Universidade de Brasília – UnB

Faculdade de Ciências e Tecnologias em Engenharia – FCTE

Engenharia de Software

ElmEsp: Desenvolvimento de uma Solução Modular para Diagnóstico Automotivo com ESP32 e ELM327

Autor: Matheus Soares Arruda, Mateus Caltabiano Neves Frauzino

Orientador: Dr. Renato Coral Sampaio

Brasília, DF

2025

Matheus Soares Arruda, Mateus Caltabiano Neves Frauzino

ElmEsp: Desenvolvimento de uma Solução Modular para Diagnóstico Automotivo com ESP32 e ELM327

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB

Faculdade de Ciências e Tecnologias em Engenharia – FCTE

Orientador: Dr. Renato Coral Sampaio

Brasília, DF

2025

Matheus Soares Arruda, Mateus Caltabiano Neves Frauzino

ElmEsp: Desenvolvimento de uma Solução Modular para Diagnóstico Automotivo com ESP32 e ELM327/ Matheus Soares Arruda, Mateus Caltabiano Neves Frauzino. – Brasília, DF, 2025-

73 p. : il. (algumas color.) ; 30 cm.

Orientador: Dr. Renato Coral Sampaio

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade de Ciências e Tecnologias em Engenharia – FCTE , 2025.

1. Manutenção Preditiva. 2. Diagnóstico Veicular. I. Dr. Renato Coral Sampaio. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. ElmEsp: Desenvolvimento de uma Solução Modular para Diagnóstico Automotivo com ESP32 e ELM327

CDU 004.5:629.28

Matheus Soares Arruda, Mateus Caltabiano Neves Frauzino

ElmEsp: Desenvolvimento de uma Solução Modular para Diagnóstico Automotivo com ESP32 e ELM327

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 21 de julho de 2025:

Dr. Renato Coral Sampaio
Orientador

Dr. Evandro Leonardo Silva Teixeira
Convidado 1

Dra. Milene Serrano
Convidado 2

Brasília, DF
2025

Agradecimentos

Gostaríamos de expressar nossa sincera gratidão a todos que estiveram ao nosso lado ao longo desta jornada, durante bons e maus momentos. Agradecemos aos familiares e amigos que nos apoiaram e motivaram, tornando possível a conclusão desta grande trajetória que, finalmente, se aproxima do seu fim. Somos gratos a todos os professores, orientadores e mentores que nos guiaram durante esses anos, sempre compartilhando todo conhecimento para nosso crescimento. Em especial ao Dr. Renato Sampaio que além de ter aceitado a desafiadora tarefa de nos mentorear, acreditou na nossa ideia e sempre buscou nos manter focados e motivados com o projeto.

E por fim, eu Matheus Soares agradeço a este amigo que sempre esteve presente comigo desde o primeiro dia de aula e pelo que tudo parece até o fim de meu bacharelado, obrigado Mateus Caltabiano.

E eu, Mateus Caltabiano, agradeço ao meu companheiro de TCC, Matheus Soares, com quem compartilhei toda essa etapa da minha vida até o momento. Ao longo de toda essa trajetória, você se tornou um irmão para mim, me dando forças nos momentos difíceis e tornando os trabalhos e estudos mais chatos em momentos mais leves e alegres. Agradeço imensamente.

"A vida é uma sucessiva sucessão de
sucessões que se sucedem sucessivamente
sem suceder o sucesso"
(Autor desconhecido)

Resumo

Este trabalho apresenta uma prova de conceito de um sistema integrado para diagnóstico veicular remoto, com foco na automação da leitura e interpretação de códigos de falha (DTCs). A proposta consiste no uso de um microcontrolador ESP32, conectado via Bluetooth ao módulo ELM327, como ponte entre a ECU do veículo e uma aplicação intermediária responsável por registrar os dados em uma base relacional. O sistema visa transformar os dados brutos extraídos da ECU em informações úteis e acionáveis para oficinas mecânicas, com potencial de integração a sistemas de gestão (ERPs) e suporte à decisão técnica. A solução desenvolvida também serve como base para futuras aplicações em manutenção preditiva, enriquecimento de bases para inteligência artificial e desenvolvimento de painéis analíticos. Os testes realizados com veículos reais permitiram validar a viabilidade técnica do fluxo de ponta a ponta, demonstrando a aplicabilidade prática do sistema em ambientes profissionais.

Palavras-chave: Manutenção Preditiva. Diagnóstico Veicular. Esp32. OBD. ECU.

Abstract

This paper presents a proof of concept for an integrated system for remote vehicle diagnostics, with a focus on automating the reading and interpretation of fault codes (DTCs). The proposal consists of using an ESP32 microcontroller, connected via Bluetooth to the ELM327 module, as a bridge between the vehicle's ECU and an intermediate application responsible for recording the data in a relational database. The system aims to transform the raw data extracted from the ECU into useful, actionable information for mechanical workshops, with the potential for integration into management systems (ERPs) and technical decision support. The solution developed also serves as a basis for future applications in predictive maintenance, enrichment of bases for artificial intelligence and development of analytical panels. The tests carried out on real vehicles made it possible to validate the technical feasibility of the end-to-end flow, demonstrating the practical applicability of the system in professional environments.

Key-words: Predictive Maintenance. Vehicle Diagnostics. ESP32. OBD. ECU.

Lista de ilustrações

Figura 1 – ELM327 utilizado para o projeto 40
Figura 2 – *Terminal Serial Bluetooth* 41
Figura 3 – Car Scanner 42
Figura 4 – ESP32 utilizada para o projeto 43
Figura 5 – Veículos usados na pesquisa 45
Figura 6 – Diagrama de Blocos 50
Figura 9 – Diagrama de Camadas 52
Figura 10 – Protótipo da aplicação intermediária 53
Figura 11 – Roadmap para o TCC 2 53
Figura 7 – Diagrama de Hierarquia de Controle 54
Figura 8 – Diagrama de Hierarquia de Controle 55
Figura 12 – ELM327 conectada no carro 57
Figura 13 – ESP32 conectada 58
Figura 14 – Log de envio do mqtt 58
Figura 15 – Log do servidor 59
Figura 16 – Tela inicial 59
Figura 17 – Tela associar veículo 61
Figura 18 – Aguardando resposta da IA 62
Figura 19 – Resposta da IA 63
Figura 20 – Formulário novo veículo 64
Figura 21 – Dashboard 65
Figura 22 – Tela configurações 65

Lista de tabelas

Tabela 1 – Cronograma de desenvolvimento por *sprints* 34
Tabela 2 – User Stories 48
Tabela 3 – Requisitos com Priorização MoSCoW 48
Tabela 4 – Custo total do experimento 66

Lista de abreviaturas e siglas

CAN	*Controller Area Network*
ECU	*Engine Control Unit*
MAC	*Media Access Control Address*
MQTT	*Message Queuing Telemetry Transport*
OBD	*On-Board Diagnostic*
OEM	*Original Equipment Manufacturer*
ORM	*Object-Relational Mapping*
PID	*Parameter Identifier*
SO	Sistema Operacional
SPP	*Serial Port Profile*
XP	*Extreme Programming*

Sumário

1 INTRODUÇÃO 21
1.1 Justificativa 22
1.2 Objetivos 23
1.2.1 Objetivos Gerais 23
1.2.2 Objetivos específicos 23

2 FUNDAMENTAÇÃO TEÓRICA 25
2.1 *Engine Control Unit* (ECU) 25
2.2 *Controller Area Network* (CAN) 26
2.3 *On Board Diagnostics* (OBD) 26
2.3.1 OBD-II 26
2.3.2 *Diagnostic Trouble Codes* (DTCs) 27
2.4 Protocolo Bluetooth 28
2.4.1 *Bluetooth Serial Port Profile* (SPP) 28
2.5 *Media Access Control Address* (Endereço MAC) 28
2.6 FreeRtos 29
2.7 *Message Queuing Telemetry Transport* (MQTT) 29
2.8 *Sistemas de Gerenciamento de Bancos de Dados* (SGBD) 30
2.8.1 *Object-Relational Mapping framework* (ORM) 30

3 METODOLOGIA, MATERIAIS E FERRAMENTAS 31
3.1 Metodologia 31
3.1.1 Tipos de Metodologias 31
3.1.2 Metodologias de Desenvolvimento 32
3.1.3 SCRUM 33
3.1.3.1 Eventos 33
3.1.3.2 Cronograma das *Sprints* 33
3.1.4 Extreme Programming (XP) 38
3.1.5 Heurística da Tentativa e Erro 39
3.2 Ferramentas 39
3.2.1 ELM327 39
3.2.2 *Serial Bluetooth Terminal* 40
3.2.3 *Car Scanner* 41
3.2.4 ESP32-WROOM-32 42
3.2.5 *Espressif IoT Development Framework* (ESP-IDF) 43
3.2.6 Arduino IDE 44

3.2.7 ELMduino . . . 44
3.2.8 Next.js . . . 44
3.2.9 Prisma . . . 44
3.2.10 Tailwindcss e Shadcn . . . 45
3.2.11 Sandero Stepway e Onix LT . . . 45

4 PROPOSTA DE SOLUÇÃO . . . 47
4.1 Requisitos do Sistema . . . 47
4.2 Arquitetura . . . 49
4.2.1 Diagrama de Blocos . . . 50
4.2.2 Diagrama de Hierarquia de Controle . . . 51
4.2.3 Diagrama de Camadas . . . 51
4.2.4 Protótipo . . . 52
4.2.5 *Roadmap* . . . 53

5 RESULTADOS FINAIS . . . 57
5.1 Funcionamento do ELM327 e ESP32 em ambiente controlado . . . 57
5.1.1 ELM327 . . . 57
5.1.2 ESP32 . . . 57
5.2 Funcionamento da aplicação web . . . 59
5.2.1 Tela inicial . . . 59
5.2.2 Tabela de Veículos Cadastrados . . . 59
5.2.3 Tabela de Códigos de Diagnóstico de Falha (DTCs) . . . 60
5.2.4 Tela associar veículo . . . 60
5.2.5 Tela mais informações . . . 61
5.2.6 Tela formulário novo veículo . . . 63
5.3 *Dashboard* de Análise e Estatísticas . . . 64
5.4 Tela de Configurações: Importação de Dados de Veículos . . . 65
5.5 Custo Total . . . 66
5.6 Possíveis Melhorias . . . 66
5.6.1 Otimização do processo de *discovery Bluetooth* . . . 66
5.6.2 Sistema de redundância para escolha de protocolo OBD-II . . . 66
5.6.3 Melhoria da infraestrutura e migração para a nuvem . . . 66
5.6.4 Integração com bases oficiais de dados veiculares . . . 67
5.6.5 Separação de dados para diferentes oficinas . . . 67
5.6.6 Salvar as respostas da inteligência artificial . . . 67

6 CONCLUSÃO . . . 69

REFERÊNCIAS . . . 71

1 Introdução

De acordo com Cromer et al. (2024), o automóvel é um veículo cuja principal função é o transporte de passageiros e é comumente propulsionado por um sistema à combustão, elétrico ou híbrido.

Para o correto funcionamento do sistema, a ECU (*Engine Control Unit* ou Unidade de Controle do Motor) realiza cálculos contínuos para buscar a otimização da injeção e ignição com a maior eficiência do combustível ao mesmo tempo que busca reduzir a emissão de poluentes e maximizar o torque e potência final, utilizando dos dados provenientes dos diversos sensores espalhados pelo carro para fazer ajustes e atingir o seu objetivo (CUATTO et al., 1998).

Esses dados, essenciais para otimização do motor, estão disponíveis externamente por meio da interface OBD-II, cuja integração com microcontroladores exige soluções especializadas devido à complexidade de comunicação e requisitos de tempo real. Desta forma, a integração de sistemas de diagnóstico veicular com microcontroladores, tem ganhado relevância em aplicações automotivas inteligentes, desde monitoramento remoto até soluções IoT embarcadas.

A evolução contínua dos sistemas eletrônicos veiculares exige soluções integradas de diagnóstico capazes de associar códigos de falhas (DTCs) a ações corretivas específicas. Embora a interface OBD-II permita acesso a esses dados, a aplicação prática de códigos de falhas em oficinas mecânicas ainda enfrenta barreiras de acessibilidade. A dificuldade não está na interpretação técnica dos códigos, mas na transformação desses dados em informações úteis e acionáveis para os profissionais de reparação.

Este trabalho investiga soluções técnicas para facilitar o acesso ao diagnóstico veicular, desenvolvendo uma ponte eficiente entre os sistemas padronizados de leitura e as necessidades reais das oficinas, com objetivo de tornar os dados da ECU tão acessíveis quanto decisivos para o processo de manutenção.

A arquitetura da solução desenvolvida foi feita de forma que a comunicação com a ECU do veículo é estabelecida através do dispositivo ELM327. Os dados de diagnóstico, especificamente os Códigos de Falha de Diagnóstico (DTCs), são então solicitados e processados por um microcontrolador ESP32. Este, por sua vez, transmite as informações para um banco de dados por meio de uma aplicação intermediária, que recebe as mensagens em formato JSON via protocolo MQTT.

Adicionalmente, para viabilizar a integração e o consumo desses dados, foi desenvolvida uma API, permitindo a busca programática dos DTCs cadastrados. Para de-

monstrar a aplicabilidade prática da solução em ambientes de oficinas, foi construído um frontend que apresenta os dados cadastrados dos veículos e DTCs extraídos, além uma interface baseada em resposta de IA, capaz de identificar, analisar e sugerir soluções para os códigos de falha. Essa funcionalidade avançada leva em consideração a marca, modelo e ano do veículo, oferecendo um primeiro contato com o real problema e as possíveis soluções, deixando claro que a aprovação de um profissional é recomendada pela resposta se basear apenas na IA.

1.1 Justificativa

A crescente complexidade dos automóveis, impulsionada pelo desenvolvimento e integração de tecnologias avançadas, faz com que os automóveis sejam grandes unidades de tratamento de dados oriundos de diversos sensores, e o advento da conexão destes veículos faz com que esses dados sejam transmitidos em grande escala.

No entanto, para Pillmann, Zarcula e Alonso (2017) o acesso as informações dos automóveis está cada vez mais limitada, devido à natureza proprietária dos dados. A escassez de detalhes claros para o usuário final dificulta a realização de manutenções corretivas e preventivas, além de restringir o acesso a informações fundamentais para o monitoramento e desempenho eficiente do veículo.

A indústria automotiva está transformando sua percepção sobre os automóveis, que deixam de ser vistos como produtos isolados e passam a integrar um ecossistema centrado em carros conectados, permitindo então um modelo de negócio baseado em dados. O grande volume de informações disponíveis nos automovéis podem ser utilizados para melhorar a experiência do usuário e criar oportunidades de negócios (STERK et al., 2024). Deste princípio surge, então, a necessidade de um sistema capaz de processar, analisar e transformar esses dados em valor para usuários e empresas.

Atualmente, o mercado automotivo é amplamente atendido por módulos ELM327, que permitem a comunicação com a interface OBD-II, oferencendo aos usuários uma alternativa acessível para obtenção de informações da ECU do veículo. No entanto, ainda há uma lacuna significativa quando se trata de soluções que viabilizem não apenas a leitura desses dados, mas também sua integração com arquiteturas modernas de software. A ausência de fluxos abertos e bem definidos que conectem a leitura dos DTCs à persistência estruturada de bancos de dados limita o desenvolvimento de aplicações mais inteligentes e interconectadas, como sistemas de ERP para oficinas, plataformas móveis e modelos preditivos baseados em IA. Este trabalho justifica-se, portanto, pela necessidade de validar uma arquitetura que una dispositivos embarcados de baixo custo com protocolos modernos de comunicação e armazenamento, abrindo caminho para um ecossistema mais acessível, interoperável e orientado a dados no setor automotivo.

1.2 Objetivos

1.2.1 Objetivos Gerais

Desta forma, o objetivo geral deste trabalho é desenvolver e validar um sistema integrado de diagnóstico veicular que auxilie oficinas mecânicas na interpretação e aplicação prática de códigos de falhas (DTCs). A solução utiliza um microcontrolador ESP32 e um módulo ELM327 para extração padronizada de dados via OBD-II, com transmissão MQTT de dados serializados em JSON que são persistidos em um banco de dados relacional, criando uma base técnica para:

- Associação automática entre DTCs e procedimento de reparo sugerido por uma Inteligência Artificial;
- Integração com sistemas de gestão de oficinas (ERPs automotivos);
- Suporte à decisão técnica com histórico de casos similares.

O sistema visa otimizar o fluxo de trabalho em oficinas, transformando dados brutos da ECU em informação acionável para mecânicos e gestores.

1.2.2 Objetivos específicos

A validação de um sistema capaz de se comunicar e transmitir informações envolve alguns objetivos específicos que devem ser claros, mensuráveis e alinhados ao contexto geral, sendo eles:

- Implementar um algoritmo em linguagem C no ESP32 para comunicação com o módulo ELM327 via *Bluetooth*, realizando a leitura dos códigos de falha (DTCs) da ECU por meio do protocolo OBD-II;
- Estruturar e formatar os dados coletados em JSON, serializando-os para transmissão via protocolo MQTT;
- Criar uma base de conhecimento de carros e seus respectivos problemas mais comuns;
- Desenvolver um aplicativo intermediário responsável por consumir os dados MQTT e armazená-los em um banco de dados relacional;
- O Aplicativo intermediário deve utilizar de inteligência artificial para sugerir, com base em dados históricos e padrões reconhecidos, as soluções mais comuns e eficazes para cada DTC específico;

- Desenvolver *dashboard* com dados mensuráveis que gerem informação útil para gestores e mecânicos;
- Validar o funcionamento do sistema completo por meio de testes com veículos reais, verificando a integridade e consistência dos dados ao longo da pipeline;
- Demonstrar a viabilidade da arquitetura proposta como base para soluções futuras voltadas à manutenção preditiva, integração com ERPs automotivos e criação de bases de dados para modelos de inteligência artificial.

2 Fundamentação Teórica

A fundamentação teórica é a base de conhecimento para o trabalho, é a coletânea de definições e menções importantes sobre os principais aspectos do projeto, citados a partir de alguns autores. Nesse contexto, as referências bibliográficas de autores especializados são fundamentais para fornecer o suporte teórico necessário à análise e à proposta apresentada.

Neste trabalho, são exploradas de forma aprofundada as tecnologias e soluções disponíveis para a captação de dados de veículos. O foco é em tecnologias como a interface OBD-II, que permite o acesso às informações do veículo, e o protocolo CAN, responsável pela comunicação interna entre os diversos módulos eletrônicos do automóvel. Essas tecnologias operam em conjunto com a *Engine Control Unit* (ECU), que desempenha um papel fundamental neste projeto ao disponibilizar os dados e DTCs necessários para a coleta e análise.

A comunicação entre o microcontrolador ESP32 e o módulo ELM327 será estabelecida através do protocolo *Bluetooth*, utilizando o perfil SPP (*Serial Port Profile*). Nesta seção, será abordado também o conceito de endereço MAC. Complementarmente, serão apresentados os fundamentos do protocolo MQTT, empregado para o envio dos dados em formato JSON a uma aplicação intermediária conectada a um banco de dados.

Por fim, a conclusão será uma breve explicação sobre o sistema operacional que irá orquestrar todo o *firmware* a ser desenvolvido, garantindo a execução eficiente das tarefas e o gerenciamento adequado dos recursos do microcontrolador.

2.1 *Engine Control Unit* (ECU)

É um controle eletrônico do motor que está presente em todos os carros comercializados. Este computador recebe dados de diversos sensores espalhados pela mecânica do veículo para realizar seu trabalho com proficiência. É responsável pelo controle da mistura de combustível, ignição, funcionamento desengatado, controle do ABS entre outros. Para carros mais tecnológicos, também existe, por exemplo, o piloto automático e o controle da transmissão automática (GEORGE; PECHT, 2013).

A programação das ECUs é individual para cada montadora e cada carro, mas em sua maioria possuem objetivos em comum, como a busca por uma maior eficiência de combustível, redução da emissão de poluentes e maximizar o torque e a potência. Embora esses requisitos sejam um tanto quanto contraditórios, o algoritmo deve buscar o melhor para atender as necessidades dentro dos limites estabelecidos (CUATTO et al., 1998).

Neste projeto, a ECU atua como ponto de origem dos dados veiculares, fornecendo informações por meio da interface OBD-II. Esses dados, mais especificamente os códigos de falhas (DTCs), serão extraídos, transformados e carregados fornecendo o necessário para alimentar a base de dados.

2.2 *Controller Area Network* (CAN)

O barramento CAN é um sistema de mensagens por transmissão que é compartilhado por toda a rede enviando várias mensagens pequenas. Foi criado pela Bosch dentro das normas ISO e originalmente desenvolvido para a indústria automotiva para substituir o complexo chicote de fiação por um sistema de barramento de apenas dois fios (CORRIGAN, 2002).

A especificação exige uma velocidade de até 1Mbps, alto isolamento contra interferência elétrica e capacidade de auto-diagnóstico e correções de erros de dados. Essas características levaram o tipo de comunicação a extrapolar a área automotiva e atuar também em áreas como indústrias medicas, aeroespaciais e marinhas (CORRIGAN, 2002).

2.3 *On Board Diagnostics* (OBD)

Se refere ao sistema do automóvel que providencia auto-diagnóstico e capacidade de resposta, dando acesso às informações dos subsistemas do veículo com intuito de monitoração e solução de problemas (GEOTAB, 2023).

2.3.1 OBD-II

Os padrões OBD-II foram inicialmente desenvolvidos por uma parceria entre CARB (*California Air Resources Board*), SAE (*Society of Automotive Engineers*), ISO (*International Organization for Standardization*) e EPA (*Environmental Protection Agency*).

Seus objetivos principais são de aumentar a economia de combustível assegurando condições ótimas de funcionamento do motor, reduzir emissões, reduzir o tempo entre uma falha no sistema e o aviso para o usuário por meio de constante monitoramento e auxiliar no diagnóstico e reparo dos equipamentos de emissão (MCCORD, 2011).

Além das capacidades básicas que seu predecessor possuía, o OBD-II é capaz de:

- Mostrar dados provenientes do sistema em tempo real.
- Armazenar os DTCs (*Diagnostic Trouble Codes*) em memória não volátil
- Armazenar quaisquer DTCs que ainda não ativaram a MIL (*Malfunction Indicator Lamp*).

- Mostrar dados capturados no exato momento em que algum DTC foi acionado.
- Capacidade de limpar quaisquer DTCs ativados por uma ferramenta de scanner.
- Listar informações sobre o veículo, como VIN (*Vehicle Identification Number*), modelo, motor, entre outros.

Em conjunto com o módulo ELM327, torna-se viável a extração estruturada dos dados da ECU via protocolo OBD-II, permitindo o desenvolvimento de um algoritmo em linguagem C na ESP32 com esse propósito. Assim, o OBD-II deixa de ser apenas um interface técnica e passa a representar a base para transformar dados veiculares em informação útil e acionável para manutenção automotiva inteligente.

2.3.2 *Diagnostic Trouble Codes* (DTCs)

Os Códigos de Falha de Diagnóstico, conhecidos pela sigla DTC(*Diagnostic Trouble Codes*), são códigos utilizados pelos módulos eletrônicos dos veículos para indicar falhas ou irregularidades detectadas nos sistemas monitorados. Padronizados por meio da especificação SAE J2012, permitem que o sistema de diagnóstico aborde não apenas falhas que afetam o funcionamento do motor, mas também os comportamentos que indicam falhas futuras (MCCORD, 2011). Essa normalização dos códigos possibilita a detecção precoce de problemas, melhorando a eficiência da manutenção e reduzindo custos futuros com reparos.

A padronização SAE J2012 possui um formato específico e de fácil interpretação para os DTCs, compostos por cinco caracteres que seguem uma estrutura com significado distinto para cada posição (MCCORD, 2011).

- Campo 1: O primeiro caractere indica a categoria do sistema onde a falha ocorreu. Sendo:

  P: *Powertrain* (Motor, transmissão, etc.);

  B: *Body* (Itens de segurança - Airbags, cintos, sensores, etc.);

  C: *Chassis* (Direção, Freio ABS, Controle de tração, etc.);

  U: *Network* (Comunicação entre módulos eletrônicos).

- Campo 2: O segundo caractere indica o padrão:

  0: Padrão genérico, conforme a norma SAE J2012;

  1: Específico do fabricante *Original Equipment Manufacturer* (OEM)

- Campo 3: O terceiro caractere aponta o subsistema específico da falha, variando de 0 a 9;

- Campos 4 e 5: Os dois últimos caracteres (de 00 a 99) indicam o código de erro em si, ou seja, a falha identificada (MCCORD, 2011).

2.4 Protocolo Bluetooth

O *Bluetooth* é um protocolo de comunicação baseado em um padrão aberto IEEE, responsável pela implementação de redes pessoais sem fio. Suas principais características são o curto alcance e o baixo consumo de energia. Opera globalmente em uma faixa de rádio de curto alcance de 2.4GHz, com distâncias entre 10 metros até 100 metros, tornando-se uma alternativa aos cabos de dispositivos locais como teclados, mouses, impressoras e etc. Também pode ser utilizado para comunicação entre dispositivos distintos, atuando como uma ponte entre conexões ou como nós em redes (SOFI, 2016).

A operação básica do protocolo baseia-se na comunicação entre dispositivos controladores e controlados. Quando um dispositivo *Bluetooth* é iniciado, ele tenta operar como um controlado de um aparelho controlador já em execução. Em seguida, o dispositivo começa a captar a solicitação do controlador e responde a essa requisição. O *Bluetooth* define vários tipos de conexão, cada uma com diferentes combinações de largura de banda, proteção contra erros e qualidade de serviço. Desta forma, o protocolo não se limita a definir uma interface de rádio, mas também possibilita que os dispositivos se localizem e divulguem os serviços que disponibilizam (SOFI, 2016).

Para os testes, o dispositivo ELM327 adquirido possui função *bluetooth* para propagação de dados e, visando a exploração da mesma, com o uso do SPP, para emular uma conexão serial com fio entre o ELM327 e o microcontrolador ESP32.

2.4.1 *Bluetooth Serial Port Profile* (SPP)

O SPP (*Serial Port Profile*) é um perfil do protocolo *Bluetooth* que define os requisitos necessários para emular uma conexão serial com fio, utilizando o protocolo RFCOMM entre os dispositivos envolvidos. Estes requisitos podem ser descritos como serviços fornecidos às aplicações envolvidas, além da definição das funcionalidade e processos necessários para a comunicação entre os dispositivos. (KIOURTIS; MAVROGIORGOU; KYRIAZIS, 2021)

2.5 *Media Access Control Address* (Endereço MAC)

Um endereço MAC é uma sequência de 12 dígitos hexadecimais, geralmente, porém não obrigatoriamente, agrupados em pares e separados por dois pontos ou hifens. Como o próprio nome diz, é o endereço físico da máquina, diferente do endereço IP, que é o virtual. Os endereços MAC são utilizados na LAN, para "pulos"(*hops*) entre dispositivos próximos

por meio da *Link Layer*, enquanto os endereços IP são o destino final das mensagens e utilizados na *Network Layer* (OFFICES, 2024).

Os primeiros 6 dígitos identificam as montadoras e são chamados de OUI (*Organizational Unique Identifier*). Os últimos 6 dígitos são designados pela montadora ao dispositivo e identificam unicamente cada um deles (GEEKS, 2024).

O MAC Address é utilizado para garantir que o microcontrolador se conecte com o dispositivo desejado que, no caso, é o ELM327 conectado à porta OBD-II do veículo. Dessa forma, evita-se a conexão indesejada com outros possíveis dispositivos com função *bluetooth* ativada e buscando emparelhamentos pela proximidade.

2.6 FreeRtos

O *FreeRTOS* é um sistema operacional em tempo real projetado para dispositivos embarcados, conhecido por seu modelo preemptivo e por seu algoritmo de escalonamento dinâmico baseado em prioridades. Neste SO, a comunicação entre processos se estabelece por meio de filas de mensagens e semáforos binários. Todos os processos bloqueados expiram, uma técnica implementada no *FreeRtos* que visa evitar *Deadlocks*. Esse mecanismo exige que os desenvolvedores configurem e ajustem os tempos de expiração, além de lidarem com falhas de alocação de recursos (ZHU, 2016).

O *FreeRtos* é praticamente todo escrito em C, com apenas algumas funções em *Assembly*, o que torna o código legível, fácil de manter e portável. Os objetivos claros desse sistema é a simplicidade, portabilidade e concisão. O Núcleo do RTOS utiliza múltiplas listas de prioridades, ao contrário de núcleos baseados em *bitmap*, que proporcionam flexibilidade alta no design da aplicação (ZHU, 2016).

Com integração nativa com a ESP-IDF, não será necessário nenhum tipo de adaptação externa, proporcionando uma maior facilidade para a utilização direta de sua capacidade, especialmente a divisão em *threads* e gerenciamento de recursos.

2.7 *Message Queuing Telemetry Transport* (MQTT)

O *Message Queuing Telemetry Transport* (MQTT) é um protocolo de mensagens leve, baseados em publicação/assinatura (*Publish/Subscribe*), amplamente utilizado em aplicações de Internet das Coisas (IoT) devido à sua eficiência em redes com largura de banda limitada ou alta latência. Segundo Banks e Gupta (2014), o MQTT opera sobre protocolos de transporte confiáveis, como o TCP/IP, garantindo a entrega ordenada e sem perdas de mensagens entre clientes e servidores (também chamados de *brokers*).

No modelo MQTT, os dispositivos atuam como clientes, capazes de publicar men-

sagens em tópicos específicos ou assinar tópicos dos quais desejam receber atualizações. Essas mensagens são associadas a um tópico e a um nível de qualidade de serviço (QoS), que determina o grau de confiabilidade na entrega. O *broker* centraliza a comunicação, recebendo as mensagens publicadas e encaminhando-as somente aos clientes que possuem uma assinatura compatível com o tópico (BANKS; GUPTA, 2014).

A leveza do protocolo e a simplicidade da arquitetura o tornam ideal para o microcontroladores, como o ESP32 utilizado neste trabalho, permitindo o envio de dados (como códigos DTC veiculares) de forma assíncrona e eficiente para outras aplicações conectadas.

2.8 *Sistemas de Gerenciamento de Bancos de Dados* (SGBD)

De acordo com Elmasri e Navathe (2018), um banco de dados é o armazenamento de fatos que podem ser registrados e relacionados entre si, além de um significado implícito. Eles tem "uma fonte da qual os dados são derivados, algum grau de intervenção com os eventos do mundo real e uma audiência que está ativamente interessada em seu conteúdo"(ELMASRI, 2018, p.4).

Agora, um Sistema de Gerenciamento de Banco de Dados é uma aplicação que facilita o gerenciamento e mantimento do banco de dados por parte do usuário. "É um sistema de *software* de uso geral que facilita o processo de definição, construção, manipulação e compartilhamento de bancos de dados entre diversos usuários e aplicações"(ELMASRI, 2018, p.5)

No contexto deste projeto, tanto o banco de dados quanto o sistema de gerenciamento do mesmo foram desenvolvidos e mantidos por meio de uma API criada para esta funcionalidade.

2.8.1 *Object-Relational Mapping framework* (ORM)

Segundo Sivakumar et al (2021), os *frameworks* ORM servem como uma ponte entre o modelo temporário de objetos das linguagens de programação e o modelo persistente relacional dos bancos de dados, permitindo que desenvolvedores manipulem dados sem escrever diretamente comandos SQL.

ORMs abstraem operações por meio de uma interface, possibilitando que desenvolvedores interajam com o banco de dados utilizando apenas as estruturas da linguagem de programação, como objetos e classes. Isso resulta em um código enxuto, manutenção facilitada e maior produtividade no desenvolvimento de aplicações. Dessa forma, a utilização de um *framework* ORM, como o Prisma, viabiliza o desenvolvimento mais ágil do aplicativo intermediário que será responsável por validar o fluxo e persistir os dados.

3 Metodologia, Materiais e Ferramentas

De acordo com Oliveira (2017), "A Metodologia é a explicação minuciosa, detalhada, rigorosa e exata de toda ação desenvolvida no método (caminho) do trabalho de pesquisa.". Partindo dessa definição, neste tópico serão explorados os tipos de metodologia disponíveis, descrevendo aquela que melhor se adequa ao projeto em questão, bem como as metodologias de desenvolvimento e ferramentas utilizadas.

A equipe optou pela escolha de uma metodologia de pesquisa aplicada com foco no desenvolvimento tecnológico, visando a validação de um sistema embarcado capaz de extrair, transmitir e armazenar dados veiculares. Além disso, algumas metodologias de desenvolvimento ágil foram adotadas, como SCRUM e XP(*Extreme Programming*), que possibilitaram um ciclo de desenvolvimento dinâmico e iterativo, com forte adaptabilidade às necessidades encontradas durante a pesquisa e implementação. Auxiliarmente, a heurística de tentativa e erro foi utilizada em momentos críticos, como na integração da ELM327 com a ESP32, permitindo a dissolução de problemas complexos por meio de ciclos de experimentos e ajustes.

Por fim, as ferramentas utilizadas ao longo do projeto incluem ELM327, *Serial Bluetooth Terminal*, *Car Scanner*, Esp32, ESP-IDF, ELMduino, Prisma ORM, Next.JS, *Tailwind CSS*, ShadCn, Sandero Stepway 2016 e Onix LTZ 2019 que suportaram todo o processo de desenvolvimento, teste e validação dos resultados. A conjunção dessas ferramentas e metodologias propiciou um ambiente de trabalho eficiente e organizado, garantindo bons resultados esperados.

3.1 Metodologia

Pesquisa científica, de acordo com Ceyda Özhan e Aslı Dönmez, é a pesquisa realizada com o propósito de contribuir com a ciência por meio de coleta e análise de dados realizada de maneira sistemática. Antes de iniciar a pesquisa científica, o objetivo do pesquisador é estabelecer o assunto, planejar e especificar a metodologia que será utilizada. A metodologia científica refere-se então ao conjunto de métodos e técnicas que orientam o pesquisador na condução do estudo, garantindo que os resultados sejam válidos, confiáveis e replicáveis (ÇAPARLAR; DöNMEZ, 2016).

3.1.1 Tipos de Metodologias

De acordo com Ceyda Özhan e Aslı Dönmez, a classificação da pesquisa pode ser realizada de diversas maneiras, sendo feita de acordo com as técnicas de coleta de dados,

na relação com tempo e no meio pelo qual elas são aplicadas (ÇAPARLAR; DöNMEZ, 2016). Já para Fontelles, a pesquisa ganha mais formas de ser classificada, sendo destrinchada em seis pontos, sendo eles: Finalidade, Natureza, Forma de abordagem, Objetivos, Procedimentos Técnicos, Desenvolvimento no tempo (FONTELLES et al., 2009).

Dentre essas classificações, Fontelles et al. (2009) destacam os seguintes tipos de pesquisa:

- **Pesquisa Exploratória**: Onde há uma primeira aproximação do tema com o pesquisador.
- **Pesquisa Descritiva**: Foca na observação, registro e descrição de eventos, fenômenos ou amostra de população.
- **Pesquisa Explicativa**: Busca entender as causas, relações e explicar os fatores determinantes.
- **Pesquisa Aplicada**: Visa resolver problemas práticos por meio de soluções tecnológicas ou inovações.
- **Pesquisa Experimental**: Envolve a manipulação de variáveis e a realização de testes controlados para estabelecer relações de causa e efeito.

No contexto deste projeto, optou-se então pela escolha da pesquisa aplicada com foco e desenvolvimento tecnológico, considerando que o objetivo principal é validar um sistema embarcado que não apenas extrai códigos de falha veicular (DTCs), mas os transforma em informação técnica acionável, integrando-se diretamente ao fluxo de trabalho de oficinas mecânicas. Além disso, a estrutura proposta serve como base para futuros sistemas de análise preditiva, onde o histórico de DTCs e soluções associadas alimentará modelos de manutenção preventiva.

Como citado anteriormente, este tipo de pesquisa baseia-se na finalidade de produção de conhecimento científico para solução de problemas específicos da vida moderna, gerando novos processos tecnológicos e produtos, com resultados na melhoria da qualidade de vida (FONTELLES et al., 2009).

A metodologia incluirá traços de uma pesquisa experimental, que serão aplicados para validar todo o fluxo de dados. Os testes envolverão a análise da extração de códigos de falha (DTCs), sua transmissão através do protocolo MQTT e posterior armazenamento em aplicação própria.

3.1.2 Metodologias de Desenvolvimento

Para a elaboração e validação do sistema proposto será adotado um processo de desenvolvimento iterativo. Cada ciclo buscará o aprimoramento das funcionalidades es-

pecíficas, com objetivo de assegurar o correto funcionamento do sistema em diferentes cenários de uso, aplicando tratamento de erros quando necessário para garantir robustez e confiabilidade da solução.

3.1.3 SCRUM

O Scrum é uma metodologia ágil que foi criada em meados dos anos 90 por Jeff Sutherland e Ken Schwaber. Baseado no empirismo e *Lean Thinking* (ou "pensamento enxuto"), assegura que o conhecimento venha da experiência e decisões sejam tomadas com base no que é observado, possibilitando o foco no essencial e evitando o desperdício de tempo (SCHWABER; SUTHERLAND, 2020).

Foi utilizada parte da metodologia ágil Scrum para realizar o desenvolvimento iterativo com definição de valores e especialmente a divisão de tarefas em *sprints*.

3.1.3.1 Eventos

De acordo com Schwaber e Sutherland (2020), a sprint é "a batida de coração do Scrum, onde ideias se tornam valor". As *sprints* costumam ter duração de 1 a 2 semanas e é onde todas as tarefas a serem realizadas para o desenvolvimento são distribuídas, permitindo uma melhor organização e definição de escopo para o desenvolvimento da aplicação.

A *Sprint Planning* marca o início da *sprint*, sendo o momento em que todos os integrantes da equipe Scrum se reúnem para definir, de forma colaborativa, o objetivo da *sprint* que está prestes a começar (SCHWABER; SUTHERLAND, 2020).

O *Sprint Review* tem como objetivo avaliar os resultados obtidos durante o *Sprint* e definir possíveis adaptações futuras. Nesse evento, o *Scrum Team* apresenta o trabalho realizado promovendo uma discussão sobre o progresso em direção ao objetivo do produto. Durante a reunião, tanto o time quanto os stakeholders analisam o que foi alcançado no *Sprint* e consideram mudanças no ambiente (SCHWABER; SUTHERLAND, 2020).

A equipe optou por adotar *sprints* semanais, com *Planning* e *Review* programadas para as Sextas-Feiras dos meses. A seguir está disposto o cronograma de desenvolvimento da equipe durante todo processo de pesquisa:

3.1.3.2 Cronograma das *Sprints*

Tabela 1 – Cronograma de desenvolvimento por *sprints*

Sprint	**Data início**	**Data fim**	**Descrição**
Sprint 1	18/10/2024	25/10/2024	Ideias de projeto, definição de objetivo
Sprint Review/Planning	25/10/2024	25/10/2024	Encerramento sprint 1, reunião com coordenador, decisões sobre dispositivo ELM e como prosseguir no desenvolvimento
Sprint 2	25/10/2024	01/11/2024	Pesquisa e compra do ELM, tentativa de conexão do ELM com a ESP32 via Espressif
Sprint Review/Planning	01/11/2024	01/11/2024	Encerramento sprint 2, decisões sobre prosseguir no ambiente Espressif ou optar outras alternativas
Sprint 3	01/11/2024	08/11/2024	Tentativa de conexão via Arduino IDE ao ELM327 utilizando as bibliotecas ELMDuino e bluetoothSerial
Sprint Review/Planning	08/11/2024	08/11/2024	Encerramento sprint 3, discussões sobre ambiente de desenvolvimento Arduino e biblioteca ELMduino
Sprint 4	08/11/2024	15/11/2024	Estudo de caso da Automatic Labs, persistência nas tentativas de conexão com módulo ELM
Sprint Review/Planning	15/11/2024	15/11/2024	Encerramento sprint 4, discussões sobre como prosseguir com módulo ELM no veículo Sandero
Sprint 5	15/11/2024	22/11/2024	Sucesso na conexão com módulo ELM via ESP32, chamadas de firmwares sendo respondidas pelo módulo

Continua na próxima página

Sprint	Data início	Data fim	Descrição
Sprint Review/Planning	22/11/2024	22/11/2024	Encerramento sprint 5, alinhamento com coordenador e conversas sobre como prosseguir para externalizar dados da ECU do veículo Sandero
Sprint 6	22/11/2024	29/11/2024	Estudo da biblioteca e testes bem sucedidos com a ECU do Renault Sandero (dados externalizados), Levantamento de requisitos
Sprint Review/Planning	29/11/2024	29/11/2024	Encerramento sprint 6, alinhamento sobre requisitos e conversas sobre futuro objetivo do projeto
Sprint 7	29/11/2024	06/12/2024	Testes mal sucedidos no Chevrolet Onix com uso da ELMduino e Prototipação de telas no Figma
Sprint Review/Planning	06/12/2024	06/12/2024	Encerramento sprint 7, discussões sobre interface OBD-II, protocolos de comunicação
Sprint 8	06/12/2024	13/12/2024	Testes bem sucedidos no Chevrolet Onix, Descoberta e estudo de diferentes protocolos de comunicação
Sprint Review/Planning	13/12/2024	13/12/2024	Encerramento sprint 8, alinhamento com coordenador, discussões sobre mudança de foco, troca para ambiente em C Espressif
Sprint 9	13/12/2024	20/12/2024	Testes de PID em Chevrolet Onix, início de testes em ambiente espressif e início da documentação

Continua na próxima página

Sprint	Data início	Data fim	Descrição
Sprint Review/Planning	20/12/2024	20/12/2024	Encerramento sprint 9, alinhamento presencial com coordenador conversas sobre futuro do projeto e fim do protótipo.
Sprint 10	20/12/2024	27/12/2024	Recesso, Progresso na documentação
Sprint 11	27/12/2024	03/01/2025	Recesso, Progresso na documentação
Sprint Review/Planning	03/01/2024	03/01/2024	Encerramento sprint 10 e 11, retorno aos trabalhos e alinhamentos
Sprint 12	03/01/2025	10/01/2025	Alteração de escopo do projeto, foco desviado apenas para a síntese da biblioteca ElmEsp, fim do protótipo usando tela de 3”
Sprint Review/Planning	10/01/2025	10/01/2025	Encerramento sprint 12, conversas sobre documentação e primeiros feedbacks sobre escrita
Sprint 13	10/01/2025	17/01/2025	Progresso na documentação e correções após feedback
Sprint 14	17/01/2025	24/01/2025	Progresso na documentação
Sprint 15	24/01/2025	31/01/2025	Progresso na documentação
Sprint 16	31/01/2025	07/02/2025	Finalização da documentação
Sprint 17	08/02/2025	14/02/2025	Construção da apresentação TCC 1
Sprint 18,19	15/02/2025	26/02/2025	Período de Apresentação TCC 1
Recesso	26/02/2025	24/03/2025	
Sprint 20	24/03/2025	28/03/2025	Atualização de códigos e refatoração

Continua na próxima página

Sprint	**Data início**	**Data fim**	**Descrição**
Sprint Review/Planning	28/03/2025	28/03/2025	Retorno ao diálogo com o orientador
Sprint 21	29/03/2025	04/04/2025	Novos testes de extração de dados via OBD-II Chevrolet Onix
Sprint Review/Planning	04/04/2025	04/04/2025	Contato com o novo cliente, mudança de escopo, início de um aplicativo de extração, transformação e carga de DTCs
Sprint 22	04/04/2025	11/04/2025	Nova proposta e estudo de viabilidade
Sprint Review/Planning	11/04/2025	11/04/2025	Reunião marcada com gestor de aplicativo de ERP de oficinas
Sprint 23	12/04/2025	18/04/2025	Após a reunião, a equipe iniciou processo de execução da nova proposta
Sprint Review/Planning	18/04/2025	18/04/2025	Decisão da arquitetura voltada ao serviço de MQTT e aplicação intermediária
Sprint 24	18/04/2025	25/04/2025	Algoritmo na ESP32 para serialização de dados retirados pela ECU em JSON
Sprint 29	24/05/2025	30/05/2025	Desenvolvimento do app intermediário e documentação
Sprint 25	25/04/2025	02/05/2025	Desenvolvimento do app intermediário e documentação
Sprint 26	03/05/2025	09/05/2025	Desenvolvimento do app intermediário e documentação
Sprint 27	10/05/2025	16/05/2025	Desenvolvimento do app intermediário e documentação

Continua na próxima página

Sprint	Data início	Data fim	Descrição
Sprint 28	17/05/2025	23/05/2025	Desenvolvimento do app intermediário e documentação
Sprint 29	24/05/2025	30/05/2025	Desenvolvimento do app intermediário e documentação
Sprint 30	30/05/2025	06/06/2025	Apresentação do MVP para o cliente
Sprint Review/Planning	06/06/2025	06/06/2025	Feedbacks recebidos pelo cliente e pesquisa para adição de nova feat envolvendo LLM
Sprint 31	07/06/2025	13/06/2025	Implementação de inteligência artificial como indicadora de soluções para histórico de DTCs e veículos

3.1.4 Extreme Programming (XP)

É uma metodologia ágil que busca qualidade de software e qualidade de vida para o time de desenvolvimento. Foi criada na década de 90 pelo engenheiro de software Kent Beck durante um projeto para uma grande companhia de automóveis. A metodologia é baseada na ideia de que o desenvolvimento de software deve estar preparado para mudanças e que essas mudanças podem surgir por meio do ciclo contínuo de desenvolvimento, sempre com *feedback* ao final (SALEM, 2024).

Enquanto o Scrum foca mais na parte geral de desenvolvimento de um projeto, a metodologia XP se estende ao desenvolvimento do software em si, trazendo diversas práticas focadas em programação, como por exemplo a programação em pares e a integração contínua.

A metodologia de implementação do projeto será baseada, principalmente, na programação em pares (*Pair Programming*) da metodologia ágil XP (*Extreme Programming*). A programação em pares consiste basicamente em programar em dupla, utilizando o mesmo teclado, mouse e monitor (MANOEL, 2006). Utilizando este método de maneira remota, um de nós compartilha a tela por meio de um ferramenta de comunicação e o outro assume o posto de observador, prestando mais atenção aos detalhes e revisando o código a medida que o outro escreve. Além de nivelar o conhecimento técnico do par, também minimiza a quantidade de erros no código e acelera a solução de problemas.

3.1.5 Heurística da Tentativa e Erro

Durante o processo de pesquisa, foi adotada uma abordagem mais próxima da Heurística da tentativa e erro, que, segundo Eryk, é fundamentalmente impulsionada pelo ciclo de ação e retorno (*feedback*), em que o resultado de uma ação te guia para o próximo passo (BRANCH, 2024).

Essa metodologia foi especialmente relevante em etapas críticas do projeto, como na integração entre o ESP32 e o módulo ELM327. Nesses momentos, a equipe recorreu a ciclos iterativos, simplificando gradualmente as camadas mais complexas do problema por meio da persistência e da análise dos resultados, até que o objetivo final fosse alcançado.

Durante testes iniciais com o Chevrolet Onix, foi observado um comportamento peculiar: o módulo ELM327 não retornava dados válidos quando configurado no modo de detecção automática de protocolo `AT SP 0`. Comandos padrões para leitura de dados, que funcionavam no Renault Sandero, retornava apenas a mensagem `"NO DATA"`. O processo de tentativa e erro teve como curso:

1. **Falha na autodetecção**: A primeira tentativa foi utilizar o `AT SP 0` para autodetecção, como havia funcionado no Renault Sandero. Como resultado o módulo identificou incorretamente o protocolo e não estabelecia comunicação estável.

2. **Configuração Manual do Protocolo**: Foi manualmente configurado o protocolo ISO 15765-4 CAN (500Kbps, 11bits) com o comando `AT SP 6`. Imediatamente foi observado que o módulo passou a reconhecer a ECU, mas as respostas ainda eram inconsistentes.

3. **Aguardo da resposta**: Foi implementado um atraso de três segundos entre o envio de comandos, isso fez com que o algoritmo aguardasse a resposta da ECU garantindo tempo suficiente para processar e responder aos comandos.

3.2 Ferramentas

3.2.1 ELM327

O ELM327 tem por objetivo atuar como uma ponte entre a porta OBD do veículo e uma interface de comunicação, como a ESP32 no caso apresentado. Ele consegue interpretar e se comunicar com 9 protocolos OBD diferentes (ELECTRONICS, 2024).

Para enviar comandos OBD para que o ELM327 possa "repassar"para a ECU, são necessários 2 pares de *byte*, que serão tratados como dígitos hexadecimais. Os comandos são enviados em um pacote de dados para o veículo e a maioria dos padrões requer 3 *bytes* para o cabeçalho e 1 *textsum* para indicar possíveis corrompimentos de dados. O ELM327

adiciona esses *bytes* automaticamente. A maioria dos comandos OBD possui 1 ou 2 *bytes* (ELECTRONICS, 2024).

O primeiro *byte* enviado seria bem definido como o "modo"e especifica o tipo de informação requisitada, como dados em tempo real, informações do veículo ou DTCs. Por lei, os veículos não obrigatoriamente precisam disponibilizar todos os PIDs existentes. Por conta disso, o PID "00"no modo "01"(dados em tempo real) deve ser suportado por todos os veículos, e seu retorno é exatamente quais PIDs são suportados pelo mesmo (ELECTRONICS, 2024).

Figura 1 – ELM327 utilizado para o projeto.
Fonte: Amazon, 2025

3.2.2 *Serial Bluetooth Terminal*

O *Serial Bluetooth Terminal* é um aplicativo desenvolvido para dispositivos Android que fornece uma interface de terminal baseada em comandos CLI (*Command-Line Interface*), permitindo a comunicação com dispositivos *Bluetooth* por meio do protocolo SPP (*Serial Port Profile*).

A funcionalidade relevante do aplicativo para o projeto foi a capacidade de identificar o endereço MAC de dispositivos *Bluetooth*, como o do módulo ELM327, sem necessidade de ferramentas adicionais. Isso também torna-se particularmente útil para testes

de versionamento de *firmware*, protocolos e PID's sem a urgência de desenvolvimento de instrumentos para manuseio e leitura dos dados.

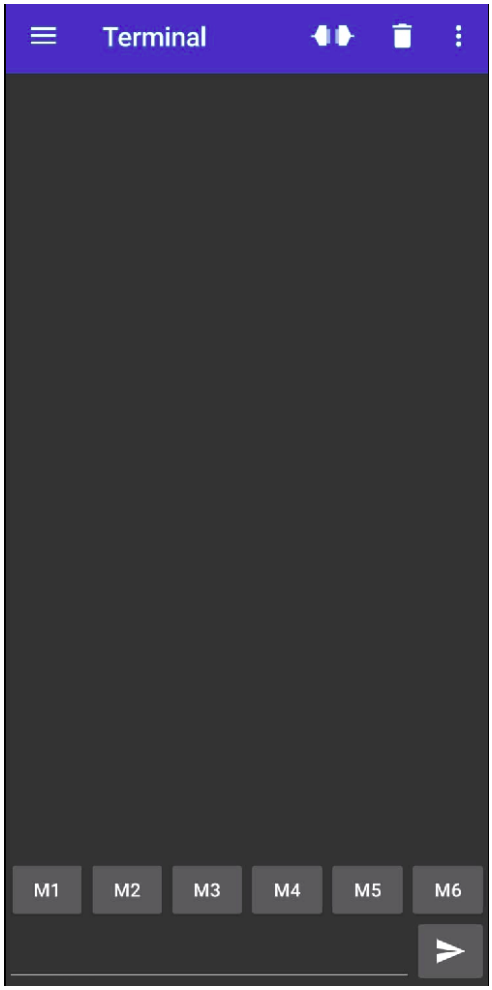

Figura 2 – *Terminal Serial Bluetooth*.
Fonte: Autoria própria

3.2.3 *Car Scanner*

O *Car Scanner* é um aplicativo disponível para Android e iOS que disponibiliza para o usuário dados provenientes da ECU do veículo. Depende de uma conexão ativa com um ELM327 conectado na porta OBD-II do veículo. Foi o meio inicial para testar o dispositivo ELM327 adquirido e o primeiro contato direto com os dados retornados pela ECU. Além disso, possui um terminal que pôde ser utilizado para testar o envio direto de PIDs.

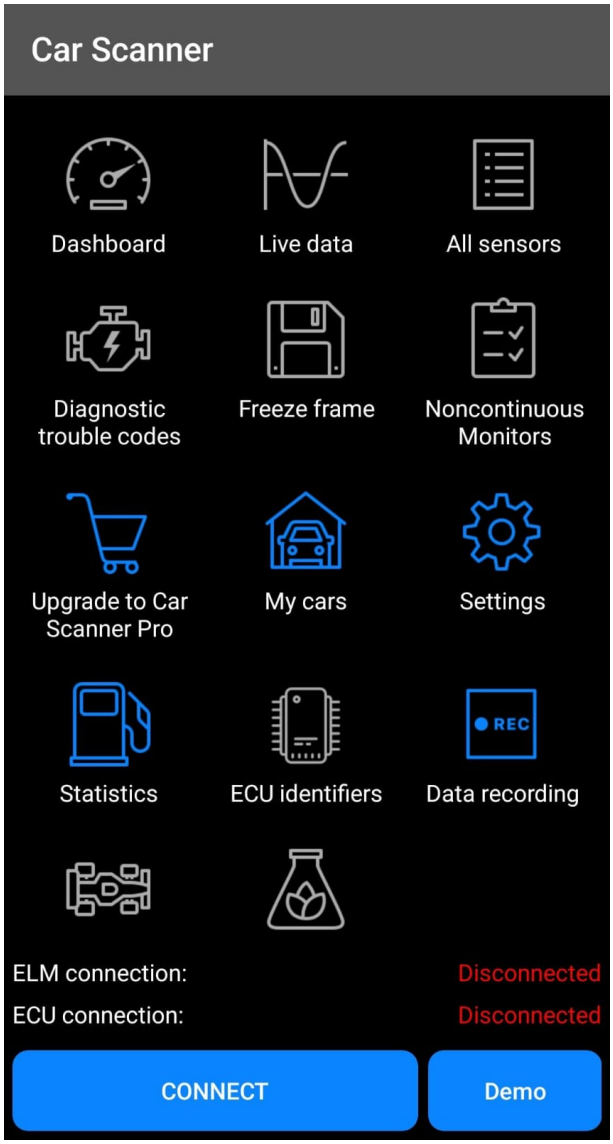

Figura 3 – Car Scanner.
Fonte: Autoria própria

3.2.4 ESP32-WROOM-32

O ESP32-WROOM-32 é um módulo de microcontrolador (MCU) com conectividade Wi-Fi, *Bluetooth®* e *Bluetooth Low Energy (BLE)*, desenvolvido para aplicações que demandam flexibilidade e alto desempenho. Baseado no chip ESP32-D0WDQ6, o módulo combina capacidade de processamento *dual-core* com arquitetura escalável, permitindo adaptação a diferentes cenários de uso. Além das funcionalidades de comunicação sem fio, o ESP32 integra periféricos essenciais, como interfaces de comunicação serial (UART, SPI, I2C), amplamente utilizadas em projetos de automação, IoT e sistemas embarcados. (SYSTEMS, 2023)

É a ferramenta central para comunicação com o veículo na pesquisa, atuando em conjunto com o dispositivo ELM327 (que serve como intermediário para o protocolo

OBD-II). Essa integração permite a requisição, processamento e interpretação dos dados do veículo, além do envio de comandos personalizados via interface serial, graças ao código gravado no módulo.

Figura 4 – ESP32 utilizado para o projeto.
Fonte: Robocore, 2025

3.2.5 *Espressif IoT Development Framework* (ESP-IDF)

A Espressif providencia um SDK (*Software Development Kit*) auto-suficiente para desenvolvimento em plataformas como a ESP-32, utilizando linguagens de programação como C e C++. Permite o desenvolvimento de aplicações simples, como lâmpadas e brinquedos, até mais complexas como equipamentos eletrodomésticos de grande porte (ESPRESSIF, 2024).

Com um processo de produção bem definido e políticas de suporte ao usuário que asseguram sempre uma versão estável, a ESP-IDF está disponível de graça pela plataforma *GitHub*. Além disso, algumas ferramentas de desenvolvimento para IDEs comumente utilizadas como *Eclipse* e *VSCode* estão presentes para facilitar mais ainda o processo de desenvolvimento (ESPRESSIF, 2024).

A extensão do *VSCode* permite uma enorme facilidade para compilar e fazer o *flash* na ESP-32, além de mostrar possíveis erros em qualquer um dos dois processos, acelerando a iteração de produção de código e testes no carro.

Por fim, a documentação da ESP-IDF é bem extensiva, não só no nível de uso mas de design também, apresentando mais de 100 exemplos bem mantidos e testados com uso de componentes e periféricos de *hardware* (ESPRESSIF, 2024).

3.2.6 Arduino IDE

É uma aplicação de código aberto para facilitar o desenvolvimento de programas para microcontroladores Arduino ou fabricados por terceiros. Com uma interface amigável, a IDE providencia um ambiente para escrita, compilação e gravação de código nas placas, além de um monitor serial, e foi utilizada no início das pesquisas para realizar testes de conexão da ESP32 com o ELM327 e testes de envio de PIDs, sendo o primeiro contato com o formato de dados recebidos pela ECU.

3.2.7 ELMduino

Trata-se de uma biblioteca implementada para abstrair a comunicação com a interface OBD-II por meio do módulo ELM327, elaborada para o ambiente de desenvolvimento integrado Arduino IDE e disponibilizada na plataforma *GitHub*. Durante a fase de testes de integração, essa biblioteca desempenhou papel importante para aprofundar o entendimento sobre o diálogo entre a ECU e o módulo, permitindo a interpretação dos PIDs retornados pela interface OBD-II. Além disso, permitiu a aplicação prática dos conceitos teóricos estudados, consolidando o conhecimento por meio de experimentação.

3.2.8 Next.js

De acordo com repositório oficial, Next.js é um *framework* que simplifica a criação de aplicações *web* completas, oferecendo otimizações de desempenho e configuração automática de ferramentas de baixo nível, como *bundlers* e compiladores, permitindo que os desenvolvedores foquem diretamente na construção e entrega do produto, sendo adequado tanto para uso individual quanto em equipes maiores (Vercel, 2024).

O Next.js foi selecionado devido ao conhecimento e facilidade prévios com o *framework* e apresentar a possibilidade de elaboração de um *frontend* utilizando React e conectado a um *backend* que consuma a API, cumprindo não apenas o desejo do cliente de possuir um banco de dados acessível, como também trazendo um aspecto visual para o trabalho e aprimorando sua apresentação.

3.2.9 Prisma

Segundo a documentação oficial, o Prisma ORM é uma ferramenta de mapeamento objeto-relacional de próxima geração, composta pelo *Prisma Client*, *Prisma Migrate* e *Prisma Studio*, oferecendo suporte a bancos relacionais e ao MongoDB. Seu Schema permite aos desenvolvedores definir os modelos da aplicação de forma intuitiva, por meio de uma linguagem de modelagem (Prisma ORM, 2024).

O uso da ferramenta na síntese projeto deu-se, principalmente, por permitir um código mais legível e conciso, facilitando a manutenção e possíveis refatorações e, além disso, também é familiar para os membros da dupla, sendo confortável a utilização no projeto.

3.2.10 Tailwindcss e Shadcn

O autor Ivaylo Gerchev (2022), em seu livro, descreve o Tailwind como um *framework* baseado em utilitários de baixo nível, reutilizáveis, que permitem a criação de qualquer tipo de *layout* de forma eficiente e flexível. Seu foco em utilitários CSS essenciais torna-o ideal tanto para prototipação rápida quanto para aplicações mais complexas. Além disso, possui documentação extensa e recursos interativos que auxiliam no aprendizado e na personalização.

O Shadcn, por sua vez, é uma ferramenta que disponibiliza diversos componentes altamente personalizáveis e que podem ser adicionados ao passo que são utilizados. Isso evita pacotes de materiais extensos e pesados em questão de espaço e que necessitam de uma instalação por completo. Por fim, os componentes disponibilizados pelo shadcn utilizam tailwindcss em sua estilização e tornam a utilização de ambos ideal para o projeto.

3.2.11 Sandero Stepway e Onix LT

Os veículos utilizados para pesquisa e teste foram um Renault Sandero Stepway 2016 e um Chevrolet Onix LT 1.0 2019, ambos os veículos comercializados no Brasil e devidamente equipados com uma interface OBD-II padrão. Ambos os carros estavam a pronta disposição dos membros e, portanto, foram a escolha mais viável. Os veículos a priori funcionaram como ferramenta de validação e iteração dos ciclos, permitindo exploração, execução de comandos e análise dos dados retornados, sendo essenciais para a aplicação prática dos conceitos teóricos discutidos no projeto.

(a) Renault Sandero Stepway 2016 (b) Chevrolet Onix LT 1.0 2019

Figura 5 – Veículos usados na pesquisa.
Fonte: Comprecar, 2016 e Carros na web, 2019

4 Proposta de Solução

4.1 Requisitos do Sistema

A elicitação de requisitos é uma das fases mais cruciais de planejamento de um sistema, onde a maioria, senão todos os requisitos necessários são corretamente identificados antes do início da implementação do sistema. De acordo com Aurum e Wohlin (2005), "todo projeto começa com uma elicitação de requisitos".

Para a elaboração arquitetural e implementação do projeto, foram inicialmente realizadas atividades de engenharia de requisitos, com foco de identificar e documentar as necessidades relevantes para o contexto do protótipo e a validação do fluxo de integração dos códigos DTCs. Para a elicitação dos requisitos foram conduzidas técnicas como User Stories, de onde foram retirados os requisitos funcionais da aplicação e sua priorização através do método MoSCoW (*Must Have, Should Have, Could Have, Won't have*), a união destas duas estratégias permitiram com que as funcionalidades essenciais fossem capturadas e alinhadas com os objetivos do projeto e auxiliaram numa gestão do escopo.

De acordo com Mike Cohn (2004), a técnica de *User Story* (História de Usuário) descreve de forma breve e simples uma funcionalidade desejada, escrita do ponto de vista do usuário final. Considerando que a proposta partiu de um empresário do ramo de oficinas mecânicas, que atua como cliente neste contexto, as *User Stories* foram elaboradas com base nas demandas por ele levantadas.

Já a estratégia MoSCoW (*Must Have, Should Have, Could Have, Won't Have*), conforme descrito por Hudaib et al. (2018), é uma técnica de priorização que teve origem no *Dynamic Software Development Method* (DSDM), um método ágil de desenvolvimento de *software*. Caracterizado por ser simples, este método é utilizado por analistas e *stakeholders* para priorização de requisitos de maneira colaborativa. De acordo com seus princípios, o MoSCoW pontua que os requisitos podem ser classificados em quatro principais prioridades, sendo elas:

- **M - *Must Have***: Os requisitos contidos nesse grupo devem ser entregues; caso não seja feito, indica que o projeto é falho.
- **S - *Should Have***: Os requisitos contidos nesse grupo são de alta prioridade, não são críticos ao sucesso do projeto, porém de alto valor e suma importância.
- **C - *Could Have***: Os requisitos contidos nesse grupo são desejáveis para o resultado final, porém não necessários.

- **W - *Won't Have***: Os requisitos contidos nesse grupo provêm pouco ou quase nenhum valor e podem ser descartados no estágio de desenvolvimento atual.

Tabela 2 – User Stories

ID	Descrição
US01	Como gestor/mecânico, eu quero inicializar a comunicação com o ELM327/Esp32 via Bluetooth.
US02	Como gestor/mecânico, eu quero configurar o protocolo OBD-II usado pelo ELM327.
US03	Como gestor/mecânico, eu quero que o sistema seja capaz de extrair e enviar os códigos de erro (DTCs) e o VIN do respectivo veículo.
US04	Como gestor/mecânico, eu quero uma interface intermediária simples de fácil manuseio.
US05	Como gestor/mecânico, eu quero que a interface me possibilite de associar carros a uma respectiva leitura de DTCs.
US06	Como gestor/mecânico, eu quero ser capaz de carregar uma base de dados no formato CSV de carros pré cadastrados no meu negócio.
US07	Como gestor/mecânico, eu quero que uma inteligência artificial seja capaz de recomendações de reparo baseadas nos DTCs e os respectivos veículos.
US08	Como gestor/mecânico, eu quero que um dashboard que me mostre dados valiosos sobre as marcas e os respectivos DTCs mais recorrentes.
US09	Como gestor/mecânico, eu quero que todos os dados sejam persistidos gerando uma base de dados histórica.
US10	Como gestor/mecânico, eu quero filtrar buscas sempre que for fazer uma associação.

Tabela 3 – Requisitos com Priorização MoSCoW

ID	Requisito	Prioridade
RF01	O sistema deve inicializar a comunicação com o ELM327/ESP32 via Bluetooth SPP.	Must have

Continua na próxima página

ID	Requisito	Prioridade
RF02	O sistema deve permitir a configuração do protocolo OBD-II usado pelo ELM327.	Should Have
RF03	O sistema deve ser capaz de extrair e enviar os códigos de erro (DTCs) e o VIN do respectivo veículo.	Must have
RF04	O aplicativo intermediário deve fornecer uma interface simples e de fácil manuseio.	Should have
RF05	O aplicativo deve ser capaz de associar carros a leituras feitas pelo sistema.	Must have
RF06	O aplicativo deve ser capaz de aceitar uma base de dados em formato CSV.	Could have
RF07	O aplicativo deve utilizar inteligência artificial para sugerir recomendações baseadas nos DTCs e veículos.	Must have
RF08	O aplicativo deve possuir um dashboard com dados valiosos sobre marcas e DTCs mais recorrentes.	Could Have
RF09	O sistema deve persistir todos os dados em uma base histórica.	Must have
RF10	O aplicativo deve permitir filtros de busca para facilitar a associação de dados.	Should have

4.2 Arquitetura

Com os requisitos definidos, foi possível a criação de três importantes diagramas para definir a arquitetura do projeto: Diagrama de Blocos, Diagrama de Hierarquia de Controle e Diagrama de Camadas. Elecia White (2024) explica que "tais diagramas permitem uma visão do sistema como um todo, ajudam a identificar dependências e providenciam percepção para a definição de novas funcionalidades.".

- **Diagrama de Blocos**: É uma representação utilizando uma notação de blocos interconectados para representar os componentes de um sistema, tanto de software quanto de hardware. Facilita a visualização das dependências e permite montar a arquitetura de maneira intuitiva e simples, dado que os blocos abstraem a complexidade de funcionamento e destacam apenas a função principal (WHITE, 2024).
- **Diagrama de Hierarquia de Controle**: Utilizando os blocos criados no diagrama anterior, é possível montar uma representação de hierarquia que permite visualizar os níveis de dependência e controle de cada componente (WHITE, 2024).
- **Diagrama de Camadas**: O diagrama de camadas busca demonstrar visualmente a complexidade de cada objeto. Deve ser feito de baixo para cima, com as camadas

inferiores representando componentes mais básicos e camadas superiores representando funcionalidades de alto nível. Além disso, os blocos dos objetos de nível superior devem tocar nos blocos de nível mais baixo que são utilizados por eles para melhor visualizar a estrutura do sistema (WHITE, 2024).

4.2.1 Diagrama de Blocos

O diagrama apresentado na figura 6 ilustra a arquitetura atualizada do sistema que compõe o ambiente da ferramenta ElmEsp, que integra *hardware* embarcado, comunicação via *Bluetooth*, mensageria via WI-FI, inteligência artificial e persistência em banco de dados.

O ESP32 é o microcontrolador responsável pela comunicação com o módulo ELM327 via *Bluetooth* utilizando o perfil SPP(*Serial Port Profile*). O algoritmo em C gerencia a comunicação, permitindo o envio de comandos PID(*Parameter Identifiers*) do protocolo OBD-II, bem como a leitura dos DTCs (*Diagnostic Trouble Codes*).

O módulo ELM327, por sua vez, atua como intermediário entre o ESP32 e a ECU(*Eletronic Control Unit*) do veículo, repassando os comandos recebidos para a ECU e retornando as respostas com dados veiculares.

Os dados coletados (como DTCs e identificadores do veículo) são processados pelo ESP32 e serializados em JSON, sendo publicados via protocolo MQTT para uma aplicação intermediária.

O Aplicativo intermediário recebe as mensagens JSON e realiza as seguintes funções:

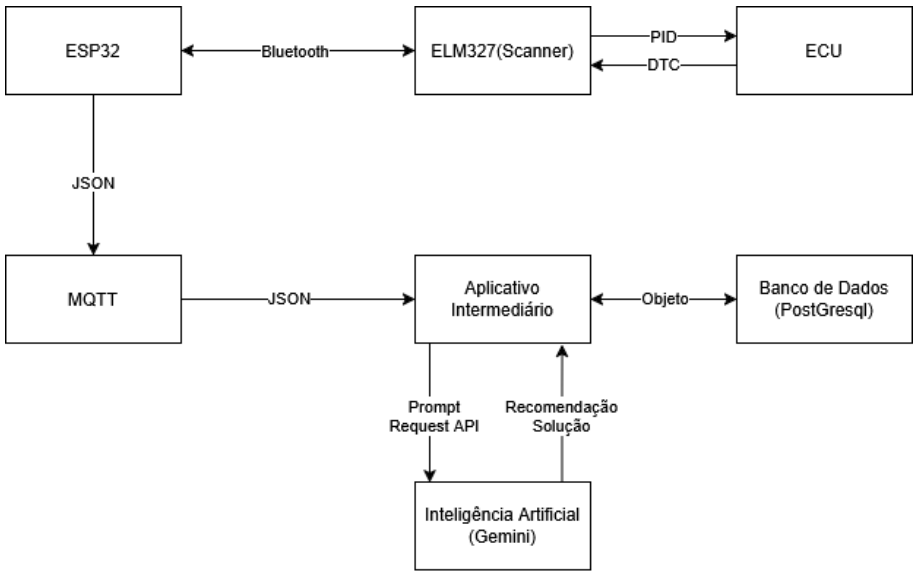

Figura 6 – Diagrama de Blocos
Fonte: Autoria própria, 2025

- **Persistência de dados e criação de base histórica**: Armazena as informações recebidas em um banco de dados PostgreSQL, organizando os dados por veículos e associações com diagnósticos.

- **Interação com inteligência Artificial**: Utiliza os dados recebidos para montar prompts que são enviados à IA Gemini via API, solicitando recomendações de soluções para os códigos de falha (DTCs) e seus respectivos veículos.

- **Disponibilidade de API**: A aplicação intermediária disponibiliza uma API para consultas ao serviço.

Essa arquitetura permite o monitoramento remoto de falhas veiculares, a recomendação automatizada de soluções e o armazenamento de dados de diagnóstico, fornecendo uma base futura para análises preditivas e integração com ERPs ou aplicações de oficina.

4.2.2 Diagrama de Hierarquia de Controle

A Figura 7 representa a hierarquia de controle do sistema de alto nível, evidenciando os módulos que compõem o aplicativo intermediário e suas interações com os demais elementos da arquitetura. Essa estrutura é projetada para lidar com os dados extraídos pela leitura veicular, realizada pelo sistema embarcado. O aplicativo intermediário divide-se em camadas, como a interface com o usuário (*Dashboard* e Tabelas), os serviços de integração (MQTT) e os mecanismos de consulta e associação de códigos de falha (DTCs). O módulo de solução de DTCs pode solicitar apoio a IA Gemini, integrando informações adicionais às entradas do sistema gerando dados históricos sobre DTCs, seus veículos e possíveis soluções. Os dados trafegados e tratados são persistidos em um banco de dados PostgreSQL.

Já a Figura 8 representa a hierarquia de controle do algoritmo embarcado, demonstrando o fluxo executado internamente no microcontrolador ESP32 para se comunicar com o ELM327 e respectivamente com a ECU do veículo, permitindo que o microcontrolador seja capaz de estabeler conexão, extrair dados e serializar os mesmos publicando-os em uma *branch* do MQTT que será responsável pela integração do aplicativo intermediário as informações.

4.2.3 Diagrama de Camadas

O diagrama de camadas representa a estrutura funcional do sistema de forma hierárquica, divida em dois grandes blocos: Algoritmo (responsável pela coleta e envio dos dados veiculares) e Aplicativo (responsável pelo enriquecimento e visualização dos dados).

Na base, encontram-se os componentes mais fundamentais, como os protocolos de comunicação, transações e armazenamento de dados. Camadas superiores representam funcionalidades de alto nível, como a interface intermediária composta por *dashboard*, tabelas e integração com IA para requisição de soluções.

Os blocos superiores estão posicionados diretamente sobre os módulos de nível inferior dos quais dependem, evidenciando a modularidade e a dependência funcional entre as partes.

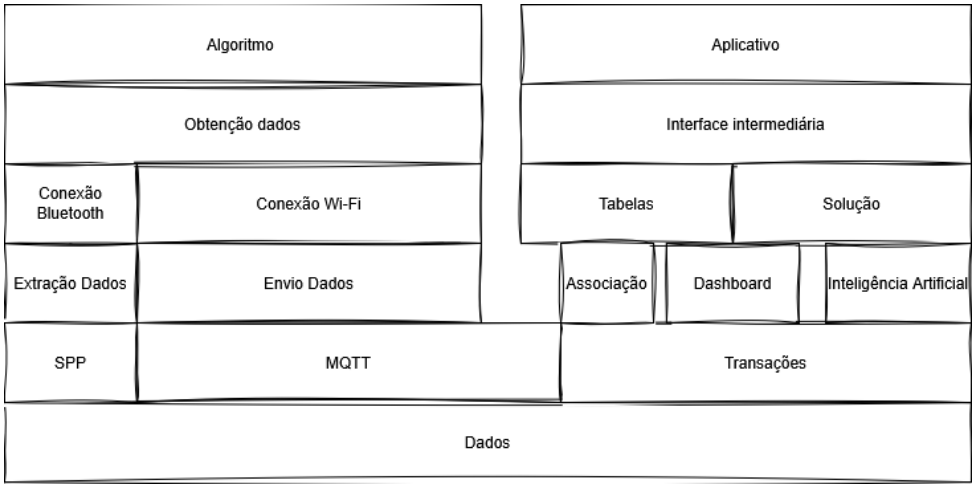

Figura 9 – Diagrama de Camadas
Fonte: Autoria própria, 2025

4.2.4 Protótipo

Um protótipo pode ser definido como uma representação inicial de um sistema, que permite a visualização, experimentação e validação de funcionalidades antes da implementação completa do software final. Segundo Bjarnason, Lang e Mjöberg (2023), a prototipagem é um elemento central do ciclo de desenvolvimento iterativo, pois facilita a captura de requisitos, a comunicação entre stakeholders e a redução de riscos por meio da experimentação e do *feedback* precoce.

Alinhado a essa perspectiva, foi desenvolvido o protótipo para o sistema proposto nesse trabalho, cujo objetivo é o monitoramento e gerenciamento de veículos e suas respectivas leituras de diagnóstico. Por meio desse protótipo (Figura 10), buscou-se oferecer uma interface clara e funcional.

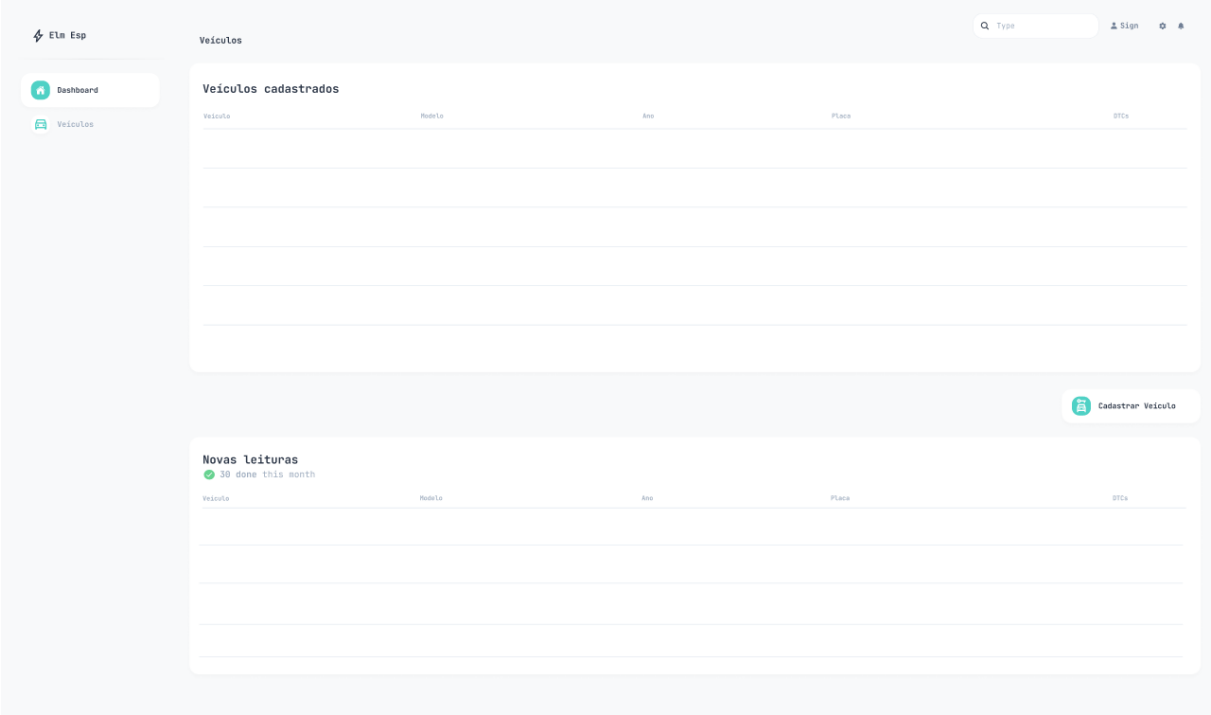

Figura 10 – Protótipo da aplicação intermediária
Fonte: Autoria própria, 2025

4.2.5 *Roadmap*

Um *Roadmap* é a representação visual das tarefas no período de tempo determinado para o projeto. As tarefas são representadas por barras que atravessam a tabela de semanas, determinando quanto tempo cada tarefa irá durar e qual sua data de início e fim. Além disso, é possível ter uma boa noção e planejamento de atividades que se sobreponham.

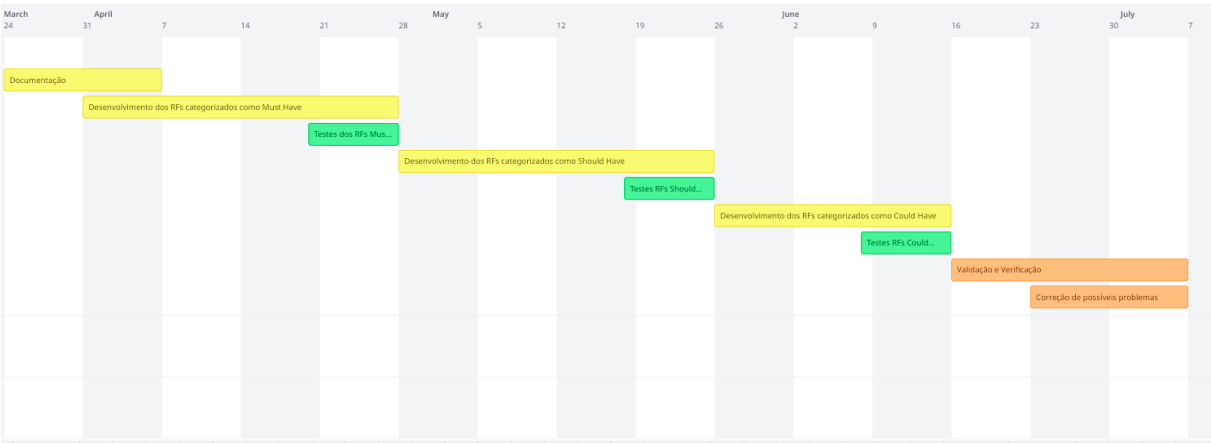

Figura 11 – Roadmap para o TCC 2
Fonte: Autoria própria, 2025

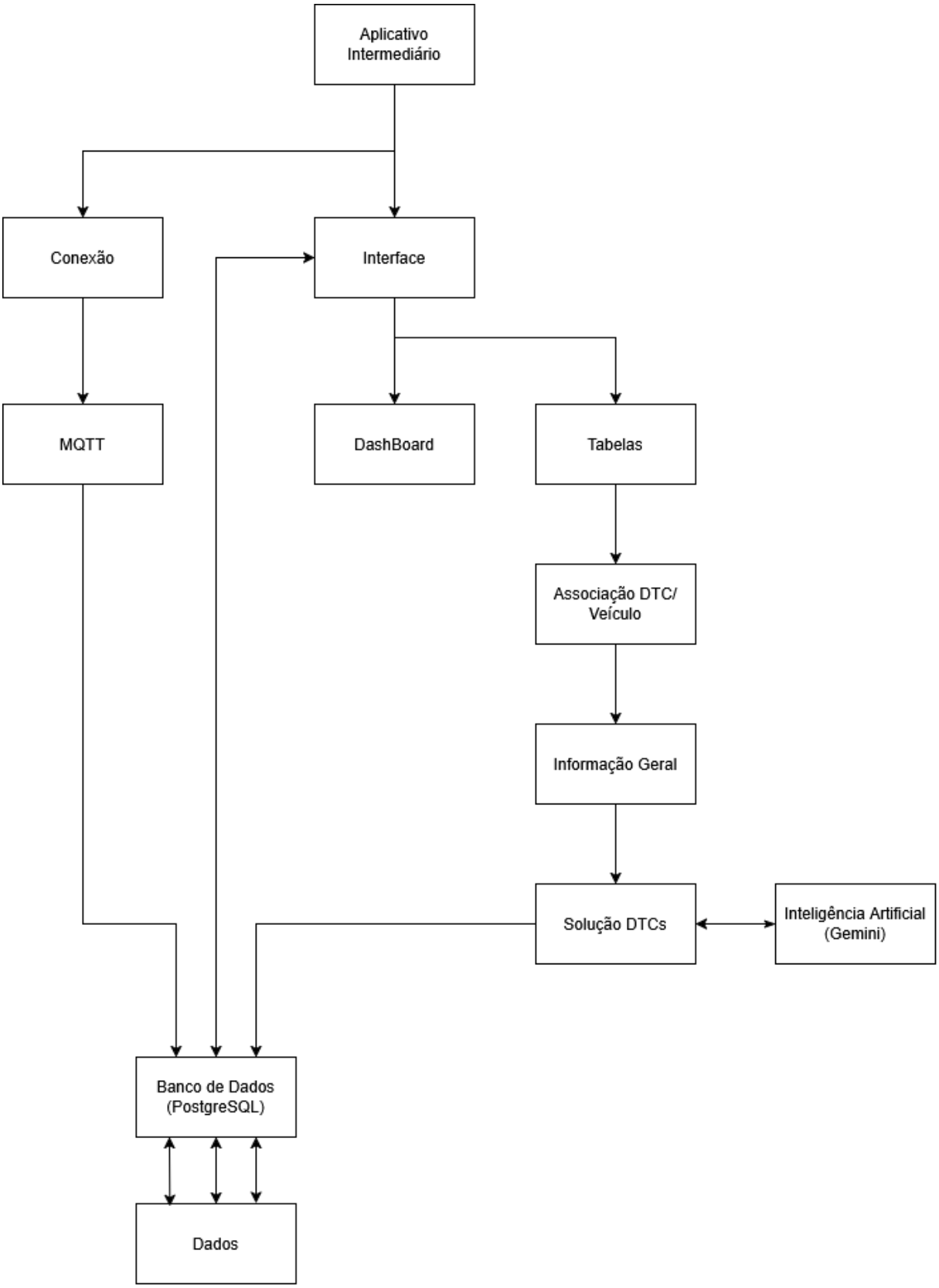

Figura 7 – Diagrama de Hierarquia de Controle Aplicativo intermediário
Fonte: Autoria própria, 2025

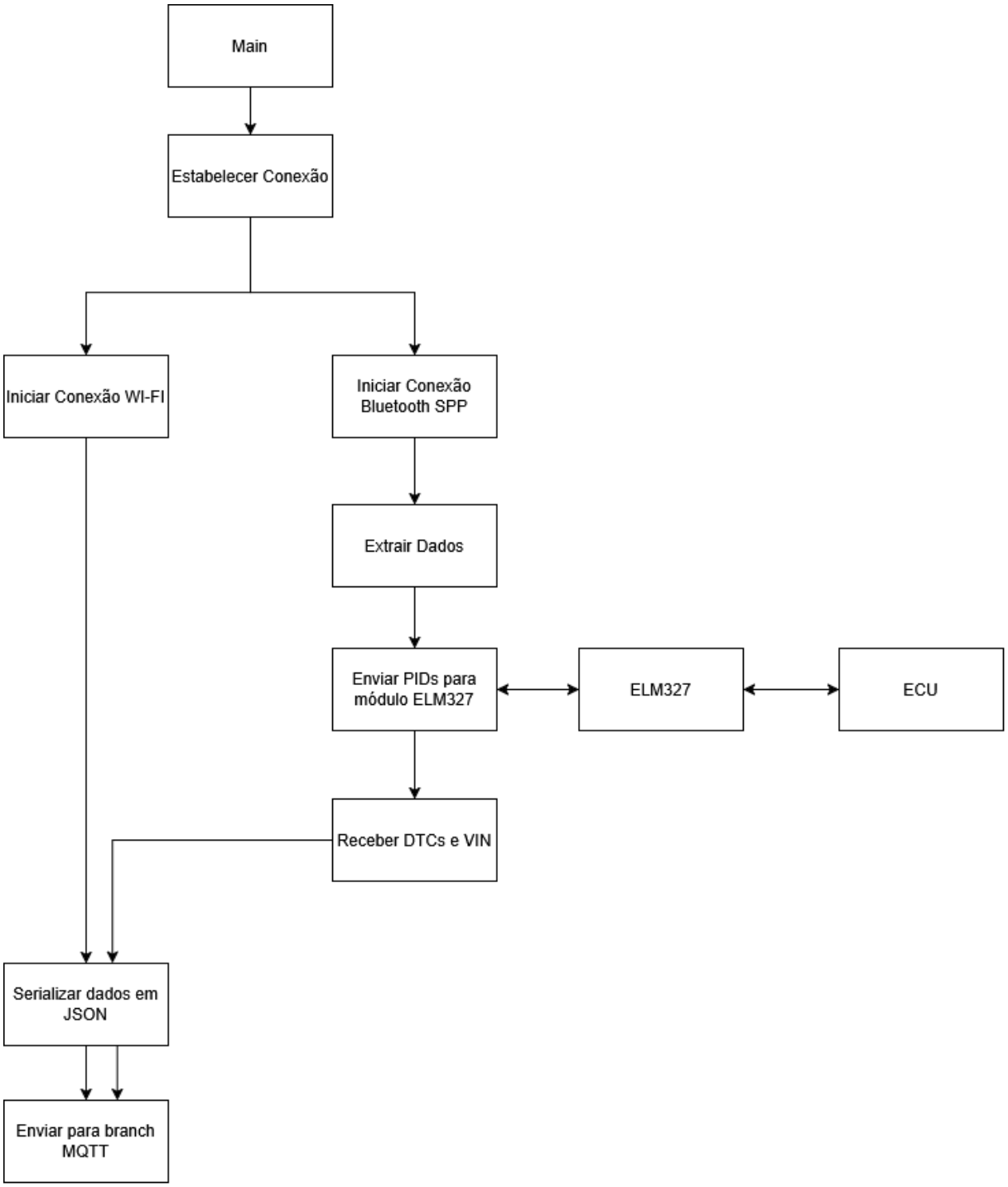

Figura 8 – Diagrama de Hierarquia de Controle Algoritmo
Fonte: Autoria própria, 2025

5 Resultados finais

5.1 Funcionamento do ELM327 e ESP32 em ambiente controlado

5.1.1 ELM327

O ELM327 deve estar conectado na porta OBD-II do veículo cujos DTCs serão extraídos e a chave deve estar na posição de acessórios. O veículo não precisa estar com o motor ligado.

Figura 12 – ELM327 conectada no carro

5.1.2 ESP32

A ESP32 deve estar conectada ao computador e configurada com acesso à uma rede wifi.

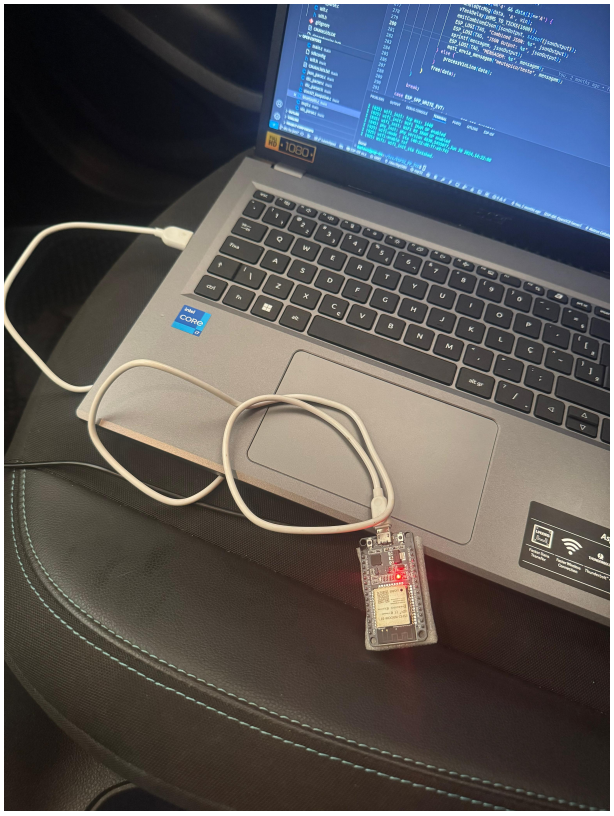

Figura 13 – ESP32 conectada

Ela tentará conexão com o ELM327 e, caso obtenha sucesso, enviará os comandos para extrair o VIN e os códigos DTC do veículo, em sequência enviando via MQTT para o servidor que estará inscrito no canal.

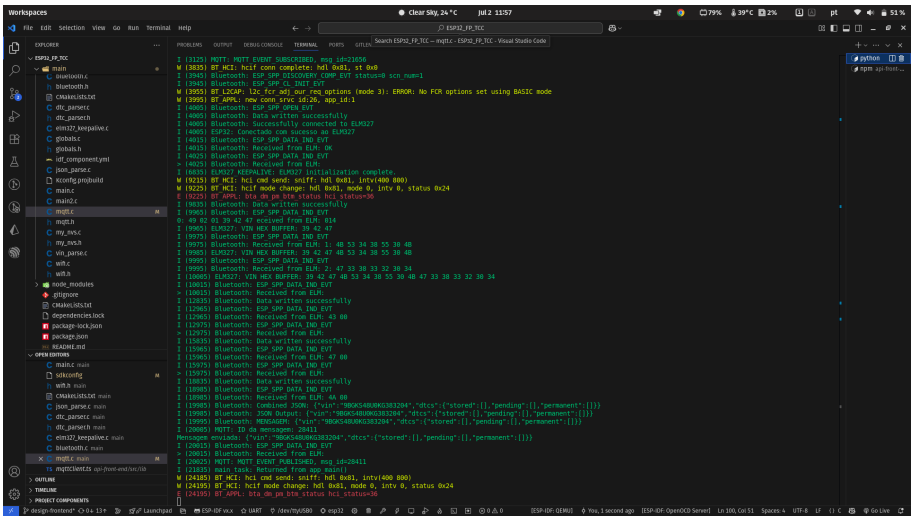

Figura 14 – Log de envio do mqtt

Ao receber a mensagem, o servidor fará a inserção no banco de dados, tornando o mesmo disponível na API e na aplicação*web*.

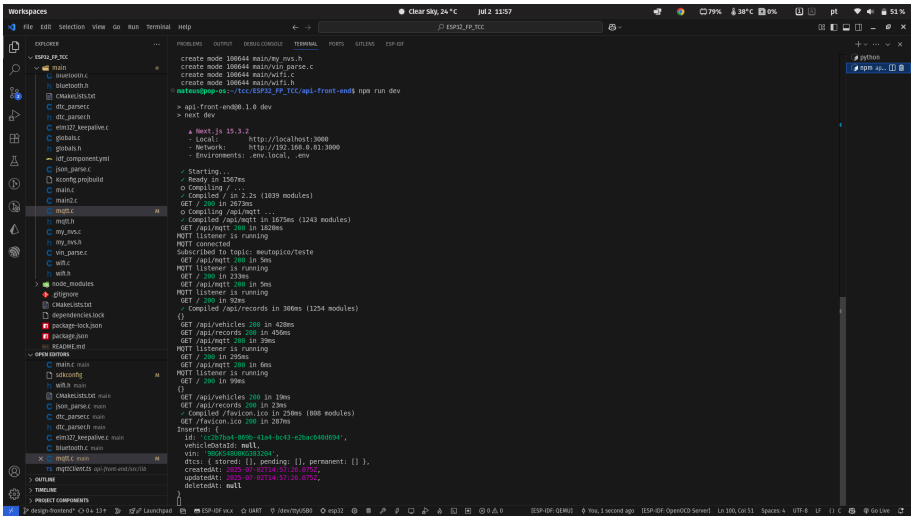

Figura 15 – Log do servidor

5.2 Funcionamento da aplicação web

5.2.1 Tela inicial

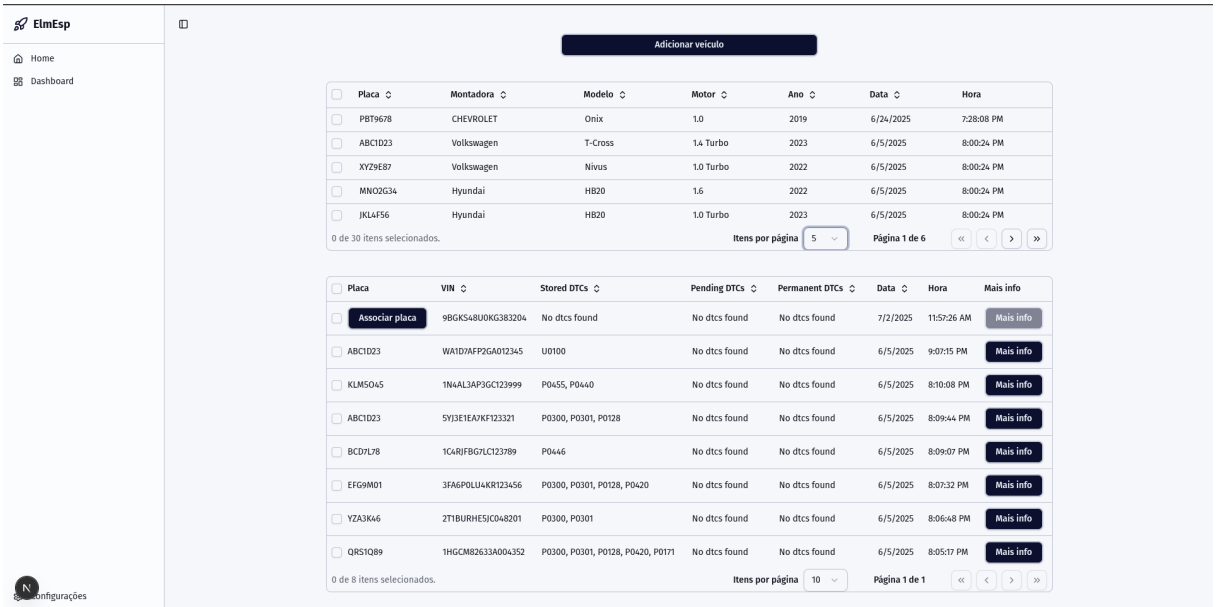

Figura 16 – Tela inicial

A tela inicial foi desenvolvida com o objetivo de proporcionar uma visualização concisa e imediata das informações mais relevantes, divididas em duas seções tabulares principais e um controle para adição de veículos.

5.2.2 Tabela de Veículos Cadastrados

A seção superior da Figura 16 exibe a tabela de "Veículos Cadastrados". Esta tabela cataloga os veículos registrados na base de dados do sistema, oferecendo um panorama

resumido das suas características. As colunas apresentadas são:

- **Placa:** Placa do veículo.
- **Montadora:** Fabricante do veículo.
- **Modelo:** Modelo do veículo.
- **Versão:** Especificação do modelo.
- **Motor:** Motorização do veículo.
- **Ano:** Ano de fabricação do veículo.
- **Data e Hora:** Timestamp do registro do veículo no sistema.

Acima desta tabela, o botão "Adicionar veículo"permite a inclusão de novos registros na base de dados.

5.2.3 Tabela de Códigos de Diagnóstico de Falha (DTCs)

A seção inferior da Figura 16 apresenta a tabela de "DTCs Extraídos dos Veículos". Esta tabela consolida os códigos de falha de diagnóstico (*Diagnostic Trouble Codes* - DTCs) obtidos dos veículos, sendo crucial para a identificação e monitoramento de anomalias. As informações exibidas incluem:

- **Placa:** Placa do veículo associado ao DTC, se o veículo estiver previamente cadastrado.
- **VIN (*Vehicle Identification Number*):** Número de Identificação do Veículo.
- **DTCs:** O(s) código(s) de falha(s) detectado(s).
- **Data e Hora:** Horário de cadastro.
- **Mais info:** Link para detalhes adicionais sobre o(s) DTC(s) específico(s).

5.2.4 Tela associar veículo

Quando as informações de diagnóstico, como o VIN (*Vehicle Identification Number*) e os DTCs, são extraídas via ESP32 e ELM327 mas ainda não foram associadas a um veículo existente na base de dados, o sistema disponibiliza uma interface para realizar essa associação. A Figura 17 ilustra esta tela.

No topo da página, são exibidas as informações do chassi (‘Chassi‘), os códigos de diagnóstico de falha (‘DTCs encontrados‘), e uma indicação caso a placa ainda não esteja associada (‘Placa: Sem placa associada‘).

A interface apresenta um campo para ‘Placa Selecionada‘, que será preenchido após a escolha do usuário, e um campo de busca (‘Buscar veículo por placa, marca ou modelo‘) para facilitar a localização do veículo desejado. A tabela principal lista os veículos já cadastrados no sistema com as colunas:

- **Placa:** Identificador alfanumérico do veículo.
- **Marca:** Fabricante do veículo.
- **Modelo:** Designação comercial do veículo.
- **Ano:** Ano de fabricação do veículo.
- **Ação:** Um botão "Selecionar"que permite vincular os DTCs e o VIN ao veículo correspondente.

Essa funcionalidade garante que todos os dados de diagnóstico sejam corretamente relacionados aos veículos, mantendo a integridade e organização das informações no sistema.

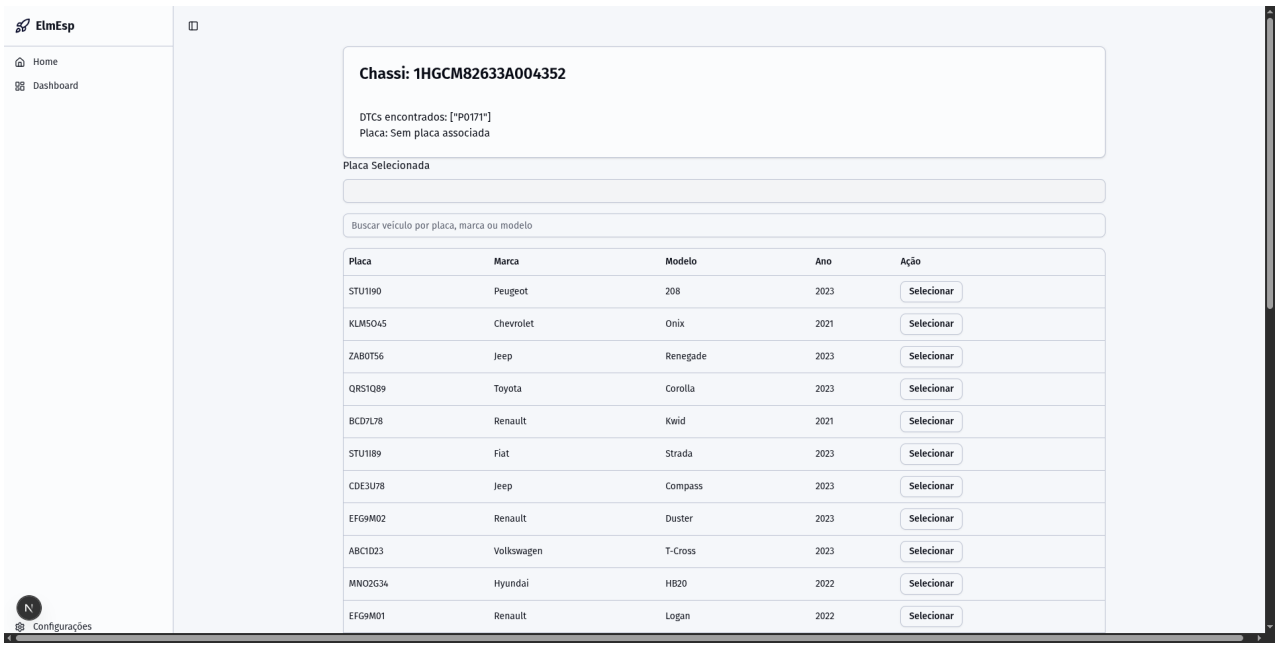

Figura 17 – Tela associar veículo

5.2.5 Tela mais informações

A funcionalidade "Mais info", acessível através do link correspondente na tabela de DTCs, direciona o usuário para uma página dedicada à análise aprofundada dos códigos de falha. Durante o processamento da requisição e a obtenção da resposta do modelo de Inteligência Artificial (IA), uma tela de carregamento é exibida, conforme ilustrado na Figura 18.

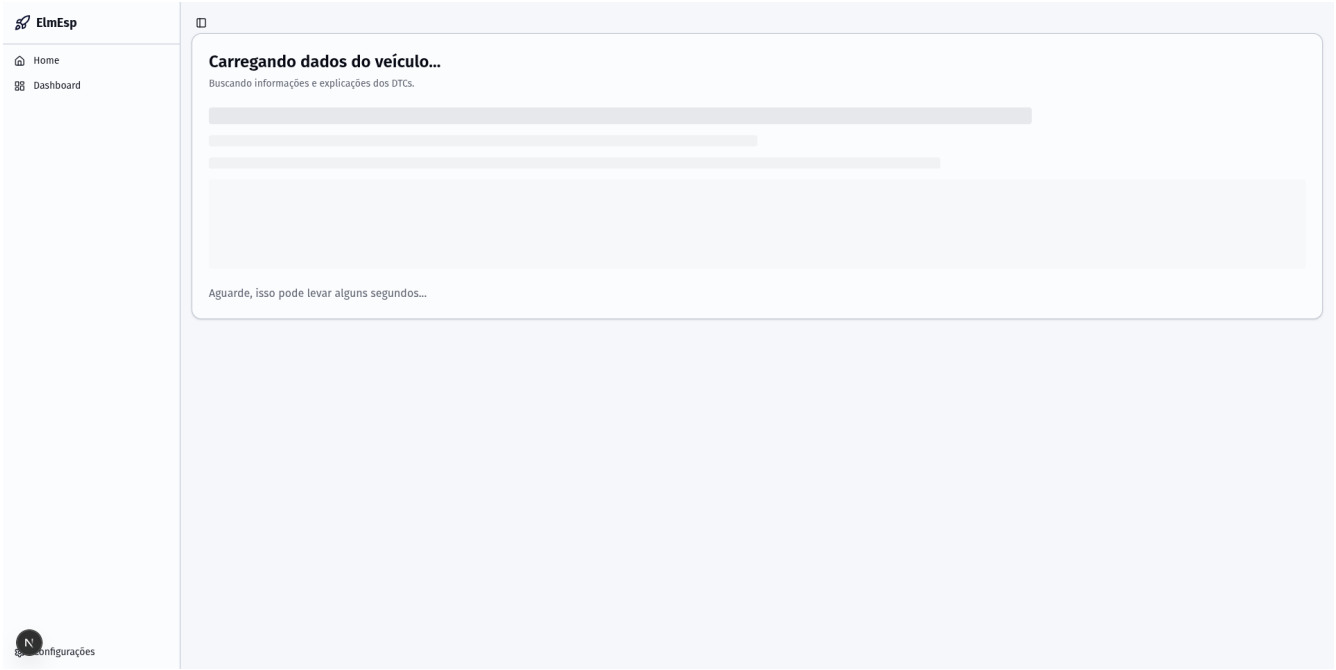

Figura 18 – Aguardando resposta da IA

Conforme ilustrado na Figura 19, a página final utiliza os dados do veículo associado (Marca, Modelo, Ano) e os códigos DTCs detectados para construir uma requisição a um modelo de Inteligência Artificial (IA).

A resposta gerada pela IA é apresentada ao usuário na tela, fornecendo informações detalhadas que visam auxiliar no diagnóstico e resolução do problema. O conteúdo da resposta tipicamente inclui:

- O significado do código de diagnóstico de falha (DTC).
- A identificação das possíveis causas subjacentes ao problema.
- Sugestões de soluções ou ações corretivas.

Esta funcionalidade integra a capacidade de processamento de linguagem natural da IA para oferecer um suporte inteligente na interpretação dos dados de diagnóstico veicular.

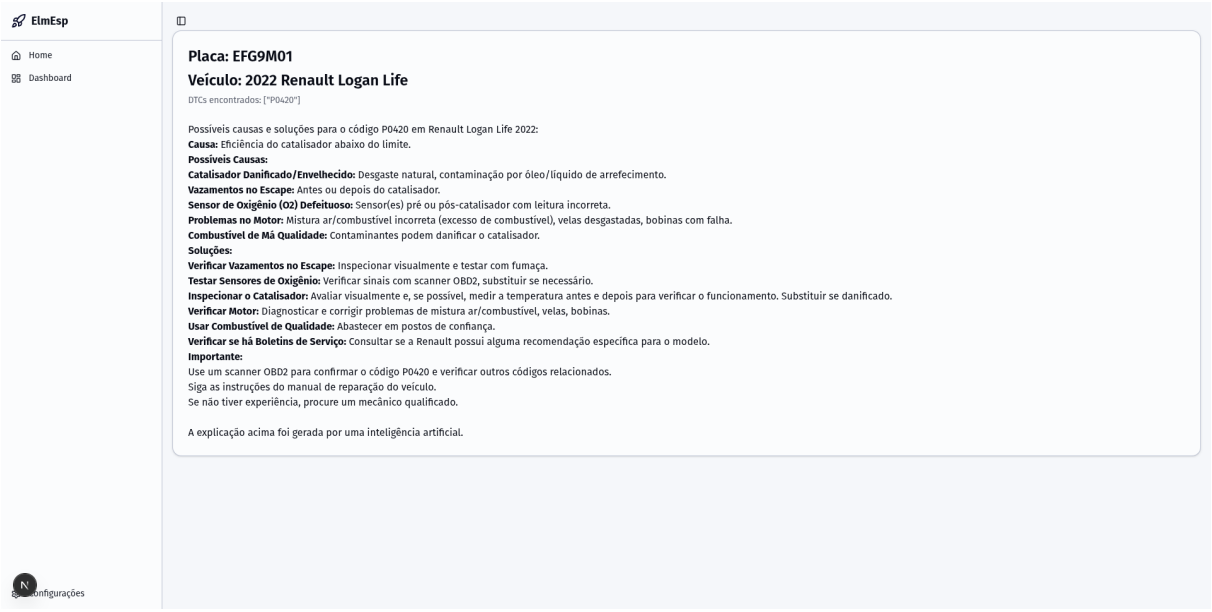

Figura 19 – Resposta da IA

5.2.6 Tela formulário novo veículo

A Figura 20 ilustra a interface para "Adicionar Novo Veículo". Esta página é acessada através do botão correspondente no *Dashboard* e permite o registro de veículos na base de dados do sistema. O formulário de entrada de dados requer as seguintes informações:

- **Placa:** Placa do veículo.
- **Marca:** Fabricante do veículo, selecionável por meio de uma lista.
- **Modelo:** Modelo do veículo.
- **Versão:** Configuração específica do modelo.
- **Ano:** Ano de fabricação do veículo, selecionável por meio de uma lista.
- **Tipo:** Categoria do veículo (e.g., carro, caminhonete), selecionável por meio de uma lista.
- **Motor:** Descrição da motorização do veículo.

Antes do envio dos dados, o sistema realiza a validação das informações inseridas no formulário para garantir a integridade e correção dos registros. Após o preenchimento e validação, o botão "Adicionar Veículo"finaliza o registro, inserindo os dados na base de dados e tornando o veículo visível na tabela de Veículos Cadastrados na página principal.

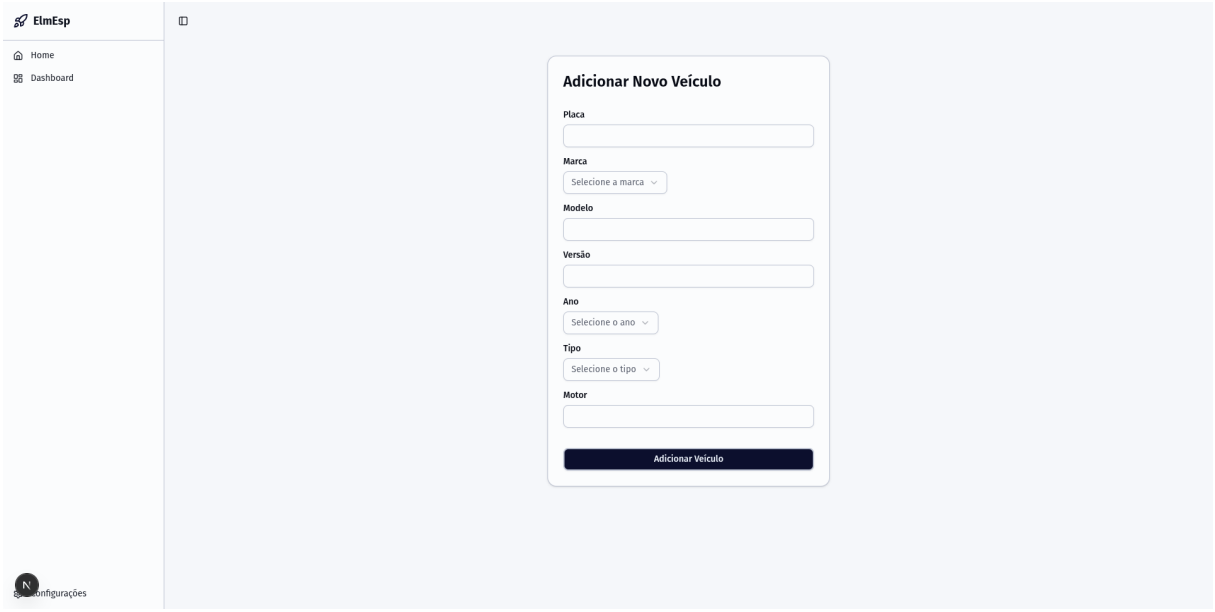

Figura 20 – Formulário novo veículo

5.3 *Dashboard* de Análise e Estatísticas

Além da tela principal de gestão de veículos e DTCs, o sistema inclui um dashboard dedicado à análise de dados e estatísticas, conforme exibido na Figura 21. Esta interface oferece uma visão consolidada de tendências e volumes de dados.

O *dashboard* é composto pelos seguintes elementos visuais:

- **DTCs por Marca:** Um gráfico de pizza que apresenta a distribuição dos códigos de diagnóstico de falha por fabricante de veículo. Isso permite identificar quais marcas geram maior volume de ocorrências de DTCs.
- **DTCs por Modelo:** Similarmente, um gráfico de pizza que exibe a distribuição dos DTCs por modelo de veículo, fornecendo insights sobre a incidência de falhas em modelos específicos.
- **Total de Registros:** Um contador que exibe o número total de registros de DTCs processados pelo sistema até o momento.

Na parte inferior da tela, observa-se um espaço destinado a um gráfico de linha. É importante salientar que este componente, atualmente intitulado "*Total Visitors*", representa um elemento de demonstração e não está funcionalmente implementado nesta versão do sistema. Ele ilustra o potencial para futuras expansões, como a visualização do número de DTCs coletados ao longo do tempo (por exemplo, meses ou semanas), permitindo uma análise de tendências de diagnóstico em um período específico.

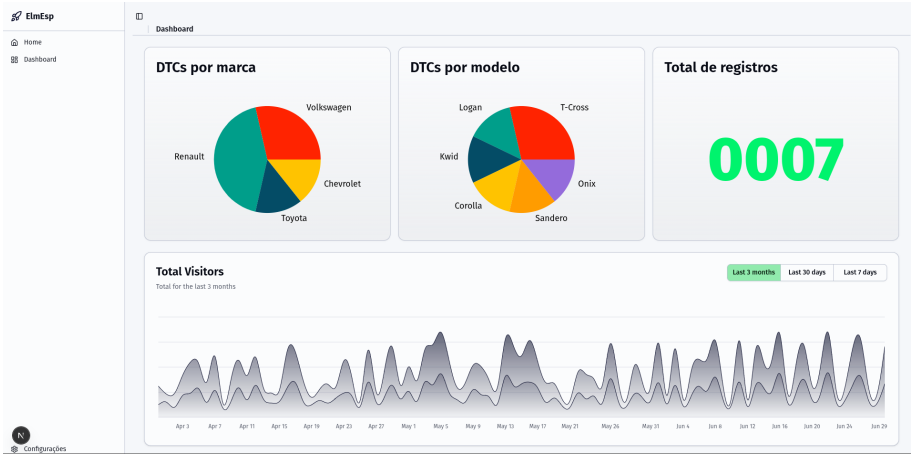

Figura 21 – Dashboard

5.4 Tela de Configurações: Importação de Dados de Veículos

A interface de configurações do sistema, acessível através do menu lateral, inclui uma funcionalidade dedicada à importação em massa de dados de veículos. Esta seção permite que o usuário carregue informações de veículos para a base de dados do sistema por meio de um arquivo no formato CSV (*Comma Separated Values*).

O objetivo principal desta tela é agilizar o processo de cadastramento de múltiplos veículos, evitando a inserção manual individual. A funcionalidade é projetada para receber um arquivo CSV que contenha os atributos dos veículos (e.g., placa, marca, modelo, ano, motor) em um formato estruturado, garantindo a integridade dos dados durante a importação para o sistema.

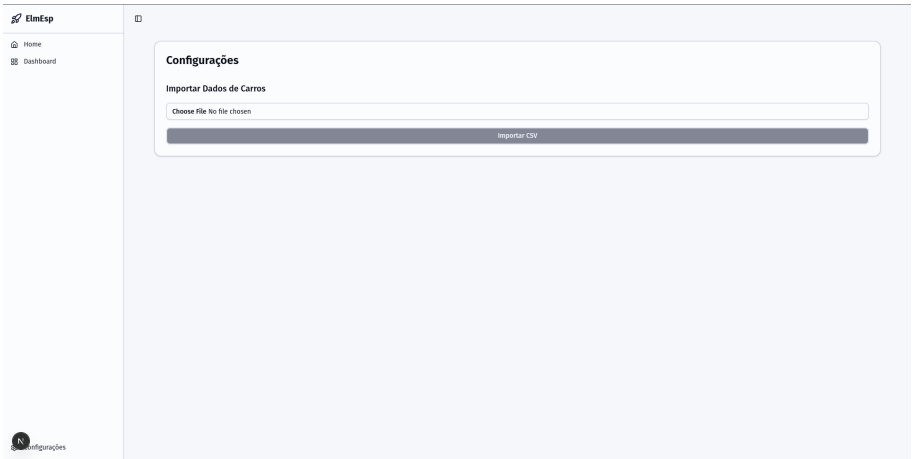

Figura 22 – Tela configurações

5.5 Custo Total

Para realização do experimento, foram adquiridos alguns componentes essenciais cujo o custo está detalhado na Tabela 4. Cabe ressaltar que outras ferramentas de apoio utilizadas ao longo do processo, como automóveis e aparelhos eletrônicos, não foram incluídas no cálculo por apresentar valores que extrapolam o escopo da proposta ou por não representarem custos diretos do projeto.

Tabela 4 – Custo total do experimento

Item	**Quantidade**	**Custo Por Unidade (R$)**
ESP32 Wi-Fi	1	46,45
Módulo ELM327	1	50,00
Cabo USB Micro	1	15,11
Total	3	**111,56**

Fonte: Autoria Própria (2025)

5.6 Possíveis Melhorias

Embora o protótipo tenha se mostrado funcional e tenha cumprido seu objetivo de validar o fluxo proposto, algumas melhorias podem ser implementadas futuramente para ampliar a robustez, escalabilidade e compatibilidade do sistema.

5.6.1 Otimização do processo de *discovery Bluetooth*

Atualmente, o pareamento com o módulo ELM327 é feito diretamente via endereço MAC, o que limita a flexibilidade do sistema e dificulta a adaptação a diferentes contextos. A implementação de um processo de descoberta mais dinâmico poderá facilitar o uso em dispositivos distintos e tornar a solução mais genérica.

5.6.2 Sistema de redundância para escolha de protocolo OBD-II

Durante os testes foi observado que o veículo Chevrolet Onix não se configura automaticamente com o ELM327, diferentemente do modelo Renault Sandero. Isso evidencia a necessidade de um sistema de redundância para escolha de protocolos OBD-II, de modo a permitir a identificação manual de protocolos alternativos caso a configuração padrão falhe.

5.6.3 Melhoria da infraestrutura e migração para a nuvem

Atualmente, todo o ambiente está operando localmente, voltada unicamente pra validação do fluxo. Para ambientes produtivos é necessário o deploy em uma infraestrutura

robusta, preferencialmente uma baseada em nuvem que ofereça elasticidade, continuidade operacional e maior confiabilidade. Esta migração também deve contemplar a implantação de um broker MQTT privado, permitindo maior controle sobre o tráfego das mensagens, melhor desempenho em aplicações distribuídas e aumento da segurança da comunicação entre dispositivos embarcados e os serviços do *backend*.

5.6.4 Integração com bases oficiais de dados veiculares

A automatização da associação entre o código *Vehicle Identification Number*(VIN) e a placa do veículo pode ser significativamente aprimorada com a integração à base de dados do SERPRO (Serviço Federal de Processamento de Dados). Embora atualmente essa base seja de acesso restrito, sua eventual liberação mediante convênios ou autorização oficial permitiria a validação automática dos dados recebidos, conferindo maior confiabilidade à aplicação e ampliando seu uso em contextos empresariais e governamentais.

5.6.5 Separação de dados para diferentes oficinas

Para um futuro modelo de negócio escalável e seguro, é crucial implementar a separação de dados por oficina. Para a viabilização deste produto, a API foi implementada de forma a cadastrar todos os dados de manutenção sem nenhuma maneira de diferenciação.

Isso não só garante a privacidade e a segurança das informações sensíveis de cada oficina, como também abre a porta para que as oficinas possam utilizar a API e o banco de dados para armazenar e gerenciar seu histórico de manutenções.

5.6.6 Salvar as respostas da inteligência artificial

Para otimizar a performance e reduzir custos, é fundamental implementar um sistema de cache para as respostas geradas pela inteligência artificial (IA). Atualmente, a API está gerando novas respostas para as mesmas combinações de carro e código de falha (DTC) a cada requisição.

Ao salvar as respostas da IA para cada combinação única de veículo e DTC, são evitadas chamadas repetidas e desnecessárias. Isso não só diminui significativamente o consumo de *tokens*, resultando em economia de custos, mas também acelera o tempo de resposta da API, oferecendo uma experiência mais fluida e eficiente para o usuário.

6 Conclusão

Este trabalho teve como objetivo o desenvolvimento e a validação de um sistema capaz de coletar códigos de falha veicular (DTCs) por meio do módulo ELM327 conectado a um ESP32, transmitindo essas informações via protocolo MQTT e armazenando-as para posterior associação, visualização e análise por meio de uma aplicação intermediária e um banco relacional. A proposta visa oferecer uma solução que possa integrar dados de diagnóstico automotivo a sistemas de gestão, como ERPs de oficinas mecânicas ou aplicações móveis voltadas para motoristas e técnicos.

Com relação aos objetivos específicos:

- Foi implementado com sucesso um algoritmo em linguagem C no ESP32 para comunicação via *Bluetooth* com o ELM327, realizando leituras de DTCs da ECU por meio do protocolo OBD-II.
- Os dados coletados foram devidamente estruturados em formato JSON e transmitidos via MQTT, validando a integridade do fluxo de dados na pipeline embarcada.
- Foi desenvolvida uma base de dados relacional e um aplicativo intermediário para consumo, armazenamento e organização das informações recebidas para síntese de base de conhecimento histórico.
- O uso de inteligência artificial foi implementado de forma experimental, com prompts baseados nos dados recebidos para sugerir soluções associadas aos códigos de falhas.
- Um dashboard foi criado, permitindo a visualização dos dados.
- Validação do sistema foi feita com veículos reais, confirmando a consistência dos dados ao longo da *pipeline*, apesar de algumas limitações técnicas.

A infraestrutura atual, baseada em execução local, atendeu ao propósito experimental, mas não representa um ambiente adequado para implantação em escala. A proposta de migração para uma infraestrutura em nuvem, incluindo o *deploy* de um *broker* MQTT privado, surge como uma alternativa essencial para garantir a plena operação do sistema.

Conclui-se, portanto, que o sistema desenvolvido apresenta viabilidade técnica e pode servir como base para soluções voltadas à manutenção preditiva e diagnóstico veicular remoto. Com as melhorias propostas espera-se aumentar a robustez, a escalabilidade e a aplicabilidade do sistema em contextos reais, contribuindo para a transformação digital do setor automotivo e reparação veicular.

Referências

AURUM, A.; WOHLIN, C. Aligning requirements with business objectives: A framework for requirements engineering decisions. *Proceedings Requirements Engineering Decision Support Workshop*, 2005. Citado na página 47.

BANKS, A.; GUPTA, R. *MQTT Version 3.1.1.* 2014. OASIS Standard. Edited by Andrew Banks and Rahul Gupta. Disponível em: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>. Citado 2 vezes nas páginas 29 e 30.

BJARNASON, E.; LANG, F.; MJÖBERG, A. An empirically based model of software prototyping: a mapping study and a multi-case study. *Empirical Software Engineering*, Springer, v. 28, n. 115, p. 1–54, August 2023. Disponível em: <https://doi.org/10.1007/s10664-023-10331-w>. Citado na página 52.

BRANCH, E. *Trial Error: A Method for Effective Solutions.* 2024. Web blog. Disponível em: <https://www.iienstitu.com/en/blog/trial-error>. Citado na página 39.

COHN, M. *User Stories Applied: For Agile Software Development.* [S.l.]: Addison-Wesley, 2004. Citado na página 47.

CORRIGAN, S. *Introduction to the Controller Area Network (CAN).* [S.l.], 2002. Disponível em: <https://masters.donntu.ru/2005/fvti/trofunenko/library/sloa101.pdf>. Citado na página 26.

CROMER, G. C. et al. *Encyclopedia Britannica*, 2024. Disponível em: <https://www.britannica.com/technology/automobile>. Citado na página 21.

CUATTO, T. et al. A case study in embedded system design: an engine control unit. *35th Design Automation Conference*, 1998. Disponível em: <https://dl.acm.org/doi/abs/10.1145/277044.277248>. Citado 2 vezes nas páginas 21 e 25.

ELECTRONICS, E. *ELM327.* [S.l.], 2024. Disponível em: <https://web.archive.org/web/20210308171217/https://www.elmelectronics.com/wp-content/uploads/2020/05/ELM327DSL.pdf>. Citado 2 vezes nas páginas 39 e 40.

ELMASRI, S. B. N. R. *Sistemas de Banco de Dados.* [S.l.]: Pearson, 2018. ISBN 978-85-430-2500-1. Citado na página 30.

ESPRESSIF. *Official IoT Development Framework.* 2024. Website article. Disponível em: <https://www.espressif.com/en/products/sdks/esp-idf>. Citado na página 43.

FONTELLES, M. J. et al. Metodologia da pesquisa científica: Diretrizes para a elaboração de um protocolo de pesquisa. *SCIENTIFIC RESEARCH METHODOLOGY: GUIDELINES FOR ELABORATION OF A RESEARCH PROTOCOL*, Revista Paulista de Pediatria, 2009. Citado na página 32.

GEEKS, G. for. *What is MAC Address?* 2024. Website article. Disponível em: <https://www.geeksforgeeks.org/mac-address-in-computer-network/>. Citado na página 29.

GEORGE, E.; PECHT, M. Tin whisker analysis of an automotive engine control unit. *Microelectronics Reliability*, 2013. Disponível em: <https://www.sciencedirect.com/science/article/abs/pii/S0026271413003107>. Citado na página 25.

GEOTAB. *What is OBDII? History of on-board diagnostics*. 2023. Website article. Disponível em: <https://www.geotab.com/blog/obd-ii/>. Citado na página 26.

GERCHEV, I. *Tailwind CSS: Craft Beautiful, Flexible, and responsive designs*. [S.l.]: SitePoint, 2022. ISBN 978-1-925836-51-6. Citado na página 45.

HUDAIB, A. et al. Requirements prioritization techniques comparison. *Modern Applied Science*, Canadian Center of Science and Education, Amman, Jordan, v. 12, n. 2, p. 62, 2018. ISSN 1913-1844. Online Published: January 16, 2018. Disponível em: <https://doi.org/10.5539/mas.v12n2p62>. Citado na página 47.

KIOURTIS, A.; MAVROGIORGOU, A.; KYRIAZIS, D. A comparative study of bluetooth spp, pan and goep for efficient exchange of healthcare data. *Emerging Science Journal*, v. 5, n. 3, p. 279, 2021. ISSN 2610-9182. Disponível em: <https://ijournalse.org/index.php/ESJ/article/view/512/pdf>. Citado na página 28.

MANOEL. *Extreme Programming – Conceitos e Práticas*. 2006. Website article. Disponível em: <https://www.devmedia.com.br/extreme-programming-conceitos-e-praticas/1498>. Citado na página 38.

MCCORD, K. *Automotive Diagnostic Systems - Understanding OBD-I OBD-II*. CarTech/SA Design, 2011. Disponível em: <https://books.google.com.br/books?hl=pt-BR&lr=&id=kyEtsrPk9ZQC&oi=fnd&pg=PP1&dq=obd+2&ots=O6rEGjTotB&sig=7eFB5Az3GAvpnqquS2w1LAKilTg#v=onepage&q=obd%202&f=false>. Citado 3 vezes nas páginas 26, 27 e 28.

OFFICES, P. I. A. *Understanding MAC Addresses: A Beginner's Guide*. 2024. Website article. Disponível em: <https://services.pitt.edu/TDClient/33/Portal/KB/ArticleDet?ID=1773>. Citado na página 29.

OLIVEIRA, L. B. de. *Como elaborar a metodologia de um projeto de pesquisa*. 2017. Pdf document. Disponível em: <https://docente.ifsc.edu.br/luciane.oliveira/MaterialDidatico/P%C3%B3s%20Gest%C3%A3o%20Escolar/Pesquisa%20em%20Educa%C3%A7%C3%A3o/Aula%2030%20de%20agosto/Como%20elaborar%20a%20metodologia%20de%20um%20projeto%20de%20pesquisa.pdf>. Citado na página 31.

PILLMANN et al. Novel common vehicle information model (cvim) for future automotive vehicle big data marketplaces. In: *IEEE Intelligent Vehicles Symposium (IV)*. Redondo Beach, CA, USA: IEEE, 2017. Disponível em: <https://doi.org/10.48550/arXiv.1802.09353>. Citado na página 22.

Prisma ORM. *What is Prisma*. 2024. <https://www.prisma.io/docs/orm/overview/introduction/what-is-prisma>. Acesso em: 15 maio 2025. Citado na página 44.

SALEM, A. B. *eXtremme Programming: A Begginer's Guide to the Agile Method*. 2024. Web blog. Disponível em: <https://scrum-master.org/en/extreme-programming-xp-a-beginners-guide-to-the-agile-method/>. Citado na página 38.

SCHWABER, K.; SUTHERLAND, J. *The Scrum Guide*. [S.l.]: Ken Schwaber and Jeff Sutherland, 2020. Citado na página 33.

SIVAKUMAR, V. et al. Object relational mapping framework performance impact. *Turkish Journal of Computer and Mathematics Education*, Turkish Journal of Computer and Mathematics Education, v. 12, n. 7, p. 2516–2519, 2021. Received: 11 January 2021; Revised: 12 February 2021; Accepted: 27 March 2021; Published online: 16 April 2021. Citado na página 30.

SOFI, M. A. Bluetooth protocol in internet of things (iot), security challenges and a comparison with wi-fi protocol: A review. *International Journal of Engineering Research Technology (IJERT)*, 2016. Disponível em: <https://www.researchgate.net/profile/Mukhtar-Sofi/publication/311086845_Bluetooth_Protocol_in_Internet_of_Things_IoT_Security_Challenges_and_a_Comparison_with_Wi-Fi_Protocol_A_Review/links/5c4044f5a6fdccd6b5b2eca2/Bluetooth-Protocol-in-Internet-of-Things-IoT-Security-Challenges-and-a-Comparison-with-Wi-Fi-P pdf>. Citado na página 28.

STERK, F. et al. Unlocking the value from car data: A taxonomy and archetypes of connected car business models. *Electron Markets*, v. 34, n. 13, 2024. Disponível em: <https://doi.org/10.1007/s12525-024-00692-5>. Citado na página 22.

SYSTEMS, E. *ESP32-WROOM-32*. [S.l.], 2023. Disponível em: <https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf>. Citado na página 42.

Vercel. *Introduction to Next.js*. 2024. Acesso em: 15 maio 2025. Disponível em: <https://nextjs.org/docs>. Citado na página 44.

WHITE, E. *Making Embedded Systems*. [S.l.]: O'Reilly Media, 2024. Citado 2 vezes nas páginas 49 e 50.

ZHU, M.-Y. Understanding freertos: A requirement analysis. *CoreTek Syst., Inc., Beijing, China, Tech. Rep*, 2016. Disponível em: <https://www.researchgate.net/profile/Ming-Yuan-Zhu/publication/308692183_Understanding_FreeRTOS_A_Requirement_Analysis/links/57eb301108ae5d93a4816184/Understanding-FreeRTOS-A-Requirement-Analysis.pdf>. Citado na página 29.

ÇAPARLAR, C. ; DöNMEZ, A. What is scientific research and how can it be done? *Turkish Journal of Anaesthesiology Reanimation*, 2016. Disponível em: <https://pmc.ncbi.nlm.nih.gov/articles/PMC5019873/>. Citado 2 vezes nas páginas 31 e 32.