Desenvolvimento API de geolocalização

Development API geolocation

Danilo Oliveira Santos¹

Lucas Zanganelli¹

Vinicius Souza¹

Ricardo Silva Campos²

**Resumo**

Este trabalho apresenta o desenvolvimento de uma API genérica de rastreamento de veículos para integrar com diversos sistemas já existentes, visando a melhoria dos processos. Para tanto, foi realizada uma revisão bibliográfica com os principais trabalhos relacionados com esse tema, o que evidenciou uma lacuna de estudos nessa área, pois o material disponível na literatura trata de todo o processo, desde o receptor de GPS até as consultas de localização. A API foi desenvolvida para que os serviços operem de modo independentes. Por esse motivo o sistema foi dividido em três partes: um banco de dados não relacional MongoDB; um *front-end* web utilizando Next.js com a biblioteca ReactLeafleat e um *back-end* desenvolvido em .NET Core 3.1 utilizando os serviços Hangfire e RabbitMQ. Por terem sido utilizados serviços desacoplados, o sistema é tolerante a falhas, o que foi comprovado através da realização de testes que serão descritos no trabalho.

Palavras-chave: Rastreamento de veículos, GPS, API, Computação em nuvem, Containers, Sistema de Informação.

**Abstract**

This work presents the development of a generic API for vehicle tracking to integrate with several existing systems, aiming to improve processes. Therefore, a literature review was carried out with the main works related to this topic, which highlighted a lack of studies in this area, as the material available in the literature deals with the entire process, from the GPS receiver to location queries. The API is designed for services to operate independently. For this reason the system was divided into three parts: a non-relational MongoDB database; a web front-end using Next.js with the ReactLeafleat library and a back-end developed in .NET Core 3.1 using Hangfire and RabbitMQ services. Due to the use of uncoupled services, the system is fault tolerant, which was proven through tests that will be described in the work.

Keywords: Vehicle Tracking, GPS, API, Cloud Computing, Containers, Information System.

**1 INTRODUÇÃO**

Com a constante evolução dos sistemas automatizados e informatizados, surgiu o rastreamento de veículos, que desde o seu surgimento em 1990, passou por constantes atualizações e mudanças para chegar ao formato atual. O formato mais conhecido que o mercado atual de rastreamento utiliza na maioria das vezes fornece um dispositivo que cabe na palma da mão e é conectado a um veículo por meio da tecnologia OBD (*On Board Diagnostic*) (COBLI, 2021).

Os primeiros registros de utilização de GPS ocorreram nos anos de 1970, quando o governo americano desenvolveu o programa NAVSTAR/GPS (Navigation Satellite with Time and Ranging / Global Positioning System) com objetivos militares (CARVALHO E ARAÚJO, 2009). Ao longo dos anos ocorreram muitas melhorias na forma de coletar e tratar os dados coletados. Atualmente existem diversos modelos de dispositivos GPS.

As informações de geolocalização coletadas através do GPS, são enviadas por meio de uma rede de comunicação podendo utilizar algumas tecnologias como 3G, 4G, GSM (*Global System for Mobile Communications*), TDMA (*Time Division Multiple Access*),  GPRS (*General Packet Radio Service*) e outras que utilizem o protocolo TCP/IP (*Transmission Control Protocol / Internet Protocol*) para transmitir os dados que serão gravados no SGBD (Sistema Gerenciador de Banco de Dados) disponível na internet (SEGANTINE, 2005).

Desta forma, observa-se que várias empresas utilizam deste mecanismo que visa resguardar e localizar bens dos clientes que adquirem essa solução, por isso utilizaremos como base para estudo o sistema implementado pela empresa *Rastrepowe*r que já está no mercado há 10 anos.

A *Rastrepowe*r foi fundada em maio de 2002 na cidade de Ubá, e atuava no início com equipamentos voltados para segurança como alarmes e câmeras. Porém em dado momento percebeu-se a necessidade de rastrear os veículos da organização visto que houve um aumento substancial no consumo de combustível dos carros que eram utilizados pelos funcionários.

A partir disso a empresa primeiro teve um contato com um sistema de rastreamento e não obtiveram o retorno desejado. Foi então que a *Rastrepowe*r começou a desenvolver seu próprio sistema para rastrear veículos corporativos e pessoais. Esse sistema é utilizado até os dias atuais. Após uma reunião do grupo com o criador do sistema foi decidido realizar o desenvolvimento de uma API genérica.

Em conciliação com o que foi apresentado sobre a empresa e o estudo de caso dela, foi identificado que atualmente a *Rastrepowe*r enfrenta alguns problemas em relação ao software disponibilizado aos clientes; destacam-se alguns desses problemas como a dificuldade de expansão e manutenção do sistema, alto tempo de resposta para solicitações, ausência de backups e padrões de projeto no código do software.

Com base no que foi abordado, esse trabalho tem como objetivo apresentar uma API de rastreamento de veículos desenvolvida para ser integrada a diversos sistemas visando a melhoria dos processos existentes e a padronização da aplicação, de modo que ela funcione atendendo às mais variadas necessidades.

O desenvolvimento da API será dividido em serviços independentes, separando as etapas do processo para serem executadas normalmente, mesmo que uma das partes esteja fora do ar em um determinado momento. Sendo assim, as informações coletadas pelos receptores GPS serão armazenadas no banco de dados e tratadas para facilitar as consultas e exibições dos resultados nos mapas, de modo que seja simples a implementação em qualquer sistema ou plataforma.

**2 REFERENCIAL TEÓRICO**

Neste capítulo abordaremos temas necessários para o entendimento deste trabalho, com conceitos sobre Geolocalização, GPS e os recursos utilizados no desenvolvimento.

Em busca de uma análise mais ampla sobre o tema GPS, encontramos, nos materiais do Figueirêdo (2005), Bernardi e Landim (2002), informações claras sobre o funcionamento do GPS, os principais segmentos, cartografia, geodésia e posicionamento.

Duarte (2020) utilizando da geometria analítica, geometria plana e geometria espacial, consegue mostrar nos teoremas utilizados os cálculos de distância em uma superfície esférica, nas triangulações e no processo de trilaceração.

**2.1 Geolocalização**

O Datum é uma referência à cartografia para elaboração de mapas, um modelo matemático teórico que representa a superfície terrestre ao nível do mar. Permitindo cálculos no sistema de coordenadas planas UTM (Universal Transversa de Mercator) (FIGUEIREDO, 2005).

O UTM é dividido em 60 fusos, representando as linhas verticais que iniciam a partir do antimeridiano de Greenwich e linhas horizontais a partir do equador. Essas linhas representam as coordenadas de latitude e longitude (FIGUEIREDO, 2005).

Conforme o (IBGE), desde 2005 o SIRGAS 2000 (Sistema de Referência Geocêntrico para as Américas) é o *Datum* oficial para tratamento de geolocalização no Brasil. E no mundo o mais utilizado é o WGS-84 (World Geodetic System/1984).

**2.2 GPS**

Os primeiros registros de utilização de GPS ocorreram nos anos de 1970, quando o governo americano desenvolveu o programa NAVSTAR/GPS (Navigation Satellite with Time and Ranging/Global Positioning System) com objetivos militares (CARVALHO E ARAÚJO, 2009). Desde então ocorreram melhorias na forma de coletar e tratar os dados capturados. De acordo com (FIGUEIREDO, 2005), os dados utilizados pelo GPS vêm do *Datum* geocêntrico, conhecido como WGS-84 (World Geodetic System/1984).

Os receptores de GPS captam os sinais emitidos por uma constelação de satélites distribuídos em seis órbitas terrestres. Dessa forma é garantido que um determinado ponto na terra seja visto pelo menos por quatro satélites ao mesmo tempo. Com a triangulação desses satélites é possível identificar a latitude e longitude trançando um eixo X e Y no sistema UTM.

Os receptores de GPS utilizam o sinal da telefonia celular, seja ela 3G, 4G, GSM (Global System for Mobile Communications), TDMA (TDMA Time Division Multiple Access), GPRS (General Packet Radio Service) desde que utilizem o protocolo de rede TCP/IP (Transmission Control Protocol/Internet Protocol) para transmitir os dados coletados até um servidor de banco de dados que armazenará estas informações (FERREIRA; GONÇALVES, 2014).

**2.3 Recurso utilizados no desenvolvimento**

A linguagem ASP .NET Core 3.1 será utilizada para o desenvolvimento da aplicação. Ela é um *framework* de código aberto desenvolvido pela Microsoft. É uma linguagem dinâmica e de alto nível, com o recurso de *cross-plataform*, que fornece funcionalidades e recursos para implementar aplicativos para os principais sistemas operacionais: Windows, Linux e MacOS.

Para Stefanov (2016) o React é definido como uma biblioteca *JavaScript* de código aberto visando principalmente a construção de interfaces de usuário baseadas em componentes para aplicações web.

A tecnologia *React* foi escolhida para esse trabalho por fornecer velocidade, escalabilidade e simplicidade e por ter como uma de suas principais características é o JSX (JavaScript XML), componentes de estado e modelo de objeto de documento virtual. (KHUAT, 2018).

*NextJS* é um *framework* criado por Vercel para adicionar várias funcionalidades ao React, como a geração de sites estáticos e renderização do lado do servidor (*NEXTJS*, 2021).

*React-Leaflet* é uma biblioteca específica de mapeamento utilizada para representar as posições geográficas e criar mapas com o *React* (REACT-LEAFLET, 2021).

*API* significa em português “Interface de Programação de Aplicativos”, sendo que a sigla provém do inglês “*Application Programming Interface*”. Basicamente a API é composta por várias funções, que na maioria dos casos é acessível somente por programação, permitindo ao utilizador tradicional consumir e utilizar características do software facilmente. A API cria um canal onde desenvolvedores podem acessar serviços para construção de aplicações de forma rápida (JACOBSON; WOODS; BRAIL, 2011).

O *Docker* é uma ferramenta *Open Source* utilizada para gerenciar e estruturar os *containers* no servidor, isolando o *kernel* com as bibliotecas e recursos necessários para rodar a aplicação sob demanda utilizando *containers,* que é uma virtualização fornecida pelo *kernel* com recursos necessários para executar uma determinada aplicação empacotada, os componentes isolam o sistema operacional, o hardware e os processos em execução. Por abstrair dependências de ambiente, iremos utilizar o *docker* para padronizar os ambientes de *deploy*; visando também a expansão, já que ele é de fácil escalonamento (VITALINO; CASTRO, 2016).

O Docker-Compose ajuda na definição de execução de aplicativos Docker em vários contêineres ao mesmo tempo. De forma que, ao executar apenas um comando, esse é replicado em todos os *containers* (DOCKER, 2021).

O Portainer é uma ferramenta para facilitar o trabalho com Containers Docker, centralizando e simplificando a instalação, configuração, gerenciamento e proteção de ambientes com muitos *clusters* (PORTAINER, 2021).

AWS são servidores virtuais, também conhecido como computação em nuvem, que oferecem recursos sob demanda computacional conforme a necessidade da aplicação. Os principais tipos de computação em nuvem são de Infraestrutura, Plataforma ou Software como serviço (ASW, 2021).

O MySQL é um banco de dados relacional, sob a licença GPL de softwares livres. Muito popular, com ampla documentação, utiliza os padrões de comandos SQL, permite multiusuários, é de fácil manutenção, alta compatibilidade, alto desempenho, *multi-threads*, estabilidade e robustez (CARVALHO, 2015).

MongoDB é um banco de dados não relacional, seus dados são armazenados em forma de documentos no lugar de tabelas, colunas e linhas. Tem como recursos nativos a replicação e a tolerância automática de falhas. Possui escalabilidade, desempenho e alta disponibilidade, podendo ser implementado em um ou vários servidores, oferecendo, portanto, uma confiabilidade e flexibilidade operacional na sua utilização (HOWS; MEMBREY; PLUGGE, 2019).

Hangfire é uma biblioteca *open source* desenvolvida para aplicações em .NET, cujo código-fonte está disponibilizado no Github. Possui uma plataforma que auxilia no gerenciamento de tarefas *background*, simplificando o processo dos *jobs*, sejam eles recorrentes ou não (HANGFIRE,2021).

RabbitMQ é um servidor open source que utiliza o protocolo Advanced Message Queuing Protocol (AMQP), proporcionando o tráfego de mensagens seguras e rápidas, compatíveis com diversas linguagens, e possui uma plataforma nativa para administração do serviço de mensageria (CEDRO, 2021).

Identity é uma API .NET para gerenciar a autenticação de usuário, com funcionalidade de perfil, senhas, tokens, confirmação de email. Ela pode utilizar seus dados armazenados ou de provedores externos como Facebook, Google, Microsoft e Twitter (MICROSOFT, 2021).

**2.4 Plataformas de mercado**

Atualmente existem diversos serviços de plataformas que integram os receptores de GPS.

Quatenus, fundada em 2011 e presente em mais de 10 países, oferece serviços de rastreamento, localização em tempo real, otimização de rotas, alcance de metas, monitoramento por câmeras, cerca eletrônica, gestão de manutenção, registros de deslocamento, controle de jornada do motorista e relatórios gerenciais (QUATENUS, 2021).

Cobli, fundada em 2017 e presente em 4 países, oferece serviços como controle de frota, gestão de combustível, controle de manutenção, monitoramento em tempo real, rastreamento, relatórios para análises, status atual do veículo, registro de velocidades, bloqueador de motor e chaveiro para identificação do motorista, (COBLI, 2021).

Rastrepower, fundada 2002 na cidade de Ubá, oferece os serviços de Rastreamento Web 24h, Controle de Velocidade, Indicadores de Performance, Visualização no Mapa, Histórico de Rota, Alarmes de Eventos, Bloqueio de Veículos, Cercas Eletrônicas, Controle de Jornada, Controle de Frota, Roteirização, Controle de Entregas, Controle de Manutenção, Logística.

**3 TRABALHOS RELACIONADOS**

Pereira (2013) propõe um receptor, utilizando uma placa embarcada com um módulo de GPS e um sistema Linux. Esse protótipo é capaz de determinar a posição de um computador devidamente cadastrado na aplicação web e em uma determinada área com sinal de GPS. Neste trabalho, o sistema embarcado foi desenvolvido como receptor GPS para PC, com plataforma Web no mesmo servidor de banco de dados. Quanto ao nosso, usamos rastreadores de veículos de mercado e APIs com serviços de desacoplamento.

Baumgarten (2016) montou um protótipo de receptor GPS com 3G. No desenvolvimento foi utilizado uma distribuição embarcada de Linux chamada OpenWRT com módulos de GPS, Webcam, modem 3G e uma plataforma web para acompanhar o rastreamento, possibilitando uma autonomia em relação aos servidores webs. O nosso trabalho apresenta APIs com serviços desacoplados que podem ser usados ​​com qualquer rastreador de veículos comerciais. Neste trabalho, foi desenvolvido um sistema embarcado com receptor GPS, webcam, 3G e plataforma de rede.

Silva (2021) tem uma proposta de protótipo de sistema embarcado utilizando uma placa Arduino com receptor GPS integrando três etapas diferentes: a primeira é o hardware responsável pela coleta da geolocalização com um módulo GPS; a segunda é um servidor web para armazenamento e exibição dos dados, e, por fim, a terceira etapa é um sistema em Android para receber as coordenadas e exibir a localização em tempo real. O objetivo deste trabalho é desenvolver uma solução desde o hardware até o aplicativo de localização, utilizando um único servidor para processar aplicações e dados. Nossa proposta é usar serviços desacoplados na nuvem para construir uma API que seja fácil de usar em qualquer sistema.

**4 ESTRUTURA DO SISTEMA**

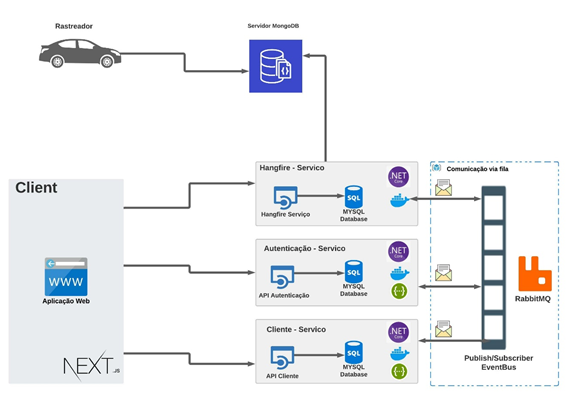
Neste capítulo apresentaremos informações referentes à estrutura utilizada no desenvolvimento do trabalho.

O sistema foi dividido em 3 partes: um banco de dados não relacional MongoDB para coletar os dados de geolocalização enviados pelos automóveis com os equipamentos de GPS configurados; um *front-end* web utilizando Next.js com a biblioteca ReactLeafleat para consultas em tempo real; e um *back-end* desenvolvido em .NET Core 3.1 utilizando os serviços *Hangfire* e RabbitMQ para tratar os dados coletados, armazená-los em um banco de dados relacional MySQL e disponibilizá-los para consultas.

A utilização de dois bancos de dados diferentes, é necessária, pois, o receptor GPS envia uma *string* com várias informações referentes à identificação e geolocalização do veículo, as quais são gravadas no MongoDB. Elas são tratadas, normalizadas e gravadas no MySQL para disponibilizar as consultas da API. Desse modo, são utilizados *containers* separados para ganhar escalabilidade e alta performance nos serviços de autenticação, cliente e *hangfire.*

No *client,* desenvolvido em NextJS, o cliente deverá inserir dados para cadastro e ou *login* e também consumir as informações das localizações. A autenticação tem como objetivo gerenciar a criação de usuários e gerar *tokens* válidos para acesso e consumo da API. O serviço de cliente tem como função registrar o cliente e obter os dados cadastrados. Todos os *containers* são desacoplados e por isso a comunicação é feita através da fila *RabbitMQ*, conforme o esquema proposto na **figura 1**.

**Figura 1:** Estrutura do Sistema

****

Fonte: Produzido pelos autores do artigo.

O rastreador instalado no veículo envia as coordenadas periodicamente para um serviço, que por sua vez armazena os dados georreferenciados para um banco não relacional *MongoDB*. O serviço do *hangfire* irá periodicamente (de 1 em 1 minuto) consultar o banco *MongoDB* para verificar se existem novos registros e, caso exista, irá tratar esses dados identificando o ID rastreador específico do cliente e com isso irá criar tabelas relacionais com os dados geográficos para facilitar o consumo.

**4.1 Serviços**

O desenvolvimento foi baseado em serviços desacoplados, ou seja, todos os serviços são independentes e realizam apenas uma função específica. Desse modo, caso ocorra falha em um serviço específico, os outros não serão afetados. Assim, a aplicação não irá parar por completo, e apenas aquela parte do processo será interrompida. Por exemplo, caso o serviço que coleta os dados de geolocalização pare por um determinado tempo, os clientes ainda conseguiriam autenticar e consultar suas localizações, já que ambos os processos são executados separadamente.

Outro benefício dessa arquitetura é a fácil escalabilidade e manutenibilidade devido ao fato de os serviços serem independentes entre si. Divididos em: autenticação, onde utilizamos a biblioteca *Identity*, com a responsabilidade de autenticar o cliente e criar um login baseado na senha; cliente, com a atribuição de armazenar os dados básicos do cliente; e *Hangfire*, um *job* agendado para consumir o *MongoDB*, tratar os dados e salvá-los em um banco MySQL.

**4.2 Comunicação**

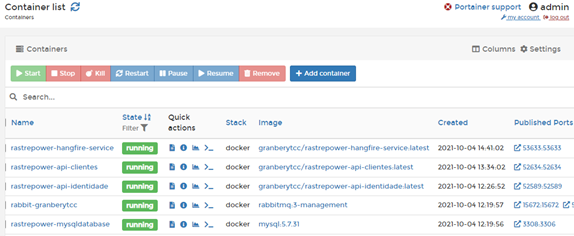
Por ser uma arquitetura desacoplada, os serviços não se comunicam entre si diretamente. Por isso, foi utilizado um gerenciador de fila chamado RabbitMQ, através do qual é possível gerenciar todas as mensagens enviadas por cada serviço e, caso haja necessidade, pode-se usar a escalabilidade para aguentar um alto fluxo de mensagens. Seu uso foi essencial para garantir que nenhuma mensagem seja perdida, mesmo em caso de falhas. Quando o serviço for restabelecido as mensagens serão entregues normalmente.

**4.3 Processo de implantação**

Para o *deploy*, que pode ser compreendido como o processo de implantação do sistema, colocamos toda a solução em *containers* Docker. Com todos os serviços em *containers*, a escalabilidade fica maior e mais fácil. Para gerenciar, iniciar e pausar containers de maneira rápida e fácil, foi utilizada uma imagem do Portainer.

Para gerar a imagem dos serviços, criou-se um *dockerfile* para cada serviço com as imagens do próprio .NET Core. A partir das imagens geradas foi criado um Docker Compose com todas as imagens para gerar os *containers* necessários de uma única vez. Como podem ser vistos os *containers* em execução na figura 2.

**Figura 2:** Painel do Container List



Fonte: Produzido pelos autores do artigo.

**4.4 Processo:**

O cliente adquire um rastreador GPS que possui um ID (número de identificação do rastreador), e é feito o cadastro do cliente na API com os seguintes dados básicos: nome, cpf/cnpj, e-mail e id do rastreador, conforme figura 3. O ID do rastreador deve ser cadastrado com o mesmo número que está instalado no carro do cliente, pois só com ele é possível cruzar as informações específicas de cada rastreador para o cliente cadastrado em nossa base.

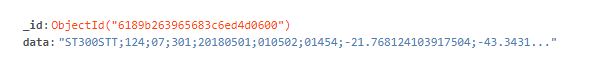
**Figura 3:** Tabela de cliente com ID do Rastreador



Fonte: Produzido pelos autores do artigo.

A configuração inicial do rastreador instalado no automóvel deve apontar diretamente para o endereço do servidor onde o banco de dados *MongoDB* está programado para receber as informações de geolocalização enviadas diretamente do veículo. Após concluída a parametrização, o servidor MongoDB irá receber os dados enviados, inicialmente a cada 30 segundos, no formato de string, demonstrado na figura 4, separados por ponto e vírgula, ou seja, o padrão utilizado pelo GPS.

**Figura 4:** String recebida no MongoDB

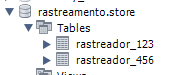


Fonte: Produzido pelos autores do artigo.

Foi configurado um serviço do *Hangfire* que irá executar a cada 15 segundos, verificando os dados existentes e obtendo os registros inseridos no banco MongoDB. Esse *job* tem a função de tratar todos os dados, já que o armazenamento nesse banco é feito de forma desnormalizada e sem separação de dados. No momento da execução do *job*, o *Hangfire* faz a leitura dos dados, que já estão com o ID do rastreador, e obtém as informações necessárias para identificar o cliente (ID do rastreador), latitude, longitude, dados de velocidade e distância. Nesse processo são utilizados somente os dados essenciais, ainda que o rastreador tenha vários outros.

Conforme demonstrado na figura 5, cada cliente possui uma tabela específica com o ID do rastreador.

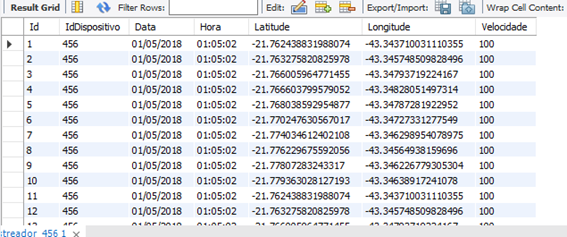
**Figura 5:** Tabelas criadas com ID do rastreador



Fonte: Produzido pelos autores do artigo.

Desse modo, a performance de consulta e escrita fica otimizada, não concorrendo com registros de outros clientes. Portanto, os dados de localização, data, hora e velocidade estão disponíveis para consulta e consumo conforme pode ser visto na figura 6.

**Figura 6:** Tabela com os dados de cada rastreador



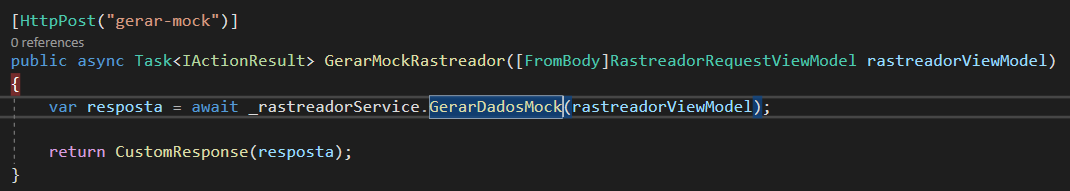
Fonte: Produzido pelos autores do artigo.

**4.5 Teste de software**

Após o desenvolvimento do aplicativo, fez-se necessária a validação das funcionalidades na solução proposta. Para isto, foi executado o *mock* dos dados (geração de dados *fakes* para teste, conforme nas figuras 8 e 9).

O Endpoint “gerar-mock” foi criado com o intuito de gerar dados automaticamente para os testes funcionais da aplicação, conforme demonstrado na figura 7.

**Figura 7:** Método De Mock



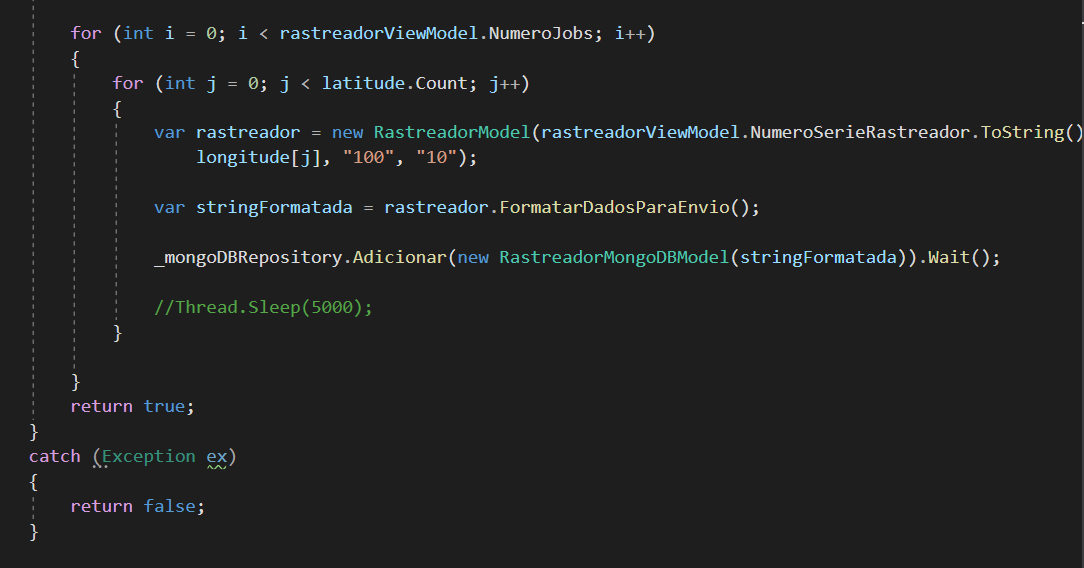
Fonte: Produzido pelos autores do artigo.

**Figura 8:** Função De Mock parte 1



Fonte: Produzido pelos autores do artigo.

**Figura 9:** Função De Mock parte 2



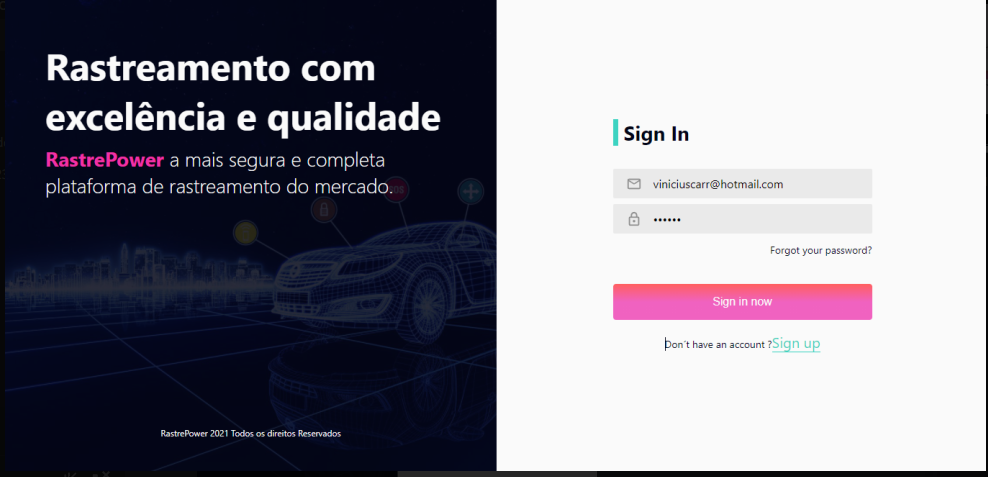
Fonte: Produzido pelos autores do artigo.

Foram realizados os cadastros dos clientes com algumas localizações simulando o rastreador, com isso criamos um *endpoint* específico para a geração desses dados e inserção no banco não relacional (*MongoDB*). Após a geração desses dados, o serviço do *hangfire* irá executar o seu processo e tratar a informação que se encontra no banco não relacional (*MongoDB*).

**4.6 Telas desenvolvidas**

Foi criada a tela de *login* do aplicativo, com um visual clean e intuitivo, com local para preenchimento do login e da senha de acesso ao sistema. demonstradas na figura 10:

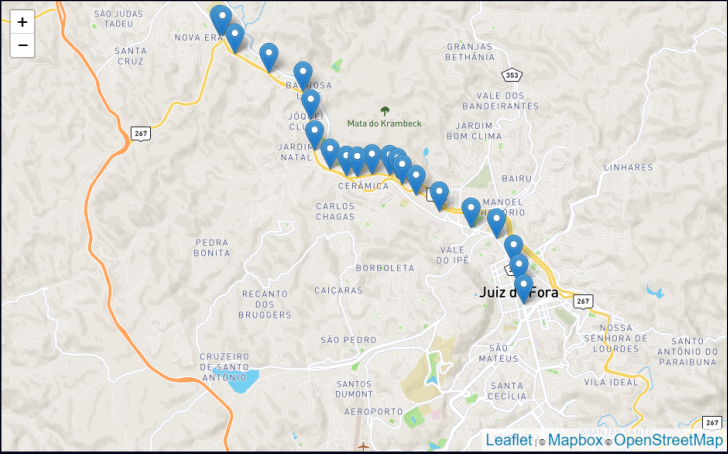
**Figura 10:** Tela de *Login*



Fonte: Produzido pelos autores do artigo.

No mapa consta a trajetória por onde o veículo passou. Os pontos azuis são baseados pela latitude e longitude enviada através da *API* conforme demonstrado na figura 11*.*

**Figura 11:** Tela do MAPA

**

Fonte: Produzido pelos autores do artigo.

**4.7 Resultados**

A *API* desenvolvida teve como resultado a facilitação das consultas e do consumo dos dados recebidos. Além disso, por ser totalmente genérica, pode ser utilizada em vários segmentos, e por possuir essa característica, pode ser utilizada por aplicativos *mobile*, *desktop* e também pelo *browser,* necessitando apenas de uma chamada em HTTP (*Hyper Text Transfer Protocol*).

Por terem sido utilizados serviços desacoplados, o sistema é tolerante a falhas, ou seja, se um serviço não estiver em condições de uso, isso não interrompe o funcionamento total da ferramenta. Para tal comprovação foram realizados os seguintes testes:

1°- O serviço de autenticação foi desligado, com isso os usuários já logados conseguiam visualizar e consultar os dados referentes ao veículo cadastrado, o único processo que ficou impossibilitado foi o de autenticação.

2°- O serviço de cliente foi desligado, com isso os usuários conseguiam logar e autenticar porém a localização e as consultas não estavam disponíveis, conforme o esperado.

3°- O serviço do *hangfire* foi desligado, com isso os usuários conseguiam autenticar, verificar as suas localizações e consultar os dados. Porém, os novos registros só ficaram disponíveis depois de ligado novamente.

**5 CONCLUSÃO**

A revisão da literatura realizada comprova que existe uma lacuna referente à falta de APIs para consumir os dados do GPS e integrá-los facilmente com qualquer sistema. Além disso, também esclarece a importância da criação de uma API genérica que não está disponível no mercado para integrar qualquer sistema ou plataforma que requeira a utilização de serviços de rastreamento.

A API deve beneficiar empresas que usam plataformas de geolocalização externas e não podem se integrar com seus sistemas de gestão. Por se tratar de requisições *web*, de instalação simples, seu uso é mais fácil e prático, permitindo realizar consultas de localização diretamente de seu sistema interno.

Como limitação, esta aplicação é utilizada apenas para receber e consultar dados de localização, sendo necessário adquirir um dispositivo de localização GPS, instalá-lo no automóvel e efetuar as configurações necessárias. Como uma possível melhoria e sugestão para o futuro, destacamos a hospedagem em um ambiente de computação em nuvem para melhor desempenho e escalabilidade. Portanto, quando um grande número de usuários começarem a utilizar a API, a nuvem tem o benefício do desacoplamento de serviço, redundância e segurança. Por ser genérica, é fácil de implementar em qualquer sistema.

**REFERÊNCIAS**

AWS, Disponível em https://aws.amazon.com/pt/ Acessado em 01/10/2021

BAUMGARTEN, Nykolas Eduardo Antonioli. **Findcar: Rastreador Veicular Utilizando Openwrt.** Orientador: Miguel Alexandre Wisintainer. 2016. Trabalho de conclusão de curso (Ciência da Computação) - Universidade Regional de Blumenau, Blumenau, 2016.

BERNARDI, José Vicente Elias; LANDIM, Paulo M. Barbosa. **Aplicação do Sistema de Posicionamento Global (GPS) na coleta de dados**. Universidade Estadual Paulista Rio Claro, Rio Claro, 2002

CARVALHO, Edilson Alves; ARAÚJO, Paulo Cesar. Leituras cartográficas e interpretações estatísticas II - Aula 8 - Noções básicas de sistema de posicionamento global GPS. Natal, RN: EDUFRN, 2009.

CARVALHO, Vinícius. MySQL: Comece com o principal banco de dados open source do mercado. Editora Casa dos Código, 2015.

CEDRO, Disponível em https://blog.cedrotech.com/rabbitmq-o-que-e-e-como-utilizar. Acesso em 10/10/2021.

COBLI, Disponível em https://www.cobli.co/blog/rastreamento-veicular/. Acesso em 21/04/2021.

DOCKER, Disponível em https://docs.docker.com/compose/. Acessado em 05/10/2021

DUARTE, Alessandra de Jesus. **Sistema de Posicionamento Global (GPS): uma aplicação da geometria analítica**. Orientadora: Lucinda Maria de Fátima Rodrigues Coelho. 2020. Trabalho de conclusão de curso (Licenciatura em Matemática) – Centro Universitário Municipal de Franca, Franca, 2020.

FERREIRA, Alan Azevedo Jacundá; GONÇALVES, Willton Oliveira. **Rastreamento Veicular com auxílio de dispositivos móveis**. Orientador: Georges Daniel Amvame Nze. 2014. Trabalho de conclusão de curso (Engenharia Eletrônica) - Universidade de Brasília, Brasília, 2014.

FIGUEIRÊDO, Divino Cristino. Curso Básico de GPS, 2005 <http://seivamt.com.br/anexo_artigos/21.pdf>. Acesso em 21/04/2021.

HANGFIRE, Disponível em https://www.hangfire.io. Acesso em 01/10/2021.

HOWS, David; MEMBREY, Peter; PLUGGE, Eelco. Introdução ao MongoDB. Editora Novatec Editora, 2019.

IBGE Redes Geodésicas https://www.ibge.gov.br/geociencias/informacoes-sobre-posicionamento-geodesico/rede-geodesica.html. Acesso em 25/04/2021.

JACOBSON, Daniel; BRAIL, Greg; WOODS, Dan. APIs: A Strategy Guide: Creating Channels with Application Programming Interfaces. Editora O'Reilly Media, 2011.

KHUAT, Tung. Developing a frontend application using ReactJS and Redux. Dissertação (Degree Programme in Business Information Technology Bache lor’s) — Laurea University of Applied Sciences, Leppävaara, 2018. https://www.theseus.fi/bitstream/handle/10024/150837/Tung\_Khuat\_1301747\_Thesis.pdf?sequence=1&isAllowed=y. Acesso em: 30/05/2021.

MICROSOFT, Disponível em https://docs.microsoft.com/pt-br/aspnet/core/security/authentication/identity?view=aspnetcore-5.0&tabs=visual-studio. Acessado em 25/10/2021.

NEXT JS, Disponível em <https://nextjs.org>. Acesso em 01/10/2021.

PEREIRA, Bárbara Dias. **Protótipo de um Sistema de Rastreamento para Computadores**. Orientador: Miguel Alexandre Wisintainer. 2013. Trabalho de conclusão de curso (Sistema de Informação) - Universidade Regional de Blumenau, Blumenau, 2013.

PORTAINER, Disponível em <https://www.portainer.io>. Acesso em 01/10/2021

QUATENUS, Disponível em <https://www.quatenusonline.com.br/>. Acesso em 23/08/2021.

REACT-LEAFLET, Disponível em https://react-leaflet.js.org. Acesso em 01/10/2021.

SEGANTINE, Paulo Cesar Lima. GPS: sistema de posicionamento global. EESC/USP, 2005.

SILVA, André Menezes Melo e. **Sistema para rastreamento veicular via GPS.** Orientador: Carlos Moratelli. 2021. Trabalho de conclusão de curso (Engenharia de Controle e Automação) - Universidade Federal de Santa Catarina, Blumenau, 2021.

STEFANOV, Stoyan. Primeiros Passos com React: Construindo aplicações Web. Editora Novatec, 2016.

VITALINO, Jeferson Fernando Noronha; CASTRO, Marcus André Nunes. Descomplicando o Docker. Editora Brasport, 2016.