Relatório de Métricas de Performance

ATIVIDADE: AV3

ALUNO: HENRY VILELA SILVA TITO

TURMA: 2DSM

INTRODUÇÃO

Este relatório apresenta as métricas obtidas para latência, tempo de resposta e testes de carga com:
1, 5 e 10 usuários simultâneos. Todos os valores estão em milissegundos (ms).
Como início, será demonstrado o Tempo de latência do servidor seguindo este pequeno exemplo.

medir latência (cliente -> servidor -> cliente)

Latência (ms): 81.69069999999999

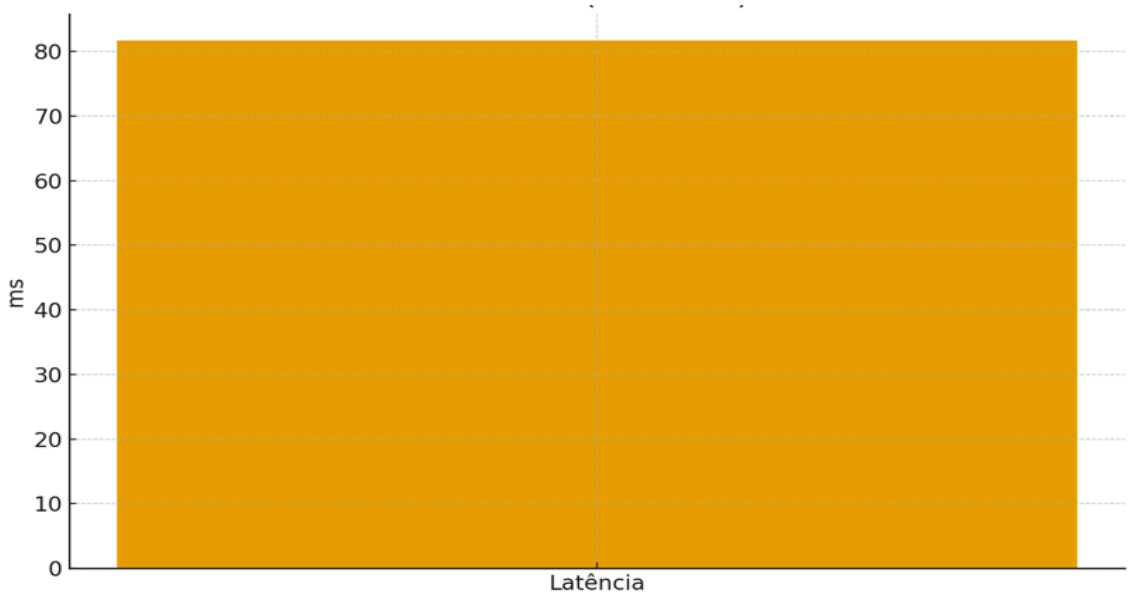

figura 1 latência do servidor.

Como próximo tópico, foi feita a medição do tempo de resposta para 1 (UM) usuário ao utilizar nosso sistema.

figura 2 gráfico tempo resposta 1 usuário.

Depois, foi feita a medição de teste de carga do sistema, mas agora com 5 (CINCO) usuários simultâneos.

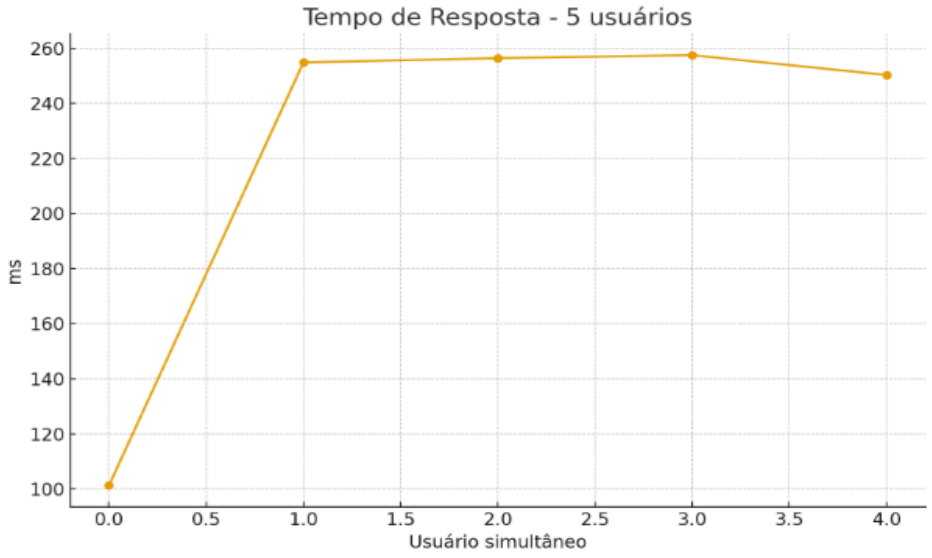

figura 3 gráfico tempo reposta 5 usuários.

Por fim, o teste de carga do sistema com 10 (DEZ) acessos de usuários simultâneos.

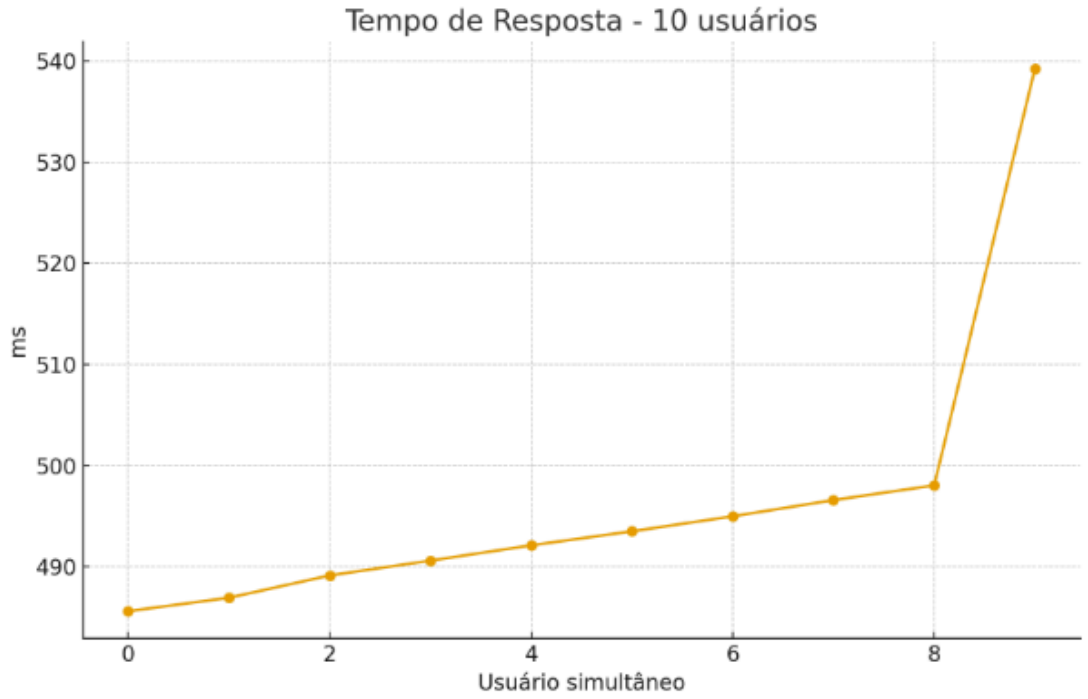

figura 4 gráfico tempo resposta 10 usuários

COMO FORAM OBTIDOS ESSES VALORES?

Metodologia de Coleta das Métricas

Para realizar as medições de latência, tempo de processamento do servidor e tempo de resposta total, foi desenvolvida uma estrutura composta por duas partes:

1. **Rotas específicas no backend**, responsáveis por fornecer medições precisas de cada etapa da requisição.

2. **Um script de testes no cliente**, que envia requisições e calcula os tempos de forma programática, simulando usuários reais.

Essa combinação permitiu medir tanto o comportamento interno do servidor quanto o impacto da comunicação cliente–servidor em cenários com diferentes níveis de carga.

---

1. Medição da Latência

A latência representa o tempo de ida e volta da requisição entre o cliente e o servidor.
Para medi-la, foi criada uma rota extremamente simples no backend:

```
router.get("/ping", (req, res) => {
  const inicio = performance.now();
  const fim = performance.now();
  res.json({ servidor: "ok", tempoServidor: fim - inicio });
});
```

Essa rota retorna imediatamente, sem realizar qualquer processamento significativo.
Assim, o tempo medido do lado do cliente corresponde quase exclusivamente ao atraso da rede (client → server → client).

No script cliente, a latência é medida da seguinte forma:

```
async function medirLatencia() {
  const inicio = performance.now();
  await fetch("http://localhost:3000/medicoes/ping");
  const fim = performance.now();
  return fim - inicio;
}
```

Esse valor foi registrado no relatório final como a **latência média observada**.

---

2. Medição do Tempo de Processamento do Servidor

Para medir somente o tempo que o servidor leva para executar uma operação interna, foi criada uma segunda rota:

```
router.get("/medir-processamento", async (req, res) => {
  const inicio = performance.now();

  const aeronaves = await prisma.aeronave.findMany({
    include: { pecas: true, etapas: true, testes: true },
  });

  const fim = performance.now();
  res.json({ tempoProcessamento: fim - inicio });
});
```

Essa rota simula uma operação real da aplicação, incluindo consultas ao banco de dados e carregamento de relacionamentos.

Ela permite medir **apenas o tempo interno gasto pelo backend**, ignorando latência da rede ou tempo adicional de resposta.

---

3. Tempo de Resposta Total (Latência + Processamento)

Para simular uma requisição real usada pela aplicação, foi criada a rota:

```
router.get("/relatorio-medicao", async (req, res) => {
  const iniProcesso = performance.now();

  const dados = await prisma.relatorio.findMany({
    include: {
      aeronave: {
        include: { pecas: true, etapas: true, testes: true },
      },
    },
  });

  const fimProcesso = performance.now();

  res.json({
    dados,
    tempoProcessamento: fimProcesso - iniProcesso,
  });
});
```

O script cliente mede o tempo total da chamada:

```
async function medirResposta() {
  const inicio = performance.now();
  await fetch("http://localhost:3000/medicoes/relatorio-medicao");
  const fim = performance.now();
  return fim - inicio;
}
```

Esse tempo inclui:

- Latência da rede
- Tempo de processamento
- Tempo de serialização/deserialização da resposta

É o valor que **realmente importa para a experiência do usuário final**.

---

4. Testes de Carga (Simulação de Múltiplos Usuários)

Para avaliar o comportamento da aplicação sob maior demanda, foi criada uma função que dispara diversas requisições simultâneas:

```
async function testarCarga(n: number) {
  const promessas = [];
  for (let i = 0; i < n; i++) {
    promessas.push(medirResposta());
  }
  return Promise.all(promessas);
}
```

Foram realizados testes com:

- **1 usuário**
- **5 usuários simultâneos**
- **10 usuários simultâneos**

Esses valores foram usados para gerar os gráficos do relatório, mostrando o impacto do aumento de carga no tempo total de resposta.

Caso queira testar por si mesmo, siga os passos:

Após toda configuração inicial explicada no read-ME, apenas dê:

cd backend

npm run dev (compilação)

abra outro terminal

coloque o comando: npx tsx testePerformance.ts

No terminal, aparecerão todos tempos/métricas solicitadas no exercìcio.