INSTITUTO FEDERAL
Rondônia | Campus Vilhena

Gustavo Vinicius Duarte Da Silva

Análise de dados de acidentes de trânsito

Vilhena - RO

2025

INSTITUTO FEDERAL
Rondônia | Campus Vilhena

Gustavo Vinicius Duarte Da Silva

Análise de dados de acidentes de trânsito

Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Educação, Ciência e Tecnologia de Rondônia – campus Vilhena, realizado em cumprimento de requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Instituto Federal de Educação, Ciência e Tecnologia de Rondônia – IFRO

Campus Vilhena

Curso Superior em Tecnologia em Análise e Desenvolvimento de Sistemas

Orientador: Esp. Erick Leonardo Weil

Vilhena - RO

2025

Ficha catalográfica elaborada pelo Sistema Gerador de Ficha Catalográfica do IFRO.

Silva, Gustavo Vinicius Duarte da.
Análise de dados de acidentes de trânsito / Gustavo Vinicius Duarte da Silva. - Vilhena, 2025.
54 f.

Orientador(a): Prof. Esp. Erick Leonardo Weil.

Trabalho de Conclusão de Curso (Superior de Tecnologia em Análise e Desenvolvimento de Sistemas) – Instituto Federal de Educação, Ciência e Tecnologia de Rondônia - IFRO, Vilhena, 2025.

1. Acidentes de trânsito. 2. Análise de dados. 3. Linguagem GO. 4. Goroutines. 5. Processamento de dados. I. Weil, Erick Leonardo (orient.). II. Instituto Federal de Educação, Ciência e Tecnologia de Rondônia - IFRO. III. Título.

Bibliotecário(a) Responsável: Rosilene Maria do Couto Marques, CRB-11/321

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE RONDÔNIA
Vilhena - Código INEP: 11107804
Rodovia BR 174, KM 3, CEP 76982-270, Vilhena (RO)
CNPJ: 10.817.343/0003-69 - Telefone: 69 2101-0703

ATA DE DEFESA DE MONOGRAFIA

Na data 04/07/2025 realizou-se a sessão pública de defesa da Monografia intitulada **Análise de dados de acidentes de trânsito** apresentada pelo aluno **Gustavo Vinicius Duarte da Silva (2022103070004)** do Curso **Superior de Tecnologia em Análise e Desenvolvimento de Sistemas (Vilhena)**. Os trabalhos foram iniciados às **16:00** pelo Professor **Erick Leonardo Weil** presidente da banca examinadora, constituída pelos seguintes membros:

- **Erick Leonardo Weil** (Orientador)
- **Wesley Jhonnes Ramos Rolim** (Examinador Interno)
- **Gilberto Pereira da Silva** (Examinador Interno)

A banca examinadora, tendo terminado a apresentação do conteúdo da Monografia, passou à arguição do candidato. Em seguida, os examinadores reuniram-se para avaliação e deram o parecer final sobre o trabalho apresentado pelo aluno, tendo sido atribuído o seguinte resultado:

[X] APROVADO **Nota: 71**

Proclamados os resultados pelo presidente da banca examinadora, foram encerrados os trabalhos e, para constar, eu **Erick Leonardo Weil** lavrei a presente ata que assino juntamente com os demais membros da banca examinadora.

VILHENA / RO, 04/07/2025

Documento assinado eletronicamente por **Gustavo Vinicius Duarte da Silva**, Discente, em 04/07/2025, às 17:06, conforme horário oficial de Rondônia, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Erick Leonardo Weil**, Orientador, em 04/07/2025, às 17:05, conforme horário oficial de Rondônia, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Wesley Jhonnes Ramos Rolim**, Examinador Interno, em 04/07/2025, às 17:06, conforme horário oficial de Rondônia, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Gilberto Pereira da Silva**, Examinador Interno, em 04/07/2025, às 17:06, conforme horário oficial de Rondônia, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Dedico este trabalho a todas as pessoas que contribuíram para minha formação.

Agradecimentos

Agradeço imensamente aos meus pais, Ginovaldo Gomes da Silva e Lisander Santos Duarte da Silva, ao meu irmão William Mateus e à minha namorada Giselly, pelo amor, incentivo e apoio incondicional, durante toda a minha jornada acadêmica. Agradeço também aos meus amigos do projeto cidades inteligentes, por todas as discussões e colaborações que enriqueceram este trabalho.

RESUMO

A crescente disponibilidade de dados sobre acidentes de trânsito no Brasil exige ferramentas eficientes para sua organização e análise. Este estudo apresenta o desenvolvimento de um sistema para processamento de grandes volumes de informações utilizando a linguagem Go e suas goroutines. A metodologia adotada envolve a coleta, estruturação e tratamento dos dados, permitindo sua organização de forma otimizada para futuras consultas e análises. O trabalho detalha o processo de desenvolvimento e as decisões técnicas adotadas, ressaltando a aplicação de programação paralela para lidar com grandes conjuntos de dados. Conclui-se que a abordagem implementada oferece uma base estruturada para análise de dados sobre acidentes de trânsito, facilitando investigações futuras sobre o tema.

Palavras-chave: acidentes de trânsito; análise de dados; linguagem Go; goroutines; processamento de dados; big data.

Abstract

The increasing availability of traffic accident data in Brazil demands efficient tools for its organization and analysis. This study presents the development of a system for processing large volumes of information using the Go programming language and its goroutines. The adopted methodology involves data collection, structuring, and processing, enabling optimized organization for future queries and analyses. The study details the development process and the technical decisions made, highlighting the application of parallel programming to handle large datasets. It is concluded that the implemented approach provides a structured basis for traffic accident data analysis, facilitating future research on the subject.

Keywords: traffic accidents; data analysis; Go programming language; goroutines; data processing; big data.

Lista de ilustrações

Lista de abreviaturas e siglas

API	Application Programming Interface
CRUD	Acrônimo para Create (criação), Read (consulta), Update (atualização) e Delete (destruição) de dados
MVP	Minimum Viable Product
QA	Quality Assurance
CSV	Comma-Separated Values
DETRAN-RO	Departamento de Trânsito de Rondônia
IBGE	Instituto Brasileiro de Geografia e Estatística
IPEA	Instituto de Pesquisa Econômica Aplicada

Sumário

1 Introdução

1.1 Contexto e problema

O aumento da mortalidade no trânsito brasileiro tem sido uma preocupação crescente nos últimos anos. Segundo o portal Metropoles (2024), com base no relatório da Organização Mundial da Saúde (OMS, 2025), o número de óbitos em acidentes rodoviários no país passou de 32.667 em 2019 para 32.716 em 2020, chegando a 33.813 em 2021. Esse crescimento representa um acréscimo de 3,5% no período, evidenciando a necessidade de políticas públicas mais eficazes para a redução dessas fatalidades.

A disponibilização de informação é uma ferramenta essencial para a conscientização da sociedade. A falta de acesso a dados relevantes pode limitar a percepção da população sobre determinados problemas e dificultar a formulação de soluções eficazes. Nesse sentido, disponibilizar dados detalhados sobre acidentes de trânsito pode contribuir para a conscientização e para a implementação de políticas públicas mais eficientes.

Entretanto, as ferramentas públicas atualmente disponíveis para acesso a dados de trânsito no Brasil apresentam sérias limitações de usabilidade e visualização. Segundo reportagem da Luany (2024), publicada na *folha de S.Paulo*, baseada em dados do Observatório Nacional de Segurança Viária, “em doze estados não há disponibilidade de informações online sobre sinistros, incluindo mortes provocadas por acidente”. Ainda segundo o levantamento, quase metade dos Departamentos Estaduais de Trânsito (DETRAN) brasileiros não publica dados online de forma estruturada ou acessível ao público geral.

Além disso, o presidente da Federação Nacional das Vistorias Automotivas (FENIVE), Everton Pedroso, critica a precariedade da coleta e publicação de dados "O que deveria ser uma prioridade básica a coleta de informações precisas sobre acidentes de trânsito é frequentemente negligenciada pelos órgãos de trânsito”, (PORTAL DO TRâNSITO, 2024).

Essas deficiências refletem não apenas a falta de transparência, mas também a ausência de painéis interativos com gráficos dinâmicos, filtros e visualizações intuitivas que permitiriam à sociedade civil, pesquisadores e gestores públicos interpretar e explorar os dados diretamente.

Relatórios como o *Índice de Dados Abertos para Cidades* (Open Knowledge Brasil e FGV DAPP) já alertavam que muitos portais municipais brasileiros não apenas disponibilizam dados incompletos, como o fazem de maneira visualmente confusa, sem documentação e sem forma de download completa (BRASIL; DAPP, 2018). Como destaca Ruediger (2024), "é um desserviço o descaso generalizado das esferas públicas com a capacidade de

traduzir números. Esses problemas prejudicam a interpretação dos dados e impedem que cidadãos e sociedade civil acompanhem e discutam políticas públicas de forma informada".

1.2 Objetivos

1.2.1 Objetivo geral

O objetivo deste trabalho é possibilitar visualização de dados de acidentes de trânsito sob diferentes perspectivas e realizar filtragens específicas para o usuário. Com este intuito, propõe-se o desenvolvimento de uma aplicação web que permitirá a visualização de estatísticas de acidentes de trânsito com base em dados disponibilizados pelo Ministério dos Transportes. A aplicação contará com um sistema back-end para processamento dos dados e um front-end para exibição das estatísticas, que processará arquivos CSV (*Comma-Separated Values*) ou em português (valores separados por vírgulas) fornecidos pelo governo, agrupando e somando as informações para apresentá-las em um WebSite.

1.2.2 Objetivos específicos

Os objetivos específicos deste trabalho são:

- Delimitar um MVP (*Minimum Viable Product*)[1] de uma solução para visualização de dados de acidente de trânsito;
- Desenvolver uma solução para agrupar e somar informações trazidas por um arquivo CSV (*Comma-Separated Values*)[2] que possa agrupar e somar de forma rápida e eficiente;
- Desenvolver site com apresentação de dados.

1.3 Justificativa

O aumento contínuo de mortes no trânsito no Brasil, somado à limitação no acesso a dados públicos estruturados, evidencia a necessidade de ferramentas tecnológicas que facilitem a compreensão e análise desses dados. Embora existam portais oficiais, como os da Polícia Rodoviária Federal (PRF) e de alguns DETRANs, muitos estados sequer disponibilizam informações online ou o fazem de forma bruta, dificultando a identificação de padrões e a formulação de políticas públicas Luany (2024), Matheus et al. (2016).

Além disso, a apresentação dos dados em formatos tabulares, como planilhas CSV, limita a interpretação a usuários com conhecimento técnico. Ribeiro (2009) em sua

[1] Produto Mínimo Viável
[2] Valores Separados por Vírgulas

pesquisa sobre visualização de dados na internet, destaca que o uso de representações cartográficas digitais pode tornar a informação mais acessível e compreensível, facilitando a análise crítica por públicos diversos. Embora o autor enfoque a cartografia digital, sua argumentação reforça a importância da visualização interativa de dados como um meio eficaz de ampliar o acesso à informação. Nesse sentido, observa-se que plataformas públicas voltadas à segurança no trânsito ainda carecem de recursos que favoreçam esse tipo de interação, justificando o desenvolvimento de soluções que combinem mapas, gráficos e tabelas interativas para apoiar a análise e a tomada de decisão.

Diante disso, este projeto propõe o desenvolvimento de uma aplicação que centraliza e apresenta dados de acidentes de trânsito de forma acessível, visual e interativa. A proposta inclui filtros dinâmicos e visualizações geográficas que tornam possível a análise por estado, ano, quantidade de envolvidos e óbitos, permitindo maior compreensão por parte da população e dos gestores públicos, com potencial de impacto na redução de acidentes e no aumento da segurança viária.

2 Fundamentação teórica

A análise de dados relacionados a acidentes de trânsito é uma ferramenta essencial para compreender os fatores que os causam e desenvolver estratégias eficazes de mitigação. A coleta e o tratamento dessas informações possibilitam uma visão mais detalhada das ocorrências, permitindo a identificação de padrões e auxiliando na tomada de decisões para a melhoria da segurança viária.

A visualização de dados desempenha um papel fundamental nesse contexto, pois transforma informações brutas em representações gráficas que facilitam a interpretação e a análise. Ferramentas interativas permitem explorar os dados de forma dinâmica, possibilitando a identificação de tendências e variáveis críticas, como localização, horário, condições climáticas e tipo de via. Dessa forma, a utilização de painéis interativos e gráficos dinâmicos contribui para a formulação de políticas públicas mais assertivas e para o desenvolvimento de estratégias preventivas. Conforme Tufte (1983), em *The Visual Display of Quantitative Information*[1], a excelência gráfica ocorre quando a apresentação visual de dados é bem projetada, permitindo a comunicação eficiente de informações complexas.

Além disso, a disponibilização pública de dados tem ganhado espaço como um mecanismo essencial para ampliar a transparência e incentivar a participação da sociedade. O acesso aberto a informações sobre acidentes de trânsito permite que pesquisadores, gestores e cidadãos utilizem esses dados para monitoramento, planejamento e implementação de soluções que possam reduzir o número de ocorrências e aprimorar a segurança no trânsito. De acordo com Berners-Lee (2009), em *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web*[2], a disponibilização de dados públicos de forma acessível e aberta tem um impacto positivo na sociedade, promovendo maior transparência e engajamento social.

Este trabalho também destaca a importância da concorrência no processamento de grandes volumes de dados, uma necessidade fundamental na análise de informações sobre acidentes de trânsito. A abordagem desenvolvida explora o benefício da execução de tarefas em concorrência, contrastando-a com o processamento linear tradicional para demonstrar ganhos significativos de performance. Segundo (TANENBAUM; STEEN, 2007), a execução concorrente permite que múltiplas tarefas sejam realizadas simultaneamente, aumentando a eficiência e escalabilidade de sistemas distribuídos.

[1] A exibição visual de informações quantitativas
[2] Tecendo a Web: O Design Original e o Destino Final da World Wide Web

Goroutine

A *goroutine* é um recurso nativo da linguagem de programação Go, projetado para facilitar a execução concorrente de múltiplas tarefas com um consumo de recursos significativamente baixo. Essa característica é crucial para otimizar o processamento de grandes volumes de dados, como os registros de acidentes de trânsito. Segundo Donovan e Kernighan (2015) a principal vantagem reside na capacidade de dividir uma carga de trabalho complexa em partes menores, que podem ser processadas em paralelo. Por exemplo, um conjunto de 8.000 linhas de dados pode ser distribuído entre 8 "trabalhadores" (goroutines), onde cada um processa 1.000 linhas, distribuindo o esforço e acelerando a conclusão da tarefa. Essa estratégia é análoga à divisão de tarefas em uma equipe humana, onde cada membro se dedica a um segmento para otimizar o resultado coletivo.

O conceito de concorrência não é novo e está presente em sistemas computacionais modernos, desde computadores pessoais até smartphones, que executam múltiplas aplicações simultaneamente. Historicamente, os processadores evoluíram de arquiteturas que permitiam a execução de apenas uma tarefa por vez (*single-threaded*)[3] para arquiteturas (*multi-core*)[4], capazes de gerenciar e dividir o trabalho entre múltiplos núcleos e threads Tanenbaum e Bos (2015).

No entanto, há uma distinção importante entre as threads tradicionais, gerenciadas pelo sistema operacional, e as goroutines de Go. As threads são gerenciadas pelo kernel do sistema operacional através de um componente chamado scheduler. Esse scheduler executa-se diversas vezes por segundo para gerenciar a troca de contexto entre as threads. Esse processo de gerenciamento do kernel é, por natureza, custoso em termos de recursos e tempo, podendo impactar a performance, conforme apontado por Temporin (2022) em "Quais as diferenças entre goroutines e threads - Aprenda Golang" e por Silberschatz, Galvin e Gagne (2018), no livro *Operating System Concept*.

Em contrapartida, a linguagem Go emprega um modelo de gerenciamento de concorrência mais eficiente, conhecido como agendamento M:N (*Many-to-Many*)[5]. Nesse modelo, o scheduler de Go é responsável por mapear M goroutines para N threads do sistema operacional. Isso significa que múltiplas goroutines podem ser executadas em uma única thread do sistema operacional. Essa abordagem reduz significativamente o custo de troca de contexto, pois o scheduler de Go pode alternar a execução entre goroutines de forma muito mais rápida e leve do que o scheduler do sistema operacional entre threads. Consequentemente, o uso de goroutines oferece ganhos notáveis de performance, especialmente em aplicações que demandam alta concorrência e processamento intensivo de dados, conforme detalhado por Kennedy, Ketelsen e Martin (2015) no livro *Go in*

[3] thread única
[4] multinúcleo
[5] muitos para muitos

action.

Redis

Com o intuito de guardar os valores processados, o Redis, configurado para persistência em disco, exerceu como solução de banco de dados para atender às demandas de armazenamento e recuperação de dados. Sua arquitetura *key-value*[6], aliada à otimização de acesso a disco, é crucial para garantir um desempenho consistente, especialmente em operações relacionadas a análises dinâmicas e em tempo real, conforme o Wagner (2025) discute em seu artigo "O Básico de Redis Cache Distribuído" publicado na plataforma Medium e por Lazoti (2020), no livro *Armazenando dados com Redis*.

Diante desse cenário, este trabalho propõe o desenvolvimento de uma solução para a visualização interativa de dados de acidentes de trânsito, integrando técnicas de análise e apresentação de informações. Ao estruturar os dados de maneira acessível e interativa, espera-se fornecer uma ferramenta que auxilie na compreensão das ocorrências e na formulação de estratégias preventivas eficazes.

2.1 Trabalhos similares

Os seguintes softwares proprietários, para apresentação de dados sobre acidentes de trânsito, foram encontrados durante esta pesquisa:

- DETRAN-RO[7]: plataforma para visualização de acidentes de trânsito do estado de Rondônia.
- IBGE [8]: plataforma para visualização de dados sobre acidentes de trânsito e outras estatísticas.
- Atlas Volvo [9]: site destinado a acidentes de trânsito do Brasil em rodovias federais e estaduais.

[6] chave-valor
[7] Disponível em https://www.detran.ro.gov.br/post/39/2023/9/12/anuario-estatistico-de-sinistros-de-transito-de-rondonia/
[8] Disponível em https://seculoxx.ibge.gov.br/populacionais-sociais-politicas-e-culturais/busca-por-palavra-chave/justica/787-transito
[9] Disponível em https://www.volvogroup.com/br/news-and-media/news/2017/aug/atlas-da-acidentalidade.html

3 Materiais e métodos

Neste capítulo, são detalhados os materiais e métodos utilizados no desenvolvimento da solução proposta para a análise eficiente de dados de acidentes de trânsito. A abordagem adotada busca otimizar o processamento e a manipulação de grandes volumes de dados, proporcionando uma análise mais ágil e performática em comparação com métodos tradicionais. Além disso, a solução inclui a visualização interativa dessas informações por meio de uma aplicação web.

3.1 Fluxo de execução da aplicação

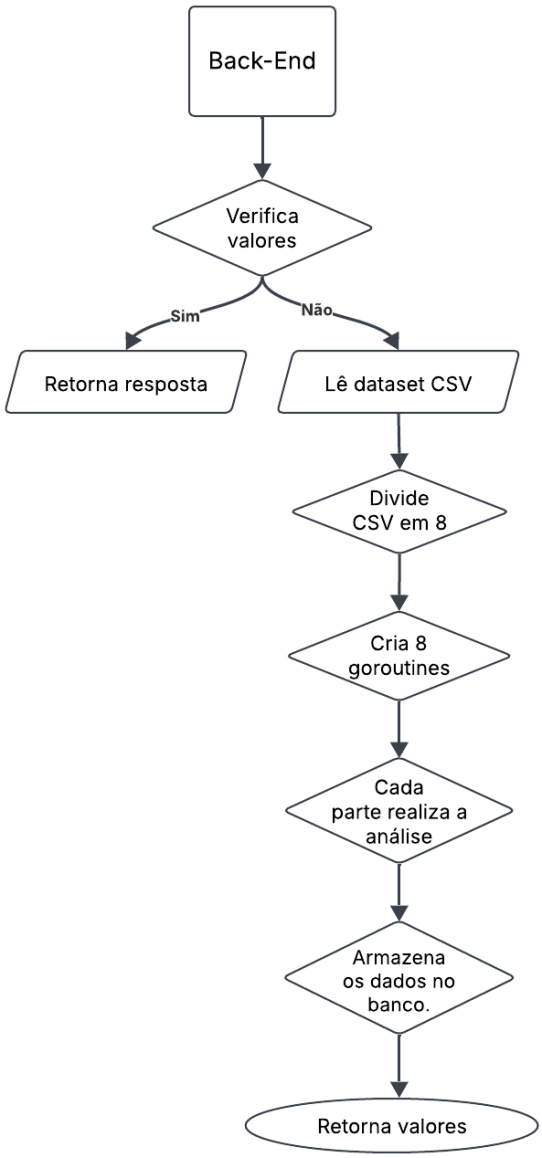


Figura 1 – Fluxo do software

Ao ser iniciada, a aplicação passa pela fase de inicialização, o qual verifica os valores como podemos observar na figura 1, que tem como objetivo principal verificar a necessidade de um novo processamento dos dados. Esta fase é marcada pela consulta ao banco de dados Redis, onde-se busca a existência de uma chave específica realizando verificação no Redis para identificar se a chave predefinida já se encontra registrada. Esta verificação atua como um ponto de controle, indicando se os dados já foram previamente processados e persistidos e caso a chave seja encontrada no Redis, o sistema pode inferir que o processamento dos dados realizou-se em um momento anterior. O comportamento subsequente da aplicação, neste cenário, envolvendo a reutilização dos dados já persistidos.

Caso a ausência da chave no Redis, sinaliza a necessidade de iniciar um novo ciclo de processamento dos dados. Neste caso, a aplicação avança para a próxima fase, iniciando ao fluxo de serviço e criação.

Esta etapa, orquestrada pelo componente service/creation_controller.go (ou componente equivalente), tem como responsabilidade coordenar as etapas subsequentes de análise e processamento.

O componente de controle de criação atua como um ponto central, delegando tarefas específicas a serviços especializados, como exemplificado pela chamada ao serviço create.go que atua para realizar toda manipulação dos dados no contexto do arquivo acidentes. Dentro do serviço create.go, destaca-se a invocação da função AnalyzeAll, pertencente ao pacote acidente. Esta função representa o ponto de entrada para a análise detalhada dos dados de acidentes, configurando o próximo estágio do fluxo.

A fase de análise de dados é um pilar central na solução proposta, responsável por processar e extrair informações significativas dos registros de acidentes de trânsito. O núcleo dessa fase é a função AnalyzeAll, localizada no pacote acidente. Essa função foi projetada para receber parâmetros cruciais que direcionam o processamento, permitindo uma customização flexível da análise conforme a necessidade específica de extração de dados.

Inicialmente, o sistema utilizava uma abordagem de processamento sequencial. Cada tipo de análise (por exemplo, dados por UF, por clima, por velocidade, etc.) era encapsulado em uma função separada (createDataUF, createDataClimate, etc.), e cada uma dessas funções realizava uma chamada independente à função AnalyzeAll. O problema fundamental dessa arquitetura era que cada chamada a AnalyzeAll resultava na reabertura e releitura completa do arquivo CSV de dados de acidentes como podemos constatar na figura 2 como o código era antes e como esta hoje na figura 3.

```go
// exemplo da abordagem anterior do creation_controller.go
    if !Validation(rdb, ctx) {
        createDataUF(rdb, ctx)
        createDataClimate(rdb, ctx)
        createDataHighway(rdb, ctx)
        //...
    }
```

Figura 2 – Antigo método CreateData

```
1  if !Validation(rdb, ctx) {
2          CreateData(rdb, ctx)
3          createDataSusAlcool(rdb, ctx)
4      }
5
```

Figura 3 – Novo método CreateData

A refatoração implementada transformou o método createData em uma função utilitária centralizada, responsável por processar o arquivo CSV de acidentes de forma abrangente. Essa alteração permitiu consolidar as múltiplas operações de leitura e análise que antes eram realizadas separadamente para cada conjunto de dados (ex:. UF, clima, velocidade).

Mesmo com a utilização de goroutines em ambas as abordagens, a diferença fundamental reside na otimização do acesso ao arquivo. Anteriormente, cada tipo de análise disparava uma nova leitura do arquivo CSV, gerando redundância e elevando o tempo de processamento. Com a nova arquitetura, o arquivo é lido e processado uma única vez, e as goroutines trabalham sobre essa única passagem dos dados para extrair todas as informações necessárias.

Como resultado direto dessa refatoração, o tempo de análise dos dados foi reduzido de 20 para 6 segundos, representando uma diminuição de 14 segundos no tempo total de execução. Essa melhoria significativa é visível ao comparar o desempenho da abordagem anterior na tabela 1 da arquitetura sem otimização e com o da arquitetura otimizada .

Ambiente

- Arquitetura: x86_64 (AMD64)
- Processador: AMD Ryzen 7 5700U with Radeon Graphics (8 núcleos / 16 threads)
- Sistema operacional: Ubuntu 24.04.1 LTS (Codinome: noble)
- Go: go1.24.4 linux/amd64

Resultado

Processo	Tamanho do arquivo	Tempo de Execução
Linear	1,5GB	37 segundos
Linear	2,9GB	63 segundos
Linear	5,9GB	125 segundos
arquitetura sem otimização	2,5GB	20 segundos
arquitetura com otimização	2,5GB	6 segundos
arquitetura com otimização	3,9GB	11 segundos
arquitetura com otimização	6,8GB	22 segundos

Tabela 1 – Teste de performance inicial

Para garantir o rigor e a validade científica dos ganhos de performance observados especialmente na comparação entre a arquitetura otimizada e a não otimizada, os testes para o arquivo de 2.5GB foram repetidos 20 vezes para cada cenário, enquanto no caso do cenário linear, com arquivo de 1,5 GB, foi adotada a mesma quantidade de repetições.. A análise de desempenho de sistemas requer rigor estatístico para que os resultados observados possam ser considerados confiáveis e reproduzíveis. Como destaca Jain (1991), a repetição de medições e a utilização de métricas como média, desvio padrão e intervalo de confiança são essenciais para validar diferenças significativas entre experimentos. Essa metodologia permitiu a aplicação de análises estatísticas para determinar a significância dos resultados.

Processo	Média (s)	Desvio Padrão (s)	IC 95% Inferior (s)	IC 95% Superior (s)
Linear 1.5GB	32.53	0.16	32.46	32.61
Arquitetura sem otimização 2.5GB	20.71	0.24	20.60	20.83
Arquitetura com otimização 2.5GB	6.17	0.08	6.13	6.21

Tabela 2 – Teste de performance com análise estatística (20 repetições)

A fase de análise de dados de acidentes é crucial para extrair informações relevantes dos registros, permitindo a compreensão de padrões e a formulação de estratégias preventivas. Antes de detalhar a implementação dessa fase, é importante contextualizar a escolha da concorrência com goroutines em detrimento de uma abordagem linear. Caso queira testar este procedimento apenas comente o código "service.Controller()"escrito na main e descomente o código "sequencial.Acidente("acidentes.csv")".

A análise revelou que, embora goroutines permitam paralelismo, sua utilização indiscriminada pode causar o efeito inverso. Em cenários específicos, o paralelismo mal

planejado gera mais *overhead*[1] do que benefício. Por isso, além de aplicar goroutines, é essencial avaliar:

- A natureza da tarefa (CPU-bound vs. I/O-bound);
- A estrutura de sincronização usada (sync.Mutex, sync.WaitGroup, etc.);
- O volume de goroutines disparadas.

Este comparativo ressalta que a simples utilização de goroutines não garante otimização, a forma como são implementadas é determinante. Muitas vezes, um gargalo de desempenho pode estar no próprio código, e não na ausência de concorrência. No entanto, em cenários com grande volume de dados, como o apresentado, a divisão do processamento via goroutines demonstrou ser altamente eficaz.

Conforme destacado por Majdi (2025), "goroutines não são mágicas, cada goroutine tem um custo imposto" (criação, agendamento, coleta de lixo) e se usadas de forma ingênua, podem até degradar a performance. Por outro lado, a computação paralela, ao permitir que múltiplas operações sejam executadas simultaneamente, é uma estratégia fundamental para alcançar melhorias significativas de desempenho em sistemas modernos, especialmente em tarefas que envolvem processamento intensivo de dados Donovan e Kernighan (2015). Estudos têm investigado a eficiência das goroutines em cenários de paralelismo. Por exemplo, em uma análise de implementação paralela de um algoritmo de programação dinâmica com grafos de tarefas acíclicas diretas, observou-se que as goroutines, apesar de seus custos inerentes de agendamento e sincronização, são eficazes para a avaliação de tais grafos, demonstrando seu potencial para ganhos de performance em certas condições como demonstra Serfass e Tang (2012) em seu estudo *Comparing parallel performance of Go and C++ TBB on a direct acyclic task graph using a dynamic programming problem.* Testes de desempenho em cenários de alto volume de dados mostram que, quando o processamento é dividido corretamente em goroutines bem configuradas, os ganhos são expressivos como mostrado em estudos como o do Ganjtabesh (2024) em *Speeding Up Go Concurrency: Unlocking the Power of Array Segmentation* (Acelerando a simultaneidade em Go: revelando o poder da segmentação de arrays), chegam a obter multiplicadores de 5x a 40x em tarefas intensivas.

Neste contexto de otimização, a função AnalyzeAll inicia realizando a leitura do cabeçalho do arquivo CSV para identificar e mapear os índices de todas as colunas de interesse como (uf_acidente, cond_meteorologica, qtde_obitos, etc.). Esse mapeamento inicial é crucial para garantir que os dados corretos sejam acessados durante o processamento. Um mecanismo de validação verifica a presença das colunas críticas, caso alguma

[1] Sobrecarga

coluna essencial não seja encontrada, o processo é interrompido com o devido registro de erro, garantindo a integridade da análise.

Após a validação do cabeçalho, a estratégia de processamento paralelo é ativada. A função AnalyzeAll determina o tamanho total do arquivo CSV e o divide logicamente em um número predefinido de partes (atualmente, 8 partes, conforme definido pela constante numParts). Para cada uma dessas partes, uma goroutine é lançada, "compartilhando recursos e melhorando a eficiência do sistema" Ribeiro (2025).

Cada goroutine recebe um segmento específico do arquivo (definido por um startOffset e endOffset) e é responsável por processar as linhas contidas naquele segmento através da função auxiliar ProcessFilePartAll. Esta função lê o segmento atribuído, extrai os dados relevantes de todas as colunas mapeadas e agrega os resultados parciais em uma estrutura sync.Map. O uso de sync.Map é essencial para garantir a segurança de acesso e gravação dos resultados parciais por múltiplas goroutines concorrentemente, evitando condições de corrida.

Ao final do processamento de todas as partes, a função AnalyzeAll utiliza um sync.WaitGroup para aguardar a conclusão de todas as goroutines lançadas. Posteriormente, os resultados parciais coletados de cada goroutine são agregados e mesclados em uma estrutura AnalysisResult unificada através da função mergeYearData. Essa agregação final consolida os dados de todas as análises por (UF, clima, velocidade, etc.) que foram processadas em paralelo.

Essa abordagem otimizada, que divide o arquivo e processa todas as análises utilizando goroutine, representa um avanço significativo em relação a métodos sequenciais de leitura ou múltiplas leituras de arquivo. Ela explora a concorrência de forma eficiente para lidar com arquivos de grande porte, reduzindo drasticamente o tempo total necessário para a fase de análise de dados de acidentes de trânsito.

A sincronização e o controle do término das goroutines são gerenciados utilizando sync.WaitGroup (wg), Este mecanismo espera que todas as goroutines sejam concluídas antes de encerrar o programa segundo Donovan e Kernighan (2015) o WaitGroup fornece uma maneira segura e eficiente de coordenar múltiplas execuções concorrentes, sendo essencial para o controle de fluxo em aplicações concorrentes escritas em Go.

Cada goroutine executa a função processFilePart, responsável pelo processamento individual de uma porção do arquivo CSV como na imagem 4.

```
for i := 0; i < numParts; i++ {
        startOffset := int64(i) * partSize
        endOffset := startOffset + partSize
        if i == numParts-1 {
            endOffset = fileSize
        }

        filterValue := "MOTORISTA"
        filterColumn := 9

        wg.Add(1)
        go processFilePart(filePath, startOffset, endOffset, idxColumn,
            dateColumnIndex, amountDeathColumn, amountInvolvedColumn,
            amountInjuredColumn, filterColumn, filterValue, &wg, &counts)
    }
```

Figura 4 – Método processFilePart

E no caso de acidentes que realiza todos em uma chamada conseguimos identificar como é na figura 5.

```go
for i := 0; i < numParts; i++ {
    startOffset := int64(i) * partSize
    endOffset := startOffset + partSize
    if i == numParts-1 {
        endOffset = fileSize
    }

    wg.Add(1)
    go ProcessFilePartAll(
        filePath,
        startOffset,
        endOffset,
        columns,
        idxUF,
        idxDate,
        idxClimate,
        idxSpeed,
        idxTrack,
        // ...
    )
}
```

Figura 5 – Método processFilePartAll

Cada goroutine, dentro de processFilePart, realiza a leitura sequencial da sua porção designada do arquivo CSV, linha por linha, utilizando bufio.Reader. Caso um filtro tenha sido configurado, cada linha lida é avaliada. A linha é processada apenas se o valor na coluna de filtro (filterColumn) corresponder ao valor de filtro desejado (filterValue) e para cada linha que passa pela etapa de filtro (ou na ausência de filtro), os seguintes dados são extraídos e validados. A extração da data da coluna dateColumnIndex, com validação para garantir que o ano se inicia com "20" (indicando anos a partir de 2000) e as linhas com datas inválidas são ignoradas e a extração do valor da coluna especificada por idxColumn. A uma extração dos valores das colunas amountDeathColumn, amountInvolvedColumn e amountInjuredColumn, com conversão para tipo inteiro e tratamento de erros de conversão (valores padrão 0 em caso de falha).

A agregação dos dados extraídos é realizada de forma concorrente utilizando estruturas de dados e mecanismos de sincronização e os dados são organizados hierarquicamente utilizando as estruturas YearData (para agrupar dados por ano) e AccidentData (para agregar dados por tipo de análise, como "susp_alcool"). Os resultados de todas as gorotinas são consolidados em um mapa concorrente (counts do tipo sync.Map), que permite acesso e modificações seguras por múltiplas goroutines, para cada linha processada, os totais de acidentes, óbitos, envolvidos e feridos são agregados. Dentro de cada YearData, no objeto AccidentData referente ao valor da coluna de análise (columnName). Para proteger o acesso concorrente ao mapa TotalAcciden dentro de cada objeto YearData, um mutex (yearData.mu) é utilizado, garantindo a integridade dos dados durante a agregação paralela.

Na fase de retorno de resultados após a conclusão do processamento paralelo do arquivo CSV, orquestrado pela função processFile, a função AnalyzeAccidentData finaliza sua execução e a função AnalyzeAccidentData retorna a estrutura de dados counts (do tipo sync.Map<string, *YearData>). Esta estrutura encapsula os resultados agregados da análise, organizados por ano e pela coluna de análise especificada, contendo os totais de acidentes, óbitos, envolvidos e feridos para cada categoria analisada.

3.2 Ferramentas e tecnologias utilizadas

A linguagem de programação Go foi escolhida para o desenvolvimento da API (*Application Programming Interface*)[2] e o método de análise de dados do sistema devido à sua alta performance e escalabilidade. Conforme discutido por Donovan e Kernighan (2015), a simplicidade da linguagem Go e sua robusta biblioteca padrão tornam o desenvolvimento eficiente e confiável, ao mesmo tempo que permitem a execução simultânea de múltiplas tarefas com baixo consumo de recursos. Tais características são fundamentais para o processamento em tempo real de grandes volumes de dados, como os dados de acidentes de trânsito utilizados neste sistema.

O Redis, configurado para persistência em disco, exerceu como solução de banco de dados para atender às demandas de armazenamento e recuperação de dados.

Para a camada de apresentação do sistema, Next.js destacou-se como escolha para construir uma interface moderna. Sua capacidade de realizar renderização no lado do servidor e cliente (*server-side rendering e client-side rendering*)[3] permite que as visualizações de dados, como mapas e relatórios de acidentes de trânsito, sejam carregadas rapidamente, mesmo em conexões de internet mais lentas, conforme Razvan (2023), essas técnicas de renderização são fundamentais para otimizar o desempenho e a escalabilidade

[2] Traduzido como interface de programação de aplicações
[3] Renderização do lado do servidor e renderização do lado do cliente

de aplicações web modernas.

Para a construção de interfaces dinâmicas e interativas, como mapas e gráficos de análise, adotou-se o React. Esta biblioteca JavaScript facilita a criação de interfaces de usuário interativas e dinâmicas por meio de componentes reutilizáveis evidenciado por Banks e Porcello (2023) em seu livro *Learning React: Modern Patterns for Developing React*. A abordagem baseada em componentes do React permite dividir a interface do usuário em partes menores e independentes, facilitando o desenvolvimento, a manutenção do código. Além disso, o React utiliza o virtual DOM para otimizar a atualização e renderização eficiente dos componentes conforme os dados mudam, proporcionando uma experiência intuitiva para os usuários finais.

Além disso, para facilitar o processo de implantação e garantir a portabilidade entre diferentes ambientes, utilizou-se o Docker. Essa plataforma de virtualização baseada em containers permite empacotar a aplicação e todas as suas dependências em uma imagem padronizada, assegurando que o sistema funcione de forma consistente em qualquer ambiente, conforme explicado por (GOMES, 2020) em seu livro *Docker para Desenvolvedores*. O uso do Docker contribui para a automação do deploy, o isolamento de ambientes do projeto, beneficiando tanto o desenvolvimento quanto a manutenção do sistema.

3.3 Requisitos

1. Análise de plataformas concorrentes e referências

   O processo de levantamento de requisitos iniciou-se com uma análise de plataformas existentes que abordam, a visualização e o acesso a informações de trânsito. A escolha das plataformas para análise baseou-se na sua relevância para o contexto do presente projeto, buscando identificar tanto boas práticas quanto lacunas que poderiam ser exploradas para a construção de uma solução inovadora e mais abrangente. Nesse sentido, foram selecionadas para análise as seguintes plataformas:

   - Atlas Volvo: O grupo Volvo América Latina criou, em 2014, o Atlas da Acidentalidade no Transporte Brasileiro, o mais completo material sobre acidentes rodoviários já publicado no Brasil segundo o trabalho publicado no site Aliens Design. Esta plataforma apresenta uma interface visualmente atraente e funcionalidades de mapa interativo para visualização de dados relacionados a rodovias federais. Durante a análise, buscou-se compreender a ideia de visualização do site proposto em gráficos a estrutura de apresentação das informações em mapas e tabelas, a forma como os dados eram filtrados e a experiência do usuário ao navegar pela plataforma Design (2025).
   - DETRAN-RO (Departamento Estadual de Trânsito de Rondônia): O site

DETRAN-RO possui uma plataforma de Power BI, proporcionando diferentes formas de busca para identificar diferentes pontos sobre dados que são extraídos dos acidentes no estado de Rondônia. A seleção desta plataforma justificou-se por trazer dados interativos, com mapa, além de ser um portal de informações de trânsito de um órgão governamental, apresentando dados relevantes para o contexto de informações sobre veículos e condutores. A análise concentrou-se na organização da informação em menus e categorias e na facilidade de acesso aos dados oferecidos DETRAN-RO (2025).

A análise de ambas as plataformas realizou-se mediante navegação extensiva em suas funcionalidades, testes práticos de interação com os elementos visuais e anotações sobre os pontos positivos, negativos e ideias inspiradoras observadas em cada uma delas.

Embora ambas as plataformas ofereçam funcionalidades relevantes para seus contextos, estudos indicam que ferramentas públicas voltadas à visualização de dados, em geral, ainda enfrentam limitações significativas de usabilidade, acessibilidade e integração de dados. Segundo Valiati (2008) em sua tese *Avaliação de usabilidade de tecnicas de visualização de informações multidimensionais*, a análise dessas interfaces demanda a definição de uma taxonomia de tarefas específica, bem como a adaptação de métodos tradicionais de usabilidade ao contexto visual e interativo, devido à complexidade envolvida. Reforçando essa perspectiva, o seminário promovido pelo IPEA (2025), intitulado *Seminário discute desafios para a consolidação do Estado digital no Brasil*, destaca a fragmentação e a falta de interoperabilidade entre bases públicas como um dos principais entraves à análise integrada em nível nacional. Complementando essa discussão, o Portal do Trânsito (2024), na reportagem *O trânsito brasileiro precisa de estatísticas confiáveis*, evidencia a ausência de transparência e a indisponibilidade de dados públicos consistentes sobre acidentes de trânsito no país.

Tais evidências reforçam a percepção, também observada empiricamente durante a análise prática das plataformas, de que há espaço para o desenvolvimento de uma solução mais coesa, intuitiva e unificada para a visualização de dados de trânsito em nível nacional.

2. Identificação da necessidade e definição do escopo

   Após a análise das plataformas referenciadas, tornou-se evidente a necessidade de uma solução que suprisse a lacuna existente na centralização e visualização de informações relevantes em âmbito nacional. Observou-se que, enquanto plataformas como o Atlas Volvo e o DETRAN-RO oferecem ferramentas valiosas em seus respectivos nichos de atuação (rodovias federais e informações estaduais de trânsito, respectivamente), não foi identificada, até o momento desta análise uma plataforma que agregue e apresente dados de trânsito de todo o território brasileiro.

Especificamente, a análise revelou que o Atlas Volvo, apesar da excelência na visualização de dados rodoviários, restringe-se a informações federais, e o DETRAN-RO, por sua vez, limita-se ao estado de Rondônia. Essa constatação motivou a definição do escopo do presente projeto, que visa desenvolver uma plataforma que se diferencie por oferecer uma visão unificada de dados de trânsito em escala nacional.

O escopo da plataforma a ser desenvolvida consiste em uma plataforma web interativa focada na visualização de dados de trânsito em âmbito federal do Brasil. A plataforma permite a visualização através de mapas interativos e tabelas comparativas, priorizando a facilidade de uso. Funcionalidades como análise estatística avançada ou download massivo de dados não serão implementadas nesta versão inicial, mantendo o foco na visualização e na exploração interativa das informações.

3.4 Arquitetura do software

A solução desenvolvida adota uma arquitetura distribuída, onde o sistema é dividido em componentes independentes que se comunicam entre si para cumprir suas funções. Neste caso, a aplicação consiste em uma API back-end, um banco de dados Redis e uma interface web front-end que são sistemas separados, comunicando-se via chamadas HTTP.

Antes da inicialização da API, é executado um método interno responsável pelo processamento dos dados brutos e pela alimentação do banco de dados, caso este esteja vazio. Este processamento é feito de forma integrada à inicialização da API.

Segundo George (2013, p. 2), "um sistema distribuído é aquele em que componentes de hardware ou software localizados em computadores interligados em rede se comunicam e coordenam suas ações apenas trocando mensagens entre si". Essa definição reforça que, apesar do método de processamento ser parte da API, a separação entre back-end e front-end em sistemas distintos caracteriza a arquitetura distribuída. A seguir, são apresentados os principais componentes e seu funcionamento:

1. API (Go)
    - Responsável por servir os dados para a aplicação web.
    - Verifica a existência dos dados no Redis ao iniciar.
    - Caso os dados não estejam disponíveis, aciona o método de processamento.
2. Método de processamento (Go)
    - Lê e analisa arquivos CSV para extrair informações.
    - Insere os dados processados no Redis para otimizar futuras consultas.
    - Após o processamento, retorna para a API.

3. Banco de dados (Redis com persistência em disco)
    - Armazena as informações processadas, permitindo buscas no futuro.
4. Interface Web (Next.js e React)
    - Exibe as informações em gráficos interativos.
    - Permite filtros e análises sobre os dados processados.

Conforme podemos visualizar na figura 6.

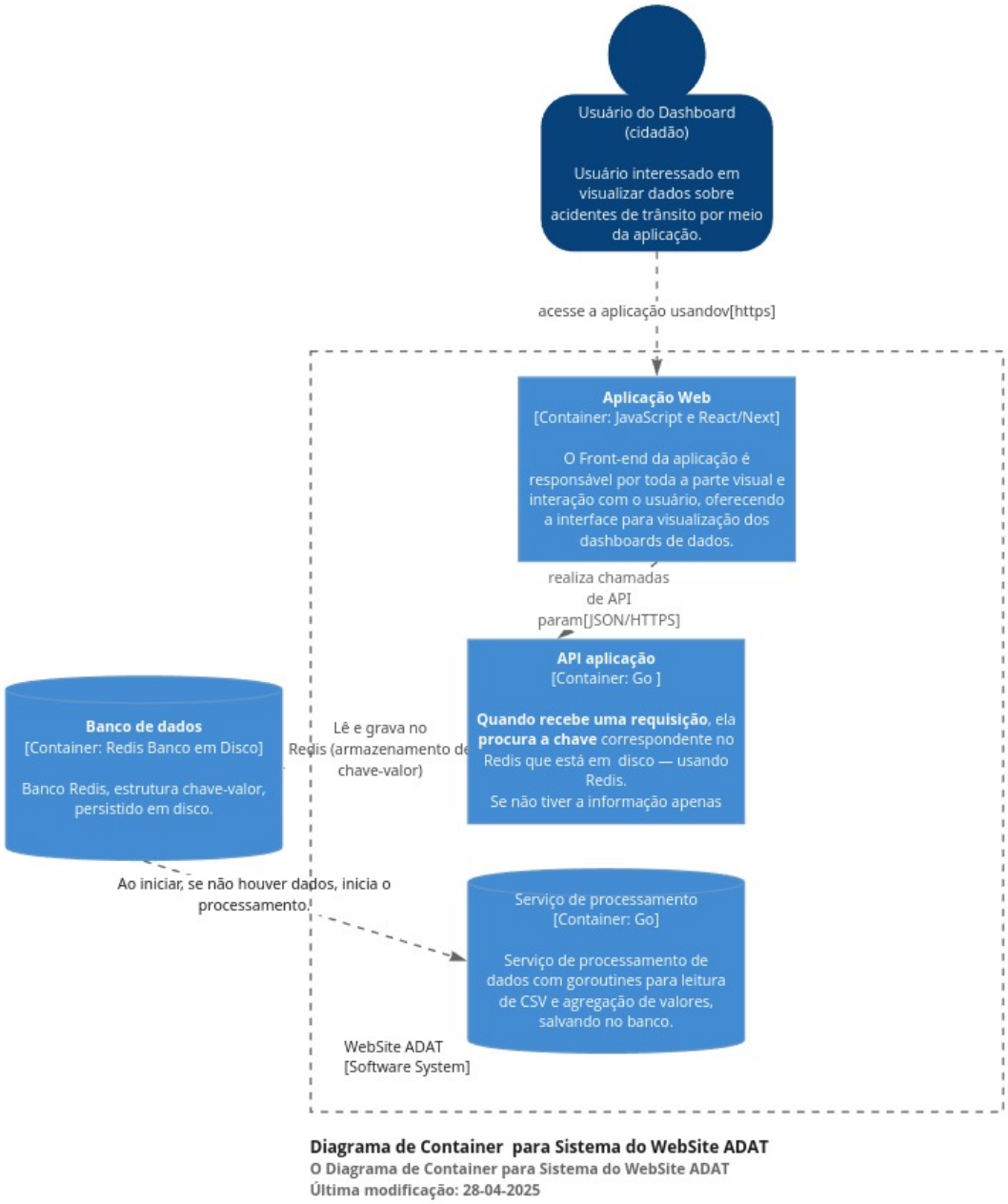


Figura 6 – Arquitetura do Software

3.5 Persistência de dados

Para garantir a precisão dos dados processados, a abordagem adotada envolve validação manual, comparando as informações geradas pela aplicação com uma tabela de referência. Os dados são armazenados no Redis seguindo uma estrutura de chave-valor, onde as chaves possuem um padrão específico para organização e recuperação eficiente. O formato das chaves segue a estrutura "data_<tipo_de_dado>_<ano>". Por exemplo:

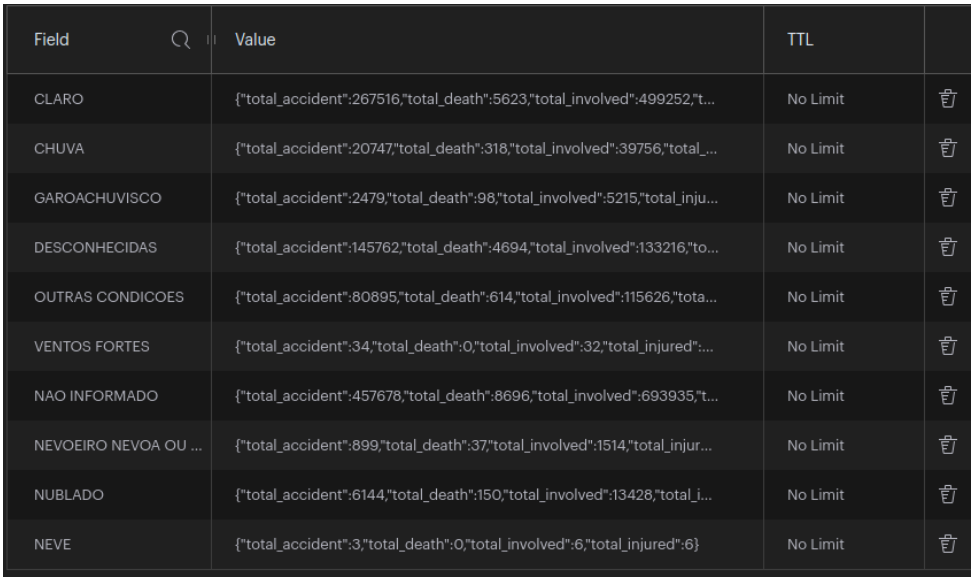
- data_climate_2023 → Contém dados climáticos do ano de 2023

Cada chave armazena informações detalhadas, incluindo, por exemplo o campo "CHUVA":

- total_accident: Número total de acidentes.
- total_injured: Número total de feridos.
- total_death: Número total de óbitos.
- total_involved: Número total de pessoas envolvidas.

Conforme podemos visualizar as chaves na figura 7 e os valores na figura 8.

Figura 7 – Valores de chave em Redis



Field	Value	TTL	
CLARO	{"total_accident":267516,"total_death":5623,"total_involved":499252,"t...	No Limit	
CHUVA	{"total_accident":20747,"total_death":318,"total_involved":39756,"total_...	No Limit	
GAROACHUVISCO	{"total_accident":2479,"total_death":98,"total_involved":5215,"total_inju...	No Limit	
DESCONHECIDAS	{"total_accident":145762,"total_death":4694,"total_involved":133216,"to...	No Limit	
OUTRAS CONDICOES	{"total_accident":80895,"total_death":614,"total_involved":115626,"tota...	No Limit	
VENTOS FORTES	{"total_accident":34,"total_death":0,"total_involved":32,"total_injured":...	No Limit	
NAO INFORMADO	{"total_accident":457678,"total_death":8696,"total_involved":693935,"t...	No Limit	
NEVOEIRO NEVOA OU ...	{"total_accident":899,"total_death":37,"total_involved":1514,"total_injur...	No Limit	
NUBLADO	{"total_accident":6144,"total_death":150,"total_involved":13428,"total_i...	No Limit	
NEVE	{"total_accident":3,"total_death":0,"total_involved":6,"total_injured":6}	No Limit	

Figura 8 – Valores armazenados

3.6 Plano de testes

O plano de teste desenvolvido para este projeto tem como objetivo garantir que todas as funcionalidades implementadas operem conforme os requisitos estabelecidos, validando tanto a integridade das respostas das rotas da API quanto a consistência dos dados processados e armazenados no Redis.

Os testes serão estruturados seguindo as práticas recomendadas para projetos desenvolvidos na linguagem Go, com os arquivos de teste organizados dentro dos pacotes correspondentes às funcionalidades avaliadas. No diretório accident, por exemplo, será incluído um arquivo de teste específico para validar a análise dos dados de acidentes. Adicionalmente, serão implementados testes para o serviço de integração com o Redis, a fim de assegurar a correta persistência e recuperação dos dados.

Para testar os endpoints da API, serão realizados testes de integração utilizando a biblioteca httptest, a qual permite simular requisições HTTP e validar as respostas conforme os requisitos funcionais. Endpoints como /uf, /climate e /guardrail serão avaliados quanto ao status HTTP retornado, à estrutura dos dados e à consistência das informações em relação aos registros armazenados no Redis. Caso sejam identificadas divergências entre os resultados obtidos e os valores esperados, a implementação da rota será revisada para correção de eventuais falhas.

A integridade do processamento dos dados será validada por meio de testes unitários, como no caso da função AnalyzeAccidentData. Esses testes verificarão se a função é capaz de processar corretamente os arquivos CSV de entrada, agregando métricas como o total de acidentes, óbitos e feridos por categoria (por exemplo, presença ou ausência de guardrail).

Os valores esperados para o ano de 2022 servirão como referência para comparação, a fim de garantir a precisão da lógica de análise implementada.

A persistência de dados no Redis será validada por meio de testes isolados. Antes de cada execução, a chave referente ao ano analisado será removida, evitando interferências de execuções anteriores. Em seguida, novos valores serão inseridos e será verificada a existência da chave, bem como a correta estruturação dos dados armazenados.

3.7 Licença de uso

A solução desenvolvida neste trabalho está licenciada sob a Licença MIT *(Massachusetts Institute of Technology License)*. Essa licença é amplamente utilizada em projetos de software de código aberto, permitindo que qualquer pessoa utilize, copie, modifique e distribua o software, desde que seja mantida a atribuição original ao autor.

Conforme publicado por Francileudo (2025) em seu trabalho "Licença MIT no Github - O que é isso?", a escolha de uma licença adequada permite que um software seja modificado, copiado e redistribuído sem implicações jurídicas para seus usuários e desenvolvedores. Nesse sentido, a Licença MIT se destaca por sua simplicidade e flexibilidade, garantindo liberdade de uso sem impor restrições excessivas.

O texto completo da Licença MIT encontra-se no Anexo A deste trabalho.

4 Resultados e discussões

Neste capítulo são apresentados os resultados e discussões acerca da aplicação proposta.

4.1 Gerenciamento de configuração e mudanças

O gerenciamento de configuração e mudanças nos repositórios adat e processamento, ocorreu utilizando o sistema de controle de versão GIT e armazenado no GitHub, conforme (CHACóN; STRAUB, 2014) explicam, o GitHub é uma plataforma baseada em nuvem que fornece um ambiente colaborativo para repositórios GIT, permitindo controle de versão distribuído e integração contínua. Ambos os projetos mantiveram o código principal na branch main, com o desenvolvimento de novas tarefas sendo realizado em uma branch separada chamada (dev). A integração das tarefas era feita através de pull requests para a branch main, garantindo a revisão do código antes da incorporação.

A organização dos commits seguiu um padrão de mensagens descritivas, facilitando o acompanhamento da evolução dos projetos. Apesar de não ter sido utilizada uma metodologia formal de gerenciamento de projetos, como Kanban, o acompanhamento das atividades ocorreu de forma individual, considerando as áreas de maior necessidade em cada etapa do desenvolvimento.

4.2 Processo de desenvolvimento

O desenvolvimento da solução seguiu uma abordagem iterativa, sem a adoção de uma metodologia formal como Scrum ou Kanban. No entanto, o fluxo de trabalho foi organizado de maneira estruturada, priorizando as funcionalidades conforme a necessidade e complexidade de implementação.

4.3 Relatório dos testes

Os arquivos de teste foram organizados dentro dos respectivos pacotes Go. O ambiente de teste foi configurado removendo chaves do Redis antes da execução para evitar interferências entre testes consecutivos e mocando dados caso necéssario em cada respectivo teste.

4.3.1 Teste de integração pacote de router

O teste verifica se as rotas registradas no pacote routes estão funcionando corretamente. Podemos encontrar esse teste no pacote "routes" no arquivo "test_routes". Utiliza-se o framework Chi para criar um roteador HTTP e testa as rotas configuradas no método RegisterRoutes. O teste simula requisições HTTP para cada rota e valida se o status da resposta corresponde ao esperado (http.StatusOK).

Os seguintes casos de teste foram definidos:

Método HTTP	Rota	Status esperado
GET	/uf	OK
GET	/climate	OK
GET	/guardrail	OK
GET	/highway	OK
GET	/median	OK
GET	/shoulder	OK
GET	/speed	OK
GET	/day_week	OK
GET	/phase_day	OK
GET	/month	OK

Tabela 3 – Rotas testadas e seus status esperados

Todas as rotas retornaram o status HTTP 200 OK, conforme esperado. Nenhum erro encontrado durante a execução do teste.

4.3.2 Teste de integração pacote service

O teste do pacote service avalia a função Validation, que é crucial para o fluxo de processamento de dados. Esta função é responsável por verificar a existência e a consistência de uma chave específica (data_uf_2021) no banco de dados Redis. A correta operação de Validation é fundamental para garantir que os dados necessários estejam disponíveis antes que qualquer processamento subsequente ocorra. Se esta função falhar, há o risco de o sistema reprocessar informações desnecessariamente ou, pior, não executar um processamento essencial quando necessário.

CENÁRIO	ENTRADA	SAÍDA ESPERADA
Chave não existe	Validation(rdb, ctx)	false
Chave existe com dados válidos	Validation(rdb, ctx)	true

Tabela 4 – Casos de teste para a função Validation no pacote service, verificando a existência e consistência da chave data_uf_2021 no Redis.

O teste da função Validation no pacote service é projetado para verificar a presença e a consistência da chave data_uf_2021 no Redis. Para garantir um ambiente de teste

isolado, a chave é sempre removida do Redis antes de cada execução, assegurando um estado limpo. Inicialmente, o teste chama Validation com a chave inexistente, esperando e confirmando o retorno false. Em seguida, para o cenário de chave existente, dados arbitrários são inseridos diretamente na data_uf_2021, simulando sua presença sem a necessidade de reprocessar o arquivo CSV. Uma nova chamada à Validation então retorna true, validando que a função detecta corretamente a chave. Por fim, a chave data_uf_2021 é novamente removida, restaurando o ambiente do Redis.

4.3.3 Teste unitario pacote config

O objetivo deste teste unitário é verificar se as variáveis de ambiente estão sendo carregadas corretamente e se os valores padrão são aplicados quando as variáveis de ambiente não estão definidas. Além disso, o teste valida a inicialização do cliente Redis com base nas configurações, sem realizar operações reais no banco de dados.

Os testes foram implementados para garantir que a lógica interna do código funcione corretamente de forma isolada, sem dependências externas. Para isso, foram utilizados mocks e stubs para simular o comportamento das dependências, garantindo que o código possa ser testado sem necessidade de conexão real com o Redis.

Todos os testes foram executados conforme esperado, garantindo que:

- As variáveis de ambiente são carregadas corretamente.
- Os valores padrão são aplicados quando necessário.
- O cliente Redis é inicializado corretamente sem conexões reais.

4.3.4 Teste funcional pacote accident

O teste funcional verifica se uma função ou módulo produz os resultados esperados com base em entradas específicas. Ele foca na saída final do sistema, sem se preocupar com os detalhes internos da implementação. Esse tipo de teste é frequentemente usado para validar a correção dos dados retornados por uma função.

E utilizando o teste TestAccident como exemplo, ele valida se a função AnalyzeAll retorna os valores corretos para cada conjunto de dados em um determinado ano (2022). O teste compara os resultados reais com os valores esperados, garantindo que a lógica de processamento de dados esteja funcionando conforme o planejado. Caso os dados retornados seja o mesmo que estão presente no map o teste é finalizado com sucesso.

Todos os documentos gravados no banco são testados, testo apenas o ano de 2022 afim de facilitar a testagem, se o ano de 2022 estiver retornando sucesso leva em conta que os demais anos também estão sendo gravado corretamente.

CHAVE	RETORNO
data_speed_2022	OK
data_shoulder_2022	OK
data_phase_day_2022	OK
data_month_2022	OK
data_guardrail_2022	OK
data_highway_2022	OK
data_climate_2022	OK
data_day_week_2022	OK
data_median_2022	OK
data_track_condition_2022	OK

Tabela 5 – Retornos esperados das chaves validadas no teste funcional TestAccident do pacote accident.

4.4 Documentação

Devido à simplicidade da aplicação e seu foco principal na apresentação intuitiva dos dados de acidentes de trânsito, a documentação da API foi construída de forma objetiva e direta. Todas as rotas utilizam exclusivamente o método HTTP GET, permitindo ao usuário ou sistema consumidor apenas realizar consultas (leitura) dos dados disponíveis.

As rotas seguem uma estrutura padronizada, variando basicamente apenas pelo nome do endpoint e pelos parâmetros de consulta. Assim, o funcionamento da API é altamente intuitivo, ao acessar a URL raiz onde o back-end está sendo executado, é possível visualizar todas as rotas disponíveis e instruções sobre como utilizá-las.

A documentação completa está disponível na própria aplicação, acessível por meio do Swagger "/swagger/index.html".

exemplo de rota:

- /climate?data=dados_climate_2022

Essa rota retorna os dados de acidentes por condições climáticas no ano de 2022.

Esse padrão de organização facilita o acesso direto pelo navegador e também a integração com o front-end e outras ferramentas que consumam a API.

4.4.1 Documentação para usuários finais

Dada a simplicidade e a natureza intuitiva da aplicação, não há uma documentação formal destinada ao usuário final. A interface teve seu design elaborado para ser clara e direta, permitindo que o usuário acesse facilmente as informações de acidentes de trânsito sem a necessidade de guias ou tutoriais extensivos.

A aplicação não exige configurações complexas ou qualquer tipo de customização avançada. O fluxo de navegação é simples, com as funcionalidades principais claramente acessíveis. Caso necessário, pequenos elementos de orientação visual estão presentes na interface, ajudando o usuário a compreender rapidamente como interagir com os dados apresentados.

Dessa forma, a experiência do usuário é proporcionada diretamente pela interface, que dispensa a necessidade de uma documentação extensa.

4.4.2 Documentação para desenvolvedores

A documentação para desenvolvedores seguiu uma abordagem simples e objetiva. Um arquivo README.md criado no repositório do projeto, contendo informações essenciais sobre a configuração e o funcionamento da aplicação, como:

- Objetivo da aplicação.
- Passos para instalação e execução.
- Dependências necessárias.
- Descrição dos principais componentes do código.

Além disso, o código-fonte é documentado com comentários explicativos, de modo que um desenvolvedor consiga entender a lógica e fazer modificações ou melhorias conforme necessário.

Para mais detalhes, o repositório completo do projeto está disponível no GitHub.

4.5 Implantação

O processo de implantação da solução foi realizado na VM *(Virtual Machine)* do IFRO (Instituto Federal de Rondônia). A aplicação está disponível para acesso através do seguinte link: https://adat.app.fslab.dev/.

4.6 Demonstração do software

Este trabalho apresenta o desenvolvimento de um website para visualização de dados. A interface do usuário foi projetada com foco na simplicidade e facilidade de uso. A página inicial é composta por um cabeçalho com o título do website e um menu que permite ao usuário selecionar diferentes filtros de visualização, como UF (Unidade Federativa) para informações do estado e por exemplo alterar para tempo/clima. Abaixo do menu,

encontra-se a área de filtragem, onde o usuário pode refinar sua busca por informações específicas conforme podemos visualizar na figura 9. Essa lógica de filtragem é consistente em todas as páginas do website, garantindo uma experiência de usuário consistente e previsível.

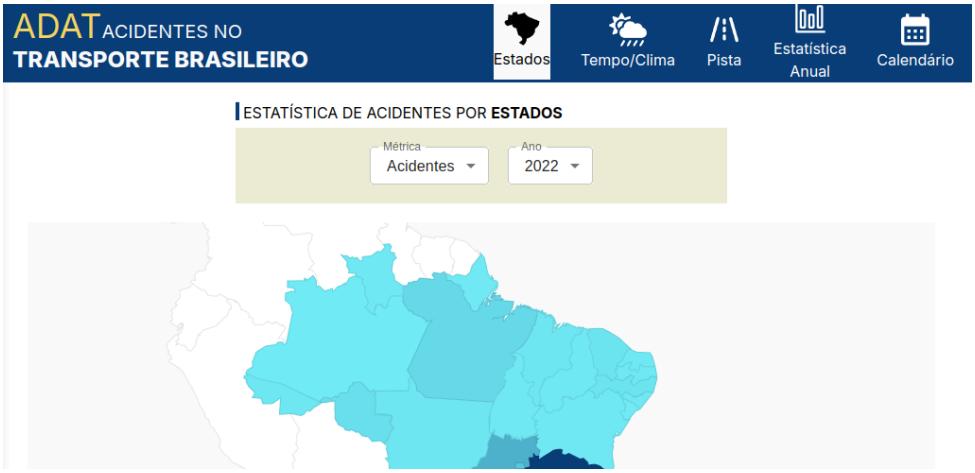


Figura 9 – Filtragem de unidade federativa

Nesta página de unidade federativa as informações são apresentadas em um mapa do Brasil, onde cada estado é colorido de acordo com a quantidade da informação registrada. Tons mais escuros representam quantidades maiores, enquanto tons mais claros indicam quantidades menores, permitindo ao usuário identificar os estados com maior e menor incidência desses dados filtrados.

Ao passar o mouse sobre um estado específico, um pop-up exibe informações detalhadas sobre aquele estado, como a quantidade de informação registrada no ano em questão 10.

A combinação do mapa interativo com os pop-ups informativos oferece uma forma de explorar os dados, permitindo ao usuário obter uma visão geral da situação em todo o país e, ao mesmo tempo, acessar informações detalhadas sobre estados individuais.

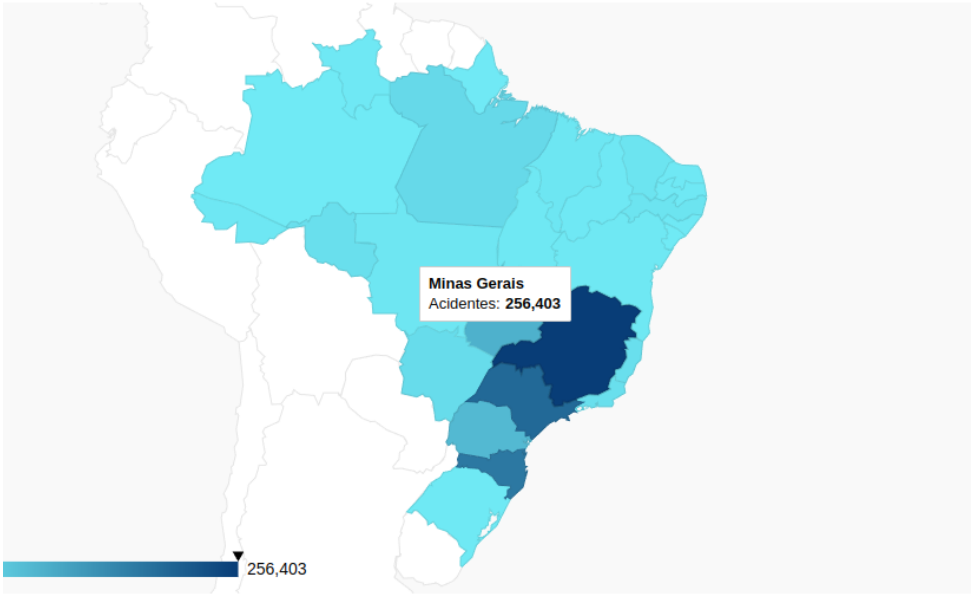


Figura 10 – Mapa unidade federativa

Na página dedicada ao tempo e clima, é apresentada uma tabela detalhada que permite filtrar informações por ano. Essa tabela inclui dados sobre as condições climáticas registradas durante os eventos, como céu limpo, neve, ou situações em que essas informações estão indisponíveis ou não foram informadas. Esses dados são fundamentais para analisar a relação entre as condições climáticas e os impactos observados, como o número de envolvidos, óbitos e acidentes ocorridos em cada condição específica. A visualização dessas informações pode ser conferida nas figuras 11 e 12.

Figura 11 – Filtragem de tempo e clima

Garoa/Chuvisco	2479	5215	5117	98
Chuva	20747	39756	39438	318
Nevoeiro/névoa/fumaça	899	1514	1477	37
Nublado	6144	13428	13278	150
Neve	3	6	6	0
Granizo	0	0	0	0
Desconhecidas	145762	133216	128522	4694
Não informado	457678	693935	685239	8696
Outras condições	80895	115626	115012	614

Figura 12 – Interface de visualização de tempo e clima

Em relação à Pista, o sistema disponibiliza duas opções de filtragem, conforme ilustrado nas Figuras 13 e 14. Os filtros permitem a escolha de dados tanto por ano quanto por métrica, com ênfase em acidentes e envolvidos. As informações são visualizadas em gráficos de linha e pizza, conforme demonstrado na figura 15.

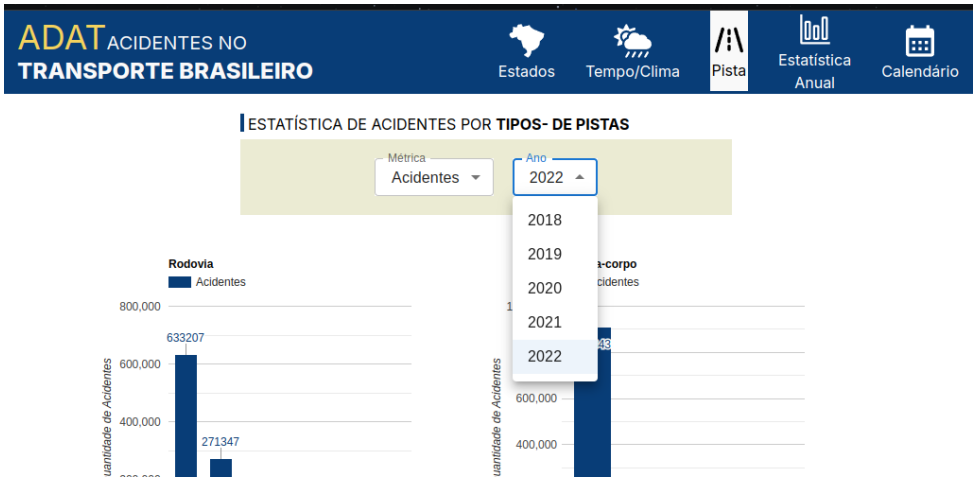


Figura 13 – Filtragem de pista ano

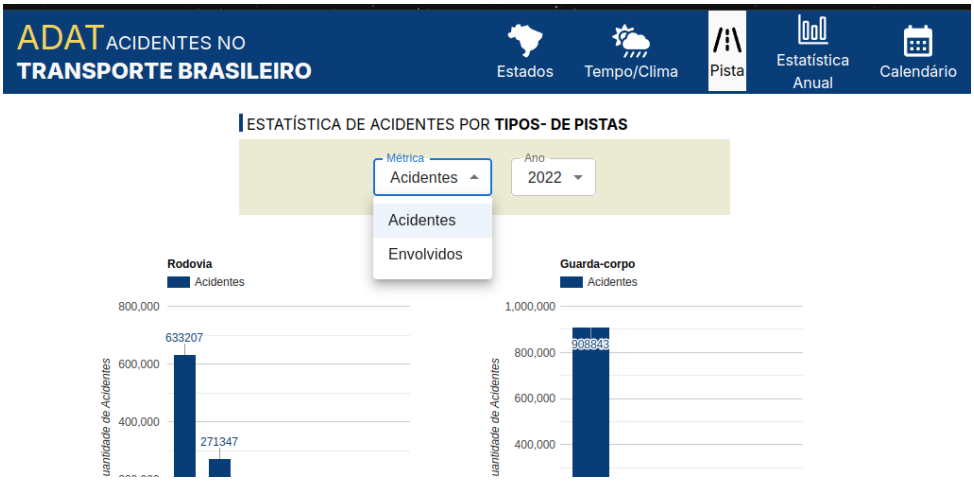


Figura 14 – Filtragem de pista métrica

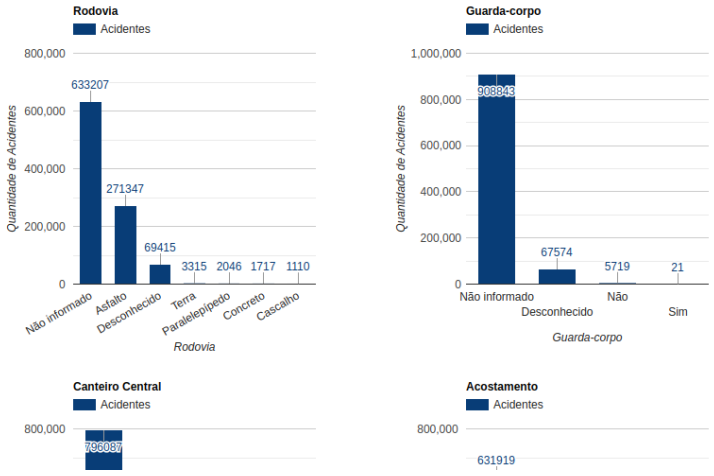


Figura 15 – Interface gráfica de dados pista

Na página estatística anual, são apresentadas informações como suspeita de álcool, tipo de pista, clima e unidade federativa. Esses dados abrangem todos os anos disponíveis no banco, como, por exemplo, de 2019 a 2024, permitindo uma análise histórica dos indicadores. Além disso, os dados são exibidos em gráficos de linha, possibilitando o acompanhamento da evolução dessas informações ao longo do tempo, conforme ilustrado nas figuras 16 e 17.

Figura 16 – Filtragem de ano

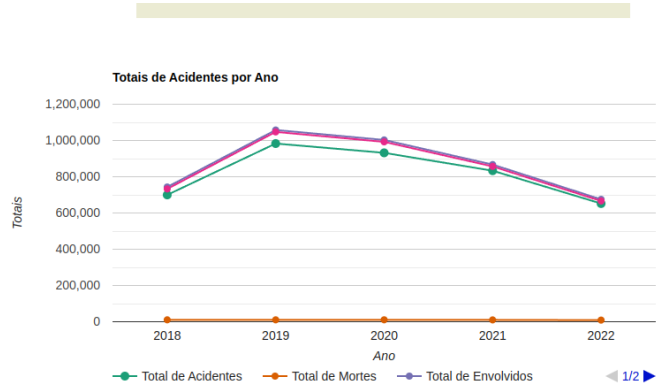


Figura 17 – Interface de dados de estatística

Por último, a página Calendário oferece filtros por fase do dia, dia da semana e mês, além da opção de selecionar o ano, possibilitando uma análise detalhada. A visualização dos filtros e das informações pode ser conferida nas figuras 18, 19, 20, 21 e 22.

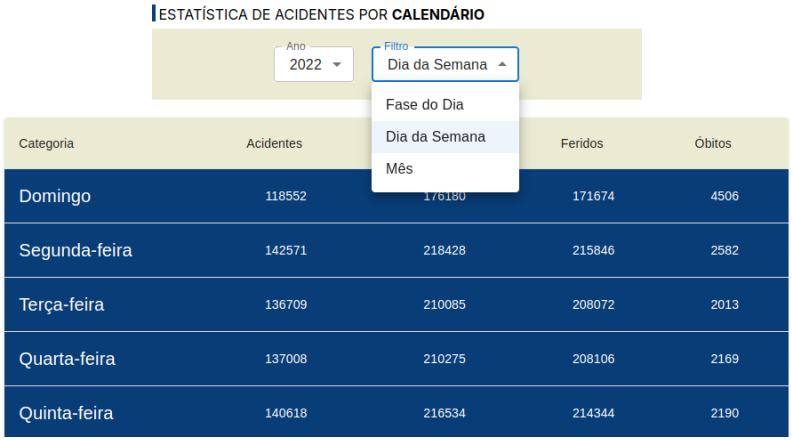


Figura 18 – Filtragem de calendário

Figura 19 – Interface de dados de calendário

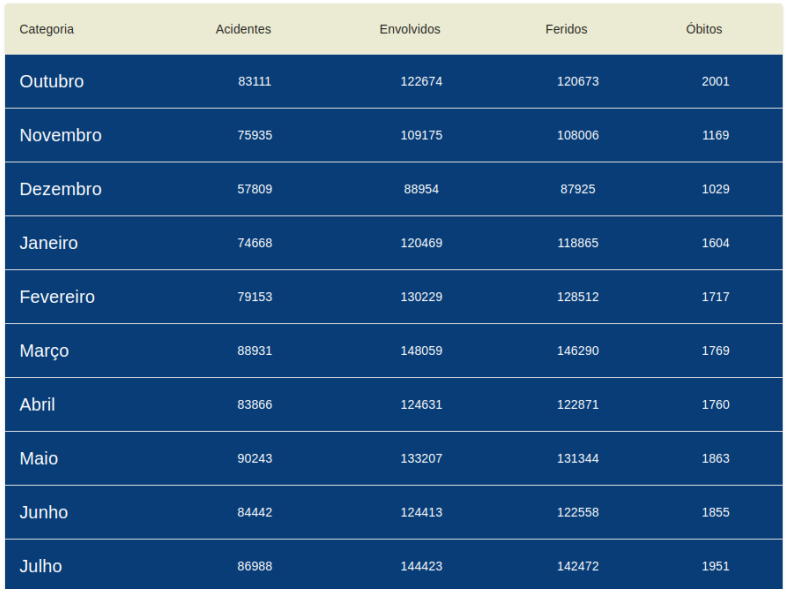


Figura 20 – Interface de dados de calendário mês

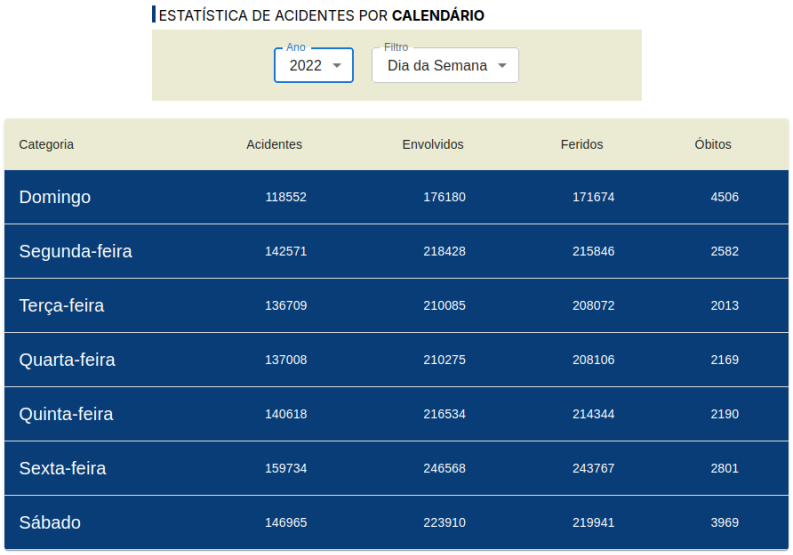


Figura 21 – Interface de dados de calendário mês

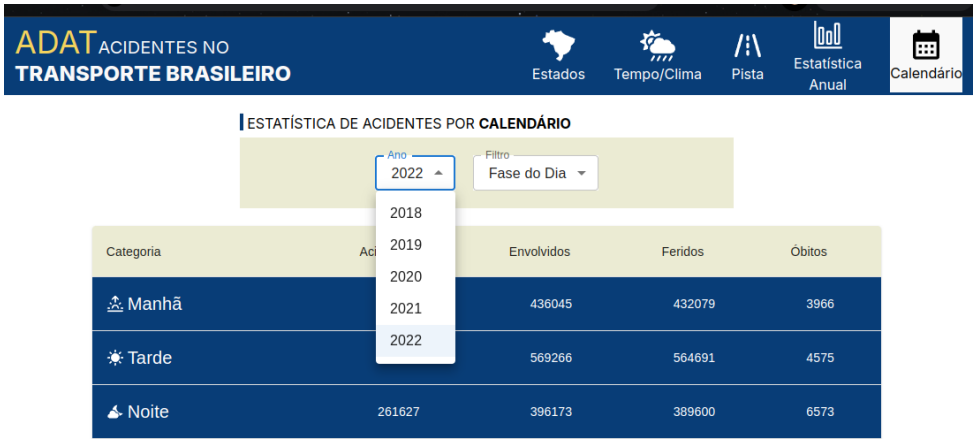


Figura 22 – Interface de dados de calendário ano

5 Considerações finais

A aplicação apresenta algumas limitações, especialmente no que diz respeito à interatividade e personalização da visualização dos dados. Atualmente, os usuários têm acesso às informações de acidentes de trânsito, mas não podem personalizar ou combinar gráficos e filtros conforme suas necessidades.

Apesar dessas limitações, o principal objetivo do projeto se concretizou. A aplicação fornece uma plataforma com informações relevantes sobre acidentes de trânsito, ajudando usuários a compreender melhor os pontos críticos e possíveis melhorias no trânsito.

A escolha do Go para a análise de dados demonstrou-se eficaz. A linguagem proporcionou um processamento ágil dos arquivos CSV, aproveitando a concorrência por meio de goroutines para otimizar a leitura e a análise dos dados. Essa abordagem permitiu um desempenho elevado, tornando a solução eficiente para lidar com grandes volumes de informação.

Durante o desenvolvimento, foi realizado um teste de performance com o objetivo de comparar a execução linear tradicional com o processamento paralelo utilizando goroutines, recurso nativo da linguagem Go. A análise foi aplicada ao processamento de grandes volumes de dados referentes a acidentes de trânsito, organizados em arquivos CSV com milhões de linhas.

Foram avaliados três cenários, execução linear, arquitetura paralela sem otimizações e arquitetura paralela com otimizações. Embora o processamento concorrente tenha potencial para aproveitar múltiplos núcleos do processador, os resultados mostraram que, em cenários sem otimizações, o tempo de execução podia até aumentar.

Essa variação no desempenho pode ser explicada por uma série de fatores técnicos. Um dos principais está relacionado ao *overhead*[1] da criação e sincronização das goroutines. Embora as goroutines sejam extremamente leves se comparadas a threads tradicionais (DONOVAN; KERNIGHAN, 2015), a criação massiva e descontrolada dessas rotinas pode levar a um consumo excessivo de recursos, como memória e tempo de CPU, apenas para coordenar sua execução. Além disso, operações de leitura e escrita concorrentes em estruturas compartilhadas exigem mecanismos de sincronização como sync.Mutex ou sync.WaitGroup, que podem gerar contenção e impactar negativamente a performance geral Pike (2016).

Outro fator importante é o próprio tipo de tarefa sendo executada. No caso do processamento de arquivos CSV, a tarefa é intensiva em I/O (entrada e saída), e pode não

[1] Sobrecarga

se beneficiar tanto da paralelização se o gargalo estiver na leitura do disco, especialmente em sistemas com discos mecânicos ou arquivos extremamente grandes. Além disso, o parser CSV não é trivialmente paralelizável, uma linha depende do encerramento correto da anterior, e a divisão de arquivos sem um controle preciso pode causar inconsistências.

Com base nos testes realizados, verificou-se que, embora o processamento linear seja mais simples conceitualmente, ele apresentou tempos de execução significativamente superiores em comparação com a versão paralela otimizada. Os resultados reforçam que, quando o paralelismo é bem estruturado e evita redundâncias como a leitura repetida de arquivos os ganhos de desempenho são expressivos, especialmente em cenários com grandes volumes de dados.

É importante destacar, no entanto, que o uso inadequado de goroutines pode acarretar em *overhead*[2] desnecessário e até mesmo piores tempos de execução, dependendo da carga e da forma de sincronização. Contudo, no cenário avaliado, a arquitetura paralela bem estruturada superou claramente a abordagem linear em todos os testes realizados.

Portanto, este teste de performance serviu não apenas como um experimento técnico, mas também como um alerta sobre o uso criterioso da concorrência. Paralelizar um processo não garante, por si só, aumento de desempenho e pode, inclusive, gerar o efeito oposto quando mal planejado.

5.1 Trabalhos futuros

Para aprimorar a aplicação, algumas melhorias podem ser exploradas em versões futuras. Uma das principais evoluções consiste na implementação de gráficos mais avançados e interativos, permitindo que o usuário tenha maior controle sobre os dados exibidos. Além disso, a possibilidade de filtragem personalizada e a combinação de diferentes conjuntos de dados tornariam a análise mais flexível e útil para diferentes perfis de usuários.

Outra melhoria relevante é a inclusão de um módulo informativo sobre trânsito, abordando leis e boas práticas de como se portar no trânsito. Esse recurso agregaria mais valor à aplicação, transformando-a não apenas em uma ferramenta de análise de acidentes, mas também em uma plataforma educativa sobre segurança no trânsito.

[2] Sobrecarga

Referências

BANKS, A.; PORCELLO, E. **Learning React: Modern Patterns for Developing React Apps**. 3. ed. [S.l.]: O'Reilly Media, 2023. Citado na página 28.

BERNERS-LEE, T. **Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web**. New York: HarperSanFrancisco, 2009. Citado na página 15.

BRASIL, O. K.; DAPP, F. **Índice de Dados Abertos para Cidades: Resultados 2018**. [S.l.], 2018. Acesso em: 20 jul. 2025. Disponível em: <https://ok.org.br/projetos/indice-dados-abertos/>. Citado na página 12.

CHACóN, S.; STRAUB, B. **Pro Git**. 2. ed. Berkeley: Apress, 2014. Acesso em: 19 fev. 2025. Disponível em: <https://git-scm.com/book/en/v2>. Citado na página 35.

DESIGN, A. **VOLVO - PORTAL ATLAS DA ACIDENTALIDADE**. 2025. Acesso em: 19 fev. 2025. Disponível em: <https://www.aliensdesign.com.br/portfolio/volvo-portal-atlas-da-acidentalidade/>. Citado na página 28.

DETRAN-RO. **Anuário Estatístico de Sinistros de Trânsito de Rondônia**. 2025. Acesso em: 13 julh. 2025. Disponível em: <https://www.detran.ro.gov.br/post/39/2023/9/12/anuario-estatistico-de-sinistros-de-transito-de-rondonia/>. Citado na página 29.

DONOVAN, A. A. A.; KERNIGHAN, B. W. **The Go Programming Language**. 1. ed. [S.l.]: Addison-Wesley Professional, 2015. ISBN 978-0134190440. Citado 5 vezes nas páginas 16, 23, 24, 27 e 47.

FRANCILEUDO, O. **Licença MIT no Github - O que é isso?** 2025. Acesso em: 24 fev. 2025. Disponível em: <https://www.dio.me/articles/licenca-mit-no-github-o-que-e-isso>. Citado na página 34.

GANJTABESH, A. **Speeding Up Go Concurrency: Unlocking the Power of Array Segmentation**. 2024. Acesso em: 20 jun. 2025. Disponível em: <https://medium.com/%40agtabesh/speeding-up-go-concurrency-unlocking-the-power-of-array-segmentation-dd51a798c9d4>. Citado na página 23.

GEORGE, C. **Sistemas distribuidos conceitos e projetos**. Porto Alegre: Bookman, 2013. Citado na página 30.

GOMES, R. **Docker para desenvolvedores**. [S.l.]: Novatec Editora, 2020. Citado na página 28.

IPEA. **Seminário discute desafios para a consolidação do Estado digital no Brasil**. [S.l.], 2025. Disponível em: <https://www.ipea.gov.br>. Citado na página 29.

JAIN, R. **The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling**. New York: John Wiley Sons, 1991. Citado na página 22.

KENNEDY, W.; KETELSEN, B.; MARTIN, E. S. **Go in Action**. Shelter Island, NY: Manning Publications, 2015. Citado na página 16.

LAZOTI, R. **Armazenando dados com Redis**. [S.l.]: Casa do Código, 2020. Citado na página 17.

LUANY, G. **Quase metade dos estados não divulga online dados sobre acidentes e infrações de trânsito**. 2024. Acesso em: 15 jul. 2025. Disponível em: <https://www1.folha.uol.com.br/cotidiano/2024/11/quase-metade-dos-estados-nao-divulga-online-dados-sobre-acidentes-e-infracoes-de-transito.shtml>. Citado 2 vezes nas páginas 12 e 13.

MAJDI, K. **Why Your Goroutines Are Slower Than You Think - Karam Majdi - Medium**. 2025. Acesso em: 20 jun. 2025. Disponível em: <https://medium.com/%40karam.majdi33/why-your-goroutines-are-slower-than-you-think-433372e6e50b>. Citado na página 23.

MATHEUS, R.; RODRIGUES, D.; VAZ, J. C.; JAYO, M. Análise do nível de abertura de dados governamentais de trânsito no brasil. **Revista Eletrônica de Sistemas de Informação**, v. 15, n. 2, p. 1–15, ago 2016. Acesso em: 17 jul. 2025. Disponível em: <https://www.periodicosibepes.org.br/index.php/reinfo/article/viewFile/2284/pdf>. Citado na página 13.

METROPOLES. **Ranking Trágico: Brasil é 3º país que mais registra mortes no trânsito**. 2024. Acesso em: 14 ago. 2024. Disponível em: <https://www.metropoles.com/brasil/ranking-tragico-brasil-e-3o-pais-que-mais-registra-mortes-no-transito>. Citado na página 12.

OMS. **Road safety project run by PAHO, PRF, Senatran, Dnit and UnB presents results of its first stage**. 2025. Acesso em: 29 fev. 2025. Disponível em: <https://www.paho.org/en/news/4-11-2024-road-safety-project-run-paho-prf-senatran-dnit-and-unb-presents-results-its-first>. Citado na página 12.

PIKE, R. **Concurrency is not Parallelism by Rob Pike**. 2016. Acesso em: 26 jun. 2025. Disponível em: <https://www.youtube.com/watch?v=oV9rvDllKEg&t=912s&ab_channel=gnbitcom>. Citado na página 47.

PORTAL DO TRâNSITO. **O trânsito brasileiro precisa de estatísticas confiáveis**. 2024. Acesso em: 15 jul. 2025. Disponível em: <https://www.portaldotransito.com.br/noticias/conscientizacao/o-transito-brasileiro-precisa-de-estatisticas-confiaveis/>. Citado 2 vezes nas páginas 12 e 29.

RAZVAN, M. **Fullstack React with TypeScript and Next.js: Build powerful, scalable web apps with modern React tools**. [S.l.]: Packt Publishing, 2023. Citado na página 27.

RIBEIRO, D. M. **Visualização de dados na internet: mapas e cartografias do ciberespaço**. Dissertação (Dissertação de Mestrado) — Pontifícia Universidade Católica de São Paulo, São Paulo, 2009. Disponível em: <https://repositorio.pucsp.br/handle/handle/18226>. Citado na página 13.

RIBEIRO, R. **Concorrência em Go - Rafael Ribeiro**. 2025. Acesso em: 19 fev. 2025. Disponível em: <https://medium.com/@rafaelmgr12/concorr%C3%AAncia-em-go-85b6a127b12f>. Citado na página 24.

RUEDIGER, M. A. **Cidades brasileiras ainda patinam ao divulgar dados públicos compreensíveis**. 2024. Acesso em: 20 jul. 2025. Disponível em: <https://www.poder360.com.br/brasil/cidades-brasileiras-ainda-patinam-ao-divulgar-dados-publicos-compreensiveis>. Citado na página 12.

SERFASS, D.; TANG, P. Comparing parallel performance of go and c++ tbb on a direct acyclic task graph using a dynamic programming problem. 03 2012. Citado na página 23.

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. **Operating System Concepts**. 9. ed. Hoboken, NJ: John Wiley & Sons, 2018. Citado na página 16.

TANENBAUM, A. S.; BOS, H. **Modern Operating Systems**. 4. ed. Boston: Pearson, 2015. Citado na página 16.

TANENBAUM, A. S.; STEEN, M. V. **Distributed Systems: Principles and Paradigms**. 2. ed. [S.l.]: Pearson Prentice Hall, 2007. Citado na página 15.

TEMPORIN, T. **Quais as diferenças entre goroutines e threads - Aprenda Golang**. 2022. Acesso em: 21 jun. 2025. Disponível em: <https://aprendagolang.com.br/quais-as-diferencas-entre-goroutines-e-threads/>. Citado na página 16.

TUFTE, E. R. **The Visual Display of Quantitative Information**. Cheshire, CT: Graphics Press, 1983. Citado na página 15.

VALIATI, E. R. d. A. **Avaliação de usabilidade de técnicas de visualização de informações multidimensionais**. Tese (Doutorado) — Universidade Federal do Rio Grande do Sul, Porto Alegre, 2008. Citado na página 29.

WAGNER, N. **O Básico de Redis Cache Distribuído - Wagner Nogueira - Medium**. 2025. Acesso em: 19 fev. 2025. Disponível em: <https://medium.com/@engnogueirawgn/o-b%C3%A1sico-de-redis-cache-distribu%C3%ADdo-7b558032bb59#:~:text=Diferencial%20do%20Redis,muito%20com%20pouca%20complexidade%20interna.> Citado na página 17.

Anexos

ANEXO A – Licença MIT