Universidade de Aveiro
2022

Ricardo Félix Abrantes

Plataforma de monitorização remota da condução e dos parâmetros dos veículos
Plataform for remote tracking of driving and vehicle parameters

Universidade de Aveiro
2022

Ricardo Félix Abrantes

Plataforma de monitorização remota da condução e dos parâmetros dos veículos
Plataform for remote tracking of driving and vehicle parameters

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestrado em Engenharia Mecânica, realizada sob orientação científica de José Paulo Oliveira Santos, Professor Auxiliar do Departamento de Engenharia Mecânica da Universidade de Aveiro e de Susana Isabel Barreto de Miranda Sargento, Professora Catedrática do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

Esta dissertação teve o apoio dos projetos UIDB/00481/2020 e UIDP/00481/2020 - Fundação para a Ciência e a Tecnologia; do CENTRO-01-0145 FEDER-022083 - Programa Operacional Regional do Centro (Centro2020), através do Portugal 2020; e do PAC Portugal AutoCluster for the Future (POCI-01-0247-FEDER-046095), financiado pelo Fundo Europeu de Desenvolvimento Regional, COMPETE2020, e Portugal2020.

O júri / The jury

Presidente / President	**Professora Doutora Margarida Isabel Cabrita Marques Coelho** Professora Auxiliar com Agregação da Universidade de Aveiro
Vogais / Committee	**Professor Doutor José Paulo Oliveira Santos** Professor Auxiliar da Universidade de Aveiro (orientador)
	Professor Doutor Rui Manuel Escadas Ramos Martins Professor Auxiliar da Universidade de Aveiro

Agradecimentos / Acknowledgements

Quero agradecer a todos os que de alguma forma ajudaram na realização deste trabalho.
Aos meus orientadores José Paulo Santos e Susana Isabel Barreto de Miranda Sargento por toda a ajuda e paciência que tiveram para comigo.
Aos meus pais pelo apoio e incentivo durante todo tempo.
Aos meus avós por tudo o que me ensinaram.
À minha irmã por ajudar sempre que é preciso.
À Margarida pelo carinho.
Ao Tiago por todas as ideias e experiências.
À professora Deolinda por me fazer entender a matemática.
A todos os meus amigos que fui fazendo durante este percurso.

Palavras-chave Monitorização de Veículos; Interface Web; Localização; Centralina; Distância

Resumo

A monitorização de veículos tem uma vasta aplicabilidade no controlo, gestão e conhecimento de transportes. A existência de uma plataforma única e universal para realizar e armazenar os dados de monitorizações é vantajosa uma vez que deixa de ser necessário criar métodos de recolha e armazenamento de dados específicos para cada projeto. Sendo também que desta forma o armazenamento dos dados recolhidos durante as monitorizações é feito de forma padronizada.

Nesta dissertação foi desenvolvida uma plataforma de monitorização flexível e escalável com base em tecnologias da Internet com o objetivo de servir diversas finalidades de forma fácil e rápida execução, como por exemplo estimar as emissões e consumos.

A solução implementada consiste numa plataforma de recolha de informação de dados de veículos. A informação a ser adquirida a partir dos veículos depende da aplicação, e poderá ser alterada modificando os sensores. A informação é recolhida e processada para enviar através de um microprocessador presente no veículo. De forma a não restringir os movimentos dos veículos a comunicação com a plataforma é feita sem fios. O servidor está programado em PHP para gerir a recolha de dados, toda a informação é armazenada numa base de dados SQL e a apresentação da informação ao utilizador é feita com recurso a HTML e Javascript.

Esta ferramenta foi utilizada para monitorizar a posição, velocidade instantânea e velocidade de rotação do motor durante uma viagem de carro tendo os valores sido utilizados para cálculos de consumo.

O repositório do código da plataforma está disponível para acesso, podendo a plataforma vir a ser utilizada para quaisquer fins.

Keywords Vehicle Tracking; Web Interface; Position; ECU; Distance

Abstract

Vehicle tracking has wide applicability in control, management and research. The existence of a single and universal platform to acquire and store data from vehicle tracking is beneficial as there is no longer a need to create specific data collection methods and data storage for each project. Also, in this way, the storage of data collected during monitoring is done in a standardized way. In this masters thesis a flexible and scalable tracking platform was developed based on Internet technologies with the aim of serving diverse goals easily and quickly, such as estimate emissions and consumption.
The implemented solution consists of a data collection platform for vehicles. The tracked data from the vehicles depends on the application, and it can be modified by changing the sensors. The information is collected and processed to be sent through a microprocessor present in the vehicle. In order not to compromise the movements of the vehicles, communication with the platform is done wirelessly. The server is programmed in PHP to manage the data collection, all the data collection Service in an SQL database and the presentation of information to the user is made using HTML and Javascript. This tool was used to monitor the position, instantaneous speed and the engine speed during a car journey and the values were used for consumption calculations. The platform code is open source so that the platform may be used for any purpose.

Índice

Lista de Tabelas

Lista de Figuras

Lista de Acrónimos

AJAX Asynchronous JavaScript And XML

API Application Programming Interface

CA Certificate Authority

CAN Controller Area Network

CSS Cascading Style Sheets

CSV Comma-Separated Values

DNS Domain Name System

EEPROM Electrically Erasable Programmable Read-Only Memory

GNSS Global Navigation Satellite System

GPIO General-Purpose Input/Output

GPRS General Packet Radio Service

GPS Global Positioning System

GSM Global System for Mobile Communications

HTML HyperText Markup Language

HTTP HyperText Transfer Protocol

HTTPS HyperText Transfer Protocol Secure

I^2C Inter-Integrated Circuit

IC Integrated Circuit

IDE Integrated Development Environment

IP Internet Protocol

ISO International Organization for Standardization

JS JavaScript

JSON JavaScript Object Notation

JWT JSON Web Token

NMEA National Marine Electronics Association

OBD On-Board Diagnostics

OOP Object-Oriented Programming

PHP PHP: Hypertext Preprocessor

PID Parameter ID

QL Query Language

REST Representational State Transfer

SHA Secure Hash Algorithm

SPA Single-Page Application

SPI Serial Peripheral Interface

SQL Structured Query Language

SSID Service Set Identifier

SSL Secure Sockets Layer

TCP Transmission Control Protocol

TLS Transport Layer Security

TPMS Tire Pressure Monitoring System

UART Universal Asynchronous Receiver/Transmitter

UML Unified Modeling Language

URL Uniform Resource Locator

USB Universal Serial Bus

VSP Vehicle-Specific Power

Capítulo 1

Introdução

1.1 Enquadramento

A monitorização de veículos é uma ferramenta necessária para adquirir dados usados, por exemplo, para fins de investigação. Esta monitorização permite a previsão de avarias, controlo de frotas, entre outros. A existência de uma plataforma única e universal para realizar e armazenar os dados de monitorizações é vantajosa uma vez que deixa de ser necessário criar métodos de recolha e armazenamento de dados específicos para cada projeto.

Um estudo realizado acerca do caudal dos gases de escape de veículos ligeiros de passageiros em condições de trabalho diferentes usou o equipamento *OEM-2100* para monitorizar as emissões de um veículo numa determinada rota. Os dados recolhidos eram depois armazenados e formatados numa folha de cálculo *Excel* e um programa informático correlacionava as medições com o momento em que foram efetuadas. [1]

Outro estudo com o objetivo de determinar se ferramentas de *croudsourcing* podem ser usadas como uma fonte de informação viável para determinar a intensidade de trânsito em certas faixas de rodagem necessitou de monitorizar, através de um Global Navigation Satellite System (GNSS) *data logger*, a velocidade e aceleração de um veículo na faixa de rodagem para estimar as emissões que cada veículo que ali circula produz. [2]

No artigo [3] foram monitorizados, a partir do Controller Area Network (CAN) bus do veículo, a aceleração longitudinal, transversal e a direção, com o intuito de determinar a ação que o condutor executa num determinado momento, como aceleração, travagem ou mudança de direção, e a sua intensidade. Tendo por objetivo classificar o estilo de condução e informar o condutor de forma a reduzir a quantidade de manobras perigosas.

Para averiguar o impacto da distância transversal, durante a manobra de ultrapassagem entre um veículo motorizado e uma bicicleta, em termos de consumos energéticos e na segurança rodoviária, realizou-se a monitorização de uma bicicleta e de um veículo motorizado. No veículo e na bicicleta foram monitorizadas a velocidade e a aceleração de forma a determinar a potência especifica que, através de cálculos intermédios, avalia o consumo energético num dado percurso com a intensidade de transito do instante monitorizado. Na bicicleta foi ainda monitorizada a distância lateral e foi também montada uma câmara. [4]

A sua utilização poderá também ser feita por mecânicos automóvel no auxilio do diagnóstico de avarias ou pelos próprios utilizadores dos veículos para controlar os consumos, encontrar rotas mais económicas ou encontrar sintomas de mal funcionamento.

Um exemplo de uma plataforma usada para este fim é a *Caruso*. Foi criada por uma empresa com o intuito de monitorizar diversos parâmetros de veículos de uma forma padronizada, através de uma Application Programming Interface (API) bem definida. Nesta plataforma é possível configurar os parâmetros de veículos de várias marcas como, por exemplo, o nível de combustível ou carga da bateria para veículos elétricos, a pressão dos pneus, a velocidade do veículo, rotações por minuto do motor, entre outros. [10] A informação é usada, por exemplo, para controlar a manutenção preventiva ou detetar avarias, agendar abastecimentos/carregamentos, controlar uma frota de veículos ou detetar acidentes e contactar serviços de emergência automaticamente.

Uma plataforma única e universal para realizar e armazenar os dados destas monitorizações seria benéfica uma vez que deixaria de ser necessário criar métodos de recolha e armazenamento de dados específicos para cada projeto.

1.2 Objetivos

Esta dissertação tem por objetivo desenvolver uma plataforma que permita:

- Aquisição de dados adquiridos no veículo para posterior tratamento;
- Visualização dos dados obtidos remotamente através de uma interface gráfica;
- A sua utilização após a conclusão da dissertação;
- Acomodar simultaneamente diferentes tipos de veículos com múltiplos parâmetros para monitorizar;
- Permitir a adição posterior de mais parâmetros para monitorizar.

A plataforma deverá ser totalmente desenvolvida com linguagens e protocolos web atuais. Esta poderá ser acedida através de web browsers que todos utilizadores saibam usar.

Nesta plataforma poderão ser utilizados diversos sensores, sendo cada um deles escolhido e programado de acordo com a função a desenvolver. O *hardware* deverá ser baseado em sensores e placas de desenvolvimento disponiveis para compra.

Preferencialmente a plataforma deverá ser desenvolvida de acordo com a norma International Organization for Standardization (ISO) 20078 - Extended Vehicle Interface, de forma a respeitar a privacidade e segurança da informação dos utilizadores.

1.3 Organização do documento

Esta dissertação é composta por seis capítulos, sendo de referir que estrangeirismos foram grafados em itálico e que todo o código incluído em frases se encontra escrito com o mesmo estilo de letra com que é normalmente apresentado no código.

No capítulo 2 são expostos e analisados outros trabalhos realizados anteriormente no mesmo âmbito desta dissertação com vista a verificar as soluções já encontradas e aspetos a melhorar de forma a servirem de base a este trabalho para construir uma melhor solução proposta.

No capítulo 3 são apresentadas as soluções desenvolvidas para a implementação da plataforma, tais como os seus elementos essenciais, os modelos para a comunicação entre os componentes, as estruturas para a implementação do software, diagramas de funcionamento de alguns componentes e esboços da interface visual da plataforma.

O capítulo 4 refere-se à implementação da solução proposta onde é explicado como foram implementados os componentes da plataforma e são expostas partes relevantes do código para melhor compreensão do que foi realizado. Neste capítulo é também definido o caso de estudo usado para a recolha de dados discutida no capitulo seguinte.

No capítulo 5 são apresentados e discutidos os resultados obtidos durante a execução do caso de estudo e são apresentadas métricas de desempenho da plataforma na recolha de dados.

O capítulo 6 é uma revisão geral do trabalho realizado nesta dissertação e sugestões para futuros trabalhos na mesma área.

Capítulo 2

Estado de Arte

2.1 Revisão Bibliográfica

Este capitulo destina-se a analisar e descrever outros trabalhos desenvolvidos no mesmo ambito desta dissertação. Serão revistos com o intuito de encontrar aspetos a melhorar que possam ser adicionados neste trabalho.

O autor [6] desenvolveu na sua dissertação uma aplicação Android que permite a recolha de informações da centralina do veiculo e envia os dados recolhidos para uma base de dados em tempo real. Teve como intuito a rastreabilidade dos veículos de forma a reduzir os consumos de combustível e também assegurar a qualidade das mercadorias perecíveis transportadas.

A recolha da informação foi feita a partir de um *smartphone*. Os dados da centralina foram obtidos através do Integrated Circuit (IC) *ELM327*, transmitidos por *bluetooth* para o *smartphone* e a posição do veículo foi obtida diretamente do sensor Global Positioning System (GPS) do *smartphone*. Todos os dados foram enviados para a base de dados através da rede de dados móvel acedida pelo *smartphone*. A figura 2.1 esquematiza o fluxo da informação proposto pelo autor.

Para isto ser possível foi desenvolvida uma aplicação *Android* que permite ao utilizador configurar as informações obtidas da centralina, ver a posição atual num mapa e enviar mensagens de texto por Transmission Control Protocol (TCP) para o servidor web. Em *background* esta aplicação efetuava as comunicações entre o *ELM327*, o sensor GPS do *smartphone* e a interface web.

A interface web que recebe a informação foi desenvolvida em Visual Basic, abre uma *socket* TCP no servidor que fica "à escuta"de uma série de caracteres que identifica o tipo de informação recebida e os valores associados. Esta interface armazena então os dados na base de dados através de instruções Structured Query Language (SQL). O utilizador poderá consultar todos os dados presentes na base de dados através de outra interface web desenvolvida em PHP: Hypertext Preprocessor (PHP), que apresenta os dados em forma de tabela ou mapa da *Google*.

Noutra dissertação realizada por [7] foi proposto o desenvolvimento de um sistema de monitorização de veículos de mercadorias que permita obter a localização através de um sensor GPS, registar a abertura/fecho de portas e a temperatura da mercadoria transportada no atrelado do veículo.

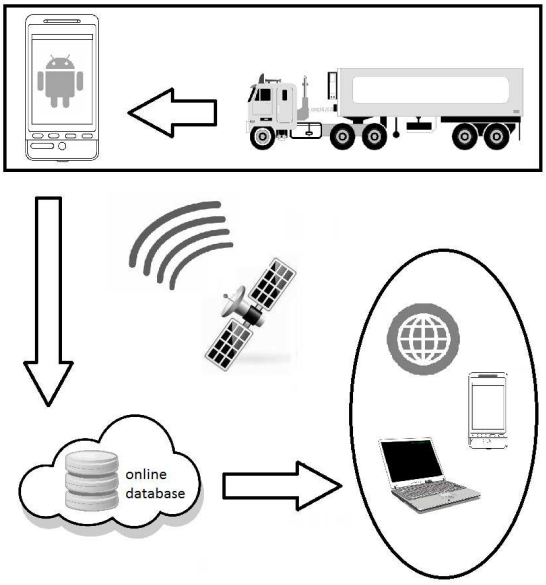

Figura 2.1: Diagrama da solução proposta por Ricardo Afonso [6]

A recolha da informação foi feita por dois módulos, um para o trator e o outro para o atrelado. Os módulos consistiam numa placa desenvolvida pelo autor, cada uma composta pelos sensores necessários e um microprocessador. O módulo do trator comunica com o módulo do reboque, com o sensor GPS e com a placa de comunicação General Packet Radio Service (GPRS). O módulo do reboque mede a temperatura no interior, verifica o estado da porta e comunica as leituras ao módulo do trator por radiofrequência. A informação recolhida é enviada via GPRS a partir do módulo do trator para uma base de dados externa instalada num servidor central através da Internet, como ilustra a figura 2.2.

A interface web foi desenvolvida em HyperText Markup Language (HTML), PHP e JavaScript (JS) e é servida por *Apache*. A informação é enviada do veículo como um pedido HTTP pelo método *GET* e é interpretada pelo script PHP que, através dos parâmetros passados no URL, identifica o veículo, cliente e rota, selecionando assim na base de dados a tabela apropriada onde deverá armazenar as informações recebidas.

Para aceder ao site o cliente necessita de se autenticar como utilizador ou administrador. O cliente tem acesso aos dados registados, podendo observar o percurso realizado no mapa e a variação da temperatura ao longo do percurso. Para além destas funcionalidades um administrador pode ainda adicionar clientes e percursos.

Na dissertação de mestrado [8] foi desenvolvido um sistema para monitorizar a posição, velocidade e temperatura de veículos de aluguer. O objetivo da dissertação foi estudar a influência de algoritmos de envio de informação no consumo de energia do microcontrolador.

A unidade presente no veículo era constituída por uma placa *Arduino* com um módulo *shield* GPRS/GPS e um sensor de temperatura. O *Arduino* comunica com o sensor de temperatura e com o módulo GPRS/GPS para obter a temperatura e posição atuais, e envia os valores através do módulo GPRS/GPS para uma base de dados na web. A

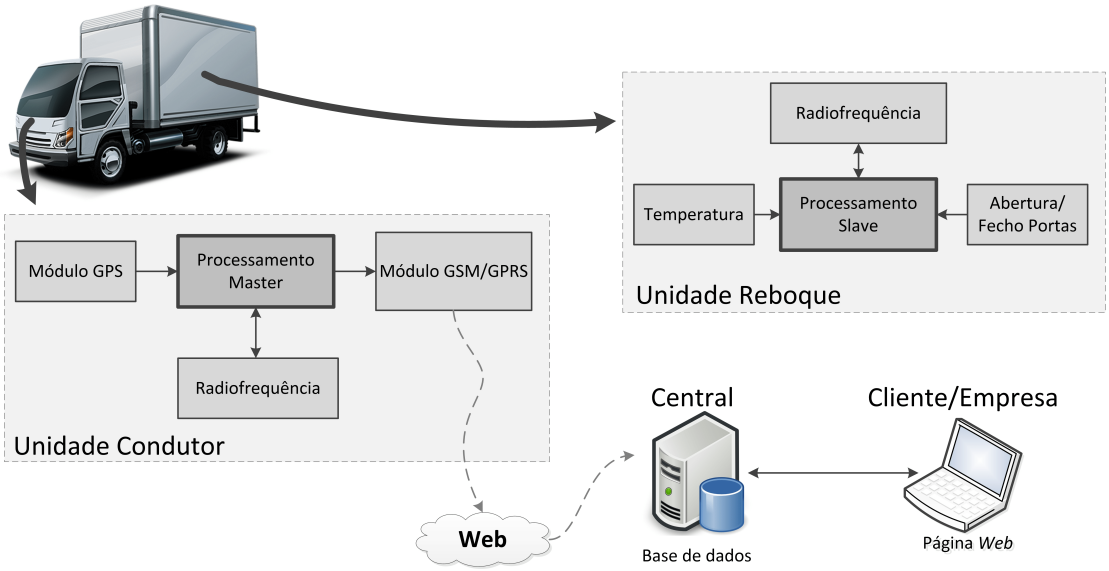

Figura 2.2: Diagrama da solução proposta por Patrick Igreja [7]

figura 2.3 ilustra o sistema desenvolvido.

O serviço web desenvolvido permite monitorizar, adicionar ou configurar veículos e clientes. O administrador adiciona um veículo fornecendo o nome, marca e modelo do veículo e pode arrendar este veículo a um cliente dando limitações como a área onde o cliente pode viajar, a velocidade máxima e a temperatura máxima que o sensor pode medir. Caso algum destes limites seja ultrapassado o administrador é avisado por email e é enviada uma notificação para o *smartphone* do cliente. Para isto foi também desenvolvida uma aplicação para *smartphone* que permite aos clientes ver a posição e temperatura atuais do veículo e consultar os limites impostos pelo administrador.

Neste trabalho foi concluído que se não se enviar os valores medidos caso sejam iguais aos últimos enviados para o servidor é possível reduzir o consumo de energia até 17%, com uma taxa de amostragem de 1 segundo.

2.1.1 Análise Critica

Da analise do trabalho desenvolvido nas outras dissertações identificaram-se tópicos que podem ser explorados com maior detalhe.

O formato da mensagem que o servidor recebe é diferente para todas as plataformas, varia consoante o tipo e número de dados a ser monitorizados e o método usado para enviar a informação. Por exemplo [7] e [8] usam um módulo GPRS para enviar a informação em pedidos HyperText Transfer Protocol (HTTP) enquanto que [6] usa a ligação de dados móvel do *smartphone* para enviar as mensagens diretamente entre duas *sockets* TCP. Com vista a simplificar estas comunicações propõe-se padronizar estas mensagens através de protocolos e formatos usados atualmente na web.

Nenhuma informação partilhada entre os veículos e a plataforma é confidencial, mesmo que algumas destas plataformas forneçam autenticação do utilizador através de *password*. As mensagens enviadas entre os veículos e o servidor central circulam pela

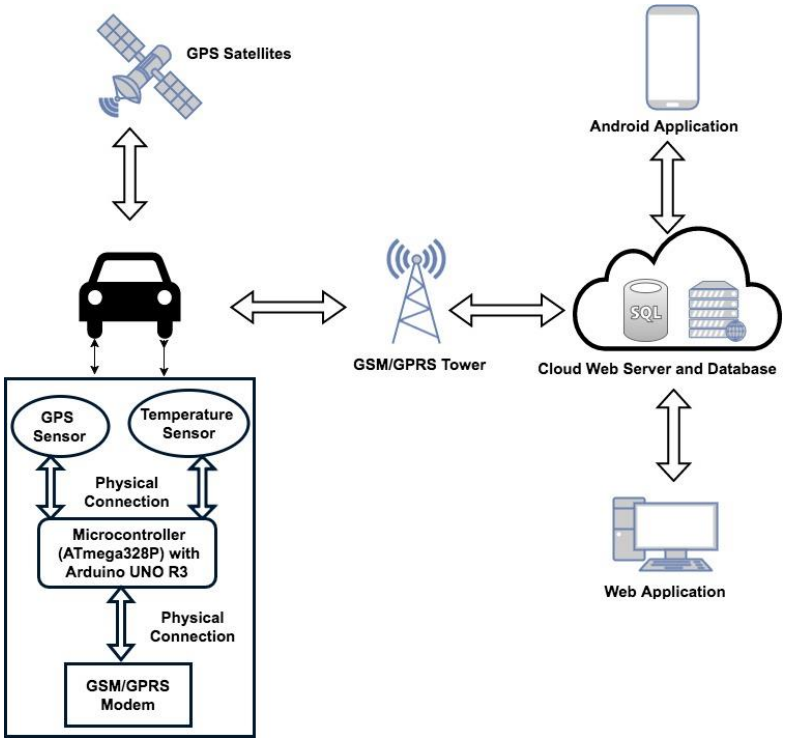

Figura 2.3: Diagrama da solução proposta por Salman Almishari [8]

Internet e, caso não estejam encriptadas, podem ser lidas por terceiros. Para aumentar a privacidade e segurança dos utilizadores todas as mensagens trocadas entre o cliente, o servidor e o veículo deverão ser protegidas por encriptação.

Por vezes é necessário incluir novos parametros de monitorização e, caso necessário, nas plataformas aqui apresentadas apenas seria possivel através da restruturação da base de dados e programação do servidor web de forma a intrepretar corretamante a nova informação. Para facilitar este tipo de operação a plataforma desenvolvida deverá possuir uma base de dados flexível e uma estrutura de comunicação bem definida que acomode este tipo de alterações.

Nenhuma destas plataformas ainda está disponivel para testar, os endereços apresentados não respondem e não foi disponibilizada nenhuma forma para as executar no computador pessoal. O trabalho necessário para realizar monitorização seria reduzido usando uma destas plataformas, dado que apenas seria necessário programar o módulo do veículo de forma a comunicar corretamente com o servidor.

Esta plataforma tem de ser desenvolvida com vista a poder ser usada por outros utilizadores no futuro. Desta forma tem de ser de fácil configuração para diferentes aplicações e tem de ser disponibilizada através de repositórios públicos para não ficar dependente de um só servidor.

Capítulo 3

Solução Proposta

Com base na pesquisa dos trabalhos feitos anteriormente e das tecnologias disponíveis descritas no capítulo 2 foi esboçada a arquitetura da plataforma. A figura 3.1 representa os componentes principais da solução, e a forma como comunicam entre si. Como se pode observar a plataforma a ser desenvolvida é constituída por 3 componentes: os veículos, o servidor e a interface visual. A arquitetura de cada um dos componentes da plataforma é apresentada nos capítulos que se seguem.

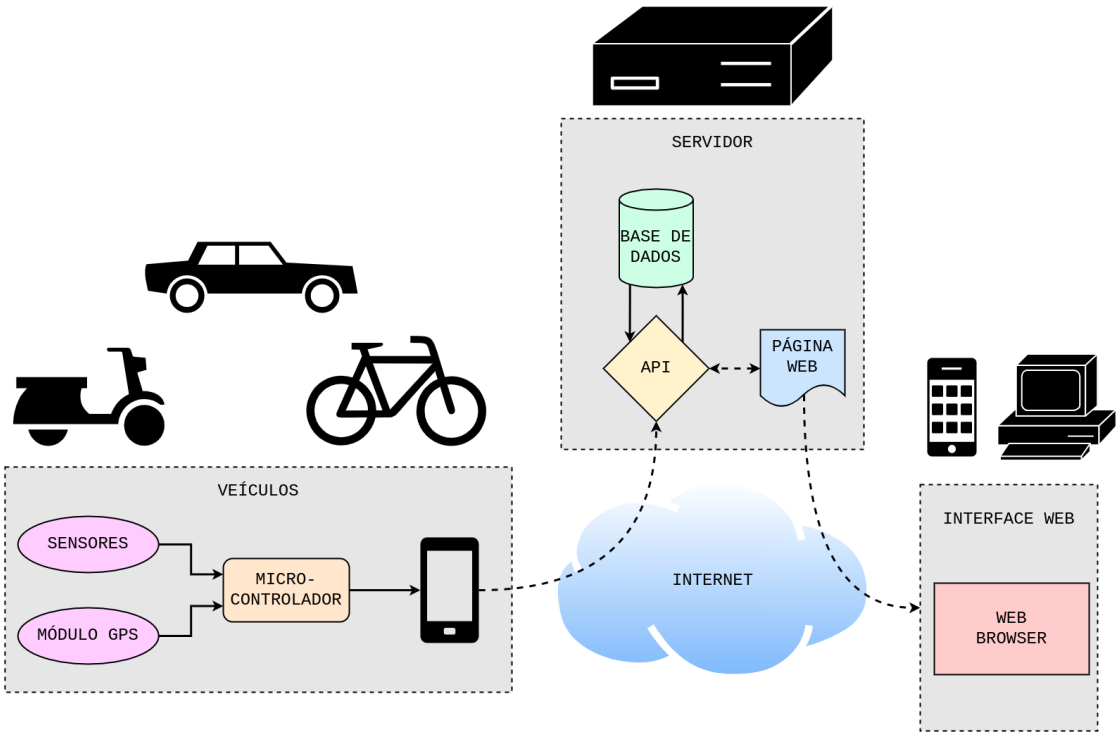

Figura 3.1: Diagrama da solução proposta

3.1 Veículos

Para realizar a monitorização remota de um veículo, em tempo real, serão obtidos dados a partir de uma série de sensores, instalados no veículo, em intervalos de tempo constantes. Para isto será programado um microcontrolador que, periodicamente, executa a leitura dos sensores e envia os dados para o servidor remoto. No esquema geral da arquitetura proposta, figura 3.1, o fluxo de informação de cada veículo é representado no grupo da esquerda, com o título *veículos*.

Conforme as necessidades do utilizador os parâmetros a ser monitorizados serão diferentes. Desta forma, para garantir que a plataforma seja versátil para todas essas necessidades não se pode limitar o tipo nem o número de sensores que podem ser utilizados nos diversos veículos. Como, na maior parte das aplicações, a posição do veículo é um parâmetro de interesse a ser monitorizado, geralmente será montado um módulo de localização GNSS em conjunto com outros sensores no microcontrolador, como é ilustrado na figura 3.1. Os outros sensores serão instalados conforme o objetivo da monitorização, por exemplo:

- Para controlar o desgaste (e prever a necessidade da troca) dos pneus de um camião serão instalados sensores Tire Pressure Monitoring System (TPMS) de pressão e temperatura nos pneus que comunicam as leituras a um recetor de rádio instalado no microcontrolador;
- Para monitorizar o consumo de combustível de um carro de aluguer será montado um leitor CAN bus na porta de diagnóstico do veículo que, através da comunicação com a centralina, obterá o nível de combustível periodicamente;
- Para capturar a distância transversal durante a manobra de ultrapassagem entre um veículo motorizado e uma bicicleta, para trabalhos de investigação como foi desenvolvido em [4], seriam instalados sensores de distância na lateral da bicicleta para medir essa distância.

Estes são apenas um exemplo de três diferentes aplicações, existem inúmeros parâmetros que podem ser monitorizados a partir dos muitos sensores disponíveis que podem ser instalados num veículo.

3.1.1 Biblioteca

Para facilitar a integração de novos sensores na plataforma deverá ser desenvolvida uma biblioteca para os microcontroladores, responsável por todas as comunicações com o servidor da plataforma, cuja interface proposta é apresentada na figura 3.2. Desta forma um utilizador da plataforma poderá programar no microcontrolador os métodos necessários à recolha de de dados dos sensores, relativos à sua aplicação, deixando as comunicações entre o servidor e o microcontrolador a cargo das funções implementadas pela biblioteca.

Esta biblioteca declara uma classe de objetos do tipo `XautoConnection` cujo construtor recebe 7 parâmetros que identificam o servidor da plataforma, o utilizador e o veículo. Nos seguintes parágrafos são apresentados cada um em detalhe.

O parâmetro `host` é uma *String* que contém o Uniform Resource Locator (URL) que identifica o servidor na Internet. Este pode ser o endereço Internet Protocol (IP) público

```
1  XautoConnection( String host, const char* certCA,
2    String username, String password,
3    int vehicleId, int[] sensorIDs, int numSensors );
4  int send( String[] values );
```

Figura 3.2: Declaração do construtor e das funções públicas da classe `XautoConnection` desenvolvida para os microcontroladores

do servidor ou, através da configuração de um registo Domain Name System (DNS), o nome do domínio do servidor. De ambas as formas o administrador do servidor terá de obter um certificado Secure Sockets Layer (SSL) de um Certificate Authority (CA) para proteger as comunicações. O certificado do Certificate Authority (CA) será usado para o segundo parâmetro do construtor, `certCA`, que é um ponteiro para o endereço de memória do primeiro carácter do certificado codificado em base 64.

Os dois parâmetros que seguem servem para autenticar o utilizador na plataforma. Tratam-se de duas *Strings* com a informação do nome de *login* do utilizador e a sua palavra-passe. Para que o utilizador possua uma conta na plataforma este terá de se registar previamente, através de métodos que serão apresentados nos próximos capítulos.

Por último, os 3 parâmetros finais identificam o veículo e os sensores nele instalados. O `vehicleID` é o número que identifica o veículo dentro da plataforma, o `sensorIDs` é um *array* de números que identificam os sensores deste veículo na plataforma e, por último, `numSensors` é o número de sensores instalados no veículo. Tal como as credenciais do utilizador, o veículo também terá de ser configurado na plataforma. O número de identificação do veículo e o número de identificação de cada sensor do *array* `sensorIDs` será atribuído automaticamente pela plataforma no processo de configuração.

3.1.2 Comunicação

O método `send()` usufruirá de a esta informação fornecida ao construtor para comunicar corretamente com o servidor. Cada valor medido pelos sensores é convertido numa *String* e é adicionado a um *array* no mesmo índice que o ID do sensor no *array* `sensorIDs`, a referência a este *array* de *Strings* é passada à função pelo parâmetro `values` onde, com a informação dos dois *arrays*, será construída e enviada a mensagem. O uso de *Strings* para representar as medições permite uma maior flexibilidade no tipo de dados que podem ser enviados, do lado do servidor a *String* recebida poderá ser interpretada como número inteiro, ponto flutuante ou *String*. Caso algum dos sensores não realize a sua medição, como acontece com os sensores GNSS quando não encontram satélites no horizonte, insere-se uma *String* vazia no índice respetivo, desta forma não é enviada leitura desse sensor em particular.

O conteúdo da mensagem serão dois *arrays* formatados em JavaScript Object Notation (JSON), o `sensors` que identifica quais as medições que estão a ser enviadas através do ID atribuído a cada sensor, e o `values` que contém o valor das medições efetuadas na mesma ordem que foi atribuída ao `sensors`. Um exemplo de uma dessas mensagens é apresentado na figura 3.3, nesta são enviados 3 valores para o servidor: o sensor com o ID 1 que no servidor estará registado como latitude o valor de 38,996817; o ID

2 que será a longitude o valor -8,737197; e o ID 3 com o valor da velocidade de 23,7Km/h.

```
{
  "sensors":[1, 2, 3],
  "values":[38.996817, -8.737197, 23.7]
}
```

Figura 3.3: Exemplo do *payload* de uma mensagem entre o veículo e o servidor

3.1.3 Funcionamento

O funcionamento do microcontrolador pode ser esquematizado através do diagrama de sequência da figura 3.4, onde estão ilustradas as interações que o microcontrolador tem com o servidor e os sensores.

Durante a inicialização o microcontrolador fará a autenticação no servidor, através da API, que resultará numa chave única que o microcontrolador deverá armazenar para apresentar ao servidor em todas as comunicações posteriores. Esta função é realizada pelo construtor do objeto `XautoConnection`, e a chave fica guardada automaticamente numa variável interna que será usada para enviar as leituras para o servidor. Para terminar a inicialização deverão ser preparados os sensores para leitura e, se necessário, realizar uma primeira medição como teste. Esta fase depende dos sensores instalados no veículo, alguns ficam prontos assim que são alimentados e outros possuem procedimentos próprios, geralmente implementados através das bibliotecas do sensor, como tal esta fase poderá ser programada pelo utilizador conforme as suas necessidades.

Quando terminada a inicialização o processamento entra no *loop*, que é um procedimento que é executado repetidamente, enquanto o microprocessador estiver ligado. Durante este ciclo o microprocessador recolhe as medições dos sensores, através de métodos próprios relativos a cada sensor, e envia-as para o servidor recorrendo à função `send()` do objeto definido durante a inicialização. Desta forma um utilizador que esteja a programar o microcontrolador para a sua própria aplicação apenas tem de implementar a inicialização leitura dos sensores.

Resumindo, o microcontrolador obtém a localização, faz a leitura dos sensores, constrói a mensagem e envia-a para o servidor mas, para que as mensagens do microcontrolador alcancem o servidor a partir dos veículos, é necessário algum meio de comunicação com o servidor remoto. A forma mais simples será através da Internet, por isso tem que se incluir no veículo algum método de acesso à Internet. Desta forma recorre-se ao telemóvel *smartphone* do condutor para agir como *router* entre uma rede local *Wi-Fi* gerada pela função *hotspot* onde se liga o microcontrolador, e a Internet através da rede de dados móvel. Isto é, a ligação ao servidor remoto será feita através do telemóvel do condutor, de forma a evitar um tarifário Global System for Mobile Communications (GSM) dedicado a cada veículo monitorizado. A figura 3.5 apresenta o diagrama de rede presente nos veículos monitorizados.

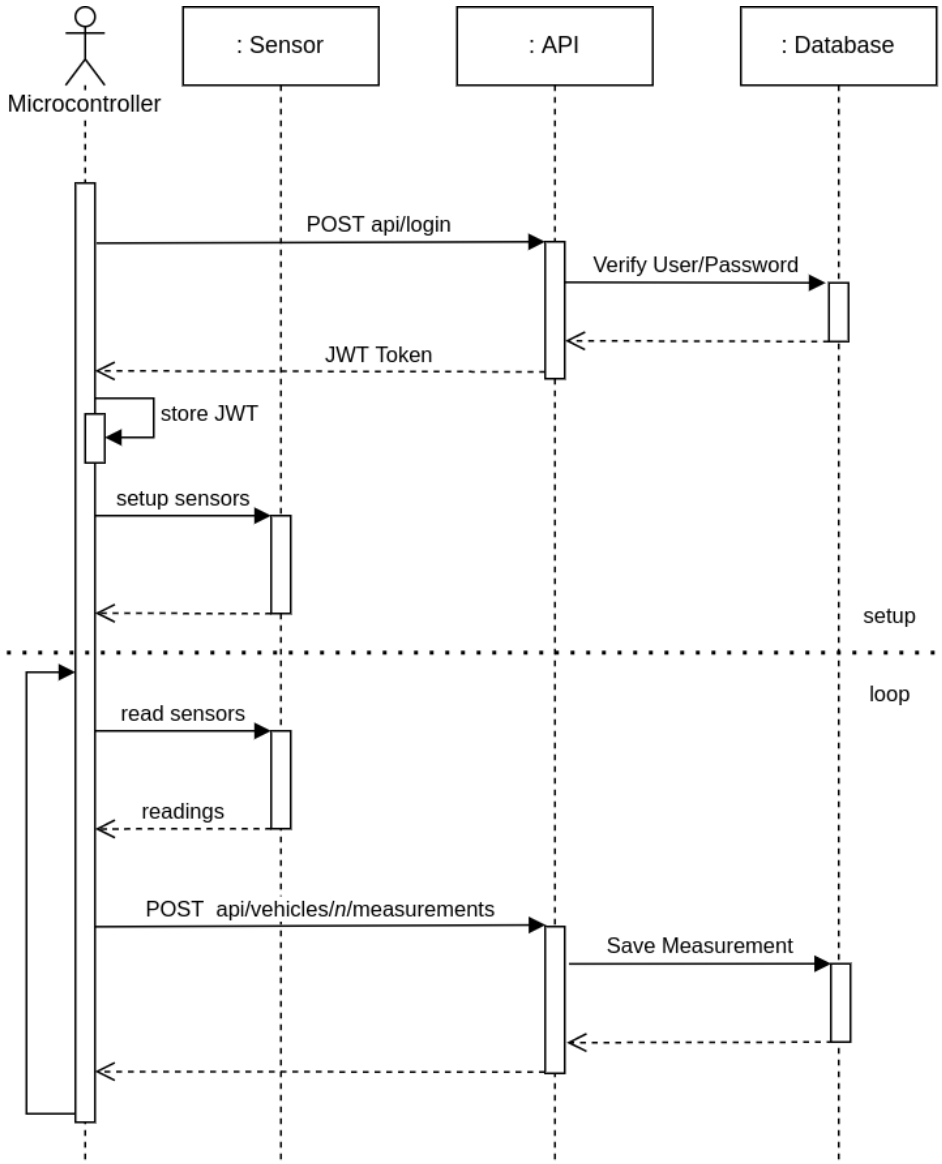

Figura 3.4: Diagrama de sequência do microcontrolador dos veículos (Unified Modeling Language (UML) 2.0 de sequência) [9]

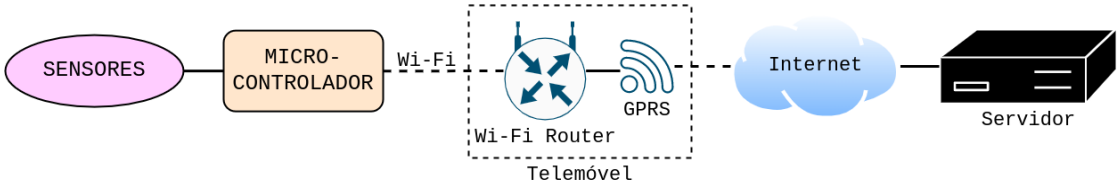

Figura 3.5: Diagrama de rede entre o veículo e o servidor

3.2 Servidor

Cada veículo monitorizado enviará os dados para um servidor central, onde serão recebidos, armazenados e fornecidos aos utilizadores através da interface visual, que os apresenta através de uma página web. Desta forma o servidor é o centro da plataforma, como ilustrada o esquema da figura 3.1, recebendo *input* dos veículos e enviando *output* para o *browser* dos utilizadores.

Dado que a plataforma deverá ser desenvolvida para ser usada por diferentes utilizadores tem de ser possível a sua instalação e configuração em múltiplos computadores. A solução proposta está em incluir o *software* do servidor numa imagem de máquina virtual, como está apresentado no esquema da figura 3.6, que é executada num computador com acesso à Internet sendo, desta forma, possível distribuir e instalar o *software* facilmente por diversos utilizadores. Incluir o *software* da plataforma numa máquina virtual é vantajoso também porque permite ser executado em qualquer tipo de computador, e em diversos sistemas operativos, desde que possua capacidades de virtualização.

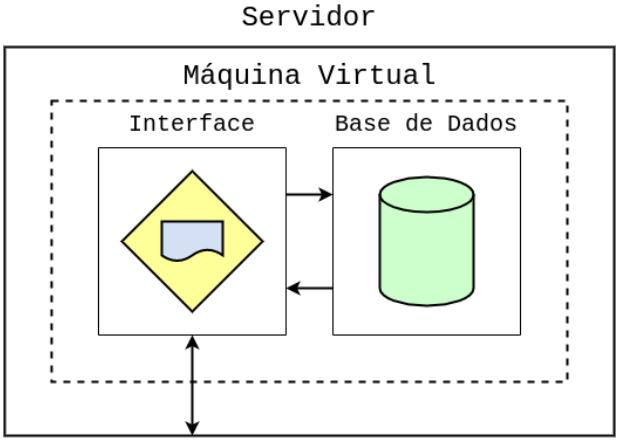

Figura 3.6: Esquema do funcionamento do *software* no servidor

O *software* do servidor é constituído por dois módulos: a interface e a base de dados. Os módulos serão executados de forma independente, ou seja, o servidor deverá executar duas máquinas virtuais que comunicam entre si através de uma rede virtual onde a interface estará exposta ao exterior e a base de dados apenas comunicará com a interface. A interface trata-se de um servidor web cujo objetivo é servir como uma ponte de informação entre os clientes (os veículos e a interface visual) e a base de dados através de uma API, traduzindo pedidos HTTP de determinados recursos em chamadas Query Language (QL) para a base de dados, assim os clientes interagem com a base de dados através da interface, sem fornecer acesso direto à base de dados. A interface deverá também fornecer o conteúdo da página da interface visual (representado a azul na figura 3.6) aos *browsers* dos utilizadores.

3.2.1 Interface

A interface deverá expor uma API, usada pela interface visual e pelos veículos, que fornece formas para manipular utilizadores, veículos, sensores e medições na plataforma.

Esta API deverá dispor os recursos segundo as regras definidas pela arquitetura Representational State Transfer (REST)[?], de forma a ser facilmente interpretada e incorporada por outros programas, a figura 3.7 ilustra a implementação desta arquitetura no desenvolvimento dos *endpoints* expostos pela interface.

A API define 5 recursos diferentes: os utilizadores (`users`), os veículos (`vehicles`), os sensores (`sensors`), as medições (`measurements`) e um recurso especial *login* (`login`). O *payload* das mensagens trocadas com a API estará formatado em JSON, quer no envio dos parâmetros das funções como nas respostas do servidor.

Do lado esquerdo do esquema estão representados os *endpoints* relativos aos utilizadores da plataforma cujo primeiro recurso, representado pelo URL `example.org/api/users`, serve para procurar utilizadores (pelo nome ou por e-mail) através do método `GET`, com os parâmetros da procura passados por URL, ou para criar novos utilizadores através do método `POST`, fornecendo o *username*, a *password* e o e-mail na *payload* da mensagem. O resultado da pesquisa é um valor booleano identificado pelo nome `result` que indica se o nome ou e-mail está usado por algum utilizador. A resposta para o pedido de registo de um novo utilizador é, como está definido pela arquitetura REST para um pedido `POST`, uma mensagem de sucesso com o estado `201 Created` e a localização do recurso do novo utilizador no *header* `Content-Location` ou, caso não seja bem sucedido, a mensagem de erro com o estado `400 Bad Request`.

O *endpoint* que segue identifica um utilizador da plataforma como, por exemplo, o utilizador com o ID 5 será acedido através do URL `example.org/api/users/5`, e este permitirá obter o nome e e-mail deste utilizador através do método `GET`, alterar esses dados e a *password* através do método `PUT` ou apagar todos os dados, incluindo os veículos registados e as suas medições, com um pedido `DELETE`. O servidor responderá a estes pedidos de acordo com o REST porém, como se tratam de procedimentos que acedem e alteram a informação de um determinado utilizador, para que o servidor execute o pedido tem de ser fornecida uma chave de autenticação JSON Web Token (JWT) válida através do *header* `Authorization`. Isto é válido para todos os *endpoints* que acedam a informação relativa a veículos, sensores e medições, o pedido tem de ser autenticado pelo utilizador que possui esses recursos, caso contrário o servidor responderá com o estado `401 Unauthorized` se não for fornecido nenhuma autenticação ou `403 Forbidden` se a chave fornecida não for válida para o recurso acedido.

A chave JWT é obtida através do *endpoint* `login`. É enviado um pedido `POST` para o URL `example.org/api/login`, com o nome do utilizador e a palavra passe no *payload* da mensagem, para o qual o servidor responde com uma nova chave ou um código de erro `400 Bad Request`, conforme a informação enviada esteja correta ou errada.

O último endereço relativo aos utilizadores, por exemplo para o utilizador 5 `example.org/api/users/5/vehicles`, representa os veículos registados em nome de um determinado utilizador, neste o cliente da API pode pedir essa lista com um pedido `GET`, ao qual o servidor responde com um *array* de objetos (no formato JSON) com o ID, o nome e a cor para cada veículo. Um pedido do tipo `POST` para o mesmo endereço serve para registar um novo veículo do utilizador referenciado e, no *payload* da mensagem, é fornecido o nome e a cor que representa o novo veículo na plataforma.

O veículo é um recurso acedido através do *endpoint* `/api/vehicles/`, seguido do ID do veículo. Representa um veículo monitorizado e expõe um nome e uma cor (usada pela interface visual), e pertence sempre a um utilizador. Neste *endpoint* o cliente pode pedir a informação relativa a um veículo através de um pedido `GET`, alterar essa informação

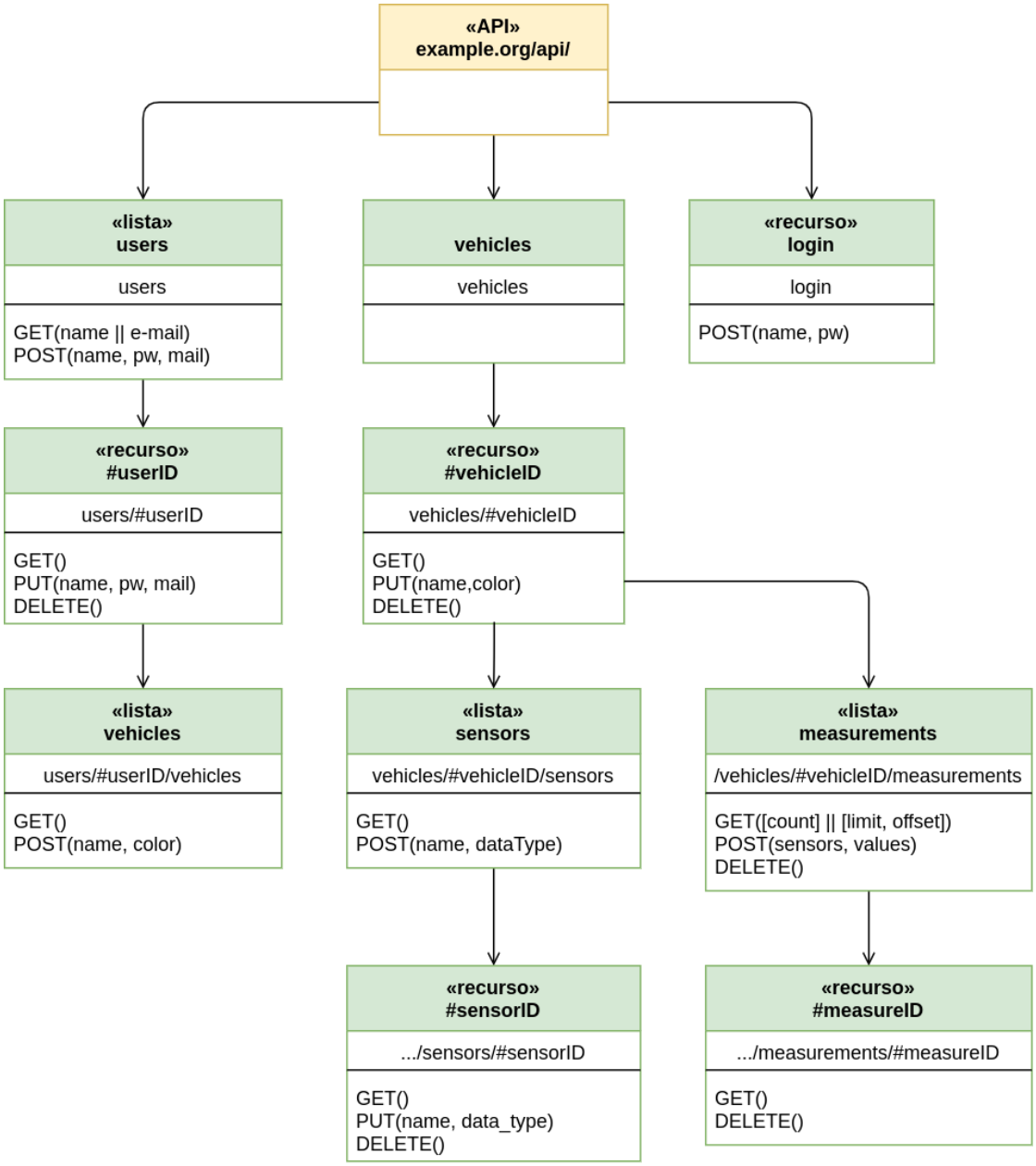

Figura 3.7: Diagrama dos *endpoints* definidos pela interface (UML 2.0 de classes, adaptado) [9]

com o método `POST` ou apagar o veículo da plataforma com um pedido `DELETE`. Cada veículo expõe um recurso relativo aos sensores nele instalados (`sensors`) e outro para medições que forem nele realizadas (`measurements`).

Uma lista de sensores instalados num veículo com o ID `N` podem ser obtida no *endpoint* `/api/vehicles/N/sensors` com o método `GET` no formato de *array* de objetos com o ID, o nome e o tipo de dados para cada sensor registado. No mesmo endereço podem ser introduzidos novos sensores na plataforma com um pedido `POST`, fornecendo o nome e o tipo de dados que o sensor mede (números inteiros, de vírgula flutuante ou strings) no *payload* da mensagem. Se o pedido for bem sucedido o servidor responde com uma referência ao novo recurso criado com o *header* `Content-Location` como, por exemplo, quando for introduzido o 16º sensor na plataforma, para o veículo com ID 5, o recurso construído pelo servidor será representado pelo URL `/api/vehicles/5/sensors/16`.

Para os recursos que representam sensores na plataforma a API permite 3 ações: obter a informação do sensor (nome e tipo de dados) através de `GET`, alterar essas informações com um `PUT`, ou apagar o sensor (e as medições efetuadas por esse sensor) com o método `DELETE`. O recurso que representa as medições (`measurements`) não pertence a um sensor, como evidencia a figura 3.7, mas referencia o ID dos vários sensores que efetuaram uma medição num dado instante de tempo.

As medições de um veículo são representadas através do *endpoint* `/api/vehicles/ID/measurements`, onde ID é o número de identificação de um veículo na plataforma. Para obter o número total de medições guardadas recorre-se a um pedido `GET`, passando como parâmetro (no URL) a palavra `count`; para obter todas as medições do veículo basta executar o pedido sem passar parâmetros, no entanto, se para esse veículo existirem muitas medições armazenadas, a resposta do servidor será lenta e pesada, por isso são disponibilizados dois parâmetros: `limit` para pedir um determinado número de medições e `offset` para escolher o número da primeira medição, estes parâmetros trabalham em conjunto para paginar todas as medições efetuadas através de transferências mais pequenas. Este *endpoint* serve também para registar novas medições através do pedido `POST`, onde os *arrays* descritos no capítulo 3.1.2 seguem no *payload* da mensagem. Aqui também se pode eliminar todo o histórico de medições do veículo através do método `DELETE`.

Cada medição também pode ser acedida através do *endpoint* do canto inferior-direito da figura 3.7, `/api/vehicles/ID/measurements/N` para a `N` medição do veículo `ID`. O método `GET` obtém os *arrays* dos sensores e dos valores da medição e uma *string* do *timestamp* do momento em que foi realizada, para apagar essa medição usa-se o método `DELETE`.

Como as mensagens trocadas entre os clientes e o servidor contém informação sensível como, por exemplo, a palavra-passe do utilizador no pedido de autenticação, todas as mensagens deverão ser trocadas usando o protocolo HTTPS, ou seja, todos os pedidos HTTP descritos nos parágrafos anteriores deverão ser feitos sobre o protocolo Transport Layer Security (TLS). Isto significa que se tem de gerar um certificado SSL para o servidor da plataforma e o assinar através de um CA, para garantir que o *browser* dos utilizadores reconhecem o servidor e que os microcontroladores nos veículos consigam enviar medições para o servidor.

3.2.2 Base de Dados

A base de dados armazena de forma permanente a informação que é enviada para o servidor pela interface (API). Foi desenvolvida com vista a ser flexível de forma a permitir adicionar e remover utilizadores, veículos, sensores e medições sem afetar o funcionamento da plataforma. Desta forma a base de dados é constituída por 5 tabelas: a dos utilizadores (`users`), a dos veículos (`vehicles`), a dos sensores (`sensors`), a das amostras (`samples`) e a das medições (`measurements`). Estas tabelas, e as relações entre elas, estão representadas no diagrama da figura 3.8.

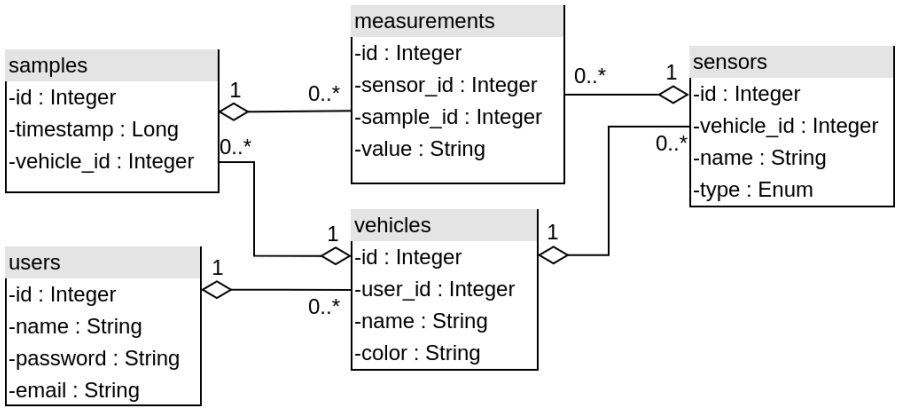

Figura 3.8: Diagrama da estrutura da base de dados
(UML 2.0 de classes) [9]

A tabela `users` armazena o nome dos utilizadores, o seu e-mail e a sua *password*, e atribui um número de identificação (ID) único a cada entrada da tabela. A informação desta tabela é usada pela API para realizar a autenticação do utilizador, o nome de utilizador recebido é usado para encontrar a entrada correta da tabela e são comparadas as *passwords* para determinar se é realmente o utilizador.

A tabela `vehicles` contém a informação acerca do nome do veículo e uma cor usada pela interface visual para o representar. A coluna `user_id` serve para identificar o utilizador que o registou na plataforma, esta informação é usada para determinar se um pedido relativo a um veículo é realizado pelo utilizador autorizado. Diferentes entradas nesta tabela podem conter o mesmo `user_id`, tornando-se numa relação de 1 utilizador para N veículos, dando flexibilidade ao utilizador para criar ou apagar veículos conforme for necessário.

A tabela `sensors` guarda o nome e o tipo de dados de cada sensor instalado nos veículos da plataforma. Cada sensor representa apenas uma grandeza física monitorizada no veículo, por exemplo para monitorizar a localização do veículo são necessárias duas entradas nesta tabela, uma para a latitude e outra para a longitude. A coluna `vehicle_id` mantém a referência ao ID do veículo onde o sensor se encontra instalado, desta forma é possível introduzir ou remover sensores em cada veículo. A coluna `type` serve para identificar como devem ser interpretados os dados monitorizados por cada sensor, como números inteiros, números de vírgula flutuante ou *strings*.

A tabela `samples` regista o momento em que os veículos efetuaram medições. Esta tabela agrupa as medições que foram realizadas por um veículo num dado instante numa amostra, agrupando os valores medidos pelos diferentes sensores do veículo. A coluna

`vehicle_id` identifica o veículo a que a amostra pertence.

A tabela `measurements` armazena os valores de todas as medições dos diversos sensores. Cada medição contém uma referência a uma amostra na coluna `sample_id` que armazena o momento em que foi efetuada, e a um sensor na coluna `sensor_id` que identifica a qual sensor é que pertence o valor guardado. Esta estrutura é muito flexível, permite acrescentar ou remover utilizadores, veículos ou sensores sem influenciar os valores já registados, e é capaz de realizar amostragens de veículos onde alguns sensores não enviam informação, como acontece quando realizam medições invalidas ou existem falhas de sinal.

3.3 Interface visual

A interface visual é uma aplicação na web que serve para o utilizador interagir com o servidor, para além de apresentar os dados da monitorização, fornece os meios para configurar novos veículos e sensores e fornece as informações necessárias para programar nos microcontroladores, através da biblioteca referida no capitulo 3.1.1, os IDs dos recursos do utilizador. Aqui também serão registados os utilizadores e as suas *passwords*, que serão usados tanto na interface visual como nos microcontroladores para autenticar os veículos no servidor.

3.3.1 Painéis

A interface visual é uma página web construída como uma Single-Page Application (SPA) que disponibiliza diferentes paineis para o utilizador consultar os valores da monitorização ou configurar os parâmetros de monitorização. A disposição dos elementos na página é apresentada na figura 3.9, no topo é apresentado o logótipo e o nome atribuídos pelo administrador de sistema, de seguida é apresentado o menu de navegação e, ao fundo, é apresentado o painel selecionado.

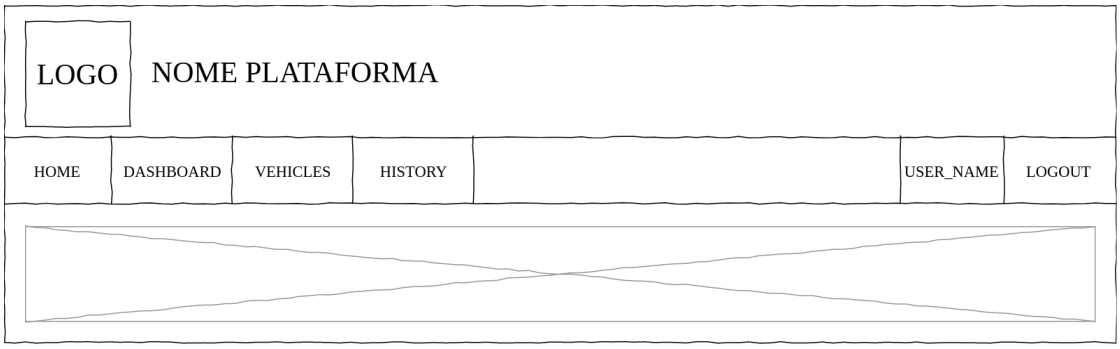

Figura 3.9: Esboço da estrutura da interface visual

Do lado direito do menu de navegação estão as funções relativas ao utilizador, antes de se autenticar são apresentados os botões *login* e *sign-up* para, respetivamente, realizar a autenticação ou realizar o registo de um novo utilizador. Quando o utilizador está autenticado apresenta, conforme a figura 3.9, um botão com o nome do utilizador e um botão de *logout* para configurar as informações do utilizador e para sair da conta, respetivamente.

Os botões do lado esquerdo selecionam o painel apresentado. Quando a página é carregada pela primeira vez o painel selecionado é o *home*, o painel relativo ao primeiro botão do menu, que contém apenas instruções para configurar os veículos pela interface e programar os microcontroladores para enviarem a informação para o servidor.

O segundo botão seleciona o painel *dashboard* que apresenta as medições mais recentes de todos os veículos do utilizador e um mapa com as posições dos veículos, para aqueles que possuam sensores com o nome de latitude e longitude. A figura 3.10 esboça a disposição dos elementos do painel da *dashboard*, à esquerda é apresentado um mapa interativo onde é desenhado um circulo da cor representativa de cada veículo na sua última posição conhecida e, à direita, mostra-se a lista de veículos do utilizador com cada sensor, e o seu valor mais recente, exposto numa caixa da cor do veículo.

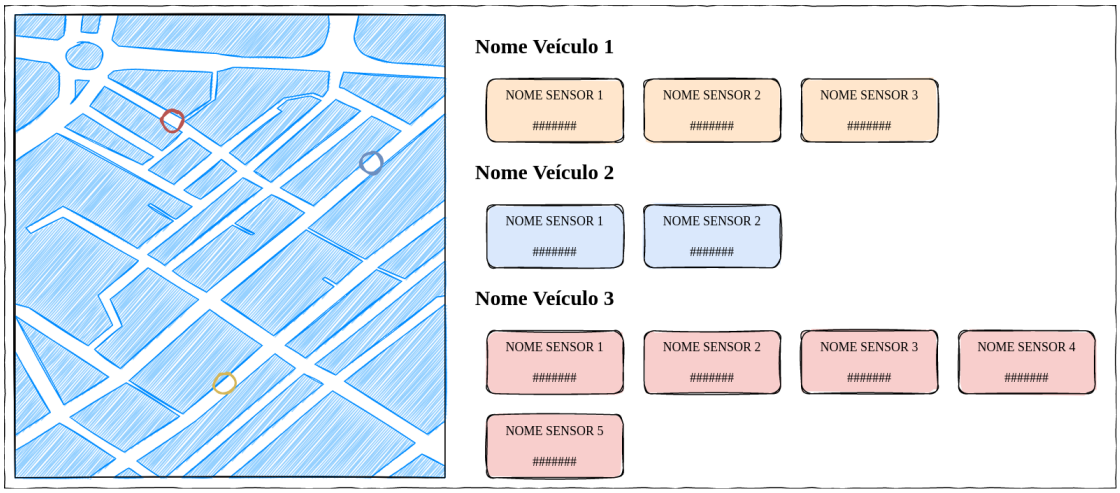

Figura 3.10: Esboço do painel *dashboard*

O próximo botão, *vehicles*, serve para ativar o painel de configuração dos veículos do utilizador, fornece ferramentas para adicionar, editar e apagar veículos e os sensores neles instalados. Este painel, ilustrado na figura 3.11, está separado em três colunas, da esquerda para a direita são: a lista de veículos do utilizador, os controlos para configuração do veículo e o editor de sensores. A lista de veículos é usada para selecionar o veículo a editar ou adicionar um novo veículo. Os controlos de configuração servem para alterar o nome do veículo selecionado, através da caixa de texto, mudar a cor que o representa na plataforma, apagá-lo ou registar um novo sensor nele instalado. O editor de sensores apresenta a lista de sensores que estão instalados no veículo selecionado e, para cada um, fornece meios para alterar o nome e o tipo de dados, ou remover o sensor.

Para além dos controlos este painel apresenta os IDs que a plataforma atribui a cada veículo e a cada sensor e, com esta informação, o utilizador poderá programar no microcontrolador do veículo as referências corretas de cada elemento. Só é possível realizar monitorização de veículos que sejam previamente registados na plataforma visto que toda a informação que é enviada para o servidor sem estar autenticada por um utilizador, ou que seja enviada para um *endpoint* errado, é ignorada.

O último painel é o histórico de monitorização, ilustrado na figura 3.12. Aqui o utilizador tem acesso a todos as medições efetuadas para cada veículo registado, sob a forma de tabela. À esquerda, à semelhança do painel da configuração de veículos, é

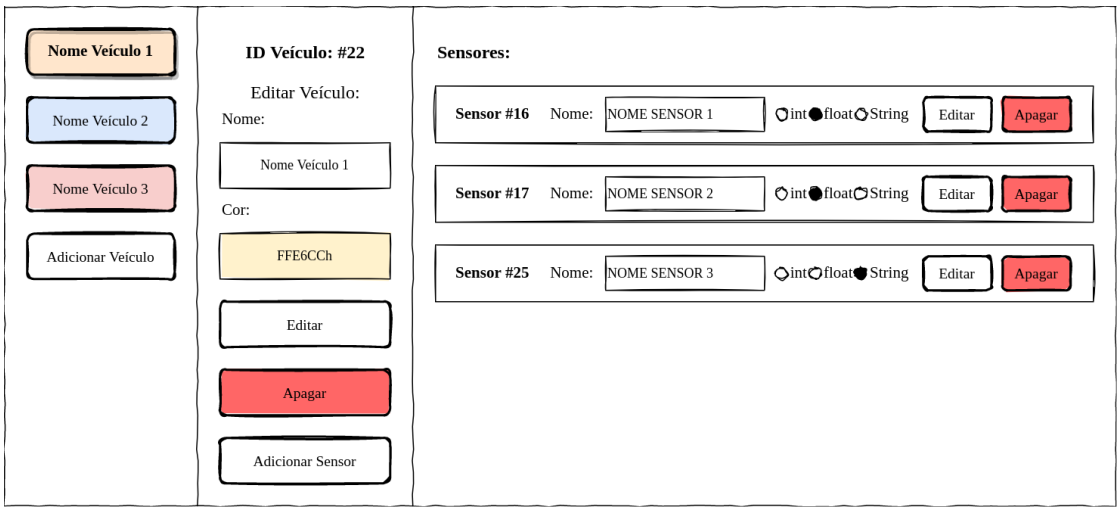

Figura 3.11: Esboço do painel de configuração de veículos

apresentada a lista de veículos do utilizador onde o utilizador seleciona o veículo que quer consultar. À direita é apresentado histórico, onde o utilizador pode descarregar todas as medições para o seu computador ou apagar as medições gravadas do veículo selecionado. As medições podem ser consultadas através da tabela onde as medições estão distribuídas em várias páginas, para que a tabela não seja comprida de mais para o ecrã do utilizador.

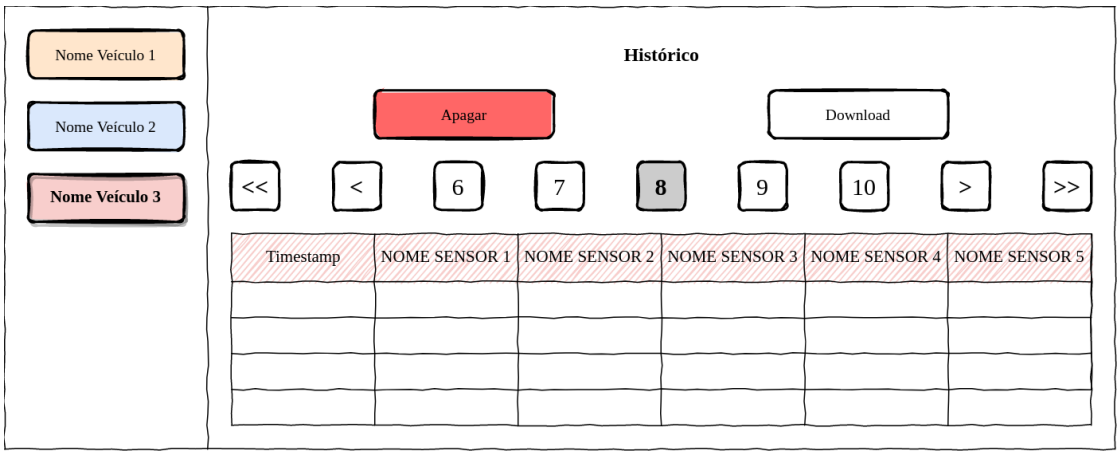

Figura 3.12: Esboço do painel de visualização do histórico de monitorização

3.3.2 Funcionamento

O funcionamento da interface visual está esquematizado no diagrama de sequência da figura 3.13 que representa a sequência de interações necessárias para o utilizador visualizar a *dashboard* com a informação mais recente dos seus veículos, entre o *browser* do utilizador, a API e, indiretamente, com a base de dados.

O primeiro passo do utilizador é realizar a autenticação no site através do preenchimento de um formulário com o nome de utilizador e palavra-passe, a informação introduzida será usada para realizar um pedido HTTP com o método `POST` para *endpoint* `login`

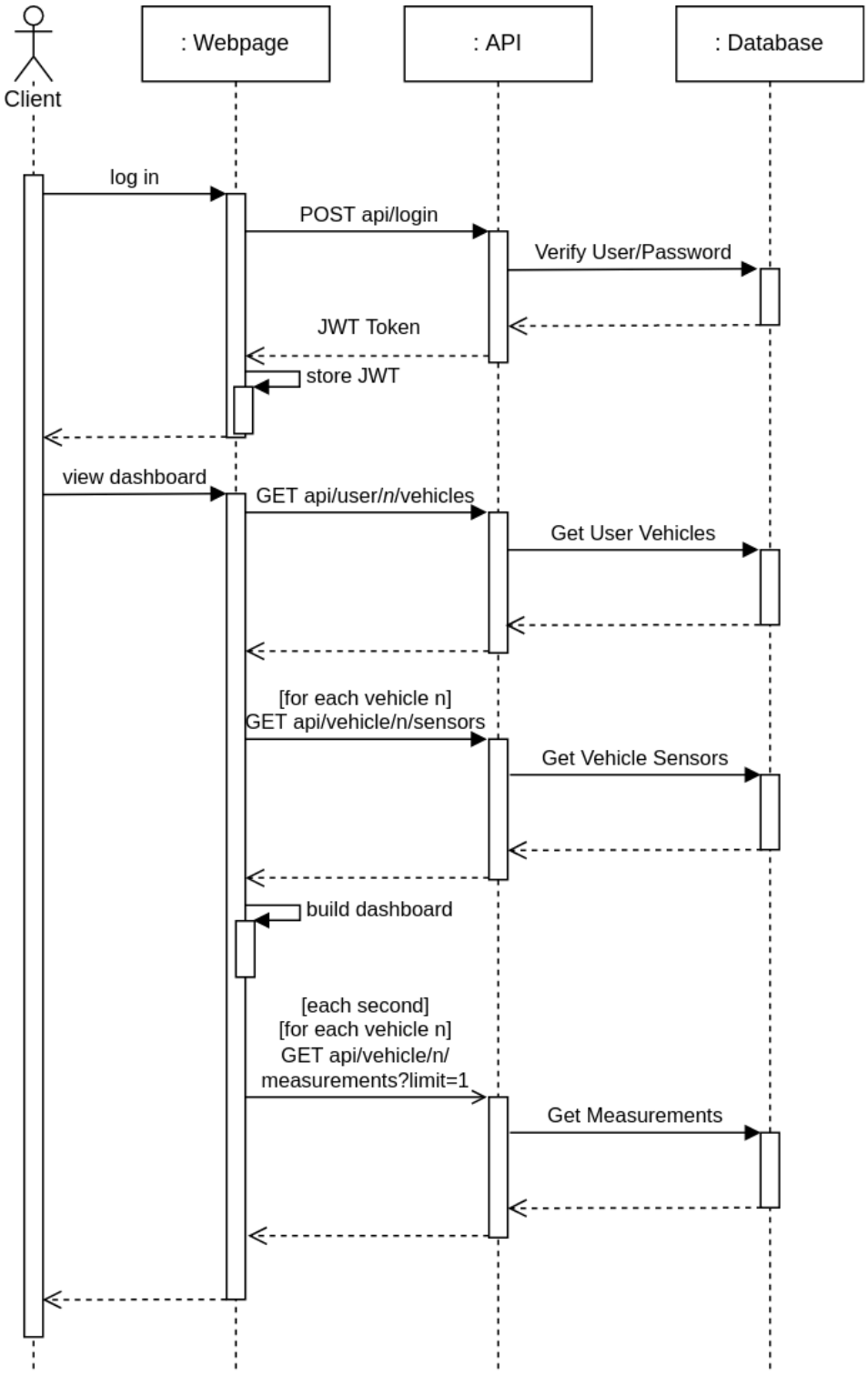

Figura 3.13: Diagrama de sequência da interface visual (UML 2.0 de sequência) [9]

da API. Se os dados de autenticação estiverem corretos a resposta do servidor contém uma chave JWT que será guardada na memória do *browser* para ser enviada para o servidor em todos os pedidos subsequentes no *header* `Authentication`.

Concluída a autenticação o utilizador terá acesso ao menu de navegação onde poderá selecionar a página a visualizar, de entre essas a *dashboard* é usada para monitorizar, em tempo real, os últimos valores enviados pelos veículos. Como mostra a figura 3.13, assim que o utilizador seleciona a página *dashboard* o *browser* executa um pedido ao servidor para obter o ID, nome e a cor de todos os veículos do utilizador. Conhecendo o ID dos veículos, para cada um, é pedido ao servidor o nome dos sensores instalados e, com esta informação, o *browser* constrói o painel com uma série de caixas, uma para cada sensor, com a cor do veículo a que pertence e o nome do sensor, como ilustrado na figura 3.10.

Com a página construída o *browser* inicia um ciclo repetido a cada segundo onde, para cada veículo, pede ao servidor a última medição registada através de Asynchronous JavaScript And XML (AJAX) e atualiza o valor na caixa de cada sensor com o valor recebido do servidor, sem que seja necessário atualizar a página. Caso durante a última amostra enviada pelo veículo algum dos sensores não tenha efetuado a medição será escrito no painel um til (~), que representa um valor inexistente. O ciclo é executado até o utilizador escolher outra pagina, fazer *logout*, fechar o *browser* ou até o JWT expirar.

O painel do histórico de monitorização trabalha de forma semelhante à apresentada na figura 3.13, o *browser* executa um pedido para construir a lista de veículos do utilizador e, quando o utilizador seleciona um veículo, faz outro pedido para obter o ID e o nome dos sensores nele instalados. O *browser* pede depois o número total de amostras gravadas e, com este número, constrói os controlos da paginação, ilustrado como os botões numerados a cima da tabela na figura 3.12, e a tabela. Os valores apresentados na tabela são obtidos através do mesmo endereço usado pela *dashboard*, mas são passados parâmetros para pedir apenas a informação para apresentar na tabela dada a página selecionada pelo utilizador. O pedido enviado para o servidor passa dois parâmetros, o `limit` para controlar o número de amostras que será igual ao número de linhas da tabela e o `offset` para selecionar a primeira amostra do conjunto, que é calculado conforme a expressão

$$offset = n \cdot (p - 1)$$

onde n é o número de linhas na tabela e p é o número da página selecionada.

No exemplo da figura 3.12, para uma tabela com 4 linhas e supondo que o veículo selecionado tem ID 32, quando o utilizador selecionou a página 8 o *browser* gera um pedido `GET` com o URL `/api/vehicles/32/measurements?limit=4&offset=27`. Se o utilizador quiser descarregar todas as medições para o seu computador, ao pressionar o botão de *download*, o *browser* faz o mesmo pedido sem passar nenhum parâmetro e, em segundo plano, constrói um ficheiro Comma-Separated Values (CSV) com os valores recebidos.

O painel da configuração de veículos, ilustrado na figura 3.11, executa pedidos ao servidor em AJAX conforme as ações do utilizador. Quando pressiona o botão de adicionar um novo veículo o *browser* executa um pedido `PUSH` para o endereço `/api/users/uID/vehicles` registando, efetivamente, um novo veículo na plataforma. Quando o utilizador seleciona um veículo da lista da esquerda é preenchido automaticamente o formulário de configuração do veículo e é atualizada a lista de sensores do lado direito da página. O utilizador, depois de editar o formulário de configuração com a informação correta, pressiona o botão editar e o browser executa um pedido `PUT` para

o recurso que identifica o veículo na plataforma (`/api/vehicles/vID`). Com um veículo selecionado o utilizador pode também lhe adicionar sensores, o *browser* envia um `PUSH` para `/api/vehicles/vID/sensors` e o servidor regista um novo sensor na plataforma, esse sensor aparece na lista de sensores da direita, onde o utilizador o poderá configurar e, ao pressionar o botão editar, o *browser* envia um pedido `PUT` com a nova informação para o endereço `/api/vehicles/vID/sensors/sID`.

Capítulo 4

Implementação

A plataforma foi implementada com base na arquitetura proposta no capitulo 3. Para cada um dos elementos da plataforma apresentam-se as escolhas do *hardware* e *software* usados.

A figura 4.1 representa de uma forma geral as soluções usadas na implementação da plataforma.

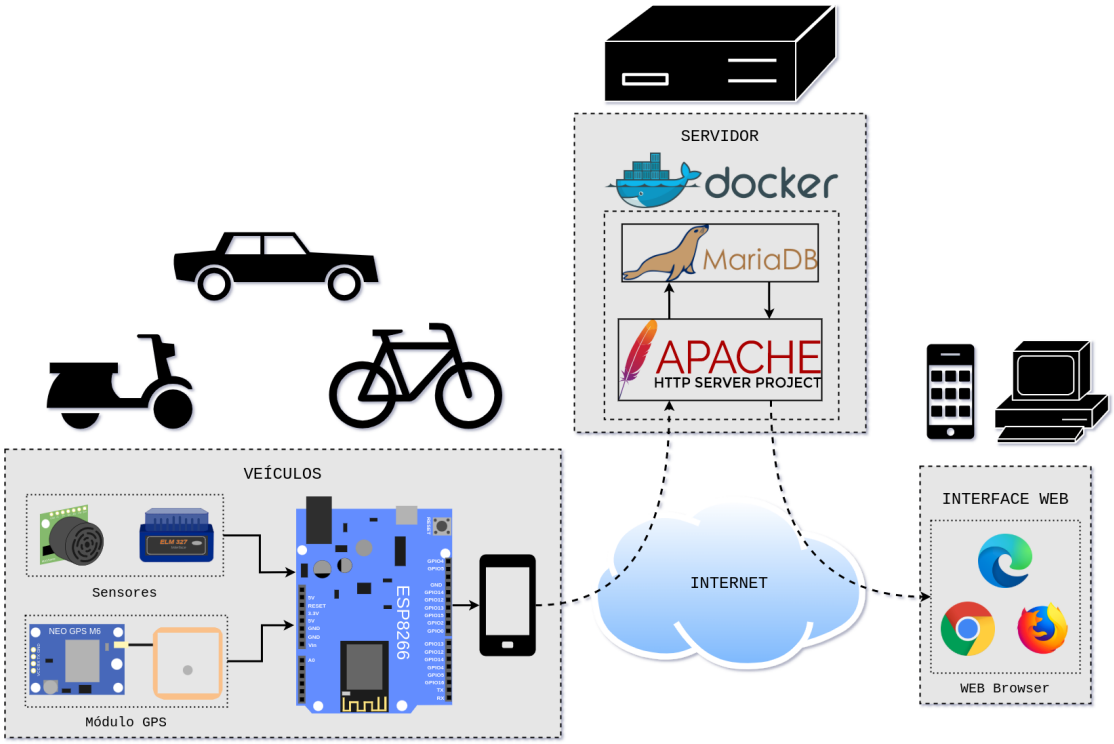

Figura 4.1: Diagrama geral da plataforma representando as escolhas de *software* e *hardware* usadas na sua implementação

O código desenvolvido para esta aplicação está disponível no repositório `https://github.com/RAbrantes706/xauto`, onde se encontra disponível o *software* do servidor, a biblioteca para os microcontroladores e o código dos microcontroladores para diferentes exemplos.

4.1 Veículos

Para monitorizar o veículo foi escolhido *hardware* disponível para compra na Internet. Isto torna a partilha desta plataforma mais fácil dado que o *hardware* está disponível para compra, e a programação dos microcontroladores é auxiliada com a biblioteca disponível no repositório, que implementa as comunicações entre o veículo e o servidor.

4.1.1 Microcontrolador

Os microcontroladores usados foram o ESP8266 e o ESP32, são usados em módulos como o Wemos D1 R2, Wemos D1 mini e o DOIT ESP32 DEVKIT V1, ilustrados na figura 4.2. O módulo usado em cada veículo depende dos parâmetros a monitorizar visto que algumas placas possuem um número maior de pinos para General-Purpose Input/Output (GPIO), ou permitem comunicar por *bluetooth* com os sensores.

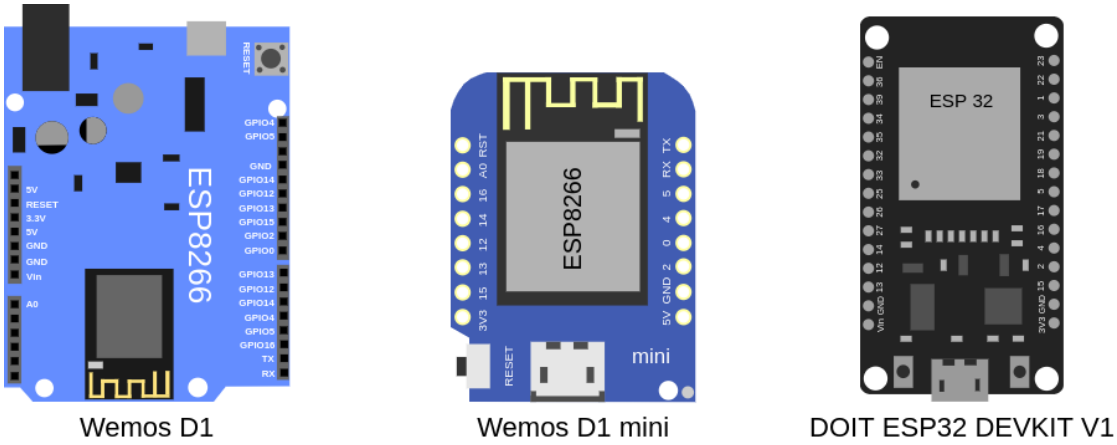

Figura 4.2: Ilustração dos módulos ESP usados na plataforma

Escolheu-se esta família de microcontroladores porque têm a capacidade de comunicar por *Wi-Fi* sem a necessidade de uma placa extra ou um *shield*, como é o caso com a família de microcontroladores ATmega328 usado nas placas *Arduino*. Usando esta funcionalidade podem comunicar as leituras para o servidor, recorrendo a uma rede *Wi-Fi* local gerada pela funcionalidade *hotspot* do *smartphone* do condutor, que encaminha os pacotes para a Internet através da rede móvel, como está descrito no capítulo 3.1.3.

As placas da *Wemos* são ambas construidas com o microcontrolador ESP8266-12F, o que significa que possuem as mesmas capacidades e número de portas GPIO. Entre elas existem duas diferenças, uma é a forma da placa, sendo a versão normal construida com o mesmo formato que a placa Arduino Uno, consideravelmente maior que a versão *mini*; outra diferença é a alimentação, a versão normal possui um regulador de tensão que permite uma alimentação entre 9V e 24V através do *jack* de alimentação para além da alimentação a 5V através da porta USB mini que ambos possuem. [11; 12] Isto torna a placa Wemos D1 R2 mais apropriada para veículos que não possuam tomadas de energia elétrica como o caso das bicicletas ou outros veículos de mobilidade suave, porque podem ser alimentadas com pilhas de 9V, por exemplo. As placas Wemos D1 mini são alimentadas através da porta USB mini, que pode ir buscar energia elétrica ás tomadas elétricas USB presentes em alguns automóveis ou, caso seja necessário, a um *powerbank*.

A placa *DOIT* possuí o microcontrolador ESP32, tem mais funcionalidades que os ESP8266, das quais a mais relevante é a capacidade de comunicar por *bluetooth*. [13] Esta funcionalidade é necessária para obter informações da centralina dos carros através do módulo *ELM327* usado, dado que a comunicação com este módulo é através de *bluetooth*.

Este microcontrolador possui dois núcleos de processamento, por isso é possível realizar tarefas como medir distâncias com sensores ultrassónicos ou processar a mensagem do sensor GPS sem interromper o fluxo do programa para ficar à escuta da resposta dos sensores. Assim é possível realizar uma frequência de amostragem maior caso a aplicação assim o necessite.

Para garantir que a informação circula confidencialmente pela Internet, desde as credenciais do utilizador aos valores dos sensores do veículo, é necessário usar o protocolo HTTPS para todas as comunicações entre o microcontrolador e o servidor. Este protocolo requer que o microcontrolador verifique se o certificado SSL enviado pelo servidor está correto [14], no ESP8266 basta comparar o *fingerprint* Secure Hash Algorithm (SHA) 1 do certificado que o servidor enviou com o *fingerprint* conhecido que está gravado na memória do microprocessador [15], no ESP32 este método foi descontinuado porque a função de *hashing* SHA-1 foi considerada insegura para este propósito [16] e, então, é usada uma cópia integral do certificado para comparar se é igual ao que é recebido do servidor.

A autenticação do utilizador pelos microcontroladores em conjunto com as comunicações encriptadas com o servidor são necessárias para que a plataforma esteja de acordo com a norma ISO20078. [17]

4.1.2 Sensores

Os sensores são os equipamentos que realizam as medições no veículo, são placas disponíveis para compra na Internet. No desenvolvimento desta dissertação foram usados sensores para obter a posição geográfica, distância a obstáculos próximos e valores de sensores da centralina do veículo.

Posição

Para obter a posição do veículo usou-se o módulo GPS NEO GPS 6M. Com este sensor GPS é possível obter a posição geográfica, velocidade, altitude e direção do veículo, para além de outras informações como o número de satélites ligados ao módulo e a que momento foi feita a medição. Pode ser realizada uma montagem simples com este módulo e um microcontrolador Wemos D1 mini para monitorizar estas informações em qualquer veículo, alimentado com um *powerbank*. Uma ilustração do esquema de montagem é apresentado na figura 4.3.

Para comunicar com o módulo usam-se as portas de comunicação Universal Asynchronous Receiver/Transmitter (UART) (as portas série) dos microcontroladores configuradas com um *baudrate* de 9600bit·s^{-1} que, segundo a biblioteca usada para comunicar e processar as mensagens do módulo [18], é a forma mais eficiente. Segundo o fabricante do circuito integrado recetor de GPS [19] este tem a capacidade de comunicar através de UART, Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I^2C) e Universal Serial Bus (USB), no entanto o módulo usado apenas disponibiliza a interface UART para comunicar com o microcontrolador. O módulo tem incluído uma fonte de alimentação

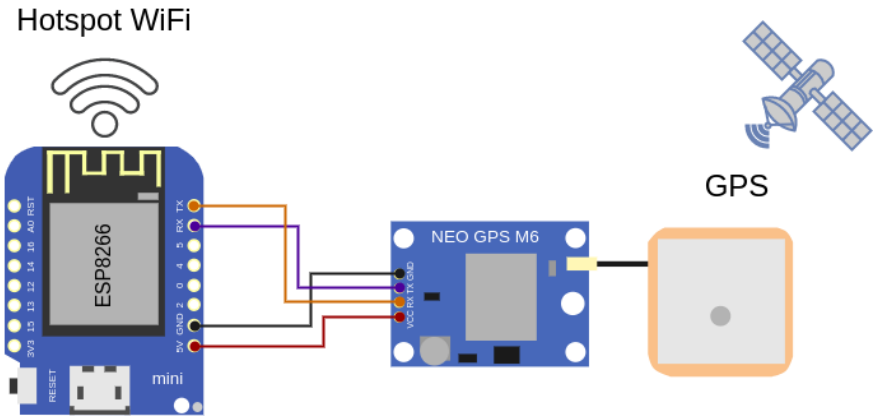

Figura 4.3: Esquema da montagem de um sensor de localização

para que este possa ser alimentado com 5V e uma Electrically Erasable Programmable Read-Only Memory (EEPROM) que vem programada de fábrica com as configurações do circuito integrado, como o *baudrate* da UART e a escolha do protocolo de mensagens, neste caso National Marine Electronics Association (NMEA) em vez de UBX (que se trata de um protocolo proprietário da *u-blox*).

A programação do microcontrolador realizou-se através do Integrated Development Environment (IDE) do *Arduino*, e para facilitar a comunicação e processamento das mensagens do módulo GPS instalou-se a biblioteca *NeoGPS* [18] que foi configurada para comunicar com o módulo pelas portas UART e para processar apenas as linhas *GPGGA* e *GPRMC* das mensagens NMEA através dos ficheiros *GPSport.h* e *NMEAGPS_cfg.h* respetivamente. Na plataforma todos os microcontroladores que usem este módulo são programados com uma estrutura de código como a apresentada na figura 4.4, onde o código foi simplificado, mantendo apenas o que é relevante ao funcionamento do módulo GPS. Nas diretivas de preprocessamento são importados os ficheiros *NMEAGPS.h* que contém as declarações das estruturas e funções para a leitura das mensagens do módulo, e o ficheiro *GPSport.h* onde estão as configurações da comunicação com o módulo. São declaradas duas variáveis: `gps` que serve de interface com a `gpsPort` que é a *stream* de informação entre o microcontrolador e o módulo GPS, e `fix` que é uma estrutura de dados que guarda de forma organizada a informação lida da *stream*. Durante o *setup* inicia-se a *stream* com um *baudrate* de 9600bit·s^{-1} e, quando terminado, o microcontrolador executa o procedimento `loop` repetidamente, onde verifica se existe informação a chegar na *stream* `gpsPort` e, caso exista, processa a mensagem recebida através da função `gps.read()` e guarda a informação na estrutura `fix`. Se essa informação for uma localização válida então é enviada para o servidor.

A montagem da figura 4.3 pode ser construída rapidamente e de forma compacta como mostra a figura 4.5. Desta forma é possível monitorizar a posição de qualquer veículo, sendo apenas necessário manter a antena do módulo GPS num local com acesso ao sinal dos satélites.

Distância

São usados sensores de distância para verificar se existe, e a que distância se encontra, algum obstáculo próximo do veículo. São montados no veículo e orientados para onde

```
#include <NMEAGPS.h>
#include <GPSport.h>
static NMEAGPS gps;
static gps_fix fix;
void setup()
{
  gpsPort.begin(9600);
}
void loop()
{
  while(gps.available(gpsPort))
  {
    fix = gps.read();
    // Executar outras medições aqui
    if(fix.valid.location)
    {
      send(); // Enviar dados para o servidor
    }
  }
}
```

Figura 4.4: Estrutura do código *Arduino* usado para a leitura do módulo GPS

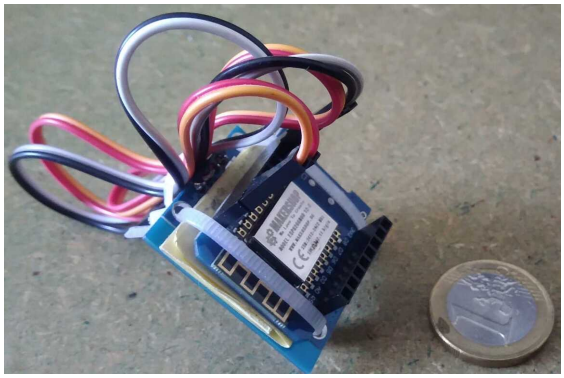

Figura 4.5: Foto da montagem do NEO GPS M6

se quer realizar as medições: à retaguarda, medindo a distância ao veículo que o segue, à esquerda, medindo a distância a um veículo que o ultrapasse, à dianteira, medindo a distância ao veículo que o precede, à direita, medindo a distancia ao veículo que é ultrapassado.

Os sensores de distância usados estão ilustrados na figura 4.6. Apenas os dois primeiros sensores apresentados são capazes de realizar medições em veículos dado que o terceiro está sujeito a interferências quando exposto à luz solar.

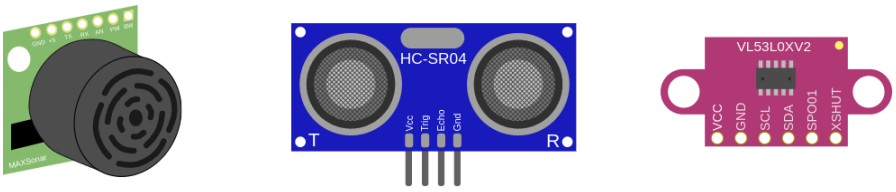

Figura 4.6: Ilustração dos sensores de distância usados

O principio de funcionamento destes sensores é igual para todos, emitem um sinal e aguardam pela reflexão do mesmo, a distância será metade do produto entre a velocidade de propagação do sinal e o tempo que o sinal demorou a alcançar o obstáculo e a voltar. Nestes sensores o sinal pode ser sonoro ou luminoso, nos dois primeiros é usado um sinal sonoro ultrassónico e no último é um sinal luminoso infravermelho.

O primeiro sensor da imagem é o MB1010 LV-MaxSonar-EZ1 com capacidade de detetar objetos até 6 metros de distância [20], é usado para detetar a distancia a que circulam outros veículos a diante ou à traseira, visto que é o sensor com maior alcance usado. É possível realizar as leituras deste sensor por 3 diferentes meios: por porta serie através da interface UART do microcontrolador, pelo valor da tensão da saída analógica do sensor ou através da leitura da duração de um impulso; no entanto nos veículos não foi usada a porta série porque não apresenta vantagens em relação aos outros dois métodos e o porto UART dos microcontroladores já é usado para comunicar com o módulo GPS. A figura 4.7 ilustra as diferentes montagens deste sensor com o microcontrolador Wemos D1 mini conforme o método de leitura usado.

Nestas montagens usou-se a porta GPIO 4 do microcontrolador para controlar os instantes em que o sensor realiza medições. Para dar a instrução para o sensor realizar uma medição eleva-se a tensão do pino *RX* de 0V a *Vcc* entre 20µs a 49ms e volta-se a baixar para 0V. A leitura da medida através da saída por impulso estará disponível cerca de 1ms depois da instrução no pino *PW* do sensor e poderá ser lida através da função `pulseIn()` que dá a duração do impulso do sensor em milissegundos, divide-se essa medição pelo valor dado pelo manual $147\text{ms}\cdot\text{in}^{-1}$ para obter a distância em polegadas e converte-se esse valor para metros, como mostra o código da figura 4.8. Para se obter a distância através da saída analógica recorre-se à instrução `analogRead` no pino *AN* do sensor cerca de 50ms depois de se dar o impulso para leitura, o valor obtido é um número inteiro da conversão do sinal analógico do sensor para digital com resolução de 10bits, ou seja, a tensão medida é dada pelo produto do número obtido pelo `analogRead()` com 5/1024; para converter a tensão na medida do sensor multiplica-se o valor da tensão por $512/5\text{in}\cdot\text{V}^{-1}$, para terminar basta converter polegadas em metros. [20] A distância em

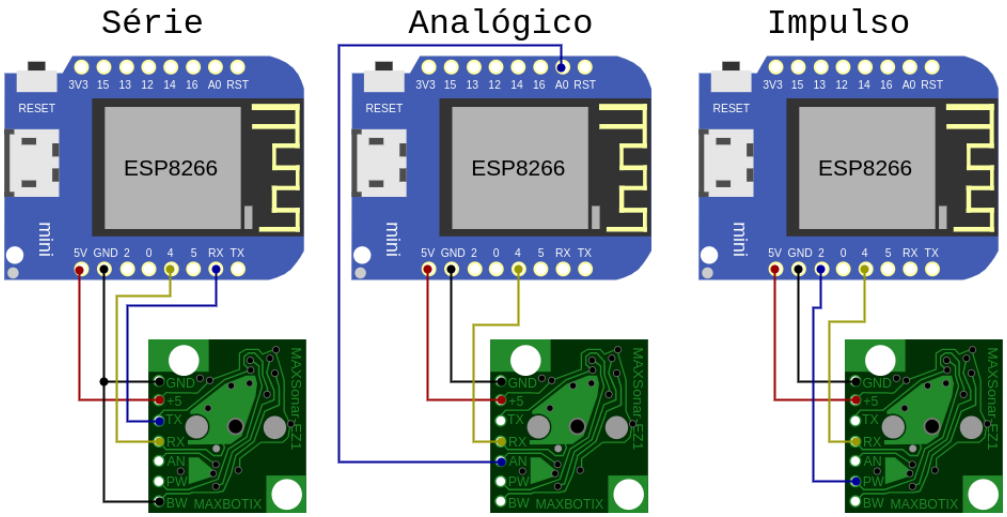

Figura 4.7: Esquema de diferentes montagens do sensor MaxSonar-EZ1 consoante o método usado para captar as leituras

metros pode ser dada pela expressão

$$d = x \cdot \frac{5}{1024}[\mathrm{V}] \cdot \frac{512}{5}[\frac{\mathrm{in}}{\mathrm{V}}] \cdot \frac{25.4}{1000}[\frac{\mathrm{m}}{\mathrm{in}}] \Leftrightarrow d = x \cdot \frac{25.4}{2 \cdot 1000}[\mathrm{m}]$$

onde x é o valor obtido pela função `analogRead()`, esta expressão é usada no código da figura 4.9.

```
#define PIN_PW 2
#define PIN_RX 4
float measure()
{
  float distance;
  digitalWrite(PIN_RX, HIGH);
  distance = pulseIn(PIN_PW, HIGH) * 25.4 / 147.0 / 1000;
  digitalWrite(PIN_RX, LOW);
  return distance;
}
```

Figura 4.8: Código *Arduino* para obter a distância através do sensor MaxSonar-EZ1 pela da saída de impulso

O segundo sensor de distância usado é o HV-SR04, segundo o fabricante tem um alcance máximo de 4 metros no entanto a distância máxima recomendada é de 2,5 metros [21], que é suficiente para medir a distância lateral entre dois veículos na manobra de ultrapassagem. Este sensor tem quatro pinos: dois para a alimentação a 5V, o *trigger* para comandar a leitura e o *echo* para ler a medição. A leitura da distância é semelhante ao MaxSonar-EZ1 na saída por impulso, o microcontrolador envia um impulso de 10µs

```
#define PIN_AN A0
#define PIN_RX 4
float measure()
{
  digitalWrite(PIN_RX, HIGH);
  delay(1);
  digitalWrite(PIN_RX, LOW);
  delay(50); //Aguardar 50ms para a leitura estar disponível
  return analogRead(PIN_AN) * 25.4 / 2000;
}
```

Figura 4.9: Código *Arduino* para obter a distância através do sensor MaxSonar-EZ1 pela saída analógica

ao pino *trigger* e mede a duração do impulso no pino *echo* do sensor, esse valor é o tempo que o sinal levou para chegar ao obstáculo e voltar [22]. A distância entre o sensor e o obstáculo é metade do produto entre o tempo e a velocidade do som ou, conforme diz o manual, multiplicar o tempo por 0,582 para obter a distância em metros. Esta medida estará sujeita a erros causados pela alteração da velocidade do som causados pela variação da temperatura do ar, no entanto como as distâncias são pequenas os desvios não serão significativamente grandes, tendo em conta a aplicação.

O esquema de montagem deste sensor com o microcontrolador Wemos D1 mini é apresentado na figura 4.10, é idêntico à montagem por impulso do sensor MaxSonar-EZ1, mas o código usado para ler a distância tem de ser alterado: o pino *trigger* não pode estar ativo após o impulso para que o sensor não execute uma nova leitura desnecessariamente, e o valor da conversão entre o tempo do impulso e a distância tem de ser corrigido. O código correto para este sensor é apresentado na figura 4.11.

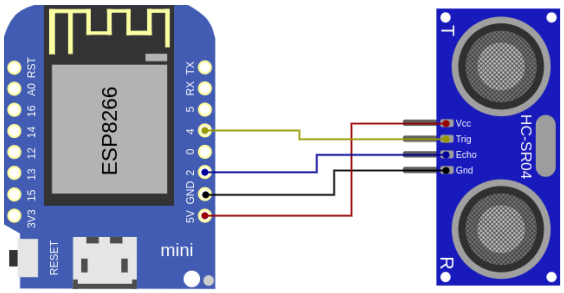

Figura 4.10: Esquema da montagem do sensor HV-SR04

O último sensor apresentado é o VL53L0X, trata-se de um sensor de distância a infravermelhos também conhecido como um sensor *time of flight* com a capacidade de medir distâncias até 2 metros. [23] Comunica por I^2C com o microcontrolador montado conforme a figura 4.12 visto que as portas GPIO 5 e 4 são as linhas *serial clock* e *serial data* respetivamente. Para gerir a interação com o sensor usa-se uma biblioteca de *Arduino* chamada *Adafruit_ VL53L0X*, esta tem métodos para realizar as medições e

```
#define PIN_ECHO 2
#define PIN_TRIG 4
float measure()
{
  digitalWrite(PIN_TRIG, HIGH);
  delayMicroseconds(10);
  digitalWrite(PIN_TRIG, LOW);
  return pulseIn(PIN_ECHO, HIGH) * 0.582;
}
```

Figura 4.11: Código *Arduino* para obter a distância através do sensor HC-SR04

estruturas de dados próprias para as guardar.

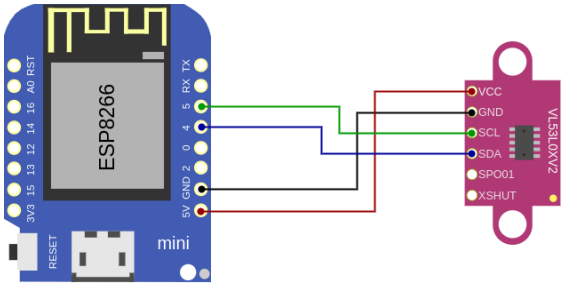

Figura 4.12: Esquema da montagem do sensor VL53L0X

Este sensor seria ideal para medir a distância lateral entre veículos na manobra de ultrapassagem, tal como o sensor HV-SR04, no entanto este não está preparado para realizar medições a distâncias superiores a 40cm quando exposto à luz solar. [23] Dado que os veículos em funcionamento normal estão expostos à luz solar e não se pode garantir que só se use este sensor durante a noite não foram realizadas medições com este sensor.

Centralina

Os automóveis modernos estão equipados com diversos computadores de bordo que processam os valores de centenas de sensores distribuídos pelos diferentes equipamentos e, com base no comportamento desejado pelos fabricantes, controlam os atuadores dos equipamentos. Nos veículos mais recentes existe uma centralina dedicada a cada equipamento do veículo, como por exemplo uma para controlar o motor, outra o sistema anti-bloqueio dos travões, entre outros. [24] As diferentes centralinas comunicam entre si através do protocolo CAN bus, o barramento é também usado para pela ficha de diagnóstico On-Board Diagnostics (OBD) e alguns sensores e atuadores são também controlados pelas centralinas através do CAN bus.

Para monitorizar valores da centralina recorre-se à ficha de diagnostico OBDII do veículo, realizando periodicamente pedidos a um determinado Parameter ID (PID), para obter o valor mais recente de um sensor. Para comunicar corretamente com a ficha de diagnostico é usado o microcontrolador ELM327 que traduz instruções AT fornecidas a

partir de UART para o barramento CAN, e recebe as respostas do barramento e remete-as para a porta série [25]. Estão disponíveis para compra diferentes conectores que usam este microcontrolador, mas é possível encontrar conectores que usam uma ligação por USB, *Wi-Fi*, e UART.

Independentemente do meio de comunicação o processo usado para obter os dados da centralina é relativamente simples, os passos necessários para realizar a monitorização de um valor da centralina são os seguintes:

1. Ligar (por *bluetooth*, *Wi-Fi* ou porta série) ao ELM327
2. Enviar comando AT para configurar a comunicação entre o ELM327 e o CAN bus do veiculo
3. Realizar o pedido, fornecendo o PID do sensor a monitorizar
4. Esperar pela resposta e processar a mensagem
5. Enviar a informação para o servidor
6. Repetir a cada intervalo de tempo a partir do passo 3

O código *Arduino* usado para realizar a monitorização da velocidade do motor através destes passos para uma ligação por *bluetooth* é apresentado na figura 4.13.

```
1  float getRPM(){
2    SerialBT.write("01 0C\r"); // Passo 3
3    String res = SerialBT.readString();
4    char *pos = strstr(res.c_str(), "41 0C");
5    if(!pos) return -1.0;
6    int sum = 0;
7    for(int i=0; i<5; i++){
8      if(i==2) continue;
9      sum <<= 4;
10     sum += (pos[6+i]-'0');
11   }
12   return sum * 0.25; // Passo 4
13 }
14 void setup(){
15   SerialBT.connect("OBDII"); // Passo 1
16   SerialBT.write("AT SP 0\r"); // Passo 2
17 }
18 void loop(){
19   float rpm = getRPM(); // Passo 5 após este ponto
20 }
```

Figura 4.13: Código *Arduino* usado para monitorizar a velocidade do motor

Na linha 16 do código é enviada a mensagem de configuração `"AT SP 0\r"`, este comando AT dá a instrução para o ELM327 encontrar o protocolo de comunicação correto

usado no veículo, este passo é essencial para que o ELM327 comunique corretamente com as centralinas do veículo.

O pedido para monitorização `"01 0C\r"` é composto em duas partes, o código 01 identifica um pedido de uma medição de um sensor, o segundo código 0C identifica o PID do sensor, que é a velocidade de rotação do motor. Enviada essa mensagem ao ELM327 o fluxo de processamento bloqueia na função `readString()` de forma a aguardar a resposta que será uma *String* como por exemplo `"41 0C 14 A7\r"`, onde `"41 0C"` identifica a mensagem como sendo a resposta ao pedido da medição do sensor com o PID 0C, e `"14 A7"` é 4 vezes o valor da velocidade do motor na representação hexadecimal.

Assim que é recebida a mensagem do ELM327 pesquisa-se na mensagem a *String* `"41 0C"`, como está na linha 3 da figura. Caso essa *String* não seja encontrada responde com um valor de erro de -1, a velocidade do motor nunca é um valor negativo logo fica evidente que um erro ocorreu. Para carregar o valor da velocidade do motor recebida para uma variável foi feito, na função `getRPM()`, um ciclo que converte o valor hexadecimal recebido como uma *String* num número inteiro, e retorna um quarto desse valor como um valor de vírgula flutuante.

4.1.3 Caso de estudo

Para o caso de estudo desta dissertação foi escolhida a realização da monitorização da velocidade, posição e velocidade de rotação do motor de um veículo ligeiro motorizado, o modelo *Ford Fiesta* de 2018. Para o efeito foi utilizada a montagem apresentada na figura 4.14.

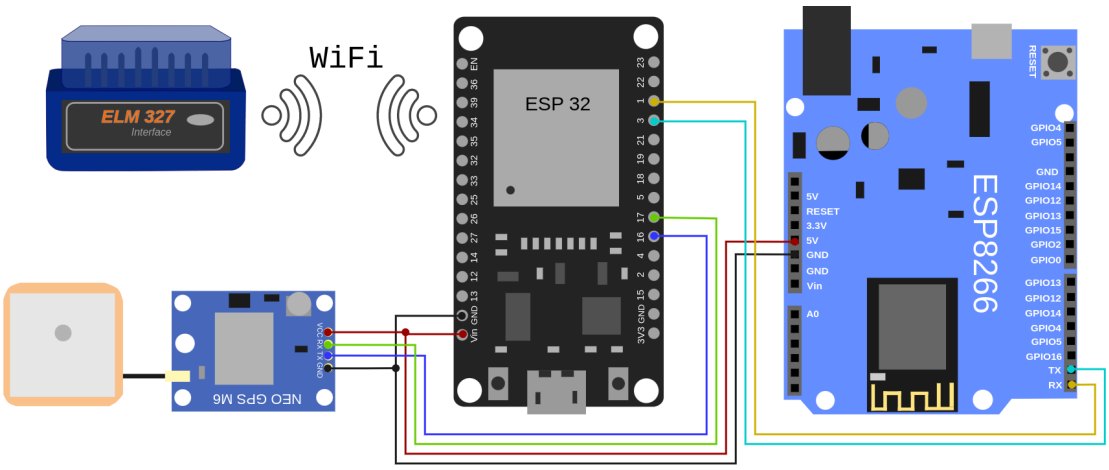

Figura 4.14: Esquema da montagem para o caso de estudo

Recorreu-se a dois módulos ESP, um dedicado ao envio dos dados para o servidor e o outro à recolha de dados do sensor GPS e da centralina do veículo. Para este caso é necessária a utilização de dois módulos porque o ESP32 está ligado à rede *Wi-Fi* do conector ELM327 e o ESP8266 liga à rede *hotspot* gerada pelo *smartphone* do condutor. Para comunicarem entre si usa-se as portas série de cada módulo, assim que o ESP32 obtém os dados envia 22 *bytes* para ESP8266, um *byte* que dá o inicio da transmissão com o valor hexadecimal 7fh, o valor das 5 medições como número de vírgula flutuante,

cada uma de 4 *bytes* de comprimento, e um *byte* para sinalizar o fim da transmissão com o valor hexadecimal de 01h. Para enviar e receber as medições de forma organizada é definida uma estrutura de dados que permite aceder facilmente à representação binária de um número de vírgula flutuante, essa definição é apresentada no código da figura 4.15.

```
1  typedef union{
2    float num;
3    byte bin[4];
4  } binFloat;
```

Figura 4.15: Definição de um número de vírgula flutuante com acesso à sua representação binária

Com esta definição é possível enviar as medições, como por exemplo a latitude, chamando a função `Serial.write(latitude.bin, 4)`, declarando a variável `latitude` como sendo do tipo `binFloat`. Com este tipo de dados o envio das medições é muito simples, no lado da receção da mensagem é necessário ter outros cuidados. O código da figura 4.16 apresenta a função de leitura.

```
1  void readFloat(binFloat *number){
2    for(int i=0;i<4;i++){
3      int r = Serial.read();
4      if(r==-1) return;
5      number->bin[i] = (byte)(r);
6    }
7  }
8
9  bool readData(){
10   int r = -1;
11   do{
12     r = Serial.read();
13     if(r==-1) return false;
14   }while(r!=127);
15
16   readFloat(&latitude);
17     (...)
18   r = Serial.read();
19   if(r!=1) return false;
20   return true;
21 }
```

Figura 4.16: Código usado para receber as leituras no ESP8266

O ciclo da linha 11 procura o *byte* de inicio de mensagem (127_{10}=7fh), descartando todos os *bytes* no *buffer* de entrada até o encontrar, se não esvaziar o *buffer* primeiro.

Este ciclo permite situar o inicio da mensagem, sincronizando a informação que vai sendo recebida, sem este *byte* seria impossível de reconstruir corretamente os números de vírgula flutuante recebidos. Estando sincronizado são lidos 5 números de virgula flutuante: a latitude, longitude, velocidade, direção e velocidade do motor, todos com através da função `readFloat()` que recebe como parâmetro o endereço de memória de um `binFloat`. Esta função lê 4 *bytes* do *buffer* da porta série e grava-os, um a um, na posição de memória recebida.

Para comunicar com o sensor de GPS é usada uma segunda porta série do ESP32, o `Serial2` que usa os pinos GPIO 16 e 17 do microcontrolador. Para comunicar com o ELM327 por *Wi-Fi* usam-se exatamente os mesmos procedimentos que os usados com a interface *bluetooth* no entanto, durante o *setup*, o microcontrolador tem que se ligar à rede gerada pelo ELM327, identificada pelo Service Set Identifier (SSID) *"OBDII"* e, por último, tem que ser aberta uma porta de comunicação TCP para o endereço IP `192.168.0.10`, usada para enviar e receber as mensagens discutidas no capitulo 4.1.2.

Para enviar as medições para o servidor recorre-se à biblioteca `XautoConnection` implementada para o Arduino IDE, desta forma o código no ESP8266 é como o apresentado na figura 4.17.

```
1  XautoConnection* xauto;
2  binFloat latitude, longitude, speed, heading, rpm;
3  void setup(){
4    Serial.begin(115200);
5    WiFi.mode(WIFI_STA);
6    WiFi.begin(ssid, password);
7    while (WiFi.status() != WL_CONNECTED) delay(500);
8    xauto = new XautoConnection(host, fingerprint, USERNAME,
9      PASSWORD, VEHICLE_ID, sensorIDs, 5);
10 }
11 void loop(){
12   while(Serial.available()){
13     if(readData()){
14       values[0]=(latitude.num<-1000.0)?"":String(latitude.num,6);
15       values[1]=(longitude.num<-1000.0)?"":String(longitude.num,6);
16       values[2]=(speed.num<0.0)?"":String(speed.num,1);
17       values[3]=(heading.num<0.0)?"":String(heading.num,0);
18       values[4]=(rpm.num<0.0)?"":String(rpm.num,1);
19       xauto->send(values);
20     }
21   }
22 }
```

Figura 4.17: Código usado para enviar as medições a partir do ESP8266

A cada ciclo é verificado se existe informação no *buffer* de entrada e, caso exista, é executada a função `readData()` para receber a informação. Recebida a informação é verificada cada uma das medições, caso possua um valor de erro a medição enviada para

o servidor é uma *String* vazia, o que dá indicação ao servidor de que se trata de um valor invalido ou ausente. Por último é chamada a função `send()`, que trata de enviar de forma correta toda a informação gravada no *array* `values` para o servidor.

Para este caso de estudo selecionou-se uma rota de Mira com destino a Aveiro pela nacional N109, com um breve percurso urbano em Aveiro e a viagem de volta pela autoestrada A17. O percurso está ilustrado na figura 4.18, à esquerda um mapa geral, mais afastado, que engloba o percurso inteiro e à direita um mapa ampliado do percurso urbano de Aveiro. As setas indicam o sentido da viagem.

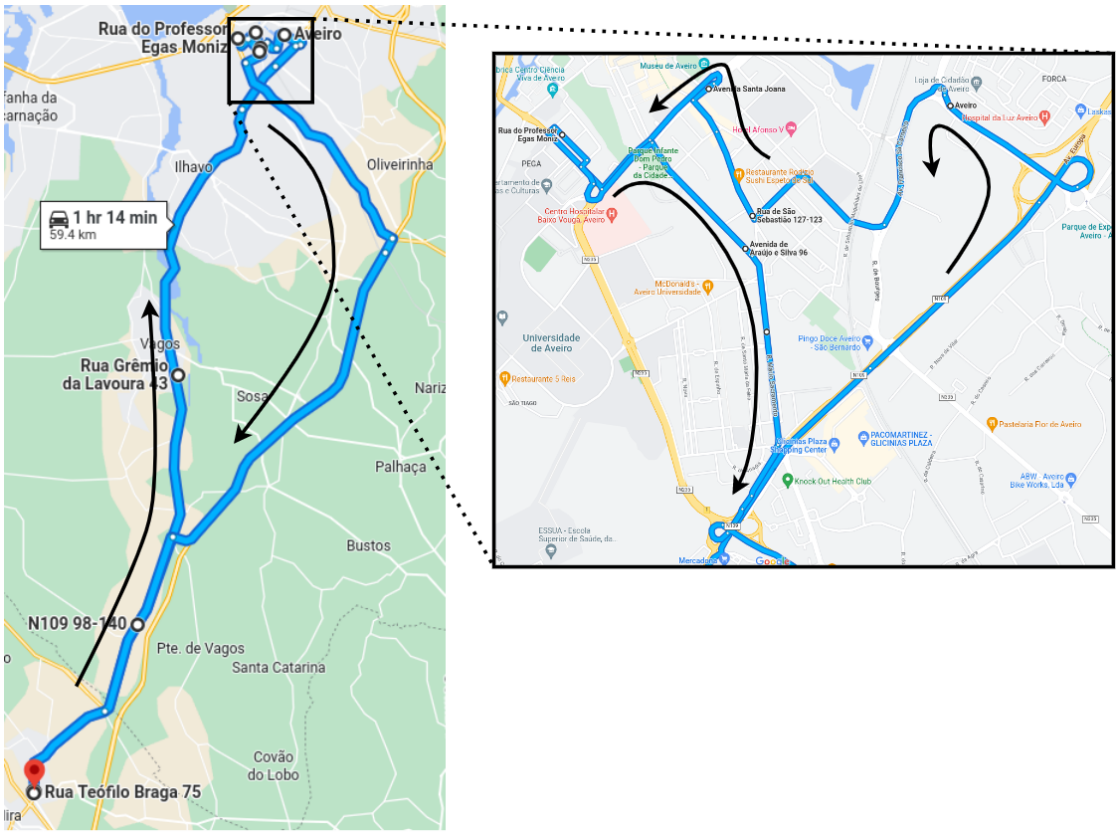

Figura 4.18: Mapa do percurso planeado para o caso de estudo

A tabela 4.1 contém os detalhes da rota segundo as informações fornecidas pelo *Google Maps*. Esta rota permite realizar medições em diferentes tipos de vias, desta forma será possível analisar como varia o comportamento do veículo em diferentes vias e, obtendo a altitude a partir da posição e as acelerações médias calculadas através das velocidades instantâneas, será possível estimar os perfis de consumo para os diferentes tipos de estrada.

4.2 Servidor

Para o servidor estão implementados os serviços descritos no capítulo 3.2. O servidor web que serve a API é o *Apache*, e está implementada em PHP. A base de dados está implementada em *MySQL* e é gerida por *MariaDB*.

Tabela 4.1: Dados relativos à rota selecionada para o caso de estudo

Tipo de Estrada	Distância [km]	Tempo Estimado [min]
Nacional	40	50
Auto-estrada	11.2	6
Urbana	8.2	18
Total	59.4	74

4.2.1 Virtualização

Como foi referido no capitulo 3.2, o *software* do servidor tem de ser facilmente distribuído e configurado, por este motivo foi escolhido o *Docker* pois fornece ferramentas como o *Docker Compose* que constroem e iniciam a execução da plataforma com um só comando.

O *software* do servidor é composto por dois *containers*, um corre o servidor web *Apache* e o outro o servidor de gestão de dados *MariaDB*. A configuração do *docker-compose* é apresentada no código da figura 4.19, onde estão declarados dois serviços: a `api` cuja imagem é um sistema *ubuntu* com o *Apache2* instalado e os ficheiros e código da interface, e a *database* que é construída a partir da imagem oficial [26] modificada para montar as tabelas da plataforma durante a primeira execução. Nesta configuração é também criada uma rede virtual entre as duas máquinas, onde apenas a porta 443 da API é exposta ao exterior.

Para a configuração dos parâmetros relativos a cada instância da plataforma é usado um ficheiro com o nome *.env* que contém variáveis como o domínio do servidor e *passwords* de administração. Cada parâmetro é carregado como variável de sistema pelas máquinas virtuais onde o código de inicialização as usa para configurar os serviços. Para além de editar este ficheiro o administrador do sistema tem de fornecer o certificado SSL associado ao domínio do servidor, que é automaticamente inserido no *container* da API. O administrador de sistema pode também editar o nome da plataforma e fornecer um logótipo personalizado para ser apresentado na página da interface visual para identificar a instituição dos utilizadores, por exemplo.

4.2.2 Interface

A interface do servidor é a API desenvolvida em PHP que implementa os serviços discutidos no capítulo 3.2.1. O funcionamento da interface pode ser separado em três fases de processamento:

1. Receber e processar o pedido do cliente
2. Executar o pedido
3. Construír e enviar a resposta ao pedido

Em primeiro lugar é necessário compreender qual é o objetivo do pedido, que é definido pelo URL e pelo método HTTP usado. Isto é concluído pelo código do ficheiro *api.php*, no entanto é necessário redirecionar internamente os pedidos para a API de forma a serem processados por este ficheiro, para isto configura-se o *Apache* de forma a

```yaml
version: "3.7"
services:
  api:
    build:
      dockerfile: ./docker/api/Dockerfile
    env_file:
      - .env
    ports:
      - "443:443"
    networks:
      xauto:
        ipv4_address: 172.0.0.2
  database:
    build:
      dockerfile: ./docker/database/Dockerfile
    env_file:
      - .env
    networks:
      xauto:
        ipv4_address: 172.0.0.3
networks:
  xauto:
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet: 172.0.0.0/24
```

Figura 4.19: Código de configuração dos serviços da plataforma para o *docker-compose*

usar o módulo `mod_rewrite` de forma a rescrever o pedido internamente para ser processado pelo ficheiro correto, com a *flag passthrough* (`[PT]`) para não alterar o URL original, como mostra o código da figura 4.20. [27]

```
<Directory /var/www/>
  ...
  RewriteEngine on
  RewriteRule "^.+/api/users.+$" "/api/src/api.php" [PT]
  RewriteRule "^.+/api/vehicles.+$" "/api/src/api.php" [PT]
  RewriteRule "^.+/api/login.+$" "/api/src/api.php" [PT]
  RewriteRule "^.+/api/src/(?!api.php).*$" "/api/src/api.php" [PT]
  ...
</Directory>
```

Figura 4.20: Configuração do *Apache* para os pedidos serem processados pelo *api.php*

Quando é feito um pedido à API o servidor processa o código *api.php* que, consoante o URL do pedido, chama os métodos relevantes à execução do pedido. A implementação desta funcionalidade está feita através do processamento da *String* do URL do pedido, primeiro o caminho é explodido nos diversos diretórios, é feita uma verificação para garantir que está bem construído, é identificado o *endpoint* pedido através de uma série de blocos `switch-case` e, por último, consoante o método HTTP usado é chamada a função relevante. Um resumo deste código, para processar os pedidos dos *endpoints* relativos aos utilizadores, está na figura 4.21.

Para processar a autenticação do utilizador no servidor é usada um JWT gerado quando são enviadas as credenciais corretas do utilizador para o *endpoint* `/login` através de um pedido do tipo `POST`. Para isso recorre-se à biblioteca *Really Simple JSON Web Tokens* [29] que implementa os métodos necessários para gerar *tokens* e verificar se os que são recebidos são válidos. No *payload* do JWT segue o ID e o nome do utilizador ao qual foi fornecido o *token*, e está também gravada a hora de expiração que é uma hora após ter sido criado. Quando um pedido é recebido o *token* é obtido a partir do *header* `Authorization`, é verificado se é válido com o método `Token::validate()`, e é confirmado que o recurso que o cliente está a tentar aceder pertence ao mesmo ID de utilizador presente no *payload* do JWT. Caso alguma destas condições não seja satisfeita é gerada uma resposta de erro e o pedido não é executado.

De forma a organizar o código este está programado segundo um design Object-Oriented Programming (OOP) onde é definida uma classe para cada tipo de recurso, cada um em ficheiros separados, que são a `User` que representa um utilizador da plataforma, a `Vehicle` que representa um veículo de um utilizador, a `Sensor` que representa um sensor que pertence a um veículo e a `Measurement` que representa uma amostra de medições enviada por um veículo. Cada classe possui os campos que definem um recurso e métodos para interagir com as tabelas na base de dados de cada recurso. Para exemplificar está apresentado no código da figura 4.22 as declarações dos campos e métodos da classe `Measurement`.

Esta classe possui 4 campos: o ID da amostra no campo `$id`, o *timestamp* do momento em que foi obtida no campo `$time`, o *array* dos IDs dos sensores que participaram na

```php
$query = explode('/', $_SERVER['REQUEST_URI']);
switch ($query[2]) {
case 'users':
  if(!isset($query[3]) || empty($query[3])) {
    /* Procurar ou registar utilizador */
  } else {
    /* Selecionar utilizador */
    $userID = intval($query[3]);
    if(!isset($query[4]) || empty($query[4])) {
      switch ($_SERVER['REQUEST_METHOD']){
      case 'GET':
        /* Responder com dados do utilizador */
      case 'PUT':
        /* Atualizar dados do utilizador */
      case 'DELETE':
        /* Eliminar utilizador */
      }
    }
...
```

Figura 4.21: Resumo do código PHP usado no ficheiro *api.txt*

```php
class Measurement {
  /* Declaração dos campos da classe e construtor */
  public int $id, $time;
  public array $values, $sensors;
  public function __construct($id, $time, $values, $sensors);
  /* Métodos auxiliares */
  public static function exists($id) : bool;
  public static function number($vehicle_id) : int;
  /* Métodos para eliminar amostras da base de dados */
  public static function deleteMeasurement($id) : void;
  public static function deleteAllMeasurements($vehicle_id) : void;
  /* Método para gravar uma amostra na base de dados */
  public static function createMeasurement($vehicle_id, $sensors,
    $values) : int;
  /* Métodos para obter amostras gravadas na base de dados */
  public static function getMeasurement($id) : Measurement;
  public static function getMeasurements($vehicleId) : array;
  public static function getMeasurementsLimitOffset($vehicleId,
    $limit, $offset = 0) : array;
}
```

Figura 4.22: Declarações de campos e métodos da classe `Measurement` em PHP

amostra no campo `$sensors` e o *array* dos valores de cada sensor no momento em que a amostra foi obtida no campo `$values`; e define vários métodos estáticos para ler e gravar amostras na base de dados, ver quantas amostras estão gravadas ou consultar se um dado ID de amostra existe na base de dados.

Quando é recebido um pedido `GET` no o *endpoint* `/vehicles/vID/measurements` no código do *api.php* é invocado o método `Measurement::getMeasurements(vID)` que faz o servidor *Apache* realizar a *query* SQL

```
SELECT id FROM samples WHERE vehicle_id = vID ORDER BY id DESC;
```

para a base de dados de forma a obter todas as amostras de um dado veículo com ID `vID` e, para cada amostra obtida, são pedidos os IDs dos sensores e os valores de cada medição dessa amostra através de uma série de *querys* com o formato

```
SELECT sensor_id, value FROM measurements WHERE sample_id = numSample;
```

onde o `numSample` é o valor do ID da amostra que é iterado entre todas as amostras recebidas pela primeiro pedido. Tendo a resposta da base de dados é chamado o construtor desta classe com a informação recebida, formando um novo objeto do tipo `Measurement` para cada amostra, que é introduzido num *array* que, quando completo com todas as amostras do veículo, será devolvido ao *script* que chamou a função onde, neste exemplo, será a função principal no *api.php*.

Por último o servidor tem de formar a resposta ao pedido. Quando o cliente realiza pedidos `GET` o servidor terá de enviar a informação pedida no formato JSON que, para o caso de programação OOP em PHP é obtida diretamente através do método `json_encode($objeto)`, reduzindo a complexidade do código. Para os outros tipos de pedido o servidor verifica se são bem executados e responde com o código de estado correto, conforme estipulado pela arquitetura REST.

4.2.3 Base de dados

A base de dados está implementada conforme a estrutura apresentada no esquema da figura 3.8. Foram acrescentados alguns detalhes para agilizar o funcionamento como a indexação de colunas para uma pesquisa mais rápida e a eliminação automática de registos filho quando é eliminado um registo pai. Nos próximos parágrafos será detalhada a implementação de cada tabela.

A tabela dos utilizadores é criada com a instrução SQL apresentada na figura 4.23, possui 4 colunas: o `id` é um número inteiro, único para cada utilizador, que identifica o utilizador na plataforma e é atribuído automaticamente pela base de dados de forma incremental; o `name` e o `email` são o nome de utilizador e o e-mail do utilizador, respetivamente, são diferentes entre utilizadores (garantido pela restrição `UNIQUE`), são variáveis do tipo `TINYTEXT` (o que significa que armazenam uma *String* de 255 caracteres) e são indexadas para realizar procuras por nome ou e-mail mais rapidamente; por último a coluna `password` armazena a *hash* gerada pela função PHP `password_hash` na palavra-passe do utilizador, é armazenada como uma variável do tipo `VARCHAR`, porque a função codifica a *hash* em base 64, e tem 255 caracteres de comprimento como recomendado pelo manual da função. [**?**]

A próxima tabela a ser criada é a `vehicles` com a instrução da figura 4.24, onde são armazenados os veículos registados na plataforma, que tem 4 colunas: o `id` único de cada

```sql
CREATE TABLE users(
  id INT NOT NULL AUTO_INCREMENT,
  name TINYTEXT UNIQUE NOT NULL,
  password VARCHAR(255) NOT NULL,
  email TINYTEXT UNIQUE NOT NULL,
  PRIMARY KEY (id),
  INDEX idx_name (name),
  INDEX idx_email (email)
);
```

Figura 4.23: Código SQL para criar a tabela dos utilizadores

veículo atribuído pela base de dados; o `name` que guarda o nome atribuído ao veículo; o `color` onde é armazenada como texto a cor que representa o veículo na plataforma na forma hexadecimal; e o `user_id` que é uma referência ao `id` do registo da tabela `users` relativo ao utilizador que criou o veículo, esta coluna possuí a opção de chave externa `ON DELETE CASCADE` que elimina automaticamente todos os veículos de um utilizador quando este for removido da base de dados.

```sql
CREATE TABLE vehicles(
  id INT NOT NULL AUTO_INCREMENT,
  user_id INT NOT NULL,
  name TINYTEXT NOT NULL,
  color TINYTEXT NOT NULL,
  PRIMARY KEY (id),
  FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
);
```

Figura 4.24: Código SQL para criar a tabela dos veículos

A tabela `sensors` armazena a informação acerca dos sensores instalados nos veículos, a instrução que cria esta tabela é apresentada na figura 4.25. A tabela possui 4 colunas: o `id` que identifica o sensor; o `name` armazena o nome do sensor; o `type` guarda o tipo de dados da informação que o sensor envia, como só pode ser *float*, *int* ou *string* é armazenado como um `ENUM` na base de dados; a coluna `vehicle_id` é a referência ao `id` do veículo onde o sensor está instalado, onde tem a opção `ON DELETE CASCADE` que apaga automaticamente da base de dados o sensor caso o veículo seja eliminado.

A tabela das amostras, chamada `samples`, armazena a data e hora do momento em que um veículo envia as medições dos seus sensores para o servidor e, para isso, necessita de 3 colunas, como mostra o código da figura 4.26: o `id` único para identificar a amostra, a `stamp` que grava automaticamente o momento em que a amostra foi recebida no servidor através da definição do valor padrão (`DEFAULT`) como o `CURRENT_TIMESTAMP(3)` que é o momento avaliado pela data e hora do sistema operativo, com precisão à milésima de segundo; e por último o `vehicle_id` que identifica o veículo a que pertence a amostra

```sql
CREATE TABLE sensors(
  id INT NOT NULL AUTO_INCREMENT,
  vehicle_id INT NOT NULL,
  name TINYTEXT NOT NULL,
  type ENUM('float', 'int', 'string'),
  PRIMARY KEY (id),
  FOREIGN KEY (vehicle_id) REFERENCES vehicles(id) ON DELETE
  CASCADE
);
```

Figura 4.25: Código SQL para criar a tabela dos sensores

recebida.

```sql
CREATE TABLE samples(
  id INT NOT NULL AUTO_INCREMENT,
  stamp TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP(3),
  vehicle_id INT NOT NULL,
  PRIMARY KEY (id),
  FOREIGN KEY (vehicle_id) REFERENCES vehicles(id) ON DELETE
  CASCADE
);
```

Figura 4.26: Código SQL para criar a tabela das amostras

A última tabela grava todas as medições enviadas para o servidor, o código usado para criar esta tabela é apresentado na figura 4.27. Esta tabela é constituída por 4 colunas: o identificador único `id`; o valor da medida gravado como uma *String* num campo do tipo `TINYTEXT`, onde fica armazenada sob a forma de texto o valor da medida (que pode ser lido como um número inteiro, de vírgula flutuante ou simplesmente como uma *String*); a referência ao sensor a que esta medida se refere na coluna `sensor_id`; e uma referência à amostra onde esta medida foi recebida na coluna `sample_id`. Ambas as referências externas desta tabela possuem a opção `ON DELETE CASCADE` o que significa que se o sensor que realizou esta medição ou a amostra onde foi enviado forem eliminado esta medição será automaticamente eliminado.

4.3 Interface Visual

A interface visual do utilizador com a plataforma é uma página web com a estrutura definida em HTML, formatada com o estilo definido em Cascading Style Sheets (CSS), onde código JS controla os diferentes painéis e informações apresentadas no *browser* conforme as ações do utilizador. A estrutura da interface está implementada de forma a estar conforme a solução discutida no capitulo 3.3.1.

```
CREATE TABLE measurements(
  id INT NOT NULL AUTO_INCREMENT,
  sensor_id INT NOT NULL,
  sample_id INT NOT NULL,
  value TINYTEXT,
  PRIMARY KEY (id),
  FOREIGN KEY (sensor_id) REFERENCES sensors(id) ON DELETE
  CASCADE,
  FOREIGN KEY (sample_id) REFERENCES samples(id) ON DELETE
  CASCADE
);
```

Figura 4.27: Código SQL para criar a tabela das medições

4.3.1 Estrutura

A estrutura da página da interface visual é definida num único ficheiro HTML, cada painel é um elemento `<div>` que pertence à classe `frame` e, quando a página é carregada, não é apresentado no ecrã porque, por defeito, está definido o atributo `style="display: none"` para todos os painéis. Na figura 4.28 é apresentado o código HTML que define o painel do *dashboard*, dentro do painel é declarado um segundo `<div>` que garante um disposição lado-a-lado entre o mapa e a lista dos sensores de cada veículo, como está ilustrado na figura 3.10. Dentro do `<div id="mapid">` é apresentado, através de JS, um mapa gerado pela biblioteca *leaflet* [30] que apresenta o local onde cada veiculo se encontra através de um circulo da cor do veículo, caso a latitude e longitude sejam definidas. Do lado direito da *dashboard* é apresentada a lista de sensores que é construída automaticamente através de JS.

```
<div id="frameDashboard" class="frame" style="display: none">
  <div style="display: flex; flex-direction: row nowarp">
    <div id="mapid" style="height: 600px; width: 30%"></div>
    <div id="dashboard" class="dashboard"></div>
  </div>
</div>
```

Figura 4.28: Código HTML que define o painel de *dashboard*

Quando o utilizador seleciona outro painel a partir do menu de navegação no topo da pagina é chamado um método JS que altera o atributo `style` do painel antigo para `"display: none"`, de forma a o esconder, e remove esse atributo do novo painel selecionado. Esse método está definido no ficheiro *page.js* como `selectFrame(frame)`, que trata da troca de painéis e chama outras funções para popular o conteúdo de cada painel conforme é selecionado. A pagina com o painel de *dashboard* selecionado está apresentada na captura de ecrã da figura 4.29, no topo da pagina é apresentado o logótipo e o nome dado à plataforma, que podem ser alterados pelo administrador do sistema. Logo

de seguida está o menu de navegação que indica o painel selecionado a cor-de-laranja e, por último, o mapa e lista de sensores.

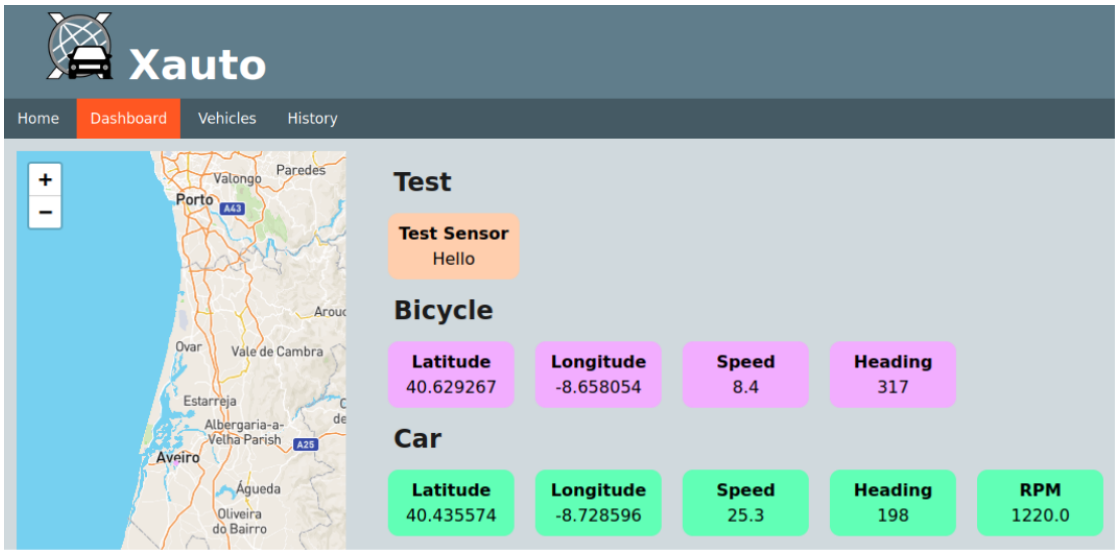

Figura 4.29: Captura de ecrã do painel de *dashboard*

Todos os painéis funcionam com o mesmo principio, a estrutura é definida em HTML e o *browser* executa código JS para obter informação e a apresentar ao utilizador. No código da figura 4.30 é definido o painel de edição de veículos e sensores como o apresentado na figura 3.11, neste são criadas 3 colunas apresentadas lado-a-lado, uma para selecionar o veículo a editar, a segunda apresenta os controlos de edição e a terceira tem os sensores do veículo.

Na linha 6 do código é declarado um botão para criar um novo veículo, quando o utilizador pressiona esse botão é chamada a função `openCreateVehicleModal()`, uma função JS que faz abrir um *modal* (uma janela dentro da página) com um formulário para preencher o nome do veiculo e selecionar a cor. Para selecionar a cor o utilizador tem acesso a uma palete de cores, uma funcionalidade implementada pela biblioteca *jscolor* [31], tanto na criação de um novo veiculo como na sua edição.

Na segunda coluna é apresentado um formulário para editar o veículo, ao submeter o formulário é chamado o procedimento `editVehicleProcedure()` que faz o *browser* enviar as novas informações para o servidor. Após o formulário de edição são apresentados dois botões, um que elimina o veiculo selecionado e outro que adiciona um sensor ao veiculo. Cada um dos botões esta associado a um procedimento JS que faz o *browser* executar pedidos HTTP ao servidor.

Na terceira coluna são apresentados os sensores registados no veiculo, cada um é apresentado como um formulário onde o utilizador pode mudar o nome e o tipo de dados de cada sensor, ou o eliminar da plataforma. A lista de sensores é gerada automaticamente pelo *browser* dentro do elemento da linha 31 após um pedido ao servidor. A figura 4.31 apresenta este painel desenhado no *browser*.

```
<div id="frameVehicles" class="frame" style="display: none">
  <div style="display: flex; justify-content: start">
    <div class="vehicles">
      <h2>Vehicles</h2>
      <div id="vehicles"></div>
      <button onclick="openCreateVehicleModal()">
        Create New
      </button>
    </div>

    <div id="vehicleOptions" class="vehicleOptions"
        style="display: none">
      <h2>Options</h2>
      <form name="editVehicleForm"
          onsubmit="editVehicleProcedure(); return false;">
        <input type="text" name="vName">
        <br>
        <input data-jscolor="{}" onInput="changePreviewColor(this)"
          name="vColor" style="width:154px">
        <br>
        <label id=vehicleIdLabel></label>
        <button type="submit">Edit</button>
      </form>
      <button onclick="deleteVehicleProcedure()">Delete</button>
      <button onclick="addSensor()">Add Sensor</button>
    </div>

    <div id="vehicleSensors" class="vehicleSensors"
        style="display: none">
      <h2>Sensors</h2>
      <div class="sensorList" id="sensorList"></div>
    </div>
  </div>
</div>
```

Figura 4.30: Código HTML que define o painel da edição de veículos e sensores

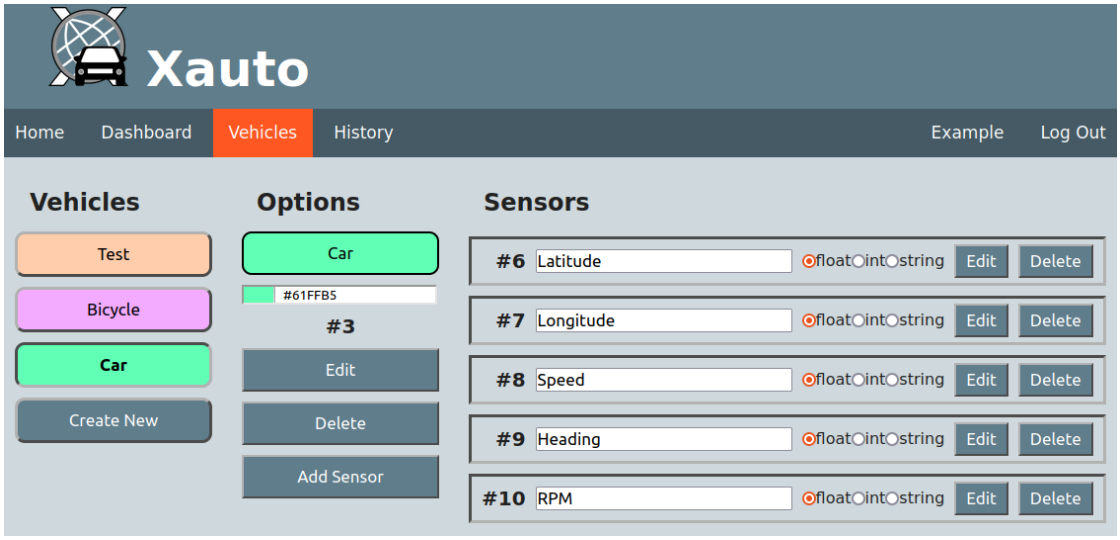

Figura 4.31: Captura de ecrã do painel da edição de veículos e sensores

4.3.2 Funcionamento

Como foi detalhado no capitulo 3.3.2 o funcionamento da interface visual baseia-se na realização de pedidos, em *backgroud*, para o servidor. Estes pedidos são realizados através da função JS `fetch(url, parametros)`, onde os parâmetros usados especificam o método HTTP usado, os *headers* passados (incluindo o JWT) e os *payloads* enviados.

O conjunto de interações entre o utilizador, a interface visual e a plataforma que foram implementadas para se realizar a monitorização, em tempo real, do conjunto de veículos do utilizador, ilustradas no diagrama de sequencia da figura 3.13, foram as seguintes:

- Autenticar o utilizador na plataforma
- Obter os veículos registados em nome do utilizador
- Obter os sensores instalados em cada veiculo
- Desenhar no ecrã uma caixa para cada sensor
- Obter, a cada segundo, os valores mais recentes de cada sensor

Para realizar a autenticação no servidor o utilizador pressiona o botão *logIn* do menu de navegação, que faz abrir uma janela *modal*, onde insere os dados do utilizador: o nome e palavra-passe. Ao pressionar o botão de validação o *browser* envia os dados inseridos no formulário ao servidor, que deverá responder com o JWT ou um código de erro, caso os dados não sejam validos. O código JS que implementa esta funcionalidade está resumido no código apresentado na figura 4.32.

A declaração da função identifica-a como sendo assíncrona, o que significa que quando é chamada o fluxo de processamento não aguarda que esta função termine a sua execução, e é definida desta maneira para permitir chamar a função `getJWT()`, declarada na linha 13, que bloquearia o fluxo à espera da resposta do servidor. A função inicia obtendo o nome e a *password* do utilizador a partir do formulário, e gera um objeto para enviar no *payload* da mensagem para o servidor.

```
async function logInProcedure(){
  let name = logInForm.name.value;
  let password = logInForm.password.value;

  let data = {name:name, password:password};
  let token = await getJWT(data);
  if(!token) return;

  window.sessionStorage.setItem("accessToken", token.accessToken);
  pageLogIn(token.accessToken);
}

async function getJWT(data){
  let response = await fetch("api/login/",
    {method: "POST", body: JSON.stringify(data),
      headers:{"Content-Type":"application/json"}});

  if(response.status != 200) return "";
  return await response.json();
}
```

Figura 4.32: Código JS usado para autenticar o utilizador na interface visual

O *browser* envia a mensagem ao servidor através da função `getJWT()`, é usado o método `POST` e são enviados os dados em JSON no corpo da mensagem. A função verifica o código da resposta do servidor, se não teve sucesso devolve a *String* vazia, se teve sucesso devolve o JWT.

Recebida a resposta da função é feita uma verificação, se se trata de uma *String* vazia a função termina e é apresentada uma mensagem de erro ao utilizador, se foi recebido o JWT este é guardado na memória da sessão do *browser* e é executada uma função que desbloqueia os painéis de *dashboard*, edição de veículos e histórico de medições.

Terminada a autenticação o utilizador pressiona o botão *dashboard* do menu de navegação. Como foi descrito anteriormente, ao selecionar este painel o *browser* tem de construir um retângulo para apresentar as medições de cada sensor, para isto é necessário primeiro obter os veículos e sensores do utilizador. O código usado para este efeito esta apresentado na figura 4.33, começa por obter o `<div>` destinado ao *dashboard* e armazena-o numa variável. De seguida pede ao servidor a lista de veículos do utilizador ao servidor através da função `getVehicles()` e, para cada um, cria um novo elemento `<div>`. O mesmo é feito para cada um dos sensores instalados em cada veículo e, no fim, os elementos construidos (e preenchidos com a informação de cada sensor) são inseridos no *dashboard*. Terminada a construção é iniciada uma função, repetida a cada segundo, que atualiza os valores das medições indicados nos elementos dos sensores automaticamente.

As funções `getVehicles()` e `getSensors()` são idênticas, apenas muda o URL do pedido. Estas funções usam o JWT para autenticarem o utilizador no servidor, enviando

```
async function buildDashboard(){
  let dashboardDiv = document.getElementById("dashboard");

  let vehicles = await getVehicles();
  for(const vehicle of vehicles){
    let vehicleDiv = document.createElement("div");
      (...)
    let sensors = await getSensors(vehicle.id);
    for(const sensor of sensors){
      let sensorDiv = document.createElement("div");
        (...)
      sensorsDiv.appendChild(sensorDiv);
    }
      (...)
    dashboardDiv.appendChild(vehicleDiv);
  }
  updateDashboardInterval = setInterval(updateDashboard, 1000);
}

async function getVehicles(){
  let auth="Bearer "+window.sessionStorage.getItem("accessToken");
  let url = "api/users/" + userID + "/vehicles/";
  let response=await fetch(url, {headers:{"Authorization": auth}});
  return await response.json();
}
```

Figura 4.33: Código JS resumido das funções usadas para gerar a *dashboard*

a *String* gravada na variável `auth` no *header Authorization* do pedido. A sequencia dos pedidos executados pelo *browser* para o servidor de forma a gerar a *dashboard* da interface visual da plataforma está ilustrada na captura de ecrã da figura 4.34.

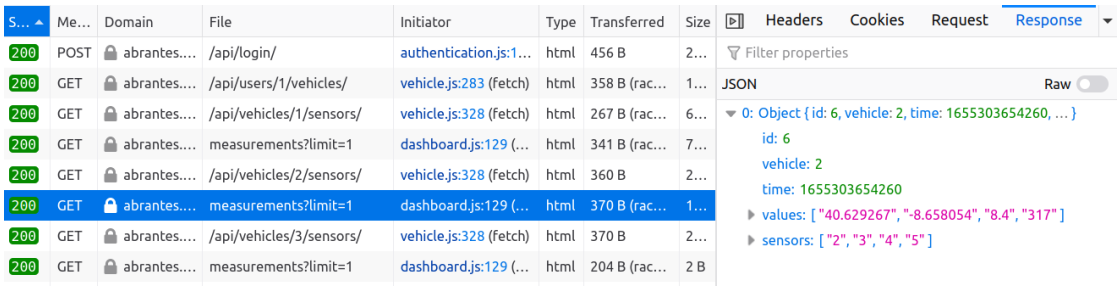

Figura 4.34: Captura de ecrã dos pedidos do *browser* para o servidor para gerar a *dashboard*

Capítulo 5

Análise de Resultados

Neste capitulo serão analisados os resultados da monitorização do caso de estudo descrito no capitulo 4.1.3. O trajeto monitorizado é apresentado no mapa da figura 5.1, com um gradiente baseado na velocidade registada em cada ponto do mapa.

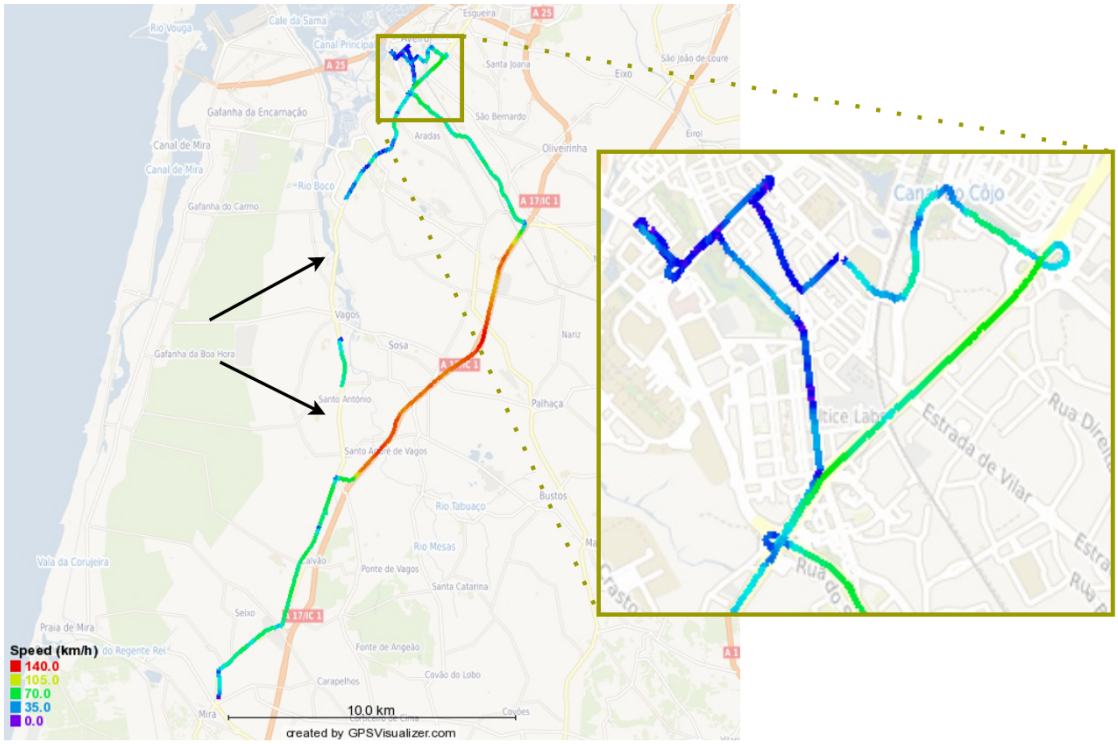

Figura 5.1: Vista geral do trajeto monitorizado

Na figura estão assinaladas duas falhas registadas no processo de monitorização, as quais ocorreram durante determinados períodos de tempo devido a medições invalidas enviadas para o servidor. Estas amostras foram enviadas sem nenhuma informação tendo sido causadas pela dessincronização da comunicação entre os dois microcontroladores. Este fenómeno ocorreu nos momentos em que a comunicação com o servidor levou mais tempo que o esperado (cerca de 10 segundos), provavelmente devido a um sinal fraco de GSM. Durante esses instantes o *buffer* da porta série encheu, corrompendo a informa-

ção, causando a dessincronização. Este problema foi resolvido com a limpeza do *buffer* sempre que a mensagem do ESP32 for incorretamente lida através da chamada à função `Serial.flush()`.

O número de amostras registadas no servidor é apresentado na tabela 5.1. De todas as mensagens recebidas apenas 2582 foram amostras válidas, sendo as restantes 1178 amostras foram recebidas sem medições durante os períodos iniciais, enquanto o sinal GPS não é válido, e durante os erros assinalados na figura 5.1. Algumas das amostras válidas não são relevantes porque foram enviadas antes do inicio da marcha do veículo, dessa forma não foram usadas na análise dos dados.

Tabela 5.1: Distribuição da amostragem

Amostras		
Registos	3760	
Válidos	2582	
Usados	2430	
Nacional	1250	(51.4%)
Urbano	948	(39.0%)
Autoestrada	232	(9.6%)

Os dados monitorizados foram usados para calcular o Vehicle-Specific Power (VSP) através da fórmula [32]

$$\text{VSP} = v \cdot (1.1 \cdot a \cdot 9.81 \cdot \sin(\phi) + 0.132) + 0.000302 \cdot v^3$$

onde v é a velocidade do veículo, a a aceleração e ϕ o ângulo do declive da estrada, obtido a partir da expressão

$$\phi = \arctan(\frac{\Delta h}{d})$$

onde Δh é a variação na altura e d a distância. A velocidade usada no cálculo do VSP é a velocidade adquirida pelo GPS, convertida em metros por segundo, a aceleração é calculada a partir da variação da velocidade monitorizada como $a = \Delta v/\Delta t$, onde Δt é o tempo entre as duas amostras. A altura foi obtida para todos os pontos através da ferramenta *GPSVisualizer* [33], e a distância d é calculada como a distância geodésica entre as duas posições.

Para estimar as emissões do veículo usa-se o valor do VSP calculado para determinar o modo VSP, usando os intervalos de valores apresentados na tabela 5.2. Os valores das emissões instantâneas usados foram obtidos para um veículo a gasóleo com características diferentes do veículo monitorizado, logo os valores obtidos não estão corretos, no entanto são uma boa estimativa para ilustrar os valores que podem ser obtidos a partir da monitorização com a plataforma desenvolvida. Através da estequiometria da combustão foi possivel estimar o consumo de combustível no percurso, para isso recorreu-se aos valores da emissão de CO_2 que, divididos por 2.64kg/l [34] deram o volume de combustível necessário para a combustão.

Em primeira análise é verificada a frequência que cada modo VSP ocorre durante a viagem para os diferentes tipos de estrada. No gráfico da figura 5.2 são apresentados os resultados normalizados para se poder comparar visualmente a distribuição da frequência do modo VSP entre os diferentes tipos de estrada. Observando o gráfico é evidente que

Tabela 5.2: Intervalos para a classificação do modo VSP e valores usados para estimar as emissões do veículo [32]

		Emissões			
Modo VSP	Intervalo VSP	CO_2 [g/s]	CO [mg/s]	NO_x [mg/s]	HC [mg/s]
1	VSP $\in]-\infty, -2[$	0.21	0.03	1.29	0.14
2	VSP $\in [-2, 0[$	0.61	0.07	2.62	0.11
3	VSP $\in [0, 1[$	0.73	0.14	3.38	0.11
4	VSP $\in [1, 4[$	1.50	0.25	6.05	0.17
5	VSP $\in [4, 7[$	2.34	0.29	9.36	0.20
6	VSP $\in [7, 10[$	3.29	0.69	12.53	0.23
7	VSP $\in [10, 13[$	4.20	0.58	15.48	0.24
8	VSP $\in [13, 16[$	4.94	0.64	17.82	0.23
9	VSP $\in [16, 19[$	5.57	0.61	21.32	0.24
10	VSP $\in [19, 23[$	6.26	1.01	32.53	0.28
11	VSP $\in [23, 28[$	7.40	1.15	55.75	0.37

em auto-estrada ocorrem os valores mais elevados do VSP, e que nas zonas urbanas o modo mais comum é o 3, o modo do veículo enquanto está parado (VSP=0 equivale ao modo 3) e em movimento a baixas velocidades e acelerações.

A velocidade do motor média em função do modo VSP está relacionada no gráfico da figura 5.3. Em primeira análise observa-se que, nas estradas nacionais, a velocidade do motor é superior em modos VSP maiores, no entanto a frequência de modos VSP superiores ao modo 5 é muito reduzida na estrada nacional, o que leva a acreditar que a relação da velocidade do motor com o modo VSP é inexistente para o conjunto de dados obtidos. A partir deste gráfico apenas se concluí que, em média, a velocidade do motor é superior em auto-estrada. Possivelmente os resultados ilustrados no gráfico estão influenciados por uma discrepância detetada durante a recolha dos dados, o valor da velocidade do motor em ralenti monitorizada é de 1200rpm sendo que o valor real deverá rondar as 800rpm, conforme a medição do tacómetro.

Com base nos valores do modo VSP realizou-se o perfil de consumo de combustível ao longo do tempo que está apresentado junto do perfil de velocidade no gráfico da figura 5.4.

Por último estão calculados, a partir do produto do valor das emissões instantâneas pelo tempo, os valores de emissões e do consumo de combustível totais que, quando divididos pela distância total percorrida, fornecem o valor médio de emissões e consumo por unidade de comprimento. Na tabela 5.3 são apresentados esses valores convertidos em unidades por 100km para comparação com valores geralmente usados. Observando apenas o consumo, valores entre 4.10l/100km e 6.38l/100km são bastante económicos, expectáveis num veículo moderno, o que indica que os cálculos efetuados e os dados monitorizados não devem estar muito longe da realidade.

Uma conclusão que pode ser retirada da tabela 5.3 é que a condução em estradas urbanas é a que tem maiores consumos (e emissões) por quilometro, o que faz sentido tendo em conta o perfil de velocidades neste tipo de estradas, como mostra a figura 5.4, geralmente o veículo encontra-se parado, em semáforos ou no trânsito, a consumir para manter o motor ligado, sem deslocar o veículo.

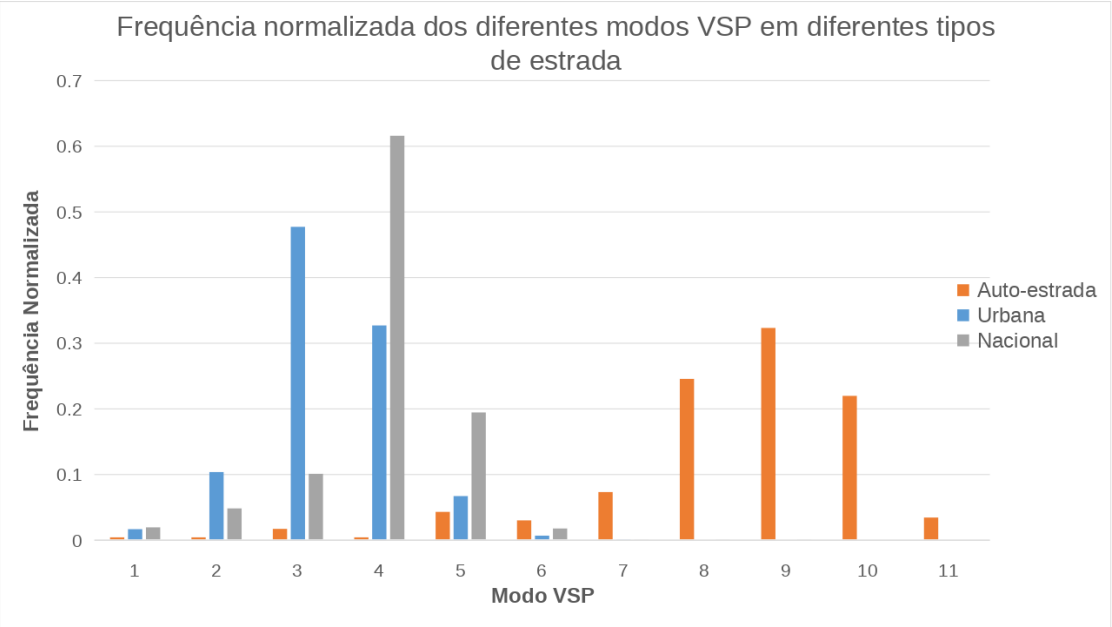

Figura 5.2: Gráfico da frequência normalizada dos diferentes modos VSP em diferentes tipos de estrada

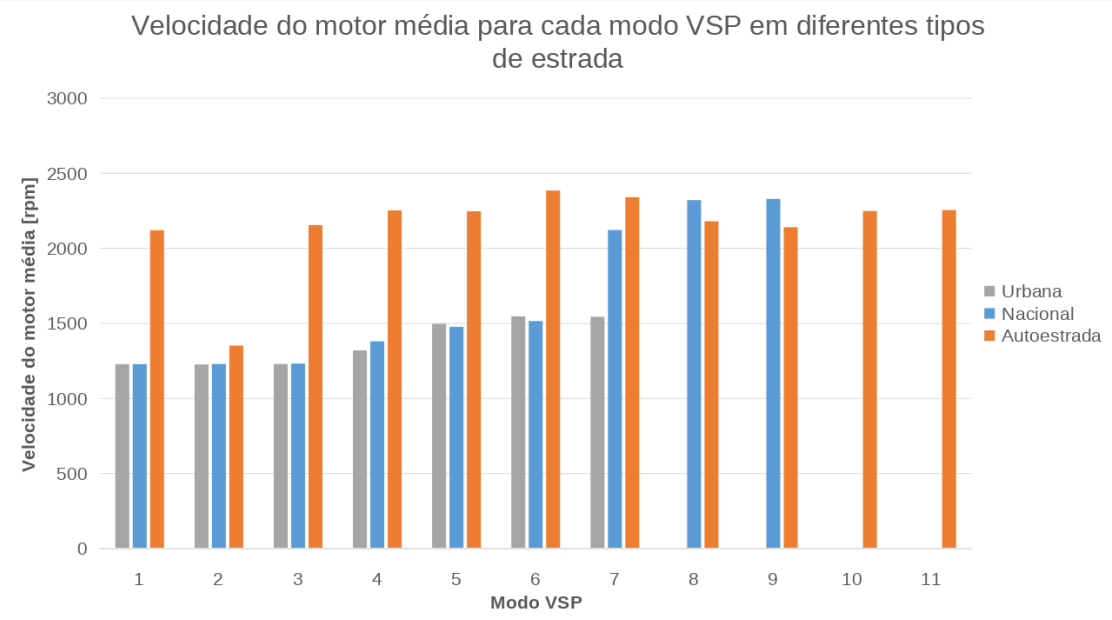

Figura 5.3: Relação entre a velocidade do motor e o modo de VSP

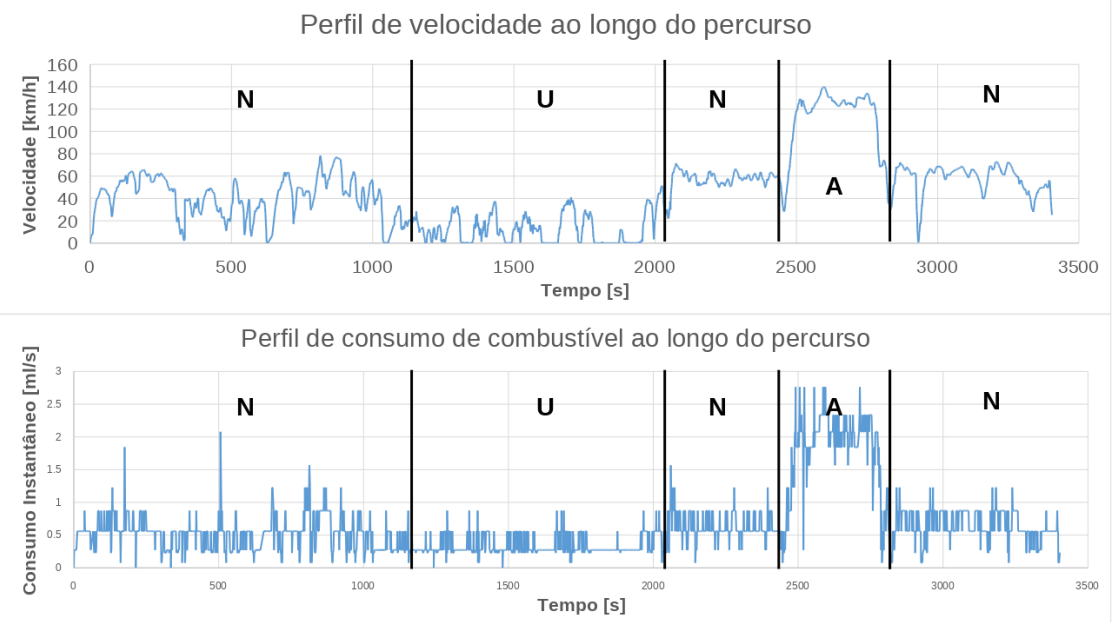

Figura 5.4: Perfil da velocidade e consumo instantâneo ao longo do trajeto, separado pelo tipo de estrada: N - Nacional; U - Urbano; A - Auto-estrada

Tabela 5.3: Emissões e consumo de combustível por 100km para cada tipo de estrada

Emissões e consumo por 100Km			
Emissão	Nacional	Urbano	Auto-estrada
CO_2 [g/100km]	108.25	168.40	154.03
CO [mg/100km]	16.64	28.00	20.62
NO_x [mg/100km]	438.34	716.60	660.52
HC [mg/100km]	11.48	21.31	7.29
Consumo [l/100km]	4.10	6.38	5.83

Capítulo 6

Conclusões

O objetivo principal deste trabalho foi desenvolver e implementar uma plataforma que permita a monitorização simultânea de múltiplos parâmetros em diversos veículos.

Durante a implementação da plataforma encontrei muitas situações que obrigaram a explorar outras ferramentas, como foi o caso do *Wireshark* [35] e *Man in the Middle Proxy* [35] que ajudaram a diagnosticar problemas no formato das mensagens recebidas no servidor. Estas situações ajudaram a compreender melhor o funcionamento dos micro-controladores, do *Docker*, apache e *mariaDB*, e das próprias linguagens de programação usadas.

A implementação da API em PHP sem recurso a uma *framework* própria para o efeito foi uma experiência construtiva, tendo aprendido o funcionamento de uma API, o que é o REST e desenvolvi conhecimento da linguagem PHP, no entanto no futuro recorrerei a *frameworks* como, por exemplo, *Flask* [**?**] para o *Python Web Server*, tanto para reduzir o tempo de desenvolvimento como para produzir uma infraestrutura mais robusta.

Para validar a implementação realizou-se a monitorização de uma viagem como caso de estudo. A plataforma ficou funcional, cumpriu com os objetivos propostos, sendo que o objetivo principal foi alcançado uma vez que foram obtidos dados de diversos tipos de sensores. No entanto alguns aspetos poderão ser explorados posteriormente, como a segurança da API onde os inputs não são filtrados e estão vulneráveis a injeções de SQL.

A plataforma desenvolvida está feita com vista a poder ser distribuída para que possa ser utilizada por outras pessoas no futuro, com uma interface simples e de fácil compreensão.

Poderão vir a ser desenvolvidas e implementadas novas funcionalidades e diferentes sensores. Alguns exemplos a considerar são:

- Implementar limites de valores, mínimos e máximos, que podem ser selecionados pelos utilizadores para cada sensor. Quando o valor monitorizado ultrapassa os limites o utilizador será alertado através de e-mail.
- Implementar no painel do histórico a visualização do percurso gravado no servidor, possibilitando também apresentar um gradiente de cor conforme o valor de algum dos sensores registados no veículo.
- Implementar a leitura de sensores TPMS com um recetor rádio será útil para controlar e prever o desgaste dos pneus em veículos pesados, esta informação é útil na gestão da manutenção de uma frota de veículos.

Bibliografia

[1] FREY, H. Christopher; UNAL, Alper; ROUPHAIL, Nagui M.; COLYAR, James D. - On-Road Measurement of Vehicle Tailpipe Emissions Using a Portable Instrument. **Journal of the Air & Waste Management Association** [Em linha] 53:8 (2003), 992-1002. [Consult. 30 Jan. 2022]. Disponível em WWW: <URL:https://www.tandfonline.com/doi/abs/10.1080/10473289.2003.10466245> ISSN 2162-2906

[2] BANDEIRA, Jorge M.; TAFIDIS, Pavlos; MACEDO, Eloísa; TEIXEIRA, João; BAHMANKHAH, Behnam; GUARNACCIA, Cláudio; COELHO, Margarida C. - Exploring the Potential of Web Based Information of Business Popularity for Supporting Sustainable Traffic Management. **Transport and Telecommunication Journal** [Em linha] 21:1 (2020), 47-60. [Consult. 31 Jan. 2022]. Disponível em WWW:<URL:https://doi.org/10.2478/ttj-2020-0004> ISSN 1407-3179

[3] VAN LY, Mihn; MARTIN, Sujitha; TRIVEDI, Mohan M. - Driver classification and driving style recognition using inertial sensors. **2013 IEEE Intelligent Vehicles Symposium (IV)** [Em linha] (2013) 1040-1045 [Consult. 30 Jan. 2022] Disponível em WWW:<URLhttps://ieeexplore.ieee.org/document/6629603> ISSN 1931-0587

[4] BAHMANKHAH, Behnam; FERNANDES, Paulo; TEIXEIRA, João; COELHO, Margarida C. - Interaction between motor vehicles and bicycles at two-lane roundabouts: a driving volatility-based analysis. **International Journal of Injury Control and Safety Promotion** [Em linha] 26:3 (2019), 205-215. [Consult. 01 Fev. 2022] Disponível em WWW:<URLhttps://ria.ua.pt/handle/10773/26254> ISSN 1745-7300

[10] **CARUSO - From Connected Cars to Connected Business** [Em linha]. [Consult. 02 Fev. 2022]. Disponível em WWW:<URL:https://www.caruso-dataplace.com/>.

[6] AFONSO, Ricardo Filipe - **Tracking and Data Recording System for Vehicles**. Aveiro: Universidade de Aveiro, 2013. 92 p. Dissertação de mestrado.

[7] CARVALHO, Patrick Igreja - **Sistema de monitorização de veículos de mercadorias**. Aveiro: Universidade de Aveiro, 2014. 92 p. Dissertação de mestrado.

[8] ALMISHARI, Salman - **Real Time Vehicle Tracking System and Energy Reduction**. Waterloo: University of Waterloo, 2017. 99 p. Dissertação de mestrado.

[9] NUNES, Mauro; O'NEIL, Henrique - **Fundamental de UML**. 4ª ed. Lisboa: FCA - Editora de Informática, 2004. ISBN 972-722-481-4

[10] **REST - MDN Web Docs Glossary** [Em linha]. [Consult. 19 Jul. 2022]. Disponível em WWW:<URL:https://developer.mozilla.org/en-US/docs/Glossary/REST>.

[11] WEMOS Electronics - **D1** [Em linha]. Arquivo da Internet [Consult. 19 Mar. 2022]. Disponível em WWW:<URL:https://web.archive.org/web/20190712145610/https://wiki.wemos.cc/products:d1:d1>.

[12] WEMOS Electronics - **D1 mini** [Em linha]. Arquivo da Internet [Consult. 19 Mar. 2022]. Disponível em WWW:<URL:https://web.archive.org/web/20170702145741/https://wiki.wemos.cc/products:d1:d1_mini>.

[13] Espressif Systems - **ESP32 Datasheet (v3.8)** [Em linha]. [Consult. 29 Out. 2021]. Disponível em WWW:<URL:https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf>.

[14] RESCORLA, Eric - **The Transport Layer Security (TLS) Protocol Version 1.3** [Em linha]. RFC Editor, 2018. [Consult. 04 Abr. 2022] Disponível em WWW:<URL:https://rfc-editor.org/rfc/rfc8446.txt>. ISSN 2070-1721

[15] Espressif Systems - **ESP8266 Arduino Core: Client Secure** [Em linha]. [Consult. 02 Jan. 2022]. Disponível em WWW:<URL: https://arduino-esp8266.readthedocs.io/en/2.4.0/esp8266wifi/client-secure-examples.html>.

[16] BARKER, Elaine; ROGINSKY, Allen - **Transitioning the Use of Cryptographic Algorithms and Key Lengths** [Em linha]. Gaithersburg, USA: National Institute of Standards and Technology, 2019. [Consult. 22 Abr. 2022] Disponível em WWW:<URLhttps://doi.org/10.6028/NIST.SP.800-131Ar2>

[17] International Organization for Standardization - **Road vehicles — Extended vehicle (ExVe) web services** (ISO 20078). 2021.

[18] **NeoGPS** [Em linha]. [Consult. 17 Dez. 2021]. Disponível em WWW:<URL:https://github.com/SlashDevin/NeoGPS>.

[19] u-blox - **NEO-6 Data Sheet** [Em linha]. [Consult. 29 Out. 2021]. Disponível em WWW:<URL:https://www.u-blox.com/sites/default/files/products/documents/NEO-6_DataSheet_(GPS.G6-HW-09005).pdf>.

[20] MaxBotix Inc. - **LV-MaxSonar-EZ Series** [Em linha]. [Consult. 01 Nov. 2021]. Disponível em WWW:<URL:https://www.maxbotix.com/documents/LV-MaxSonar-EZ_Datasheet.pdf>.

[21] Adafruit Industries - **HC-SR04 Ultrasonic Sonar Distance Sensor** [Em linha]. [Consult. 15 Jan. 2022]. Disponível em WWW:<URL:https://media.digikey.com/pdf/Data%20Sheets/Adafruit%20PDFs/3942_Web.pdf>.

[22] Adafruit Industries - **Distance Measurement with Ultrasound** [Em linha]. [Consult. 15 Jan. 2022]. Disponível em WWW:<URL:https://cdn-learn.adafruit.com/downloads/pdf/distance-measurement-ultrasound-hcsr04.pdf>.

[23] STMicroelectronics - **VL53L0X** [Em linha]. [Consult. 16 Jan. 2022]. Disponível em WWW:<URL:https://www.st.com/resource/en/datasheet/vl53l0x.pdf>.

[24] bauaeletric - **What is Electronic control Unit and what does it do** [Em linha]. [Consult. 22 Jan. 2022]. Disponível em WWW:<URL:https://bauaelectric.com/cars/what-is-electronic-control-unit-and-what-does-it-do/>.

[25] Elm Electronics - **ELM327** [Em linha]. [Consult. 18 Jan. 2022]. Disponível em WWW:<URL:https://www.elmelectronics.com/wp-content/uploads/2020/05/ELM327DSL.pdf>.

[26] **Docker Library Official Image for MariaDB** [Em linha]. [Consult. 27 Mai. 2022]. Disponível em WWW:<URL:https://github.com/MariaDB/mariadb-docker>.

[27] HTTP Server Project - **Apache HTTP Server Documentation** [Em linha]. [Consult. 20 Mai. 2022]. Disponível em WWW:<URL:https://httpd.apache.org/docs/>.

[29] **Really Simple JSON Web Tokens** [Em linha]. [Consult. 25 Mai. 2022]. Disponível em WWW:<URL:https://github.com/RobDWaller/ReallySimpleJWT>.

[29] The PHP Documentation Group - **PHP Manual** [Em linha]. [Consult. 18 Mai. 2022]. Disponível em WWW:<URL:https://www.php.net/manual/en/>.

[30] **Leaflet - a JavaScript library for interactive maps** [Em linha]. [Consult. 14 Jun. 2022]. Disponível em WWW:<URL:https://leafletjs.com/>.

[31] **jscolor: JavaScript Color picker with Opacity** [Em linha]. [Consult. 16 Jun. 2022]. Disponível em WWW:<URL:https://jscolor.com/>.

[32] COELHO, Margarida C.; FREY, H. Christopher; ROUPHAIL, Nagui M.; ZHAI, Haibo; PELKMANS, Luc - Assessing methods for comparing emissions from gasoline and diesel light-duty vehicles based on microscale measurements. **Transportation Research Part D: Transport and Environment** [Em linha] 14:2 (2009), 91-99. [Consult. 25 Jun. 2022]. Disponível em WWW:<URL:https://doi.org/10.1016/j.trd.2008.11.005> ISSN 1361-9209

[33] **GPS Visualizer** [Em linha]. [Consult. 20 Jun. 2022]. Disponível em WWW:<URL:https://www.gpsvisualizer.com/>.

[34] Ecoscore - **How to calculate the CO2 emission from the fuel consumption?** [Em linha]. [Consult. 20 Jul. 2022]. Disponível em WWW:<URL:https://ecoscore.be/en/info/ecoscore/co2>.

[35] **Wireshark** [Em linha]. [Consult. 20 Jul. 2022]. Disponível em WWW:<URL:https://www.wireshark.org/>.

[36] **mitmproxy** [Em linha]. [Consult. 20 Jul. 2022]. Disponível em WWW:<URL:https://mitmproxy.org/>.