UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DIRETORIA DE PESQUISA E PÓS-GRADUAÇÃO
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO DE ESPECIALIZAÇÃO EM INTERNET DAS COISAS

DANIEL NEGREIROS CANGIANELLI

PROTOCOLO DE COMUNICAÇÃO GPS-SERVIDOR UTILIZANDO MQTT

MONOGRAFIA DE ESPECIALIZAÇÃO

CURITIBA
2019

DANIEL NEGREIROS CANGIANELLI

PROTOCOLO DE COMUNICAÇÃO GPS-SERVIDOR UTILIZANDO MQTT

Monografia de Especialização, apresentada ao Curso de Especialização em Internet das Coisas, do Departamento Acadêmico de Eletrônica – DAELN, da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Especialista.

Orientador: Prof. Dr. Guilherme Luiz Moritz

CURITIBA
2019

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Curitiba

Diretoria de Pesquisa e Pós-Graduação
Departamento Acadêmico de Eletrônica
Curso de Especialização em Internet das Coisas

TERMO DE APROVAÇÃO

PROTOCOLO DE COMUNICAÇÃO GPS-SERVIDOR UTILIZANDO MQTT

por

DANIEL NEGREIROS CANGIANELLI

Esta monografia foi apresentada em 22 de Novembro de 2019 como requisito parcial para a obtenção do título de Especialista em Internet das Coisas. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Prof. Dr. Guilherme Luiz Moritz
Orientador

_____
Prof. M. Sc. Danillo Leal Belmonte
Membro titular

_____
Prof. M. Sc. Omero Francisco Bertol
Membro titular

- O Termo de Aprovação assinado encontra-se na Coordenação do Curso –

Dedico este trabalho à Lucia e Gabriela que me apoiam e incentivam em todos os momentos.

AGRADECIMENTOS

A minha mãe, por acreditar e incentivar meus estudos e crescimento. A minha namorada por toda paciência, compreensão, auxílio e motivação.

A todos os professores do Curso de Especialização em Internet das Coisas da UTFPR, destacando o professor Dr. Guilherme Luiz Moritz por concordar em orientar essa monografia.

RESUMO

CANGIANELLI, Daniel Negreiros. **Protocolo de comunicação GPS-Servidor utilizando MQTT.** 2019. 41 p. Monografia de Especialização em Internet das Coisas, Departamento Acadêmico de Eletrônica, Universidade Tecnológica Federal do Paraná. Curitiba, 2019.

Serviços de segurança para veículos são itens de interesse para a população devido à grande falta de segurança enfrentada no país. O uso da tecnologia da informação para o desenvolvimento de soluções tem grande potencial no mercado. Com a evolução da IoT, o protocolo MQTT, por exemplo, trata-se de um recurso que oferece baixo consumo de energia, pacotes de dados minimizados e distribuição eficiente de informações para um ou vários receptores. Dessa forma, o objetivo do trabalho foi desenvolver um protocolo, padronizando a comunicação entre dispositivo rastreador e servidor baseado em MQTT. Configurou-se um servidor para funcionar como *broker* MQTT, desenvolveu-se o código para que a placa ESP8266 se inscrevesse e publicasse mensagens referente a localização do veículo em um tópico específico do *broker* através da interface Mosquito, após isso implementou-se a programação para que uma aplicação se inscrevesse no tópico e realizasse a leitura e armazenamento dos valores de localização. Em seguida esta mesma aplicação foi configurada para funcionar como um *servlet* e responder à uma requisição HTTP do tipo GET, devolvendo os valores armazenados. Por fim foi desenvolvida uma aplicação móvel para dispositivos Android, que recebeu implementações para realizar a requisição de dados e apresentar a localização do veículo para o usuário uma lista com todos seus veículos e suas localizações no mapa.

Palavras-chave: IoT. Rastreamento. Segurança. Software. Veículo.

ABSTRACT

CANGIANELLI, Daniel Negreiros. **GPS-Server communication protocol using MQTT.** 2019. 41 p. Monografia de Especialização em Internet das Coisas, Departamento Acadêmico de Eletrônica, Universidade Tecnológica Federal do Paraná. Curitiba, 2019.

Vehicle security services are items of interest to the population due to the great lack of security faced in the country. The use of information technology to develop solutions has great potential in the market. With the evolution of IoT, the MQTT protocol, for example, is a feature that offers low power consumption, minimized data packets and efficient information distribution to one or several receivers. Thus, the objective of the work was to develop a protocol, standardizing communication between MQTT-based server and crawler device. A server was configured to function as an MQTT broker, code was developed for the ESP8266 board to subscribe and to publish vehicle location messages on a specific broker topic via the Mosquito interface, after which programming for an application to subscribe to the topic and read and store location values. Then this same application was configured to function as a servlet and respond to a GET HTTP request, returning the stored values. Finally, a mobile application for Android devices was developed, which received implementations to request data and present vehicle location to the user with a list of all their vehicles and their locations on the map.

Keywords: IoT. Security. Software. Tracking. Vehicle.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

API	*Application Programming Interface*
CoAP	*Constrained Application Protocol*
DNS	*Domain Name System*
GMS	*Global System for Mobile*
GPS	*Global Positioning System*
HTTP	*Hypertext Transfer Protocol*
IBM	*International Business Machines Corporation*
IDE	*Integrated Development Environment*
IoT	*Internet of Things*
IP	*Internet Protocol*
IPv4	*Internet Protocol version 4*
IPv6	*Internet Protocol version 6*
JSON	*JavaScript Object Notation*
LoRa	*Long Range*
LoRaWAN	*Long Range Wide Area Network*
MQTT	*Message Queuing Telemetry Transport*
MQTT-SN	*Message Queuing Telemetry Transport for Sensor Network*
REST	*Representational State Transfer*
SDK	*Software Development Kits*
TCP	*Transmission Control Protocol*
TLS	*Transport Layer Security*

SUMÁRIO

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

As tecnologias e serviços de rastreamento veicular tratam-se de um mercado que desperta interesse. Há um grande aumento no número de roubo de cargas no território nacional e os impactos refletem não apenas na perda do veículo ou produto roubado, mas também no custo do transporte das cargas (RIBEIRO, 2009).

O mercado de rastreamento de veículos atende diferentes tipos de clientes, como frotas de caminhões e automóveis particulares. Para tanto, diversas tecnologias são aplicadas no desenvolvimento desses serviços, como a obtenção de dados, transmissão, comunicação e apresentação ao usuário. Moura (2004) destaca as principais funções de um serviço de rastreamento sendo a comunicação com a frota, referência de posição com mapa, aviso de emergência e bloqueio de veículo. Esses pontos tratam-se de variáveis que produzem diferenças de preços e de qualidade do serviço oferecido, levando em consideração critérios como precisão e confiabilidade.

O desenvolvimento de software como solução tecnológica abrange as mais diversas áreas, desde controle e gestão de uma empresa até uma aplicação completa de rastreamento de veículos.

Um sistema que realiza rastreamento de automóveis é incorporado por outros subsistemas: aquisição de dados de localização, armazenamento de dados e apresentação de dados que se comunicam e garantem o funcionamento da aplicação.

A Internet das Coisas (*Internet of Things* - IoT) é uma tecnologia que conecta à rede a pessoas, máquinas e pessoas, entrelaçando esses sistemas, fazendo eles serem capazes de interagir entre si, com o objetivo de criar novos serviços e aplicativos (PATEL; PATEL, 2016). IoT apresenta um grande potencial nesse setor, pois conta com tecnologias de fácil implementação. O protocolo MQTT, por exemplo, centraliza a recepção de dados em um elemento principal, que permite a escrita e leitura desses dados a partir de diversos outros pontos. O protocolo é ideal para aplicações móveis devido a baixo consumo de energia, pacotes de dados

minimizados e distribuição eficiente de informações para um ou vários receptores (MQTT-V5.0, 2019).

1.2 PROBLEMA

Existe uma grande diversidade de ferramentas e soluções que podem ser utilizadas no desenvolvimento de aplicações de rastreamento de automóveis. Esse fato se torna uma dificuldade para as empresas deste segmento, visto que não existe um padrão a ser seguido ou um conjunto de regras para determinado processo.

Para o processo de comunicação do rastreador para o servidor de dados, as empresas adquirem uma solução que presta esse serviço ou implementam sua solução de maneira específica.

Consequentemente alterações em código-fonte, sejam por atualizações ou manutenções, se tornam custosas para a empresa e trabalhosas para o desenvolvedor.

1.3 OBJETIVOS

1.3.1 Objetivo Geral

Desenvolver um protocolo, padronizando a comunicação entre dispositivo rastreador e servidor baseado em MQTT.

1.3.2 Objetivos Específicos

Etapas a serem realizadas para atingir-se o objetivo geral:

- Desenvolver código fonte do protocolo de comunicação entre o dispositivo rastreador e o servidor;
- Desenvolver uma aplicação móvel para Android a partir do Android Studio;
- Implementar funcionamento do modelo de arquitetura REST entre servidor e aplicação;
- Apresentar localização do veículo em tempo real na aplicação móvel.

1.4 JUSTIFICATIVA

Em virtude do avanço tecnológico, tornou-se possível utilizar ferramentas e protocolos criados para internet das coisas com o intuito de otimizar processos já existentes. Deste modo, aplicar um protocolo de comunicação com baixo custo energético como base para padronizar um processo, garantindo confiabilidade dos dados e fácil implementação, resulta em benefícios tanto para a comunidade de desenvolvedores, quanto para empresas deste segmento.

1.5 ESTRUTURA DO TRABALHO

Esta monografia de especialização está dividida em 5 (cinco) seções. Nesta primeira seção foi introduzido o assunto tema do trabalho e também foram abordados a motivação e os objetivos geral e específicos da pesquisa, a justificativa e a estrutura geral do trabalho.

Já na segunda seção, Fundamentação Teórica, foram descritas as tecnologias utilizadas para o desenvolvimento deste trabalho.

A terceira seção, Desenvolvimento da Aplicação, todos os passos para a realização desse presente trabalho foram descritos.

Na quarta seção: Apresentação e Análise dos Resultados, tendo como base os procedimentos dos tópicos Fundamentação Teórica e Desenvolvimento da Aplicação, neste capítulo serão descritos os resultados obtidos através do Protocolo de comunicação GPS-servidor utilizando MQTT.

Por último na quinta seção, denominada Conclusão, foi apresentada a importância e viabilidade do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 IoT

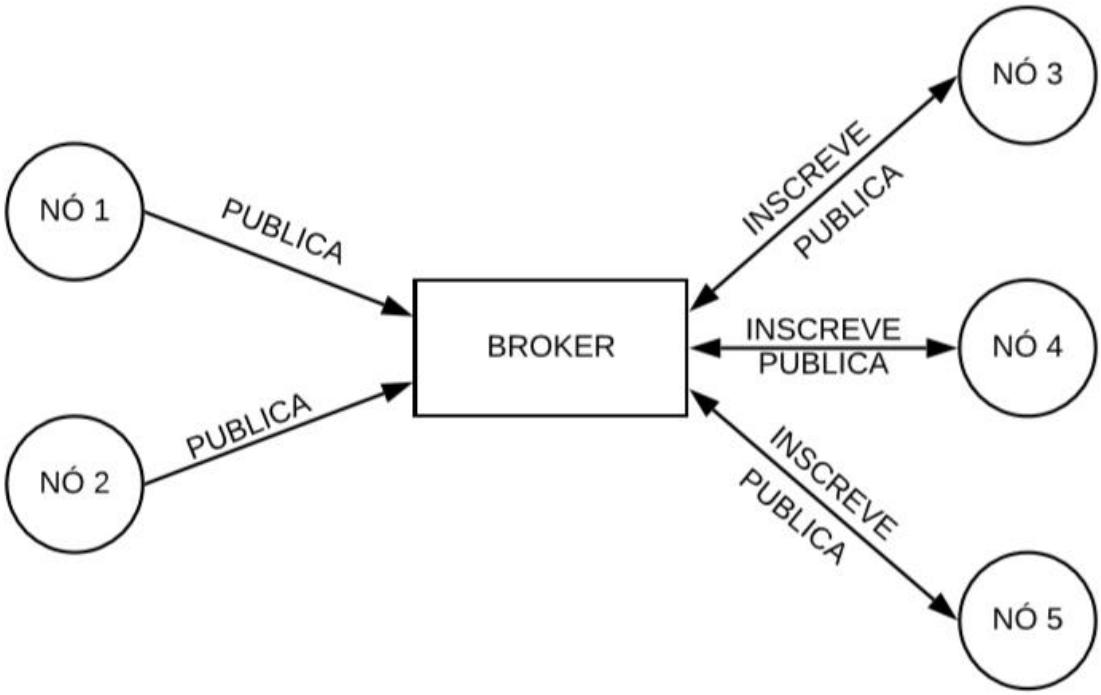
Internet das coisas (*Internet of Things* - IoT) é uma ligação entre os sensores e dispositivos que possuem a capacidade de armazenamento e compartilhamento de informações, através de uma estrutura única, permitindo transferência e análises das mesmas em nuvens (GUBBI *et al.*, 2013). Contudo de acordo com Evans (2011), “IoT é o momento exato em que foram conectados à internet ‘coisas ou objetos’ do que pessoas”. As principais aplicações da IoT são praticidade do desenvolvimento de aplicativos que conectam os dispositivos entre si e com o usuário, ter a garantia de que mensagens e informações sejam recebidas e colocadas em prática de maneira ágil, monitoramento de dispositivos, identificar problemas e ser capaz de resolvê-los sem auxílio humano (LEE; LEE, 2015).

2.2 MQTT

Existem conceitos e tecnologias que incorporam a internet das coisas, uma delas é o padrão de comunicação MQTT (*Message Queue Telemetry Transport*), desenvolvido pela IBM no final do século 90. É uma tecnologia que se baseia num “protocolo de mensagens com suporte para a comunicação assíncrona entre as partes, garantindo ser escalável em ambientes de rede que não são confiáveis." (YUAN, 2017). Yuan (2017) continua definindo o protocolo como um “modelo de publicação e assinatura, tendo um *broker* como servidor e diversos nós.

Além disso, o funcionamento desse modelo, apresentado na Figura 1, pode se dividir em três partes, a primeira sendo onde o cliente se conecta ao *broker* e escolhe um tópico para se inscrever, podendo ser uma conexão TCP/IP simples ou TLS criptografada. A segunda parte é definida pelas publicações de mensagens do cliente direcionadas ao tópico escolhido e ao *broker*. A última parte é definida pelo encaminhamento da mensagem a todos os clientes que são inscritos nesse tópico. Essa ferramenta pode ser considerada mais simples que o módulo Python Mosquito, uma vez que fornece SDKs e bibliotecas do MQTT em diversas linguagens de programação (YUAN, 2017).

Figura 1 - Ilustração do funcionamento do modelo MQTT



Fonte: Autoria própria.

2.3 MQTT-SN

A tecnologia MQTT-SN foi desenvolvida para ocupar o lugar do MQTT, sendo melhor nas particularidades dos sistemas de comunicação sem fio, baixa largura de banda, defeitos de link alto e pequeno tamanho de mensagens, tendo baixo custo para a execução dessa tecnologia, utilizando dispositivos que utilizam baterias, com recursos de processamento e armazenamento (STANFORD-CLARK; TRUONG, 2013).

2.4 REST

Existem alguns modelos de arquitetura para troca de informações entre cliente e servidor, dentre eles destaca-se o REST. Mumbaikar e Padiya (2013) descrevem a arquitetura REST como um modelo de solicitações entre servidor e cliente baseado em HTTP, logo o cliente envia uma solicitação para o servidor, que processa a mesma e devolve uma resposta para o cliente.

2.5 ANDROID STUDIO

A equipe de desenvolvimento Android, define o Android Studio como “o ambiente de desenvolvimento integrado (*Integrated Development Environment* - IDE) oficial para o desenvolvimento de aplicativos Android” (ANDROID, 2019). Esta plataforma permite que o desenvolvedor utilize a linguagem JAVA ou Kotlin, apresenta ferramentas de *debug* e possibilidade de emulação da aplicação.

2.6 ESP 8266

Os módulos baseados no controlador ESP 8266 representam um avanço na relação custo/recurso no cenário de internet das coisas. Dentre alguns dos recursos que esta placa abrange, destacam-se as suas configurações de WiFi e rádio (OLIVEIRA, 2017).

2.7 PESQUISA DE MERCADO

Algumas empresas já utilizam sistemas parecidos no rastreamento de veículos. A AllTech Rastreamento Veicular faz uso do GPS, satélites e softwares fornecendo acesso em tempo real das informações de movimentação, localização, paradas e rodagens. Com isso é possível fazer um monitoramento da frota/veículo via internet (ALLTECH, 2015).

A empresa Rastremar Rastreamento e Logística, possui um sistema que obtém latitude e longitude via satélite GPS e envia para seu próprio servidor, fornecendo esses dados ao usuário através da internet (RASTREMAR, 2014). Já a empresa Intersept apresenta as seguintes informações sobre o veículo em tempo real: velocidade, trajeto percorrido e o tempo que permaneceu em determinado local e, obtém elas por meio de um sistema GPS (geolocalização), e todas as imagens disponibilizadas por satélite. Com o veículo ligado, os dados são enviados, a partir de GPS, a cada três (3) minutos. Quando está desligado, são enviados a cada trinta (30) minutos (INTERSEPT, 2017).

A empresa Maxtrack utiliza Rastreadores GSM para obtenção de dados e realiza comunicação utilizando LoRa (MAXTRACK, 2019). LoRa é uma tecnologia

LPWAN que atua como transceptor, com o objetivo de coletar dados de sensores de leitura de um determinado local geográfico (BOR; VIDLER; ROEDIG, 2016).

A empresa Calsat Telemetria e Gestão de Frota, apresenta uma solução de rastreamento utilizando IoT, onde o sistema obtém informações de localização via GPS, porém sua comunicação utiliza LoRaWAN (CALSAT, 2019). LoRaWAN é um protocolo que incorpora o LoRa a uma infraestrutura de rede (WIXTED *et al.*, 2016).

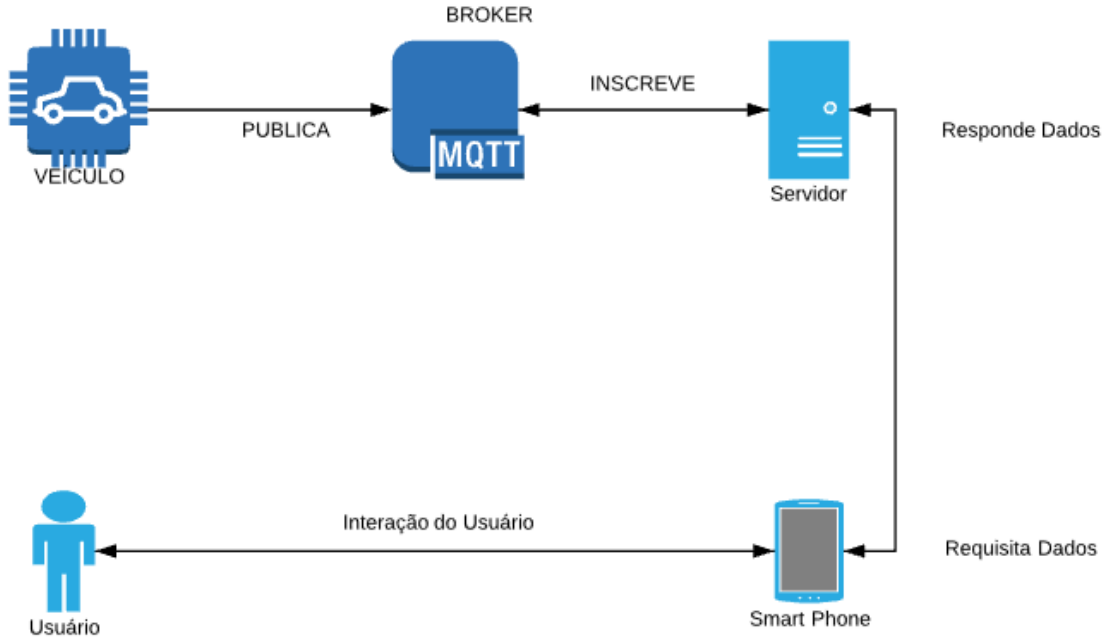
3 DESENVOLVIMENTO DA APLICAÇÃO

O projeto desenvolvido visou servir de protótipo para validar a utilização do protocolo MQTT em atividades de rastreamento veicular a partir disso as configurações e implementações foram realizadas.

O Apêndice A especifica respectivamente os requisitos para o hardware responsável pela publicação de dados, os requisitos técnicos para o *broker* e os requisitos técnicos para o *smartphone* que irá executar o aplicativo e consumir as informações.

A Figura 2 ilustra o funcionamento do projeto, desde a aquisição de dados até a entrega de informações para o usuário final.

Figura 2 - Desenvolvimento e funcionamento deste projeto



Fonte: Autoria própria.

Deve-se configurar o *broker* MQTT em um dispositivo (remoto ou local), a partir disso a placa eletrônica deve receber a implementação para que se conecte ao *broker*, crie um tópico e passe a publicar periodicamente valores da posição do veículo no mesmo.

Posteriormente, define-se ao servidor o comportamento de realizar a inscrição no tópico criado pelo hardware e passar a efetuar a leitura de todas as mensagens que são publicadas. Em paralelo a isso ele fica responsável por

responder a requisições HTTP do tipo GET com o valor mais atual de localização do veículo.

Essas requisições foram implementadas para serem realizadas pela aplicação móvel, o aplicativo ao iniciar envia uma requisição ao servidor, caso a resposta não seja nula, o valor é armazenado na memória interna do smartphone, persistindo assim os dados, toda vez que a localização é atualizada, esta também é adicionada a memória interna do telefone.

O valor limite pré-estabelecidos de posições que podem ser armazenadas é de cinco, com intuito de não sobrecarregar o telefone móvel. No caso do usuário executar a aplicação estando *offline*, as últimas posições recebidas do servidor serão listadas.

3.1 PROTOCOLO DE COMUNICAÇÃO

O protocolo deve garantir que a comunicação aconteça da seguinte maneira: a placa ESP8266 deve permitir o cadastro de um veículo a partir da sua placa e modelo, após isso, deve realizar a leitura serial de um dispositivo GPS e transmitir os valores recebidos para o *broker*, publicando-os em um tópico específico.

Os valores que devem ser processados pela placa tratam-se de latitude e longitude fornecidos pelo dispositivo GPS. O tópico a ser analisado porta o nome de “veículo”, quando alguma placa tenta publicar nele, caso ele ainda não exista é realizada sua criação, a partir de então as publicações são destinadas à ele.

Os valores publicados são referentes ao modelo, a placa, a latitude e a longitude do veículo, a mensagem contendo-os é codificada para o formato JSON e enviada para o tópico “veiculo”.

Codificando o seguinte veículo modelo FORD EDGE, portando placa ABC 1234, e com valores de latitude e longitude de -999, para o formato JSON tem-se o seguinte resultado:

```
{
        “modelo”: “FORD EDGE”
        “placa”: “ABC1234”
        “latitude”: “-999.0”
        “longitude”: “-999.0”
}
```

A padronização do protocolo está no fato de que o sistema apenas exige em sua implantação que seja informado os valores de placa e modelo do veículo, os valores de latitude e longitude são processados pelo servidor que a partir disso disponibiliza informações sobre a posição do veículo.

3.2 HARDWARE

A escolha do hardware responsável pelo rastreamento foi realizada considerando os seguintes pontos: preço, possibilidade de implementação de protocolos IoT, familiaridade com a placa, com base nisso adotou-se o módulo NodeMCU V2 Amica composto por um controlador ESP8266, ilustrado na Figura 3.

Figura 3 - Placa ESP8266

Fonte: Autoria própria.

No processo de elaboração do código fonte compilado na placa, adotou-se a plataforma Arduino IDE como compilador (Figura 4), configurando o gerenciador de placas da IDE para que fosse possível criar o código utilizando-a com a linguagem C++.

O código recebe a implementação que permite a placa se registrar em um tópico do *broker* e enviar valores JSONs para ele. Esses pacotes JSONs são preenchidos com informações que simulam a leitura de um módulo rastreador como latitude e longitude, além disso valores de modelo e placa do veículo também foram enviados, estes por sua vez têm a finalidade de identificar os veículos no servidor.

O código fonte que foi desenvolvido para atuar de acordo com a implementação descrita, está apresentado no Apêndice B.

Figura 4 - Novo projeto Arduino IDE

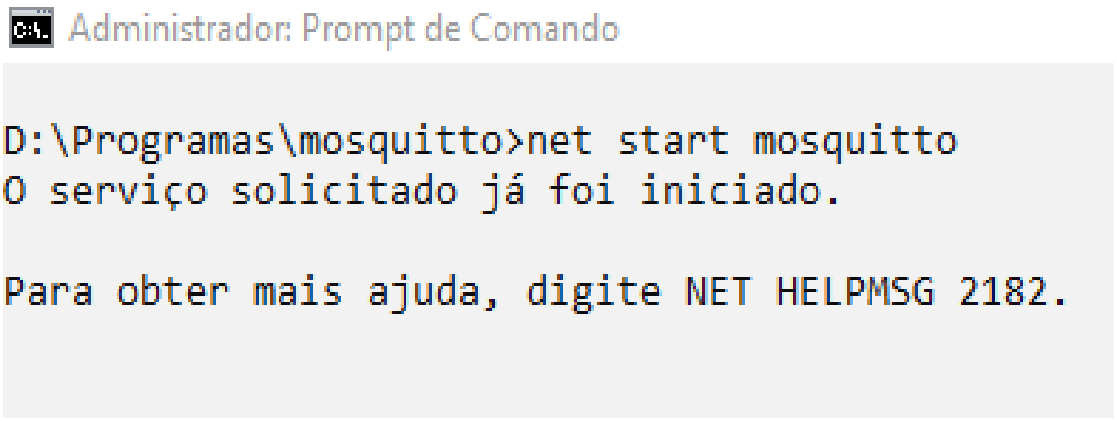
Fonte: Autoria própria.

3.3 BROKER

Para implementar a comunicação MQTT adotou-se o pacote Mosquito e a partir disso, configurou-se um computador pessoal, portando o sistema operacional Windows como servidor, em um primeiro momento o acesso ao mesmo fez-se utilizando o endereçamento IPv4, entretanto devido a alterações de endereço dinâmicas a forma de acesso foi alterada para utilizar o protocolo DNS.

A configuração do *broker* MQTT local foi efetuada fazendo o uso *prompt* de comandos como indica a Figura 5.

Figura 5 - Configuração de *broker* usando Mosquito

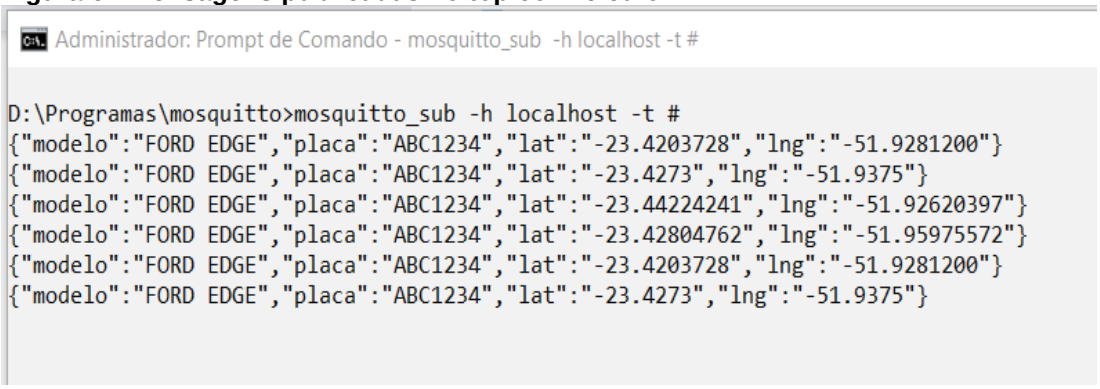


Fonte: Autoria própria.

Apesar de ter-se configurado um servidor local, existe a alternativa de realizar a configuração de um servidor na nuvem, utilizando-se de serviços conhecidos como por exemplo: Amazon Web Services, Google Clould Platform, Digital Ocean, entre outros.

Para garantir que a comunicação *publisher*/*broker* estava funcionando, utilizou-se outra janela do *prompt* de comando para se inscrever no tópico específico e, com isso receber todas as mensagens publicadas no mesmo (Figura 6).

Figura 6 - Mensagens publicadas no tópico "veículo"



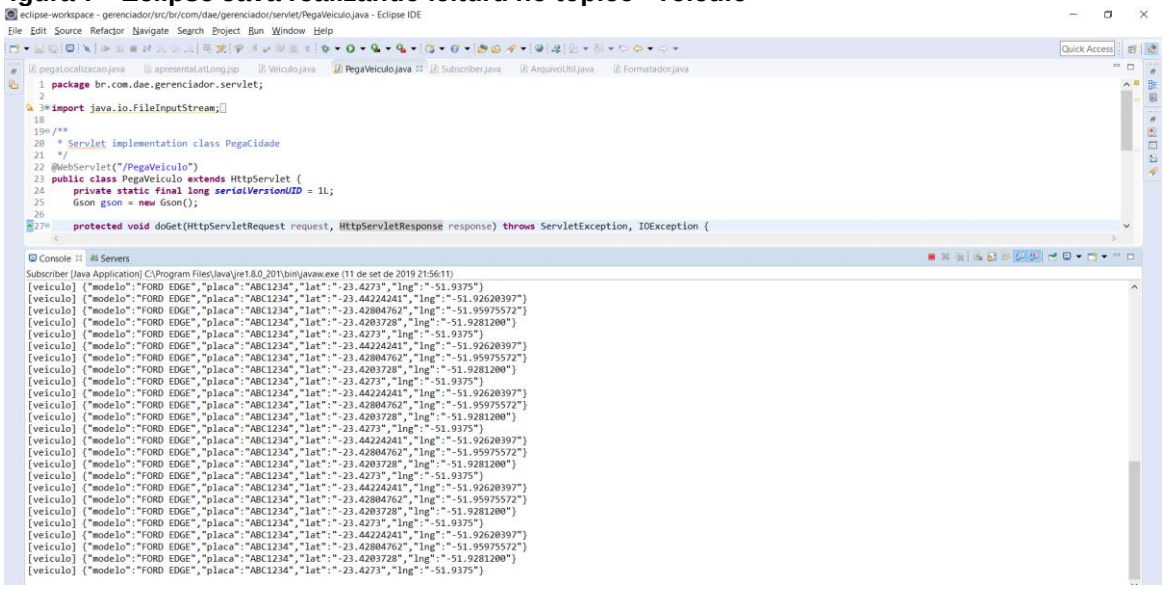
Fonte: Autoria própria.

3.4 SERVIDOR

Adotou-se o mesmo computador utilizado como servidor *broker*, para receber as configurações de servidor de dados.

A fim de captar os dados publicados no tópico, foi desenvolvido um programa utilizando-se da linguagem JAVA 8 e da IDE Eclipse (Figura 7). Neste programa os seguintes passos são executados: configura-se uma conexão MQTT com o *broker*, para tal fez-se o uso da biblioteca PAHO. MQTT disponível em Eclipse Foundation, uma organização sem fins lucrativos que atua como administradora da comunidade Eclipse (ECLIPSE, 2019); realiza-se a inscrição no tópico especificado como "veículo"; efetua-se a leitura dos valores presentes no tópico e armazena-os no próprio computador. A implementação para esta funcionalidade está apresentada no Apêndice C.

Figura 7 - Eclipse Java realizando leitura no tópico “veiculo”



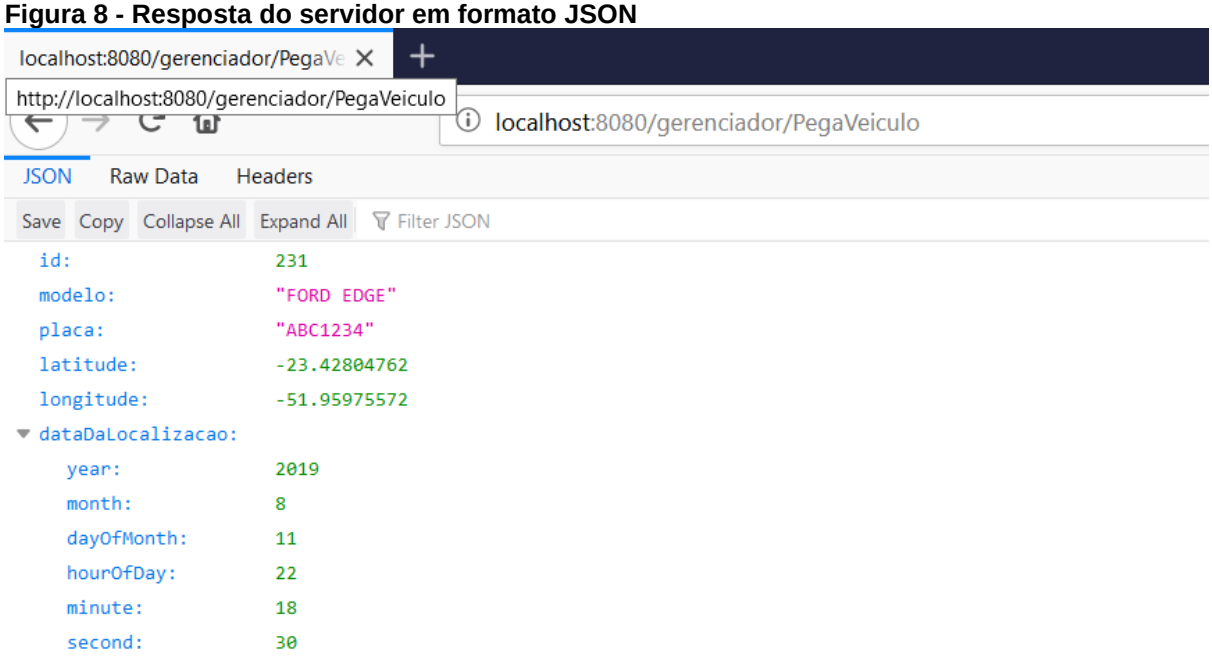
Fonte: Autoria própria.

3.4.1 SERVLET

Com intuito de disponibilizar esses dados para uma aplicação móvel, implementou-se um *servlet* no mesmo projeto do programa, descrito na sessão 3.4, definiu-se o *end-point* para obter os valores do veículo como “pegaVeiculo”, onde uma requisição HTTP do tipo GET está configurada no *servlet*.

Um objeto JSON com os mesmos valores enviados pela placa para o *broker*, neste momento é enviado do *servlet* a partir do *endpoint* configurado (Figura 8).

Figura 8 - Resposta do servidor em formato JSON



Fonte: Autoria própria.

A Tabela 1 apresenta o nome que cada valor codificado em JSON recebeu, acompanhado de sua descrição.

Tabela 1 - Campos de JSON

Valor JSON	Descrição
Id	Identificador único do veículo
Modelo	Fabricante e modelo do veículo
Placa	Placa do veículo
Latitude	Valor numérico da latitude do veículo
Longitude	Valor numérico da longitude do veículo
dataDaLocalizacao	Data da localização, contendo ano, mês, dia, hora, minuto e segundo que o valor foi adicionado no servidor.

Fonte: Autoria própria.

O código fonte que configura o funcionamento do *servlet* está apresentado no Apêndice D.

3.5 APLICATIVO

No processo de produção do aplicativo capaz de receber e processar os dados enviados pelo *servlet*, os seguintes passos foram seguidos: criação das telas; configuração de comportamento das telas; persistência interna de dados; comunicação com o *servlet*.

As tecnologias utilizadas implementar persistência interna de dados e comunicação com o servidor foram respectivamente “room” e “retrofit”. O “room” fornece uma camada de abstração sobre o SQLite para permitir acesso fluente ao banco de dados (ANDROID, 2019). “Retrofit” é a classe através da qual suas interfaces de API REST são transformadas em objetos que podem ser chamados em Java (SQUARE, 2019).

As dependências que foram necessárias ser incluídas no projeto da aplicação móvel estão expostos no Apêndice E. Já a aplicação, foi estruturada em pacotes e esta apresentada no Apêndice F.

O aplicativo foi programado para funcionar da seguinte maneira: ao abri-lo pela primeira vez, o mesmo tentará se comunicar com o servidor e solicitar a localização do veículo, caso ele obtenha uma resposta, esse valor será armazenado na memória interna do telefone e será possível visualizar o veículo em uma lista de

veículos, caso não obtenha uma resposta do servidor, a lista de veículos ficará vazia e uma mensagem de falha na comunicação será apresentada.

Na impossibilidade de se comunicar com o servidor, ou de se conectar a internet, a aplicação deve levar ao usuário as localizações que foram atualizadas na última comunicação com o servidor.

Todas as vezes que o aplicativo for executado, ele tentará buscar um veículo e sua localização no servidor, caso ele já tenha este veículo em memória, o valor da localização e também do horário em que ela foi obtida será atualizado na lista de veículos.

Ao clicar em um veículo da lista, uma nova tela será aberta, na qual será apresentado um mapa com as últimas 5 localizações do veículo, permitindo que ao clicar no veículo, o endereço e horário da localização sejam apresentados para o usuário.

Essa funcionalidade da aplicação foi configurada a partir do uso de uma Interface de Programação de Aplicativos, a API JavaScript do Google Maps, “permite personalizar mapas com seu próprio conteúdo e imagens para exibição em páginas da web e dispositivos móveis” (GOOGLEMAPS, 2019a). Para utiliza-la foi necessário gerar uma credencial da mesma, chamada de chave API ou API *key.*

Para gerar a chave da API do Google Maps, foi necessário acessar o site do Google para desenvolvedores, criar um projeto no qual a chave deve ser adicionada, abrir a página credenciais e criar a chave de API JavaScript do Google Maps (GOOGLEMAPS, 2019b).

Após isso, a chave foi configurada adicionando restrição de uso para aplicativos móveis com sistema Android, como é o caso da aplicação desenvolvida. Esse serviço fornecido pela Google, não é gratuito, porém para esse projeto foi utilizada a avaliação gratuita dos serviços de Google Cloud Platform com validade de um ano.

Após obter a chave, a mesma foi adicionada no projeto como uma dependência, a partir disso o funcionamento da aplicação na tela de mapas foi configurado e garantido.

O ciclo de funcionamento do projeto foi configurado para funcionar em classes chamadas “*ACTIVITIES*”, estas divisões estão presentes no Apêndice G.

4 APRESENTAÇÃO E ANÁLISE DOS RESULTADOS

Executando o projeto como um todo, os comportamentos esperados foram alcançados. Dessa forma, ordenando os acontecimentos, a placa ESP8266 se conecta ao *broker* e passa a publicar dados referentes a latitude e longitude do veículo em um intervalo de 30 em 30 segundos.

Nos testes relacionados a essa comunicação algumas inconsistências foram encontradas, pois o endereçamento estava sendo realizado a partir dos protocolos IPv4/IPv6, por conta disso, eventualmente o endereço da máquina que executava o *broker* sofria alterações, sendo necessário alterar o código do hardware.

O método adotado para essa situação foi definir como protocolo de acesso ao servidor o DNS, com isso independente de alterações de IP da máquina *broker*, o nome para acesso se mantinha o mesmo.

Em paralelo ao funcionamento da placa, o servidor se inscreve no *broker* realiza a leitura das mensagens publicadas no tópico veículo constantemente, em cada leitura os valores são armazenados. A biblioteca utilizada para comunicação com o *broker* permite a utilização do protocolo DNS, evitando o problema anteriormente apontado.

No que cabe a aplicação, durante a tela de carregamento, requisições de dados são direcionadas ao servidor, estes dados referentes a localização mais recente do veículo, é feito o tratamento e armazenamento no telefone móvel.

Os dados persistem na aplicação mesmo em situações de travamentos ou quando acabar a bateria do dispositivo.

Caso o telefone não consiga acessar a internet, apenas os veículos e localizações que serão apresentados são referentes à última resposta recebida do servidor.

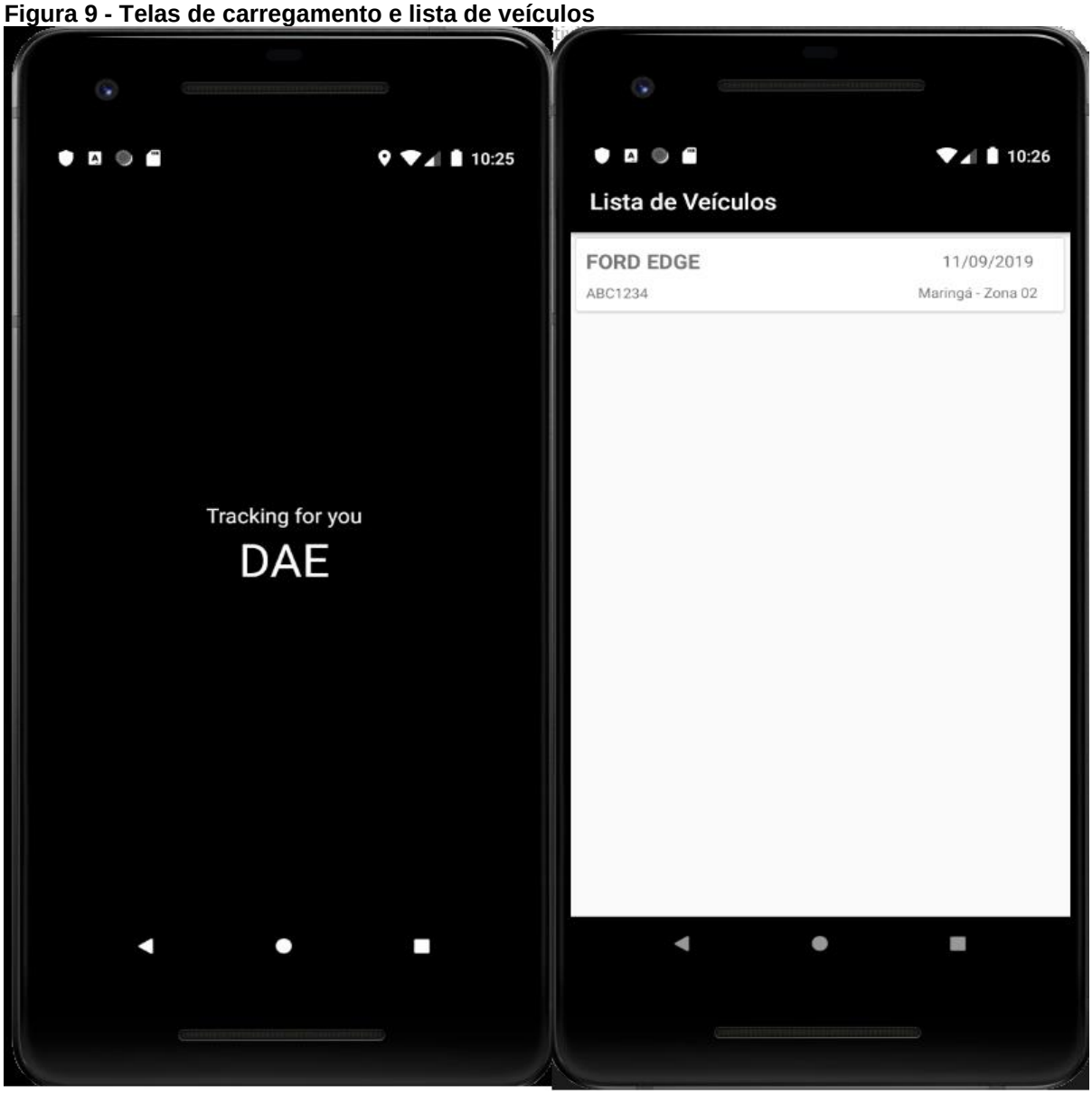
Após receber a resposta do servidor, o aplicativo monta uma lista com os veículos cadastrados, na qual a data e o endereço de localização do veículo são apresentados.

Ao efetuar um clique em um dos itens da lista, uma nova tela é apresentada: o mapa com a posição mais atual recebendo foco.

É possível diminuir o zoom e, assim visualizar as localizações anteriores do veículo, cada uma delas responde ao evento de clique apresentando a data e horário em que foram atualizadas.

A Figura 9 exibe as telas de carregamento e a lista de veículos citadas anteriormente.

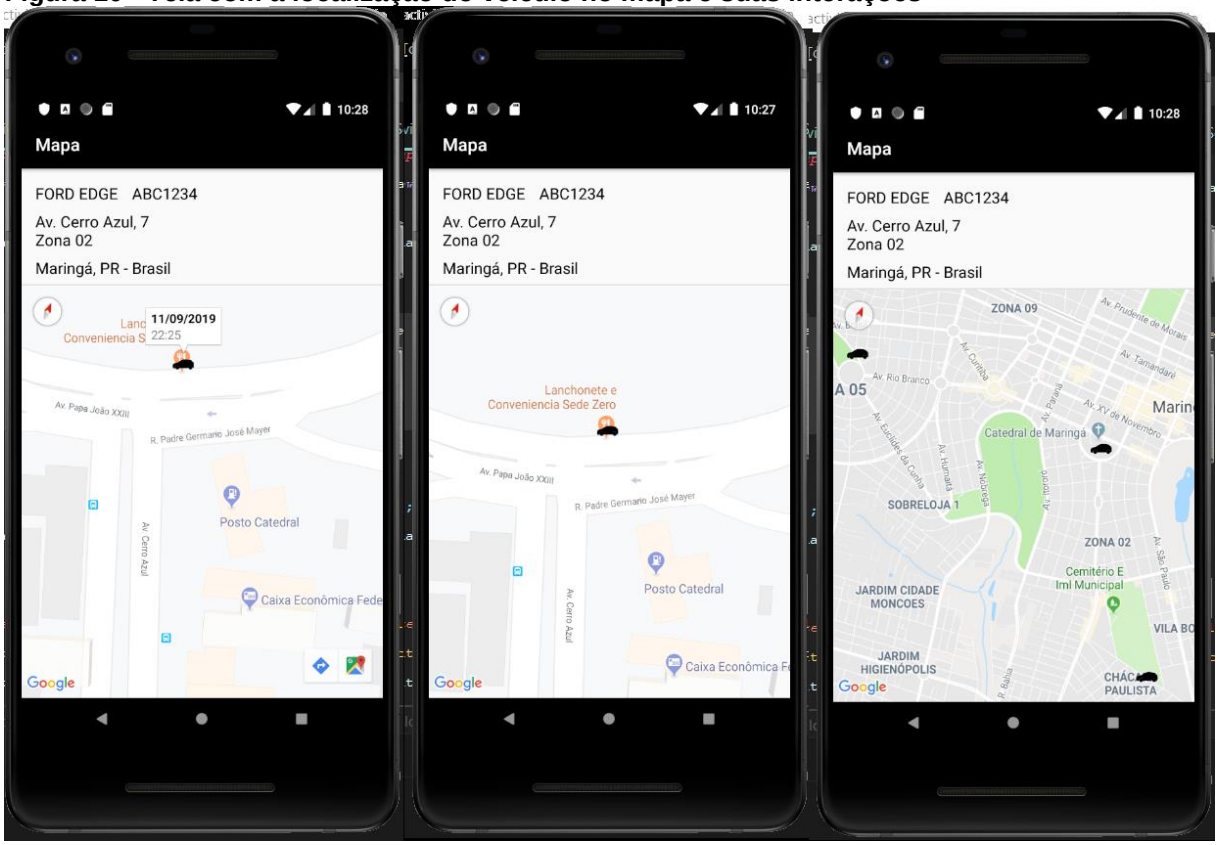
Figura 9 - Telas de carregamento e lista de veículos



Fonte: Autoria própria.

Os comportamentos previamente mencionados relacionados a tela com a localização do veículo no mapa são apresentados na Figura 10.

Figura 10 - Tela com a localização do veículo no mapa e suas interações



Fonte: Autoria própria.

5 CONCLUSÃO

Por ser um mercado amplo e tão diversificado no quesito de tecnologias aplicadas, como também na faixa de preço dos serviços, torna-se um campo favorável para a implementação e desenvolvimento de processos utilizando a tecnologia IoT.

Visto que, um número massivo de veículos será monitorado, enviando e recebendo dados através de algum tipo de serviço específico, com a implantação do protocolo IPv6, demonstra-se uma oportunidade para o uso de protocolos de comunicação como COAP e MQTT.

Neste trabalho prototipou-se o funcionamento do processo de rastreamento veicular utilizando tecnologias voltadas ao domínio IoT, os testes e análises realizados tratavam-se de um hardware utilizar o MQTT para enviar os dados de localização para um centralizador e um servidor de dados, bem como o servidor utilizar o protocolo para realizar a leitura e aquisição destes dados, que após tratados são enviados para uma aplicação móvel.

Dentre os resultados obtidos, destaca-se a facilidade de implementação da comunicação entre rastreador e servidor, contando com utilidades pré-definidas capazes de lidar com problemas de falha de conexão. O projeto apresentou uma solução satisfatória, sendo capaz de exibir para o usuário as localizações de acordo com os objetivos estabelecidos inicialmente.

Em contrapartida fez-se evidente quesitos relacionados à segurança da informação e integridade dos dados, tornando-se pontos significativos e objeto de estudos futuros.

REFERÊNCIAS

ALLTECH. **Alltech**: Rastreamento veicular. Copyright© Alltech Rastreamento Veicular 2015. Disponível em: <http://alltechrastreamento.com.br/2015/rastreamentos.php>. Acesso em: 23 out. 2019.

ANDROID. **Android Studio**: Conheça o Android Studio. 2019. Disponível em: <https://developer. android.com/studio/intro>. Acesso em: 01 jun. 2019.

BOR, M. C.; VIDLER, J.; ROEDIG, U. **LoRa for the Internet of Things**. International Conference on Embedded Wireless Systems and Networks (EWSN), p. 361-366. 2016. Disponível em: <https://dl.acm.org/citation.cfm?id=2893802>. Acesso em: 15 out. 2019.

CALSAT. **IoT**: Internet of Things. Copyright© CalSat Sistema de Rastreamento LTDA 2019. Disponível em: <https://www.calsat.com.br/iot>. Acesso em: 23 out. 2019.

ECLIPE. **Eclipse Foundation**: About the Eclipse Foundation. Copyright© Eclipse Foundation, Inc 2019. Disponível em: <https://www.eclipse.org/org/>. Acesso em: 23 out. 2019.

EVANS, D. **A Internet das Coisas**: Como a próxima evolução da Internet está mudando tudo. CISCO IBSG, abr. 2011. Disponível em: <https://www.cisco.com/c/dam/global/pt_br/assets/executives/pdf/internet_of_things_iot_ibsg_0411final.pdf>. Acesso em: 01 jun. 2019.

GOOGLEMAPS. **Google Maps Platform**: Overview. 2019a. Disponível em: <https://developers.goog le.com/maps/documentation/javascript/tutorial>. Acesso em: 23 out. 2019.

GOOGLEMAPS. **Google Maps Platform**: Get an API Key. 2019b. Disponível em: <https://developers.google.com/maps/documentation/javascript/get-api-key>. Acesso em: 23 out. 2019.

GUBBI, J.; *et al*. Internet of Things (IoT): A vision, architectural elements, and future directions. **Future generation computer systems**, vol. 29, n.7, p. 1645-1660. 2013.

INTERSEPT. **Rastreamento de veículos**. Copyright© Grupo intersept 2017. Disponível em: <https://www.intersept.com.br/servicos/rastreamento-de-veiculos-e-familiar/rastre amento-de-veiculos/>. Acesso em: 23 out. 2019.

LEE, I.; LEE, K. **The Internet of Things (IoT)**: Applications, investments, and challenges for enterprises. Business Horizons, v. 58, n. 4, p. 431-440, 2015. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0007681315000373>. Acesso em: 23 out. 2019.

MAXTRACK. **Maxtrack**: Rastreadores GSM + LoRa. Copyright© Maxtrack Industrial LTDA 2019. Disponível em: <https://maxtrack.com.br/>. Acesso em: 23 out. 2019.

MOURA, L. C. B. **Avaliação do impacto do sistema de rastreamento de veículos na logística.** Dissertação (Mestrado). Pontifícia Universidade Católica. Rio de Janeiro, 2004.

MQTT-V5.0. **MQTT Version 5.0**. Editado por Andrew Banks, Ed Briggs, Ken Borgendale, e Rahul Gupta. Copyright© OASIS Open 2019. Publicado em: 07 mar. 2019. Disponível em: <https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html>. Acesso em: 29 set. 2019.

MUMBAIKAR, S.; PADIYA, P. **Web services based on SOAP and REST principles**. International Journal of Scientific and Research Publications, v. 3, n. 5, p. 1-4, mai. 2013. Disponível em: <http://www.ijsrp.org/research-paper-0513/ijsrp-p17115.pdf>. Acesso em: 20 set. 2019.

OLIVEIRA, S. de. **Internet das Coisas com ESP8266, Arduino e Raspberry Pi**. 1. ed. São Paulo. Novatec Editora Ltda, 2017.

PATEL, K. K.; PATEL S. M. **Internet of things-IOT: definition, characteristics, architecture, enabling technologies, application & future challenges**. International journal of engineering science and computing 6.5, v. 6, n. 5, p. 6122-6131. 2016.

RASTREMAR. Rastreador Veicular. Copyright© Rastremar 2014. Disponível em: <http://www.rastremar.com.br/rastreadorveicular>. Acesso em: 23 out. 2019.

RIBEIRO, F. A. **O roubo de cargas nas rodovias do estado de São Paulo**. Trabalho de Conclusão de Curso. Pontifícia Universidade Católica de Campinas. Campinas, 2009.

SQUARE. **Retrofit**: A type-safe HTTP client for Android and Java. Copyright© Square, Inc 2009-2019. Disponível em: <https://square.github.io/retrofit/>. Acesso em: 23 out. 2019.

STANFORD-CLARK, A.; TRUONG, H. L. **MQTT for sensor networks (MQTT-SN) protocol specification**. International business machines (IBM) Corporation version, v. 1, nov. 2013. Disponível em: <http://www.mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf>. Acesso em: 18 set. 2019.

WIXTED, A. J.; *et al*. **Evaluation of LoRa and LoRaWAN for wireless sensor networks**. Institute of Electrical and Electronics Engineers (IEEE), p. 1-3. 2016. Disponível em: <https://ieeexplore.ieee.org/abstract/document/7808712/>. Acesso em: 18 out. 2019.

YUAN, M. **Conhecendo o MQTT**: Por que o MQTT é um dos melhores protocolos de rede para a Internet das Coisas? IBM**.** 2017. Disponível em: <https://www.ibm.com/developerworks/br/library/iot-mqtt-why-good-for-iot/index.html>. Acesso em: 01 jun. 2019.

APÊNDICE A: LISTA DE REQUISITOS PARA O SISTEMA

Requisitos de hardware para publicação de dados

Nome do Requisito	Descrição do Requisito	Prioridade
HW_001	Módulo GPS com saída de dados serial	0
HW_002	Placa microcontroladora deve possuir suporte à comunicação WiFi.	0
HW_003	Microcomputador ou computador com acesso a internet.	0

Requisitos técnicos para o broker

Nome do Requisito	Descrição do Requisito	Prioridade
BR_001	Microcomputador/ computador deve contar com o ambiente de execução Java (JRE) versão 8 ou superior.	0
BR_002	Microcomputador/ computador deve portar Apache Tomcat versão 8.0 ou superior.	1

Requisitos de sistema para o smartphone

Nome do Requisito	Descrição do Requisito	Prioridade
SP_001	Smartphone deve possuir sistema operacional Android.	0
SP_002	Para melhor desempenho é indicável que a versão do sistema android seja 7.0 ou superior.	1
SP_003	Recomenda-se que o aparelho tenha ao menos 8 gigabytes de armazenamento interno.	1

APÊNDICE B: CÓDIGO FONTE ESP8266

Configuração de variáveis

```
1
2  #include <ESP8266WiFi.h> // Importa a Biblioteca ESP8266WiFi
3  #include <PubSubClient.h> // Importa a Biblioteca PubSubClient
4  #include <ArduinoJson.h>
5
6  //defines:
7  //defines de id mqtt e tópicos para publicação e subscribe
8  #define TOPICO_SUBSCRIBE "veiculo"     //tópico MQTT de escuta
9  #define TOPICO_PUBLISH   "veiculo"    //tópico MQTT de envio de informações para Broker
10 //IMPORTANTE: recomendamos fortemente alterar os nomes
11 //            desses tópicos. Caso contrário, há grandes
12 //            chances de você controlar e monitorar o NodeMCU
13 //            de outra pessoa.
14 #define ID_MQTT  "dani"     //id mqtt (para identificação de sessão)
15 //IMPORTANTE: este deve ser único no broker (ou seja,
16 //            se um client MQTT tentar entrar com o mesmo
17 //            id de outro já conectado ao broker, o broker
18 //            irá fechar a conexão de um deles).
19
20 //defines - mapeamento de pinos do NodeMCU
21 #define D0    16
22 #define D1    5
23 #define D2    4
24 #define D3    0
25 #define D4    2
26 #define D5    14
27 #define D6    12
28 #define D7    13
29 #define D8    15
30 #define D9    3
31 #define D10   1
32
33 // WIFI
34 const char* SSID = "BROTHER12"; // SSID / nome da rede WI-FI que deseja se conectar
35 const char* PASSWORD = "1234senha"; // Senha da rede WI-FI que deseja se conectar
36 // MQTT
37 const char* BROKER_MQTT = "192.168.25.43"; //URL do broker MQTT que se deseja utilizar
38 int BROKER_PORT = 1883; // Porta do Broker MQTT
39 int escolha = 1;
40
41 //Variáveis e objetos globais
42 WiFiClient espClient; // Cria o objeto espClient
43 PubSubClient MQTT(espClient); // Instancia o Cliente MQTT passando o objeto espClient
44 char EstadoSaida = '0';  //variável que armazena o estado atual da saída
```

Implementação de funções

```
46 //Prototypes
47 void initSerial();
48 void initWiFi();
49 void initMQTT();
50 void reconectWiFi();
51 void mqtt_callback(char* topic, byte* payload, unsigned int length);
52 void VerificaConexoesWiFIEMQTT(void);
53 void InitOutput(void);
54
55 /*
```

Incialização de funções

```
58  void setup()
59  {
60    //inicializações:
61    InitOutput();
62    initSerial();
63    initWiFi();
64    initMQTT();
65  }
66
67  void initSerial()
68  {  Serial.begin(115200);}
69
70  void initWiFi()
71  {
72    delay(10);
73    Serial.println("------Conexao WI-FI------");
74    Serial.print("Conectando-se na rede: ");
75    Serial.println(SSID);
76    Serial.println("Aguarde");
77    reconectWiFi();
78  }
79  void initMQTT()
80  {  MQTT.setServer(BROKER_MQTT, BROKER_PORT);}
81
82  void reconnectMQTT()
83  {
84    while (!MQTT.connected())
85    {
86      Serial.print("* Tentando se conectar ao Broker MQTT: ");
87      Serial.println(BROKER_MQTT);
88      if (MQTT.connect(ID_MQTT, "daniel", "565953"))
89      {
90        Serial.println("Conectado com sucesso ao broker MQTT!");
91        MQTT.subscribe(TOPICO_SUBSCRIBE);
92      }
93      else
94      {
95        Serial.println("Falha ao reconectar no broker.");
96        Serial.println("Havera nova tentatica de conexao em 2s");
97        delay(2000);
98      }
99    }
100 }
```

Configuração de comunicação

```
void reconectWiFi()
{
  if (WiFi.status() == WL_CONNECTED)
    return;

  WiFi.begin(SSID, PASSWORD);

  while (WiFi.status() != WL_CONNECTED)
  {
    delay(100);
    Serial.print(".");
  }

  Serial.println();
  Serial.print("Conectado com sucesso na rede ");
  Serial.print(SSID);
  Serial.println("IP obtido: ");
  Serial.println(WiFi.localIP());
}
void VerificaConexoesWiFIEMQTT(void)
{
  if (!MQTT.connected())
    reconnectMQTT(); //

  reconectWiFi(); //
}
```

Publica MQTT

```
void EnviaEstadoOutputMQTT(int escolha)
{
  StaticJsonDocument<300> JSONdocument;
  JsonObject JSONencoder = JSONdocument.to<JsonObject>();

  JSONencoder["modelo"] = "FORD EDGE";
  JSONencoder["placa"] = "ABC1234";
  if(escolha == 1){
    JSONencoder["lat"] = "-23.4203728";
    JSONencoder["lng"] = "-51.9281200";
  }
  else if(escolha == 2){
    JSONencoder["lat"] = "-23.4273";
    JSONencoder["lng"] = "-51.9375";
  }
  else if(escolha == 3){
    JSONencoder["lat"] = "-23.44224241";
    JSONencoder["lng"] = "-51.92620397";
  }
  else if(escolha == 4){
    JSONencoder["lat"] = "-23.42804762";
    JSONencoder["lng"] = "-51.95975572";
  }
  char JSONmessageBuffer[100];
  serializeJson(JSONdocument, JSONmessageBuffer, sizeof(JSONmessageBuffer));
  MQTT.publish(TOPICO_PUBLISH,  JSONmessageBuffer, false);
  delay(10000);
  delay(10000);
  delay(10000);
}
```

Execução principal

```
void loop()
{
  VerificaConexoesWiFIEMQTT();
  EnviaEstadoOutputMQTT(escolha);
  if(escolha < 4)
    escolha = escolha +1;
  else
    escolha = 1;
  MQTT.loop();
}
```

APÊNDICE C: CÓDIGO FONTE DO SERVIDOR – CONFIGURAÇÃO MQTT

```
 1  package br.com.dae.gerenciador.servlet;
 2⊕ import java.io.File;
12
13  public class Subscriber implements MqttCallback {
14      private final int qos = 1;
15      private String topic = "veiculo";
16      private MqttClient client;
17
18⊖     public Subscriber(String uri) throws MqttException, URISyntaxException {
19          this(new URI(uri));
20      }
21⊖     public Subscriber(URI uri) throws MqttException {
22          String host = String.format("tcp://%s:%d", uri.getHost(), uri.getPort());
23          String[] auth = this.getAuth(uri);
24          String username = auth[0];
25          String password = auth[1];
26          String clientId = "MQTT-Java-Example";
27          if (!uri.getPath().isEmpty()) {
28              this.topic = uri.getPath().substring(1);
29          }
30          MqttConnectOptions conOpt = new MqttConnectOptions();
31          conOpt.setCleanSession(true);
32          conOpt.setUserName(username);
33          conOpt.setPassword(password.toCharArray());
34
35          this.client = new MqttClient(host, clientId, new MemoryPersistence());
36          this.client.setCallback(this);
37          this.client.connect(conOpt);
38
39          this.client.subscribe(this.topic, qos);
40      }
41⊖     private String[] getAuth(URI uri) {
42          String a = uri.getAuthority();
43          String[] first = a.split("@");
44          return first[0].split(":");
45      }
46⊖     public void sendMessage(String payload) throws MqttException {
47          MqttMessage message = new MqttMessage(payload.getBytes());
48          message.setQos(qos);
49          this.client.publish(this.topic, message);
50      }
51⊖     public void connectionLost(Throwable cause) {
52          System.out.println("Connection lost because: " + cause);
53          System.exit(1);
54      }
55      public void deliveryComplete(IMqttDeliveryToken token) {}
56⊖     public void messageArrived(String topic, MqttMessage message) throws MqttException {
57          System.out.println(String.format("[%s] %s", topic, new String(message.getPayload())));
58          ArquivoUtil.gravar(new String(message.getPayload()));
59      }
60⊖     public static void main(String[] args) throws MqttException, URISyntaxException {
61          Subscriber s = new Subscriber("http://localhost:1883");
62          s.sendMessage("Servlet Conectado!");
63      }
64  }
```

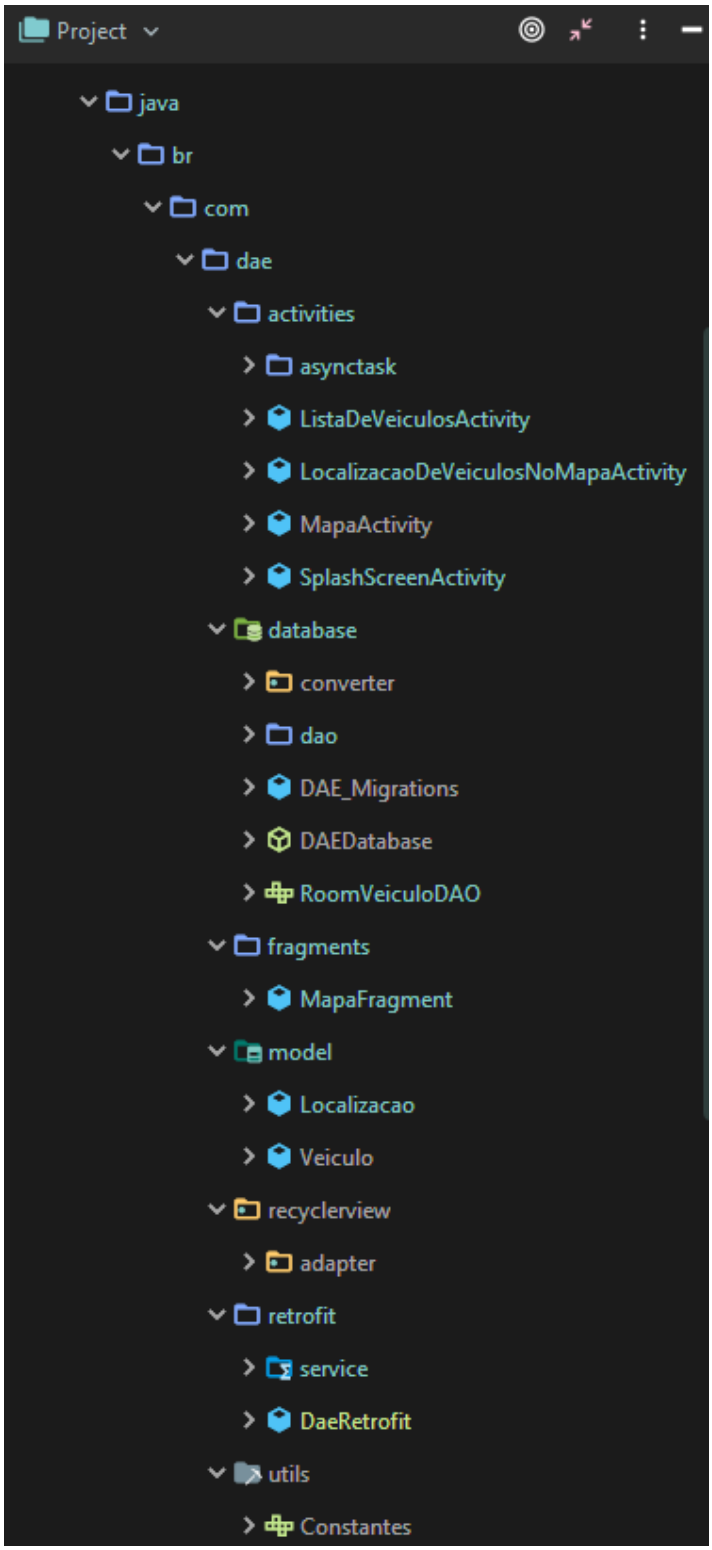
APÊNDICE D: SERVIDOR – CONFIGURAÇÃO SERVLET

```java
package br.com.dae.gerenciador.servlet;

import java.io.FileInputStream;
import java.io.IOException;
import java.nio.CharBuffer;
import java.nio.channels.FileChannel;
import java.nio.charset.Charset;
import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.google.gson.Gson;
import com.sun.corba.se.impl.ior.ByteBuffer;

/**
 * Servlet implementation class PegaCidade
 */
@WebServlet("/PegaVeiculo")
public class PegaVeiculo extends HttpServlet {
    private static final long serialVersionUID = 1L;
    Gson gson = new Gson();

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("application/json;charset=UTF-8");

        ServletOutputStream out = response.getOutputStream();
        String valorCarro = ArquivoUtil.ler("D:\\Dev\\Programacoes\\"
                + "workspaceEE\\gerenciador\\src\\br\\com\\dae"
                + "\\gerenciador\\servlet\\veiculo.txt");

        Veiculo v1 = Formatador.getModelo(valorCarro);
        String json = gson.toJson(v1);
        out.print(json);
    }
}
```

APÊNDICE E: DEPENDÊNCIAS DA APLICAÇÃO MOBILE

```
def room_version = "1.1.1"


implementation "android.arch.persistence.room:runtime:$room_version"
implementation 'com.google.android.gms:play-services-maps:17.0.0'
annotationProcessor "android.arch.persistence.room:compiler:$room_version"
implementation fileTree(dir: 'libs', include: ['*.jar'])
implementation 'androidx.appcompat:appcompat:1.1.0-rc01'
implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
testImplementation 'junit:junit:4.12'
androidTestImplementation 'androidx.test:runner:1.1.0-alpha4'
androidTestImplementation 'androidx.test.espresso:espresso-core:3.1.0-alpha4'
implementation 'com.google.android.material:material:1.0.0'
implementation 'com.squareup.retrofit2:retrofit:2.5.0'
implementation 'com.squareup.retrofit2:converter-gson:2.5.0'
implementation 'com.squareup.okhttp3:logging-interceptor:3.13.1'
```

APÊNDICE F: ORGANIZAÇÃO DO PROJETO MOBILE



APÊNDICE G: ACTIVITIES MOBILE

1. SplashScreenActivity: apresenta a logo do aplicativo e carrega os dados do servidor;
2. ListaDeVeiculosActivity: exibe todos os veículos em formato de lista, com detalhes de modelo, placa, horário de atualização e endereço. Responde ao evento de clique a um item na lista chamando a LocalizacaoDeVeiculosNoMapaActivity;
3. Nessa activity, em segundo plano é realizada a tomada de decisão de inserção ou atualização de valores de localização no banco de dados do dispositivo;
4. LocalizacaoDeVeiculosNoMapaActivity: apresenta endereços por extenso e um fragmento de mapa com as últimas localizações do veículo. Cada uma dessas localizações e representada por um desenho de um automóvel preto e respondem ao evento de clique apresentando o horário que o veículo esteve nessa posição;
5. MapaActivity: Classe responsável por configurar e implementar configurações próprias do mapa que será exibido para o usuário.