INSTITUTO FEDERAL DE SANTA CATARINA
CURSO DE ENGENHARIA ELÉTRICA

SERGIO SILVA MACEDO

DESENVOLVIMENTO DE UM SISTEMA DE RASTREAMENTO EM TEMPO REAL APLICADO AO CONTEXTO DA FF TRANSPORTES

TRABALHO DE CONCLUSÃO DE CURSO

ITAJAÍ
2026

Sergio Silva Macedo

DESENVOLVIMENTO DE UM SISTEMA DE RASTREAMENTO EM TEMPO REAL APLICADO AO CONTEXTO DA FF TRANSPORTES

Monografia apresentada ao curso de Engenharia Elétrica do Instituto Federal de Santa Catarina, para obtenção do título de bacharel em Engenharia Elétrica.

Área de concentração: Desenvolvimento de Produto

Orientador(a): Prof. Dr. Tiago Drummond Lopes

ITAJAÍ
2026

Ficha de Identificação da obra elaborada pelo autor, através do cadastro de ficha de identificação disponível no portal discente do Sistema Integrado de Gestão Acadêmica - SIGAA, do IFSC.

Macedo, Sergio Silva
Desenvolvimento de um sistema de rastreamento em tempo real aplicado ao contexto da ff transportes / Sergio Silva Macedo; Orientador(a): Prof. Dr. Tiago Drummond Lopes. - Itajaí, SC, 2026.
80 p.

Trabalho de conclusão de curso (Graduação) - Instituto Federal de Santa Catarina, Campus Itajaí. Curso de Bacharelado Em Engenharia Elétrica.

Inclui referências.

1. Sistema de Rastreamento. 2. Node. 3. js. 4. React. 5. React Native. I. Lopes, Prof. Dr. Tiago Drummond. II. Instituto Federal de Santa Catarina. Curso de Bacharelado Em Engenharia Elétrica. III. Título.

Àqueles que buscam na tecnologia uma ferramenta para aprimorar o cotidiano e otimizar processos. Que o conhecimento aqui gerado possa contribuir para a eficiência logística e a segurança de todos que dependem do movimento, do transporte e da informação em tempo real.

AGRADECIMENTOS

A concretização deste trabalho representa não apenas a conclusão de uma etapa acadêmica, mas o reflexo de um suporte inestimável e contínuo. Sou profundamente grato, em primeiro lugar, a Deus, pela força, sabedoria e oportunidades que pavimentaram este caminho.

Minha eterna gratidão aos meus pais, Kleyton de Mattos Macedo e Elizabeth Aparecida Campus Silva Macedo. O amor incondicional, os valores transmitidos e o apoio incansável foram pilares essenciais para cada passo da minha jornada, e esta conquista é, em grande parte, fruto da dedicação e dos sacrifícios de vocês.

À minha amada namorada, Julia Pedro Matsunaga. Seu companheirismo, paciência e incentivo constante foram um refúgio e uma fonte de inspiração, especialmente nos momentos mais desafiadores. Sua presença fez toda a diferença, tornando este percurso mais leve e significativo.

Aos meus amigos, pela amizade, pelos momentos de descontração e por sempre me lembrarem da importância de equilibrar os desafios com a leveza da vida.

Aos meus colegas de trabalho, pelo ambiente colaborativo, pelas trocas de conhecimento e pelo apoio mútuo que tornaram o dia a dia mais produtivo e agradável.

Aos meus professores, pela dedicação, pelo conhecimento compartilhado e pela orientação fundamental que me guiou ao longo de toda a formação acadêmica.

A todos que, direta ou indiretamente, contribuíram para a realização deste projeto, meu sincero muito obrigado.

RESUMO

Este trabalho de conclusão de curso apresenta o desenvolvimento de um sistema de rastreamento em tempo real utilizando a tecnologia Global Positioning System (GPS) de dispositivos celulares (smartphones), especificamente projetado como solução personalizada e de baixo custo para otimizar as operações logísticas da empresa FF Transportes LTDA. O problema central abordado foi a necessidade de monitoramento contínuo e preciso da frota, visando aprimorar a gestão de riscos, a eficiência operacional e o serviço ao cliente. Para tal, o objetivo principal foi construir uma solução tecnológica integrada, capaz de fornecer visibilidade instantânea sobre a localização de veículos e condutores. A metodologia de desenvolvimento adotou uma arquitetura full-stack, fundamentada na linguagem JavaScript em todas as camadas. O backend foi implementado utilizando Node.js e o framework Express.js, com comunicação em tempo real via WebSocket (Socket.io), aproveitando sua natureza assíncrona e não-bloqueante para gerenciar um grande volume de operações de Input/Output (I/O) e conexões simultâneas, essenciais para uma Application Programming Interface (API) Representational State Transfer (RESTful) de alta performance. Para a persistência de dados, optou-se pelo banco de dados NoSQL MongoDB, reconhecido por sua flexibilidade de esquema (schema-less) e escalabilidade horizontal, adequadas para armazenar informações dinâmicas, incluindo dados geoespaciais em tempo real. A comunicação bidirecional de baixa latência foi estabelecida por meio do protocolo WebSocket; as requisições Hypertext Transfer Protocol (HTTP) pontuais e o fluxo de autenticação foram gerenciados pela biblioteca Axios. A interface web administrativa para visualização e gestão da frota, incluindo cadastro e gestão de condutores, clientes e endereços, foi desenvolvida com a biblioteca React e o framework Next.js, que possibilitou renderização server-side (Server-Side Rendering – SSR) e geração de sites estáticos (Static Site Generation – SSG), otimizando o desempenho e o SEO. A aplicação móvel para condutores foi inicialmente prototipada com Expo; devido às limitações na execução de tarefas em segundo plano, especialmente na captura contínua da localização, optou-se pela migração para o React Native Command Line Interface (CLI), permitindo acesso a funcionalidades nativas como a captura precisa de coordenadas GPS em segundo plano. O processo de validação incluiu testes funcionais e de usabilidade em ambos os ambientes (navegador Brave para web e Android Studio com emuladores para mobile), focando na integração entre frontend, backend e na precisão do rastreamento em tempo real. Os resultados obtidos demonstram a eficácia de um sistema de rastreamento sólido e escalável; a configuração do intervalo de captura de posição mostrou-se fundamental para equilibrar a precisão do rastreamento e o volume de dados armazenados. O sistema proporciona à FF Transportes LTDA uma ferramenta estratégica para a otimização de rotas, a prevenção de perdas e o aprimoramento geral

da cadeia de suprimentos, consolidando uma solução de alto valor agregado para os desafios logísticos contemporâneos.
Palavras-chave: sistema de rastreamento; Node.js; React; React Native; geolocalização.

ABSTRACT

This final paper presents the development of a real-time tracking system using smartphone Global Positioning System (GPS) technology, specifically designed as a customized, low-cost solution to optimize the logistical operations of FF Transportes LTDA. The central problem addressed was the need for continuous and precise fleet monitoring, aiming to improve risk management, operational efficiency, and customer service. To this end, the main objective was to build an integrated technological solution capable of providing instant visibility into the location of vehicles and drivers. The development methodology adopted a full-stack approach, utilizing JavaScript as the primary language across all layers. The backend was implemented using Node.js and the *Express.js* framework, with real-time communication via *WebSocket* (*Socket.io*), leveraging their asynchronous and non-blocking architecture to manage a high volume of I/O operations and concurrent connections, essential for a high-performance RESTful API. For data persistence, the *NoSQL MongoDB* database was chosen, recognized for its schema-less flexibility and horizontal scalability, suitable for storing dynamic information, including real-time geospatial data. Low-latency, bidirectional communication was established via the *WebSocket* protocol; punctual HTTP requests and the authentication flow were managed by the *Axios* library. The administrative web interface for fleet visualization and management, including registration and management of drivers, clients, and addresses, was developed with the React library and the *Next.js* framework, which enabled server-side rendering (*SSR*) and static site generation (*SSG*), optimizing performance and SEO. The mobile application for drivers was initially prototyped with *Expo*; due to limitations in background task execution, especially for continuous location capture, the project was migrated to *React Native CLI*, allowing access to native functionalities such as precise GPS coordinate capture in the background. The validation process included functional and usability tests in both environments (*Brave* browser for web and Android Studio with emulators for mobile), focusing on the integration between frontend, backend, and the accuracy of real-time tracking. The obtained results demonstrate the effectiveness of a robust and scalable tracking system; the configuration of the position capture interval proved essential to balance tracking precision and the volume of stored data. The system provides FF Transportes LTDA with a strategic tool for route optimization, loss prevention, and overall supply chain improvement, solidifying a high-value solution for contemporary logistical challenges.
Keywords: tracking system; Node.js; React; React Native; geolocalization.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
API	*Application Programming Interface*
AVD	*Android Virtual Device*
BSON	*Binary JSON*
CLI	*Command Line Interface*
CNH	Carteira Nacional de Habilitação
CPF	Cadastro de Pessoas Físicas
CPU	*Central Processing Unit*
CSRF	*Cross-Site Request Forgery*
CSS	*Cascading Style Sheets*
CT-e	Conhecimento de Transporte Eletrônico
DOM	*Document Object Model*
FMS	*Fleet Management System*
GPS	*Global Positioning System*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
I/O	*Input/Output*
IDE	*Integrated Development Environment*
IoT	*Internet of Things*
JSON	*JavaScript Object Notation*
JWT	*JSON Web Token*
MB	Megabyte
MVC	*Model-View-Controller*
ODM	*Object-Document Mapper*

RAM	*Random Access Memory*
RENAVAM	Registro Nacional de Veículos Automotores
RESTful	*Representational State Transfer*
RG	Registro Geral
SDK	*Software Development Kit*
SGBD	Sistema de Gerenciamento de Banco de Dados
SSG	*Static Site Generation*
SSR	*Server-Side Rendering*
TCC	Trabalho de Conclusão de Curso
TCP	*Transmission Control Protocol*
UI	*User Interface*
UX	*User Experience*
WPS	*Wi-Fi Positioning System*

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos, a eficiência no setor de transporte tornou-se um fator crucial para a competitividade e sustentabilidade das empresas. Para Novaes (2007), a logística empresarial, através do planejamento, organização e controle efetivo das atividades de movimentação, armazenagem e distribuição, pode prover melhor nível de serviço e satisfação ao cliente. Já no que diz respeito às tecnologias aplicadas ao transporte, Giannopoulos (2004) destaca que os principais impactos são pertinentes à melhoria no planejamento e controle do processo de transporte, otimização de custos e serviços, redução de papéis e esforço manual e agilidade no fluxo das informações.

A evolução tecnológica tem desempenhado um papel fundamental no setor de transporte, permitindo que as empresas aprimorem o controle e o monitoramento de suas operações. Tecnologias como a telemetria, Fleet Management System (FMS) e Internet of Things (IoT) têm revolucionado a maneira como as transportadoras acompanham suas frotas, oferecendo dados em tempo real sobre a localização dos veículos, condições de tráfego e desempenho dos motoristas. Essas inovações garantem uma supervisão contínua e detalhada, possibilitando ajustes imediatos e uma gestão mais precisa das operações (Tao e Xing, 2011).

Embora as tecnologias de rastreamento tenham se tornado mais acessíveis, o custo ainda é um fator determinante, especialmente quando comparado entre empresas de pequeno e grande porte. Para grandes transportadoras, os sistemas de rastreamento geralmente são personalizados e integrados a uma ampla infraestrutura tecnológica, o que, apesar de custoso, oferece maior controle e eficiência operacional. Em contrapartida, pequenas empresas frequentemente enfrentam dificuldades em arcar com os elevados custos de soluções personalizadas, o que muitas vezes as impede de acessar essas tecnologias avançadas (Liu et al., 2019). A falta de acesso a serviços mais sofisticados pode resultar em uma gestão menos eficiente, limitando o potencial de otimização da frota e reduzindo o retorno sobre o investimento.

Diante dessas considerações, a proposta de um sistema de rastreamento em tempo real utilizando o Global Positioning System (GPS) de celular apresenta-se como uma solução viável e eficiente. Ao oferecer uma plataforma de monitoramento que alia precisão, baixo custo e flexibilidade, este sistema atende às necessidades das transportadoras que buscam melhorar sua eficiência operacional, reduzir custos e aumentar a segurança da frota. A combinação dessas vantagens justifica a adoção da tecnologia proposta e destaca seu potencial para transformar a gestão de frotas no setor de transporte. Neste contexto, o presente trabalho visa apresentar o desenvolvimento e implementação de tal sistema para a empresa FF Transportes LTDA.

1.1 OBJETIVOS

1.1.1 Objetivo geral

Desenvolver um sistema de rastreamento em tempo real, utilizando a tecnologia GPS de dispositivos celulares, com foco em oferecer uma solução personalizada, flexível e de baixo custo, de modo a suprir as demandas operacionais e financeiras da empresa FF Transportes LTDA, considerando seu contexto específico e a necessidade de otimização dos recursos disponíveis.

1.1.2 Objetivos específicos

1. Projetar uma solução de rastreamento em tempo real que utilize o GPS do celular como base tecnológica.
2. Implementar um servidor capaz de lidar com as posições recebidas em tempo real, garantindo a precisão e confiabilidade dos dados.
3. Desenvolver o módulo de cadastro para condutores, clientes e endereços, permitindo a gestão eficiente dessas informações.
4. Criar uma interface web que possibilite a visualização das informações de rastreamento, com atualização sob demanda, além de permitir, de forma intuitiva, a realização de cadastros, edições e remoções dos dados apresentados.
5. Integrar todas as interfaces do sistema, assegurando que operem de maneira coesa e eficiente.

A crescente demanda por soluções de rastreamento em tempo real na gestão de frotas evidencia a relevância do desenvolvimento de um sistema específico para a empresa FF Transportes LTDA, baseado na utilização do GPS de dispositivos celulares. Este projeto tem como objetivo proporcionar uma ferramenta prática e acessível para otimizar o controle da localização e movimentação dos veículos, integrando funcionalidades como o cadastro de condutores, clientes e endereços, além de uma interface web intuitiva para monitoramento, com consulta às informações de rastreamento sob demanda.

1.2 ORGANIZAÇÃO DO TRABALHO

Este documento está organizado em cinco capítulos, conforme descrito a seguir. O Capítulo 1 contextualiza o tema, apresenta os objetivos e esta organização do trabalho. O Capítulo 2 apresenta o referencial teórico que fundamenta o desenvolvimento do sistema de rastreamento em tempo real, abordando geolocalização e tecnologia GPS, JavaScript, arquitetura de sistemas de rastreamento, além das tec-

nologias utilizadas no projeto: Node.js, Express, WebSocket, MongoDB, Axios, React, Next.js e React Native, incluindo React Native CLI e Android Studio. O Capítulo 3 descreve a metodologia adotada, com ênfase no levantamento de requisitos junto à FF Transportes LTDA, no fluxo de dados do sistema, na configuração do ambiente de desenvolvimento e na implementação do backend, do frontend web e do aplicativo móvel. O Capítulo 4 apresenta a análise e discussão dos resultados, contemplando a comunicação em tempo real, a persistência de dados, o desempenho da interface web, o desenvolvimento da aplicação móvel e a configuração do intervalo de captura de posição. Por fim, o Capítulo 5 traz as conclusões e considerações finais, com as lições aprendidas e sugestões de trabalhos futuros para aprimoramento do sistema.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta a base teórica e tecnológica que fundamenta o desenvolvimento do sistema de rastreamento em tempo real abordado neste trabalho. São expostos, na sequência, conceitos de processo e metodologia de desenvolvimento de software; em seguida, os conceitos de geolocalização e de sistemas de rastreamento; as tecnologias de backend (JavaScript, Node.js, Express, WebSocket, MongoDB) e de frontend (Axios, React, Next.js); por fim, as abordagens de desenvolvimento móvel e as ferramentas utilizadas (React Native CLI e Android Studio).

2.1 DESENVOLVIMENTO DE SOFTWARE

2.1.1 Ciclo de vida e processo de software

O desenvolvimento de software é um processo que envolve a aplicação sistemática de atividades, métodos e práticas para conceber, especificar, implementar e evoluir sistemas de software (Pressman e Maxim; Sommerville, 2014, 2015). Conforme Pressman e Maxim (2014), um processo de software define quem faz o quê, quando e como alcançar um conjunto de objetivos de projeto. Sommerville (2015) destaca que o processo de desenvolvimento engloba as atividades fundamentais de especificação de requisitos, projeto e implementação, validação e evolução do software.

O levantamento e a análise de requisitos constituem a fase inicial e crítica do ciclo de vida, pois definem o que o sistema deve fazer a partir das necessidades dos stakeholders (Pressman e Maxim, 2014). A comunicação direta com os usuários e a observação do ambiente de trabalho permitem identificar demandas reais e priorizar funcionalidades, reduzindo o risco de entregar uma solução desalinhada às expectativas (Sommerville, 2015). O planejamento das fases subsequentes — projeto da arquitetura, implementação e testes — depende da qualidade e da estabilidade dos requisitos levantados (Pressman e Maxim, 2014).

2.1.2 Metodologias ágeis e desenvolvimento iterativo

As metodologias tradicionais de desenvolvimento, frequentemente associadas ao modelo em cascata, organizam o projeto em fases sequenciais e lineares (requisitos, projeto, implementação, testes), com entrega ao final do ciclo (Pressman e Maxim, 2014). Em contextos em que os requisitos evoluem ou em que o feedback do usuário é necessário ao longo do desenvolvimento, abordagens mais flexíveis tornam-se mais adequadas (Schwaber, 2004).

As metodologias ágeis priorizam indivíduos e interações, software em funcionamento, colaboração com o cliente e resposta a mudanças, em detrimento de processos rígidos e documentação extensiva (Beck et al., 2001). O desenvolvimento iterativo e incremental permite que o produto seja construído em ciclos curtos, com entregas parciais e frequentes, favorecendo a validação contínua e o ajuste às necessidades reais do negócio (Schwaber; Sommerville, 2004, 2015). Schwaber (2004) enfatiza que essa abordagem é particularmente útil em projetos nos quais o problema a ser resolvido e as prioridades podem ser refinados com base no feedback dos usuários e no uso efetivo do sistema.

A adoção de uma arquitetura modular, com separação clara entre camadas (por exemplo, frontend, backend e aplicação móvel), facilita o desenvolvimento iterativo e a manutenção do sistema (Sommerville, 2015). Padrões de projeto como o Model-View-Controller (MVC) contribuem para essa organização, ao dividir a aplicação em componentes de modelo (dados e lógica de negócio), visão (apresentação) e controlador (intermediação), promovendo reutilização e clareza de responsabilidades (Gamma et al.; Pressman e Maxim, 1994, 2014).

2.2 GEOLOCALIZAÇÃO

Conforme Câmara et al. (2004), o GPS é um sistema de posicionamento por satélite que fornece informações de localização e tempo para um receptor em qualquer lugar do mundo, desde que haja visada para satélites. Essa funcionalidade é crucial em diversas aplicações modernas, incluindo sistemas de rastreamento, navegação, logística e serviços baseados em localização (Câmara et al.; Wang, Li e Zhang, 2004, 2009). O principal sistema empregado para essa finalidade é o GPS, que opera por meio de uma constelação de satélites que emitem sinais captados por receptores na superfície terrestre (Ferreira, 2012). Ferreira (2012) aborda a navegação via satélite como base para o posicionamento global, enquanto Bernardi e Landim (2002) destacam a aplicação do GPS na coleta de dados em campo.

2.3 JAVASCRIPT

JavaScript é uma linguagem de programação multiparadigma, interpretada e de alto nível, amplamente utilizada no desenvolvimento web e, mais recentemente, em diversas outras áreas, como o desenvolvimento server-side e mobile (Flanagan, 2011). Criada originalmente para adicionar interatividade a páginas web, sua evolução e a padronização via ECMAScript a tornaram uma das linguagens mais populares e versáteis do mundo (Flanagan; Lei, Ma e Tan, 2011, 2014). Segundo Flanagan (2011), o JavaScript é a linguagem de programação da web, e a essência da programação

dinâmica e interativa em navegadores.

Sua arquitetura baseada em eventos e sua natureza assíncrona são características que a tornam particularmente adequada para aplicações que exigem comunicação em tempo real e manipulação de dados de forma não-bloqueante (Pereira; Flanagan, 2014, 2011). Lei et al. (2014) comparam tecnologias de desenvolvimento web e destacam o JavaScript no Node.js como alternativa de alto desempenho. Pereira (2014) reforça que aplicações web em tempo real se beneficiam do modelo assíncrono da linguagem. A popularidade do JavaScript foi impulsionada pelo surgimento de ambientes de execução como o Node.js para o lado do servidor, e frameworks de frontend como React, permitindo que a mesma linguagem seja utilizada em todas as camadas de uma aplicação full-stack (Lei, Ma e Tan; Pereira, 2014, 2014). Essa ubiquidade reduz a curva de aprendizado e otimiza a produtividade no desenvolvimento de sistemas complexos (Pereira, 2014).

2.4 SISTEMA DE RASTREAMENTO

Sistemas de rastreamento em tempo real representam uma ferramenta estratégica fundamental na gestão logística moderna, permitindo o monitoramento contínuo e a localização precisa de ativos, veículos ou pessoas (Christopher; Novaes, 2016, 2007). A capacidade de obter informações instantaneamente sobre a posição de elementos em movimento proporciona um controle operacional aprimorado e a tomada de decisões ágeis (Christopher, 2016). Conforme Christopher (2016), a visibilidade na cadeia de suprimentos é um fator crítico, e o rastreamento em tempo real permite que as empresas tenham uma visão contínua de seus ativos, o que é fundamental para a gestão de riscos, otimização de rotas e melhoria do serviço ao cliente. Novaes (2007) enfatiza o papel do gerenciamento da cadeia de distribuição e da informação em tempo real na logística. Zhang et al. (2015) apresentam sistemas de rastreamento em tempo real baseados em GPS e GPRS para transporte e cadeia de frio. A implementação desses sistemas visa a otimização de rotas, a prevenção de perdas ou furtos e a maximização da eficiência operacional, resultando em redução de custos e aumento da segurança (Wang, Li e Zhang, 2009).

2.5 NODE.JS

Node.js é um ambiente de execução JavaScript server-side, open-source e multiplataforma, construído sobre o motor V8 do Google Chrome (Tilkov e Vinoski; Teixeira, 2010, 2013). Ele permite que desenvolvedores utilizem JavaScript para escrever código de servidor, expandindo o ecossistema da linguagem além do navegador (Tilkov e Vinoski, 2010). A principal característica que diferencia o Node.js é sua arquitetura

de *I/O* não-bloqueante e orientada a eventos, que o torna altamente eficiente para lidar com um grande número de conexões simultâneas e operações de entrada/saída intensivas (Tilkov e Vinoski; Teixeira, 2010, 2013). Conforme Tilkov e Vinoski (2010), o Node.js permite a construção de programas de rede de alta performance utilizando JavaScript. Pereira (2014) e Teixeira (2013) destacam o modelo não bloqueante e a construção de software escalável em JavaScript no servidor.

Essa arquitetura é ideal para aplicações em tempo real, como API RESTful, chatbots e sistemas de streaming de dados, onde a baixa latência e a alta capacidade de processamento concorrente são cruciais (Pereira; Tilkov e Vinoski, 2014, 2010). A popularidade do Node.js também se deve ao seu vasto ecossistema de pacotes (módulos) acessíveis via npm, que acelera o desenvolvimento e permite a integração com diversas ferramentas e tecnologias de banco de dados, como o MongoDB (Teixeira; Pereira, 2013, 2014). Ele serve como o motor para o backend de muitas aplicações modernas, incluindo sistemas de rastreamento que precisam processar e rotear dados de localização em tempo real (Pereira, 2014).

2.6 EXPRESS

Express.js, ou simplesmente Express, é um framework para construção de servidores e aplicações web em Node.js. Ele atua sobre o módulo HTTP nativo do Node.js, oferecendo uma camada de abstração que facilita o tratamento de requisições, o roteamento por URL e a composição de lógica reutilizável por meio de middlewares (Hahn, 2016). O Express é rápido, minimalista e “sem opiniões” (*unopinionated*), pois não impõe uma estrutura rígida ao projeto e permite que o desenvolvedor escolha como organizar rotas, middlewares e integrações com banco de dados ou outros serviços (Hahn, 2016).

O conceito central do Express é o roteamento, isto é, associar um método HTTP (GET, POST, PUT, DELETE, entre outros) e um caminho (URL) a uma função que processa a requisição e envia a resposta. Assim, a aplicação define explicitamente como responde a cada endpoint, o que é a base para a construção de APIs RESTful (Mardan; Hahn, 2014, 2016). Outro pilar é o uso de middlewares, funções que interceptam a requisição (e a resposta) antes de ela chegar à rota final. Middlewares permitem centralizar tarefas como autenticação, leitura e análise do corpo da requisição (*body parsing*), compressão de dados, registro de logs e tratamento de erros, mantendo as rotas focadas na lógica de negócio (Mardan; Hahn, 2014, 2016).

Essa combinação de roteamento e middlewares torna o Express adequado tanto para aplicações web completas quanto para backends que expõem apenas uma API consumida por clientes web ou móveis. No contexto de um sistema de rastreamento em tempo real, o Express pode ser responsável pelas rotas REST

(cadastro de usuários, viagens, autenticação), enquanto a comunicação em tempo real fica a cargo de mecanismos como WebSocket; a mesma instância Node.js pode hospedar ambos (Hahn; Mardan, 2016, 2014).

2.7 WEBSOCKET

WebSocket é um protocolo de comunicação que permite uma comunicação *full-duplex* (bidirecional) sobre uma única conexão *Transmission Control Protocol* (TCP) (Fette e Melnikov, 2011). O protocolo foi padronizado pela IETF no RFC 6455 (Fette e Melnikov, 2011), definindo o handshake e o quadro de dados para comunicação bidirecional. Diferente do modelo tradicional de requisição-resposta do HTTP, onde o cliente inicia cada comunicação, o WebSocket permite que tanto o cliente quanto o servidor enviem mensagens a qualquer momento após o estabelecimento da conexão. Isso o torna ideal para aplicações que exigem comunicação em tempo real e baixa latência, como jogos online, chats, notificações em tempo real e, crucialmente, sistemas de rastreamento (Melkote e D'Souza; Moss, 2012, 2016).

Conforme Moss (2016), *WebSockets* estabelecem uma conexão persistente que permite a troca eficiente de dados, eliminando a sobrecarga de cabeçalhos HTTP repetitivos e reduzindo a latência. Melkote e D'Souza (2012, p. 35-39) corroboram, afirmando que "*WebSockets* fornecem uma comunicação mais eficiente e de baixa latência em comparação com as técnicas de *polling HTTP*, tornando-os ideais para aplicações que exigem atualizações instantâneas". Almeida (2018) exemplifica a implementação de sistemas em tempo real que se beneficiam de canais bidirecionais. No contexto de um sistema de rastreamento, o WebSocket é fundamental para a transmissão contínua de coordenadas geográficas do dispositivo móvel para o servidor e para a atualização imediata das posições na interface de monitoramento (Wang, Li e Zhang; Melkote e D'Souza, 2009, 2012).

2.8 MONGODB

MongoDB é um Sistema de Gerenciamento de Banco de Dados (SGBD) NoSQL orientado a documentos (Chodorow, 2013). Diferente dos bancos de dados relacionais que utilizam tabelas, o MongoDB armazena dados em documentos flexíveis no formato Binary JSON (BSON), que podem variar em estrutura e são facilmente mapeados para objetos em linguagens de programação (Chodorow, 2013). Essa característica de esquema flexível (schema-less) é uma de suas maiores vantagens, permitindo que a estrutura dos dados seja adaptada e evolua sem a necessidade de migrações complexas (Chodorow, 2013). Andreoli et al. (2021) analisam o desempenho diferenciado de bancos NoSQL como o MongoDB em cenários de tempo real.

Segundo Chodorow (2013), o MongoDB oferece alta performance, alta disponibilidade e escalabilidade automática, tornando-o adequado para lidar com grandes volumes de dados e altas taxas de escrita e leitura. Lopes et al. (2024) utilizam MongoDB em aplicativo mobile de rastreamento de transportes, evidenciando sua adequação a baixa latência e dados dinâmicos. Sua capacidade de armazenar dados geoespaciais e realizar consultas baseadas em localização o torna particularmente útil para aplicações de rastreamento (Chodorow; Lopes et al., 2013, 2024). Além disso, a escalabilidade horizontal, que permite distribuir dados por múltiplos servidores (*sharding*), garante que o banco de dados possa crescer junto com a demanda do sistema, sendo uma opção adequada para persistir informações de usuários, veículos e, principalmente, os dados de geolocalização em tempo real (Chodorow, 2013).

2.9 AXIOS

Axios é uma biblioteca JavaScript baseada em Promises para fazer requisições HTTP, funcionando tanto em navegadores quanto em ambientes Node.js (e, por extensão, em *React Native*) (Flanagan; Pereira, 2011, 2014). O uso de Promises para requisições assíncronas é um fundamento do ecossistema web em JavaScript (Flanagan, 2011). O Axios encapsula essas ideias em uma API unificada, simplificando a interação com APIs RESTful e fornecendo uma interface concisa para enviar dados, recuperar informações e manipular cabeçalhos e respostas HTTP (Pereira, 2014). A biblioteca se destaca pela facilidade de uso e pela flexibilidade para diversas operações de rede (Hahn, 2016).

Entre suas principais características, o Axios oferece (Pereira, 2014):

- Suporte a Promises, que facilita a escrita de código assíncrono mais limpo e legível.
- Interceptadores de requisição e resposta, que permitem a execução de funções antes que uma requisição seja enviada ou antes que uma resposta seja tratada, útil para adicionar tokens de autenticação ou lidar com erros globalmente.
- Transformação automática de dados JSON.
- Proteção contra *Cross-Site Request Forgery* (CSRF).

No contexto de aplicações full-stack e mobile, o Axios é a ferramenta de escolha para comunicação com o backend, lidando com o fluxo de dados entre o cliente (web ou mobile) e a API, como no caso de autenticação de usuários e gerenciamento de cadastros de viagens (Pereira; Hahn, 2014, 2016).

2.10 REACT

React é uma biblioteca JavaScript de código aberto para a construção de User Interface (UIs) declarativas, eficientes e flexíveis (Gackenheimer; Wieruch, 2015, 2019). Criada pelo Facebook (atualmente Meta), o React popularizou o conceito de desenvolvimento baseado em componentes, onde a interface é dividida em partes independentes e reutilizáveis, cada uma com sua própria lógica e estado (Gackenheimer, 2015). Segundo Wieruch (2019), o React simplifica o desenvolvimento de UIs complexas ao permitir que os desenvolvedores pensem em componentes independentes e reutilizáveis. Gackenheimer (2015) introduz o React e seu modelo de componentes, e Pinto et al. (2023) analisam comparativamente o React em relação a outras tecnologias de front-end, destacando sua adoção e vantagens.

A principal inovação do React é o Virtual DOM (Pontes; Tostes e Costa, 2018, 2020). Em vez de manipular o Document Object Model (DOM) diretamente (o que pode ser lento), o React cria uma representação virtual da User Interface (UI) na memória e a compara com o *DOM* real. Apenas as alterações necessárias são aplicadas ao *DOM* real, otimizando o desempenho e a renderização (Pontes; Tostes e Costa, 2018, 2020). Sua abordagem declarativa, onde o desenvolvedor descreve como a UI deve parecer para um determinado estado e o React se encarrega de atualizá-la, facilita a criação de interfaces dinâmicas e responsivas (Wieruch; Pontes, 2019, 2018). O React é a base para frameworks como Next.js (para web) e *React Native* (para mobile), permitindo o desenvolvimento full-stack com uma única linguagem e um paradigma de programação consistente (Pinto, Hoffmann e Uriarte; Tostes e Costa, 2023, 2020).

2.11 NEXT.JS

Next.js é um framework de React de código aberto que estende as capacidades do React para a construção de aplicações web de produção, oferecendo funcionalidades avançadas de renderização e otimização (Thakkar, 2020). Ele resolve muitos dos desafios enfrentados no desenvolvimento de aplicações React de larga escala, como a otimização de desempenho e o SEO (Thakkar, 2020). Segundo Thakkar (2020), o Next.js permite que desenvolvedores React construam aplicações web com *Server-Side Rendering* (SSR) e Static Site Generation (SSG), com desempenho otimizado e flexibilidade de renderização.

As principais características do Next.js incluem (Thakkar, 2020):

- *Server-Side Rendering* (SSR): Permite que as páginas sejam pré-renderizadas no servidor antes de serem enviadas ao cliente, melhorando o tempo de carregamento inicial e o SEO.
- *Static Site Generation* (SSG): Gera páginas HyperText Markup Language

(HTML) estáticas em tempo de build, ideal para conteúdo que não muda frequentemente.
- Roteamento baseado em sistema de arquivos: Simplifica a criação de rotas, onde cada arquivo em uma pasta específica se torna uma rota.
- Otimização de imagem e fontes: Recursos embutidos para melhorar o carregamento de ativos.
- *API Routes*: Permite a criação de endpoints de API diretamente no projeto Next.js.

Essas funcionalidades tornam o Next.js uma alternativa sólida para o frontend de sistemas full-stack, garantindo uma experiência de usuário fluida e um bom posicionamento em mecanismos de busca, além de interagir de forma eficiente com as APIs do backend (Thakkar, 2020).

2.12 APLICAÇÕES MÓVEIS HÍBRIDAS E NATIVAS

O desenvolvimento de aplicações móveis pode seguir diferentes abordagens, sendo as mais proeminentes as nativas, híbridas e multiplataforma (Winterfeldt; Eisenman, 2017, 2018). Aplicações nativas são desenvolvidas com as linguagens e Software Development Kit (SDKs) específicas de cada plataforma (Java/Kotlin para Android, *Swift*/*Objective-C* para iOS), oferecendo desempenho e acesso irrestrito a recursos do dispositivo (Eisenman; Andrews, 2018, 2012). Por outro lado, aplicações híbridas utilizam tecnologias web (*HTML*, Cascading Style Sheets (CSS), JavaScript) encapsuladas em um *WebView*, resultando em menor custo de desenvolvimento para múltiplas plataformas, mas com potenciais limitações de desempenho e acesso a recursos nativos (Winterfeldt, 2017). Winterfeldt (2017) discute frameworks de aplicações híbridas e multiplataforma no contexto de sensoriamento móvel, e Eisenman (2018) aborda o *React Native* e a ponte entre JavaScript e componentes nativos.

A abordagem multiplataforma, como a oferecida pelo *React Native*, surge como um intermediário (Eisenman, 2018). O *React Native* permite que desenvolvedores construam aplicativos móveis nativos utilizando JavaScript e a sintaxe declarativa do React, compilando o código para componentes nativos das plataformas iOS e Android (Eisenman, 2018). Isso significa que o desempenho e a experiência do usuário são semelhantes aos de aplicativos nativos, mas com o benefício de uma única base de código (Eisenman, 2018).

2.13 REACT NATIVE CLI

React Native CLI representa a ferramenta oficial para iniciar e gerenciar projetos *React Native* sem as abstrações adicionais de plataformas como o *Expo* (Eisenman,

2018). Ele oferece aos desenvolvedores controle total sobre o projeto nativo (Android e iOS), permitindo a configuração manual de módulos, bibliotecas e dependências específicas da plataforma (Eisenman, 2018). Segundo Eisenman (2018), o CLI é a ferramenta padrão para criar e executar projetos *React Native*, fornecendo uma base sólida para aplicativos que exigem personalização profunda ou acesso a funcionalidades nativas não expostas por outras camadas de abstração, com a configuração do ambiente e a integração com código nativo por meio do CLI.

A principal vantagem do uso do React Native CLI é a flexibilidade para integrar módulos nativos personalizados ou bibliotecas de terceiros que dependem de código nativo (Java/Kotlin para Android, *Swift*/*Objective-C* para iOS) (Eisenman, 2018). Isso é crucial para cenários complexos, como o acesso a APIs de geolocalização com alta precisão, o gerenciamento de serviços em segundo plano (como *foreground services* no Android) ou a integração com hardware específico do dispositivo (Andrews, 2012). Embora exija um conhecimento mais aprofundado do ambiente de desenvolvimento nativo e um processo de build e configuração mais manual, o CLI garante que o desenvolvedor tenha controle total sobre o desempenho e a funcionalidade do aplicativo, sendo a escolha ideal para projetos que necessitam de otimização de baixo nível e funcionalidades específicas de plataforma (Eisenman, 2018).

2.14 ANDROID STUDIO

Android Studio é o Integrated Development Environment (IDE) oficial e mais utilizado para o desenvolvimento de aplicativos na plataforma Android (Lima; Andrews, 2022, 2012). Baseado no software *IntelliJ IDEA* da JetBrains, o *Android Studio* oferece um conjunto abrangente de ferramentas para o desenvolvimento, depuração, teste e build de aplicações Android (Andrews, 2012). Segundo Andrews (2012), trata-se do *IDE* que fornece as ferramentas necessárias para a construção de aplicativos em cada tipo de dispositivo Android. Lima (2022) caracteriza o papel de um IDE no fluxo de desenvolvimento, e Andrews (2012) apresenta o desenvolvimento profissional de aplicações Android e o uso do ambiente oficial.

Suas principais funcionalidades incluem (Andrews; Lima, 2012, 2022):

- Um editor de código inteligente com *auto-completion*, refatoração e análise de código estática.
- Ferramentas de build poderosas baseadas no *Gradle*.
- Emuladores Android rápidos e configuráveis para testar aplicações em diferentes dispositivos e versões do Android.
- Um depurador robusto para identificar e corrigir problemas de código.
- Ferramentas de profiling para analisar o desempenho (uso de *CPU*, memória, rede) do aplicativo.

- Um layout editor visual para criar interfaces de usuário.

Para projetos React Native CLI, o *Android Studio* é indispensável para gerenciar o projeto Android nativo subjacente, configurar permissões, integrar bibliotecas nativas, depurar problemas específicos da plataforma e realizar o build final do aplicativo para distribuição (Andrews; Eisenman, 2012, 2018). Ele serve como o principal ambiente para qualquer configuração ou desenvolvimento que precise ser feito diretamente no código Java/Kotlin do Android (Andrews, 2012).

3 METODOLOGIA

A metodologia adotada neste trabalho caracteriza-se como uma pesquisa aplicada, de abordagem qualitativa, com procedimentos baseados em pesquisa de campo e no desenvolvimento empírico de software. O objetivo principal foi desenvolver uma solução para um problema real enfrentado pela empresa FF Transportes LTDA por meio da construção de uma ferramenta tecnológica personalizada.

O processo de desenvolvimento apoiou-se nos conceitos de ciclo de vida, levantamento de requisitos, desenvolvimento iterativo e arquitetura modular apresentados no Capítulo 2, em especial na Seção 2.1. Conforme discutido no referencial teórico, o levantamento e a análise de requisitos constituem a fase inicial e crítica do ciclo de vida do software, definindo o que o sistema deve fazer a partir das necessidades dos stakeholders (Pressman e Maxim, 2014). A comunicação direta com os usuários e a observação do ambiente de trabalho permitem identificar demandas reais e priorizar funcionalidades, reduzindo o risco de uma solução desalinhada às expectativas (Sommerville, 2015).

Neste capítulo são detalhados os processos, ferramentas, linguagens de programação e bibliotecas utilizadas ao longo do desenvolvimento, bem como as abordagens adotadas para autenticação de usuários, comunicação entre cliente e servidor (via HTTP e WebSocket) e manipulação de dados cadastrados e em tempo real.

3.1 LEVANTAMENTO DE REQUISITOS

Em consonância com a importância atribuída à fase de requisitos no referencial teórico (Seção 2.1.1), o levantamento de requisitos deste projeto foi realizado por meio de comunicação direta com os stakeholders da FF Transportes LTDA. Foi realizada uma reunião presencial com a presença de representantes de todos os setores da empresa: diretor operacional, CEO, atendentes e demais membros da equipe. O objetivo foi levantar informações detalhadas sobre os processos internos, as rotinas operacionais, os métodos utilizados para rastreamento e controle de viagens e as ferramentas tecnológicas então empregadas.

Durante o encontro, foram utilizadas perguntas abertas e direcionadas para compreender os fluxos de trabalho existentes, identificar etapas críticas no processo de monitoramento logístico e registrar as demandas apontadas por cada setor. A condução da etapa envolveu escuta ativa dos participantes, categorização das necessidades por área de atuação e consolidação dos dados em tópicos organizados, de modo a orientar o planejamento das funcionalidades do sistema de rastreamento e a priorização do que seria implementado nas fases seguintes do ciclo de vida.

Com base nas informações coletadas, foram identificados os seguintes pontos como principais demandas da equipe da FF Transportes LTDA:

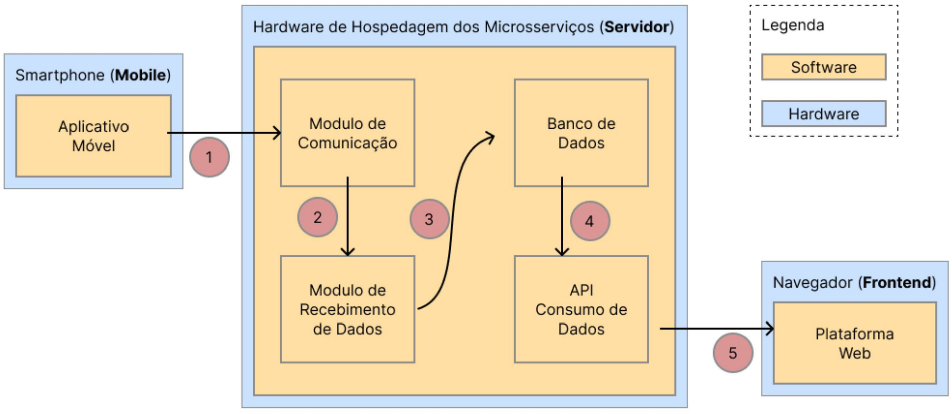
- Ausência de um sistema centralizado para visualização em tempo real da localização dos veículos;
- Falta de controle no histórico de viagens e rotas percorridas;
- Inexistência de rastreamento próprio em cargas que não possuem exigência contratual, resultando em falta de visibilidade operacional;
- Necessidade de uma solução que cubra todas as cargas, inclusive as que não contam com sistemas externos de rastreamento;
- Interesse em reduzir a dependência de ferramentas terceirizadas e concentrar os dados em uma plataforma própria e acessível.

3.2 FLUXO DOS DADOS DO SISTEMA

A proposta de sistema adotada para o desenvolvimento da solução de rastreamento da FF Transportes LTDA baseia-se em uma arquitetura modular, projetada para assegurar clareza estrutural, eficiência operacional e facilidade de escalabilidade. Essa abordagem permite a divisão do sistema em componentes distintos, cada um responsável por uma função específica dentro do fluxo de captura, transmissão, armazenamento e visualização das coordenadas geográficas dos veículos em tempo real.

Conforme ilustrado na Figura 1, a seguir são descritas as etapas que compõem o fluxo proposto de funcionamento do sistema, destacando o papel de cada componente dentro da arquitetura modular adotada.

Figura 1 – Diagrama simplificado do funcionamento do sistema de rastreamento



Fonte: (Autor, 2024)

1. O dispositivo móvel captura as coordenadas geográficas (latitude e longitude) utilizando o GPS e transmite essas informações ao servidor por meio de um módulo de comunicação. Esse processo ocorre de forma contínua, garantindo atualizações em tempo real.
2. O módulo de comunicação estabelece uma conexão eficiente entre o smartphone e o servidor, assegurando a transmissão segura e confiável dos dados de geolocalização.
3. O módulo de recebimento de dados no servidor processa as coordenadas recebidas, valida sua integridade e, em seguida, as armazena em um banco de dados otimizado para dados de localização.
4. O banco de dados armazena os dados de maneira segura e flexível, possibilitando futuras consultas ou atualizações. Ele é projetado para lidar com uma alta taxa de atualizações em tempo real, essencial para sistemas de rastreamento.
5. A API de consumo de dados fornece os meios necessários para que sistemas externos, como a plataforma web, possam acessar, modificar ou excluir os dados armazenados no banco. A API facilita a comunicação entre o banco de dados e outras aplicações, mesmo sem uma interface gráfica própria.
6. A plataforma web permite ao usuário final acessar os dados de rastreamento de forma prática, mediante consulta à API sob demanda, além de realizar cadastros, atualizações ou exclusões de dados, conforme necessário.

3.3 INTERFACE DE DESENVOLVIMENTO

Para a implementação do sistema, optou-se pelo Visual Studio Code (VS Code) como IDE. Essa escolha foi motivada por suas características de leveza, gratuidade e ampla compatibilidade com diversas linguagens de programação e frameworks utilizados no projeto, como JavaScript e Next.js. Conforme LIMA (2022), uma IDE unifica ferramentas essenciais para o desenvolvimento, como edição de código-fonte, acesso a terminais e depuração. O VS Code se destaca justamente por essa integração, oferecendo recursos como destaque de sintaxe, autocompletar e um depurador integrado que foram cruciais para a agilidade e eficiência no desenvolvimento do sistema.

O Apêndice A ilustra a interface do Visual Studio Code (VS Code), apresentando uma seção do código-fonte do backend. A figura destaca a funcionalidade de autocompletar (também conhecido como IntelliSense) que o VS Code oferece.

3.4 LINGUAGENS DE PROGRAMAÇÃO

A linguagem predominante no desenvolvimento deste projeto é o JavaScript, amplamente utilizada no backend, frontend e aplicação mobile. Sua popularidade e

versatilidade a tornam ideal para o desenvolvimento full-stack. O JavaScript é uma linguagem interpretada, orientada a objetos e baseada em eventos, características que facilitam a criação de interfaces dinâmicas e a interação em tempo real entre cliente e servidor. A ampla gama de frameworks e bibliotecas disponíveis contribui para o desenvolvimento rápido e eficiente de aplicações estáveis e escaláveis.

3.5 BACKEND

3.5.1 Design pattern

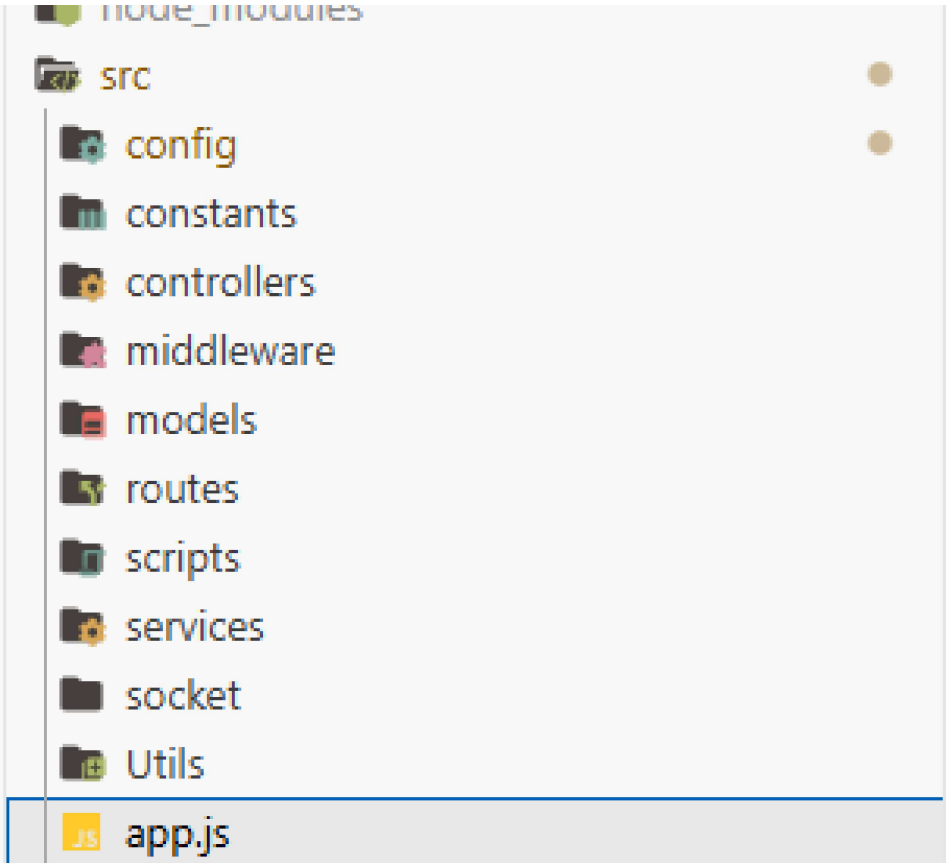
O padrão de arquitetura Model-View-Controller (MVC) foi adotado para estruturar o desenvolvimento do sistema, proporcionando uma separação clara de responsabilidades entre as diferentes camadas da aplicação.

O MVC foi aplicado na camada do backend, seguindo sua definição clássica e dividindo o sistema em três componentes interligados:

- Models: No sistema, o Model abrigou as entidades Endereço, Admin, Cliente, Condutor e Viagem. Os Models gerenciaram a lógica de negócios e as interações com o banco de dados.
- Views: Na arquitetura do backend, a View foi representada pelas respostas formatadas, como em formato JavaScript Object Notation (JSON), enviadas ao cliente. Assim, foram responsáveis apenas pela apresentação dos dados processados pelos Controllers.
- Controllers: No backend, os Controllers receberam as requisições do cliente, tanto da interface web quanto da aplicação móvel. Exemplos de Controllers implementados incluem adminController, authController, driverController e travelsController, cada um contendo a lógica específica para suas respectivas funcionalidades e a preparação das respostas.

A estrutura geral do design pattern MVC aplicado neste sistema pode ser visualizada na Figura 2. Esta imagem ilustra a relação e interação entre esses três componentes para formar a aplicação.

Figura 2 – Design Pattern MVC aplicado no backend



Fonte: (Autor, 2024)

3.5.2 Ferramentas e tecnologias de desenvolvimento

A seleção das ferramentas e tecnologias para o desenvolvimento do backend do sistema foi um processo guiado pelos requisitos específicos do projeto, visando garantir a performance, escalabilidade, segurança e manutenibilidade da aplicação. Esta subseção detalha as principais escolhas tecnológicas e o papel de cada uma na construção da solução.

3.5.2.1 Bibliotecas e versões

O Quadro 1 detalha as principais bibliotecas utilizadas no backend, juntamente com suas respectivas versões, garantindo a rastreabilidade do ambiente de desenvolvimento.

Quadro 1 – Bibliotecas utilizadas e suas versões.

Bibliotecas	Versões
Node.js	20.15.1
Express	4.18.2
Socket.io	4.7.2
Body-parser	1.20.2
Bcrypt	5.1.1
Jsonwebtoken	9.0.2
Mongoose	8.0.2

Fonte: (Autor, 2024)

3.5.2.2 Plataforma e servidor

O Node.js foi adotado como a plataforma principal para a execução do backend. Sua arquitetura baseada em eventos e I/O não-bloqueante foi crucial para lidar com o grande volume de requisições simultâneas inerentes a um sistema de rastreamento em tempo real, proporcionando respostas rápidas e eficiência no processamento de dados. Para a gestão das rotas e a comunicação HTTP, o Express, um framework web minimalista, foi empregado. Ele facilitou a estruturação das APIs RESTful e a implementação de middlewares para o tratamento padronizado das requisições, como a autenticação e a validação de dados. O Apêndice B ilustra a aplicação de middlewares em uma rota.

O Express também foi fundamental na definição da arquitetura RESTful do sistema, permitindo a criação de um conjunto de rotas para gerenciamento e interação com o backend. O Quadro 2 sumariza as principais rotas da API de administração implementadas:

Quadro 2 – Principais rotas da API de Administração.

Rota	Método HTTP	Descrição
/register/admin	POST	Registra administrador.
/register/client	POST	Registra cliente.
/register/driver	POST	Registra motorista.
/user	GET	Lista usuários.
/user/:type/:id	GET	Obtém usuário por ID.
/user/:type/:id	DELETE	Exclui usuário.
/user/:type/:id	PUT	Edita usuário.
/register/address	POST	Registra endereço.
/address	GET	Lista endereços.
/address/:id	DELETE	Exclui endereço.
/register/travel	POST	Registra viagem.
/travel/:id	PUT	Atualiza viagem.
/travels	GET	Lista viagens.
/travel/:id	GET	Obtém viagem por ID.
/travels/:id	DELETE	Exclui viagem.

Fonte: (Autor, 2024)

Por fim, o body-parser foi utilizado para processar os dados recebidos em requisições HTTP, especificamente nos formatos JSON e URL-encoded, simplificando o tratamento das informações enviadas pelos clientes e dispositivos móveis.

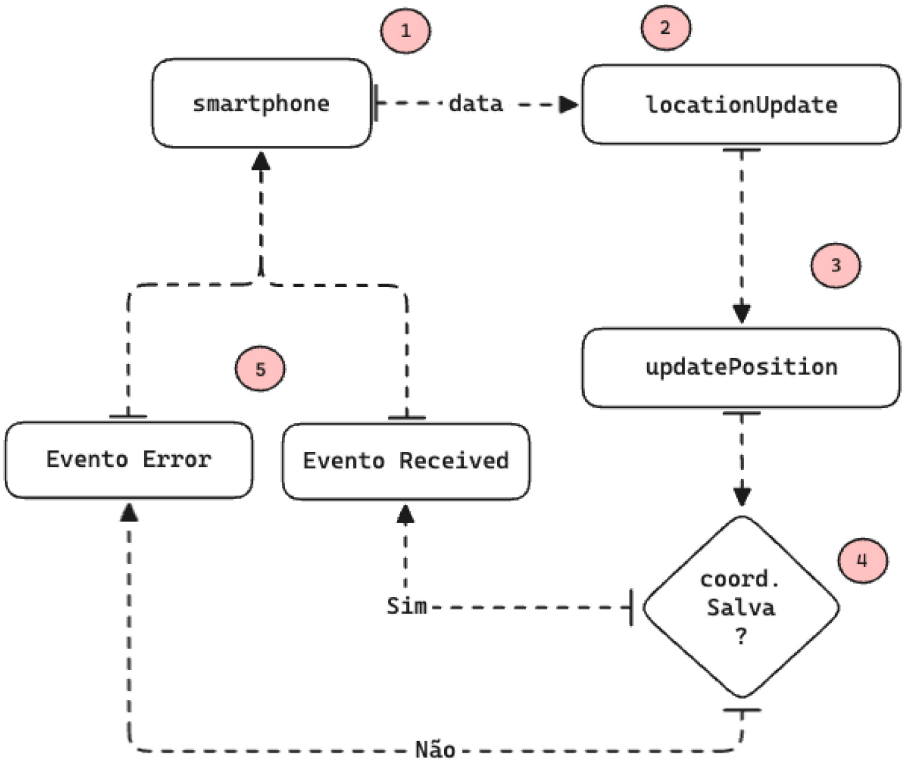
3.5.2.3 Comunicação em tempo real

O rastreamento de veículos exige que a posição do dispositivo seja enviada e atualizada com frequência, de modo que a transportadora visualize a localização em tempo (quase) real no painel web. Requisições HTTP tradicionais (cliente pergunta, servidor responde) não são ideais para esse cenário: seria necessário fazer várias requisições em intervalo curto (polling), sobrecarregando o servidor e gerando atraso. Por isso, complementando a estrutura de comunicação descrita anteriormente, a comunicação em tempo real foi implementada com o Socket.io, permitindo um canal de comunicação bidirecional e persistente entre o aplicativo móvel (motorista) e o servidor. Assim, o servidor pode receber atualizações de posição assim que o dispositivo as envia e, quando necessário, retransmiti-las aos clientes conectados ao painel.

O Socket.io baseia-se no conceito de eventos: o cliente emite um evento (por exemplo, *locationUpdate*) com um payload (coordenadas e identificador da viagem), e o servidor escuta esse evento e reage. O fluxo de processamento das atualizações de localização no servidor foi projetado para garantir a integridade e a confiabilidade dos

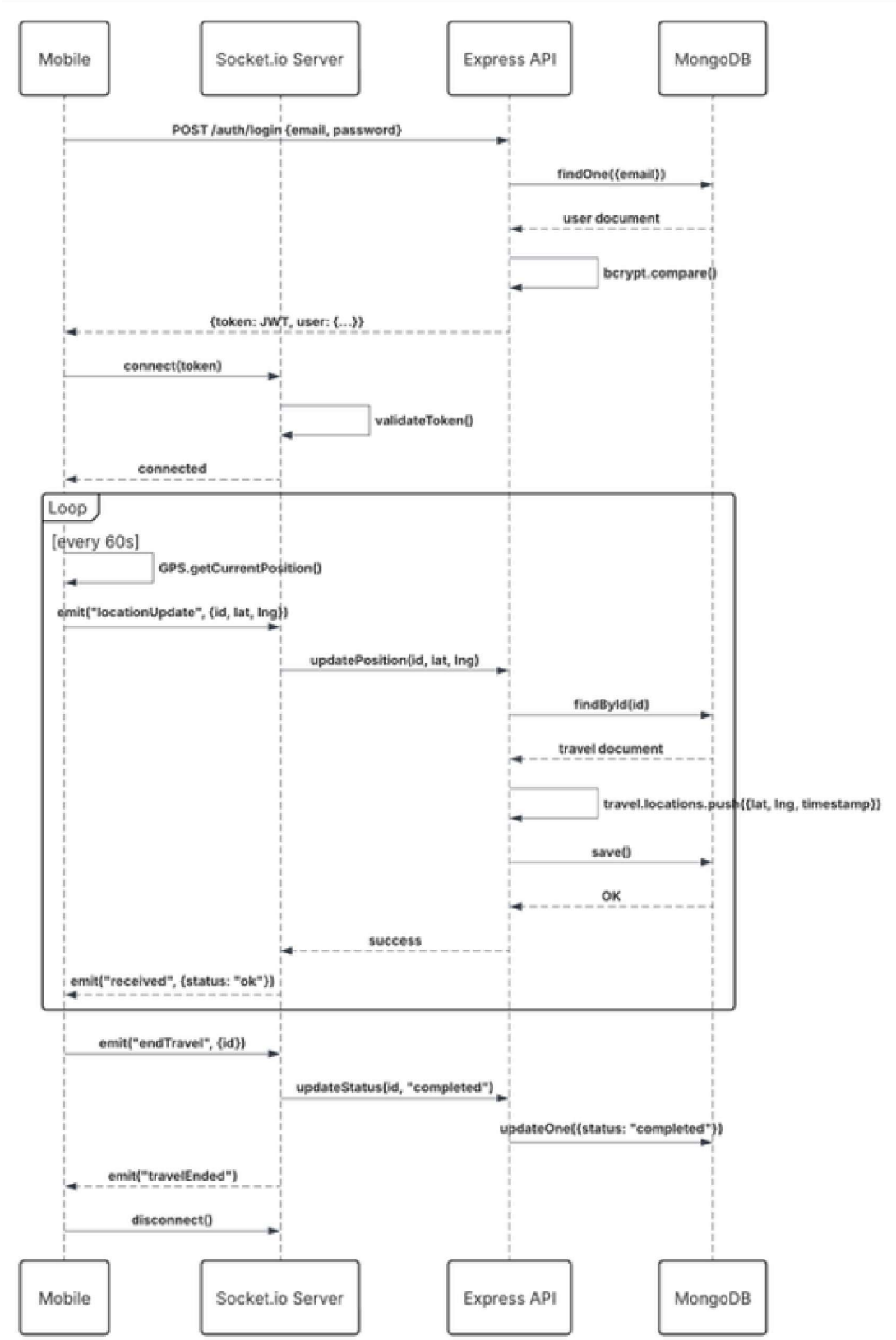
dados recebidos, validando as informações antes de persistir e enviando confirmação ou mensagem de erro ao cliente. As Figuras 3 e 4 ilustram esse fluxo em dois níveis de detalhe: a primeira mostra o processamento da atualização no servidor; a segunda apresenta o diagrama de sequência completo, da autenticação à desconexão.

Figura 3 – Fluxo de processamento da atualização de localização no servidor



Fonte: (Autor, 2024)

Figura 4 – Diagrama de sequência do rastreamento GPS



Fonte: (Autor, 2024)

O processo no servidor inicia-se com o recebimento das coordenadas geográficas (latitude e longitude) e do identificador do dispositivo ou da viagem emissor. Os dados são desestruturados e encaminhados à rotina de atualização no banco de dados. A implementação do listener do evento *locationUpdate* está ilustrada no Apêndice C.

A lógica central de atualização fica na função *updatePosition* (Apêndice D). Ela é responsável por:

- validar a presença dos dados essenciais (ID da viagem, latitude e longitude);
- buscar na coleção de viagens o documento correspondente ao ID fornecido;
- adicionar as novas coordenadas ao array de localizações (*locations*) desse documento e persistir as alterações no banco.

Em caso de sucesso, uma confirmação é enviada ao cliente por meio do evento *received*, o que permite ao aplicativo móvel saber que a posição foi registrada. Em caso de dados inválidos ou de exceção durante a busca ou o salvamento, a falha é tratada de forma segura e o cliente recebe uma mensagem de erro padronizada, evitando que o sistema assuma que a atualização foi bem-sucedida quando não foi.

Dessa forma, a comunicação em tempo real consolidou-se como base do rastreamento contínuo dos veículos: o motorista envia a posição periodicamente pelo aplicativo, o servidor valida e persiste os dados e pode repassá-los ao painel da transportadora, permitindo acompanhamento eficiente e seguro da frota.

3.5.2.4 Segurança e autenticação

No quesito segurança, a criptografia de senhas foi realizada com a biblioteca bcrypt. Este algoritmo de hashing foi escolhido para proteger dados sensíveis, impedindo que senhas sejam armazenadas em texto plano e prevenindo ataques de força bruta ou dicionário. Conforme demonstrado no Apêndice E, a senha é convertida em um hash antes de ser persistido no banco de dados.

Para a autenticação e autorização no sistema, a biblioteca jsonwebtoken foi adotada. Essa abordagem permitiu a geração e validação de tokens JSON Web Token (JWT), garantindo a autenticidade e a integridade das requisições. Após a validação das credenciais do usuário na rota de login, um JWT é gerado (Apêndice F) e fornecido ao cliente. As requisições subsequentes à API são então validadas por meio da inspeção desse token, verificando sua assinatura e expiração para assegurar que apenas usuários autorizados acessem recursos protegidos (Apêndice G).

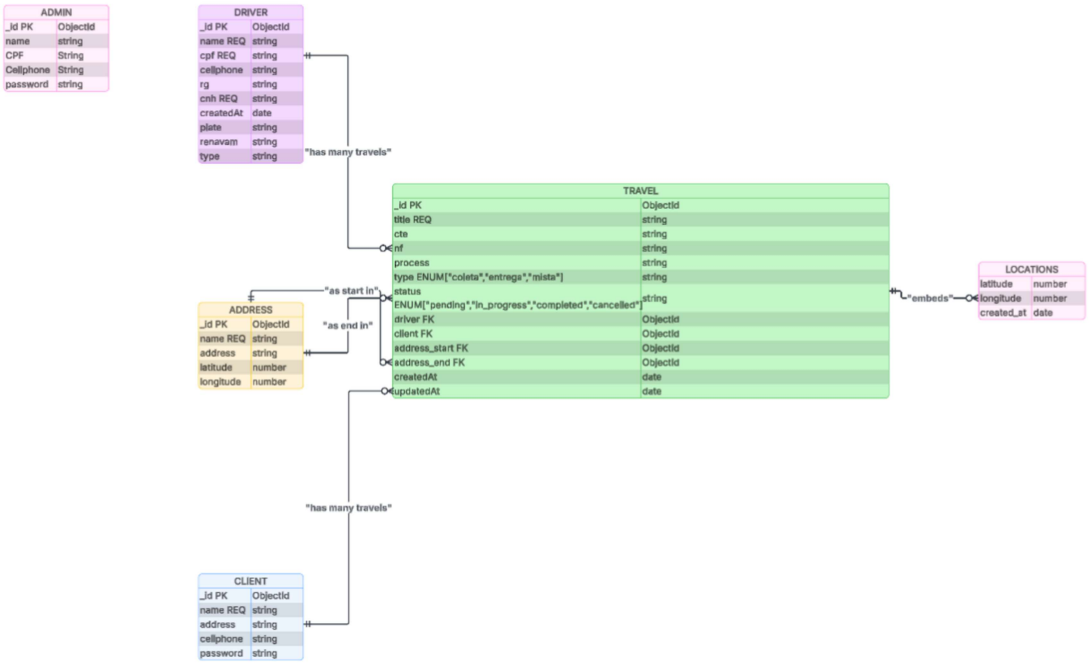
3.5.2.5 Persistência de dados

A persistência dos dados do sistema foi implementada com o banco MongoDB, um sistema NoSQL orientado a documentos. A escolha do MongoDB deveu-se à sua

flexibilidade para armazenar dados heterogêneos, como informações de geolocalização e de usuários, que se adequam bem ao modelo de documentos. Para a interação com o banco, foi empregado o Mongoose como camada de abstração Object-Document Mapper (ODM), responsável pela definição de esquemas, pela manipulação das entidades e pela conexão com o banco (Apêndice H).

O modelo de dados e os relacionamentos entre as entidades do sistema estão representados na Figura 5. Como exemplo de entidade central, o *TravelSchema* e sua definição com parâmetros e tipagem são apresentados no Apêndice I.

Figura 5 – Diagrama de entidade-relacionamento do sistema

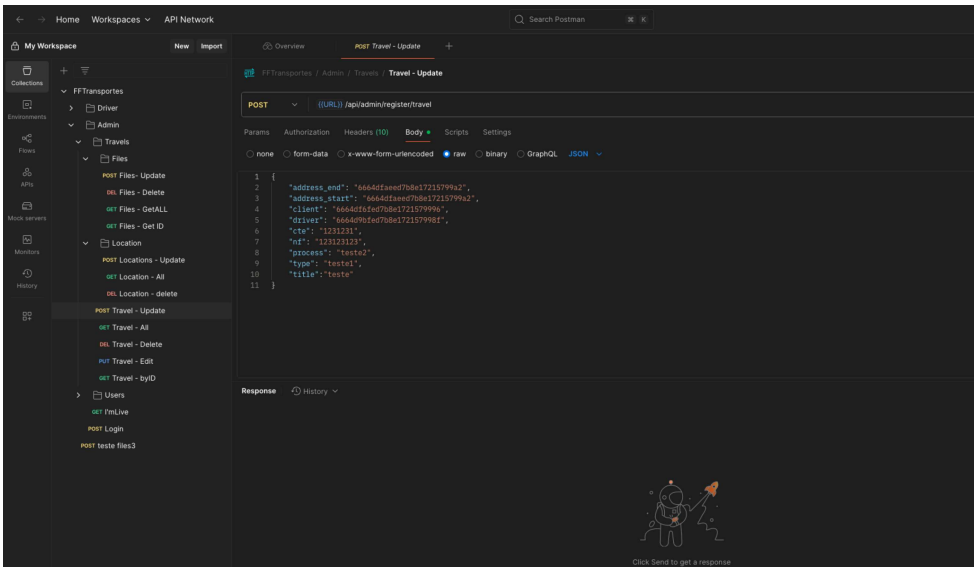


Fonte: (Autor, 2024)

3.5.3 Testes aplicados

Foi utilizado o software Postman como ferramenta de teste das rotas do backend durante a fase de desenvolvimento. Todas as requisições HTTP dos métodos POST, GET, PUT e DELETE foram cadastradas no ambiente da aplicação com o objetivo de simular os comportamentos esperados do sistema. A Figura 6 ilustra a interface geral do Postman e a organização das requisições.

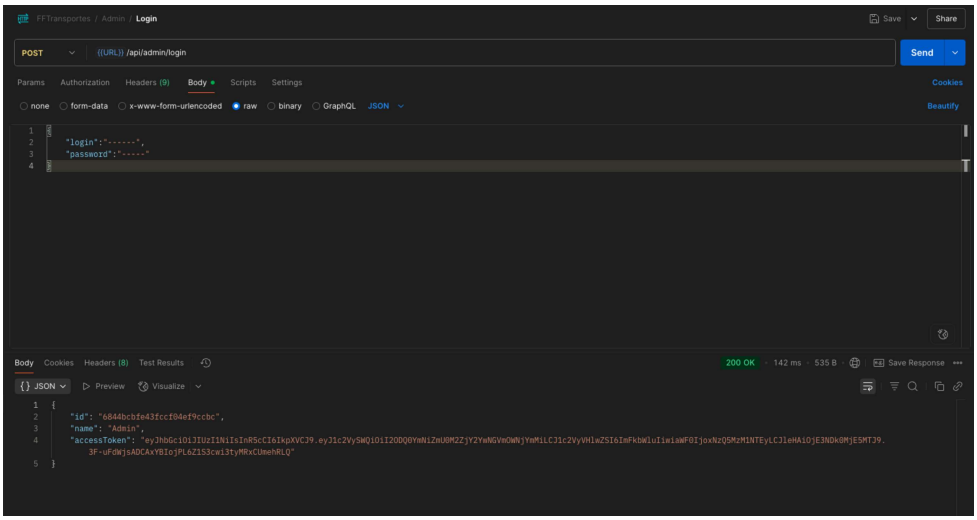
Figura 6 – Interface do Postman



Fonte: (Autor, 2023)

Foram realizados testes de criação e manipulação das entidades de usuário, de cliente e de viagem, antecedendo a etapa de integração com as interfaces web e mobile. Esses testes incluíram a verificação prévia de todas as validações das informações enviadas em formato JSON, abrangendo campos obrigatórios, tipos de dados e a estrutura esperada nas requisições, bem como a validação dos códigos de status HTTP retornados. A Figura 7 ilustra um teste realizado na rota de autenticação de usuário, evidenciando o envio dos dados e a resposta retornada pelo servidor.

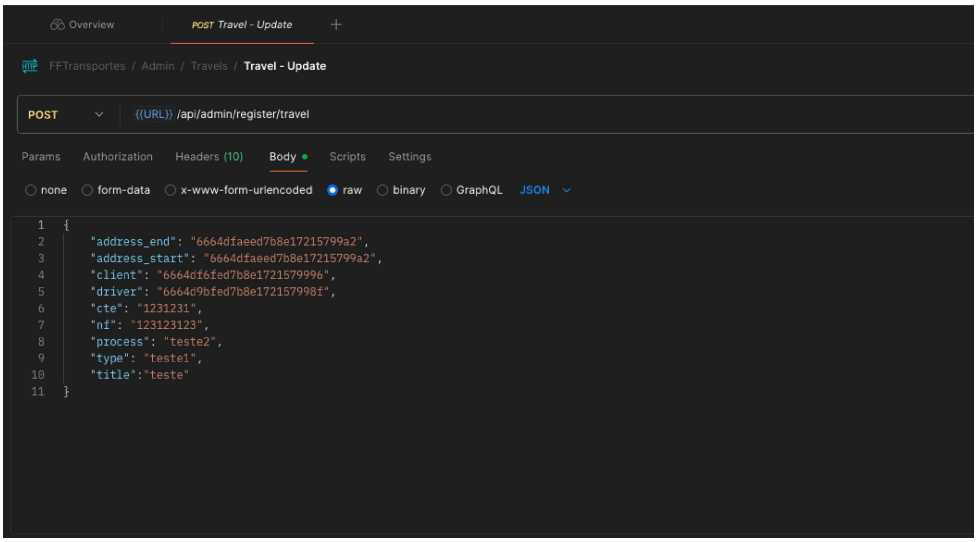
Figura 7 – Teste da rota de autenticação de usuário no Postman



Fonte: (Autor, 2023)

Um exemplo adicional de teste realizado foi a criação de uma nova entidade de viagem. A Figura 8 demonstra a requisição POST enviada à rota /register/travel, contendo os dados necessários para o registro da viagem e a resposta de sucesso retornada pelo servidor, incluindo o identificador único da viagem recém-criada. Para essa operação, os campos essenciais enviados no corpo da requisição incluem os IDs dos endereços de início e fim da viagem (address_start e address_end), o ID do cliente (client), o ID do motorista (driver), além de dados específicos da viagem como cte, nf, process, type e title.

Figura 8 – Teste da rota de criação de viagem no Postman



Fonte: (Autor, 2023)

3.6 FRONTEND

Nesta etapa do projeto, foi desenvolvido um portal voltado à transportadora, com foco em funcionalidades essenciais para o controle de viagens e a gestão do sistema. Entre os principais recursos implementados estão o cadastro, a edição e a exclusão de usuários, bem como o gerenciamento de endereços conhecidos pela empresa.

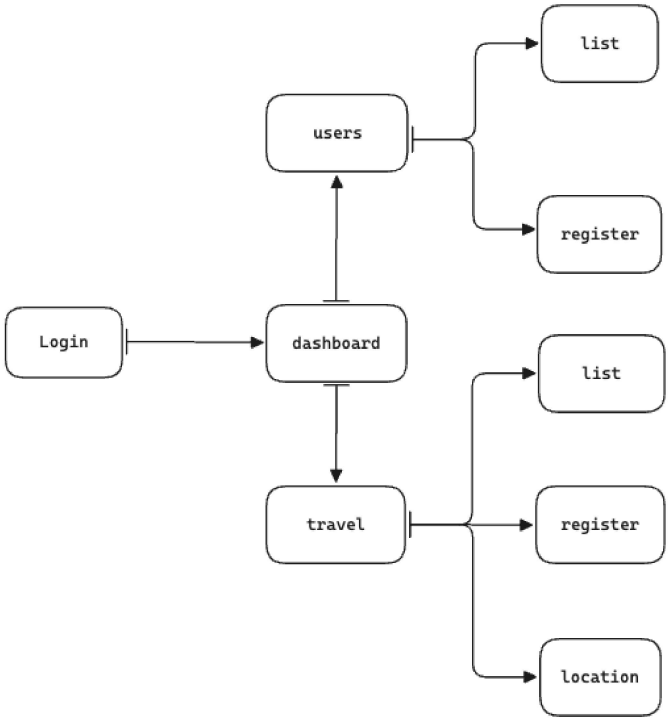
Durante a execução do sistema, as seguintes páginas foram implementadas no frontend da aplicação:

- login: responsável pela autenticação de usuários e controle de acesso à plataforma;
- *dash/travel/register*: destinada ao cadastro de novas viagens no sistema;
- *dash/travel/list*: exibe a listagem das viagens registradas, com filtros e paginação;
- *dash/travel/list/[id]*: exibe os detalhes da viagem em questão, apresentando as informações sobre os dados da viagem e seu rastreamento.
- *dash/travel/location*: destinada ao cadastro de novas localizações conhecidas no sistema;
- *dash/user/register*: responsável pelo cadastro de novos usuários no sistema;
- *dash/user/list*: apresenta a listagem dos usuários cadastrados, com opções de edição e exclusão.

Conforme ilustrado na Figura 9, o fluxo de navegação da aplicação inicia-se

sempre pela página de login. Essa página foi utilizada para verificar se o usuário já está autenticado junto ao servidor e se o token de acesso encontra-se armazenado localmente no *localStorage*. Se essas condições forem atendidas, o sistema redireciona automaticamente o usuário para a página principal da aplicação: /dash/travel/list.

Figura 9 – Fluxo de navegação da aplicação



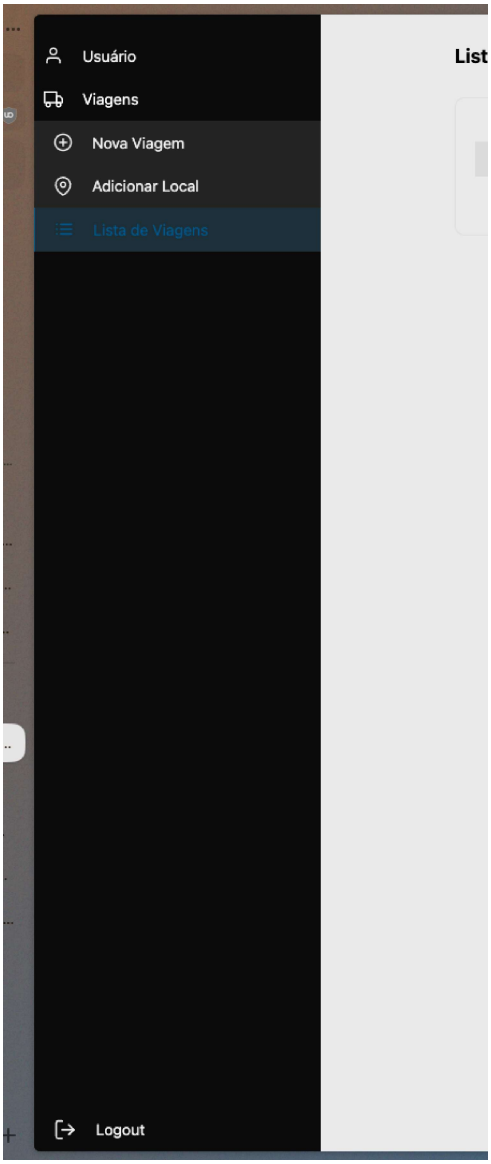
Fonte: (Autor, 2023)

Após a autenticação, o usuário é redirecionado para a página principal do sistema. Nela, é apresentado o menu lateral, que constitui o principal meio de navegação da aplicação, conforme demonstrado na Figura 10. Esse menu reúne os acessos às diferentes seções do sistema, permitindo uma navegação eficiente e organizada. A opção que representa a página atualmente acessada é ressaltada por uma coloração destacada, facilitando a identificação visual da seção em uso e contribuindo para uma melhor experiência de usabilidade.

Para uma melhor organização e usabilidade, as funcionalidades foram agrupadas em duas seções principais: Usuário e Viagens. A seção Usuário abrange as páginas relacionadas à gestão de diferentes perfis do sistema, incluindo clientes, condutores e administradores. Já a seção Viagens contempla as funcionalidades voltadas ao controle de rotas e destinos, como o cadastro de endereços e o gerenciamento das

viagens propriamente ditas.

Figura 10 – Menu lateral com navegação estruturada



Fonte: (Autor, 2023)

3.6.1 Página de cadastro de usuários

A página dash/user/register permite o cadastro de novos usuários no sistema, classificados em três perfis: cliente, condutor e administrador. O formulário adapta-se conforme o tipo de perfil selecionado, exigindo diferentes campos de preenchimento. Por padrão, apenas usuários com perfil administrativo têm acesso a essa funcionalidade.

3.6.2 Página de cadastro de endereços

A página dash/travel/location foi implementada com o objetivo de permitir o registro de endereços previamente conhecidos e utilizados com frequência pela transportadora. Essa funcionalidade visa otimizar o processo de criação de novas viagens, evitando o retrabalho na inserção de dados e garantindo padronização nas rotas cadastradas.

3.6.3 Página de cadastro de viagens

A página dash/travel/register representa uma das principais funcionalidades da aplicação, sendo responsável pelo registro de novas viagens no sistema. Seu papel é central no fluxo operacional da transportadora, pois reúne todas as informações necessárias para a execução e rastreamento das viagens.

O formulário de cadastro é estruturado para garantir a integridade dos dados e a padronização das informações. Nele, o usuário deve selecionar o condutor responsável por meio de uma lista de condutores previamente cadastrados no sistema, o que assegura a consistência entre os dados pessoais e operacionais do motorista. Da mesma forma, o cliente associado à viagem também é selecionado a partir de uma listagem existente.

Além disso, as localizações de origem e destino da viagem são escolhidas com base em endereços previamente registrados, facilitando o preenchimento e evitando redundância de dados. Esses endereços são vinculados diretamente ao processo de rastreamento e logística.

Complementando o cadastro, o sistema solicita informações obrigatórias referentes ao transporte, tais como: número do Conhecimento de Transporte Eletrônico (CT-e), número da nota fiscal associada, número do processo logístico interno e o tipo da viagem (por exemplo: entrega, coleta ou rota mista). Todos esses campos são validados antes do envio, e o sistema exibe mensagens de retorno em caso de erro ou preenchimento incompleto.

3.6.4 Página de detalhes da viagem

A página dash/travel/list/[id] é responsável por exibir os detalhes completos de uma viagem previamente cadastrada no sistema. Essa interface permite que administradores e usuários com permissão visualizem todas as informações inseridas no momento do cadastro, bem como dados complementares capturados ao longo da execução da viagem.

Entre os dados apresentados, estão: condutor responsável, cliente vinculado,

endereço de origem e destino, número do CT-e, número da nota fiscal, número do processo logístico e o tipo de viagem. Esses dados são exibidos de forma estruturada, visando facilitar a leitura e conferência por parte do usuário.

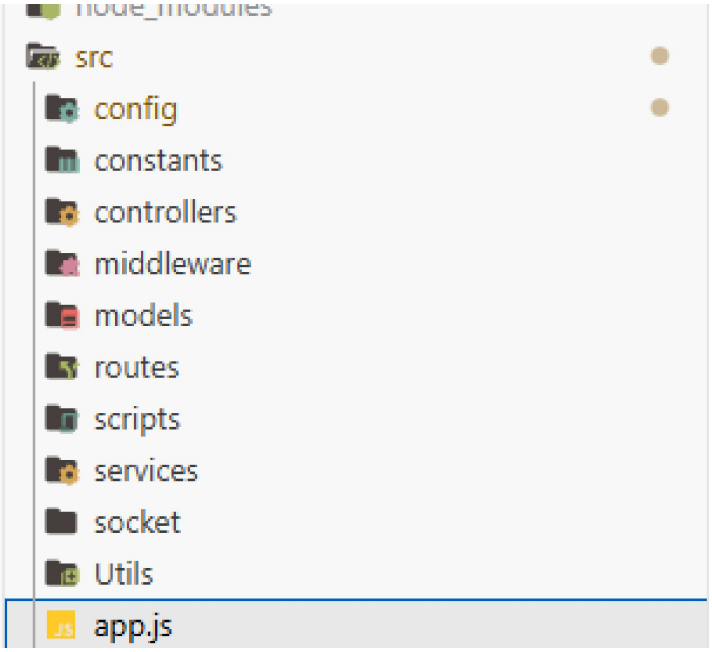
Além dessas informações estáticas, a página também exibe os dados de localização associados à viagem, incluindo coordenadas geográficas como latitude e longitude. Essas coordenadas são obtidas por meio do sistema de rastreamento embarcado no dispositivo do condutor e armazenadas automaticamente durante o percurso.

Para facilitar a análise e o monitoramento da viagem, a interface oferece a funcionalidade de redirecionamento para o Google Maps. Ao clicar sobre um botão específico, o administrador pode abrir as coordenadas em uma nova aba do navegador, utilizando o serviço do Google Maps para visualizar a localização exata do ponto registrado.

3.6.5 COMPONENTIZAÇÃO DA INTERFACE

A aplicação foi desenvolvida seguindo o paradigma de componentização do React. Interfaces complexas foram divididas em componentes menores, reutilizáveis e com responsabilidades bem definidas, como cabeçalhos, formulários de login e quadros de dados de viagens. Isso promoveu a modularidade e a manutenibilidade. A Figura 11 apresenta a estrutura de pastas do projeto, destacando a separação lógica entre os componentes reutilizáveis utilizados na interface.

Figura 11 – Estrutura de pastas de componentes do frontend



Fonte: (Autor, 2023)

Como exemplo prático de componentização, pode-se citar a construção da página de autenticação, ilustrada no Apêndice J. Nessa tela, tanto os campos de entrada (inputs) quanto o botão de envio são instâncias de componentes previamente definidos, nomeadamente o InputCustom, apresentado no Apêndice L, e o ButtonCustom, ilustrado no Apêndice K.

3.6.6 Comunicação com backend

A biblioteca Axios foi empregada para realizar as requisições HTTP assíncronas ao backend. Sua API simples e baseada em Promises facilitou a comunicação com os endpoints da API RESTful.

Para organizar as interações com o backend, os serviços de comunicação foram componentizados. Cada serviço, responsável por um conjunto específico de operações (como autenticação, gestão de usuários ou viagens), encapsula as configurações do Axios e as chamadas aos respectivos endpoints, utilizando um componente principal de conexão implementado com Axios.

O Apêndice M apresenta o componente principal de comunicação. Este componente é projetado para receber configurações específicas da solicitação desejada, tais como a rota (endpoint), o método de requisição (method), os dados a serem enviados (data) e as configurações de cabeçalhos (headers).

Já no Apêndice N, pode-se observar uma das implementações que se utilizam deste componente central de comunicação para o envio do registro de uma nova viagem. Após o envio, os tratamentos de resposta são aplicados, utilizando o handleOpenAlert, um modal de alerta em formato *pop-up*, para informar o usuário. Em caso de sucesso, a mensagem "Viagem cadastrada com sucesso"é exibida; em caso de erro, a mensagem "Viagem não cadastrada"é apresentada. Essa abordagem permitiu a centralização da lógica de requisições e o tratamento padronizado das respostas em toda a aplicação.

3.6.7 Formulários e validação

Para o gerenciamento dos formulários da aplicação, foi adotada a biblioteca React Hook Form, amplamente utilizada em aplicações React pela sua leveza e eficiência no controle de formulários. No contexto deste sistema, o React Hook Form foi empregado na construção dos formulários de cadastro de usuários, cadastro de viagens, cadastro de endereços e autenticação (login). Essa ferramenta foi responsável por capturar e controlar as mudanças nos campos de entrada (inputs) desses formulários, além de gerenciar seus estados internos de forma performática.

Em conjunto com o React Hook Form, foi utilizada a biblioteca Yup, que atua na definição dos esquemas de validação. Com ela, foi possível estabelecer regras específicas de tipagem e obrigatoriedade para cada campo dos formulários, garantindo a integridade dos dados antes de sua submissão ao servidor. A integração entre as duas bibliotecas permitiu validar os formulários de maneira declarativa e consistente, fornecendo mensagens de erro claras ao usuário final sempre que alguma regra fosse violada.

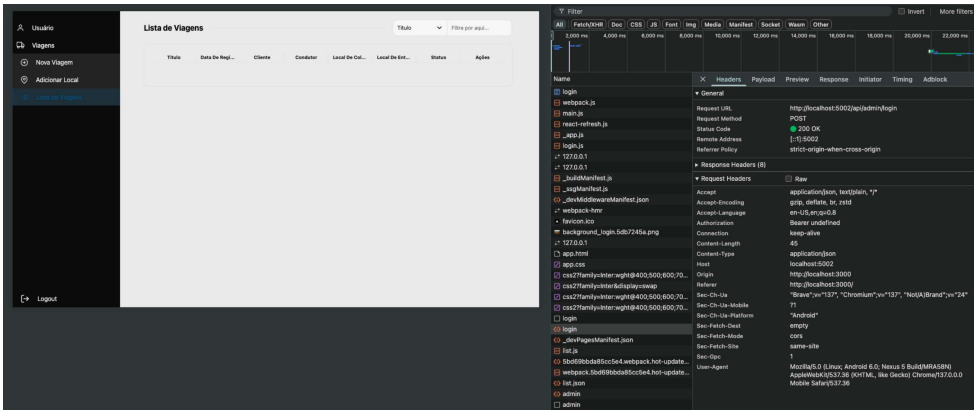
3.6.8 Testes aplicados

Durante a fase de testes da interface web do sistema, o navegador Brave foi empregado como ambiente principal para simular e validar o funcionamento da aplicação. A escolha do Brave, baseado na arquitetura Chromium, garantiu acesso a um conjunto abrangente de ferramentas de desenvolvimento nativas, cruciais para a inspeção e análise detalhada do comportamento do frontend.

Foram realizados testes funcionais com o objetivo principal de validar a integração da plataforma com a API do backend. Para isso, o componente de rede (*network*) das ferramentas de desenvolvimento do navegador foi utilizado de forma sistemática. Essa abordagem permitiu a inspeção detalhada tanto das informações enviadas (*requests*) quanto das respostas recebidas (*responses*), garantindo a integridade e a correção da comunicação entre o frontend e o backend. Em paralelo, foi possível visualizar a renderização em tela dos dados manipulados, assegurando que

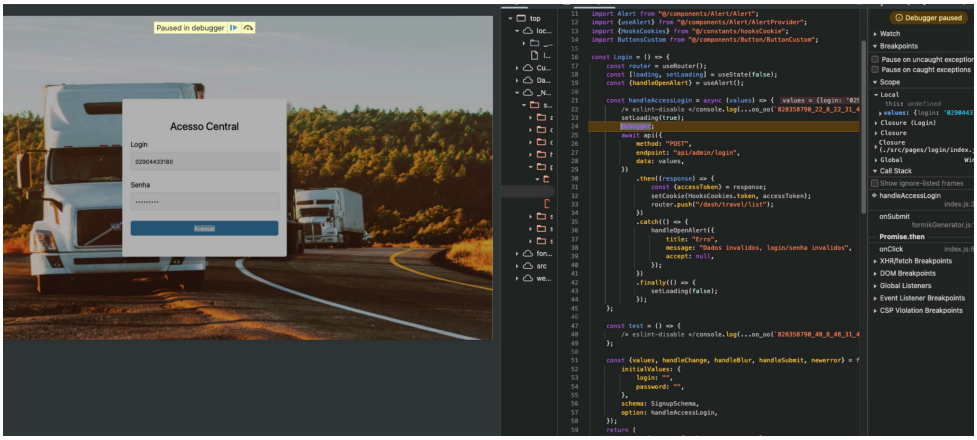
a interface de usuário apresentasse as informações de maneira correta e esperada. Adicionalmente, foram conduzidos testes de usabilidade a fim de garantir uma *User Experience* (UX) intuitiva e fluida, focando na facilidade de navegação e na clareza das interações.

Figura 12 – Detalhes da Comunicação de Rede na Autenticação do Usuário



Fonte: (Autor, 2023)

Figura 13 – Análise detalhada de componentes por meio do *Debugger*



Fonte: (Autor, 2023)

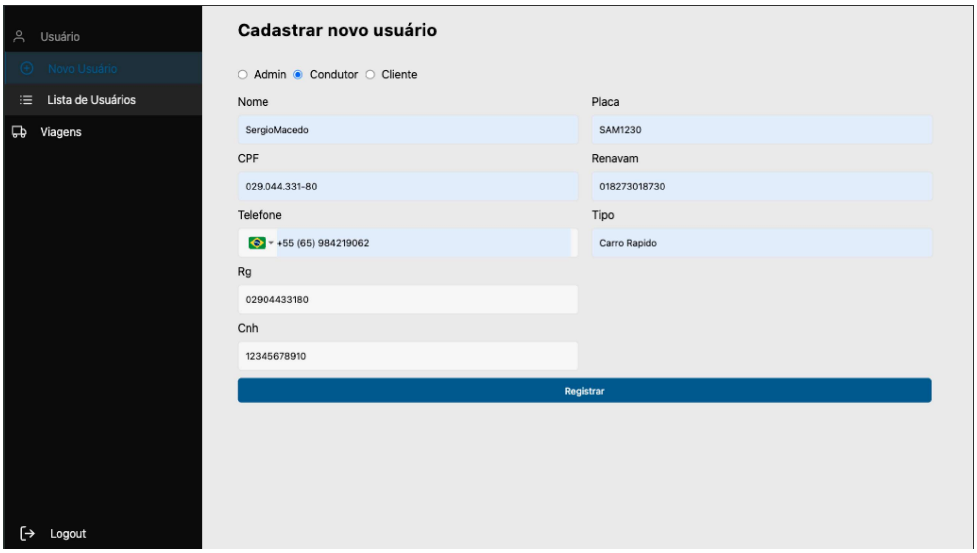
As ferramentas de inspeção do navegador desempenharam um papel fundamental durante todo o processo de desenvolvimento e validação, sendo empregadas para:

- Monitorar requisições HTTP e respostas: Verificando o status, os cabeçalhos e os corpos das requisições para depurar problemas de comunicação com a API, conforme ilustrado na Figura 12.
- Depurar *scripts JavaScript*: Utilizando *breakpoints* e observando o fluxo de execução para identificar e corrigir falhas lógicas no código do frontend, conforme ilustrado na Figura 13.
- Analisar e ajustar o estilo *CSS*: Inspecionando elementos, modificando propriedades *CSS* em tempo real e avaliando o impacto visual das alterações na apresentação da interface.

Além dos testes mencionados, diversas funcionalidades críticas do sistema foram meticulosamente testadas para assegurar sua operação correta e robustez.

A funcionalidade de cadastro de usuário foi exaustivamente verificada. Isso incluiu a validação de campos obrigatórios para cada tipo de usuário a ser cadastrado, formatos de dados (*e-mail*, senha, Registro Geral (RG), Cadastro de Pessoas Físicas (CPF), Carteira Nacional de Habilitação (CNH), Registro Nacional de Veículos Automotores (RENAVAM)) e o tratamento de usuários já existentes, garantindo que o fluxo completo de criação de contas ocorresse sem falhas e assegurando a integridade e segurança dos dados de autenticação. A Figura 14 apresenta o formulário de cadastro para o perfil de Condutor, exemplificando a interface para a inserção desses dados.

Figura 14 – Formulário de cadastro de usuário (Condutor)

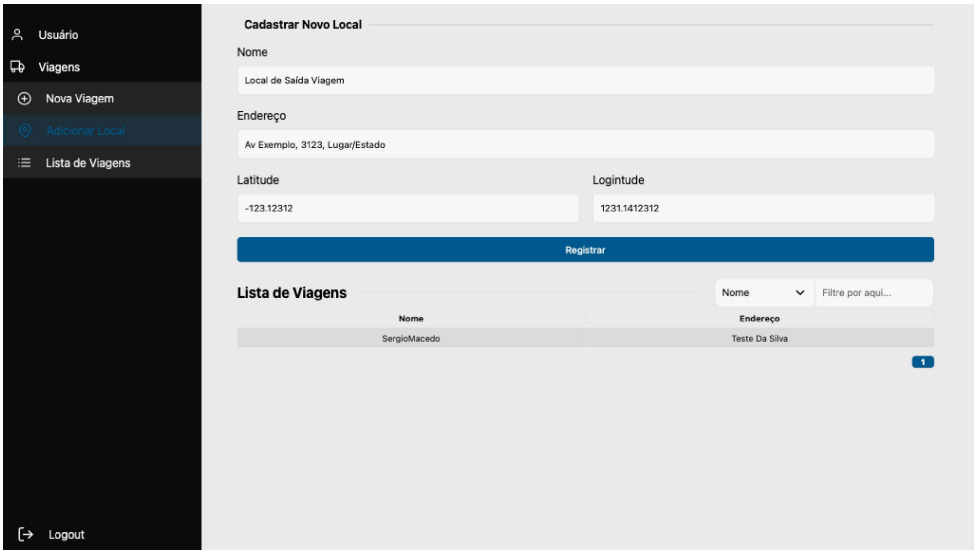


Fonte: (Autor, 2023)

O processo de cadastro de localizações também foi rigorosamente testado. A validação focou na correta inserção, atualização e persistência dos dados de pontos de interesse. Testes específicos verificaram a precisão da entrada de coordenadas

geográficas (latitude e longitude), a correta atribuição de nomes descritivos às localizações e a garantia de que essas informações eram armazenadas de forma confiável no banco de dados e podiam ser recuperadas sem erros. A Figura 15 ilustra a interface de cadastro de endereços, enquanto a Figura 16 apresenta a visualização das localizações cadastradas.

Figura 15 – Formulário de cadastro de endereços



Fonte: (Autor, 2023)

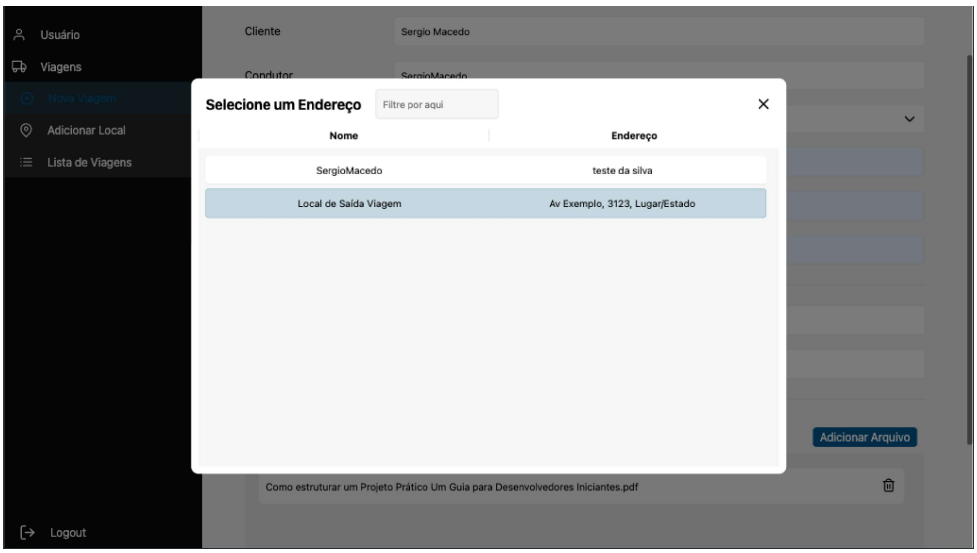
Figura 16 – Listagem dos endereços cadastrados

Fonte: (Autor, 2023)

Similarmente, a funcionalidade de cadastro de viagem foi minuciosamente validada. Isso envolveu a verificação da seleção correta das localizações de origem e destino por meio de interfaces intuitivas, como a apresentada na Figura 17. Foi também testada a correta associação da viagem com o condutor que irá efetuá-la,

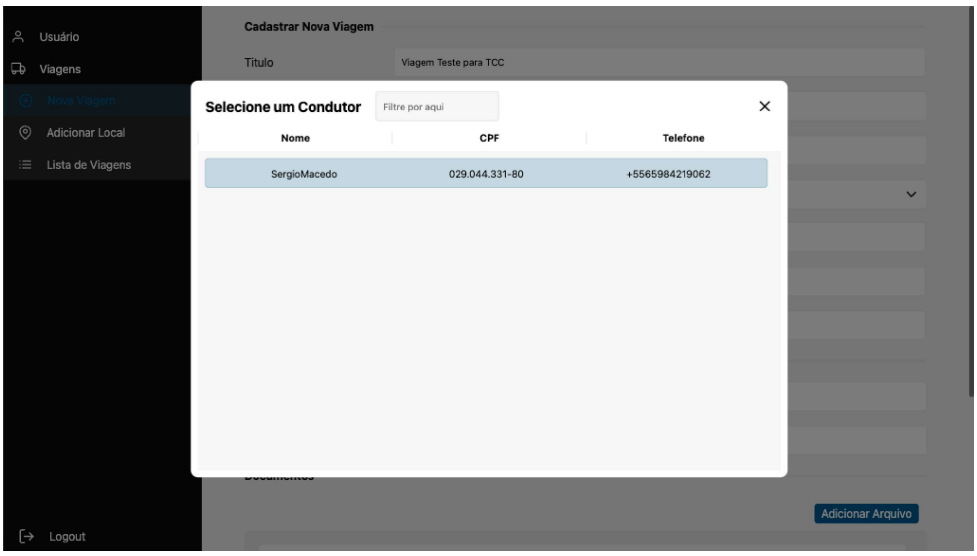
conforme ilustrado na Figura 18. Os testes asseguraram que todas as informações essenciais para a criação de uma viagem fossem capturadas e armazenadas de maneira consistente, preparando o terreno para o rastreamento e a visualização. A Figura 19 exemplifica o formulário de criação de viagem com a seleção de localizações, condutor e demais informações.

Figura 17 – Menu de seleção de endereço de Coleta/Entrega



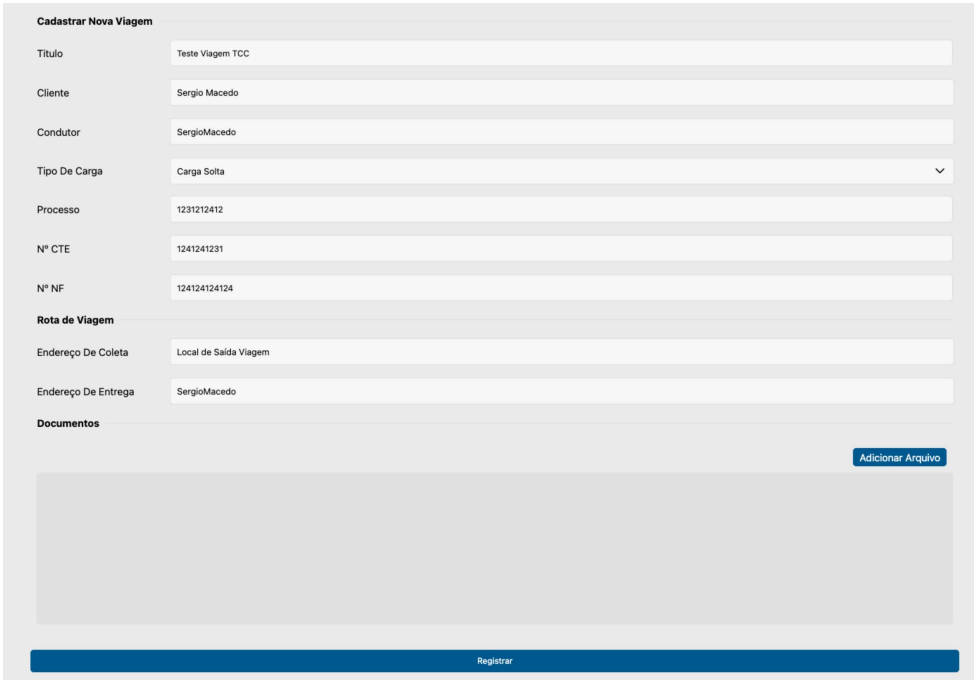
Fonte: (Autor, 2023)

Figura 18 – Menu de seleção de condutor



Fonte: (Autor, 2023)

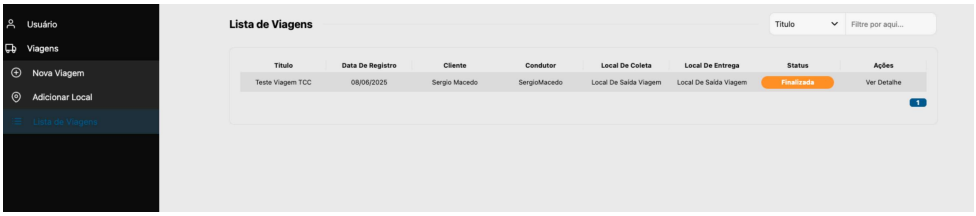
Figura 19 – Formulário de cadastro de viagem preenchido



Fonte: (Autor, 2023)

Em seguida, os detalhes da viagem foram verificados para confirmar que todas as informações pertinentes a uma viagem específica fossem exibidas de forma completa e precisa ao ser selecionada na listagem. Primeiramente, verificou-se a lista de viagens cadastradas, onde são apresentadas as principais informações, conforme ilustrado na Figura 20.

Figura 20 – Listagem das viagens cadastradas



Fonte: (Autor, 2023)

Os testes incluíram a validação da apresentação das localizações de origem e destino, bem como do status atual da viagem, conforme apresentado na Figura 21, e de quaisquer outras informações pertinentes. Essas verificações asseguraram não apenas

a consistência dos dados exibidos com o que foi previamente cadastrado, mas também a correta interpretação dos diferentes estados da viagem pelo sistema, garantindo a precisão das informações acessadas pelos usuários, conforme a Figura 22.

Figura 21 – Barra de status da viagem

Fonte: (Autor, 2023)

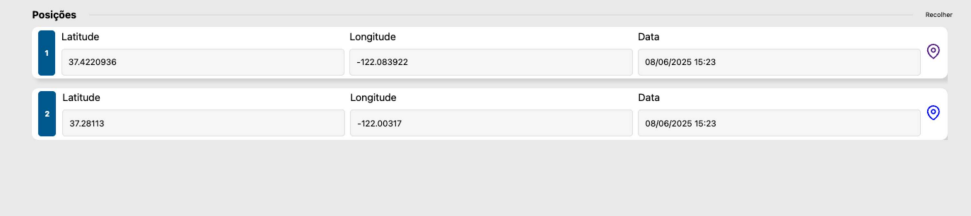
Figura 22 – Detalhes da viagem cadastrada

Fonte: (Autor, 2023)

Por fim, testou-se a visualização das localizações nos detalhes da viagem, um ponto crucial para a funcionalidade de rastreamento. Como a interface web não possui carregamento automático das posições, a validação foi realizada por meio da atualização manual da página , com o objetivo de assegurar que as informações de coordenadas geográficas, enviadas em tempo real pelo smartphone e persistidas no backend, estivessem disponíveis para consulta no frontend da interface administrativa.

Essa etapa confirmou a consistência e a sincronização dos dados entre as diferentes partes do sistema.

Figura 23 – Listagem das Coordenadas Cadastradas



Fonte: (Autor, 2023)

3.7 APLICATIVO MÓVEL

A aplicação móvel foi desenvolvida utilizando o *React Native CLI*, sendo destinada exclusivamente ao uso dos condutores da transportadora. A solução contempla as seguintes funcionalidades principais:

- Tela de autenticação: permite a validação do condutor por meio de credenciais cadastradas no sistema, utilizando requisições HTTP via Axios para comunicação com o backend. O token de autenticação é armazenado localmente após o login bem-sucedido.
- Tela principal: exibe o status da viagem associada ao condutor, indicando se há uma viagem cadastrada ou em andamento, além de disponibilizar as opções para iniciar ou encerrar uma viagem.
- Tela de detalhes da viagem: apresenta informações adicionais relevantes, tais como endereços de coleta e entrega, cliente responsável e outros dados importantes para o condutor.
- Captura de coordenadas: utiliza a biblioteca *react-native-geolocation-service* para a obtenção periódica das coordenadas geográficas (latitude e longitude) do dispositivo, realizadas de forma contínua durante a execução do aplicativo.

3.7.1 Execução em segundo plano

Para garantir a operação do serviço de rastreamento mesmo com o aplicativo em segundo plano, foi empregada a biblioteca *react-native-foreground-service*, que possibilita a execução contínua de tarefas em primeiro plano no sistema operacional móvel, assegurando que a captura das coordenadas não seja interrompida enquanto o usuário utiliza outros aplicativos ou bloqueia a tela do dispositivo.

3.7.1.1 Comunicação com o servidor

A comunicação com o backend ocorre por dois canais distintos, adequados a diferentes tipos de operação:

- Requisições HTTP pontuais, como autenticação e recuperação de dados relacionados à viagem ativa, são realizadas por meio da biblioteca Axios, que oferece uma interface leve e simples para operações assíncronas;
- A transmissão contínua e em tempo real das coordenadas geográficas é realizada via protocolo WebSocket, que estabelece uma conexão persistente e bidirecional entre o aplicativo e o servidor. Essa abordagem garante baixa latência e atualização constante dos dados de localização durante toda a execução da viagem.

3.7.2 Intervalo de captura de posição

Durante o desenvolvimento do aplicativo móvel, a configuração do intervalo de captura da localização foi um ponto de atenção crucial, pois influencia diretamente o desempenho do sistema e o volume de dados armazenados no banco de dados. A abordagem adotada foi buscar um equilíbrio entre a precisão do rastreamento em tempo real e a eficiência no consumo de recursos, como bateria e espaço de armazenamento.

No código-fonte da aplicação móvel, a configuração do rastreamento foi definida com base nas propriedades da biblioteca react-native-geolocation-service utilizada para a captura das coordenadas. O intervalo para a coleta de dados de localização e o *distanceFilter* foram estabelecidos em diálogo direto com a FF Transportes LTDA, que solicitou o uso de sessenta segundos e cinco metros respectivamente, em conformidade com suas necessidades operacionais. Esses valores foram incorporados ao código conforme ilustrado no Apêndice O. A combinação assegurou que uma nova posição só fosse registrada se o dispositivo se movesse pelo menos cinco metros desde a última captura, ou se o intervalo de sessenta segundos fosse atingido.

A escolha do valor de sessenta segundos, além de atender à solicitação da empresa, mostrou-se coerente com testes de usabilidade e integração realizados durante o desenvolvimento, visando mitigar a sobrecarga no banco de dados. Um intervalo menor, como quinze ou trinta segundos, foi inicialmente considerado, mas a alta frequência de envios resultaria em um grande volume de dados. Por outro lado, um intervalo muito longo, como cinco minutos, comprometeria a precisão do rastreamento, pois o caminho percorrido entre os pontos se tornaria menos detalhado.

Essa configuração se mostrou eficaz para o contexto operacional da FF Transportes LTDA, proporcionando um rastreamento preciso o suficiente para a gestão de rotas e o monitoramento da frota, sem gerar um volume de dados excessivo que

pudesse comprometer a performance e a escalabilidade do sistema a longo prazo. Essa decisão validou a importância de ajustar os parâmetros de rastreamento com base nas necessidades específicas do negócio.

3.7.3 Testes aplicados

Durante a fase de testes da aplicação móvel, o Android Studio foi empregado como ferramenta de simulação e depuração. Por meio do emulador do Android Studio, a execução e verificação das funcionalidades do aplicativo foram realizadas em um ambiente controlado, simulando as configurações de dispositivos físicos e visando a compatibilidade com uma ampla gama de aparelhos reais.

Conforme Umar (2020), a adoção de distintas categorias de testes é fundamental para assegurar a qualidade e a robustez do software, pois a combinação adequada de abordagens permite cobrir diferentes aspectos do sistema. Nesse sentido, os testes conduzidos abrangeram diversas categorias:

- Testes Funcionais Gerais: Validação do comportamento correto das funcionalidades, como login, navegação e visualização de dados.
- Testes de UI: Avaliação da usabilidade e apresentação visual da interface, assegurando uma experiência intuitiva e agradável para o usuário.
- Testes de Integração com o backend: Verificação da comunicação e troca de dados entre o aplicativo e o servidor, garantindo a integridade e consistência das informações.

As principais configurações do dispositivo virtual *Android Virtual Device* (AVD) utilizado neste ambiente de simulação estão detalhadas no Quadro 3.

Quadro 3 – Configurações principais do AVD utilizado para os testes da aplicação móvel

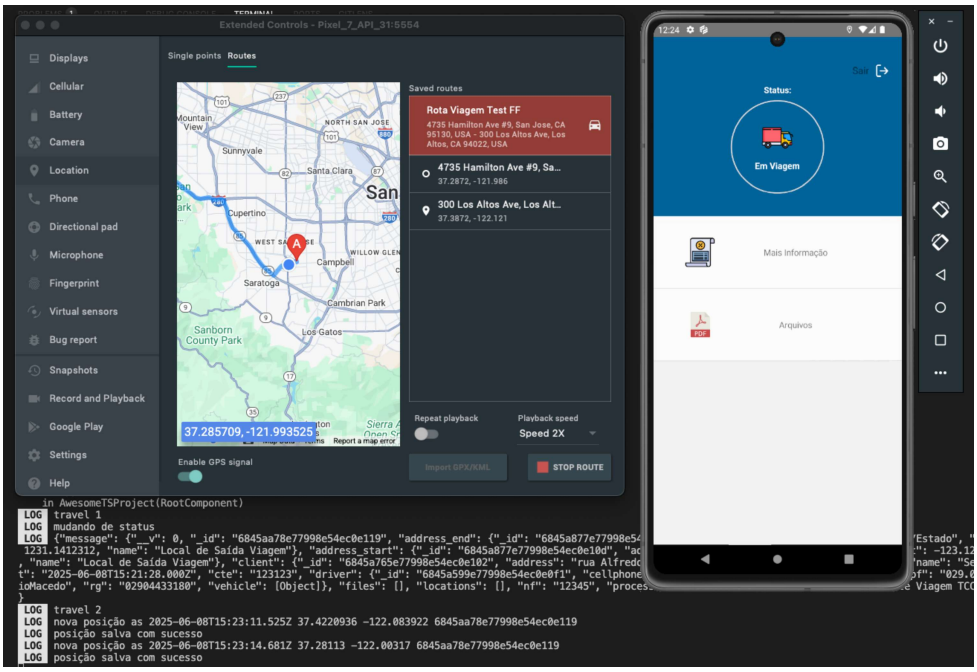
Propriedade	Valor
Nome do dispositivo	*Pixel 7 API 31*
Identificador do *AVD*	*Pixel_7_API_31*
Número de núcleos da *Central Processing Unit* (CPU)	4
Tamanho da memória *Random Access Memory* (RAM)	2048 megabytes (MB)

Fonte: (Autor, 2024)

A ferramenta de controle de GPS integrada ao Android Studio foi essencial para simular situações de rastreamento, uma funcionalidade central do sistema. Rotas com coordenadas dinâmicas foram configuradas, permitindo que o aplicativo recebesse atualizações contínuas de localização e replicasse deslocamentos reais. Este procedimento foi fundamental para validar o funcionamento preciso e em tempo real do sistema

de rastreamento, conforme ilustrado na Figura 24, em que se observa a atualização da localização na ferramenta e a resposta do sistema confirmando o recebimento das coordenadas simuladas.

Figura 24 – Simulação de Deslocamento GPS e Validação de Coordenadas no Android Studio



Fonte: (Autor, 2023)

3.7.4 Validações em operações reais

Além dos testes realizados em ambiente controlado com o emulador do Android Studio, foram conduzidas validações em operações logísticas reais pela empresa FF Transportes LTDA, complementando a verificação do sistema em cenários de uso efetivo.

Caso 1: Backup de rastreamento. Durante uma viagem, o equipamento de rastreamento original perdeu sinal. O sistema desenvolvido, utilizando o smartphone do motorista, manteve o monitoramento ativo e permitiu a localização do veículo, atuando como redundância de segurança.

Caso 2: Teste de longa distância. Foi realizada viagem de teste com carga real até São Paulo. O sistema rastreou todo o percurso com sucesso, validando a estabilidade da comunicação em trajetos de longa distância e confirmando o rastreamento contínuo em viagem interestadual.

Esses testes foram conduzidos de forma independente pela própria equipe da

FF Transportes LTDA, em suas operações logísticas reais, sem a participação do autor deste trabalho.

4 ANÁLISE E DISCUSSÃO DOS RESULTADOS

4.1 COMUNICAÇÃO EM TEMPO REAL

O backend desenvolvido com Node.js, Express e WebSocket (implementado no servidor com Socket.io) demonstrou-se eficiente para manter a comunicação contínua entre o aplicativo móvel e o servidor. Os testes funcionais indicaram que as mensagens foram transmitidas em tempo real, com resposta rápida e estabilidade, mesmo com o aplicativo em segundo plano. Esses resultados confirmam a adequação da escolha do WebSocket para atender aos requisitos de comunicação em tempo real no contexto da FF Transportes LTDA.

4.2 PERSISTÊNCIA DE DADOS

O banco de dados MongoDB mostrou-se apropriado para o armazenamento e a recuperação das informações do sistema. Durante os testes realizados, as operações de escrita e leitura ocorreram sem inconsistências, e os dados armazenados refletiram corretamente as informações referentes às viagens, aos motoristas e às posições geográficas. A estruturação dos dados contribuiu para uma manipulação e consulta eficiente ao longo da execução do sistema.

4.3 INTERFACE FRONTEND

A interface implementada com React e Next.js apresentou bom desempenho em termos de responsividade e usabilidade. Os formulários de cadastro e de atualização de informações funcionaram de maneira adequada, com validação eficiente dos dados de entrada. A comunicação com o backend, realizada por meio da biblioteca Axios, ocorreu sem falhas, garantindo que os dados exibidos estivessem atualizados e consistentes.

4.4 DESENVOLVIMENTO DA APLICAÇÃO MÓVEL

A prototipação da aplicação móvel foi inicialmente realizada com a ferramenta *Expo*. Contudo, devido às limitações encontradas na execução de tarefas em segundo plano, especialmente no que diz respeito à captura contínua da localização, optou-se pela migração para o React Native CLI. Essa mudança permitiu maior controle sobre os serviços nativos, aspecto essencial para garantir o envio estável da localização mesmo com o aplicativo minimizado.

A biblioteca *react-native-geolocation-service* atendeu às necessidades de obtenção das coordenadas geográficas, enquanto o react-native-foreground-service possibilitou a continuidade da execução em segundo plano. A comunicação por meio do WebSocket manteve-se estável durante as simulações de trajeto.

4.5 CONFIGURAÇÃO DO INTERVALO DE CAPTURA DE POSIÇÃO

Durante os testes da aplicação móvel, verificou-se que o intervalo de captura da localização, determinado pela diferença de tempo ou de distância entre as capturas, influencia diretamente no desempenho do sistema e no volume de dados armazenados. Foram realizados testes com diferentes configurações a fim de equilibrar a frequência de atualização e o uso do espaço no banco de dados.

A configuração do intervalo mostrou-se fundamental para evitar a sobrecarga do banco, que poderia ocorrer caso os dados fossem capturados e armazenados com frequência excessiva. Recomenda-se que esse parâmetro seja ajustável pelo usuário, permitindo adequações conforme a necessidade operacional, de modo a equilibrar a precisão do rastreamento com o consumo de recursos.

5 CONCLUSÃO E CONSIDERAÇÕES FINAIS

5.1 CONCLUSÃO

Este trabalho apresentou o desenvolvimento de um sistema de rastreamento em tempo real utilizando smartphones, aplicado ao contexto da empresa FF Transportes LTDA. A implementação integrada do backend, frontend e aplicação móvel permitiu atender aos requisitos definidos, garantindo comunicação eficiente, persistência dos dados e usabilidade necessária para o acompanhamento das viagens realizadas pelos condutores.

Durante o desenvolvimento, foi possível observar a importância da adaptação tecnológica frente às limitações das ferramentas utilizadas. Inicialmente, utilizou-se a plataforma Expo devido à sua abstração e facilidade de implementação. Contudo, suas limitações no acesso a serviços nativos, especialmente para execução de tarefas em segundo plano, exigiram a migração para o React Native CLI. Apesar da maior complexidade, essa solução se mostrou indispensável para garantir o funcionamento estável da aplicação, especialmente na captura contínua da localização.

Outro aprendizado fundamental foi a necessidade de ajustes finos na configuração do intervalo de captura da posição, a fim de evitar sobrecarga do banco de dados e garantir a performance do sistema. Esse processo evidenciou a importância de considerar detalhes técnicos que impactam diretamente na estabilidade e escalabilidade da solução.

5.2 TRABALHOS FUTUROS

Para aprimorar e expandir a solução, sugerem-se as seguintes implementações futuras:

- Ampliação das funcionalidades do dashboard, incluindo indicadores avançados para análise detalhada do desempenho operacional.
- Implementação de alertas inteligentes baseados em geofencing dinâmico, permitindo notificações automáticas em situações específicas.
- Otimização de rotas utilizando algoritmos preditivos para melhorar a eficiência dos trajetos.
- Aprimoramentos na interface do usuário, visando maior intuitividade e usabilidade.
- Automação da configuração do intervalo de captura de localização, possibilitando ajustes dinâmicos conforme o contexto operacional.

REFERÊNCIAS

ALMEIDA, D. A. F. ***Implementação em tempo real de um sistema de avaliação automática de leitura de crianças***. Dissertação (Dissertação de Mestrado), 2018.

ANDREOLI, R.; CUCINOTTA, T.; PEDRESCHI, D. **Rt-mongodb: A nosql database with differentiated performance**. In: HELFERT, M.; FERGUSON, D.; PAHL, C. (Ed.). *Proceedings of the 11th International Conference on Cloud Computing and Services Science (CLOSER)*. [S.l.]: SciTePress, 2021. p. 77–86.

ANDREWS, S. ***Professional Android 4 Application Development***. [S.l.]: Wrox, 2012.

BECK, K. et al. ***Manifesto para Desenvolvimento Ágil de Software***. 2001. Acesso em: 11 fev. 2025. Disponível em: <https://agilemanifesto.org/iso/ptbr/manifesto.html>.

BERNARDI, J. V. E.; LANDIM, P. M. B. ***Aplicção do Sistema de Posicionamento Global (GPS) na Coleta de Dados***. Rio Claro, SP: Laboratório de Geomatemática, UNESP, 2002. Texto Didático 10.

CHODOROW, K. ***MongoDB: The Definitive Guide***. 2. ed. Sebastopol, CA: O'Reilly Media, 2013.

CHRISTOPHER, M. ***Logística e gerenciamento da cadeia de suprimentos: estratégias para a redução de custos e melhoria dos serviços***. 4. ed. São Paulo: Cengage Learning, 2016.

CâMARA, G. et al. ***Geoprocessamento: teoria e aplicações***. São José dos Campos: INPE, 2004.

EISENMAN, B. ***Learning React Native***. [S.l.]: O'Reilly Media, 2018. ISBN 978-1-491-98914-2.

FERREIRA, A. G. ***Princípios de Aviônica e Navegação***. [S.l.]: Editora da Universidade Federal de São Carlos, 2012.

FETTE, I.; MELNIKOV, A. ***The WebSocket Protocol***. 2011. RFC 6455, IETF. Acesso em: 5 fev. 2025. Disponível em: <https://datatracker.ietf.org/doc/html/rfc6455>.

FLANAGAN, D. ***JavaScript: The Definitive Guide***. 6. ed. Sebastopol, CA: O'Reilly Media, 2011.

GACKENHEIMER, C. ***Introduction to React***. 1. ed. New York: Apress Media, 2015.

GAMMA, E. et al. ***Design Patterns: Elements of Reusable Object-Oriented Software***. Reading, MA: Addison-Wesley, 1994.

GIANNOPOULOS, G. A. **The application of information and communication technologies in transport**. *European Journal of Operational Research*, v. 152, n. 2, p. 302–320, 2004.

HAHN, E. M. ***Express in Action***. [S.l.]: Manning, 2016.

LEI, K.; MA, Y.; TAN, Z. **Performance comparison and evaluation of web development technologies in php, python, and node.js**. In: *IEEE 17th International Conference on Computational Science and Engineering*. Chengdu, China: [s.n.], 2014. p. 661–668.

LIMA, G. ***Saiba tudo sobre o IDE - Integrated Development Environment***. 2022. Acesso em: 07 de outubro de 2024. Disponível em: <https://www.alura.com.br/artigos/o-que-e-uma-ide>.

LIU, Z. et al. **Research on logistics enterprises' selection of intelligent logistics equipment**. In: IEEE. *2019 International Conference on Industrial Engineering and Engineering Management (IEEM)*. Macau, China, 2019. p. 953–957.

LOPES, A. T. et al. **Followbus: Aplicativo mobile para rastreamento de transportes públicos**. *Revista Científica da UNIFENAS*, v. 5, n. 3, p. 23, set 2024.

MARDAN, A. ***Express.js Guide: The Comprehensive Book on Express.js***. 1. ed. [S.l.: s.n.], 2014.

MELKOTE, H. V.; D'SOUZA, L. J. **Real-time data synchronization using websockets for a collaborative framework**. In: *Proceedings of the International Conference on Computer Communication and Management (ICCCM)*. [S.l.: s.n.], 2012. p. 35–39.

MOSS, A. ***WebSockets: For the Modern Web***. Berkeley, CA: Apress, 2016.

NOVAES, A. G. N. ***Logística e Gerenciamento da Cadeia de Distribuição***. 3. ed. Rio de Janeiro: Campus, 2007.

PEREIRA, C. R. ***Aplicações web real-time com Node.js***. [S.l.]: Editora Casa do Código, 2014.

PINTO, L. A.; HOFFMANN, S.; URIARTE, L. R. **Análise comparativa entre as tecnologias de front-end react, angular e vue**. *Revista de Extensão e Iniciação Científica da UNISOCIESC*, v. 10, n. 1, 2023.

PONTES, G. ***Progressive Web Apps: Construa Aplicações Progressivas com React***. [S.l.]: Casa do Código, 2018. ISBN 978-85-94188-56-4.

PRESSMAN, R. S.; MAXIM, B. R. ***Software Engineering: A Practitioner's Approach***. 8. ed. New York: McGraw-Hill Education, 2014.

SCHWABER, K. ***Agile Project Management with Scrum***. Redmond, WA: Microsoft Press, 2004.

SOMMERVILLE, I. ***Software Engineering***. 10. ed. Boston: Pearson, 2015.

TAO, Z.; XING, X. **A research on vehicle positioning and tracking system based on gps and gprs**. In: IEEE. *2011 International Conference on Automation and Logistics (ICAL)*. Chongqing, China, 2011. p. 433–437.

TEIXEIRA, P. ***Professional Node.js - Building Javascript Based Scalable Software***. [S.l.: s.n.], 2013.

THAKKAR, M. ***Building React Apps with Server-Side Rendering*: Use react, redux, and next to build full server-side rendering applications**. Berkeley, CA: Apress, 2020. ISBN 978-1-4842-5868-2.

TILKOV, S.; VINOSKI, S. **Node.js: Using javascript to build high-performance network programs**. *IEEE Internet Computing*, v. 14, n. 6, p. 80–83, 2010.

TOSTES, R. N.; COSTA, R. M. D. R. **Aplicações multiplataforma com react-desenvolvimento de uma aplicação educacional**. *ANALECTA-Centro Universitário Academia*, v. 5, n. 5, 2020.

UMAR, M. A. ***A Study of Software Testing: Categories, Levels, Techniques, and Types***. TechRxiv, 2020. Pré-publicação. Acesso em: fev. 2025. Disponível em: <https://www.researchgate.net/publication/342579919_A_Study_of_Software_Testing_Categories_Levels_Techniques_and_Types>.

WANG, J.; LI, K.; ZHANG, J. **Real-time tracking system for logistics based on gps and gprs**. *Proceedings of the 2009 International Conference on Wireless Communications, Networking and Mobile Computing*, p. 1–4, 2009.

WIERUCH, R. ***The Road to React***. 2. ed. Berlin: self-published, 2019.

WINTERFELDT, J. ***Context-Aware Mobile Crowd Sensing using Mobile Hybrid Application Frameworks***. 2017. Germany: Universität Ulm. Disponível em: <http://dbis.eprints.uni-ulm.de/1486/1/MA_WIN_2017.pdf>.

ZHANG, R. et al. **A real-time logistics tracking system based on gps and gprs for cold chain transportation**. *Journal of Advanced Transportation*, v. 49, n. 1, p. 1–16, 2015.

APÊNDICE A – Ferramenta de autocompletar do VS Code

Figura 25 – Ferramenta de autocompletar do VS Code

Fonte: (Autor, 2024)

APÊNDICE B – Exemplo de implementação de rota com middleware de autenticação no Express

Figura 26 – Exemplo de implementação de rota com middleware de autenticação no Express

```
const upload = require("../config/multer");

const fileController = require("../controllers/fileController");
const adminController = require("../controllers/adminContoller");
const travelController = require("../controllers/travelsController");
const authController = require("../controllers/authController");
const { authMiddleware } = require("../middleware/authMiddleware");
//AUTH
router.post("/login", authController.Adminlogin);

router.use(authMiddleware);
//FILE CONTROLLERS
router.post("/file", upload.single("file"), fileController.create);
router.delete("/file/:id", fileController.remove);
router.get("/file/:id", fileController.getFile);
router.get("/file", fileController.findAll);
```

Fonte: (Autor, 2024)

APÊNDICE C – Implementação do evento locationUpdate do Socket.io

Figura 27 – Implementação do evento locationUpdate do Socket.io

```
io.on("connection", (socket) => {
  setDriverConnectionState(true);

  socket.on("locationUpdate", async (data) => {
    try {
      const { latitude, longitude, id, date } = data;

      const response = await updatePosition(latitude, longitude, id, date);

      if (response) {
        socket.emit("received", { status: 'ok', id, timestamp: new Date().toISOString() });
      } else {
        socket.emit("received", { status: 'error', message: 'Dados inválidos ou viagem não encontrada' });
      }

      return;
    } catch (error) {
      socket.emit("error", {
        message: "Erro ao processar atualização de localização",
        error: error.message
      });
    }
  });

  socket.on("disconnect", (reason) => {
    setDriverConnectionState(false);
  });

  socket.on("error", (error) => {
    // Erro no socket
  });
});
```

Fonte: (Autor, 2024)

APÊNDICE D – Implementação da função updatePosition de atualização de localização

Figura 28 – Implementação da função updatePosition de atualização de localização

```
const updatePosition = async (lat, long, id, date) => {
  try {
    if (!id || !lat || !long || !date) {
      return false;
    }

    const travel = await Travel.findById(id);

    if (!travel) {
      return false;
    }

    travel.locations.push({ latitude: lat, longitude: long, created_at: date });
    await travel.save();

    return true;
  } catch (error) {
    return false;
  }
};
```

Fonte: (Autor, 2024)

APÊNDICE E – Exemplo do Uso do Bcrypt no Controlador de Registro de Usuário Administrador

Figura 29 – Exemplo do Uso do Bcrypt no Controlador de Registro de Usuário Administrador

```
const adminController = {
  adminRegister: async (req, res) => {
    try {
      const { cpf, password, ...data } = req.body;
      console.log(req.body);

      // Verificação se o CPF já está em uso
      const existingAdmin = await Admin.findOne({ cpf });
      if (existingAdmin) {
        return res.status(400).json({ error: "Este CPF já está em uso" });
      }

      // Verificação se a senha está presente nos dados
      if (!password) {
        return res.status(400).json({ error: "Senha não fornecida" });
      }

      // Criptografar a senha
      const hashedPassword = await bcrypt.hash(password, 10);

      // Criação do novo administrador
      const newAdmin = new Admin({
        ...data,
        cpf,
        password: hashedPassword,
      });

      await newAdmin.save();
      res.status(200).json({ message: "Usuário cadastrado com sucesso" });
    } catch (error) {
      console.log(error);
      res.status(500).json({ error: "Erro ao cadastrar usuário" });
    }
  },
```

Fonte: (Autor, 2024)

APÊNDICE F – Exemplo da Geração de JWT na Rota de Login

Figura 30 – Exemplo da Geração de JWT na Rota de Login

```
  const auth_user = await user.findOne(body);
  console.log(auth_user);
  if (!auth_user) {
    throw new Error("Credenciais inválidas");
  }
  const isPasswordMatch = await bcrypt.compare(
    password,
    auth_user.password
  );
  if (!isPasswordMatch) {
    console.log("password");
    throw new Error("Credenciais inválidas");
  }

  // Gera o token JWT
  const token = jwt.sign(
    { userId: user._id, userType },
    process.env.SECRET_KEY_JWT,
    {
      expiresIn: "24h",
    }
  );
  const { id, name } = user;
  res.json({ id, name, accessToken: token });
} catch (error) {
  res.status(500).json({ message: error.message });
}
```

Fonte: (Autor, 2024)

APÊNDICE G – Exemplo da Validação de JWT aplicado ao middleware de autenticação

Figura 31 – Exemplo da Validação de JWT aplicado ao middleware de autenticação

```
const authMiddleware = async (req, res, next) => {
  try {
    const authHeader = req.headers.authorization;
    if (!authHeader) return res.status(401).send({ message: "Unauthorized" });

    const parts = authHeader.split(" ");
    if (parts.length !== 2)
      return res.status(401).send({ message: "Invalid token!" });

    const [scheme, token] = parts;
    if (!/^Bearer$/i.test(scheme))
      return res.status(401).send({ message: "Malformatted Token!" });

    const decoded = await authenticate(token);
    req.userId = decoded.id;
    return next();
  } catch (error) {
    console.log(error);
    res.status(401).send({ message: "Unauthorized" });
  }
};
```

Fonte: (Autor, 2024)

APÊNDICE H – Configuração da Conexão com o Banco de Dados Utilizando Mongoose

Figura 32 – Configuração da Conexão com o Banco de Dados Utilizando Mongoose

```
const mongoose = require('mongoose')
const { initializeAdminUser } = require('../scripts/initializeAdmin')
const connectDB = async () => {
    try {

        const DB_ADMIN_USER = encodeURIComponent(process.env.MONGO_INITDB_ROOT_USERNAME)
        const DB_ADMIN_PASSWORD = encodeURIComponent(process.env.MONGO_INITDB_ROOT_PASSWORD)
        const DB_HOST = process.env.MONGO_INITDB_HOST
        const DB_APP_NAME = process.env.MONGO_INITDB_APP_NAME
        const MONGODB_URI = `mongodb+srv://${DB_ADMIN_USER}:${DB_ADMIN_PASSWORD}@${DB_HOST}/fftransportes?appName=${DB_APP_NAME}`;

        console.log("tentando conexão com", MONGODB_URI)
        await mongoose.connect(MONGODB_URI, {
            useNewUrlParser: true,
            useUnifiedTopology: true
        })

        initializeAdminUser()
        console.log('Conexão com o banco de dados MongoDB estabelecida com sucesso!')
    } catch (error) {
        console.error('Erro de conexão com o banco de dados MongoDB:', error)
        process.exit(1)
    }
}
```

Fonte: (Autor, 2024)

APÊNDICE I – Definição do Esquema da Entidade Viagem com Mongoose

Figura 33 – Definição do Esquema da Entidade Viagem com Mongoose

```
const mongoose = require('mongoose')

const locationSchema = new mongoose.Schema({
    latitude: { type: Number },
    longitude: { type: Number },
    created_at: { type: Date, default: Date.now },
})

const travelSchema = new mongoose.Schema({
    title: { type: String, required: true },
    address_start: { type: mongoose.Schema.Types.ObjectId, ref: 'Address', required: true },
    address_end: { type: mongoose.Schema.Types.ObjectId, ref: 'Address', required: true },
    client: { type: mongoose.Schema.Types.ObjectId, ref: 'Client', required: true },
    driver: { type: mongoose.Schema.Types.ObjectId, ref: 'Conductor', required: true },
    cte: { type: String, required: true },
    nf: { type: String, required: true },
    process: { type: String, required: true },
    type: { type: String, required: true },
    files: [{ type: mongoose.Schema.Types.ObjectId, ref: 'File' }],
    locations: [locationSchema],
    created_at: { type: Date, default: Date.now },
    status: { type: String, required: true },
})

const Travel = mongoose.model('Travel', travelSchema)

module.exports = Travel
```

Fonte: (Autor, 2024)

APÊNDICE J – Página de login composta por componentes reutilizáveis

Figura 34 – Página de login composta por componentes reutilizáveis

```
<main className={styles.mainContainer}>
  <section className={styles.loginContainer}>
    <span className={styles.titleLoginForm}>Acesso Central</span>
    <div className={styles.formContainer}>
      <InputCustom
        type="text"
        title="login"
        name="login"
        value={values.login}
        onBlur={handleBlur}
        onChange={handleChange}
        error={newerror.login()}
        placeholder="Digite seu login"
      />
      <InputCustom
        type="password"
        title="senha"
        name="password"
        value={values.password}
        onBlur={handleBlur}
        onChange={handleChange}
        error={newerror.password()}
        placeholder="Digite sua senha"
      />

      <ButtonsCustom
        title='Acessar'
        onClick={() => handleSubmit()}
        type="submit" />
    </div>
  </section>
</main >
```

Fonte: (Autor, 2023)

APÊNDICE K – Visualização do componente ButtonCustom

Figura 35 – Visualização do componente ButtonCustom

```
import styles from '@/styles/Components/Button/button.module.css'
import CircleLoader from '../Loader/Loaders'

const ButtonsCustom = ({ onClick, type, loading = false, className = {}, icon =
() => { }, title }) => {
    console.log(!loading)

    return (
        <button
            disabled={loading}
            type={type}
            className={
                typeof className == 'object'
                    ? styles.buttonContainer
                    : className
            }
            onClick={onClick}
        >
            {loading ? <CircleLoader /> : <>{title}{icon()}</>}
            {loading && "carregando"}
        </button>
    )
}

export default ButtonsCustom
```

Fonte: (Autor, 2023)

APÊNDICE L – Visualização do componente InputCustom

Figura 36 – Visualização do componente InputCustom

```
const InputCustom = ({
    title,
    type,
    placeholder,
    value,
    ref,
    name = '',
    onChange = () => {},
    error,
    row = false,
    onBlur,
    className,
    disable = false,
    maxLength,
}) => {
    return (
        <div
            className={styles.inputCustomContainer}
            style={{
                flexDirection: `${row ? 'row' : 'column'}`,
            }}
        >
            {title && (
                <span
                    style={{
                        minWidth: `${row && '200px'}`,
                    }}
                    className={styles.titleInput}
                >       Sergio Silva Macedo, 19 months ago • Update SIte …
                    {title}
                </span>
            )}
            <input
                placeholder={placeholder}
                name={name}
                value={value}
                type={type}
                onBlur={onBlur}
                onChange={(e) => onChange(e)}
                disabled={disable}
                ref={ref}
                maxLength={maxLength}
                className={
                    className
                        ? className
                        : disable
                        ? styles.disabledInput
                        : error
                        ? styles.inputErrorLine
                        : styles.inputContainer
                }
            />
            {error && <span className={styles.errorFormMessage}>*{error}</span>}
        </div>
    )
}
```

Fonte: (Autor, 2023)

APÊNDICE M – Componente Principal de Comunicação com o Backend (Axios)

Figura 37 – Componente Principal de Comunicação com o Backend (Axios)

```
const api = async ({
  method,
  endpoint,
  data,
  headers,
  onUploadProgress,
  responseType,
}) => {

  if (!headers) {
    const token = getCookie(HooksCookies.token)
    server.defaults.headers.common['Authorization'] = `Bearer ${token}`;
  }

  try {
    const response = await server.request({
      method,
      url: endpoint,
      data,
      headers,
      onUploadProgress,
      responseType,
    });
    const parsed = await parseResponse(response); // Passando diretamente a resposta
    return parsed;
  } catch (error) {
    if (error.response) {
      const parsedError = await parseResponse(error.response);
      throw parsedError;
    }
    throw {
      response: { message: "Erro na requisição", details: error.message },
    };
  }
};
```

Fonte: (Autor, 2024)

APÊNDICE N – Exemplo de Implementação de um Serviço Específico Utilizando o Componente Central

Figura 38 – Exemplo de Implementação de um Serviço Específico Utilizando o Componente Central

```
const startRegisterTravel = async (data) => {
  try {
    setLoading(true)
    await api({
      data,
      endpoint: "/api/admin/register/travel",
      method: "POST",
    });
    setLoading(false)
    handleOpenAlert({
      title: "Sucesso",
      message: "Viagem Cadastrada com sucesso",
      accept: () => {
        router.push("list");
      },
    });
  } catch (error) {
    setLoading(false)
    handleOpenAlert({
      title: "Alerta",
      message: "Viagem não cadastrada",
    });
  }
};
```

Fonte: (Autor, 2024)

APÊNDICE O – Trecho de código da função de captura de posição

Figura 39 – Trecho de código da função de captura de posição

```
{
    accuracy: { android: 'high', ios: 'best' },
    enableHighAccuracy: true,
    distanceFilter: 5,
    interval: 60000,
    fastestInterval: 300,
    forceLocationManager: false,
    showLocationDialog: true,
    useSignificantChanges: false,
    showsBackgroundLocationIndicator: true,
}
```

Fonte: (Autor, 2023)