Relatório – Parte 2 (Gestão de Eventos)

1. Introdução

Este projeto corresponde à **Parte 2** do trabalho de Desenvolvimento Web 1 e tem como objetivo o desenvolvimento de uma **API RESTful design-first**, documentada em **OpenAPI 3.0**, para a gestão de eventos.

O trabalho exige a utilização de **MySQL** como sistema de gestão de base de dados, **Node.js/Express** como servidor de aplicação, e a implementação de **CRUD completo** sobre quatro recursos principais: - **Users** - **Venues** - **Events** - **Tickets**

Além do CRUD, foi implementada pelo menos uma **relação 1:n** e adicionados **filtros avançados** no recurso `/events`. A solução deve ser entregue com **Docker Compose multi-serviço** (MySQL + API) e validada com uma **coleção Postman**.

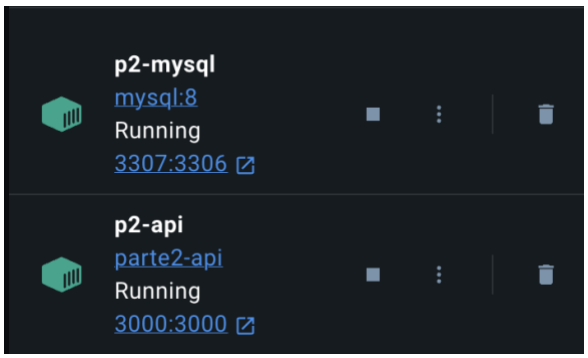
---

2. Arquitetura

A arquitetura segue um modelo **cliente-servidor** clássico, com duas camadas principais:

- **Camada de Dados**: MySQL, com schema e seeds automáticos.
- **Camada de Aplicação**: API em Node.js/Express, que expõe endpoints REST documentados em Swagger.

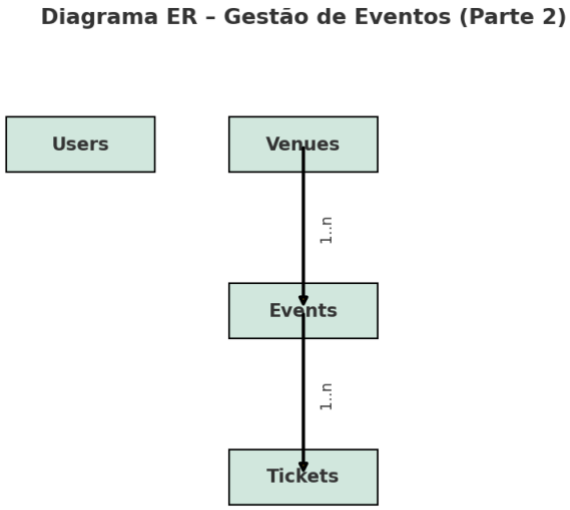
```
[Cliente / Postman] → [API REST Node.js/Express] → [MySQL]
```



3. Modelo de Dados

O modelo de dados foi concebido com quatro entidades principais e duas relações **1:n**:

- **User**: armazena utilizadores registados.
- **Venue**: locais de eventos.
- **Event**: eventos associados a um local (1 Venue → n Events).
- **Ticket**: bilhetes associados a um evento (1 Event → n Tickets).



4. Execução

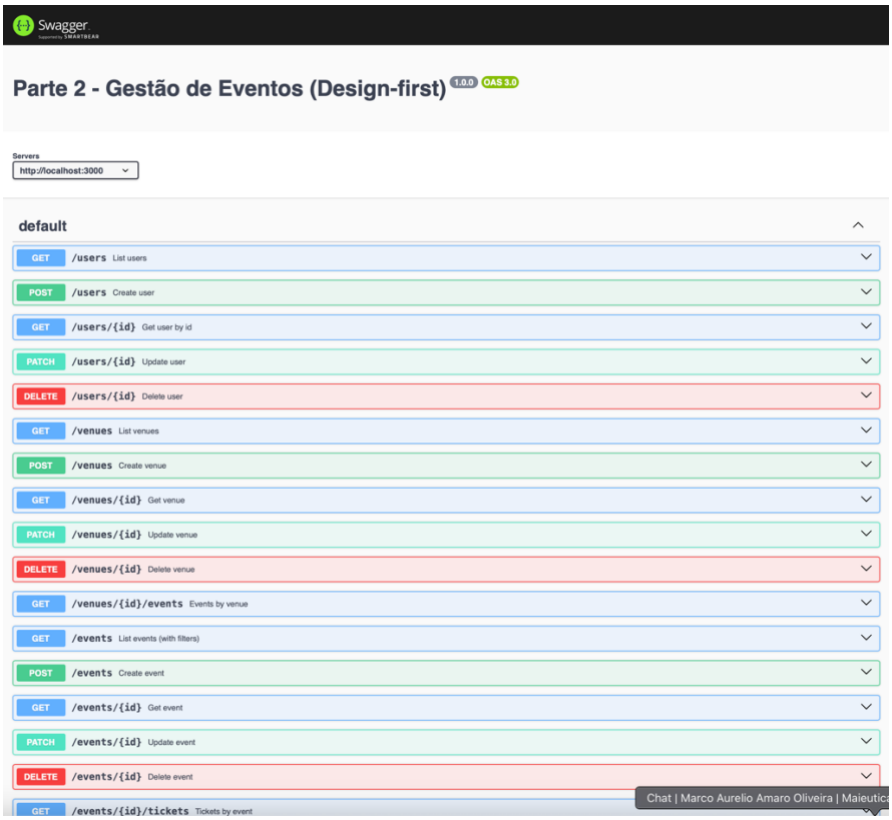
Passos de execução:

1. Construir e iniciar os serviços:

   ```
   docker compose up --build
   ```

2. A API fica disponível em:

   - http://localhost:3000
   - Swagger em http://localhost:3000/explorer

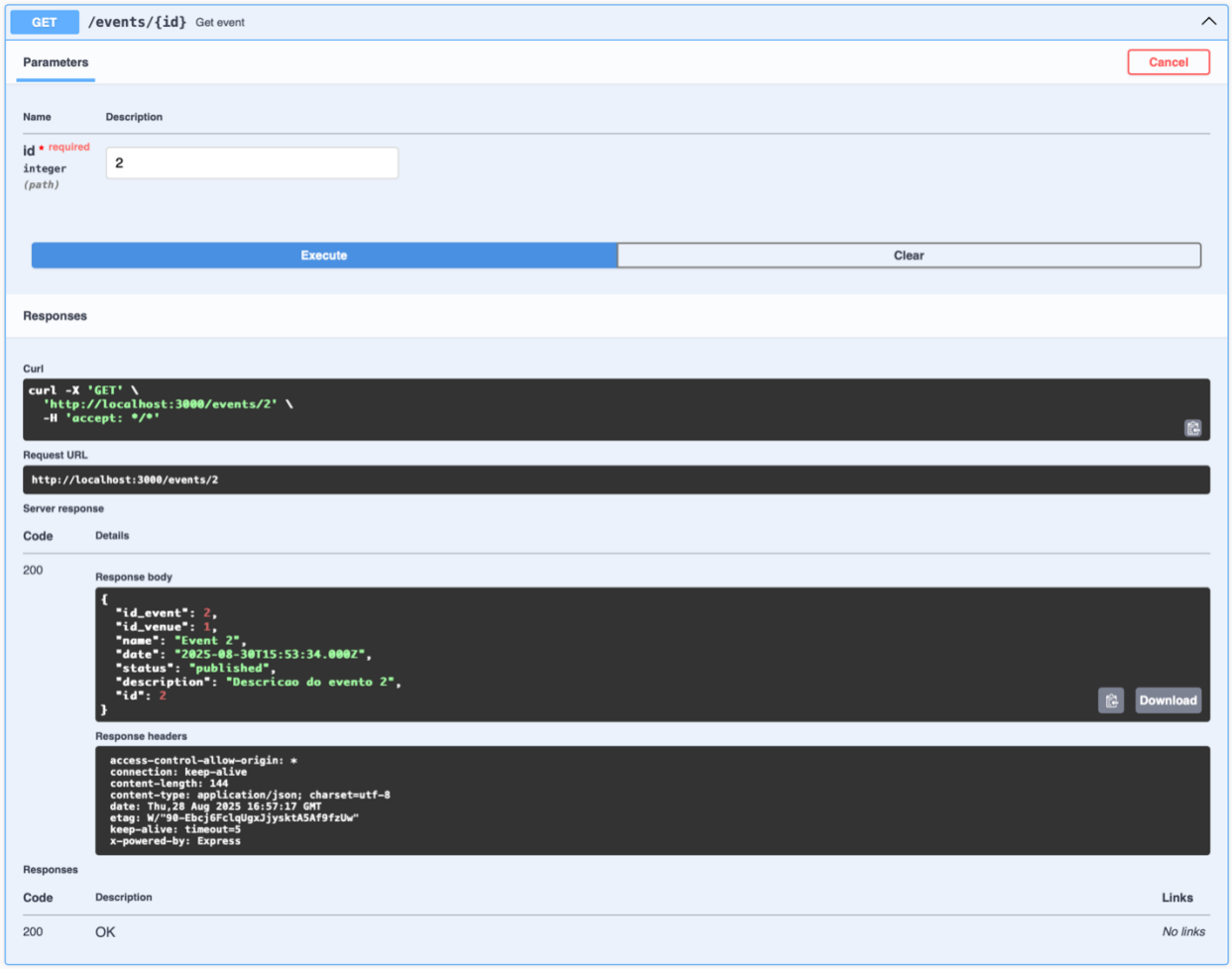


5. Funcionalidades e Endpoints

A API disponibiliza CRUD completo sobre quatro recursos:

- `/users` (create, read, update, delete)
- `/venues` (create, read, update, delete, `/venues/{id}/events`)
- `/events` (CRUD + filtros por `q`, `status`, `venueId`, `dateFrom`, `dateTo`)
- `/tickets` (CRUD + `/events/{id}/tickets`)

Formato de dados: **JSON**.



6. Testes (Postman)

Foi criada uma coleção Postman que cobre:

- **Sanidade** (GETs básicos)
- **CRUD Encadeado** (Venue → Event → Ticket → Patch → Delete)
- **Users CRUD** (ciclo completo num user)
- **Filtros & Relações**
- **Validações** (400 e 404)

Todos os testes passaram com sucesso.

7. Conclusão

A Parte 2 cumpre integralmente os requisitos do enunciado: - API design-first com documentação OpenAPI 3.0
- CRUD completo em 4 recursos
- Relações 1:n implementadas
- Filtros avançados em `/events`
- Docker Compose multi-serviço (MySQL + API)
- Seeds com ≥30 registos por tabela
- Coleção Postman validada com sucesso