, pode-se perceber através do Quadro 11 que os trabalhos correlatos possuem a funcionalidade extra de reprodução de áudio através de palavras digitadas pelo usuário, além de possuírem a possibilidade da reprodução de áudio encadeada. Porém, o trabalho desenvolvido se mostrou mais interessante em relação a diversidade de plataformas que pode suportar. Através do uso do *framework* Phonegap a aplicação desenvolvida pode ser facilmente disponibilizada para diversas plataformas, permitindo que o usuário tenha uma maior liberdade de escolha da plataforma que gostaria de utilizar a aplicação.

Além disso, o trabalho presente permite ao usuário a criação de um perfil específico para utilizar a aplicação, podendo escolher entre tutor ou paciente. Isto facilita o uso da aplicação em um ambiente de aprendizado, onde existe o papel de tutor/especialista e o papel de paciente.

4 CONCLUSÕES

O objetivo principal de aprimorar a aplicação Tagarela desenvolvida por Marco (2014) e migrar sua implementação para um *framework* de desenvolvimento único foi atendido, através da utilização do PhoneGap que permite a codificação de novas funcionalidades da aplicação Tagarela em um ambiente de desenvolvimento integrado. A aplicação desenvolvida se mostrou eficaz no que diz respeito à criação e utilização de pranchas de comunicação, permitindo também a criação de perfis de usuário distintos para atender os papéis existentes no cenário de aprendizado e comunicação alternativa. Primeiramente a aplicação iria permitir a criação de três perfis, o perfil de especialista, tutor e paciente, porém foi percebido ao longo deste trabalho que os perfis de especialista e tutor poderiam ser agrupados em um único perfil, já que possuem o mesmo papel dentro da aplicação. Portanto, optou-se por disponibilizar aos usuários a criação de dois perfis distintos, o tutor e o paciente.

Um dos objetivos deste trabalho era criar uma interface mais acessível para os usuários, já que em sua maioria serão pessoas com alguma dificuldade física ou mental. Este objetivo foi atingido através do uso do conceito de pranchas de comunicação, onde as informações de determinadas telas são apresentadas ao usuário apenas através de símbolos, evitando ao máximo o uso textual e também mantendo um padrão de interface intuitiva. No entanto, não foi possível se desprender totalmente do uso textual, pois caso contrário algumas informações não poderiam ser claramente repassadas aos usuários.

As ferramentas utilizadas no desenvolvimento, de forma geral, se mostraram adequadas para a criação de uma aplicação de comunicação alternativa multiplataforma. O uso do PhoneGap foi efetivo para a aplicação obter um comportamento equivalente nas plataformas trabalhadas. Apesar de ter sido encontrada algumas dificuldades na utilização da tecnologia WebSQL para a realização da persistência local nos dispositivos móveis, pois a mesma está descontinuada, os demais *plugins* do PhoneGap se mostraram eficazes para realizar o que propõem. Dentre eles pode ser citado o *plugin* `file-transfer`, que atende a necessidade da aplicação de realizar a atualização dos arquivos de aúdio e imagem entre o dispositivo local e o servidor.

As ferramentas utilizadas pela aplicação servidora também se mostraram efetivas através da utilização do servidor OpenShift a aplicação *web* e o banco de dados MySQL puderam ser armazenadas na nuvem e disponibilizadas através da criação de um domínio próprio, gratuito e de maneira geral, robusto. Os *scripts* PHP também foram disponibilizados

no servidor OpenShift e não houve situações onde o servidor estava indisponível ou instável durante o desenvolvimento do trabalho.

Acredita-se que as principais contribuições deste trabalho são disponibilizar uma aplicação de comunicação alternativa acessível e multiplataforma, que permita a criação e interação do usuário com pranchas de comunicação, junto à criação de perfis distintos para que a aplicação melhor se adeque a necessidade do usuário e que possa ser desenvolvida e aperfeiçoada através de um ambiente de desenvolvimento único e integrado. Suas principais limitações estão na capacidade de reprodução de áudio encadeada, uso de texto puro em algumas telas da aplicação e na impossibilidade de capturar uma imagem da câmera para criar um símbolo.

4.1 EXTENSÕES

São sugeridas as seguintes extensões para a continuidade do trabalho:

a) implementar a persistência dos dados locais no dispositivo utilizando uma outra técnica de armazenamento, sugere-se a utilização do banco de dados SQLite;
b) aperfeiçoar a rotina de consistência dos dados da base local, de modo que as operações de alteração (*update*) possam ser sincronizadas corretamente;
c) utilizar técnicas de desenvolvimento *web* mais recentes e aprimoradas para a aplicação *web* que vão além do uso de JQuery e AJAX, desde que sejam suportadas pelo PhoneGap, a fim de manter ao máximo a equivalência de código para todas as plataformas;
d) utilizar o *plugin camera* do PhoneGap para possibilitar a captura de imagens através da câmera do dispositivo no momento da criação de um símbolo;
e) implementar a reprodução de áudio encadeada nas pranchas de comunicação;
f) utilizar uma miniatura (*tumbnail*) da própria prancha como imagem para as pranchas de comunicação;
g) estender a aplicação para a plataforma Windows Phone, utilizando o PhoneGap;
h) tornar a interface da aplicação ainda mais acessível, retirando ao máximo os componentes de texto, sem afetar o comportamento da aplicação;
i) adicionar a descrição da observação a data e hora de quando a mesma foi criada e na tela de visualização das observações, ordená-las através da data e hora;
j) implementar os arquivos Javascript para serem carregados dinamicamente, conforme a plataforma onde a aplicação está sendo executada, desta forma pode ser utilizado exatamente o mesmo código para todas as plataformas.

REFERÊNCIAS

ADOBE. PhoneGap. San Jose, 2014. Disponível em: <http://phonegap.com>. Acesso em: 12 set. 2014.

CIVIAM. Sono Flex - Comunicação Alternativa para Ipad e Tablet Android. São Paulo, 2014. Disponível em: <http://www.civiam.com.br/civiam/index.php/necessidadesespeciais/sono-flex-comunicacao-alternativa-para-ipad-tablet-android.html#>. Acesso em: 06 set. 2014.

FABENI, Alan Filipe C. Tagarela: Aplicativo para Comunicação Alternativa no iOS. 2012. 107 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

FOTON. Evolução da Comunicação Humana e dos Meios de Comunicação. Campinas, 2008. Disponível em: <http://www.foton.com.br/empresa.php?id=informacoes&sid= dados_administrativos>. Acesso em: 12 set. 2014.

GONÇALVES, Maria de J. Comunicação alternativa na fonoaudiologia: uma área em expansão. São Paulo, 2008. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S1516-18462008000300002>. Acesso em: 12 set. 2014.

HELPTALK. HelpTalk. Guimarães, 2014. Disponível em <http://www.helptalk.mobi/pt>. Acesso em: 11 set. 2014.

MARCO, Darlan D. de. Tagarela: Aplicativo de Comunicação Alternativa na Plataforma Android. 2014. 94 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

MATERIALIZE. Sobre o Materialize. Pittsburgh, 2015. Disponível em: <http://materializecss.com/about.html>. Acesso em: 09 nov. 2015.

PELOSI, Miryam. Recursos na CAA. Rio de Janeiro, 2011. Disponível em: <http://www.comunicacaoalternativa.com.br/recursos-na-caa>. Acesso em: 12 set. 2014.

SARTORETTO, Mara L.; BERSCH, Rita. O que é comunicação alternativa? Porto Alegre, 2014. Disponível em: <http://www.assistiva.com.br/ca.html>. Acesso em: 12 set. 2014.

TAGARELA. Tagarela: rede social de comunicação alternativa. Blumenau, 2014. Disponível em <http://gcg.inf.furb.br/tagarela>. Acesso em: 11 set. 2014.

TEIXEIRA, Milene. A importância da fala. Primavera do Leste, 2013. Disponível em: <http://www.jornalodiario.com.br/TNX/conteudo.php?sid=204&cid=30649>. Acesso em: 12 set. 2014.

APÊNDICE A – Descrição dos casos de uso, condições, cenários e exceções

Os casos de uso da aplicação Tagarela estão descritos a seguir:

a) UC01 – Aceitar convites, este caso de uso descreve como o `Tutor` aceita os convites enviados por outros usuários.
b) UC02 – Enviar convites, este caso de uso descreve como o `Paciente` envia convites para outros usuários que possuem o perfil `Tutor`.
c) UC03 – Alterar dados pessoais, este caso de uso descreve como o usuário pode alterar seus dados pessoais dentro da aplicação.
d) UC04 – Usar prancha, este caso de uso descreve como o usuário interage com suas pranchas de comunicação.
e) UC05 – Criar prancha, este caso de uso descreve como o usuário cria uma nova prancha de comunicação.
f) UC06 – Reutilizar prancha, este caso de uso descreve como o usuário pode reutilizar uma prancha já existente de outro usuário.
g) UC07 – Visualizar observações, este caso de uso descreve como o usuário pode visualizar suas observações.
h) UC08 – Criar observação, este caso de uso descreve como o `Tutor` cria uma nova observação.
a) UC09 – Visualizar históricos, este caso de uso descreve como o usuário visualiza seu histórico de uso de pranchas.

Dos quadros 12 a 20 a seguir, são demonstradas as condições, cenários e exceções de cada caso de uso.

Quadro 12 – Caso de uso UC01

UC01 – Aceitar convites	
Requisitos atendidos	RF2
Pré-condições	1. O usuário deve ter o perfil `Tutor`. 2. Algum usuário com o perfil `Paciente` precisa ter enviado um convite para este usuário (UC02).
Cenário principal	1. São apresentados para o usuário todos os `Pacientes` que enviaram convites para ele. 2. O usuário seleciona de qual `Paciente` deseja aceitar o convite e visualiza sua mensagem. 3. O usuário toca no botão `Confirmar` para aceitar o convite.
Exceção	Se o usuário não possuir nenhum convite é apresentada a mensagem `Não há convites pendentes`.
Pós-condição	O `Paciente` ficará vinculado ao usuário que aceitou o convite, permitindo então realizar outras operações no aplicativo.

Quadro 13 – Caso de uso UC02

UC02 – Enviar convites	
Requisitos atendidos	RF2
Pré-condições	1. O usuário precisa ter perfil de `Paciente`.
Cenário principal	1. O `Paciente` escolhe para qual usuário deseja enviar o convite, podendo também escrever uma mensagem para ele. 2. O usuário toca no botão `Confirmar` para enviar o convite.
Exceção	Não existindo nenhum usuário com perfil `Tutor` cadastrado, o convite não será criado.
Pós-condição	O `Tutor` escolhido pelo `Paciente` receberá o convite junto com a mensagem digitada.

Quadro 14 – Caso de uso UC03

UC03 – Alterar dados pessoais	
Requisitos atendidos	RF3
Pré-condições	Nenhuma.
Cenário principal	1. O usuário toca em sua foto de perfil. 2. Após visualizar suas informações pessoais, ele seleciona o botão `Alterar`. 3. Será mostrada uma tela onde o usuário poderá alterar seu nome, telefone, local, função e símbolo.
Exceção	Não há.
Pós-condição	O usuário terá seus novos dados pessoais gravados.

Quadro 15 – Caso de uso UC04

UC04 – Usar prancha	
Requisitos atendidos	RF5
Pré-condições	1. O usuário deve possuir ao menos uma prancha de comunicação.
Cenário principal	2. O usuário seleciona o plano no qual a prancha desejada está vinculada. 3. Seleciona a prancha. 4. Serão mostrados os símbolos da prancha selecionada, no formato 3x3, nesta tela o usuário poderá interagir com os

	símbolos.
Exceção	Não há.
Pós-condição	Será gravado no histórico de atividades que o usuário interagiu com a prancha selecionada.

Quadro 16 – Caso de uso UC05

UC05 – Criar prancha	
Requisitos atendidos	RF4
Pré-condições	Nenhuma.
Cenário principal	1. O usuário toca no símbolo `Criar Prancha`. 2. É apresentada a ele uma prancha 3x3, sem símbolos. O usuário então insere os símbolos que desejar tocando no símbolo `Adicionar`. 3. Após adicionar os símbolos desejados, pressiona o botão `Confirmar`. 4. Escolhe um símbolo para a prancha. 5. Escolhe um plano já existente ou cria um novo plano, para vincular a prancha. 6. Se desejar criar um novo plano, deve selecionar um símbolo para este plano, assim como foi feito para a prancha no passo 4. Retorna então para o passo 5.
Exceção	Caso o usuário não selecionou nenhum símbolo para a prancha, no passo 3 o botão `Confirmar` não ficará habilitado, impossibilitando o usuário de criar a prancha.
Pós-condição	Será criada a nova prancha e ela ficará vinculada ao plano selecionado no passo 5.

Quadro 17 – Caso de uso UC06

UC06 – Reutilizar prancha	
Requisitos atendidos	RF4
Pré-condições	1. Deve existir ao menos uma prancha criada na base.
Cenário principal	1. O usuário toca no símbolo `Compartilhar Prancha`. 2. O usuário deve escolher reutilizar uma prancha sua ou de outro usuário. 3. Conforme o filtro acima, são apresentados todos os planos do próprio usuário ou dos outros usuários da base. O usuário seleciona um plano. 4. São apresentadas todas as pranchas do plano selecionado. O usuário seleciona uma prancha. 5. É apresentada a ele a prancha selecionada, sem os símbolos da categoria `Pessoa`. O usuário então altera os símbolos que desejar. 6. Após obter os símbolos desejados, pressiona o botão `Confirmar`. 7. Escolhe um símbolo para a prancha. 8. Escolhe um plano já existente ou cria um novo plano, para vincular a prancha. 9. Se desejar criar um novo plano, deve selecionar um símbolo para este plano, assim como foi feito para a prancha no passo 7. Retorna então para o passo 8.
Exceção	Caso a prancha não possuir nenhum símbolo, no passo 6 o botão `Confirmar` não ficará habilitado, impossibilitando o usuário de criar a nova prancha.

Pós-condição	Será criada a nova prancha e ela ficará vinculada ao plano selecionado no passo 8.

Quadro 18 – Caso de uso UC07

UC07 – Visualizar observações	
Requisitos atendidos	RF6
Pré-condições	1. Deve existir ao menos uma observação criada para o usuário.
Cenário principal	1. O usuário toca no símbolo `Observações`. 2. Seleciona uma observação através da sua descrição. 3. O texto da observação selecionada será mostrado logo abaixo da descrição.
Exceção	Não há.
Pós-condição	A observação é visualizada pelo usuário.

Quadro 19 – Caso de uso UC08

UC08 – Criar observação	
Requisitos atendidos	RF6
Pré-condições	1. O usuário deve possuir o perfil `Tutor`.
Cenário principal	1. O usuário toca no símbolo `Observações`. 2. Toca no símbolo `Adicionar Obs`. 3. Digita uma descrição para a nova observação, um texto e toca no botão `Confirmar`.
Exceção	Não há.
Pós-condição	A nova observação é criada.

Quadro 20 – Caso de uso UC09

UC09 – Visualizar históricos	
Requisitos atendidos	RF7
Pré-condições	Nenhuma.
Cenário principal	1. O usuário toca no símbolo `Histórico Atividades`. 2. Será apresentada uma tela com todos os históricos de uso das pranchas de comunicação. No formato "[`data - hora`] - `prancha utilizada`".
Exceção	Não há.
Pós-condição	O histórico das pranchas utilizadas é visualizado pelo usuário.