

## Relatório – Parte 3 (Aplicação Completa com Frontend)

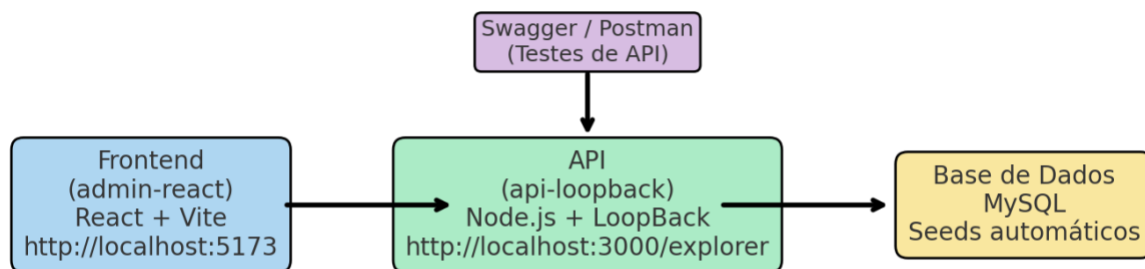
### 1. Introdução

Este projeto corresponde à Parte 3 do trabalho de Desenvolvimento Web 1 e tem como objetivo a integração completa da aplicação, incluindo o desenvolvimento de um frontend funcional que consome a API criada na Parte 2, bem como a entrega de um sistema coeso com todas as funcionalidades solicitadas.

### 2. Arquitetura

A arquitetura adotada nesta fase é composta por três camadas principais:

- Camada de Dados: Base de dados MySQL, com o schema definido e seeds automáticos.
- Camada de Aplicação: API RESTful em Node.js/Express, exposta através de endpoints documentados.
- Camada de Apresentação (Frontend): Aplicação web construída em React (ou tecnologia equivalente), que consome os serviços da API e disponibiliza uma interface amigável ao utilizador.



#### Legenda:

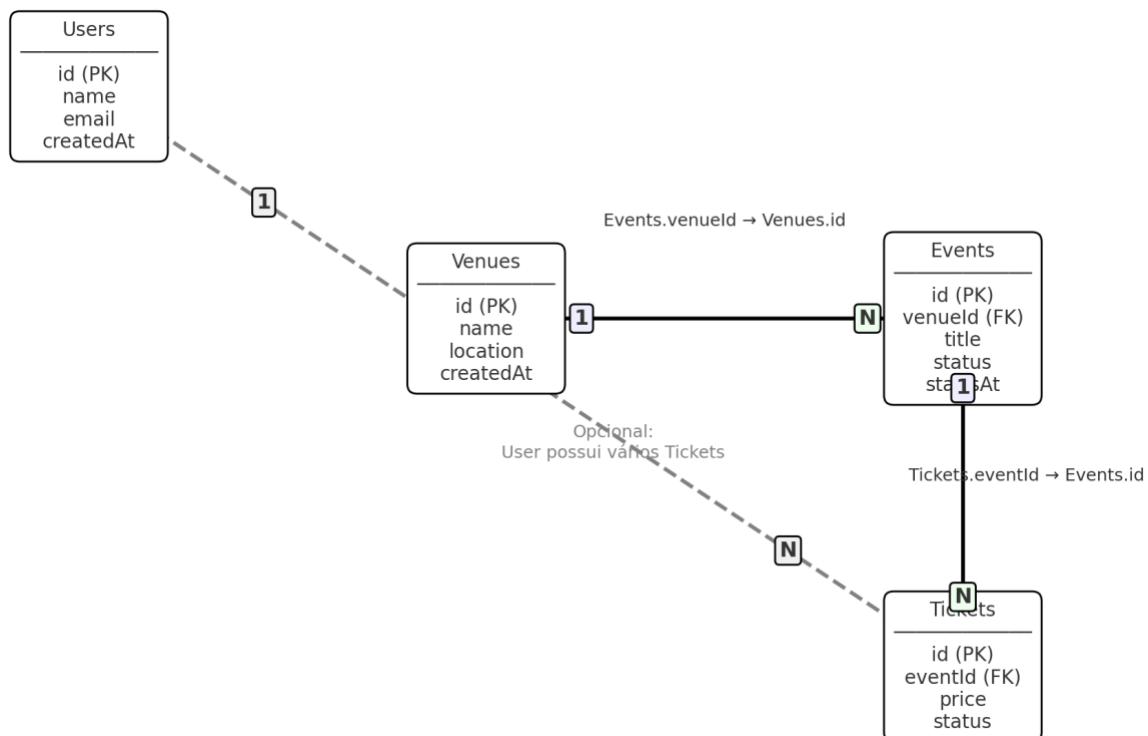
O diagrama representa a arquitetura completa da aplicação desenvolvida na Parte 3.

- O Frontend (admin-react) é uma aplicação em React + Vite que disponibiliza a interface gráfica ao utilizador, acessível via browser em <http://localhost:5173>.
- O API Backend (api-loopback), construído em Node.js/LoopBack, expõe os serviços REST documentados em Swagger ([/explorer](http://localhost:3000/explorer)) e implementa a lógica de negócio e operações CRUD.

- A Base de Dados MySQL armazena a informação persistente das entidades Users, Venues, Events e Tickets, com seeds automáticos.
- O Swagger/Postman funciona como cliente externo de teste e validação da API.

### 3. Modelo de Dados

O modelo de dados mantém as quatro entidades principais (Users, Venues, Events e Tickets) com as respetivas relações 1:n. Foram ainda feitas otimizações no desenho do modelo para garantir a consistência e integridade referencial.

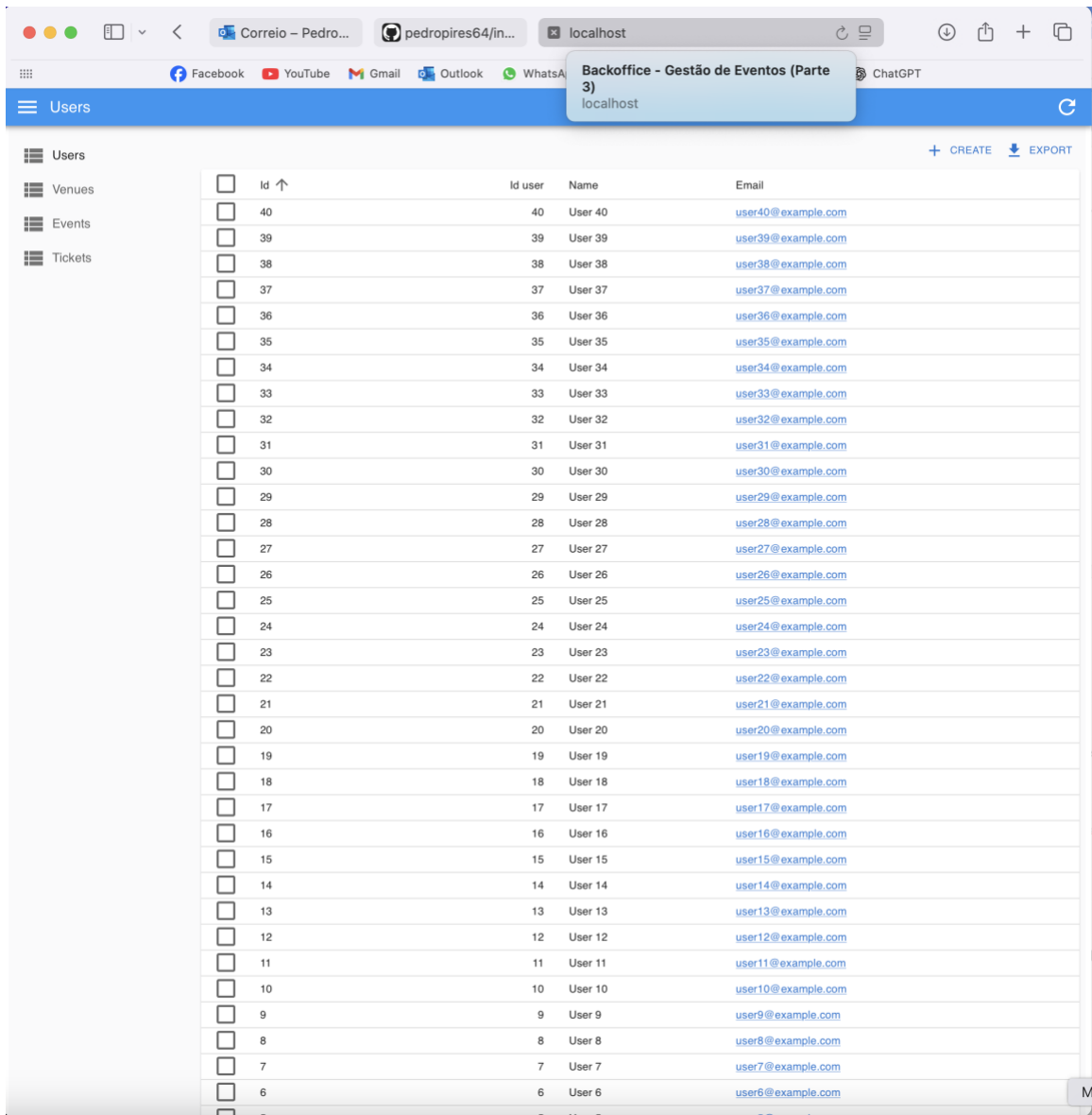


### 4. Execução

Passos para executar a aplicação completa:

1. Construir e iniciar os serviços:  
`docker compose up --build`
2. A API fica disponível em:  
`http://localhost:3000`
3. O Swagger pode ser acedido em:  
`http://localhost:3000/explorer`

4. O frontend pode ser acessado em:  
<http://localhost:5173> (ou a porta definida no projeto)



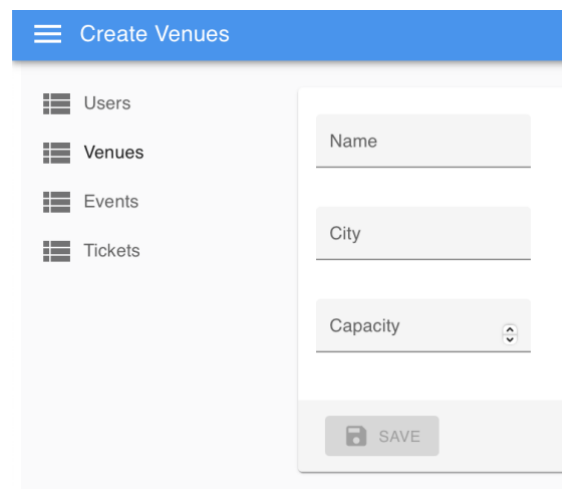
## 5. Funcionalidades e Endpoints

A aplicação disponibiliza um conjunto completo de funcionalidades, acessíveis tanto via API como via interface gráfica no frontend:

- Gestão de Users (CRUD)
- Gestão de Venues (CRUD + associação de eventos)
- Gestão de Events (CRUD + filtros avançados: q, status, venueId, dateFrom, dateTo)
- Gestão de Tickets (CRUD + associação a eventos)

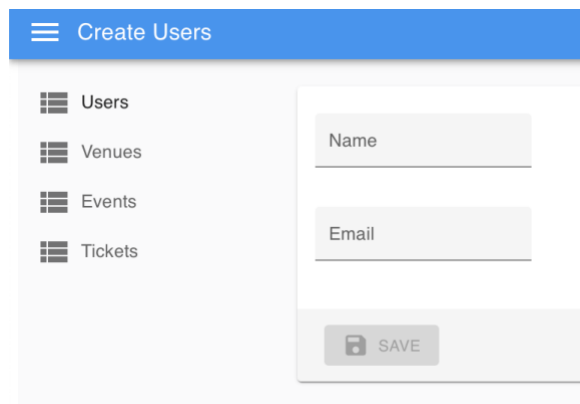
No frontend, o utilizador consegue:

- Registar e autenticar-se
- Visualizar e filtrar eventos
- Gerir venues e tickets



The screenshot shows the 'Create Venues' form. It has a blue header bar with a hamburger menu icon and the text 'Create Venues'. On the left, there is a sidebar with a list of menu items: 'Users', 'Venues', 'Events', and 'Tickets', each preceded by a small grid icon. The main area of the form contains three input fields: 'Name', 'City', and 'Capacity'. The 'Capacity' field has a small up/down arrow icon on its right side. At the bottom right of the form is a 'SAVE' button with a floppy disk icon.

Criar Locais



The screenshot shows the 'Create Users' form. It has a blue header bar with a hamburger menu icon and the text 'Create Users'. On the left, there is a sidebar with a list of menu items: 'Users', 'Venues', 'Events', and 'Tickets', each preceded by a small grid icon. The main area of the form contains two input fields: 'Name' and 'Email'. At the bottom right of the form is a 'SAVE' button with a floppy disk icon.

Criar Utilizadores

Create Events

Users

Venues

Events

Tickets

Id venue

Name

Date

04/09/2025, 12:30

Status

Description

SAVE

## Criar Eventos

Create Tickets

Users

Venues

Events

Tickets

Id event

Type

Price

Status

SAVE

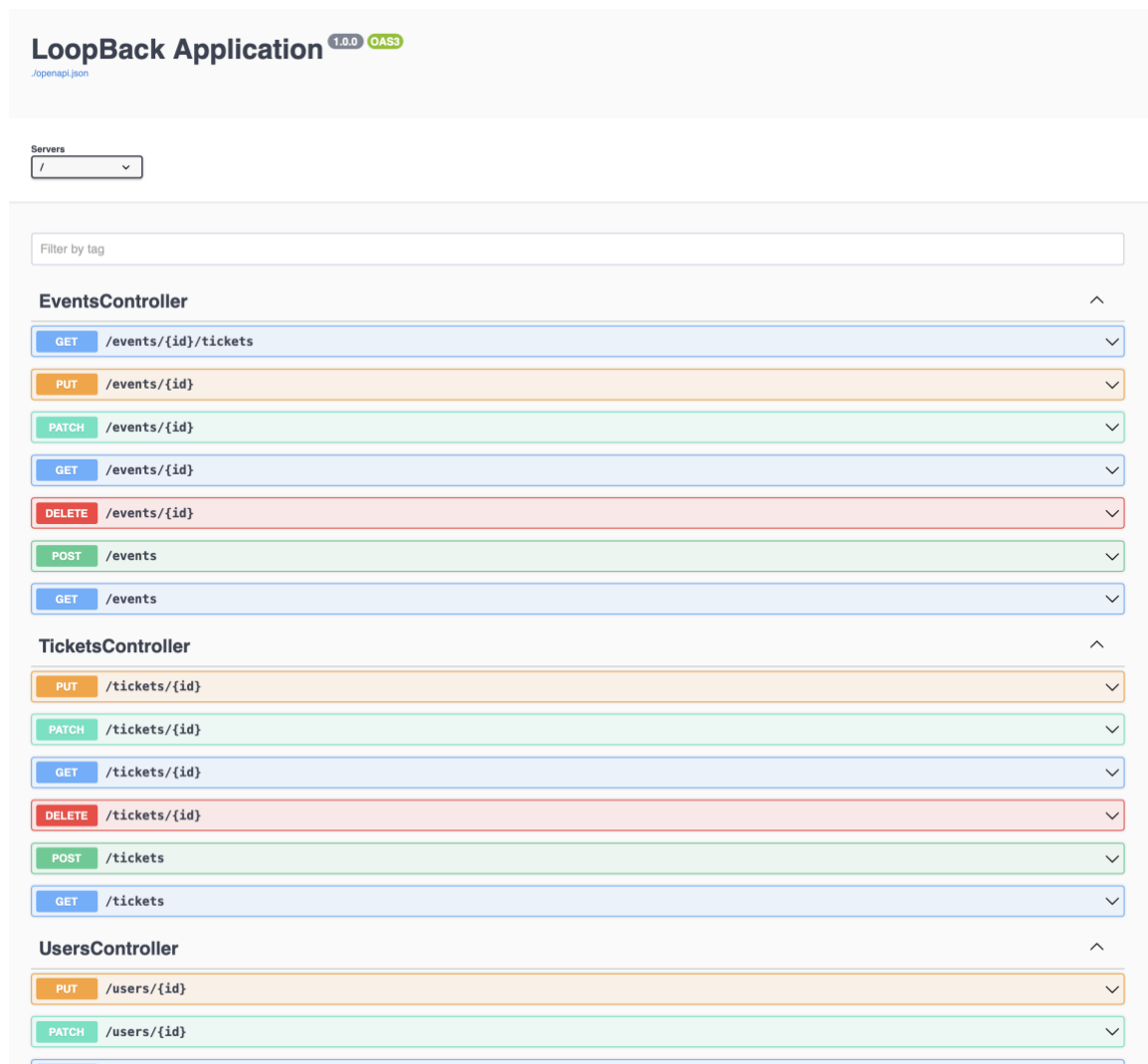
## Criar Bilhetes

## 6. Testes (Swagger e Postman)

Foram realizados testes tanto via Swagger UI como via Postman:

- Testes básicos de sanidade (GETs)
- CRUD encadeado (Venue → Event → Ticket)
- Validações de erros (400, 404)
- Filtros avançados

Todos os testes passaram com sucesso.



- O **Swagger UI** a correr (LoopBack Application com OpenAPI 3.0).

- Os controladores principais (**EventsController**, **TicketsController**, **UsersController**), cada um com os métodos **GET**, **POST**, **PUT**, **PATCH**, **DELETE** visíveis.

The screenshot shows a Postman interface for a POST request to `http://localhost:3001/venues`. The request is successful, returning a **200 OK** status with a response time of **14 ms** and a body size of **336 B**. The response body is a JSON object representing a venue.

**Query Params**

Key	Value	Description	...	Bulk Edit
Key	Value	Description		

**Body**

200 OK 14 ms 336 B Save as example

Pretty Raw Preview Visualize JSON

```

1 {
2   "id_venue": 35,
3   "name": "Venue QA 1756945935949",
4   "city": "Porto",
5   "capacity": 7777,
6   "id": 35
7 }
```

- Uma chamada **POST /venues** no Postman para `http://localhost:3001/venues`.
- O corpo da resposta com o novo registo criado (`id_venue: 35`, `name: "Venue QA..."`, `city: "Porto"`, `capacity: 7777`).
- O código de sucesso **200 OK** com tempo de execução e tamanho da resposta.

GET

http://localhost:3001/users

Send

ParamsAuthHeaders (7)BodyPre-req. Tests●SettingsCookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

Body200 OK16 ms2.91 KBSave as example

PrettyRawPreviewVisualizeJSON

```
1  [
2    {
3      "id_user": 40,
4      "name": "User 40",
5      "email": "user40@example.com",
6      "id": 40
7    },
8    {
9      "id_user": 39,
10     "name": "User 39",
11     "email": "user39@example.com",
12     "id": 39
13   },
14   {
15     "id_user": 38,
16     "name": "User 38",
17     "email": "user38@example.com",
18     "id": 38
19   },
20   {
21     "id_user": 37,
22     "name": "User 37",
23     "email": "user37@example.com",
24     "id": 37
25   },
26   {
27     "id_user": 36,
28     "name": "User 36",
29     "email": "user36@example.com",
30     "id": 36
31   },
32   {
33     "id_user": 35,
34     "name": "User 35",
35     "email": "user35@example.com",
36     "id": 35
37   },
38   {
39     "id_user": 34,
40     "name": "User 34",
41     "email": "user34@example.com",
42     "id": 34
43   },
44   {
45     "id_user": 33,
46     "name": "User 33",
47     "email": "user33@example.com",
48     "id": 33
49   }
50 ]
```



- Uma chamada **GET /users** feita no **Postman** para `http://localhost:3001/users`.
- A resposta JSON com a lista de utilizadores (vários registos, id, name, email).
- O código **200 OK**, tempo de resposta e tamanho da resposta.

## 7. Conclusão

A Parte 3 cumpre integralmente os requisitos do enunciado:

- Integração do frontend com a API desenvolvida na Parte 2
- CRUD completo sobre os recursos principais
- Relações 1:n corretamente implementadas
- Funcionalidades avançadas de pesquisa e filtros
- Testes validados com sucesso via Swagger e Postman
- Deploy local da aplicação com Docker Compose