Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

Painel analítico dos dados do projeto BikeSP

Filipe Tressmann Velozo

Monografia Final

mac 499 — Trabalho de Formatura Supervisionado

Supervisor: Prof. Dr. Fabio Kon
Cossupervisor: Prof. Dr. Paulo Meireles

São Paulo
2025

Sumário

Capítulo 1

Introdução

Em setembro de 2016 a cidade de São Paulo sancionou a Lei nº 16.547 (São Paulo, 2016), que criou o Programa BikeSP — uma iniciativa que prevê a concessão de créditos de mobilidade para cidadãos que utilizem a bicicleta como meio de transporte. A implementação prática da iniciativa teve início apenas em 2023 (Lima, 2023), quando o IME-USP passou a desenvolver um aplicativo capaz de registrar viagens de bicicleta e enviá-las à SPTrans com o propósito de conceder créditos aos usuários.

A aplicação, embora tenha sido validada por cerca de mil usuários em testes-piloto realizados em 2025, ainda carece de ferramentas de análise de dados que permitam aos administradores acompanhar e monitorar a evolução de seu uso. Além disso, a exploraçaão dos dados de viagem tem potencial para gerar insights sobre possíveis caminhos para a expansão de ciclofaixas e ciclovias na cidade de São Paulo.

Nesse sentido, este trabalho propõe a criação de uma ferramenta de análise de dados que auxilie os administradores do aplicativo a monitorar a evolução do projeto e que ofereça subsídios a agentes públicos para identificar lacunas na infraestrutura cicloviária da cidade.

A ferramenta desenvolvida consiste em um painel online que permite aplicar diversos tipos de filtros e agregações aos dados do BikeSP. Adicionalmente, o painel inclui um mapa de calor da cidade de São Paulo, que possibilita visualizar os locais mais visitados por bicicleta.

1.1 Contextualização

Em setembro de 2016, a cidade de São Paulo sancionou a Lei nº 16.547, que instituiu o Programa BikeSP, com previsão de concessão de créditos de mobilidade a usuários que utilizem a bicicleta como meio de transporte. Embora a lei devesse entrar em vigor em 1º de janeiro de 2017, sua implementação foi postergada por falta de base científica e de dados empíricos necessários para definir critérios operacionais e mecanismos de fiscalização adequados. Na época, não existiam métodos claros para comprovar o uso efetivo da bicicleta, determinar o valor a ser pago por viagem ou estabelecer a forma de pagamento. Também não estava bem definido quais recortes da população teriam direito

ao benefício, qual valor seria pago, durante quanto tempo nem qual seria o custo anual para os cofres públicos deste programa.

Com o objetivo de suprir essa lacuna, o Instituto de Matemática e Estatística da Universidade de São Paulo (IME–USP) iniciou, em 2023, o desenvolvimento de um aplicativo para registrar viagens, validar se elas foram efetivamente realizadas de bicicleta e gerenciar os pagamentos correspondentes. Após testes iniciais, o aplicativo entrou em fase piloto em junho de 2025, com aproximadamente mil usuários.

Apesar dos resultados positivos dessa fase piloto, o projeto ainda necessita de ferramentas que permitam a exploração dos dados gerados, de modo a validar experimentos e apoiar a definição de critérios operacionais. Ademais, os dados de geolocalização armazenados têm grande potencial para o planejamento de políticas públicas, pois podem ser usados para identificar trechos mais movimentados e indicar locais prioritários para a implantação de novas ciclovias e ciclofaixas..

1.2 Objetivos

O objetivo principal deste trabalho é fornecer uma ferramenta que possibilite a exploração dos dados do aplicativo BikeSP e permita o monitoramento do uso da plataforma pelos administradores. Para tanto, a ferramenta exibe gráficos com métricas de interesse — como número de viagens, duração média das viagens e distância média percorrida — agregadas por diferentes categorias — como nível de renda, raça/cor e hora do dia.

Além disso, a ferramenta apresenta um mapa de calor da cidade de São Paulo que mostra a concentração de pontos amostrados em cada local, facilitando a identificação dos locais mais visitados pelos usuários e apoiando o planejamento de novas ciclorotas.

1.3 Organização

Este trabalho está organizado em quatro capítulos:

- Capítulo 1: Introdução (este capítulo).
- Capítulo 2: Painel analítico do BikeSP — analisa plataformas semelhantes para entender possibilidades e identificar padrões de uso dos usuários, a partir dos quais são extraídos requisitos funcionais e não funcionais para a aplicação.
- Capítulo 3: Implementação — descreve como a plataforma foi construída, apresenta os principais desafios encontrados e as soluções adotadas.
- Capítulo 4: Conclusão — discute o impacto do projeto e aponta possíveis trabalhos futuros.

Capítulo 2

Painel analítico do BikeSP

Com o objetivo de compreender o que será desenvolvido e definir os requisitos da aplicação, este capítulo apresenta a análise de algumas plataformas semelhantes. A partir dessa análise, são identificados requisitos funcionais que irão guiar a implementação do sistema.

2.1 Plataformas semelhantes

Após uma breve pesquisa, foram identificadas algumas plataformas de exploração de dados que poderiam servir de inspiração para este trabalho. Esta seção descreve o funcionamento de cada uma delas, de modo a orientar as capacidades desejáveis para o painel analítico do BikeSP.

2.1.1 HealthDashboard

O *HealthDashboard* (InterSCity, 2016), criado no contexto do projeto InterSCity, reúne dados disponibilizados pelo Sistema Único de Saúde (SUS) e permite sua visualização por meio de gráficos e mapas de calor. Por exemplo, o mapa de calor da figura 2.1 apresenta a quantidade de internações realizadas na rede pública do Rio de Janeiro:

ALém disso, a plataforma também permite aplicar filtros sobre os dados exibidos no mapa, como o estabelecimento onde ocorreu a internação ou o diagnóstico principal, acessíveis pelo menu à direita como pode ser visto na figura 2.2

Apesar da interface intuitiva, o painel apresenta um tempo de carregamento elevado — geralmente acima de 10 segundos. Isso ocorre porque a página carrega todos os dados do mapa de calor de uma só vez. Embora essa estratégia funcione para volumes menores de dados, ela representa um alerta para este projeto, pois o volume de dados do BikeSP tende a ser significativamente maior e a aumentar com o tempo, tornando essa abordagem inviável para o caso deste trabalho.

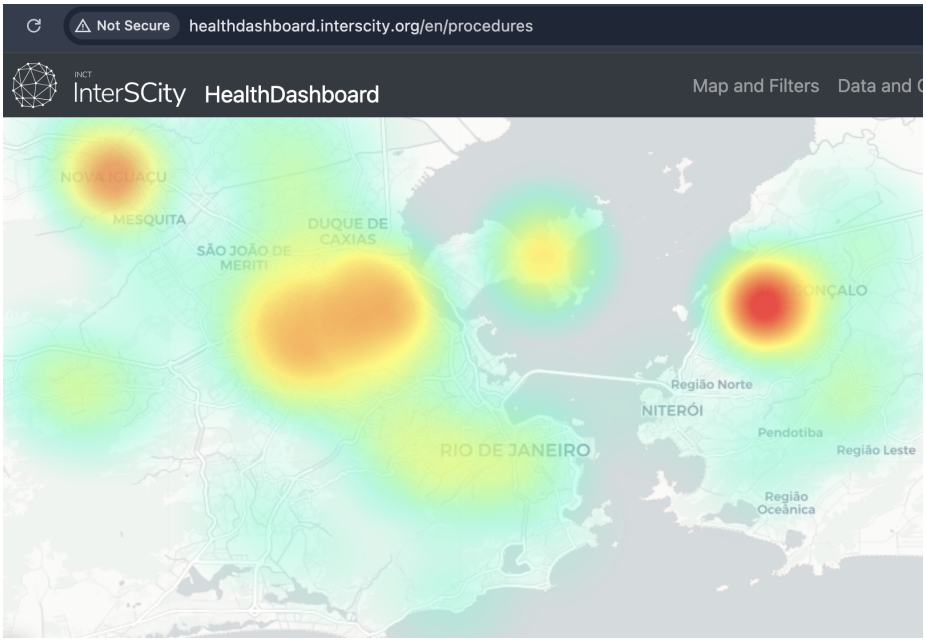

Figura 2.1: *Internações Hospitalares por Dengue na Grande Rio de Janeiro em 2019*

2.1.2 BikeScienceWeb

O *BikeScienceWeb* (InterSCity, 2023) é um painel online que permite visualizar as pesquisas Origem–Destino (OD) realizadas pelo Metrô de São Paulo. A ferramenta representa deslocamentos por meio de setas entre pontos e possibilita aplicar diversos filtros, como faixa etária dos viajantes e tempo de viagem. Uma exemplo do mapa apresentado pela aplicação pode ser visto na figura 2.3.

Diferentemente do HealthDashboard, o BikeScienceWeb permite adicionar camadas ao mapa, como linhas de metrô e ciclovias, possibilitando a visualização da infraestrutura de transporte da cidade. Essa funcionalidade pode ser interessante para o painel do BikeSP, pois permitiria verificar se determinadas ciclovias estão sendo utilizadas ou identificar a necessidade de novas rotas.

2.1.3 Strava

O Strava (Strava, 2023) é uma plataforma digital voltada ao acompanhamento e análise de atividades físicas, amplamente utilizada por praticantes de corrida, ciclismo e outros esportes ao ar livre. O serviço integra monitoramento por GPS com funcionalidades de caráter social, permitindo registrar treinos, avaliar desempenho e interagir com outros usuários.

Com acesso a uma grande quantidade de amostras de localização, a empresa disponibiliza mapas de calor que indicam a prática de cada esporte, incluindo o ciclismo.

O mapa de calor do Strava é global e possui tempo de carregamento reduzido, proporcionando uma experiência fluida ao usuário. No entanto, não é possível aplicar filtros sobre os dados exibidos e os mapas disponibilizados são exclusivamente de densidade, não contemplando informações como velocidade média dos ciclistas em uma dada localização.

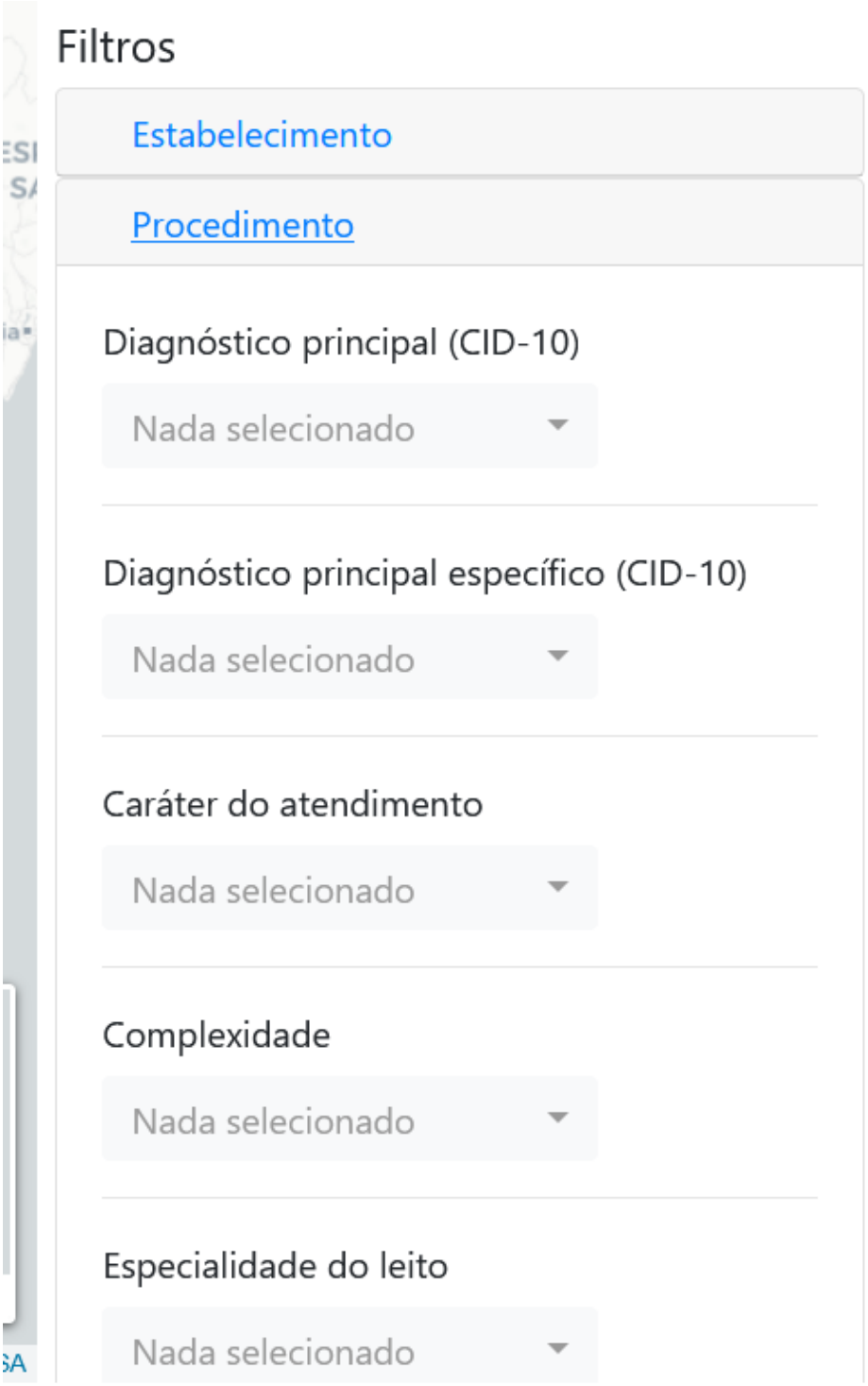

Figura 2.2: *Exemplo de interface do HealthDashboard*

2.2 Requisitos funcionais

Com base nas capacidades observadas nas plataformas analisadas, é possível definir algumas funcionalidades essenciais. A capacidade de exibir gráficos com algumas métricas como é feito no Healthdashboard é interessante. No caso deste trabalho, visualizar a quan-

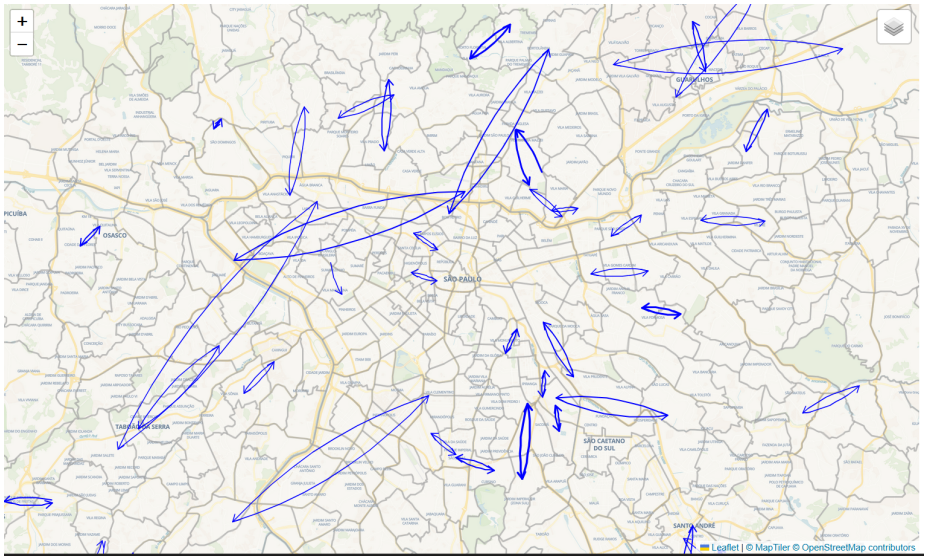

Figura 2.3: *Fluxos cicloviários mostrados pelo BikeScienceWeb*

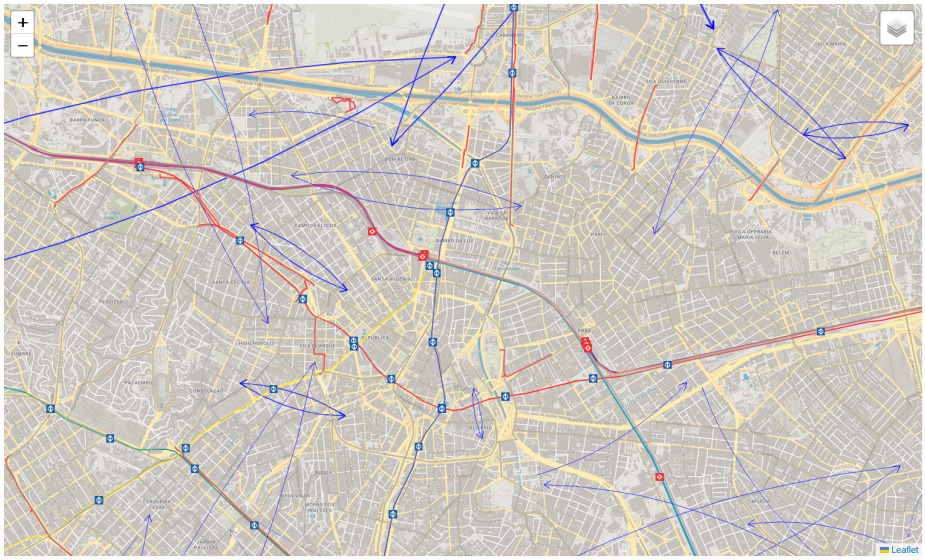

Figura 2.4: *O BikeScienceWeb permite adicionar camadas com a infraestrutura de transporte de São Paulo*

tidade de viagens realizadas ou a distância média percorrida sobre diferentes agregações e filtros forneceria aos administradores do aplicativo uma visão geral sobre como seus usuários se comportam. Por exemplo, seria possível visualizar horários de pico de uso, quanto eles devem se locomover para chegar ao destino e quanto tempo leva.

Além disso, ter um mapa de calor que indique os percursos feitos pelos usuários, assim como no strava, possibilitaria encontrar algumas lacunas na infraestrutura cicloviaria da cidade, o que poderia apoiar administradores publicos na tomada de decisões. Nesse sentido, é importante que também seja possível visualizar a infraestrutura de transportes existente, como ciclovias e linhas metrô assim como é feito no BikeScienceWeb.

Sobre o mapa de calor, um ponto importante a se ressaltar é a privacidade. Como a

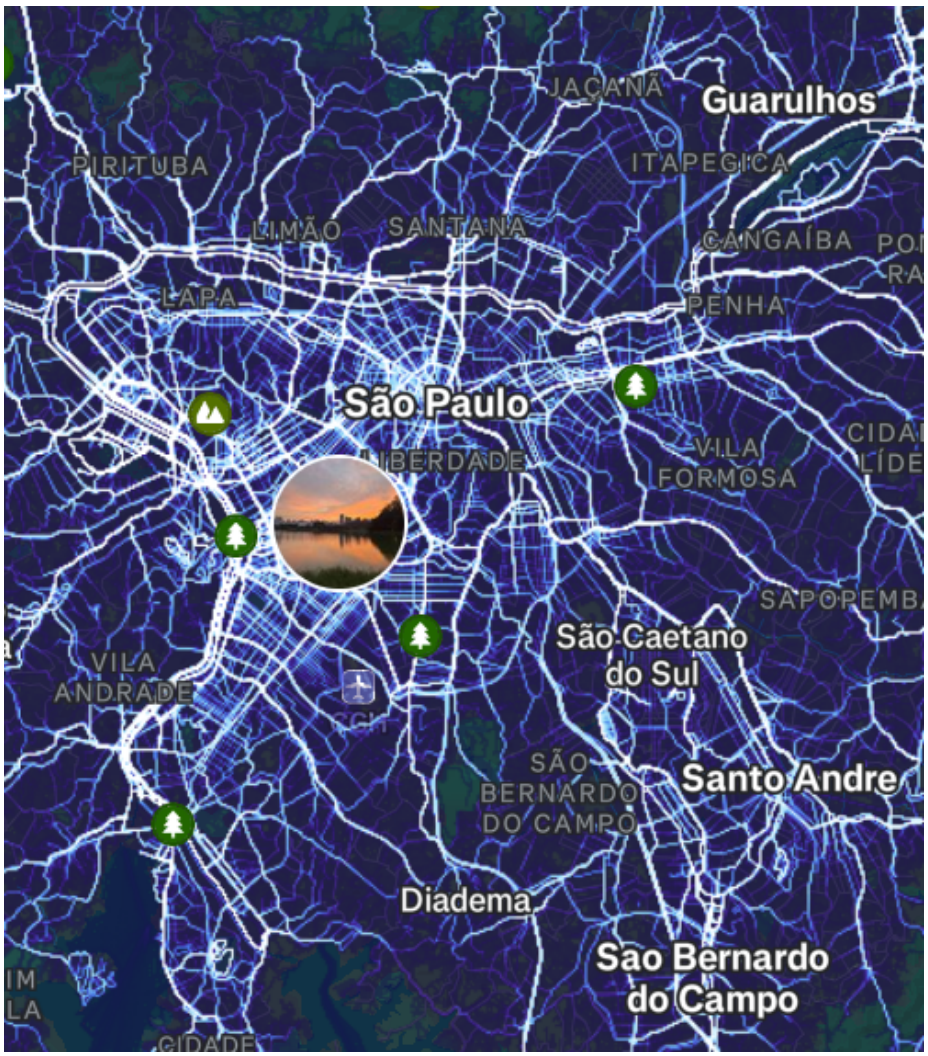

Figura 2.5: *Mapa de calor do uso de bicicleta disponibilizado pelo Strava*

localização é uma informação de identificação pessoal, a plataforma de analise de dados deve anonimizar esses informações. Não deveria ser possível identificar uma pessoa com base no mapa de calor.

Outro ponto importante é que a implementação deve levar em conta o desempenho do painel a fim de entregar uma experiência fluida de navegação. Simplesmente carregar todos os dados na interface gráfica do usuário como é feito no HealthDashboard e no BikeScienceWeb não será escalavel nesse caso já que o volume de dados facilmente ultrapassa 1 GB.

Tendo em vista essas considerações, podemos definir os seguintes requisitos funcionais e não funcionais:

1. **Visualizações e agregações:** o painel deve exibir gráficos com as seguintes métricas:
    - número de viagens;
    - duração média das viagens;
    - distância média percorrida.

Essas métricas devem estar disponíveis com agregações por:

- data;
- dia da semana;
- hora do dia;
- gênero;
- raça/cor;
- nível de remuneração da viagem.

2. **Filtragem:** o sistema deve permitir a aplicação de filtros com base nos seguintes critérios:
   - data;
   - dia da semana;
   - hora do dia;
   - gênero;
   - raça/cor;
   - nível de remuneração da viagem.

   Deve ser possível aplicar múltiplos filtros simultaneamente a qualquer forma de visualização.
3. **Mapa de calor**: o painel deve incluir um mapa de calor da cidade de São Paulo exibindo as localizações amostradas dos usuários.
4. **Visualização da infraestrutura:** o painel deve incluir um mapa de calor da cidade de São Paulo exibindo as localizações amostradas dos usuários.

2.3 Requisitos não funcionais

Como a aplicação lida com dados de geolocalização, classificados como sensíveis segundo a LGPD, é necessário definir métodos adequados de anonimização, garantindo que nenhum usuário do BikeSP possa ser identificado.

Além disso, considerando o grande volume de dados, a escalabilidade do sistema deve ser tratada como uma prioridade. Para isso, estabelece-se como requisito não funcional que todas as requisições tenham tempo de resposta inferior a 10 segundos.

1. **Anonimização:** Os dados apresentados no mapa de calor devem ser anonimizados a fim de não ser possível identificar um usuário particular. Por exemplo, não deveria ser possível inferir o domicilio e local de trabalho de um usuário.
2. **Performance:** O painel deve entregar experiência fluida. O tempo máximo de carregamento deve ser de no máximo 10 segundos para alguma visualização de dados particular.

Com esse embasamento inicial, é possível iniciar o desenvolvimento do sistema proposto.

Capítulo 3

Implementação

Neste capítulo será apresentada a arquitetura do sistema, detalhes da implementação e alguns dos desafios encontrados, bem como suas soluções. Todo o codigo desenvolvido pode ser encontrado https://github.com/filipetressmann/tcc e o sistema pode ser acesso em https://interscity.org/bikesp/piloto/dashboard

3.1 Arquitetura de software

Esta seção tem como objetivo fornecer uma visão inicial da arquitetura do sistema. As próximas subseções apresentam as principais decisões que fundamentam a implementação deste trabalho.

3.1.1 Componentes do sistema

O sistema é estruturado como uma aplicação web distribuída, composta pelos seguintes componentes principais:

- **Interface web (Frontend):** Executada no navegador do usuário, é responsável pela visualização dos dados e pela interação com o sistema. Esse componente apresenta mapas de calor, gráficos e controles de filtragem, enviando requisições ao servidor conforme as ações do usuário.
- **Servidor de aplicação (Backend):** Atua como intermediário entre a interface web e o banco de dados. Esse componente recebe as requisições do frontend, processa os parâmetros de consulta, executa operações de agregação e retorna apenas os dados já processados, reduzindo a quantidade de informação transmitida ao cliente.
- **Banco de dados geoespacial:** Responsável pelo armazenamento dos dados de trajetos e demais informações necessárias para a análise. Foi adotado o PostgreSQL com a extensão PostGIS, devido ao seu suporte a consultas geoespaciais e operações espaciais avançadas, permitindo filtros e agregações eficientes sobre grandes volumes de dados.

- **Etapa de transformação de dados:** Componente responsável pela reorganização dos dados originais do BikeSP em um esquema adequado para consultas geoespaciais. Nessa etapa, os dados são limpos, transformados e anonimizados antes de serem carregados no banco de dados, garantindo melhor desempenho nas consultas e maior cuidado com a privacidade dos usuários.

O fluxo de funcionamento do sistema inicia-se na interface web, quando o usuário seleciona uma área do mapa ou aplica filtros de análise. Essas ações resultam em requisições enviadas ao servidor de aplicação, que interpreta os parâmetros recebidos e consulta o banco de dados geoespacial. As consultas são executadas sobre os dados previamente transformados e agregados, e apenas os resultados consolidados são retornados ao frontend para visualização.

A etapa de transformação de dados ocorre previamente ao uso do sistema, alimentando o banco de dados com informações estruturadas e otimizadas para consultas espaciais. Esse processo evita que dados brutos sejam manipulados diretamente em tempo de execução, contribuindo para o desempenho e a escalabilidade da aplicação.

3.1.2 Diagrama do sistema

Tendo em vista as considerações das subseções anteriores, temos o seguinte diagrama do sistema:

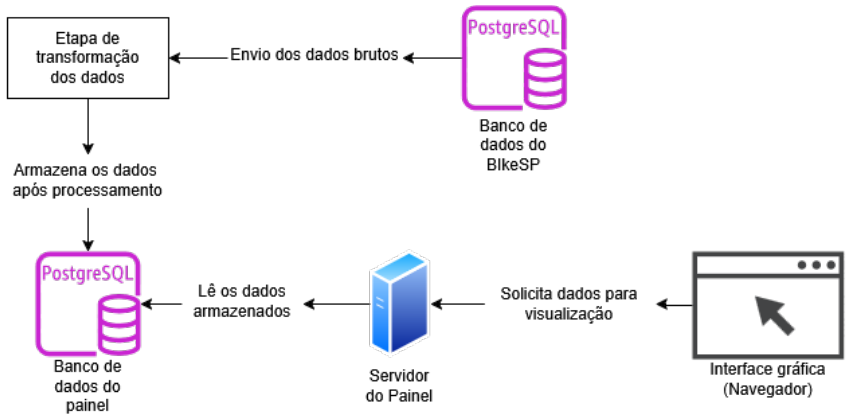

Figura 3.1: *Diagrama do painel analitico*

3.1.3 Decisões arquiteturais

Escolha do banco de dados

Assim como nas outras plataformas estudadas, o sistema implementado também será uma aplicação web. A interface de usuário, executada no navegador, fará chamadas para um servidor, que será responsável por interpretar cada requisição e retornar os dados solicitados.

Para manter a interface rápida, os dados não serão enviados em sua totalidade para o cliente, como ocorre no HealthDashboard e no BikeScienceWeb. Considerando que a quantidade total de dados aqui é bem maior (superior a 1GB), torna-se mais eficiente

armazená-los em um banco de dados, realizando a leitura e o processamento no servidor e enviando apenas os dados já agregados para a interface. Isso reduz a quantidade de memória necessária para o navegador renderizar a página, tornando a experiência mais fluida.

Dessa forma, é necessário escolher um sistema gerenciador de banco de dados adequado. Como o projeto precisa lidar com dados de geolocalização, o banco escolhido deve oferecer bom suporte a esse tipo de informação. Algumas opções avaliadas foram:

- **PostgreSQL (com a extensão PostGIS):** O PostgreSQL (PGDG, 2025) é um banco relacional de código aberto. Ele oferece suporte a uma grande variedade de consultas sem exigir tanto planejamento de índices quanto um banco não relacional. Além disso, a extensão PostGIS (PSC, 2025) lida muito bem com dados de geolocalização e permite executar diferentes tipos de operações espaciais de forma eficiente.
- **MongoDB**: O MongoDB (MongoDB, 2024) é um banco de dados não relacional. Embora ofereça suporte a consultas geoespaciais, exige um planejamento mais cuidadoso dos índices. A variedade de consultas que podem ser executadas de forma eficiente também é menor, o que limita filtros e agregações mais complexas.
- **Elasticsearch:** O Elasticsearch (Elasticsearch, 2024) é um sistema de busca distribuído que também pode funcionar como um banco de dados orientado a documentos. Ele suporta consultas geoespaciais, como busca por pontos próximos ou filtragem por áreas, e funciona muito bem com grandes volumes de dados. Seu principal ponto forte é a rapidez nas consultas. Porém, sua configuração e manutenção são mais trabalhosas, e ele não oferece operações espaciais avançadas como as do PostGIS.

Considerando essas opções, a melhor alternativa para este trabalho é o PostgreSQL, devido s sua simplicidade de configuração, flexibilidade para realizar consultas com diferentes filtros e agregações e a suporte a dados geoespaciais por meio da extensão PostGIS.

Etapa de transformação dos dados

Os dados armazenados nas tabelas do servidor do BikeSP não estão em um formato adequado para as consultas de geolocalização que deverão ser realizadas. Por exemplo, todas as amostras de localização dos usuários estão em campos do tipo *jsonb*. Isso significa que, mesmo para consultas simples — como buscar todos os pontos dentro de uma determinada distância do centro de São Paulo — seria necessário ler todo o dataset, tornando o processo lento e ineficiente.

Por esse motivo, a arquitetura do sistema deve incluir uma etapa de transformação de dados, na qual o dataset do BikeSP será reorganizado em um novo esquema de tabelas mais adequado para consultas geoespaciais e qualquer dado de identificação pessoa filtrado. Essa etapa também permitirá realizar a limpeza dos dados, já que é possível que existam registros inválidos nas tabelas originais.

Além disso, essa etapa também ajuda a na questão de privacidade já que com essa passo seria possível retirar qualquer informação de identificação pessoal antes de carrega-la no banco de dados.

Reúso do BikeScienceWeb

Com o objetivo de reduzir o tempo de desenvolvimento e facilitar a integração com o BikeScienceWeb, o painel analítico do BikeSP será implementado como uma extensão da aplicação já existente. Assim, será possível reutilizar partes do backend, do frontend e também a infraestrutura que já está em produção.

3.2 Esquema de banco de dados

Com a decisão de utilizar um banco de dados, é necessário definir um esquema adequado para armazenar as informações do projeto. Para isso, é preciso analisar o esquema das tabelas relacionais utilizadas pelo aplicativo BikeSP a fim de compreender quais dados estão disponíveis e quais serão úteis para o painel. As tabelas disponíveis podem ser encontradas no arquivo *tables.sql*.

Após analise, apenas essas tabelas possuem dados uteis para o painel:

```
CREATE TABLE PESSOA (
   idPessoa integer,
   genero varchar(2),
   raca varchar(10)
);

CREATE TABLE VIAGEM (
   idpessoa integer,
   remuneracao money,
   idViagem integer,
   data timestamp,
   deslocamento double precision,
   idOrigem integer,
   idDestino integer,
   status varchar(30),
   motivoStatus varchar(100),
   trajeto jsonb
);
```

A tabela `PESSOA` pode ser reutilizada na aplicação com o mesmo esquema original. Entretanto, o campo `trajeto` da tabela `VIAGEM` apresenta um desafio. Esse campo armazena os dados do percurso, representados por um vetor de objetos, em que cada objeto contem uma data e uma localização geoespacial (latitude e longitude). Por exemplo:

```
[
    {
        "Data": "2025-01-01 00:00:00 GMT-03:00",
        "Posicao": {
            "latitude": -23.5456768,
            "longitude": -46.6411423
        }
```

```
    }
]
```

Por estar em formato `jsonb`, esse campo não permite a realização de consultas geoespaciais diretamente no banco de dados. Por exemplo, para identificar todos os pontos localizados em uma determinada região, será necessário carregar todos os objetos `jsonb` e verificar cada amostra, o que torna as consultas excessivamente lentas e inviáveis para uso em uma interface web interativa.

Diante desse problema, optou-se por normalizar[1] a estrutura da tabela, removendo o campo `trajeto`. Essa normalização consiste em extrair cada objeto do vetor para uma nova tabela denominada `LOCATIONS`, além da criação de uma nova tabela `TRIP`, que não conteria mais o campo `jsonb`:

```
1   CREATE TABLE TRIP (
2       payout money NOT NULL,
3       idTrip serial PRIMARY KEY,
4       idPerson integer NOT NULL,
5       date timestamp NOT NULL,
6       distance double precision,
7       idOrigin integer,
8       idTarget integer,
9       status varchar(30) NOT NULL CHECK (status in ('EmAnalise', '
            Reprovado','Aprovado')),
10      statusReason varchar(100) NOT NULL,
11      duration interval,
12      payoutLevel double precision,
13      meanSpeed double precision
14  );
15
16  CREATE TABLE LOCATIONS (
17      idSample serial PRIMARY KEY,
18      idTrip integer NOT NULL REFERENCES TRIP (idTrip),
19      pointGeom GEOGRAPHY(Point, 4326) NOT NULL,
20      pointTimestamp timestamp,
21      geohash TEXT
22  );
```

Na nova tabela `LOCATIONS`, o campo `idSample` representa um identificador único para cada amostra. O campo `pointGeom` é um objeto do *PostGIS* responsável por armazenar a localização geoespacial do ponto. O campo `pointTimestamp` indica o momento em que a amostra foi registrada, enquanto o campo `geohash` armazena o código geográfico correspondente à posição. Por fim, o campo `idTrip` faz referência à tabela `TRIP`, indicando a qual viagem cada amostra pertence.

Esse novo esquema melhora significativamente o desempenho em consultas geoespaciais, pois a separação dos dados permite o uso completo das funcionalidades do

[1] https://en.wikipedia.org/wiki/Database_normalization

PostGIS. Além disso, o esquema possibilita a criação de índices especializados, que serão apresentados nas seções seguintes.

3.3 Estrutura do Backend

Esta seção descreve como o backend foi implementado, destacando alguns dos desafios encontrados e as soluções adotadas.

3.3.1 Modelo de requisições e chamadas de API

Para prover os dados de visualização ao usuário é necessário definir algumas APIs. Esta subseção foca em definir quais APIs são necessárias para prover todas as funcionalidades da interface, descrevendo seu comportamento e o modelo das requisições e respostas.

A API usada para recuperar os dados a serem visualizados em gráficos é a `fetch_metric`. Para chamar essa API o usuário deve informar um ou mais filtros, assim como o tipo de dado a ser recuperado e sua agregação. Diferentemente dos filtros, apenas uma agregação e um tipo de dado podem ser informados. Por exemplo, a requisição abaixo recupera a quantidade de viagens por dia da semana, filtrando apenas por viagens realizadas entre 01:15 e 15:52:

```
{
  "data_type": "TRIP_COUNT",
  "aggregation": "DAY_OF_WEEK",
  "filters": {
    "hour_filter": {
      "min": "01:15",
      "max": "15:52"
    }
  }
}
```

Essa chamada retorna um vetor de objetos em que cada objeto contém um rótulo e o valor do dado. No caso da chamada acima o rótulo seria o dia da semana e o valor seria a quantidade de viagens realizadas naquele dia. Por exemplo:

```
{
  "data": [
    {
        "label": "MONDAY",
        "value": 1203
    },
    {
        ...,
    },
    ...
  ]
}
```

Esse modelo de requisição/resposta funciona bem para gráficos, pois é fácil integrá-lo com bibliotecas de visualização. Entretanto, esse mesmo modelo é incompatível com requisições de dados de geolocalização.

Para requisições geoespaciais a agregação geralmente é implícita (os dados devem ser agregados por localização) e o formato "rótulo/valor" não é adequado para mapas de calor. A maioria das ferramentas de visualização de mapas de calor trabalha com tuplas onde o primeiro e o segundo item são latitude e longitude e o terceiro item é a intensidade naquele ponto. Além disso, para agregar os dados com resolução adequada ao nível de zoom é necessário informar esse nível na requisição; também é útil informar o centro da pesquisa para limitar o conjunto de dados retornado e assim poupar banda e recursos do servidor.

Dessa forma, uma API distinta foi criada para dados espaciais chamada *fetch_geographic_data*. A requisição inclui o tipo de dado a ser visualizado (por exemplo, velocidade média no ponto ou número de amostras), o nível de zoom, o centro da pesquisa, a distância máxima a partir do centro (em metros) e os filtros. Exemplo:

```
{
  "zoom_level": 8,
  "center": {
    "lat": -23
    "lng": -46
  },
  "max_distance": 14000,
  "data_type": "TOTAL_SAMPLES",
  "filters: {
    "hour_filter": {
      "min": "01:15",
      "max": "15:52"
    }
  }
}
```

O modelo de resposta pode retornar um vetor de tuplas `[lat, lng, intensity]`:

```
{
  "data": [
    [-23, -46, 1000]
  ]
}

Esse modelo facilita a construção do mapa de calor na interface final já que integra co
```

3.3.2 Integração com o PostgreSQL

Para que o servidor consiga retornar os dados solicitados para a interface de usuário, é necessário que o servidor traduza as requisições para uma consulta SQL e então envie a consulta para o banco de dados. Essa tradução não pode ser feita de maneira direta, por

exemplo, mapeando cada requisição diferente para uma consulta especifica, já que há uma grande quantidade de combinações possíveis entre filtros e agregações diferentes. Além disso, fazer esse mapeamento direto dificultaria a adição de novas métricas para o painel.

Para solucionar essa questão, optou-se pela implementação de uma estratégia de construção dinâmica de consultas. Essa abordagem baseia-se na definição da classe *Query*, que encapsula métodos para adicionar, de forma modular, as diferentes partes de uma instrução SQL, além de um método responsável pela montagem final da consulta. O exemplo a seguir ilustra a classe, destacando métodos para a definição do dado de saída (como *add_mean_speed*), bem como para a filtragem (*filter_by_gender*) e agregação (*aggregate_by_month*) dos dados:

```
class Query:
  ...
  def add_mean_speed(self):
       self.select_parts.append('AVG(t.meanSpeed) AS meanSpeed')
       self.table = "TRIP t"

  def aggregate_by_month(self):
       self.select_parts.append('DATE_TRUNC(\'month\', date) AS
          month')
       self.group_by_parts.append('month')

  def filter_by_gender(self, genders: list):
       self.join_person()
       self.add_filter('p.gender', 'IN', genders)
  def join_person(self):
       query = 'JOIN PERSON p ON t.idPerson = p.idPerson'
       if self.joins.count(query) > 0:
          return

       self.joins.append(query)
  ...
```

Uma vez que os componentes da consulta tenham sido definidos pelos métodos ilustrados, a consulta final é construída por meio do método apresentado a seguir. Este método consolida todas as informações acumuladas pelas chamadas anteriores, como as colunas a serem selecionadas e as operações de *JOIN* necessárias:

```
class Query:
  def build_query(self):
       self.froms.append(self.table)
       query_string = ''
       if len(self.withs) > 0:
          query_string = f"WITH {', '.join(self.withs)}"
       query_string += f"SELECT {', '.join(self.select_parts)}
          FROM {', '.join(self.froms)}"

```

```
        if self.joins:
            query_string += ' ' + '\n'.join(self.joins)

        if self.where_parts:
            query_string += ' WHERE ' + ' AND '.join(self.
                where_parts)

        if self.group_by_parts:
            query_string += ' GROUP BY ' + ', '.join(self.
                group_by_parts)
            if self.orderByGroups:
                query_string += ' ORDER BY ' + ', '.join(self.
                    group_by_parts)

        query_string += ';'
        return query_string
```

Após a construção, esse método retorna a consulta SQL completa, pronta para ser enviada ao banco de dados. O envio é realizado por meio do método *execute* da classe *Query*, que utiliza a biblioteca *psycopg2*. Note que todos os parâmetros incluídos na query são repassados separadamente nesta etapa, o que previne vulnerabilidades como a injeção de SQL.

```
class Query:
  def execute(self):
        query_string = self.build_query()
        self.params.extend(self.groupByParams)
        try:
            cursor = postgres.get_cursor()
            cursor.execute(query_string, self.params)
            data = cursor.fetchall()
            cursor.close()
            return data
        except Exception as e:
            print(f"Error executing query: {e}")
            raise
```

Com a classe *Query* finalizada, torna-se possível automatizar a criação e a execução da consulta por meio da análise do objeto da requisição. Para adicionar agregações e tipos de dados, o servidor consulta um mapa de chaves que correlaciona a campos da requisição com uma função específica da classe *Query*. Por exemplo, o mapa abaixo estabelece a relação entre os tipos de agregação solicitados e seus métodos associados na classe:

```
@operation_adder('aggregation')
def add_query_aggregation_operations(q: Query):
    return {
        'WEEK': q.aggregate_by_week,
        'HOUR': q.aggregate_by_hours,
        'DAY_OF_WEEK': q.aggregate_by_day_of_week,
```

```
        'GENDER': q.aggregate_by_gender,
        'RACE': q.aggregate_by_race,
        'PAYOUT_LEVEL': q.aggregate_by_payout_level,
        'REMUNERATION': q.aggregate_by_remuneration
    }
```

Esse método utiliza o decorator *operation_adder*, que consulta o mapa e a requisição para invocar o método especificado da classe *Query*. Com isso, é possível adicionar automaticamente a função de agregação solicitada na requisição, garantindo uma extensibilidade simplificada. Se for necessário criar uma nova agregação, basta implementar o novo método na classe *Query* e associá-lo a uma chave no mapa de operações.

Para os outros campos da requisição o mapeamento funciona de forma analoga. Para construir a consulta final, esta função é usada:

```
def create_query(request: dict):
    q = query.Query()
    add_query_data_type(q, request) # equivalente a
        add_query_aggregation_operations para tipos de dados
    add_query_aggregation_operations(q, request)
    add_filters(q, request.get('filters')) # equivalente a
        add_query_aggregation_operations para filtros
    return q
```

Com esse esquema de mapeamento, o custo de adiconar novos filtros, agregações ou tipos de dados é reduzido e o sistema se torna mais extensível.

3.4 Estrutura do Frontend

Esta seção foca em descrever o trabalho feito na interface gráfica.

3.4.1 Filtros, agregações e gráficos

A primeira parte do trabalho foi focada na definição de um menu que pudesse ser usado tanto para configurar a visualização das métricas em gráficos quanto em mapas de calor. Após analise, foi decidido que seria seguido o padrão do HealthDashboard e do BikeScienceWeb onde há um menu lateral estreito e um painel mais extenso onde os dados são visualizados.

O menu se baseia em uma série de campos do tipo *select*. Nesses campos pode se escolher o tipo de visualização, agregação, o tipo de dados, os filtros a serem aplicados e um botão para aplicar a visualização configurada como mostra a figura 3.2. Cada campo contém uma lista de valores onde apenas um pode ser aplicado (como na figura 3.3), com exceção do campo de filtros.

O campo de filtros funciona de uma maneira um pouco diferente. Como vários filtros podem ser aplicados de uma vez, o campo aceita a seleção de multiplos valores. Além disso, toda vez que um novo valor é adicionado na seleção um novo campo especifico para

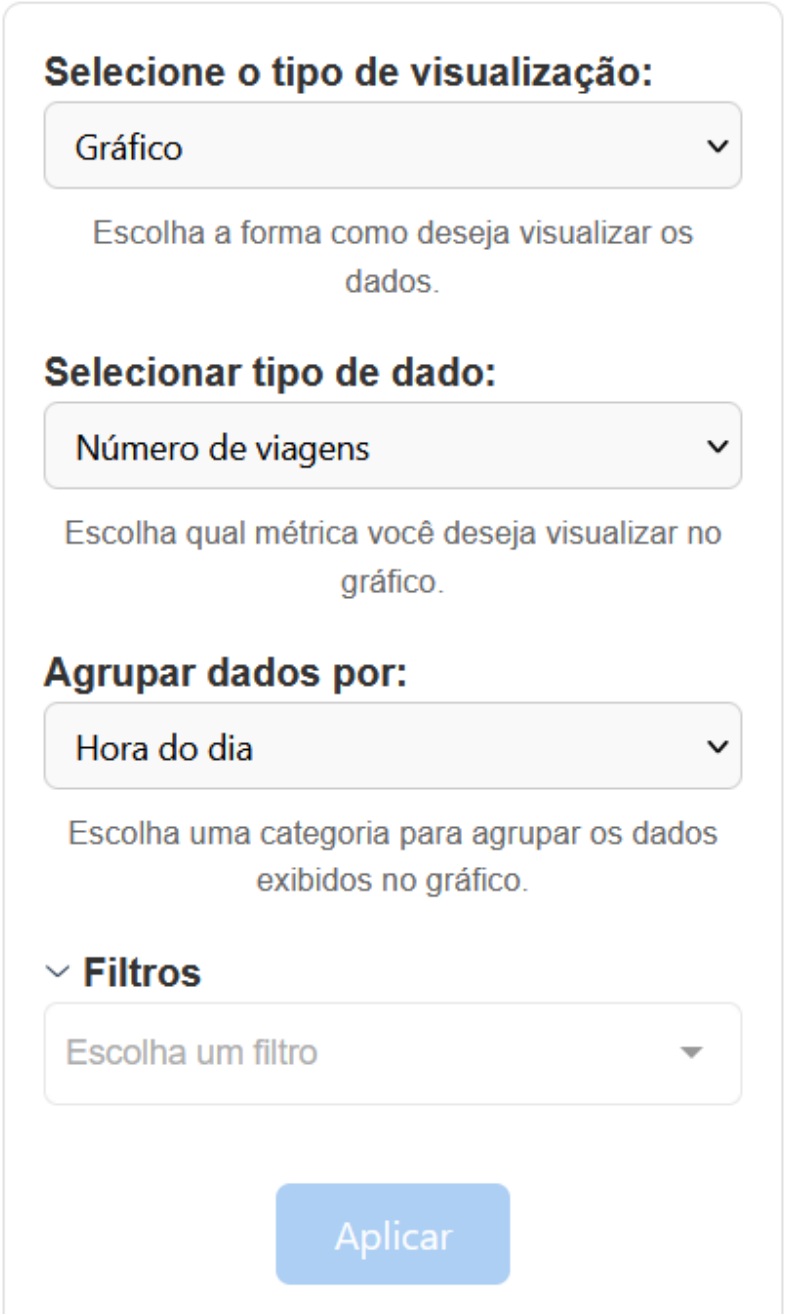

Figura 3.2: *Menu da aplicação*

o tipo de filtro é criado. Por exemplo, se o filtro de data é selecionado um componente aparece para que o usuário selecione a data final e inicial do filtro. Entretanto, se o filtro é sobre a recompensa do usuário, um slider de 0 a 1 aparece para o usuário. Um exemplo desse componente pode ser visto na figura 3.4

Para a parte de visualização dos gráficos, foi escolhida a biblioteca vue-data-ui. Essa biblioteca facilita a criação de diferentes tipos de gráficos por meio de um arquivo de configuração. Esse arquivo permite configurar o tipo de gráfico, cores, legendas, entre outros. Essa capacidade da biblioteca permitiu que diferentes formatos de gráficos fossem

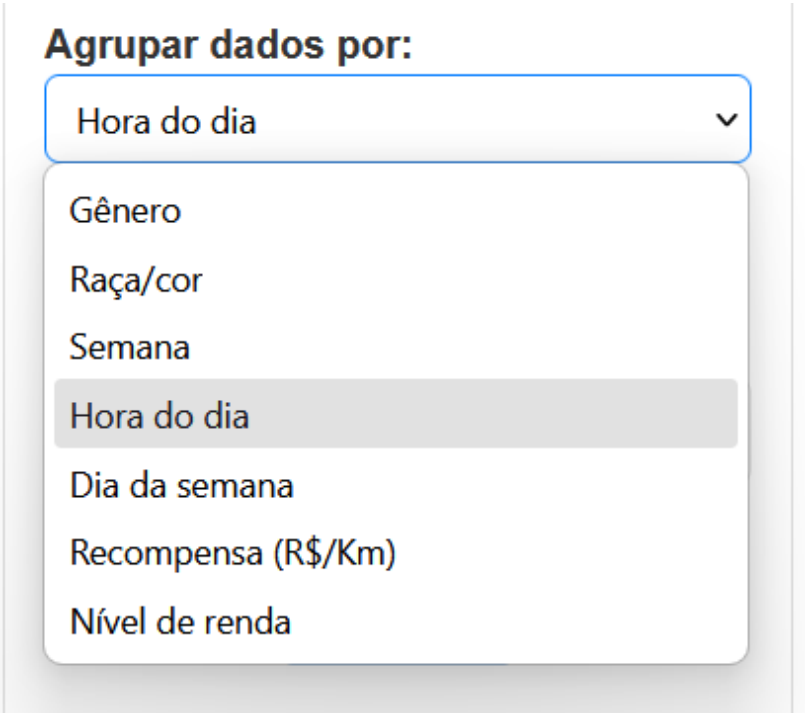

Figura 3.3: *Lista de valores de agregação*

configurados para cada agregação a fim de respeitar algumas caracteristicas dos dados. Por exemplo, para datas é melhor que haja uma angulação na visualização para que elas não se sobreponham como pode ser visto na figura 3.5

3.4.2 Mapa de calor

Para que a aplicação pudesse apresentar um mapa de calor a abordagem escolhida foi carregar um mapa a partir do Leaflet (Leaflet, 2025) e então adicionar uma camada extra por cima do mapa com o mapa de calor. A principio, foi escolhida a biblioteca Leaflet.heat (Leaflet.heat, 2024) para fazer isso. Essa biblioteca é integrada com os mapas do Leaflet e automaticamente cria a camada de calor por cima do mapa baseando-se nos dados providos pelo usuário.

Entretanto essa biblioteca não era apropriada para o caso em questão pois ela é usada para casos em que os dados são fixos. Se os dados mudam conforme o usuário interage com a aplicação, a visualização fica estranha pois a biblioteca agrega os dados para otimizar o desenho da camada. Além disso, a intensidade das cores do mapa é relacionada apenas a quantidade de pontos na localização não importando o peso da amostra.

Por estes motivos foi implementada uma estrategia própria de visualização do mapa de calor com base na implementação existente do Leaflet.heat que melhor se adequa a o caso presente.

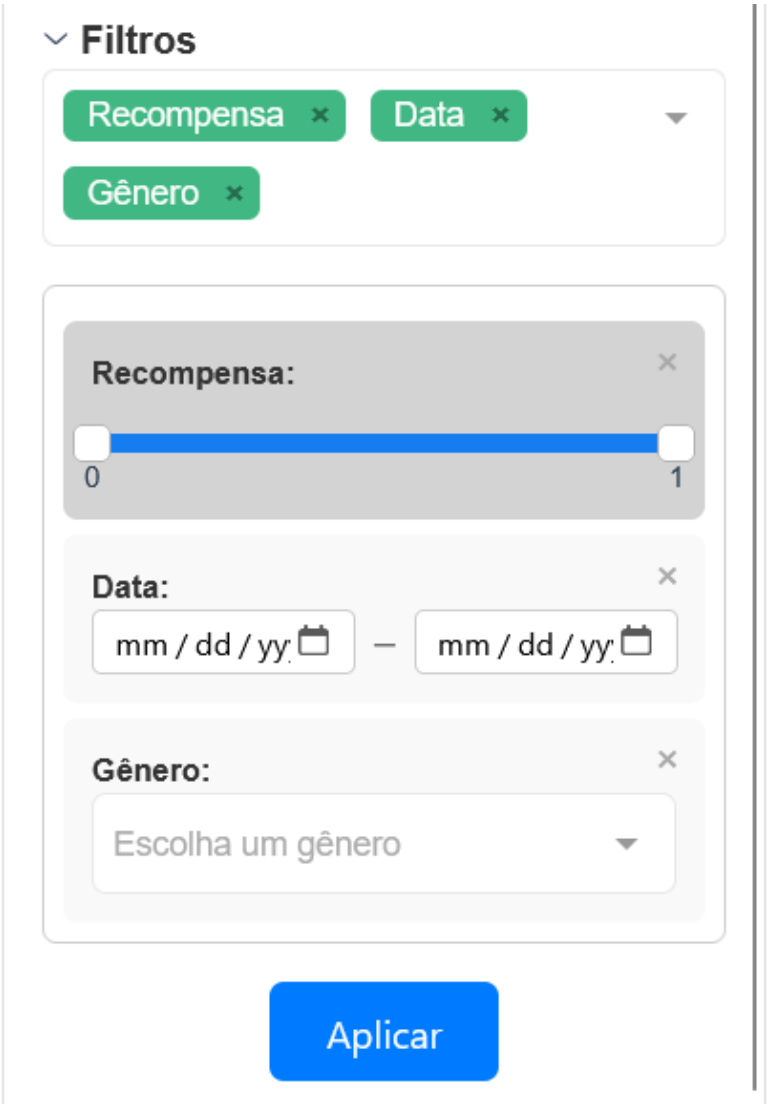

Figura 3.4: *Exemplo de uso do campo de filtro. Múltiplos valores podem ser adicionados e removidos.*

3.5 Problemas de escalabilidade

Após a criação de uma versão inicial da aplicação, o crescimento do *dataset* do BikeSP, impulsionado pelo projeto piloto, revelou diversos problemas de escalabilidade. Esta seção detalha os principais gargalos e as soluções implementadas.

3.5.1 Tamanho da resposta do servidor para visualização do mapa de calor

Dependendo da área que o usuário visualizava, o tamanho da resposta do servidor poderia passar de 1MB, o que aumentava o tempo de resposta e sobrecarregava o navegador. Duas ações foram tomadas para resolver essa limitação:

- **Compactação:** A compressão dos dados utilizando o Gzip foi ativada no servidor para todas as respostas na API de *fetch_geographic_data*. O cliente no navegador

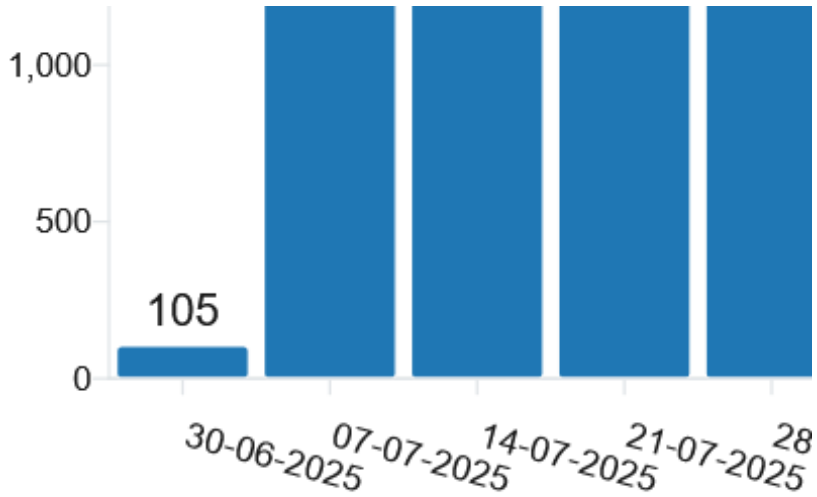

Figura 3.5: *Valores de datas inclinados para evitar sobreposição*

também foi configurado para habilitar a descompactação.

- **Resolução Ótima de Agregação:** O servidor passou a calcular a resolução ideal para a agregação dos dados. Visto que cada ponto do mapa de calor tinha um raio fixo de 15 *pixels* e o servidor recebia o nível de *zoom* do mapa renderizado no navegador, foi possível calcular a quantidade de quilômetros por *pixel* (k/p). Isso permitiu definir o nível de agregação das amostras, que passaram a ser agrupadas em quadrados de lado equivalente a $15 \times k/p$. Essa otimização limitou o número de itens enviados no vetor de resposta a aproximadamente 2.000 entradas.

Com essas ações, o tamanho médio da resposta do servidor foi reduzido para cerca de 30 KB.

3.5.2 Amostras Repetidas

Um segundo ponto de gargalo era o grande volume de amostras de localização repetidas no dados do BikeSP. A parada de ciclistas em semáforos ou a passagem de múltiplos usuários pelo mesmo local geravam amostras redundantes. Essas amostras criavam problemas de desempenho no **PostGIS**, que precisava processar e agregar dados repetidos.

Para resolver essa questão, o modelo de dados foi alterado para promover a unicidade das localizações. Em vez de utilizar apenas as tabelas *LOCATIONS* e *TRIP*, foi introduzida a tabela associativa *TRIP_LOCATION*. O objetivo dessa alteração foi garantir que a tabela *LOCATIONS* contivesse apenas localizações não repetidas, sendo agregadas às viagens individuais por meio da tabela de relacionamento *TRIP_LOCATION*.

Essa mudança resultou no seguinte esquema de dados:

```

  CREATE TABLE LOCATIONS (
     idLocation serial PRIMARY KEY,
     point_geom GEOGRAPHY(Point, 4326) NOT NULL,
```

```
    geohash TEXT
);

CREATE TABLE TRIP (
    payout money NOT NULL,
    idTrip serial PRIMARY KEY,
    idPerson integer NOT NULL,
    date timestamp NOT NULL,
    distance double preci sion,
    idOrigin integer,
    idTarget integer,
    status varchar(30) NOT NULL,
    statusReason varchar(100) NOT NULL,
    duration interval,
    payoutLevel double precision,
    meanSpeed double precision
);

CREATE TABLE TRIP_LOCATION (
    idTrip integer NOT NULL REFERENCES TRIP (idTrip),
    idLocation integer NOT NULL REFERENCES LOCATIONS (idLocation)
        ,
    point_timestamp timestamp NOT NULL,
    seq serial,
    speed double precision
);
```

Neste novo esquema, a tabela *LOCATIONS* armazena apenas o identificador, o objeto geométrico (*point_geom*) e o *geohash*. A tabela *TRIP_LOCATION* realiza o relacionamento, contendo o ID da viagem e o ID da localização, além do *timestamp* e da ordem da amostra. Essa separação permite que o PostGIS otimize a busca pelos pontos geográficos únicos, deixando a operação de junção com os dados da viagem a cargo do PostgreSQL.

A aplicação utiliza um geohash de precisão 8 como critério para determinar se dois pontos são considerados idênticos ou diferentes.

Com a implementação dessa normalização e das medidas de compactação, o tempo de resposta máximo da aplicação foi reduzido para 3 segundos.

3.6 Resultado final

Após concluídas todas as etapas de implementação, o sistema final atendeu aos requisitos propostos. A aplicação é capaz de exibir todos dados com diferentes agregações, filtros e visualizações. Além disso, os requisitos não funcionais de privacidade e desempenho foram atendidos considerando que não é possível identificar usuários particulares pela analise do painel.

Considerando esta etapa finalizada, o próximo capitulo analisa os resultados obtidos e propõe trabalhos futuros.

Capítulo 4

Conclusão

Com a finalização da implementação, este capitulo se propõe a discutir os resultados obtidos.

4.1 Análise dos resultados

Esta seção foca em analisar os resultados obtidos. Ela será dividida em duas subseções, sendo que a primeira discutirá os gráficos e a segunda o mapa de calor.

4.1.1 Analise dos gráficos

Por meio da visualização dos gráficos gerados pelo dashboard analítico é possível extrair diferentes padrões de comportamento dos usuários. Por exemplo, o grafico da figura 4.1 exibe o número de viagens por hora do dia, o que ajuda os administradores a entender qual o horário de pico de uso do aplicativo.

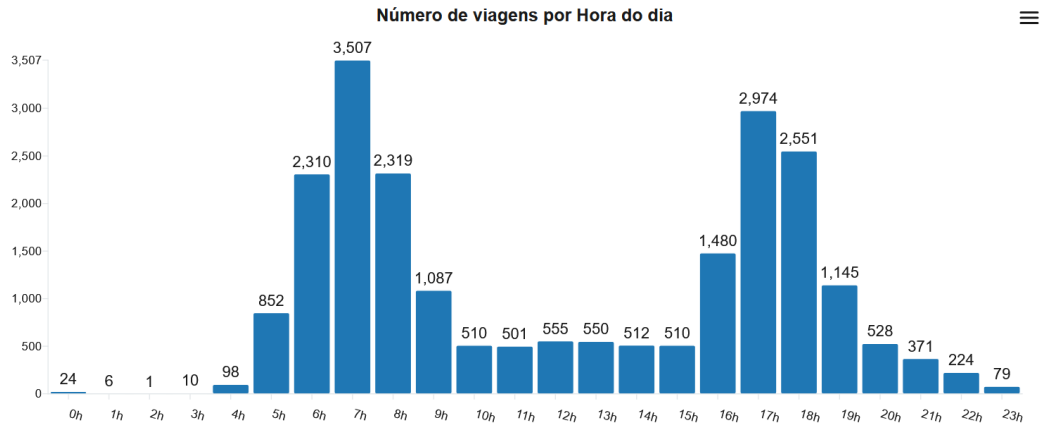

Figura 4.1: *Quantidade de viagens coletadas por hora do dia*

Uma outra métrica interessante a se analisar é a duração média das viagens. Considerando que durante todo o percurso da viagem o aplicativo se comunica com o servidor,

entender a duração média das viagens permite saber o impacto que cada usuário tem na aplicação em termos de consumo de recursos da infraestrutura. Além disso, esse número pode servir de insumo para criação de iniciativas de redução do tempo de locomoção da cidade de São Paulo.

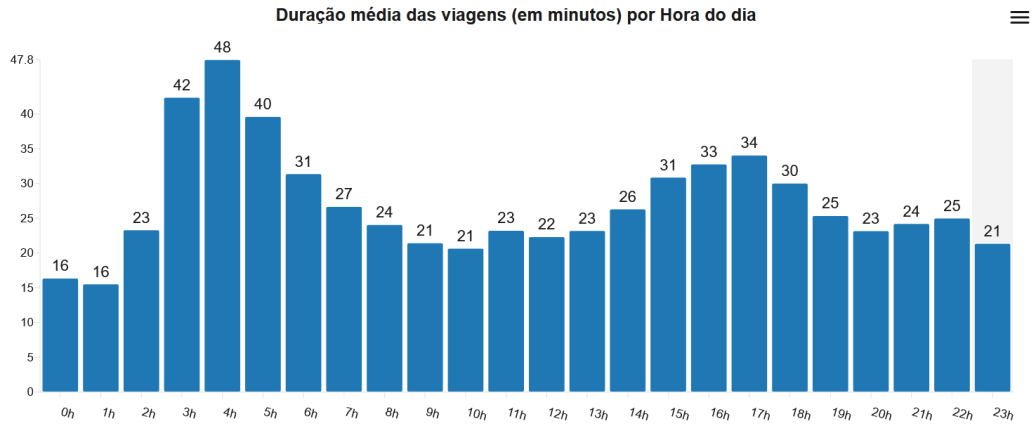

Figura 4.2: *Duração média das viagens por hora do dia*

É interessante notar que o tempo médio de viagem é menor para indivíduos com maior poder aquisitivo (veja a figura 4.3) e maior para pretos, pardos e indígenas (veja a figura 4.4). Além disso, pretos e pardos são os que tiveram, em média, as viagens mais longas como pode ser visto na figura 4.5.

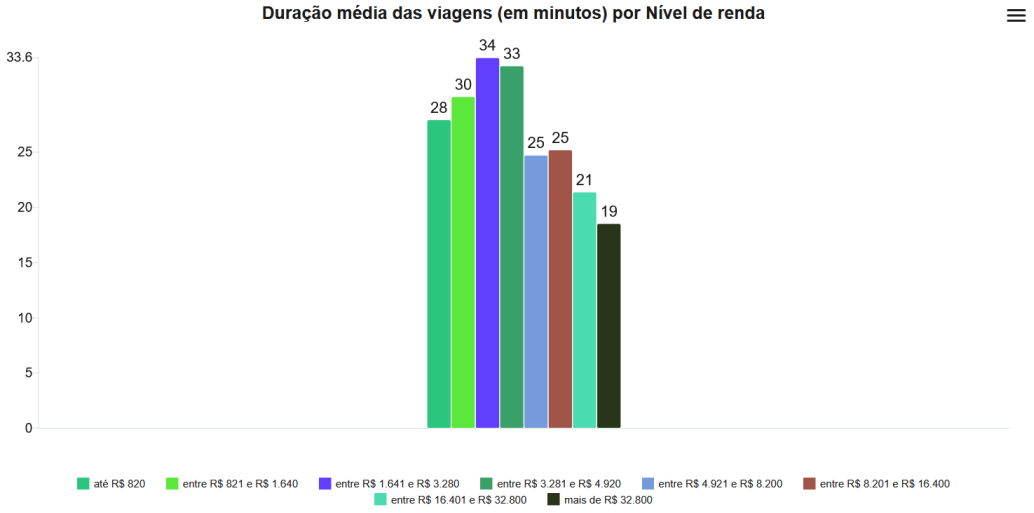

Figura 4.3: *Duração da média das viagens por renda*

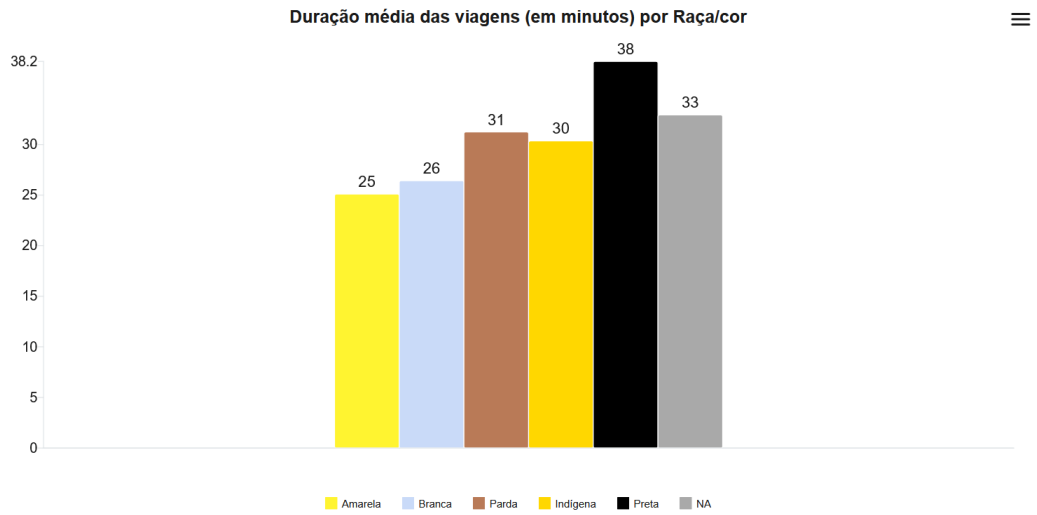

Figura 4.4: *Duração média das viagens por raça/cor*

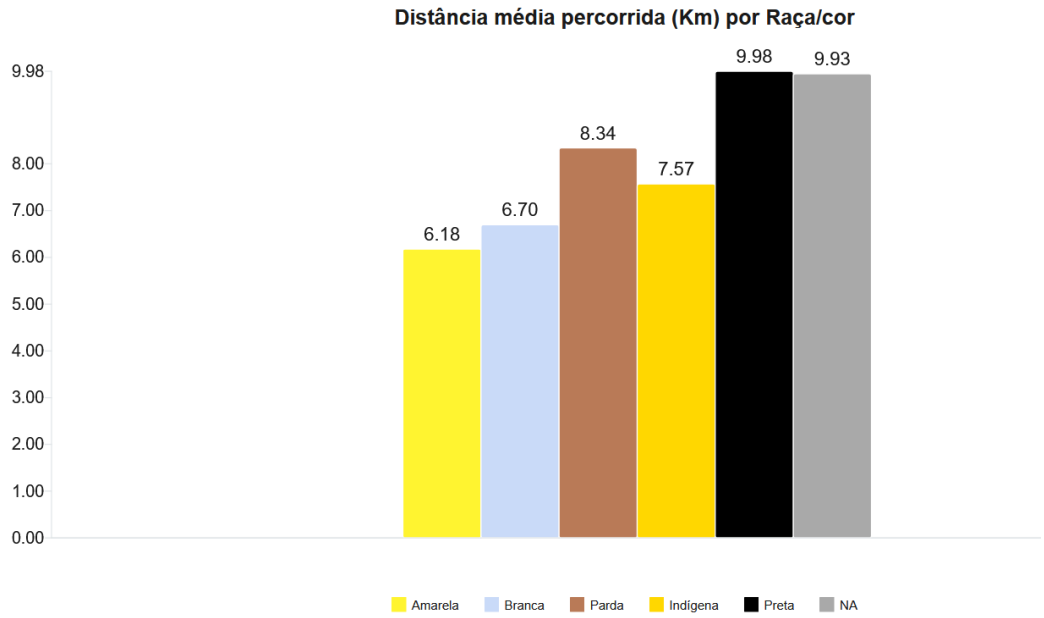

Figura 4.5: *Distância média percorrida por raça/cor*

Essas métricas evidenciam a existência de desigualdades significativas associadas a fatores socioeconômicos e raciais no uso da bicicleta como meio de transporte, sugerindo que diferentes grupos enfrentam condições distintas de deslocamento no espaço urbano. A maior duração e distância média das viagens observadas para pretos, pardos e indígenas podem estar relacionadas tanto à distribuição desigual da infraestrutura cicloviária quanto à localização residencial e às oportunidades de trabalho, indicando que a mobilidade ativa também reflete padrões estruturais da cidade. Dessa forma, os resultados reforçam a importância de análises orientadas por dados para subsidiar políticas públicas que busquem reduzir desigualdades e promover uma mobilidade urbana mais equitativa.

4.1.2 Analise do mapa de calor

No geral, o mapa de calor auxiliou na identificação de pontos com maior concentração de ciclistas na cidade de São Paulo, como é o caso da Avenida Faria Lima e da Avenida Paulista, exemplificados na figura 4.6. Nessa visualização, as linhas em vermelho representam a malha de ciclovias existente na região, evidenciando a forte sobreposição entre a infraestrutura cicloviária e as áreas de maior intensidade de uso. Essas avenidas concentram tanto uma infraestrutura adequada para o deslocamento por bicicleta quanto uma elevada densidade de atividades comerciais e de serviços, o que contribui para explicar o padrão observado.

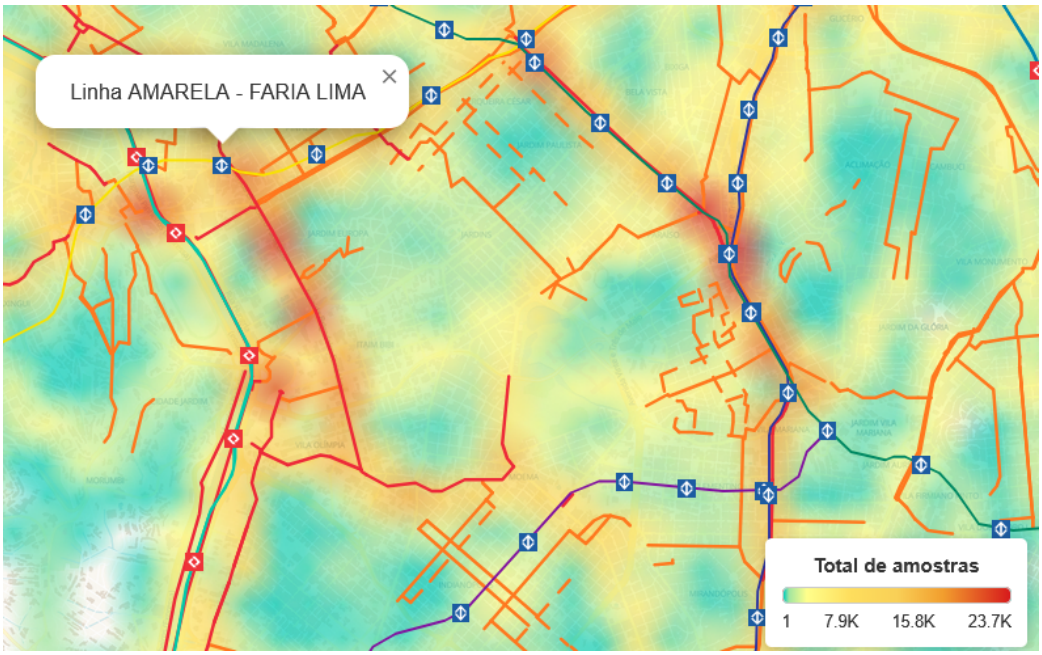

Figura 4.6: *A Avenida Paulista e a Avenida Faria Lima são as regiões com maior concentração de ciclistas na cidade*

Além disso, o mapa de calor permitiu identificar áreas que apresentam demanda significativa por deslocamentos cicloviários, mas que não dispõem de ciclovias ou ciclofaixas. Um exemplo é a Avenida Alcântara Machado, na região central da cidade, ilustrada na Figura 4.7. Apesar do fluxo elevado de ciclistas indicado pelo mapa de calor, não há infraestrutura dedicada para esse tipo de deslocamento ao longo da via.

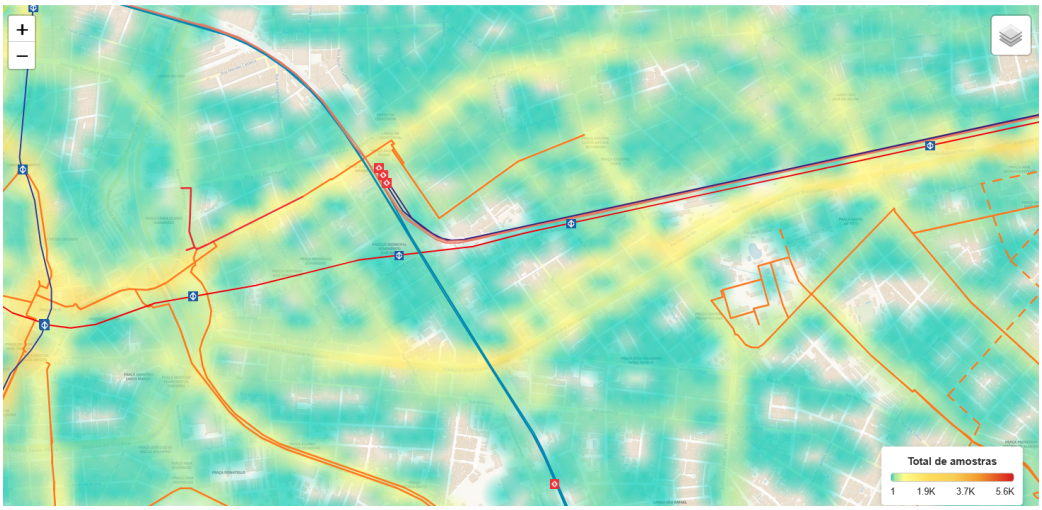

Figura 4.7: *A Avenida Alcântara Machado não possui ciclovias apesar da demanda*

4.2 Trabalhos futuros

Apesar de o sistema desenvolvido atender aos objetivos propostos neste trabalho, diversas extensões e melhorias podem ser exploradas como trabalhos futuros, tanto do ponto de vista técnico quanto da experiência do usuário.

Um primeiro ponto relevante refere-se ao desempenho da visualização de mapas de calor, que atualmente pode apresentar atrasos perceptíveis ao usuário dependendo da área selecionada e da densidade de dados, comprometendo a experiência de exploração interativa. Como trabalho futuro, seria interessante investigar técnicas mais avançadas de otimização para visualização geoespacial, com foco tanto na redução do tempo de resposta quanto na eficiência do processamento no backend. Entre as abordagens possíveis, destacam-se a implementação de mecanismos de cache para resultados frequentemente acessados, como agregações espaciais associadas a regiões e níveis de zoom mais comuns, bem como a otimização de consultas no PostgreSQL a partir da análise de planos de execução utilizando a ferramenta `EXPLAIN`. Essa análise permitiria identificar gargalos e orientar melhorias, como a reescrita de consultas e o uso mais eficiente de índices, reduzindo o volume de dados processados e transmitidos ao cliente e proporcionando uma interação mais fluida, especialmente em áreas com grande concentração de trajetos.

Além disso, um aspecto importante a ser considerado é a implementação de um sistema de permissionamento mais robusto. Atualmente, o acesso às informações é relativamente homogêneo, o que limita a possibilidade de disponibilizar dados mais sensíveis. Como trabalho futuro, seria interessante adotar um modelo de controle de acesso baseado em papéis, permitindo diferenciar usuários comuns, pesquisadores e administradores. Dessa forma, administradores poderiam ter acesso a informações mais detalhadas, como rotas individuais ou dados não anonimizados, enquanto usuários finais continuariam visualizando apenas informações agregadas. A implementação desse modelo envolveria tanto a definição clara de perfis de usuário quanto a integração de mecanismos de autenticação e autorização no backend.

Referências

[Lima 2023] Ana Yoon Faria de Lima, Fabio Kon. *Cycling promotion using financial incentives.* 2023. URL: https://thesis-bikesp.netlify.app/assets/pdfs/report.pdf (acesso em 08/12/2025) (citado na pg. 1).

[MongoDB 2024] MongoDB Inc. *MongoDB: Document Database Platform.* 2024. URL: https://www.mongodb.com (acesso em 08/12/2025) (citado na pg. 13).

[São Paulo 2016] *Lei nº 16.547, de 20 de dezembro de 2016.* Institui o Programa Bike SP no âmbito do município de São Paulo, 2016. 2016. URL: https://legislacao.prefeitura.sp.gov.br/leis/lei-16547-de-21-de-setembro-de-2016 (citado na pg. 1).

[Elasticsearch 2024] Elastic NV. *Elasticsearch: Distributed Search and Analytics Engine.* 2024. URL: https://www.elastic.co/elasticsearch (acesso em 08/12/2025) (citado na pg. 13).

[PSC 2025] PostGIS Project Steering Committee. *PostGIS Spatial Extension for PostgreSQL.* 2025. URL: https://postgis.net/ (acesso em 23/11/2025) (citado na pg. 13).

[PGDG 2025] PostgreSQL Global Development Group. *PostgreSQL Database Management System.* 2025. URL: https://www.postgresql.org/ (acesso em 23/11/2025) (citado na pg. 13).

[InterSCity 2016] Projeto InterSCity e contribuidores. *HealthDashboard: A Urban Public Health Geospatial Visualization Platform.* 2016. URL: https://interscity.org/assets/healthdashboard-VAHC-2020.pdf (acesso em 08/12/2025) (citado na pg. 3).

[Strava 2023] Strava. *Strava Global Heatmap.* Mapa de calor global gerado por atividades dos usuários. 2023. URL: https://www.strava.com/heatmap (acesso em 08/12/2025) (citado na pg. 4).

[InterSCity 2023] Thiago J. B. Pena, Higor A. de Souza, Letícia L. Lemos, Fabio Kon. *BikeScienceWeb: a tool for bicycle-related urban planning.* 2023. URL: https://repositorio.usp.br/directbitstream/93aa9acc-85bf-4558-864e-879f8637e887/3174735.pdf (acesso em 08/12/2025) (citado na pg. 4).

[Leaflet.heat 2024] Vladimir Agafonkin and contributors. *Leaflet.heat: A tiny, simple and fast heatmap plugin for Leaflet*. Biblioteca JavaScript. 2024. URL: https://github.com/Leaflet/Leaflet.heat (acesso em 23/11/2025) (citado na pg. 22).

[Leaflet 2025] Vladimir Agafonkin and contributors. *Leaflet: An Open-Source JavaScript Library for Interactive Maps*. 2025. URL: https://leafletjs.com/ (acesso em 23/11/2025) (citado na pg. 22).