Gerenciamento de Campanhas e Desempenho com Altas Requisições

Cenário

A empresa executa campanhas como:

- "Frete grátis"
- "Pedido com 15% de desconto"
- "Pedido com 10% no frete"

Empresas parceiras podem participar de uma ou mais campanhas.

Desafios:

1. Alto volume de requisições: ~10.000 por minuto.
2. Complexidade no frontend para lidar com orientação a objetos (OO).
3. Backend e banco de dados sobrecarregados.

Tecnologias Atuais

- **Frontend:** React
- **Backend:** PHP com GraphQL
- **Banco de Dados:** MySQL

Objetivo: Transferir a lógica de OO para o backend, mantendo alta performance e simplicidade no frontend.

Estrutura do Banco de Dados

Criaremos uma estrutura eficiente para gerenciar campanhas e parcerias:

Tabela de Campanhas

```
CREATE TABLE Campanhas (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(255),
    tipo ENUM('frete_gratis', 'desconto', 'desconto_frete'),
    valor DECIMAL(10,2),
    ativo BOOLEAN DEFAULT TRUE
```

```
);
```

Tabela de Parcerias

```
CREATE TABLE ParceirosCampanhas (
    id INT AUTO_INCREMENT PRIMARY KEY,
    parceiro_id INT,
    campanha_id INT,
    FOREIGN KEY (campanha_id) REFERENCES Campanhas(id)
);
```

Essa estrutura permite que cada empresa esteja vinculada a uma ou mais campanhas.

Backend

Resolver GraphQL

Centralize a lógica de consulta no backend:

```
public function resolveCampanhasPorParceiro($parceiroId) {
    // Verifica se as campanhas estão no cache
    $campanhas = $this->cache-
>get("campanhas_parceiro_{$parceiroId}");

    if (!$campanhas) {
        // Consulta ao banco de dados
        $query = "
            SELECT c.*
            FROM Campanhas c
            JOIN ParceirosCampanhas pc ON c.id = pc.campanha_id
            WHERE pc.parceiro_id = ? AND c.ativo = 1
        ";
        $campanhas = $this->db->query($query, [$parceiroId]);

        // Armazena o resultado no cache por 10 minutos
        $this->cache->set("campanhas_parceiro_{$parceiroId}",
$campanhas, 600);
    }
```

```
    return $campanhas;
}
```

Descrição

1. **Cache (Redis):** Verifica primeiro no cache para reduzir consultas ao banco.
2. **Consulta ao Banco:** Busca todas as campanhas ativas vinculadas ao parceiro.
3. **Armazenamento no Cache:** Salva o resultado para futuras consultas.

Frontend

Consumo de Dados via GraphQL

O frontend consome as campanhas por meio de uma query simplificada:

```
query {
    campanhasPorParceiro(parceiroId: 123) {
        id
        nome
        tipo
        valor
    }
}
```

Exibição no Frontend

Utilize componentes React para exibir as campanhas:

```
import React from 'react';

const Campanhas = ({ campanhas }) => (
    <ul>
        {campanhas.map(campanha => (
            <li key={campanha.id}>
                <strong>{campanha.nome}</strong> - {campanha.tipo} -
{campanha.valor}%
            </li>
        ))}
    </ul>
);
```

```
export default Campanhas;
```

Consumo com Fetch

Faça a requisição ao backend e passe os dados ao componente:

```
fetch('/graphql', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
        query: `
          query {
              campanhasPorParceiro(parceiroId: 123) {
                  id
                  nome
                  tipo
                  valor
              }
          }
        `
    })
})
  .then(response => response.json())
  .then(data => {
      console.log(data.data.campanhasPorParceiro);
  });
```

Cache

Estrutura no Redis

Armazene os dados das campanhas por parceiro:

- **Chave:** `campanhas_parceiro_<parceiroId>`
- **Valor:** [
  { "id": 1, "nome": "Frete grátis", "tipo": "frete_gratis", "valor": 0 },
  { "id": 2, "nome": "15% desconto", "tipo": "desconto",

```
"valor": 15 },
  { "id": 3, "nome": "10% no frete", "tipo": "desconto_frete",
"valor": 10 }
]
```

Benefícios do Cache

1. **Redução de Latência:** Dados são recuperados mais rapidamente.
2. **Redução de Carga no Banco:** Diminui consultas repetitivas.
3. **Atualizações Temporais:** Atualize o cache quando houver alterações nas campanhas.

Fluxo de Dados

1. **Cadastro de Campanhas:**
    a. Backend processa a criação de campanhas e atualiza o cache.
2. **Consulta de Campanhas:**
    a. Frontend chama o backend via GraphQL.
    b. Backend verifica o cache e retorna os dados processados.
3. **Atualização de Campanhas:**
    a. Backend atualiza o banco e o cache simultaneamente.

Escalabilidade

1. **Cache Redis:** Reduz a carga no MySQL e acelera consultas.
2. **Filas de Mensagens:** Utilize RabbitMQ ou SQS para processamento assíncrono de atualizações em larga escala.
3. **Escalabilidade Horizontal:** Configure múltiplos servidores backend para lidar com altas requisições.

Benefícios da Solução

- **Frontend Simples:** Dados prontos, sem lógica complexa.
- **Backend Centralizado:** Lógica OO bem organizada e eficiente.
- **Desempenho Otimizado:** Uso de cache para minimizar o impacto de altas requisições.
- **Flexibilidade:** Permite adicionar novas campanhas facilmente.

Conclusão

Essa solução distribui responsabilidades de forma eficiente, mantendo o frontend leve, o backend escalável e o banco de dados com alta performance.