# Artigo RotaExpress - P.I. V

**Leonardo Pertile Follador, Lucas Luís Stasiak, Luiz Felipe Buaszczyk**

Departamento de ciência da computação
Universidade Regional Integrada do Alto Uruguai e das Missões (URI)
Câmpus 2 - Erechim, RS - Brazil

{066477,101188,104104}@aluno.uricer.edu.br

***Abstract**. This chapter will present the main concepts for understanding project development, providing all the theoretical knowledge necessary to understand this project.*

***Resumo**. Este capítulo apresentará os principais conceitos para a compreensão do desenvolvimento do projeto, fornecendo todo o conhecimento teórico necessário para entender o desenvolvimento deste projeto.*

## 1. INTRODUÇÃO

Com o avanço da tecnologia e a crescente demanda por soluções logísticas eficientes, o rastreamento de produtos tornou-se uma ferramenta essencial para empresas que desejam oferecer maior transparência e controle aos seus clientes. Nesse contexto, o projeto RotaExpress foi desenvolvido como uma aplicação web que visa facilitar o gerenciamento de entregas e o acompanhamento de produtos em tempo real. O sistema permite que empresas se cadastrem na plataforma e tenham acesso a um painel onde podem registrar seus produtos, atualizar o status das entregas e disponibilizar essas informações aos seus clientes de forma prática e intuitiva. Este artigo apresenta o desenvolvimento do RotaExpress, abordando desde a concepção da ideia até as etapas de implementação, destacando as tecnologias utilizadas, os desafios enfrentados e os benefícios que a plataforma oferece ao setor logístico.

## 2. REFERENCIAL TEÓRICO

Neste capítulo é apresentado todo o conteúdo requisitado para compreender o fluxo de desenvolvimento do projeto.

### 2.1. HTML (HyperText Markup Language)

Para criação de sites é necessário utilizar uma linguagem específica para realizar essa tarefa. Essa linguagem é denominada **HTML** (HyperText Markup Language). Contudo, ela não é uma linguagem de programação, ou seja, em vez de comandos para execução de cálculos ou processamento de dados, ela possui

marcadores, que são utilizados para formatar um texto, uma imagem ou qualquer outro objeto que faça parte da estrutura de um documento HTML (ALVES, 2021).

Esses marcadores são envolvidos pelos sinais '**<**' e '**/>**' para indicar o início e o fim da formatação, respectivamente, embora existam algumas exceções, como é o caso da tag **<br>** ALVES (2021). Confira um exemplo de marcação em html na **Figura 1**.

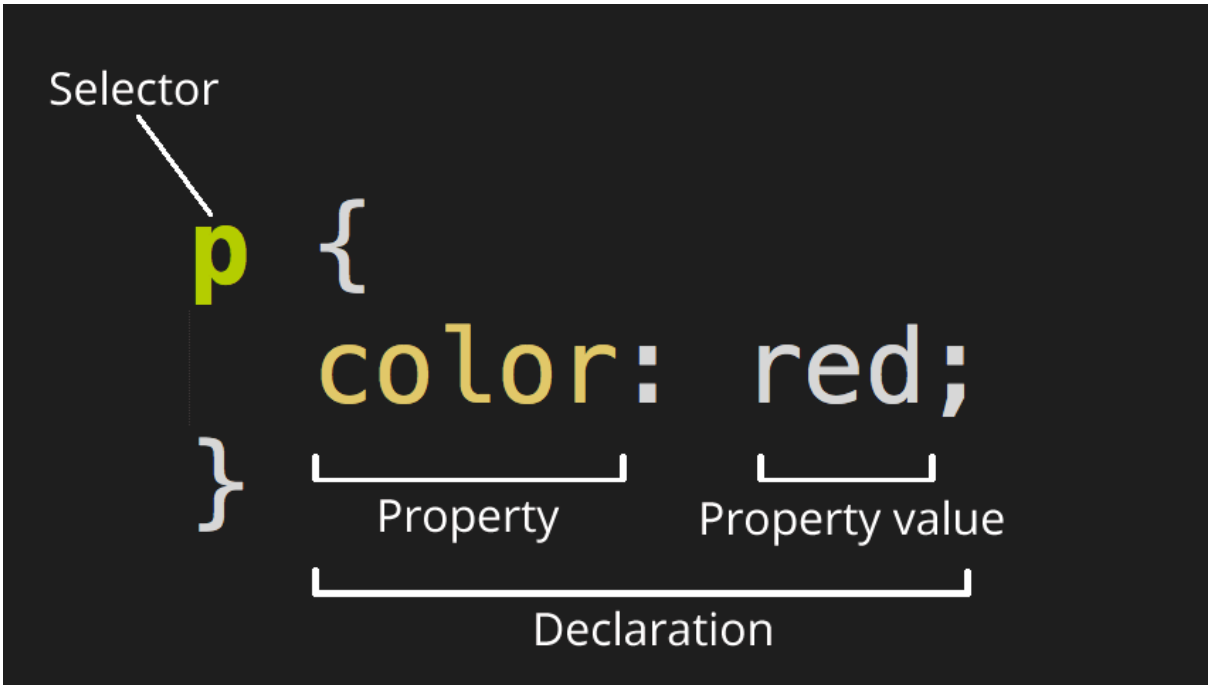
**Figura 1** - Exemplo de utilização da tag de marcação.


**Fonte**: ALVES (2021).

### 2.2. CSS (Cascading Style Sheets)

Assim como o HTML, o CSS não é realmente uma linguagem de programação. Também não é uma linguagem de marcação — é uma linguagem de folhas de estilos. Isso significa que o CSS permite aplicar estilos seletivamente a elementos em documentos HTML (MDN).

**Figura 2** - Exemplo de estilização com CSS.




**Fonte**: MDN

Seletor (**Selector**)

O nome do elemento HTML no começo do conjunto de regras. Ele seleciona o(s) elemento(s) a serem estilizados (nesse caso, elementos <p>). Para dar estilo a um outro elemento, é só mudar o seletor (MDN).

Declaração (**Declaration**)

Uma regra simples como color: red; especificando quais das propriedades do elemento você quer estilizar (MDN).

Propriedades (**Property**)

Forma pela qual você estiliza um elemento HTML. (Nesse caso, color é uma propriedade dos elementos <p>.) Em CSS, você escolhe quais propriedades você deseja afetar com sua regra (MDN).

Valor da propriedade (**Property value**)

À direita da propriedade, depois dos dois pontos, nós temos o valor de propriedade, que escolhe uma dentre muitas aparências possíveis para uma determinada propriedade (MDN).

Cada linha de comando deve ser envolvida em chaves ({}), dentro de cada declaração, você deve usar dois pontos (:) para separar a propriedade de seus valores e dentro de cada conjunto de regras, você deve usar um ponto e vírgula (;) para separar cada declaração da próxima (MDN).

### 2.3. JS (JavaScript)

JavaScript é uma linguagem de programação usada para fazer páginas interativas na Internet. As funções de JavaScript melhoram a experiência do usuário durante a navegação em um site, como, por exemplo, desde a atualização do feed na página da mídia social até a exibição de animações e mapas interativos (AMAZON).

Antes do javascript, as páginas da Web eram estáticas como páginas em um livro. Uma página estática exibia informações em um layout fixo e não fazia tudo que esperamos de um site moderno hoje em dia. O JavaScript surgiu como uma tecnologia para tornar as aplicações Web mais dinâmicas. Ao usar JavaScript, os navegadores passaram a ser capazes de responder a interações do usuário e alterar o layout do conteúdo na página (AMAZON).

### 2.4. PostgreSQL

É um banco de dados com código aberto no qual possui uma forte reputação por sua confiabilidade, flexibilidade e suporte a padrões técnicos abertos, é um dos bancos de dados relacionais mais compatíveis, estáveis e maduros disponíveis atualmente (IBM).

### 2.5. Git

Desenvolvido por Linus Torvalds, o Git é um sistema para o controle de versão distribuído que permite rastrear e gerenciar mudanças em arquivos ao longo do tempo. O Git pensa nas mudanças de maneira mais eficiente e armazena os dados de forma mais eficiente em relação a outros versionadores, o que contribui para sua flexibilidade e rapidez (CHACON; STRAUB, 2014).

### 2.6. GitHub

O GitHub é uma plataforma de hospedagem de código-fonte baseada na web que utiliza o Git como sistema de controle de versões. Ele permite que equipes de desenvolvimento gerenciem e compartilhem código de forma colaborativa, possibilitando a criação de repositórios remotos, onde é possível armazenar versões do código ao longo do tempo. No contexto deste projeto, o GitHub, juntamente com o Git, foi fundamental para o gerenciamento eficiente das versões do código, garantindo a organização, a rastreabilidade e a integridade do trabalho desenvolvido por toda a equipe, além de facilitar o processo de integração e entrega contínua (Github Docs).

### 2.7. NodeJs

Node.js permite a execução de código JavaScript fora do navegador. Ele é especialmente utilizado para o desenvolvimento de aplicações web escaláveis e de alto desempenho, uma vez que é baseado em um modelo de I/O não bloqueador e orientado a eventos. Isso significa que o Node.js pode processar múltiplas requisições simultaneamente sem precisar esperar que uma tarefa seja concluída antes de iniciar outra, o que o torna muito eficiente para aplicações que lidam com um grande número de conexões simultâneas. A grande vantagem do Node.js é que ele utiliza JavaScript tanto no front-end quanto no back-end, proporcionando uma única linguagem para todo o processo de desenvolvimento da aplicação (Node.js).

### 2.8. Express

Express.js é um framework minimalista e flexível para o Node.js, utilizado no desenvolvimento de aplicações web e APIs. Ele oferece um conjunto robusto de funcionalidades para facilitar o roteamento de requisições HTTP, manipulação de respostas, e a organização do código em aplicações web. O Express é amplamente reconhecido por sua simplicidade e escalabilidade, permitindo a construção de aplicações rápidas e de alto desempenho. Ele facilita a criação de rotas, e o tratamento de requisições HTTP, tudo com uma API limpa e de fácil compreensão. Express também é altamente extensível, permitindo a integração com diversas bibliotecas e módulos, o que torna o desenvolvimento de aplicações web mais ágil e produtivo (Express.js).

### 2.9. Token JWT

JWT, ou JSON Web Token, é um meio compacto e seguro para URL de representar reivindicações a serem transferidas entre duas partes. As informações contidas em um JWT são codificadas como um objeto JSON, que é então assinado digitalmente usando um algoritmo criptográfico para garantir sua veracidade. JWTs podem ser assinados usando um segredo (com o algoritmo HMAC ) ou um par de chaves pública/privada usando RSA ou ECDSA. Eles consistem em três partes: um cabeçalho, uma carga útil e uma assinatura (JWT).

#### 2.9.1. Cabeçalho(Header)

O cabeçalho normalmente consiste em duas partes: o tipo do token, que é JWT, e o algoritmo de assinatura (JWT).

#### 2.9.2. Carga útil(Payload)

A carga útil contém as reivindicações, que são declarações sobre uma entidade (normalmente, o usuário) e metadados adicionais (JWT).

#### 2.9.3. Assinatura(Signature)

A assinatura é usada para verificar se o remetente do JWT é quem ele diz ser e para garantir que a mensagem não foi alterada ao longo do caminho (JWT).

Este formato compacto permite que JWTs sejam facilmente passados por uma URL, pelo parâmetro POST ou dentro de um cabeçalho HTTP. Além disso, os JWTs são autocontidos, carregando todas as informações necessárias dentro de si, reduzindo assim a necessidade de um backend para recuperar informações do usuário, o que pode ser benéfico para escalabilidade e desacoplamento (JWT).

## 3. Nginx

O Nginx é um servidor web e proxy reverso que opera em um modelo de processamento assíncrono. Quando uma solicitação HTTP chega no Nginx ele irá criar uma conexão com o cliente e irá decidir como responder, isso pode variar desde buscar um arquivo em disco ou redirecionar para outro serviço, servindo assim como um **Proxy reverso** (GUERRA, 2023).

## 4. METODOLOGIA

A primeira fase do projeto consistiu no levantamento de requisitos, a partir dos requisitos, a arquitetura do sistema foi definida, levando em consideração a escalabilidade, segurança e a integração entre as tecnologias escolhidas, como Node.js, PostgreSQL e Express.

Para controle de versionamento foi utilizado Git e GitHub. A ferramenta Postman foi usada para validar a funcionalidade das APIs, enquanto testes manuais foram realizados na interface do usuário para garantir a experiência desejada.

Após o desenvolvimento de cada funcionalidade, foi feito o deploy do sistema em ambiente de testes e  feito a validação. Feedback dos testes foi usado para ajustes necessários.

As entregas foram feitas de modo a sempre ter um protótipo funcional. A manutenção do sistema será realizada conforme o uso, com atualizações baseadas em feedback e novas necessidades.

## 5. OBJETIVOS

O site tem como objetivo facilitar o gerenciamento de fretes, especialmente para pequenos empreendedores que realizam entregas de seus próprios produtos. O projeto é dividido em duas partes: **frontend** e **backend**.

O **frontend** é desenvolvido com HTML, CSS e JavaScript, e seus arquivos estáticos são servidos pelo Nginx, que atua como servidor web. Enquanto o **backend** é implementado em Node.js com o framework Express. O acesso ao backend também passa pelo Nginx, que nesse caso funciona como proxy reverso.

As **principais funcionalidades** definidas que o site deve dispor estão listadas abaixo:

- **Login e autenticação com JWT**: O sistema de login será realizado por meio de e-mail e senha. O backend deve validar as credenciais e, caso estejam corretas, gerar um par de tokens JWT (auth e refresh). Esses tokens serão retornados ao frontend, que os armazenará nos cookies do navegador.

- **Painel para usuários**: Essa é a função principal do site, onde estarão disponíveis todas as funcionalidades destinadas a usuários autenticados. O sistema deve ser projetado de forma a garantir que apenas usuários com tokens válidos de autenticação consigam acessar essa seção. Qualquer tentativa de acesso sem os tokens apropriados deve ser bloqueada.

- **Gerenciamento dos Fretes**: Esta é a funcionalidade principal do sistema e deve estar integrada ao painel do usuário. Nela, o usuário poderá criar, atualizar e visualizar todas as rotas de entregas que precisam ser realizadas. Cada rota possui diversas entregas, para cada entrega, o usuário deve adicionar os produtos e informar o endereço do cliente correspondente. Essa funcionalidade centraliza o controle logístico, oferecendo uma visão clara e organizada das rotas e das entregas planejadas.

## 6. DESENVOLVIMENTO

O projeto teve o backend e o frontend desenvolvidos separadamente. Embora façam parte de um mesmo sistema, a comunicação entre eles foi estabelecida por meio de uma API.

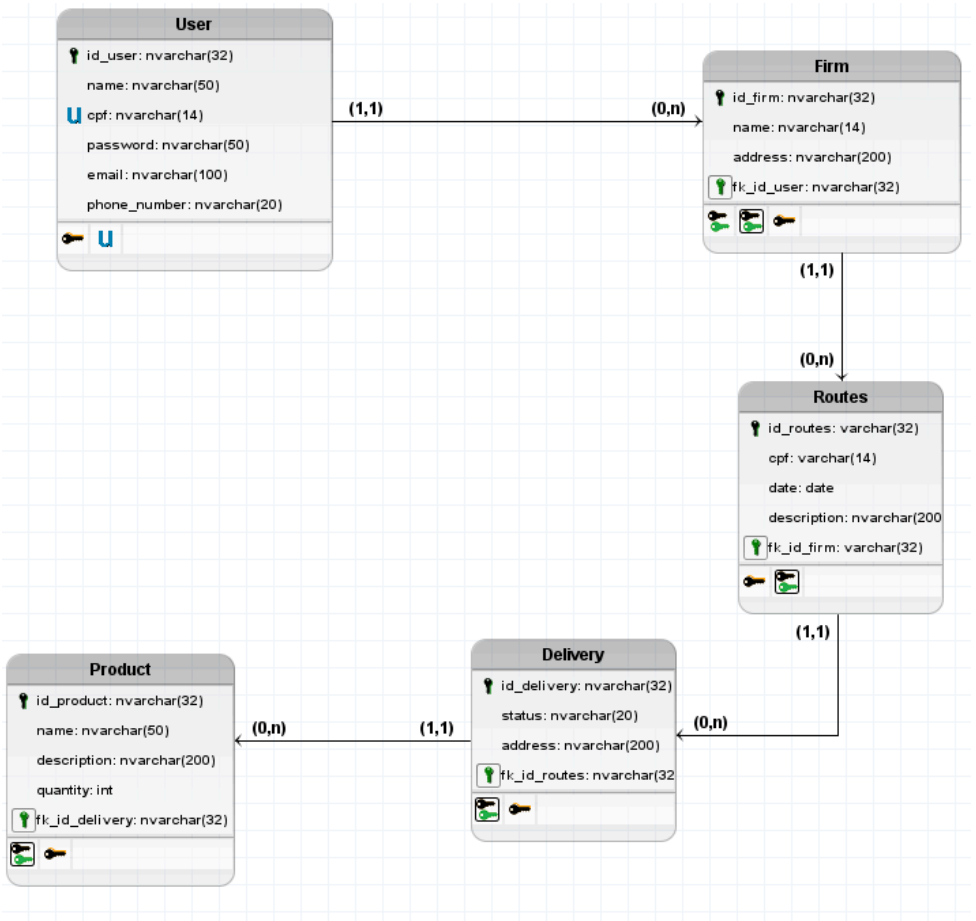
### 6.1. Banco de dados

O banco de dados foi desenvolvido em duas partes, a primeira parte foi a construção da estrutura básica para funcionamento. Enquanto a segunda parte se

deve às tabelas extras que o time em conjunto resolveu adicionar para implementar novas funcionalidades e facilitar a estrutura de armazenamento.

O banco de dados foi estruturado utilizando a ferramenta brModelo, inicialmente foi desenvolvido um modelo ER para poder visualizar a arquitetura básica do BD, como é possível ver na **figura 3**.

**Figura 3** - Modelo ER inicial.




**Fontes**: Autores (2025)

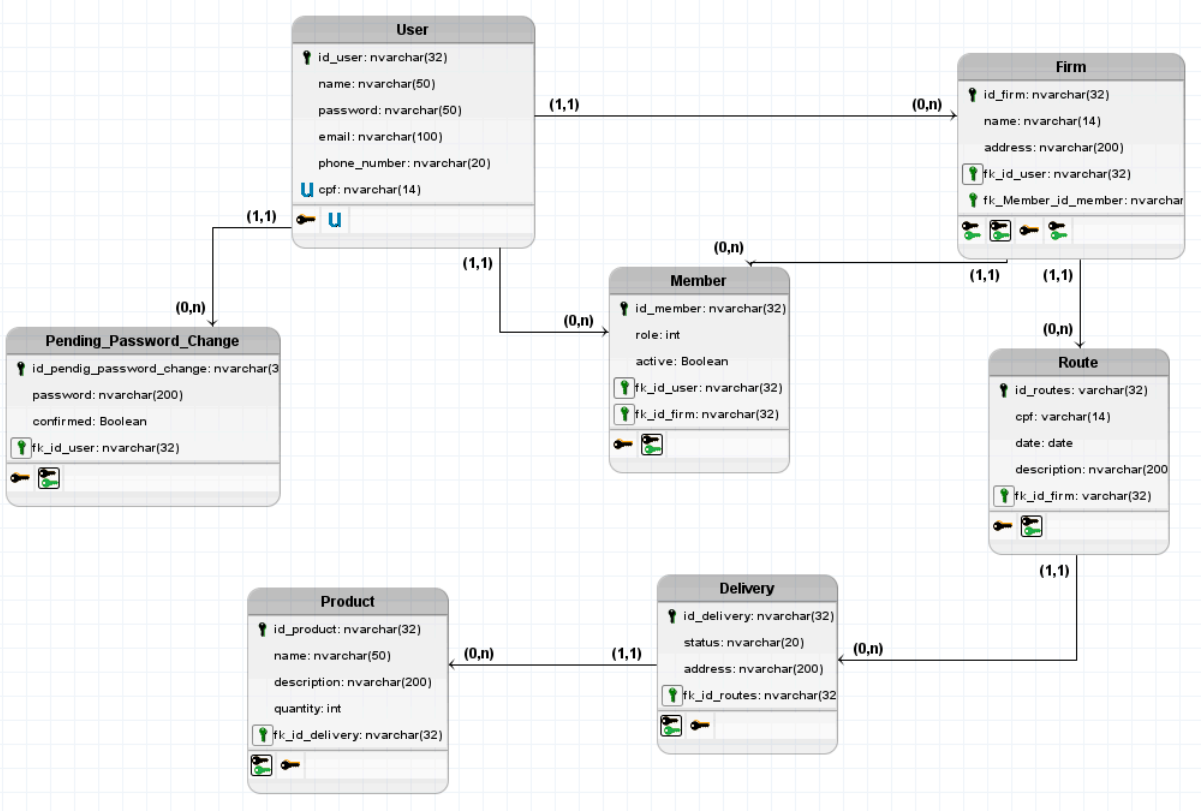
Depois que o banco de dados inicial foi planejado, foi dado início o desenvolvimento, com isso o grupo encontrou novas tabelas que tiveram que ser incluídas, pois, foi necessário se adaptar a novas funcionalidades e melhorar a organização de outras funcionalidades.

As novas tabelas adicionadas foram:

- **Pendig_Password_Change**: Adicionada para a funcionalidade de mudança de senha;
- **Member**: Adicionada para ter a oportunidade de adicionar membros a uma organização.

É possível ver como ficou a estrutura de tabelas na **Figura 4**.

**Figura 4** - Modelo ER Aprimorado




**Fontes:** Autores (2025)

### 6.2. Backend

O backend desenvolvido em Node JS foi arquitetado para ser facilmente escalável e organizável. As funcionalidades principais do backend estão estruturados da seguinte maneira:

- **Routes**: Responsáveis por mapear os endpoints da aplicação. Cada rota define um caminho específico da API e encaminha as requisições para o controlador correspondente..
- **Controller**: Atua como intermediário entre as rotas e os serviços. Recebe os dados das requisições e repassa para os devidos métodos na camada de serviço.
- **Service**: Contém toda a regra de negócio da aplicação para tratar as requisições da melhor maneira possível.
- **Repository**: Possui métodos que são responsáveis por interagir com o Banco de Dados.
- **Models**: Define as estruturas modelo que compõem o Banco de Dados da aplicação.
- **Middlewares**: São middlewares utilizados antes das rotas serem acessadas.
- **Utils**: Contém funções úteis consumidas pela aplicação.
- **Validators**: Contém as validações feitas através da biblioteca zod.
- **Modules**: Contém arquivos de integração de serviços terceiros.

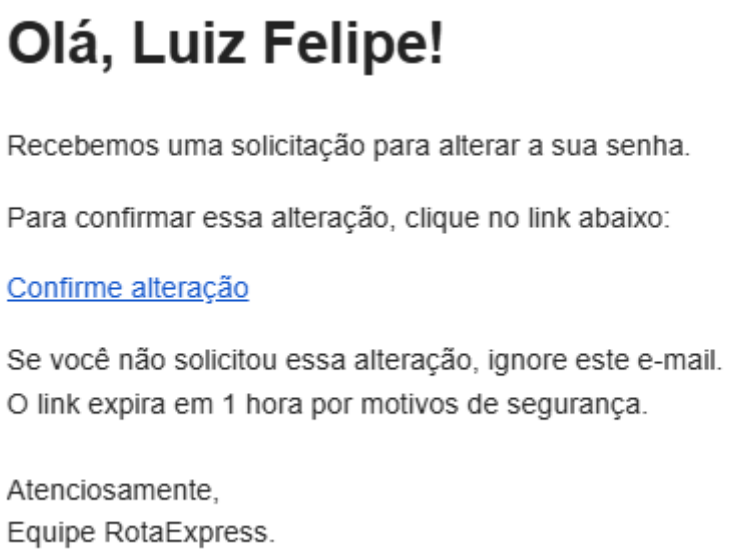
- **BD**: Onde contém o banco de dados e arquivos relacionados com o banco de dados.

### 6.2.1. GMAIL API

O sistema possui integração com GMAIL API, o que possibilita enviar emails para os usuários com informações tratadas importantes para o usuário saber. A integração deve ser inteiramente feita no backend, pois tem dados sensíveis rodando, como tokens.

A integração foi facilitada utilizando a SDK googleapis para o NodeJS. Está integração possibilita enviar mensagens para qualquer e-mail que tiver cadastrado, essa funcionalidade foi utilizada para confirmar a mudança de senhas. Com isso, foi necessário implementar uma nova tabela para armazenar as senhas pendentes, contendo também, um histórico de alterações.

**Figura 5** - E-mail para confirmar alteração da senha




**Fonte**: Autores (2025)

## 6.3. Frontend

O frontend da aplicação foi desenvolvido com foco em simplicidade, organização e facilidade de manutenção. A estilização das páginas foi feita utilizando CSS puro, permitindo controle total sobre o layout e o design visual. O desenvolvimento das interfaces seguiu um planejamento prévio, com os protótipos sendo inicialmente desenhados no papel e também utilizando a ferramenta Figma, o que proporcionou uma visão clara da estrutura e da usabilidade das páginas.

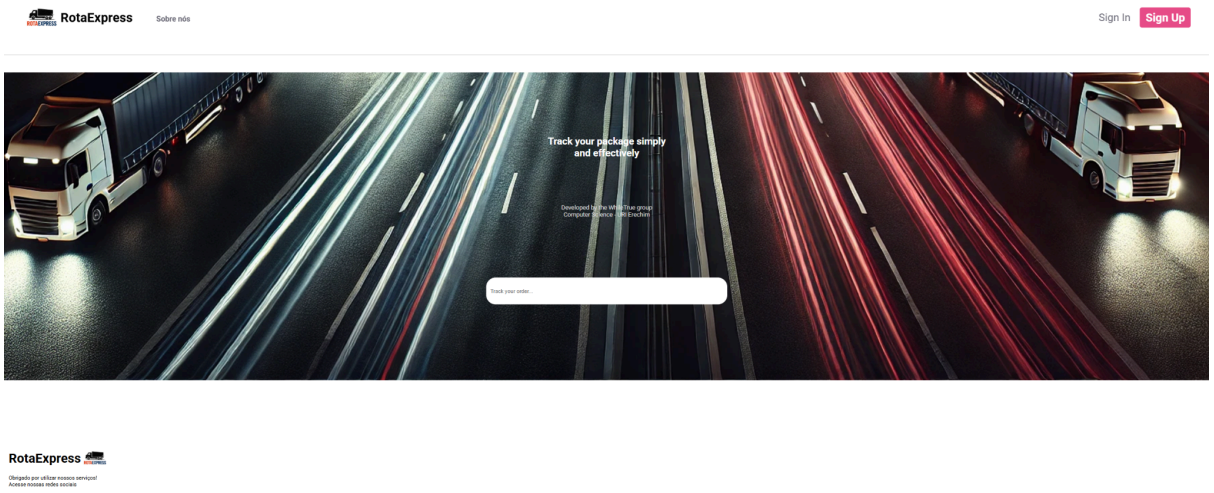
As funcionalidades do frontend estão estruturadas da seguinte maneira:

- **HTML**: Responsável pela estrutura base das páginas, organizando os elementos visuais e garantindo a semântica do conteúdo.

- **CSS**: Utilizado para a personalização visual da aplicação, cuidando de cores, posicionamentos, espaçamentos, responsividade e demais aspectos estéticos.

- **Interatividade**: A navegação entre as páginas e as interações com os dados da API backend foram desenvolvidas de forma simples, prezando por uma experiência fluida para o usuário final.

**figura 6 -** Home page RotaExpress




**Fonte**: Autores (2025)

### 6.4. Integração Frontend x Backend

Essa integração foi realizada por meio de uma API RESTful, onde o frontend consome os endpoints expostos pelo backend para realizar todas as operações de dados.

O fluxo de comunicação ocorre da seguinte maneira: o frontend, através de JavaScript, envia requisições HTTP (como GET, POST, PUT, DELETE) para a API do backend. Por exemplo, quando um usuário preenche o formulário de login, o JavaScript captura os dados e envia uma requisição POST para o endpoint /login da API. O backend, por sua vez, processa essa requisição, valida as credenciais e, em caso de sucesso, retorna um JSON Web Token (JWT), que é armazenado nos cookies do navegador. Para acessar rotas protegidas, como o painel de gerenciamento, o Token é anexado ao cabeçalho de autorização (Authorization) de

cada requisição, permitindo que o backend verifique a autenticidade do usuário antes de liberar o acesso aos recursos.

Para organizar essa arquitetura em um ambiente de desenvolvimento e produção, o Nginx foi configurado para atuar como um proxy reverso. Todas as requisições externas são direcionadas ao Nginx. Ele é responsável por servir os arquivos estáticos do frontend (HTML, CSS, JavaScript) diretamente ao cliente.

Quando uma requisição é destinada à API (por exemplo, um caminho prefixado com /api), o Nginx a redireciona para a aplicação backend, que está em execução em sua própria porta.

Por fim o projeto foi containerizado com Docker, tanto a aplicação backend (Node.js), quanto o servidor web/proxy reverso (Nginx) e o banco de dados (PostgreSQL) foram configurados para rodar em seus próprios contêineres, porém na mesma rede. Utilizando o Docker Compose, todo o ecossistema da aplicação pode ser iniciado e interligado com um único comando, garantindo que a comunicação entre os serviços ocorra de forma padronizada e eficiente, independentemente da máquina onde o projeto é executado.

## 7. CONCLUSÃO

Através deste projeto podemos criar um sistema para gerenciar rotas para entregas, ele atende o objetivo de otimizar operações logísticas de entregas bem como todos os objetivos e funcionalidades pretendidas, bem como algumas extras como o envio de e-mail para confirmação de troca de senha, se trata de um projeto bem arquitetado e robusto, desse modo demonstramos a viabilidade do conceito e execução do plano de desenvolvimento.

No decorrer do projeto enfrentamos e superamos desafios técnicos encontrados durante o desenvolvimento com destaque para a integração do backend com frontend e autenticação de usuários, este desafio foi superado através da implementação de uma arquitetura desacoplada e da construção de uma APIRestfull assim garantindo que a interface do usuário e a lógica do servidor se comunicassem de forma segura e eficiente.

É importante também, reconhecer as limitações que podem atingir o sistema, o principal impedimento está relacionado com a utilização da API do Google para o cálculo de rotas. Embora a cota gratuita de 10.000 chamadas mensais seja suficiente para operações de pequena escala e para a validação do projeto, ela representa um desafio direto à escalabilidade caso o projeto continue se desenvolvendo. Um futuro crescimento no volume de operações levaria a custos de operações que precisam ser considerados, assim os próximos passos para o projeto poderiam envolver a pesquisa de APIs alternativas ou o desenvolvimento de uma solução de rotas proprietária para garantir a sustentabilidade a longo prazo.

## 8. REFERÊNCIAS

ALVES, William P. “**HTML & CSS: aprenda como construir páginas web**”. Rio de Janeiro: Expressa, 2021. E-book. p.7. ISBN 9786558110187. Disponível em:

https://integrada.minhabiblioteca.com.br/reader/books/9786558110187/. Acesso em: 27 mar. 2025.

IBM **“O que é PostgreSQL?”**. Disponível em <https://www.ibm.com/br-pt/topics/postgresql>. Acesso em 25 de mar. de 2025

CHACON, Scott; STRAUB Ben “**Pro GIT**”. 2ª Ed. E-Book. São Francisco: Apress, 2014. ISBN 1484200772.

MDN “**Então, o que realmente é CSS?**”. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn_web_development/Getting_started/Your_first_website/Styling_the_content>. Acesso em 31 de mar. de 2025

Amazon “**O que é o JavaScript (JS)?**”. Disponível em: <https://aws.amazon.com/pt/what-is/javascript/>. Acesso em 31 de mar. 2025

GitHub Docs “**Help for wherever you are on your GitHub journey**”. Disponível em: <https://docs.github.com/en>. Acesso em 1 de abr. de 2025

NodeJs “**Node.js v23.10.0 documentation**”. Disponível em: <https://nodejs.org/docs/latest/api/>. Acesso em 1 de abr. de 2025

Express “**Express5.1.0 Fast, unopinionated, minimalist web framework for Node.js**”. Disponível em: <https://expressjs.com/>. Acesso em 1 de abr. de 2025

JSON Web Token (JWT) “**What is a JWT?**” Disponível em: <https://jwt.is/>. Acesso em 1 de abr. de 2025

GUERRA, Glaucio. “**Nginx Descomplicado: Guia Prático para Configuração e Otimização de Servidores Web**”. 1. ed. São Paulo, 2023. Acesso em 20 de maio de 2025.