MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

SISTEMA PARA MONITORAMENTO DE UTILIZAÇÃO DE APLICAÇÕES PARA A INTERNET VOLTADO AO PROJETO DE UX: BACK END

Rafaela Vieira Cabral

Relatório de Iniciação Científica do programa PIBIT, orientada pelo Dr. Fabrício Galende Marques de Carvalho

URL do documento original:

< >

INPE

São José dos Campos

2023

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

SISTEMA PARA MONITORAMENTO DE UTILIZAÇÃO DE APLICAÇÕES PARA A INTERNET VOLTADO AO PROJETO DE UX: BACK END

Rafaela Vieira Cabral

Relatório de Iniciação Científica do programa PIBIT, orientada pelo Dr. Fabrício Galende Marques de Carvalho

URL do documento original:

< >

INPE

São José dos Campos

2023

RESUMO

O presente trabalho tem como objetivo pesquisar aspectos referentes à experiência do usuário, presentes em aplicações desenvolvidas pelo INPE, na qual se faz necessário o desenvolvimento de componentes para o monitoramento do uso de interfaces pelo usuário, sendo seu foco em *back end*, incluindo a criação de um dashboard para a análise dos dados de utilização.

O projeto de pesquisa utiliza predominantemente o enfoque qualitativo e exploratório, em que o problema de monitoramento de utilização de aplicações é estudado sob diferentes perspectivas. O trabalho é baseado em revisão de literatura e análise de aplicações para a Internet, preferencialmente aquelas desenvolvidas pelo INPE, de modo a se compreender os principais aspectos que devem ser monitorados de modo a se extrair conclusões que ajudem no projeto de UX e, consequentemente, melhorem a eficácia dos serviços prestados pelo instituto à população, através de seus sistemas. Sob o ponto de vista de desenvolvimento de software, está sendo utilizada a abordagem incremental e iterativa.

Até o presente momento, foram implementados componentes para a gravação de eventos no banco de dados. Tais eventos são capturados por componentes de *front end*, que também estão sendo desenvolvidos no escopo do projeto. Os dados armazenados no banco de dados podem ser listados e filtrados por meio de consultas específicas definidas pelo usuário. Essas funcionalidades foram desenvolvidas objetivando a construção de um dashboard, a fim de permitir a análise dos dados de utilização capturados e, consequentemente, a tomada de decisões importantes com relação ao aprimoramento de determinada aplicação. É importante ressaltar que o projeto ainda está em desenvolvimento e possui aspectos a serem melhorados e adicionados até sua conclusão. As etapas a serem realizadas incluem a implementação do controle de sessão, o aprimoramento e a generalização dos filtros de busca, a criação de análises de dados de acesso e sua disponibilização no dashboard, a modelagem dos dados necessários para representar eventos associados a mapas e demais informações em interfaces mais específicas, entre outros. Além das funcionalidades a serem desenvolvidas, espera-se que o sistema possa ser testado no contexto de uma aplicação de interesse do instituto que esteja publicamente disponível ao cidadão (ex. sistema Queimadas) ou então que testes de validação e verificação em um sistema contendo tais aspectos seja efetuado.

Palavras-chave: experiência de usuário, análise de dados, jornada de usuário.

LISTA DE FIGURAS

Pág.

SUMÁRIO

1. INTRODUÇÃO

A experiência de usuário, vem a desempenhar um grande papel no desenvolvimento de aplicações tecnológicas atualmente, uma vez que irá promover a convergência entre a funcionalidade do sistema e a satisfação do usuário. É sob esse contexto que o presente projeto tem como principal propósito a investigação de diversos aspectos relacionados à experiência do usuário nas aplicações desenvolvidas pelo INPE. Com esse objetivo em mente, foi concebido um componente destinado a monitorar a interação dos usuários com as interfaces disponibilizadas, viabilizando assim uma análise minuciosa dos dados coletados.

Portanto, a primeira etapa do projeto deu-se por meio da compreensão das necessidades, expectativas e preferências dos usuários nas aplicações do INPE, após realizar essa compreensão, busca-se aprimorar a elaboração das aplicações, criando interfaces mais intuitivas e eficientes.

É de extrema importância ressaltar que o projeto tem como objetivo desenvolver componentes especializados para o monitoramento detalhado do uso das interfaces de usuário, com ênfase no backend. Os componentes devem efetuar a captura e análise dos dados de utilização da interface, para oferecer uma visão abrangente e precisa dos padrões de interação do usuário. Como forma de deixar esses dados visíveis para análise, foi realizado a criação de um dashboard a fim de promover uma otimização das aplicações, uma vez que sua mudança será estritamente baseada em dados.

Em suma, o projeto de pesquisa busca inovar a experiência de usuário nas aplicações do INPE, por meio de investigações mais amplas através de ferramentas de análise e avaliação fundamentada. Sendo assim, seu objetivo é alcançar uma experiência mais fluída e eficiente para os usuários, impulsionando a excelência tecnológica.

2. DESENVOLVIMENTO

2.1 REVISÃO DE LITERATURA

A experiência de usuário, vem desempenhando um grande papel no desenvolvimento de aplicações tecnológicas atualmente, uma vez que irá promover a convergência entre a funcionalidade do sistema e a satisfação do usuário (BASRI ET AL, 2016). Segundo a definição oficial da ISO FDIS 9241 o UX pode ser entendido como “percepções e respostas de uma pessoa que resultam do uso naquele exato momento ou/e do uso antecipado de um produto, sistema ou serviço”. Segundo Barsi (2016) essa seria uma boa definição, mas incompleta, para o autor o UX deve ser uma soma de características pragmáticas como uma visão holística para o usuário resultando em um sistema, produto ou serviço que seja atrativo e de fácil assimilação.

Para iniciar o projeto, estudos foram feitos na linguagem JavaScript para o aprendizado da mesma, uma vez que, ela será utilizada para embasamento ao utilizar a efetiva linguagem TypeScript. De acordo com Bierman (2014), o Typescript é uma extensão do Javascript com o intuito de facilitar o desenvolvimento de aplicações em larga escala.

Ele diz que o não é difícil realizar transição do Javascript para o Typescript, pois mantém idiomas de programação estabelecidos.

2.2 TÉCNOLOGIAS UTILIZADAS

Para a execução do projeto, foram utilizadas tecnologias específicas com o intuito de atender os requisitos propostos de forma eficiente. Na etapa de documentação, foi utilizado o Git e a plataforma GitHub, pois com eles é possível efetuar o controle de versão dos códigos para o desenvolvimento de programas, na qual um deles é o sistema para rastrear as mudanças feitas no código e o outro é um serviço de hospedagem, respectivamente.

A linguagem adotada foi o TypeScript, uma extensão do JavaScript com recursos adicionais de tipagem estática, na qual proporciona códigos mais robustos e seguros. É importante ressaltar que a escolha do TypeScript foi motivada também por sua biblioteca de Mapeamento Objeto Relacional (ORM) chamada TypeORM, em que simplifica a interação com o banco de dados.

Para os testes, foi escolhido o framework Jest, uma excelente ferramenta em JavaScript que permite a criação de testes automatizados para verificar o funcionamento correto das funções desenvolvidas no arquivo principal do projeto. Com ele é possível garantir a qualidade e confiabilidade do código, identificando eventuais erros ou problemas de execução.

Portanto, as escolhas das tecnologias foram feitas com base na experiência e nas melhores práticas de desenvolvimento, para realizar um projeto bem estruturado, de fácil manutenção e alto desempenho.

2.2 ESTUDOS EFETUADOS

Ao iniciar o projeto, estudos foram feitos na linguagem JavaScript para o aprendizado da mesma, uma vez que, ela será utilizada para embasamento ao utilizar a efetiva linguagem TypeScript. Portanto, foi disponibilizado pelo orientador um tutorial seguido de alguns exemplos para análise e teste, de modo que, após o estudo e execução do mesmo houvesse o atendimento a respeito da linguagem estudada e conseguisse realizar outros códigos seguindo os exemplos.

Para concretizar os conhecimentos adquiridos, foi desenvolvida algumas atividades, dentre elas pode-se citar a criação de exemplos simples, frontend e backend de tal forma que demonstre as estruturas de controle, sendo elas seleção e repetição, e os operadores aritméticos, respectivamente. As estruturas de controle são aquelas que definem a ordem em que as instruções, expressões e chamadas de funções devem ser executadas e avaliadas pelo computador, essas estruturas de controle são divididas entre duas partes, a estrutura de seleção e de repetição, onde uma irá receber uma condição dentro de um parêntese, e se for verdadeira, será executado o código dentro de suas chaves, caso contrário, irá executar outra parte do código, e a outra irá repetir as ações enquanto uma condição permanecer verdadeira, respectivamente.

Portanto, no frontend a primeira demonstração realizada foi por meio de um código executável no próprio arquivo HTML, chamado de arquivo interno, nele foi feito

códigos das estruturas de controle de repetição, o exemplo com o Do While irá executa o loop primeiro e depois irá verificar a condição, já o exemplo com For irá executar o loop apenas se a condição é verdadeira. Na Figura 1 e Figura 2 é possível verificar o código realizado para a o estudo das estruturas de controle de repetição.

Figura 1. Exemplo de frontend interno com a estrutura de repetição Do While

```
<!doctype html>
<html>
<head>
    <title> Exemplo front end interno</title>
    <meta charset="utf-8">
</head>
<body>
    <script>
        x=0
        do{
            document.write("Valores de x:" + x + "<br/>");
            ++x;
        }while(x<5);

    </script>
</body>
</html>
```

Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

Figura 2. Exemplo de frontend interno com a estrutura de repetição For

```
<!doctype html>
<html>
<head>
    <title> Exemplo front end interno</title>
    <meta charset="utf-8">
</head>
<body>
    <script>
        x=0
        for(y=0; y<5; y++){
            x++;
            document.write(["Valores para x: "] + x, "<br/>");
        }

    </script>
</body>
</html>
```

Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

Para efetuar a demonstração com as estruturas de controle de seleção, foi utilizado um arquivo HTML com um arquivo externo, presente o código de execução, nele foi feito demonstrações com as estruturas If - Else e Switch. Portanto, na Figura 3 e Figura 4 é possível ver o código realizado para os estudos das estruturas de controle de seleção.

Figura 3. Exemplo frontend externo com a estrutura de seleção If - Else

```
<!doctype html>
<html lang="pt-br">
<head>
    <title> Exemplo front end externo</title>
    <meta charset="utf-8">
    <script type='text/javascript' src='IF-ELSE.js'></script>
</head>
<body>
</body>
</html>
```

```
var numero = prompt("Digite um número:");
if (numero%2==0){
    document.write("O número escolhido é par:" + numero)
}
else{
    document.write("O número escolhido é ímpar: " + numero);
}
```

(a) Código html (b) Código JavaScript

Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

Figura 4. Exemplo frontend externo com a estrutura de seleção Switch

```html
<!doctype html>
<html lang="pt-br">
<head>
    <title> Exemplo front end externo</title>
    <meta charset="utf-8">
    <script type='text/javascript' src='Switch.js'></script>
</head>
<body>
</body>
</html>
```

```javascript
var pessoa= prompt("Digite seu nome nome:");
switch(pessoa){
    case "Fabricio":
        document.write("Ola professor " + pessoa);
        break;
    case "Rafaela":
        document.write("Bem vinda novamente " + pessoa + "<br/>");
        break;
    default:
        document.write ("Ola " + pessoa);

}
```

(a) Código html (b) Código JavaScript

Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

Iniciando os estudos de backend, foi realizado um código para exemplificar as estruturas de controle e os operadores aritméticos, na qual ele irá realizar um loop enquanto o valor adotado pela variável estiver sendo verdadeira, dentro do loop irá realizar operações com o número escolhido, conforme é mostrado na Figura 5.

Figura 5. Exemplo backend de estruturas de controle e operadores aritméticos

```javascript
var numero = 0;
console.log("O valor inicial do número 1 é: " + numero);
i=0;
while(i<10){
    i++;
    if (numero > 5){
        numero-=2;
   }
    else if(numero<i){
        numero+=2;
    }
    else if(numero%2 ==0){
        numero/=2;
    }
    else{
        numero*=2;
    }


}
console.log("Número de repetições: " + i);
console.log("Valor final do número: " + numero);
```

Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

Após a familiarização com o JavaScript, foi desenvolvido atividades ligadas ao projeto, primeiro foi feita a criação de um web app que quando é inicializado faz a exibição de um nome na tela, para realizá-lo foi utilizado o framework express pois permite a criação de servidores HTTP, conforme mostrado na Figura 6.

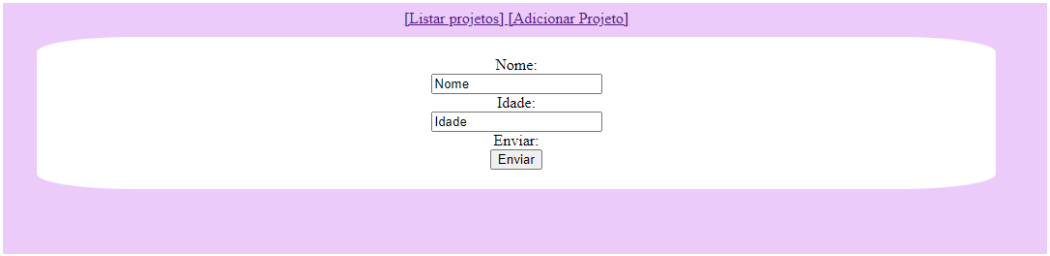
Figura 6. Exemplo de Web App utilizando express para a exibição de nome na tela

```
const express = require('express');
const app = express();

app.get("/", pagina);

app.listen(8080, conectar);


async function pagina(req, res){
    res.send("Rafaela");
}

function conectar(){
    console.log("Servidor iniciado na porta 8080: http://localhost:8080");
}
```

Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

Além disso, para o estudo do armazenamento de dados enviados por uma aplicação, foi realizado exemplo em web app na qual contém uma página de formulário, para o usuário digitar as informações requisitadas e envia-las para o backend que irá armazenar em um banco de dados as informações inseridas. Na Figura 7 é possível observar o formulário com os campos para preenchimento e após isso salva-las no backend.

Figura 7. Exemplo de Web App com formulário para armazenamento no backend



Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

No exemplo, possuí a opção de adicionar projeto e listar os projetos, no backend foi realizada funções para efetuar ambas as solicitações, a listagem é feita com base nas informações adicionadas no banco e será mostrada na tela para o usuário. Na Figura 8 é possível observar o código das funções de listagem e adição de projetos.

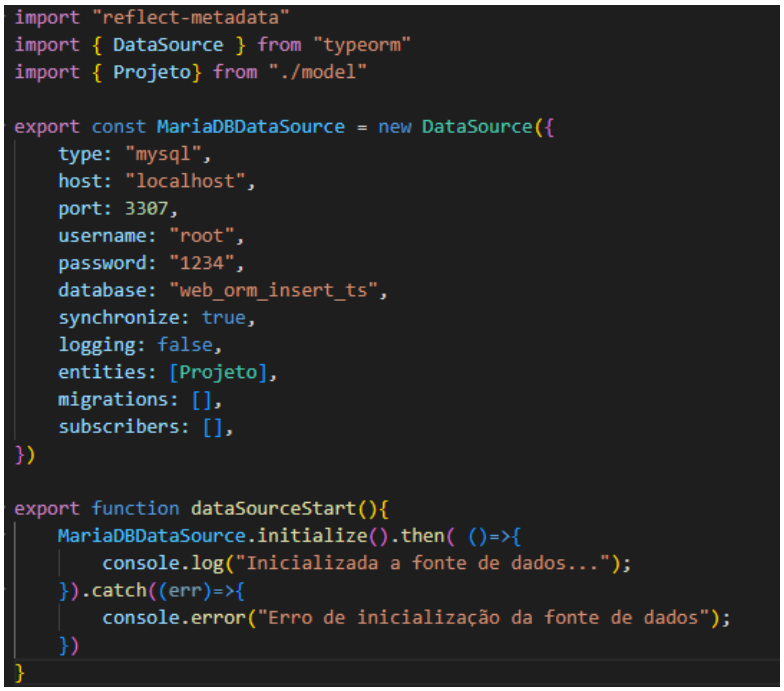
Figura 8. Código das funções para adição e listagem de projeto no formulário

```
function addProjectHandler(req, res) {
    let novo_projeto = new Projeto();
    novo_projeto.nome = req.body.nome;
    novo_projeto.idade = req.body.idade;
    service.insert(novo_projeto);
    res.render('adicionar_projeto_confirm.ejs', { projeto: novo_projeto });
}

async function listProjectHandler(req, res) {
    let projetos = await service.listAll();
    console.log(projetos);
    res.render('listar_projetos.ejs', { lista_projetos: projetos });
}
```

Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

Entretanto, no código pode ser realizada a alteração do banco de dados pela qual o usuário preferir e isso se dá pelo fato da utilização de um Mapeamento Objeto Relacional (ORM), conhecido como TypeORM, já que ele irá definir o modo como os dados serão mapeados, acessados e gravados. Dessa forma ele irá diminuir o tempo de desenvolvimento, pois não será necessário ficar fazendo outro código para trocar a configuração. Na Figura 9 é possível ver como é feita a configuração da base de dados escolhida com o TypeORM.

Figura 9. Exemplo de configuração da base de dados com TypeORM no arquivo DataSource



```
import "reflect-metadata"
import { DataSource } from "typeorm"
import { Projeto} from "./model"

export const MariaDBDataSource = new DataSource({
    type: "mysql",
    host: "localhost",
    port: 3307,
    username: "root",
    password: "1234",
    database: "web_orm_insert_ts",
    synchronize: true,
    logging: false,
    entities: [Projeto],
    migrations: [],
    subscribers: [],
})

export function dataSourceStart(){
    MariaDBDataSource.initialize().then( ()=>{
        console.log("Inicializada a fonte de dados...");
    }).catch((err)=>{
        console.error("Erro de inicialização da fonte de dados");
    })
}
```

Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

Portanto, após um estudo aprofundado em JavaScript, preparando para o uso de TypeScript, foram desenvolvidos exemplos de estruturas de controle no frontend e backend, bem como um web app com armazenamento de dados, para que houvesse o avanço necessário no desenvolvimento do projeto. Além disso, vale ressaltar que o uso do TypeORM permitiu a flexibilidade e agilidade no desenvolvimento da interação com o banco de dados.

2.3 EXECUÇÃO DO PROJETO

Após todo o processo de aprendizado das linguagens de programação, foi iniciado definitivamente os exemplos que tivessem relevância para o projeto. Portanto, o primeiro exemplo que feito foi de inserir informações no banco de dados e lista-los utilizando em um arquivo JSON. Na Figura 10 é possível observar o código que é feita adição das informações no banco de dados, na Figura 11 é possível observar o código que irá fazer a listagem das informações na tela, na Figura 12 é possível visualizar a forma que é feita o método de adição no banco de dados sem a necessidade de um frontend, foi utilizado o postman para isso, já na Figura 13 é possível ver a execução da listagem na tela.

Figura 10. Código da função de adição de informações no banco de dados

```
export async function addEventtHandler(req,res){
    console.log("Requisição de inserção recebida..");
    let novo_evento = new Eventos();
    for(let key in req.body){
        novo_evento[key] = req.body[key];
    }
    await service.insert(novo_evento);
    let novo_evento_i = JSON.stringify(novo_evento);
    res.setHeader('Content-Type', 'application/json');
    res.end(novo_evento_i);
}
```

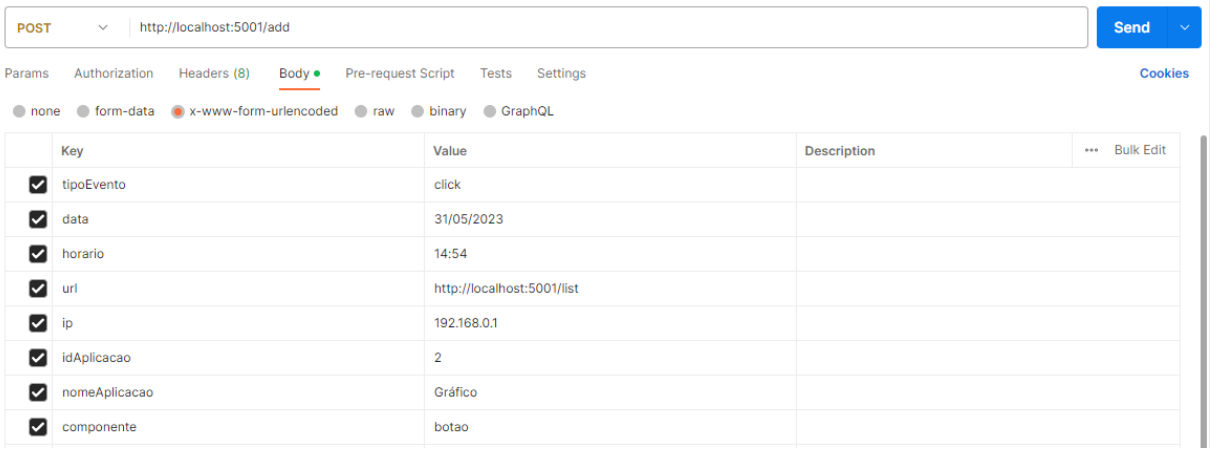
Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

Figura 11. Código da função de listagem das informações presentes no banco de dados

```
export async function listEventHandler(req, res){
    console.log("Requisição de listagem recebida.");
    let eventos = await service.listAll();
    let json_evt_list = JSON.stringify(eventos);
    res.setHeader('Content-Type', 'application/json');
    res.end(json_evt_list);
}
```

Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

Figura 12. Método de adição no banco de dados com o postman sem a necessidade de frontend



Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

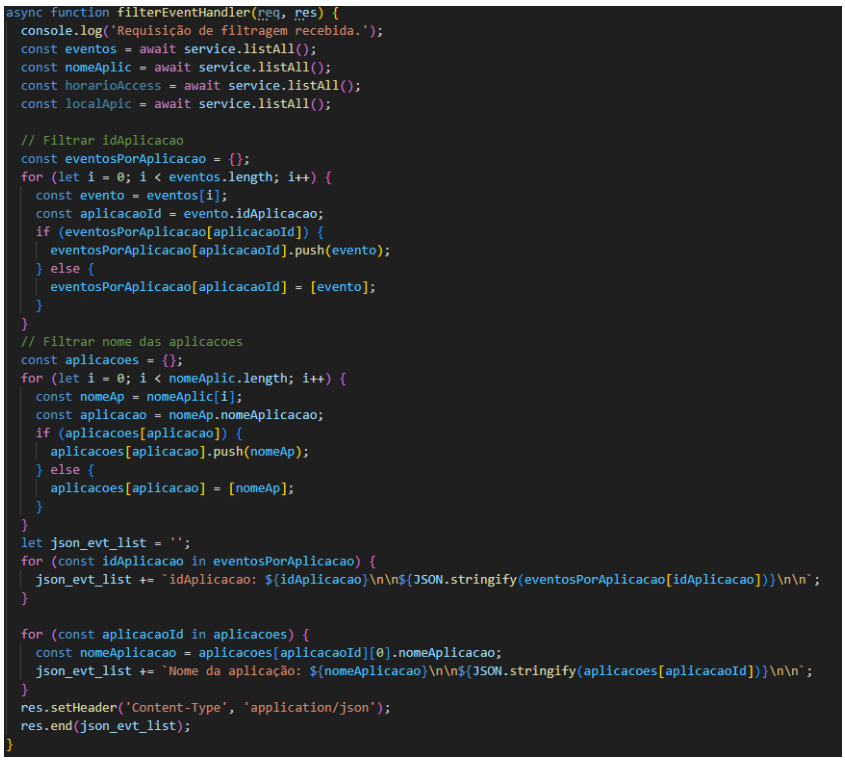
Figura 13. Listagem dos eventos presentes no banco de dados em um JSON

Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

Avançando no desenvolvimento, inicialmente foi criado uma função para efetuar a filtragem dos dados que foram inseridos no banco de dados de acordo com o que fosse solicitado pelo usuário, entretanto esses dados filtrados são retornados em um único json, acabando por ser ineficiente ao projeto. Na Figura 14 é possível analisar o código

da filtragem dos dados presentes no banco e na Figura 15 é possível analisar os resultados obtidos após a filtragem.

Figura 14. Código inicial de filtragem de eventos retornados em um único json



```
async function filterEventHandler(req, res) {
  console.log('Requisição de filtragem recebida.');
  const eventos = await service.listAll();
  const nomeAplic = await service.listAll();
  const horarioAccess = await service.listAll();
  const localApic = await service.listAll();

  // Filtrar idAplicacao
  const eventosPorAplicacao = {};
  for (let i = 0; i < eventos.length; i++) {
    const evento = eventos[i];
    const aplicacaoId = evento.idAplicacao;
    if (eventosPorAplicacao[aplicacaoId]) {
      eventosPorAplicacao[aplicacaoId].push(evento);
    } else {
      eventosPorAplicacao[aplicacaoId] = [evento];
    }
  }
  // Filtrar nome das aplicacoes
  const aplicacoes = {};
  for (let i = 0; i < nomeAplic.length; i++) {
    const nomeAp = nomeAplic[i];
    const aplicacao = nomeAp.nomeAplicacao;
    if (aplicacoes[aplicacao]) {
      aplicacoes[aplicacao].push(nomeAp);
    } else {
      aplicacoes[aplicacao] = [nomeAp];
    }
  }
  let json_evt_list = '';
  for (const idAplicacao in eventosPorAplicacao) {
    json_evt_list += `idAplicacao: ${idAplicacao}\n\n${JSON.stringify(eventosPorAplicacao[idAplicacao])}\n\n`;
  }

  for (const aplicacaoId in aplicacoes) {
    const nomeAplicacao = aplicacoes[aplicacaoId][0].nomeAplicacao;
    json_evt_list += `Nome da aplicação: ${nomeAplicacao}\n\n${JSON.stringify(aplicacoes[aplicacaoId])}\n\n`;
  }
  res.setHeader('Content-Type', 'application/json');
  res.end(json_evt_list);
}
```

Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

Figura 15. Resultado da filtragem de eventos inicial retornada em um único json

localhost:5001/filter

idAplicacao: 1

[{"id":15,"idAplicacao":1,"nomeAplicacao":"tabela","tipoEvento":"click","data":"31/05/2023","horario":"14:54","url":"http://localhost:5001/list","ip":"192.168.0.1","componente":"botao"},
{"id":16,"idAplicacao":1,"nomeAplicacao":"tabela","tipoEvento":"click","data":"31/05/2023","horario":"14:54","url":"http://localhost:5001/list","ip":"192.168.0.1","componente":"botao"}]

idAplicacao: 2

[{"id":17,"idAplicacao":2,"nomeAplicacao":"Gráfico","tipoEvento":"click","data":"31/05/2023","horario":"14:54","url":"http://localhost:5001/list","ip":"192.168.0.1","componente":"botao"}]

Nome da aplicação: tabela

[{"id":15,"idAplicacao":1,"nomeAplicacao":"tabela","tipoEvento":"click","data":"31/05/2023","horario":"14:54","url":"http://localhost:5001/list","ip":"192.168.0.1","componente":"botao"},
{"id":16,"idAplicacao":1,"nomeAplicacao":"tabela","tipoEvento":"click","data":"31/05/2023","horario":"14:54","url":"http://localhost:5001/list","ip":"192.168.0.1","componente":"botao"}]

Nome da aplicação: Gráfico

[{"id":17,"idAplicacao":2,"nomeAplicacao":"Gráfico","tipoEvento":"click","data":"31/05/2023","horario":"14:54","url":"http://localhost:5001/list","ip":"192.168.0.1","componente":"botao"}]

Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

Após o entendimento de como deveria funcionar a filtragem dos dados e com objetivo de aprimorar o código tornando o eficiente de fato, foi utilizado o método de queryString para facilitar essa filtragem, uma vez que o usuário irá precisar apenas inserir na url do navegador o que se deseja filtrar, o que foi passado para ela irá para o backend que irá tratar a requisição e retornar as informações desejadas. Portanto, na Figura 16 é possível ver o código da filtragem com queryString e as informações presentes e na Figura 17 é possível analisar os dados que foram filtrados após a solicitação.

Figura 16. Código para filtragem de eventos por meio de queryString

```
export async function filterEventHandler(req, res) {
  console.log('Requisição de filtragem recebida.');

  const filters = {};
  const queryParams = req.query;

  for (const key in queryParams) {
    if (queryParams.hasOwnProperty(key)) {
      if (key === 'count') {
        continue;
      }
      filters[key] = queryParams[key];
    }
  }

  const dataSource = MariaDBDataSource;
  const eventosRepository = dataSource.getRepository(Eventos);
  const eventos = await eventosRepository.find({ where: filters });
  const quantidadeEventos = eventos.length;

  res.setHeader('Content-Type', 'application/json');
  res.end(JSON.stringify({ eventos, quantidadeEventos }));
}
```

Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

Figura 17. Resultado da filtragem de eventos por queryString

Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

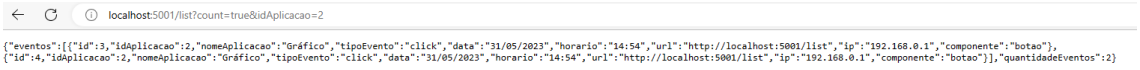
Com a função retornando o resultado desejado, foi proposto para que a filtragem de eventos estivesse implementada na função de listagem, uma vez que a filtragem deverá retornar os valores desejados pelo usuário na queryString. Para que isso seja feito de forma eficiente, no código foi implementado um comando If-Else para quando o usuário passar o comando na queryString "/list?count=true" ele faz a filtragem com as propriedades e valores desejados pelo usuário, entretanto, se o usuário passar o comando "/list?count=false" ou apenas "/list", ele irá listar todos os eventos presentes no banco de dados. Portanto, dessa maneira o código se torna eficiente e útil para o projeto. Na Figura 18 está o novo código para a função de listagem com a filtragem de informações no banco de dados, na Figura 19 a listagem com os dados filtrados e na Figura 20 a listagem sem os dados filtrados.

Figura 18. Código da função de listagem com comando para filtragem dos dados

```
export async function listEventHandler(req, res) {
  const queryParams = req.query;

  if (queryParams.count === 'true') {
    const filters = {};
    for (const key in queryParams) {
      if (key !== 'count') {
        filters[key] = queryParams[key];
      }
    }
    console.log("Requisição de filtragem recebida.");
    const dataSource = MariaDBDataSource;
    const eventosRepository = dataSource.getRepository(Eventos);
    const eventos = await eventosRepository.find({ where: filters });
    const quantidadeEventos = eventos.length;

    res.setHeader('Content-Type', 'application/json');
    res.end(JSON.stringify({ eventos, quantidadeEventos }));
  } else {
    console.log("Requisição de listagem recebida.");
    const eventos = await service.listAll();
    const json_evt_list = JSON.stringify(eventos);
    res.setHeader('Content-Type', 'application/json');
    res.end(json_evt_list);
  }
}
```

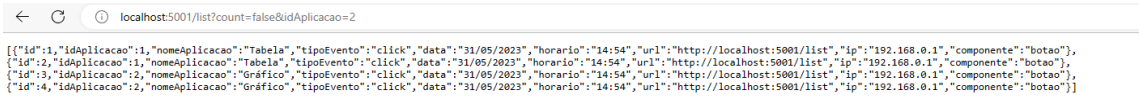
Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

Figura 19. Resultado da queryString com count adotado como true na rota de listagem



Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

Figura 20. Resultado da queryString com count adotado como false na rota de listagem



Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

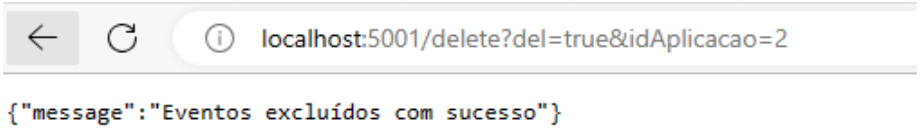
Avançando com os exemplos, foi feito um código para que o usuário consiga apagar um evento pela queryString de acordo com os parâmetros que foram passados, a exclusão irá funcionar de fato apenas se o usuário passar a rota “/delete?del=true” caso contrário, não terá exclusão. Após executar o comando, irá aparecer uma mensagem indicando que o evento foi excluído com sucesso. Na Figura 21 é possível ver o código da função que é responsável por efetuar a exclusão por meio da queryString, na Figura 22 é possível verificar a mensagem exibida no navegador após a exclusão das informações e na Figura 23 o antes e depois do banco de dados após a ação ser realizada.

Figura 21. Código da função para exclusão do evento no banco de dados

```
export async function deleteEventHandler(req, res) {
  const queryParams = req.query;

  if (queryParams.del === 'true') {
    const filters = {};
    for (const key in queryParams) {
      if (key !== 'del') {
        filters[key] = queryParams[key];
      }
    }

    console.log("Requisição de exclusão recebida.");

    const dataSource = MariaDBDataSource;
    const eventosRepository = dataSource.getRepository(Eventos);
    const eventos = await eventosRepository.find({ where: filters });

    for (const evento of eventos) {
      await eventosRepository.remove(evento);
    }

    res.setHeader('Content-Type', 'application/json');
    res.end(JSON.stringify({ message: 'Eventos excluídos com sucesso' }));
  } else {
    console.log("Requisição inválida para exclusão.");
    res.status(400).json({ error: 'Requisição inválida' });
  }
}
```

Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

Figura 22. Resultado da ação de exclusão por queryString



Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

Figura 23. Banco de Dados antes e depois da ação de exclusão das informações

```
MariaDB [eventos]> select * from eventos;
+----+-------------+--------------+-----------+------------+---------+--------------------------+-------------+-----------+
| id | idAplicacao | nomeAplicacao | tipoEvento | data       | horario | url                      | ip          | componente |
+----+-------------+--------------+-----------+------------+---------+--------------------------+-------------+-----------+
|  1 |           1 | Tabela        | click      | 31/05/2023 | 14:54   | http://localhost:5001/list | 192.168.0.1 | botao      |
|  2 |           1 | Tabela        | click      | 31/05/2023 | 14:54   | http://localhost:5001/list | 192.168.0.1 | botao      |
|  3 |           2 | Gráfico       | click      | 31/05/2023 | 14:54   | http://localhost:5001/list | 192.168.0.1 | botao      |
|  4 |           2 | Gráfico       | click      | 31/05/2023 | 14:54   | http://localhost:5001/list | 192.168.0.1 | botao      |
+----+-------------+--------------+-----------+------------+---------+--------------------------+-------------+-----------+
4 rows in set (0.000 sec)

MariaDB [eventos]> select * from eventos;
+----+-------------+--------------+-----------+------------+---------+--------------------------+-------------+-----------+
| id | idAplicacao | nomeAplicacao | tipoEvento | data       | horario | url                      | ip          | componente |
+----+-------------+--------------+-----------+------------+---------+--------------------------+-------------+-----------+
|  1 |           1 | Tabela        | click      | 31/05/2023 | 14:54   | http://localhost:5001/list | 192.168.0.1 | botao      |
|  2 |           1 | Tabela        | click      | 31/05/2023 | 14:54   | http://localhost:5001/list | 192.168.0.1 | botao      |
+----+-------------+--------------+-----------+------------+---------+--------------------------+-------------+-----------+
2 rows in set (0.000 sec)
```

Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

Com o objetivo de facilitar para desenvolvedores a manutenção de possíveis erros futuros na web service, scripts de teste foram sendo desenvolvidos para que as funções desenvolvidas no arquivo principal fossem testadas. Inicialmente foi feito apenas dois casos de teste, já que o arquivo principal possuía apenas duas funções, uma para listagem e outra para adição. Após a criação de outras funções no arquivo principal, o script de teste recebeu 3 acréscimos para casos de teste, totalizando assim 5 casos de teste. Na Figura 24 é possível analisar o arquivo com script de teste inicial das funções

de adição e listagem presentes no arquivo principal e na Figura 25 é possível ver os seus resultados exibidos no console após a execução dos casos de teste no console da aplicação.

Figura 24. Script de teste inicial das funções de adição e listagem do arquivo principal

```
const axios = require ('axios');


describe('Testes da API', () => {
    test('Adicionar Eventos', async () => {
      const novoEvento = {
        nome: 'Evento Teste',
        local: 'Click',
        idAplicacao: 1,
        nomeAplicacao: 'Mapa',
        data: '2023-05-05',
        horario: '15:00',
        url: 'http://localhost:5001/list',
        ip: '192.168.0.1'
      };
      const response = await axios({
        method: 'post',
        url: 'http://localhost:5001/add',
        data: novoEvento,
        headers: { 'Content-Type': 'application/x-www-form-urlencoded' }
      })
      expect(response.status).toBe(200);
      expect(response.data).toBeDefined();
      console.log(response.data);
    });
    test('Listar Eventos', async () => {
        const response = await axios.get('http://localhost:5001/list');
        expect(response.status).toBe(200);
        expect(response.data).toBeDefined();
        console.log(response.data);
        console.log(response.data.length);
        console.log(response.data[response.data.length - 1]);
    });

});
```

Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

Figura 25. Resultado do script de teste inicial exibido no console.

```
 PASS  test/app.test.js
  Testes da API
    √ Adicionar Eventos (106 ms)
    √ Listar Eventos (11 ms)

  console.log
    5

      at Object.log (test/app.test.js:31:17)

  console.log
    {
      id: 7,
      idAplicacao: 1,
      nomeAplicacao: 'Mapa',
      local: 'Click',
      data: '2023-05-05',
      horario: '15:00',
      url: 'http://localhost:5001/list',
      ip: '192.168.0.1'
    }

      at Object.log (test/app.test.js:32:17)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        1.381 s, estimated 2 s
Ran all test suites.
```

Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

Sendo assim, até o presente momento essas representam as etapas significativas alcançadas ao longo da evolução do projeto. Portanto, elas abrangem desde a implementação das funcionalidades essenciais que seriam a inclusão de dados no banco até a criação de filtros avançados e a capacidade de exclusão seletiva dos dados.

Vale ressaltar que a melhoria do código para a utilização de queryString nos filtros de busca contribuiu para o aprimoramento da eficiência e a flexibilidade do sistema. Além disso, a inclusão de scripts de teste também desempenharam um papel crucial para o projeto, garantindo o bom funcionamento das funções desenvolvidas e possibilitando futuras melhorias ou manutenções no projeto.

2.4 CRIAÇÃO DO BANCO DE DADOS

Para a confecção do banco de dados foi utilizado o TypeORM, como dito anteriormente, já que permite trabalhar com banco de dados relacionais usando a abordagem orientada a objetos ao invés de ter que lidar com scripts SQL.

Portanto, o banco de dados foi projetado para fazer o armazenamento dos dados conforme iria ser recebido do frontend. Primeiramente, para casos de estudo foi feito um banco de dados simples com informações de acordo com o que era esperado para receber do frontend, na Figura 26 mostra as informações iniciais que se desejava receber do frontend.

Figura 26. Banco de dados inicial com informações básicas para recebimento do frontend

Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

A primeira modelagem do banco foi apenas para testar se estava fazendo as operações desejadas, como adição e listagem. Após o sucesso na primeira etapa, foi dado o andamento no projeto, no caminho foi analisada a importância da presença de alguns dados que deveriam ser armazenados e a inutilidade do armazenamento de alguns,

sendo assim, na Figura 27 é possível visualizar a nova modelagem do banco com a presença dos novos campos para armazenagem dos dados.

Figura 27. Modelagem do banco de dados após análise de informações necessárias

```
import { Entity, PrimaryGeneratedColumn, Column } from "typeorm"

@Entity()
export class Eventos {
    @PrimaryGeneratedColumn()
    id: number

    @Column()
    appNome: string

    @Column()
    appFileName: string

    @Column()
    elementName: string

    @Column()
    idElement: string

    @Column()
    classElement: string

    @Column()
    insertValue: string

    @Column()
    dateBack: string

    @Column()
    timeBack: string
}
```

Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

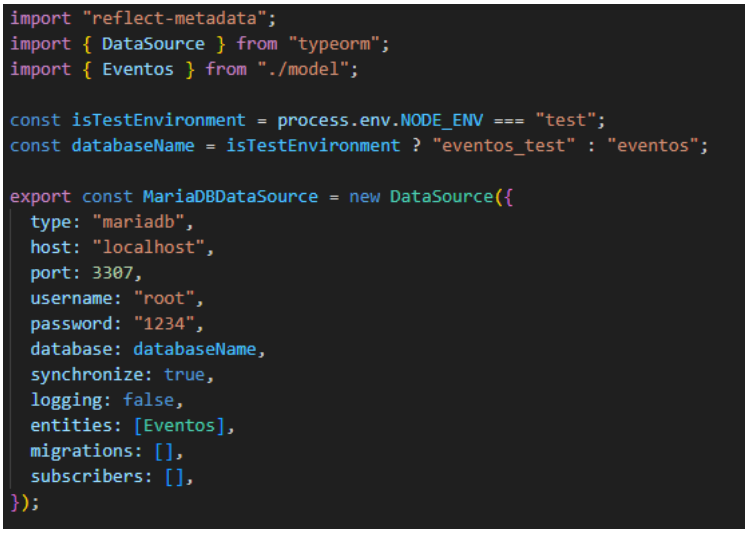
Portanto, essa é a parte do TypeORM onde será feita a criação das colunas no banco de dados em que o arquivo foi chamado de model, porém existe outros dois arquivos para a criação do banco, chamado de service e data_source, o arquivo service é onde existe as funções que irão tratar exatamente com o banco de dados, seja para inserção, listagem e remoção de dados. Já no arquivo data_source é onde será feita a configuração do banco, o arquivo inicial do data_source pode ser visualizado na Figura 9, porém quando foi visto a necessidade da presença dos casos de teste no projeto, o arquivo data_source teve que ser analisado, pois houve a necessidade de mudar automaticamente o nome da base de dados na hora de execução dos scripts de teste, ou seja, enquanto o teste for inicializado normalmente irá executar com uma base de dados x, quando for executado como script de teste sua base de dados deverá mudar automaticamente para uma com o sufixo _test, como exemplo eventos_test. Dessa forma, na Figura 28 é possível analisar o arquivo service com as funções que irão executar o tratamento no banco de dados, já na Figura 29 é possível ver o arquivo data_source atualizado para executar a mudança na base de dados ao inicializar como script de teste.

Figura 28. Arquivo service com funções para tratamento no banco de dados

```
import { MariaDBDataSource } from "./data_source";
import {Eventos} from "./model";

export class Service{
    start(){
            MariaDBDataSource.initialize().then( ()=>{
                console.log("Inicializada a fonte de dados...");
            }).catch((err)=>{
                console.error("Erro de inicialização da fonte de dados!!");
            })
    }
    async insert(eventos: Eventos,){
        await MariaDBDataSource.manager.save(eventos);
    }
    async listAll(){
       let list = await MariaDBDataSource.manager.find(Eventos);
       return list;
    }
    async remove(evento: Eventos) {
        await MariaDBDataSource.manager.remove(evento);
      }
    async resetDatabase() {
        await MariaDBDataSource.dropDatabase();
        await MariaDBDataSource.synchronize();
    }
}
```

Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

Figura 29. Arquivo data_source com a mudança da base de dados ao executar script de teste



```
import "reflect-metadata";
import { DataSource } from "typeorm";
import { Eventos } from "./model";

const isTestEnvironment = process.env.NODE_ENV === "test";
const databaseName = isTestEnvironment ? "eventos_test" : "eventos";

export const MariaDBDataSource = new DataSource({
  type: "mariadb",
  host: "localhost",
  port: 3307,
  username: "root",
  password: "1234",
  database: databaseName,
  synchronize: true,
  logging: false,
  entities: [Eventos],
  migrations: [],
  subscribers: [],
});
```

Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

Sendo assim, pode-se dizer que o TypeORM facilitou completamente a confecção do banco de dados, uma vez que permitiu a manipulação do banco de dados de forma orientada a objetos, evitando com que precisasse de códigos em SQL.

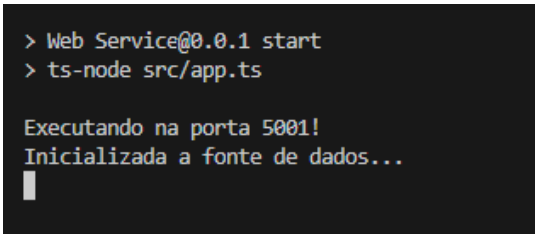
3. RESULTADOS

Após o início de fato do projeto, os resultados começaram a aparecer e avanços foram sendo feitos. O primeiro avanço foi na adição de um evento no banco de dados, ao projetar o código de inserção, assim que recebe a chamada da função ele informa no console que a requisição para inserção foi feita, portanto, o evento foi inserido com

sucesso. É possível observar o código da função de inserção no banco de dados descrita na Figura 10.

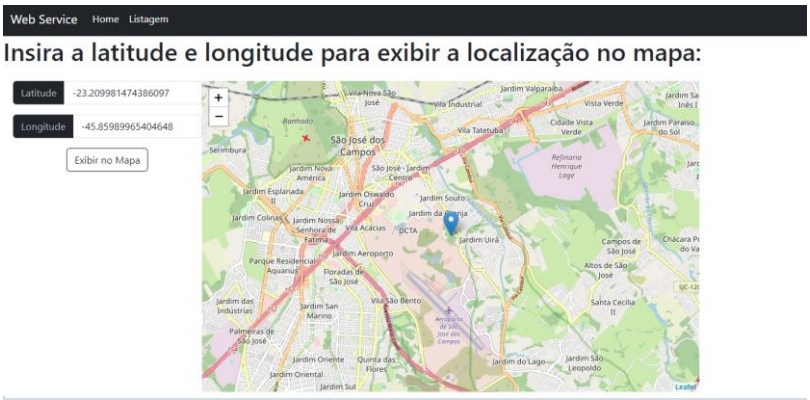
Após ser feita a integração do projeto do frontend e backend, na qual o frontend foi desenvolvido pelo aluno Thiago Frederico da Silva Zani, foi possível efetuar de fato os testes da aplicação, já que anteriormente estava se utilizando a ferramenta postman para as inserções. No frontend desenvolvido pelo mesmo, ao clicar em qualquer parte da tela é enviado para o banco de dados uma requisição de inserção (adição do evento), portanto, na Figura 30 ao inicializar a aplicação é exibido no console identificando a porta que está sendo utilizada e quando inicializou a fonte de dados na Figura 31 é possível observar que onde possui um "x" em vermelho no mapa é o local que foi efetuado um clique na página, na Figura 32 é possível ver a mensagem de requisição feita no console após as mensagens de inicialização e na Figura 33 o evento que foi inserido no banco de dados.

Figura 30. Mensagem de inicialização da aplicação exibida no console



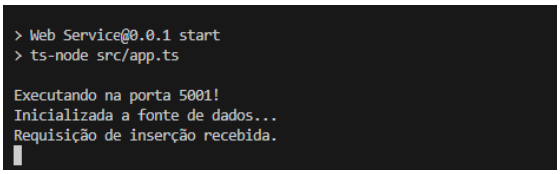
Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

Figura 31. Frontend com mapa exibindo onde foi efetuado o evento de clique



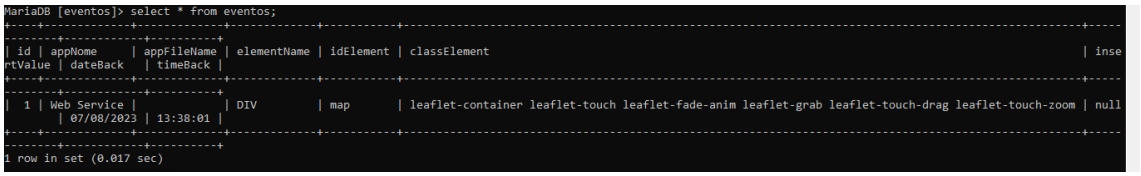
Fonte: Desenvolvido por Thiago Frederico da Silva Zani (2023).

Figura 32. Mensagem de requisição de inserção exibida no console após inicialização



Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

Figura 33. Exibição do evento de click inserido no banco de dados após a requisição.

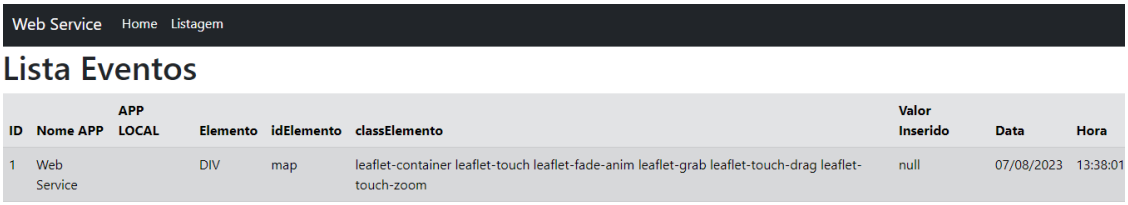


```
MariaDB [eventos]> select * from eventos;
+----+-------------+-------------+-------------+-----------+-----------------------------------------------------------------------------------------------------------------------+-----
-------+-----------+----------+
| id | appNome     | appFileName | elementName | idElement | classElement                                                                                                          | inse
rtValue | dateBack   | timeBack |
+----+-------------+-------------+-------------+-----------+-----------------------------------------------------------------------------------------------------------------------+-----
-------+-----------+----------+
|  1 | Web Service |             | DIV         | map       | leaflet-container leaflet-touch leaflet-fade-anim leaflet-grab leaflet-touch-drag leaflet-touch-zoom | null
       | 07/08/2023 | 13:38:01 |
+----+-------------+-------------+-------------+-----------+-----------------------------------------------------------------------------------------------------------------------+-----
-------+-----------+----------+
1 row in set (0.017 sec)
```

Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

Além disso, no frontend também foi desenvolvido uma página para verificar todos os eventos que estão presentes no banco de dados, portanto, foi desenvolvido pelo aluno Thiago Frederico da Silva Zani. Na Figura 34 é possível observar todos os eventos presentes no banco de dados sendo listados pela função da Figura 11 no frontend.

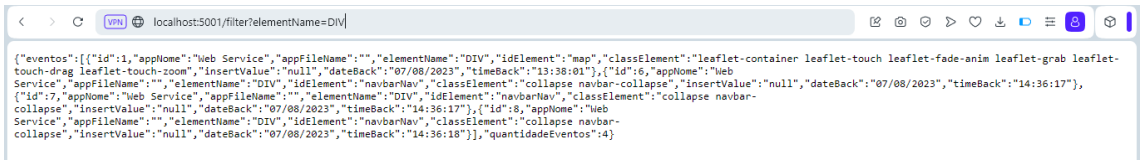
Figura 34. Frontend com listagem dos eventos presentes no banco de dados



Fonte: Desenvolvido por Thiago Frederico da Silva Zani (2023).

Após a realização das funções de captura e listagem dos eventos, foi necessário a criação de uma função para efetuar a filtragem dos dados do banco de dados, na qual foi filtrada passando os parâmetros desejados em uma queryString. Na Figura 16 é possível observar o código da filtragem dos dados por meio de uma queryString e na Figura 17 o resultado que foi obtido após a filtragem. Essa necessidade se deu pelo fato de conseguir contabilizar a quantidade de eventos com aquele mesmo valor, um exemplo claro é quando existe o duplo clique na tela. Portanto, na Figura 35 é possível observar a filtragem e contabilização dos eventos do banco de dados sendo exibidos em tela.

Figura 35. Filtragem e contabilização dos eventos presentes no banco de dados



Fonte: Desenvolvido pela autora Rafaela Vieira Cabral (2023).

Observando a necessidade de excluir um evento do banco de dados, foi feito também uma função com queryString para realizar a operação de exclusão do evento, na Figura 21 é possível observar o código da função capaz de efetuar a exclusão, portanto, o resultado obtido foi conforme o desejado e isso pode ser visto na Figura 21.

Concluindo a etapa de criação das funções necessárias que restavam, scripts de teste foram realizados, uma vez que para a manutenção da web service, desenvolvedores necessitariam testá-lo para ver sua funcionalidade. Com isso, a web service ao executar o comando de teste, conhecido como npm test, muda sua database para uma que

contenha o sufixo _test criada apenas para testar e verificar se os dados que foram inseridos nela estão corretos. Sendo assim, na Figura 24 é possível analisar a forma em que foram feitos os scripts de teste e na Figura 25 os resultados obtidos após o comando npm test.

4. CONCLUSÃO

Durante o projeto, o processo de pesquisa e desenvolvimento revelou a complexidade do processo de modelagem do banco de dados e a determinação dos dados que devem ser armazenados no banco de dados e posteriormente utilizados. Portanto, fica claro que a tarefa não é trivial, principalmente considerando o crescimento exponencial dos dados de utilização, que inevitavelmente afetará a estrutura do banco de dados.

Com isso, os objetivos estabelecidos foram alcançados, resultando no desenvolvimento de componentes para o monitoramento da interação do usuário com as interfaces. Além disso, um painel de controle (dashboard) foi criado, com o propósito de possibilitar o uso de dados e suas utilizações.

Vale ressaltar, entretanto, que existe espaço para efetuar aprimoramentos no projeto, são eles: otimização da modelagem de dados e por consequência, das classes utilizadas no backend, a implementação do controle de sessão, o aprimoramento e a generalização dos filtros de busca, a criação de análises de dados de acesso e sua disponibilização no dashboard, a modelagem dos dados necessários para representar eventos associados a mapas e demais informações em interfaces mais específicas, entre outros.

Nesse contexto, espera-se que o sistema possa ser testado em uma aplicação de interesse do instituto que esteja publicamente disponível ao cidadão (ex. sistema Queimadas) ou então que testes de validação e verificação em um sistema contendo tais aspectos seja efetuado.

5. REFERÊNCIAS BIBLIOGRÁFICAS

Axios Docs. Disponível em: https://axios-http.com/docs/intro. Acesso em: 27 jun. 2023.

BASRI, Nasrah Hassan et al. Conceptualizing and understanding user experience. In: 2016 4th International Conference on User Science and Engineering (i-USEr). IEEE, 2016. p. 81-84.

BIERMAN, Gavin; ABADI, Martín; TORGERSEN, Mads. Understanding typescript. In: ECOOP 2014–Object-Oriented Programming: 28th European Conference, Uppsala, Sweden, July 28–August 1, 2014. Proceedings 28. Springer Berlin Heidelberg, 2014. p. 257-281.

Express. Disponível em: https://expressjs.com/pt-br/. Acesso em: 05 de mar. 2023.

Git. Disponível em: https://git-scm.com/. Acesso em: 28 jun. 2023.

Introdução Express/Node - Aprendendo desenvolvimento web | MDN. Disponível em: https://developer.mozilla.org/pt-BR/docs/Learn/Server side/Express_Nodejs/Introduction. Acesso em: 27 jun. 2023.

Jest. Disponível em: https://jestjs.io/pt-BR/. Acesso em: 27 jun. 2023.

Oliveira Silva, Flávio de. Mapeamento objeto-relacional. Local: Universidade Federal de Uberlândia. 05 mar. 2023. Notas de aula

TypeORM. Disponível em: https://typeorm.io/. Acesso em: 27 jun. 2023.