Gestão e qualidade de software

Refatoração de Código e Boas Práticas do *Clean Code*

URL GitHub:https://github.com/LucasFMSuarez/Projeto-A3-Gestao-e-qualidade-de-software.git
Apresentação YouTube:https://youtu.be/1B6kC_q07a8

Guilherme Arantes Nascimento - 825155575
Lucas Felipe Monteiro Suarez - 824138683
Luiz Washington de Jesus Muraro - 824148694
Mariana Cristina Silva Oliveira - 825123486

São Paulo
2025

Introdução

O presente documento descreve a arquitetura, o funcionamento e o propósito de um sistema distribuído baseado em microsserviços, desenvolvido com o objetivo de gerenciar lembretes e observações de maneira eficiente, modular e escalável. A solução adota um modelo orientado a eventos, no qual a comunicação entre os componentes ocorre por meio de um barramento central responsável por receber, registrar e retransmitir eventos a todos os serviços participantes.

Esta implementação corresponde a uma refatoração estruturada e à melhoria integral de uma versão anterior do sistema. O código original, embora funcional, apresentava organização limitada, acoplamento excessivo e ausência de mecanismos adequados de padronização interna. Assim, o processo de refatoração buscou aprimorar a clareza, a manutenibilidade e a robustez da aplicação, sem alterar sua lógica fundamental de operação.

A nova arquitetura foi reorganizada em serviços independentes, cada qual responsável por uma unidade específica do domínio: criação de lembretes, registro e atualização de observações, classificação automática de conteúdo e consolidação de dados para consulta. A intermediação entre esses módulos é realizada pelo barramento de eventos, o qual assegura que mudanças ocorridas em qualquer serviço sejam propagadas de forma assíncrona e não bloqueante para os demais.

Além disso, esta versão introduz um padrão mais consistente de tratamento de erros, modularização do código fonte, isolamento das regras de negócio e melhoria da legibilidade geral da aplicação. Também foi incorporada uma ferramenta auxiliar para agregação automática do código, com a finalidade de facilitar auditorias e análises globais do projeto.

Dessa forma, o sistema resultante apresenta maior aderência a boas práticas de engenharia de software, promove facilidade de expansão futura e oferece uma base mais sólida para experimentações, manutenção contínua e ensino de conceitos relacionados a microsserviços e comunicação orientada a eventos.

Metodologias de clean code usadas(DRY, KISS e SOLID):

No processo de refatoração do código legado, foram aplicadas diversas metodologias de *clean code* para tornar o código mais simples, legível e fácil de manter. A primeira delas foi o princípio DRY (Don't Repeat Yourself), que busca eliminar repetições desnecessárias. Em vários pontos, blocos duplicados foram substituídos por uma estrutura de mapa de funções, permitindo que chamadas como funcoes[req.body.tipo](req.body.dados) substituíssem múltiplos ifs ou trechos redundantes. Essa abordagem reduziu a repetição, eliminou código igual em diferentes *handlers* de eventos e facilitou a manutenção.

Outro princípio aplicado foi o KISS (Keep It Simple, Stupid), que trouxe simplicidade ao código em diversos aspectos. O uso do express.json() em substituição ao antigo bodyParser tornou a configuração mais direta. Além disso, as responsabilidades foram separadas por serviço, de forma que cada microsserviço passou a focar em um evento específico. O tratamento de erros também foi simplificado com blocos try/catch minimalistas, com menos etapas desnecessárias e livre de configurações antigas.

Por fim, alguns elementos do SOLID foram parcialmente aplicados. O princípio mais evidente foi o Single Responsibility Principle (SRP), já que cada microsserviço passou a ter uma responsabilidade única: o serviço de lembretes cria lembretes, o de observações cria e atualiza observações, o de classificação analisa textos, o de consultas gera visões consolidadas e o barramento envia eventos. Essa divisão garante aderência total ao SRP. No entanto, outros princípios do SOLID, como OCP, LSP, ISP e DIP, não foram implementados de forma clara, principalmente pela ausência de interfaces, inversão de dependência e uma estrutura orientada a objetos mais robusta.

Em resumo, a aplicação de DRY e KISS trouxe ganhos significativos de simplicidade e manutenção, enquanto o SOLID foi seguido parcialmente, com destaque para o SRP.

Código Original (Legado) e suas deficiências

O código legado apresenta várias limitações:

Problemas estruturais e técnicos

- **Uso de variáveis globais**
    - Exemplos: eventos = [], lembretes = {}, contador = 0, baseConsulta = {}.
    - **Problema:** Pode gerar conflitos em ambiente concorrente ou quando múltiplos microsserviços acessam dados simultaneamente.
    - **consequência:** Risco de inconsistência de dados e difícil manutenção/testes.
- **Ausência de banco de dados**
    - Todos os dados são armazenados em memória.
    - **Problema:** Se o servidor reinicia, todos os lembretes, observações e eventos são perdidos.
    - **consequência:** Perda de dados, impossibilidade de escalar para produção.
- **Código repetitivo**
    - Cada serviço implementa funções de armazenamento, envio de eventos e tratamento de rotas de forma isolada.
    - **Problema:** Dificulta manutenção e evolução, aumenta chance de bugs.
- **Tratamento de erros insuficiente**
    - Exemplo: try { ... } catch (err) {} vazio.
    - **Problema:** falta de notificação de falhas e sem log.
    - **consequência:** Debug muito difícil e serviços podem falhar sem sinalizar.
- **Falta de modularização**
    - Toda a lógica do serviço está no index.js.
    - Não há separação clara entre:
        - Lógica do serviço
        - Servidor.
        - Envio de Eventos.
- **Código síncrono ou parcial assíncrono**
    - Exemplo: envio de eventos com axios.post sem await em alguns serviços.

- **Problema:** Pode ocorrer perda de eventos se o serviço cair antes do envio.

- **Escalabilidade limitada**
  - IDs gerados manualmente (contador) e arrays/objetos em memória não funcionam bem em múltiplas instâncias.
  - Sem banco de dados, não há suporte para índices ou consultas complexas.
- **Ausência de validações**
  - Eventos recebidos e dados do corpo das requisições não são validados.
  - Pode receber dados inconsistentes e quebrar o serviço.

Código Refatorado e suas Mudanças

O novo código resolve a maioria dos problemas do legado.

principais mudanças:

Arquitetura e Organização

1. **Modularização clara**
   - Cada serviço possui:
     - routes.js → rotas HTTP
     - servico*.js → Função do serviço
     - distribuidorEventos.js → envio de eventos
   - Barramento separado (servicoBarramento.js).
2. **Integração com MongoDB**
   - Uso do mongoose para salvar (Lembretes e Observações):
     - Lembretes (LembreteModel.js)
     - Observações (ObservacaoModel.js)
   - IDs gerenciados pelo banco (id sequencial ou UUID).
3. **Classe de serviço**

- Ex: ClassificacaoService, ObservacoesService, LembretesService.
- **Benefício:** Encapsula a lógica de negócios e tratamento de eventos.

4. **Envio de eventos robusto**
   - await axios.post(...) com tratamento de erro e timeout.
   - Eventos não bloqueiam o fluxo principal e logs claros de falhas.
5. **Validação de dados**
   - Ex: checa se eventos ou campos obrigatórios existem antes de processar.
   - Ex: if (!evento || !evento.tipo).
6. **Melhoria no tratamento de erros**
   - Logging consistente (console.error) com mensagens claras.
   - Resposta HTTP adequada (400, 500) em caso de erro.
7. **Uso de padrões modernos**
   - async/await em todo o código.
   - Classes POO para serviços.
   - Separação clara entre persistência, lógica de negócio e rotas.
8. **Adição da função de exclusão de dados**
   - Eventos de exclusão (LembreteExcluido, ObservacaoExcluida) implementados.
   - Antes não havia remoção de dados.

9. **Funcionalidades adicionais**
   - Consulta de lembretes + observações usando métodos assíncronos (listarLembretesComObservacoes).
   - aplicação de ID único pelo MongoDB evitando repetições.
   - IDs de observações usando UUID (uuidv4), garantindo unicidade.
10. **Escalabilidade e manutenção**

- Cada serviço agora pode ser escalado independentemente.
- Armazenamento centralizado no MongoDB.
- Lógica de envio de eventos isolada, facilitando alterações.

Testes Unitários e Integração realizados no Código

barramento.test.js – Testes do Barramento de Eventos.

Testa o comportamento do serviço central que recebe eventos e redistribui para todos os microsserviços.

Testes cobrem:

Erro quando envia evento sem “tipo”

Garante validação de entrada.

Aceitar evento válido e encaminhar para 4 serviços

- Testa se o barramento envia eventos usando axios.post.
- Confirma que foram feitas **4 chamadas**, 1 para cada serviço inscrito.

Armazenar eventos localmente

Garante que /eventos (GET) retorna o histórico.

Não quebrar quando axios dá erro

Mesmo se um serviço estiver fora do ar, o barramento:

- não cai.
- continua retornando { msg: "ok" }.

integracao.test.js – Teste de integração do serviço Lembretes.

- Usa MongoDB em memória.
- Cria lembrete e salva no banco.

- Processar evento LembreteClassificado e atualizar no Mongo.

testeRoutes.test.js – Testa as rotas do serviço de Lembretes.

- GET retorna lista usando mock.
- PUT cria lembrete e chama criarLembrete.

POST /eventos chama processarEvento.**testeServico.test.js – Testa a lógica interna do serviço Lembretes.**

criarLembrete:

- Salva no banco.
- Envia evento LembreteCriado.

processarEvento:

- Atualiza status.
- chama save()

envia LembreteAtualizado**integracaoObservacoes.test.js – Testes simples da lógica de Observações**

- Criar observação.
- Atualizar observação via evento.
- Ignorar tipos de evento desconhecidos.

testeRoutesObservacoes.test.js – Rotas das Observações

- Criar observação (PUT).
- Validar erro quando falta texto.
- Listar observações.
- Receber evento.
- Evento sem tipo → 400.

testeServicoObservacoes.test.js – Serviço de Observações

- Criar observação com UUID mockado

- Listar observações
- Atualizar quando recebe ObservacaoClassificada
- Ignorar eventos desconhecidos

integracaoClassificacao.test.js – Teste de Integração do serviço Classificação.

Testa o fluxo completo dentro do serviço:

Classificar Lembrete

- Recebe LembreteCriado.
- Processa

  Envia LembreteClassificado.

Classificar Observação

- Recebe ObservacaoCriada.
- Envia ObservacaoClassificada.

testeRoutesClassificacao.test.js – Testa as rotas de classificação.

- POST /eventos chama processarEvento.
- Retorna 400 quando tipo não existe.
- Retorna 500 se processarEvento lançar erro.

testeServicoClassificacao.test.js – Testa a lógica interna da classificação.

- Classifica lembretes com palavra-chave como "importante".
- Classifica sem palavra-chave como "comum".
- Classifica observações corretamente.

testeRoutesConsulta.test.js – Rotas do serviço Consulta.

- GET /lembretes retorna dados do banco com mock.

- POST /eventos chama processarEvento.
- Continue funcionando mesmo com erro interno.

testeServicoConsulta.test.js – Serviço de consulta.

Junta lembretes + suas observações corretamente

Simula:

- Lembretes retornados do banco (MongoDB).
- Observações retornadas do banco (MongoDB).
- Resultado é um objeto combinando ambos voltando do banco (MongoDB).

RESUMO DOS TESTES

1. Validação de entrada das rotas

Evitar requisições inválidas.

2. Integração completa entre serviços

Principalmente Lembretes e Classificação.

3. Fluxo de eventos via barramento

Testando:

- recebimento
- envio
- armazenamento
- tolerância a falhas

4. Lógica de negócio pura

Classificação de textos (importante/comum), atualização de status, etc.

5. Acesso ao banco

Com MongoMemoryServer para não sujar o banco real.

6. Mocks de dependências externas

Axios, Models Mongoose, uuid, distribuidores de eventos.

Justificativas para as mudanças

Mudança	Justificativa
Uso de MongoDB	Evita perda de dados, garante persistência e suporta consultas complexas.
Classes de serviço	Encapsula lógica de negócios, melhora manutenção e testabilidade.
Modularização (routes, services, models)	Facilita leitura, testes unitários e manutenção futura.
Validação de eventos	Garante a integridade dos dados recebidos.
Uso de async/await em todos os fluxos	Evita inconsistências ao enviar ou processar eventos assíncronos.
Eventos de exclusão	Permite remover lembretes e observações de forma consistente.
IDs únicos com UUID / incremento seguro	Evita conflito de IDs em múltiplas instâncias ou reinicializações.
Logging e tratamento de erros	Facilita monitoramento e depuração em produção.
Indexação no MongoDB	Evita duplicatas e melhora performance de consultas.

Por que Clean Code é importante na manutenção de software

A manutenção de software depende diretamente da clareza e da organização do código. Em grande parte dos projetos, a maior parte do esforço não está na criação de novas funcionalidades, mas sim na leitura, compreensão e alteração do código existente. Por esse motivo, práticas de Clean Code tornam o processo de manutenção mais eficiente.

Código com nomes claros, funções pequenas e responsabilidades bem definidas reduz o tempo necessário para localizar um comportamento específico ou identificar a origem de um erro. Quando o código é estruturado de forma consistente, a compreensão do sistema é mais rápida e menos sujeita a interpretações incorretas.

Além disso, código limpo diminui a probabilidade de introdução de novos bugs durante uma alteração. Funções curtas e componentes com responsabilidades isoladas facilitam a criação de testes, a análise de impacto e a identificação de efeitos colaterais.

Outro benefício importante é a velocidade de adaptação de novos desenvolvedores. Documentação interna clara e estruturas previsíveis reduzem o tempo de onboarding e diminuem a dependência de explicações informais.

Em projetos de longo prazo, a tendência natural é o aumento da complexidade. A aplicação contínua dos princípios de Clean Code ajuda a evitar um acúmulo de soluções improvisadas, o que pode resultar em código difícil de modificar, testar ou estender.

Do ponto de vista econômico, manutenção costuma representar grande parte do custo total de um software. Código limpo reduz retrabalho, facilita correções e permite evolução contínua com menor esforço.

De forma resumida, Clean Code contribui diretamente para a sustentabilidade do sistema, reduz riscos e melhora a eficiência da equipe. Manter o código limpo não é apenas uma boa prática, mas um requisito para garantir a durabilidade e a qualidade do software ao longo do tempo.

Código antigo:

barramento-de-eventos

index.js

```
const express = require('express');
const bodyParser = require('body-parser');
//para enviar eventos para os demais microsserviços
const axios = require('axios');
eventos = []

const app = express();
app.use(bodyParser.json());
app.post('/eventos', (req, res) => {
const evento = req.body;
eventos.push(evento)

//envia o evento para o microsserviço de lembretes
axios.post('http://localhost:4000/eventos', evento);
//envia o evento para o microsserviço de observações
```

```
axios.post('http://localhost:5000/eventos', evento);
//envia o evento para o microsserviço de consulta
//axios.post("http://localhost:6000/eventos", evento);
//envia o evento para o microsservico de classificacao
postEventoConsulta(evento);

async function postEventoConsulta(evento) {
    const url = "http://localhost:6000/eventos";
    const timeout = 5000; // Contagem em millisegundos

    try {
        await axios.post(url, evento, { timeout });
    } catch (error) {
        // Ignore todos os erros

    }
}

axios.post("http://localhost:7000/eventos", evento);
res.status(200).send({ msg: "ok" });
});

app.get('/eventos', (req, res) => {
res.send(eventos)})


app.listen(10000, () => {
console.log('Barramento de eventos. Porta 10000.')
console.log(eventos);
})
```

classificacao

index.js

```
const express = require("express");
const axios = require("axios");
const app = express();

app.use(express.json());
palavraChave = "importante";

const funcoes = {
```

```
 ObservacaoCriada: (observacao) => {
 observacao.status =
observacao.texto.includes(palavraChave)
 ? "importante"
 : "comum";

axios.post("http://localhost:10000/eventos", {
tipo: "ObservacaoClassificada",
dados: observacao,
});
},
};

app.post("/eventos", (req, res) => {
try {
funcoes[req.body.tipo](req.body.dados);
} catch (err) {}
res.status(200).send({ msg: "ok" });
});

app.listen(7000, () => console.log ("Classificação. Porta 7000"));
```

consulta

index.js

```
const express = require("express");
const axios = require("axios");

const app = express();
app.use(express.json());



baseConsulta = {};

const funcoes = {
    LembreteCriado: (lembrete) => {
    baseConsulta[lembrete.contador] = lembrete;
    },
    ObservacaoCriada: (observacao) => {
    const observacoes =
    baseConsulta[observacao.lembreteId]["observacoes"] ||
```

```
    [];
    observacoes.push(observacao);
    baseConsulta[observacao.lembreteId]["observacoes"] =
    observacoes;
    },

    ObservacaoAtualizada: (observacao) => {
     const observacoes =
baseConsulta[observacao.lembreteId]["observacoes"];
     const indice = observacoes.findIndex((o) => o.id ===
observacao.id);
     observacoes[indice] = observacao;
     },
    };



app.get("/lembretes", (req, res) => {
 res.status(200).send(baseConsulta);
 });

app.post("/eventos", (req, res) => {
    try {
    funcoes[req.body.tipo](req.body.dados);
    } catch (err) {}
    res.status(200).send({ msg: "ok" });
 });


app.listen(6000, async () => {
console.log("Consultas. Porta 6000")
const resp = await
axios.get("http://localhost:10000/eventos");
//axios entrega os dados na propriedade data
resp.data.forEach((valor, indice, colecao) => {
try {
funcoes[valor.tipo](valor.dados);
} catch (err) {}
});
});
```

lembretes

index.js

```
const express = require ('express');
const bodyParser = require('body-parser');
const axios = require("axios");
const app = express();
app.use(bodyParser.json());
lembretes = {};
contador = 0;

app.get ('/lembretes', (req, res) => {
    res.send(lembretes);

});
app.put("/lembretes", async (req, res) => {
contador++;
const { texto } = req.body;
lembretes[contador] = {
contador, texto
 };
 await axios.post("http://localhost:10000/eventos", {
tipo: "LembreteCriado",
dados: {
contador,
texto,
},
});
res.status(201).send(lembretes[contador]);
});

app.post("/eventos", (req, res) => {
console.log(req.body);
res.status(200).send({ msg: "ok" });
});

app.listen(4000, () => {
console.log('Lembretes. Porta 4000');
});
```

observacoes

index.js

```
const express = require ('express');
const bodyParser = require('body-parser');
```

```
const axios = require ('axios');
const app = express();
const { v4: uuidv4 } = require('uuid');

observacoesPorLembreteId = {};

const funcoes = {
 ObservacaoClassificada: (observacao) => {
 const observacoes =
observacoesPorLembreteId[observacao.lembreteId];
 const obsParaAtualizar = observacoes.find(o => o.id ===
observacao.id)
 obsParaAtualizar.status = observacao.status;
 axios.post('http://localhost:10000/eventos', {
 tipo: "ObservacaoAtualizada",
 dados: {
 id: observacao.id,
 texto: observacao.texto,
 lembreteId: observacao.lembreteId,
 status: observacao.status
 }
 });
 }
 }




app.use(bodyParser.json());

 //:id é um placeholder
 //exemplo: /lembretes/123456/observacoes

app.put('/lembretes/:id/observacoes', async (req, res) => {
const idObs = uuidv4();
const { texto } = req.body;
 //req.params dá acesso à lista de parâmetros da URL
 observacoesDoLembrete =
 observacoesPorLembreteId[req.params.id] || [];
 observacoesDoLembrete.push({ id: idObs, texto, status:"aguardando" });
 observacoesPorLembreteId[req.params.id] =
 observacoesDoLembrete;
```

```
await axios.post('http://localhost:10000/eventos', {
tipo: "ObservacaoCriada",
dados: {
id: idObs, texto, lembreteId: req.params.id,
status:"aguardando"
}
})
res.status(201).send(observacoesDoLembrete);
});

app.get('/lembretes/:id/observacoes', (req, res) => {
res.send(observacoesPorLembreteId[req.params.id] || []);
});

app.post("/eventos", (req, res) => {
try{
funcoes[req.body.tipo](req.body.dados);
}
catch (err){}
res.status(200).send({ msg: "ok" });
});

app.listen(5000, (() => {
 console.log('Lembretes. Porta 5000');
 }));
```

Código atual:

banco

conexao.js

```
// banco/conexao.js
const mongoose = require("mongoose");

// Classe POO
class Database {
  constructor(url) {
    this.url = url;
  }

  async conectar() {
    try {
```

```
      await mongoose.connect(this.url, {
        useNewUrlParser: true,
        useUnifiedTopology: true
      });
      console.log(" MongoDB conectado com sucesso!");
    } catch (err) {
      console.error(" Erro ao conectar no MongoDB:", err);
    }
  }
}

const db = new Database(
  "mongodb://localhost:27017/a3"
);

// Exporta a função db
module.exports = () => db.conectar();
```

lembreteModel.js

```
const mongoose = require("mongoose");

const LembreteSchema = new mongoose.Schema({
  id: { type: Number, required: true, unique: true },
  texto: String,
  status: { type: String, default: "aguardando" }
});

module.exports = mongoose.model("Lembrete", LembreteSchema);
```

ObservacaoModel.js

```
const mongoose = require("mongoose");

const ObservacaoSchema = new mongoose.Schema({
  id: {
    type: String,
    required: true
  },
  texto: {
    type: String,
    required: true
  },
  lembreteId: {
    type: String,
```

```
    required: true
  },
  status: {
    type: String,
    default: "aguardando"
  }
});
ObservacaoSchema.index({ lembreteId: 1, texto: 1 }, { unique: true });
module.exports = mongoose.model("Observacao", ObservacaoSchema);
```

barramento-de-eventos

routes.js

```
// barramento/src/rotas.js
const express = require("express");
const router = express.Router();
const { enviarEvento, armazenarEvento, listarEventos } =
require("./servicoBarramento");

router.post("/eventos", async (req, res) => {
  const evento = req.body;

  if (!evento || !evento.tipo) {
    console.log("Evento inválido recebido:", evento);
    return res.status(400).send({ erro: "Evento inválido: faltando
'tipo'" });
  }

  console.log("Evento recebido:", evento);
  armazenarEvento(evento);

  // envia para os microsserviços
  try {
    await enviarEvento("http://localhost:4000/eventos", evento);
    await enviarEvento("http://localhost:5000/eventos", evento);
    await enviarEvento("http://localhost:6000/eventos", evento);
    await enviarEvento("http://localhost:7000/eventos", evento);
  } catch (err) {
    console.log("Erro enviando evento:", err.message);
  }

  res.status(200).send({ msg: "ok" });
});
```

```
router.get("/eventos", (req, res) => {
  res.send(listarEventos());
});

module.exports = router;
```

server.js

```
// barramento/src/server.js
const express = require("express");
const rotas = require("./routes");

const app = express();
app.use(express.json());
app.use(rotas);

const PORT = 10000;
app.listen(PORT, () => {
  console.log(`Barramento de eventos rodando na porta ${PORT}.`);
});
```

servicoBarramento.js

```
// barramento/src/servicoBarramento.js
const axios = require("axios");

// Classe
class BarramentoService {
  constructor() {
    this.eventos = [];
  }

  async enviarEvento(url, evento) {
    try {
      await axios.post(url, evento, {
        headers: { "Content-Type": "application/json" },
        timeout: 5000
      });
    } catch (error) {
      console.error(`Erro enviando evento para ${url}:
${error.message}`);
    }
```

```
  }

  armazenarEvento(evento) {
    this.eventos.push(evento);
  }

  listarEventos() {
    return this.eventos;
  }
}

const barramento = new BarramentoService();

// Exportamos as funções
module.exports = {
  enviarEvento: (url, evento) => barramento.enviarEvento(url, evento),
  armazenarEvento: (evento) => barramento.armazenarEvento(evento),
  listarEventos: () => barramento.listarEventos()
};
```

classificacao

distribuidorEventos.js

```
// classificacao/src/distribuidorEventos.js
const axios = require("axios");

async function enviarEvento(tipo, dados) {
  try {
    await axios.post("http://localhost:10000/eventos", {
      tipo,
      dados
    });
  } catch (err) {
    console.error("Erro ao enviar evento (classificação):",
err.message);
  }
}

module.exports = { enviarEvento };
```

routes.js

```
// classificacao/src/routes.js
```

```
const express = require("express");
const { processarEvento } = require("./servicoClassificacao");

const router = express.Router();

router.post("/eventos", async (req, res) => {
  try {
    let tipo, dados;

    // aceita os dois formatos que o nosso barramento envia
    if (req.body.tipo && typeof req.body.tipo === "object") {
      tipo = req.body.tipo.tipo;
      dados = req.body.tipo.dados;
    } else {
      tipo = req.body.tipo;
      dados = req.body.dados;
    }

    if (!tipo) {
      return res.status(400).send({ erro: "Evento sem tipo" });
    }

    console.log("Classificação recebeu evento:", tipo, dados);

    await processarEvento(tipo, dados);

    res.status(200).send({ msg: "ok" });
  } catch (err) {
    console.error("Erro na rota de classificação:", err);
    res.status(500).send({ erro: "Erro ao processar evento." });
  }
});

module.exports = router;
```

server.js

```
// classificacao/src/server.js
const express = require("express");
const routes = require("./routes");

const app = express();
app.use(express.json());
app.use(routes);
```

```
if (require.main === module) {
  const PORT = process.env.PORT || 7000;
  app.listen(PORT, () => console.log(`Classificação. Porta ${PORT}`));
}

module.exports = app;
```

servicoClassificacao.js

```
// classificacao/src/servicoClassificacao.js
const { enviarEvento } = require("./distribuidorEventos");

class ClassificacaoService {
  constructor() {
    this.palavraChave = "importante";

    this.funcoes = {
      LembreteCriado: async (dados) => {
        await this.classificarLembrete(dados);
      },
      ObservacaoCriada: async (dados) => {
        await this.classificarObservacao(dados);
      }
    };
  }

  // Classifica lembrete
  async classificarLembrete(dados) {
    const texto = dados && dados.texto ? String(dados.texto) : "";
    const status = dados.status
      ? dados.status
      : (texto.includes(this.palavraChave) ? "importante" : "comum");
    const atualizado = { ...dados, status };

    console.log(" Lembrete classificado:", atualizado);

    await enviarEvento("LembreteClassificado", atualizado);
  }

  // Classifica observação
  async classificarObservacao(dados) {
    const texto = dados && dados.texto ? String(dados.texto) : "";
```

```
    const status = dados.status
      ? dados.status
      : (texto.includes(this.palavraChave) ? "importante" : "comum");
    const atualizado = { ...dados, status };

    console.log(" Observação classificada:", atualizado);

    await enviarEvento("ObservacaoClassificada", atualizado);
  }

  // Processa eventos recebidos
  async processarEvento(tipo, dados) {
    const fn = this.funcoes[tipo];
    if (fn) {
      try {
        await fn(dados);
      } catch (err) {
        console.error("Erro ao processar evento na classificação:",
err.message);
      }
    } else {
      console.log(" Tipo ignorado pela classificação:", tipo);
    }
  }
}

const classificacaoService = new ClassificacaoService();

module.exports = {
  processarEvento: (tipo, dados) =>
classificacaoService.processarEvento(tipo, dados),
  palavraChave: classificacaoService.palavraChave
};
```

consulta

app.js

```
const express = require("express");
const routes = require("./routes");

const app = express();
app.use(express.json());
app.use(routes);
```

```
module.exports = app;
```

funcoesEventos.js

```
// consulta/src/funcoesEventos.js
const Lembrete = require("../../banco/lembreteModel");
const Observacao = require("../../banco/observacaoModel");

const funcoes = {
  LembreteCriado: async (dados) => {
    await Lembrete.create({
      id: dados.id,
      texto: dados.texto,
      status: dados.status,
    });
    console.log("Consulta armazenou lembrete:", dados);
  },

  LembreteAtualizado: async (dados) => {
    await Lembrete.findOneAndUpdate(
      { id: dados.id },
      { status: dados.status }
    );
    console.log("Consulta atualizou lembrete:", dados);
  },

  ObservacaoCriada: async (dados) => {
    await Observacao.findOneAndUpdate(
      { id: dados.id },
      {
        texto: dados.texto,
        status: dados.status,
        lembreteId: dados.lembreteId,
      },
      { upsert: true, new: true }
    );
    console.log("Consulta armazenou/atualizou observação:", dados);
  },

  ObservacaoAtualizada: async (dados) => {
    await Observacao.findOneAndUpdate(
      { id: dados.id },
      { texto: dados.texto, status: dados.status }
```

```
    );
    console.log("Consulta atualizou observação:", dados);
  },

  //  LembreteExcluido → apaga lembrete e observações
  LembreteExcluido: async (dados) => {
    const { id } = dados;

    await Lembrete.findOneAndDelete({ id });
    await Observacao.deleteMany({ lembreteId: id });

    console.log("Consulta excluiu lembrete e observações:", id);
  },

  //  ObservacaoExcluida → apaga só a observação
  ObservacaoExcluida: async (dados) => {
    const { id } = dados;

    await Observacao.findOneAndDelete({ id });

    console.log("Consulta excluiu observação:", id);
  },
};

async function processarEvento(tipo, dados) {
  const fn = funcoes[tipo];

  if (fn) {
    await fn(dados);
  } else {
    console.log(" Consulta ignorou tipo:", tipo);
  }
}

module.exports = { processarEvento };
```

routes.js

```
// consulta/src/routes.js
const express = require("express");
const router = express.Router();

const { listarLembretesComObservacoes } = require("./servicoConsulta");
const { processarEvento } = require("./funcoesEventos");
```

```
// GET /lembretes
router.get("/lembretes", async (req, res) => {
  try {
    const resultado = await listarLembretesComObservacoes();
    res.status(200).send(resultado);
  } catch (err) {
    console.error("Erro ao listar lembretes:", err);
    res.status(500).send({ erro: "Erro ao listar." });
  }
});

// POST /eventos
router.post("/eventos", async (req, res) => {
  console.log("Consulta recebeu evento:", req.body);
  const { tipo, dados } = req.body;

  try {
    await processarEvento(tipo, dados);
  } catch (err) {
    console.error("Erro ao processar evento na consulta:",
err.message);
  }

  res.status(200).send({ msg: "ok" });
});

module.exports = router;
```

server.js

```
const conectarBanco = require("../../banco/conexao");
const app = require("./app");

conectarBanco();

app.listen(6000, () => console.log("Consulta. Porta 6000"));
```

servicoConsulta.js

```
// consulta/src/servicoConsulta.js
const Lembrete = require("../../banco/lembreteModel");
const Observacao = require("../../banco/observacaoModel");
```

```
// Classe
class ConsultaService {
  // Retorna lembretes + observações
  async listarLembretesComObservacoes() {
    const lembretes = await Lembrete.find();
    const resultado = {};

    for (const lembrete of lembretes) {
      const observacoes = await Observacao.find({ lembreteId:
lembrete.id });

      resultado[lembrete.id] = {
        id: lembrete.id,
        texto: lembrete.texto,
        status: lembrete.status,
        observacoes,
      };
    }

    return resultado;
  }
}

// Instância
const consultaService = new ConsultaService();

// Exportação
module.exports = {
  listarLembretesComObservacoes: () =>
    consultaService.listarLembretesComObservacoes(),
};
```

frontend

index.html

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <meta charset="UTF-8" />
  <title>Lembretes</title>
  <link rel="stylesheet" href="style.css" />
</head>
<body>
```

```
  <h1>Meus Lembretes</h1>

  <form id="formLembrete">
    <input type="text" id="novoLembrete" placeholder="Digite um
lembrete" required />
    <button type="button" id="btnImportanteLembrete"
class="important">Importante</button>
    <button type="submit">Adicionar</button>
  </form>

  <ul id="listaLembretes"></ul>

  <script src="script.js"></script>
</body>
</html>
```

script.js

```
const API_LEMBRETES = "http://localhost:4000";
const API_OBSERVACOES = "http://localhost:5000";

const lista = document.getElementById("listaLembretes");
const form = document.getElementById("formLembrete");
const input = document.getElementById("novoLembrete");
const btnImportanteLembrete =
document.getElementById("btnImportanteLembrete");

let lembreteImportante = false;

// alterna o visual do importante
btnImportanteLembrete.addEventListener("click", () => {
  lembreteImportante = !lembreteImportante;
  btnImportanteLembrete.classList.toggle("active", lembreteImportante);
});

async function carregarLembretes() {
  lista.innerHTML = "";

  try {
    const res = await fetch(`${API_LEMBRETES}/lembretes`);
    const lembretes = await res.json();

    for (const lembrete of lembretes) {
      const li = document.createElement("li");
```

```
      const titulo = document.createElement("strong");
      titulo.textContent = lembrete.texto;
      li.appendChild(titulo);

      if (lembrete.status === "importante") {
        const tag = document.createElement("span");
        tag.textContent = "IMPORTANTE";
        tag.classList.add("tag");
        li.appendChild(tag);
      }

      const btnDel = document.createElement("button");
      btnDel.textContent = "Excluir";
      btnDel.classList.add("delete");
      btnDel.addEventListener("click", async () => {
        await fetch(`${API_LEMBRETES}/lembretes/${lembrete.id}`, {
method: "DELETE" });
        carregarLembretes();
      });
      li.appendChild(btnDel);

      const obsDiv = document.createElement("div");
      obsDiv.classList.add("observacoes");

      const obsRes = await
fetch(`${API_OBSERVACOES}/lembretes/${lembrete.id}/observacoes`);
      const observacoes = await obsRes.json();

      observacoes.forEach(obs => {
        const p = document.createElement("p");

        const spanTxt = document.createElement("span");
        spanTxt.textContent = obs.texto;
        p.appendChild(spanTxt);

        if (obs.status === "importante") {
          const tag = document.createElement("span");
          tag.textContent = "IMPORTANTE";
          tag.classList.add("tag");
          p.appendChild(tag);
        }
```

```
        const btnDelObs = document.createElement("button");
        btnDelObs.textContent = "Excluir";
        btnDelObs.classList.add("delete");
        btnDelObs.addEventListener("click", async () => {
          await fetch(`${API_OBSERVACOES}/observacoes/${obs.id}`, {
method: "DELETE" });
          carregarLembretes();
        });
        p.appendChild(btnDelObs);

        obsDiv.appendChild(p);
      });

      // Form de nova observação
      const formObs = document.createElement("form");
      formObs.style.display = "flex";
      formObs.style.gap = "8px";

      formObs.innerHTML = `
        <input type="text" placeholder="Nova observação" required />
        <button type="button" class="btnObsImportant
important">Importante</button>
        <button type="submit" class="addObs">Adicionar</button>
      `;

      let obsImportante = false;
      const btnObsImportant =
formObs.querySelector(".btnObsImportant");

      btnObsImportant.addEventListener("click", () => {
        obsImportante = !obsImportante;
        btnObsImportant.classList.toggle("active", obsImportante);
      });

      formObs.addEventListener("submit", async (e) => {
        e.preventDefault();

        const textoObs = formObs.querySelector("input").value.trim();
        if (!textoObs) return;

        await
fetch(`${API_OBSERVACOES}/lembretes/${lembrete.id}/observacoes`, {
          method: "PUT",
```

```
          headers: { "Content-Type": "application/json" },
          body: JSON.stringify({
            texto: textoObs,
            status: obsImportante ? "importante" : "comum"
          })
        });

        carregarLembretes();
      });

      obsDiv.appendChild(formObs);
      li.appendChild(obsDiv);
      lista.appendChild(li);
    }
  } catch (err) {
    console.error("Erro:", err);
  }
}

form.addEventListener("submit", async (e) => {
  e.preventDefault();

  const texto = input.value.trim();
  if (!texto) return;

  await fetch(`${API_LEMBRETES}/lembretes`, {
    method: "PUT",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({
      texto,
      status: lembreteImportante ? "importante" : "comum"
    })
  });

  input.value = "";
  lembreteImportante = false;
  btnImportanteLembrete.classList.remove("active");

  carregarLembretes();
});

carregarLembretes();
```

style.css

```css
/* ==== RESET ==== */
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
  font-family: "Inter", Arial, sans-serif;
}

/* ==== BODY ==== */
body {
  background: #eef1f5;
  padding: 30px;
}

/* ==== TÍTULO ==== */
h1 {
  text-align: center;
  font-size: 32px;
  margin-bottom: 25px;
  color: #2c3e50;
}

/* ==== FORM PRINCIPAL ==== */
form {
  display: flex;
  justify-content: center;
  gap: 10px;
  margin-bottom: 25px;
}

input {
  padding: 10px;
  width: 330px;
  border-radius: 8px;
  border: 1px solid #ccc;
  background: #fff;
}

/* ==== BOTÕES ==== */
button {
  padding: 10px 14px;
```

```
  border-radius: 8px;
  border: none;
  cursor: pointer;
  font-weight: bold;
  transition: 0.15s ease;
}

/* Botão importante */
button.important {
  background: #ffdd57;
}
button.important:hover {
  background: #f1c40f;
}
button.important.active {
  background: #d35400;
  color: white;
}

/* Botão adicionar */
button[type="submit"] {
  background: #3498db;
  color: white;
}
button[type="submit"]:hover {
  background: #2980b9;
}

/* ==== LISTA ==== */
ul {
  list-style: none;
  max-width: 680px;
  margin: 0 auto;
}

/* ==== CARTÃO DE LEMBRETE ==== */
li {
  background: #fff;
  padding: 16px;
  margin-bottom: 15px;
  border-radius: 12px;
  box-shadow: 0 3px 8px rgba(0,0,0,0.08);
}
```

```css
/* Título do lembrete */
li strong {
  font-size: 18px;
  color: #2c3e50;
}

/* TAG DE IMPORTANTE */
.tag {
  margin-left: 10px;
  padding: 3px 10px;
  background: #f1c40f;
  border-radius: 6px;
  color: #000;
  font-size: 12px;
  font-weight: bold;
}

/* ==== BOTÕES DENTRO DO CARD ==== */
button.delete,
button.addObs {
  margin-left: 10px;
  padding: 7px 12px;
  border-radius: 6px;
}

button.delete {
  background: #e74c3c;
  color: white;
}
button.delete:hover {
  background: #c0392b;
}

button.addObs {
  background: #3498db;
  color: white;
}
button.addObs:hover {
  background: #2980b9;
}

/* ==== OBSERVAÇÕES ==== */
```

```css
.observacoes {
  margin-top: 15px;
  padding-left: 12px;
  border-left: 3px solid #dcdcdc;
}

.observacoes p {
  background: #f9f9f9;
  padding: 10px;
  border-radius: 6px;
  margin-bottom: 8px;
  display: flex;
  align-items: center;
  gap: 10px;
}
```

lembretes

distribuidorEventos.js

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

async function enviarEvento(tipo, dados) {
  await axios.post("http://localhost:10000/eventos", {
    tipo,
    dados,
  });
}

module.exports = { enviarEvento };
```

routes.js

```
const observacoesService =
require("../../observacoes/src/servicoObservacoes");
  const express = require("express");
  const axios = require("axios");
  const lembreteService = require("./servicoLembretes");

  const router = express.Router();

  // Lista todos os lembretes
  router.get("/lembretes", async (req, res) => {
    try {
      const lista = await lembreteService.listarLembretes();
```

```
      res.send(lista);
    } catch (err) {
      console.error("Erro ao listar lembretes:", err);
      res.status(500).send({ erro: "Erro ao listar lembretes." });
    }
  });

  // Cria um novo lembrete
  router.put("/lembretes", async (req, res) => {
    const { texto, status } = req.body;
    try {
      const lembrete = await lembreteService.criarLembrete(texto,
status);
      res.status(201).send(lembrete);
    } catch (err) {
      res.status(500).send({ erro: err.message });
    }
  });

  // Criar observação
  router.put("/lembretes/:id/observacoes", async (req, res) => {
    const { texto, status } = req.body;
    try {
      const obs = await
observacoesService.criarObservacao(req.params.id, texto, status);
      res.status(201).send(obs);
    } catch (err) {
      res.status(500).send({ erro: err.message });
    }
  });

  // Recebe eventos do barramento
  router.post("/eventos", async (req, res) => {
    const { tipo, dados } = req.body;
    console.log("Lembretes recebeu evento:", tipo, dados);

    try {
      await lembreteService.processarEvento(tipo, dados);
      res.status(200).send({ msg: "ok" });
    } catch (err) {
      console.error("Erro ao processar evento:", err);
      res.status(500).send({ erro: "Erro ao processar evento." });
    }
```

```
  });

  // Excluir lembrete
  router.delete("/lembretes/:id", async (req, res) => {
    try {
      const id = req.params.id;

      // Enviar evento para excluir
      await axios.post("http://localhost:10000/eventos", {
        tipo: "LembreteExcluido",
        dados: { id }
      });

      res.send({ msg: "Lembrete excluído (evento enviado ao
barramento)" });
    } catch (err) {
      console.error("Erro ao excluir lembrete:", err);
      res.status(500).send({ erro: "Erro ao excluir lembrete." });
    }
  });

  module.exports = router;
```

server.js

```
const express = require("express");
const routes = require("./routes");
const cors = require("cors");

const app = express();
app.use(express.json());

// conectarBanco();
app.use(cors());
app.use(routes);

if (require.main === module) {
  const conectarBanco = require("../../banco/conexao");
  conectarBanco();
  app.listen(4000, () => console.log("Lembretes. Porta 4000"));
}

module.exports = app;
```

servicoLembretes.js

```
// servicoLembretes.js
const Lembrete = require("../../banco/lembreteModel");
const { enviarEvento } = require("./distribuidorEventos");

// Classe POO lembretes
class LembretesService {
  constructor() {
    this.funcoes = {
      LembreteClassificado: async (dados) => {
        await this.atualizarStatusLembrete(dados.id, dados.status);
      }
    };
  }

  // Gera ID incremental
  async gerarIdSequencial() {
    const ultimo = await Lembrete.findOne().sort({ id: -1 }).lean();
    const ultimoId =
      ultimo && typeof ultimo.id !== "undefined"
        ? Number(ultimo.id)
        : 0;

    return Number.isFinite(ultimoId) ? ultimoId + 1 : 1;
  }

  // Cria lembrete e envia evento
async criarLembrete(texto, status = "aguardando") {
  const id = await this.gerarIdSequencial();

  const novo = await Lembrete.create({
    id,
    texto,
    status,
  });

  await enviarEvento("LembreteCriado", {
    id: novo.id,
    texto: novo.texto,
    status: novo.status
  });

  return novo;
```

```
}

  // Lista todos os lembretes
  async listarLembretes() {
    return Lembrete.find();
  }

  // Atualiza status após classificação
  async atualizarStatusLembrete(id, status) {
    const lembrete = await Lembrete.findOne({ id });
    if (!lembrete) return;

    lembrete.status = status;
    await lembrete.save();

    await enviarEvento("LembreteAtualizado", {
      id: lembrete.id,
      texto: lembrete.texto,
      status: lembrete.status
    });
  }

  // Processa eventos recebidos
  async processarEvento(tipo, dados) {
    const fn = this.funcoes[tipo];
    if (fn) await fn(dados);
  }
}


const lembretesService = new LembretesService();


module.exports = {
  criarLembrete: (texto, status) =>
lembretesService.criarLembrete(texto, status),
  listarLembretes: () => lembretesService.listarLembretes(),
  processarEvento: (tipo, dados) =>
lembretesService.processarEvento(tipo, dados)
};
```

observacoes

distribuidorEventos.js

```
// src/distribuidorEventos.js
const axios = require("axios");

async function enviarEvento(tipo, dados) {
  try {
    await axios.post("http://localhost:10000/eventos", {
      tipo,
      dados
    });
  } catch (err) {
    console.error("Erro ao enviar evento:", err.message);
  }
}

module.exports = { enviarEvento };
```

routes.js

```
// src/routes.js
const express = require("express");
const axios = require("axios");
const router = express.Router();

const {
  criarObservacao,
  listarObservacoes,
  processarEvento
} = require("./servicoObservacoes");

// Cria observação com PUT
router.put("/lembretes/:id/observacoes", async (req, res) => {
  try {
    const { texto, status } = req.body;

    if (!texto) return res.status(400).send({ erro: "O campo 'texto' é
obrigatório." });

    const observacao = await criarObservacao(
      req.params.id,
      texto,
      status || "comum"
    );
```

```
    res.status(201).send(observacao);
  } catch (err) {
    console.error("Erro ao criar observação:", err);
    res.status(500).send({ erro: "Erro ao criar observação." });
  }
});

// Listar observações
router.get("/lembretes/:id/observacoes", async (req, res) => {
  try {
    const observacoes = await listarObservacoes(req.params.id);
    res.send(observacoes);
  } catch (err) {
    console.error("Erro ao listar observações:", err);
    res.status(500).send({ erro: "Erro ao listar observações." });
  }
});

// Receber eventos do barramento
router.post("/eventos", async (req, res) => {
  try {
    const { tipo, dados } = req.body;
     console.log(" Observações recebeu:", tipo, dados);
    if (!tipo) {
      return res.status(400).send({ erro: "Evento sem tipo" });
    }

    await processarEvento(tipo, dados);
    res.send({ msg: "ok" });
  } catch (err) {
    console.error("Erro ao processar evento:", err);
    res.status(500).send({ erro: "Erro ao processar evento." });
  }
});

// Excluir observação
router.delete("/observacoes/:id", async (req, res) => {
  try {
    const id = req.params.id;

    // Evento para o barramento
    await axios.post("http://localhost:10000/eventos", {
      tipo: "ObservacaoExcluida",
```

```
      dados: { id }
    });

    res.send({ msg: "Observação excluída (evento enviado ao
barramento)" });
  } catch (err) {
    console.error("Erro ao excluir observação:", err);
    res.status(500).send({ erro: "Erro ao excluir observação." });
  }
});


module.exports = router;
```

server.js

```
// src/server.js
const conectarBanco = require("../../banco/conexao");
const express = require("express");
const routes = require("./routes");
const cors = require("cors");

const app = express();
app.use(express.json());

conectarBanco();
app.use(cors());
app.use(routes);

app.listen(5000, () => console.log("Observações. Porta 5000"));
```

servicoObservacoes.js

```
// servicoObservacoes.js
const Observacao = require("../../banco/ObservacaoModel");
const { enviarEvento } = require("./distribuidorEventos");
const { v4: uuidv4 } = require("uuid");

class ObservacoesService {
  constructor() {
    this.funcoes = {
      ObservacaoClassificada: async (observacao) => {
        await this.atualizarStatusObservacao(observacao.id,
observacao.status);
      }
```

```
    };
  }

  // Cria nova observação
async criarObservacao(lembreteId, texto, status = "aguardando") {
  const id = uuidv4();

  const nova = await Observacao.create({
    id,
    texto,
    lembreteId,
    status, // <- usar status recebido
  });

  await enviarEvento("ObservacaoCriada", {
    id,
    texto,
    lembreteId,
    status,
  });

  return nova;
}


  // Lista observações de um lembrete específico
  async listarObservacoes(lembreteId) {
    return Observacao.find({ lembreteId });
  }

  // Atualiza status da observação
  async atualizarStatusObservacao(id, status) {
    const obs = await Observacao.findOne({ id });
    if (!obs) return;

    obs.status = status;
    await obs.save();

    await enviarEvento("ObservacaoAtualizada", {
      id: obs.id,
      texto: obs.texto,
      lembreteId: obs.lembreteId,
      status: obs.status,
```

```
    });
  }

  // Processa eventos recebidos
  async processarEvento(tipo, dados) {
    const fn = this.funcoes[tipo];
    if (fn) await fn(dados);
  }
}


const observacoesService = new ObservacoesService();

// Exportação
module.exports = {
  criarObservacao: (lembreteId, texto, status) =>
observacoesService.criarObservacao(lembreteId, texto, status),
  listarObservacoes: (lembreteId) =>
observacoesService.listarObservacoes(lembreteId),
  processarEvento: (tipo, dados) =>
observacoesService.processarEvento(tipo, dados)
};
```