Centro Universitário de Excelência
Sistemas de Informação

Dendê Eventos API

Autor: Lucas Ferreira Lan
Cesár Filipe Gomes
Rodrigo Souza Guimarães
Italo Yan Mendes da Silva
Valnei Souza

Feira de Santana, 2026

Agenda

Apresentar a implementação das principais rotas da API, destacando a arquitetura adotada, o uso do padrão DTO, segurança e eficiência na comunicação entre as camadas do sistema.

1. Contexto:
Contextualização do projeto, implementações e regras.

2. Diagrama de Classe:
Apresentação do diagrama de classe.

3. DTO:
Definição e importância do Data Transfer Object.

4. Arquitetura de endpoints:
Apresentação da arquitetura dos endpoints e seus métodos.

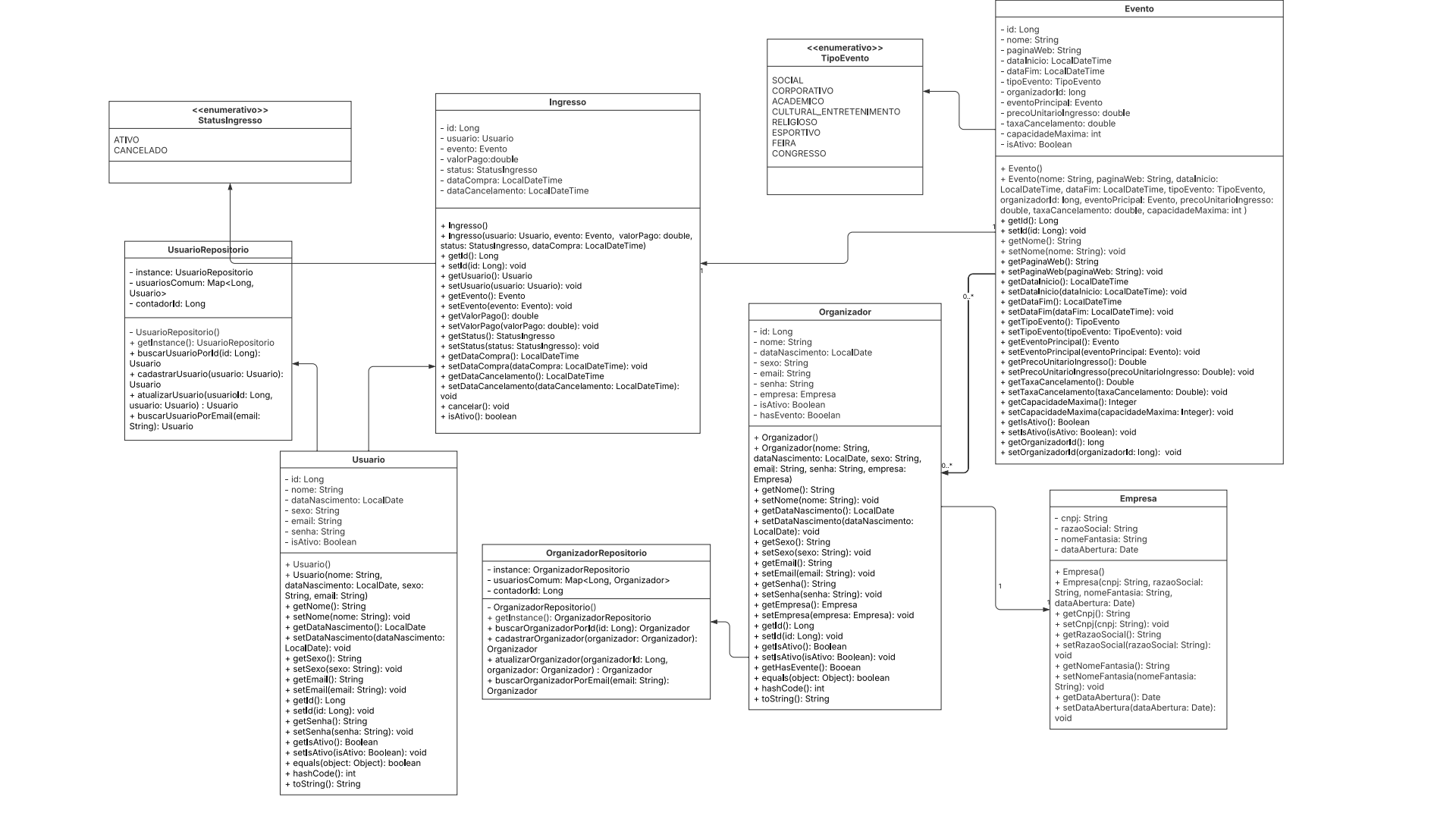
5. Teste funcional:
Resultados dos testes feitos em algumas rotas.

5. Referências:
Referências do projeto.

Contexto

- Dendê Eventos API
- Implementação de endpoints
- Sem frameworks externos

Diagrama de Classe


<<enumerativo>>
StatusIngresso
ATIVO
CANCELADO
<<enumerativo>>
TipoEvento
SOCIAL
CORPORATIVO
ACADEMICO
CULTURAL_ENTRETENIMENTO
RELIGIOSO
ESPORTIVO
FEIRA
CONGRESSO
Evento
- id: Long
- nome: String
- paginaWeb: String
- dataInicio: LocalDateTime
- dataFim: LocalDateTime
- tipoEvento: TipoEvento
- organizadorId: long
- eventoPrincipal: Evento
- precoUnitarioIngresso: double
- taxaCancelamento: double
- capacidadeMaxima: int
- isAtivo: Boolean
+ Evento()
+ Evento(nome: String, paginaWeb: String, dataInicio: LocalDateTime, dataFim: LocalDateTime, tipoEvento: TipoEvento, organizadorId: long, eventoPricipal: Evento, precoUnitarioIngresso: double, taxaCancelamento: double, capacidadeMaxima: int)
+ getId(): Long
+ setId(id: Long): void
+ getNome(): String
+ setNome(nome: String): void
+ getPaginaWeb(): String
+ setPaginaWeb(paginaWeb: String): void
+ getDataInicio(): LocalDateTime
+ setDataInicio(dataInicio: LocalDateTime): void
+ getDataFim(): LocalDateTime
+ setDataFim(dataFim: LocalDateTime): void
+ getTipoEvento(): TipoEvento
+ setTipoEvento(tipoEvento: TipoEvento): void
+ getEventoPrincipal(): Evento
+ setEventoPrincipal(eventoPrincipal: Evento): void
+ getPrecoUnitarioIngresso(): Double
+ setPrecoUnitarioIngresso(precoUnitarioIngresso: Double): void
+ getTaxaCancelamento(): Double
+ setTaxaCancelamento(taxaCancelamento: Double): void
+ getCapacidadeMaxima(): Integer
+ setCapacidadeMaxima(capacidadeMaxima: Integer): void
+ getIsAtivo(): Boolean
+ setIsAtivo(isAtivo: Boolean): void
+ getOrganizadorId(): long
+ setOrganizadorId(organizadorId: long): void
Ingresso
- id: Long
- usuario: Usuario
- evento: Evento
- valorPago:double
- status: StatusIngresso
- dataCompra: LocalDateTime
- dataCancelamento: LocalDateTime
+ Ingresso()
+ Ingresso(usuario: Usuario, evento: Evento, valorPago: double, status: StatusIngresso, dataCompra: LocalDateTime)
+ getId(): Long
+ setId(id: Long): void
+ getUsuario(): Usuario
+ setUsuario(usuario: Usuario): void
+ getEvento(): Evento
+ setEvento(evento: Evento): void
+ getValorPago(): double
+ setValorPago(valorPago: double): void
+ getStatus(): StatusIngresso
+ setStatus(status: StatusIngresso): void
+ getDataCompra(): LocalDateTime
+ setDataCompra(dataCompra: LocalDateTime): void
+ getDataCancelamento(): LocalDateTime
+ setDataCancelamento(dataCancelamento: LocalDateTime): void
+ cancelar(): void
+ isAtivo(): boolean
UsuarioRepositorio
- instance: UsuarioRepositorio
- usuariosComum: Map<Long, Usuario>
- contadorId: Long
- UsuarioRepositorio()
+ getInstance(): UsuarioRepositorio
+ buscarUsuarioPorId(id: Long): Usuario
+ cadastrarUsuario(usuario: Usuario): Usuario
+ atualizarUsuario(usuarioId: Long, usuario: Usuario) : Usuario
+ buscarUsuarioPorEmail(email: String): Usuario
Usuario
- id: Long
- nome: String
- dataNascimento: LocalDate
- sexo: String
- email: String
- senha: String
- isAtivo: Boolean
+ Usuario()
+ Usuario(nome: String, dataNascimento: LocalDate, sexo: String, email: String)
+ getNome(): String
+ setNome(nome: String): void
+ getDataNascimento(): LocalDate
+ setDataNascimento(dataNascimento: LocalDate): void
+ getSexo(): String
+ setSexo(sexo: String): void
+ getEmail(): String
+ setEmail(email: String): void
+ getId(): Long
+ setId(id: Long): void
+ getSenha(): String
+ setSenha(senha: String): void
+ getIsAtivo(): Boolean
+ setIsAtivo(isAtivo: Boolean): void
+ equals(object: Object): boolean
+ hashCode(): int
+ toString(): String
Organizador
- id: Long
- nome: String
- dataNascimento: LocalDate
- sexo: String
- email: String
- senha: String
- empresa: Empresa
- isAtivo: Boolean
- hasEvento: Booelan
+ Organizador()
+ Organizador(nome: String, dataNascimento: LocalDate, sexo: String, email: String, senha: String, empresa: Empresa)
+ getNome(): String
+ setNome(nome: String): void
+ getDataNascimento(): LocalDate
+ setDataNascimento(dataNascimento: LocalDate): void
+ getSexo(): String
+ setSexo(sexo: String): void
+ getEmail(): String
+ setEmail(email: String): void
+ getSenha(): String
+ setSenha(senha: String): void
+ getEmpresa(): Empresa
+ setEmpresa(empresa: Empresa): void
+ getId(): Long
+ setId(id: Long): void
+ getIsAtivo(): Boolean
+ setIsAtivo(isAtivo: Boolean): void
+ getHasEvente(): Booean
+ equals(object: Object): boolean
+ hashCode(): int
+ toString(): String
Empresa
- cnpj: String
- razaoSocial: String
- nomeFantasia: String
- dataAbertura: Date
+ Empresa()
+ Empresa(cnpj: String, razaoSocial: String, nomeFantasia: String, dataAbertura: Date)
+ getCnpj(): String
+ setCnpj(cnpj: String): void
+ getRazaoSocial(): String
+ setRazaoSocial(razaoSocial: String): void
+ getNomeFantasia(): String
+ setNomeFantasia(nomeFantasia: String): void
+ getDataAbertura(): Date
+ setDataAbertura(dataAbertura: Date): void
OrganizadorRepositorio
- instance: OrganizadorRepositorio
- usuariosComum: Map<Long, Organizador>
- contadorId: Long
- OrganizadorRepositorio()
+ getInstance(): OrganizadorRepositorio
+ buscarOrganizadorPorId(id: Long): Organizador
+ cadastrarOrganizador(organizador: Organizador): Organizador
+ atualizarOrganizador(organizadorId: Long, organizador: Organizador) : Organizador
+ buscarOrganizadorPorEmail(email: String): Organizador
1
1
0..*
0..*
1
1

DTO

- Data Transfer Object
- Objeto usado para transferir dados entre camadas
- Reduzir acoplamento e proteger entidades
- Segurança, organização e controle de dados

```
public record AtualizarEventoRequestDTO(
        String nome,
        String paginaWeb,
        LocalDateTime dataInicio,
        LocalDateTime dataFim,
        TipoEvento tipoEvento,
        Evento eventoPrincipal,
        Double precoUnitarioIngresso,
        Double taxaCancelamento,
        Integer capacidadeMaxima
) {
}
```

Arquitetura de Endpoints

- Padrão RESTful
- Métodos HTTP
  - POST
  - PATCH
  - PUT
  - GET
- Estrutura e nomeação
  - Uso de substantivos
  - Pluralização
  - Hierarquia

```
PUT - ATUALIZAÇÃO DE EVENTO


/{organizadorId}/eventos/{eventoId}
```

POST

- Endpoint para cadastro de evento por um organizador.

```
@PostMapping(path = "/{organizadorId}/eventos")
    public ResponseEntity<Object> cadastrarEvento(@PathVariable(parameter = "organizadorId") long organizadorId, @RequestBody Evento evento) {
        try{
            Organizador organizador = this.organizadorRepositorio.buscarOrganizadorPorId(organizadorId);

            evento.setOrganizadorId(organizadorId);

            Evento novoEnvento = this.eventoRepositorio.cadastrarEvento(evento);
            return ResponseEntity.status(201, novoEnvento);
        }catch (NotFoundException e){
            return ResponseEntity.status(404, new ErroDTO(e.getMessage()));
        }catch (Exception e){
            return ResponseEntity.status(500, new ErroDTO(e.getMessage()));
        }
    }
```

PATCH

- Endpoint para desativar um usuário.

```java
@PatchMapping(path = "/{usuarioId}/desativar")
    public ResponseEntity<String> desativarUsuario(@PathVariable(parameter = "usuarioId") long usuarioId) {

        try {
            Usuario usuarioExiste = this.usuarioRepositorio.buscarUsuarioPorId(usuarioId);

            if (Boolean.FALSE.equals(usuarioExiste.getIsAtivo())) {
                return ResponseEntity.status(400, "Usuário já está inativo.");
            }

            usuarioExiste.setIsAtivo(false);

            return ResponseEntity.status(204, null);
        }catch (NotFoundException e) {
            return ResponseEntity.status(404, e.getMessage());
        } catch (Exception e) {
            return ResponseEntity.status(500, e.getMessage());
        }
    }
```

PUT

- Endpoint para atualizar um evento.

```
@PutMapping(path = "/{organizadorId}/eventos/{eventoId}")
    public ResponseEntity<Object> atualizarEvento(@PathVariable(parameter = "organizadorId") long organizadorId,
    @PathVariable(parameter = "eventoId") long eventoId, @RequestBody AtualizarEventoRequestDTO evento) {

        try{
            Evento eventoExiste = this.eventoRepositorio.buscarEventoPorId(eventoId);

            if (eventoExiste.getOrganizadorId() != organizadorId) {
                return ResponseEntity.status(401, new ErroDTO("Usuário sem permissão para alterar o evento."));
            }

            this.eventoRepositorio.atualizarEvento(eventoId, eventoExiste);

            return ResponseEntity.status(204, null);
        }catch (NotFoundException e){
            return ResponseEntity.status(404, new ErroDTO(e.getMessage()));
        }catch (Exception e){
            return ResponseEntity.status(500, new ErroDTO(e.getMessage()));
        }
    }
```

GET

- Endpoint para visualizar o perfil de um organizador.

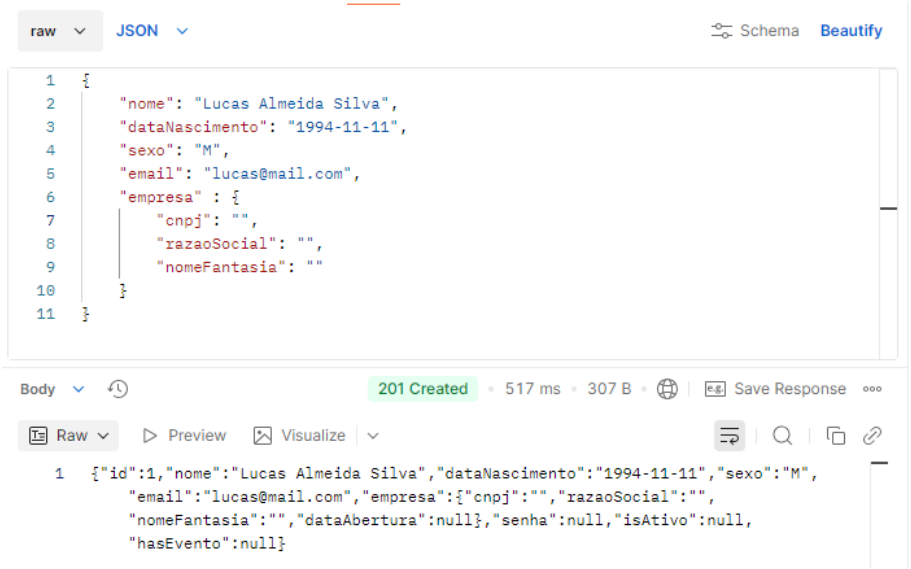
```
@GetMapping(path = "/{organizadorId}")
    public ResponseEntity<Object> visualizarPerfil(@PathVariable(parameter = "organizadorId") long organizadorId) {
        try {
            Organizador organizadorExiste = this.organizadorRepositorio.buscarOrganizadorPorId(organizadorId);

            OrganizadorPerfilResponseDTO response = new OrganizadorPerfilResponseDTO(organizadorExiste);

            return ResponseEntity.ok(response);
        } catch (NotFoundException e) {
            return ResponseEntity.status(404, new ErroDTO(e.getMessage()));
        } catch (Exception e) {
            return ResponseEntity.status(500, new ErroDTO(e.getMessage()));
        }
    }
```

Teste funcional

- Cadastrando um organizador



Teste funcional

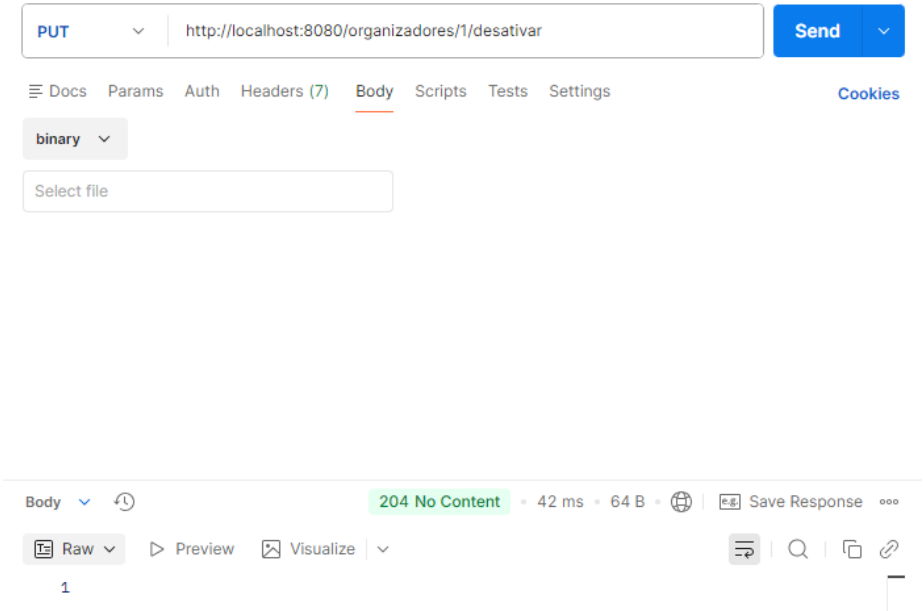
- Cadastrando um organizador com email já cadastrado

Teste funcional

- Alterando um organizador cadastrado

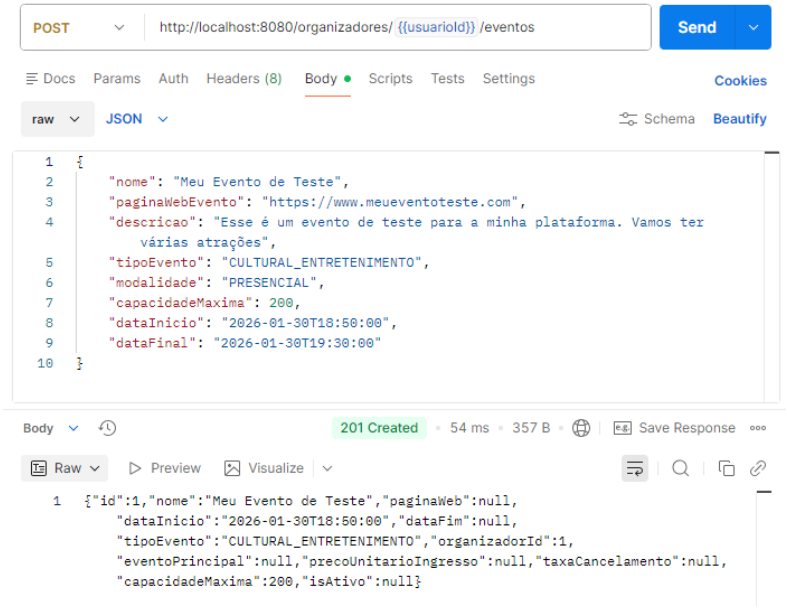
Teste funcional

- Desativando um organizador



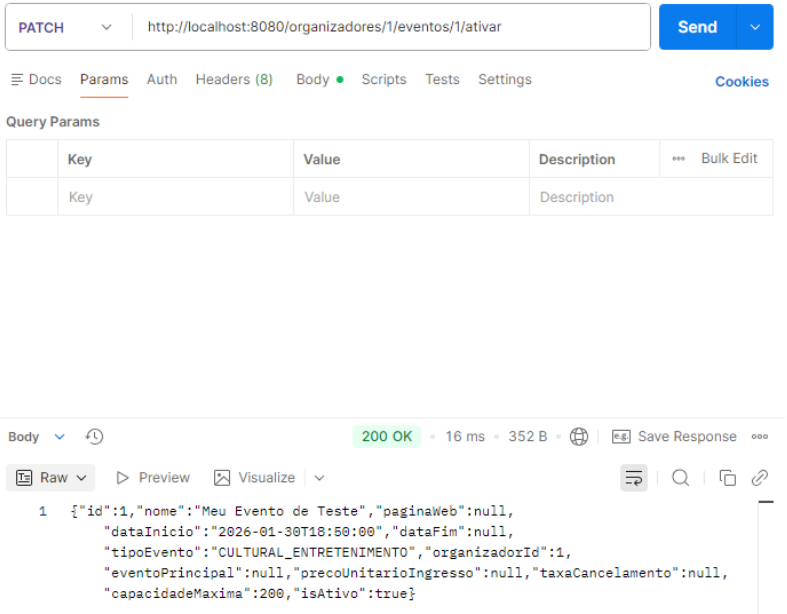
Teste funcional

- Cadastrando um evento



Teste funcional

- Ativando um evento



Referências

- SCHILDT, Herbert. Java para iniciantes. 6. ed. Porto Alegre: Bookman, 2015. E-book. Disponível em: https://integrada.minhabiblioteca.com.br/reader/books/9788582603376. Acesso em: 17 fev. 2026.

- NITHACK, Andrey. A mágica por trás da conversão de dados: o padrão DTO explicado. Medium, 6 jul. 2023. Disponível em: https://devnit.medium.com/a-m%C3%A1gica-por-tr%C3%A1s-da-convers%C3%A3o-de-dados-o-padr%C3%A3o-dto-explicado-25f68a718b5a. Acesso em: 20 fev. 2026.