UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE SISTEMAS DE INFORMAÇÃO – BACHARELADO

TAGARELA: APLICATIVO PARA COMUNICAÇÃO ALTERNATIVA USANDO ARQUITETURA BASEADA EM COMPONENTES

RAFAEL DE MOURA PACHECO

BLUMENAU
2019

RAFAEL DE MOURA PACHECO

TAGARELA: APLICATIVO PARA COMUNICAÇÃO ALTERNATIVA USANDO ARQUITETURA BASEADA EM COMPONENTES

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

Prof. Dalton Solano dos Reis, Mestre – Orientador

BLUMENAU
2019

TAGARELA: APLICATIVO PARA COMUNICAÇÃO ALTERNATIVA USANDO ARQUITETURA BASEADA EM COMPONENTES

Por

RAFAEL DE MOURA PACHECO

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(a). Dalton Solano dos Reis – Orientador(a), FURB
Membro:	Prof(a). Joyce Martins – FURB
Membro:	Prof(a). Simone Erbs da Costa – FURB

Blumenau, 10 de dezembro de 2019

Dedico este trabalho a todas as pessoas da minha família que sempre me deram todo o apoio necessário para a conclusão dele.

AGRADECIMENTOS

À minha família, a que sou grato por tudo que fizeram por mim, por todo seu apoio e suporte quando fosse necessário.

À minha namorada Gislaine, por estar sempre ao meu lado, e me apoiar e incentivar em qualquer momento de necessidade.

E a todos que de alguma forma contribuíram para a conclusão deste trabalho e na jornada até a chegada dele.

Ninguém baterá tão forte quanto a vida. Porém, não se trata de quão forte pode bater, se trata de quão forte pode ser atingido e continuar seguindo em frente. É assim que a vitória é conquistada.

Rocky Balboa

RESUMO

Este trabalho apresenta o levantamento de informações, especificação, desenvolvimento e operacionalidade de um aplicativo multiplataforma para comunicação alternativa. O objetivo principal deste trabalho é reconstruir partes de um aplicativo existente de maneira que facilitará futuras expansões no código utilizando uma arquitetura de componentes. O servidor foi desenvolvido utilizando a linguagem de programação JavaScript, mais especificamente Node.js. Para armazenamento de dados foi utilizando o banco de dados MongoDB. O aplicativo foi desenvolvido utilizando Ionic, *framework* responsável por gerar aplicativos Android e iOS. A ideia para o desenvolvimento deste trabalho foi motivada pelo fato de existirem seis aplicativos diferentes, sem qualquer ligação entre eles, com intuito de fornecer um ambiente no qual tutor e paciente possam interagir de forma que haja uma evolução na capacidade de comunicação do paciente. O levantamento de informações foi realizado por meio das experiências do orientador do projeto com os outros seis aplicativos já desenvolvidos. Foram utilizados diagramas da Unified Modeling Language (UML) para representar estruturas e fluxos desenvolvidos. A implementação do trabalho foi realizada utilizando as ferramentas VisualStudioCode, Heroku Cloud Platform, mLab e Amazon S3, todo seu código fonte foi armazenado no website Github. Por fim, conclui-se vantagens oferecidas pelo desenvolvimento deste trabalho, bem como foram levantadas melhorias futuras e extensões para este trabalho.

Palavras-chave: Comunicação alternativa. Arquitetura de componentes. Multiplataforma.

ABSTRACT

This work presents the information gathering, specification, development and operability of a multiplatform application for alternative communication. The main purpose of this paper is to reconstruct parts of an existing application in a way that will facilitate future code expansions using a component architecture. The server was developed using the JavaScript programming language, more specifically Node.js. For data storage was using MongoDB database. The application was developed using Ionic, framework responsible for generating Android and iOS applications. The idea for the development of this work was motivated by the fact that there are six different applications, without any connection between them, in order to provide an environment in which tutor and patient can interact so that there is an evolution in the patient's communication skills. Information gathering was carried out through the project advisor's experiences with the other six applications already developed. Unified Modeling Language (UML) diagrams were used to represent developed structures and flows. The implementation of the work was performed using the tools VisualStudioCode, Heroku Cloud Platform, mLab and Amazon S3, all their source code was stored on the Github website. Finally, the advantages offered by the development of this work are concluded, as well as future improvements and extensions to this work were raised.

Key-words: Alternative communication. Components architecture. Multiplatform.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API – Application Programming Interface

CA – Comunicação Alternativa

CLI – Command Line Tools

CRUD – Create, Read, Update and Delete

CSS – Cascading Style Sheets

DCU – Diagrama de Caso de Uso

HTML – HyperText Markup Language

HTTP – HyperText Transfer Protocol

JSON – JavaScript Object Notation

NPM - NodeJS package manager

RF – Requisito Funcional

RNF – Requisito Não-Funcional

SDK – Software Development Kits

SGBD – Sistema Gerenciados de Base de Dados

TA – Tecnologia Assistiva

TEA – Transtorno do Espectro Autista

UC – Use Case

UML – Unified Modeling Language

URI - Uniform Resource Identifier

SUMÁRIO

1 INTRODUÇÃO

No ano de 2015, dados do Instituto Brasileiro de Geografia e Estatística (2018) revelam que 6,2% da população brasileira possui algum tipo de deficiência e 0,8% deste percentual tem algum tipo de deficiência intelectual, na qual a maioria (0,5%) já nasceu com essas limitações. Ao analisar o total de pessoas com deficiência intelectual, 54,8% tem grau intenso ou muito intenso de limitação e apenas cerca de 30% frequentam algum tipo de instituição que ofereça serviços de reabilitação (VILLELA, 2015). Já no âmbito nacional, a população brasileira é composta por 45 milhões de brasileiros com algum tipo de deficiência, segundo o Instituto Brasileiro de Geografia e Estatística (IBGE) (OLIVEIRA, 2012), o Nordeste contendo a maior a proporção (26,63%) e a menor com o Sul (22,50%).

A comunicação é a principal forma de transmissão de conhecimentos, sendo um dos grandes desafios enfrentados pelas pessoas portadoras de necessidades especiais, ela pode comprometer a interação, aprendizado, compreensão do mundo e impossibilitando o compartilhamento de suas ideias e sentimentos (CARNIEL et al., 2018). É a comunicação que humaniza a criança pequena, que a coloca como ser humano em uma sociedade, que nomeia tudo o que existe e acontece à sua volta (TEIXEIRA, 2013). A dificuldade na comunicação tende a impedir as pessoas com deficiência manifestar suas necessidades e podendo ocasionar o afastamento do convívio em sociedade (CARNIEL et al., 2018). Não existe solução única para estimular a comunicação, mas para esses casos, mas uma alternativa é fazer uso de Tecnologia Assistiva (TA), que segundo Bersch (2007), auxilia na inclusão escolar de pessoas com alguns tipos de deficiência.

A TA é composta de ferramentas e serviços utilizado para pessoas com deficiência que facilitam, ou mesmo, tornam possível, o acesso ao computador de pessoas com diferentes tipos de limitação motora, comunicação e linguagem (GALVÃO FILHO, 2009). A Comunicação Alternativa (CA) é uma categoria da TA que visa dar assistência para deficiências de comunicação, buscando auxiliar pessoas com dificuldades de comunicação utilizando de recursos como as pranchas de comunicação, construídas com simbologia gráfica para expressar suas questões, desejos, sentimentos e entendimentos (BERSCH, 2017).

Assim como outras ferramentas de TA, o Tagarela, desenvolvido por acadêmicos da Universidade Regional de Blumenau, tem como objetivo fornecer uma alternativa para auxiliar no ensino de pessoas com necessidades especiais. O Tagarela, possui várias funcionalidades, entre elas, as que auxiliam na comunicação, na coordenação motora fina, aquisição de linguagens para crianças autistas, entre outros. Atualmente estas várias

funcionalidades estão divididas em seis aplicativos diferentes, sem qualquer ligação entre eles. Por exemplo, o trabalho mais recente da funcionalidade de prancha de comunicação desenvolvido por Wippel (2015), não interage com os aplicativos que desenvolveram as outras funcionalidades do Tagarela.

Diante do exposto, este projeto disponibiliza um novo aplicativo Tagarela, fornecendo uma nova interface para os usuários e disponibilizando um novo módulo de prancha de comunicação. E assim, e preparando-o para receber novos módulos integrados em apenas um aplicativo utilizando uma arquitetura baseada em componentes.

1.1 OBJETIVOS

O objetivo deste trabalho é desenvolver um aplicativo híbrido para ser utilizado em mais de uma plataforma e integrar os módulos do Tagarela utilizando uma arquitetura baseada em componentes.

Os objetivos específicos são:

a) criar uma interface para adicionar novos módulos;
b) criar uma interface para controlar a visibilidade dos módulos;
c) refatorar o módulo de prancha de comunicação utilizando está interface.

1.2 ESTRUTURA

Este trabalho está organizado em quatro capítulos.

O primeiro capítulo apresenta a introdução ao objeto de estudo, definição dos principais objetivos e apresentação da estrutura do trabalho.

No segundo capítulo é apresentada a fundamentação teórica utilizada durante o projeto. São abordados assuntos referentes a ferramenta atual do Tagarela, aplicativos híbridos e nativos, e arquitetura de componentes. Além disso relata-se alguns trabalhos correlatos utilizados como base para estudo de TA.

O capítulo três contempla as etapas de desenvolvimento do aplicativo, no qual são apresentados os requisitos, os diagramas para melhor entendimento, a implementação, demonstrando alguns quadros com código fonte, os resultados e discussões.

O quarto e último capítulo apresenta os resultados e as discussões. Por fim, as principais conclusões do trabalho desenvolvido são colocadas e as possíveis extensões para serem implementadas no futuro.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os conceitos e fundamentos mais importantes para a pesquisa em questão, estando organizado da seguinte forma: a seção 2.1 apresenta a ferramenta atual; na seção 2.2 é discutido sobre aplicativos nativos e híbridos, apresentando as vantagens de utilizar tecnologias híbridas para aplicativos móveis; a seção 2.3 apresenta as vantagens de uma arquitetura baseada em componentes; por fim, a seção 2.4 expõe os trabalhos correlatos estudados.

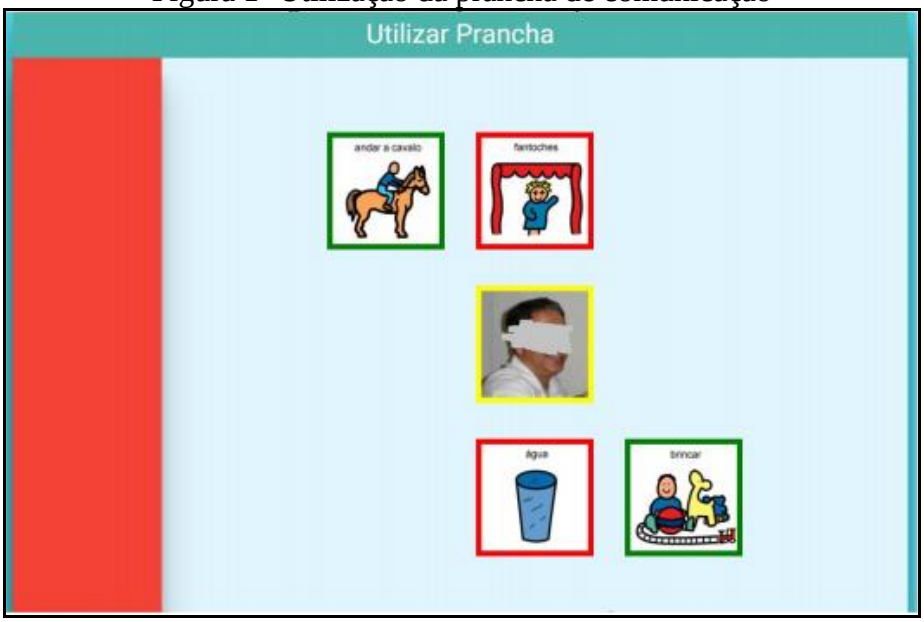
2.1 FERRAMENTA ATUAL

O Tagarela é um conjunto de aplicativos desenvolvido por alunos da Universidade Regional de Blumenau, tendo como objetivo fornecer uma alternativa para auxiliar no ensino de pessoas com necessidades especiais (TAGARELA, 2014). A primeira versão deste projeto foi desenvolvida por Fabeni (2012), um aplicativo para a plataforma iOS. Um dos principais objetivos deste aplicativo é disponibilizar um ambiente no qual o especialista possa trocar experiências com o usuário em conjunto com seu tutor montar um plano de atividade para estimular a capacidade de comunicação do usuário (TAGARELA, 2014).

Marco (2014) continuou o desenvolvimento do aplicativo Tagarela para a plataforma Android. Nesta etapa, o objetivo foi tornar a experiência do usuário no aplicativo interativa, utilizando pranchas de comunicação, além de possibilitar a sincronização das informações entre dispositivos e permitir o uso do aplicativo de forma off-line (MARCO, 2014).

Wippel (2015) teve como objetivo integrar o desenvolvimento das versões em um único *framework*, o PhoneGap. Esse *framework*, utiliza um ambiente de desenvolvimento integrado, no qual é realizado uma única codificação para aplicativos web e móvel (Android e iOS) (WIPPEL, 2015). As funcionalidades desenvolvidas para pranchas de comunicação nas versões anteriores foram disponibilizadas na versão atual do aplicativo, por meio de uma interface acessível (WIPPEL, 2015). Contudo, a versão desenvolvida por Wippel (2015) não integra os outros módulos do Tagarela, concentrando o desenvolvimento no módulo da prancha de comunicação conforme a Figura 1 (WIPPEL, 2015).

Figura 1- Utilização da prancha de comunicação



Fonte: Wippel (2015).

2.2 APLICATIVOS NATIVOS E HÍBRIDOS

Nos últimos anos, o uso de dispositivos móveis ao uso rotineiro cresceu muito (MAHATO, 2016). Com isto, aplicativos baseados em acesso à internet vêm aumentando muito na última década (MAHATO, 2016). Para acompanhar o avanço tecnológico, foram criadas ferramentas para desenvolvimento de aplicativos em plataformas móveis, visando atender tanto a forma de desenvolvimento nativo para o dispositivo quanto para o desenvolvimento focado em atender um número mais abrangente de plataformas mantendo compatibilidade (PAULINO, 2015).

Aplicativos construídos de forma nativa, são desenvolvidos para utilização em uma plataforma especifica, sendo capaz de explorar as potencialidades da plataforma na qual foi criado, conseguindo ter acesso a recursos do aparelho como leitor biométrico, lista de contatos, calendário, entre outros (MADUREIRA, 2017). Dentre as vantagens pode-se destacar o desenvolvimento nativo, possibilitando que o desenvolvedor consuma recursos do dispositivo (DEVICELAB, 2018). Além disto, o desenvolvimento nativo tem como objetivo alcançar um nível de performance superior aos oferecidos por aplicativos híbridos, justamente por ter acesso direto aos recursos da plataforma (VENTEU; PINTO, 2018). Isso proporciona um desempenho superior aos aplicativos nativos. Porém, o custo de desenvolvimento de um aplicativo nativo é superior ao desenvolvimento híbrido, visto que a equipe de desenvolvimento necessita ter um domínio sobre cada plataforma em que o aplicativo será desenvolvido, assim como as peculiaridades de seus dispositivos compatíveis (PAULINO, 2015).

Manter a experiência de usuário, identidade visual, compatibilidade de dependências de um aplicativo nativo, dado que existem muitos tipos de dispositivos e ambientes diferentes, se tornou um desafio para os desenvolvedores (MAHATO, 2016). Devido às dificuldades de um aplicativo ser desenvolvido de forma nativa, os aplicativos híbridos acabaram se tornando uma opção bem vista para o desenvolvimento de novos aplicativos nos últimos anos (PAULINO, 2015).

Aplicativos desenvolvidos de forma híbrida são concebidos em uma arquitetura Web, possuindo tecnologias como HyperText Markup Language (HTML), Cascading Style Sheets (CSS) e TypeScript, que são tecnologias comuns em desenvolvimento de sites (PAULINO, 2015). A utilização dessas tecnologias se torna possível em um dispositivo móvel pois há um navegador embutido dentro do próprio dispositivo, possibilitando a renderização do aplicativo em forma de site (PAULINO, 2015). Uma das formas de criar este site embutido dentro do dispositivo pode ser feita pela ferramenta Ionic. Com a utilização desta ferramenta, existe a possibilidade de utilização de componentes próprios da ferramenta que se adaptam de acordo com a plataforma do dispositivo, facilitando padronizar a experiência do usuário, a identidade visual e a compatibilidade de dependências (IONIC FRAMEWORK, 2018).

Mesmo o aplicativo híbrido sendo um site, ele também possui acesso aos recursos nativos do dispositivo, da mesma forma que um aplicativo nativo. Para que o acesso a recursos nativos dos dispositivos seja possível, por meio do site criado pelo Ionic, existe a ferramenta Cordova. Esta ferramenta possibilita a compilação de um código nativo, escrito em HTML5, CSS e TypeScript para diferentes plataformas híbridas utilizando a mesma base de código de origem. Desta forma, a ferramenta fica responsável por encapsular todas as funcionalidades nativas de cada plataforma (CORDOVA, 2018).

2.3 ARQUITETURA DE COMPONENTES

Com o passar dos anos, o processo de desenvolvimento de software exige cada vez mais produtividade e agilidade em construções de aplicativos de alta complexidade (KLEEMANN, 2019). Medeiros (2016) coloca que para aumentar a produtividade, bem como reduzir custos no desenvolvimento de aplicativos, que sejam adotadas novas metodologias e práticas para reaproveitamento de código.

A reutilização de código é um aspecto que deve ser considerado no desenvolvimento de aplicativos, apresentando um impacto positivo na qualidade, produtividade e na redução de custos (SANTOS, 2015). Ao desenvolver, é importante o reconhecimento de pontos do

aplicativo que apresentam componentes idênticos ou similares aos quais já foram desenvolvidos para conduzir a reutilização de tais componentes (KLEEMANN, 2019).

A arquitetura baseada em componentes baseia-se basicamente em componentes. Componentes são artefatos que são identificados claramente nas aplicações, eles têm uma interface, encapsulamento interno detalhado e são documentados separadamente (MEDEIROS, 2016). Pode-se dizer, que o desenvolvimento de aplicativos baseado em componentes adere ao princípio da divisão e conquista, diminuindo a complexidade, pois um problema é dividido em partes menores, resolvendo estas partes menores e construindo soluções mais elaboradas a partir de soluções mais simples (FEIJÓ, 2017).

Dessa forma, tanto o uso de componentes quanto o desenvolvimento de componentes têm como preocupação o reaproveitamento de código para facilitar manutenções futuras e principalmente para o aumento de produtividade durante o desenvolvimento de novas funcionalidades nas aplicações. Assim um componente deve ser projetado com foco em uma tarefa específica, tendo assim uma maior chance de ser reutilizado novamente.

Aplicativos desenvolvidos utilizando o *framework* Ionic são indicados a utilizar uma arquitetura de componentes, pois este *framework* disponibiliza um aplicativo de modularização orientado a componentes, conhecido como NgModules (IONIC FRAMEWORK, 2018). NgModules são containers para um bloco de código dedicado a um domínio do aplicativo, ou sejam um fluxo de trabalho ou conjunto estreitamente relacionado a um certo recurso do aplicativo (IONIC FRAMEWORK, 2018).

A utilização de componentes no Ionic auxilia no reaproveitamento de código e no carregamento sobre demanda desses componentes. Uma vez que um módulo é declarado em uma página, os demais módulos que não são daquele domínio do aplicativo não iram carregar, deixando o aplicativo com um carregamento de mais rápido pois não há um processamento de todos os componentes definidos no aplicativo (IONIC FRAMEWORK, 2018).

2.4 TRABALHOS CORRELATOS

Nesta seção são apresentados trabalhos com características semelhantes aos principais objetivos do estudo desenvolvido. A subseção 2.4.1 traz um conceito de plataforma que reúne aplicativos, conhecida por Super-Apps, a subseção 2.4.2 e a subseção 2.4.3 trazem aplicativos de comunicação alternativa para auxiliar na alfabetização de pessoas com necessidades especiais. Por fim, a subseção 3.4.2 traz um comparativo entre os trabalhos relacionados.

2.4.1 Super-Apps

Os `superaplicativos` são plataformas que reúnem diferentes serviços em uma única plataforma, instalando a plataforma, o usuário passa a ter acesso a outros módulos integrados com a plataforma (EXAME, 2017). Segundo Fleximize (2018), WeChat é uma plataforma desenvolvida e implantada pela empresa Tencent, que oferece integração entre vários tipos de aplicativos em um único aplicativo. A plataforma consiste em uma rede social, na qual constam outros aplicativos para contratar serviços, realizar pagamentos, compras e troca de mensagens entre usuários (FLEXIMIZE, 2018). WeChat se difere por ser: acessível de dispositivos móveis e ter multi-módulos.

Com o mesmo intuito, o grupo Movile é responsável pelo desenvolvimento de aplicativos de vários segmentos, como o iFood, para realizar pedidos on-line de comida e o Superplayer, aplicativo de música, também desenvolveu a plataforma Rapiddo, que antigamente era apenas um serviço de entregas e agora consiste em um agregador de serviços. A plataforma tem o intuito de levar para as pessoas apenas um único aplicativo, juntando os demais aplicativos do grupo, para gerenciar tanto serviços off-line quanto on-line. A plataforma permite realizar o pedido de comida, chamar um taxi, ouvir músicas, realizar recargas de celular, comprar ingressos e realizar pagamentos (EXAME, 2018). Essas opções podem ser visualizadas na Figura 2, na parte superior (de cima para baixo), na qual são listados os serviços: `Almoço no Trabalho`, `Táxi`, entre outros.

Figura 2 - Aplicativo Rapiddo

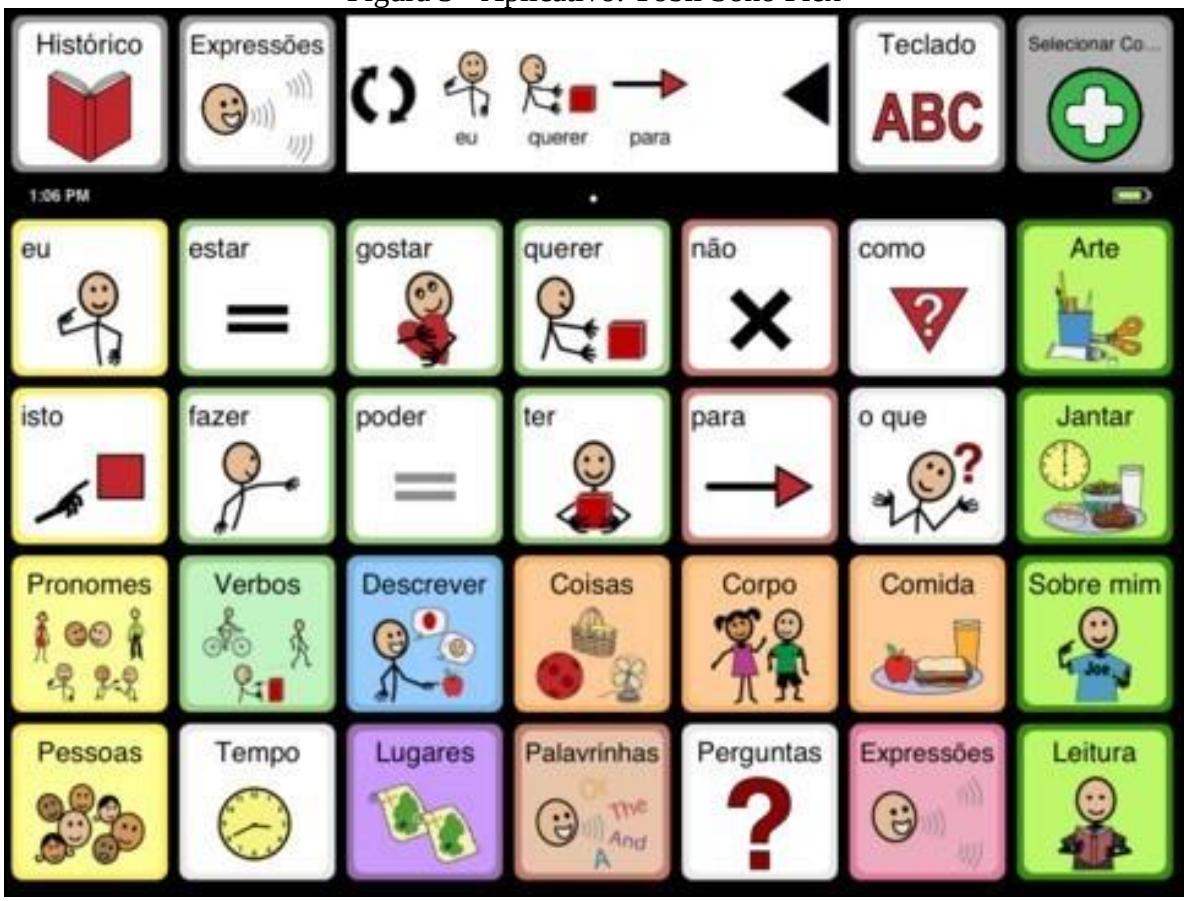
Fonte: Google Play (2018).

2.4.2 TOBII Sono Flex

O Tobii Sono Flex é um aplicativo de comunicação alternativa e assertiva, que foi desenvolvido pela empresa Comércio de Materiais Esportivos e Educativos Civiam Ltda, com o intuito de oferecer recursos de comunicação para os usuários sem capacidade verbal e ainda sem total controle da alfabetização (CIVIAM, 2014). O Tobii Sono Flex é disponibilizado para as plataformas Windows e iOS. Civiam (2014) se destaca por: ser acessível de dispositivos móveis, e ter acesso à câmera e galeria de imagens.

O aplicativo permite a utilização da câmera e galeria de imagens disponíveis no dispositivo, para personalização das imagens do aplicativo. Além de permitir a customização das imagens, o aplicativo também permite a personalização do vocabulário, para atender mais assertivamente as necessidades de cada usuário (CIVIAM, 2014). Cabe desacatar, que o Tobii Sono Flex possui uma experiência de utilização baseada em símbolos, tornando o entendimento do usuário mais intuitivo e possibilitando que usuários com necessidades especiais ou sem alfabetização consigam utilizar das funções do aplicativo. A Figura 3 traz a página inicial, contemplando as categorias principais de vocabulários, codificado por cores para ajudar a associação.

Figura 3 - Aplicativo: Tobii Sono Flex

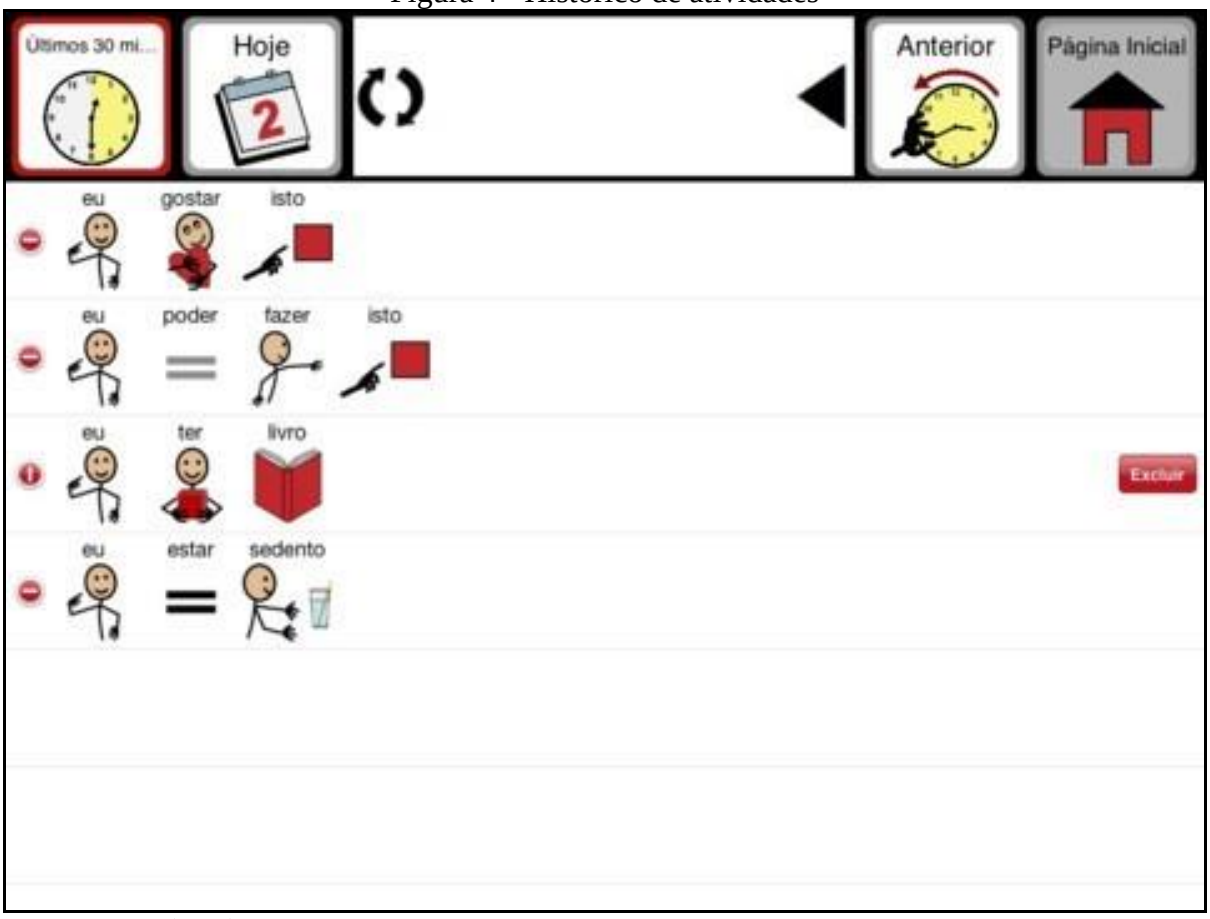


Fonte: Civiam (2014).

Está página é formada por dois grupos, um mais acima com uma linha de símbolos e outro abaixo com quatro linhas de símbolos. O grupo acima são o menu do aplicativo, disponibilizando acesso as telas de histórico, expressões, teclado, entre outras. Já o grupo abaixo é dividido em três regiões: pela última coluna de símbolos com o fundo verde, que são: contextos em que estarão os vocabulários disponibilizados aos usuários; pelas duas primeiras linhas deste grupo, que são utilizadas em diversas situações diferentes do dia-dia, como `eu`, `não`, `como`, `fazer`; e pelas duas últimas linhas deste grupo são temas que são comuns para o usuário (CIVIAM, 2014).

A Figura 4 traz a tela do histórico das atividades do usuário, permitindo que as ações de consultas e aquelas mais utilizadas pelo usuário sejam realizadas de forma mais rápida. Essa tela oferece a opção `Anterior`, permitindo consultar as ações feitas nos últimos 30 minutos, no dia atual ou a última atividade feita; a opção `Excluir` (botão excluir que aparece no meio do lado direito – da esquerda para direita e de cima para baixo), possibilitando apagar o histórico das atividades (CIVIAM, 2014).

Figura 4 - Histórico de atividades



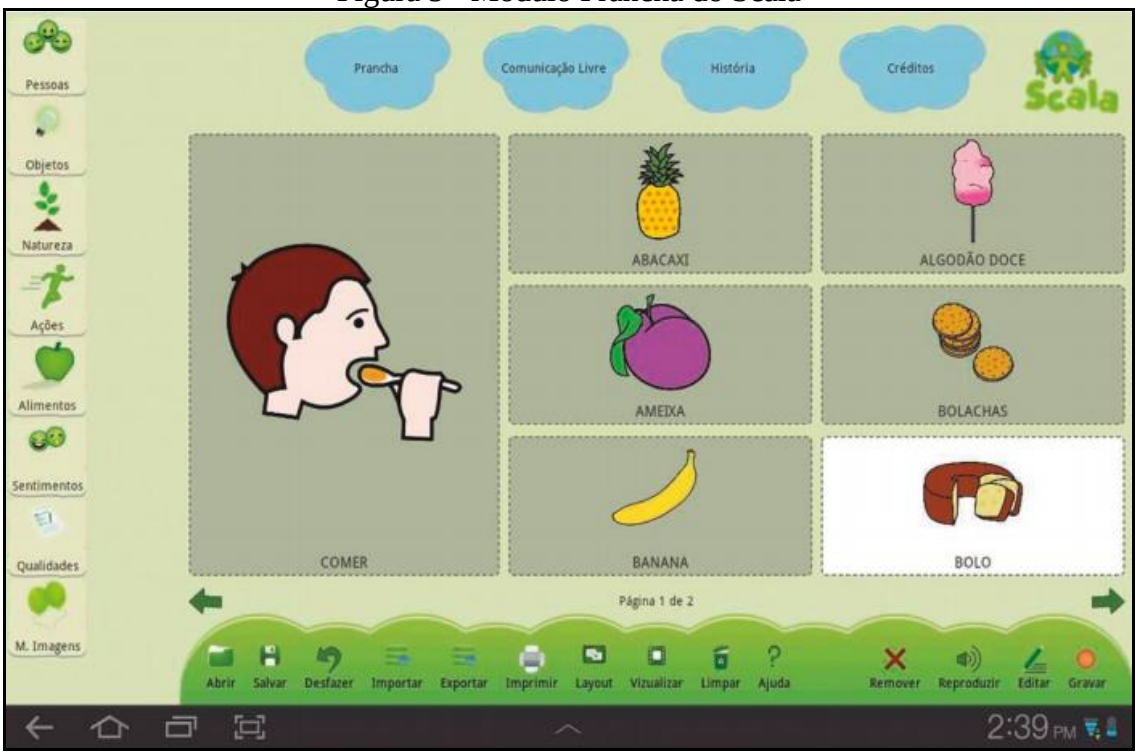
Fonte: Civiam (2014).

2.4.3 Scala

Scala é um aplicativo acadêmico, desenvolvido na Universidade Federal do Rio Grande do Sul, que tem como público alvo alunos com dificuldades de interação social, como Transtorno do Espectro Autista (TEA) (PASSERINO, 2015). O TEA é um problema psiquiátrico que costuma ser identificado na infância, entre 1 ano e meio e 3 anos, embora os sinais iniciais às vezes apareçam já nos primeiros meses de vida (SAÚDE, 2018). O distúrbio afeta a comunicação e capacidade de aprendizado e adaptação da criança (SAÚDE, 2018). Saúde (2018) se destaca por: ter prancha de comunicação; ser acessível para funcionamento pela internet e para dispositivos móveis (sistema operacional Android); reprodução áudio ao interagir; ter multi-módulos; ter acesso à câmera e galeria de imagens.

O aplicativo possui o módulo Prancha que pode ser estruturado com as rotinas diárias da criança (PASSERINO, 2015). Na Figura 5 pode ser visto que as pranchas com as rotinas diárias do usuário possuem imagens claras, facilitando a utilização da ferramenta por usuários sem alfabetização.

Figura 5 - Módulo Prancha do Scala



Fonte: Passerino (2015).

O módulo Prancha possui recursos de layout, que possibilitam a configuração para organizar a prancha e as figuras dispostas nela. Também são disponibilizadas as opções de criação e edição das pranchas, podendo escolher as `categorias`, `figuras` e `layouts`. Na Figura 6 pode ser vista a tela de criação de pranchas.

Figura 6 - Tela de criação de prancha do Scala

Fonte: Passerino (2015).

As categorias do Scala estão dispostas em: Pessoas, Objeto, Natureza, Ações, Alimentos, Sentimentos, Qualidades e Minhas imagens. A categoria Minhas imagens contém as figuras importadas pelo usuário no aplicativo, que podem ser acessadas por meio da galeria de imagens ou da câmera do celular. O usuário demonstra na prancha qual ação que ele deseja expressar. Para isto, basta clicar nas figuras que expressam seu desejo. Ao realizar clique na figura, o seu fundo é destacado para mostrar ao usuário sua ação (PASSERINO, 2015).

O Scala também possui um módulo, chamado Narrativas Visuais, que tem como intuito apoiar a alfabetização de crianças com autismo a partir da contação da construção de histórias (PASSERINO, 2015). Na Figura 7 pode ser visto o módulo de histórias do Scala, que permite a ilustração de ambientes e personagens de forma mais dinâmica. Assim, o usuário pode expressar melhor como foi a relação dele com ambientes em que ele frequentou, e o tutor pode demonstrar como funciona ambientes em que ele futuramente irá frequentar.

Figura 7 - Módulo História do Scala

Fonte: Passerino (2015).

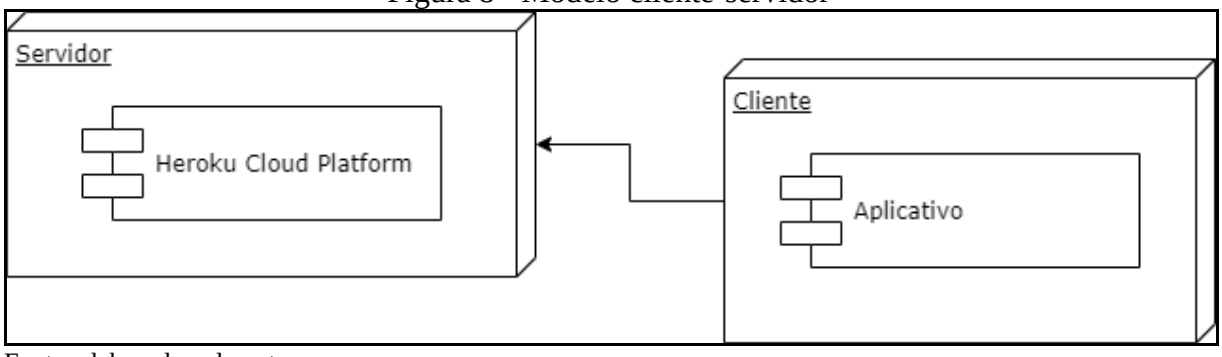
3 DESENVOLVIMENTO

Neste capítulo são apresentadas as etapas de desenvolvimento do aplicativo. A seção 3.1 apresenta os requisitos funcionais e não-funcionais. A seção 3.2 contém a especificação do aplicativo, utilizando diagramas da UML. A seção 3.3 detalha a implementação do aplicativo, apresentando os principais trechos de código e exemplos de uso das telas. Por fim, a seção 3.4 aborda os resultados obtidos deste trabalho

3.1 LEVANTAMENTO DE INFORMAÇÕES

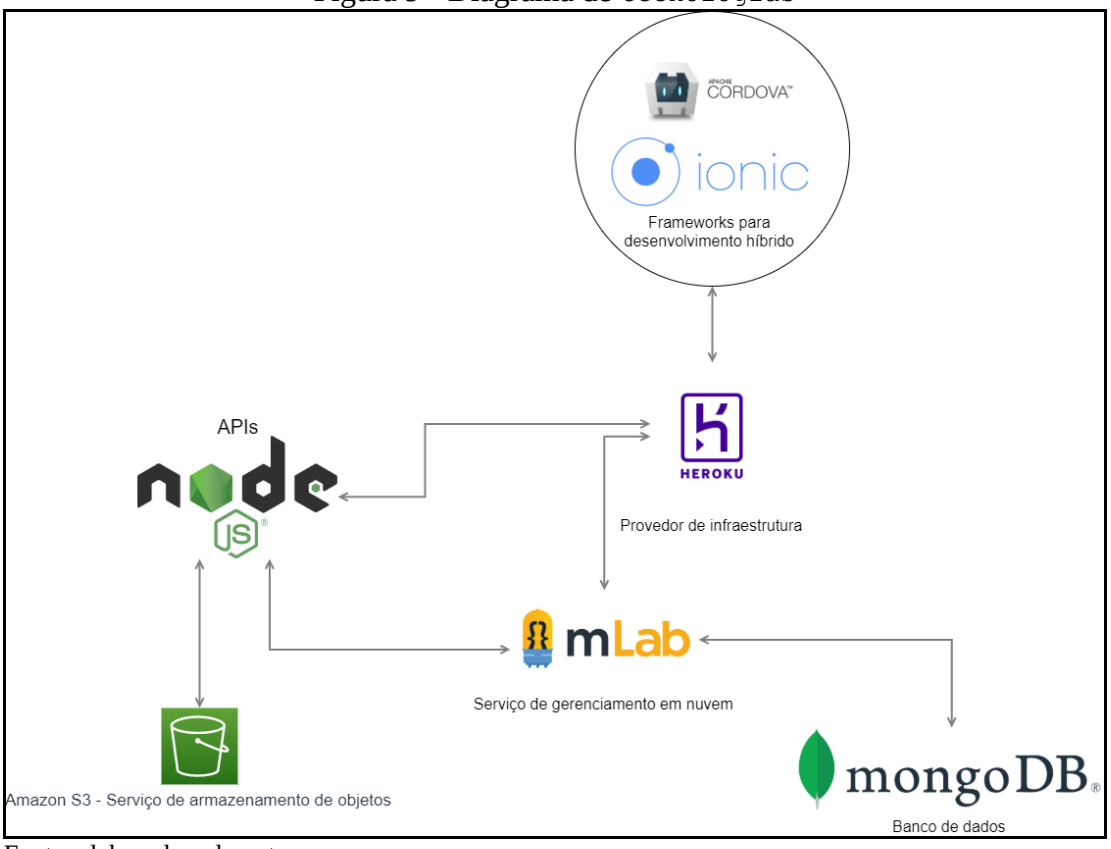
Para desenvolver o aplicativo Tagarela foi utilizado o modelo `cliente-servidor`, como pode ser visto na Figura 8. No lado do `cliente` foi utilizada a linguagem `Typescript` e os *frameworks* `Ionic` e `Cordova`, que viabilizam a criação de aplicações moveis hibridas para a plataforma `iOS` e `Android`. No lado do `servidor` foi utilizado o `Heroku Cloud Platform` como provedor de infraestrutura em nuvem para garantir a comunicação com as Application Programming Interface (API) que foram desenvolvidas em `Node.JS`. Para comunicação com a base de dados, o `Heroku Cloud Platform` possui uma conexão com o `mLAB`, um serviço para hospedagem do banco de dados. O Sistema Gerenciador de Base de Dados (SGBD) utilizado foi o MongoDB, banco de dados não relacional, que armazena os dados como `Javascript Object Notation` (JSON) e sincronizados em tempo real com todos os clientes conectados. Como forma de facilitar o entendimento do projeto, a Figura 9 traz cada uma das tecnologias utilizadas no desenvolvimento, bem como, sua respectiva infraestrutura de execução.

Figura 8 - Modelo cliente-servidor



Fonte: elaborado pelo autor.

Figura 9 - Diagrama de `tecnologias`



Fonte: elaborado pelo autor.

Para a realização deste trabalho cumpriram-se os seguintes requisitos:

a) permitir o cadastro de usuários com o perfil de tutor ou paciente (Requisitos Funcionais - RF);
b) permitir realizar o login dos usuários no aplicativo (RF);
c) permitir o vínculo entre usuários (RF);
d) permitir a criação de planos de atividades (RF);
e) permitir filtrar os planos de atividade de acordo com usuários vinculados (RF);
f) permitir a criação de pranchas de comunicação (RF);
g) permitir a criação de símbolos para as pranchas de comunicação (RF);
h) permitir a criação de categorias para os símbolos (RF);
i) permitir a reutilização de pranchas de comunicação (RF);
j) permitir a interação dos usuários com as pranchas (RF);
k) permitir a inserção de novos módulos no Tagarela (RF);
l) permitir o controle de visibilidade dos módulos disponibilizados no aplicativo (RF);
m) disponibilizar a API em um servidor para acesso externo, fornecendo às informações para o aplicativo (RF);

n) possuir um aplicativo para dispositivo móvel construído com Ionic f*ramework* 3 (Requisitos Não Funcionais - RNF);
o) ser implementado com uma arquitetura baseada em componentes (RNF);
p) utilizar Node.JS para desenvolvimento das APIs (RNF);
q) ser implementado com a linguagem de programação TypeScript (RNF);
r) utilizar banco de dados MongoDB para persistência dos dados (RNF).

3.2 ESPECIFICAÇÃO

Esta seção contém a especificação do aplicativo desenvolvido. Apresenta-se os Diagramas de Caso de Uso (DCU) na subseção 3.2.1 e suas respectivas rastreabilidade no Quadro 1. Nas subseções subsequentes são demonstrados os demais diagramas utilizados durante o desenvolvimento do aplicativo.

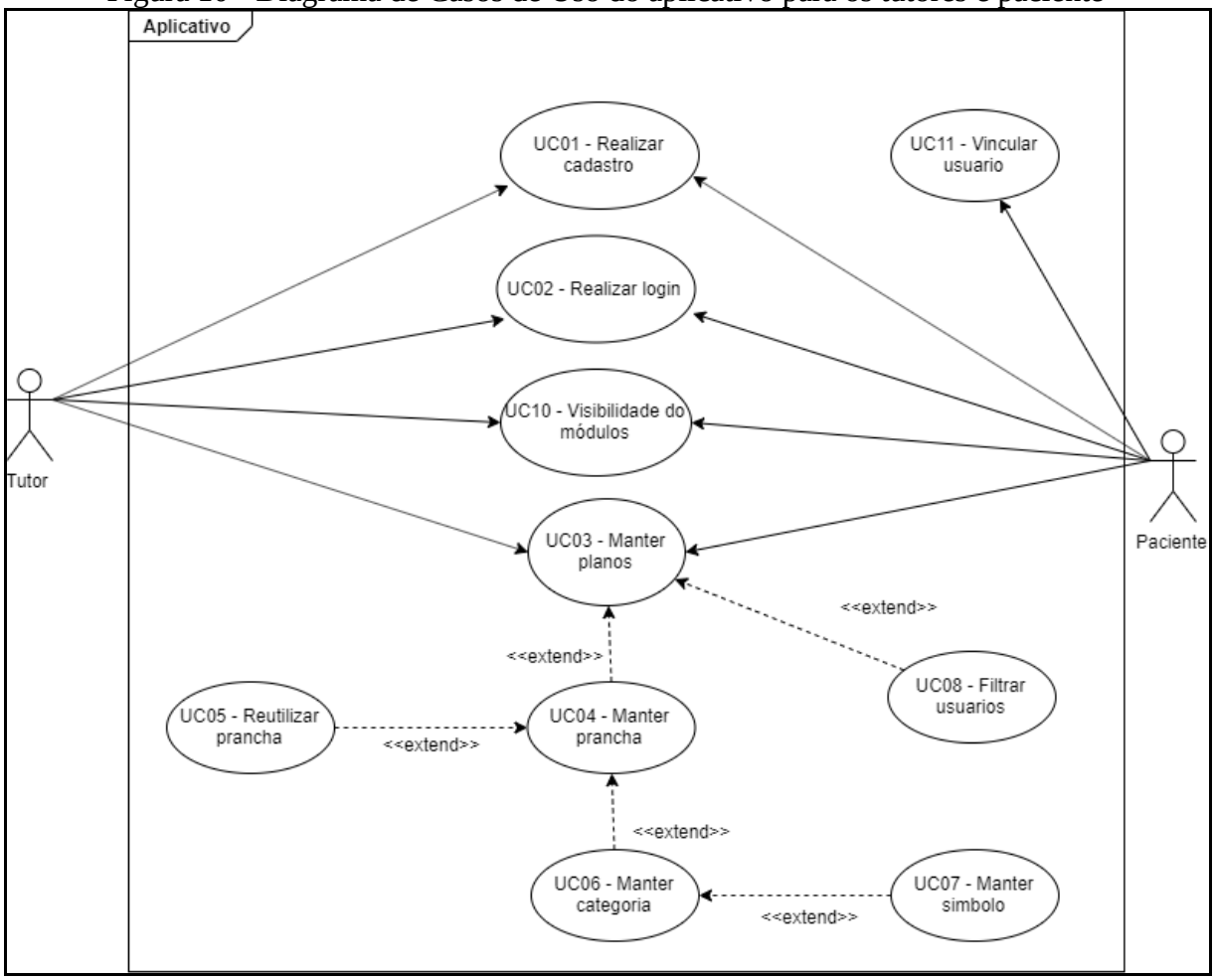
3.2.1 Diagrama de Caso de Uso

Esta subseção apresenta os Diagramas de Casos de Uso (DCU) para as funcionalidades do aplicativo. No aplicativo existem três perfis de usuário, o `Tutor`, o `Paciente` e o `Administrador`. Na Figura 10 apresenta o DCU do aplicativo para os tutores e pacientes, que com os atores `Tutor` e `Paciente`, já na Figura 11 apresenta o DCU da aplicativo o ator `Administrador`. No DCU do aplicativo para os tutores e pacientes Figura 10, o caso de uso referente a tela de cadastro de novo usuário é o `UC01 – Realizar cadastro`. Inclui-se também o `UC02 – Realizar login` que disponibilizara ao usuário o acesso as funcionalidades do aplicativo. O `UC03 – Manter planos` apresenta a lista de planos criados e a opção de criar um plano. É disponibilizado um filtro com a lista de usuários vinculados para apresentar os planos do usuário por meio do `UC08 – Filtrar usuários`.

Após a escolha de um plano, as pranchas são exibidas por meio do caso de uso `UC04 – Manter prancha`. Ao selecionar uma prancha, o aplicativo exibe os símbolos cadastrados para esta prancha. O `UC05 – Reutilizar prancha` disponibiliza a opção de duplicar prancha em que o usuário está posicionado para reutilização. O usuário pode interagir com os símbolos cadastrados na prancha, caso tenha algum símbolo faltante na prancha o usuário pode adicionar um novo. Ao selecionar a opção adicionar, o aplicativo exibe as categorias cadastradas no aplicativo por meio do `UC06 – Manter categoria` podendo criar uma categoria. Após a escolha de uma categoria são listados os símbolos por meio do caso de uso `UC07 – Manter símbolo`. Por fim, o aplicativo possibilita que usuário controle a visibilidade dos módulos exibidos no seu menu com o `UC10 – Visibilidade dos módulos`. Além disso,

o Paciente pode vincular os seus tutores para compartilhar seus planos por meio do caso e uso `UC11 – Vincular usuário`.

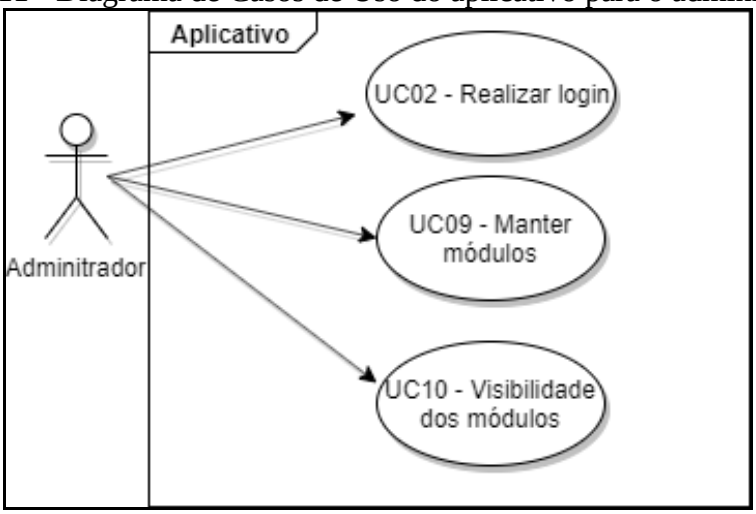
Figura 10 - Diagrama de Casos de Uso do aplicativo para os tutores e paciente



Fonte: elaborador pelo autor.

No DCU do aplicativo para administradores Figura 11, o caso de uso referente a tela de *login* que disponibilizara ao usuário o acesso as funcionalidades do aplicativo é o `UC02 – Realizar login`. O `UC09 – Manter módulos` apresenta um Create, Read, Update and Delete (CRUD) para que o `Administrador` possa cadastrar os módulos do aplicativo. Por fim, o aplicativo possibilita que usuário controle a visibilidade dos módulos exibidos no seu menu com o `UC10 – Visibilidade dos módulos`.

Figura 11 - Diagrama de Casos de Uso do aplicativo para o administrador



Fonte: elaborado pelo autor.

3.2.2 Matriz de rastreabilidade dos RFs e sua relação com os Casos de Uso (UC)

Nesta subseção é apresentado o Quadro 1 com a matriz de rastreabilidade dos Requisitos Funcionais com os UC referente ao aplicativo.

Quadro 1 - Matriz de rastreabilidade dos RF com UC

RF	**Descrição**	**UC**
RF01	Permitir o cadastro com o perfil de tutor ou paciente	UC01
RF02	Permitir login dos usuários	UC02
RF03	Permitir o vínculo entre usuários	UC11
RF04	Permitir a criação de planos de atividade	UC03
RF05	Permitir filtrar os planos de atividade por usuário vinculado	UC08
RF06	Permitir a criação de pranchas de comunicação	UC04
RF07	Permitir a criação de símbolos para as pranchas de comunicação	UC07
RF08	Permitir a criação de categorias para os símbolos	UC06
RF09	Permitir a reutilização de pranchas de comunicação	UC05
RF10	Permitir a interação dos usuários com as pranchas	UC04
RF11	Permitir a inserção de novos módulos no Tagarela	UC09
RF12	Permitir o controle de acesso aos módulos a serem disponibilizados no aplicativo	UC10

Fonte: elaborado pelo autor.

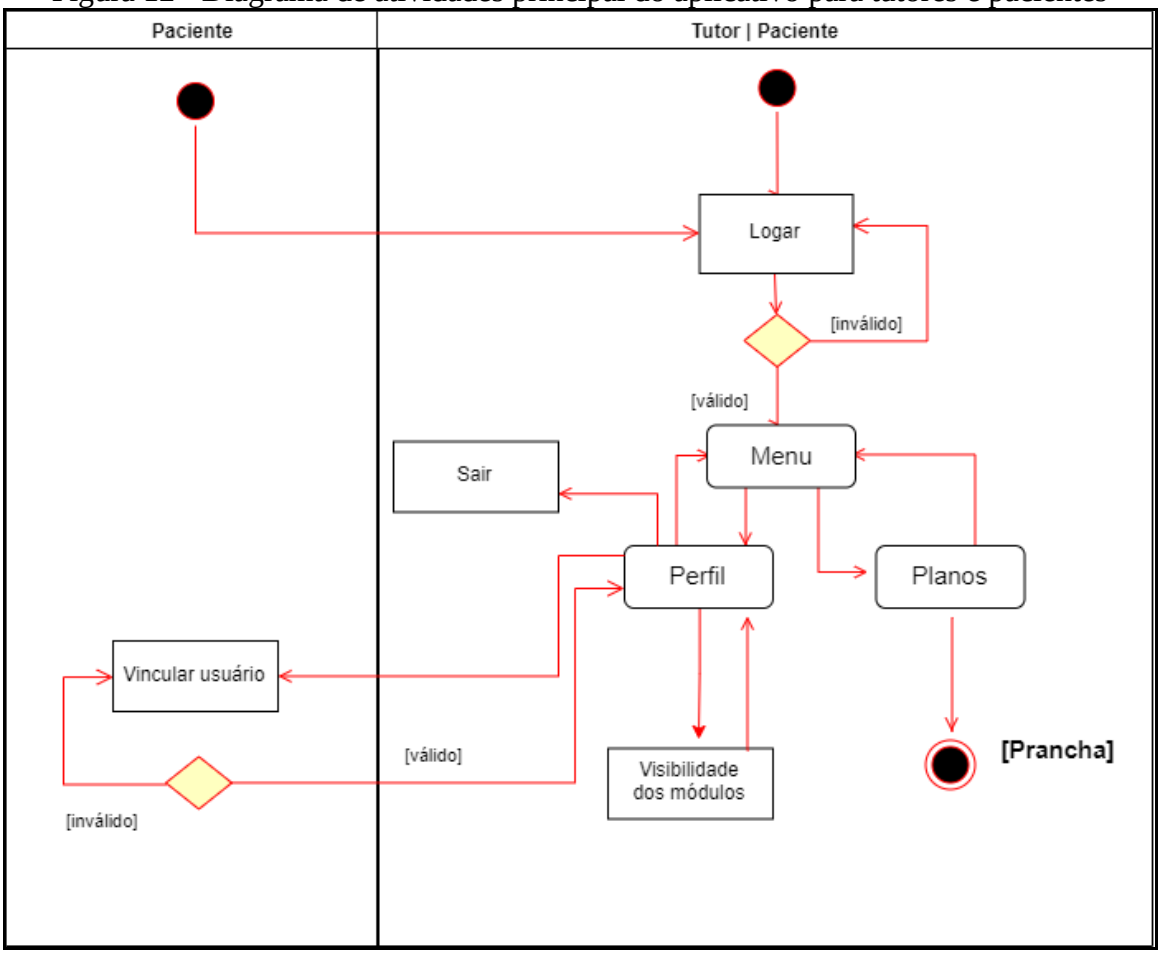
3.2.3 Diagrama de Atividades

O diagrama de atividades demonstra de forma geral a utilização de todos os recursos do aplicativo pelos três atores envolvidos. Este diagrama divide-se em três partes, a primeira

contém as atividades principais dos atores `Tutor` e `Paciente`, a segunda parte trata das atividades principais do ator Administrador e a terceira parte refere-se das atividades relacionadas a prancha de comunicação, que é uma funcionalidade importante do aplicativo. A Figura 12 retrata de forma prática este cenário por meio do diagrama de atividades por colunas. A coluna da esquerda apresenta as atividades realizadas apenas pelo ator `Paciente` e a coluna da direita apresenta as atividades realizadas por ambos os atores, tanto `Tutor` quanto `Paciente`.

No diagrama da Figura 12 também encontra-se atividades que estão diretamente relacionadas com os casos de uso definidos na Figura 10. As atividades no aplicativo iniciam-se com a atividade `Logar` relacionada ao `UC02`. No Menu ele terá acesso aos Planos, podendo criar pranchas, reutilizar pranchas ou interagir com elas. Caso o usuário opte por acessar a tela de `Perfil`, ele terá acesso ao controle de `visibilidade dos módulos` disponíveis no `menu` e poderá `sair` do aplicativo. Se o usuário possuir o papel de `Paciente`, ele terá acesso a tela de `vincular usuário`, neste processo deve ser informado um usuário que possuirá vínculo com o paciente, podendo ter acesso as suas informações nos planos de atividades.

Figura 12 - Diagrama de atividades principal do aplicativo para tutores e pacientes



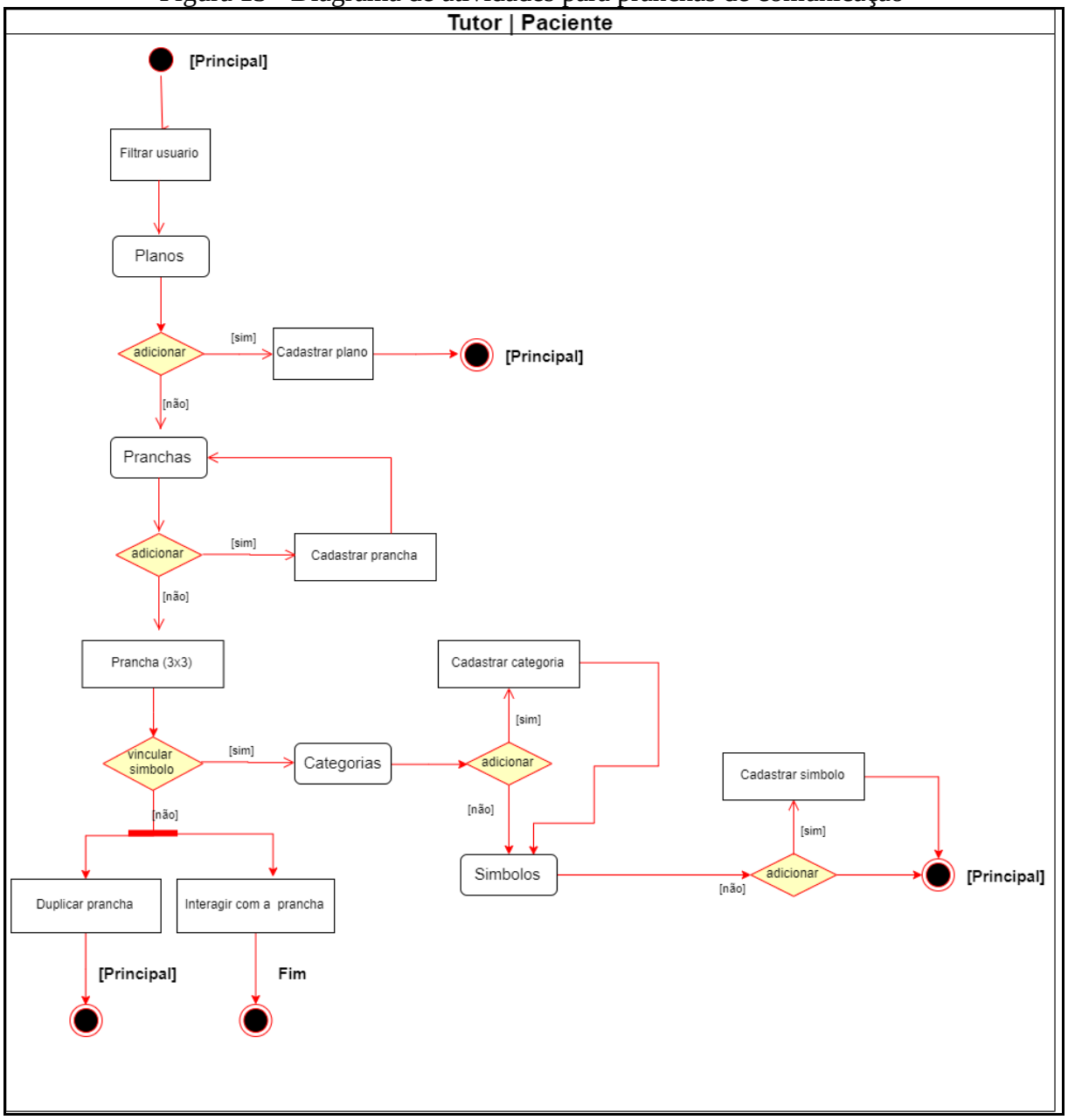
Fonte: elaborado pelo autor.

A atividade de interação com uma prancha de comunicação está relacionada ao UC03 – Manter planos e está demonstrado na Figura 13. O primeiro passo do processo ao iniciar a interação com os planos de atividades é filtrar usuário, podendo filtrar os planos de atividades por usuários vinculados ou utilizar os seus próprios planos de atividades. Com isto, é apresentado a lista de planos de atividades, caso o usuário já possua planos, pode-se prosseguir para as pranchas de comunicação vinculadas ao plano de atividade, porém se não possui planos, é necessário realizar o processo de cadastro antes de continuar para as pranchas.

Com o plano de atividades cadastrado, é apresentada as pranchas de comunicação do plano. Se o usuário não possui pranchas, é necessário realizar o processo de cadastro antes de continuar para a prancha 3x3. As pranchas 3x3 são constituídas de símbolos vinculados, caso o usuário possua algum símbolo na prancha, ele poderá interagir com ele para que seja executado o áudio vinculado ao símbolo. O usuário terá a opção de duplicar a prancha onde está posicionado.

Para adicionar um novo símbolo na prancha 3x3, é necessário escolher uma categoria, porém se não possui nenhuma categoria, é necessário realizar o cadastro para continuar o processo de adicionar símbolo. Com a categoria criada, é possível fazer a escolha de um símbolo já cadastro. Caso o usuário não possua nenhum símbolo cadastro, ele terá a opção de adicionar um novo, nesse processo deve ser informada uma imagem e áudio relacionada ao símbolo, e assim que estiver incluída as informações, o cadastro do símbolo pode ser finalizado.

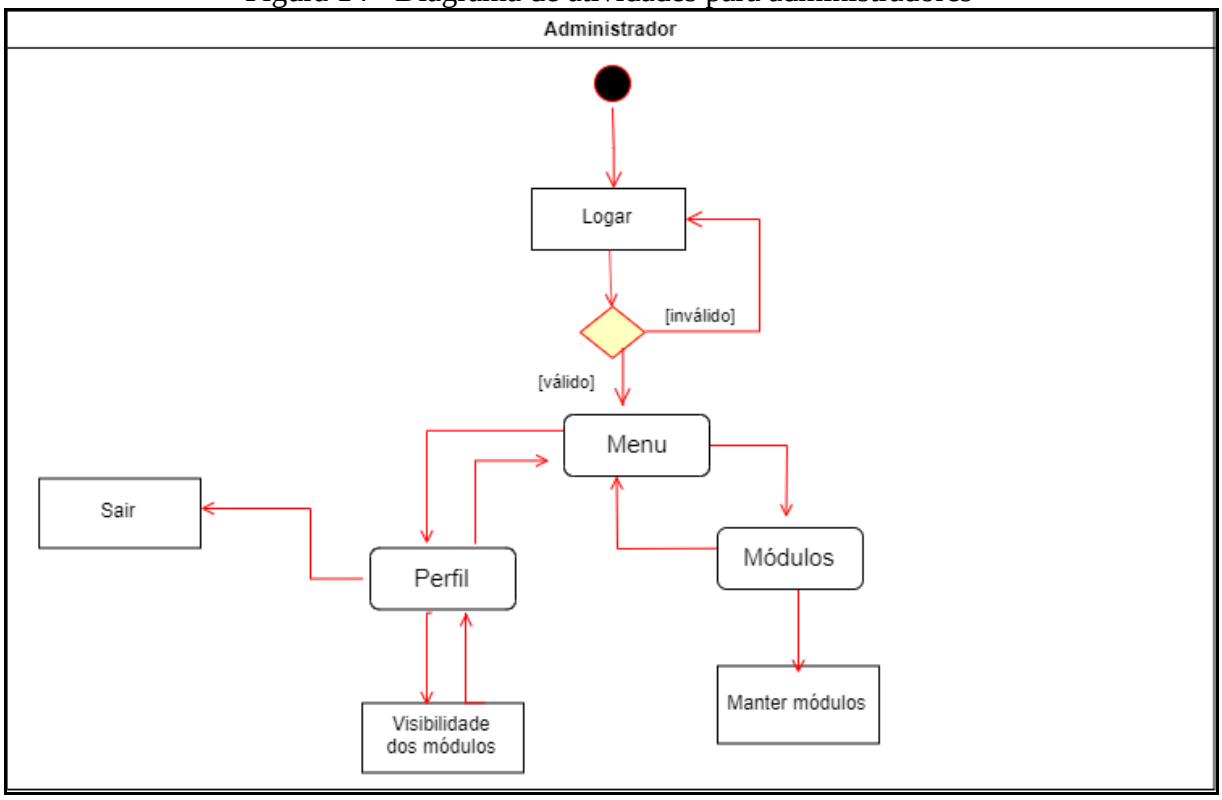
Figura 13 - Diagrama de atividades para pranchas de comunicação



Fonte: elaborado pelo autor.

A Figura 14 traz o fluxo de atividades com o ator `Administrador` e encontra-se atividades que estão diretamente relacionadas com os casos de uso definidos na Figura 11. As atividades no aplicativo iniciam-se com a atividade `Logar` relacionada ao `UC02`. Será apresentado o `menu` principal do sistema para o `Administrador`, possibilitando-o de acessar a tela de `Módulos`. A tela de `módulos` é responsável por criar, editar e listar os módulos cadastrados no sistema. Caso o usuário opte por acessar a tela de `Perfil`, ele terá acesso ao controle de `visibilidade dos módulos` disponíveis no `menu` e poderá `sair` do aplicativo.

Figura 14 - Diagrama de atividades para administradores



Fonte: Elaborado pelo autor.

3.2.4 Diagrama de Classes

A Figura 15 traz o diagrama de classes que compõem o aplicativo e seus relacionamentos. Apesar da linguagem JavaScript não possuir classes, apenas objetos, cada arquivo principal foi simulado como uma classe no diagrama, separados por funcionalidades ou assuntos, com intuito de organizar a estrutura do aplicativo e atender os requisitos especificados de maneira mais clara.

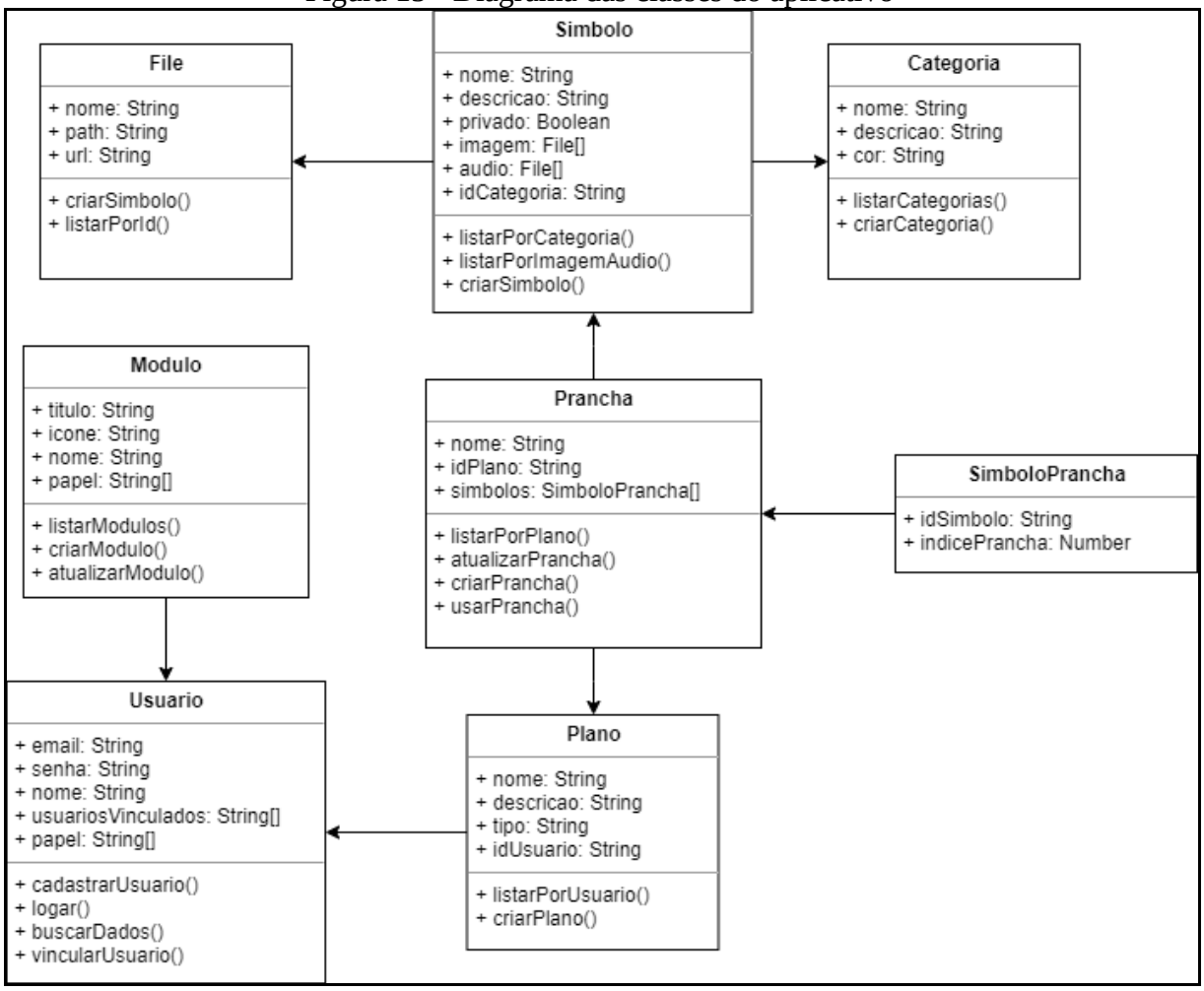
A primeira classe definida é a classe `Usuário`, que tem como responsabilidade o gerenciamento dos usuários do aplicativo, possuindo um método para acessar o aplicativo, bem como criar um usuário escolhendo seu perfil, informação esta que será armazenada no atributo `papel`. Além disso, pelo método `vincularUsuario` é possível criar um vínculo entre dois usuários com perfis distintos. Este vínculo será tratado pela classe `Plano`, que irá buscar os usuários vinculados para utilizar como filtro no método `listarPorUsuario`. A classe `Prancha` possui o método `usarPrancha` que permite ao usuário interagir com a prancha de comunicação. Este método é responsável por reproduzir o áudio que é vinculado a imagem da prancha ao ser pressionado, utilizando as informações estão presentes no atributo `simbolos`.

Uma prancha de comunicação deve possuir um ou mais símbolos, este relacionamento é determinado com a classe `Simbolo`, que faz o gerenciamento dos símbolos cadastrados no

aplicativo. O método `criarSimbolo` implementa o *plugin* câmera do Ionic para permitir ao usuário a seleção de uma imagem, fazendo uso da galeria de fotos de seu dispositivo. Este método também é responsável por realizar a gravação de um áudio para vincular ao símbolo criado, utilizando o *plugin* `media` do Ionic. Esta classe também possui o atributo `privado` informando se o símbolo é privado, o que caracteriza o símbolo como sendo uma informação sensível e que não pode ser compartilhada entre pranchas posteriormente entre os demais usuários do aplicativo. Todos os símbolos devem ter um vínculo com alguma categoria, denominando o tipo dos símbolos, informação está presente na classe `Categoria`.

Por fim, a classe `modulo` possui os módulos cadastrados no aplicativo, que pelo método `listarModulos` são listados para apresentar no menu principal do aplicativo para acesso das funcionalidades. O administrador do aplicativo poderá manter os módulos do aplicativo, bem como habilitar novos módulos no aplicativo.

Figura 15 - Diagrama das classes do aplicativo



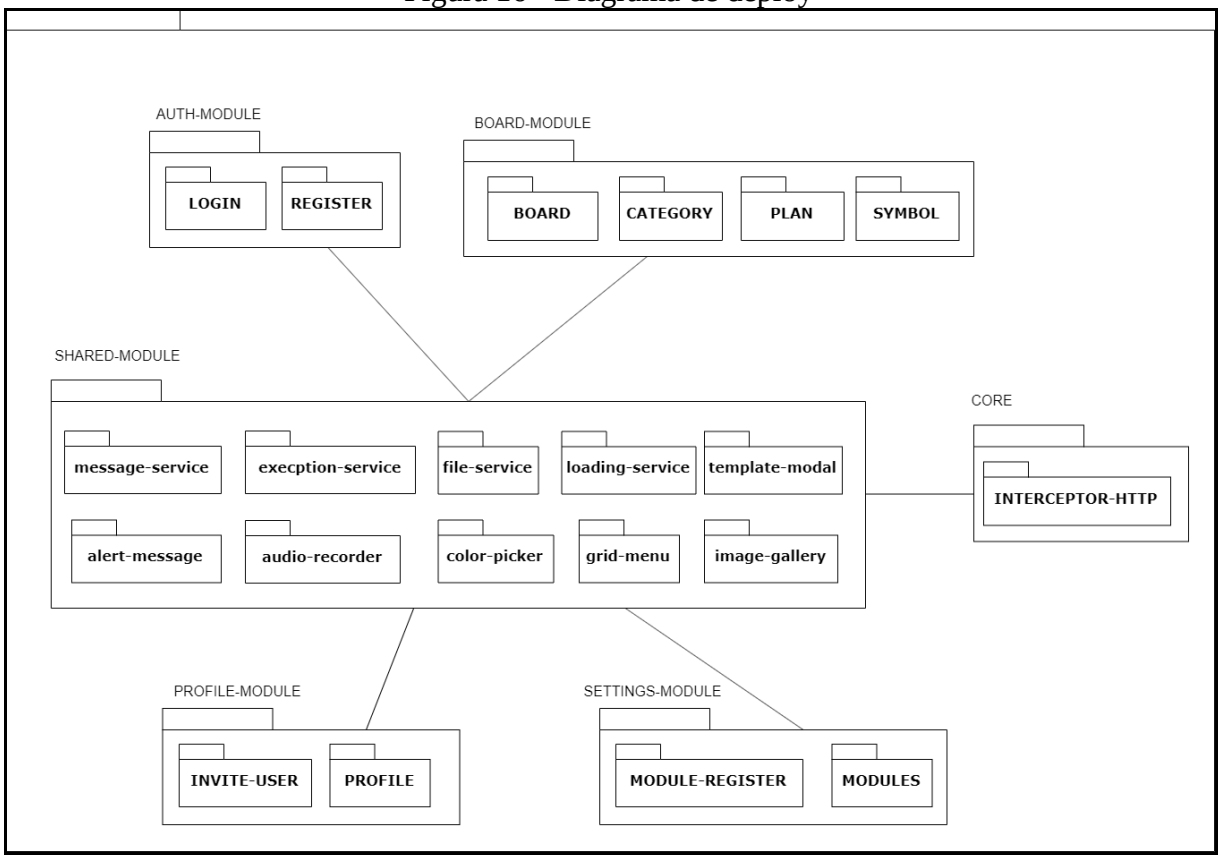
Fonte: elaborado pelo autor.

3.2.5 Diagrama de Deploy

Neste trabalho foi desenvolvida uma arquitetura de componentes. Como pode ser visto na subseção 2.3 a arquitetura se dá pela reutilização de componentes comuns a fim de reaproveitar o código desenvolvido para compartilhamento entre os módulos do aplicativo. Na Figura 16 é possível visualizar os módulos e componentes, sendo:

a) `Auth-Module`: módulo responsável pelo cadastro de usuário e login;
b) `Board-Module`: módulo responsável pelo fluxo de Pranchas de comunicação, descrito na Figura 13;
c) `Profile-Module`: módulo responsável pelo vínculo entre usuários e controle de visibilidade dos módulos no menu;
d) `Settings-Module`: módulo responsável pelo cadastro dos módulos que o usuário irá interagir no aplicativo;
e) `Core`: serviço responsável por atuar como middleware de todas as requisições HyperText Transfer Protocol (HTTP) feitas pelo aplicativo
f) `Shared-Module`: módulo orquestrador de todos os componentes e serviços reutilizáveis entre os demais módulos do aplicativo.

Figura 16 - Diagrama de deploy



Fonte: elaborado pelo autor.

3.3 IMPLEMENTAÇÃO

Nesta seção são descritas as técnicas e ferramentas utilizadas para o desenvolvimento do trabalho, além de ser detalhado o processo de implementação do aplicativo. Na subseção 3.3.1 mostra as técnicas e ferramentas utilizadas no processo de criação do trabalho. Já a subseção 3.3.2 apresenta a operacionalidade da implementação, com o funcionamento do aplicativo e suas respectivas telas.

3.3.1 Técnicas e ferramentas utilizadas

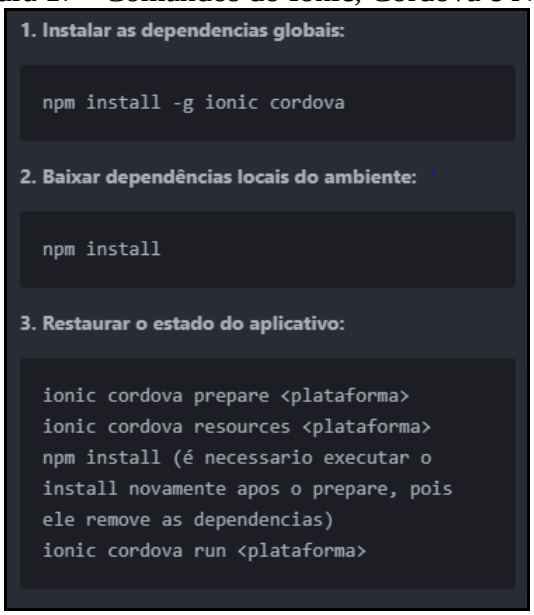
Para desenvolver o aplicativo Tagarela foi utilizado o modelo cliente-servidor e a arquitetura de componentes (seção 2.3). A interface foi construída com auxílio do Ionic que é uma ferramenta que viabiliza a criação de aplicativos moveis híbridos para as plataformas iOS e Android (IONIC FRAMEWORK, 2018) (seção 2.2) . O Sistema Gerenciados de Base de Dados (SGBD) utilizado foi o MongoDB. Foi utilizado o serviço Amazon S3 para possibilitar o armazenamento de arquivos em nuvem. O ambiente de desenvolvimento escolhido para a criação do trabalho foi o Visual Studio Code, no qual foram utilizadas as linguagens TypeScript 2.6.0, HTML5, CSS 3.0, Node.JS 10.16.0 e JavaScript. Para desenvolvimento do cliente foi utilizado *framework* Ionic v3.2.2 usando o Angular 5.2.11.

3.3.1.1 Utilização do Ionic

O Ionic é uma ferramenta de desenvolvimento híbrido, porém é necessário efetuar algumas configurações no seu ambiente antes de começar a utilizar. Primeiramente deve ser decidido com quais plataformas deseja-se trabalhar, pois apesar dos aplicativos serem desenvolvidos utilizando recursos *web*, é necessário obter os Software Development Kits (SDK) específicos de 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 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 Ionic e do Cordova. Para isto, é executado o comando `npm install -g ionic cordova`, dentro do aplicativo `Terminal`. Desta forma será efetuada a instalação do Ionic Command Line Tools (CLI) e Cordova CLI em seu ambiente de trabalho. A partir deste momento o Ionic e o Cordova estão prontos para uso, e todas as operações são realizadas por meio de comandos via `Terminal`, conforme mostra a Figura 17.

Figura 17 - Comandos do Ionic, Cordova e NPM



Fonte: elaborado pelo autor.

Primeiro executou-se o comando `npm install -g ionic cordova` para instalação do Ionic e Cordova. Após isso é executado o comando `npm install` para realizar o download de todas as dependências necessários para a execução do aplicativo. O aplicativo não pode ser executado ou simulado nos dispositivos desejados até que sejam adicionadas as plataformas. Isto é realizado com o comando `ionic cordova prepare`, sendo necessário passar por parâmetro as plataformas desejadas, neste caso Android ou 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 quatro passos necessários, que são mostrados no tópico 3 da Figura 17. O primeiro comando, `ionic cordova resources`, gera as imagens de ícones e *splash screen* para a plataforma desejada dentro do subdiretório `platforms` do projeto. O segundo passo se dá por executar o comando `ionic cordova prepare`, responsável por compilar o aplicativo para a plataforma desejada.

Ao executar o `ionic cordova prepare`, será necessário rodar o terceiro comando, o `npm install`, pois o Ionic possui um bug em sua versão 3 no qual apaga todas as dependências após o comando `ionic cordova prepare`. Ao instalar as dependências novamente, pode-se então executar o quarto passo, responsável por rodar o aplicativo, tanto em um simulador quanto em um dispositivo móvel conectado ao computador, respectivamente por meio do comando `ionic cordova run`, informando via parâmetro qual a plataforma desejada.

3.3.1.2 Criação e reutilização de pranchas

Nesta subseção será detalhada a implementação das funcionalidades de criação e reutilização de pranchas de comunicação do Tagarela. Estas duas funcionalidades compartilham os mesmos métodos, e a reutilização se dá por meio de um botão na tela de `Prancha`, no qual o usuário poderá duplicar a prancha atual. Para efetuar a reutilização será carregado todos os símbolos da prancha já cadastrada. O Quadro 2 mostra um trecho do método `getMultipleSymbols` da Classe `BoardService`, implementado em TypeScript que fará uma requisição HTTP para o servidor a fim de buscar o endereço de hospedagem de cada símbolo da prancha.

Quadro 2 - Método getMultipleSymbols

```
15  public getMultipleSymbols(symbols: string[]): Observable<any> {
16    let url = `${API.URL}/symbolsById`;
17    return this.httpClient.post(url, symbols);
18  }
```

Fonte: elaborado pelo autor.

Neste trecho de código é realizada a busca dos símbolos desta prancha no serviço `S3`. A URL passada por parâmetro no método `post` condiz com o script em JavaScript que realizara o acesso ao banco de dados. Este script está localizado dentro do servidor `Heroku Cloud Platform`. Após a requisição retornar com sucesso, as variáveis `boardImagesUrl` na `linha 71` e `boardAudios` na `linha 72` que armazenam os símbolos e áudios da prancha em memória são preenchidas (Quadro 3).

Quadro 3 - Método getSymbols

```
54  getSymbols(symbolIds, loading) {
55    this.boardService
56      .getMultipleSymbols(symbolIds)
57      .pipe(
58        map(response => (this.symbols = response)),
59        switchMap(() => this.boardService.getCategories()),
60        map(response => (this.categories = response))
61      )
62      .subscribe(
63        () => {
64          for (let index = 0; index < this.board.symbols.length; index++) {
65            const element = this.board.symbols[index];
66
67            for (let index = 0; index < this.symbols.length; index++) {
68              const symbol = this.symbols[index];
69
70              if (element.symbolId === symbol._id) {
71                this.boardImagesUrl[element.boardIndex] = symbol.image[0].url;
72                this.boardAudios[element.boardIndex] = symbol.audio[0].url;
73
74                for (let index = 0; index < this.categories.length; index++) {
75                  const category = this.categories[index];
76
77                  if (category._id === symbol.categoryId) {
78                    this.boardColors[element.boardIndex] = category.color;
79                  }
```

Fonte: elaborador pelo autor.

Durante o processo de reutilização da prancha de comunicação feita uma interação sobre os símbolos já cadastros, como pode ser visto nas `linhas 116` e `117` do Quadro 4, para que possa ser removido os símbolos privados da prancha, no qual será aberto um espaço na prancha 3x3 para adicionar um novo símbolo. Isto possibilita que o usuário siga o processo normal de criação de uma prancha, como pode ser verificado entre a `linha 119` e `126` do Quadro 4.

Quadro 4 - Método duplicateBoard

```
113 duplicateBoard() {
114   const newBoard = { ...this.board };
115
116   for (let index = 0; index < this.symbols.length; index++) {
117     const symbol = this.symbols[index];
118
119     for (let index = 0; index < newBoard.symbols.length; index++) {
120       const newBoardSymbol = newBoard.symbols[index];
121
122       if (symbol._id === newBoardSymbol.symbolId && symbol.isPrivate) {
123         newBoardSymbol.symbols = newBoard.symbols.splice(index, 1);
124       }
125     }
126   }
127   delete newBoard._id;
128   newBoard.name += ' - Duplicado';
129
130   this.board = newBoard;
131 }
```

Fonte: elaborador pelo autor.

Após o usuário selecionar os símbolos desejados para a sua nova prancha, o método `saveBoard` da classe `BoardRegister` é chamado. O Quadro 5 apresenta um trecho da

implementação deste método. A variável `board`, na `linha 134`, armazena as informações que serão enviadas ao servidor em formato JSON. Estas informações dizem respeito aos símbolos e ao plano que o próprio usuário selecionou à medida que realizada o processo de criação de prancha nas telas anteriores.

Quadro 5 - Método saveBoard

```
133   saveBoard(newBoard?): void {
134     const board = newBoard ? newBoard : this.board;
135
136     if (!board.name) {
137       this.messageService.showMessage('É necessario informar um nome à prancha');
138       return;
139     }
140
141     let observable = this.boardService.saveBoard(board);
142     if (board._id) {
143       observable = this.boardService.updateBoard(board);
144     }
145
146     let loading: any = this.loadingService.createLoadingPage('Aguarde...');
147     loading.present();
148
149     observable.subscribe(
150       () => {
151         this.navCtrl.push(PlanPage);
152         loading.dismiss();
153       },
154       () => loading.dismiss()
155     );
156   }
```

Fonte: elaborado pelo autor.

O trecho do *script* que realiza a gravação da prancha no banco de dados é demonstrado no Quadro 6, em que é feito um *Create* no *model* `Board` com as informações que foram enviadas pela requisição HTTP, conforme pode ser visto na `linha 5`.

Quadro 6 - Script de gravação da prancha na base de dados

```
4   async store(req, res) {
5     const board = await Board.create(req.body);
6     return res.json(board);
7   }
```

Fonte: elaborado pelo autor.

3.3.1.3 Interação com as pranchas de comunicação

A utilização das pranchas pelos usuários é também implementada dentro da classe `BoardRegister`, no método `selectedImage`. No Quadro 7 é apresentado um trecho de código com a implementação deste método no aplicativo.

Para a implementação desta funcionalidade foi utilizado a API `Audio` fornecida pelo JavaScript. Primeiramente, pode-se notar que é criada uma variável com a instância de um `Audio` do JavaScript na `linha 107`. Após isto é adicionada a URL do áudio que está armazenado no serviço `S3` da Amazon na propriedade `src` do `Audio` na `linha 108`. A partir desta variável pode ser realizada a chamada a função `load` na `linha 109`, que irá fazer o *re-*

load do objeto a fim de garantir que o áudio será iniciado do começo, e a função `play` que irá reproduzir o áudio carregado em memória na `linha 110`.

Quadro 7 - Reprodução de áudio do símbolo

```
106   selectedImage(index) {
107     let audio = new Audio();
108     audio.src = this.boardAudios[index];
109     audio.load();
110     audio.play();
111   }
```

Fonte: elaborado pelo autor.

3.3.1.4 Criação de símbolos

A rotina de criação de um novo símbolo faz uso de dois *plugins* do Ionic, o *plugin* `camera` e o *plugin* `media`. A partir deles foi possível permitir ao usuário a seleção de uma imagem em seu dispositivo e a gravação do áudio que será vinculado ao símbolo. O Quadro 8 apresenta a utilização do *plugin* `media` na implementação da rotina para gravação de um áudio. No método `startRecord` é executada a chamada da função `create`, que recebe como parâmetro o caminho no dispositivo na qual será armazenado o áudio, como visto na `linha 35`.

Quadro 8 - Métodos para gravar áudio

```
31  startRecord() {
32    this.audioFileName = `record${Date.now()}.3gp`;
33
34    this.getFilePath(this.audioFileName);
35    this.audioMediaObject = this.media.create(this.audioFilePath);
36
37    this.audioMediaObject.startRecord();
38    this.recording = true;
39  }
40
41  stopRecord() {
42    this.audioMediaObject.stopRecord();
43
44    let loading: any = this.loadingService.createLoadingPage('Aguarde...');
45    loading.present();
46
47    this.fileService
48      .mediaObjectToBlob(this.audioFilePath, this.audioFileName)
49      .then(response => {
50        this.audio = response;
51        this.recording = false;
52        this.audioChange.emit(this.audio);
53        loading.dismiss();
54      })
55      .catch(() => loading.dismiss());
56  }
```

Fonte: elaborado pelo autor.

Para se ter a informação do local em que será armazenado o áudio, é necessário fazer uma verificação de qual plataforma o usuário está operando, como apresentado entre as `linhas 67` e `71`, pois o caminho será diferente para as duas plataformas suportadas. Por meio

do *plugin* `file` do Ionic é possível se ter o caminho padrão para os diretórios das plataformas Android e iOS, como poder ser visto no Quadro 9.

Quadro 9 - Método getFilePath

```
66  getFilePath(file) {
67    if (this.platform.is('ios')) {
68      this.audioFilePath = this.file.documentsDirectory.replace(/file:\/\//g, '') + file;
69    } else if (this.platform.is('android')) {
70      this.audioFilePath = this.file.externalDataDirectory.replace(/file:\/\//g, '') + file;
71    }
72  }
```

Fonte: elaborador pelo autor.

Após a criação do objeto que terá a instância do *plugin* `media`, será utilizada a função `startRecord`, dando início a gravação do áudio. Para finalizar a gravação do áudio foi disponibilizado o método `stopRecord`. Neste método é executada a função `stopRecord` do *plugin* `media` na linha 42 do Quadro 8. Ele será responsável por criar o arquivo de áudio no diretório anteriormente fornecido para o *plugin* `media`. Ao criar o arquivo no diretório local do dispositivo é chamado o método `mediaObjectToBlob` da classe `FileService`. Este método é responsável por buscar o arquivo no diretório local e convertê-lo para `Blob` como pode ser visto no Quadro 10.

Quadro 10 - Método mediaObjectToBlob

```
39  mediaObjectToBlob(filePath, fileName?, isImage = false): Promise<any> {
40    if (!isImage) {
41      if (this.platform.is('ios')) {
42        filePath = this.file.documentsDirectory + fileName;
43      } else if (this.platform.is('android')) {
44        filePath = this.file.externalDataDirectory + fileName;
45      }
46    }
47    let mime = isImage ? 'image/' : 'audio/';
48
49    return new Promise((resolve, reject) => {
50      let fileName: string;
51      this.file
52        .resolveLocalFilesystemUrl(filePath)
53        .then(fileEntry => {
54          let { name, nativeURL } = fileEntry;
55          let path = nativeURL.substring(0, nativeURL.lastIndexOf('/'));
56          fileName = name;
57          mime += fileName.match(/\.[A-z0-9]+$/i)[0].slice(1);
58          return this.file.readAsArrayBuffer(path, name);
59        })
60        .then(buffer => {
61          let blob = new Blob([buffer], {
62            type: mime
63          });
64          resolve(blob);
65        })
66        .catch(e => reject(e));
67    });
```

Fonte: elaborado pelo autor.

Essa conversão se faz necessário para posteriormente ser possível enviar o arquivo de áudio para o servidor. Este método se comporta em forma de `Promise`, recurso do JavaScript para realizar processamentos assíncronos no código. Isso se faz necessário pois é utilizado o método `readAsArrayBuffer` do *plugin* `file`. Função responsável por transformar o arquivo salvo em diretório local como um *array* de buffers, que se trata de um tipo de dado usado para representar dados binários de tamanho fixo. Para realizar a transformação é necessário informar o caminho do arquivo e seu nome, como visto na linha 58. Ao realizar a transformação, será criado um objeto do tipo `Blob` e atribuído a variável `audio` conforme visto entre as `linhas 41` a `43` do Quadro 11. Para a inserção de imagens no cadastro do símbolo, foi utilizado o *plugin* `camera` do Ionic, no qual suporta a seleção de imagens da galeria do dispositivo móvel. O Quadro 11 apresenta a implementação utilizando o *plugin*.

Quadro 11 - Método newImage

```
24  newImage(): void {      You, 12 days ago • new components
25    let loading: any = this.loadingService.createLoadingPage('Aguarde...');
26    loading.present();
27
28    const options: CameraOptions = {
29      destinationType: this.camera.DestinationType.FILE_URI,
30      sourceType: this.camera.PictureSourceType.PHOTOLIBRARY,
31      allowEdit: true,
32      saveToPhotoAlbum: true,
33      targetWidth: 2210,
34      targetHeight: 2210,
35      quality: 100
36    };
37
38    this.camera
39      .getPicture(options)
40      .then(imagePath =>
41        this.fileService
42          .mediaObjectToBlob(imagePath, '', true)
43          .then(response => (this.image = response))
44      )
45      .then((imageBlob: any) => {
46        var reader = new FileReader();
47        reader.readAsDataURL(imageBlob);
48
49        reader.onloadend = function() {
50          var base64data = reader.result;
51          this.imageB64 = base64data;
52          this.imageChange.emit(this.image);
53          loading.dismiss();
54        }.bind(this);
```

Fonte: elaborado pelo autor.

No método `newImage` é executada a chamada da função `getPicture`, que recebe como parâmetro um objeto do tipo `CameraOptions`, 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 Uniform Resource

Identifier (URI) da imagem no dispositivo, entre outros. Caso não seja especificado nenhuma opção será considerado o conjunto padrão de opções do *plugin*. Ao retornar com sucesso do método `getPicture`, será acionado o método `mediaObjectToBlob` da classe `FileService` para realizar a conversão do arquivo para o tipo `Blob`. O retorno do método `mediaObjectToBlob` é atribuído a variável `image` como pode ser visto entre as linhas 41 e 43. Ao realizar a conversão e atribuição a variável `image` é necessário também uma segunda conversão. Para exibição da imagem na tela de cadastro de símbolo é necessário ter uma imagem em formato `base-64`.

Com intuído de realizar a conversão de `Blob` para `base-64` é utilizado o `FileReader` para criar uma variável com sua instância que fornecera acesso a função `readAsDataUrl`, responsável por transformar o arquivo binário, como visto na linha 47. Após a execução da função `readAsDataUrl` será acionado um *call-back* também da instância do `FileReader`, se trata do `onloadend`, como visto na linha 49. Esse *call-back* irá retornar à informação em `base-64`, na qual será atribuída na variável `imageB64` na linha 51 para possibilitar a exibição no HTML.

3.3.1.5 Envio de arquivos para `Amazon S3`

Para armazenamento dos arquivos de imagem e áudio necessários para as pranchas de comunicação, foi utilizado o serviço de armazenamento de objetos S3 da Amazon. Primeiramente, para que o servidor consiga receber os arquivos eles devem ser enviados como binário, a fim de aprimorar a performance da requisição HTTP. Para isto é utilizada a API `XMLHttpRequest`, que possibilita realizar uma requisição HTTP com `body` do tipo `FormData`. O `FormData` é uma API também do JavaScript que fornece uma maneira mais fácil de construir um conjunto de pares chave/valor representando os campos de um elemento `form` e seus valores, os quais podem ser facilmente enviados utilizando o método `send` do `XMLHttpRequest`. No Quadro 12, entre as linhas 17 e 18, são adicionados os atributos `audioFile` e `imageFile` na instância de uma variável do tipo `FormData`, essas variáveis contém os arquivos de áudio e imagem do tipo `Blob`.

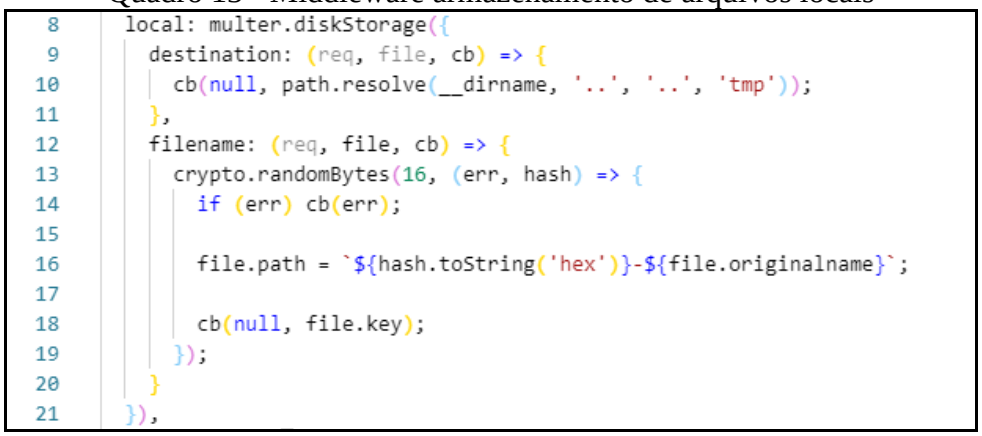
Quadro 12 - Método uploadSymbol

```
11  uploadSymbol(symbolId: string, audioFile: any, imageFile: any): Observable<any> {
12    return Observable.create(observer => {
13      const token = localStorage.getItem('token');
14      const url = `${API.URL}/saveSymbol/${symbolId}`;
15
16      const formData = new FormData();
17      formData.append('audioFile', audioFile, 'record.3gp');
18      formData.append('imageFile', imageFile, 'img.jpg');
19
20      var xhr: XMLHttpRequest = new XMLHttpRequest();
21
22      xhr.onreadystatechange = () => {
23        if (xhr.readyState === 4) {
24          if (xhr.status === 200) {
25            observer.next(JSON.parse(xhr.response));
26            observer.complete();
27          } else {
28            observer.error(xhr.response);
29          }
30        }
31      };
32
33      xhr.open('POST', url, true);
34      xhr.setRequestHeader('Authorization', `Bearer ${token}`);
35      xhr.send(formData);
36    });
37  }
```

Fonte: elaborado pelo autor.

Ao ser efetuada a requisição HTTP com sucesso os arquivos irão passar por um middleware no servidor. Esse middleware é responsável por realizar o envio dos arquivos para o local de armazenamento, sendo localmente caso o servidor esteja sendo executado em uma máquina local, ou para o serviço `S3`. Para que seja possível o envio dos arquivos para o `S3` é necessário a utilização dos *plugins* `multer`, `multer-s3` e `aws-sdk`. O *plugin* `multer-s3` é responsável por lidar como middleware de requisições do com conteúdo `FormData` que são recebidas no servidor e enviá-las ao serviço `S3`. No Quadro 13 é ilustrada a implementação do middleware para salvar os arquivos localmente.

Quadro 13 - Middleware armazenamento de arquivos locais



```
8   local: multer.diskStorage({
9     destination: (req, file, cb) => {
10      cb(null, path.resolve(__dirname, '..', '..', 'tmp'));
11    },
12    filename: (req, file, cb) => {
13      crypto.randomBytes(16, (err, hash) => {
14        if (err) cb(err);
15
16        file.path = `${hash.toString('hex')}-${file.originalname}`;
17
18        cb(null, file.key);
19      });
20    }
21  }),
```

Fonte: elaborado pelo autor.

Entre as linhas 8 e 21 é utilizado o *call-back* `diskStorage` do *plugin* `multer`. O `multer` por sua vez recebe um objeto contendo mais dois *call-backs* opcionais. O primeiro

call-back é o `destination`, responsável por resolver o caminho local do servidor que é informado por parâmetro e salvar o arquivo no diretório. O segundo *call-back* é o `filename`, responsável por realizar um parse do objeto que fora enviado na requisição e que será salvo no banco de dados. Ainda responsável por realizar o envio para o `S3` se tem os *plugins* `aws-sdk` e `multer-s3`. O *plugin* `aws-sdk` é utilizado para criar uma instância com as configurações padrões necessárias para se conectar ao `S3`, e essa instância será informada no *plugin* `multer-s3`. Para realizar o envio ao `S3`, o *plugin* `multer-s3` necessita de algumas informações, que podem serem vistas no Quadro 14 entre as linhas 22 a 33, são elas:

a) `S3`: instância contendo configurações da SDK da Amazon;
b) `Bucket`: nome do `bucket` criado no servidor da Amazon;
c) `Acl`: tipo de leitura dos arquivos que serão enviados para o serviço, está confirmação é opcional;
d) `contentType`: tipo do arquivo que será enviado, foi informado `AUTO_CONTENT_TYPE` para que o próprio `multer` identifique o tipo do arquivo que está sendo enviado;
e) `key`: *call-back* para realizar *parse* do `body` da requisição antes de enviar o objeto que será salvo no banco de dados.

Quadro 14 - Middleware para armazenamento de arquivos no `S3`

```
22    s3: multerS3({
23      s3: new aws.S3(),
24      bucket: process.env.AWS_BUCKET_NAME,
25      contentType: multerS3.AUTO_CONTENT_TYPE,
26      acl: 'public-read',
27      key: (req, file, cb) => {
28        crypto.randomBytes(16, (err, hash) => {
29          if (err) cb(err);
30
31          const fileName = `${hash.toString('hex')}-${file.originalname}`;
32
33          cb(null, fileName);
34        });
35      }
36    })
37  };
```

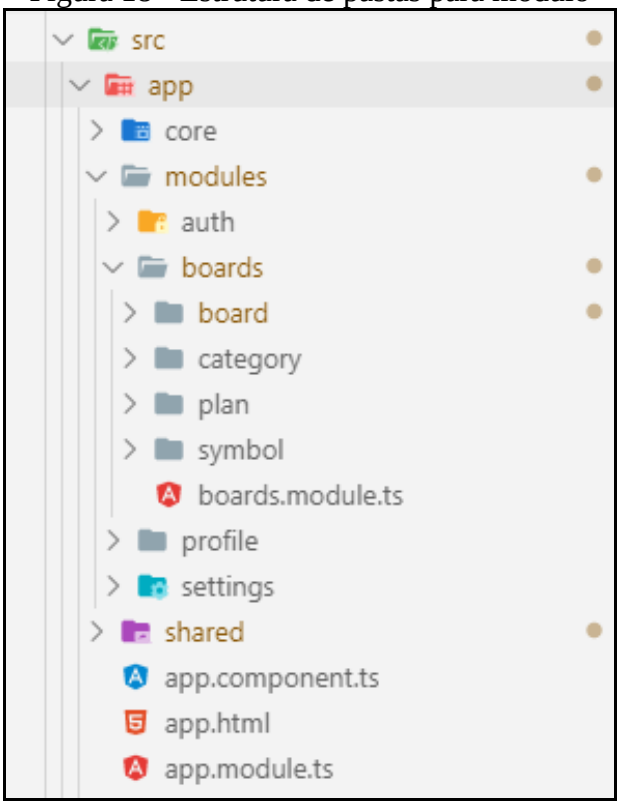
Fonte: elaborado pelo autor.

3.3.1.6 Criação de componentes e módulos

Com intuído de facilitar a implementação de novos módulos no aplicativo Tagarela, foi estabelecido algumas boas práticas a serem seguidas para se desenvolver no aplicativo. Para se adicionar um novo módulo deverá ser criada uma pasta com o nome do módulo na

estrutura estabelecida no projeto. A Figura 18 ilustra a hierarquia de pastas e mostra que os novos módulos devem ser criados dentro da pasta `modules`.

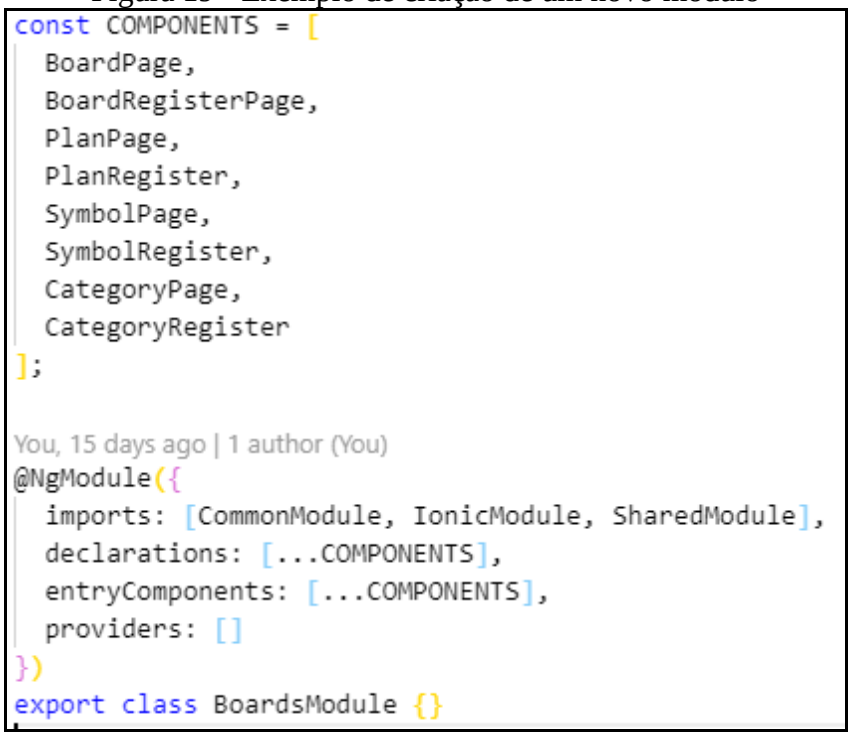
Figura 18 - Estrutura de pastas para módulo



Fonte: elaborador pelo autor.

Após a criação da pasta do módulo deverá ser criado um arquivo com a extensão `module.ts`. Nele irá conter todos os componentes e páginas criados para a sua execução, tais componentes irão ser declarados em uma constante que será atribuído aos atributos `declarations` e `entryComponentes`, responsáveis por tornar eles visíveis em tempo de compilação do aplicativo. Este arquivo é responsável por importar o módulo `SharedModule`, no qual estão declarados todos os componentes reutilizáveis feitos no projeto. Para isto, é necessário informá-lo no atributo `imports`, assim como os módulos `CommonModule` e `IonicModule`, necessários para qualquer tipo de módulo criado utilizando o Ionic. Na Figura 19 pode-se observar a implementação do módulo `BoardsModule`, exemplificando a criação de um novo módulo no Tagarela.

Figura 19 - Exemplo de criação de um novo módulo



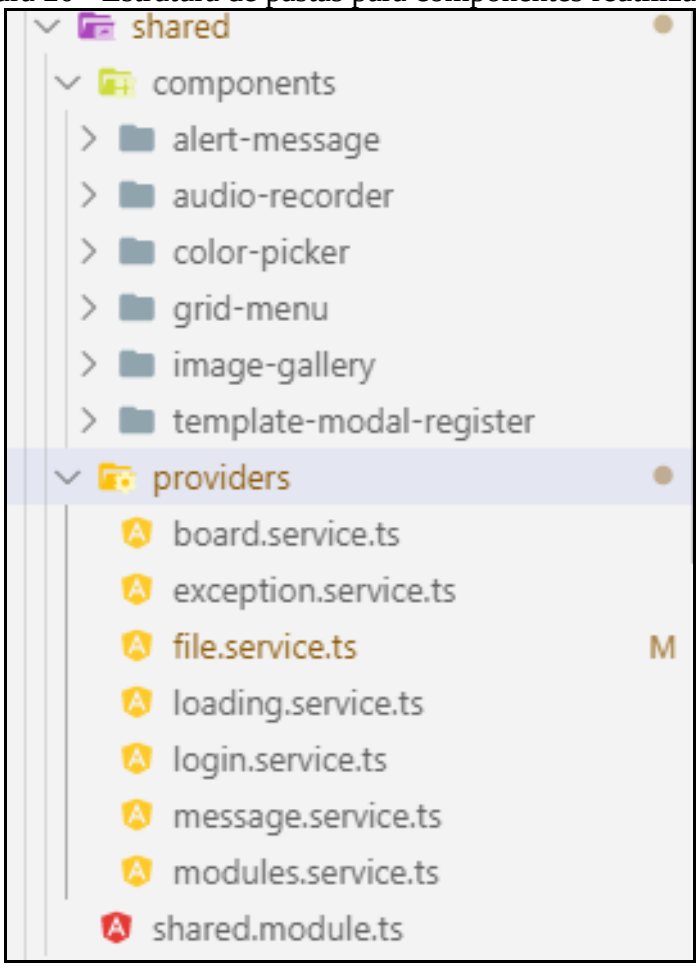
```
const COMPONENTS = [
  BoardPage,
  BoardRegisterPage,
  PlanPage,
  PlanRegister,
  SymbolPage,
  SymbolRegister,
  CategoryPage,
  CategoryRegister
];

You, 15 days ago | 1 author (You)
@NgModule({
  imports: [CommonModule, IonicModule, SharedModule],
  declarations: [...COMPONENTS],
  entryComponents: [...COMPONENTS],
  providers: []
})
export class BoardsModule {}
```

Fonte: elaborado pelo autor.

A organização de todos os componentes e serviços reutilizáveis do projeto é de responsabilidade do o `SharedModule`, que possui duas pastas em sua estrutura. A primeira pasta é a pasta `componentes`, `que` é responsável por todos os componentes que podem ser reutilizáveis entre um ou mais módulos do Tagarela. A segunda pasta é denominada como `providers`, e nela deverá estar presente os serviços que também podem ser utilizados em um ou mais módulos do aplicativo. A Figura 20 ilustra a estrutura de pastas a ser seguida para componentes e serviços reutilizáveis.

Figura 20 - Estrutura de pastas para componentes reutilizáveis



Fonte: elaborador pelo autor.

Para que um componente ou serviço reutilizável seja encontrado pelos demais módulos do aplicativo em tempo de compilação, é necessário realizar as suas declarações. A Figura 21 ilustra as variáveis responsáveis pelas declarações. Para isto, foram criadas as seguintes constantes:

a) `COMPONENTS`: responsável por declarar componentes que podem ser utilizados no HTML de outros módulos;
b) `PROVIDERS`: responsável por declarar serviços que podem ser informados nos construtores de qualquer página, componente ou serviço do aplicativo;
c) `IONIC_PROVIDERS`: responsável por declarar dependências do Ionic para funcionamento geral do aplicativo;
d) `PAGES`: responsável por declarar páginas que podem ser utilizadas por outros módulos no aplicativo.

Figura 21 - Declaração de componentes e serviços reutilizáveis

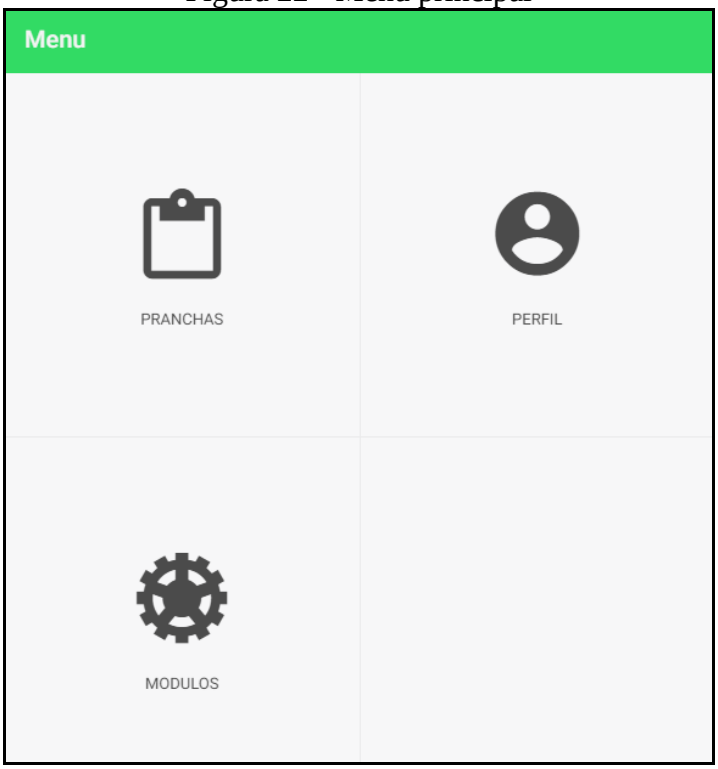
```
const COMPONENTS = [
  ColorPickerComponent,
  TemplateModalRegisterComponent,
  AudioRecorderComponent,
  ImageGalleryComponent,
  AlertMessageComponent
];
const PROVIDERS = [
  LoadingService,
  MessageService,
  ExceptionService,
  LoginService,
  BoardService,
  ModulesService
];
const IONIC_PROVIDERS = [
  StatusBar,
  SplashScreen,
  Media,
  FileTransfer,
  FileTransferObject,
  File,
  Camera,
  FileService
];
const PAGES = [GridMenuComponent];
You, a few seconds ago | 1 author (You)
@NgModule({
  imports: [CommonModule, IonicModule],
  declarations: [...COMPONENTS, ...PAGES],
  exports: [...COMPONENTS],
  entryComponents: [...PAGES],
  providers: [
    ...PROVIDERS,
    ...IONIC_PROVIDERS
  ]
})
export class SharedModule {}
```

Fonte: elaborador pelo autor.

3.3.2 Operacionalidade da implementação

Nesta subseção será demonstrado o funcionamento da implementação por meio de telas e menus. A primeira tela apresentada será o menu do aplicativo, conforme pode ser visto na Figura 22. Nesta tela pode-se verificar os menus disponíveis do aplicativo. O menu `Módulos` é apresentado apenas para o usuário com o papel de `Administrador`. Também há a possibilidade de acessar o `Perfil` e `Pranchas`, na qual está presente o fluxo de pranchas de comunicação.

Figura 22 - Menu principal

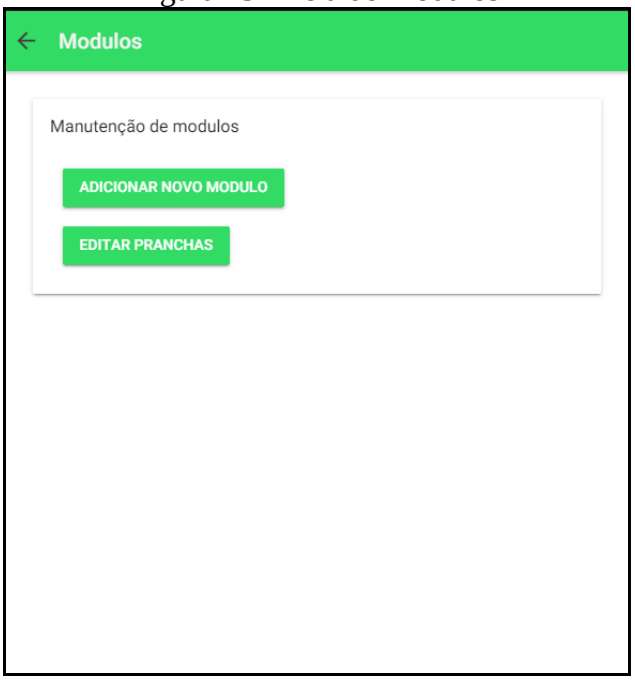


Fonte: elaborado pelo autor.

3.3.2.1 Criação de módulos

Ao entrar no menu `Módulos` o usuário será redirecionado para a tela de `Módulos` conforme Figura 23. Neste momento o usuário terá disponível a lista de módulos já criados, assim como poderá adicionar um novo.

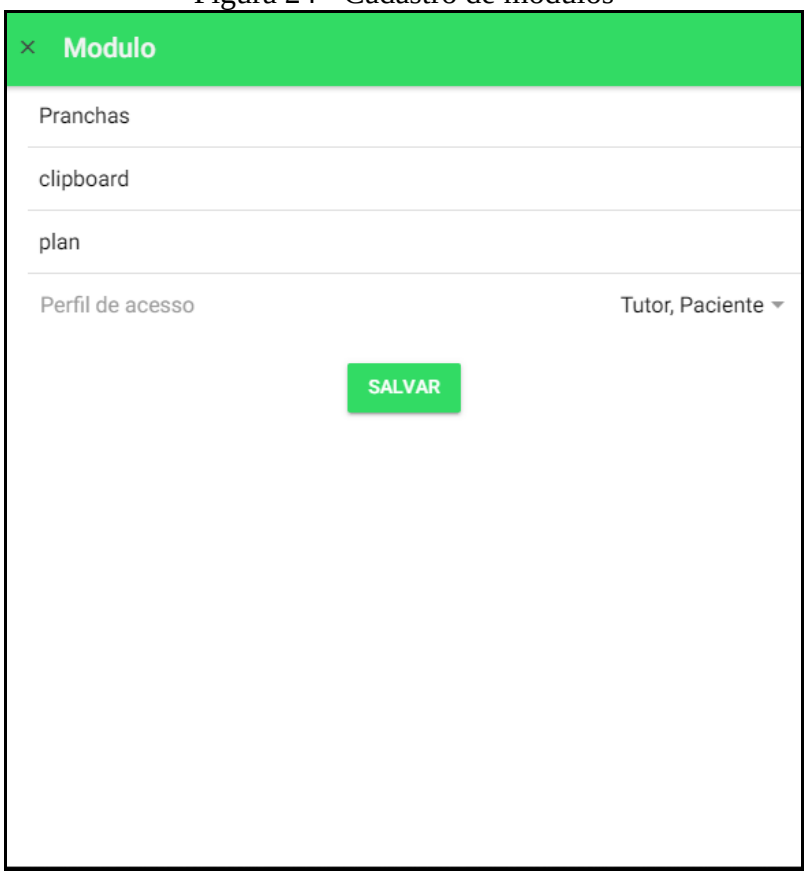
Figura 23 - Tela de Módulos



Fonte: elaborado pelo autor.

Selecionando o botão `Adicionar novo módulo` o usuário será redirecionado para a tela de cadastro de um módulo. Informações do `título`, `nome`, `ícone` e `papeis` serão necessárias. O título será apresentado no menu inicial, assim como o ícone. As características do ícone a ser inserido está disponível na documentação oficial do Ionic, como pode ser visto na Figura 24. Após a criação do módulo ele já estará disponível no menu para qualquer usuário que acessar o aplicativo.

Figura 24 - Cadastro de módulos

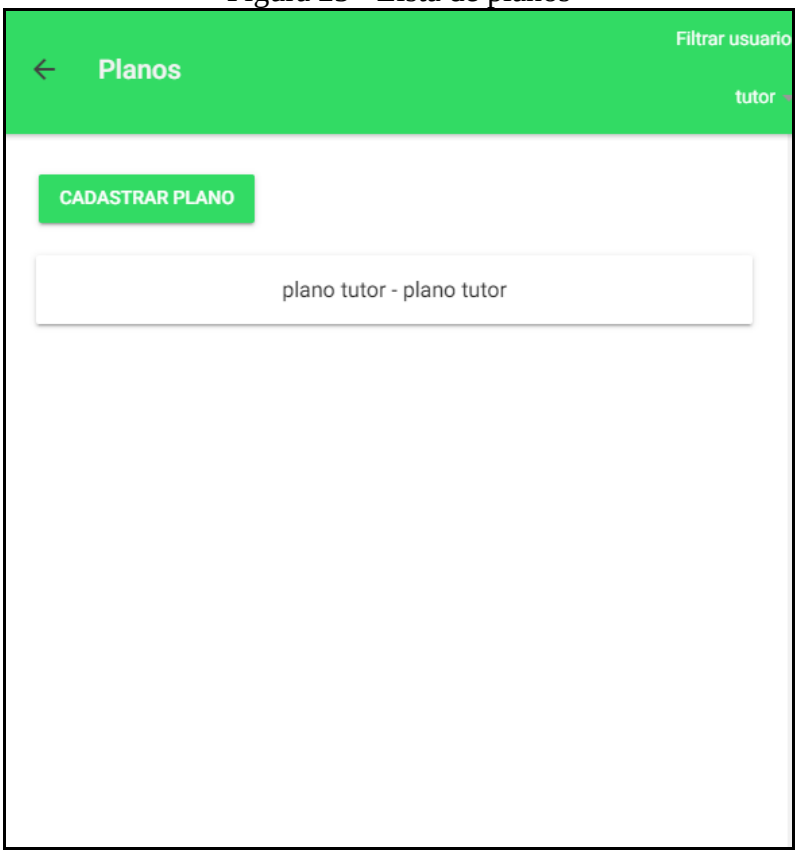


Fonte: elaborado pelo autor.

3.3.2.2 Criação de pranchas de comunicação

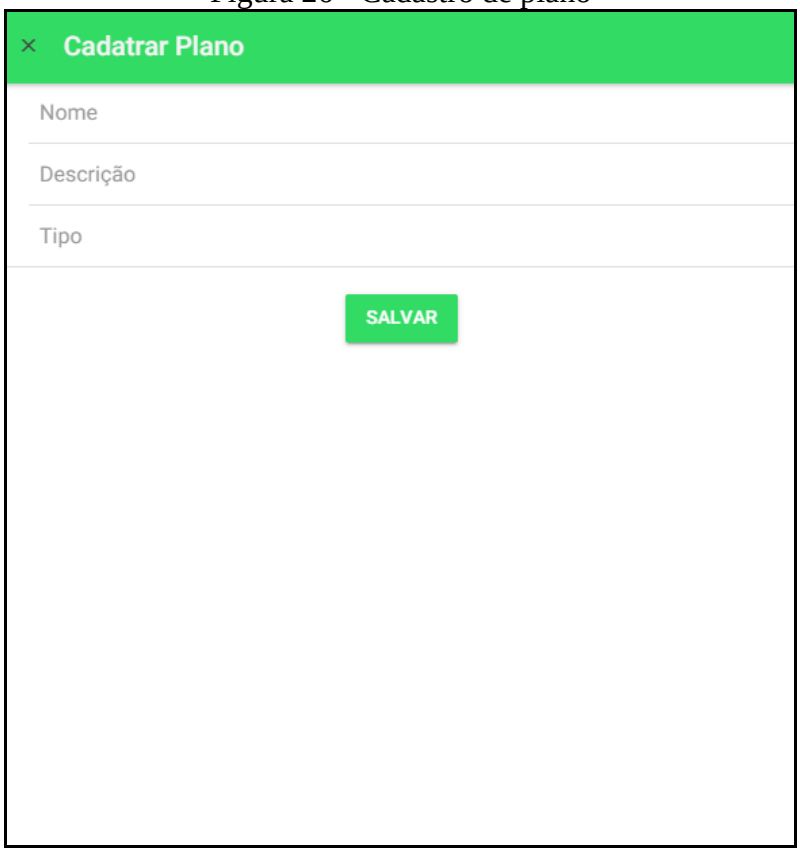
O processo de criação de uma prancha de comunicação é realizado a partir do menu `Pranchas`. Após entrar no menu de `Pranchas` o usuário será redirecionado para a lista de planos, na qual poderá selecionar um dos seus planos ou optar por filtrar por planos de usuários vinculados a ele, conforme Figura 25. No menu superior é disponibilizada a opção de `filtrar usuário`, por meio desta opção o usuário poderá visualizar os planos de quem estiver vinculado a ele. O botão `cadastrar plano` está presente na tela para que o usuário possa adicionar um novo plano. Selecionando o usuário será redirecionado para a tela de `cadastro de plano`. Informações de `nome`, `descrição` e `tipo` serão necessárias para finalizar o cadastro, conforme pode ser visto na Figura 26.

Figura 25 - Lista de planos



Fonte: elaborado pelo autor.

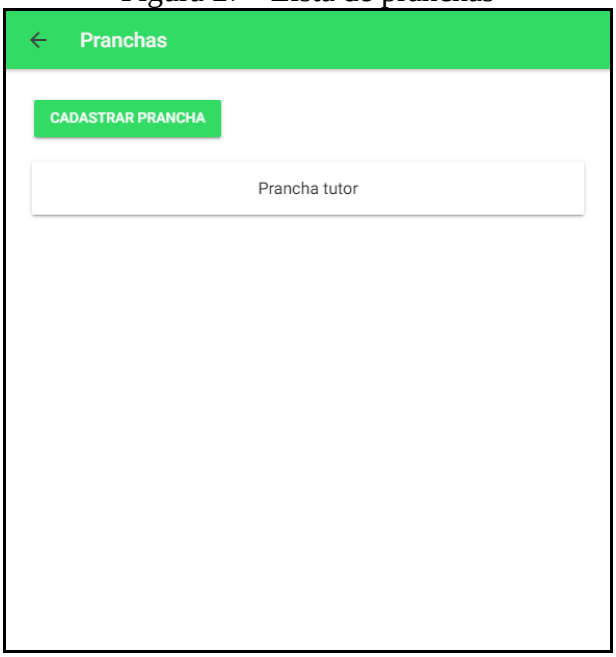
Figura 26 - Cadastro de plano



Fonte: elaborado pelo autor.

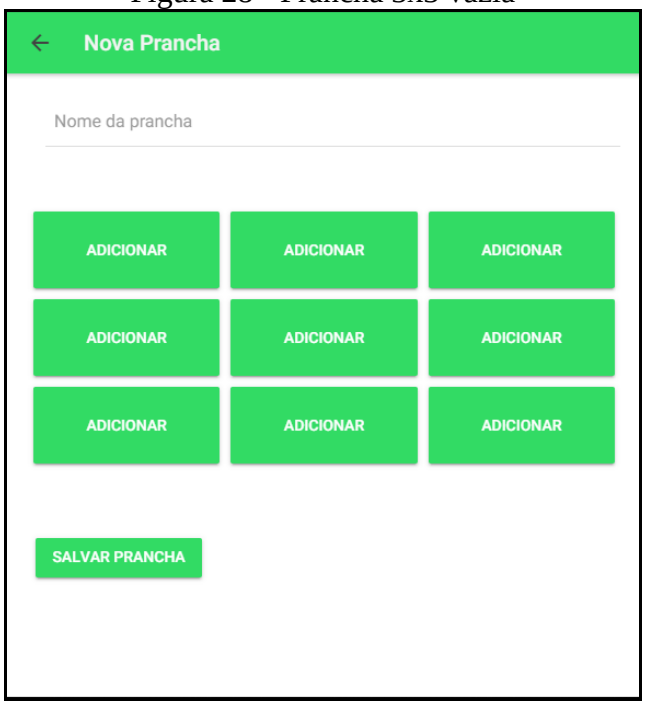
Com a escolha do plano o usuário passa a ter acesso a lista de pranchas vinculadas ao plano escolhido. Esta tela possibilita ao usuário selecionar um item da lista ou adicionar uma nova prancha por meio do botão cadastrar prancha (Figura 27). Ao pressionar o botão `cadastrar prancha` será mostrado ao usuário uma prancha 3x3 vazia (Figura 28). O usuário então deve selecionar dentro de uma categoria quais os símbolos farão parte desta nova prancha pressionando o botão `Adicionar`.

Figura 27 - Lista de pranchas



Fonte: elaborado pelo autor.

Figura 28 - Prancha 3x3 vazia



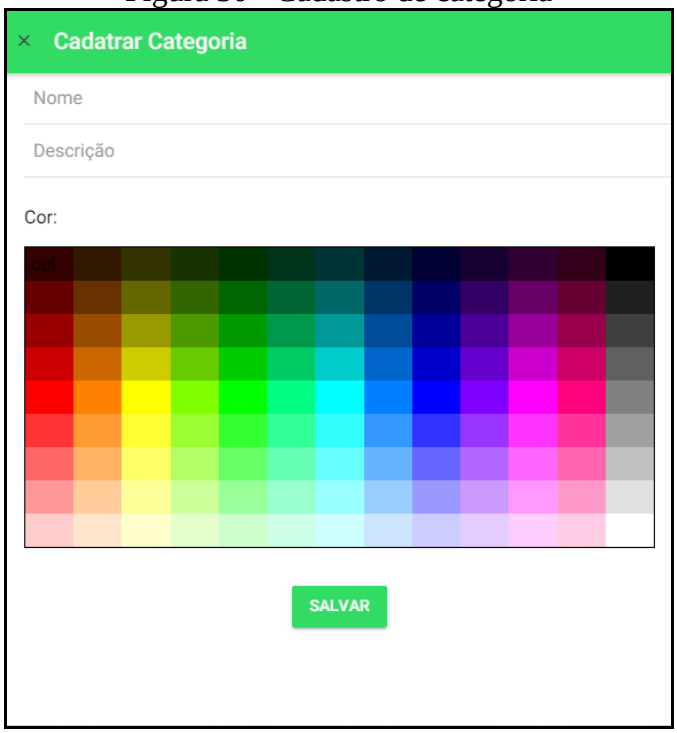
Fonte: elaborado pelo autor.

A seleção do símbolo é realizada a partir de uma lista de categorias definidas pelos usuários (Figura 29). O botão `cadastrar categoria` está presente na tela para que o usuário possa adicionar uma nova categoria. Selecionando este botão o usuário será redirecionado para a tela de `cadastro de categoria`. Informações de `nome`, `descrição` e `cor da categoria` serão necessárias para finalizar o cadastro (Figura 30).

Figura 29 - Lista de categorias

Fonte: elaborado pelo autor.

Figura 30 - Cadastro de categoria

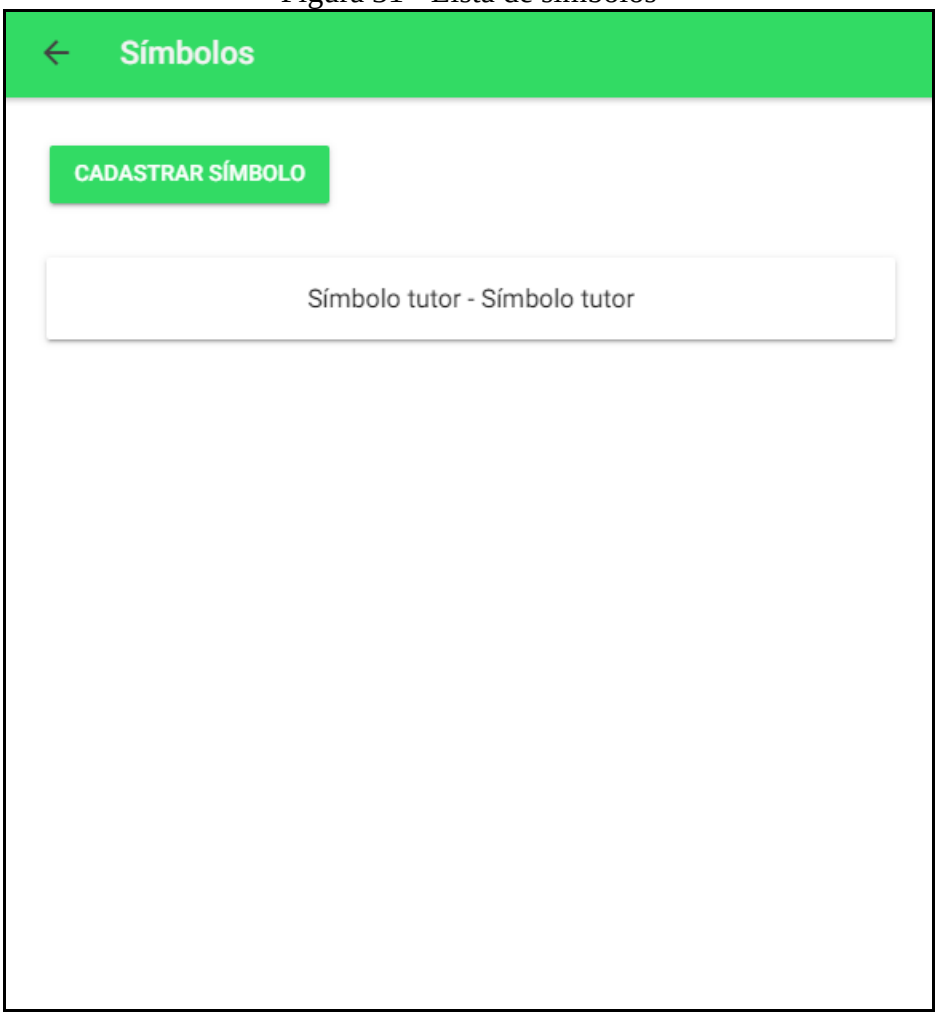


Fonte: elaborado pelo autor.

Com a escolha da categoria o usuário passa a ter acesso a lista de símbolos vinculados a categoria escolhida. Esta tela possibilita ao usuário selecionar um item da lista ou adicionar um novo símbolo por meio do botão `cadastrar símbolo` (Figura 31) e o seu cadastro pode ser visto na Figura 32. Um símbolo contém as seguintes informações:

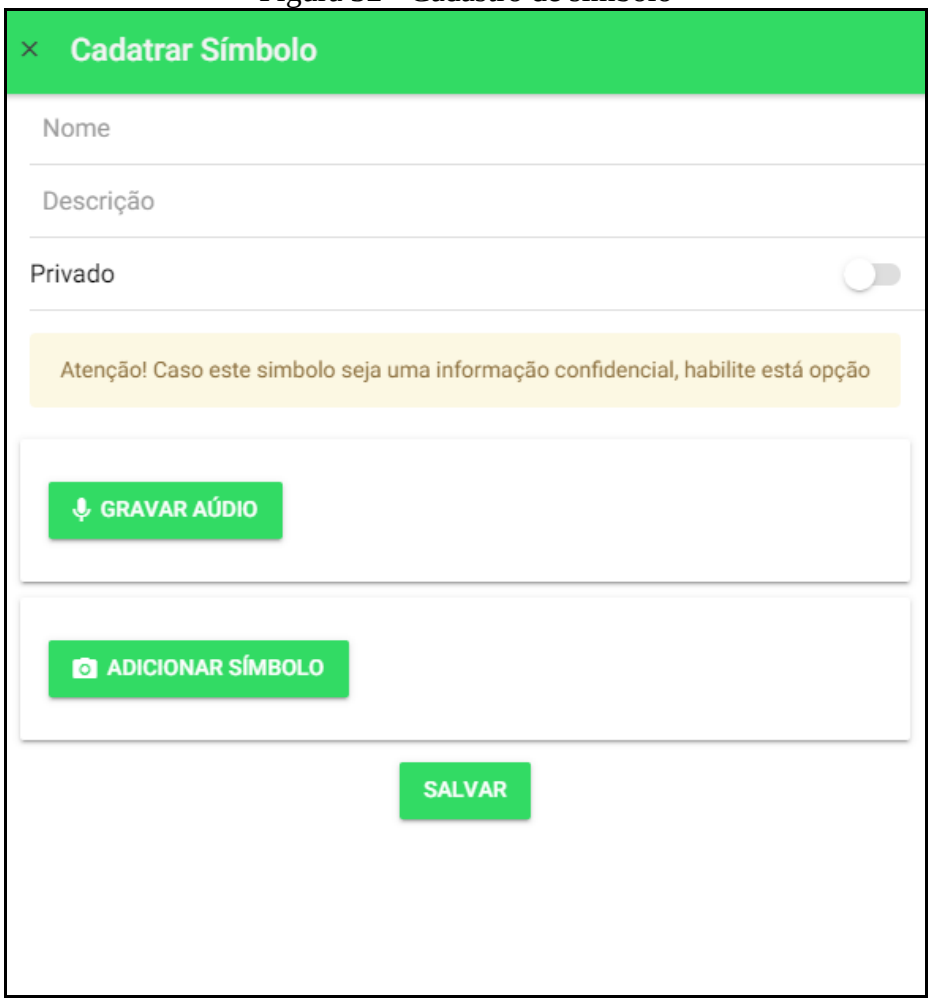
a) `Nome`;
b) `Descrição`;
c) `Indicativo`: se o símbolo é privado, para que ele não seja exibido para outros usuários;
d) `Áudio`: que é gravado a partir do microfone do dispositivo móvel, para testar o áudio gravado pode ser pressionado o botão de play que é disponibilizado após a gravação;
e) `Imagem`: que é escolhida a partir da galeria de imagens do dispositivo móvel.

Figura 31 - Lista de símbolos



Fonte: elaborado pelo autor.

Figura 32 - Cadastro de símbolo

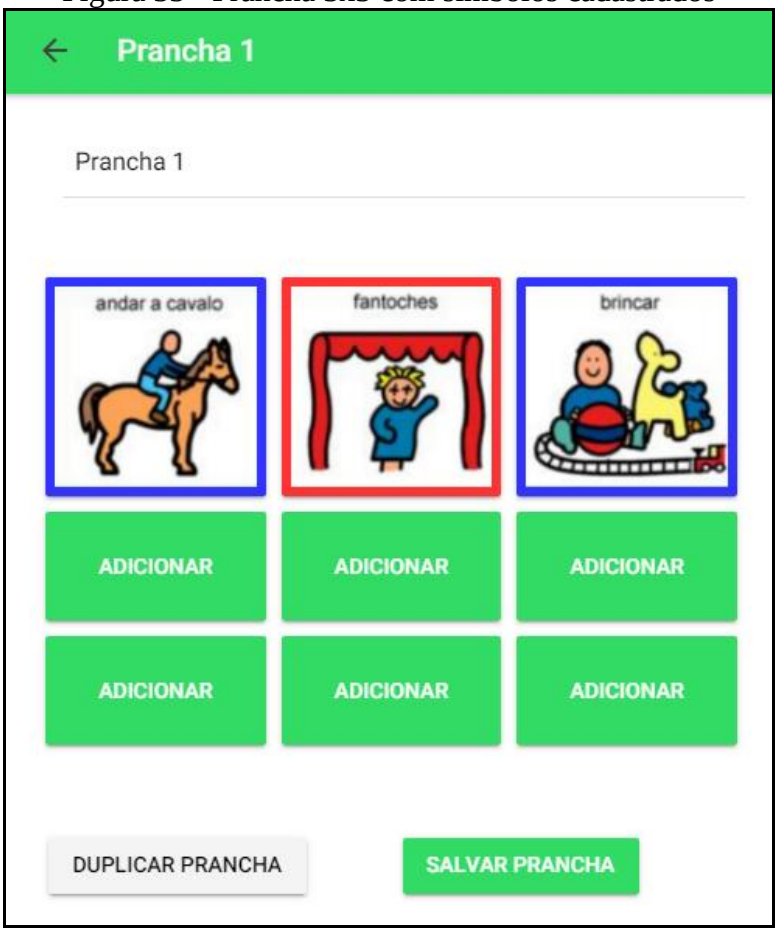


Fonte: elaborado pelo autor.

3.3.2.3 Utilização das pranchas de comunicação

A interação com as pranchas de comunicação é realizada a partir da lista de pranchas do usuário. Primeiramente, deve-se selecionar um plano de atividades, objeto que agrupa as pranchas de comunicação, para então selecionar a prancha desejada, conforme demonstrado na Figura 33. E assim será apresentada uma tela para o usuário com os símbolos da prancha 3x3. Ao pressionar um símbolo o áudio vinculado a ele será reproduzido. As posições da prancha que não possuírem símbolos ficarão com o botão `adicionar` e não reproduzirão nenhum áudio ao ser pressionado. O botão redirecionara o usuário para a lista de categorias para que possa cadastrar um novo símbolo naquela posição.

Figura 33 - Prancha 3x3 com símbolos cadastrados



Fonte: elaborado pelo autor.

3.4 RESULTADOS E DISCUSSÕES

Nesta seção são apresentados os testes de usabilidade realizados com a arquitetura de componentes do aplicativo e uma comparação do trabalho desenvolvido com os trabalhos correlatos. Os testes deste trabalho podem ser na utilização das pranchas de comunicação e criação de módulos e componentes. Em relação a utilização das pranchas de comunicação não se justifica fazer testes pois segue a refatoração usando os mesmos padrões utilizados em trabalhos anteriores como o de Wippel (2015). Na parte de criação de módulos e componentes foi realizado um questionário a fim de testar a usabilidade da arquitetura de componentes para o desenvolvimento de novas funcionalidades no aplicativo.

3.4.1 Testes de usabilidade

A fim de testar a usabilidade da arquitetura do aplicativo, foi solicitado para três programadores desenvolverem um simples módulo seguindo os padrões de desenvolvimento estabelecidos. A sua avaliação foi dividida em três etapas, sendo elas a avaliação dos conhecimentos dos programadores, avaliação do desenvolvimento de um novo módulo,

avaliação do desenvolvimento de um novo módulo e a avaliação geral da arquitetura (Apêndice A).

3.4.1.1 Avaliação dos conhecimentos dos programadores

Inicialmente foram avaliados os conhecimentos dos programados em relação as tecnologias necessárias para o desenvolvimento de aplicações utilizando o aplicativo. Os resultados obtidos a partir dessa avaliação são apresentados na Tabela 1.

Tabela 1 - Avaliação do conhecimento das tecnologias

Qual o seu nível de escolaridade?	0% Ensino médio incompleto. 0% Ensino médio completo. 100% Ensino superior incompleto. 0% Ensino superior completo.
Como você avalia o seu conhecimento em relação ao Ionic f*ramework*?	0% Não conheço o Ionic f*ramework*. 33,3% Conheço, mas nunca implementei nada. 33,3% Conheço e já desenvolvi pequenas aplicações. 33,3% Conheço bem, já desenvolvi aplicações complexas.
Como você avalia o seu conhecimento em relação a Arquitetura de componentes?	0% Não conheço. 0% Conheço o conceito, mas nunca utilizei. 100% Conheço e já utilizei.
Como você avalia o seu conhecimento em relação ao Angular?	0% Não conheço o Angular. 0% Conheço, mas nunca implementei nada. 66,7% Conheço e já desenvolvi pequenas aplicações. 33,3% Conheço bem, já desenvolvi aplicações complexas.

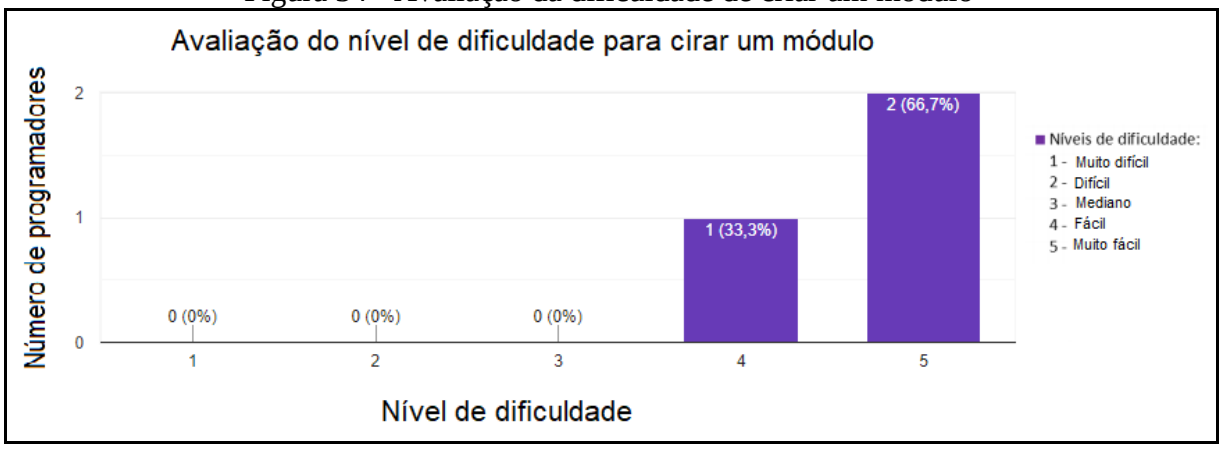
Fonte: elaborado pelo autor.

A partir dos dados apresentados, pode-se observar que apenas um programador possui conhecimento no Ionic f*ramework*, porém todos eles possuem conhecimentos em Angular e Arquitetura de componentes, levando em consideração que o Ionic utilizado do Angular para seu desenvolvimento, todos eles têm os conhecimentos básicos necessários para desenvolver na arquitetura do aplicativo. É necessária esta avaliação pois caso o programador não possuísse os conhecimentos básicos necessários, isto iria gerar impactos negativos no momento do desenvolvimento de um novo módulo no aplicativo.

3.4.1.2 Avaliação do desenvolvimento de um novo módulo

A Figura 34 apresenta os resultados obtidos em relação ao nível de dificuldade para o desenvolvimento de um novo módulo no aplicativo. A partir dos resultados apresentados, é possível constatar que não houve problemas na criação de um novo módulo, podendo assim se afirmar que a arquitetura desenvolvida possui um uso fácil e prático para o desenvolvimento de futuras funcionalidades do aplicativo.

Figura 34 - Avaliação da dificuldade de criar um módulo

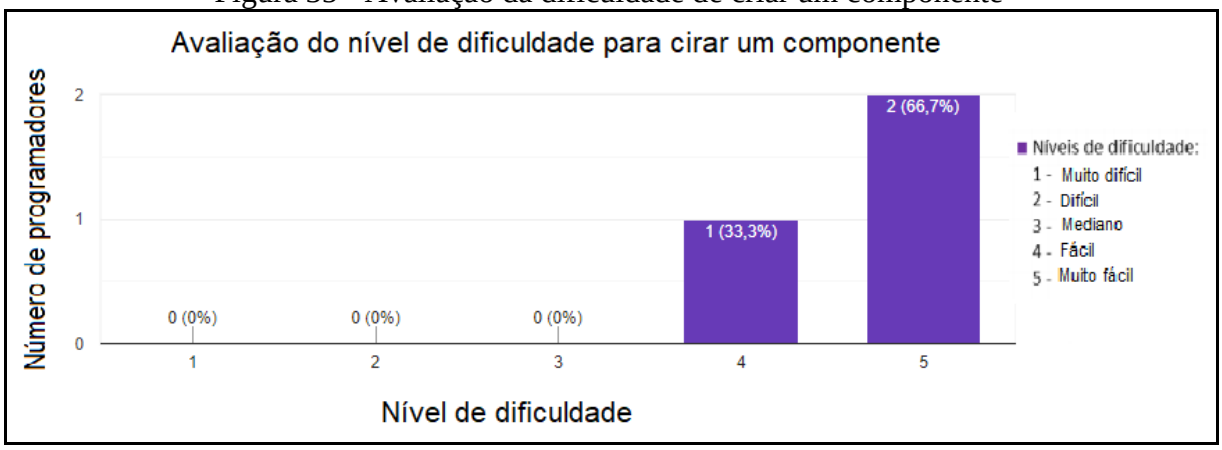


Fonte: elaborador pelo autor.

3.4.1.3 Avaliação do desenvolvimento de um novo componente

A Figura 35 apresenta os resultados obtidos em relação ao nível de dificuldade para o desenvolvimento de um novo componente no aplicativo. É possível constatar que não houve problemas na criação de um novo componente, podendo assim se afirmar que programadores que utilizarem a arquitetura de componentes possivelmente não enfrentaram problemas ao tentar desenvolver um novo.

Figura 35 - Avaliação da dificuldade de criar um componente

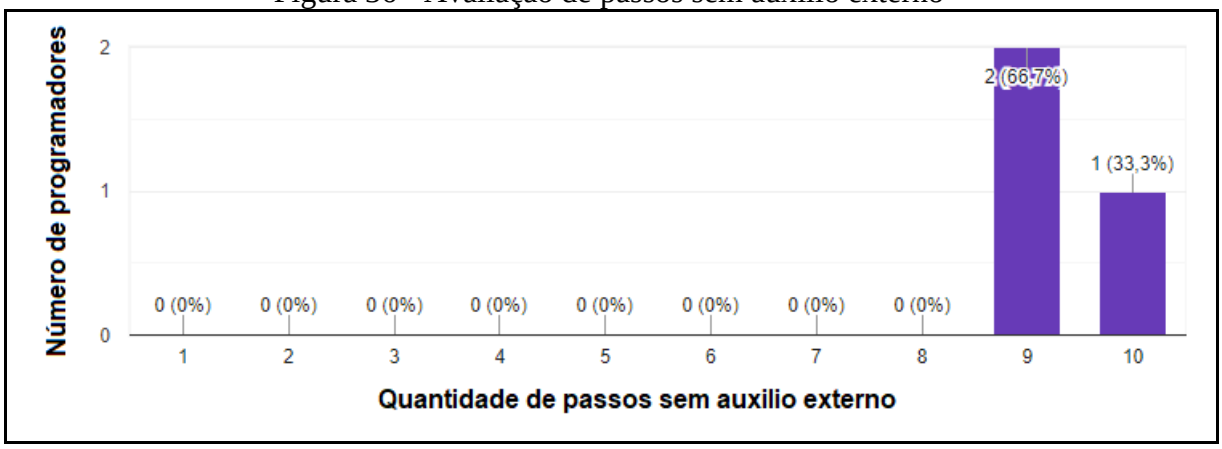


Fonte: elaborado pelo autor.

3.4.1.4 Avaliação geral da arquitetura

Por fim, foi realizada uma avaliação geral da arquitetura, que teve o objetivo de avaliar a utilidade de modo geral, bem como a dificuldade de execução de todos os passos para se ter uma nova tela funcional no aplicativo. A Figura 36 apresenta a avaliação de passos concluídos pelo programador sem auxilio externo, a partir dela é possível afirmar que os programadores não possuíram dificuldades em executar os passos necessários para desenvolver uma nova funcionalidade no aplicativo.

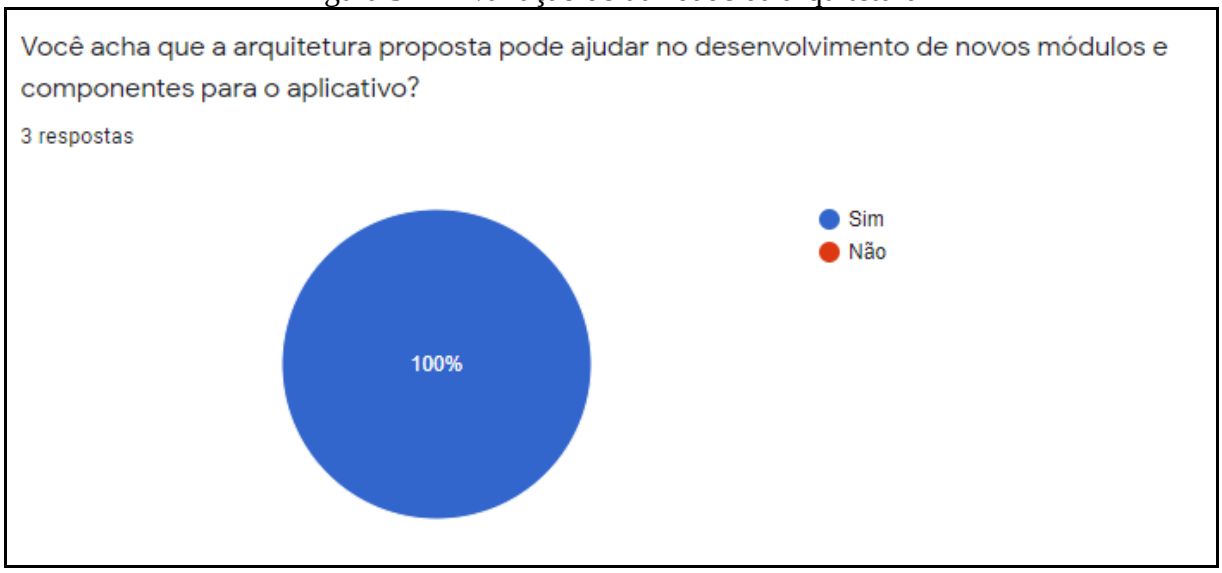
Figura 36 - Avaliação de passos sem auxílio externo



Fonte: elaborado pelo autor.

A Figura 37, que apresenta se a arquitetura proposta auxilia no desenvolvimento e gera produtividade para a implementação de novas funcionalidades no aplicativo, reforça a afirmação da utilidade da arquitetura proposta.

Figura 37 - Avaliação de utilidade da arquitetura



Fonte: elaborado pelo autor.

3.4.2 Comparativo entre os trabalhos correlatos

É realizada nesta subseção uma comparação entre os trabalhos correlatos apresentados na seção 2.4 e o trabalho presente, sendo possível visualizar pelo Quadro 15 as características dos trabalhos correlatos relacionadas com a arquitetura e funcionalidades desenvolvidas no aplicativo. Observa-se pelo quadro comparativo que o trabalho alcançou seus objetivos, em virtude de todas as características destacadas dos trabalhos correlatos terem sido implementadas.

Quadro 15 - Características dos trabalhos relacionados

Trabalhos Relacionados / Características	Fleximize (2018)	Civiam (2014)	Passerino (2015)	Trabalho presente
Tratamento diferenciado para perfis distintos de usuários	X	X	X	✓
Prancha de comunicação	X	X	✓	✓
Acessível de dispositivos móveis	✓	✓	✓	✓
Reprodução de áudio ao interagir	X	X	✓	✓
Multi-módulos	✓	X	✓	✓
Acesso à galeria de imagens	✓	✓	✓	✓

Fonte: elaborado pelo autor.

Cada uma das características destacadas pode ser alcançada de diferentes formas com o trabalho desenvolvido. O `tratamento diferenciado para perfis distintos de usuários` foi alcançado com a possibilidade de cadastros de perfis distintos para que somente os usuários do perfil `Paciente` possa fazer o vínculo com seus tutores, enquanto a `Prancha de comunicação` foi alcançada da mesma forma que o trabalho de Passerino (2014), implementação do módulo de pranchas de comunicação para que os usuários possam interagir com elas.

A característica `Acessível de dispositivos moveis` está presente no trabalho presente neste trabalho pois é disponibilizado um aplicativo móvel, assim como os demais trabalhos relacionados. A `Reprodução de áudio ao interagir` como em Passerino (2014), foi alcançada pela implementação da interação com imagens na prancha de comunicação, dado que ao usuário clicar em alguma imagem, o áudio relacionado a aquela imagem será reproduzido.

A característica de `Multi-módulos` como em Fleximе (2018) e Passerino (2015), foi alcançada pela implementação da funcionalidade de cadastro de módulos, possibilitando ao usuário Administrador adicionar novos módulos ao aplicativo. Além do cadastro de módulos, temos a arquitetura do aplicativo que possibilita a inserção de novos módulos independentes dos já criados, seguindo os padrões definidos para facilitar a implementação futura de novos

módulos. Por fim o `Acesso à galeria de imagens`, Fleximize (2018), Civiam (2014) e Passerino (2015) trazem a possibilidade de adicionar imagens por meio da galeria durante o cadastro de um novo símbolo para as pranchas de comunicação.

4 CONCLUSÕES

Neste trabalho é apresentado o Tagarela, uma nova versão do aplicativo desenvolvido por Wippel (2015), com uma arquitetura baseada em componentes. Trata-se da refatoração de um aplicativo de uma maneira em que seja possível a inserção de novos módulos, bem como a aplicação de conceitos para facilitar reutilização de recursos já implementados no aplicativo.

O objetivo geral descrito na seção 1.1 para este trabalho foi desenvolver um aplicativo que possibilite a inserção de mais de um módulo, sendo cumprida conforme feedback dos programadores apresentados na seção 3.4. Ademais, foram desenvolvidos os objetivos específicos descritos para este trabalho.

Referente ao objetivo específico de implementar uma interface para adicionar novos módulos, foi implementada a interface conforme apresentado na seção 3.3.2.1. Esta interface tem seu acesso para o usuário com papel de Administrador, possibilitando que um novo módulo seja adicionado e visualizado no aplicativo. O objetivo específico de controlar a visibilidade dos módulos, foi realizada a implementação de um controle de quais módulos será apresentado no menu para o usuário, ele pode controlar os módulos exibidos pelo seu perfil. O objetivo específico de refatoração do módulo de prancha de comunicação foi realizado pela implementação do módulo Pranchas, disponibilizado para o usuário pelo menu principal.

A fundamentação teórica deste trabalho representou parte essencial do desenvolvimento deste trabalho, pois esta trouxe conhecimento, e com este foi possível materializar a arquitetura de componentes dentro de um aplicativo híbrido. A fundamentação teórica deste trabalho também serve também como base para trabalhos futuros referentes a arquitetura de componentes aplicado à desenvolvimento móvel híbrido. Ainda por se tratar de uma arquitetura, muito do que se imagina são apenas conceitos, porem estes puderam ser aplicados e demonstram vantagens na utilização.

As ferramentas utilizadas no desenvolvimento, de forma geral, se mostraram adequadas para a criação de um aplicativo de comunicação alternativa multiplataforma. O uso do Ionic f*ramework* foi efetivo para o aplicativo obter um comportamento equivalente nas plataformas trabalhadas. Os *plugins* `camera` e `media` se mostraram eficazes para realizar a atualização dos arquivos de áudio e imagem entre o dispositivo local e o servidor.

As ferramentas utilizadas no servidor também se mostraram efetivas por meio da utilização do servidor Heroku Cloud Platform para hospedagem das APIs, o mLab para o banco de dados MongoDB e o Amazon S3 para armazenamento de arquivos. Com a utilização

destas ferramentas foi possível realizar o armazenamento em nuvem e disponibilização de um domínio próprio, gratuito sem que houvesse indisponibilidade durante o desenvolvimento do trabalho.

Este trabalho justifica-se pela necessidade de um aplicativo que possibilite a expansão com novas funcionalidades de forma integrada. A necessidade da implementação ocorre pela universidade sempre buscar novas formas de aprimorar a capacidade de comunicação e auxiliar inclusão escolar de pessoas com alguns tipos de deficiência.

Acredita-se que a principal contribuição deste trabalho é referente disponibilizar um aplicativo de comunicação alternativa acessível e multiplataforma. Este aplicativo permite a criação e interação do usuário com pranchas de comunicação, junto a criação de perfis distintos para que o aplicativo melhor se adeque a necessidade do usuário e que possa ser desenvolvida novas funcionalidades em um ambiente de desenvolvimento único e integrado. Suas principais limitações estão na capacidade de utilização off-line, registro de históricos e anotações do usuário.

4.1 EXTENSÕES

São sugeridas as seguintes extensões para a continuidade do trabalho:

a) implementar a persistência dos dados locais no dispositivo utilizando a tecnologia SQLite;
b) atualização do Ionic *framework* para a versão 4.0;
c) estender a plataforma para uma versão Web*;*
d) tornar a interface do aplicativo mais acessível;
e) adicionar captura de interações com os módulos para registrar históricos;
f) adicionar forma de anotação para que o Tutor possa manter um controle sobre as atividades do paciente.

REFERÊNCIAS

BERSCH, Rita et al. **Atendimento educacional especializado – Deficiência Física**. Brasília, 2007. Disponível em: http://portal.mec.gov.br/seesp/arquivos/pdf/aee_df.pdf. Acesso em: 15 set. 2018.

BERSCH, Rita. **Introdução à tecnologia assistiva**. 2017. Disponível em: http://www.assistiva.com.br/Introducao_Tecnologia_Assistiva.pdf. Acesso em: 11 nov. 2019.

CARNIEL, Andrei et al. O uso da comunicação aumentativa e alternativa para apoiar o diálogo de pessoas com deficiência intelectual. **Revista Brasileira de Computação Aplicada**, [s.l.], v. 10, n. 1, p.53-65, 1 maio 2018. UPF Editora. http://dx.doi.org/10.5335/rbca.v10i1.7678.

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-comunicac-o-alternativa-para-ipad-e-windows.html. Acesso em: 05 set. 2018.

CORDOVA. **Apache Cordova**. [S.l], 2018. Disponível em: https://cordova.apache.org/. Acesso em: 24 set. 2018.

DEVICELAB. **Conheça as diferenças entre aplicativos nativos, mobile e híbridos**. [S.l], 2018. Disponível em: http://www.devicelab.com.br/2018/06/22/app-nativo_web-app. Acesso em: 24 set. 2018.

EXAME. **A bilionária Movile, dona do iFood, quer sua atenção total**. [S.1.], 2018. Disponível em: https://exame.abril.com.br/negocios/a-bilionaria-movile-do-ifood-quer-sua-atencao-total-tera-2/. Acesso em: 25 set. 2018.

EXAME. **Superaplicativo permite pedir comida, pagar conta, chamar táxi**. [S.1.], 2017. Disponível em: https://exame.abril.com.br/revista-exame/superaplicativo-permite-pedir-comida-pagar-conta-chamar-taxi. Acesso em: 09 set. 2018.

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.

FEIJÓ, Rafael Heider Barros. **Uma arquitetura de software baseada em componentes para visualização de informações industriais.** 2017. 88 f. Monografia (Especialização) - Curso de Pós-graduação em Engenharia Elétrica, Universidade Federal de Rio Grande do Norte, Natal, 2017. Disponível em: http://wdz.eng.br/ArqSoftComponeteVisualizacao.pdf. Acesso em: 08 abr. 2019.

FLEXIMIZE. **The Super-App That's Transforming Tech**. [S.1.], 2018. Disponível em: https://fleximize.com/articles/006663/chinese-super-app-changing-tech. Acesso em: 09 set. 2018.

GALVÃO FILHO, Teófilo Alves. **Tecnologia Assistiva para uma Escola Inclusiva:** Apropriação, Demandas e Perspectivas. 2009. Disponível em: https://repositorio.ufba.br/ri/bitstream/ri/10563/1/Tese%20Teofilo%20Galvao.pdf. Acesso em: 11 nov. 2019.

GOOGLE PLAY. **Rapiddo:** delivery e recarga com cashback. [S.1.], 2018. Disponível em: https://play.google.com/store/apps/details?id=com.lamppy.recargabonus&hl=pt_BR/. Acesso em: 25 set. 2018.

IBGE. **IBGE Instituto Brasileiro de Geografia e Estatística**. [S. l.], 2018. Disponível em: http://www.ibge.gov.br/home/. Acesso em 15 set. 2018.

IONIC FRAMEWORK. **Build Amazing Native Apps and Web Apps with Ionic Framework and Angular**. [S.l], 2018. Disponível em: https:// https://ionicframework.com/. Acesso em: 24 set. 2018.

KLEEMANN, Jean Wagner. **Engenharia de Componentes.** 2019. Disponível em: http://www.linhadecodigo.com.br/artigo/3119/engenharia-de-componentes-parte-1.aspx. Acesso em: 08 abr. 2019.

MADUREIRA, Daniel. **Aplicativo nativo, web App ou aplicativo híbrido?** [S.l.], 2017. Disponível em: https://usemobile.com.br/aplicativo-nativo-web-hibrido/. Acesso em: 11 nov. 2019.

MAHATO, Rakesh. **Hybrid Mobile Application Development**. 2016. 41 f. Tese (Doutorado) - Curso de Bachelor Of Engineering, Information Technology, Helsinki Metropolia University Of Applied Sciences, Helsínquia, 2016. Disponível em: http://www.theseus.fi/bitstream/handle/10024/110286/Mahato_Rakesh.pdf. Acesso em: 24 set. 2018.

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.

MEDEIROS, Mariane Affonso. **Otimização de arquitetura de software utilizando sistema de colônia de formigas.** 2016. 60 f. Tese (Doutorado) - Curso de Bacharelado em Ciência da Computação, Universidade Tecnológica Federal do Paraná, Campo Mourão, 2016. Disponível em: http://repositorio.roca.utfpr.edu.br/jspui/bitstream/1/5170/1/CM_COCIC_2016_1_02.pdf. Acesso em: 21 nov. 2019.

OLIVEIRA, Luiza M. B. **Cartilha do censo 2010 - pessoas com deficiência**. Brasília, 2012. Disponível em: http://www.unievangelica.edu.br/novo/img/nucleo/cartilha-censo-2010-pessoas-com-deficienciareduzido.pdf. Acesso em Set. 2018.

PASSERINO, Liliane M.; BEZ, Maria R. **Comunicação alternativa**: Mediação para uma inclusão social a partir do SCALA; Rio Grande do Sul: Editora UPF, 2015. 323 p.

PAULINO, Fábio. **Design and development of a hybrid-based mobile app for ISCTEIUL**. 2015. 88 f. Dissertação (Mestrado) - Curso de Mestrado em Ciência da Computação, ISCTE-IUL, Lisboa, 2015. Disponível em:< https://repositorio.iscte-iul.pt/handle/10071/11163 >. Acesso em: 24 set. 2018.

SANTOS, Sérgio Adriano Oliveira. **Um Estudo sobre o Desenvolvimento Baseado em Componentes**: Uma visão prática. 2015. 39 f. Monografia (Especialização) - Curso de Licenciatura em Computação, Universidade Estadual da Paraíba, Campina Grande, 2015. Disponível em: http://dspace.bc.uepb.edu.br/jspui/bitstream/123456789/8169/1/PDF%20-%20S%c3%a9rgio%20Adriano%20Oliveira%20Santos.pdf. Acesso em: 21 nov. 2019.

SAUDE. **Autismo**. [S.1.], 2018. Disponível em: https://saude.abril.com.br/mente-saudavel/o-que-e-autismo-das-causas-aos-sinais-e-o-tratamento. Acesso em: 09 set. 2018.

TAGARELA. **Tagarela:** rede social de comunicação alternativa. Blumenau, 2014. Disponível em: http://gcg.inf.furb.br/tagarela. Acesso em: 05 nov. 2018.

TEIXEIRA, Milene. **A importância da fala.** Primavera do Leste, 2013. Disponível em: http://www.cliquef5.com.br/conteudo.php?cid=30649. Acesso em: 15 set. 2018.

TOBII Sono Flex - **Symbol Vocabulary App for iPad and iPhone**. S.i.: Tobii Aac, 2011. Son., color. Disponível em: https://www.youtube.com/watch?v=ibgSEx4AIIg. Acesso em: 04 nov. 2018

VENTEU, Kelly Cristina; PINTO, Giuliano Scombatti. Desenvolvimento Móvel Híbrido. **Revista Interface Tecnológica**, [s.l.], v. 15, n. 1, p.11-96, 12 jul. 2018. Interface Tecnologica. http://dx.doi.org/10.31510/infa.v15i1.337.

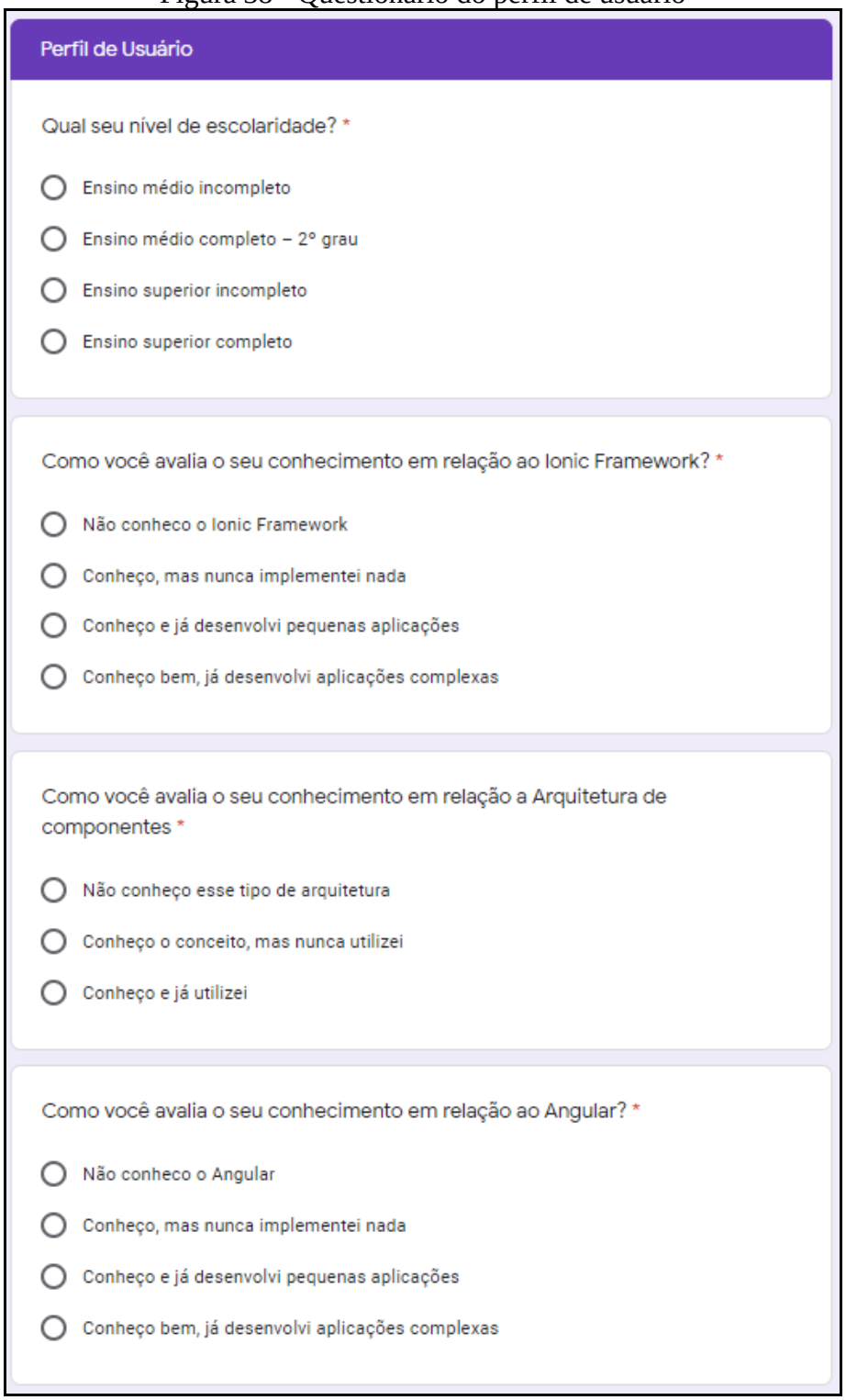
VILLELA. **IBGE**: 6,2% da população têm algum tipo de deficiência. Rio de Janeiro, 2015. Disponível em: http://agenciabrasil.ebc.com.br/geral/noticia/2015-08/ibge-62-dapopulacao-tem-algum-tipo-de-deficiencia. Acesso em 15 set. 2018.

WIPPEL, André F. **Tagarela**: Integração e melhorias no aplicativo de rede de comunicação alternativa. 2015. 65 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

APÊNDICE A – Questionário da arquitetura desenvolvida

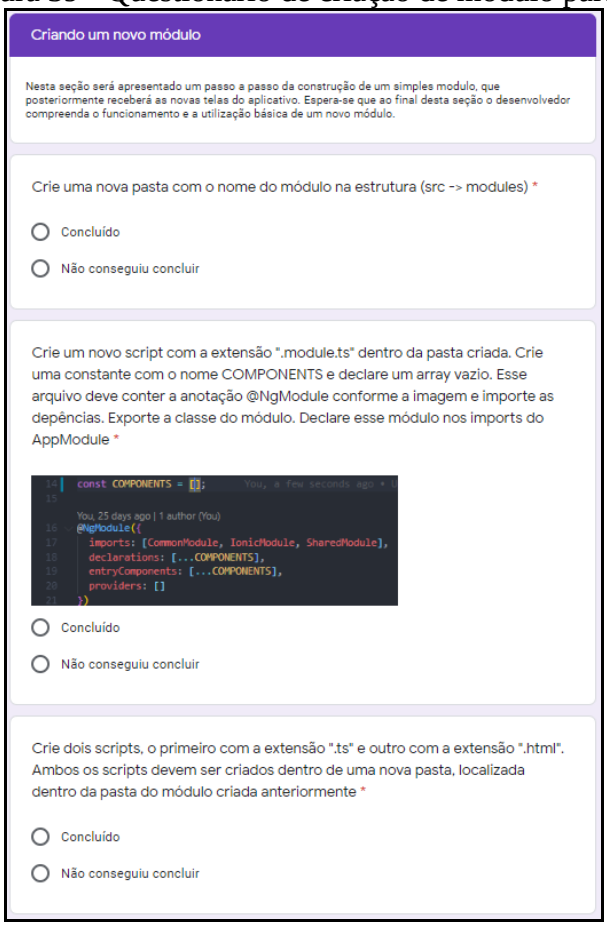
As Figuras 38 a 44 apresentam o questionário realizado com os programadores para avaliar a usabilidade da arquitetura de componentes para desenvolver uma nova funcionalidade no aplicativo.

Figura 38 - Questionário do perfil de usuário



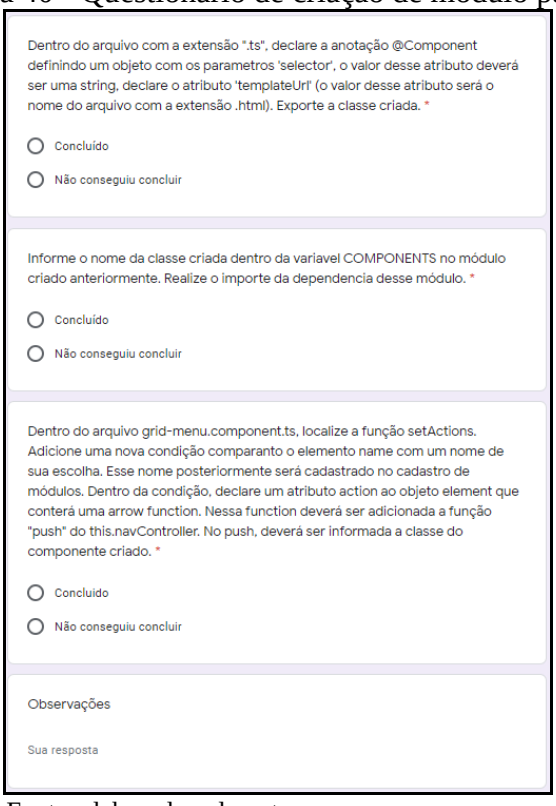
Fonte: elaborado pelo autor.

Figura 39 - Questionário de criação de módulo parte 1



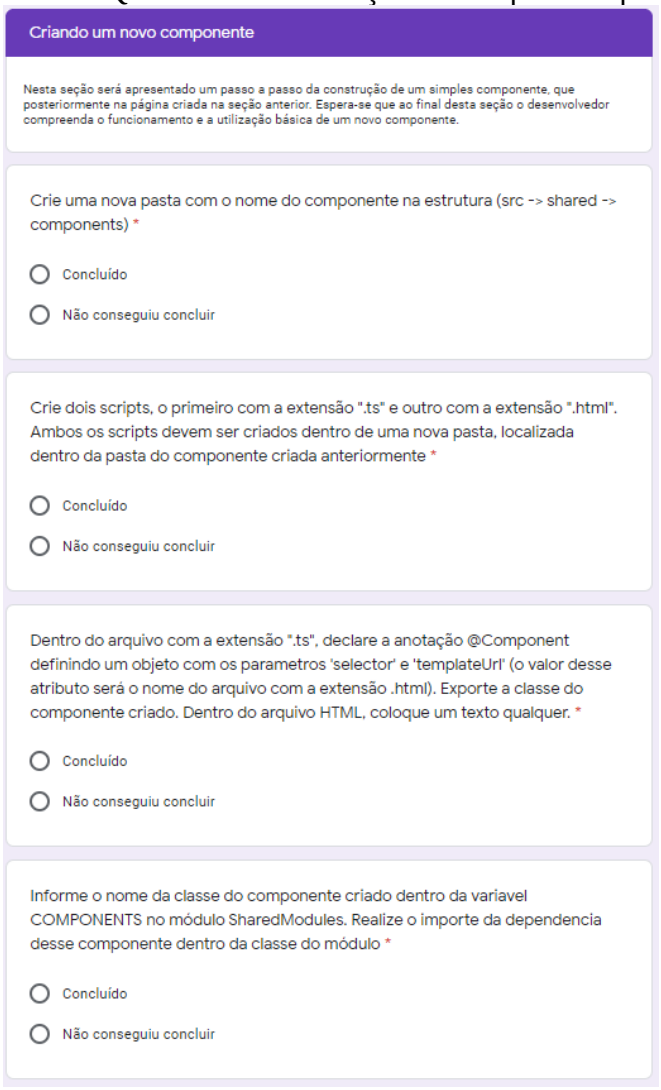
Fonte: elaborado pelo autor.

Figura 40 - Questionário de criação de módulo parte 2



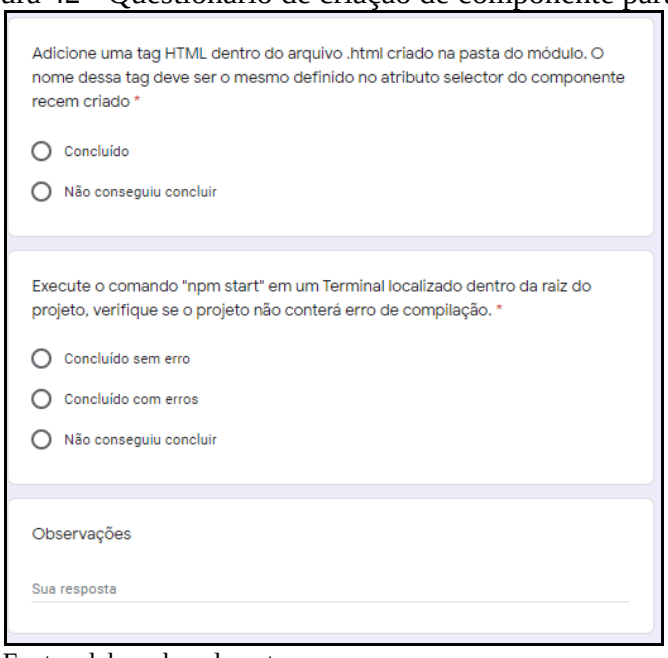
Fonte: elaborado pelo autor.

Figura 41 - Questionário de criação de componente parte 1



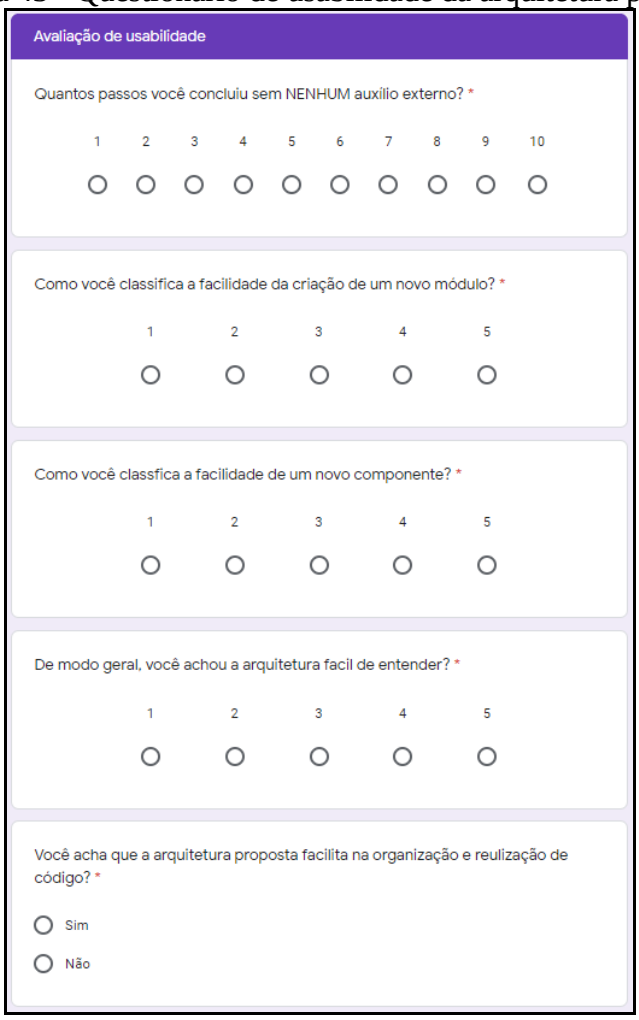
Fonte: elaborado pelo autor.

Figura 42 - Questionário de criação de componente parte 2



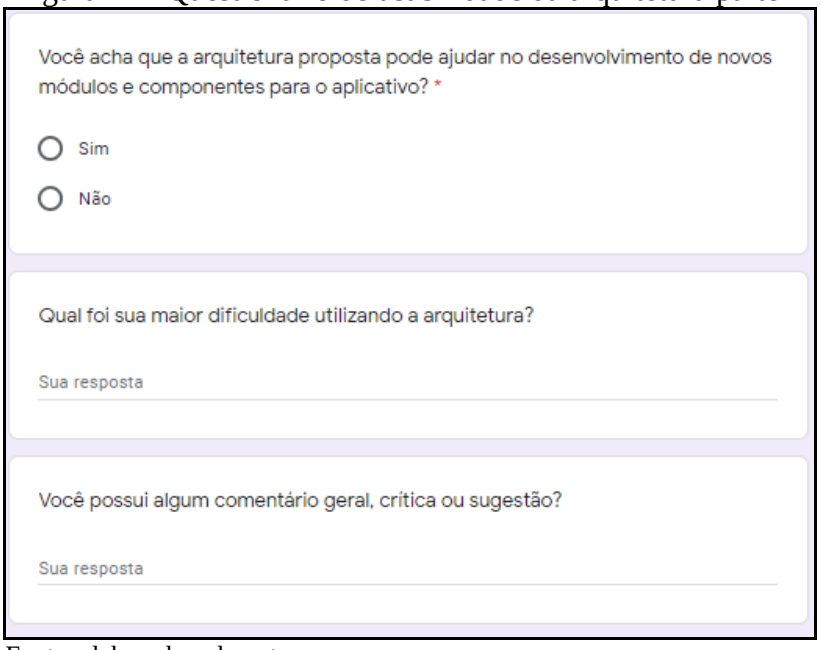
Fonte: elaborado pelo autor.

Figura 43 - Questionário de usabilidade da arquitetura parte 1



Fonte: elaborado pelo autor.

Figura 44 - Questionário de usabilidade da arquitetura parte 2



Fonte: elaborado pelo autor.