www.revistaft.com.br

Revista FT | ISSN 1678-0817 | v. 30, n. 157, 2026 | DOI: 10.69849/qh9ngv53

Desenvolvimento de um sistema IoT para gerenciamento da jornada de trabalho do motorista

Development of an IoT-based system for driver workload management

Isac de Lima Feliciano[1]; Jairo da Silva Sobrinho Junior[2]; Lucas Vinícius Teixeira de Gois[3]; Thaíse Miriele Silva Nogueira[4]; Thayná dos Santos Bittencourt[5]; Everson Mizael Cortez Silva[6]; Leonardo Gomes de Paiva Amorim[7]; Wagner de Oliveira[8]

Resumo

O presente artigo apresenta o desenvolvimento e a modernização da plataforma TechPS, uma solução integrada de *software* e *hardware* voltada para o gerenciamento logístico e controle de jornada de motoristas. O estudo visa solucionar limitações de escalabilidade de um sistema legado, propondo uma nova arquitetura que assegure a integridade dos dados e a conformidade com a legislação trabalhista vigente. A metodologia aplicada baseia-se no desenvolvimento de um dispositivo embarcado utilizando Raspberry Pi 5 e a linguagem Python, integrado a módulos de comunicação 4G, leitores RFID e biométricos para identificação segura dos condutores. Esse dispositivo é responsável por identificar o motorista e monitorar as jornadas através de uma interface touchscreen interativa e notificações por voz. Os dados são armazenados em um banco de dados local e enviados ao banco de dados da empresa por meio de uma API. Os resultados obtidos são promissores, pois demonstram a eficácia da infraestrutura proposta, com validação em andamento da comunicação entre o dispositivo IoT e o servidor, além da implementação funcional do módulo de gestão de motoristas. Conclui-se que a base tecnológica estabelecida, fundamentada em padrões de projeto modernos e validação rigorosa de dados, oferece a robustez necessária para a operação *offline* e

[1] Discente do Curso de Formação Inicial e Continuada (Curso FIC) em Residência Tecnológica em Software Embarcado, na modalidade a distância. e-mail: isaclimaf2@gmail.com
[2] Discente do Curso de Formação Inicial e Continuada (Curso FIC) em Residência Tecnológica em Software Embarcado, na modalidade a distância. e-mail: juniorsilvasobrinho1999@gmail.com
[3] Discente do Curso de Formação Inicial e Continuada (Curso FIC) em Residência Tecnológica em Software Embarcado, na modalidade a distância. e-mail: lucas.vinicius@escolar.ifrn.edu.br
[4] Discente do Curso de Formação Inicial e Continuada (Curso FIC) em Residência Tecnológica em Software Embarcado, na modalidade a distância. e-mail: thaise.nogueira.116@ufrn.edu.br
[5] Discente do Curso de Formação Inicial e Continuada (Curso FIC) em Residência Tecnológica em Software Embarcado, na modalidade a distância. e-mail: thayryu@gmail.com
[6] Docente do Curso de Formação Inicial e Continuada (Curso FIC) em Residência Tecnológica em Software Embarcado, na modalidade a distância. e-mail: everson.cortez@ifrn.edu.br
[7] Docente do Curso de Formação Inicial e Continuada (Curso FIC) em Residência Tecnológica em Software Embarcado, na modalidade a distância. e-mail: leonardo.amorim@ifrn.edu.br
[8] Docente do Curso de Formação Inicial e Continuada (Curso FIC) em Residência Tecnológica em Software Embarcado, na modalidade a distância. e-mail: wagner.oliveira@ifrn.edu.br

sincronização assíncrona, viabilizando a expansão futura das funcionalidades de gestão de jornadas.

Palavras-chave: Gestão Logística; Internet das coisas; Sistemas embarcados.

ABSTRACT

This article presents the development and modernization of the TechPS platform, an integrated software and hardware solution aimed at logistics management and driver working-hours control. The study seeks to address the scalability limitations of a legacy system by proposing a new architecture that ensures data integrity and compliance with current labor legislation. The applied methodology is based on the development of an embedded device using Raspberry Pi 5 and the Python programming language, integrated with 4G communication modules, RFID readers, and biometric devices for secure driver identification. This device is responsible for identifying the driver and monitoring working hours through an interactive touchscreen interface and voice notifications. The data are stored in a local database and sent to the company's database through an API. The obtained results are promising, as they demonstrate the effectiveness of the proposed infrastructure, with ongoing validation of communication between the IoT device and the server, in addition to the functional implementation of the driver management module. It is concluded that the established technological foundation, based on modern design patterns and rigorous data validation, provides the robustness required for offline operation and asynchronous synchronization, enabling the future expansion of working-hours management functionalities.

Keywords: Logistics Management; Internet of Things; Embedded Systems

1 INTRODUÇÃO

A gestão logística e o controle de jornada de trabalho no setor de transportes têm passado por uma transformação significativa com o avanço da Internet das Coisas (IoT) e dos Sistemas de Informação Corporativos. A necessidade de monitoramento preciso das atividades de transporte, aliada à exigência de conformidade com legislações trabalhistas rigorosas, demanda soluções tecnológicas que ultrapassem o simples rastreamento veicular. Nesse

cenário, a integração entre dispositivos móveis embarcados e sistemas centrais de gerenciamento torna-se um requisito fundamental para garantir a confiabilidade e a integridade das informações operacionais.

Entretanto, o desenvolvimento de tais soluções enfrenta desafios técnicos consideráveis, especialmente no que tange à comunicação segura e eficiente entre o *hardware* em campo e os servidores de processamento. A arquitetura de sistemas antigos muitas vezes carece de escalabilidade e não suporta as demandas modernas de validação de dados e operação *offline*. A complexidade aumenta ao se considerar a necessidade de interfaces amigáveis para os motoristas e a robustez necessária para operar em ambientes veiculares, exigindo uma separação clara de responsabilidades entre as camadas de apresentação, regras de negócio e persistência de dados.

A presente pesquisa justifica-se pela necessidade de modernização da infraestrutura tecnológica de monitoramento de jornada utilizada pela empresa TECH PS, adotando uma abordagem baseada em *Single Board Computers* (SBC) e arquitetura de microsserviços. A escolha por uma API RESTful para o *backend* fundamenta-se na promoção da separação de responsabilidades, facilitando o desenvolvimento paralelo e a integração, conceito defendido por Pressman e Maxim (2016) como essencial para a manutenibilidade de *softwares* complexos. Além disso, a arquitetura modular proposta, tanto no *hardware* quanto no *software*, alinha-se aos princípios de Sommerville (2011), permitindo que evoluções na interface do usuário ocorram sem impactar a lógica de comunicação ou o banco de dados.

O objetivo geral deste trabalho é desenvolver uma solução integrada de IoT para o controle de jornada de motoristas, composta por um sistema embarcado inteligente e um *webservice* para recepção e validação de dados. A solução busca garantir a integridade das informações desde a coleta biométrica no veículo até a persistência no banco de dados relacional.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 INTERNET DAS COISAS (IOT) E SISTEMAS EMBARCADOS NA LOGÍSTICA

A Internet das Coisas (IoT) refere-se à interconexão digital de objetos cotidianos com a internet, permitindo que esses objetos coletem e troquem dados. No contexto da logística e transporte, o uso de IoT permite o monitoramento em tempo real de frotas, cargas e, crucialmente, do comportamento e jornada dos condutores (Kurose; Ross, 2013).

Diferente de computadores de uso geral, os sistemas embarcados são projetados para realizar tarefas dedicadas, geralmente com restrições de tempo real, consumo de energia e tamanho. No cenário de controle de jornada, um sistema embarcado atua como o ponto de borda (edge computing), processando dados críticos localmente antes de enviá-los para a nuvem, garantindo operação mesmo em zonas de sombra de conectividade (Tanenbaum, 2013).

2.2 COMPUTADORES DE PLACA ÚNICA (SBC)

Para a implementação de sistemas embarcados que exigem maior poder de processamento, como o manuseio de múltiplos periféricos e conexão de rede robusta, utilizam-se os Computadores de Placa Única (SBC - Single Board Computers). Um SBC é um computador completo construído em uma única placa de circuito, contendo microprocessador, memória, entrada/saída (I/O) e outros recursos funcionais (Monk, 2013).

A plataforma Raspberry Pi, utilizada neste trabalho, exemplifica essa tecnologia. Ela oferece a capacidade de rodar sistemas operacionais completos (baseados em Linux), o que facilita a integração com leitores biométricos e módulos 4G através de linguagens de alto nível como Python, superando as limitações de microcontroladores tradicionais em tarefas de processamento de dados mais complexos (Upton; Halfacree, 2013).

2.3 TECNOLOGIAS DE IDENTIFICAÇÃO E SEGURANÇA DE DADOS

A Lei nº 13.103, de 2 de março de 2015, conhecida como “Lei do Motorista”, estabelece normas específicas para o exercício da profissão de motorista profissional, disciplinando aspectos como jornada de trabalho, tempo de direção e períodos de descanso. Entre suas disposições, destaca-se a obrigatoriedade de controle fidedigno da jornada, mediante registros em diário de bordo, papeleta ou sistemas eletrônicos instalados nos veículos (BRASIL, 2015). Nesse contexto, a legislação reforça a necessidade de mecanismos que assegurem a identificação inequívoca do condutor, de modo a garantir a veracidade dos registros e a responsabilização adequada pelo cumprimento das normas de direção e descanso.

Em conformidade com a lei nº 13.103/2025, a validação inequívoca da identidade do condutor é um requisito legal e indispensável para o controle de jornada. Duas tecnologias principais suportam essa função. A primeira é o RFID (Radio Frequency Identification), que utiliza campos eletromagnéticos para transferir dados automaticamente, identificando e

rastreando etiquetas (tags) anexadas a objetos ou cartões. É uma tecnologia madura, amplamente utilizada para controle de acesso rápido (Finkenzeller, 2010).

A segunda tecnologia é a Biometria, que se refere à identificação de humanos por suas características ou traços. A leitura de impressão digital oferece um nível superior de segurança e não-repúdio em comparação a senhas ou cartões, garantindo que o motorista registrado é quem está, de fato, operando o veículo (Stallings, 2015).

2.4 ARQUITETURA DE BACKEND E APIS RESTFUL

Para centralizar os dados coletados pelos dispositivos embarcados, adota-se uma arquitetura cliente-servidor baseada em APIs (Application Programming Interfaces). O estilo arquitetural REST (Representational State Transfer) constitui o padrão predominante para serviços web, fundamentando-se em princípios como a ausência de estado (statelessness) e o uso do formato JSON (JavaScript Object Notation) para a troca de dados, o que proporciona leveza e eficiência na transmissão sobre redes móveis (Sommerville, 2019).

No lado do servidor, a implementação do backend é realizada em PHP puro, sem o emprego de frameworks. Essa decisão decorre do fato de o sistema já estar previamente desenvolvido nessa linguagem, exigindo a incorporação das novas funcionalidades sobre uma base existente, conforme diretrizes institucionais da empresa responsável pelo projeto. Mesmo sem o uso de um framework formal, são aplicados conceitos consolidados de separação de responsabilidades, organização modular do código e camadas lógicas para acesso a dados e regras de negócio. Tal abordagem possibilita a manutenção da segurança, a legibilidade do sistema e a evolução gradual da aplicação, preservando sua estabilidade e garantindo condições adequadas de escalabilidade e manutenção (Pressman; Maxim, 2021).

2.5 MÉTODOS HTTP EM APIS RESTFUL

As APIs RESTful utilizam os métodos definidos pelo protocolo HTTP como forma padronizada de interação entre cliente e servidor. Cada método representa semanticamente uma operação sobre um recurso, promovendo clareza, previsibilidade e padronização na comunicação entre sistemas distribuídos (Fielding, 2000).

Os principais métodos empregados são:

- GET: utilizado para a recuperação de informações, sem provocar alterações no estado do servidor;
- POST: destinado à criação de novos recursos;
- PUT: empregado para atualização completa de um recurso existente;
- PATCH: utilizado para atualizações parciais;
- DELETE: responsável pela remoção de recursos.

Essa padronização favorece a interoperabilidade entre diferentes plataformas e linguagens, permitindo que clientes heterogêneos — como aplicações web, móveis ou dispositivos embarcados — interajam com o mesmo serviço de forma uniforme. Além disso, a correta utilização desses métodos contribui para a previsibilidade do comportamento do sistema e para a adoção de boas práticas de engenharia de software em ambientes distribuídos (Sommerville, 2019).

2.6 SEGURANÇA EM APIs RESTFUL

A segurança constitui um aspecto crítico em sistemas baseados em APIs, especialmente quando operam sobre redes públicas ou móveis. A exposição de serviços exige mecanismos capazes de garantir autenticidade, integridade e confidencialidade das informações transmitidas.

Entre as práticas mais comuns, destacam-se:

- Autenticação por tokens (como JWT – JSON Web Token);
- Validação rigorosa de entradas para evitar injeções de código;
- Utilização de protocolos seguros, como HTTPS, para criptografia do tráfego.

O JSON Web Token (JWT) é um padrão aberto amplamente utilizado para autenticação e autorização em sistemas distribuídos. Ele consiste em um token compacto e auto contido, codificado em formato JSON, que armazena informações sobre a identidade do usuário e seus privilégios de acesso. Por ser transmitido a cada requisição, o JWT permite que o servidor valide o cliente sem manter estado entre as comunicações, alinhando-se ao princípio de statelessness das APIs RESTful (Fielding, 2000; Sommerville, 2019).

Mesmo em implementações sem frameworks, é possível adotar tais mecanismos por meio de bibliotecas e rotinas próprias, mantendo a integridade do sistema. Segundo Pressman e Maxim (2021), a incorporação de práticas de segurança desde as camadas iniciais do projeto reduz significativamente vulnerabilidades e custos futuros de manutenção.

Dessa forma, a segurança em APIs RESTful não deve ser tratada como um componente adicional, mas como parte integrante da arquitetura do backend, assegurando confiabilidade e robustez ao sistema.

2.7 CONTEINERIZAÇÃO E AMBIENTES DE DESENVOLVIMENTO COM DOCKER

A conteinerização consiste em uma técnica de virtualização em nível de sistema operacional, na qual aplicações e suas dependências são empacotadas em unidades isoladas denominadas contêineres. Diferentemente das máquinas virtuais tradicionais, os contêineres compartilham o mesmo núcleo do sistema operacional hospedeiro, tornando-se mais leves e eficientes em termos de consumo de recursos (Merkel, 2014).

O Docker é a plataforma mais difundida para a implementação dessa abordagem, sendo amplamente adotado para padronização de ambientes de desenvolvimento. Por meio de arquivos de configuração declarativos, como o Dockerfile, é possível definir de forma precisa as dependências necessárias ao funcionamento da aplicação, assegurando que todos os desenvolvedores utilizem um ambiente homogêneo, independentemente do sistema operacional utilizado (Pahl, 2015).

No contexto deste trabalho, o Docker é empregado exclusivamente no ambiente de desenvolvimento local, não sendo utilizado como mecanismo de implantação na produção. Sua função é garantir a paridade entre os ambientes dos desenvolvedores, reduzindo inconsistências relacionadas a versões de bibliotecas, servidores web ou configurações do sistema operacional.

Essa abordagem contribui para a reprodutibilidade do ambiente de testes, facilita a integração entre os membros da equipe e minimiza problemas decorrentes de divergências de configuração, sem interferir na infraestrutura já consolidada do sistema em produção. Dessa forma, a conteinerização atua como um suporte metodológico ao desenvolvimento, preservando a arquitetura existente e assegurando maior confiabilidade durante a implementação das novas funcionalidades.

3 METODOLOGIA

Este capítulo descreve os procedimentos, ferramentas e a arquitetura específica utilizada na implementação do sistema, detalhando como os conceitos teóricos foram aplicados no hardware e software do projeto. Vale salientar que a equipe foi dividida em duas: uma

responsável pelo software e hardware embarcado e outra responsável pela conexão deste dispositivo com o banco de dados da empresa, por meio de uma API.

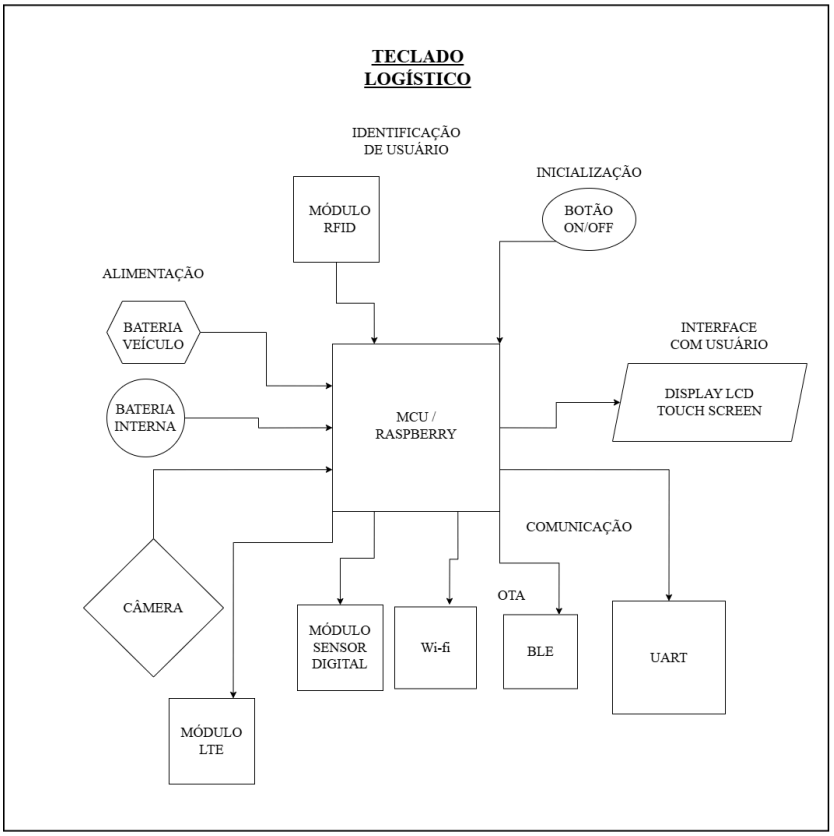
3.1 ETAPAS INICIAIS

Anteriormente ao desenvolvimento do sistema, realizou-se um estudo com a finalidade de se alcançar o entendimento do problema. Esse estudo aconteceu por meio da consulta às necessidades que foram apresentadas pelo *Product Owner* (PO) em reuniões e manuais de sistemas parecidos da marca Teltonika e Positron. Com isso, identificou-se a necessidade de um dispositivo que tornasse o acesso ao sistema de gerenciamento de jornada da empresa mais acessível e confiável aos motoristas.

Dessa forma, fez-se um levantamento de requisitos para o sistema e destacaram-se as seguintes necessidades. Primeiramente, o dispositivo deve ser um teclado logístico, que é um dispositivo já conhecido no mercado, e que consiste num display com botões usado para gerenciar a jornada de motoristas. Em segundo, ele deve ter a tela touchscreen, sendo esse um diferencial com relação aos dispositivos do mercado. Em terceiro, para realizar a autenticação dos motoristas, foi idealizada uma autenticação em dois fatores, utilizando de uma tag RFID e da digital do motorista. Em quarto, a comunicação do dispositivo deve ocorrer via BLE, Wi-Fi e LTE, por meio de um chip SIM.

Essas tecnologias de comunicação foram escolhidas pois permitem ao dispositivo ter acesso à rede para atualização e envio de dados em diversas situações. Também pensou-se que ele deveria ter uma câmera e comunicação com o barramento CAN do veículo. Mas, essas últimas especificações ficaram como possíveis adições futuras devido ao tempo que havia para o desenvolvimento do trabalho e visando ter algum produto funcional ao final. Na Figura 1, observa-se um diagrama que representa todos os pontos aqui levantados.

Figura 1: Diagrama do Sistema



Fonte: Autores.

Como pode ser observado (Figura 1), a alimentação do sistema foi planejada para ocorrer por meio de uma bateria interna do dispositivo e por meio da bateria do veículo.

Após o levantamento de requisitos, foi feita uma lista de componentes e, posteriormente, prosseguiu-se para a compra. Em seguida, foi iniciado o desenvolvimento do projeto.

3.2 DESENVOLVIMENTO DO SISTEMA EMBARCADO

Para o desenvolvimento do sistema embarcado, a equipe foi dividida em duas frentes de trabalho. A primeira ficou responsável pelo desenvolvimento da interface gráfica de usuário (GUI) e pela lógica de funcionamento do sistema, enquanto a segunda concentrou-se na integração dos módulos de biometria, RFID e comunicação LTE 4G com a Raspberry Pi 5, além do desenvolvimento do código responsável pelo funcionamento desses módulos no sistema. Quanto ao desenvolvimento do código, optou-se pelo uso do *Python* em conjunto com

o uso de um ambiente virtual, a fim de garantir maior controle das dependências e padronização do processo de implementação.

3.2.1 Desenvolvimento da interface gráfica de usuário

A interface GUI foi desenvolvida utilizando-se o módulo PyQt6 da linguagem *Python.* O sistema foi estruturado e dividido em seis módulos distintos, cada módulo possui uma responsabilidade única. A divisão em módulos tem como objetivo garantir um código mais limpo e manutenível. A comunicação entre os módulos é feita de forma controlada, evitando-se a dependência direta entre os componentes e assegurando independência das implementações internas de cada um. Os seis módulos criados são os seguintes.

O primeiro módulo, denominado *Interface*, tem a função de gerir a interface GUI. Ele é composto por classes PyQt que definem as janelas, botões e outros componentes visuais sem conter a lógica de negócio. Quando um botão é clicado, ele emite um sinal que é tratado por outro componente.

O segundo módulo é o *núcleo.* Sua função é gerir toda a aplicação, agindo como intermédio entre a interface, o hardware e os dados. Ele é composto de uma classe principal, GerenciadoJornada, que recebe os sinais da interface, os eventos do hardware e toma as decisões de negócio.

Como terceiro módulo, tem-se o *hardware.* A sua função é isolar toda a complexidade da comunicação com os periféricos físicos. Ele é composto de classes como: Leitor*RFID*, *LeitorLBIO,* que são responsáveis por gerir os módulos externos. Dessa forma, o módulo abstrai os detalhes de implementação dos periféricos, fornecendo uma interface simplificada para os demais componentes do sistema.

O quarto módulo, denominado *dados*, é responsável por gerenciar o armazenamento e a recuperação dos dados no banco local. Optou-se pelo uso do SQLite devido à sua leveza, simplicidade e compatibilidade com projetos embarcados, eliminando a necessidade de um servidor dedicado. Esse módulo é composto por funções específicas para salvar e buscar eventos no banco de dados, sendo acessado exclusivamente pelo núcleo.

O quinto e sexto módulos são: *serviços e utilitários.* Eles são responsáveis por gerir toda a comunicação com a API web externa e armazenar código e dados que são usados em várias partes do sistema, respectivamente.

Por fim, o código é composto por um arquivo *main* que é responsável por inicializar toda a aplicação e criar o seu loop de execução.

3.2.2 Desenvolvimento do hardware do sistema

O desenvolvimento do hardware do sistema foi conduzido por meio do estudo da documentação técnica de cada um dos módulos selecionados para compor o projeto, seguido da análise e integração individual de cada um dos módulos com a Raspberry Pi. Para viabilizar a integração dos módulos com o sistema embarcado, foram estudadas API's em Python, que haviam sido desenvolvidas pela comunidade, e tinham como função permitir o controle dos módulos pela raspberry ou qualquer sistema programável nessa linguagem.

O módulo RFID selecionado para o projeto foi o RDM 6300. Ele é um módulo RFID de baixa potência, compacto e econômico. Ele é capaz de realizar a leitura, não a escrita, de cartões e tags no padrão EM4100. Sua comunicação com o SBC se dá através de uma interface UART. Durante a operação de leitura, o módulo envia um pacote de dados composto por 14 Bytes em HEX. Eles são divididos em: um Bytes de preâmbulo, dez Bytes de dados, dois Bytes de Checksum, e um Bytes de Final. Para realizar o controle do módulo por meio da Raspberry Pi, fez uso de um módulo em Python chamado *RDM6300.*

O módulo biométrico utilizado é o SFM-V1.7, do tipo capacitivo. Ele possui uma arquitetura própria embarcada que realiza a leitura, processamento e armazenamento das digitais de forma local. Essa característica garante a redução do processamento exigido da Raspberry Pi, otimizando o desempenho geral do sistema. Para seu controle, foi utilizado o módulo em Python, *Adafruit_fingerprint.*

O módulo LTE escolhido é o Lilygo Ttgo T-call A7670e 4g Esp32. Esse módulo possui comunicação LTE nos padrões europeus e possui GPS integrado. No estado atual do projeto, a sua integração à placa da Raspberry ainda não foi concluída, uma vez que sua documentação técnica se encontra em fase de estudo.

3.3 DESENVOLVIMENTO DA API E INTEGRAÇÃO COM BANCO DE DADOS

A construção do backend adota uma abordagem incremental, onde a infraestrutura base é estabelecida priorizando a segurança e a autenticação, para posteriormente expandir os serviços voltados ao hardware (IoT). O processo metodológico divide-se nas seguintes etapas de implementação:

3.3.1 Tecnologias e Arquitetura Base

A API está sendo desenvolvida em PHP puro, integrada diretamente à base de código já existente do sistema corporativo. Essa escolha metodológica visa garantir compatibilidade imediata com a estrutura em produção, evitando processos de migração tecnológica e reduzindo riscos associados à substituição de componentes consolidados. Assim, as novas funcionalidades são incorporadas de forma progressiva ao sistema, respeitando sua organização interna e preservando sua estabilidade operacional.

Mesmo sem o uso de um framework formal, a arquitetura adotada segue princípios consagrados de organização de software, como a separação entre camadas de acesso a dados, regras de negócio e interface de comunicação. A API é estruturada segundo o estilo arquitetural RESTful, utilizando o protocolo HTTP e o formato JSON para a troca de mensagens, de modo a garantir comunicação stateless (sem estado) entre cliente e servidor.

O ambiente de desenvolvimento é containerizado por meio do Docker, sendo essa ferramenta utilizada exclusivamente em nível local. Seu uso tem como objetivo padronizar o ambiente entre os desenvolvedores, assegurando que todos trabalhem sob as mesmas configurações de servidor, dependências e versões de bibliotecas. Essa estratégia metodológica contribui para a reprodutibilidade do ambiente de testes e reduz inconsistências decorrentes de diferenças entre máquinas, sem interferir na infraestrutura já consolidada do sistema em produção.

3.3.2 Estado Atual da Implementação (Web e Autenticação)

No estágio atual do projeto, os esforços iniciais concentraram-se na estruturação dos mecanismos de controle de acesso da aplicação web (TechPS), com a validação dos endpoints de autenticação (login) e gestão básica via tokens (JWT). Consolidada esta etapa, o propósito do desenvolvimento volta-se agora para a expansão da API e a implementação de novas regras de negócio voltadas ao sistema embarcado. Este avanço demanda a modificação do esquema do banco de dados, especificamente na tabela de usuários (users), que receberá novas colunas para o armazenamento de dados biométricos (digital) e identificadores RFID, viabilizando a integração futura com os sensores de hardware.

3.3.3 Planejamento da Integração com o Sistema Embarcado

Para a comunicação com o dispositivo IoT (Raspberry Pi), a API encontra-se em fase de **modelagem e expansão**. Diferente da aplicação web, o embarcado requer estruturas de dados específicas que ainda não foram totalmente integradas ao esquema de produção. A metodologia definida para os próximos passos prevê a refatoração da tabela de usuários no banco de dados **MySQL**. Utilizando o recurso de Migrations do Laravel, serão adicionadas novas colunas para armazenar:

- **Hash da Biometria (Digital):** Para validação *offline* e sincronização com o módulo SFM-V1.7.
- **Identificador RFID (UID):** Para vínculo com os cartões/tags utilizados no módulo RDM6300.

Atualmente, não existem endpoints ativos dedicados exclusivamente ao consumo do hardware embarcado. A arquitetura dos controllers está desenhada para receber essas requisições futuramente, mas a implementação do código aguarda a conclusão da atualização do esquema do banco de dados citada acima.

4 RESULTADOS E DISCUSSÕES

Esta seção detalha o ciclo de implementação do dispositivo de monitoramento de jornada para a empresa TechPS, demonstrando a tradução dos requisitos teóricos em funcionalidades práticas, bem como a apresentação dos resultados alcançados através de testes e simulações.

4.1 FUNCIONAMENTO PRÁTICO DO SISTEMA IOT

O funcionamento prático do sistema IoT desenvolvido segue um fluxo linear e orientado a eventos, onde a interação entre hardware, lógica de software e interface ocorre de maneira orquestrada. Abaixo, descreve-se o ciclo de operação completo, utilizando como exemplo o cenário de identificação de um motorista via RFID. O texto abaixo descreve o fluxo técnico real implementado, destacando a navegação entre telas (QStackedWidget), a gestão de estado (logica_ativa) e a delegação de tarefas para os gerenciadores específicos. A lógica central do sistema é governada pela classe NucleoAplicacao, que atua como o controlador mestre,

gerenciando o ciclo de vida da aplicação, a navegação entre telas e a delegação de eventos para módulos de lógica especializados. O funcionamento prático ocorre através de um fluxo de estados bem definido, conforme detalhado a seguir:

4.2 INICIALIZAÇÃO E ESTRUTURA DE NAVEGAÇÃO

Ao ser instanciado, o *NucleoAplicacao* configura a janela principal e inicializa um *QStackedWidget* (pilha de widgets). Este componente permite que múltiplas interfaces (telas) sejam carregadas na memória, mas apenas uma seja exibida por vez. O sistema prepara cinco interfaces principais, a seguir elas serão apresentadas:

Inicialmente temos o índice 0 (Figura 2), nessa tela é possível selecionar o método de autenticação que o usuário deseja utilizar.

Figura 2: Tela de identificação

Fonte: Autores.

Nela temos RFID e Biometria Digital, onde o usuário utilizará a tag cadastrada no banco de dados da empresa e confirmará com a digital. O usuário e senha é o padrão da empresa no momento, enquanto que a funcionalidade de identificação por Biometria Facial está sendo desenvolvida por um dos funcionários da empresa.

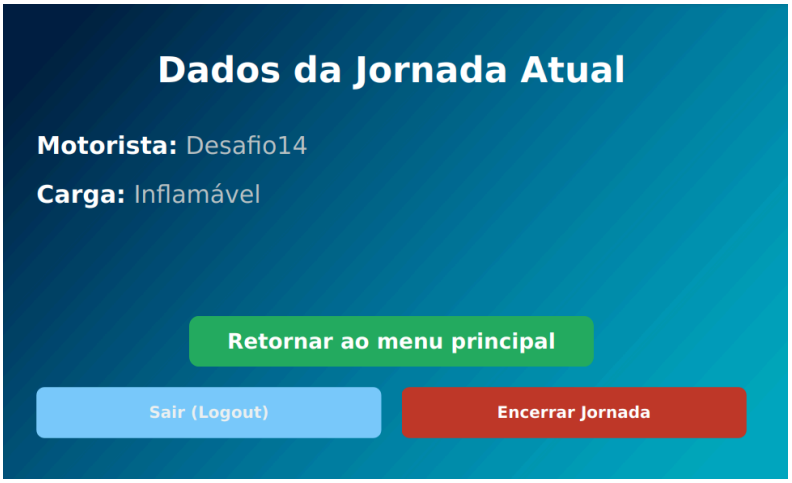
A tela principal chamamos de índice 1 (Figura 3), ela é o painel de controle da jornada, ou menu principal, nesta tela o funcionário seleciona o evento (Espera → Carga; Descarga; Fiscalização. Pausas → Almoço; Descanso; Repouso) que deseja registrar ou se deseja consultar informações sobre a jornada em andamento (Dados da jornada).

Figura 3: Tela do menu principal

Fonte: Autores.

O índice 2 (Figura 4), é onde o funcionário poderá acessar as informações da jornada. No momento ainda estão sendo filtradas quais serão essas informações, mas por enquanto estão sendo exibidos apenas o nome do motorista e o tipo de carga.

Figura 4: Tela de dados da jornada



Fonte: Autores.

O prompt RFID/BIOMETRIA, que vamos denominar de índice 3 (Figura 5), é a tela de aguardo para leitura do cartão RFID (tag) e da biometria, nela é solicitado a aproximação do RFID e a confirmação com a digital, caso falhe em alguma das etapas aparecerá na tela um aviso e será possível realizar uma nova tentativa**.**

Figura 5: Telas de solicitação de rfid e biometria digital

Fonte: Autores.

O Índice 4 é a tela de espera (Figura 6), ela possibilita que o motorista registre o tipo e o tempo de espera, esse registro - contendo tipo, início e fim - é registrado em um banco de dados local (sqlite), esses dados serão enviados para a API da empresa. Enquanto não é confirmado que o envio foi realizado de forma bem sucedida, ou seja, que o registro esteja no banco de dados da empresa, ele permanecerá salvo no banco de dados local. Essa mecânica é orquestrada de forma lógica pelo núcleo, a lógica de pausa tem o seu próprio gerenciador lógico.

Figura 6: Tela de Esperas

Fonte: Autores.

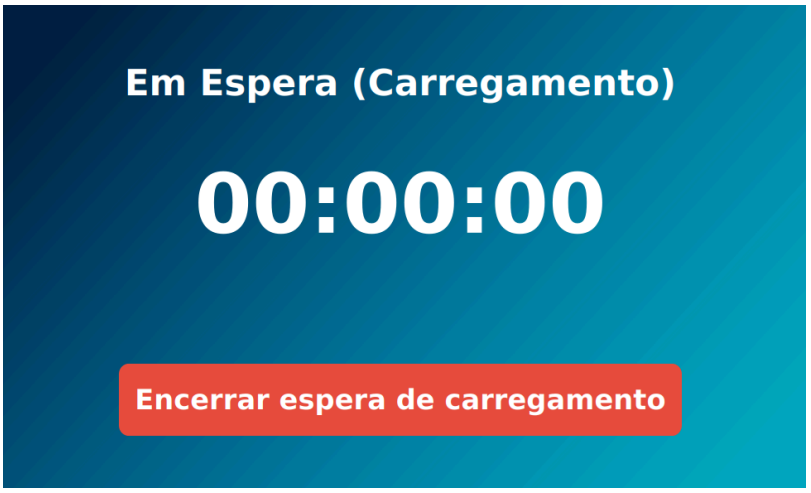
De forma semelhante ao informado acima, o índice 5, responsável pela tela de pausa (Figura 7), funciona da mesma forma.

Figura 7: Tela de Pausas

Fonte: Autores.

Quando um dos eventos é clicado, seja em pausa ou espera, a tela de cronômetro aparece (Figura 8), assim, o motorista consegue visualizar o tempo decorrido. Esse cronômetro é importante visto que a legislação tem algumas exigências como tempo mínimo e máximo de duração de pausa para o almoço ou com quanto tempo a espera pode ser enquadrada como descanso por exemplo.

Figura 8: Tela de cronômetro



Fonte: Autores.

Simultaneamente, ao carregamento das telas, o núcleo instância os **Gerenciadores de Lógica** (*GerenciadorJornada, RfidBiometriaLogica, GerenciadorPausa, GerenciadorEspera, LogicaJornada, entre outros*), injetando neles as dependências de hardware (simuladores ou drivers reais) e de persistência de dados (*DadosJornada*).

4.3 FLUXO DE AUTENTICAÇÃO

O processo de login inicia-se na Tela de Seleção. O sistema opera em um modelo de **Lógica Ativa**, onde o núcleo define qual módulo deve responder às interações do usuário naquele momento:

- **Via RFID/Biometria:** Ao selecionar esta opção, o núcleo altera a visualização para a Tela 3 e define *self._logica_ativa* como *RfidBiometriaLogica*. O método *iniciar_fluxo*() é disparado, ativando a escuta dos leitores de hardware.
- **Via Usuário/Senha:** Ao selecionar esta opção, o sistema navega para a tela de usuário e senha e define self._logica_ativa como UsuarioSenhaLogica. O sistema então aguarda o sinal de *login_solicitado* vindo do formulário.

Quando qualquer uma das lógicas valida o usuário com sucesso, o sinal *autenticacao_concluida* é emitido. O núcleo captura este sinal, inicializa o registro da jornada no banco de dados local e transita a interface para a Tela Principal (Índice 1).

4.4 ORQUESTRAÇÃO DE JORNADA E DELEGAÇÃO

Uma vez logado, o usuário interage com a Tela Principal. O *NucleoAplicacao* não processa as regras de negócio diretamente; ele atua como uma ponte (proxy).

Quando um botão de "Início de Refeição" é pressionado, o sinal é conectado diretamente ao método correspondente no *GerenciadorJornada*. Este, por sua vez, utiliza sub-gerenciadores especializados (*GerenciadorPausas* e *GerenciadorEspera*) para validar a transição de estado e registrar o evento, assim acontece com todos os tipos de eventos.

4.5 GESTÃO CENTRALIZADA DE DIÁLOGOS, INTERRUPÇÕES E ENCERRAMENTO

Um aspecto crítico do funcionamento é o sistema de roteamento de respostas. Como a interface gráfica opera de forma assíncrona, quando o sistema precisa fazer uma pergunta ao usuário (ex: "Confirmar início de descanso?"), o *NucleoAplicacao* exibe o diálogo genérico.

A resposta do usuário não é processada no núcleo, mas sim encaminhada através do método *_rotear_resposta_dialogo*. Além do gerenciamento da interface, o *NucleoAplicacao* atuará como orquestrador da comunicação externa. A integração com o servidor central não ocorre via conexão direta ao banco de dados, mas através de uma API Restful.

Quando um evento crítico ocorre (como o início de uma jornada via leitor RFID), o sistema captura os dados brutos da porta serial, processa a informação e monta um payload em formato JSON. Este pacote é enviado via requisição HTTPS segura (usando o módulo de comunicação 4G) para o backend.

Para garantir a integridade dos dados em áreas sem sinal ou com sinal muito fraco, foi implementada uma estratégia de guardar os dados na memória interna e, caso a conexão esteja boa o suficiente, os dados são enviados via API e armazenados no banco de dados da TechPS.

4.5 FUNCIONAMENTO DO HARDWARE

Atualmente, o hardware se encontra em fase final de desenvolvimento. O desenvolvimento do software do sistema embarcado acontece, prioritariamente, em ambiente virtual – com uso de simuladores – enquanto que a testagem acontece na raspberry integrada pelos respectivos módulos. Foram realizados testes envolvendo a placa da Raspberry Pi

conectada a um display touchscreen de sete polegadas, alto-falantes e os módulos RFID e de Biometria conectados às portas GPIO da placa.

Para realizar a integração entre os módulos e a placa, fez-se uso de uma matriz de contatos e jumpers durante o processo de prototipagem. Esse processo teve como objetivo realizar a validação do funcionamento do sistema em conjunto com o hardware. Os resultados obtidos indicam o correto desempenho da aplicação, demonstrando sua capacidade de atender aos requisitos estabelecidos.

5 CONCLUSÃO/CONSIDERAÇÕES FINAIS

O desenvolvimento do projeto TechPS cumpriu seu objetivo fundamental ao estabelecer uma base tecnológica robusta para a modernização do sistema logístico, validando a arquitetura proposta como uma solução segura e escalável. A integração entre o sistema embarcado, baseado em Raspberry Pi 5, e a plataforma web centralizada comprovou a viabilidade técnica do modelo de computação de borda (edge computing), assegurando a autonomia operacional e a persistência local de dados via SQLite, requisitos essenciais para ambientes com conectividade intermitente.

Os resultados obtidos confirmam o compromisso da equipe com a entrega do produto. Foi possível validar funcionalmente a autenticação de condutores através dos módulos físicos de RFID e biometria, bem como a interação do usuário por meio de uma interface gráfica (GUI) desenvolvida em Python e de um sistema de notificações por voz. A implementação da API RESTful visa garantir a separação eficiente de responsabilidades, permitindo que o processamento biométrico e as regras de jornada fossem executados sem comprometer o desempenho do banco de dados central.

Para trabalhos futuros, sugere-se a expansão dos testes para cenários de campo com rastreamento real via barramento CAN e a conclusão dos módulos de relatórios gerenciais. Conclui-se que a infraestrutura entregue — composta por uma "fábrica de software" interna com componentes reutilizáveis e um hardware validado — oferece a solidez necessária para a continuidade ágil do desenvolvimento, atendendo às exigências de auditoria e modernização da TechPS.

REFERÊNCIAS

BRASIL. Lei nº 13.103, de 2 de março de 2015. Dispõe sobre o exercício da profissão de motorista; altera a Consolidação das Leis do Trabalho – CLT, aprovada pelo Decreto-Lei nº 5.452, de 1º de maio de 1943, e as Leis nº 9.503, de 23 de setembro de 1997 – Código de Trânsito Brasileiro, e nº 11.442, de 5 de janeiro de 2007; e dá outras providências. Diário Oficial da União: seção 1, Brasília, DF, 3 mar. 2015. Disponível em: <https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2015/lei/l13103.htm>. Acesso em: 29 jan. 2026.

FINKENZELLER, Klaus. **RFID Handbook**: fundamentals and applications in contactless smart cards, radio frequency identification and near-field communication. 3. ed. Chichester: John Wiley & Sons, 2010.

FIELDING, R. T. **Architectural styles and the design of network-based software architectures.** 2000. 162 f. Tese (Doutorado em Informática) – University of California, Irvine, 2000. Disponível em: https://ics.uci.edu/~fielding/pubs/dissertation/top.htm. Acesso em: 18 jan. 2026.

KUROSE, James F.; ROSS, Keith W. **Redes de computadores e a Internet**: uma abordagem top-down. 6. ed. São Paulo: Pearson Education do Brasil, 2013.

MONK, Simon. **Programando o Raspberry Pi**: primeiros passos com Python. São Paulo: Novatec, 2013.

PRESSMAN, Roger S.; MAXIM, Bruce R. **Engenharia de software**: uma abordagem profissional. 9. ed. Porto Alegre: Bookman, 2021.

SOMMERVILLE, Ian. **Engenharia de software**. 10. ed. São Paulo: Pearson Education do Brasil, 2019.

STALLINGS, William. **Criptografia e segurança de redes**: princípios e práticas. 6. ed. São Paulo: Pearson Education do Brasil, 2015.

TANENBAUM, Andrew S. **Organização estruturada de computadores**. 6. ed. São Paulo: Pearson Prentice Hall, 2013.

UPTON, Eben; HALFACREE, Gareth. **Raspberry Pi**: guia do usuário. São Paulo: Novatec, 2013.