UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Thiago Henrique da Silva

TECNOLOGIA ASSISTIVA: USO DE INTERNET DAS COISAS PARA AUXÍLIO A DEFICIENTES VISUAIS

Uberlândia, Brasil

2021

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Thiago Henrique da Silva

TECNOLOGIA ASSISTIVA: USO DE INTERNET DAS COISAS PARA AUXÍLIO A DEFICIENTES VISUAIS

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Alexsandro Santos Soares

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2021

Thiago Henrique da Silva

TECNOLOGIA ASSISTIVA: USO DE INTERNET DAS COISAS PARA AUXÍLIO A DEFICIENTES VISUAIS

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Alexsandro Santos Soares
Universidade Federal de Uberlândia - UFU

Prof. Dr. Lásaro Jonas Camargos
Universidade Federal de Uberlândia - UFU

Prof. Dr. Pedro Frosi Rosa
Universidade Federal de Uberlândia - UFU

Uberlândia, Brasil
2021

Agradecimentos

Agradeço aos meus pais por me acompanharem me apoiando em toda jornada do curso; agradeço aos meus avós por me apoiarem e me impulsionarem a ir mais longe, aos meus tios por me darem suporte e inspiração durante minha trajetória.

Agradeço à minha companheira pelo apoio, compreensão e compartilhamento de ideias para a realização deste; ao meu orientador por conduzir todo seu conhecimento dentro e fora deste trabalho e pela sua resiliência durante o processo de construção dele. E agradeço também a todos que me ajudaram ou contribuíram de alguma forma durante o processo.

Resumo

Este trabalho apresenta uma arquitetura e implementação para aplicação de Tecnologia Assistiva com objetivo de auxiliar deficientes visuais em sua locomoção em ambientes externos. A solução foi implementada por meio de um aplicativo que utiliza informações posicionais e sobre seus arredores com uso de Internet das Coisas. Para exibição da arquitetura foi utilizada a linguagem de modelagem de sistemas (SysML). Pesquisas sobre as principais soluções de Tecnologia Assistiva e Internet das Coisas apontam uma constante evolução de ambas as áreas nos últimos anos. Portanto, parece lógico que a Internet das Coisas seja utilizada como ferramenta para implementações de Tecnologias Assistivas.

Palavras-chave: Tecnologia Assistiva; Internet das coisas; IoT; Deficientes Visuais; SysML; Aplicativo; Computação em Nuvem; API Web; MQTT; Raspberry Pi; ESP8266; React Native; Clojure; JavaScript.

Abstract

This work presents an architecture and implementation of an Assistive Technology application with the objective of helping the visually impaired walking outdoors. The solution was implemented through an application that uses positional information and information about its surroundings using the Internet of Things. The Systems Modeling Language (SysML) was used for modeling the architecture. Research on the main Assistive Technology and Internet of Things solutions point to a constant evolution of both areas in recent years. Therefore, it seems logical that Internet of Things can be used as a tool for Assistive Technology implementations.

Keywords: Assistive Technology; Internet of Things; IoT; Visually Impaired ; SysML; App; Cloud Computing; Web API; MQTT; Raspberry Pi; ESP8266; React Native; Clojure; JavaScript.

Lista de ilustrações

Figura 1 – Usuário com o *headset* desenvolvido pela Microsoft. Fonte: Microsoft (2015). 20
Figura 2 – Requisição HTTP. Fonte: Kurose e Ross (2017). 24
Figura 3 – Resposta HTTP. Fonte: Kurose e Ross (2017). 24
Figura 4 – Visão geral do protocolo MQTT com a comunicação do *broker* entre vários clientes. Fonte: adaptado de Micro Controllers Lab (2021). . . . 25
Figura 5 – Troca de pacotes MQTT entre dois clientes e um *broker*. Fonte: adaptado de Wikimedia (2021). 26
Figura 6 – *Handshake* de três partes do protocolo TCP. 27
Figura 7 – Troca de mensagens do protocolo TCP para uma aplicação Telnet. Fonte: adaptado de Kurose e Ross (2017). 28
Figura 8 – Funcionamento do HTTP e WebSockets. Fonte: adaptado de Scaleway (2021). 29
Figura 9 – Inicio do estabelecimento de uma conexão TLS entre cliente e servidor. 30
Figura 10 – Troca de mensagens do protocolo CoAP. Fonte: Adaptado de Shelby K. Hartke (2014). 31
Figura 11 – Fluxo de dados de uma consulta em uma rede local utilizando o protocolo mDNS. Fonte: Adaptado de Dentella (2021). 32
Figura 12 – Definição de pinos do NodeMcu. Fonte: NodeMcu (2019). 34
Figura 13 – Definição de pinos do Arduino Nano 33 IoT. Fonte: Arduino (2019). . 34
Figura 14 – Definição de pinos do Raspberry Pi. Fonte:Raspberry Pi (2021). 35
Figura 15 – Bloco SysML para a entidade assistente virtual. 36
Figura 16 – Relação um para um que indica que uma assistente virtual consulta um serviço de localização. 36
Figura 17 – Relação um para muitos, indicando que uma assistente virtual consulta um ou mais semáforos. 37
Figura 18 – Composição SysML indicando que a entidade aplicativo é composta por três elementos. 37
Figura 19 – Requerimento SysML para a entidade assistente virtual. 37
Figura 20 – Diagrama de requerimentos para a assistente virtual, indicando os sub requerimentos: navegação, salvamento de pontos de interesse e informações posicionais. 38
Figura 21 – Pacote que engloba um diagrama de blocos para o aplicativo 38
Figura 22 – Diagrama de atividades indicando a inicialização do serviço de rota. . . 39
Figura 23 – Nó de evento seguido por um nó de ação em um diagrama de atividades. 39

Figura 24 – Execução simplificada do inicio do serviço de rota, mostrando inicio e fim de um fluxo, direção do fluxo e divisão de fluxo em um diagrama de atividades. 40
Figura 25 – Diagrama de atividades com um nó de decisão para verificação de rota ativa e um nó de pausa. 40
Figura 26 – Arquitetura geral que mostra todos os pontos da arquitetura com o semáforo o aplicativo e a nuvem. 43
Figura 27 – Inicialização do semáforo com conexão *WiFi*. 45
Figura 28 – Conexão do semáforo com o *broker* MQTT e controle de estados para ativação de LEDs. 45
Figura 29 – Visão geral da aplicação local com a inicialização da mesma, onde é feita a conexão em dois *brokers*, publicação de estados e recebimento de sinais indicando que existe um pedestre no local. 46
Figura 30 – Diagrama de atividades com cálculo de tempo dos estados do semáforo na aplicação local. 46
Figura 31 – Modelo de entidade e relacionamento das tabelas do banco de dados . . 47
Figura 32 – Diagrama de atividades com a inicialização controlador geral, mostrando a conexão com o *broker* global e a inscrição no tópico de atualizações de estado. 48
Figura 33 – Diagrama de atividades com o recebimento do evento de estado do semáforo e atualização no banco de dados. 49
Figura 34 – Diagrama de atividades com os métodos de consulta ao semáforo na API Web. 51
Figura 35 – Diagrama de atividade com o método de consulta de localidade por identificador. 52
Figura 36 – Diagrama de atividade com o método de consulta de localidade por coordenadas. 52
Figura 37 – Diagrama de atividade com o método de consulta de localidade por coordenadas e tipo. 53
Figura 38 – Diagrama de atividade com o método de consulta de localidade por nome. 53
Figura 39 – Diagrama de requisitos geral para o aplicativo 54
Figura 40 – Diagrama de blocos com a arquitetura do aplicativo indicando todos os serviços e dependências. 54
Figura 41 – Diagrama de atividades com o registro de todos os eventos necessários para funcionamento da assistente virtual. 56
Figura 42 – Diagrama de atividades com o fluxo de controle e reconhecimento de voz da assistente virtual, mostrando o processamento de perguntas e reconhecimento de comandos. 57

Figura 43 – Diagrama de atividades com o processamento de comandos relacionados a consulta de localizações nos arredores. 59
Figura 44 – Diagrama de atividade com o reconhecimento dos comandos relacionados a rota, como inicialização, finalização, informações e configuração da rota. 59
Figura 45 – Diagrama de atividades com a inicialização do serviço de rota indicando o registro do evento de atualização do GPS e a inicialização do temporizador para atualização de direção da rota. 60
Figura 46 – Diagrama de atividades com o processo de consulta de localização para iniciar uma rota. 61
Figura 47 – Tradução do ângulo da direção em um texto que pode ser utilizado para direcionamento de um usuário. 62
Figura 48 – Diagrama de atividades com o fluxo de atualização da rota, mostrando o processo desde o evento de atualização de GPS até a tradução da direção a ser seguida. 64
Figura 49 – Diagrama de blocos com a representação da comunicação do aplicativo com o semáforo, indicando os dois caminhos possíveis, pelo broker MQTT e pela API Web. 65
Figura 50 – Diagrama de atividades com a busca de dispositivos MQTT em uma rede local por meio do mDNS. 65
Figura 51 – Diagrama de atividades com o fluxo de conexão pela API Web, quando não existe um broker MQTT na rede local. 66
Figura 52 – Diagrama de atividades com o fluxo do processo de cadastro de usuário. 66
Figura 53 – Diagrama de atividades com a inserção e remoção de locais na base de dados local do aplicativo. 67
Figura 54 – Circuito do semáforo projetado no Fritzing e circuito real implementado alimentado por uma porta USB. 68
Figura 55 – Esquema do circuito do semáforo com o NodeMcu projetado no Fritzing. 68

Lista de abreviaturas e siglas

API	Application Programming Interface
AWS	Amazon Web Services
CoAP	Constrained Application Protocol
DNS	Domain Name System
GPIO	General Purpose Input Output
GPS	Global Positioning System
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IBGE	Instituto Brasileiro de Geografia e Estatística
IoT	Internet of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
LED	Light Emitting Diode
MQTT	Message Queuing Telemetry Transport
OASIS	Organization for the Advancement of Structured Information Standards
OMG	Object Management Group
RAM	Random Access Memory
REST	Representational State Transfer
RFC	Request for Comments
SDK	Software Development Kit
SQL	Structured Query Language
SysML	Systems Modeling Language
TA	Tecnologia Assistiva

TCP	Transmission Control Protocol
TLS	Transport Layer Security
TTS	Text to Speech
UART	Universal Asynchronous Receiver Transmitter
UDP	User Datagram Protocol
UFU	Universidade Federal de Uberlândia
UML	Unified Modeling Language
URL	Uniform Resource Locator
USB	Universal Serial Bus
WPS	Wi-Fi Protected Setup

Sumário

1 INTRODUÇÃO . . . 14
1.1 Objetivo Geral . . . 14
1.2 Objetivo Específico . . . 14
1.3 Justificativa . . . 15
1.4 Organização do trabalho . . . 15

2 REVISÃO BIBLIOGRÁFICA . . . 16
2.1 Tecnologia assistiva . . . 16
2.1.1 Tipos de Tecnologia Assistiva . . . 16
2.1.2 Tecnologias Assistivas para deficientes visuais . . . 18
2.1.2.1 Sistemas desenvolvidos . . . 19
2.1.3 Requisitos de software para deficientes visuais . . . 21
2.2 Internet das Coisas . . . 22
2.2.1 IoT como suporte a recursos de Tecnologia Assistiva . . . 22
2.2.2 Protocolos para IoT . . . 23
2.2.2.1 HTTP . . . 23
2.2.2.2 MQTT . . . 25
2.2.2.2.1 Funcionamento MQTT . . . 25
2.2.2.3 TCP . . . 27
2.2.2.4 WebSockets . . . 28
2.2.2.5 TLS . . . 29
2.2.2.6 Conclusão protocolos de transporte para o MQTT . . . 30
2.2.2.7 CoAP . . . 30
2.2.2.8 mDNS . . . 31
2.2.2.9 Conclusão protocolos IoT . . . 32
2.2.3 Dispositivos IoT . . . 33
2.2.3.1 ESP8266 . . . 33
2.2.3.2 Arduino . . . 34
2.2.3.3 Raspberry Pi . . . 35
2.2.3.4 Conclusão Dispositivos IoT . . . 35
2.3 SysML . . . 35
2.3.1 Diagramas estruturais . . . 36
2.3.1.0.1 Diagrama de definição de blocos . . . 36
2.3.1.0.2 Diagrama de requerimentos . . . 37
2.3.1.0.3 Diagrama de pacotes . . . 38

2.3.2 Diagramas comportamentais . . . 38
2.3.2.0.1 Diagrama de atividades . . . 39
2.4 Definições Gerais . . . 40
2.4.1 Texto para fala . . . 40
2.4.2 JSON . . . 41
2.4.2.1 GeoJSON . . . 41
2.4.3 API Web . . . 42

3 ARQUITETURA . . . 43
3.1 Semáforo . . . 43
3.1.1 Dispositivo IoT . . . 44
3.1.1.1 Aplicação local . . . 44
3.1.1.2 Broker MQTT . . . 47
3.1.1.3 mDNS . . . 47
3.2 Nuvem . . . 47
3.2.1 Banco de Dados . . . 47
3.2.1.1 Tabelas Controlador Geral . . . 47
3.2.1.2 Tabelas API Web . . . 48
3.2.2 *Broker* global . . . 48
3.2.3 Controlador Geral . . . 48
3.2.4 API Web . . . 49
3.2.4.1 Métodos semáforo . . . 49
3.2.4.2 Métodos Localização . . . 50
3.3 Aplicativo . . . 53
3.3.1 Requisitos . . . 53
3.3.2 Arquitetura . . . 54
3.3.3 Assistente virtual . . . 55
3.3.4 Localização . . . 58
3.3.5 Rota . . . 59
3.3.6 Interação semáforo . . . 63
3.3.6.1 MQTT . . . 64
3.3.6.2 API . . . 64
3.3.7 Informações usuário . . . 66
3.4 Protótipo . . . 67
3.4.1 Semáforo . . . 67
3.4.2 Dispositivo IoT . . . 68
3.4.3 Broker Global . . . 69
3.4.4 Banco de Dados . . . 69
3.4.5 Controlador Geral . . . 70
3.4.6 API Web . . . 70

3.4.7 Aplicativo 70
3.4.8 Anexos Aplicativo 71
3.4.8.1 Google Maps API 71
3.4.8.1.1 Pontos de Interesse 71
3.4.8.1.2 Endereço 72
3.4.8.1.3 Busca localização 72
3.4.8.2 MapBox API 72
3.4.9 Demonstração Protótipo 73
3.4.10 Considerações finais 73

4 CONCLUSÃO 75
4.1 Contribuições 75
4.2 Trabalhos futuros 76

REFERÊNCIAS 77

1 Introdução

Pessoas com algum tipo de deficiência enfrentam dificuldades em suas vidas diariamente, sejam essas relacionadas à comunicação, locomoção, acesso à informação, acessibilidade, entre outros. Recursos de Tecnologia Assistiva surgem com o objetivo de ajudar pessoas com deficiência e consistem, por exemplo, em bengalas inteligentes, softwares de leitura, próteses, cadeiras de roda, etc. Com o avanço tecnológico nota-se também um avanço na área de Tecnologias Assistivas.

Este trabalho foi desenvolvido com foco na área de aplicativos para deficientes visuais como recurso de Tecnologia Assistiva. Serão abordadas dificuldades que deficientes visuais encontram ao se locomoverem em um ambiente externo. Em específico, será apresentada uma arquitetura de um aplicativo que contém uma assistente virtual para auxiliar o usuário em sua locomoção de um ponto ao outro. Durante a locomoção será feito o uso de Internet das Coisas para conectar semáforos à internet com o objetivo de informar ao usuário se é seguro ou não atravessar uma via em que ele se encontra.

Além disso, será mostrada uma pesquisa sobre os principais trabalhos relacionados à Tecnologia Assistiva para deficientes visuais e também um apanhado geral sobre o mundo da Internet das Coisas e seus principais protocolos.

1.1 Objetivo Geral

O objetivo da presente pesquisa é apresentar uma solução de Tecnologia Assistiva com o uso de Internet das Coisas para auxiliar deficientes visuais em sua locomoção em ambientes externos.

1.2 Objetivo Específico

1. Desenvolver uma pesquisa sobre as principais soluções de Tecnologia Assistiva e Internet das Coisas no cenário tecnológico atual.

2. Modelar uma arquitetura para um sistema de Tecnologia Assistiva na forma de um aplicativo para *smartphones*, que seja capaz de receber informações do estado de um semáforo por meio da internet utilizando uma técnica formal.

3. Mostrar a viabilidade da arquitetura por meio do desenvolvimento de um protótipo.

1.3 Justificativa

A partir dessa pesquisa notou-se a ausência de uma solução que, além de auxiliar deficientes visuais a se locomoverem em um ambiente externo, também englobe todas as funcionalidades a seguir: informações sobre seus arredores enquanto está se movendo entre dois pontos, instruções por meio de texto falado e reconhecimento de comandos por meio de uma assistente virtual.

Os trabalhos relacionados com a pesquisa abordam temas que visam solucionar, em partes, os problemas apontados anteriormente. Contamos com o exemplo do trabalho de Ihejimba e Wenkstern (2020), no qual foi apresentada uma solução para notificação de semáforos à deficientes visuais. No entanto, essa solução não engloba nenhum mecanismo para navegação entre dois pontos. Outros exemplos, como Siddesh et al. (2016) e Carrasco, Salazar e Ramirez (2019), se baseiam no desenvolvimento de uma bengala inteligente, que tem como objetivo auxiliar na navegação do usuário. Porém, tal solução pode contar com custos elevados para aquisição. Autores como Almeida (2015) e Coughlan e Shen (2013) apontam os métodos de computação visual, que dependem do uso da câmera do celular, para detecção de obstáculos e semáforos em um ambiente externo. Este tipo de solução conta com dificuldades na sua usabilidade para com um deficiente visual, pois depende que a câmera seja apontada aos lugares corretos para detecção. Assim, observa-se uma necessidade de unificar diferentes soluções, mas com objetivos semelhantes, a fim de simplificar o uso das mesmas para com seus usuários.

Além disso, este trabalho tem como objetivo contribuir com a melhora da qualidade de vida de deficientes visuais de maneira que este trabalho possa contribuir com futuros projetos e soluções da área.

1.4 Organização do trabalho

No capítulo 2 são apresentados os conceitos principais no âmbito de Tecnologia Assistiva e Internet das Coisas. Neste capítulo, temos também uma pesquisa com as principais soluções na área, além de explicações sobre os principais protocolos utilizados em Internet das Coisas. Contamos também com uma breve introdução à linguagem de modelagem de sistemas (SysML).

No capítulo 3 temos a arquitetura proposta para o sistema juntamente com a sua implementação por meio de um protótipo.

Por fim, o capítulo 4 é composto pelas conclusões do trabalho além de indicar possíveis melhorias e pontos de atenção para trabalhos futuros.

2 Revisão bibliográfica

Neste capítulo será mostrada uma pesquisa acerca dos principais termos e trabalhos no âmbito de Tecnologia Assistiva e Internet das Coisas. Será apresentado também uma introdução a linguagem de modelagem de sistemas (SysML).

2.1 Tecnologia assistiva

A Tecnologia Assistiva ou TA é um termo que abrange vários aspectos. Para melhor entendê-lo a seguir serão apresentadas algumas definições de diferentes autores.

Segundo Bersch (2017) podemos definir a Tecnologia Assistiva como um conjunto de recursos e serviços destinados a proporcionar ou ampliar habilidades de pessoas com algum tipo de deficiência. Seu principal objetivo é possibilitar que uma pessoa com deficiência tenha maior independência, mais qualidade de vida e melhor inclusão social mediante ao aumento de sua comunicação, mobilidade e habilidades.

Ainda segundo Bersch (2017) em 2007 o Comitê de Ajudas Técnicas (2007) aprovou o conceito brasileiro para TA como sendo:

> "Tecnologia Assistiva é uma área do conhecimento, de característica interdisciplinar, que engloba produtos, recursos, metodologias, estratégias, práticas e serviços que objetivam promover a funcionalidade, relacionada à atividade e participação, de pessoas com deficiência, incapacidades ou mobilidade reduzida, visando sua autonomia, independência, qualidade de vida e inclusão social."

Já para Bouck (2017) a Tecnologia Assistiva pode se referir tanto a dispositivos e ferramentas quanto a serviços, que podem ser utilizados na seleção, aquisição e manutenção de dispositivos de TA. Um exemplo para um serviço seria efetuar uma personalização ou um reparo em um dispositivo de TA.

2.1.1 Tipos de Tecnologia Assistiva

Por ser um tema amplo, os recursos de TA podem ser divididos em grupos de acordo com tipos de deficiências e de tecnologias. Bersch (2017) define 12 divisões para os recursos, que são:

1. Auxílios para o dia a dia:

Neste grupo se encontram os recursos que se destinam a ajudar as pessoas em sua rotina no dia a dia, como barras de apoio, talheres adaptados, dentre outros.

2. Comunicação aumentativa e alternativa:

   Engloba recursos destinados às pessoas que tem dificuldade com a fala, onde pode ser utilizado para ampliar sua comunicação, seja por meio de textos, pranchas de comunicação ou softwares que simulam uma prancha de comunicação.

3. Recursos de acessibilidade ao computador:

   Inclui meios de acessibilidade à computadores por meio de hardwares ou softwares. Como exemplo podemos citar teclados adaptados para facilitar a digitação, ou ainda softwares de lupa, que permitem melhor visibilidade dos dados na tela.

4. Sistemas de controle de ambiente:

   Constitui efetuar o controle do ambiente por meio de um controle remoto ou outro dispositivo, de forma que possa ser acessível a uma pessoa com deficiência. Como exemplo, temos o controle de luzes, ar-condicionado, portas, sistemas de segurança, entre outros.

5. Projetos arquitetônicos para acessibilidade:

   Se trata de alterações em locais ou projetos arquitetônicos destinados a funcionalidade e mobilidade de pessoas com deficiência, como por exemplo a construção de rampas, corrimões, elevadores, entre outros.

6. Órteses e próteses:

   Equipamentos que são colocados no corpo como próteses para substituir uma parte do corpo, ou órteses para ampliar a coordenação ou mobilidade de um membro ou parte do corpo.

7. Adequação postural:

   Grupo que engloba recursos que auxiliam na correção e conservação de uma postura, que deve ser alinhada, confortável e estável. Um exemplo seria um encosto de cadeira de rodas feito sob medida para uma pessoa.

8. Auxílios de mobilidade:

   Qualquer tipo de dispositivo que assista à uma pessoa com deficiência em sua locomoção, como por exemplo uma cadeira de rodas, uma bengala, dentre outros.

9. Auxílios para ampliação da função visual e recursos que traduzem conteúdos visuais em áudio ou informação tátil:

   Aqui temos recursos para auxílio na leitura e observação de informações, como por exemplo: lupas, mapas táteis e softwares de leitura.

10. Auxílios para melhorar a função auditiva e recursos utilizados para traduzir os conteúdos de áudio em imagens, texto e língua de sinais.

    Engloba serviços destinados à tradução de recursos de áudio e transcrição de textos. Como exemplo temos softwares que transformam texto em voz e voz em texto, aparelhos auditivos e legendas.

11. Mobilidade em veículos

    Se trata das alterações feitas em veículos para melhorar a acessibilidade de pessoas com deficiência, como rampas e elevadores para adentrar ao veículo, além de englobar também as adaptações em veículos destinadas a deficientes, possibilitando que possam dirigi-los.

12. Esporte e Lazer

    Grupo que contém os recursos que facilitam exclusivamente a prática de esportes, como próteses e cadeiras de rodas adaptadas.

2.1.2 Tecnologias Assistivas para deficientes visuais

Segundo um estudo feito por Bourne et al. (2021) é estimado que cerca de 43,3 milhões de pessoas em todo o mundo possuem deficiência visual. No Brasil o último estudo feito pelo IBGE (2010)[1] apontou que mais de 6 milhões de brasileiros apresentam grande dificuldade em enxergar. Além disso, mais de 500 mil brasileiros não conseguem enxergar de forma alguma. Com isso a TA tem uma grande demanda para melhorar a qualidade de vida das pessoas com deficiência visual.

Deficientes visuais contam com dificuldades diárias em suas vidas. Tarefas simples como ir ao trabalho, pegar ônibus, atravessar um cruzamento movimentado, usar o celular dentre outras. Vários recursos de Tecnologia Assistiva já existem na tentativa de remediar tais dificuldades, como por exemplo bengalas inteligentes como o UltraCane (2021), ou ainda auxiliadores de leitura como por exemplo o JAWS (2021).

Bhowmick e Hazarika (2016) chegaram à conclusão que apesar de estarmos em constante avanço tecnológico apenas uma parcela de recursos chegaram para os deficientes visuais. Problemas como localização em um ambiente interno, encontrar objetos perdidos, dentre outros, ainda não tem uma solução viável em TA.

O *smartphone* é um recurso interessante que pode ser usado de várias maneiras para substituir interações visuais. Atualmente existem vários aplicativos nativos e terceiros que são utilizados como TA para deficientes visuais e que permitem uma vida mais independente e com mais qualidade para seus usuários. Além disso, o uso do *smartphone*

[1] O estudo do IBGE seria atualizado em 2020, porém devido a pandemia do COVID-19 não foi possível realizá-lo.

como ferramenta de TA permite que o deficiente visual fique à vontade ao utilizá-lo, por ser um dispositivo mais comum e discreto quando comparado a outros recursos como lupas por exemplo. Como exemplos de aplicações nativas podemos citar o recurso de *TalkBack* no Android e o *Voice Over* para IOS. Para aplicativos de terceiros, existem inúmeras opções com funções de lupa, verificação de notas e moedas, navegação por GPS, leitura dentre muitos outros. (SENJAM, 2021)

De acordo com Senjam (2021), o número de aplicativos com acessibilidade a deficientes visuais tem crescido com o tempo, e trazem aplicações mais acessíveis e inovadoras. Porém, esse crescimento é acompanhado de um aumento na complexidade de uso do *smartphone* visto que cada aplicação tem sua função específica. Portanto, é importante definir regras e diretrizes para atender de forma mais simples as necessidades de pessoas com baixa visão.

2.1.2.1 Sistemas desenvolvidos

A seguir serão apresentados alguns trabalhos que buscam por meio da TA melhorar a qualidade de vida de deficientes visuais.

Coughlan e Shen (2013) desenvolveram o Crosswatch, que é uma ferramenta focada no auxílio da travessia de cruzamentos para deficientes visuais. A ferramenta utiliza a câmera de um *smartphone* para conseguir informações sobre os arredores do pedestre e o auxiliar na travessia. Foram utilizados métodos de computação visual para alinhar o pedestre com a faixa, e além disso, a ferramenta é capaz de informar se é seguro ou não efetuar a travessia.

Almeida (2015) também propõe um modelo para reconhecimento semáforos. O modelo faz o uso de computação visual através do processamento de imagem para conseguir o estado atual do semáforo. Com a ajuda de uma foto da câmera do *smartphone*, apontada para um semáforo, o método utiliza um algoritmo de mapa de saliência para identificar onde está o semáforo e verificar seu estado atual.

Tapu, Mocanu e Zaharia (2013) propuseram um aplicativo para *smartphone* que utiliza computação visual classificar para detectar obstáculos dinâmicos e estáticos. O principal diferencial da ferramenta é que a análise do ambiente é feita em tempo real pela câmera do *smartphone*. Foram utilizados algoritmos para identificação de pontos de interesse (Lucas-Kanade) e um algoritmo de detecção de movimentos (RANSAC).

Soluções que utilizam modelos e métodos de processamento de imagens para detectar informações, como as apontadas anteriormente tem a desvantagem de sempre necessitarem que a câmera esteja apontada na direção correta. Esta tarefa pode ser complicada para pessoas com deficiência visual, além disso existe uma dependência em condições de visibilidade e luminosidade para que as soluções funcionem corretamente.

A tecnologia 3D-Soundscape, desenvolvida pela Microsoft (2015) funciona com a ajuda de um *headset* especial (Figura 1), que é pareado com o *smartphone*. O *headset* se comunica com diversos dispositivos no ambiente com o objetivo de guiar o usuário e transmite o som diretamente por meio do osso do maxilar, evitando assim a obstrução da audição, que nos casos de pessoas com deficiência visual é de extrema importância. Assim ao utilizar o dispositivo ainda é possível ouvir o ambiente ao seu redor e se comunicar com as pessoas normalmente.

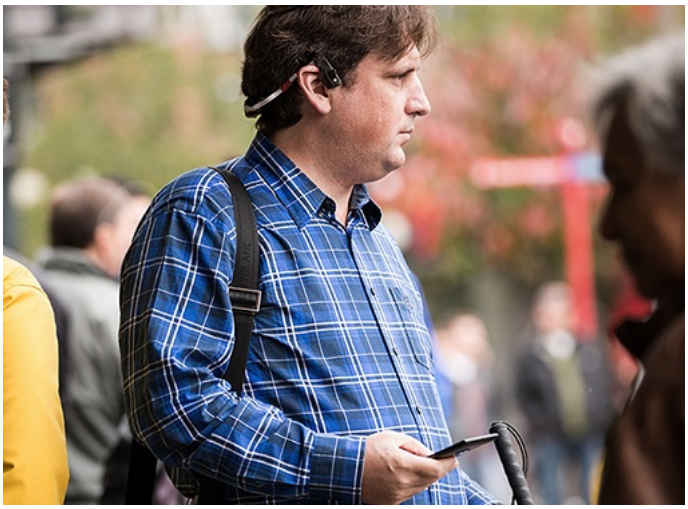

Figura 1 – Usuário com o *headset* desenvolvido pela Microsoft. Fonte: Microsoft (2015).

Siddesh et al. (2016) propõe um modelo para criação de um sistema para auxiliar deficientes visuais a se locomoverem. O modelo é constituído de 3 partes, um servidor central que armazena as informações de rotas, um aplicativo para celulares e uma bengala inteligente. Um usuário do tipo administrador poderá cadastrar rotas pelo aplicativo e programar mensagens com informações úteis sobre a rota para serem tocadas pelo celular em momentos pré-definidos. Um usuário que possui deficiência poderia então utilizar essa rota para se locomover, e com a ajuda da bengala verificar possíveis obstáculos no caminho.

O 3D-Soundscape da Microsoft (2015) e o modelo proposto por Siddesh et al. (2016) são importantes pois podem dar informações de arredores a pessoas com deficiência visual, porém estes armazenam informações estáticas, onde não é possível por exemplo capturar uma ação de um objeto dinamicamente no ambiente.

Carrasco, Salazar e Ramirez (2019) sugerem um sistema que aproveita do uso de dispositivos de Internet das Coisas em um âmbito de cidades inteligentes, com objetivo de possibilitar ao usuário uma melhor percepção de seus arredores ao se locomover pelas ruas. O sistema é composto por uma bengala inteligente que se comunica direto com o celular do usuário e pode reconhecer dispositivos IoT nos arredores e apresentar sinais quando próximo a um dispositivo.

Ihejimba e Wenkstern (2020) apresentam um modelo interessante para notificar deficientes visuais sobre semáforos que possam encontrar ao se movimentarem de um ponto a outro. O modelo utiliza da IoT por meio de um Raspberry Pi para conectar o semáforo à internet e assim poder avisar ao usuário em tempo real o estado atual diretamente em um aplicativo em seu celular. O Raspberry envia o estado do semáforo por meio do protocolo MQTT para um serviço de nuvem da AWS (Amazon Web Services). Com o estado processado pela AWS, ele é enviado diretamente ao aplicativo que possui funções de acessibilidade como leitura de texto e alertas vibratórios para notificar o usuário.

As soluções apresentadas por Carrasco, Salazar e Ramirez (2019) e Ihejimba e Wenkstern (2020) utilizam de Internet das Coisas para capturar informações dos arredores e apresentar essas informações de forma dinâmica ao usuário.

Assistentes virtuais como a Alexa vem sendo cada vez mais utilizadas por deficientes visuais. A assistente produzida pela Amazon (2021) conta com vários modelos e utiliza reconhecimento de comandos por voz para auxiliar em várias tarefas do dia a dia. Dentre as funcionalidades da assistente, ressaltamos algumas de importância para deficientes visuais, como fazer compras, efetuar ligações, criação de rotinas para automação residencial, controle do ambiente doméstico, realizar leitura e prover entretenimento por meio de jogos, filmes e músicas. Uma função também muito importante é a chamada "*show and tell*", disponibilizada para dispositivos que possuem uma câmera e permite a identificação de objetos segurados pelo usuário.

Segundo Ramadan, Farah e Essrawi (2020), a Alexa permite que pessoas com deficiência possuam uma maior independência e as permite executar tarefas que antes não poderiam ou que executavam com certa dificuldade. Além disso, a assistente passa a fazer parte do ambiente familiar, provendo um senso de segurança pois a assistente está disponível a qualquer momento ao usuário.

Com a análise dos sistemas e soluções apontados anteriormente, foi notada a falta de uma solução que dê informações sobre arredores de forma dinâmica e ainda que tenha uma assistente virtual para facilitar o uso para deficientes visuais.

2.1.3 Requisitos de software para deficientes visuais

Segundo Campana (2017) a competitividade no mercado de software para usuários com deficiência visual tem crescido, e com isso alguns aspectos devem ser considerados a fim de qualificação de qualidade de software, sendo esses: acessibilidade, usabilidade e experiência do usuário

Segundo Almeida e Araujo (2013 apud SANTOS, 2019), usuários que não possuem visão não conseguem se aproveitar de uma tela com informações visuais, e com isso, ao desenvolver aplicações de *software* para deficientes visuais é necessário ter cuidados

especiais. Como mostra Gomes (2007), a fala é um dos meios mais significativos quando se trata de obtenção e propagação de conhecimento, e isso é importante pois ao interagir com um software nos parece natural utilizar a fala.

Uma forma de garantir a acessibilidade e usabilidade de um *software* a um deficiente visual é por meio de uma assistente virtual. Desta forma, qualquer ação no sistema pode ser feita por meio da voz, permitindo um controle total sem a necessidade de interação com uma interface gráfica.

2.2 Internet das Coisas

Não há uma definição geral para o termo Internet das Coisas (*Internet Of Things* - IoT).

Segundo Mulani e Pingle (2016), IoT é o termo utilizado para descrever cenários em que se aplica capacidade computacional (sem ou com o uso de conectividade à internet) à diversos tipos de objetos, sensores e itens utilizados no dia-a-dia.

Para Serpanos e Wolf (2018), a IoT é formada por dispositivos físicos ativados pela internet, redes de sensores em tempo real e redes dinâmicas de dispositivos embarcados.

A Internet das Coisas pode ser definida também como uma rede de elementos físicos que podem ser compostos por: sensores, identificadores, software e conectividade com a internet. Segundo Rayes e Salam (2019),"IoT é uma rede de coisas, com identificação de dispositivos, inteligência embarcada e capacidade sensorial e de ação, conectando coisas e pessoas na internet"[2].

Contamos também com o estudo de Mohamed (2019), onde o mesmo informa que a IoT se trata de qualquer coisa que tenha capacidade de se comunicar com qualquer coisa em diversos lugares através de protocolos. O autor afirma que a IoT é uma espécie de "sociedade humana" porém com o mínimo de intervenção humana.

A seguir vamos discutir sobre como internet das coisas pode ser utilizada como ferramenta para auxílio em recursos de tecnologia assistiva.

2.2.1 IoT como suporte a recursos de Tecnologia Assistiva

O número de dispositivos IoT vem crescendo nos últimos anos. De acordo com um estudo feito pelo IoT Analytics (2020) no ano de 2020, mesmo com a pandemia de COVID-19, o número de conexões de dispositivos IoT ultrapassou pela primeira vez o número de conexões não IoT. A estimativa para o fim de 2020 foi um total de 21.7 bilhões

[2] *IoT is the network of things, with clear element identification, embedded with software intelligence, sensors, and ubiquitous connectivity to the Internet.*

de dispositivos conectados, dos quais 11.7 bilhões seriam dispositivos IoT. A estimativa para 2025 é que a marca de 30 milhões de conexões seja ultrapassada.

Com isso, temos uma oportunidade de utilizar a IoT como ferramenta de tecnologias assistivas para vários segmentos. Bansal e Garg (2021) citam como exemplo um sistema de navegação que pode aproveitar do grande número de dispositivos para encontrar objetos e pessoas em um espaço. A ideia é que os dispositivos possam se comunicar com sistemas de TA e com base nas informações coletadas possa determinar a localização exata de objetos e pessoas.

Domingo (2011) apresenta uma arquitetura, mostrando a composição de um sistema baseado em tecnologias assistivas IoT. A arquitetura é composta das três seguintes camadas:

1. Camada de percepção:

   Tem como função identificar objetos e coletar informações sobre um ambiente.

2. Camada de rede:

   Responsável por transmitir os dados coletados na camada de percepção para a internet.

3. Camada de aplicação:

   Se trata do conjunto de soluções que aplicam IoT para atender seus usuários.

2.2.2 Protocolos para IoT

Como realçado por Rayes e Salam (2019), as tecnologias baseadas em IoT possuem necessidades e características diferentes. Devem ser levados em consideração a capacidade computacional, mobilidade, tamanho, complexidade, dispersão de calor, consumo de energia, conectividade, entre outros. Assim, ao trabalhar com protocolos de rede, estes devem ser simples e que demandem pouca capacidade computacional e pouco consumo de energia.

Levando em consideração os protocolos usados em IoT, são apresentados os protocolos: HTTP, MQTT, TCP, WebSockets, mDNS e CoAP. O protocolo HTTP, embora não tenha amplo uso em IoT, é fundamental para entendimento e comparação com os demais protocolos apresentados.

2.2.2.1 HTTP

O *HyperText Transfer Protocol* ou simplesmente HTTP, como explicado por Kurose e Ross (2017) é um protocolo da camada de aplicação que define como a troca de mensagens HTTP deve ser feita entre um servidor e vários clientes. A qualquer momento

o cliente pode solicitar objetos do servidor por meio de uma requisição HTTP. Estes objetos podem ser páginas da web, imagens, arquivos HTML dentre outros. Ao receber a requisição, o servidor envia uma resposta HTTP contendo os objetos requisitados.

O HTTP utiliza como protocolo de transporte o TCP e pode suportar conexões entre cliente e servidor do tipo persistente ou não persistente. Dizemos que a conexão é persistente quando dentro de uma única conexão TCP, são feitas várias requisições HTTP. Uma conexão não persistente se dá quando para cada requisição HTTP temos uma nova conexão TCP sendo iniciada.

Como mostrado por Kurose e Ross (2017), abaixo temos o formato de uma mensagem de requisição do HTTP que solicita uma página HTML, seguida de sua respectiva resposta.

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/5.0
Accept-language: fr
```

Figura 2 – Requisição HTTP. Fonte: Kurose e Ross (2017).

Na primeira linha temos a requisição em si, que é composta por três campos: o tipo da requisição, a URL com o objeto requisitado e a versão do HTTP utilizada. As demais linhas da requisição são chamadas de cabeçalhos, e estes podem ser utilizados para indicar a preferência ou não por uma conexão persistente dentre outras informações adicionais.

```
HTTP/1.1 200 OK
Connection: close
Date: Tue, 18 Aug 2015 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html
(data data data data data ...)
```

Figura 3 – Resposta HTTP. Fonte: Kurose e Ross (2017).

A mensagem de resposta do HTTP é composta por três partes: uma linha para informar o status da resposta, seguida de linhas de cabeçalho, e por fim o corpo da resposta. No exemplo temos o status 200 na primeira linha indicando que houve sucesso na requisição, seguida de cabeçalhos com informações adicionais, e por fim o corpo da resposta que contém o objeto solicitado pelo cliente.

2.2.2.2 MQTT

MQTT (*Message Queue Telemetry Transport*) é um protocolo ideal para ambientes limitados de processamento e de comunicação com a rede. O protocolo foi definido pela OASIS (2021)(*Organization for the Advancement of Structured Information Standards*).

De acordo com Hillar (2017) o protocolo é baseado em um modelo de publicação e assinatura, que tem como intuito ser leve para possibilitar a transmissão de um volume grande de dados de maneira otimizada, ter baixo consumo de energia e permitir uma maior escalabilidade na situação em que for utilizado.

Ainda segundo Hillar (2017), no modelo de publicação e assinatura, um cliente publica uma mensagem e ela é enviada apenas à clientes destinados aquele tipo de mensagem. Os clientes taxados como destinatários assinam o tipo de mensagem que vão receber. Esse padrão requer um *broker*, ou servidor. Todos os clientes estabelecem uma conexão com o *broker* e o cliente que envia as informações ao *broker* é chamado de *publisher*. O *broker* então distribui a mensagem dentre todos os clientes, chamados de *subscribers*. Um cliente do protocolo MQTT pode ser ao mesmo tempo um *publisher* e um *subscriber*.

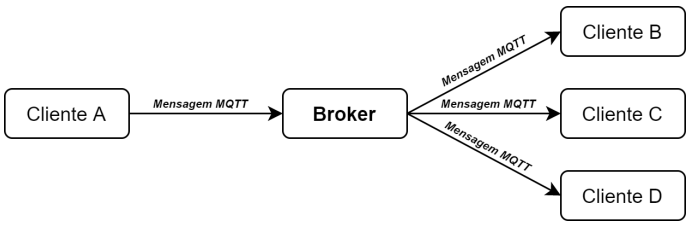

Figura 4 – Visão geral do protocolo MQTT com a comunicação do *broker* entre vários clientes. Fonte: adaptado de Micro Controllers Lab (2021).

2.2.2.2.1 Funcionamento MQTT

A seguir serão apresentados alguns conceitos que foram definidos pela OASIS (2021) sobre o funcionamento do MQTT.

O MQTT funciona com a troca de pacotes de controle em diversas formas. O formato geral desses pacotes tem até 3 partes, que são: um cabeçalho fixo, presente em todos os tipos de pacotes, um cabeçalho variável e um corpo, onde serão colocados os dados.

A primeira troca de pacotes entre cliente e servidor, se dá através de um pacote do tipo “CONNECT”, enviado do cliente para o servidor após uma conexão de rede ser estabelecida. No pacote são informadas as propriedades para a conexão MQTT, como por exemplo: identificador para o cliente, informações de qualidade do serviço, usuário e senha caso sejam necessários, dentre outros.

Após a requisição de conexão ser recebida pelo servidor, ele irá iniciar uma conexão MQTT com o cliente e em seguida enviar um pacote de reconhecimento da conexão, com o tipo "CONNACK". Caso o cliente não receba esse pacote em um tempo hábil, a conexão deverá ser descartada pelo cliente.

Com a conexão MQTT estabelecida, é possível, pelo cliente, enviar mensagens pelo servidor através de um pacote do tipo "PUBLISH". Este pacote é enviado tanto do cliente para o servidor e do servidor para o cliente com intuito de carregar as mensagens. Esse tipo de pacote, contém um cabeçalho de tamanho fixo onde são armazenadas informações importantes para o controle de mensagens e um cabeçalho variável que irá carregar o tópico em que a mensagem deve ser publicada. O conteúdo desse pacote contém a mensagem em si que é transportada pela rede. Uma vez recebida uma mensagem pelo cliente ou servidor, deve ser enviado um pacote de reconhecimento do recebimento, nesse caso do tipo "PUBACK".

Para receber as mensagens dos tópicos, o cliente precisa se inscrever em um ou mais tópicos de interesse. Para isso é necessário enviar ao servidor um pacote do tipo "SUBSCRIBE" e no conteúdo do pacote informar os tópicos em que deseja receber as mensagens enviadas. A partir do recebimento deste pacote, o servidor deverá confirmar a subscrição ao cliente através de outro pacote, do tipo "SUBACK". De maneira similar, o cliente pode querer não receber mais as mensagens de um tópico, questão que pode ser resolvida por meio de um envio ao servidor do pacote "UNSUBSCRIBE", que pode ser confirmado com o pacote do tipo "UNSUBACK" pelo servidor.

A qualquer momento a conexão pode ser encerrada por parte do cliente ou do servidor, que é feita com o pacote do tipo "DISCONNECT", representando o fim da troca dos pacotes de controle do MQTT.

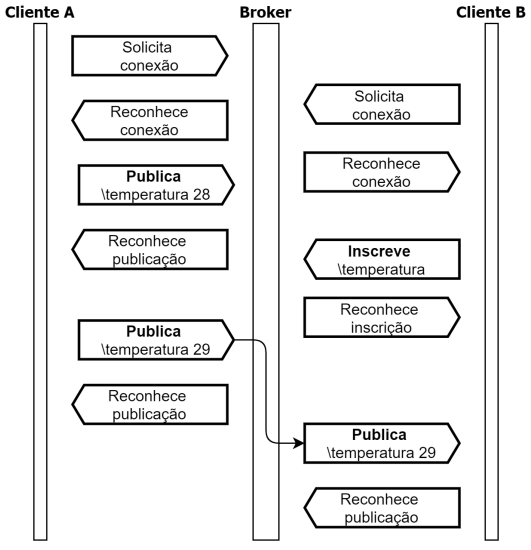

Figura 5 – Troca de pacotes MQTT entre dois clientes e um *broker*. Fonte: adaptado de Wikimedia (2021).

Como condição mínima para funcionamento, o protocolo de rede que funciona logo abaixo do MQTT precisa proporcionar a entrega confiável e ordenada dos pacotes de controle. É preciso que o protocolo também ofereça recursos para controle da qualidade de serviço. Na especificação da OASIS (2021), são citados os seguintes protocolos de transporte: TCP, WebSockets e TLS. A seguir será apresentada uma breve descrição destes.

2.2.2.3 TCP

O TCP (*Transmission Control Protocol*), definido pela RFC793 (1981) é um protocolo que atua na camada de transporte. Seu principal objetivo é estabelecer uma conexão entre dois pontos de forma a trocar dados de forma confiável e assíncrona.

Para conexão inicial, o protocolo usa um *handshake* de três partes, com intuito de evitar que conexões sejam duplicadas. Com a conexão estabelecida os dados são enviados em quadros e esses quadros são enviados com números de sequência que são utilizados para encontrar eventuais falhas na rede, caso seja detectada alguma falha os quadros são retransmitidos.

Na Figura 6 temos o *handshake* de três partes do TCP e na Figura 7 temos a troca de mensagens para uma aplicação Telnet com o uso do TCP.

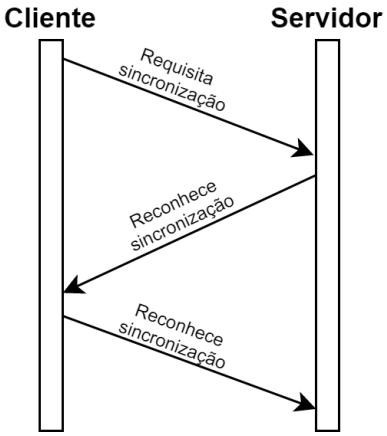

Figura 6 – *Handshake* de três partes do protocolo TCP.

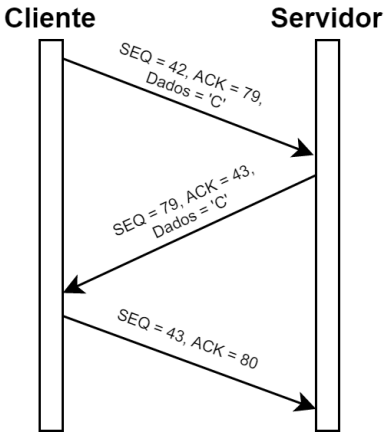

Figura 7 – Troca de mensagens do protocolo TCP para uma aplicação Telnet. Fonte: adaptado de Kurose e Ross (2017).

2.2.2.4 WebSockets

O protocolo WebSocket foi definido pela RFC 6455 (2011) e permite que seja aberta uma conexão entre cliente e servidor. Nessa conexão podem ser enviados dados de ambas as partes. O protocolo foi construído com intuito de oferecer um meio para comunicação de sistemas baseados em web.

Como definido na RFC 6455 (2011), seu funcionamento consiste inicialmente de um *handshake* entre cliente e servidor por meio de uma requisição HTTP. Essa requisição gera uma conexão TCP e o WebSocket aproveita dessa conexão para funcionar em cima do TCP. A conexão HTTP é inicialmente usada com intuito de utilizar a infraestrutura e métodos de segurança que já existem no HTTP. Com a conexão estabelecida, o protocolo reconstrói os quadros do TCP, de forma a criar quadros de mensagens, que são trocadas entre cliente e servidor.

Na Figura 8 temos a comparação entre as mensagens trocadas entre os protocolos HTTP e WebSockets.

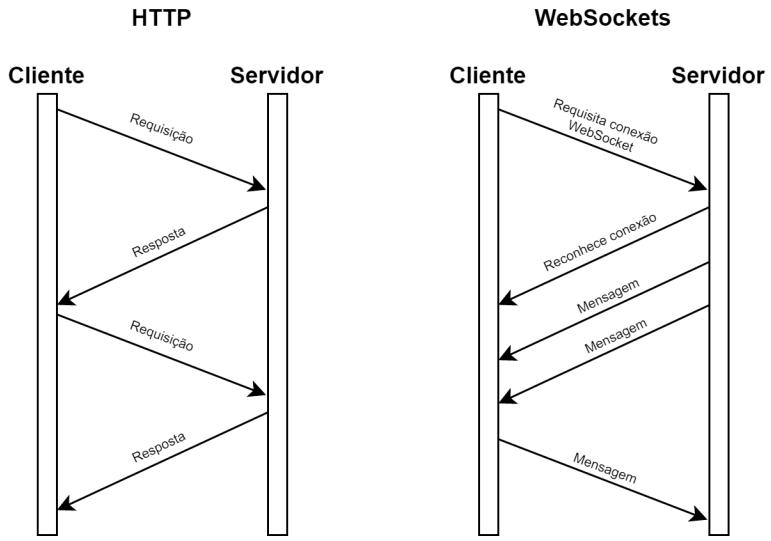

Figura 8 – Funcionamento do HTTP e WebSockets. Fonte: adaptado de Scaleway (2021).

2.2.2.5 TLS

De acordo com a RFC 8446 (2018), o TLS (*Transport Layer Security*) é um protocolo que permite a comunicação segura entre dois pontos. Como condição para funcionamento, o protocolo oferece mecanismos para autenticação, confiabilidade e integridade. Basicamente, o protocolo é composto por duas partes: um protocolo de autenticação, que é responsável por negociar parâmetros de criptografia entre os dois pontos e um protocolo de dados, que usa os parâmetros negociados na autenticação para troca de dados de forma segura.

O TLS pode ser entendido como um protocolo independente que funciona abaixo da camada de aplicação, porém acima da camada de transporte. A camada de transporte requerida para funcionar com TLS necessita que a entrega dos dados seja feita de forma segura e ordenada.

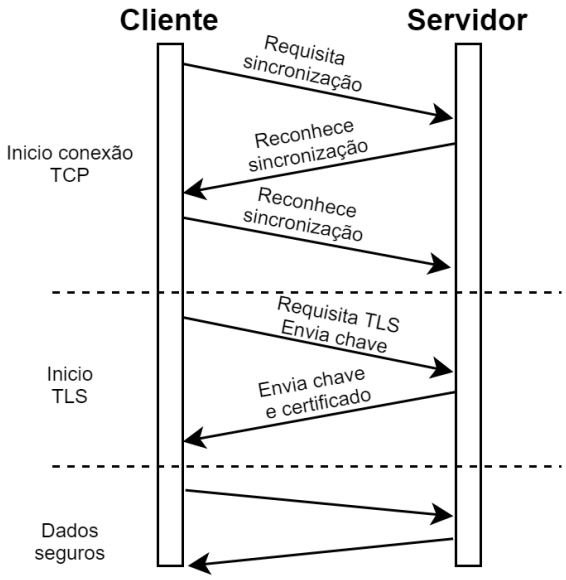

Figura 9 – Inicio do estabelecimento de uma conexão TLS entre cliente e servidor.

2.2.2.6 Conclusão protocolos de transporte para o MQTT

De acordo com os protocolos citados pela OASIS (2021) e que podem ser usados para transporte do MQTT, levantamos as seguintes vantagens e desvantagens do TCP, WebSocket e TLS.

O TCP tem a vantagem de ser rápido na troca de mensagens pois se trata de um protocolo leve e que não necessita de nenhum controle extra para troca das mensagens MQTT. Como desvantagem o protocolo não oferece nenhum mecanismo de segurança de dados.

Já utilizando o Websockets no MQTT, temos a vantagem de conseguir utilizá-lo em navegadores, com sistemas baseados em web, porém o protocolo não oferece segurança dos dados. Quando comparado ao TCP, o protocolo tem um custo adicional para troca de dados, pois adiciona informações extras para controle das mensagens.

O TLS já oferece mecanismos que permitem que os dados trocados sejam protegidos enquanto trafegam pela rede. A segurança vem com a desvantagem de ter custos adicionais para controle da conexão e criptografia entre os dados enviados.

2.2.2.7 CoAP

CoAP (*Contrained Application Protocol*), como definido por Shelby K. Hartke (2014) é um protocolo baseado em web desenvolvido especificamente para aplicações com recursos limitados. O protocolo utiliza o modelo requisição/resposta e foi desenvolvido para ter uma fácil integração com o protocolo HTTP oferecendo pouco custo adicional e simplicidade. A interação entre cliente e servidor no CoAP é similar ao modelo do HTTP.

O cliente faz uma requisição em um recurso no servidor; e o servidor envia uma mensagem de resposta ao cliente com o recurso solicitado. Este processo pode ser visto na Figura 10.

Assim como mostrado por Shelby K. Hartke (2014), o CoAP faz trocas de mensagens entre o cliente e servidor de forma assíncrona, diferentemente do HTTP. Além disso, faz o uso do UDP como protocolo de transporte da camada de rede podendo oferecer confiabilidade na troca de mensagens. Essa confiabilidade é garantida por meio de mecanismos que oferecem retransmissão de pacotes e eliminação de pacotes duplicados de forma mais otimizada e que não implemente todos os recursos do TCP por exemplo.

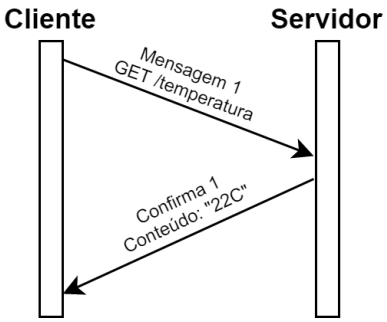

Figura 10 – Troca de mensagens do protocolo CoAP. Fonte: Adaptado de Shelby K. Hartke (2014).

2.2.2.8 mDNS

Antes de introduzir o mDNS (*Multicast Domain Name System*) é importante entender como o DNS funciona. Como mostrado por Kurose e Ross (2017), existem duas formas de se identificar um recurso na internet: pelo seu endereço de IP como por exemplo 121.7.106.83 ou então por seu identificador como por exemplo “www.umsitequalquer.com”. Máquinas tem mais facilidade de entender endereços de IP, enquanto humanos tem mais facilidade de entender o identificador. O DNS se trata de um serviço que traduz um identificador em um endereço de IP. Isso é feito por meio da implementação de um banco de dados distribuído e de uma aplicação que permite efetuar consultas neste banco de dados.

O mDNS (*Multicast DNS*) como definido na RFC6762 (2013) é um protocolo que permite a utilização de recursos de DNS em uma rede local que não contém um servidor de DNS. Seu funcionamento consiste em utilizar os computadores em uma rede local para traduzir endereços que são finalizados com “.local.” como por exemplo “meucomputador.local.”.

Ao iniciar uma consulta de DNS, a aplicação que implementa o mDNS deve verificar se o endereço contém “.local.”. Em caso de sucesso, a consulta de DNS deve ser enviada por meio de uma mensagem *multicast* para toda a rede local. Uma mensagem *multicast* é uma mensagem que é enviada a todos os recursos em uma rede local ao mesmo

tempo. Como a mensagem chega a todos os recursos da rede, o recurso consultado pode responder por meio de outra mensagem *multicast* contendo seu endereço de IP.

A Figura 11 mostra uma consulta feita por meio de mDNS em uma rede local, sendo enviada de um computador para todos os dispositivos na rede. A impressora que atende pelo nome "impressora.local". envia seu endereço como resposta da consulta para todos os dispositivos na rede.

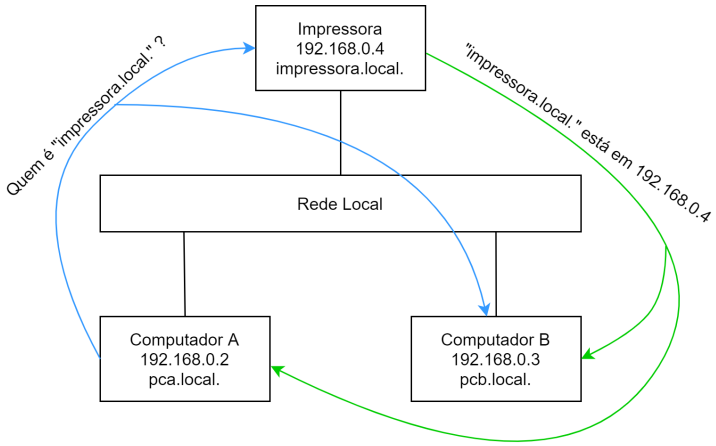

Figura 11 – Fluxo de dados de uma consulta em uma rede local utilizando o protocolo mDNS. Fonte: Adaptado de Dentella (2021).

2.2.2.9 Conclusão protocolos IoT

Com relação aos protocolos apresentados anteriormente, vamos comparar apenas os protocolos de trocas de mensagens que são: HTTP, CoAP e MQTT.

De acordo com um estudo feito por Naik (2017), destacamos alguns cenários de comparação. O estudo não leva em consideração cenários em que existe uma necessidade de retransmissão de dados, o que pode mudar completamente os resultados obtidos.

Com relação a mensagens trocadas entre os protocolos, o MQTT apresenta custos adicionais, pois utiliza como protocolo de transporte o TCP. O CoAP, por sua vez, utiliza o UDP, que o deixa mais leve. O HTTP fica na última posição com relação ao tamanho da mensagem, pois não foi projetado para aplicações IoT, deixando sua mensagem com um tamanho maior.

Considerando o custo de processamento, temos um resultado similar em que notamos que o CoAP requer menos processamento, seguido do MQTT, e por fim com maior custo de processamento temos o HTTP.

Quando analisamos a confiabilidade dos protocolos, o MQTT apresenta o maior índice, uma vez que disponibiliza recursos de qualidade de serviço, além de utilizar como protocolo de transporte o TCP, garantindo a entrega das mensagens.

Naik (2017) mostra também que o MQTT é um protocolo popular no meio de IoT, e que é utilizado por grandes empresas como a IBM, o Facebook, a Cisco, a Amazon Web Services, dentre outras.

2.2.3 Dispositivos IoT

A seguir serão apresentados alguns dispositivos que são utilizados para soluções de Internet das Coisas, sendo eles o ESP8266, o Arduino e o Raspberry Pi.

2.2.3.1 ESP8266

O ESP8266 é um microcontrolador produzido pela Espressif (2021) e que implementa o TCP/IP, possibilitando trabalhar com tecnologia *Wi-Fi*. Sua construção foi feita levando em consideração ser compacto, durável e que tenha economia de energia, fatores que contribuem para o chip ser utilizado no meio de IoT. Para processamento, o chip conta com um processador de 32 bits e pode chegar a um *clock* máximo de 160 MHz. Para memória RAM o ESP8266 disponibiliza cerca de 50kB.

O chip geralmente é montado em uma PCB (*Printed Circuit Board*) e conta com vários pinos (GPIO) que possibilitam comunicação com dispositivos de entrada e saída, como controlar LEDs, botões, controle de energia, dentre outros.

Uma opção para desenvolvimento com o ESP8266 é o NodeMcu (2019), uma plataforma de código aberto que disponibiliza um *firmware* para desenvolvimento de programas para o ESP8266 na linguagem de programação Lua. Diversos módulos foram desenvolvidos para o projeto como, suporte aos protocolos: MQTT, CoAP, HTTP, UART, WPS, entre outros.

Além do *firmware*, foi criado também um kit de desenvolvimento que permite conexão do controlador direto com USB de um computador, permitindo gravar programas dentro do chip, tornando o processo de desenvolvimento mais fácil e ágil. Devido à sua plataforma de código aberto, vários fabricantes oferecem o kit de desenvolvimento com baixo custo, como por exemplo o fornecedor Amica.

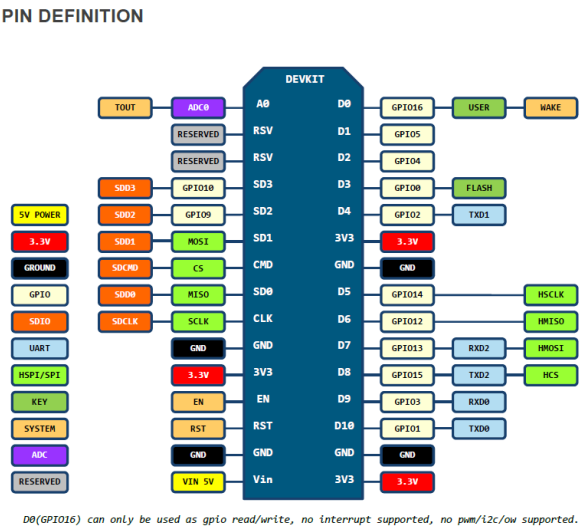

Figura 12 – Definição de pinos do NodeMcu. Fonte: NodeMcu (2019).

2.2.3.2 Arduino

Assim como o NodeMcu, o Arduino (2019) é uma plataforma *open-source* que disponibiliza recursos de hardwares e softwares capaz de ler dados e produzir ações por meio de instruções enviadas a um microcontrolador. Os principais objetivos da plataforma são: possibilitar uma construção rápida de protótipos de maneira fácil, baixo custo de produção, multiplataforma e hardware modularizado.

O Arduino pode ser utilizado em uma gama muito grande de aplicações, uma vez que disponibiliza diversos modelos com diferentes funcionalidades e módulos. Como exemplo podemos citar o Arduino Nano 33 IoT, que foi projetado especialmente para aplicações IoT e utiliza um processador ARM Cortex de 32 bits com um *clock* máximo de 48MHz e 32kB de memória RAM. Para conectividade Wi-Fi o chip utiliza o NINA-10, que permite operar na frequência de 2.4GHz com baixo consumo de energia.

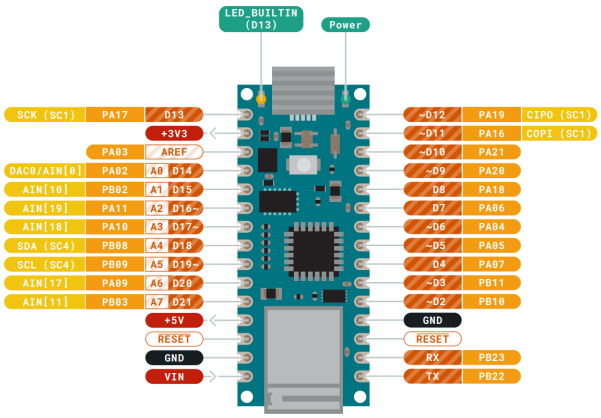

Figura 13 – Definição de pinos do Arduino Nano 33 IoT. Fonte: Arduino (2019).

2.2.3.3 Raspberry Pi

O Raspberry Pi (2021) é um dispositivo desenvolvido pela fundação Raspberry Pi, e tem como objetivo oferecer recursos de hardware e software com grande capacidade computacional pelo menor valor possível. O dispositivo é robusto e pode funcionar como um computador normal para uso no dia a dia, uma vez que é possível usá-lo com um sistema operacional. O dispositivo também é popular no meio de IoT, pelo seu baixo custo e capacidade de modularização de hardware.

Como exemplo temos o modelo Raspberry Pi 4, que conta com um processador Cortex A72 com 4 núcleos que operam em um *clock* de 1.5GHz. Para memória RAM existem opções com 2GB, 4GB ou ainda 8GB. O dispositivo ainda conta com opções para conectividade via cabo e *Wi-Fi* e pode operar nas frequências 2.4GHz e 5.0GHz.

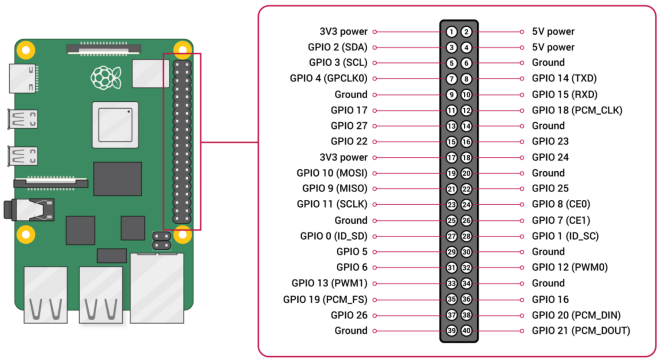

Figura 14 – Definição de pinos do Raspberry Pi. Fonte:Raspberry Pi (2021).

2.2.3.4 Conclusão Dispositivos IoT

Dentre as opções de controladores para IoT apresentadas, a opção de menor custo encontrada foi o ESP8266 por meio do kit de desenvolvimento do NodeMcu e que tem um poder de processamento razoável, porém não tem maiores opções de modularização.

O Arduino tem a vantagem de possuir uma grande quantidade de módulos, porém com menos poder de processamento e um custo mais alto quando comparado ao NodeMcu.

Já o Raspberry Pi tem um poder de processamento muito maior dentre os controladores comparados anteriormente, e de modo geral se trata de uma plataforma mais robusta para desenvolvimento no meio de IoT, porém é a opção com maior custo dentre as mostradas.

2.3 SysML

O OMG (*Object Management Group*) define o SysML (2021) como sendo uma linguagem de modelagem de propósito geral para engenharia de sistemas. A linguagem se

trata de uma extensão do UML.

De acordo com Holt (2018) os diagramas SysML podem ser divididos entre duas características, os que abrangem a parte estrutural e os que abrangem a parte comportamental.

O SysML foi adotado como padrão para especificação e modelagem do sistema desenvolvido. O principal motivo da escolha é que a ferramenta nos possibilita modelar tanto a parte do software quanto a parte do hardware, que foram utilizadas na construção do sistema.

Os principais diagramas utilizados para especificação deste projeto foram descritos nas seções a seguir. Nem todos os diagramas disponíveis do SysML foram utilizados.

2.3.1 Diagramas estruturais

Holt (2018) diz que os diagramas estruturais podem ser utilizados para modelar qualquer parte da arquitetura de um sistema, sendo diagramas de sistemas de alto nível até baixo nível, como por exemplo componentes específicos.

2.3.1.0.1 Diagrama de definição de blocos

Segundo Holt (2018), o diagrama de definição de blocos é utilizado para representar os aspectos estruturais de um projeto e como estes aspectos se relacionam entre si. São formados por dois elementos principais, os blocos e suas relações. Um bloco é caracterizado pela marcação «block» e representam elementos dentro de um sistema, como por exemplo o bloco: assistente virtual.

Figura 15 – Bloco SysML para a entidade assistente virtual.

Holt (2018) afirma que as relações são representadas por uma linha entre os blocos. É possível ter no meio da linha uma associação que quando lida em conjunto com os blocos, forma uma frase, como por exemplo: assistente virtual consulta serviço de localização. A seta representada por um triângulo preto indica a direção em que a associação deve ser lida.

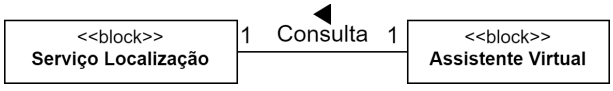

Figura 16 – Relação um para um que indica que uma assistente virtual consulta um serviço de localização.

Além de relacionar os elementos, Holt (2018) diz que as associações podem também mostrar a multiplicidade entre os elementos, que são representadas por números ou símbolos no fim de cada lado da linha. Podemos ter uma relação 1 para 1 ("1..1"), como por exemplo: 1 serviço de localização consulta 1 assistente virtual. Mas podemos ter também casos de uma relação de 1 para muitos("1..*"), como por exemplo: 1 assistente virtual consulta muitos semáforos. Os valores sempre podem ser alterados para que a associação mostre a multiplicidade entre os elementos.

Figura 17 – Relação um para muitos, indicando que uma assistente virtual consulta um ou mais semáforos.

Como demonstrado por Holt (2018), além da associação, temos também a composição que pode ser utilizada para definir relações entre os blocos. Ela é representada pelo diamante e indica que um bloco faz parte de outro. No exemplo é possível observar que o aplicativo é formado por todos os serviços ao seu redor.

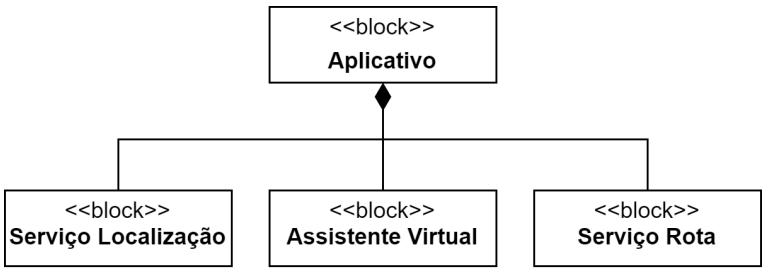

Figura 18 – Composição SysML indicando que a entidade aplicativo é composta por três elementos.

2.3.1.0.2 Diagrama de requerimentos

O diagrama de requerimentos demonstrado por Holt (2018) é um diagrama estrutural e que permite que um requerimento seja relacionado aos demais elementos do modelo. O diagrama de requerimentos também usa os blocos que vimos no diagrama anterior, porém agora com a marcação «requirement».

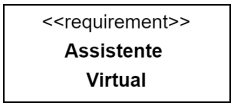

Figura 19 – Requerimento SysML para a entidade assistente virtual.

No exemplo temos o requerimento principal do aplicativo, que é uma assistente virtual. A assistente está relacionada com outros três requerimentos, navegação, salvamento

de pontos de interesse e informações posicionais, que por sua vez são sub requerimentos da assistente virtual. Para representar a relação de sub requerimento, a ponta da relação do requerimento principal é marcada com um círculo, que pode ser visto na Figura 20.

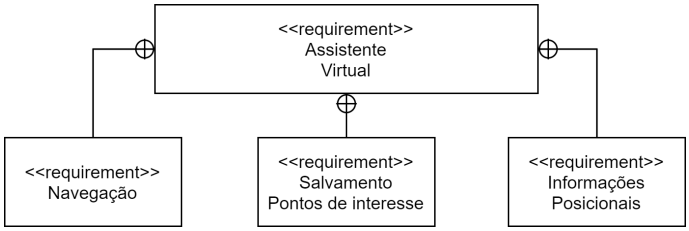

Figura 20 – Diagrama de requerimentos para a assistente virtual, indicando os sub requerimentos: navegação, salvamento de pontos de interesse e informações posicionais.

2.3.1.0.3 Diagrama de pacotes

Como apresentado por Holt (2018), o diagrama de pacotes é utilizado principalmente para organização. Cada pacote mostra um conjunto de elementos, que estão incluídos dentro do pacote. Os pacotes ainda podem ser relacionados entre si, com relações de dependência e importação em outros pacotes.

Além disso, de acordo com Holt (2018) esse diagrama pode ser mostrado em qualquer outro diagrama SysML, com intuito de mostrar em qual parte estrutural do projeto aquele diagrama se encontra. No exemplo da Figura 21, o retângulo que engloba todos os elementos é um diagrama de pacote.

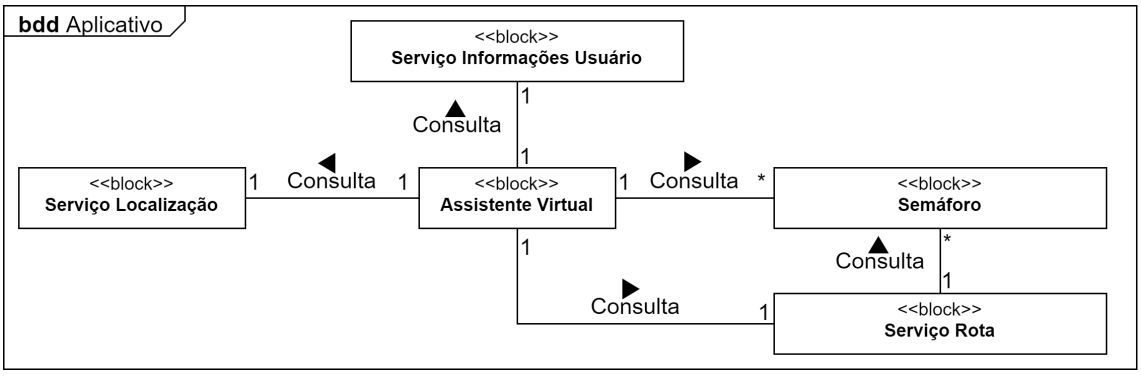

Figura 21 – Pacote que engloba um diagrama de blocos para o aplicativo

2.3.2 Diagramas comportamentais

Os diagramas comportamentais, como destacado por Holt (2018), são utilizados para apresentar comportamentos entre níveis do sistema ou entre seus elementos, mostrando, por exemplo, operações realizadas entre elementos.

2.3.2.0.1 Diagrama de atividades

O diagrama de atividades apontado por Holt (2018), é um diagrama comportamental e que é utilizado para mostrar comportamento de uma operação específica. Com ele, pode ser feita uma modelagem de baixo nível. No exemplo da Figura 22 há um diagrama de atividades que representa a inicialização e funcionamento do serviço de rota do aplicativo, que será apresentado na seção 3.3.5.

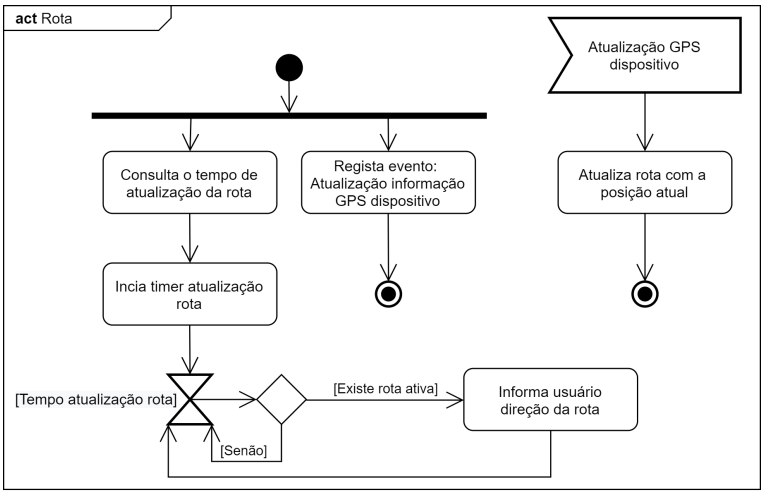

Figura 22 – Diagrama de atividades indicando a inicialização do serviço de rota.

Em um diagrama de atividades, um nó de ação é definido por um retângulo com os cantos arredondados e pode ser utilizado para representar um comportamento que será executado. Na Figura 23 é possível observar uma ação que efetua a atualização da rota. Além das ações, Holt (2018) destaca que podemos utilizar também símbolos para mostrar eventos e sinais. Eventos são utilizados para mostrar um comportamento que se inicia fora do diagrama e os sinais são usados para mostrar um comportamento que irá continuar em outro diagrama. No exemplo da Figura 23 temos um evento de atualização do GPS, que é disparado fora do diagrama.

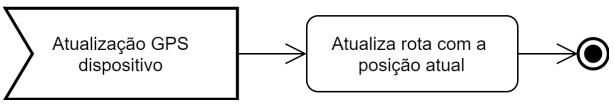

Figura 23 – Nó de evento seguido por um nó de ação em um diagrama de atividades.

Como retratado por Holt (2018), para mostrar início e fim do fluxo de execução do diagrama, temos o símbolo de um círculo colorido e um círculo com bordas brancas, que representam respectivamente início e fim. Já para indicar a direção do fluxo, são utilizadas linhas com setas apontadas para cada direção. Um fluxo pode ser divido em dois e essa divisão pode ser representada com um traço, que gera dois ou mais fluxos de saída.

A Figura 24 mostra uma execução simplificada para exemplificar o uso dos mecanismos de fluxo de um diagrama de atividades.

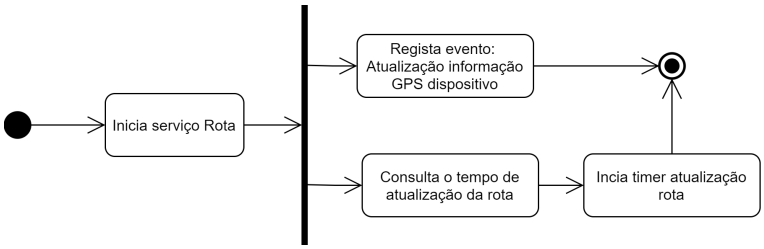

Figura 24 – Execução simplificada do inicio do serviço de rota, mostrando inicio e fim de um fluxo, direção do fluxo e divisão de fluxo em um diagrama de atividades.

Segundo Holt (2018) temos ainda um nó de decisão que é representado por um losango e é utilizado para executar fluxos sob determinada condição. No exemplo da Figura 25 temos uma condição, em que a ação "Informa usuário direção da rota" é executada apenas quando existe uma rota ativa.

Delligatti (2014) diz que para representar uma pausa no fluxo, podemos utilizar um temporizador que é representado pelo símbolo da ampulheta. Na Figura 25 temos uma pausa logo antes do nó de decisão.

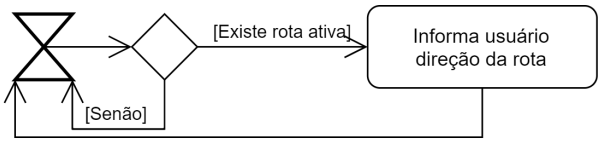

Figura 25 – Diagrama de atividades com um nó de decisão para verificação de rota ativa e um nó de pausa.

2.4 Definições Gerais

2.4.1 Texto para fala

Segundo Taylor (2009), o termo "texto para fala", do inglês "*text to speech*", ou simplesmente TTS, se trata da conversão de um texto escrito para uma fala gerada por computador. O termo "fala para texto" se trata do processo contrário, onde uma fala é convertida para um texto escrito.

Ainda como dito por Taylor (2009), estes mecanismos são importantes, pois podem ser utilizados em diversas aplicações. Um dos primeiros usos da tecnologia foi no auxílio de deficientes visuais, para auxiliar na leitura de textos e livros. A evolução dos sistemas baseados em texto para fala possibilitou que deficientes visuais consigam utilizar dispositivos e sistemas de maneira fácil e rápida.

2.4.2 JSON

O JSON (*JavaScript Object Orientation*) é um formato para serialização de dados, que são representados por estruturas do tipo objeto ou *array*. Um objeto é uma coleção de pares com nome e valor, onde o nome é um texto e os valores podem conter dados como textos, números, valores booleanos e nulo. Um *array* é uma coleção de um ou mais valores. (RFC7159, 2014)

Em JSON o início e fim de um *array* é representado pela abertura e fechamento de um colchete ("[ ]") e os elementos dentro do *array* são separados por vírgula. Para representar o início e fim de um objeto, são utilizados a abertura e fechamento de uma chave ("{ }") e os pares de valores dentro dos objetos são separados por vírgula. (RFC7159, 2014)

Como exemplo temos o JSON a seguir, que mostra a representação de um *array* com dois objetos. Dentro dos objetos temos os pares de valores que representam os dados serializados.

```
1  [
2      {
3          "locationid": 1,
4          "latitude":  -18.91867,
5          "longitude": -48.25967,
6          "locationname": "Bloco 1B"
7      },
8      {
9          "locationid": 2,
10         "latitude":  -18.91796,
11         "longitude": -48.25910,
12         "locationname": "Bloco 5R-A"
13     }
14 ]
```

2.4.2.1 GeoJSON

Um GeoJSON é um formato de representação baseado no JSON, utilizado para representar informações geográficas. Com ele, podem ser representadas estruturas como pontos, linhas e polígonos. (RFC7946, 2016)

O GeoJSON introduz um tipo de objetos para cada estrutura representada. Um desses tipos é o "LineString", que é utilizado para representar um conjunto de linhas utilizando um *array* de pontos. Um ponto em GeoJSON é representado por um par de coordenadas dentro de um *array*, representando latitude e longitude respectivamente, como por exemplo: [1.01 , 1.02]. (RFC7946, 2016)

O exemplo a seguir mostra um objeto GeoJSON retornado pela MapBoxAPI, que foi utilizada para execução deste trabalho e será detalhada na seção 3.4.8.2. No objeto temos caminho composto por três pontos representados em uma "LineString".

```
{
    "geometry": {
        "type": "LineString"
        "coordinates": [
          [-48.258245, -18.916893],
          [-48.258219, -18.917002],
          [-48.258312, -18.91702]
        ]
    }
}
```

2.4.3 API Web

De acordo com Sohan, Anslow e Maurer (2015), uma API Web é uma interface de programação utilizada para conectar serviços de softwares na internet. Uma API Web pode seguir vários formatos de padronização, como exemplo podemos citar o modelo SOAP e o modelo RESTful. Salvadori (2015) diz que as APIs Web se aproveitam da estrutura existente do HTTP e utilizam seus métodos para implementação de seus serviços. Seu uso possibilita que seja feita uma integração de dados transparente e independente de uma base de dados entre uma ou mais aplicações.

3 Arquitetura

Neste capítulo será proposta uma arquitetura para o aplicativo, seguido das informações utilizadas para implementação de um protótipo baseado nessa arquitetura.

A arquitetura do protótipo desenvolvido conta com três partes principais (Figura 26) que são: a parte do semáforo que é composta pelo semáforo em si e um dispositivo Raspberry PI; a parte da nuvem, onde ficarão hospedados os programas responsáveis pela comunicação com todas as partes do sistema juntamente com o banco de dados; e por fim o aplicativo, que será usado pelo usuário final. Essas partes serão detalhadas nas seções a seguir.

Para um melhor entendimento da arquitetura, algumas partes foram nomeadas de acordo com a tecnologia/equipamento utilizados no protótipo desenvolvido.

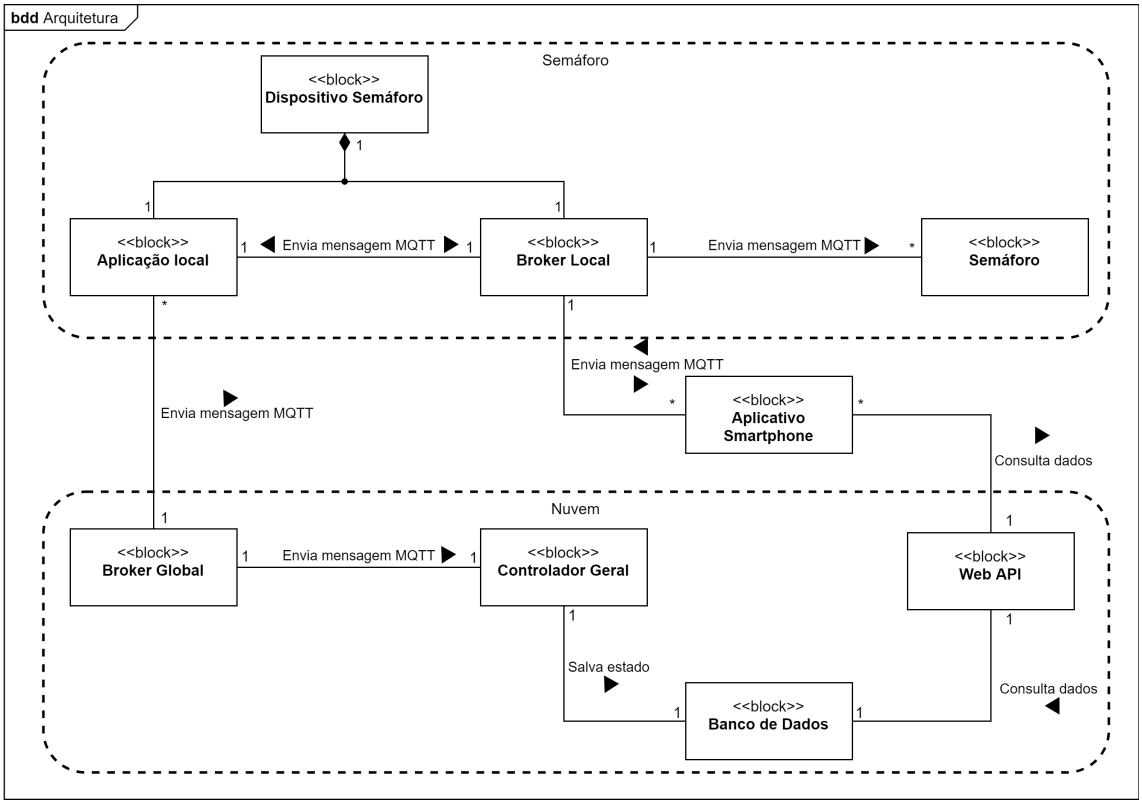

Figura 26 – Arquitetura geral que mostra todos os pontos da arquitetura com o semáforo o aplicativo e a nuvem.

3.1 Semáforo

O semáforo em si, trata-se de um dos pontos mais simples da arquitetura. É proposto que um ou mais semáforos sejam conectados a uma rede local e que essa rede

local possua um dispositivo que controle o estado (verde, amarelo ou vermelho) de cada semáforo.

Para atualizar seu estado, o semáforo deve possuir conectividade com uma rede local para se comunicar com um *broker* local e esperar por mensagens de atualização de estado. Ao receber a mensagem, a luz referente ao estado recebido deve ser ligada. Toda a parte de controle dos estados e temporização é feita em uma aplicação que roda no dispositivo IoT e será apresentada na próxima seção. Nas Figuras 27 e 28 é mostrado todo o fluxo executado dentro do semáforo

Na Figura 27 temos a inicialização do dispositivo, onde primeiramente são registrados eventos para controle da conexão de rede, e em seguida a conexão é iniciada. Caso não haja sucesso na conexão a execução é interrompida.

Com o sucesso na conexão de rede, segue-se os passos para controle dos estados do semáforo propostos na figura 28. Em um primeiro momento é feita a conexão com o *broker* MQTT na rede local e em seguida é feita a inscrição em um tópico para receber as mensagens de atualização. A mensagem recebida é então decodificada e o LED correspondente ao estado do semáforo será ativado.

A mensagem com os dados do semáforo sempre será a mesma em todas as partes da arquitetura, se trata de um JSON com os seguintes campos:

```
{
    "trafficlightid":1,
    "lastupdate":1633950344,
    "status":3,
    "latitude":0.0,
    "longitude":0.0
}
```

3.1.1 Dispositivo IoT

Na arquitetura, o dispositivo IoT faz parte do semáforo, sendo responsável por controlar os estados desse semáforo e fazer a troca de mensagens MQTT entre a rede local e a nuvem. Com isso temos três partes importantes dentro: uma aplicação responsável por controlar o semáforo; um *broker* MQTT; e uma aplicação responsável por implementar o protocolo mDNS.

3.1.1.1 Aplicação local

Na inicialização da aplicação, é feita a conexão com dois *brokers* MQTT, sendo o primeiro o *broker* global, que recebe os estados de todos os semáforos do sistema, e o segundo o *broker* local, que é utilizado pelo semáforo para atualizar as luzes, e pode ser

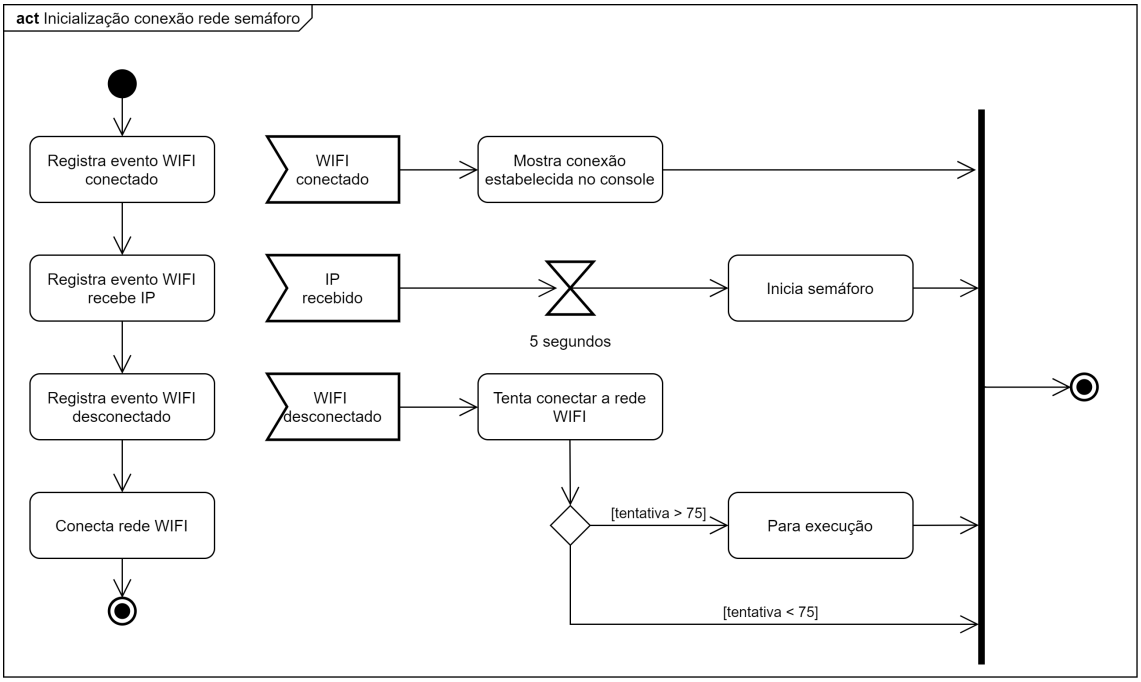

Figura 27 – Inicialização do semáforo com conexão *WiFi*.

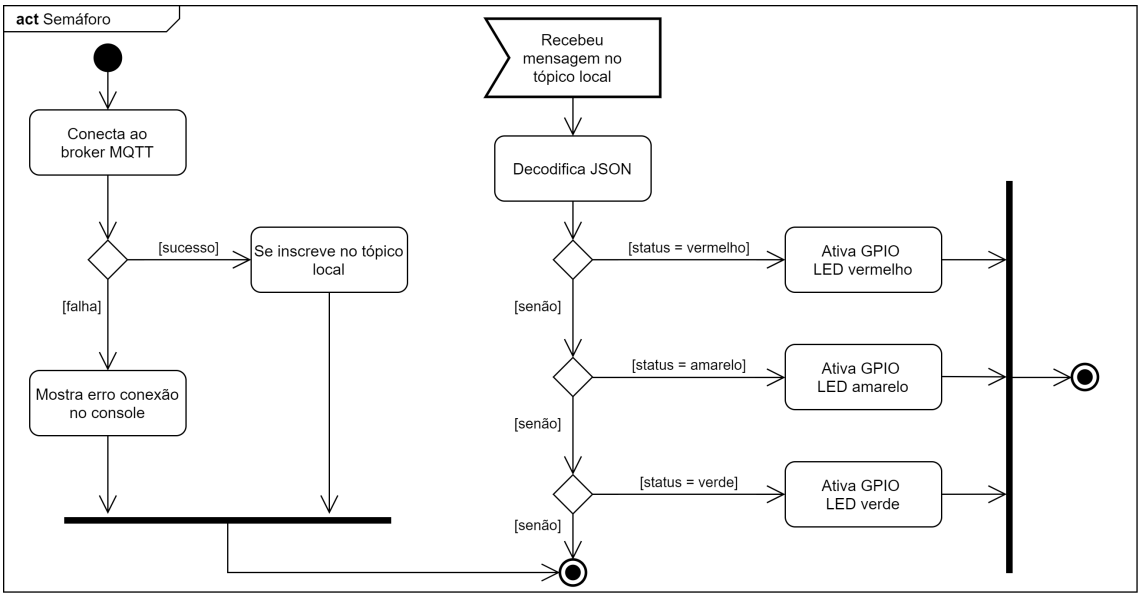

Figura 28 – Conexão do semáforo com o *broker* MQTT e controle de estados para ativação de LEDs.

utilizado pelo aplicativo do usuário. Após a conexão, a aplicação se inscreve no tópico "/crossing" do *broker* local. Caso tenha recebido uma mensagem nesse tópico significa que existe um pedestre a efetuar a travessia no semáforo. Nesse caso, a aplicação fica por um tempo maior no estado vermelho para que o usuário tenha um tempo extra para atravessar a via.

Com a inicialização finalizada, se inicia a atualização dos estados do semáforo, onde a aplicação calcula os próximos estados e o tempo que deve permanecer naquele

estado. Além disso, a aplicação faz a publicação de uma mensagem MQTT para os dois *brokers* que contém o estado atual do semáforo.

Na Figura 29 temos o processo de inicialização da aplicação que faz a conexão com os dois *brokers* MQTT e a cada troca de estado publica as atualizações. O cálculo de tempo de cada estado foi detalhado na Figura 30 e sempre irá retornar um tempo fixo para cada estado, exceto quando existe um pedestre a efetuar a travessia. Neste caso, o tempo do estado vermelho será multiplicado por dois, para que a pessoa tenha mais tempo para atravessar a via.

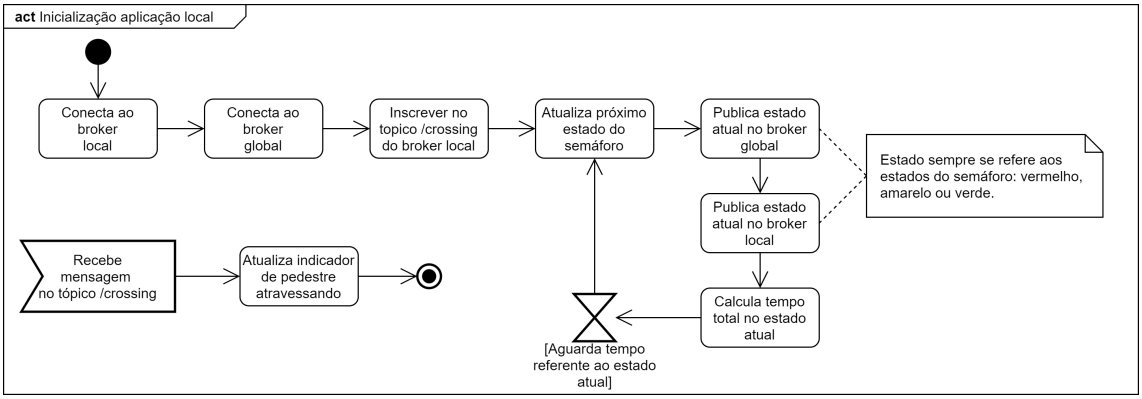

Figura 29 – Visão geral da aplicação local com a inicialização da mesma, onde é feita a conexão em dois *brokers*, publicação de estados e recebimento de sinais indicando que existe um pedestre no local.

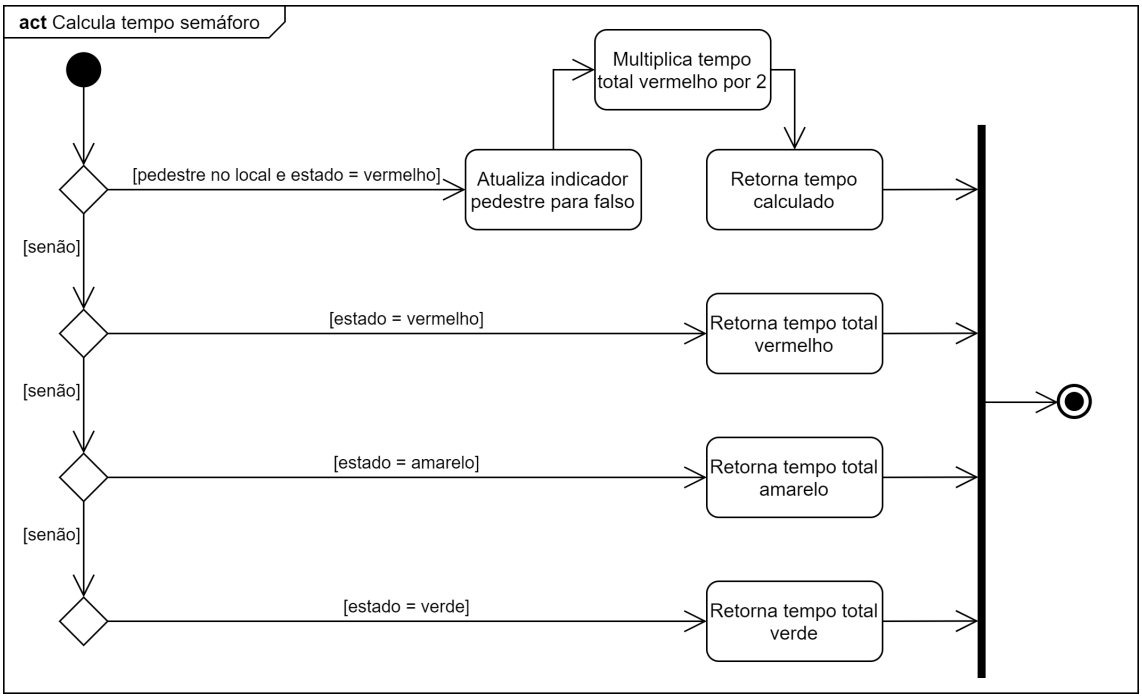

Figura 30 – Diagrama de atividades com cálculo de tempo dos estados do semáforo na aplicação local.

3.1.1.2 Broker MQTT

No dispositivo IoT temos também um *broker* MQTT que é responsável pela comunicação direta com os dispositivos na rede local, sendo estes os semáforos e os celulares de usuários finais.

3.1.1.3 mDNS

Para que o aplicativo possa se conectar de maneira transparente ao *broker* do dispositivo IoT, é preciso que o celular tenha conhecimento do endereço de rede do dispositivo. Para isso, utiliza-se o protocolo mDNS. Então, se torna necessário no dispositivo IoT a implementação do mDNS para publicar o endereço do *broker* local na rede. A implementação do aplicativo será detalhada na seção 3.3.6.1.

3.2 Nuvem

A seguir serão detalhados os pontos da arquitetura que devem estar hospedados na nuvem para que possam ser acessados de qualquer lugar a qualquer momento.

3.2.1 Banco de Dados

O banco de dados deve ser hospedado na nuvem para que possa ser acessado a qualquer momento pelo controlador geral e a API Web; ambos serão detalhados nas próximas seções. Na Figura 31 temos a especificação de todas as tabelas utilizadas e seus relacionamentos.

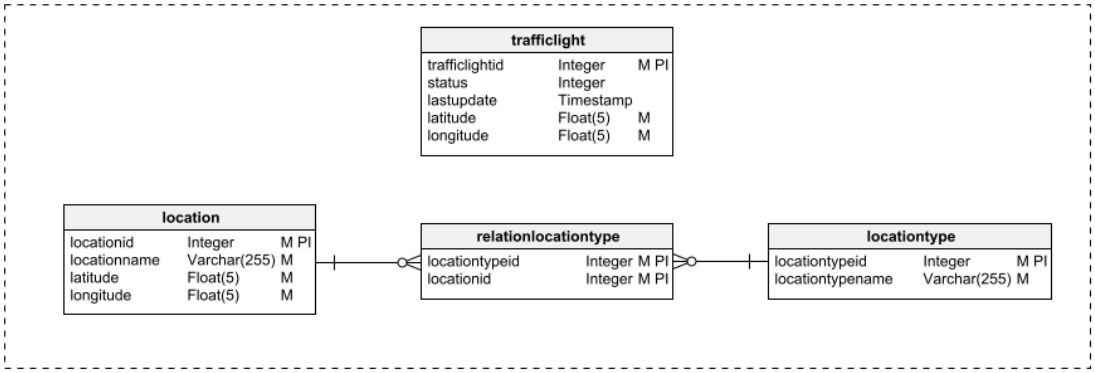

Figura 31 – Modelo de entidade e relacionamento das tabelas do banco de dados

3.2.1.1 Tabelas Controlador Geral

Para mapear os dados do semáforo no banco de dados será utilizada a tabela "*trafficlight*". Onde temos as informações do semáforo como: localização, data da última

atualização e um identificador que indica o estado atual do semáforo. A atualização do estado do semáforo é feita no controlador geral que será mostrado na seção 3.2.3.

3.2.1.2 Tabelas API Web

A API Web acessa o banco de dados em dois momentos: para consultar pontos de interesse e para consultar informações de estado dos semáforos. Maiores detalhes são mostrados na Seção 3.2.4.

Para consulta de pontos de interesse foram criadas três tabelas no banco, que estão especificadas na Figura 31. A primeira tabela é chamada de "*locationtype*" e armazena um texto com o tipo da localidade, como por exemplo: restaurante, ponto de ônibus, escolas etc. A segunda tabela, chamada de "*location*", guarda informações de localidades, com seus respectivos nomes e localizações. São exemplos de localidades: bloco 3Q, biblioteca central etc.

Por fim, foi criada a tabela "*relationlocationtype*" para relacionar uma localidade com um ou mais tipos de localidade, como por exemplo a localidade bloco 3Q, que pode estar associada ao tipo de localidade universidade e também a bloco de estudos.

3.2.2 *Broker* global

O *broker* MQTT tem como função receber as mensagens publicadas por todos os dispositivos IoT e repassá-las ao controlador geral, que será explicado em detalhes na próxima seção.

3.2.3 Controlador Geral

Trata-se da aplicação que utiliza o protocolo MQTT para receber atualizações de estados de todos os semáforos. Na inicialização, o controlador se conecta ao *broker* global e se inscreve em um tópico para receber as mudanças de estados enviadas pela aplicação do dispositivo IoT. A inicialização do controlador é mostrada no diagrama de atividades da Figura 32.

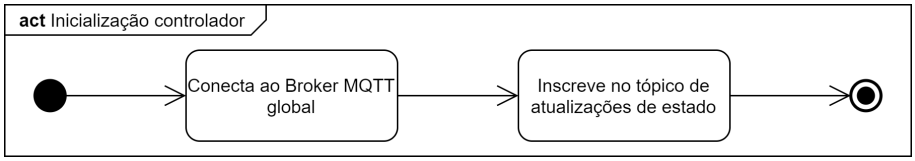

Figura 32 – Diagrama de atividades com a inicialização controlador geral, mostrando a conexão com o *broker* global e a inscrição no tópico de atualizações de estado.

Ao receber uma atualização de estado, a aplicação desserializa a mensagem JSON recebida no tópico de atualização e em seguida atualiza o estado direto na tabela de

semáforo no banco de dados. O formato do JSON segue o mesmo padrão mostrado na seção 3.1. Esse processo é mostrado no diagrama de atividades da Figura 33.

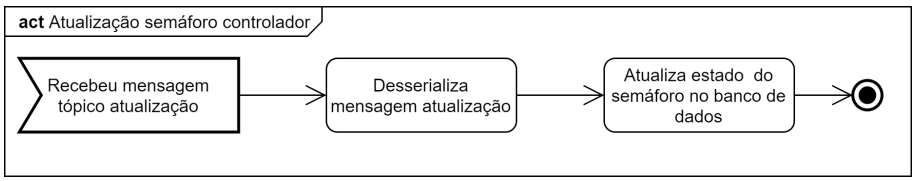

Figura 33 – Diagrama de atividades com o recebimento do evento de estado do semáforo e atualização no banco de dados.

3.2.4 API Web

A API Web é utilizada para comunicação direta com o aplicativo. Foram implementados métodos na API que são responsáveis por consultar os semáforos e também as localidades salvas no banco de dados.

Para os métodos relacionados ao semáforo é possível consultar cada um pelo seu identificador e também é possível verificar se existe um semáforo nas proximidades dada uma localização. Para os métodos associados às localidades do banco de dados, é possível consultar uma localização por seu identificador, proximidade, proximidade por tipo e nome. Na tabela 1 temos uma visão geral com todos os métodos da API Web, e nas próximas seções o detalhamento de cada um.

3.2.4.1 Métodos semáforo

Para a consulta dos dados do semáforo são necessários dois métodos. O retorno para ambos os métodos é o mesmo, sempre uma lista em JSON com zero ou mais semáforos, os atributos de cada semáforo que estão no retorno estão representados na tabela 2. Na seção 3.1 foi mostrado um JSON com a mesma estrutura que será retornado pela API.

O primeiro método, implementado na rota: "/trafficlight/:id", é utilizado pelo aplicativo para consultar o estado atual de um semáforo específico. Os parâmetros de entrada do método foram mostrados na tabela 3. Na implementação deste método é feita uma consulta ao banco de dados de acordo com o identificador passado por parâmetro, e então o semáforo é retornado, este processo é mostrado na Figura 34.

O segundo método da rota: "/trafficlight/:latitude/:longitude/:heading", tem como função retornar semáforos mais próximos as coordenadas repassadas, levando em consideração sempre a direção que é passada como parâmetro, que deve corresponder a direção do semáforo. Os parâmetros foram especificados na tabela 4. Para implementação como mostrado na Figura 34, é feito um cálculo de distância e outro de direção entre todos

Tabela 1 – Visão geral com todas as rotas da API Web e a função de cada uma.

Mét.	**Rota**	**Descrição**
GET	/trafficlight/:id	Retorna o semáforo com identificador passado por parâmetro.
GET	/trafficlight/:lat/:long/:heading	Verifica se existe um semáforo na direção informada por parâmetro.
GET	/location/:id	Retorna uma localização com o identificador passado por parâmetro.
GET	/location/points-of-interest/:lat/:long	Retorna pontos de interesse próximos a localização geográfica passada por parâmetro.
GET	/location/points-of-interest/:lat/:long/:type	Retorna pontos de interesse por tipo que estão próximos a localização geográfica passada por parâmetro.
GET	/location/search/:locationname	Retorna uma localização que corresponde ao nome passado por parâmetro.

Tabela 2 – Atributos de retorno da requisição /trafficlight

Nome atributo	**Tipo dados**	**Descrição**
latitude	Decimal	Latitude
longitude	Decimal	Longitude
status	Inteiro	Status atual do semáforo
lastUpdate	Data Hora	Data da última atualização de status

Tabela 3 – Parâmetros da requisição /trafficlight/:id

Nome parâmetro	**Tipo parâmetro**	**Tipo dados**	**Descrição**
id	Query	Inteiro	Id do semáforo

os semáforos e as coordenadas repassadas. Com isso, é possível pegar o semáforo mais próximo, caso as coordenadas apontem para a direção dele. Maiores detalhes referente aos cálculos de direção e distância são explicados na seção 3.3.5.

3.2.4.2 Métodos Localização

Para a consulta de dados relacionados a localidades, foram criados três métodos na rota "/location". Todos sempre retornam o mesmo tipo de dados: um JSON com uma lista de localidades. Seus atributos foram mostrados na tabela 5. A seguir temos um exemplo de JSON com a lista de localidades que é retornada.

Tabela 4 – Parâmetros da requisição /trafficlight/:latitude/:longitude/:heading

Nome parâmetro	Tipo parâmetro	Tipo dados	Descrição
latitude	Query	Decimal	Latitude
longitude	Query	Decimal	Longitude
heading	Query	Decimal	Direção em graus

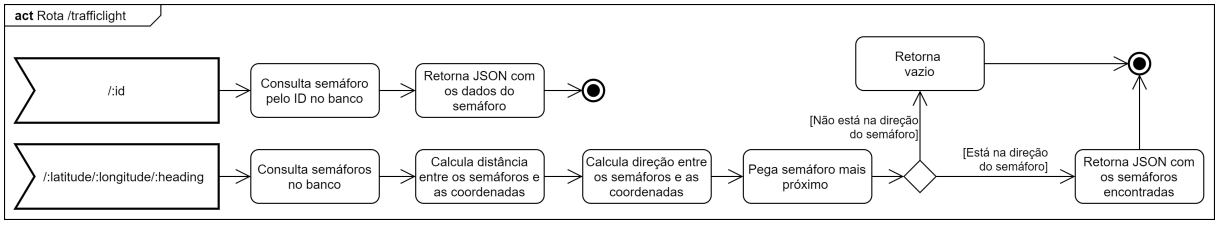

Figura 34 – Diagrama de atividades com os métodos de consulta ao semáforo na API Web.

Tabela 5 – Atributos de retorno da requisição /location

Nome parâmetro	Tipo dados	Descrição
locationid	Inteiro	Id da localização
locationname	Texto	Nome da localização
latitude	Decimal	Latitude
longitude	Decimal	Longitude

```
[
    {
        "locationid":1,
        "locationname":"bloco 1b ufu",
        "latitude":-18.91867,
        "longitude":-48.25967
    },
    {
        "locationid":3,
        "locationname":"bloco 5ra ufu",
        "latitude":-18.91796,
        "longitude":-48.25910
    }
]
```

Com o objetivo de consultar uma localização por seu identificador, pode ser usada a rota "/location/:id". Essa rota faz uma consulta ao banco de dados com o identificador repassado e retorna o local pesquisado. Esse processo pode ser visto na Figura 35.

Para consultar pontos de interesse próximos a uma localização foi criada a rota

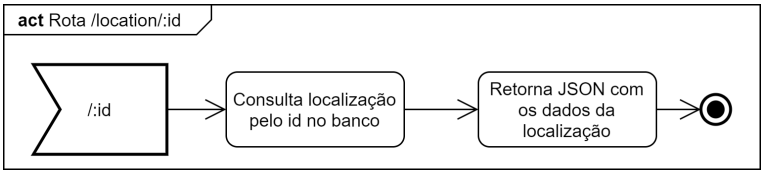

Figura 35 – Diagrama de atividade com o método de consulta de localidade por identificador.

"/location/points-of-interest/:latitude/:longitude". Sempre levando em consideração as coordenadas do local, é retornada uma lista de localidades próximas. Os parâmetros de entrada foram mostrados na tabela 6. Para implementação é necessário fazer o cálculo de distância considerando a localização repassada, como mostrado na Figura 36. O cálculo é mostrado na seção 3.3.5.

Tabela 6 – Parâmetros da requisição /location/points-of-interest/:latitude/:longitude

Nome parâmetro	Tipo parâmetro	Tipo dados	Descrição
latitude	Query	Decimal	Latitude
longitude	Query	Decimal	Longitude

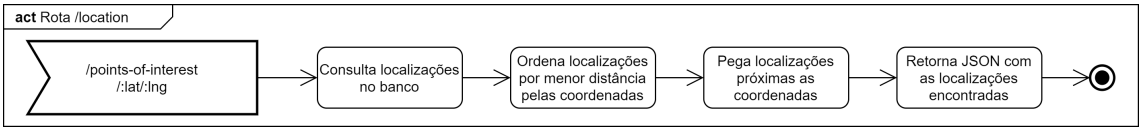

Figura 36 – Diagrama de atividade com o método de consulta de localidade por coordenadas.

Para a consulta de pontos de interesse próximos com um tipo específico, foi criada a rota "/location/points-of-interest/:latitude/:longitude/:type". Muito simular ao anterior, com a diferença de um parâmetro extra para especificar o tipo de locais a serem retornados. Os parâmetros de entrada foram mostrados na tabela 7. Para sua implementação, é necessário consultar no banco de dados apenas as localizações que tem o tipo do parâmetro, e a seguir, o cálculo de distância é repetido para cada uma destas localizações. O cálculo será mostrado na seção 3.3.5. O diagrama de atividades da Figura 37 tem informações deste processo.

Tabela 7 – Parâmetros da rota /location/points-of-interest/:latitude/:longitude/:type

Nome parâmetro	Tipo parâmetro	Tipo dados	Descrição
latitude	Query	Decimal	Latitude
longitude	Query	Decimal	Longitude
type	Query	Texto	Tipo da localização

O último método relacionado à localização está disponível em "/location/search/:locationname", e consulta localizações com nomes similares ao repassado na consulta.

Figura 37 – Diagrama de atividade com o método de consulta de localidade por coordenadas e tipo.

O parâmetro de nome foi mostrado na tabela 8. Para sua implementação, deve ser feita uma consulta no banco de dados por nomes similares ao repassado, e então retornar todos estes, processo mostrado na Figura 38.

Tabela 8 – Parâmetros da requisição /location/search/:locationname

Nome parâmetro	Tipo parâmetro	Tipo dados	Descrição
locationname	Query	Text	Nome da localização

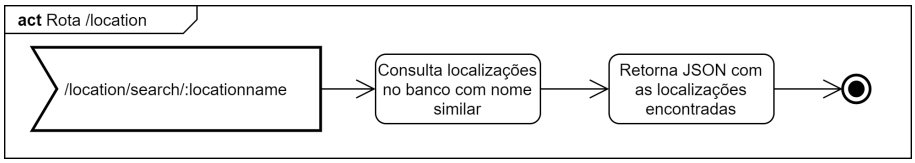

Figura 38 – Diagrama de atividade com o método de consulta de localidade por nome.

3.3 Aplicativo

A seguir será apresentada a arquitetura do aplicativo para *smartphones* que tem o objetivo de guiar o usuário em diferentes cenários por meio de uma assistente virtual utilizando os elementos ao seu redor.

3.3.1 Requisitos

Como requisito principal temos uma assistente virtual que será crucial para a interação com o usuário e além de mecanismos que permitam a percepção dos arredores com serviços de navegação, informações posicionais e de semáforo além de cadastramento de pontos de interesse. Na Figura 39 temos os requisitos para construção.

O cenário mínimo para execução da aplicação é conseguir guiar o usuário entre dois pontos fixos dentro da universidade. Exemplo: dar instruções para movimentação da estação UFU João Naves até o bloco 1B.

O cenário máximo para execução da aplicação é conseguir guiar o usuário para qualquer local relacionado a universidade. Exemplo: Dar instruções para movimentação de qualquer bloco da UFU até a estação de ônibus mais próxima.

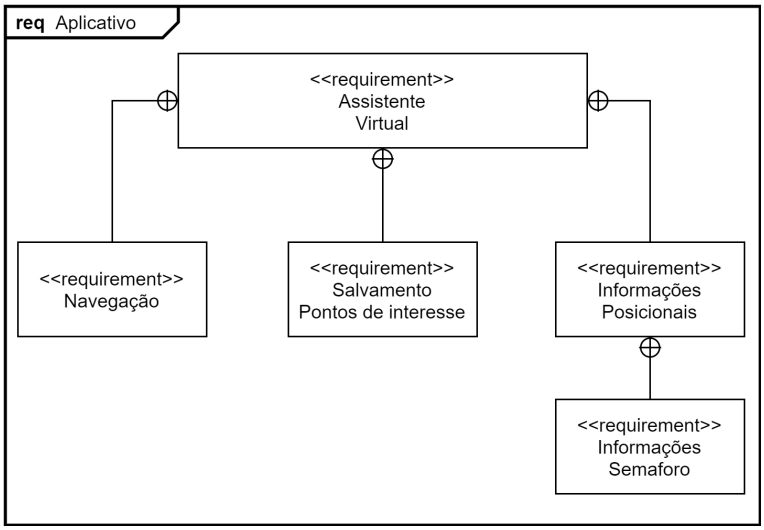

Figura 39 – Diagrama de requisitos geral para o aplicativo

3.3.2 Arquitetura

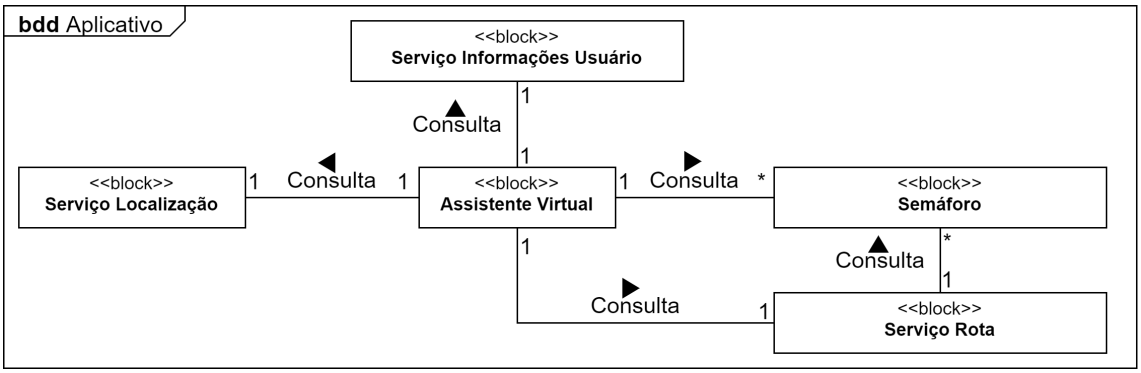

Figura 40 – Diagrama de blocos com a arquitetura do aplicativo indicando todos os serviços e dependências.

A arquitetura do aplicativo é composta por 5 elementos principais:

1. Serviço localização: responsável por consultar os serviços relacionados a localização, mapas e rotas.

2. Serviço informações usuário: responsável pela inserção e consulta de dados que serão importantes para usabilidade e controle de navegação do usuário.

3. Serviço rota: tem como principal objetivo controlar uma rota escolhida pelo usuário. Mantém controle de localização e de elementos ao redor. Além disso faz o cálculo de direção que o usuário deve seguir para concluir a rota.

4. Semáforo: o aplicativo pode se comunicar diretamente ao semáforo, caso esteja em suas proximidades, e assim auxiliar o usuário caso ele precise atravessar a via que tem um semáforo conectado.

5. Assistente virtual: responsável pela comunicação direta com o usuário, liga todos os pontos da aplicação.

   A seguir estes elementos serão detalhados.

3.3.3 Assistente virtual

Uma das principais funcionalidades do aplicativo é uma assistente virtual batizada com nome Cida. A assistente deve utilizar mecanismos de conversão de texto em fala (text to speech, em inglês) e fala para texto (speech to text, em inglês) para comunicação com o usuário. Desta forma, toda a interação com a assistente é feita via comandos de voz.

O texto para fala (text to speech, em inglês) é utilizada para reconhecer os comandos de voz do usuário e transformá-los em texto, e assim identificar a ação requisitada pelo usuário. Já a fala para texto (speech to text, em inglês) é aplicada para dar a resposta ao usuário das informações solicitadas ou confirmar as ações efetuadas.

Para requisitar a assistente é necessário um mecanismo de fácil acesso ao usuário, como exemplo o gesto de tremer o dispositivo.

Na inicialização do aplicativo, e consequentemente da assistente virtual, são registradas funções para execução após disparo de eventos relacionados aos mecanismos de texto para fala, fala para texto e a gestos. Os eventos são mostrados na Figura 41.

Para auxílio no reconhecimento e no processamento dos comandos da assistente virtual precisamos de dois mecanismos, um que efetua perguntas ao usuário e outro que possa fazer confirmações de determinadas ações. O diagrama de atividades representado pela Figura 42 mostra como estes mecanismos podem ser implementados.

Os comandos disponíveis por meio da assistente virtual são:

1. Semáforo: Verifica a existência de um semáforo nas proximidades e informa se é seguro ou não atravessar a via. A consulta com o semáforo é detalhada na seção 3.3.6. Comando: semáforo.

2. Proximidade: Verifica locais e estabelecimentos relevantes na proximidade do usuário. Também é possível encontrar tipos de locais específicos como restaurantes e bancos. A Figura 43 mostra como implementar os dois tipos de consultas de proximidade.

   Exemplos de comandos:

   - Usuário: Proximidade.

   - Cida: Local encontrado: Biblioteca Central Santa Mônica UFU.

   - Cida: Local encontrado: Agência Banco do Brasil.

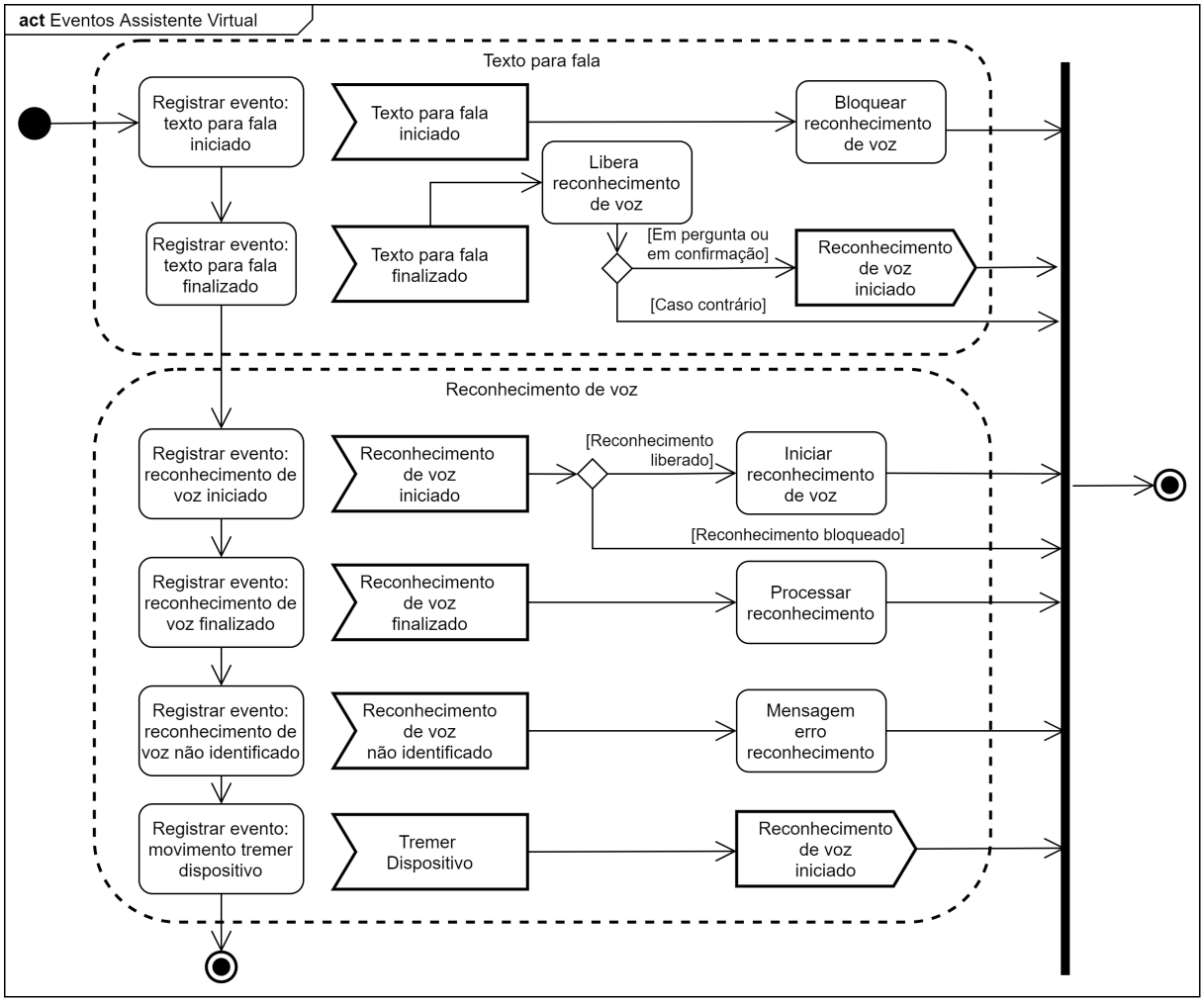

Figura 41 – Diagrama de atividades com o registro de todos os eventos necessários para funcionamento da assistente virtual.

- Cida: Local encontrado: Bloco 3Q UFU.

- Usuário: Restaurantes próximos.

- Cida: Local encontrado: Tenda do Café Restaurante.

- Cida: Local encontrado: Espetinho do Brizolla.

- Cida: Local encontrado: Recanto Universitário.

3. Localização: Informa o endereço atual do usuário. Na Figura 43 temos o diagrama de atividade para implementação do comando.

   Exemplo de comando:

   - Usuário: Minha localização.

   - Cida: Você está próximo a 3C - UFU, Campus, Av. João Naves de Ávila - Santa Mônica, Uberlândia - MG, 38405, Brasil.

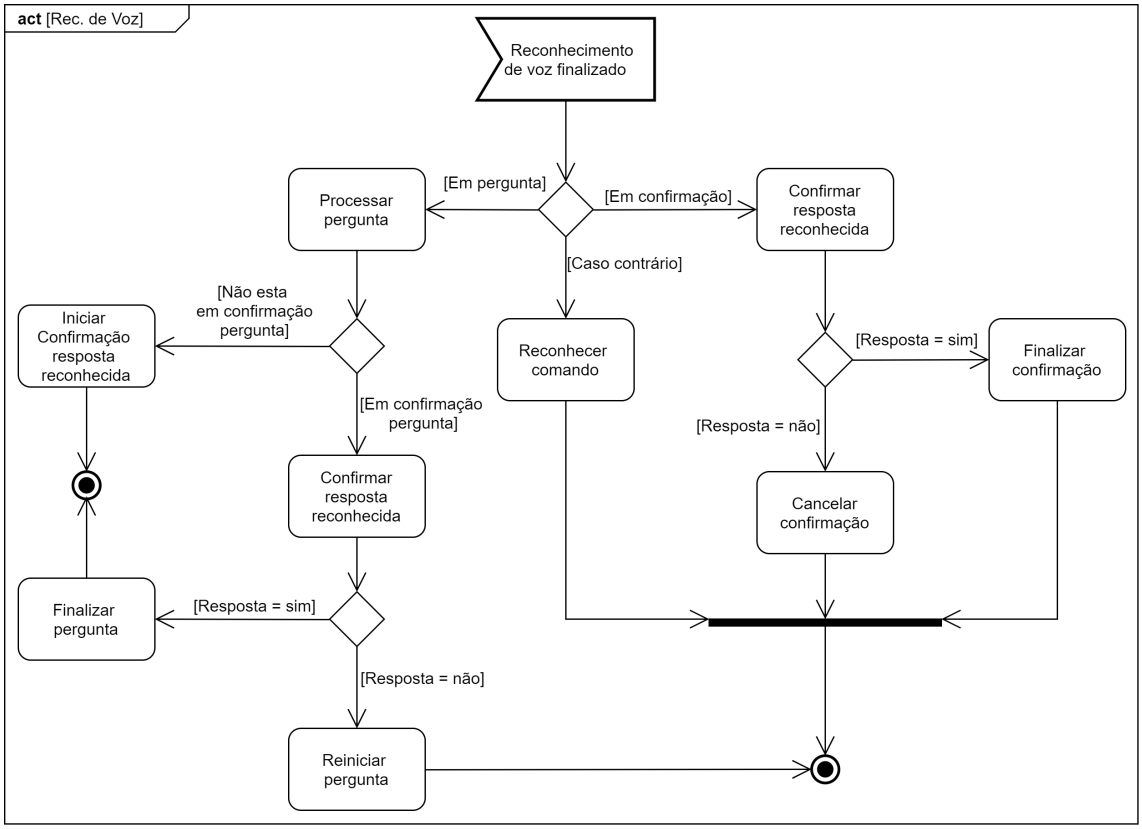

Figura 42 – Diagrama de atividades com o fluxo de controle e reconhecimento de voz da assistente virtual, mostrando o processamento de perguntas e reconhecimento de comandos.

4. Salvar localização: Salva a localização atual do usuário e exige por meio de uma pergunta que informe um nome para identificar a localização.

   Exemplo de comandos:

   - Usuário: Salvar minha localização.

   - Cida: Informe um nome para localização.

   - Usuário: Casa.

   - Cida: Confirma o nome casa?

   - Usuário: Sim.

   - Cida: Local salvo com sucesso.

5. Remover localização: Remove uma localização salva anteriormente pelo usuário.

   Exemplo de comandos:

   - Usuário: Remover localização.

   - Cida: Qual o nome do local que deseja remover?

   - Usuário: Casa.

- Cida: Confirma o nome casa?

- Usuário: Sim.

- Cida: Local removido com sucesso.

6. Iniciar rota: Inicia a rota até o local desejado. A busca do local é feita em diferentes bases de dados. A seção 3.3.5 contém informações detalhadas sobre o serviço de rota.

   Exemplo de comandos:

   - Usuário: Ir para casa.

   - Cida: Confirma o local pesquisado: casa?

   - Usuário: Sim.

   - Cida: Rota iniciada.

7. Configurações rota: Permite o usuário alterar a configuração de período em que recebe as informações de direção caso exista uma rota ativa.

   Exemplo de comandos:

   - Usuário: Configuração atualização rota.

   - Cida: Qual o tempo de atualização em segundos deseja receber as informações?

   - Usuário: 5.

   - Cida: Confirma o tempo em segundos igual a 5.

   - Usuário: Sim.

   - Cida: A configuração foi alterada, o aplicativo precisa ser reiniciado para que a atualização tenha efeito.

3.3.4 Localização

O serviço de localização se trata de um conjunto de funções que são utilizadas pela assistente virtual para consulta de informações nos arredores. É possível consultar o endereço atual e locais nas proximidades, como restaurantes, bibliotecas etc.

Para consultar o endereço atual, podemos utilizar a API do Google Maps. Já a busca pelos locais é feita primeiramente na API Web, onde busca os locais previamente cadastrados no banco de dados. Como busca complementar, caso nenhum local seja encontrado na API Web, uma nova busca é feita na API do Google Maps para que as informações sejam buscadas.

Os comandos que desencadeiam a chamada das funções de localização são mostrados na Figura 43.

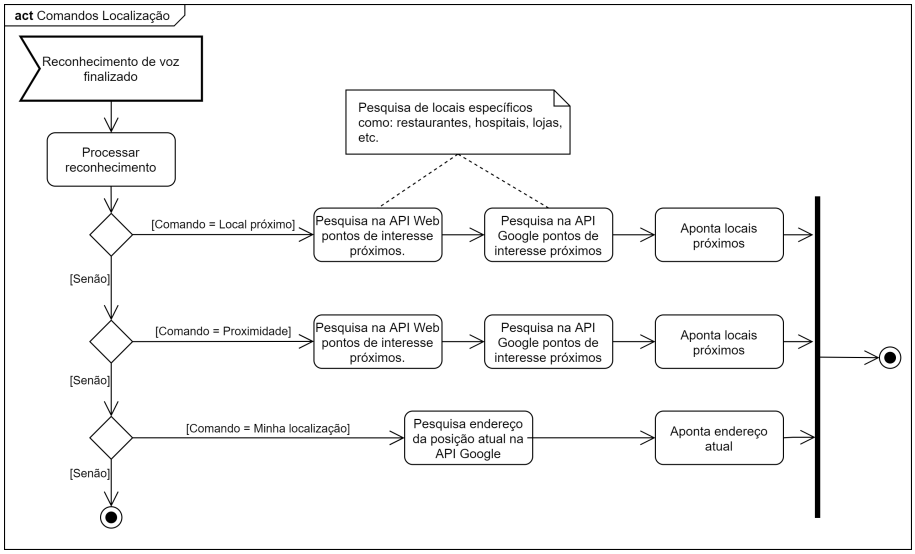

Figura 43 – Diagrama de atividades com o processamento de comandos relacionados a consulta de localizações nos arredores.

3.3.5 Rota

O serviço de rota é responsável por controlar todas as interações relacionadas a busca e navegação para locais escolhidas pelo usuário, para que ele possa se locomover. A assistente virtual se comunica diretamente com o serviço de rota para iniciar e consultar informações da rota. Os comandos disponíveis são mostrados na Figura 44

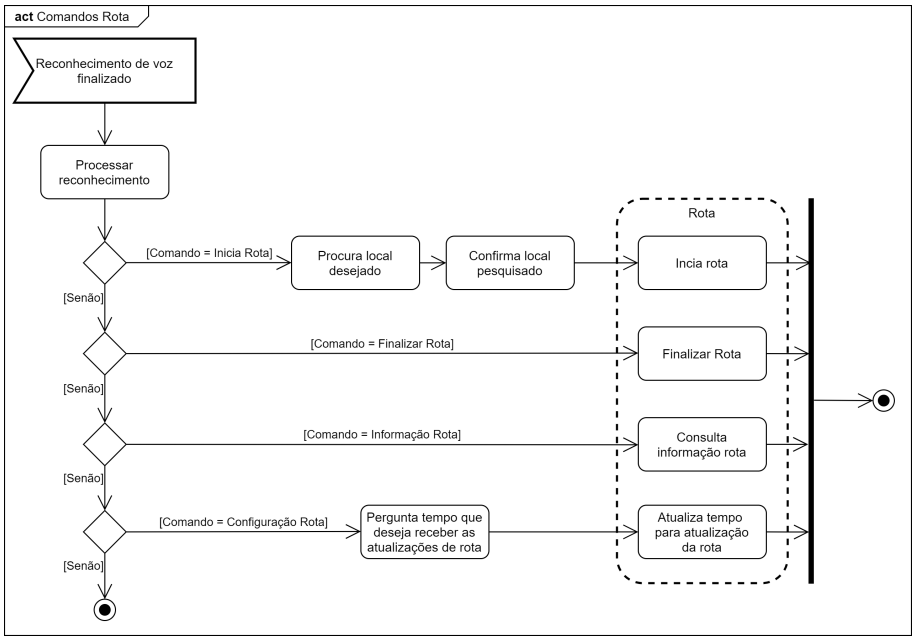

Figura 44 – Diagrama de atividade com o reconhecimento dos comandos relacionados a rota, como inicialização, finalização, informações e configuração da rota.

Na inicialização do serviço de rota é registrado um temporizador que define o tempo em que as informações de rota serão ditas ao usuário. Esse temporizador pode ser

alterado pelo usuário por meio de um comando da assistente virtual, que foi apresentado na seção 3.3.3. O diagrama de atividade da Figura 45 mostra a inicialização do serviço.

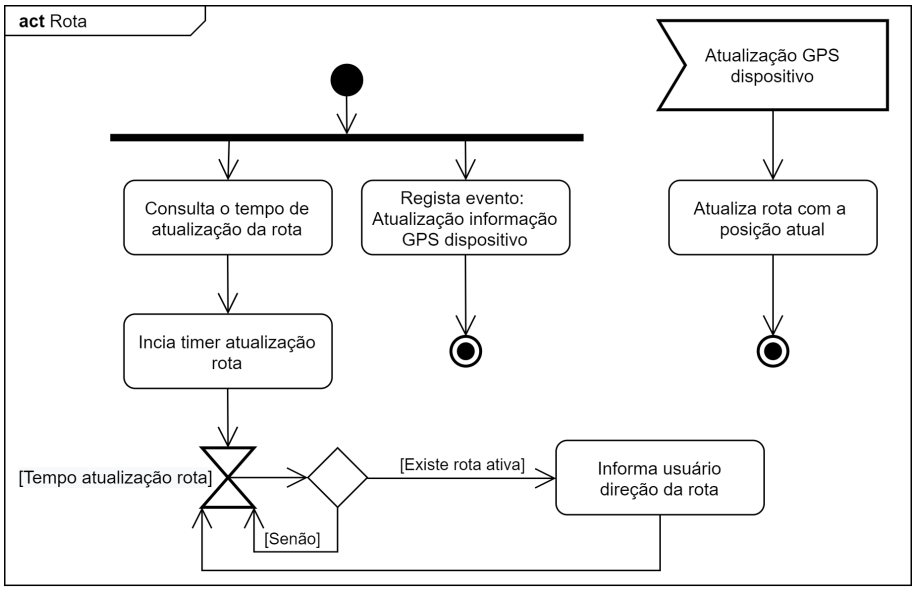

Figura 45 – Diagrama de atividades com a inicialização do serviço de rota indicando o registro do evento de atualização do GPS e a inicialização do temporizador para atualização de direção da rota.

Para iniciar a rota, o usuário pode escolher tanto um local salvo previamente, como um local a ser pesquisado. Com o comando de início feito por meio da assistente virtual, é feita uma varredura na base de dados do aplicativo. Caso o local não tenha sido previamente salvo pelo usuário, é feita uma nova busca de localidade na API Web. Em último caso é feita uma busca por meio da API do Google Maps para identificar o local desejado, como visto na Figura 46.

Com o local identificado e confirmado pelo usuário, a rota é calculada por meio da API do MapBox, que retorna um GeoJson com uma lista de pares de coordenadas geográficas para chegar ao local requisitado. Essa lista é utilizada em conjunto com a atual localização do usuário para calcular e informar qual a direção deverá ser seguida para chegar ao destino. Nesse momento, a rota é iniciada e a assistente virtual não aceita mais comandos até que a rota seja encerrada pelo usuário ou que ela seja finalizada.

Para informar a direção a qual o usuário deve seguir, o serviço de rota leva em consideração a posição atual do usuário, bem como sua direção. Dada sua posição atual, é necessário saber em qual direção se deve seguir para alcançar o próximo ponto da rota. Para calcular a direção, ou o rumo em que se deve seguir, podemos usar a fórmula a seguir, mostrada por Veness (2021), que resulta em um angulo em relação ao norte, a qual deve ser seguido para chegar ao ponto desejado. Considerando o ponto (A_1, A_2) como posição atual e o ponto (B_1,B_2) como ponto que queremos alcançar, onde as coordenadas são

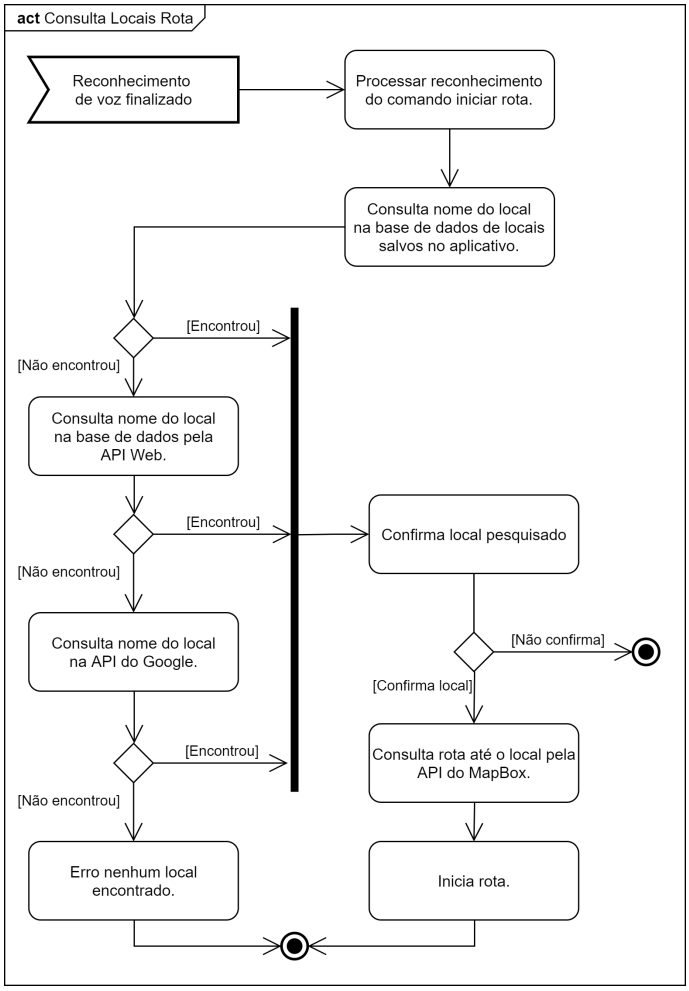

Figura 46 – Diagrama de atividades com o processo de consulta de localização para iniciar uma rota.

expressas em radianos a fórmula se dá:

$$Y = sen(B_2 - A_2) * cos(B_1)$$
$$X = cos(A_1) * sen(B_1) - sen(A_1) * cos(B_1) * cos(B_2 - A_2)$$
$$\theta = atan2(Y,X)$$

Com isso chegamos a θ, que é a direção dada em radianos. Precisamos converter esse resultado para um ângulo da bússola, que tem um intervalo em graus de 0 a 360. O primeiro passo é converter de radiano para graus. Nesse caso, basta multiplicar o resultado por 180/PI. Com isso temos o resultado em graus no intervalo de -180 a 180 decorrente da função atan2. Porém, para traduzir este resultado para uma direção da bússola é necessário converter esse resultado para um intervalo de 0 até 360, que pode ser feito por meio de uma normalização, como apresentado a seguir:

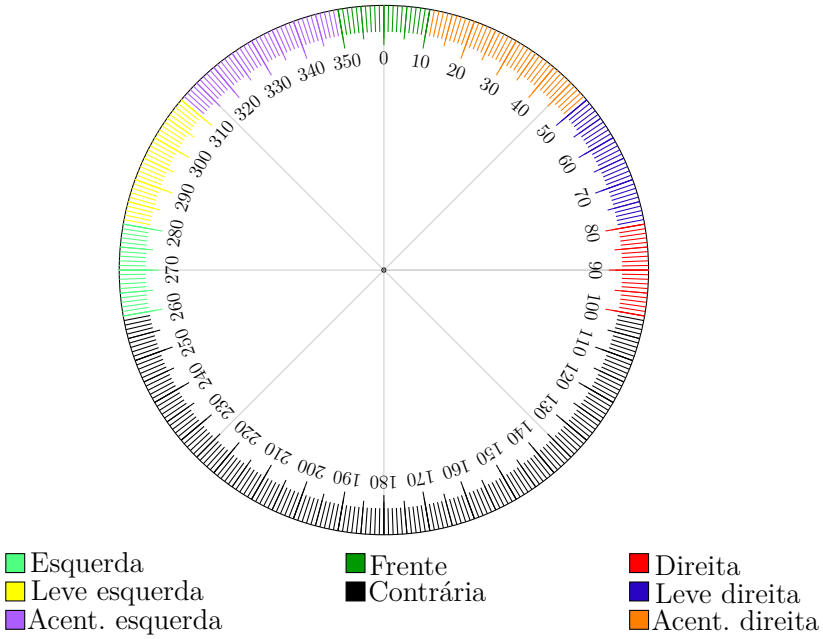

Figura 47 – Tradução do ângulo da direção em um texto que pode ser utilizado para direcionamento de um usuário.

$$Rumo = mod((\theta \times (180/\pi) + 360), 360)$$

Com o angulo calculado, é feito uma tradução do angulo em graus para uma resposta mais amigável ao usuário, com intervalos pré-definidos que foram mostrados na Figura 47

Para a navegação, além do rumo em que o usuário deverá seguir, devemos verificar também qual ponto da rota o usuário está. Para isso, é verificada a distância entre o usuário e todos os pontos não alcançados da rota. Caso ele esteja a menos de dois metros de um ponto, é considerado que ele chegou até esse marco, e então ele pode ser guiado na direção do próximo ponto da rota. Para o cálculo de distância, foi utilizado a fórmula de Haversine mostrada por Veness (2021), como descrita a seguir.

Considerando o ponto (A_1, A_2) como posição atual e o ponto (B_1,B_2) como um ponto qualquer dentro da rota, onde as coordenadas estão em graus a distância entre os pontos é:

$$r = 6371000$$
$$lat_A = A_1 \times (\pi/180)$$
$$lat_B = B_1 \times (\pi/180)$$
$$\Delta_{Lat} = (A_2 - A_1) \times (\pi/180)$$
$$\Delta_{Lng} = (B_2 - B_1) \times (\pi/180)$$
$$a = sen(\Delta_{Lat}/2) \times sen(\Delta_{Lat}/2) + sen(\Delta_{Lng}/2) \times sen(\Delta_{Lng}/2) \times cos(lat_A) \times cos(lat_B)$$
$$c = 2 \times atan2(\sqrt{a}, \sqrt{1-a})$$
$$d = r \times c$$

Quando o usuário chega ao último ponto da rota, ela é considerada como finalizada, e então os demais comandos da assistente virtual são liberados.

No diagrama de atividades é ilustrado o processo de atualização da rota, como visto na Figura 48. Com o evento de atualização do GPS disparado, é necessário percorrer todos os pontos disponíveis da rota e calcular a distância do usuário entre cada ponto. Assim, é possível verificar se o usuário atingiu determinado ponto. Com isso podemos calcular o rumo em que o usuário deve seguir para alcançar o próximo ponto da rota e então traduzir estes dados de forma que fique mais claro ao usuário.

3.3.6 Interação semáforo

A interação entre o aplicativo e o semáforo é muito importante quando o usuário está se locomovendo com o serviço de rota. Caso um semáforo seja encontrado nas proximidades, além de informar sobre qual direção deve ser seguida, o serviço de rotas informa se existe algum semáforo nas proximidades e se é seguro efetuar a travessia. A comunicação entre o semáforo e o aplicativo é feita de duas formas, utilizando tanto a API Web quanto o protocolo MQTT. Essa comunicação é representada pela Figura 49.

Como prioridade sempre será verificado se existe um semáforo na rede local por meio de um broker MQTT. A conexão local pode ser vantajosa pois não exige que o usuário tenha uma conexão de dados móveis e ainda permite tempos de resposta mais rápido na atualização de estados do semáforo. Este cenário pode ser interessante em um ambiente universitário que oferece conexão via Wi-Fi para grande parte do campus. Caso nenhum dispositivo MQTT seja encontrado, ou não é possível conectar à rede Wi-Fi a busca de estados acontece por meio da API Web.

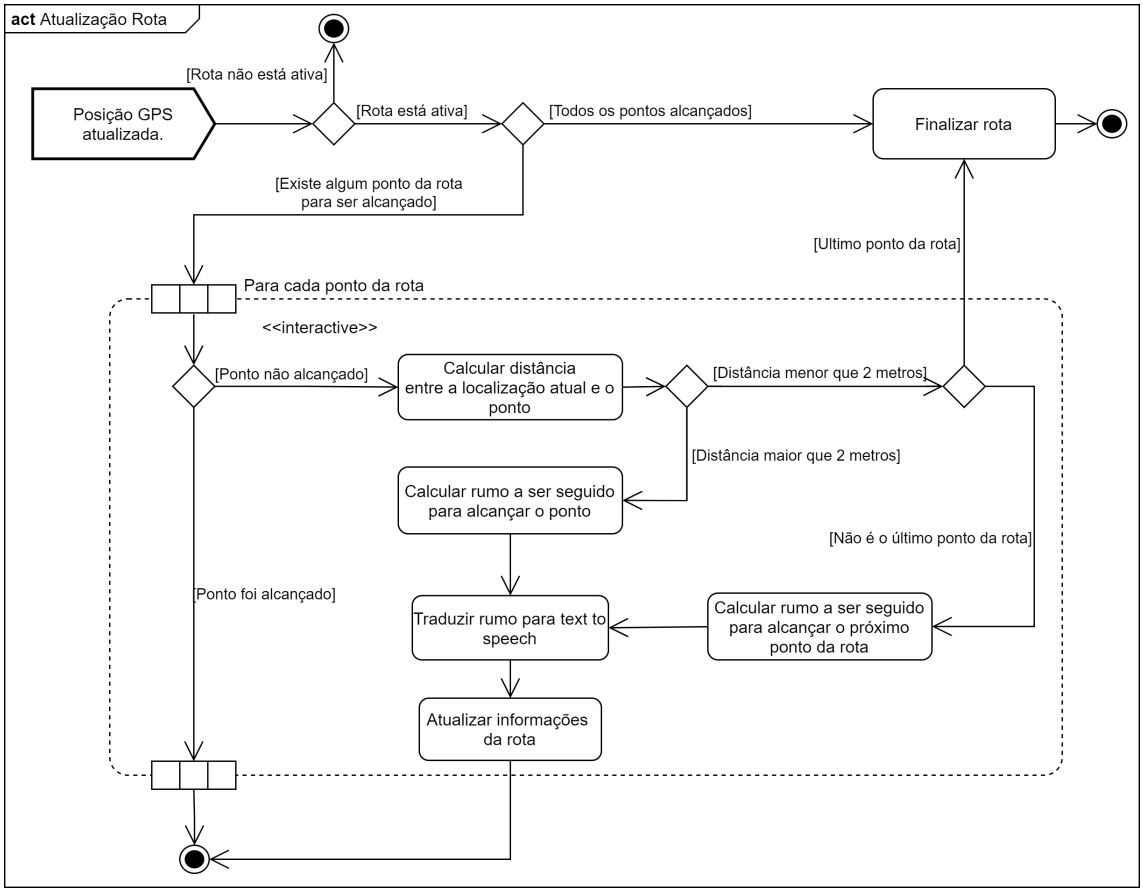

Figura 48 – Diagrama de atividades com o fluxo de atualização da rota, mostrando o processo desde o evento de atualização de GPS até a tradução da direção a ser seguida.

3.3.6.1 MQTT

A conexão com o semáforo por meio do MQTT consiste em duas partes: a procura por um broker MQTT na rede e a atualização do estado do semáforo após se conectar ao broker MQTT.

Primeiramente a busca por dispositivos na rede local que oferecem comunicação via MQTT pode ser feita por meio do protocolo mDNS. Caso algum dispositivo seja encontrado, o aplicativo deve se conectar ao broker MQTT e se inscrever no tópico local para receber as atualizações de estado. O diagrama que detalha a busca e a conexão estão representados na Figura 50.

Maiores detalhes referentes a configuração e a publicação de informações do semáforo no broker MQTT foram mostrados na seção 3.1.1

3.3.6.2 API

Caso nenhum dispositivo MQTT seja encontrado na rede local, o aplicativo deve consultar o semáforo mais próximo pela API Web repassando sua localização atual. A API

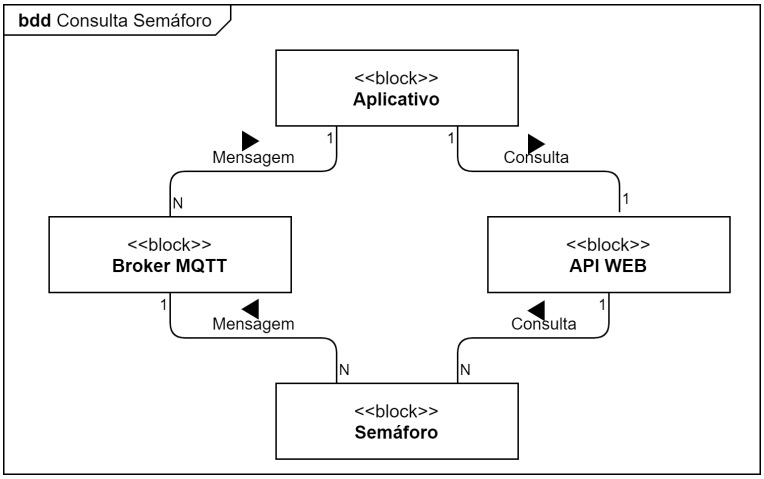

Figura 49 – Diagrama de blocos com a representação da comunicação do aplicativo com o semáforo, indicando os dois caminhos possíveis, pelo broker MQTT e pela API Web.

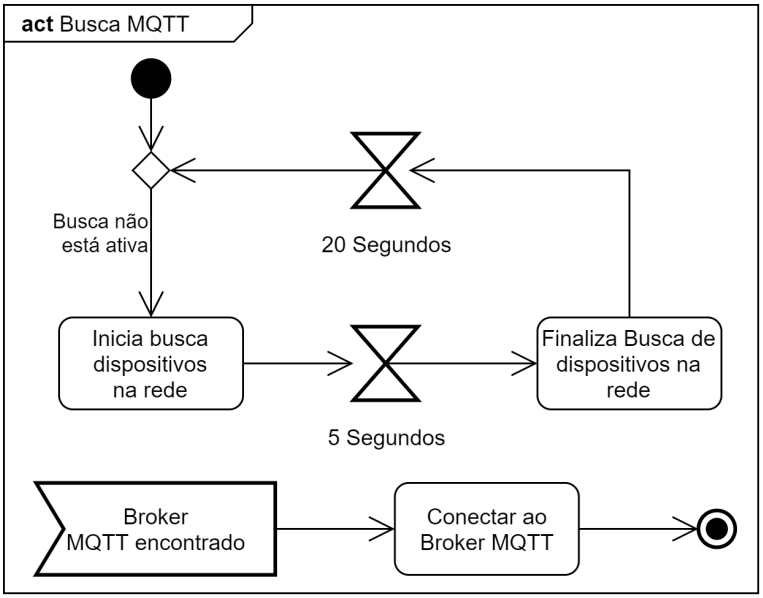

Figura 50 – Diagrama de atividades com a busca de dispositivos MQTT em uma rede local por meio do mDNS.

fica responsável por verificar se existe um semáforo próximo ao usuário, e caso existir, retorna suas informações, que são guardadas pelo aplicativo. Essas informações serão utilizadas pela assistente virtual por meio do serviço de rota. Essa consulta é repetida a cada 3 segundos, para atualizar o estado do semáforo. O diagrama de atividades da Figura 51 contem informações relevantes sobre este processo. Maiores detalhes sobre a construção da API Web foram mostrados na seção 3.2.4.

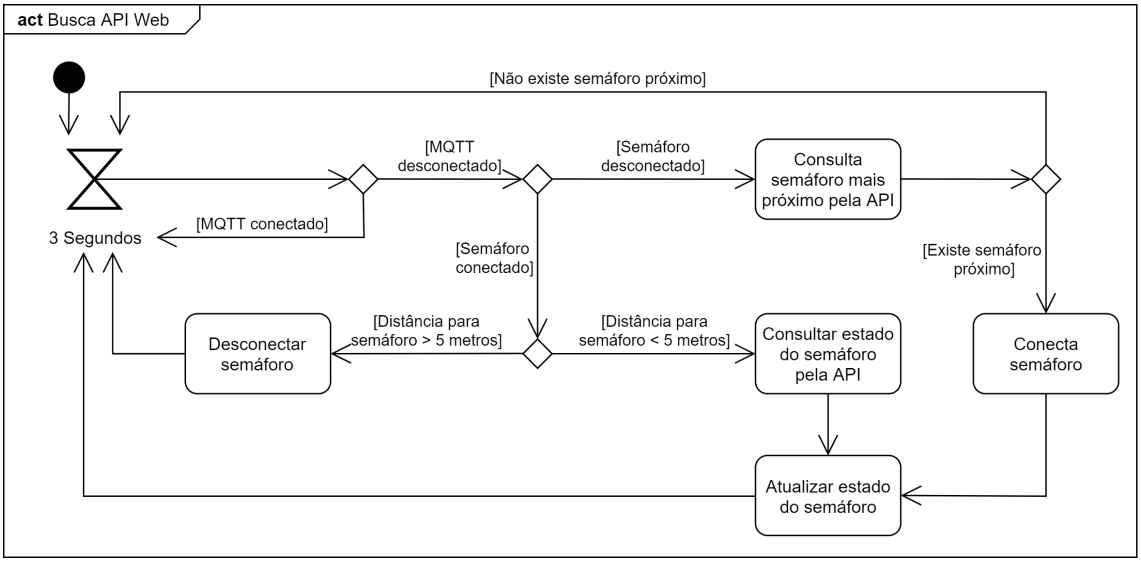

Figura 51 – Diagrama de atividades com o fluxo de conexão pela API Web, quando não existe um broker MQTT na rede local.

3.3.7 Informações usuário

O serviço de informações do usuário é responsável por armazenar e utilizar dados no armazenamento interno do celular, com objetivo de auxiliar o usuário em sua navegação.

Ao inicializar pela primeira vez, a assistente pede ao usuário seu nome, e em seguida cria o conjunto de dados do usuário juntamente com uma lista de locais vazia. A inicialização foi mostrada no diagrama de atividade da Figura 52.

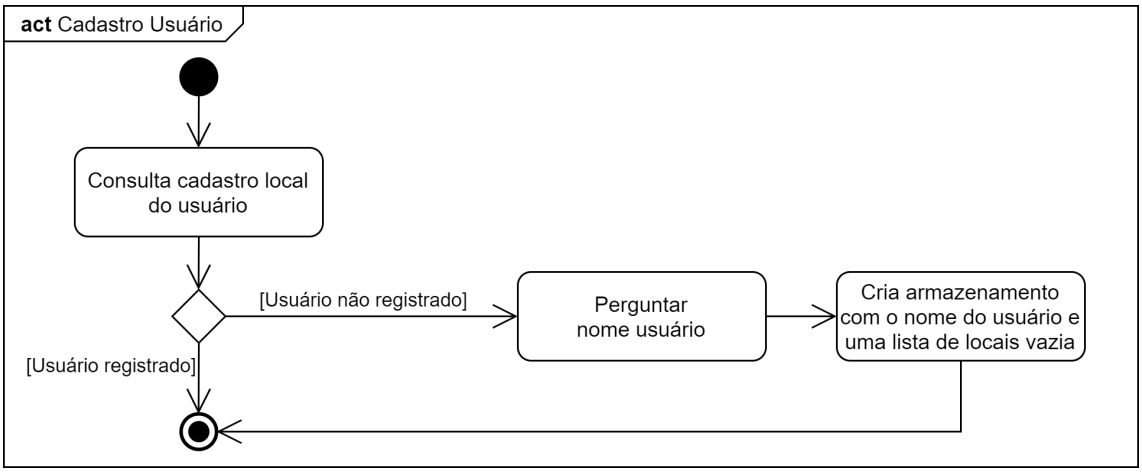

Figura 52 – Diagrama de atividades com o fluxo do processo de cadastro de usuário.

Após o cadastro inicial, fica liberado por meio da assistente virtual os comandos de salvamento e remoção de locais gerenciados pelo usuário para auxílio na navegação. A inserção e remoção de locais está apresentada pelo diagrama de atividade na Figura 53.

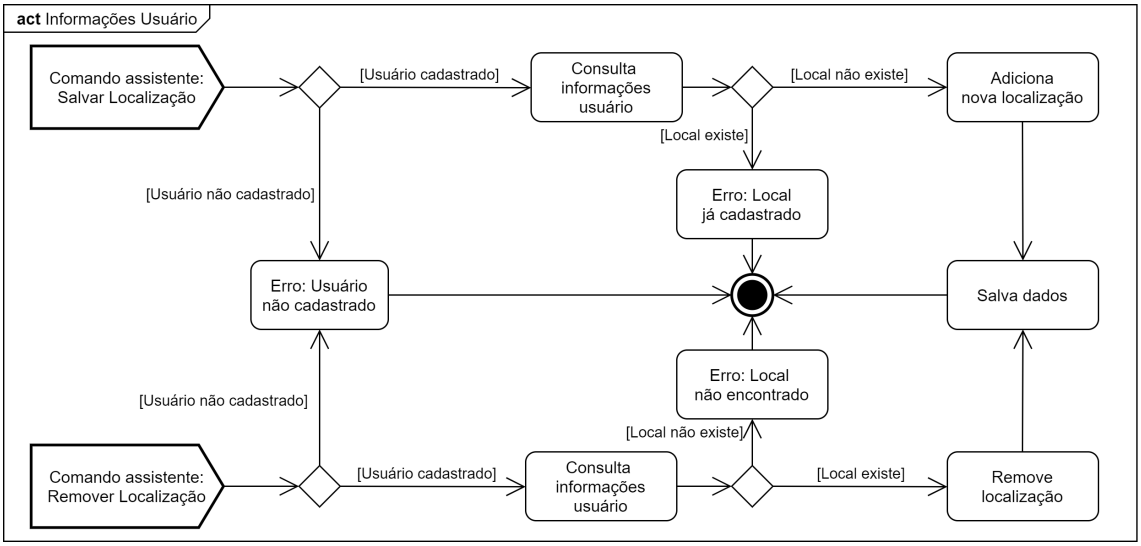

Figura 53 – Diagrama de atividades com a inserção e remoção de locais na base de dados local do aplicativo.

3.4 Protótipo

A seguir serão apresentados os dispositivos e tecnologias utilizadas para a montagem do protótipo da arquitetura descrita anteriormente. Todos os códigos fonte foram disponibilizados no GitLab[1].

Para implementação do circuito do semáforo a linguagem escolhida foi a linguagem Lua, devido a ser a linguagem oficial suportada pelo kit de desenvolvimento NodeMcu.

Para as aplicações do dispositivo IoT e da nuvem foi utilizada a linguagem Clojure. A escolha foi feita devido a linguagem proporcionar um processo de desenvolvimento rápido devida sua sintaxe concisa e paradigma funcional.

No aplicativo, a escolha foi o React Native, que é um framework utilizado para construção de aplicativos. Além de ser popular, a escolha do React Native foi feita por oferecer compatibilidade para vários sistemas operacionais como o Android e IOS, o que torna mais ágil o processo de desenvolvimento.

3.4.1 Semáforo

Para prototipação do semáforo , um circuito com LEDs foi montado em uma protoboard para simulação. O NodeMcu citado na seção 2.2.3.1 foi utilizado, e os LEDs ativados através de suas portas GPIO, simulando um semáforo. É ilustrado por meio da Figura 54 o projeto do circuito e sua implementação. Na Figura 55 temos o esquema do circuito.

[1] https://gitlab.com/tcc-assistente-virtual

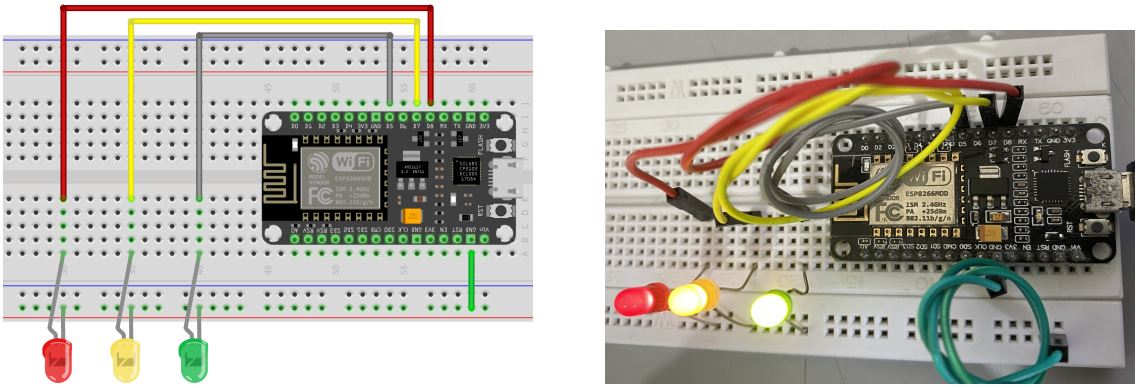

Figura 54 – Circuito do semáforo projetado no Fritzing e circuito real implementado alimentado por uma porta USB.

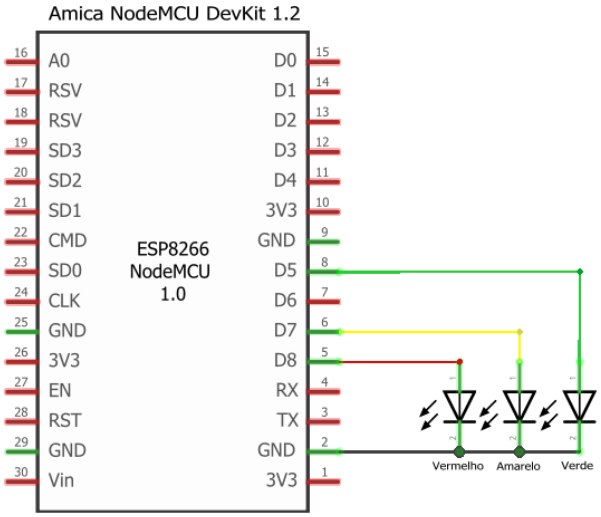

Figura 55 – Esquema do circuito do semáforo com o NodeMcu projetado no Fritzing.

O código escrito para o NodeMcu foi feito na linguagem Lua, foi implementado de acordo com os diagramas apresentados na seção 3.1. e pode ser encontrado no repositório do GitLab. Como o espaço do NodeMcu é limitado, é interessante colocar em seu firmware apenas funções utilizadas. Para este caso devemos ter as seguintes funções WiFi, JSON, MQTT e GPIO. O firmware pode ser montado online por meio do site NodeMcu-Build [2]. Outras opções e informações podem ser encontradas na documentação do NodeMcu [3].

3.4.2 Dispositivo IoT

Para uso neste trabalho, foi utilizado o modelo Raspberry Pi 3, que oferece comunicação via Wi-Fi. Como sistema operacional, foi utilizado a distribuição Ubuntu Core [4], que se trata de uma versão mais simples e leve do Ubuntu projetada para dispositivos com recursos limitados.

[2] https://nodemcu-build.com/
[3] https://nodemcu.readthedocs.io/
[4] https://ubuntu.com/core

Como mostrado na seção 3.1.1, no dispositivo IoT temos um broker MQTT e dois programas escritos em Clojure. A seguir vamos detalhar como a implementação destes foi feita. O código fonte para todos estes projetos está disponível no GitLab.

O broker MQTT é utilizado pelos dispositivos na rede local, e foi configurado no Raspberry Pi com a ajuda do pacote Mosquitto, com configurações de transporte por TCP e WebSockets ativadas. As informações como download e configurações podem ser encontradas no site do Mosquitto [5]. A conexão TCP é utilizada pelo NodeMcu, e a conexão por websockets é utilizada pelo aplicativo, visto que a biblioteca MQTT utilizada no React Native não suporta a conexão via TCP. O Mosquitto foi escolhido por ser um broker de código aberto, popular e robusto.

A aplicação em Clojure responsável por controlar os estados do semáforo foi feita utilizando a versão 1.10.0 da linguagem. Para implementação do cliente MQTT foi utilizada a biblioteca Machine Head [6], que permite a conexão em dois brokers para envio dos estados do semáforo. A biblioteca foi escolhida pois conta com funções mais simples quando comparada a biblioteca padrão Paho MQTT.

A aplicação Clojure que envia as mensagens do protocolo mDNS também utilizou a versão 1.10.0 da linguagem. Para implementação do protocolo mDNS foi utilizada a biblioteca JmDNS [7], uma vez que provê uma interface simples para implementação do protocolo.

3.4.3 Broker Global

Para o broker global foi utilizado o CloudMQTT [8], que é um serviço simples e rápido que disponibiliza brokers com Mosquitto hospedados na nuvem. O CloudMQTT contava com um plano gratuito com intuito de ser utilizado para pequenos protótipos e testes, porém atualmente o plano não se encontra disponível, portanto o serviço não é mais gratuito. Como alternativa temos o HiveMQ [9] que disponibiliza um broker público ao se registrar no site.

3.4.4 Banco de Dados

No protótipo foi utilizado o banco de dados relacional PostgreSQL na versão 12 e o banco foi escolhido devido a sua gratuidade, seu código aberto, robusto e performático. No GitLab está disponível o código SQL para criação das tabelas como especificado na seção 3.2.1, além disso também está disponível o SQL para inserção de dados para teste.

[5] https://mosquitto.org/
[6] https://github.com/clojurewerkz/machine_head
[7] https://github.com/jmdns/jmdns
[8] https://www.cloudmqtt.com/
[9] https://www.hivemq.com/mqtt-cloud-broker/

3.4.5 Controlador Geral

Serviço construído de acordo com a especificação na seção 3.2.3 na versão 1.10.0 do Clojure. Assim como na aplicação do dispositivo IoT, aqui também foi utilizada a biblioteca Machine Head para conexão MQTT com o broker global. Já para conexão e consultas ao banco de dados, foi utilizada a biblioteca HugSQL [10], uma vez que permite uma separação organizada entre o código Clojure e o SQL. O código fonte da aplicação está disponível no GitLab.

3.4.6 API Web

A API foi construída na versão 1.10.0 do Clojure seguindo a arquitetura proposta na seção 3.2.4. Para comunicação com o banco de dados e com o Controlador Geral foi utilizada a biblioteca HugSQL. Para atender as requisições HTTP foi utilizada a biblioteca Ring [11] com o serviço web Jetty. Essa biblioteca foi escolhida pois permite que os métodos da API sejam construídos de forma modular e simples. Para configuração das rotas foi utilizada a biblioteca Compojure API [12] que é uma extensão da biblioteca Ring, que permite uma configuração mais simples da API e que possui recursos de geração de documentação de forma automática. O código da API está disponível no GitLab.

3.4.7 Aplicativo

O aplicativo foi construído em React Native com a linguagem JavaScript para possibilitar que o mesmo código seja reutilizado tanto para sistemas operacionais baseados em Android quanto para iOS. A versão utilizada foi a 0.61.5. A construção do aplicativo seguiu as especificações da seção 3.3 e o código fonte está disponibilizado no Gitlab.

Para implementação dos recursos de texto para fala e fala para texto foram utilizados dois componentes que são "react-native-community/voice"[13] e "react-native-tts"[14]. A ativação da assistente virtual foi feita com ajuda do componente "react-native-shake"[15], que permite capturar o evento de tremer o dispositivo.

A busca de dispositivos na rede com o protocolo mDNS foi feita por meio do componente "react-native-zeroconf" [16]. O componente disponibiliza uma função que permite fazer uma varredura da rede local, procurando por endereços que disponibilizam um serviço especificado, neste caso o MQTT.

[10] https://www.hugsql.org/
[11] https://github.com/ring-clojure/ring
[12] https://github.com/metosin/compojure-api
[13] https://github.com/react-native-voice/voice
[14] https://github.com/ak1394/react-native-tts
[15] https://github.com/Doko-Demo-Doa/react-native-shake
[16] https://github.com/balthazar/react-native-zeroconf

Já a conexão pelo protocolo MQTT foi feita com a ajuda da biblioteca Paho MQTT [17]. Neste caso a conexão é disponibilizada apenas por meio de websockets, tornando obrigatório a disponibilidade para websockets no broker do Raspberry Pi.

Para controlar os dados salvos localmente no aplicativo, foi utilizado o componente "react-native-async-storage" [18], que utiliza o SQLite no Android e um dicionário de dados para o iOS.

A consulta de posicionamento de coordenadas de GPS foi realizada pelo componente "react-native-geolocation-service" [19]. O uso desta, permite que a API "FusedLocationProviderClient"do Google Play seja utilizada, tal API permite uma maior precisão nos dados de GPS, que são extraídos de uma junção de vários provedores como satélites de GPS, redes de telefonia e redes de internet.

Para o cálculo de caminho a seguir pelo serviço de rotas, o componente "react-native-sensors" [20] foi utilizado. Este permite a extração dos dados de magnetômetro do celular, nos permitindo ter a todo momento a informação de direção da bússola.

3.4.8 Anexos Aplicativo

A seguir serão apresentadas chamadas a APIs externas que foram utilizadas para construção do aplicativo.

3.4.8.1 Google Maps API

A API do Google Maps disponibiliza dados referentes a estabelecimentos, geolocalização ou pontos de interesse. Para utilização é necessário efetuar o cadastro para obter uma chave de acesso à API. Para utilização dos serviços, existem limites de quantidades de requisições que podem ser feitas mensalmente. No aplicativo foram utilizados 3 endpoints que serão detalhados a seguir.

3.4.8.1.1 Pontos de Interesse

- *endpoint*: GET https://maps.googleapis.com/maps/api/place/nearbysearch/json

A busca de pontos de interesse foi utilizada na busca de locais próximos, com os seguintes parâmetros:

- "key=token": Chave ou token disponibilizado ao efetuar o cadastro.

- "radius=10": Raio de busca dos locais, em metros.

[17] https://github.com/eclipse/paho.mqtt.javascript
[18] https://github.com/react-native-async-storage/async-storage
[19] https://github.com/Agontuk/react-native-geolocation-service
[20] https://github.com/react-native-sensors/react-native-sensors

- “type=point_of_interest”: Indica que a busca será feita para encontrar pontos de interesse.

- “location=latitude,longitude”: Atual localização do usuário.

- “keyword=restaurante”: Parâmetro opcional, utilizado na busca de locais específicos, como restarantes, bibliotecas, academias, etc.

3.4.8.1.2 Endereço

- *endpoint*: GET https://maps.googleapis.com/maps/api/geocode/json

Consulta utilizada para informar o atual endereço do usuário. Parâmetros utilizados:

- “key=token”: Chave ou token disponibilizado ao efetuar o cadastro.

- “latlgn=latitude,longitude”: Atual localização do usuário.

3.4.8.1.3 Busca localização

- *endpoint*: GET https://maps.googleapis.com/maps/api/place/findplacefromtext/json

Busca o endereço de um local ou ponto de interesse. Ao buscar uma localização para iniciar a rota, o aplicativo utiliza a consulta do Google como última opção, caso o local não esteja cadastrado nem na base de dados local do aplicativo ou na base de dados da API REST.

Para consulta, foram utilizados os parâmetros:

- “key=token”: Chave ou token disponibilizado ao efetuar o cadastro.

- “inputtype=textquery”: Indica que a busca de local será feita por meio de um texto.

- “fields=formatted_address,geometry,name”: Campos a serem retornados pela API, nesse caso o endereço, as coordenadas geográficas e o nome do local.

- “input=texto”: Texto informativo do local a ser pesquisado.

3.4.8.2 MapBox API

O MapBox (2021) é um serviço que oferece diversas soluções de mapas e de geolocalização com SDKs e APIs. Esses serviços são pagos para utilização em grande escala. Porém, é possível criar uma conta gratuita no site e utilizar a maioria dos serviços. A conta gratuita dá acesso a uma chave, ou token que pode ser utilizada na API.

O serviço utiliza dados do OpenStreetMap (2021), que é um projeto de dados abertos com informações geográficas mantido pela comunidade. Nele estão disponíveis dados sobre ruas, estradas, trilhas e muito mais, sobre todo o mundo.

A consulta de rota é feita pela API do Mapbox. O endpoint utilizado para a consulta foi: https://api.mapbox.com/directions/v5/mapbox/walking/lat,lng;destLat,destLng

No ínicio do link temos a base da URL do Mapbox: "api.mapbox.com/directions/v5/", seguida pelo endpoint definido por "mapbox/walking", o que diz à API que esperamos o retorno de uma rota que seja feita por um pedestre. Em seguida temos as informações de latitude e longitude, onde o primeiro par de coordenadas se refere ao ponto inicial da rota, e o segundo par define o ponto de destino da rota. Em seguida temos os parâmetros, que são:

- "geometry": Informa o tipo de retorno desejado, nesse caso as coordenadas descritas com GeoJSON.

- "steps": Indica se deve retornar um texto descritivo de locomoção para cada ponto da rota. Para o nosso caso a descrição foi feita no aplicativo, então o parâmetro é passado como falso.

- "access_token": Token de acesso disponível após cadastrar no site do Mapbox.

3.4.9 Demonstração Protótipo

Foram disponibilizados no YouTube[21] vídeos mostrando as principais funcionalidades do aplicativo com a assistente virtual Cida. Além disso, foi mostrado também o circuito de simulação de semáforos com os LEDs em uma protoboard.

3.4.10 Considerações finais

Para implementação do protótipo, temos como um ponto de vantagem a utilização da linguagem Clojure para a maioria das aplicações desenvolvidas, tonando a implementação mais simples, organizada e uniforme.

Além disso, o framework React Native nos permite uma maior simplicidade no desenvolvimento, pois conta com um ecossistema de diferentes bibliotecas que simplificam o uso da tecnologia mobile. Porém, como melhoria podemos citar a refatoração do código do aplicativo para ClojureScript, que tem uma sintaxe muito semelhante ao Clojure, mas que permite que a linguagem seja compilada para o JavaScript. Isso permite a sua utilização junto ao React Native e, consequentemente a unificação do protótipo como citado anteriormente na vantagem da utilização do Clojure.

[21] www.youtube.com/watch?v=oEop06oONEM e www.youtube.com/watch?v=B72DURcTZxw

Como ponto de dificuldade na implementação do protótipo, tivemos a extração de dados posicionais do smartphone, onde os dados de GPS não são atualizados em uma frequência satisfatória para o serviço de rotas.

4 Conclusão

Neste trabalho foi apresentada uma pesquisa geral no capítulo 2 sobre Tecnologias Assistivas e soluções para deficientes visuais. Na pesquisa foi verificado que atualmente os recursos apresentados na forma de aplicativos tem tido um crescimento significativo. Porém, estes recursos têm sofrido uma grande segmentação, onde para cada função deve ser utilizado um aplicativo diferente. Isso complexifica o uso do celular como ferramenta para Tecnologia Assistiva.

Além disso, no capítulo 2 também foi feita uma pesquisa sobre o tema de Internet das Coisas, mostrando e comparando seus principais protocolos e dispositivos utilizados para prototipação. Com isso se observou que o protocolo MQTT é utilizado no meio de IoT por se tratar de um protocolo leve, orientado a mensagens e que disponibiliza mecanismos para controle de qualidade de serviço. Para dispositivos de prototipação com melhor custo-benefício foi encontrado o ESP8266 e para maior desempenho foi encontrado o Raspberry Pi.

Tivemos também a apresentação de uma arquitetura no capítulo 3 para Tecnologia Assistiva com o uso de SysML, com objetivo de auxiliar deficientes visuais em sua locomoção em ambientes externos. Foi construído um protótipo para validar o funcionamento e usabilidade da arquitetura em um cenário real. Com isso foi constatado que é possível implementar uma solução de Tecnologia Assistiva na forma de um aplicativo de forma mais unificada para tornar menos complexa a experiência do usuário.

Como pontos de dificuldade na execução deste trabalho temos o processo de definição da arquitetura e desenvolvimento do projeto, que foi árduo devido a uma quantidade significativa de diferentes tecnologias e protocolos que precisaram ser estudados e implementados para conclusão do trabalho.

4.1 Contribuições

Este trabalho contribuiu com o ambiente acadêmico com a pesquisa dos temas de Tecnologia Assistiva e Internet das Coisas, temas estes que estão em constante crescimento e evolução. A pesquisa fica como ponto de partida para pesquisadores e desenvolvedores da área.

Este autor tem a intenção de que este trabalho possa ser utilizado por pessoas no futuro para evolução da arquitetura proposta, e que o protótipo desenvolvido possa ser uma base para evolução da solução.

4.2 Trabalhos futuros

A seguir são enumerados alguns pontos de melhorias para trabalhos futuros:

1. Atenção aos requisitos de segurança visando a troca de dados segura e confiável pelas diferentes partes do sistema.
2. Permitir que a arquitetura do sistema seja escalável para que possa ser utilizado por cidades inteiras sem afetar o tempo de resposta e confiabilidade.
3. Utilizar modelos de Inteligência Artificial por meio de Ontologia para melhorar a comunicação com a assistente virtual.
4. Retirar dependência da API MapBox, para que as rotas sejam calculadas pelo próprio sistema, utilizando os dados do OpenStreetMap.
5. Melhorar a arquitetura na parte de dispositivos IoT, para que possam ser incluídos diferentes tipos de dispositivos de maneira simples. Como exemplo, seria a criação de um dispositivo que seria colocado em ônibus. Tal dispositivo seria utilizado na navegação para que a assistente virtual possa direcionar o usuário ao ônibus correto, e avisar o mesmo de sua chegada.
6. Criar uma interface web para acesso administrativo onde seria possível efetuar o cadastro de locais e dispositivos IoT.
7. Analisar viabilidade e usabilidade do modelo para pessoas com deficiência visual.
8. Buscar melhorar a extração de dados posicionais no aplicativo, com uma maior frequência de coleta de dados de GPS e ajuste para extrair o norte real do magnetômetro.
9. Colocar semáforo com dois pontos ao invés de um, permitindo que este seja localizado precisamente em qualquer um dos lados da rua que o pedestre se encontra.
10. Permitir que a rota seja recalculada quando o usuário se afastar da mesma.

Referências

ALMEIDA, T. D. S. Modelo de detecção e reconhecimento de semáforos baseado em atenção visual e inteligência artificial. *TCC (Bacharelado em Sistemas de Informação) - Universidade Federal de Sergipe, Itabaiana, SE*, 2015. Disponível em: <https://www.sigaa.ufs.br/sigaa/verProducao?idProducao=1113370&key=9f6139103cbfd4b94f032beed9ea4878>. Acesso em: 16 nov. 2017. Citado 2 vezes nas páginas 15 e 19.

ALMEIDA, T. S.; ARAUJO, F. V. Diferenças experienciais entre pessoas com cegueira congênita e adquirida: Uma breve apreciação. *Revista Interfaces: Saúde, Humanas e Tecnologia*, v. 1, 2013. Disponível em: <https://interfaces.leaosampaio.edu.br/index.php/revista-interfaces/article/view/24/29>. Acesso em: 11 Out. 2021. Citado na página 21.

AMAZON. *Alexa Accessibility*. 2021. Site Amazon. Disponível em: <https://www.amazon.com/AlexaAccessibility>. Acesso em: 18 set. 2021. Citado na página 21.

ARDUINO. *Arduino Website*. 2019. Disponível em: <https://www.arduino.cc/en/Guide/Introduction>. Acesso em: 14 set. 2019. Citado 2 vezes nas páginas 6 e 34.

BANSAL, M.; GARG, S. Internet of things (iot) based assistive devices. *2021 6th International Conference on Inventive Computation Technologies (ICICT)*, 2021. Disponível em: <https://ieeexplore.ieee.org/document/9358662>. Acesso em: 11 out. 2021. Citado na página 23.

BERSCH, R. Introdução a tecnologia assistiva. *Porto Alegre: CEDI*, 2017. Disponível em: <http://www.inf.ufes.br/~zegonc/material/Comp_Sociedade/ZEGONC_Tecnologias_Assistivas_Livro_Introducao_TA.pdf>. Acesso em: 16 set. 2021. Citado na página 16.

BHOWMICK, A.; HAZARIKA, S. M. An insight into assistive technology for the visually impaired and blind people: state-of-the-art and future trends. *Journal on Multimodal User Interfaces, v. 11, n. 2, p. 149-172*, 2016. Disponível em: <https://link.springer.com/article/10.1007%2Fs12193-016-0235-6>. Acesso em: 11 out. 2021. Citado na página 18.

BOUCK, E. C. Assistive technology. *SAGE Publications*, 2017. Citado na página 16.

BOURNE, R. R. A. et al. Trends in prevalence of blindness and distance and near vision impairment over 30 years: an analysis for the global burden of disease study. *The Lancet Global Health. 2021 v. 9. n. 2. p. 130-43*, 2021. Disponível em: <https://www.thelancet.com/journals/langlo/article/PIIS2214-109X(20)30425-3/fulltext>. Acesso em: 15 set. 2021. Citado na página 18.

CAMPANA, A. R. Análise da qualidade e usabilidade dos softwares leitores de tela visando a acessibilidade tecnológica às pessoas com deficiência visual. 2017. Disponível em: <https://www.pessoacomdeficiencia.sp.gov.br/wp-content/uploads/2020/03/TA_SOFTWARES-LEITORES-DE-TELA_ANDERSON-ROG%C3%89RIO.pdf>. Acesso em: 28 set. 2021. Citado na página 21.

CARRASCO, I. G.; SALAZAR, L. R.; RAMIREZ, A. R. G. An iot-based contribution to improve mobility of the visually impaired in smart cities. *Computing, v. 103, n. 6, p. 1233-1254*, 2019. Disponível em: <https://link.springer.com/article/10.1007/s00607-021-00947-5>. Acesso em: 11 out. 2021. Citado 3 vezes nas páginas 15, 20 e 21.

Comitê de Ajudas Técnicas. Ata vii reunião do comitê de ajudas técnicas – cat corde/sedh/pr/. *Paraná*, 2007. Disponível em: <https://www.assistiva.com.br/Ata_VII_Reuni%C3%A3o_do_Comite_de_Ajudas_T%C3%A9cnicas.pdf>. Acesso em: 16 set. 2021. Citado na página 16.

COUGHLAN, J. M.; SHEN, H. Crosswatch: a system for providing guidance to visually impaired travelers at traffic intersections. *Journal of assistive technologies*, 2013. Disponível em: <https://pdfs.semanticscholar.org/f276/fd4a8cd9e6919c5b27189fe818036e2ecc0a.pdf?_ga=2.242796034.1443144484.1512653765-859139634.1509065416>. Acesso em: 16 nov. 2017. Citado 2 vezes nas páginas 15 e 19.

DELLIGATTI, L. Sysml distilled. a brief guide to the systems modeling language. *Addison-Wesley*, 2014. Citado na página 40.

DENTELLA, L. *ESP32 (15) – mDNS*. 2021. Site Luca Dentella. Disponível em: <https://www.lucadentella.it/en/2017/04/29/esp32-15-mdns/>. Acesso em: 18 set. 2021. Citado 2 vezes nas páginas 6 e 32.

DOMINGO, M. C. An overview of the internet of things for people with disabilities. *Journal of Network and Computer Applications, v. 35, n. 2, p. 584-596*, 2011. Disponível em: <http://www.inf.ufes.br/~zegonc/material/Comp_Sociedade/ZEGONC_Tecnologias_Assistivas_Art1_An_overview_of_the_IoT_for_people_with_disabilities.pdf>. Acesso em: 11 out. 2021. Citado na página 23.

ESPRESSIF. *Especificações ESP8266*. 2021. Disponível em: <https://www.espressif.com/en/products/modules/esp8266>. Acesso em: 21 set. 2021. Citado na página 33.

GOMES, R. J. R. Teste de interface de voz. dissertação (mestrado). *Universidade do Porto – Porto*, 2007. Disponível em: <https://repositorio-aberto.up.pt/bitstream/10216/12254/2/Texto%20integral.pdf>. Acesso em: 11 set. 2021. Citado na página 22.

HILLAR, G. C. Mqtt essentials - a lightwheight iot protocol. *Packt Publishing Ltd*, 2017. Citado na página 25.

HOLT, S. P. J. Sysml for systems engineering. a model-based approach. *The Institution of Engineering and Technology, London, United Kingdom*, 2018. Citado 5 vezes nas páginas 36, 37, 38, 39 e 40.

IBGE. *Censo Demográfico 2010*. 2010. Site do IBGE. Disponível em: <https://www.ibge.gov.br/estatisticas-novoportal/sociais/populacao/9662-censo-demografico-2010.html?edicao=9749&t=destaques>. Acesso em: 16 nov. 2017. Citado na página 18.

IHEJIMBA, C.; WENKSTERN, R. Z. Detectsigna: A cloud-based traffic signal notification system for the blind and visually impaired. *2020 IEEE International Smart Cities Conference (ISC2). p. 1-6*, 2020. Disponível em: <https://ieeexplore.ieee.org/document/9239004>. Acesso em: 14 out. 2021. Citado 2 vezes nas páginas 15 e 21.

IoT Analytics. *State of the IoT 2020: 12 billion IoT connections, surpassing non-IoT for the first time*. 2020. Disponível em: <https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/>. Acesso em: 21 set. 2021. Citado na página 22.

JAWS. 2021. Site do JAWS. Disponível em: <https://www.freedomscientific.com/Products/software/JAWS/>. Acesso em: 28 set. 2021. Citado na página 18.

KUROSE, J. F.; ROSS, K. W. *Computer Networking: A Top-Down Approach*. 2017. Pearson. Citado 5 vezes nas páginas 6, 23, 24, 28 e 31.

MAPBOX. *MapBox*. 2021. Site MapBox. Disponível em: <https://docs.mapbox.com/api/navigation/directions/>. Acesso em: 24 jul. 2021. Citado na página 72.

Micro Controllers Lab. *What is MQTT and How MQTT Works?* 2021. Site Micro Controllers Lab. Disponível em: <https://microcontrollerslab.com/what-is-mqtt-and-how-it-works/>. Acesso em: 18 set. 2021. Citado 2 vezes nas páginas 6 e 25.

MICROSOFT. *Microsoft 3D-Soundscape*. 2015. Disponível em: <https://news.microsoft.com/stories/independence-day/>. Acesso em: 16 nov. 2017. Citado 2 vezes nas páginas 6 e 20.

MOHAMED, K. S. The era of internet of things towards a smart world. *Springer, Cham. p. 1-9*, 2019. Citado na página 22.

MULANI, T. T.; PINGLE, S. V. Internet of things. *International Research Journal Of Multidisciplinary Studies And Sppps. v. 2, p. 2–4*, v. 2, p. 2–4, 2016. Disponível em: <http://www.irjms.in/sites/irjms/index.php/files/article/view/270/256>. Acesso em: 16 nov. 2017. Citado na página 22.

NAIK, N. Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. *2017 IEEE international systems engineering symposium (ISSE). p. 1-7)*, 2017. Disponível em: <https://core.ac.uk/download/pdf/160743474.pdf>. Citado 2 vezes nas páginas 32 e 33.

NODEMCU. *NodeMcu Website*. 2019. Disponível em: <www.nodemcu.com>. Acesso em: 8 set. 2019. Citado 3 vezes nas páginas 6, 33 e 34.

OASIS. *OASIS*. 2021. Site OASIS. Disponível em: <https://www.oasis-open.org/>. Acesso em: 4 set. 2021. Citado 3 vezes nas páginas 25, 27 e 30.

OPENSTREETMAP. 2021. Site OpenStreetMap. Disponível em: <https://www.openstreetmap.org/about>. Acesso em: 19 set. 2021. Citado na página 73.

RAMADAN, Z.; FARAH, M. F.; ESSRAWI, L. E. From amazon.com to amazon.love: How alexa is redefining companionship and interdependence for people with special needs. *Psychology & Marketing. n. 38. v. 4. p. 596-609*, 2020. Citado na página 21.

Raspberry Pi. *Raspberry Pi*. 2021. Site Raspberry Pi. Disponível em: <https://www.raspberrypi.org/about/>. Acesso em: 07 set. 2021. Citado 2 vezes nas páginas 6 e 35.

RAYES, A.; SALAM, S. Internet of things from hype to reality the road to digitization. *Springer International Publishing*, 2019. Citado 2 vezes nas páginas 22 e 23.

RFC 6455. *RFC 6455*. 2011. Site IETF. Disponível em: <https://tools.ietf.org/html/rfc6455>. Acesso em: 07 set. 2021. Citado na página 28.

RFC 8446. *RFC 8446*. 2018. Site IETF. Disponível em: <https://tools.ietf.org/html/rfc8446>. Acesso em: 23 set. 2021. Citado na página 29.

RFC6762. *RFC 6762*. 2013. Site IETF. Disponível em: <https://tools.ietf.org/html/rfc6762>. Acesso em: 25 set. 2021. Citado na página 31.

RFC7159. *RFC 7159*. 2014. Site IETF. Disponível em: <https://tools.ietf.org/html/rfc7159>. Acesso em: 07 set. 2021. Citado na página 41.

RFC793. *RFC 793*. 1981. Site IETF. Disponível em: <https://tools.ietf.org/html/rfc793>. Acesso em: 07 set. 2021. Citado na página 27.

RFC7946. *RFC 7946*. 2016. Site IETF. Disponível em: <https://tools.ietf.org/html/rfc7946>. Acesso em: 07 set. 2021. Citado na página 41.

SALVADORI, I. L. Desenvolvimento de Web APIS RESTFUL semânticas baseadas em JSON. *Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico.*, 2015. Disponível em: <https://www.researchgate.net/publication/280558908_Desenvolvimento_de_Web_APIs_RESTful_Semanticas_Baseadas_em_JSON>. Acesso em: 14 set. 2021. Citado na página 42.

SANTOS, A. C. S. dos. Avaliação da experiência do usuário nas interfaces conversacionais(chatterbots) no contexto dos deficientes visuais. *TCC (Bacharelado em Ciência da Computação) - Universidade do Extremo Sul Catarinense, UNESC*, 2019. Disponível em: <http://repositorio.unesc.net/handle/1/8198>. Acesso em: 14 out. 2021. Citado na página 21.

Scaleway. *IoT Hub: What Use Case for WebSockets?* 2021. Site Scaleway. Disponível em: <https://blog.scaleway.com/iot-hub-what-use-case-for-websockets/>. Acesso em: 18 set. 2021. Citado 2 vezes nas páginas 6 e 29.

SENJAM, S. S. Smartphones as assistive technology for visual impairment. *Eye. n. 17. p. 1-3*, 2021. Citado na página 19.

SERPANOS, D.; WOLF, M. Internet-of-things (iot) systems. architectures, algorithms, methodologies. *Springer*, 2018. Citado na página 22.

SHELBY K. HARTKE, C. B. Z. *RFC 7252*. 2014. Site IETF. Disponível em: <https://tools.ietf.org/html/rfc7252>. Acesso em: 03 set. 2019. Citado 3 vezes nas páginas 6, 30 e 31.

SIDDESH et al. Application for assisting mobility for the visually impaired using iot infrastructure. *2016 International Conference on Computing, Communication and Automation (ICCCA). IEEE. p. 1244-1249*, 2016. Disponível em: <http://www.inf.ufes.br/~zegonc/material/Comp_Sociedade/ZEGONC_Tecnologias_Assistivas_Art2_Application%20for%20Assisting%20Mobility%20for%20the%20Visually%20Impaired%20using%20IoT%20Infrastructure.pdf>. Acesso em: 14 out. 2021. Citado 2 vezes nas páginas 15 e 20.

SOHAN, S. M.; ANSLOW, C.; MAURER, F. A case study of web api evolution. *2015 IEEE World Congress on Services. IEEE. p. 245-252*, 2015. Disponível em: <http://anslow.cpsc.ucalgary.ca/papers/services2015-sohan.pdf>. Acesso em: 14 out. 2021. Citado na página 42.

SYSML. *RFC 7252*. 2021. Site SysML. Disponível em: <https://sysml.org/>. Acesso em: 28 jul. 2021. Citado na página 35.

TAPU, R.; MOCANU, B.; ZAHARIA, T. A computer vision system that ensure the autonomous navigation of blind people. *2013 E-Health and Bioengineering Conference (EHB). IEEE. p. 1-4*, 2013. Disponível em: <http://ieeexplore.ieee.org/document/6707267/>. Acesso em: 08 dez. 2017. Citado na página 19.

TAYLOR, P. Text-to-speech synthesis. *Cambridge university press*, 2009. Citado na página 40.

ULTRACANE. 2021. Site do UltraCane. Disponível em: <https://ultracane.com/about_the_ultracane>. Acesso em: 28 set. 2021. Citado na página 18.

VENESS, C. *Movable Type*. 2021. Site Movable Type. Disponível em: <http://www.movable-type.co.uk/scripts/latlong.html>. Acesso em: 14 jul. 2021. Citado 2 vezes nas páginas 60 e 62.

Wikimedia. *MQTT protocol example without QoS*. 2021. Site Wikimedia. Disponível em: <https://commons.wikimedia.org/wiki/File:MQTT_protocol_example_without_QoS.svg>. Acesso em: 18 set. 2021. Citado 2 vezes nas páginas 6 e 26.