FACULDADE DE TECNOLOGIA SENAI "MARIANO FERRAZ"
CURSO DE PÓS-GRADUAÇÃO EM INTERNET DAS COISAS

RAFAEL GOMES DE PAULA

Estudo da viabilidade de análise de condução veicular através de dispositivo IOT

SÃO PAULO
2021

RAFAEL GOMES DE PAULA

Estudo da viabilidade de análise de condução veicular através de dispositivo IOT

Trabalho de Conclusão de Curso apresentado à Faculdade de Tecnologia Senai "Mariano Ferraz" como requisito parcial para obtenção do título de Especialista em Internet das Coisas.
Orientador: Prof. Esp. CAIO VINICIUS RIBEIRO DA SILVA

SÃO PAULO

ESTA PÁGINA DEVERÁ CONTER A FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DA INSTITUIÇÃO.
ESTE ITEM DEVERÁ SER IMPRESSO NO VERSO DA FOLHA DE ROSTO

RAFAEL GOMES DE PAULA

Estudo da viabilidade de análise de condução veicular através de dispositivo IOT

Trabalho de Conclusão de Curso apresentado à FACULDADE DE TECNOLOGIA SENAI "MARIANO FERRAZ" como requisito parcial para obtenção do título de Especialista em Internet das Coisas

Banca examinadora

_____

Mestre André
Faculdade de Tecnologia SENAI "Mariano Ferraz"

_____

Especialista Alexandro
Faculdade de Tecnologia SENAI "Mariano Ferraz"

_____

Especialista Caio
Faculdade de Tecnologia SENAI "Mariano Ferraz"

São Paulo, _____ de _____20

DEDICATÓRIA

Dedico este trabalho primeiramente a Deus e aos meus pais, pois é graças ao seu esforço que hoje posso concluir o meu curso.

AGRADECIMENTOS

Aos professores e colegas de curso, que contribuíram para a realização deste trabalho com dedicação e conhecimento.

Agradecimentos especiais à minha esposa, pela paciência e carinho.

A toda equipe da Faculdade de Tecnologia Senai Mariano Ferraz.

RESUMO

A internet das coisas surge no mundo de tecnologia para uma revolução tecnologia, que tem como principal objetivo a melhora da qualidade de vida da população que a consome. Através deste conceito, criou-se a ideia de implementar internet das coisas nos automóveis, que apesar de estarem a muito tempo no mercado, não possuem qualquer interação com a comunidade de tecnologia, restringindo suas tecnologias as montadoras. Pensando nisso, este trabalho vem para integrar a tecnologia IoT aos veículos. O trabalho tem como objetivo pesquisar a viabilidade de se analisar o comportamento de um condutor, através de dispositivo IoT, o trabalho foi realizado utilizando tecnologias convencionais que são amplamente utilizadas neste contexto. Através de testes e simulações feias durante o trabalho, é possível constatar que é esta análise é viável e de fácil aplicação. Deste modo, o trabalho concluiu seu objetivo e deixa oportunidades de melhorias e novas implementações que tenham como principal objetivo, a melhoria da qualidade de vida.

Palavras chave: IoT. Veículo. Condução.

ABSTRACT

The internet of things appears in the world of technology for a technology revolution, whose main objective is to improve the quality of life of the population that consumes it. Through this concept, the idea of implementing the internet of things in cars was created, which despite being in the market for a long time, do not have any interaction with the technology community, restricting their technologies to automakers. With that in mind, this work comes to integrate IoT technology to vehicles. The work aims to investigate the feasibility of analyzing the behavior of a conductor, through an IoT device, the work was carried out using conventional technologies that are widely used in this context. Through tests and ugly simulations during the work, it is possible to verify that this analysis is feasible and easy to apply. Thus, the work completed its objective and leaves opportunities for improvements and new implementations whose main objective is to improve the quality of life.
Keywords: IoT. Veiche. Driving

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

IoT	*Internet of Things* - Internet das Coisas
IP	*Internet Protocol* - Protocolo de Internet
TCP	*Transmission Control Protocol* - Protocolo de Controle de Transmissão
TCP/IP	*Transmission Control Protocol/Internet Protocol* - Protocolo de Controle de Transmissão/Protocolo de Internet
IPv4	*Internet Protocol version 4* - Protocolo de Internet versão 4
IPv6	*Internet Protocol version 6* - Protocolo de Internet versão 6
CI	*Circuito Integrado*
SO	Sistema Operacional
ARM	*Advanced RISC Machine*
API	*Application Programming Interface*
JSON	*JavaScript Object Notation*
MQTT	*Message Queue Telemetry Transport*
OASIS	*Organization for the Advancement of Structured Information Standards*
CAN BUS	Barramento *Controller Area Network*

1 INTRODUÇÃO

Desde 1886, quando a primeira patente de veículo automotor foi registrada, a indústria automobilística vem revolucionando a forma como a sociedade se locomove e executam as tarefas diárias, porém, por muito tempo a implementação de tecnologia embarcada nos carros não foi uma prioridade, pois os esforços eram direcionados principalmente aos motores e ao design. Em 1996, 100 anos após a primeira patente, a indústria automobilística deu o primeiro passo para a padronização da tecnologia embarcada nos veículos, a implementação do OBD - On-Board Diagnostic, sistema de autodiagnóstico do veículo. No Brasil, a implementação do OBD, se tornou obrigatório apenas em 2010. Desde então, diversas fabricantes estão implementando novas tecnologias em seus veículos que variam de sensores até sistemas inteligentes de condução semiautônoma do veículo. Apesar do desenvolvimento tecnológico na indústria automobilística ter avançado bastante desde 1886, a tecnologia utilizada nos veículos, ainda é bastante restrita ao fabricante e pouco divulgada no contexto acadêmico.

Se por um lado, a tecnologia embarcada demorou para ser prioridade na indústria automobilística, na indústria eletrônica, os microcontroladores tiveram um desenvolvimento muito rápido. O primeiro microcontrolador foi desenvolvido pela Intel em 1971 e tinha apenas 4 bits para processamento. Após 45 anos do primeiro microcontrolador, em 2016, já era possível encontrar microcontroladores com 32 bits de processamento, ou seja, 8 vezes a capacidade de processamento do primeiro microcontrolador.

Com a evolução dos microcontroladores e da conectividade dos dispositivos, surgiu uma nova categoria de dispositivos inteligentes, a Internet das Coisas (*Internet of Things* - IoT), baseada em dispositivos pequenos, conectados e eficientes.

O paradigma de comunicação Máquina a Máquina (*Machine to Machine* – M2M), é um conceito simples e versátil que sugere integração entre dispositivos eletrônicos, como o próprio nome sugere. A base do conceito é a comunicação entre máquinas através de um meio de comunicação.

A junção do IoT com o M2M, gera diversas possibilidades de aplicações que podem ser criadas. A Internet das Coisas se define com dispositivos e sistemas embarcados que possuem tecnologia de conectividade através da Internet e são interconectados para a execução de determinada tarefa com seu potencial máximo, criando espaço para a inovação de serviços e automações.

Dispositivos IoT já são populares na sociedade, são facilmente encontrados em forma de assistentes virtuais, lâmpadas, sensores de clima, vestíveis, entre outros dispositivos que fazem parte do dia a dia da sociedade moderna. A diversidade de dispositivos IoT associado à internet, gera uma vasta gama de possibilidades de automações, dando oportunidade para integrações entre dispositivos que antes disso, era pouco viável, como é o caso das automações nos veículos automotores. Com a chegada da IoT, se torna viável a integração entre as tecnologias do carro com um dispositivo inteligente, que a partir desta automação, é possível realizar a leitura dos sensores, se conectando a rede CAN Bus. Nesse contexto, é possível criar análises dos dados para a obtenção de aspectos que variam da condução feita por uma motorista em uma viagem, até a customização dos módulos mecânicos do veículo.

Dado o contexto, a pesquisa realizada e descrita neste artigo, propor realizar a análise da condução veicular, que será baseada em dados coletados do veículo durante viagem , usando como parâmetro, os dados estabelecidos no manual do carro para uma boa condução, que visa estabelecer um perfil comportamental para o condutor do veículo.

1.1 Problematização e hipóteses

Em 2010, as fabricantes atuantes no Brasil, adotaram a padronização OBD II nos seus veículos. O OBD II possui diversos protocolos de comunicação que dão acesso as informações do carro em tempo real. Apesar de essa tecnologia estar estabelecida há mais de 10 anos no nosso país, o acesso a esta tecnologia e o conhecimento sobre a como extrair as informações ainda é muito escasso. A problematização deste trabalho é explorar as possibilidades de comunicação entre dispositivos IoT e os veículos que possuem essa tecnologia embarcada. As hipóteses sobre essa comunicação seguem abaixo:

a) A leitura dos dados do veículo é possível?

b) A leitura dos dados do veículo não será possível?

c) Os dados do veículo estarão criptografados pela montadora?

d) O dispositivo poderá interagir com o sistema do veículo, alterando dados?

1.2 Objetivos da pesquisa

1.2.1 Objetivo geral

Realizar um estudo sobre a capacidade de determinar a condução veicular através de um dispositivo IoT, usando algoritmos que faça predições sobre a conduta do motorista baseada nos dados coletados do veículo.

1.2.2 Objetivos específicos

- Construir um dispositivo protótipo IoT, usando tecnologias e arquiteturas de redes em sistemas que estão presentes no contexto de Internet das Coisas;
- Coletar os dados de sensores do veículo, processar e enviar para análise de comportamento;
- Demostrar os benefícios de forma clara na utilização de Internet das Coisas.

1.3 Justificativa e delimitação da pesquisa

A tecnologia está constantemente evoluindo para dispositivos menores, mais rápidos e cada vez mais inteligentes. Através dessa evolução, surgiu a Internet das Coisas, como uma nova habilidade da internet em se conectar e coletar dados de dispositivos de onde não eram possíveis as comunicações com a rede de computadores.

Com dispositivos pequenos, com alta capacidade de processamento, baixo consumo de energia e conexão sem fio, a Internet das Coisas abriu uma porta para diversas inovações tecnológicas.

Cada vez mais a sociedade tem suas rotinas automatizada por dispositivos inteligentes como os celulares, wearables e até cafeteiras inteligentes, porém, veículos automotivos, apesar de possuírem tecnologia embarcada, ainda não estão inseridos no contexto da Internet das Coisas, isso foi a principal causa que motivou esse estudo a explorar novas possibilidades de automação e análises feitas a partir da coleta de dados do automóvel.

O estudo tem seus limites definidos dentro de cada contexto, como é mostrado a seguir:

a) Dispositivo IoT, se limitando a coleta de velocidade atual e rotação do motor, com a finalidade de enviar essas informações somente durante o percurso em que o carro está em movimento.

b) Nuvem privada, se limitando a comunicação com o dispositivo IoT, com foco no recebimento e processamento dos dados coletados para análise de condução do veículo.

c) Pesquisa e Estudo, se limitando a um único veículo específico utilizado.

d) Comunicação IoT, se limitando a rede GSM para troca de informações

1.4 Metodologia

a) Análises bibliográficas

b) Compreensão sobre o tema

c) Protótipo de dispositivo e rede IoT

1.5 Viabilidade da pesquisa

1.5.1 Cronograma

Tabela 1 - Cronograma de Atividades

#	Atividades	jan/21		fev/21		mar/21		abr/21		mai/21		jun/21	
1.	Desenvolvimento Monografia												
2.	Desenvolvimento Hardware												
3.	Desenvolvimento Software												
4.	Desenvolvimento Firmware												
5.	Testes												

Fonte: Autoria Própria

1.5.2 Recursos

Para o desenvolvimento da pesquisa e prototipação do dispositivo, foram necessários os itens apresentados na tabela a seguir.

Tabela 2 - Recursos

Recurso	Quantidade	Custo	Já Possuía?
Protoboard	2	R$ 20,00	Sim
Jumps	50	R$ 9,90	Não
LilyGo T-Call (ESP32)	1	R$ 150,00	Não
Raspberry Pi 4	1	R$ 450,00	Não
SDCard 64GB	1	R$ 70,00	Não
Módulo CAN BUS SN65HVD230	1	R$ 25,90	Não
OLED SSD1306	1	R$ 35,00	Não
RTC 3231	1	R$ 25,00	Não
Total	**58**	**R$ 785,80**	

Fonte: Autoria Própria

O presente estudo será dividido em cinco capítulos, sendo o primeiro, a introdução aos objetivos, metodologia e delimitação do problema. Em seguida, o segundo capítulo irá apresentar o referencial teórico sobre definição, arquitetura e tecnologias utilizadas. O Terceiro capítulo descreverá como foi o desenvolvimento do projeto em todos seus aspectos. Em Seguida, o quarto demonstrará os testes efetuados e os resultados obtidos. Ao final, o quinto capítulo irá tratar da conclusão do projeto.

2 REVISÃO BIBLIOGRÁFICA

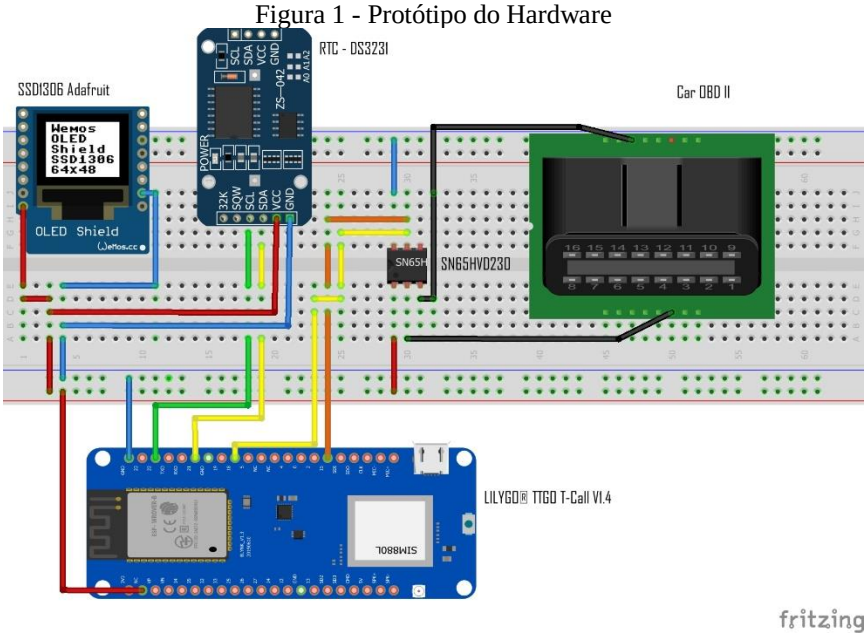
Neste capítulo são apresentados os principais conceitos teóricos necessários para a compreensão deste trabalho além de contextualizar a relevância da pesquisa para o meio acadêmico. A seção 2.1 descreve todos os hardwares que foram utilizados no projeto. A seção 2.2 descreve todos os softwares e protocolos que foram utilizados no projeto

2.1 Hardware

Para a construção do protótipo deste projeto, foi necessário realizar a separação da arquitetura em duas partes, onde a primeira parte é responsável pela coleta de dados do veículo e a segunda é responsável pelo processamento das informações coletadas.

A coleta de informações é feita através da interface OBDII do carro pelo transceptor SN65HVD230, utilizando a rede CANBUS para comunicação. Os dados recebidos pelo transceptor são enviados ao LilyGo, que utiliza ESP32 como MCU. No processamento dos dados, o MCU coleta as informações de data e hora, através do RTC DS3231, exibe os dados no OLED SSD1306, e envia um JSON com as informações através do módulo GSM SIM800L integrado a placa LilyGo. A imagem a seguir demonstra as ligações dos componentes citados.

Figura 1 - Protótipo do Hardware



Fonte: Autoria Própria

Todo o processamento dos dados e exposição da API para consulta dos dados é feito no Raspberry PI 4.

2.1.1 SD Card

Um cartão SD, abreviação de cartão "*Secure Digital"*, é um tipo de cartão de memória removível usado para ler e gravar grandes quantidades de dados em uma ampla variedade de aparelhos eletrônicos móveis, câmeras, dispositivos inteligentes e muito mais. O cartão SD foi lançado em 1999 e é o sucessor do agora obsoleto "*Multi Media Card"* (MMC). Era um dos vários formatos de cartão de memória concorrentes em uso por eletrônicos de consumo, como o extinto "*Memory Stick"* da Sony e o cartão "*Compact Flash*", que, embora ainda em uso, é muito menos comum do que nas décadas anteriores.

2.1.2 Raspberry PI 4

O Raspberry PI é um pequeno computador de baixo custo, criado pela Federação Raspberry PI Em 2006 no Reino Unido. Sua primeira versão disponível para venda, teve o preço de $35 dólares, muito abaixo dos valores de computadores pessoais encontrados nas lojas. A ideia dos criadores era desenvolver um produto com um preço acessível, que fosse pequeno e funcional, capaz de integrar facilmente o desenvolvimento de projetos eletrônicos com software. Atualmente esta placa contém 4 grandes versões, cada versão lançada se refere a atualização do modelo anterior. A última versão disponível do Raspberry PI é a 4, ela possui apenas o modelo B, que é equipado com o processador da Broadcom, o BCM2711, Quad core Cortex-A72 (ARM v8), processamento de 64-bit SoC e frequência de 1.5GHz, três opções de escolha para a memória RAM, podendo ser de 2GB , 4GB ou 8GB LPDDR4-3200 SDRAM, Conectividade WiFi 2.4 GHz e 5.0 GHz IEEE 802.11ac, Bluetooth 5.0, BLE, Gigabit Ethernet, 2 portas USB 3.0 e 2 portas USB 2.0. O Raspberry também possui 2 saídas de vídeo micro-HDMI, com capacidade de transmissão 4K em cada monitor. O dispositivo não contém nenhum equipamento para o armazenamento de dados integrado a placa, porém, é compatível com cartões micro SD, sendo necessário que o usuário insira um cartão com o sistema operacional para que ele funcione. Na figura 1, é possível ver um exemplar do Raspberry 4 model B.

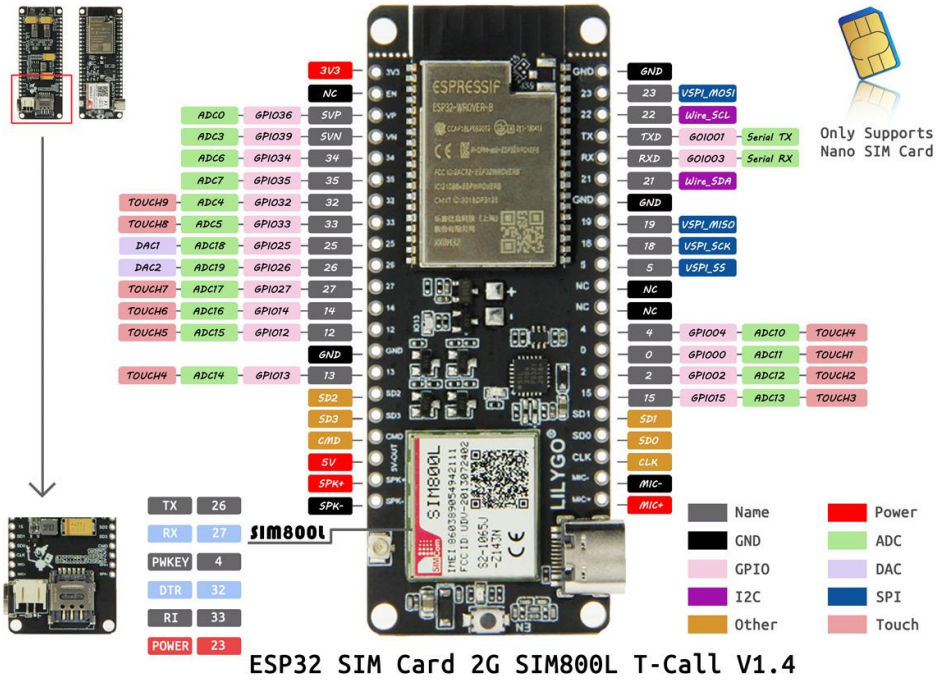
Figura 2 - Raspberry PI 4 - Model B

Fonte: Raspberry PI Foundation (2021)

2.1.3 LilyGo TTGO T-Call

A LilyGo TTGO T-Call é uma placa de desenvolvimento de hardware baseado no MCU ESP32, integrado a uma placa GSM SIM800L. O ESP32 é fabricado pela Espressif e foi criado em 2016 com o intuito de facilitar e baratear o desenvolvimento de dispositivos inteligentes, além de melhorar seu antecessor, o ESP8266. Para isso o ESP32 possui as seguintes configurações: conectividade WIFI, Bluetooth e BLE, têm processamento ajustável, podendo variar de 80 MHz a 240MHz, baixo consumo de energia e memória flash suficiente para grandes aplicações. A versão utilizada do ESP32 é a WROVER-B, com 4MB de Flash SPI e 8MB de PSRAM. O SIM800L, que também integra o LilyGo, é um módulo para comunicação GSM que necessita de um chip para seu funcionamento e possui interfaces de entrada e saída de áudio que estão integradas nos pinos do LilyGo. A placa de desenvolvimento é alimentada por uma entrada USB-C e têm tensão de funcionamento entre 2.7v e 3.6v. Na figura 2 é possível ver um exemplar da placa.

Figura 3 - LilyGo TTGO T-Call

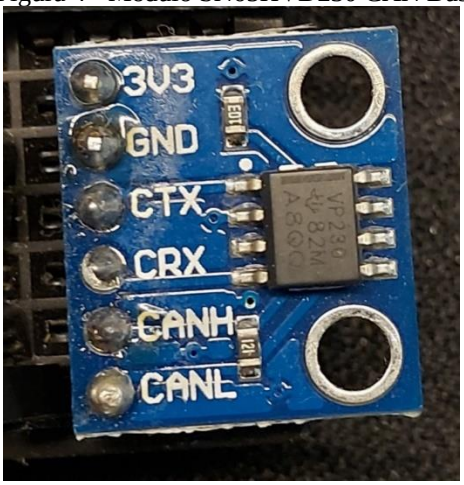


Fonte: Retirado de LilyGo, 2021

2.1.4 SN65HVD230

O SN65HVD230 é um transceptor para comunicação com a rede CAN (*Controller Area Network*). Esse dispositivo foi projetado para trafegar dados a taxas de até 1 megabit por segundo (Mbps), e incluem muitos recursos de proteção que fornecem aos dispositivos e a rede CAN mais robustez. Os dispositivos destinam-se ao uso em aplicações que empregam a camada física de comunicação serial CAN de acordo com o padrão ISO 11898. Na figura 3 é possível ver um exemplar do módulo.

Figura 4 - Módulo SN65HVD230 CAN Bus

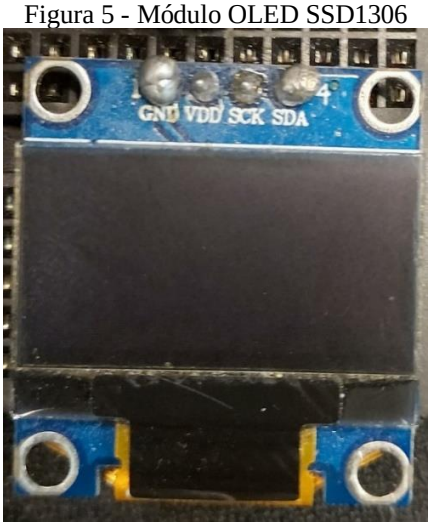


Fonte: Autoria Própria

2.1.5 SSD1306

O SSD1306 é um driver CMOS OLED / PLED de chip único com controlador para emissão de luz (orgânica / polímero) em sistema de exibição gráfica por matriz de pontos de diodo. Consiste em 128 segmentos e 64 comuns. Este IC é projetado para painel OLED tipo cátodo comum. O SSD1306 incorpora controle de contraste, RAM de exibição e oscilador, o que reduz o número de componentes externos e consumo de energia. Possui controle de brilho de 256 etapas. Dados / comandos são enviados do MCU geral através da interface paralela compatível com a série 6800/8000 selecionável por hardware, I2C ou Serial. É adequado para muitas aplicações portáteis compactas, como sub-display de telefone celular, MP3 player, calculadora etc. Na figura 4 é possível ver um exemplar do visor.

Figura 5 - Módulo OLED SSD1306

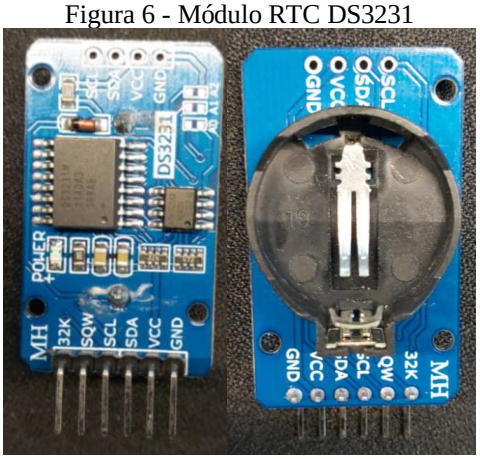


Fonte: Autoria Própria

2.1.6 RTC DS3231

O DS3231 é um relógio de tempo real I2C (RTC – *Real Time Clock*) extremamente preciso e de baixo custo, possui um oscilador de cristal com compensação de temperatura (TCXO) e cristal integrado. O dispositivo incorpora uma entrada de bateria e mantém uma cronometragem precisa quando a alimentação principal do dispositivo é interrompida. A integração do ressonador de cristal aumenta a precisão de longo prazo do dispositivo e reduz a contagem de peças em uma linha de fabricação. Na figura 5 é possível ver um exemplar do módulo.

Figura 6 - Módulo RTC DS3231



Fonte: Autoria Própria

2.2 Software

Esta seção aborda assuntos referentes à camada de software que orquestra a comunicação e o processamento de dados. Os protocolos de comunicação entre os dispositivos, o IDE utilizado na construção do firmware e o sistema operacional, são o tema principal desta seção. Na contextualização destes softwares fica claro a preferência por ferramentas open-source, que geralmente possuem uma grande comunidade de desenvolvedores envolvidas.

2.2.1 Balena Etcher

Etcher é um software open-source gerador de imagem de sistemas operacionais. Amplamente usado pela comunidade de software para a instalação de sistemas operacionais em unidades flash como o SD Card e outras unidades de disco via USB. O software faz parte de uma coleção de softwares open source da empresa balena-io.

2.2.2 Linux

O Linux é um sistema operacional de código aberto desenvolvido por Linus Torvalds e lançado sua primeira versão em 1992. O desenvolvimento do sistema operacional ainda é liderado pelo seu criador, porém, conta com a ajuda de diversos programadores ao redor do mundo, uma vez que está disponível na internet para que qualquer pessoa possa fazer modificações. O objetivo do sistema Linux era inicialmente replicar e superar um sistema operacional já existente na época, o Minix, porém, com a política de código aberto, o sistema se tornou muito popular e hoje em dia é possível encontrar diversas versões deste SO rodando em computadores pessoais e servidores. O Kernel do Linux recebe diversas atualizações da comunidade de código aberto e em 2002 eles lançaram uma atualização para o Arch Linux, que

tornou ele compatível com os processadores de arquitetura ARM, o que permitiu que vários dispositivos que utilizam processadores baseados nessa arquitetura pudessem agora rodar um sistema operacional Linux.

A Raspberry PI Foundation desenvolveu o sistema operacional Raspberry PI OS para seus dispositivos a partir do código fonte do Debian, uma distribuição do Linux.

2.2.3 GIT

O GIT é um sistema de controle de versão de arquivos. Através deles é possível desenvolver projetos na qual diversas pessoas podem contribuir simultaneamente, editando e criando arquivos, permitindo que eles possam existir sem o risco de suas alterações serem sobrescritas. Desenvolvido em 2005 por Linus Torvalds precisamente para a criação do Kernel do Linux, atualmente ele é amplamente utilizado para colaboração de times de desenvolvimento nas empresas e em comunidades de software.

2.2.4 GitHub

O GitHub é uma plataforma web que oferece hospedagem de código, além da implementação gratuita dos mecanismos do GIT. Ele oferecer um espaço limitado, porém gratuito, para que qualquer usuário possa armazenar seus códigos-fonte de forma segura na nuvem. O GitHub é uma das ferramentas mais conhecidas na comunidade da internet pela quantidade de softwares open-source que eles hospedam. O GitHub foi desenvolvido por Chris Wanstrath, J. Hyett, Tom Preston-Werner e Scott Chacon, e foi lançado em fevereiro de 2008. A empresa, GitHub, Inc., existe desde 2007 e está sediada em São Francisco.

2.2.5 VS Code

O Visual Studio Code é um editor de código-fonte (texto) multiplataforma disponibilizado pela Microsoft para o desenvolvimento de aplicações com suporte a diversas linguagens. foi anunciado, com uma versão prévia lançada, em 29 de abril de 2015 pela Microsoft na conferência Build de 2015.

2.2.6 PlatformIO

O PlatformIO é uma plataforma para desenvolvimento de firmwares em sistemas embarcados. A plataforma é open-source e possui seu código aberto para que outros desenvolvedores possam contribuir para seu desenvolvimento. A ferramenta tem o principal

objetivo de transformar a experiencia do desenvolvimento de firmware em algo simples. Teve sua primeira versão lançada em 2014 no GitHub, até o momento, já foram lançadas mais de 107 versões do software.

A plataforma é compatível com as mais modernas placas de desenvolvimento de hardware como o Arduino e o ESP32. Além de promover um ambiente amigável ao desenvolvedor, a ferramenta conta com alguns recursos de auto completar, análises estáticas de código e sugestões de melhores práticas durante o desenvolvimento.

2.2.7 Heroku

O Heroku é uma plataforma em nuvem como serviço (PaaS) que permite que as empresas e desenvolvedores construam, entreguem, monitorem e dimensionem aplicativos. A plataforma é gratuita para a comunidade de desenvolvedores além de ser totalmente integrada ao GitHub.

O software foi inicialmente desenvolvido por James Lindenbaum, Adam Wiggins, e Orion Henry para suportar projetos desenvolvidos em Ruby que utilizavam Rack como *middleware*. Em dezembro de 2010, A Salesforce.com adquiriu a Heroku como uma subsidiária integral da Salesforce.com.

2.2.8 Diagrams.Net

O Diagrams.net é software online de código aberto desenvolvido para criar diversos tipos de diagramas. O sistema foi construído pelo Google com o intuito de dar acesso a softwares de qualidade para diagramação de uma forma gratuita e descomplicada.

2.2.9 Fritzing

O Fritzing é um software de código aberto, para desenvolvimento de protótipos de circuitos eletrônicos. Ele foi desenvolvido na *Fachhochschule Potsdam* (*University of Applied Sciences Potsdam*), na Alemanha, e lançado em sua primeira versão 0.1b em novembro de 2008. Ele permite que sejam criados layouts de circuitos eletrônicos de maneira simples e rápida além de permitir ao usuário realizar a documentação dos esquemáticos e seus componentes.

2.2.10 Node-Red

O Node-RED é uma ferramenta de programação open-source que tem como objetivo conectar dispositivos de hardware, API (*Application Programming Interface*) e serviços online de maneira inovadora e simples. Ele fornece um editor web, acessado pelo navegador, que torna fácil a experiencia do usuário. A ferramenta possui diversos módulos pré-configurados que são capazes de executar funções que muitas vezes são demoradas em sua implementação, como conectores de banco de dados e exposição de end-points de API. As funções desenvolvidas na ferramenta podem ser facilmente exportadas para JSON (JavaScript Object Notation) e importadas novamente na ferramenta.

2.2.11 No-IP

O No-IP é um provedor DNS com pacotes de serviços gerenciados com suporte a DNS dinâmico (DDNS). É adequado para usuários que desejam acessar câmeras e outros dispositivos de forma remota, ou seja, de qualquer lugar com conexão com a Internet.

O No-IP foi criado em 1999 por Dan Durrer, fundador e hoje CEO da empresa que atende a mais de 25 milhões de usuários. Tornou-se uma rede com mais de 100 pontos locais de hospedagem DNS, que prometem desempenho inigualável, com garantia de 100% de tempo de atividade.

2.2.12 MongoDB

MongoDB é um banco de dados não relacional open-source, que armazena seus dados no formato de documentos, ele foi projetado para facilitar o desenvolvimento e o dimensionamento de aplicações que precisam de alta velocidade de resposta. Os dados armazenados no MongoDB estão no formato JSON, que permite criar dados com alta complexidade de informações e estrutura de dados.

2.2.13 MQTT

O MQTT (*Message Queue Telemetry Transport*) foi inventado e desenvolvido pela IBM no final dos anos 90. Sua aplicação original era vincular sensores em pipelines de petróleo a satélites. Como seu nome sugere, ele é um protocolo de mensagem com suporte para a comunicação assíncrona entre as partes. Um protocolo de sistema de mensagens assíncrono desacopla o emissor e o receptor da mensagem tanto no espaço quanto no tempo e, portanto, é escalável em ambientes de rede que não são confiáveis. Apesar de seu nome, ele não tem nada

a ver com filas de mensagens, na verdade, ele usa um modelo de publicação e assinatura. No final de 2014, ele se tornou oficialmente um padrão aberto OASIS (*Organization for the Advancement of Structured Information Standards*), com suporte nas linguagens de programação populares, usando diversas implementações de software livre. Com isso, muitos softwares foram lançados implementando essa tecnologia que se popularizou entre os dispositivos de IoT.

2.2.14 MQTT Mosquitto

O *Broker* MQTT Mosquitto é um software open-source de mensageria que implementa o protocolo MQTT versões 5.0, 3.1.1 e 3.1. O Mosquitto foi projeto para ser leve e adequado para uso em todos os dispositivos, desde microcontroladores, com o ESP32 até Servidores.

2.2.15 Protocolo CAN

Muito utilizado na indústria automobilística, o protocolo CAN Bus (Barramento Controller Area Network) foi desenvolvido pela empresa alemã Robert BOSCH e disponibilizado nos anos 80. Sua aplicação inicial foi em ônibus e caminhões. Atualmente, é utilizado em veículos automotivos, navios e até tratores.

O CAN é um protocolo de comunicação serial síncrono. O sincronismo entre os módulos conectados à rede é feito em relação ao início de cada mensagem lançada ao barramento, evento que ocorre em intervalos de tempo regulares. Trabalha baseado no conceito multi-mestre, onde todos os módulos podem se tornar mestres em determinado momento e escravos em outro, além de suas mensagens serem enviadas em regime multicast, caracterizado pelo envio de toda e qualquer mensagem para todos os módulos existentes na rede. Outro ponto forte deste protocolo é o fato de ser fundamentado no conceito CSMA/CD with NDA (*Carrier Sense Multiple Access / Collision Detection with Non-Destructive Arbitration*). Isto significa que todos os módulos verificam o estado do barramento, analisando se outro módulo está ou não enviando mensagens com maior prioridade. Caso isto seja percebido, o módulo cuja mensagem tiver menor prioridade cessará sua transmissão e o de maior prioridade continuará enviando sua mensagem deste ponto, sem ter que reiniciá-la. As mensagens que trafegam nesta rede possuem quatro tipos de pacotes de informação, são eles:

- Quadro de dados (Data Frame)
- Quadro Remoto (Remote Frame)
- Quadro de Erro (Error Frame)

- Quadro de sobrecarga (Overload Frame)

Neste trabalho, foi explorado apenas o quadro de dados, uma vez que ele foi suficiente para o desenvolvimento da pesquisa. Ele é composto por sete grupos de bytes que formam um pacote, são eles:

- *Start of Frame (SoF)*: Início do quadro, com apenas um bit que denota o início da transmissão do frame de dados.
- *Arbitration Field*: Composto por 12 bits, sendo 11 bits reservados para a identificação única do pacote, e um bit para o RTR - *Remote transmission request*, que determina se o Quadro de Dados é dominante, quando 0, ou recessivo, quando 1.
- *Control*: Composto por 6 bits, sendo 1 bit reservado para identificação do formato do campo anterior, 0 quando o identificador contiver 11 dígitos e 1 para identificador estendido, em seguida há um bit reservado com valor 0 e por último há mais 4 bits que determinam o tamanho do grupo a seguir, que varia de 0 a 64 bits.
- *Data:* Composto por 64 bits, é o dado que será transmitido pelo frame.
- *Cyclic redundancy check (CRC):* Composto por 16 bits, sendo 15 deles utilizados para validação de erros de transmissão na rede CAN e 1 bit como delimitador.
- *Acknowledge (ACK):* Composto por 2 bits, sendo 1 bit para o ACK Slot e o outro bit para ACK Delimiter. É usado pela rede pra que o destinatário possa indicar o correto recebimento do pacote.
- *End of Frame (EoF):* Composto por 7 bits de valor 1 (Recessivos) que delimitam o fim do quadro de dados.

3 DESENVOLVIMENTO

Dado a contextualização da pesquisa, a seção tem como objetivo se aprofundar nos detalhes da construção do projeto, esclarecendo a linha de raciocínio utilizada, definindo procedimentos para as etapas e demonstrando os resultados obtidos.

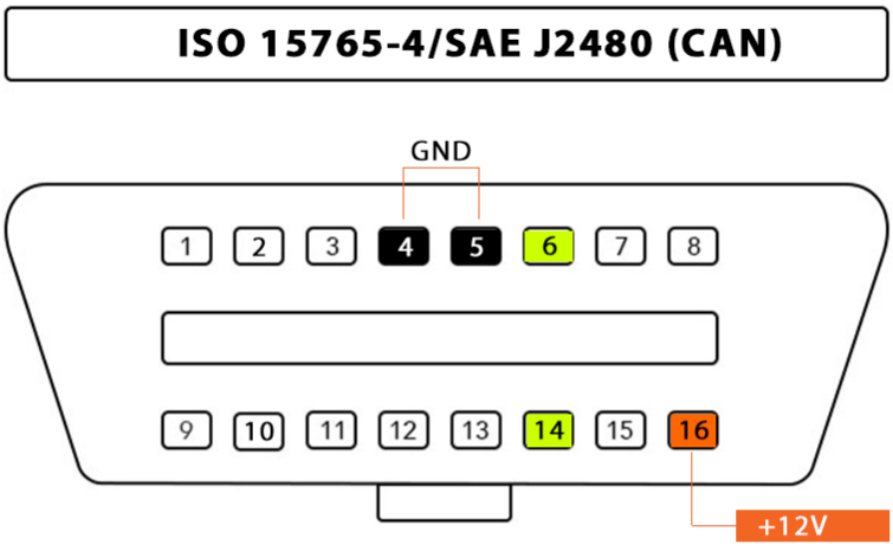
Os protótipos apresentados nesta seção foram testados em apenas um veículo, limitando os resultados obtidos. Dentro deste contexto, os resultados obtidos serão considerando verdadeiro apenas e tão somente quando o veículo for o mesmo utilizado nesta pesquisa: VW – Polo Highline 200 TSI 1.0 Flex 12V Aut.

3.1 Compreensão da rede CAN no contexto automotivo

Amplamente implementada pela indústria automotiva, a rede CAN se encontra na maioria dos carros que hoje circulam nas ruas. O acesso a rede se dá através da interface OBDII,

que se tornou obrigatória desde 2010 no Brasil. A interface, muitas vezes escondida e de difícil acesso dentro dos carros, tem a forma de um trapézio com 16 entradas para conexões que dão suporte a implementação de protocolos na rede de comunicação do veículo, dentre eles, o protocolo CAN, conforme exibido na figura 6.

Figura 7 - Interface OBD II



Fonte: Retirado de OBD Station, 2021

Com a ISO 15765-4, determinou-se os requerimentos para a aplicação da rede CAN em OBD. Sua velocidade pode chegar até 500 kbps, onde são utilizados os pinos 6 (*CAN High*) e 14 (*CAN Low*) do conector SAE J2480, conforme Figura 7.

A rede CAN automotiva possui 10 serviços previstos para implementação, porém nem todos os sistemas suportam todos os serviços e cada veículo ou módulo de motor/transmissão automática pode ser configurado pelo fabricante para atender aos serviços suportados, alinhado com o requerimento de sua legislação.

Neste artigo, será abordado apenas o serviço 1, pois para o desenvolvimento da pesquisa não foi necessário a exploração dos demais serviços, uma vez que este módulo supre as necessidades da pesquisa.

O serviço 1 oferece os dados de informações correntes, relacionadas ao *powertrain*[1], onde é possível solicitar as informações através de um código de identificação, chamado *PID*

[1] Powertrain: Trem de força: São os componentes responsáveis por gerar potência para entregar às rodas, sendo composto pelo motor, transmissão, eixo cardã e diferencial.

– *Parameter Identification*. Como a implementação de serviços e informações podem ser customizadas dependendo do veículo, é possível descobrir quais informações estarão disponíveis através de uma requisição feita com o *PID 00* para o *MCU* do veículo. A lista de informações disponíveis para consulta, pode ser encontrada no livro "*THE CAR HACKER'S HANDBOOK*" [6]. Seguindo a proposta da pesquisa, os dados necessários para o desenvolvimento são: *VIN Code (Veiche Identification Number)*, a rotação do motor em RPM e a velocidade do veículo em Km/h. Todas informações necessárias podem ser acessadas por qualquer dispositivo que se comunique nos padrões estabelecidos pelo *CAN*, basta conhecer o *PID* correspondente a informação que deseja consultar.

3.2 Montagem Inicial do Hardware – IoT

A montagem do hardware foi iniciada com foco na principal funcionalidade em que ele foi planejado, coletar informações do veículo. As seções a seguir descrevem bem o processo de montagem do que será o primeiro protótipo.

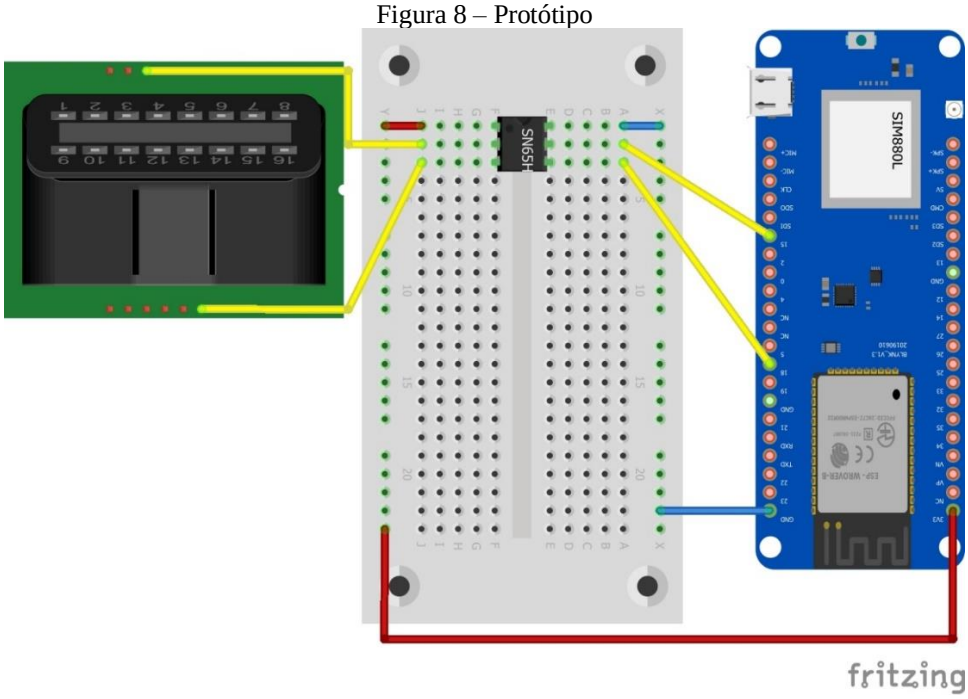
3.2.1 Prototipação

Na prototipação, a primeira etapa foi de separação de todos os hardwares necessários montagem do dispositivo IoT. Foram escolhidas as seguintes peças:

- 1x Mini Protoboard de 170 pontos
- 1x Cabo USB-C
- 1x LilyGo TTGO T-Call
- 1x SN65HVD230
- 8x Jumpers Macho – Fêmea

Com as peças em mãos, a etapa seguinte foi a criação do protótipo na ferramenta Fritzing. O protótipo final, que pode ser visto na Figura 7, foi utilizado para a montagem correta dos componentes na protoboard.

Figura 8 – Protótipo



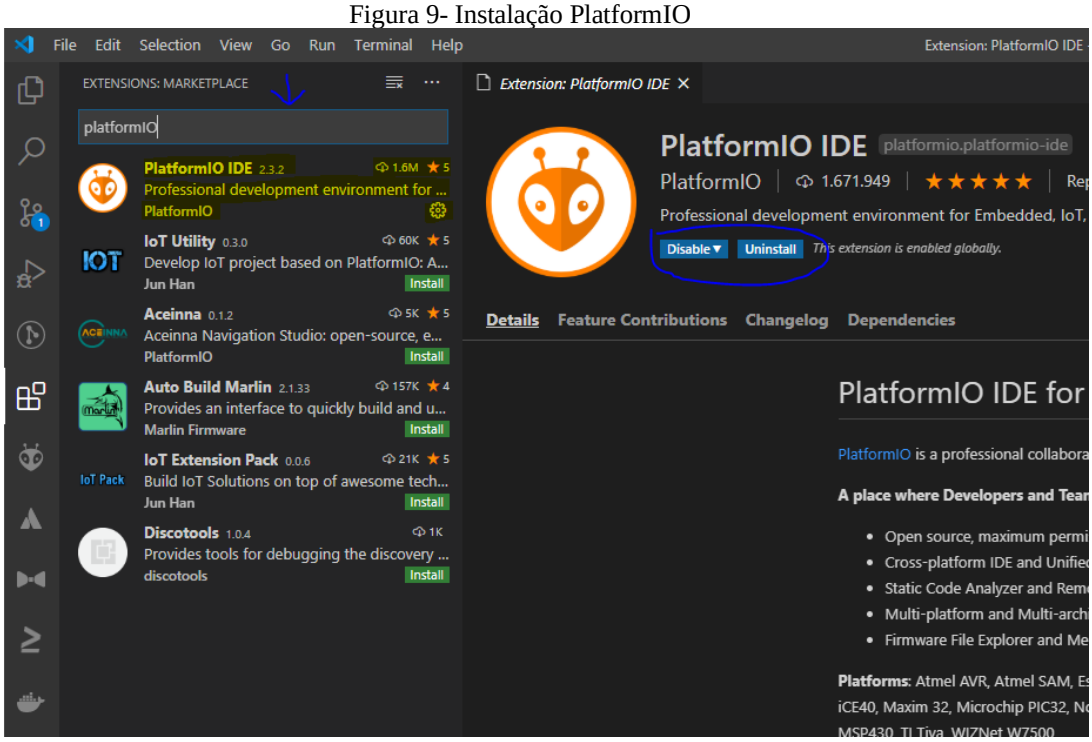
Fonte: Autoria Própria

3.2.2 Setup IDE

A IDE escolhida para o projeto foi o PlataformIO, uma plataforma de desenvolvimento de firmwares integrado ao editor de código fonte VS Code da Microsoft. Inicialmente foi necessário o download do VS Code, através do link https://code.visualstudio.com/download. A Microsoft disponibiliza o software em versões não instaláveis, ou seja, em pasta compactada no formato zip, basta extrair o arquivo para uma pasta e o software já está pronto para uso. A versão do software usada no desenvolvimento da pesquisa foi a V1.45.1. Com o software aberto, é necessário a configuração do PlatformIO, que pode ser encontrado na aba "Extensões", nos ícones da "*Activity Bar*" localizada no canto esquerdo do software, conforme exibido na figura 8. A instalação do IDE é feito de forma automática pelo VS Code. Após a instalação, é necessário fechar e abrir novamente o software. Ao reabrir e concluir a instalação, é possível ver na barra *"Activity Bar"* um novo ícone com o logotipo do PlatformIO, clicando nele, é possível ver a opção para criação de um novo projeto de firmware. Para criar um projeto é necessário seguir o seguinte caminho: PlatformIO (*"Activity Bar") > Projects > New Project*, conforme exibido na Figura 9. Ao abrir a tela de informações do novo projeto, é necessário selecionar a placa (*"Board"*) que será utilizada no desenvolvimento, no caso desta pesquisa, foi utilizado o *ESP32 WROVER Kit*, compatível com o *LilyGo TTGO T-Call*. Além da placa a ser selecionada, também é necessário selecionar o *Framework* que será utilizado durante o desenvolvimento.

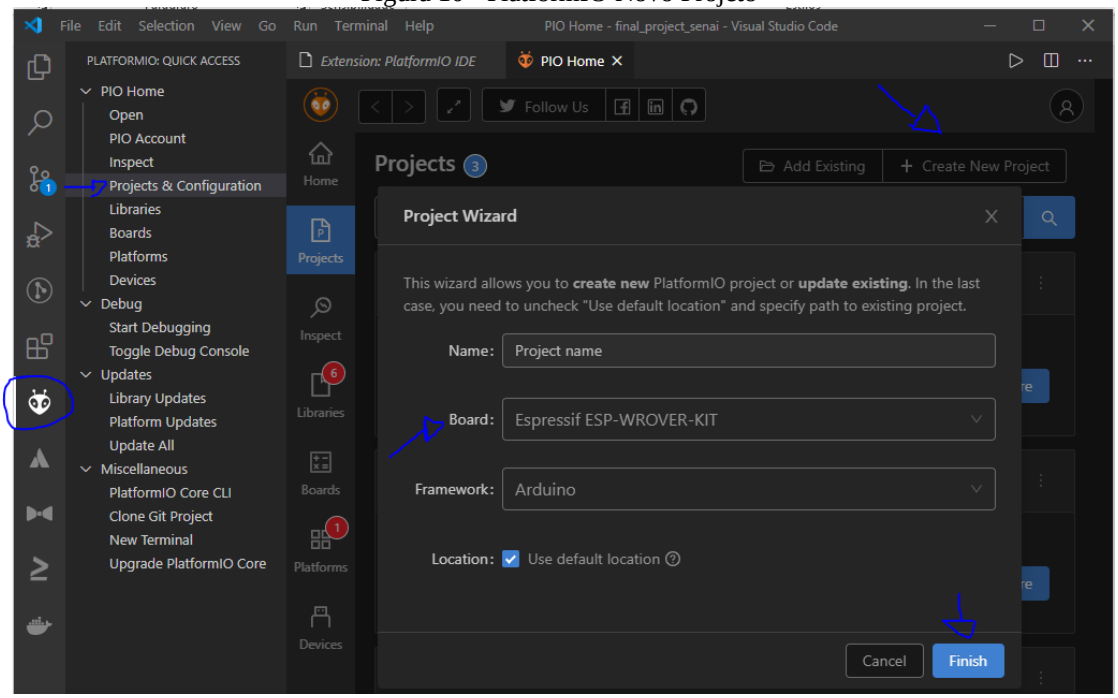
A ferramenta disponibiliza dois frameworks para o desenvolvimento de firmwares, quando a placa selecionada é o da família *ESP32*, são eles: Framework Arduino e *Framework Espressif*. O utilizado no desenvolvimento do firmware foi o Arduino, o mesmo utilizado nas classes do curso. Feito isso, o projeto está pronto para desenvolvimento.

Figura 9- Instalação PlatformIO



Fonte: Autoria Própria

Figura 10 - PlatformIO Novo Projeto



Fonte: Autoria Própria

3.3 Desenvolvimento do Firmware

O desenvolvimento do firmware foi segmentado em duas etapas, onde o primeiro protótipo teve como objetivo o entendimento da comunicação *CAN* entre o dispositivo e o carro. Após a compreensão sobre o funcionamento da rede, foi desenvolvido uma versão final para o protótipo do trabalho.

3.3.1 Primeiro protótipo

Inicialmente a proposta do firmware foi a comunicação feita entre o dispositivo IoT e o Carro, e para tanto, o código que foi implementado, visando atender estes requisitos, utilizou-se a biblioteca "*CAN*" do autor "*Sandeep Mistry*" na versão 0.3.1, hospedado no *GitHub* através do link: https://github.com/sandeepmistry/arduino-CAN. Nesta biblioteca há um código de exemplo, que foi utilizado no trabalho, chamado "*OBDII\EngineRPM.ino*". Ele resgata a informação do motor do veículo e devolve as informações de rotações por minuto do motor ao dispositivo IoT. A compilação do firmware e o upload dos binários para o dispositivo IoT pode ser feito através do botão de upload que fica na barra de status – "*Status Bar*,". O código utilizado no exemplo pode ser visto no Apêndice 1 – *Engine RPM*.

Para que o código funcionasse com o *LilyGo TTGO T-Call*, foi necessário a realização de um ajuste na biblioteca com um novo mapeamento dos pinos *RX* e *TX* padrões. A mudança é feita através do código "*#define DEFAULT_CAN_RX_PIN GPIO_NUM_15*" e "*#define DEFAULT_CAN_TX_PIN GPIO_NUM_18*", onde os pinos *RX* e *TX* são mapeados novamente para os pinos 15 e 18 respectivamente.

O código citado deve ser inserido logo em seguida da importação da biblioteca "*CAN.h*". Após essa modificação, o código foi testado e os resultados obtidos foram conforme o esperado. É possível ver através da instrução "*CAN.write(0x0C)*", dentro da rotina "*loop*" do código em apêndice, que o firmware solicita ao veículo o PID "0C", correspondente ao número de rotações do motor. Além deste exemplo, também foi utilizado o exemplo para captura do *VIN* do veículo. O exemplo tem o nome "*OBDII\VINReader.h*" e pode ser visto no Apêndice 2 – *VIN Reader*.

Igualmente ao primeiro exemplo, o código precisou de adaptação para os pinos *RX* e *TX* do *LilyGo TTGO T-Call*. Após a modificação, foi possível coletar o *VIN* do veículo corretamente conforme o esperado.

3.3.2 Versão final do protótipo

O desenvolvimento de uma versão final do firmware exigiu que diversas outras funcionalidades sejam implementadas no dispositivo, além de uma simples comunicação com

o veículo. Para isso, algumas capacidades foram incluídas no firmware, tais como a conexão com internet, gerenciamento de configuração e mensagens, monitoramento de logs, entre outras funções. É possível verificar a implementação de alguma destas funcionalidades através dos métodos "*setup*" e "*loop*" da classe principal do código-fonte, disponível no GitHub através do link https://github.com/rafargp/CarMonitor-Firmware.

3.3.2.1 *Setup*

O método "setup" é de implementação obrigatória para o desenvolvimento de firmware que tem o Arduino como framework de desenvolvimento. Ele é executado apenas uma vez durante a inicialização do firmware no dispositivo. O principal objetivo deste método é realizar etapas de configuração antes de executar qualquer outra tarefa. Logo abaixo está é possível ver um trecho do código fonte contendo apenas o método mencionado.

```
void setup()
{
  log_d("begin setup");
  Serial.begin(BAUD_RATE);
  log_d("setup serial baud rate %i", BAUD_RATE);

  log_d("begin setup CanBus");
  setupCanBus();
  log_d("begin setup CanBus - OK");

  log_d("setup OLED Screen");
  if (!setupOled()) ESP.restart();
  log_d("setup OLED Screen Completed");

  printOledTextSingleLine("Iniciando Sistema");

  log_d("configuring file system");
  if (!setupFile()) ESP.restart();
  log_d("configuration file system completed");

  log_d("getting Configuration File");
  getConfig();
  log_d("getting Configuration File - OK");

  log_d("configuring GSM");
  setup_gsm();
  log_d("configuration GSM - OK");
```

```
    log_d("setup RTC");
    if (!setupRTC()) ESP.restart();
    log_d("setup RTC Completed");

    log_d("update Configuration");
    updateConfiguration(false);
    log_d("update Configuration - OK");

    log_d("checking for updates");
    check_update = config["check_update"].as<bool>();
    if(check_update) checkUpdate("");
    log_d("checking for updates - OK");

    log_d("setup MQTT");
    setupMQTT();
    log_d("setup MQTT - OK");

    log_d("setting variables");
    healthInterval = config["health_interval"].as<long>();
    sensorInterval = config["sensor_interval"].as<long>();
    log_d("setting variables - OK");

    disableCore0WDT();

    xTaskCreatePinnedToCore(sendMQTTData, "Send MQTT Data", 5000, NULL, 0, NULL, 0);

    log_w("begin setup complete");
    delay(100);
}
```

Com o intuito de esclarecer as funcionalidades do código-fonte acima, esta sessão está dedicada a explicação dos métodos. Como uma forma de melhorar a depuração do código-fonte, em praticamente todos os métodos citados, eles estão circundados pelo método "*log_d()*", que é apenas uma saída de texto para a comunicação serial, com o objetivo de facilitar o entendimento para o desenvolvedor que estiver fazendo a inspeção do dispositivo.

O primeiro método é o "*setupCanBus()*", responsável por garantir a inicialização do módulo SN65HVD230 para que haja comunicação entre o veículo e o dispositivo.

O segundo método "*setupOLED()*" se trata da inicialização do display SSD1306, onde é estabelecido uma conexão com o display e também a parametrização dos tamanhos e cor do texto a ser exibido na tela.

Os métodos "*setupFile()*" e "*getConfig()*" garantem a correta inicialização e carregamento das configurações em memória que ficam armazenadas através de arquivos. O sistema gerenciador destes arquivos é o SPIFFS (*SPI File Flash System*).

A rede GSM é configurada através do método "*setupGSM()*", onde é feito a parametrização dos pinos que ligam o módulo SIM800L ao ESP32, além de inicializar o modem e estabelecer uma conexão GPRS - *General Packet Radio Services* com a rede operadora.

A data e hora são inicializadas no método "*setupRTC()*", onde é estabelecido uma comunicação com o módulo RTC DS3231. Apesar deste módulo possuir habilidade para reter a informação de data e hora, sempre que a rede GSM é estabelecida, é feito uma requisição de data e hora da rede para atualização destas informações no RTC DS 3231. Isso acontece devido ao risco da falta de energia na bateria no módulo ou até mesmo a ausência dela.

Apesar do armazenamento das configurações do sistema estarem disponível no arquivo, também é possível resgatar as configurações no servidor, através de API, e para isso, é utilizado o método "*updateConfiguration()*".

Assim como é possível atualizar as configurações do sistemas através de API, o método "*checkUpdate()*" permite ao dispositivo validar se há alguma versão de firmware mais recente no servidor, e havendo, realiza essa atualização.

Após ter a conexão GPRS estabelecida e todas as configurações carregadas, o método "*setupMQTT()*" inicializa a comunicação MQTT com o servidor, onde é feito o login no Broker para publicação e consumo de mensagens, além de enviar as informações do dispositivo e do veículo para registro.

Os últimos dois métodos "*disableCore0WDT()*" e "*xTaskCreatePinnedToCore()*" são utilizados para a parametrização de uma nova linha de execução independente no ESP32. A parametrização aponta para uma tarefa que irá executar independente do que estiver sendo executado na outra linha de processamento. Desta forma foi possível separar a coleta de dados do veículo, do envio de dados para processamento.

3.3.2.2 Loop

Da mesma forma que o método "*setup*" é de implementação obrigatória, o método "*loop*" também é, quando se utiliza o Arduino como framework de desenvolvimento. Ele é

executado infinitamente até que o dispositivo seja desligado ou programaticamente, através de métodos que interrompam as atividades do microcontrolador. O principal objetivo deste método é executar sua função constantemente. Abaixo é possível ver um trecho do código fonte com um recorte apenas do método loop.

```
void loop()
{
   unsigned long currentMillis = millis();

   if (currentMillis - healthPreviousMillis >= healthInterval)
   {
      sendHealthStatus();
      healthPreviousMillis = currentMillis;
   }

   if (currentMillis - sensorPreviousMillis >= sensorInterval){
      getSensorData();
      sensorPreviousMillis = currentMillis;
   }
   String text = getTimeStampString() + "\n\n";
   text += "Pacotes Enviados: " + String(dataSent) + "\n";
   text += "Pacotes Perdidos: " + String(dataLoss) + "\n\n";
   text += "Device Registrado: " + String(DEVICE_ID != "" ? "S":"N") + "\n";
   text += "Carro Registrado: " + String(TRAVEL_ID != "" ? "S":"N");
   printOledTextSingleLine(text,false);
}
```

Devido a sua característica de repetição, a implementação deste método foi feita com validações de tempo, onde cada comando é executado após um determinado tempo decorrido. Esse controle foi feito por duas variáveis, as quais "*healthInterval*" é responsável pelo controle do momento exato em que o dispositivo coleta dados de saúde do seu funcionamento, e a variável "*sensorInterval*", que é responsável por controlar o momento exato de coletar os dados do veículo e armazenar. Apesar dos controles mencionados acima, a atualização de dados na tela OLED, é feito toda vez que o método é executado.

3.4 Montagem do Hardware – Raspberry

A utilização do Raspberry PI 4, ao invés da nuvem, se deu devido aos custos de utilização dos recursos disponíveis na nuvem. A escolha de usar o Raspberry como uma "nuvem privada", implica no aumento de trabalho em realizar a configuração e manutenção deste Hardware periodicamente.

Por padrão, o Raspberry não possui nenhum disco para armazenamento de um sistema operacional, portanto, é necessário a utilização de um cartão SD com um S.O instalado e que seja compatível com a arquitetura ARM.

3.4.1 Instalação do Linux

Para esta pesquisa, foi escolhido o sistema operacional Linux, devido a sua compatibilidade com o Hardware e vasta comunidade de desenvolvedores que contribui para atualizações constantes do sistema, além de facilitar o suporte em caso de dúvidas nos sistemas. A instalação do sistema operacional no cartão SD é feita separadamente do hardware, em outra máquina, uma vez que é necessário fazer o Raspberry não tenha ferramentas de boot para fazer o deploy da imagem no SD diretamente do Raspberry.

A primeira etapa da instalação do SO é o download da imagem da distribuição do Linux escolhida, que para esta pesquisa, foi o "*Kali Linux*", que pode ser encontrado em https://images.kali.org/arm-images/kali-linux-2021.1-rpi4-nexmon-64.img.xz. Com a imagem do SO baixado, foi necessário utilizar o software Balena Etcher para realizar o deploy do S.O no cartão SD conectado ao computador. Estando completo o processo de deploy no Balena Etcher, remove-se o cartão SD com segurança do computador, insere o SD no Raspberry Pi 4, o sistema operacional já deve funcionar corretamente. Ao iniciar o sistema operacional, é necessário a atualização dos repositórios através dos comandos listados abaixo:

```
apt update -y

apt upgrade -y
```

Estes comandos fazem com que todas as referências de softwares sejam atualizadas, permitindo que novos softwares possam ser instalados com facilidade.

3.4.2 Setup Node-Red

A utilização do Node-Red foi baseada na praticidade que o software disponibiliza para a criação de novas API e canais de comunicação com dispositivos IoT. A instalação do software é feita através dos seguintes comandos no terminal do Linux:

```
bash<(curl -sL https://raw.githubusercontent.com/node-red/linux-installers/master/deb/update-nodejs-and-nodered)
```

O comando citado acima faz uma requisição para o link, que retorna um script a ser executado. Após a execução completa do script, o sistema já fica pronto para uso na porta 1880

por padrão. Diversas configurações podem ser alteradas através do arquivo de configuração do software, porém, para esta pesquisa, foram utilizados os valores padrões estabelecidos pelo provedor do software.

3.4.3 Setup MQTT Mosquitto Broker

A escolha do Broker MQTT Mosquitto foi feita considerando a praticidade de sua instalação, facilidade de utilização e popularidade na comunidade IoT. A instalação do broker Mosquitto MQTT também é feita através de comandos do Linux, conforme orientação do fornecedor. Abaixo estão os comandos executados:

```
sudo apt-get update

sudo apt-get install mosquitto -y
```

Após a execução do script, o software já está funcional e pronto para uso através da porta 1883. Diversas configurações podem ser alteradas através do arquivo de configuração do software, porém, para esta pesquisa, foram utilizados os valores padrões estabelecidos pelo provedor do software.

3.4.4 Setup MongoDB

O uso do MongoDB como banco de dados foi baseada na facilidade de manipulação dos dados, velocidade de resposta e modelo de dados não relacional, que garante a velocidade esperada para o projeto. A instalação do software é feita através de linhas de comando no Linux, conforme sequência abaixo:

```
sudo apt-get install gnupg

wget -qO - https://www.mongodb.org/static/pgp/server-4.4.asc | sudo apt-key add –

echo "deb http://repo.mongodb.org/apt/debian stretch/mongodb-org/4.4 main" | sudo tee /etc/apt/sources.list.d/mongodb-org-4.4.list

sudo apt-get update

sudo apt-get install -y mongodb-org

sudo systemctl enable mongod

sudo systemctl start mongod

sudo systemctl daemon-reload
```

Após a execução do script, o software já está funcional e pronto para uso através da porta 27017. Diversas configurações podem ser alteradas através do arquivo de configuração do software, porém, para esta pesquisa, foram utilizados os valores padrões estabelecidos pelo provedor do software.

3.4.5 NO-IP

Devido a escolha de utilização do Raspberry PI como central de processamento de dados, ao invés de utilizar serviços em Cloud, se faz necessário a utilização de um provedor de *DDNS - Dynamic Domain Name System* para que o dispositivo IoT tenha como referência um endereço fixo (*DNS - Domain Name System*) e não mais um IP, que pode alterar a qualquer momento. Antes da instalação do software, é necessário criar uma conta no site, através do link: https://www.noip.com/pt-BR/sign-up. Através do site, é possível escolher um DNS para sua utilização. A instalação do software é feita através de linhas de comando no Linux, conforme sequência abaixo:

```
cd /usr/local/src

wget http://www.no-ip.com/client/linux/noip-duc-linux.tar.gz

tar xzf noip-duc-linux.tar.gz

cd noip-2.1.9-1

make

make install

/usr/local/bin/noip2 -C
```

Após a execução do último comando, o software irá requisitar os dados de login, além de outras informações relacionadas ao funcionamento do software. Ao terminar as configurações, o software deve estar funcionando corretamente, redirecionando o DNS para o IP externo da rede. Diversas configurações podem ser alteradas através do arquivo de configuração do software, porém, para esta pesquisa, foram utilizados os valores padrões estabelecidos pelo provedor do software.

3.4.6 Arquitetura do projeto

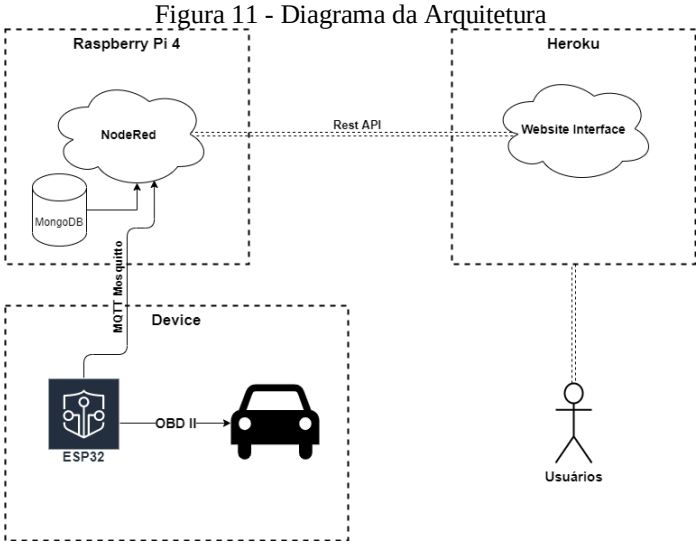
A arquitetura do projeto foi dividida em três grandes partes, conforme é possível ver na figura 10, onde, cada parte tem seu objetivo definido conforme listado abaixo:

Device IoT: responsável pela coleta de dados do veículo e envio de dados para o servidor;

Raspberry PI: responsável pelo recebimento e processamento dos dados;

Interface Web: responsável por disponibilizar as informações processadas ao usuário de forma amigável.

Figura 11 - Diagrama da Arquitetura



Fonte: Autoria Própria

3.5 Desenvolvimento do Software

Conforme descrito na figura 10, a arquitetura foi dividida em três partes, porém, apenas em duas delas foi necessário o desenvolvimento de software, são elas: Node-Red e Website.

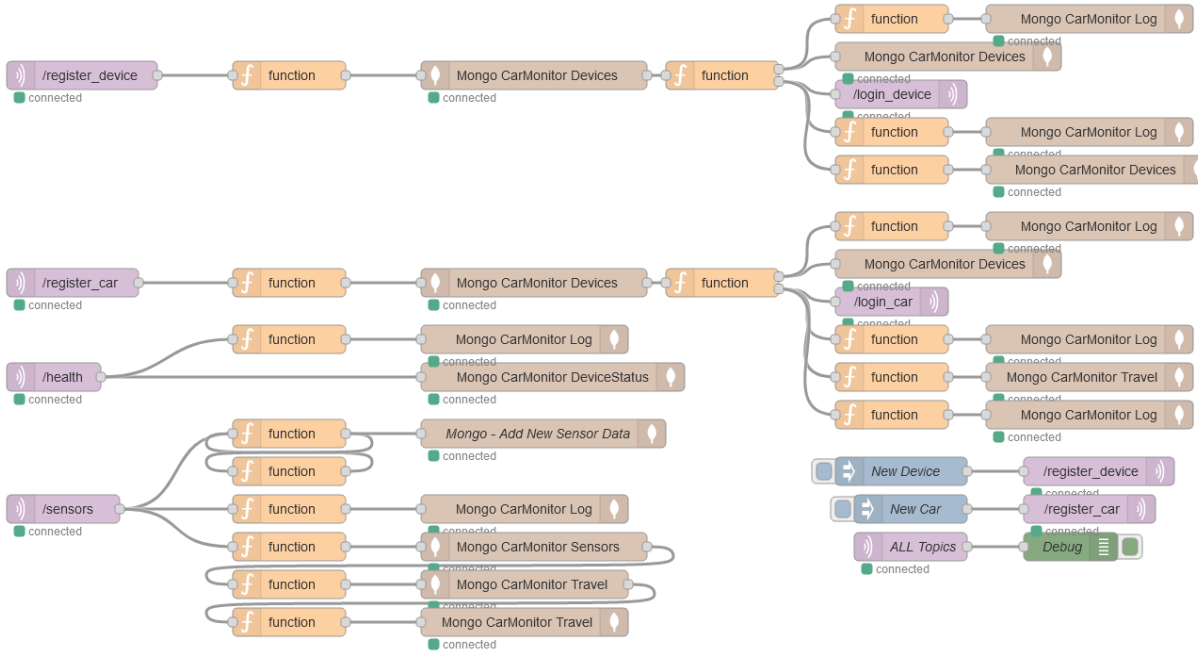
3.5.1 Node-Red

O Node-Red permite ao usuário implementar rapidamente funções que seriam trabalhosas se fosses feitas de forma manual, através da programação. Essa facilidade de implementação se dá através dos chamados “Nós” da ferramenta, que nada mais são que módulos com funções pré-programadas, como operadores de banco de dados, sockets HTTP e MQTT, dashboard de dados e muitos outros “Nós” que permitem a customização de sua função a nível de programação. Através destes nós foi construída a todas as funções que suportam a comunicação e o processamento dos dados coletados pelo dispositivo IoT.

O desenvolvimento dos nós foram separados em duas partes, sendo a primeira responsável pelo recebimento dos dados e a outra, uma API para consumo dos dados coletados. É possível ver nas Figuras 12 e 13 a separação dos nós.

3.5.1.1 Processamento dos Dados

Figura 12 - Nós Node-Red - Parte I



Fonte: Autoria Própria

Na Figura 11 é possível ver que foram utilizados quatro nós (roxos) como portas de entrada para a recepção dos dados do dispositivo IoT. Abaixo está a descrição de cada nó:

O Nó "*/register_device*" é responsável por receber os dados do dispositivo e criar um registro dentro do banco de dados, caso não exista. Existindo o registro, o nó apenas devolve os valores conforme na Tabela 4. Para o processamento, a entrada de dados esperado pelo nó está conforme tabela 3.

Tabela 3 - Node-Red - *Register Device* - Dados de Entrada

Campo	**Descrição**	**Tipo**
IMEI	IMEI da placa GSM SIM800L	Texto
CCID	CCID da placa GSM SIM800L	Texto
IMSI	IMSI da placa GSM SIM800L	Texto
Operator	Operadora GSM do SIM Card	Texto
Board	Placa Utilizada (Fixo: ESP32)	Texto
Version	Versão do Firmware	Decimal
Unix_Time	Data e Hora Atual (Formato Unix)	Inteiro

Fonte: Autoria Própria

Ao processar os dados e havendo sucesso na operação, o nó deve retornar os dados conforme definido na Tabela 4. Havendo falha, o nó é interrompido e não retorna nenhuma resposta.

Tabela 4 - Node-Red - *Register Device* - Dados de Saída

Campo	**Descrição**	**Tipo**
ID	Identificação Gerada no Sistema	Texto

Fonte: Autoria Própria

O nó "*/register_car*" é responsável por receber os dados do veículo e criar um registro dentro do banco de dados, caso não exista. Existindo o registro, o nó é interrompido. Para o processamento, a entrada de dados esperado pelo nó está conforme tabela 5.

Tabela 5 - Node-Red - *Register Car* - Entrada de Dados

Campo	**Descrição**	**Tipo**
DeviceID	Código do Dispositivo (Vide Tabela 4)	Texto
CarVIN	VIN do Carro	Texto
Unix_Time	Data e Hora Atual (Formato Unix)	Inteiro

Fonte: Autoria Própria

Ao processar os dados e havendo sucesso na operação, o nó deve retornar os dados conforme definido na Tabela 6.

Tabela 6 - Node-Red – *Register Car* - Saída de Dados

Campo	**Descrição**	**Tipo**
Travel_ID	Identificação de Viagem Iniciada	Texto

Fonte: Autoria Própria

O nó "Health" é responsável por receber os dados do dispositivo que indicam a saúde do dispositivo. Para o processamento, a entrada de dados esperado pelo nó está conforme tabela 6.

Tabela 7 - Node-Red - *Health* - Entrada de Dados

Campo	**Descrição**	**Tipo**
Device_ID	Código do Dispositivo (Vide Tabela 4)	Texto
IP	IP da Conexão GSM	Texto
Signal	Força do Sinal GSM	Inteiro
GSM_Date	Data na Rede GSM	Texto
GSM_Time	Hora na Rede GSM	Texto
Location_LO	Longitude da Triangulação GSM	Decimal
Location_LA	Latitude da Triangulação GSM	Decimal
Location_Accuracy	Precisão da Geolocalização (em metros)	Decimal
Data_Loss	Quantidade de Mensagens Perdidas	Inteiro
Data_Sent	Quantidade de Mensagens Enviadas	Inteiro
Unix_Time	Data e Hora Atual (Formato Unix)	Inteiro

Fonte: Autoria Própria

O nó "*/Sensors*" é responsável por receber os dados dos sensores do veículo coletados pelo dispositivo. Para o processamento, a entrada de dados esperado pelo nó está conforme tabela 8.

Tabela 8 - Node-Red - *Sensors* - Entrada de Dados

Campo	**Descrição**	**Tipo**
Travel_ID	Código da Viagem (Vide Tabela 6)	Texto
Sensor_PID (Array)	PID do Processo	Texto
Sensor_Value (Array)	Valor do Sensor	Texto
Unix_Time	Data e Hora Atual (Formato Unix)	Inteiro

Fonte: Autoria Própria

Ao processar os dados e havendo sucesso na operação, o nó deve calcular uma pontuação para esta condução, baseando-se na leitura dos sensores e no perfil do veículo.

3.5.1.2 Cálculo de Condução

O cálculo da pontuação do condutor foi implementado no desenvolvimento do software devido a modelagem de dados estar concluída apenas nesta etapa. A implementação da pontuação se deu através do nós do Node-Red, onde é pontuado cada registro de leitura dos sensores do veículo, por exemplo, para cada medição de rotação do motor ou velocidade do veículo, é calculado uma pontuação de 0 a 100, onde 0 é considerado um comportamento ruim e 100 é considerado bom. Para cada sensor há um critério diferente:

- Rotações do Motor

  As rotações do motor são avaliadas de acordo com o perfil atrelado ao veículo, onde é especificado a rotação máxima do veículo, conforme manual de instruções da montadora. Caso o valor do sensor seja maior que o informado no perfil do veículo, a pontuação deste dado será 0, caso contrário, será 100;

- Velocidade do Veículo

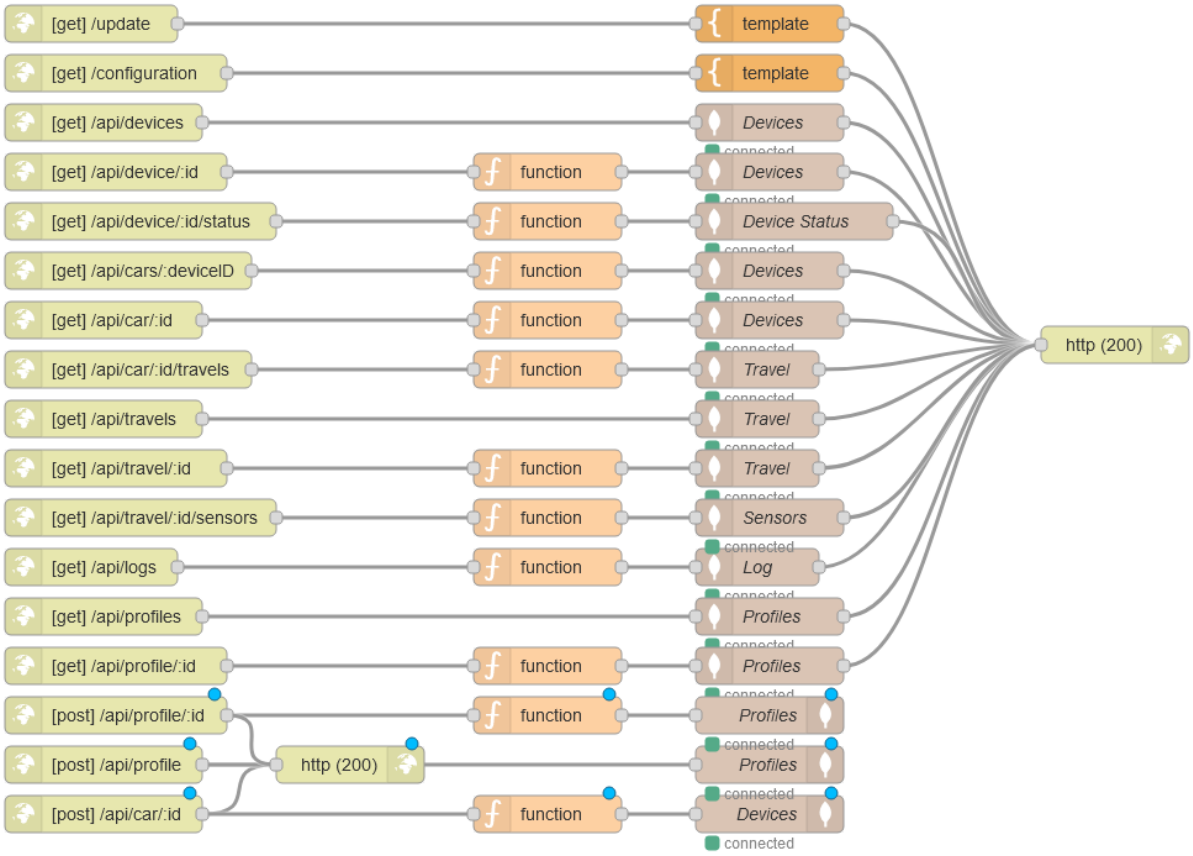
  A velocidade do veículo não pode estar associada a nenhuma medida do motor, portanto, é utilizado o limite máximo de velocidade permitido nas vias, onde neste trabalho, foi considerado apenas os limites do estado de São Paulo, logo, se o valor do sensor for maior que 120 km/h, a pontuação deste dado será 0, caso contrário, será 100.

A informação que é visualizada pelo condutor, na plataforma web do trabalho, considera uma média de todas as pontuações dos sensores da viagem em que o usuário seleciona.

3.5.2 API

Segundo a definição da empresa RedHat, “API é um conjunto de definições e protocolos usados no desenvolvimento e na integração de software de aplicações”. O objetivo de uma API está explicita no seu próprio nome, que é um acrônimo em inglês para interface de programação de aplicações. A criação de uma interface é extremamente útil quando se faz necessário duas aplicações se comunicarem entre si de forma rápida e segura. O Node-Red, utilizado no trabalho, facilita a implementação de uma API através de nós, conforme figura a seguir.

Figura 13 - Nós Node-Red – Parte II



Fonte: Autoria Própria

Na Figura acima é possível ver que a API foi construída baseando-se no conjunto de dados do projeto. Abaixo está descrito de forma detalhada o fluxo de dados das funções disponíveis na API. Abaixo temos uma tabela descritiva com todos os endpoints da API e suas funções.

Tabela 9 - Descritivo das API

Endpoint	**Método HTTP**	**Descrição**
/Update	GET	Responsável por informar última versão do firmware disponível para download. Esta API não possui dados de entrada, apenas saída no formato JSON, conforme tabela em apêndice.
/Configuration	GET	Responsável por enviar as configurações padrões estabelecidas pelo projeto. Esta API não possui dados de entrada, apenas saída no formato JSON, conforme tabela em apêndice.
/api/devices	GET	Responsável por listar os dispositivos registrados no banco de dados. Esta API não possui dados de entrada, apenas um array de saída no formato JSON, conforme tabela em apêndice.
/api/device/{id}	GET	Responsável por listar o dispositivo registrados no banco de dados, conforme ID informado no Endpoint. Esta API possui dados de entrada e saída, conforme tabelas em apêndice. Ambos retornos são no formato JSON.
/api/device/{id}/status	GET	Responsável por listar o status do dispositivo registrado no banco de dados, conforme ID informado no Endpoint. Esta API possui dados de entrada e saída, conforme tabelas em apêndice. Ambos retornos são no formato JSON.

/api/cars/{deviceId}	GET	Responsável por listar todos os veículos do dispositivo registrados no banco de dados, conforme ID informado no Endpoint. Esta API possui dados de entrada e saída, conforme tabelas em apêndice. Ambos retornos são no formato JSON.
/api/car/{id}	GET	Responsável por listar o veículo registrado no banco de dados, conforme ID informado no Endpoint. Esta API possui dados de entrada e saída, conforme tabelas em apêndice. Ambos retornos são no formato JSON.
/api/car/{id}/travels	GET	Responsável por listar as viagens do veículo registrado no banco de dados, conforme ID informado no Endpoint. Esta API possui dados de entrada e saída, conforme tabelas em apêndice. Ambos retornos são no formato JSON.
/api/travels	GET	Responsável por listar as viagens registradas no banco de dados. Esta API não possui dados de entrada, apenas um array de saída no formato JSON, conforme tabela em apêndice.
/api/travel/{id}	GET	Responsável por listar a viagem registrada no banco de dados, conforme ID informado no Endpoint. Esta API possui dados de entrada e saída, conforme tabelas em apêndice. Ambos retornos são no formato JSON.
/api/travel/{id}/sensors	GET	Responsável por listar os sensores monitorados durante a viagem registrada no banco de dados, conforme ID informado no Endpoint. Esta API possui dados de entrada e saída, conforme tabelas em apêndice. Ambos retornos são no formato JSON.
/api/logs	GET	Responsável por listar os logs registrados no banco de dados. Esta API não possui dados de entrada, apenas um array de saída no formato JSON, conforme tabela em apêndice.
/api/profiles	GET	Responsável por listar os perfis de veículos registrados no banco de dados. Esta API não possui dados de entrada, apenas um array de saída no formato JSON, conforme tabela em apêndice.
/api/profile/{id}	GET	Responsável por listar o perfil registrado no banco de dados, conforme ID informado no Endpoint. Esta API possui dados de entrada e saída, conforme tabelas em apêndice. Ambos retornos são no formato JSON.
/api/profile	POST	Responsável por registrar um perfil no banco de dados. Esta API não possui dados de saída, apenas uma entrada no formato JSON, conforme tabela em apêndice.
/api/profile/{id}	POST	Responsável por alterar o perfil registrado no banco de dados conforme ID informado no Endpoint. Esta API não possui dados de saída, apenas uma entrada no formato JSON, conforme tabela em apêndice.
/api/car/{id}	POST	Responsável por alterar o veículo registrado no banco de dados conforme ID informado no Endpoint. Esta API não possui dados de saída, apenas uma entrada no formato JSON, conforme tabela em apêndice.

Fonte: Autoria Própria

3.5.3 Website

O desenvolvimento do website tem como principal função a interação do usuário com a plataforma. Na construção da ferramenta, foi utilizado algumas bibliotecas que facilitam o desenvolvimento de software, tais como jQuery, Bootstrap e HighCharts, conforme descrito a seguir.

3.5.3.1 jQuery

jQuery é uma biblioteca popular do JavaScript que foi criada por John Resig em 2006 com o propósito de facilitar a vida dos desenvolvedores que usam JavaScript nos seus sites. Não é uma linguagem de programação separada, funciona em conjunto com o JavaScript

3.5.3.2 Bootstrap

Bootstrap é um framework para interface e de código-aberto que foi inicialmente criado por Mark Otto e Jacob Thornton para o desenvolvimento web mais rápido e prático.

Ele contém templates baseados em HTML e CSS para várias funções e componentes. Como por exemplo, navegação, sistema de grades, carrosséis de imagens e botões.

3.5.3.3 HighCharts

O HighCharts é uma biblioteca escrita em JavaScript para geração de gráficos. Ele é baseado em SVG e vem sido desenvolvido desde 2009 pela empresa Highsoft Solutions

3.5.3.4 API

Para que o website pudesse interagir com a API criada no projeto, foi necessário a criação de uma biblioteca em JavaScript que implementasse os endpoints disponíveis, conforme item 3.5.1.2. O código desenvolvido pode ser encontrado no Apêndice C.

3.5.4 Setup Heroku

O Heroku é uma plataforma online que facilita a hospedagem de sites e por este motivo foi utilizada no trabalho. Para que se tenha acesso a plataforma, é necessário estar logado no site através de um credenciais que pode ser criadas no link https://signup.heroku.com/login. A interface do site é amigável e intuitiva. Para fazer o deploy da aplicação no Heroku, é necessário que o código-fonte esteja em um repositório compatível com a plataforma, no caso deste trabalho, foi utilizado o GitHub, com o repositório https://github.com/rafargp/CarMonitor-Web. Ao criar uma aplicação no Heroku, a plataforma automaticamente solicita a conexão com um repositório de código para listar todos os projetos contidos no repositório. Feito a conexão, a plataforma exige a escolha de um projeto específico, e automaticamente identifica a linguagem utilizada no projeto, facilitando o *deploy* da aplicação. No Website, não foram utilizadas linguagens de programação que necessitam de servidor para serem executadas, porém, para a exposição do site em um servidor é necessário utilizar um software compatível com alguma linguagem de programação, e para isso, foi criado o arquivo "*index.php*" onde a plataforma identifica o projeto e consegue realizar o deploy e a exposição na internet. O conteúdo do arquivo "*index.php*" é bem simples, possuindo apenas uma linha com o seguinte

conteúdo: “*<?php include_once("index.html"); ?>*”. O arquivo nada mais faz do que indicar a página principal, que pode ser acessada no seguinte link: https://senai-iot.herokuapp.com/

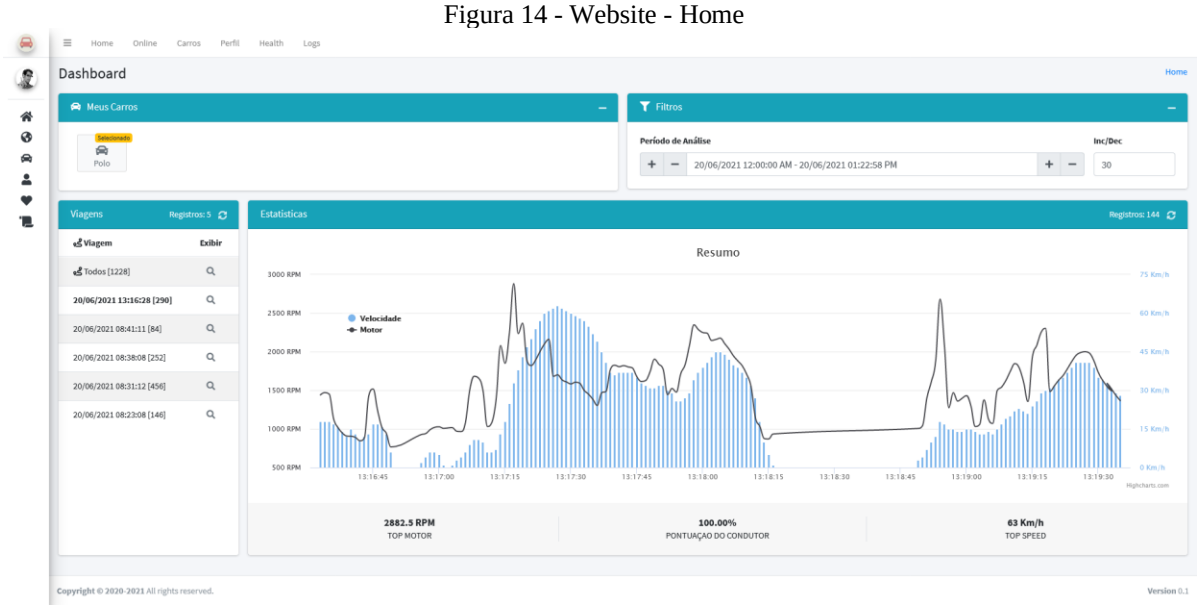
3.6 Interface com o usuário

O website foi construído com o objetivo principal de exibir as informações do usuário de uma forma clara e intuitiva. Ele é composto de seis páginas com funcionalidades especificas, são elas:

- *Home*

Na Página “*Home*” o usuário tem a possibilidade de visualizar sua pontuação como condutor além de poder ver os dados do veículo, que foram coletados e processados, através de tabelas e gráficos. É necessário escolher um veículo para se ter acesso aos dados de viagens. As viagens são filtradas por data, que podem ser customizadas pelo usuário, conforme é exibido na figura abaixo.

Figura 14 - Website - Home

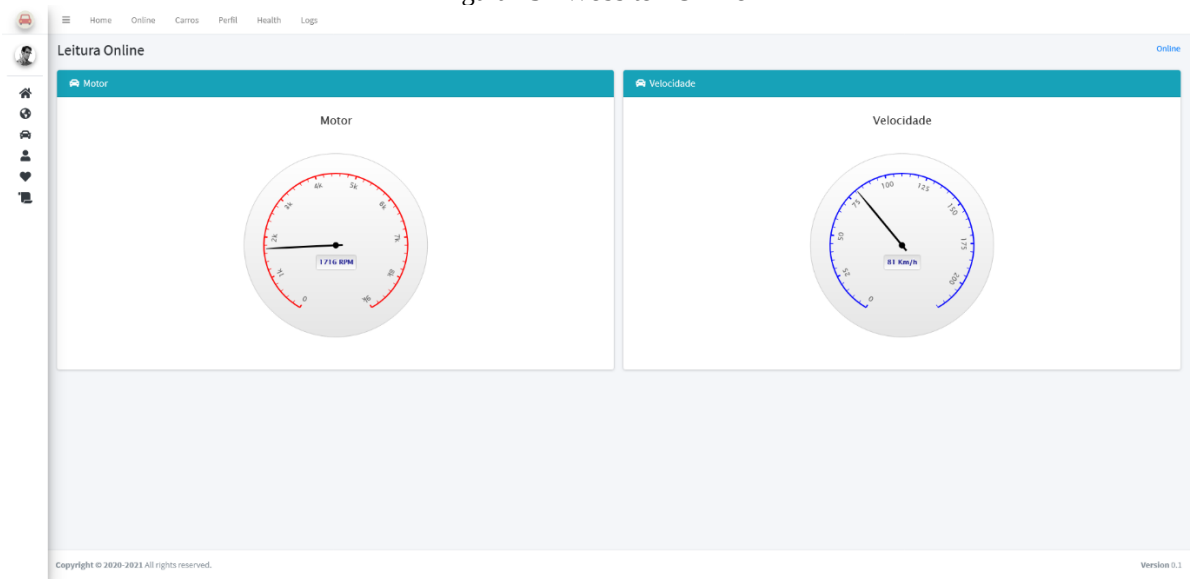


Fonte: Autoria Própria

- *Online*

A página “*Online*” reflete o momento atual do veículo em movimento, é possível acompanhar os valores de RPM do motor e Velocidade do atual conforme é possível ver na figura abaixo.

Figura 15 - Website - Online

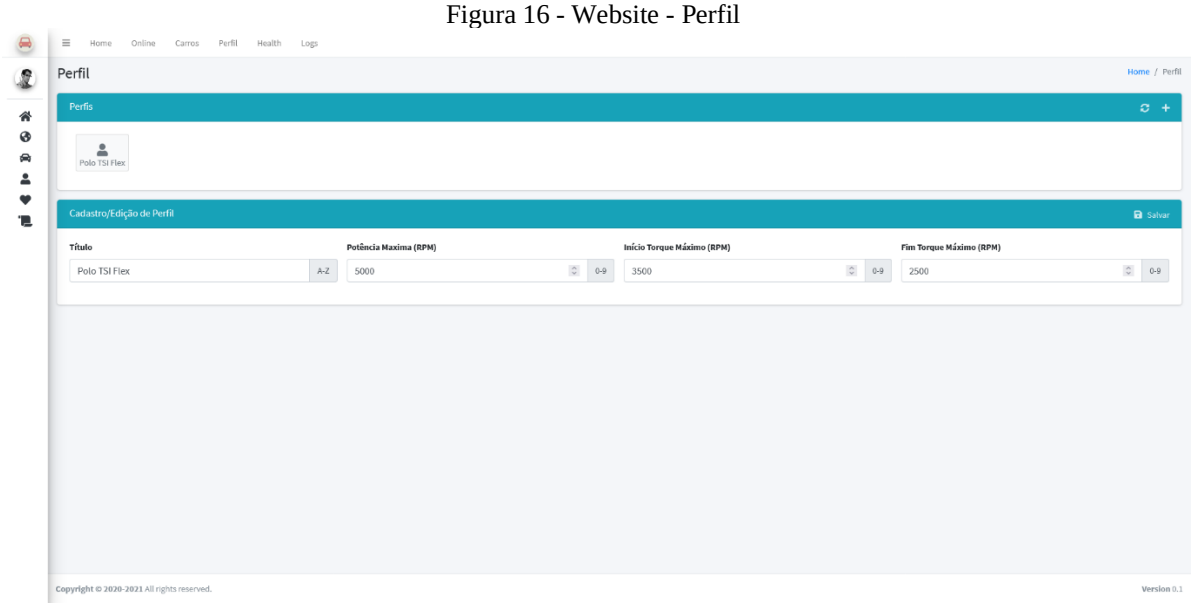


Fonte: Autoria Própria

- Perfil

A pontuação do condutor é calculada com base no perfil atrelado ao veículo, onde são definidos os parâmetros que estabelecem os limites do motor. Na página “Perfil”, é possível cadastrar diversos perfis de automóveis conforme demonstrado abaixo.

Figura 16 - Website - Perfil

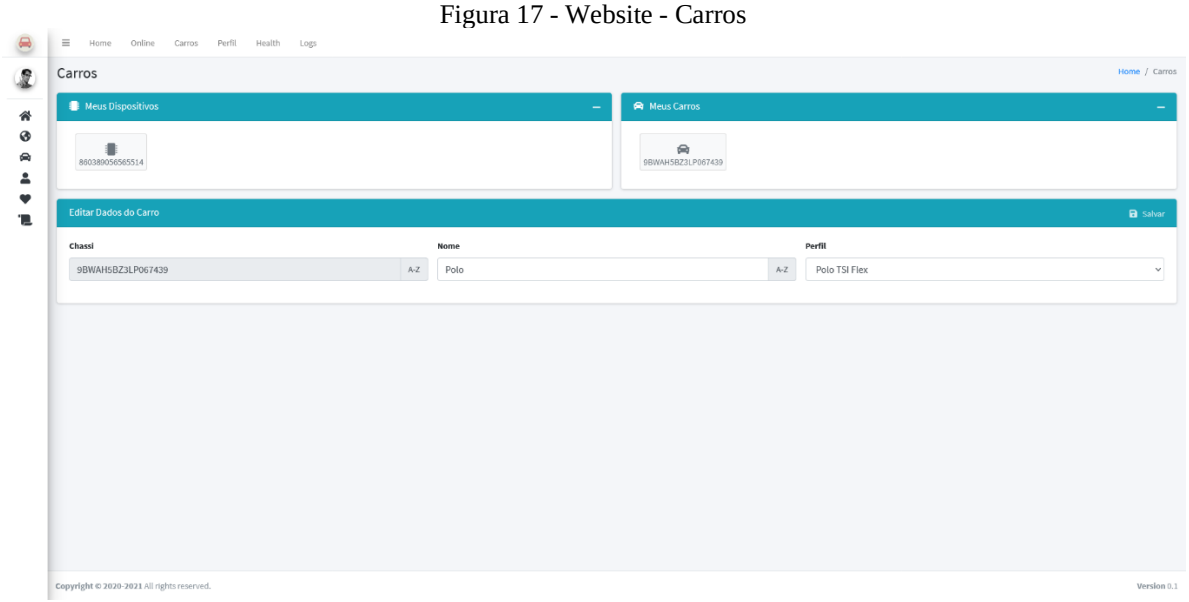


Fonte: Autoria Própria

- Carros

A listagem dos veículos cadastrados, pode ser acessada na página “Carros”. O cadastro dos veículos pode ser editado nessa página, podendo customizar o nome do veículo e o perfil de condução atrelado a ele, conforme é possível ver na imagem abaixo.

Figura 17 - Website - Carros

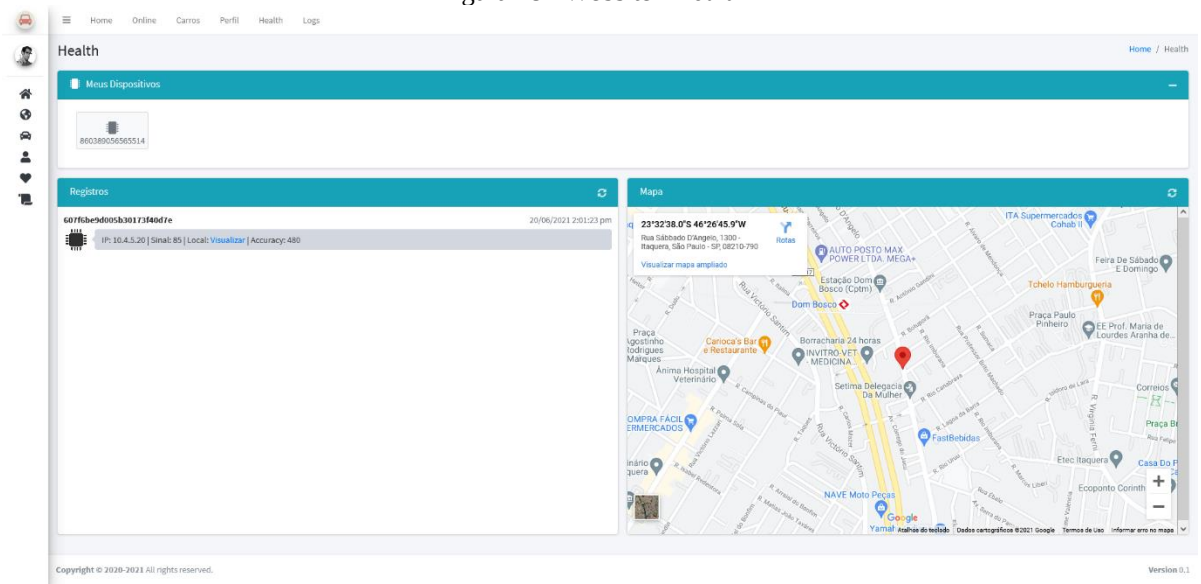


Fonte: Autoria Própria

- *Health*

Durante o funcionamento do dispositivo IoT, ele envia informações sobre a saúde do dispositivo com dados de localização, rede GSM, qualidade do sinal entre outros, esses dados podem ser visualizados na página “Health”, como é possível na imagem abaixo.

Figura 18 - Website - *Health*

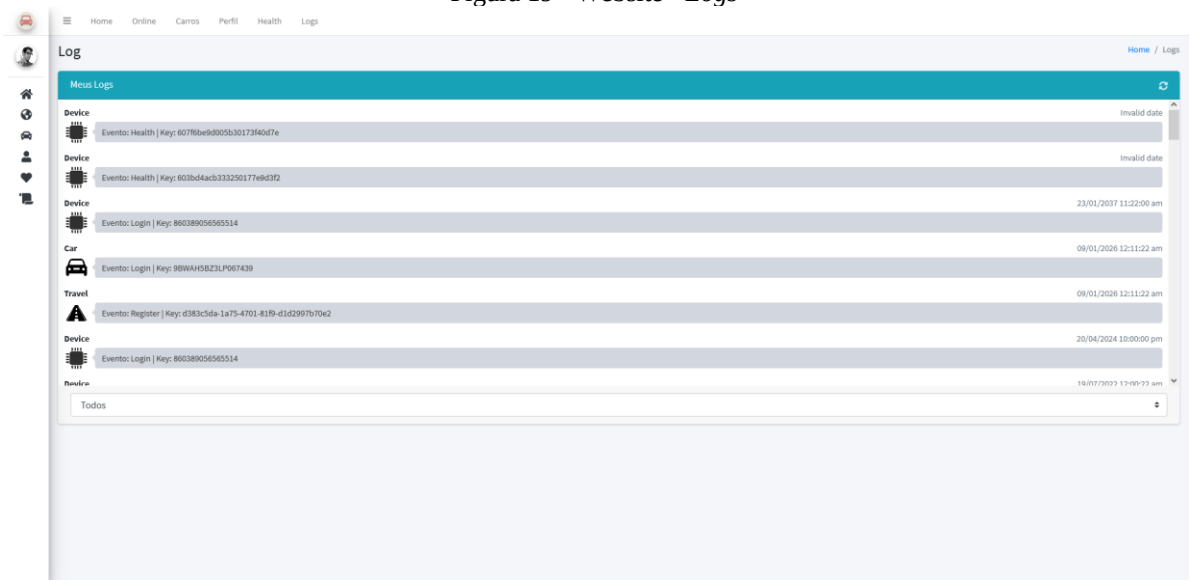


Fonte: Autoria Própria

- *Logs*

O acompanhamento das ações efetuadas pelo dispositivo, pode ser feito através da página de "*Logs*", onde é possível filtrar por origem as ações que geraram log, conforme é possível ver na imagem abaixo.

Figura 19 - Website - *Logs*



Fonte: Autoria Própria

3.7 Testes e Resultados

Os testes efetuados no trabalho e os resultados alcançados, garantem que os objetivos estabelecidos foram cumpridos. Abaixo está a lista de testes efetuados e seus resultados.

- Estabelecer uma conexão GSM entre o dispositivo IoT e a rede da Operadora

Inicialmente essa conexão não obteve sucesso, quando utilizado o ESP32, devido a incompatibilidade de tensão e corrente elétrica do módulo utilizado para este fim, o SIM800L. A implementação do LiLyGo TTGO T-Call resolveu este problema, permitindo que a conexão fosse estabelecida.

- Fonte de alimentação do dispositivo no Carro

A fonte de energia do dispositivo havia sido inicial planejada para ser via porta OBD II, porém, foi notado que esta conexão permanece ativa ainda que a ignição tenha sido desligada, o que poderia ocasionalmente gerar um problema de falta de bateria no veículo. No modelo específico de veículo utilizado neste trabalho é possível encontrar portas USB que são ativadas apenas quando a ignição do veículo é acionada, e para a resolução deste problema, foi utilizada uma destas portas para a alimentação do dispositivo.

- Estabelecer uma conexão com o veículo

A Leitura de dados do veículo foi inicialmente realizada a partir do firmware de exemplo da biblioteca “CAN.h”, como explicado na sessão do primeiro protótipo. A leitura dos sensores funcionou corretamente, como proposto no exemplo, foi possível coletar a rotação do motor e a velocidade do veículo.

- Cálculo de pontuação do condutor

A pontuação do condutor, é baseada no perfil atrelado a veículo, portanto, levando em consideração o modelo específico do veículo, foi criado o perfil com base no manual da montadora, com os seguintes valores na tabela abaixo.

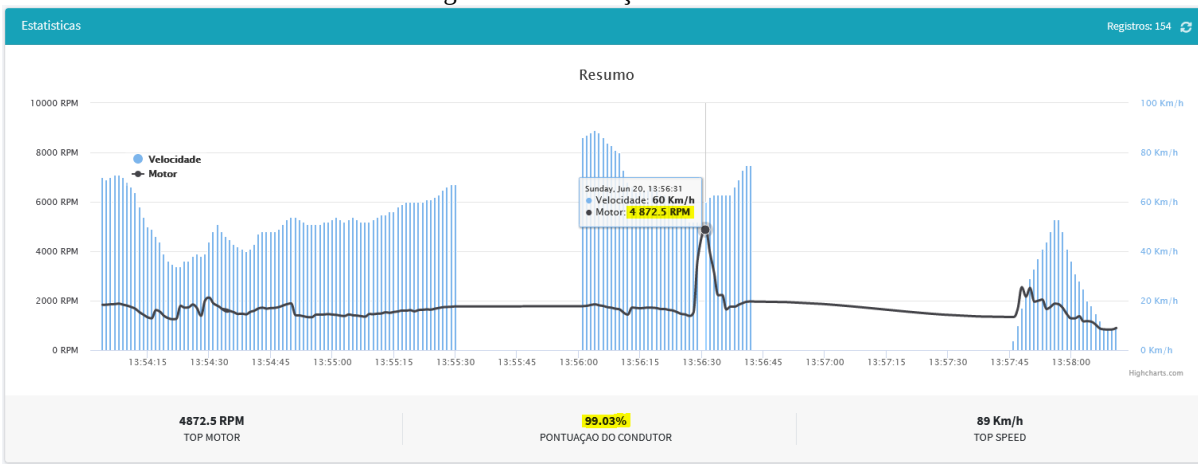
Tabela 10 - Perfil - Polo TSI

Campo	Valor
Potência Máxima (RPM)	5000
Início Torque Máximo (RPM)	3500
Fim Torque Máximo (RPM)	2500

Fonte: Autoria Própria

Dados os valores do perfil acima, foi possível calcular as viagens que haviam sido feitas durantes os testes. Na figura abaixo, é possível notar um exemplo de desvio da pontuação quando as rotações do motor excedem o limite do perfil.

Figura 20 - Pontuação Condutor



Fonte: Autoria Própria

4 CONCLUSÃO

A internet das coisas é uma tendência tecnológica que tem se mostrando útil e com muita relevância para a população. Neste trabalho, foi abordado a possibilidade de se analisar a condução veicular através de dispositivo IoT conectado ao veículo, onde, através de uma pontuação é possível determinar se o condutor está realizando uma condução segura ou não. A relevância deste trabalho está nas oportunidades em que ele cria com a análise destes dados.

Isso se reflete em ações preditivas que podem alertar sobre determinado condutor em uma ocasião específica ou até corriqueira. Também é possível que estes dados sirvam de histórico para empresas de seguros ou até mesmo para o motorista que deseja melhorar sua condução.

Através de testes realizados no trabalho, foi possível constatar que a análise de condução de um veículo, com base nos dados coletados pelo dispositivo IoT, é possível de ser realizada. No trabalho, coletamos os dados do veículo e aplicamos o algoritmo de pontuação, onde foi possível pontuar de 0 a 100 a viagem realizada pelo motorista.

Ainda que a proposta do trabalho tenha sido concluída com sucesso, existem ainda diversas oportunidades de melhorias para se fazer no trabalho as quais deixo listada abaixo para futuras implementações.

4.1 Sugestões de melhorias e trabalhos futuros

Este trabalho tinha o objetivo de validar a viabilidade da análise de condução do veículo através de um dispositivo IoT, o que se provou possível, porém, existem algumas melhorias que podem ser implementadas no trabalho para que o cálculo da pontuação e a análise e de condução possam ser mais assertivos, tais como

- Implementação de GPS

Os dados de localização em tempo real, podem melhorar a assertividade da pontuação devido a precisão e velocidade em que o GPS pode determinar o local exato do veículo. Isso enriquece o cálculo da pontuação com informações que antes não era possível, tais como frenagem brusca e aumento de rotação do motor por ultrapassagem em via expressa.

- Integração com informações de velocidade das vias

A pontuação calculada com base na velocidade do motor, considera apenas o limite máximo das vias do estado de São Paulo, porém, se houvesse integração com a velocidade da via combinada com o GPS, seria possível determinar quando o condutor ultrapassa o limite da via em que está circulando.

- Leitura do Pedal de freio

Os sensores lidos pelo dispositivo, consome apenas o módulo do motor, logo, não é possível fazer a leitura do pedal de freio. Essa implementação é possível, porém não foi explorada no trabalho, caso fosse, poderia ajudar com o critério de frenagem brusca.

- Login no Website

O website desenvolvido no trabalho não implementa login, portanto, não é possível a separação de dados fazendo com que todos os usuários possam ver os dados do outro usuário.

- Implementação de LGPD

Os dados coletados do usuário e do veículo estão atrelados ao dispositivo, porém, não há nenhuma função que possibilite a remoção destes dados mediante solicitação do usuário.

- Implementação de rede GSM 3G ou 4G

A comunicação entre o dispositivo e o servidor é feito através da rede GSM, porém, a latência desta conexão é muito alta, fazendo com que o dispositivo não consiga enviar seus dados mais rápido que uma vez por segundo. A implementação de uma rede mais rápida, permitiria que o dispositivo enviasse dados mais rápidos.

- Uso de Provedor Cloud

Por questões de custo, foi utilizado o Raspberry PI 4 para o processamento de dados, além de centralizar o recebimento de informações via MQTT e HTTP, porém a utilização de um provedor em Cloud, a comunicação poderia ser feita com uma latência muito menor.

REFERÊNCIAS

[1] Raspberry PI Foundation. In: Product Brief. **Raspberry Pi 4 Model B.** 2021. Disponível em: https://datasheets.raspberrypi.org/rpi4/raspberry-pi-4-product-brief.pdf. Acesso em 15 mai. 2021.

[2] LilyGo. LILYGO® TTGO T-Call V1.4. Disponível em: http://www.lilygo.cn/prod_view.aspx?TypeId=50044&Id=1127&FId=t3:50044:3. Acesso em 15 mai. 2021.

[3] SN65HDV2X. SN65HDV23. Disponível em: https://www.ti.com/lit/ds/symlink/sn65hvd23.pdf. Acesso em 15 mai. 2021.

[4] *OBD2 Explained - A Simple Intro (2021)*. Disponível em: https://www.csselectronics.com/screen/page/simple-intro-obd2-explained/language/en. Acesso em 24 mai. 21

[5] *Which OBD2 Protocol Is Supported By My Vehicle?* Disponível em: https://obdstation.com/obd2-protocols/. Acessado em 24 mai. 21

[6] *THE CAR HACKER'S HANDBOOK*. Disponível em: http://opengarages.org/handbook/ebook/. Acesso em 24 mai. 21

[7] MQTT Protocol. Disponível em: https://mqtt.org. Acesso em 24 mai. 21

[8] SSD1306. Disponível em: https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf. Acesso em 24 mai. 21

[9] DS3231. Disponível em: https://www.alldatasheet.com/datasheet-pdf/pdf/254832/MAXIM/DS3231.html. Acesso em 24 mai. 21

[10] Balena Etcher. Disponível em: https://www.balena.io/etcher/. Acesso em 24 mai. 21

[11] Kali Linux. Disponível em: https://www.kali.org/docs/. Acesso em 01 jun. 21

[12] GIT. Disponível em: https://git-scm.com/doc. Acesso em 01 jun. 21

[13] VSCode. Disponível em: https://code.visualstudio.com/. Acesso em 01 jun. 21

[14] PlatformIO. Disponível em: https://docs.platformio.org/en/latest/tutorials/index.html. Acesso em 01 jun. 21

[15] Heroku. Disponível em: https://devcenter.heroku.com/start. Acesso em 01 jun. 21

[16] Diagrams.Net. Disponível em: https://www.diagrams.net/blog. Acesso em 01 jun. 21

[17] Fritzing. Disponível em: https://fritzing.org/. Acesso em 01 jun. 21

[18] Node-Red. Disponível em: https://nodered.org/. Acesso em 15 jun. 21

[19] No-IP. Disponível em: https://www.noip.com/pt-BR. Acesso em 15 jun. 21

[20] MongoDB. Disponível em: https://www.mongodb.com/. Acesso em 15 jun. 21

[21] MQTT Mosquitto. Disponível em: https://mosquitto.org/. Acesso em 15 jun. 21

[22] CAN Protocol. Disponível em: https://www.ti.com/lit/an/sloa101b/sloa101b.pdf. Acesso em 15 jun. 21

APÊNDICES

Apêndice A - Código Fonte de Exemplo - EngineRPM.ino

```
// Copyright (c) Sandeep Mistry. All rights reserved.
// Licensed under the MIT license. See LICENSE file in the project root for full license information.
//
//
// This examples queries the engine RPM (OBD-II PID 0x0c) once a seconds and
// prints the value to the serial monitor
//
#include <CAN.h>

// Most cars support 11-bit adddress, others (like Honda),
// require 29-bit (extended) addressing, set the next line
// to true to use extended addressing
const bool useStandardAddressing = true;

void setup() {
  Serial.begin(9600);

  Serial.println("CAN OBD-II engine RPM");

  // start the CAN bus at 500 kbps
  if (!CAN.begin(500E3)) {
    Serial.println("Starting CAN failed!");
    while (1);
  }

  // add filter to only receive the CAN bus ID's we care about
  if (useStandardAddressing) {
    CAN.filter(0x7e8);
  } else {
    CAN.filterExtended(0x18daf110);
  }
}

void loop() {
  if (useStandardAddressing) {
    CAN.beginPacket(0x7df, 8);
  } else {
    CAN.beginExtendedPacket(0x18db33f1, 8);
  }
  CAN.write(0x02); // number of additional bytes
  CAN.write(0x01); // show current data
  CAN.write(0x0c); // engine RPM
  CAN.endPacket();

  // wait for response
  while (CAN.parsePacket() == 0 ||
         CAN.read() < 3 ||         // correct length
         CAN.read() != 0x41 ||     // correct mode
         CAN.read() != 0x0c);      // correct PID

  float rpm = ((CAN.read() * 256.0) + CAN.read()) / 4.0;

  Serial.print("Engine RPM = ");
  Serial.println(rpm);

  delay(1000);
```

```
}
```

Apêndice B - Código Fonte de Exemplo - VINReader.ino

```
// Copyright (c) Sandeep Mistry. All rights reserved.
// Licensed under the MIT license. See LICENSE file in the project root for full license information.
//
//
// This examples queries the ECU for the car's Vehicle Identification Number (VIN) and
// prints it out to the serial monitor using Mode 09 and OBD-II PID 0x02
//
#include <CAN.h>

// Most cars support 11-bit adddress, others (like Honda),
// require 29-bit (extended) addressing, set the next line
// to true to use extended addressing
const bool useStandardAddressing = true;

void setup() {
  Serial.begin(9600);

  Serial.println("CAN OBD-II VIN reader");

  // start the CAN bus at 500 kbps
  if (!CAN.begin(500E3)) {
    Serial.println("Starting CAN failed!");
    while (1);
  }

  // add filter to only receive the CAN bus ID's we care about
  if (useStandardAddressing) {
    CAN.filter(0x7e8);
  } else {
    CAN.filterExtended(0x18daf110);
  }
}

void loop() {
  // send the request for the first chunk
  if (useStandardAddressing) {
    CAN.beginPacket(0x7df, 8);
  } else {
    CAN.beginExtendedPacket(0x18db33f1, 8);
  }
  CAN.write(0x02); // Number of additional bytes
  CAN.write(0x09); // Request vehicle information
  CAN.write(0x02); // Vehicle Identification Number (VIN)
  CAN.endPacket();

  // wait for response
  while (CAN.parsePacket() == 0 ||
         CAN.read() != 0x10 || CAN.read() != 0x14 || // correct length
         CAN.read() != 0x49 ||                      // correct mode
         CAN.read() != 0x02 ||                      // correct PID
         CAN.read() != 0x01);

  // print out
  while (CAN.available()) {
    Serial.write((char)CAN.read());
  }
```

```
  // read in remaining chunks
  for (int i = 0; i < 2; i++) {
    // send the request for the next chunk
    if (useStandardAddressing) {
      CAN.beginPacket(0x7e0, 8);
    } else {
      CAN.beginExtendedPacket(0x18db33f1, 8);
    }
    CAN.write(0x30);
    CAN.endPacket();

    // wait for response
    while (CAN.parsePacket() == 0 ||
           CAN.read() != (0x21 + i)); // correct sequence number

    // print out
    while (CAN.available()) {
      Serial.write((char)CAN.read());
    }
  }

  Serial.println("That's all folks!");

  while (1); // all done
}
```

Apêndice C – Código Fonte do Website

- API.js

```
const endpoint = "http://rafaelgomes.ddns.net:1880/api";
const Devices = {
    getAll: function(){
        var result = $.ajax({
                                url: `${endpoint}/devices`,
                                type: "GET",
                                async: false,
                                headers: { "ACCEPT": "application/json;odata=verbose" },
                        });
                        return result.responseJSON;
    },
    getById: function(id){
        var result = $.ajax({
                                url: `${endpoint}/device/${id}`,
                                type: "GET",
                                async: false,
                                headers: { "ACCEPT": "application/json;odata=verbose" },
                        });
                        return result.responseJSON;
    },
    getStatus: function(id){
        var result = $.ajax({
                                url: `${endpoint}/device/${id}/status`,
                                type: "GET",
                                async: false,
                                headers: { "ACCEPT": "application/json;odata=verbose" },
                        });
                        return result.responseJSON;
    }
```

```
};
const Travels = {
        getAll: function(){
    var result = $.ajax({
                        url: `${endpoint}/travels`,
                        type: "GET",
                        async: false,
                        headers: { "ACCEPT": "application/json;odata=verbose" },
                });
                return result.responseJSON;
        },
        getById: function(id){
    var result = $.ajax({
                        url: `${endpoint}/travel/${id}`,
                        type: "GET",
                        async: false,
                        headers: { "ACCEPT": "application/json;odata=verbose" },
                });
                return result.responseJSON;
        },
        getSensors: function(id,pid,startDate=null,endDate=null,state=null){
    var result = $.ajax({
                        url: `${endpoint}/travel/${id}/sensors`,
                        data: { pid:pid, startDate: startDate, endDate: endDate, state: state },
                        type: "GET",
                        async: false,
                        headers: { "ACCEPT": "application/json;odata=verbose" },
                });
                return result.responseJSON;
        }
};
const Cars = {
   getAll: function(){
      let devices = Devices.getAll();
      let cars = [];
      $(devices).each(function(i,device){
         $(device.cars).each(function(x,car){
            let result = cars.find(x => x.carVIN == car.carVIN);
            if(result == undefined) cars.push(car);
         })
      });
                return cars;
   },
   getByDeviceId: function(deviceId){
      var result = $.ajax({
                        url: `${endpoint}/cars/${deviceId}`,
                        type: "GET",
                        async: false,
                        headers: { "ACCEPT": "application/json;odata=verbose" },
                });
                let cars = [];
                $(result.responseJSON[0].cars).each(function(x,car){
                        let result = cars.find(x => x.carVIN == car.carVIN);
                        if(result == undefined) cars.push(car);
                })
                return cars;
   },
   getById: function(id){
      var result = $.ajax({
                        url: `${endpoint}/car/${id}`,
```

```
                type: "GET",
                async: false,
                headers: { "ACCEPT": "application/json;odata=verbose" },
            });
            return result.responseJSON;
    },
    getTravels: function(id,startDate=null,endDate=null){
        var result = $.ajax({
                url: `${endpoint}/car/${id}/travels`,
                type: "GET",
                async: false,
                data: { startDate: startDate, endDate: endDate },
                headers: { "ACCEPT": "application/json;odata=verbose" },
            });
            return result.responseJSON;
    },
        update: function(id,data){
            var result = $.ajax({
                url: `${endpoint}/car/${id}`,
                data: data,
                type: "POST",
                async: false,
                headers: { "ACCEPT": "application/json;odata=verbose" },
            });
            return result.responseJSON;
        },
        getSensors: function(id,pid,startDate=null,endDate=null,state=null){
            let travels = this.getTravels(id,startDate,endDate).sort((a,b) => new Date(a.timestamp) - new
Date(b.timestamp));
            let full = [];
        $(travels).each(function(i,travel){
                let result = Travels.getSensors(travel.id,pid,startDate,endDate,state);
                full = full.concat(result);
            });
            return full;
        }
};
let Profile = {
        getAll: function(filter=null){
        var result = $.ajax({
                url: `${endpoint}/profiles`,
                type: "GET",
                data: filter,
                async: false,
                headers: { "ACCEPT": "application/json;odata=verbose" },
            });
            return result.responseJSON;
        },
        getById: function(id){
        var result = $.ajax({
                url: `${endpoint}/profile/${id}`,
                type: "GET",
                async: false,
                headers: { "ACCEPT": "application/json;odata=verbose" },
            });
            return result.responseJSON;
    },
        newProfile: function(data){
            var result = $.ajax({
                url: `${endpoint}/profile`,
```

```
                data: data,
                type: "POST",
                async: false,
                headers: { "ACCEPT": "application/json;odata=verbose" },
            });
            return result.responseJSON;
        },
        editProfile: function(id, data){
            var result = $.ajax({
                url: `${endpoint}/profile/${id}`,
                data: data,
                type: "POST",
                async: false,
                headers: { "ACCEPT": "application/json;odata=verbose" },
            });
            return result.responseJSON;
        }
};
let Logs = {
        getAll: function(filter=null){
    var result = $.ajax({
                url: `${endpoint}/logs`,
                type: "GET",
                data: filter,
                async: false,
                headers: { "ACCEPT": "application/json;odata=verbose" },
            });
            return result.responseJSON;
   }
}
```

- Site.js

```
const menu = [
   { id: "mnhHome", title: "Home", url: "index.html", type: "H" },
   { id: "mnhOnline", title: "Online", url: "online.html", type: "H" },
   { id: "mnhCar", title: "Carros", url: "cars.html", type: "H" },
   { id: "mnhProfile", title: "Perfil", url: "profile.html", type: "H" },
   { id: "mnhHealth", title: "Health", url: "health.html", type: "H" },
   { id: "mnhLogs", title: "Logs", url: "logs.html", type: "H" },
   { id: "mnvHome", title: "Home", url: "index.html", css: "fa-home" ,type: "V" },
   { id: "mnvOnline", title: "Online", url: "online.html", css:"fa-globe-americas", type: "V" },
   { id: "mnvCar", title: "Carros", url: "cars.html", css: "fa-car", type: "V" },
   { id: "mnvProfile", title: "Perfil", url: "profile.html", css:"fa-user",  type: "V" },
   { id: "mnvHealth", title: "Health", url: "health.html", css:"fa-heart",  type: "V" },
   { id: "mnvLogs", title: "Logs", url: "logs.html", css:"fa-scroll", type: "V" }
];
const resources = [
   { id: "Car", image: "/dist/img/resources/car.webp" },
   { id: "Device", image: "/dist/img/resources/device.png" },
   { id: "Sensor", image: "/dist/img/resources/sensor.png" },
   { id: "Travel", image: "/dist/img/resources/travel.png" }
]

const Toast = Swal.mixin({
   toast: true,
   position: 'top-end',
   showConfirmButton: false,
   timer: 3000
});
```

```
$(document).ready(function(){
    LTE.init();
});

const LTE = {
    init: function(){
        $(menu).each(function(i,item){
            if(item.type == "H") LTE.newHorizontalMenuItem(item.id,item.title,item.url);
            if(item.type == "V") LTE.newVerticalMenuItem(item.id,item.title,item.url,item.css);
        });
    },
    newVerticalMenuItem: function (id, name, url, icon) {
        if ($(`#${id}`).length > 0) return false;

        var container = $("#MenuVContainer");
        container.append(`<li class="nav-item"><a href="${url}" class="nav-link" id="${id}"><i class="nav-
icon fas ${icon}"></i><p>${name}</p></a></li>`);
        return true;
    },
    newHorizontalMenuItem: function (id, name, url) {
        if ($(`#${id}`).length > 0) return false;

        var container = $("#MenuHContainer");
        container.append(`<li class="nav-item d-none d-sm-inline-block" id="${id}"><a href="${url}"
class="nav-link">${name}</a></li>`);
        return true;
    },
    newVerticalMenuTreeItem: function (id, name, icon, subIds, subNames, subUrls, subIcons) {
        if ($(`#${id}`).length > 0) return false;

        var print = "";
        var container = $("#MenuVContainer");
        print += `<li class="nav-item has-treeview" id="${id}">`;
        print += `<a href="#" class="nav-link">`;
        print += `<i class="nav-icon fas ${icon}"></i>`;
        print += `<p>`;
        print += `${name}`;
        print += `<i class="right fas fa-angle-left"></i>`;
        print += `</p>`;
        print += `</a>`;
        print += `<ul class="nav nav-treeview">`;

        $(subIds).each(function (i, item) {
            print += `<li class="nav-item" id="${item}">`;
            print += `<a href="${subUrls[i]}" class="nav-link">`;
            print += `<i class="fas ${subIcons[i]} nav-icon"></i>`;
            print += `<p>${subNames[i]}</p>`;
            print += `</a>`;
            print += `</li>`;
        });
        print += `</ul>`;
        print += `</li>`;
        container.append(print);
    },
    getResourceImage: function(id){
        let resource = resources.find(x => x.id == id);
        if(resource == undefined) return "";
        return resource.image;
    }
```

```
};
```

- Mqtt.js

```
//Cliente MQTT para gereciamento do protocolo
var mqttClient;

//Objeto MQTT com funções pertinentes ao protocolo
const MQTT = {
   connect: function (host,port,user,pass,clientId,onMessageArrived=this.onMessageArrived) {
      let mqtt = new Paho.MQTT.Client(host, port, clientId);
      mqtt.onConnectionLost = this.onConnectionLost;
      mqtt.onMessageArrived = onMessageArrived;
      mqtt.onConnected = this.onConnected;
      var options = {
         timeout: 10,
         cleanSession: false,
         onSuccess: this.onConnect,
         onFailure: this.onFailure,
         userName: user,
         password: pass,
      };
      mqtt.connect(options);
      mqttClient = mqtt;
   },
   onConnectionLost: function (responseObject) {
      if (responseObject.errorCode !== 0) {
         console.log(responseObject)
         console.log("MQTT -> onConnectionLost:" + responseObject.errorMessage);
      }
   },
   onMessageArrived: function (message) {
      var msg = message.payloadString;
      console.log(`MQTT -> onMessageArrived: ${msg}`)
      console.log(`MQTT -> processando mensagem`)
   },
   onFailure: function (message) {
      console.log("MQTT -> onFailure: Falha");
      setTimeout(MQTTconnect, 2000);
   },
   onConnected: function (recon, url) {
      console.log("MQTT -> onConnected: " + reconn);
   },
   onConnect: function () {
      console.log("MQTT -> onConnect: " + mqttClient.isConnected());
   },
   subscribe: function (stopic, sqos) {

      if (!mqttClient.isConnected()) {
         console.log("MQTT -> Não conectado, nao poderá subscrever");
         return;
      }

      if (sqos > 2) sqos = 0;
      var soptions = { qos: sqos };
      mqttClient.subscribe(stopic, soptions);
   },
   disconnect: function () {
      if (mqttClient.isConnected()) mqttClient.disconnect();
   },
```

```
    sendMessage: function (topic, msg, retain_flag, pqos) {
        if (!mqttClient.isConnected()) {
            console.log("MQTT -> Não conectado, nao poderá enviar");
            return;
        }
        console.log(`MQTT -> Enviando mensagem: "${msg}", Tópico: "${topic}", Retain: "${retain_flag}", QoS:
${pqos}`);
        if (pqos > 2) pqos = 0;
        message = new Paho.MQTT.Message(msg);
        message.destinationName = topic;
        message.qos = pqos;
        message.retained = retain_flag;
        mqttClient.send(message);
    }
}
```

- Index.js

```
let carChart = null;
let startDate = moment().startOf('day').subtract('hour',3).valueOf();
let endDate = moment().subtract('hour',3).valueOf();
let selectedCar = -1;
let selectedTravel = -1;
let carState = "All";


$(document).ready(function () {


    $(document).on("click", "[name=btnCar]", function (e) {
        e.preventDefault();
        let id = $(this).data("id");
        selectedCar = id;
        $("#carsContainer .badge").each(function (i, badge) { badge.remove(); });
        $(this).append(`<span class="badge bg-warning">Selecionado</span>`);
        client.selectCar(id, startDate, endDate, carState);
    });
    $(document).on("click", "[name=btnTravel]", function (e) {
        e.preventDefault();
        let id = $(this).data("id");
        selectedTravel = id;
        $("[name=btnTravel] .font-weight-bold").removeClass("font-weight-bold");
        $(this).find('td').first().addClass(`font-weight-bold`);
        client.selectTravel(id, startDate, endDate, carState);
    });
    $(document).on("click","[name=btnRefreshData]",function(e){
        client.selectCar(selectedCar, startDate, endDate, carState);
    });
    $(document).on("click","#startAddTime",function(e){
```

```
    let increment = $("#timeIncrement").val();
    let nDateTime = moment.unix(startDate/1000).add(increment,'minutes');
    startDate = nDateTime.valueOf();
    nDateTime = nDateTime.add('hour',3);
    $("#reservationtime").data('daterangepicker').setStartDate(nDateTime);
    client.setupTravels(selectedCar);
    client.selectTravel(selectedTravel, startDate, endDate, carState);
});
$(document).on("click","#startDelTime",function(e){
    let increment = $("#timeIncrement").val();
    let nDateTime = moment.unix(startDate/1000).subtract(increment,'minutes');
    startDate = nDateTime.valueOf();
    nDateTime = nDateTime.add('hour',3);
    $("#reservationtime").data('daterangepicker').setStartDate(nDateTime);
    client.setupTravels(selectedCar);
    client.selectTravel(selectedTravel, startDate, endDate, carState);
});
$(document).on("click","#endAddTime",function(e){
    let increment = $("#timeIncrement").val();
    let nDateTime = moment.unix(endDate/1000).add(increment,'minutes');
    endDate = nDateTime.valueOf();
    nDateTime = nDateTime.add('hour',3);
    $("#reservationtime").data('daterangepicker').setEndDate(nDateTime);
    client.setupTravels(selectedCar);
    client.selectTravel(selectedTravel, startDate, endDate, carState);
});
$(document).on("click","#endDelTime",function(e){
    let increment = $("#timeIncrement").val();
    let nDateTime = moment.unix(endDate/1000).subtract(increment,'minutes');
    endDate = nDateTime.valueOf();
    nDateTime = nDateTime.add('hour',3);
    $("#reservationtime").data('daterangepicker').setEndDate(nDateTime);
    client.setupTravels(selectedCar);
    client.selectTravel(selectedTravel, startDate, endDate, carState);
});

$('#reservationtime').daterangepicker(
    {
        timePicker: true,
        timePickerIncrement: 1,
```

```
        //minDate: moment(),
        //maxDate: moment().add(1, 'month'),
        //dateLimit: { days: 5 },
        startDate: moment.unix(startDate/1000).add('hour',3),
        endDate: moment.unix(endDate/1000).add('hour',3),
        locale: {
          format: 'DD/MM/YYYY hh:mm:ss A'
        },
      },
      function (start, end) {
        startDate = start.subtract('hour',3).valueOf();
        endDate = end.subtract('hour',3).valueOf();
        client.setupTravels(selectedCar);
        client.selectTravel(selectedTravel, startDate, endDate, carState);
      }
    );

    client.setup();
});
let client = {
    setup: function () {
        $("#carsCard .overlay").removeClass("d-none");
        $("#travelsCard .overlay").removeClass("d-none");

        let cars = Cars.getAll();
        let html = "";
        $(cars).each(function (i, item) {
            html += `<a class="btn btn-app" name="btnCar" data-id="${item.carVIN}"><i class="fas fa-
car"></i>${item.name == undefined ? item.carVIN : item.name}</a>`;
        });
        $("#carsContainer").html(html);

        html = "";
        html += `<table class="table table-striped table-valign-middle">`;
        html += `<thead>`;
        html += `<tr>`;
        html += `<th><i class="fa fa-route"></i> Viagem</th>`;
        html += `<th class="text-center">Exibir</th>`;
        html += `</tr>`;
        html += `</thead>`;
```

```
        html += `<tbody>`;
        html += `<tr data-id="All" name="btnTravel">`;
        html += `<td><i class="fa fa-route"></i> Todos</td>`;
        html += `<td class="text-center"><a href="#" class="text-muted"><i class="fas fa-search"></i></a></td>`;
        html += `</tr>`;
        html += `</tbody>`;
        html += `</table>`;
        $("#travelsContainer").html(html);

        $("#carsCard .overlay").addClass("d-none");
        $("#travelsCard .overlay").addClass("d-none");

        this.setupChart();
    },
    setupChart: function () {
        if (carChart != null) lineChart.clearChart(carChart);
        carChart = lineChart.setup("carChartContainer", "Resumo");
    },
    selectCar: function (id) {
        if (id == -1) {
            Toast.fire({ icon: 'warning', title: 'Selecione um Carro!' });
            return;
        }

       client.setupTravels(id);
       if(selectedTravel != -1) client.selectTravel(selectedTravel, startDate, endDate, carState);
    },
    setupTravels: function(carId){
        $("#travelsCard .overlay").removeClass("d-none");

        let travels = Cars.getTravels(carId, startDate, endDate);

        let html = "";
        html += `<table class="table table-striped table-valign-middle table-head-fixed">`;
        html += `<thead>`;
        html += `<tr>`;
        html += `<th><i class="fa fa-route"></i> Viagem</th>`;
        html += `<th class="text-center">Exibir</th>`;
        html += `</tr>`;
```

```
        html += `</thead>`;
        html += `<tbody>`;
        html += `<tr data-id="All" name="btnTravel">`;
        html += `<td><i class="fa fa-route"></i> Todos [${travels.reduce((t,i) => t + i.records,0)}]</td>`;
        html += `<td class="text-center"><a href="#" class="text-muted"><i class="fas fa-search"></i></a></td>`;
        html += `</tr>`;
        let selected = false
        $(travels).each(function (i, item) {
            if(item.id == selectedTravel) selected = true;
            let ts = new Date((item.timestamp+10800)*1000);
            let title = `${ts.toLocaleDateString()} ${ts.toLocaleTimeString()}`;
            html += `<tr data-id="${item.id}" name="btnTravel">`;
            html += `<td class="${item.id == selectedTravel?"font-weight-bold":""}"> ${title} [${item.records}]</td>`;
            html += `<td class="text-center"><a href="#" class="text-muted"><i class="fas fa-search"></i></a></td>`;
            html += `</tr>`;
        });
        if(selectedTravel != "All" && !selected) selectedTravel = -1;
        html += `</tbody>`;
        html += `</table>`;
        $("#travelsContainer").html(html);
        $("#countTravels").html(`Registros: ${travels.length}`);
        $("#travelsCard .overlay").addClass("d-none");
    },
    selectTravel: function (id, startDate = null, endDate = null, carState = null) {
        if (id == -1) {
            Toast.fire({ icon: 'warning', title: 'Selecione uma Viagem!'});
            return;
        } else if (selectedCar == -1) {
            Toast.fire({ icon: 'warning', title: 'Selecione um Carro!' });
            return;
        }
        $("#chartCard .overlay").removeClass("d-none");
        this.setupChart();

        let dataRPM, dataKmh;

        if (id == "All") {
```

```
            dataRPM = Cars.getSensors(selectedCar, "0C", startDate, endDate, carState);
            dataKmh = Cars.getSensors(selectedCar, "0D", startDate, endDate, carState);
        } else {
            dataRPM = Travels.getSensors(id, "0C", startDate, endDate, carState);
            dataKmh = Travels.getSensors(id, "0D", startDate, endDate, carState);
        }

        let sumRPM = dataRPM.reduce((a, b) => a + b.behaviour, 0);
        let avgRPM = (sumRPM / dataRPM.length) || 0;
        let sumKmh = dataKmh.reduce((a, b) => a + b.behaviour, 0);
        let avgKmh = (sumKmh / dataKmh.length) || 0;
        let avgScore = (avgRPM+avgKmh) / 2;
        $("#score").html(`${avgScore.toFixed(2)}%`);

        dataRPM = dataRPM.map(x => [x.receive_date*1000, x.value]);
        dataKmh = dataKmh.map(x => [x.receive_date*1000, x.value]);

        carChart.series[0].setData(dataKmh);
        carChart.series[1].setData(dataRPM);

        let maxRPM = 0;
        if (dataRPM.length > 0) maxRPM = Math.max.apply(Math, dataRPM.map(function (o) { return o[1];
}))
        let maxKMH = 0;
        if (dataKmh.length > 0) maxKMH = Math.max.apply(Math, dataKmh.map(function (o) { return o[1];
}))

        $("#TopKmh").html(`${maxKMH} Km/h`)
        $("#TopRpm").html(`${maxRPM} RPM`);
        $("#countRecords").html(`Registros: ${dataRPM.length}`);
        $("#chartCard .overlay").addClass("d-none");
    }
};
let lineChart = {
    setup: function (id, text) {
        return Highcharts.chart(`${id}`, {
            chart: { zoomType: 'xy' },
            title: { text: text },
            xAxis: [{ type: 'datetime', crosshair: true }],
            yAxis: [{
```

```
    labels: {
        format: '{value} RPM',
        style: { color: Highcharts.getOptions().colors[1] }
    },
    title: {
        text: 'Motor',
        style: { color: Highcharts.getOptions().colors[1] },
        enabled: false
    }
}, {
    title: {
        text: 'Velocity',
        style: { color: Highcharts.getOptions().colors[0] },
        enabled: false
    },
    labels: {
        format: '{value} Km/h',
        style: { color: Highcharts.getOptions().colors[0] }
    },
    opposite: true
}],
tooltip: {
    shared: true
},
legend: {
    layout: 'vertical',
    align: 'left',
    x: 120,
    verticalAlign: 'top',
    y: 100,
    floating: true,
    backgroundColor: Highcharts.defaultOptions.legend.backgroundColor || 'rgba(255,255,255,0.25)'
},
series: [{
    yAxis: 1,
    type: 'column',
    name: 'Velocidade',
    tooltip: { valueSuffix: ' Km/h' }
}, {
    name: 'Motor',
```

```
            type: 'spline',
            tooltip: { valueSuffix: ' RPM' }
        }],
    });
  },
  setPoint(chart, rpm, kmh) {
    chart.series[0].addPoint(kmh);
    chart.series[1].addPoint(rpm);
  },
  clearChart: function (chart) {
    while (chart.series.length > 0) chart.series[0].remove(true)
  }
}
let Helper = {
  groupBy: function (collection, property) {
    var i = 0, val, index,
      values = [], result = [];
    for (; i < collection.length; i++) {
      val = collection[i][property];
      index = values.indexOf(val);
      if (index > -1)
        result[index].push(collection[i]);
      else {
        values.push(val);
        result.push([collection[i]]);
      }
    }
    return { groups: result, keys: values };
  }
}
```

- Online.js

```
let rpmChart = null;
let kmhChart = null;
let mqtt;
$(document).ready(function () {
  client.setup();
});
let client = {
  setup: function () {
```

```
        this.setupChart();
        MQTT.connect("rafaelgomes.ddns.net", 9001, "", "", "web_client",client.onMessageArrivedMQTT);
        var trySubscribe = setInterval(function () {
            if (mqttClient.isConnected()) {
                console.log(`MQTT -> Subscrever ao tópico "/sensors"`)
                MQTT.subscribe("/sensors", 0);
                clearInterval(trySubscribe);
            }
        }, 500);
    },
    onMessageArrivedMQTT: function (message) {
        let data = JSON.parse(message._getPayloadString());
        let dataRPM = data.sensors.find(x => x.pid == "0C").value;
        let dataKmh = data.sensors.find(x => x.pid == "0D").value;

        gaugeChart.setPoint(rpmChart, dataRPM);
        gaugeChart.setPoint(kmhChart, dataKmh);
    },
    setupChart: function () {
        if (rpmChart != null) gaugeChart.clearChart(rpmChart);
        if (kmhChart != null) gaugeChart.clearChart(kmhChart);

        rpmChart = gaugeChart.setup("rpmChartContainer", 0, 9000, "Motor", "RPM", "RPM", "#F00");
        kmhChart = gaugeChart.setup("kmhChartContainer", 0, 220, "Velocidade", "velocidade", "Km/h",
"#00F");
    }
};
let gaugeChart = {
    setup: function (id, min, max, title, name, unit, color) {
        var json = {};
        json.chart = {
            type: 'gauge',
            plotBackgroundColor: null,
            plotBackgroundImage: null,
            plotBorderWidth: 0,
            plotShadow: false
        };
        json.credits = { enabled: false };
        json.title = { text: title, enabled: false };;
        json.pane = { startAngle: -150, endAngle: 150 };
```

```
        json.yAxis = [{
            min: min,
            max: max,
            lineColor: color,
            tickColor: color,
            minorTickColor: color,
            offset: -25,
            lineWidth: 2,
            labels: { distance: -20, rotation: 'auto' },
            tickLength: 5,
            minorTickLength: 5,
            endOnTick: false
        }];

        json.series = [{
            name: name,
            data: [0],
            dataLabels: {
                formatter: function () {
                    return `<span style="color:#339">${this.y} ${unit}</span>`;
                },
                backgroundColor: {
                    linearGradient: { x1: 0, y1: 0, x2: 0, y2: 1 },
                    stops: [[0, '#DDD'], [1, '#FFF']]
                }
            },
            tooltip: { valueSuffix: ` ${unit}` }
        }];

        return $(`#${id}`).highcharts(json);
    },
    setPoint: function (chart, point) {
        $(chart).highcharts().series[0].points[0].update(point);
    },
    clearChart: function (chart) {
        this.setPoint(chart, 0);
    }
}
let Helper = {
    groupBy: function (collection, property) {
```

```
            var i = 0, val, index,
                values = [], result = [];
            for (; i < collection.length; i++) {
                val = collection[i][property];
                index = values.indexOf(val);
                if (index > -1)
                    result[index].push(collection[i]);
                else {
                    values.push(val);
                    result.push([collection[i]]);
                }
            }
            return { groups: result, keys: values };
        }
    }
```

- Cars.js

```
$(document).ready(function () {

    $(document).on("click", "[name=btnDevice]", function (e) {
        e.preventDefault();
        let id = $(this).data("id");
        client.selectMicrochip(id);
    });
    $(document).on("click", "[name=btnCar]", function (e) {
        e.preventDefault();
        let id = $(this).data("id");
        client.selectCar(id);
    });
    $(document).on("click","#btnSaveCar",function(e){
        let id = $(this).data("id");
        let name = $("#txtDeviceName").val();
        let profile = $("#slProfile option:selected").val();
        client.editCar(id,name,profile);
    });

    client.setup();

});
let client = {
    setup: function () {
        $("#devicesCard .overlay").removeClass("d-none");

        let devices = Devices.getAll();
        let html = "";
        $(devices).each(function (i, item) {
            html += `<a class="btn btn-app" name="btnDevice" data-id="${item._id}"><i class="fas fa-
microchip"></i>${item.imei}</a>`;
        });
        $("#devicesContainer").html(html);

        let option = new Option("Nenhum",-1,false, true);
        $("#slProfile").append(option);
```

```
        let profiles = Profile.getAll();
        $(profiles).each(function(i,item){
            let option = new Option(item.title,item._id,false, false);
            $("#slProfile").append(option);
        });

        $("#devicesCard .overlay").addClass("d-none");
    },
    selectMicrochip: function(id){
        $("#carsCard").removeClass("d-none");
        $("#carsCard .overlay").removeClass("d-none");

        let cars = Cars.getByDeviceId(id);

        let html = "";
        $(cars).each(function (i, item) {
            html += `<a class="btn btn-app" name="btnCar" data-id="${item.carVIN}"><i class="fas fa-
car"></i>${item.Name == undefined ? item.carVIN : item.Name}</a>`;
        });
        $("#carsContainer").html(html);
        $("#carsCard .overlay").addClass("d-none");
    },
    selectCar(id){
        $("#carForm").removeClass("d-none");
        let device = Cars.getById(id);
        if(device.length == 0) return;
        let car = device[0].cars.filter(x => x.carVIN == id);
        if(car.length == 0) return;
        else car = car[0];
        if(car.name != undefined) $("#txtDeviceName").val(car.name);
        if(car.profile != undefined) $(`#slProfile option[value='${car.profile}']`).prop('selected', true);
        $("#txtCarVin").val(car.carVIN);
        $("#txtCarVin").prop("disabled",true);
        $("#btnSaveCar").data("id",id);
    },
    editCar(id,name,profile){
        let device = Cars.getById(id);
        if(device.length == 0) return;
        let carIndex = device[0].cars.findIndex(x => x.carVIN==id);
        if(carIndex == -1) return;
        device[0].cars[carIndex].name = name;
        if(profile != -1) device[0].cars[carIndex].profile = profile;
        else delete device[0].cars[carIndex].profile;

        Cars.update(id,device[0]);
        Toast.fire({
            icon: 'success',
            title: 'Carro Atualizado!'
        })
    }
};
```

- Profile.js

```
$(document).ready(function () {
    $(document).on("click","#btnRefresh",function(e){
        client.setup();
    });
    $(document).on("click","#btnNewProfile",function(e){
        $("#title").val("");
        $("#maxPower").val("");
        $("#maxTorqueStart").val("");
```

```
        $("#maxTorqueEnd").val("");
        $("#btnSaveProfile").data("operation",1);
        $("#formProfileCard").show();
    });
    $(document).on("click","#btnSaveProfile",function(e){
        let id = $(this).data("id");
        let operation = $(this).data("operation");
        let title = $("#title").val();
        let maxPower = $("#maxPower").val();
        let maxTorqueStart = $("#maxTorqueStart").val();
        let maxTorqueEnd = $("#maxTorqueEnd").val();
        if(operation == 1) client.newProfile(title,maxPower,maxTorqueStart,maxTorqueEnd);
        else client.editProfile(id, title,maxPower,maxTorqueStart,maxTorqueEnd);
    });
    $(document).on("click","[name='btnProfile']",function(e){
        let id = $(this).data("id");
        let profile = Profile.getById(id);
        if(profile == undefined){
            Toast.fire({
                icon: 'error',
                title: `Pefil não encontrado`
            });
            return;
        }

        $("#title").val(profile[0].title);
        $("#maxPower").val(profile[0].maxPower);
        $("#maxTorqueStart").val(profile[0].maxTorque.start);
        $("#maxTorqueEnd").val(profile[0].maxTorque.end);

        $("#btnSaveProfile").data("id",id);
        $("#btnSaveProfile").data("operation",2);
        $("#formProfileCard").show();
    });
    client.setup();
});
const client = {
    setup: function(){
        $("#profileCard .overlay").removeClass("d-none");

        let profiles = Profile.getAll();
        let html = "";
        $(profiles).each(function (i, item) {
            html += `<a class="btn btn-app" name="btnProfile" data-id="${item._id}"><i class="fas fa-
user"></i>${item.title}</a>`;
        });
        $("#profilesContainer").html(html);

        $("#profileCard .overlay").addClass("d-none");
    },
    newProfile: function(title,maxPower,maxTorqueStart,maxTorqueEnd){
        let data = {
            title: title,
            maxPower: maxPower,
            maxTorque: {
                start: maxTorqueStart,
                end: maxTorqueEnd
            }
        };
```

```
        Profile.newProfile(data);

        Toast.fire({
            icon: 'success',
            title: `Novo Pefil Adicionado`
        });

        client.setup();
    },
    editProfile: function(id,title,maxPower,maxTorqueStart,maxTorqueEnd){
        let data = Profile.getById(id)[0];
        data.title = title;
        data.maxPower = maxPower;
        data.maxTorque = {
            start: maxTorqueStart,
            end: maxTorqueEnd
        };

        Profile.editProfile(id, data);

        Toast.fire({
            icon: 'success',
            title: `Alterado Pefil ${title}`
        });

        client.setup();
    }
};
```

- Health.js

```
$(document).ready(function () {

    $(document).on("click", "[name=btnDevice]", function (e) {
        e.preventDefault();
        let id = $(this).data("id");
        client.selectMicrochip(id);
    });
    $(document).on("click", "[name=btnLocation]", function (e) {
        e.preventDefault();
        $("#mapCard .overlay").removeClass("d-none");

        let location = $(this).data("location");
        let html = `<iframe width='100%' height='100%' frameborder='0' scrolling='no' marginheight='0'
marginwidth='0' src='https://maps.google.com/maps?&amp;q=${encodeURIComponent( location
)}&amp;output=embed'></iframe>`
        $("#mapsContainer").html(html);
        $("#mapCard .overlay").addClass("d-none");
    });
    client.setup();

});
let client = {
    setup: function () {

        $("#devicesCard .overlay").removeClass("d-none");

        let devices = Devices.getAll();
        let html = "";
        $(devices).each(function (i, item) {
            html += `<a class="btn btn-app" name="btnDevice" data-id="${item._id}"><i class="fas fa-
microchip"></i>${item.imei}</a>`;
```

```
        });
        $("#devicesContainer").html(html);

        $("#devicesCard .overlay").addClass("d-none");
    },
    selectMicrochip: function(id){
        $("#logCard .overlay").removeClass("d-none");

        let result = Devices.getStatus(id);

        let html = `<div class="direct-chat-messages" style="height: 100%;">`;
        $(result).each(function (i, item) {
            html += `<div class="direct-chat-msg">`;
            html += `   <div class="direct-chat-infos clearfix">`;
            html += `       <span class="direct-chat-name float-left">${item.device_id}</span>`;
            html += `       <span class="direct-chat-timestamp float-right">${moment(`${item.gsm_date}
${item.gsm_time}`).format('DD/MM/YYYY h:mm:ss a')}</span>`;
            html += `   </div>`;
            html += `   <img class="direct-chat-img" src="${LTE.getResourceImage("Device")}">`;
            html += `   <div class="direct-chat-text">`;
            html += `       IP: ${item.ip} | Sinal: ${item.signal} | Local: <a href="#" data-
location="${item.location_lo},${item.location_la}" name="btnLocation">Visualizar</a> | Accuracy:
${item.location_accuracy}`;
            html += `   </div>`;
            html += `</div>`;
        });
        html += `</div>`;
        $("#logsContainer").html(html);
        $("#logCard .overlay").addClass("d-none");

    }
};
```

- Log.js

```
$(document).ready(function () {

    client.init();

    $(document).on("click","#btnRefresh",function(e){
        let selected = $("#slLogResources option:selected").val();
        if(selected == "All") client.getAllLogs();
        else client.getAllLogs({ resource: selected});
    });
    $(document).on("change","#slLogResources",function(e){
        e.preventDefault();
        let selected = $("#slLogResources option:selected").val();
        if(selected == "All") client.getAllLogs();
        else client.getAllLogs({ resource: selected});
    });
});
let client = {
    init: function(){
        $("#slLogResources").append(new Option("Todos","All"));
        let r = resources.map(x => x.id);
        $(r).each(function(i,resource){
            $("#slLogResources").append(new Option(resource,resource));
        });
        client.getAllLogs();
    },
    getAllLogs: function (filter=null) {
```

```
        $("#logCard .overlay").removeClass("d-none");
        let logs = Logs.getAll(filter);

        let html = `<div class="direct-chat-messages" style="height: 100%;">`;
        $(logs).each(function (i, item) {
            let timestamp = moment.unix(item.timestamp).add(3, 'hours');
            html += `<div class="direct-chat-msg">`;
            html += `   <div class="direct-chat-infos clearfix">`;
            html += `       <span class="direct-chat-name float-left">${item.resource}</span>`;
            html += `       <span class="direct-chat-timestamp float-right">${timestamp.format('DD/MM/YYYY
h:mm:ss a')}</span>`;
            html += `   </div>`;
            html += `   <img class="direct-chat-img" src="${LTE.getResourceImage(item.resource)}">`;
            html += `   <div class="direct-chat-text">`;
            html += `       Evento: ${item.event} | Key: ${item.key}`;
            html += `   </div>`;
            html += `</div>`;
        });
        html += `</div>`;
        $("#logsContainer").html(html);
        $("#logCard .overlay").addClass("d-none");
    }
};
```

Apêndice D – Firmware

```
#include <Arduino.h>

//Definitions
#define BAUD_RATE 115200

//Structs
struct MQTT_ITEM {
    String message;
    char* topic;
};

//Variables
const float FW_VERSION = 1.0;
const int queueSize = 50;

unsigned long healthPreviousMillis = 0;
unsigned long sensorPreviousMillis = 0;
unsigned long healthInterval = 0;
unsigned long sensorInterval = 0;

float lat = 0, lon = 0, accuracy = 0, timezone = 0;
int year = 0, month = 0, day = 0;
int hour = 0, minute = 0, sec = 0, tz = 0;

bool check_update;

int lastAdded = -1;
int lastSent = 0;
int dataLoss = 0;
int dataSent = 0;

String DEVICE_ID;
String TRAVEL_ID;
MQTT_ITEM MQTT_QUEUE[queueSize];
```

```
#include <CAN.h>

#define TIMEOUT 3000 //Seconds

const bool useStandardAddressing = true;

/*
Caso a Biblioteca Atualize
#define DEFAULT_CAN_RX_PIN GPIO_NUM_15
#define DEFAULT_CAN_TX_PIN GPIO_NUM_18
*/

String getSupportedPIDs()
{
  log_d("getting supprted PIDs");
  String PIDs = "";
  for (int pid = 0x00; pid < 0xe0; pid += 0x20)
  {
    if (useStandardAddressing) CAN.beginPacket(0x7df, 8);
    else CAN.beginExtendedPacket(0x18db33f1, 8);

    CAN.write(0x02); CAN.write(0x01); CAN.write(pid); CAN.endPacket();

    unsigned long currentMillis = millis();
    while (CAN.parsePacket() == 0 || CAN.read() < 6 || CAN.read() != 0x41 || CAN.read() != pid){
      if(millis() - currentMillis > TIMEOUT) {
        log_d("getting supprted PIDs timeout");
        break;
      }
    }

    unsigned long pidsSupported = 0;

    for (int i = 0; i < 4; i++)
    {
      pidsSupported <<= 8;
      pidsSupported |= CAN.read();
    }

    for (unsigned int i = 31; i > 0; i--)
    {
      if (pidsSupported & (1UL << i))
      {
        int pidSupported = pid + (32 - i);

        PIDs += "0x";
        if (pidSupported < 16) PIDs += "0";
        PIDs += String(pidSupported) + ",";
      }
    }

    if ((pidsSupported & 0x00000001) == 0x00000000) break;
  }
  log_d("Supported PIDs: %s",PIDs.c_str());
  return PIDs;
}
String getVINCar()
{
  log_d("getting car information");
```

```
  String VIN_Car = "";

  if (useStandardAddressing) CAN.beginPacket(0x7df, 8);
  else CAN.beginExtendedPacket(0x18db33f1, 8);

  CAN.write(0x02);CAN.write(0x09);CAN.write(0x02); CAN.endPacket();
  unsigned long currentMillis = millis();
  while (CAN.parsePacket() == 0 || CAN.read() != 0x10 || CAN.read() != 0x14 || CAN.read() != 0x49 ||
CAN.read() != 0x02 || CAN.read() != 0x01){
    if(millis() - currentMillis > TIMEOUT) {
      log_d("getting car information timeout");
      return "";
    }
  }

  while (CAN.available()) VIN_Car += (char) CAN.read();

  for (int i = 0; i < 2; i++)
  {
    if (useStandardAddressing) CAN.beginPacket(0x7e0, 8);
    else CAN.beginExtendedPacket(0x18db33f1, 8);

    CAN.write(0x30); CAN.endPacket();

    while (CAN.parsePacket() == 0 || CAN.read() != (0x21 + i));

    while (CAN.available()) VIN_Car += (char)CAN.read();
  }
  log_d("Chassi: %s",VIN_Car.c_str());
  return VIN_Car;
}
void setupCanBus()
{
  log_d("start Setup CanBus");

  log_d("starting the CAN bus at 500 kbps");

  if (!CAN.begin(500E3))
    ESP.restart();

  if (useStandardAddressing) CAN.filter(0x7e8);
  else CAN.filterExtended(0x18daf110);

  log_d("start Setup CanBus - OK");
}
float getRPM()
{
  if (useStandardAddressing) CAN.beginPacket(0x7df, 8);
  else CAN.beginExtendedPacket(0x18db33f1, 8);

  CAN.write(0x02); CAN.write(0x01); CAN.write(0x0c); CAN.endPacket();

  unsigned long currentMillis = millis();
  while (CAN.parsePacket() == 0 || CAN.read() < 3 || CAN.read() != 0x41 || CAN.read() != 0x0c) {
    if(millis() - currentMillis > TIMEOUT) {
      log_d("Read RPM timeout");
      return 0;
    }
  }
```

```
  return (float) ((CAN.read() * 256.0) + CAN.read()) / 4.0;
}
float getKmh()
{
  if (useStandardAddressing) CAN.beginPacket(0x7df, 8);
  else CAN.beginExtendedPacket(0x18db33f1, 8);

  CAN.write(0x02); CAN.write(0x01); CAN.write(0x0D); CAN.endPacket();

  unsigned long currentMillis = millis();
  while (CAN.parsePacket() == 0 || CAN.read() < 3 || CAN.read() != 0x41 || CAN.read() != 0x0d) {
   if(millis() - currentMillis > TIMEOUT) {
     log_d("Read KM/h timeout");
     return 0;
   }
  }
  return (float)CAN.read();
}
int getAirTemp()
{
  if (useStandardAddressing) CAN.beginPacket(0x7df, 8);
  else CAN.beginExtendedPacket(0x18db33f1, 8);

  CAN.write(0x02); CAN.write(0x01); CAN.write(0x46); CAN.endPacket();

  unsigned long currentMillis = millis();
  while (CAN.parsePacket() == 0 || CAN.read() < 3 || CAN.read() != 0x41 || CAN.read() != 0x46) {
   if(millis() - currentMillis > TIMEOUT) {
     log_d("Read Air Temperature timeout");
     return 0;
   }
  }
  return CAN.read() - 40;
}
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

//Definitions
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1

//Variables
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

String buffer;

bool setupOled()
{
   log_d("begin OLED");
   if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))
   {
      log_e("Error when begin OLED");
      Serial.println(F("Falha na alocação do OLED SSD1306"));
      return false;
   }
   log_d("begin OLED Competed");
   log_d("setup OLED parameters");
```

```
    display.setTextSize(1);
    display.setTextColor(WHITE);
    log_d("setup OLED parameters Completed");
    delay(100);
    return true;
}
void printOledTextSingleLine(String text,bool serial=true)
{
    //log_d("print OLED (Serial | Text): (%d | %s)",serial,text.c_str());
    if(serial) Serial.println(text);
    if(buffer == text) return;
    buffer = text;
    display.clearDisplay();
    display.setCursor(0, 0);
    display.println(text);
    display.display();
    delay(100);
}
#include <FS.h>
#include <SPIFFS.h>
#include <ArduinoJson.h>

//Variables
DynamicJsonDocument config(3500);

bool setupFile(){
  log_i("begin filesystem SPIFFS");
  if (!SPIFFS.begin(true))
  {
    log_e("SPIFFS Mount Failed");
    printOledTextSingleLine("Falha ao montar SPIFFS");
    return false;
  }
  log_i("begin filesystem completed");
  return true;
}

bool fileExist(fs::FS &fs, const char *dirname, const char *filename, uint8_t levels)
{
  log_i("if file exists: directory -> %s | filename -> %s",dirname,filename);
  bool exist = false;
  File root = fs.open(dirname);
  if (!root)
  {
    log_w("Failed to open directory");
    printOledTextSingleLine("Falha ao abrir deretório");
    return exist;
  }

  if (!root.isDirectory())
  {
    log_i("Not a directory");
    printOledTextSingleLine("Não é um diretório");
    return exist;
  }

  File file = root.openNextFile();

  while (file)
  {
```

```
    log_d("found file %s looking for %s",file.name(),filename);
    exist = strcmp(file.name(), filename) == 0;
    if (exist) break;
    file = root.openNextFile();
  }
  log_d("file exist -> %d",exist);
  return exist;
}

void listDir(fs::FS &fs, const char *dirname, uint8_t levels)
{
  log_d("Listing directory: %s", dirname);
  Serial.printf("Listing directory: %s\n", dirname);

  File root = fs.open(dirname);

  if (!root)
  {
    log_w("Failed to open directory");
    printOledTextSingleLine("Falha ao abrir diretírio");
    return;
  }
  if (!root.isDirectory())
  {
    log_i("Not a directory");
    printOledTextSingleLine("Não é um diretório");
    return;
  }

  File file = root.openNextFile();
  while (file)
  {
    if (file.isDirectory())
    {
      log_d("found directory %s",file.name());
      if (levels) listDir(fs, file.name(), levels - 1);
    }
    else
    {
      log_d("found file %s %f",file.name(),file.size());
    }
    file = root.openNextFile();
  }
}

String readFile(fs::FS &fs, const char *path)
{
  log_d("Reading file: %s", path);
  File file = fs.open(path);
  if (!file || file.isDirectory())
  {
    log_e("Failed to open file for reading");
    printOledTextSingleLine("Falha ao ler o arquivo");
    return "";
  }
  String out;
  while (file.available()) out += (char)file.read();
  return out;
}
```

```
void writeFile(fs::FS &fs, const char *path, const char *message)
{
  log_d("Writing file: %s", path);
  File file = fs.open(path, FILE_WRITE);
  if (!file)
  {
    log_e("Failed to open file for writing");
    printOledTextSingleLine("Falha ao escrever o arquivo");
    return;
  }
  if (file.print(message))
  {
    log_i("file written");
  }
  else
  {
    log_e("Write failed");
    file.close();
    SPIFFS.format();
  }
}

void appendFile(fs::FS &fs, const char *path, const char *message)
{
  log_d("appending to file: %s | %s", path, message);

  File file = fs.open(path, FILE_APPEND);
  if (!file)
  {
    printOledTextSingleLine("Falha ao acrescentar info ao arquivo");
    return;
  }
  if (file.print(message))
  {
    log_i("message appended to file");
  }
  else
  {
    log_e("failed to append message to file");
    printOledTextSingleLine("Falha ao escrever arquivo");
  }
}

void renameFile(fs::FS &fs, const char *path1, const char *path2)
{
  log_d("renaming file %s to %s", path1, path2);
  if (fs.rename(path1, path2))
  {
    log_i("file were renamed");
    printOledTextSingleLine("Arquivo Renomeado");
  }
  else
  {
    log_e("failed to rename file");
    printOledTextSingleLine("Falha ao renomear");
  }
}

void deleteFile(fs::FS &fs, const char *path)
{
```

```
  log_d("Deleting file: %s\n", path);
  if (fs.remove(path))
  {
    log_d("file deleted");
    printOledTextSingleLine("Arquivo Removido");
  }
  else
  {
    log_e("failed to delete file");
    printOledTextSingleLine("Falha ao remover arquivo");
  }
}
#define TINY_GSM_MODEM_SIM800

#include <TinyGsmClient.h>
#include <HardwareSerial.h>

//Definitions
#define MODEM_RST 5
#define MODEM_PWRKEY 4
#define MODEM_POWER_ON 23
#define MODEM_TX 27
#define MODEM_RX 26

//Variables
HardwareSerial SerialAT(1);
TinyGsm modem(SerialAT);
TinyGsmClient gsm_client(modem);
String gsm_ip,response;

const char* gsm_apn;
const char* gsm_user;
const char* gsm_pass;

void updateGSMDateTime(){

  log_d("Getting Status GSM");
  modem.sendAT(GF(""));
  modem.waitResponse(6000,response);
  log_d("Raw AT Response: %s",response.c_str());
  int16_t startOk = response.indexOf("O");
  if(response.substring(startOk,startOk+2) != "OK") {
    response = "";
    log_d("Set Date Time From GSM Network");
    modem.sendAT(GF("+CLTS=1;&W"));
    modem.waitResponse(1000,response);
    log_d("Raw AT Response: %s",response.c_str());
    ESP.restart();
  }
  response = "";

  log_d("Getting Date Time From GSM Network");
  modem.sendAT(GF("+CCLK?"));
  modem.waitResponse(6000,response);
  log_d("Raw AT Response: %s",response.c_str());

  int16_t a = response.indexOf("\"") + 1;
  int16_t b = response.indexOf("\"", a);
  response = response.substring(a, b);
  year = response.substring(0,2).toInt() + 2000;
```

```
  month = response.substring(3,5).toInt();
  day = response.substring(6,8).toInt();
  hour = response.substring(9,11).toInt();
  minute = response.substring(12,14).toInt();
  sec = response.substring(15,17).toInt();
  sec = response.substring(15,17).toInt();
  tz = response.substring(17,20).toInt() / 4;
  log_d("Date/Time: %i/%i/%i %i:%i:%i (%i)", day, month, year, hour, minute, sec, tz);
  response = "";
 }
 bool connect_gprs(){
  log_d("connceting to GPRS network");
  if(modem.isGprsConnected()) {
    log_d("modem already connected to the GPRS network");
    return true;
  }else{
    gsm_ip = "";
  }

  printOledTextSingleLine("Conctando a Internet\nAPN:"+String(gsm_apn));
  if (!modem.gprsConnect(gsm_apn, gsm_user, gsm_pass)) {
    log_e("gsm failed to connect to the GRPS network");
    ESP.restart();
  }
  return true;
 }
 bool getLocation(){
  log_d("getting date and location");
  bool result = modem.getGsmLocation(&lat, &lon, &accuracy, &year, &month, &day, &hour, &minute, &sec);

  log_d("Requesting current GSM location");
  if (result) {
     updateGSMDateTime();
     log_d("Accuracy: %f", accuracy);
     log_d("Location: %s,%s", String(lon, 10).c_str(), String(lat, 10).c_str());
     log_d("Date/Time: %i/%i/%i %i:%i:%i", day, month, year, hour, minute, sec);
  } else {
     log_d("Couldn't get GSM location");
  }
  return result;
 }
 void setup_gsm() {
  log_d("setting GSM variables");
  gsm_apn = config["gsm_apn"].as<char *>();
  gsm_user = config["gsm_user"].as<char *>();
  gsm_pass = config["gsm_pass"].as<char *>();

  log_d("config pin GSM modem");
  pinMode(MODEM_RST, OUTPUT);
  pinMode(MODEM_PWRKEY, OUTPUT);
  pinMode(MODEM_POWER_ON, OUTPUT);
  digitalWrite(MODEM_RST, HIGH);
  digitalWrite(MODEM_POWER_ON, HIGH);
  digitalWrite(MODEM_PWRKEY, HIGH);
  delay(500);
  digitalWrite(MODEM_PWRKEY, LOW);
  delay(1000);
  digitalWrite(MODEM_PWRKEY, HIGH);

  log_d("begin serial modem");
```

```
  SerialAT.begin(115200,SERIAL_8N1,MODEM_RX,MODEM_TX);
  delay(3000);

  log_d("begin GSM Modem");
  printOledTextSingleLine("Iniciando Modem GSM");

  log_d("getting Modem Informations");
  printOledTextSingleLine(modem.getModemInfo());

  log_d("waiting for operator network");
  printOledTextSingleLine("Aguardando Rede GSM");
  if (!modem.waitForNetwork()) {
    log_e("GSM Network failed");
    ESP.restart();
  }
  log_d("connected to operator network");
  printOledTextSingleLine("Conectado a rede");

  connect_gprs();
  log_i("connected to GSM Internet");

  log_i("Getting DateTime from GSM");

  printOledTextSingleLine("Setup GSM OK");
}

String getIP(){
  if(gsm_ip != "") return gsm_ip;
  gsm_ip = modem.localIP().toString();
  log_d("New IP: %s",gsm_ip.c_str());
  return gsm_ip;
}
int16_t getSignalQuality(){
  return modem.getSignalQuality();
}
#include <RTClib.h>

RTC_DS3231 rtc;
int unix_reference = -1;

bool setupRTC()
{
    if (!rtc.begin())
    {
        log_e("Couldn't find RTC");
        return false;
    }
    delay(50);
    printOledTextSingleLine("Atualizando Data e Hora");

    delay(50);
    while(year < 2021) updateGSMDateTime();

    delay(50);
    DateTime now = DateTime(year,month,day,hour,minute,sec);
    unix_reference = now.unixtime();
    rtc.adjust(now);

    return true;
}
```

```
int getCurrentUnixtime(){
   DateTime now = rtc.now();
   int currntUnix = now.unixtime();
   while(currntUnix - unix_reference < 0) currntUnix = rtc.now().unixtime();
   return currntUnix;
}
String getTimeStampString(){
   DateTime now = rtc.now();
   char buff[] = "DD/MM/YYYY hh:mm";
   const char* result = now.toString(buff);
   return String(result);
}
#include <HttpClient.h>
#include <Update.h>

void FW_Update(const char *host, int port, const char *path, const char *method)
{
   log_d("update firmaware");
   int err = 0;
   log_d("making a requesting to host (%s): %s:%i%s",method,host,port,path);
   HttpClient http(gsm_client,host,port);
   err = http.startRequest(path, method);
   if (err == 0) {
      err = http.responseStatusCode();
      log_d("Request Status: %i",err);
      if (err >= 0) {
         err = http.skipResponseHeaders();
         if (err >= 0) {
            int bodyLen = http.contentLength();
            int remaining = bodyLen;
            int totalReceived = 0;
            int received;
            uint8_t buff[2048] = {0};
            log_w("Start Update Firmware");
            printOledTextSingleLine("Iniciando Atualizacao de Firmware");
            delay(500);
            if (!Update.begin(bodyLen)) Update.printError(Serial);

            while (http.available() && remaining > 0)
            {
               printOledTextSingleLine("Atualizando..." + String(totalReceived * 100 / bodyLen) + "%");
               received = http.readBytes(buff, ((remaining > sizeof(buff)) ? sizeof(buff) : remaining));

               Update.write(buff, received);
               if (remaining > 0) remaining -= received;
               totalReceived += received;
               yield();
            }
            if (Update.end(true))
            {
               log_i("Update Successful");
               printOledTextSingleLine("Firmware Atualizado!\nReiniciando...");
               delay(500);
               ESP.restart();
            }
            else
            {
               log_e("Update Error");
               Update.printError(Serial);
            }
```

```
        }
        else
        {
            log_e("Failed to skip response headers: %i",err);
        }
    }
    else
    {
        log_e("Getting response failed: %i",err);
    }
}
else
{
    log_e("HTTP Request failed: %i",err);
}
http.stop();
return;
}

String requestBody(const char *host, int port, const char *path, const char *method)
{
    log_d("HTTP Request");
    int err = 0;
    HttpClient http(gsm_client,host, port);
    String body = "";
    log_d("making a requesting to host (%s): http://%s:%i%s",method,host,port,path);
    err = http.startRequest(path, method, NULL);
    if (err == 0) {
        err = http.responseStatusCode();
        log_d("Request Status: %i",err);
        if (err >= 0) {
            err = http.skipResponseHeaders();
            log_d("Request Skip Headers: %i",err);
            if (err >= 0) {
                int bodyLen = http.contentLength();

                while ((http.connected() || http.available()))
                {
                    if (http.available())
                    {
                        body += (char)http.read();
                        bodyLen--;
                    }
                }
                log_d("Response Body: %s",body.c_str());
            }
            else
            {
                log_e("Failed to skip response headers: %i",err);
            }
        }
        else
        {
            log_e("Getting response failed: %i",err);
        }
    }
    else
    {
        log_e("HTTP Request failed: %i",err);
    }
```

```
    http.stop();
    return body;
}
void setupFileConfig(bool reset)
{
  log_d("setting configuration file");
  if (!fileExist(SPIFFS, "/", "/config.json", 0))
  {
    printOledTextSingleLine("Arquivo de Configuracao\nconfig.json nao existe");
    log_e("File config.json does not exists");
    log_e("Please, upload configuration file");
    ESP.restart();
  }
  log_d("setting configuration file OK");
}

void getConfig()
{
  log_d("getting configuration file");
  printOledTextSingleLine("Carregando Arquivo:\nconfig.json");
  setupFileConfig(false);
  log_d("reading configuration file");
  printOledTextSingleLine("Lendo Arquivo de Configuracao");
  String configFile = readFile(SPIFFS, "/config.json");
  log_d("deserializing configuration file to JSON");
  DeserializationError err = deserializeJson(config, configFile);
  if (err)
  {
    log_e("error reading configuration: %s",err);
    delay(500);
    log_e("Removing Configuration File");
    deleteFile(SPIFFS, "/config.json");
    log_e("Restarting...");
    ESP.restart();
  }
  log_i("configuration file: %s",configFile.c_str());
  printOledTextSingleLine("Arquivo de Configuracao - OK");
}
void updateConfiguration(bool force_sync){
  log_d("updating configuration");
  bool sync = (bool)config["sync"].as<int>();

  if (sync && !force_sync) {
    log_i("skip update due variables (sync:%d | force_sync:%d)",sync,force_sync);
    return;
  }
  printOledTextSingleLine("Atualizando Configuracao");

  String response = requestBody(config["config_host"].as<char *>(), config["config_port"].as<int>(),
config["config_path"].as<char *>(), "GET");

  if (response.isEmpty())
  {
    log_e("error to connect to the server: %s",config["config_host"].as<char *>());
    setupFileConfig(true);
    delay(1000);
    ESP.restart();
  }
  log_i("rewriting configuration file with payload: %s",response.c_str());
  writeFile(SPIFFS, "/config.json", (const char *)response.c_str());
```

```
  log_d("deserializing configuration file to JSON");
  DeserializationError err = deserializeJson(config, response);
  if (err)
  {
    log_e("error reading configuration: %s",err);
    delay(500);
    log_e("Removing Configuration File");
    deleteFile(SPIFFS, "/config.json");
    log_e("Restarting...");
    ESP.restart();
  }
  log_i("updated configuration file");
  printOledTextSingleLine("Arquivo de Configuracao Atualizado");
}
void checkUpdate(String payload)
{
  log_d("checking for updates");
  if (payload.isEmpty())
  {
    log_i("no payload for update");
    payload = requestBody(config["update_host"].as<char *>(), config["update_port"].as<int>(),
config["update_path"].as<char *>(), config["update_method"].as<char *>());
  }
  DynamicJsonDocument update(300);
  if (!deserializeJson(update, payload))
  {
    if (update["version"].as<float>() > FW_VERSION)
    {
      log_d("found update (%f) for this version %f",update["version"].as<float>(),FW_VERSION);
      FW_Update(update["update_host"].as<char *>(), update["update_port"].as<int>(),
update["update_path"].as<char *>(), update["update_method"].as<char *>());
    }
    else
    {
      log_i("System is up-to-date: V%f",FW_VERSION);
      printOledTextSingleLine("Sistema Atualizado\n\nFirmware: V"+String(FW_VERSION));
    }
  }
  delay(1000);
}
#include <PubSubClient.h>

#define LOGIN_TIMEOUT 20000
//Variables
uint8_t message_char_buffer[MQTT_MAX_PACKET_SIZE];
const char *mqtt_server;
int mqtt_port;
const char *mqtt_user;
const char *mqtt_pass;
char *mqtt_sensors;
char *mqtt_register_device;
char *mqtt_register_car;
char *mqtt_login_device;
char *mqtt_login_car;
char *mqtt_health;
unsigned long loginPreviousMillis = 0;


PubSubClient client(gsm_client);
```

```
void getSensorData(){
    if (TRAVEL_ID == "") return;

    log_d("send sensor data");

    log_d("requesting sensor data");

    float rpm = getRPM();
    float kmh = getKmh();

    if(rpm == 0 || kmh == 0) return;

    log_d("creating json payload");
    StaticJsonDocument<256> json;
    json["travel_id"] = TRAVEL_ID;
    json["unix_time"] = getCurrentUnixtime();
    JsonArray sensors = json.createNestedArray("sensors");
    JsonObject sensors_0 = sensors.createNestedObject();
    sensors_0["pid"] = "0C";
    sensors_0["value"] = rpm;
    JsonObject sensors_1 = sensors.createNestedObject();
    sensors_1["pid"] = "0D";
    sensors_1["value"] = kmh;

    String payload = "";
    serializeJson(json, payload);

    lastAdded++;
    if (lastAdded >= queueSize) lastAdded = 0;
    if (MQTT_QUEUE[lastAdded].message != "") dataLoss++;
    else {
        MQTT_QUEUE[lastAdded].message = payload;
        MQTT_QUEUE[lastAdded].topic = mqtt_sensors;
    }
}
void sendHealthStatus(){
    if (DEVICE_ID == "") return;

    log_d("send health status");

    log_d("update gsm data to sent");
    if(!getLocation()) return;

    log_d("creating json payload");
    StaticJsonDocument<512> json;

    json["device_id"] = DEVICE_ID;
    json["ip"] = getIP();
    json["signal"] = getSignalQuality();
    json["gsm_date"] = String(year) + "-" + String(month) + "-" + String(day);
    json["gsm_time"] = String(hour) + ":" + String(minute) + ":" + String(sec);
    json["location_lo"] = String(lon, 8);
    json["location_la"] = String(lat, 8);
    json["location_accuracy"] = String(accuracy);
    json["data_loss"] = dataLoss;
    json["data_sent"] = dataSent;
    json["unix_time"] = getCurrentUnixtime();
    String payload = "";
    serializeJson(json, payload);
    lastAdded++;
```

```
    if (lastAdded >= queueSize) lastAdded = 0;
    if (MQTT_QUEUE[lastAdded].message != "") dataLoss++;
    else {
      MQTT_QUEUE[lastAdded].message = payload;
      MQTT_QUEUE[lastAdded].topic = mqtt_health;
    }
}
void registerDevice()
{
    if (DEVICE_ID != "") return;

    unsigned long currentMillis = millis();
    if (currentMillis - loginPreviousMillis < LOGIN_TIMEOUT) return;
    loginPreviousMillis = currentMillis;

    log_d("register device");
    log_d("creating json payload");
    StaticJsonDocument<256> json;
    json["imei"] = modem.getIMEI();
    json["ccid"] = modem.getSimCCID();
    json["imsi"] = modem.getIMSI();
    json["operator"] = modem.getOperator();
    json["board"] = "ESP32";
    json["version"] = FW_VERSION;
    json["unix_time"] = getCurrentUnixtime();
    String payload = "";
    serializeJson(json, payload);
    lastAdded++;
    if (lastAdded >= queueSize) lastAdded = 0;
    if (MQTT_QUEUE[lastAdded].message != "") dataLoss++;
    else {
      MQTT_QUEUE[lastAdded].message = payload;
      MQTT_QUEUE[lastAdded].topic = mqtt_register_device;
    }
}
void registerCar()
{
    if (DEVICE_ID == "") return;
    if (TRAVEL_ID != "") return;

    unsigned long currentMillis = millis();
    if (currentMillis - loginPreviousMillis < LOGIN_TIMEOUT) return;
    loginPreviousMillis = currentMillis;

    log_d("register car");
    log_d("creating json payload");

    String VIN = "";
    int max_try = 5;
    while(VIN == "") {
      if(max_try <= 0) {
        log_w("Could not get VIN CAR");
        ESP.restart();
      }
      log_d("Trying to get VIN CAR: %i",(5-max_try)+1);
      VIN = getVINCar();
      max_try--;
    }

    StaticJsonDocument<256> json;
```

```
    json["deviceId"] = DEVICE_ID;
    json["carVIN"] = VIN;
    json["unix_time"] = getCurrentUnixtime();
    String payload = "";
    serializeJson(json, payload);
    lastAdded++;
    if (lastAdded >= queueSize) lastAdded = 0;
    if (MQTT_QUEUE[lastAdded].message != "") dataLoss++;
    else {
        MQTT_QUEUE[lastAdded].message = payload;
        MQTT_QUEUE[lastAdded].topic = mqtt_register_car;
    }
}
void loginDevice(byte *payload)
{
    log_d("Login device");
    DynamicJsonDocument device(128);
    if (!deserializeJson(device, payload))
    {
        DEVICE_ID = device["id"].as<String>();
        log_d("unsubscribing login topic");
        boolean result = client.unsubscribe(mqtt_login_device);
        log_d("unsubscribe login topic: %d", result);
        log_d("Calling Register Car");
        loginPreviousMillis = millis() + LOGIN_TIMEOUT;
        registerCar();
    }
    else
    {
        log_e("error to deserialize login payload");
        registerDevice();
    }
    delay(500);
}
void loginCar(byte *payload)
{
    log_d("Login car");
    DynamicJsonDocument device(128);
    if (!deserializeJson(device, payload))
    {
        TRAVEL_ID = device["travel_id"].as<String>();
        log_d("unsubscribing login topic");
        boolean result = client.unsubscribe(mqtt_login_car);
        log_d("unsubscribe login topic: %d", result);
    }
    else
    {
        log_e("error to deserialize login payload");
        registerCar();
    }
    delay(500);
}
void mqttCallback(char *topic, byte *payload, unsigned int length)
{
    log_d("MQTT Callback");
    String strPayload = "";

    for (int i = 0; i < length; i++) strPayload += (char)payload[i];
    log_d("Receive MQTT message (topic: %s | payload: %s)", topic, strPayload.c_str());
```

```
    if (strcmp(topic, mqtt_login_device) == 0) loginDevice(payload);
    else if (strcmp(topic, mqtt_login_car) == 0) loginCar(payload);
    else log_e("Topic is not implemented %s", topic);
}
void setupMQTT()
{
    log_d("setting up MQTT");
    printOledTextSingleLine("Configurando MQTT");

    mqtt_server = config["mqtt_server"].as<char *>();
    mqtt_port = config["mqtt_port"].as<int>();
    mqtt_user = config["mqtt_user"].as<char *>();
    mqtt_pass = config["mqtt_pass"].as<char *>();

    log_i("MQTT_Server: mqtt://%s:%i", mqtt_server, mqtt_port);
    client.setServer(mqtt_server, mqtt_port);
    log_d("Setting MQTT Callback");
    client.setCallback(mqttCallback);
    printOledTextSingleLine("Configurando MQTT - OK");
}
void connectMQTT()
{
    while (!client.connected())
    {
        log_d("MQTT is not connected");
        connect_gprs();

        log_d("Connecting to MQTT server");
        if (client.connect(DEVICE_ID.c_str(), mqtt_user, mqtt_pass))
        {
            log_i("MQTT Connected");
            JsonArray topics = config["mqtt_topics"].as<JsonArray>();
            log_d("MQTT subscribing to topics");
            for (int x = 0; x <= topics.size() - 1; x++)
            {
                const char *method = topics[x]["method"].as<const char *>();
                const char *isFor = topics[x]["for"].as<const char *>();

                if (strcmp(isFor, "register_device") == 0) {
                    if(DEVICE_ID != "") continue;
                    mqtt_register_device = (char *)topics[x]["topic"].as<char *>();
                }
                else if (strcmp(isFor, "register_car") == 0) {
                    if(TRAVEL_ID != "") continue;
                    mqtt_register_car = (char *)topics[x]["topic"].as<char *>();
                }
                else if (strcmp(isFor, "sensors") == 0) mqtt_sensors = (char *)topics[x]["topic"].as<char *>();
                else if (strcmp(isFor, "login_device") == 0) mqtt_login_device = (char *)topics[x]["topic"].as<char
*>();
                else if (strcmp(isFor, "login_car") == 0) mqtt_login_car = (char *)topics[x]["topic"].as<char *>();
                else if (strcmp(isFor, "health") == 0) mqtt_health = (char *)topics[x]["topic"].as<char *>();
                if (strcmp(method, "publish") == 0) continue;

                const char *topic = topics[x]["topic"].as<const char *>();
                boolean result = client.subscribe(topic);

                log_i("MQTT topic %s | result: %d", topic, result);

                delay(100);
            }
```

```
        }
        else
        {
            log_e("MQTT connecting - Error: %s", client.state());
            delay(500);
        }
    }
    if(DEVICE_ID == "") registerDevice();
    if(TRAVEL_ID == "") registerCar();
    client.loop();
}
void sendMQTTData(void * pvParameters){
    log_i("Start Send Data");

    while (true)
    {
        connectMQTT();
        for (int x = lastSent; x < queueSize; x++)
        {
            if (lastSent == (queueSize - 1)) lastSent = 0;
            if (MQTT_QUEUE[x].message == "") continue;
            const char *c_payload = MQTT_QUEUE[x].message.c_str();
            log_d("payload: %s", c_payload);
            int p_length = strlen(c_payload);
            log_d("publishing payload to topic: %s", MQTT_QUEUE[x].topic);
            boolean result = false;
            while(!result){
                result = client.publish(MQTT_QUEUE[x].topic, c_payload, p_length);
                if (result) log_i("payload sent");
                else {
                    connectMQTT();
                    log_e("failed to publish sent payload");
                }
            }
            MQTT_QUEUE[x].message = "";
            MQTT_QUEUE[x].topic = "";
            // memset(MQTT_QUEUE[x].topic, 0, sizeof(MQTT_QUEUE[x].topic));
            dataSent++;
            lastSent = x;
        }
    }
    vTaskDelay(5);
}

void setup()
{
    log_d("begin setup");
    Serial.begin(BAUD_RATE);
    log_d("setup serial baud rate %i", BAUD_RATE);

    log_d("begin setup CanBus");
    setupCanBus();
    log_d("begin setup CanBus - OK");

    log_d("setup OLED Screen");
    if (!setupOled()) ESP.restart();
    log_d("setup OLED Screen Completed");

    printOledTextSingleLine("Iniciando Sistema");
```

```
    log_d("configuring file system");
    if (!setupFile()) ESP.restart();
    log_d("configuration file system completed");

    log_d("getting Configuration File");
    getConfig();
    log_d("getting Configuration File - OK");

    log_d("configuring GSM");
    setup_gsm();
    log_d("configuration GSM - OK");

    log_d("setup RTC");
    if (!setupRTC()) ESP.restart();
    log_d("setup RTC Completed");

    log_d("update Configuration");
    updateConfiguration(false);
    log_d("update Configuration - OK");

    log_d("checking for updates");
    check_update = config["check_update"].as<bool>();
    if(check_update) checkUpdate("");
    log_d("checking for updates - OK");

    log_d("setup MQTT");
    setupMQTT();
    log_d("setup MQTT - OK");

    log_d("setting variables");
    healthInterval = config["health_interval"].as<long>();
    sensorInterval = config["sensor_interval"].as<long>();
    log_d("setting variables - OK");

    disableCore0WDT();

    xTaskCreatePinnedToCore(sendMQTTData, "Send MQTT Data", 5000, NULL, 0, NULL, 0);

    log_w("begin setup complete");
    delay(100);
}
void loop()
{
    unsigned long currentMillis = millis();

    if (currentMillis - healthPreviousMillis >= healthInterval)
    {
        sendHealthStatus();
        healthPreviousMillis = currentMillis;
    }

    if (currentMillis - sensorPreviousMillis >= sensorInterval){
        getSensorData();
        sensorPreviousMillis = currentMillis;
    }
    String text = getTimeStampString() + "\n\n";
    text += "Pacotes Enviados: " + String(dataSent) + "\n";
    text += "Pacotes Perdidos: " + String(dataLoss) + "\n\n";
    text += "Device Registrado: " + String(DEVICE_ID != "" ? "S":"N") + "\n";
    text += "Carro Registrado: " + String(TRAVEL_ID != "" ? "S":"N");
```

```
   printOledTextSingleLine(text,false);
}
```

Apêndice E – Tabelas de definição da API (Node-Red)

Tabela 11 - API - *Update*

Campo	Descrição	Tipo
Version	Última Versão Disponível para Download	Decimal

Fonte: Autoria Própria

Tabela 12 - API - *Configuration*

Campo	Descrição	Tipo
sync	Última Versão Disponível para Download	Decimal
config_host	Endereço URL para API de Configuração	Texto
config_port	Porta da URL de API de Configuração	Inteiro
config_path	Endpoint da URL de API de Configuração	Texto
mqtt_server	Endereço URL para Broker MQTT	Texto
mqtt_port	Porta da URL do Broker MQTT	Inteiro
mqtt_user	Usuário do Broker MQTT	Texto
mqtt_pass	Senha do Broker MQTT	Texto
check_update	Verificação de Atualização de Firmware	Inteiro
update_host	Endereço URL da API de Atualização de Firmware	Texto
update_port	Porta da URL da API de Atualização	Inteiro
update_path	Endpoint da URL da API de Atualização	Texto
update_method	Verbo HTTP do Endpoint de Atualização	Texto
health_interval	Intervalo de Envio de Mensagem Health (Vide Tabela 7)	Inteiro
sensor_interval	Intervalor de Envio de Mensagem Sensor (Vide Tabela 8)	Inteiro
gsm_apn	Endereço APN da rede GSM	Texto
gsm_user	Usuário APN da rede GSM	Texto
gsm_pass	Senha APN da rede GSM	Texto
mqtt_topics	Array de tópicos MQTT	Array
mqtt_topics/method	Método de uso do Tópico MQTT (*Publish* ou *Subscribe*)	Texto
mqtt_topics/mqtt_topics	Nome do Tópico MQTT	Texto
mqtt_topics/topic	Endpoint do Tópico	Texto

Fonte: Autoria Própria

Tabela 13 - API - *Devices* - Listagem - Saída

Campo	Descrição	Tipo
_id	Identificação do registro no banco de dados	Texto
IMEI	IMEI da placa GSM SIM800L	Texto
CCID	CCID da placa GSM SIM800L	Texto
IMSI	IMSI da placa GSM SIM800L	Texto
Operator	Operadora GSM do SIM Card	Texto
Board	Placa Utilizada (Fixo: ESP32)	Texto
Version	Versão do Firmware	Decimal
Unix_Time	Data e Hora do registro (Formato Unix)	Inteiro
Cars	Array de veículos cadastrados	Array
Cars/carVIN	VIN do Carro	Texto
Cars/Unix_Time	Data e Hora do Registro (Formato Unix)	Inteiro

Fonte: Autoria Própria

Tabela 14 - API - *Device* – Listagem - Entrada

Campo	Descrição	Tipo
ID	Identificação do Dispositivo	Texto

Fonte: Autoria Própria

Tabela 15 - API - *Device* - Listagem - Saída

Campo	Descrição	Tipo
_id	Identificação do registro no banco de dados	Texto
IMEI	IMEI da placa GSM SIM800L	Texto
CCID	CCID da placa GSM SIM800L	Texto
IMSI	IMSI da placa GSM SIM800L	Texto
Operator	Operadora GSM do SIM Card	Texto
Board	Placa Utilizada (Fixo: ESP32)	Texto
Version	Versão do Firmware	Decimal
Unix_Time	Data e Hora do registro (Formato Unix)	Inteiro
Cars	Array de veículos cadastrados	Array
Cars/carVIN	VIN do Carro	Texto
Cars/Unix_Time	Data e Hora do Registro (Formato Unix)	Inteiro

Fonte: Autoria Própria

Tabela 16 - API - *Device* - *Status* - Entrada

Campo	Descrição	Tipo
ID	Identificação do Dispositivo	Texto

Fonte: Autoria Própria

Tabela 17 - API - *Device* - *Status* - Saída

Campo	Descrição	Tipo
Device_ID	Código do Dispositivo (Vide Tabela 4)	Texto
IP	IP da Conexão GSM	Texto
Signal	Força do Sinal GSM	Inteiro
GSM_Date	Data na Rede GSM	Texto
GSM_Time	Hora na Rede GSM	Texto
Location_LO	Longitude da Triangulação GSM	Decimal
Location_LA	Latitude da Triangulação GSM	Decimal
Location_Accuracy	Precisão da Geolocalização (em metros)	Decimal
Data_Loss	Quantidade de Mensagens Perdidas	Inteiro
Data_Sent	Quantidade de Mensagens Enviadas	Inteiro
Unix_Time	Data e Hora do Registro (Formato Unix)	Inteiro

Fonte: Autoria Própria

Tabela 18 - API - *Cars* - Listagem - Entrada

Campo	Descrição	Tipo
ID	Identificação do Dispositivo	Texto

Fonte: Autoria Própria

Tabela 19 - API - *Cars* - Listagem - Saída

Campo	Descrição	Tipo
_id	Identificação do registro no banco de dados	Texto
IMEI	IMEI da placa GSM SIM800L	Texto
CCID	CCID da placa GSM SIM800L	Texto
IMSI	IMSI da placa GSM SIM800L	Texto
Operator	Operadora GSM do SIM Card	Texto
Board	Placa Utilizada (Fixo: ESP32)	Texto
Version	Versão do Firmware	Decimal
Unix_Time	Data e Hora do registro (Formato Unix)	Inteiro
Cars	Array de veículos cadastrados	Array
Cars/carVIN	VIN do Carro	Texto
Cars/Unix_Time	Data e Hora do Registro (Formato Unix)	Inteiro

Fonte: Autoria Própria

Tabela 20 - API - *Car* - Listagem - Entrada

Campo	Descrição	Tipo
ID	Identificação do Veículo	Texto

Fonte: Autoria Própria

Tabela 21 - API - *Car* - Listagem - Saída

Campo	Descrição	Tipo
_id	Identificação do registro no banco de dados	Texto
IMEI	IMEI da placa GSM SIM800L	Texto
CCID	CCID da placa GSM SIM800L	Texto
IMSI	IMSI da placa GSM SIM800L	Texto
Operator	Operadora GSM do SIM Card	Texto
Board	Placa Utilizada (Fixo: ESP32)	Texto
Version	Versão do Firmware	Decimal
Unix_Time	Data e Hora do registro (Formato Unix)	Inteiro
Car/carVIN	VIN do Carro	Texto
Car/Unix_Time	Data e Hora do Registro (Formato Unix)	Inteiro

Fonte: Autoria Própria

Tabela 22 - API - *Car* - Viagens - Entrada

Campo	Descrição	Tipo
ID	Identificação do Veículo	Texto

Fonte: Autoria Própria

Tabela 23 - API - *Car* - Viagens - Saída

Campo	Descrição	Tipo
_id	Identificação da viagem no banco de dados	Texto
Id	Identificação da viagem no dispositivo	Texto
carVIN	VIN do Carro	Texto
deviceID	Identificação do dispositivo no banco de dados	Texto
Timestamp	Data e Hora do Registro (Formato Unix)	Inteiro
Records	Quantidade de registros de Sensores	Inteiro

Fonte: Autoria Própria

Tabela 24 - API - *Travels* - Listagem - Saída

Campo	Descrição	Tipo
_id	Identificação da viagem no banco de dados	Texto
Id	Identificação da viagem no dispositivo	Texto
carVIN	VIN do Carro	Texto
deviceID	Identificação do dispositivo no banco de dados	Texto
Timestamp	Data e Hora do Registro (Formato Unix)	Inteiro
Records	Quantidade de registros de Sensores	Inteiro

Fonte: Autoria Própria

Tabela 25 - API - *Travel* - Listagem - Entrada

Campo	Descrição	Tipo
ID	Identificação da Viagem	Texto

Fonte: Autoria Própria

Tabela 26 - API - *Travel* - Listagem - Saída

Campo	Descrição	Tipo
_id	Identificação da viagem no banco de dados	Texto
Id	Identificação da viagem no dispositivo	Texto
carVIN	VIN do Carro	Texto
deviceID	Identificação do dispositivo no banco de dados	Texto
Timestamp	Data e Hora do Registro (Formato Unix)	Inteiro
Records	Quantidade de registros de Sensores	Inteiro

Fonte: Autoria Própria

Tabela 27 - API - *Travel* - Sensores - Entrada

Campo	Descrição	Tipo
ID	Identificação da Viagem	Texto

Fonte: Autoria Própria

Tabela 28 - API - *Travel* - Sensores - Saída

Campo	**Descrição**	**Tipo**
_id	Identificação do Sensor no banco de dados	Texto
Pid	PID do Sensor	Texto
Value	Valor do Sensor	Decimal
car_status	Estado do Carro (*Moving* ou *Stop*)	Texto
receive_date	Data e Hora da Leitura (Formato Unix)	Inteiro
travel_id	Id da Viagem no banco de dados	Texto
Behaviour	Pontuação da Condução	Inteiro

Fonte: Autoria Própria

Tabela 29 - API - *Logs* – Listagem – Saída

Campo	**Descrição**	**Tipo**
_id	Identificação do Log no banco de dados	Texto
Timestamp	Data e Hora do registro (Formato Unix)	Inteiro
Event	Evento que gerou o Log	Texto
Resource	Origem do Log	Texto
Key	Identificação Relacionado ao *Resource*	Texto

Fonte: Autoria Própria

Tabela 30 - API - *Profiles* - Listagem - Saída

Campo	**Descrição**	**Tipo**
_id	Identificação do Perfil no banco de dados	Texto
Title	Nome do Perfil	Texto
maxPower	RPM Máxima de Potência	Inteiro
MaxTorque/Start	RPM Máximo Início Torque	Inteiro
MaxToque/End	TPM Máximo Fim Torque	Inteiro

Fonte: Autoria Própria

Tabela 31 - API - *Profile* - Listagem - Entrada

Campo	**Descrição**	**Tipo**
ID	Identificação do Perfil	Texto

Fonte: Autoria Própria

Tabela 32 - API - *Profile* - Listagem - Saída

Campo	**Descrição**	**Tipo**
_id	Identificação do Perfil no banco de dados	Texto
Title	Nome do Perfil	Texto
maxPower	RPM Máxima de Potência	Inteiro
MaxTorque/Start	RPM Máximo Início Torque	Inteiro
MaxToque/End	TPM Máximo Fim Torque	Inteiro

Fonte: Autoria Própria

Tabela 33 - API - *Profile* - Cadastro - Entrada

Campo	**Descrição**	**Tipo**
Title	Nome do Perfil	Texto
maxPower	RPM Máxima de Potência	Inteiro
MaxTorque/Start	RPM Máximo Início Torque	Inteiro
MaxToque/End	TPM Máximo Fim Torque	Inteiro

Fonte: Autoria Própria

Tabela 34 - API - *Profile* - Atualização - Entrada

Campo	**Descrição**	**Tipo**
Title	Nome do Perfil	Texto
maxPower	RPM Máxima de Potência	Inteiro
MaxTorque/Start	RPM Máximo Início Torque	Inteiro
MaxToque/End	TPM Máximo Fim Torque	Inteiro

Fonte: Autoria Própria

Tabela 35 - API - *Car* - Alteração - Entrada

Campo	**Descrição**	**Tipo**

IMEI	IMEI da placa GSM SIM800L	Texto
CCID	CCID da placa GSM SIM800L	Texto
IMSI	IMSI da placa GSM SIM800L	Texto
Operator	Operadora GSM do SIM Card	Texto
Board	Placa Utilizada (Fixo: ESP32)	Texto
Version	Versão do Firmware	Decimal
Unix_Time	Data e Hora do registro (Formato Unix)	Inteiro
Cars	Array de veículos cadastrados	Array
Cars/carVIN	VIN do Carro	Texto
Cars/Unix_Time	Data e Hora do Registro (Formato Unix)	Inteiro

Fonte: Autoria Própria