

Dendê Eventos: Solução POO em Gestão de Eventos

Autor: Anna Beatriz Silva Lima
Fátima Pereira Santos Pinho
Ian Salomão da Silva Carneiro
Rebeca Helen Batista Amorim
Valeria Soares Santos

Agenda

Demonstrar a arquitetura técnica e as regras de negócio da API Dendê Eventos, validando a modelagem orientada a objetos e a implementação de endpoints REST funcionais.

1. A Dendê Eventos:

Visão Geral do Produto e Contexto

2. Diagrama de Classes:

Estrutura e Relacionamentos

3. Mapeamento:

Classes, Atributos e Métodos

4. Endpoints e Verbos HTTP

Rotas para gestão e fluxo de entidades

5. Regras de Negócio Implementadas

Garantias de funcionamento do sistema.

6. Demonstração e Validação:

Simulação de cenários reais de uso através do Postman.

A Dendê Eventos:



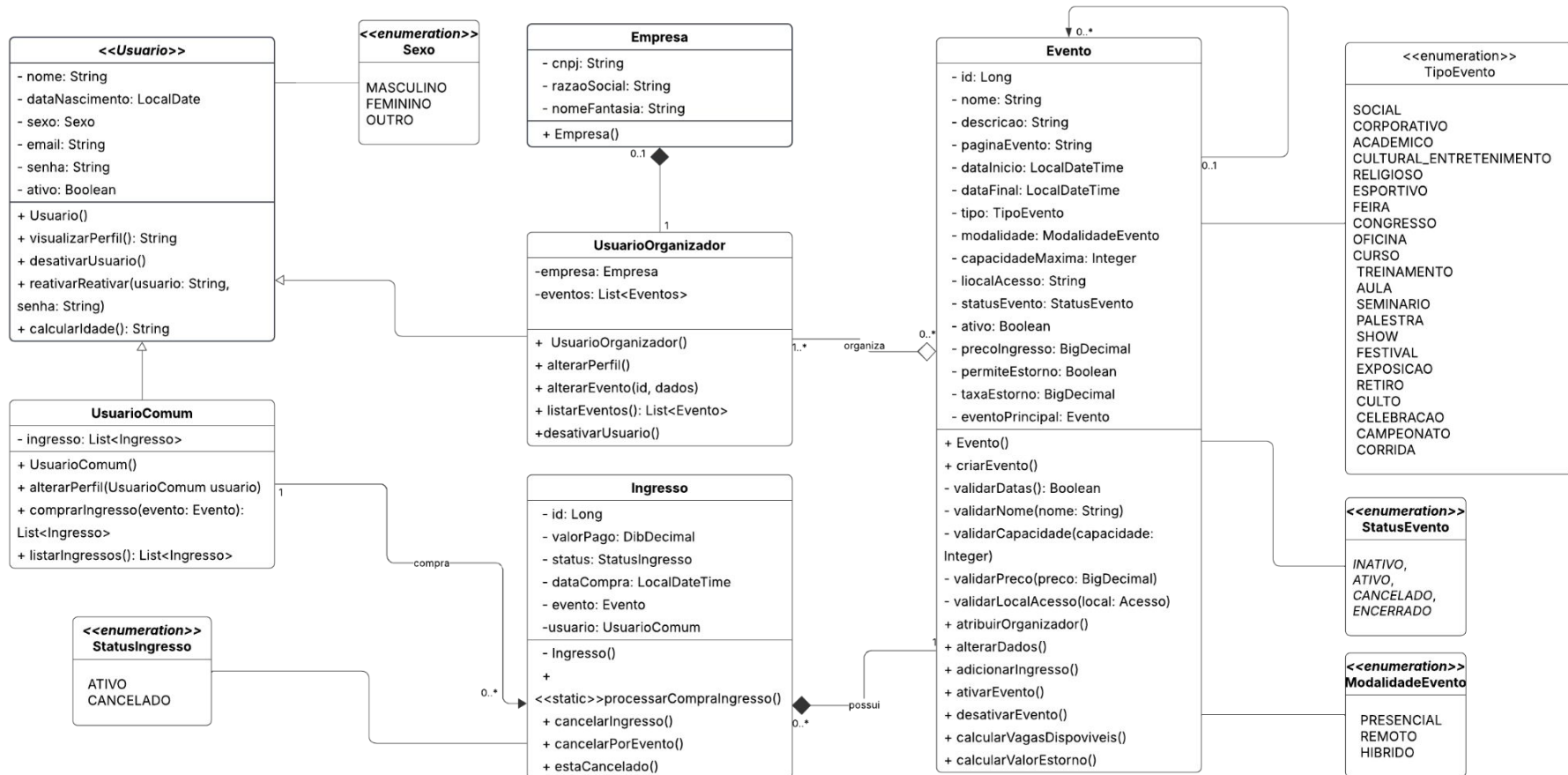
- **A empresa:** Uma softhouse baiana focada em soluções digitais com impacto local.
- **O produto:** Uma plataforma completa para divulgação, organização e gestão de eventos.
- **A missão:** construir um backend robusto em Java, garantindo o fluxo seguro de cadastros, eventos e venda de ingressos.



Diagrama de Classes



- **Paradigma Orientado a Objetos:** Aplicação de abstração, encapsulamento e herança na modelagem.
- **Hierarquia de Usuários:**
 - **Superclasse Usuario:** Centraliza atributos e métodos comuns.
 - **Subclasses:** Derivação em UsuarioComum e UsuarioOrganizador.
- **Mapeamento de Relacionamentos:**
 - **Composição (1 : 0..1):** Vínculo entre UsuarioOrganizador e Empresa.
 - **Associações Diretas (1 : 0..*):** Listas de Ingressos e Eventos.
 - **Auto-relacionamento (0..1):** Suporte a sub-eventos na classe Evento.



Dendê Events: Solução POO em Gestão de Eventos

Mapeamento de Domínio



Tipagem e Nomenclatura

- **Convenções e Padrões (Java):** Adoção de camelCase e nomenclaturas que refletem ações de negócio claras.
- **Tipagem Estratégica:**
 - BigDecimal: Precisão em operações financeiras (preços e taxas).
 - LocalDateTime e LocalDate: Manipulação cronológica exata.

```
public class Ingresso {  
  
    private Long id;  
    private BigDecimal valorPago;  
    private StatusIngresso status;  
    private LocalDateTime dataCompra;  
    private Evento evento;  
    private UsuarioComum usuario;  
}
```

```
public int calcularVagasDisponiveis(){  
    long ativos = this.ingressos.stream()  
        .filter(i -> i.getStatus() == StatusIngresso.ATIVO)  
        .count();  
    return this.capacidadeMaxima - (int) ativos;  
}
```

Preservação de Relacionamentos no Código

- **Mapeamento de Multiplicidade:** Tradução fiel das cardinalidades do diagrama UML para as estruturas de dados no backend.
- **Associações e Composições em Java:**
 - **Relação (1 : 0..1):** Instanciação direta de objetos (ex: Empresa em UsuarioOrganizador).
 - **Relação (1 : 0..*):** Utilização de List<> (Coleções) para agregação de múltiplos itens.

```
public class UsuarioOrganizador extends Usuario {  
  
    private Empresa empresa;  
    private final List<Evento> eventos = new ArrayList<>();  
  
    public UsuarioOrganizador() {  
        super();  
    }  
  
    public UsuarioOrganizador(Long id, String... ) {  
        super(id, nome, dataNascimento, sexo, email, senha);  
        this.empresa = empresa;  
    }  
}
```

Arquitetura de Endpoints e Padrão REST



- **Construção de Rotas:** Mapeamento semântico de recursos baseado em substantivos e hierarquias lógicas (ex: /usuarios/{email}/ingressos).
- **Semântica dos Verbos HTTP:**
 - **POST:** Criação e processamento de novas entidades.
 - **GET:** Recuperação segura e idempotente de recursos.
 - **PUT:** Substituição ou atualização integral de uma entidade.
 - **PATCH:** Modificação parcial e transições de estado específicas.

POST

- **Propósito:** Inserção de novos dados no domínio (Cadastros, Compra de Ingressos).

```
@PostMapping
public ResponseEntity<String> cadastrarUsuario(@RequestBody UsuarioComum usuarioComum) {
    try {
        repositorio.salvarUsuario(usuarioComum);
        return ResponseEntity.ok("Usuario " + usuarioComum.getEmail() + " cadastrado com sucesso!");
    } catch (IllegalArgumentException e) {
        return ResponseEntity.status(400, e.getMessage());
    }
}
```

GET

- **Propósito:** Consulta idempotente e segura (Feed, Visualização de Perfil, Listagens).

```
@GetMapping
public ResponseEntity<List<EventoResponseDTO>> feedEventos() {
    List<EventoResponseDTO> lista = repositorio.feedEventos()
        .stream().map(evento -> new EventoResponseDTO(
            evento.getNome(),
            evento.getDescricao(),
            evento.getDataInicio(),
            evento.getDataFinal(),
            evento.getOrganizador().getNome()
        ))
        .toList();

    return ResponseEntity.ok(lista);
}
```

- **Propósito:** Alteração completa de um recurso existente.

```
@PutMapping(path = "{email}/eventos/{eventoId}")
public ResponseEntity<String> alterarEvento(
    @PathVariable(parameter = "email") String email,
    @PathVariable(parameter = "eventoId") long eventoId,
    @RequestBody Evento novosDados) {
    UsuarioOrganizador organizador = repositorio.buscarOrganizador(email);
    if (organizador == null) return ResponseEntity.status(404, "Organizador nao encontrado.");
    try {
        repositorio.buscarEventoPorId(eventoId);
        organizador.alterarEvento(eventoId, novosDados);
        return ResponseEntity.ok("Evento alterado com sucesso.");
    } catch (IllegalArgumentException e) {
        if (e.getMessage().contains("não encontrado")) return ResponseEntity.status(404, e.getMessage());
        return ResponseEntity.status(400, e.getMessage());
    }
}
```

PATCH

- **Propósito:** Modificações parciais ou transições de estado de negocio (Ativar/Inativar/Cancelar).

```
@PatchMapping(path =("/{email}/desativar")
public ResponseEntity<String> desativarUsuario(@PathVariable(parameter = "email") String email) {
    Usuario usuario = repositorio.buscarUsuarioComum(email);
    if (usuario == null) return ResponseEntity.status(404, "Usuario nao encontrado.");
    if (!usuario.isAtivo()) return ResponseEntity.status(400, "Usuario ja esta inativo.");
    try {
        usuario.desativarUsuario();
        return ResponseEntity.ok("Usuario desativado com sucesso.");
    } catch (IllegalArgumentException e) {
        return ResponseEntity.status(400, e.getMessage());
    }
}
```

Regras de Negócio



Usuários e Validações Temporais

- **Gestão de Usuários:**

- Garantia de unicidade de e-mails na plataforma.
- Bloqueio de inativação para organizadores com eventos ativos.
- Cálculo dinâmico de idade (Anos, Meses e Dias) na visualização do perfil.

- **Integridade de Eventos:**

- **Validação estrita de cronogramas:** fim posterior ao início, ambos no futuro.
- Restrição de duração mínima estipulada em **30 minutos**.

```
public String calcularIdade() {  
    if (dataNascimento == null) return null;  
    Period periodo = Period.  
        between(dataNascimento, LocalDate.now());  
    return periodo.getYears() + " anos, "  
        + periodo.getMonths() + " meses e "  
        + periodo.getDays() + " dias";  
}
```

```
private void validarDatas(LocalDateTime dataInicio, LocalDateTime dataFinal) {  
  
    if (dataInicio == null || dataFinal == null)  
        throw new IllegalArgumentException(  
            "Datas e Horários não podem ser nulos.");  
    if (dataInicio.isBefore(LocalDate.now()))  
        throw new IllegalArgumentException(  
            "Data e horário iniciais não podem ser anteriores as atuais");  
  
    long duracaoMinutos = Duration.between(dataInicio, dataFinal).toMinutes();  
    if (duracaoMinutos < 0)  
        throw new IllegalArgumentException(  
            "Data e horário finais não podem ser " +  
            "anteriores a data e horário iniciais.");  
    if (duracaoMinutos < 30) throw new IllegalArgumentException("Evento não pode " +  
        "durar menos de 30 min.");  
}
```

Regras de Negócio



Transações e Listagens Estruturadas

- **Transações (Ingressos e Estornos):**
 - **Compras combinadas:** processamento atômico de ingressos para evento principal e sub-evento.
 - **Motor de cancelamento:** cálculo percentual dinâmico para estornos baseado nas regras do evento.
- **Máquina de Estados e Listagens:**
 - Feeds públicos **ocultam eventos inativos ou lotados.**
 - **Implementação de interfaces Comparator para ordenação multicritério:** prioridade de status, data e ordem alfabética.



```
public double calcularValorEstorno(Ingresso ingresso) {  
    if (permiteEstorno == null || !permiteEstorno) {  
        return 0;  
    }  
    if (taxaEstorno == null) {  
        return ingresso.getValorPago().doubleValue();  
    }  
    double valor = ingresso.getValorPago().doubleValue();  
    return valor - (valor * this.taxaEstorno.doubleValue());  
}
```

Validação Prática



Cadastro de usuário

```
1  {
2    "nome": "Lucas Almeida Silva",
3    "dataNascimento": "1994-11-11",
4    "sexo": "MASCULINO",
5    "email": "lucas@mail.com",
6    "senha": "senha123"
7  }
```

Body 200 OK • 25 ms • 124 B • | Save Response

Raw Preview Visualize |

```
1  "Usuario lucas@mail.com cadastrado com sucesso!"
```

Validação Prática

Cadastro de usuário com email duplicado

```
1  {  
2    "nome": "Outro Lucas",  
3    "dataNascimento": "1990-01-01",  
4    "sexo": "MASCULINO",  
5    "email": "lucas@mail.com",  
6    "senha": "senha999"  
7  }
```

Body ▾ ↺

400 Bad Request

•

11 ms

•

138 B

•



e.g. Save Response



Raw ▾

▶ Preview

🐞 Debug with AI ▾



1 "Já existe um usuário com o e-mail: lucas@mail.com"

Validação Prática



Cadastro de organizador (CPF e CNPJ)

```
1 {  
2   "nome": "Carlos Organizador",  
3   "dataNascimento": "1985-06-20",  
4   "sexo": "MASCULINO",  
5   "email": "org@mail.com",  
6   "senha": "senha123"  
7 }
```

Body 200 OK • 39 ms • 126 B • | Save Response

Raw Preview Visualize

```
1 "Organizador org@mail.com cadastrado com sucesso!"
```

```
1 {  
2   "nome": "Carlos Organizador Empresa",  
3   "dataNascimento": "1985-06-20",  
4   "sexo": "MASCULINO",  
5   "email": "empresa@mail.com",  
6   "senha": "senha123",  
7   "empresa": {  
8     "cnpj": "12.345.678/0001-99",  
9     "razaoSocial": "Eventos Ltda",  
10    "nomeFantasia": "Eventos Show"  
11  }  
12 }
```

Body 200 OK • 19 ms • 130 B • | Save Response

JSON Preview Visualize

```
1 "Organizador empresa@mail.com cadastrado com sucesso!"
```

Cloud View Console Terminal

Runner Vault

Validação Prática



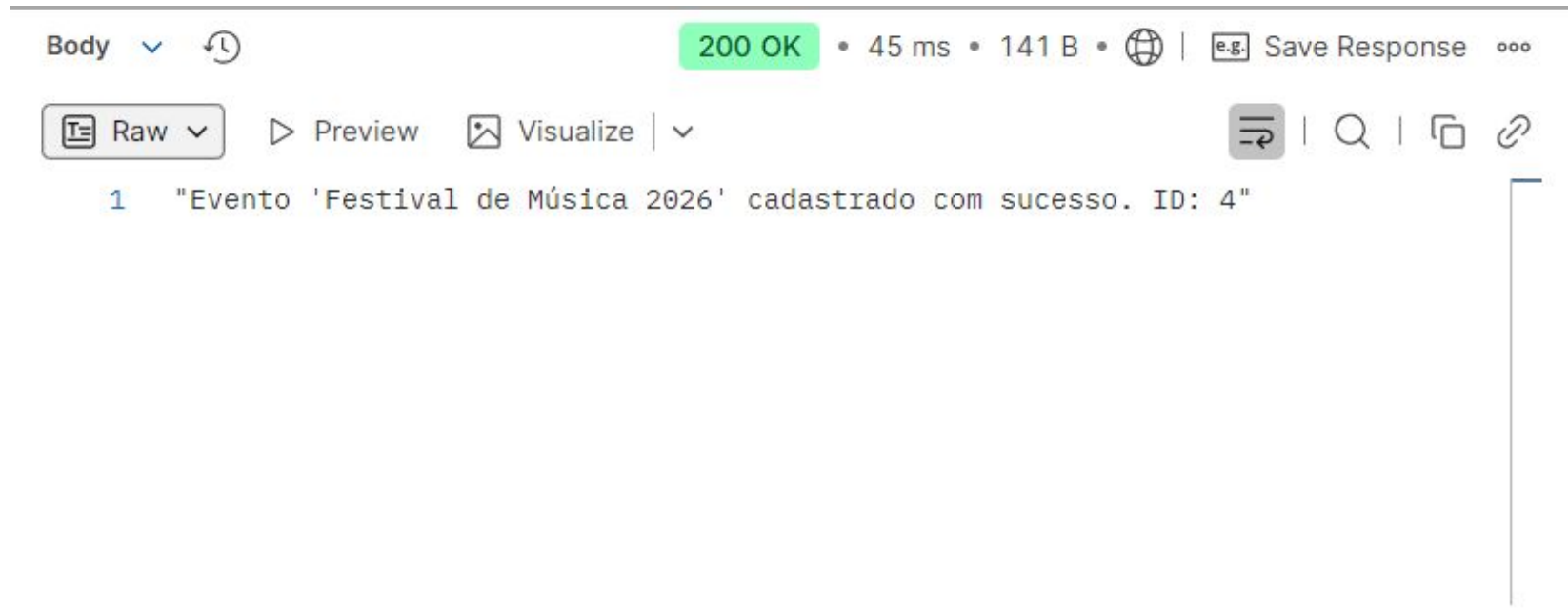
Desativar organizador com eventos ativos



Validação Prática



Cadastro de evento



Validação Prática

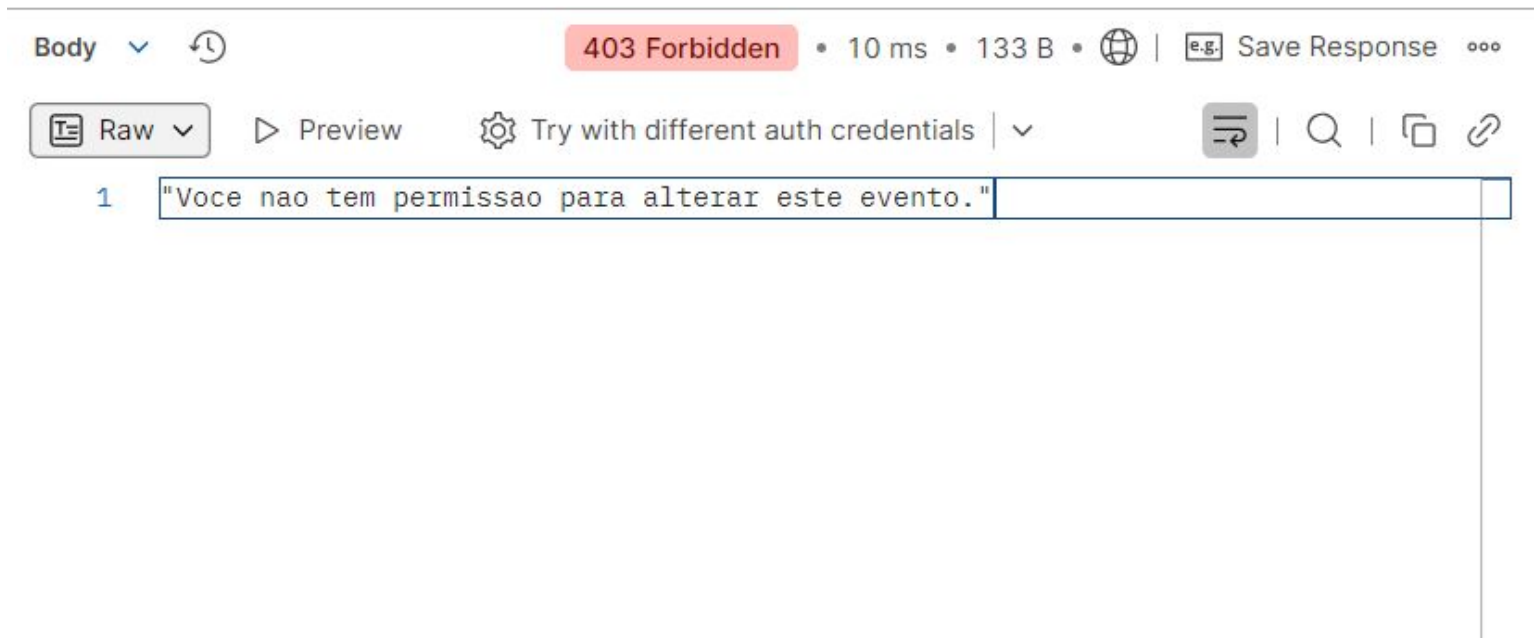
Feed de Eventos

```
Body 200 OK • 4 ms • 800 B | Save Response ...
{} JSON Preview Visualize
1  [
2      {
3          "nome": "Festival de Comida 2026",
4          "descricao": "O maior festival de comida do ano com culinárias de
5              vários países.",
6          "dataHoraInicio": "2026-06-15T18:00:00",
7          "dataHoraFim": "2026-06-15T23:00:00",
8          "nomeOrganizador": "Carlos Organizador"
9      },
10     {
11         "nome": "Festival de Música 2026",
12         "descricao": "O maior festival de música do ano com várias atrações
13             nacionais.",
14         "dataHoraInicio": "2026-06-15T18:00:00",
15         "dataHoraFim": "2026-06-15T23:00:00",
16         "nomeOrganizador": "Carlos Organizador"
17     },
18     {
19         "nome": "IntegraSI 2026",
20         "descricao": "Semana de Integração com atividades entre veteranos e
21             calouros do curso de Sistemas de Informação.",
```

Validação Prática



Um organizador pode alterar apenas o próprio evento



Validação Prática



Compra de ingresso com evento inativo



Validação Prática



Tentativa de cancelar um ingresso já cancelado



Referências



- ORACLE. **Java Platform, Standard Edition 17: API Specification.** [Redwood Shores]: Oracle, 2021.
Disponível em: <https://docs.oracle.com/en/java/javase/17/docs/api/index.html>. Acesso em: 20 fev. 2026.