

## 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**.

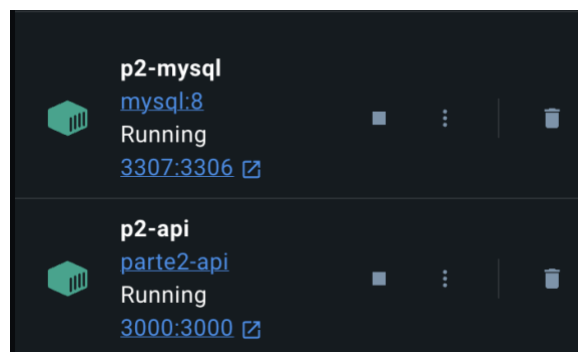
---

### 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]

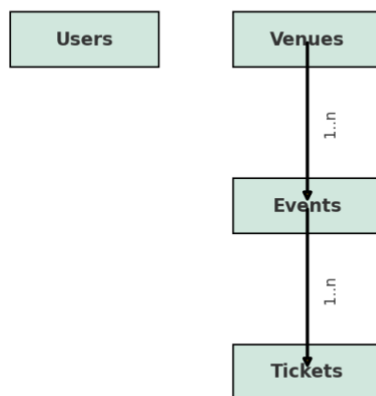


### 3. Modelo de Dados

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  $\rightarrow$  n Events).
- **Ticket**: bilhetes associados a um evento (1 Event  $\rightarrow$  n Tickets).

Diagrama ER - Gestão de Eventos (Parte 2)



## 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**.

The screenshot shows a REST client interface with the following sections:

- GET /events/{id} Get event**
- Parameters**: A table with columns 'Name' and 'Description'. It contains one parameter: 

Name	Description
id * required integer (path)	2
- Execute** and **Clear** buttons.
- Responses**:
  - Curl**: 

```
curl -X 'GET' \
'http://localhost:3000/events/2' \
-H 'accept: */*'
```
  - Request URL**: `http://localhost:3000/events/2`
  - Server response**:

Code	Details
200	<p><b>Response body</b></p> <pre>{   "id_event": 2,   "id_venue": 1,   "name": "Event 2",   "date": "2025-08-30T15:53:34.000Z",   "status": "published",   "description": "Descricao do evento 2",   "id": 2 }</pre> <p><b>Response headers</b></p> <pre>access-control-allow-origin: * connection: keep-alive content-length: 144 content-type: application/json; charset=utf-8 date: Thu, 28 Aug 2025 16:57:17 GMT etag: W/"90-Ebcj6FcIqUgxJjysktASAf9fzUw" keep-alive: timeout=5 x-powered-by: Express</pre>
- Responses** table:

Code	Description	Links
200	OK	No links

## 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.

Parte 2 - TESTS (Completa) - Run results				
Ran today at 17:00:36 · <a href="#">View all runs</a>				
Source	Environment	Iterations	Duration	All tests
Runner	Local-Parte2	1	1s 23ms	62
RUN SUMMARY				
1				
▶ GET	GET /users		2   0	
▶ GET	GET /venues		2   0	
▶ GET	GET /events		2   0	
▶ GET	GET /tickets		2   0	
▶ POST	POST /venues (create)		5   0	
▶ GET	GET /venues/:id		4   0	
▶ POST	POST /events (create for venue)		5   0	
▶ GET	GET /venues/:id/events		3   0	
▶ POST	POST /tickets (create for event)		5   0	
▶ GET	GET /events/:id/tickets		2   0	
▶ PATCH	PATCH /events/:id (cancelled)		2   0	
▶ DELETE	DELETE /tickets/:id		1   0	
▶ DELETE	DELETE /events/:id		1   0	
▶ DELETE	DELETE /venues/:id		1   0	
▶ POST	POST /users (create)		4   0	
▶ GET	GET /users/:id		4   0	
▶ PATCH	PATCH /users/:id		2   0	
▶ PUT	PUT /users/:id		2   0	
▶ DELETE	DELETE /users/:id		1   0	
▶ GET	GET /events?q=Event		2   0	
▶ GET	GET /events?status=published		2   0	
▶ GET	GET /venues/1/events		2   0	
▶ GET	GET /events/1/tickets		2   0	
▶ GET	GET /events?dateFrom&dateTo		2   0	
▶ GET	GET /tickets/999999 (deve dar 404)		1   0	
▶ POST	POST /users sem email (400)		1   0	

## 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  $\geq 30$  registos por tabela
- Coleção Postman validada com sucesso