Event Management System

Desenvolvimento WEB I

Event Managment System

A nossa API (Application Programming Interface) foi desenvolvida para auxiliar empresas de gestão de eventos a se organizarem de forma eficiente.

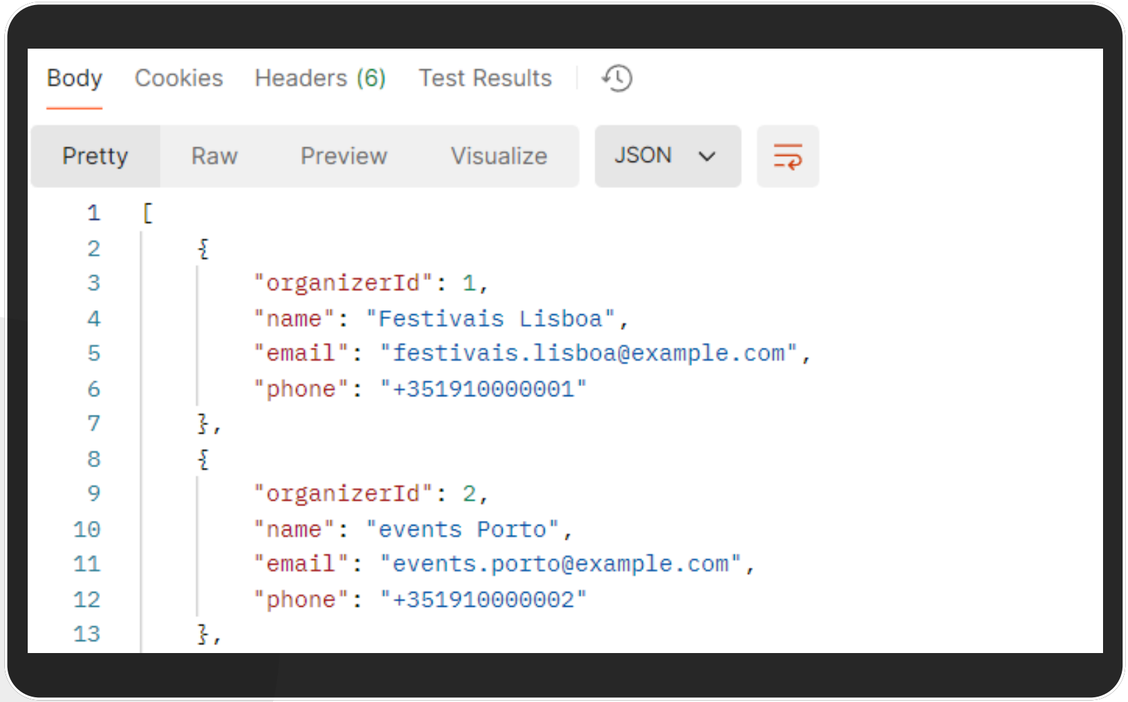
Esta oferece acesso a quatro principais recursos: Organizadores, Eventos, Bilhetes e Participantes. Para cada um desses recursos, estão disponíveis quatro métodos: **<GET>**, **<POST>**, **<PUT>** **e** **<DELETE>**, permitindo a realização de todas as funcionalidades necessárias, como criar um evento, atualizar suas informações, gerir tickets e participantes, entre outras operações.

Com esta API, todas essas ações podem ser realizadas de forma centralizada e simplificada, otimizando o gerenciamento com um único sistema de pedidos.

Organizers

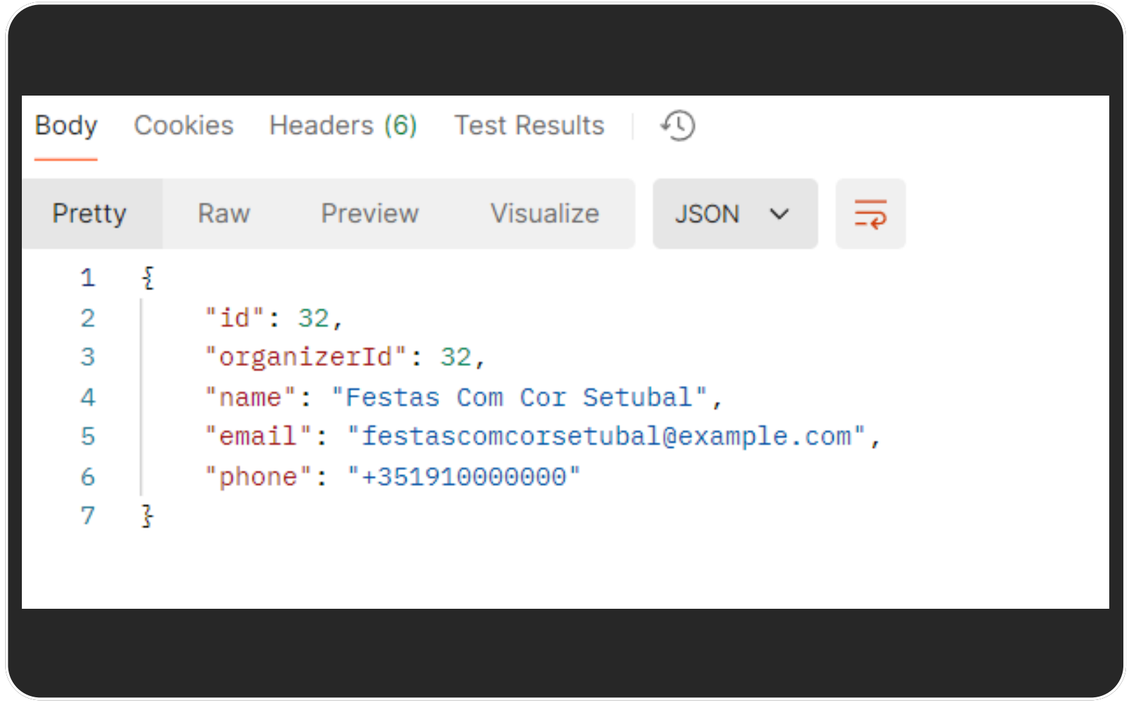
ORGANIZERS (GET)

Obtém a Lista de todos os Organizadores



CREATE ORGANIZER (POST)

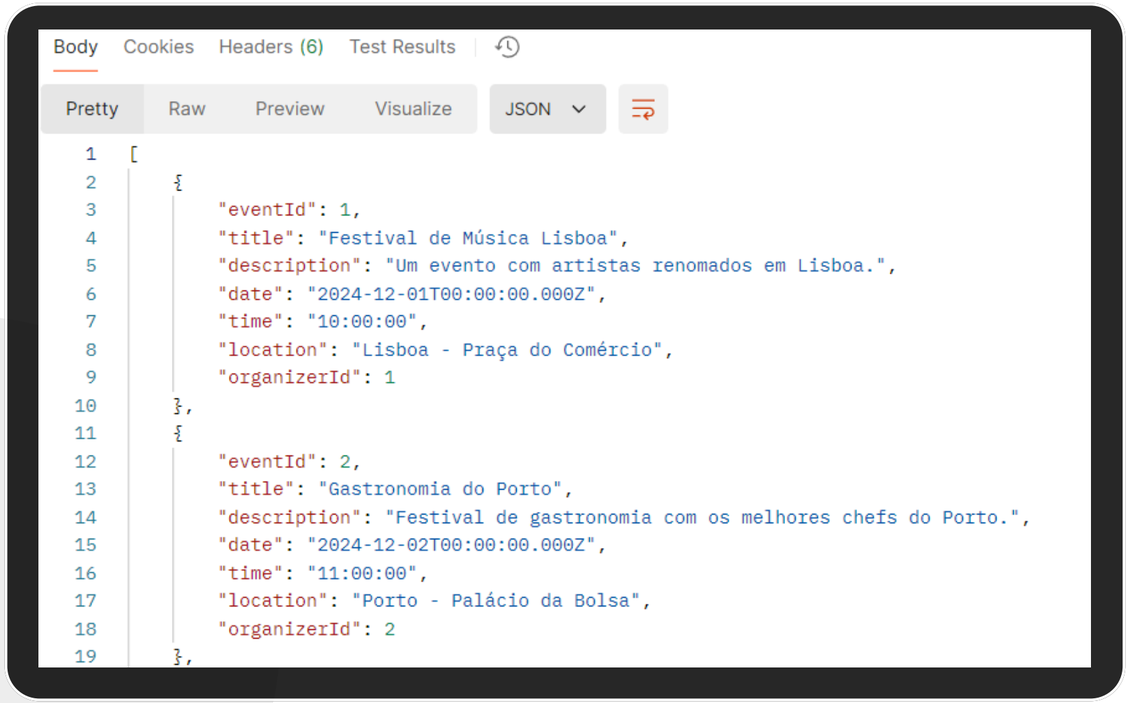
Cria um novo Organizador



Events

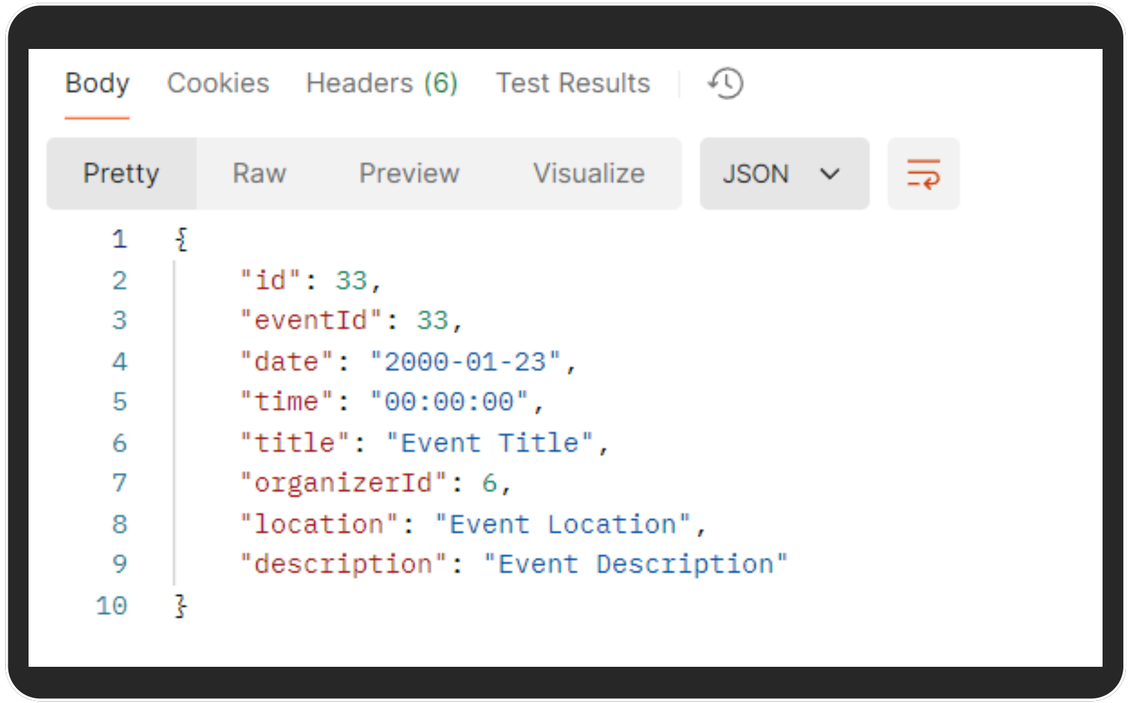
EVENTS (GET)

Obtém a Lista de todos os Eventos



CREATE EVENT (POST)

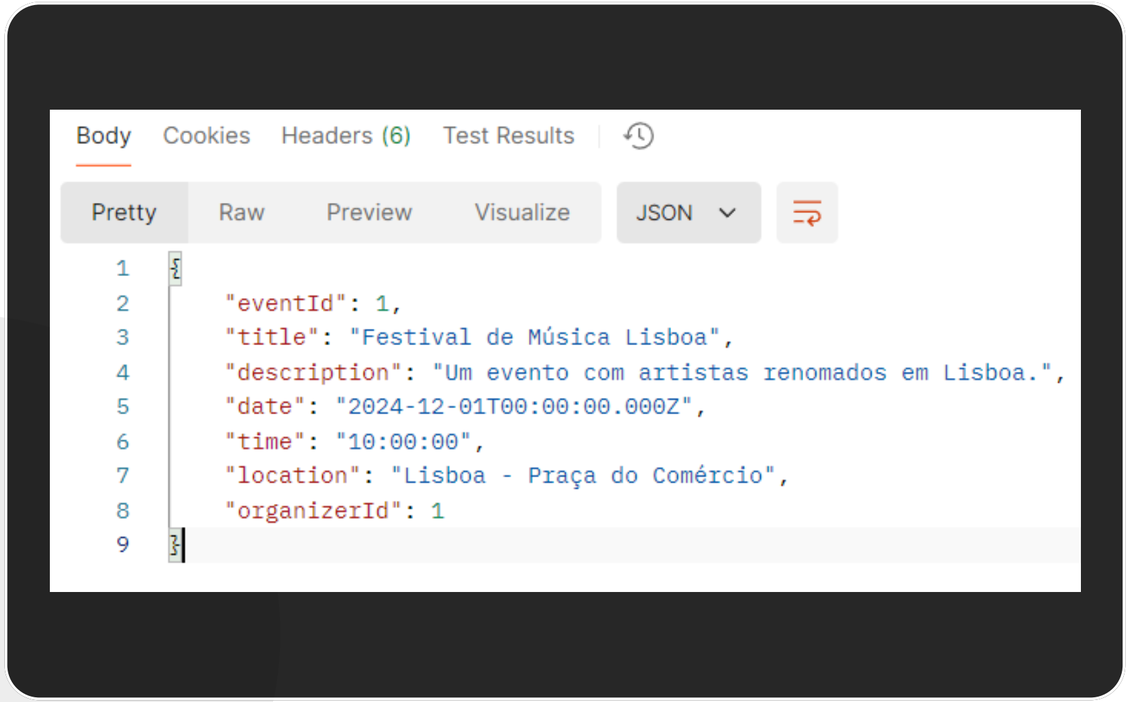
Cria um novo Evento



Events

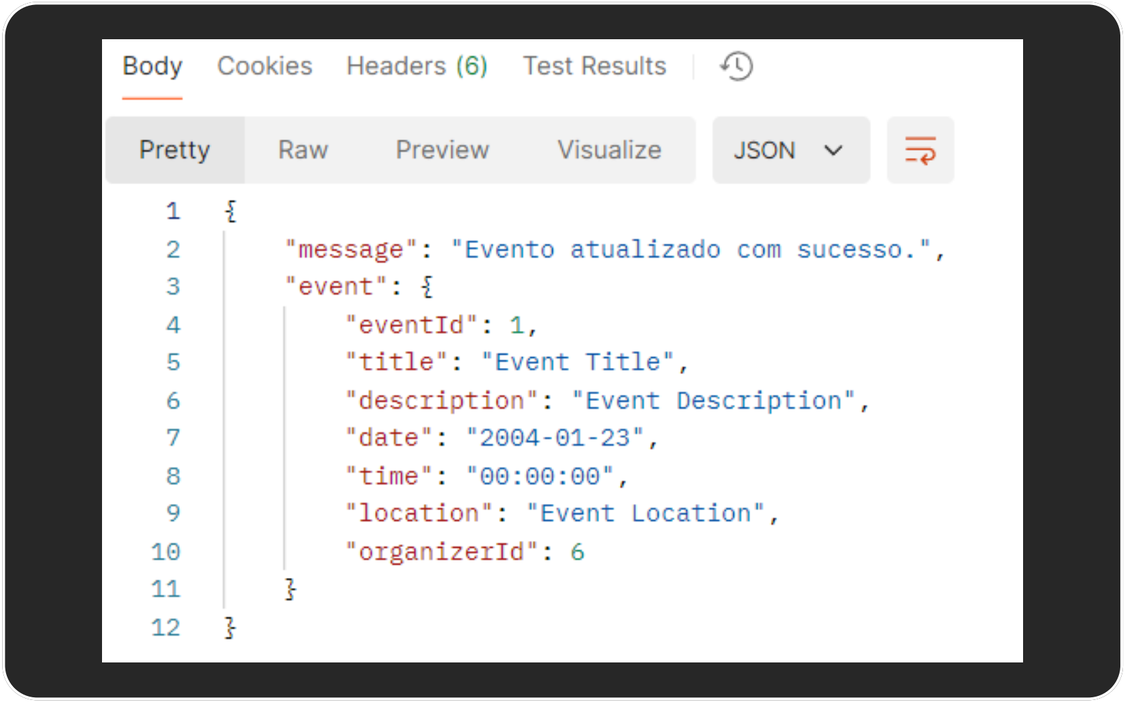
SPECIFIC EVENTS (GET)

Obtém um Evento Específico (“eventId”)



UPDATE SPECIFIC EVENT (PUT)

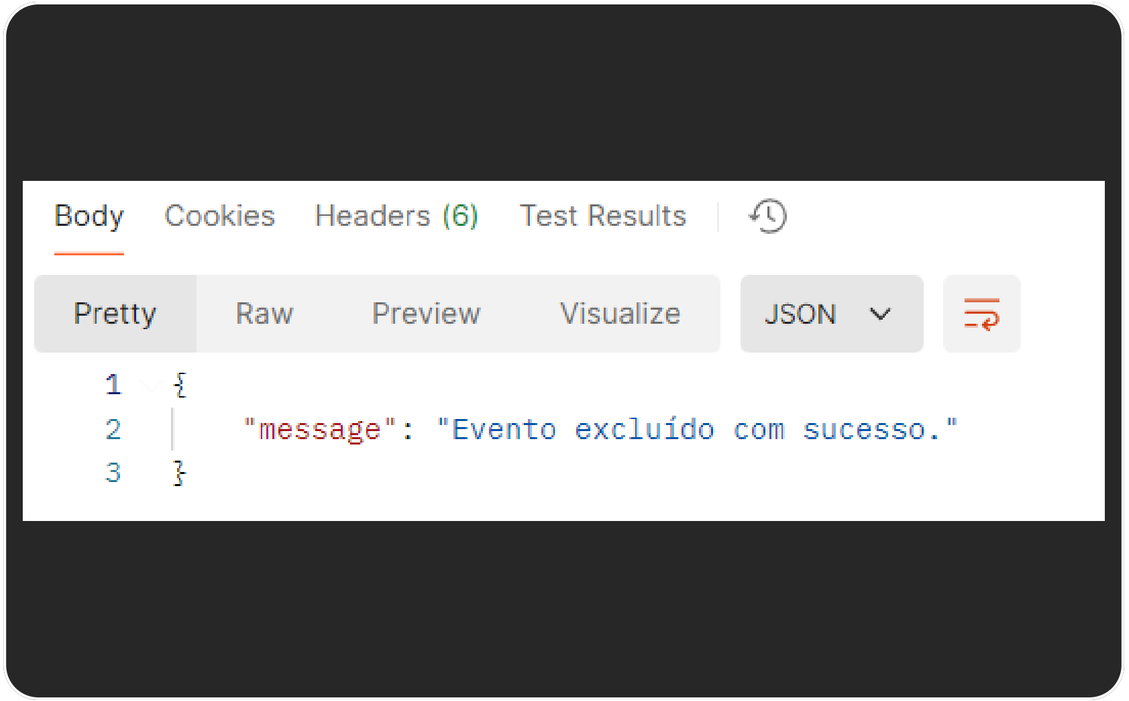
Atualiza informação de um Evento Específico (“eventId”)



Events

DELETE SPECIFIC EVENT (DELETE)

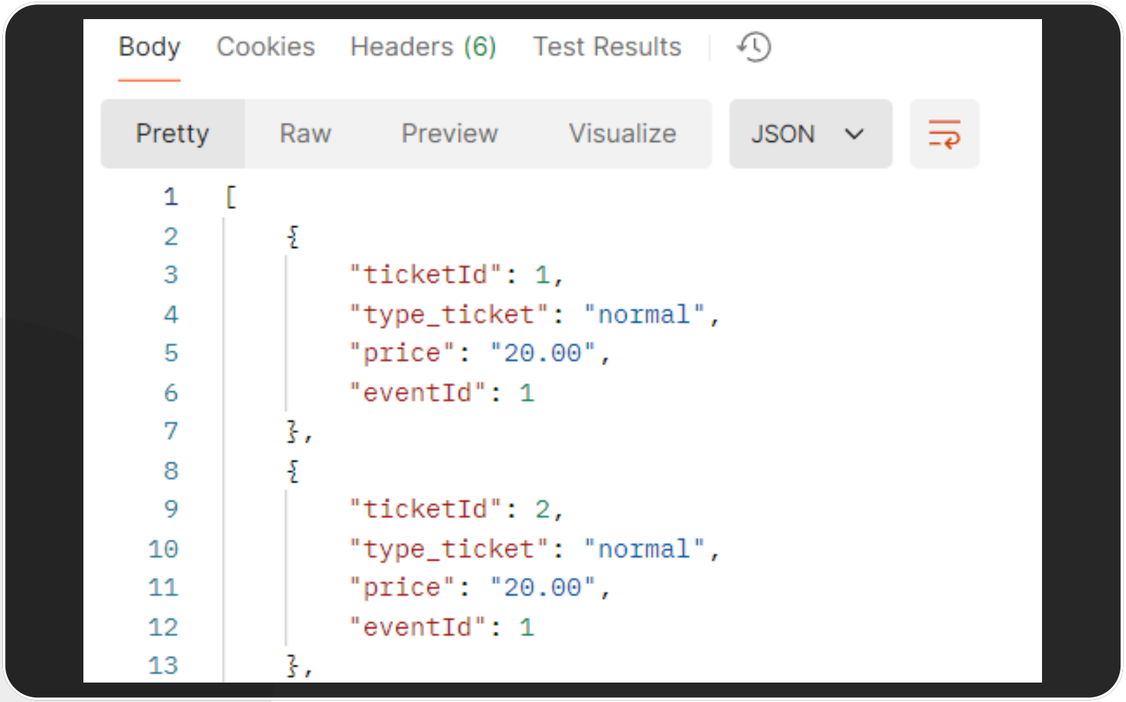
Apaga um Evento específico (“eventId”)



Tickets

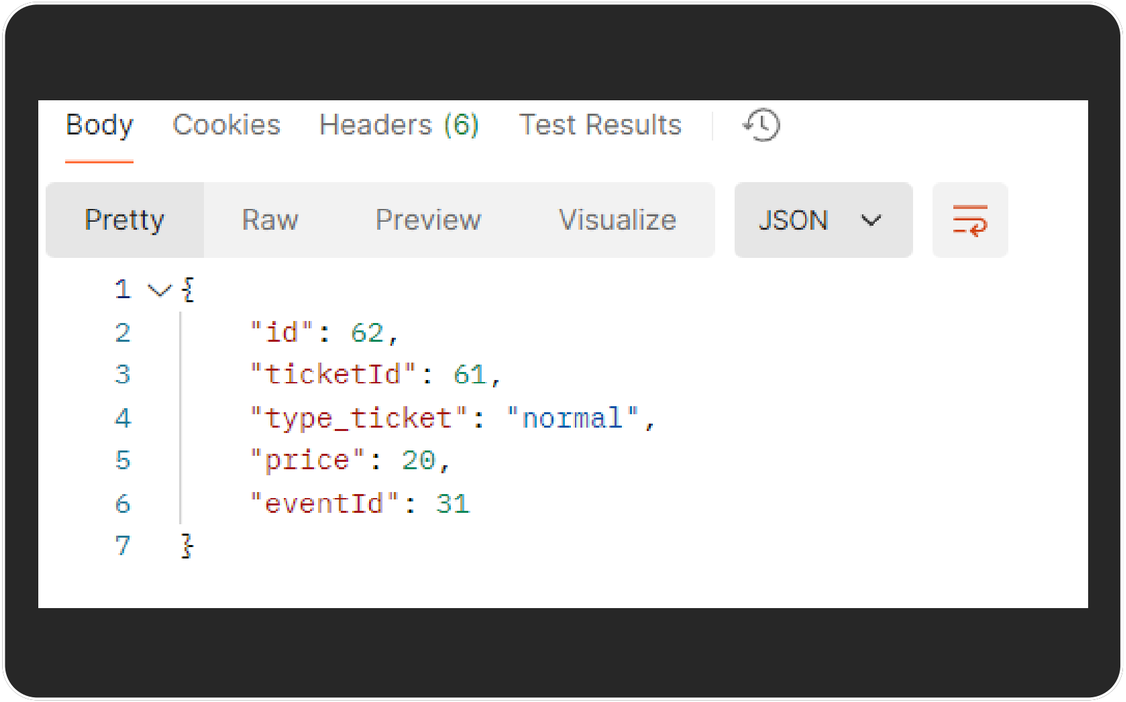
TICKETS (GET)

Obtém a Lista de todos os bilhetes



CREATE TICKET (POST)

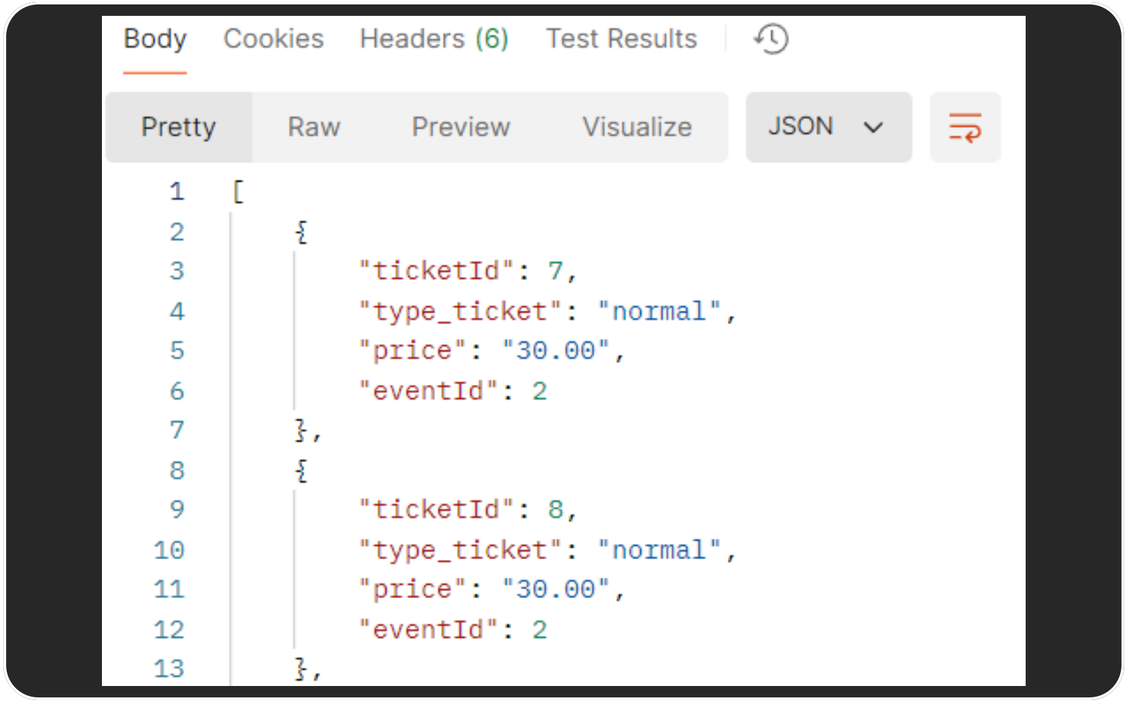
Cria um novo Bilhete



Tickets

SPECIFIC TICKET (GET)

Obtém Bilhete específico (ticketId)



Participants

PARTICIPANTS (GET)

Obtém a Lista de todos os Participantes

Body Cookies Headers (6) Test Results

Pretty Raw Preview Visualize JSON


```
[
    {
        "participantId": 1,
        "name": "JoÃ£o Silva",
        "email": "joao.silva@example.com",
        "status": "confirmado",
        "ticketId": 1
    },
    {
        "participantId": 2,
        "name": "Maria Oliveira",
        "email": "maria.oliveira@example.com",
        "status": "pendente",
        "ticketId": 2
    },
```


CREATE PARTICIPANT (POST)

Cria um novo Participante

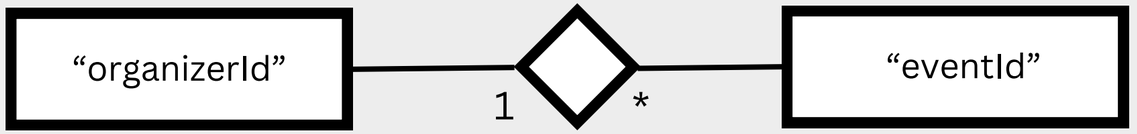
Body Cookies Headers (6) Test Results

Pretty Raw Preview Visualize JSON

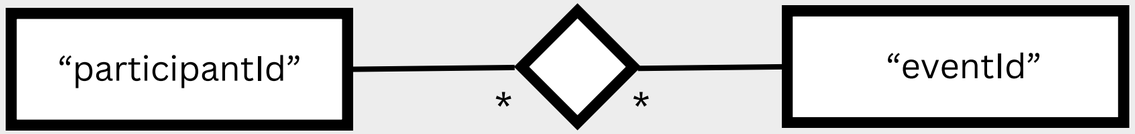

```
{
  "participantId": 31 ,
  "name": "John Doe",
  "email": "johndoe@example.com",
  "status": "confirmado",
  "ticketId": 62
}
```

Cardinality Realationship

Um organizador pode gerenciar vários eventos.

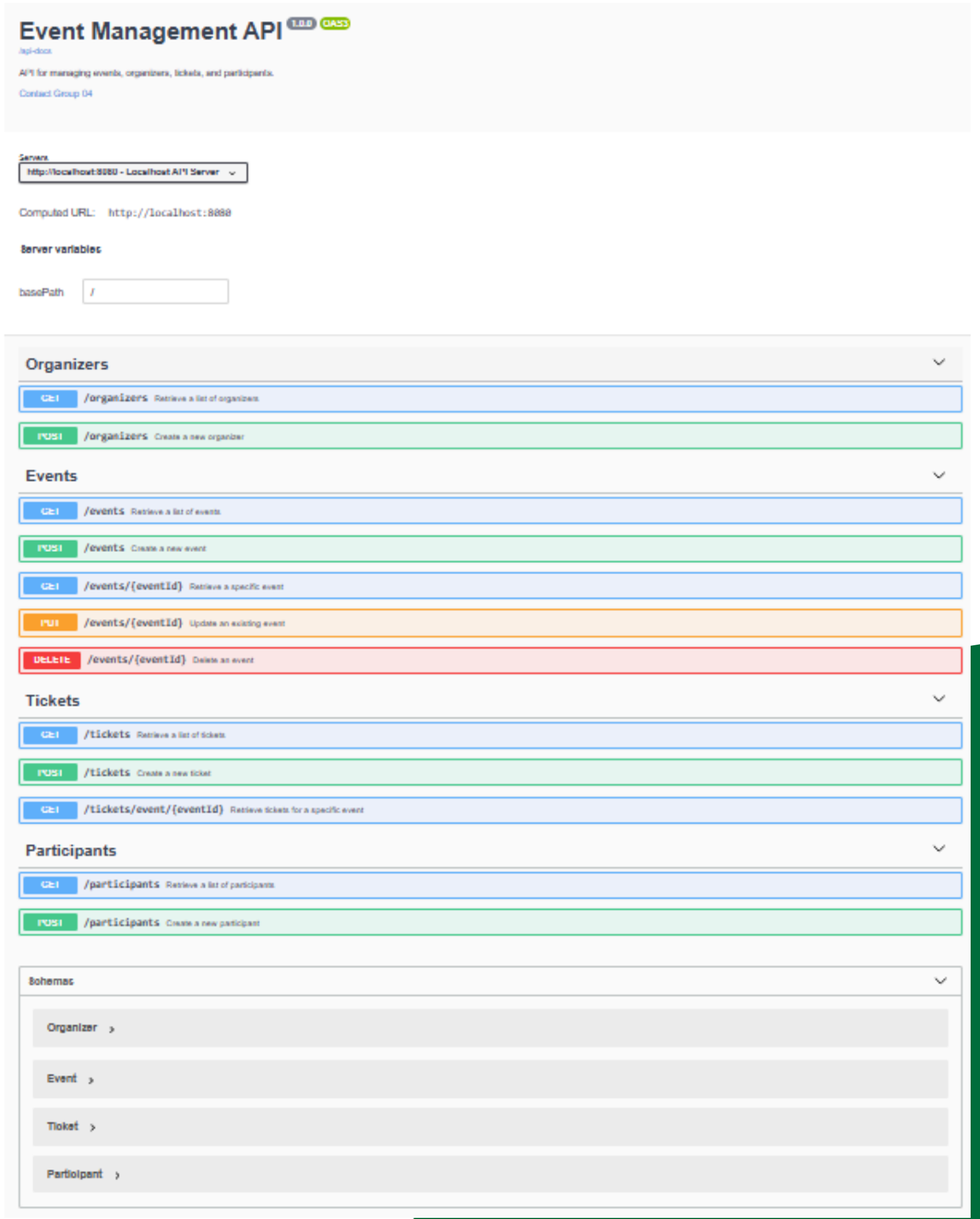


Participantes podem se inscrever em vários eventos, e eventos podem ter vários participantes.



Swagger e Node.js

Utilizamos o Swagger Editor para desenvolver e visualizar nossa API. Após validar o resultado, aproveitamos a função "generate server" disponível na ferramenta para gerar o código inicial do servidor em Node.js.

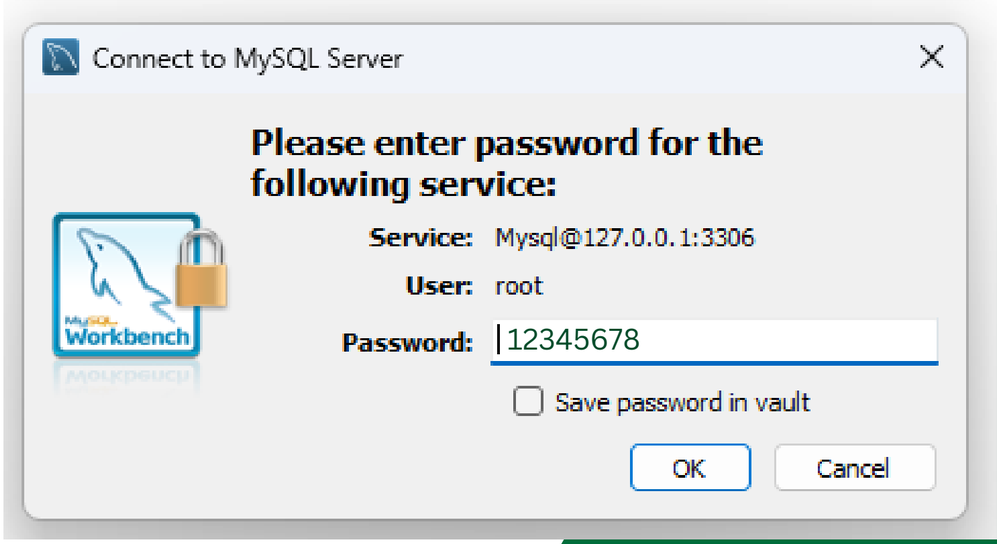


Connection to DataBase

db.js

```
'use strict';

var mysql = require('mysql2');

var connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: '12345678',
  database: 'sistema_gestao_events'
});

connection.connect(function (err) {
  if (err) {
    console.error('Erro ao conectar ao banco de dados: ' + err.stack);
    return;
  }
  console.log('Conectado ao banco de dados como ID ' + connection.threadId);
});

module.exports = connection;
```

MySQL - Workbench



Data Base (MYSQL)

STRUCTURE

1. Organizadores

- Representa os organizadores responsáveis pelos eventos.
- **Atributos**:
  - `organizerId` : Identificador único do organizador.
  - `name` : Nome do organizador.
  - `email` : Endereço de e-mail do organizador.
  - `phone` : Telefone para contato.

---

2. Eventos

- Representa os eventos disponíveis no sistema (presenciais ou online).
- **Atributos**:
  - `eventId` : Identificador único do evento.
  - `title` : Nome do evento.
  - `description` : Descrição do evento.
  - `date` : Data do evento.
  - `time` : Hora do evento.
  - `location` : Local onde o evento será realizado (ou link, se online).
  - `organizerId` : Referência ao organizador responsável.

---

3. Ingressos

- Representa os ingressos disponíveis para um evento.
- **Atributos**:
  - `ticketId` : Identificador único do ingresso.
  - `type_ticket` : Tipo de ingresso (normal, VIP, etc.).
  - `price` : Preço do ingresso
  - `eventId` : Referência ao evento associado.

---

4. Participantes

- Representa os usuários que participam dos eventos.
- **Atributos**:
  - `id` : Identificador único do participante.
  - `name` : Nome do participante.
  - `email` : Endereço de e-mail do participante.
  - `status` : Status de presença no evento (confirmado/pendente).
  - `tichetId` : Referência ao ingresso em que o participante está inscrito.

Data Base (MYSQL)

```sql
-- BASE DE DADOS

CREATE DATABASE sistema_gestao_events;
USE sistema_gestao_events;

CREATE TABLE organizers (
    organizerId INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    phone VARCHAR(20) NOT NULL
);

CREATE TABLE events (
    eventId INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    description TEXT NOT NULL,
    date DATE NOT NULL,
    time TIME NOT NULL,
    location VARCHAR(255) NOT NULL,
    organizerId INT NOT NULL,
    FOREIGN KEY (organizerId) REFERENCES organizers(organizerId)
);

CREATE TABLE tickets (
    ticketId INT AUTO_INCREMENT PRIMARY KEY,
    type_ticket ENUM('normal', 'VIP', 'premium'),
    price DECIMAL(10, 2) NOT NULL,
    eventId INT NOT NULL,
    FOREIGN KEY (eventId) REFERENCES events(eventId)
);

CREATE TABLE participants (
    participantId INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    status ENUM('confirmado', 'pendente') NOT NULL,
    ticketId INT NOT NULL,
    FOREIGN KEY (ticketId) REFERENCES tickets(ticketId)
);
```

```sql
-- Inserindo organizers
INSERT INTO organizers (organizerId, name, email, phone) VALUES
(1, 'Festivais Lisboa', 'festivais.lisboa@example.com', '+351910000001'),
(2, 'events Porto', 'events.porto@example.com', '+351910000002'),
```

•••

```sql
-- Inserindo events
INSERT INTO events (eventId, title, description, date, time, location, organizerId) VALUES
(1, 'Festival de Música Lisboa', 'Um evento com artistas renomados em Lisboa.', '2024-12-01', '10:00:00', 'Lisboa - Praça do Comércio', 1),
(2, 'Gastronomia do Porto', 'Festival de gastronomia com os melhores chefs do Porto.', '2024-12-02', '11:00:00', 'Porto - Palácio da Bolsa', 2),
```

•••

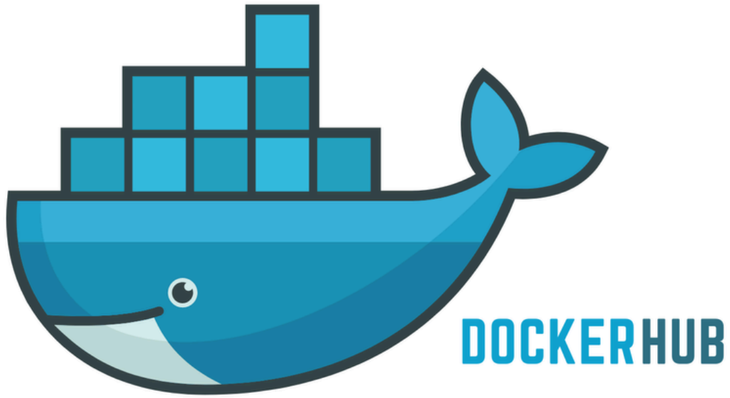
```sql
-- Inserindo tickets
INSERT INTO tickets (ticketId, type_ticket, price, eventId) VALUES
-- Evento 1
(1, 'normal', 20.00, 1),
(2, 'normal', 20.00, 1),
(3, 'normal', 20.00, 1),
(4, 'VIP', 50.00, 1),
(5, 'VIP', 50.00, 1),
(6, 'premium', 75.00, 1),
-- Evento 2
(7, 'normal', 30.00, 2),
(8, 'normal', 30.00, 2),
(9, 'normal', 30.00, 2),
(10, 'VIP', 80.00, 2),
(11, 'VIP', 80.00, 2),
(12, 'premium', 100.00, 2),
```

•••

```sql
-- Inserindo participants
INSERT INTO participants (participantId, name, email, status, ticketId) VALUES
(1, 'João Silva', 'joao.silva@example.com', 'confirmado', 1),
(2, 'Maria Oliveira', 'maria.oliveira@example.com', 'pendente', 2),
```

•••

DOCKER HUB USERS



PEDRO

A045464

CAROL

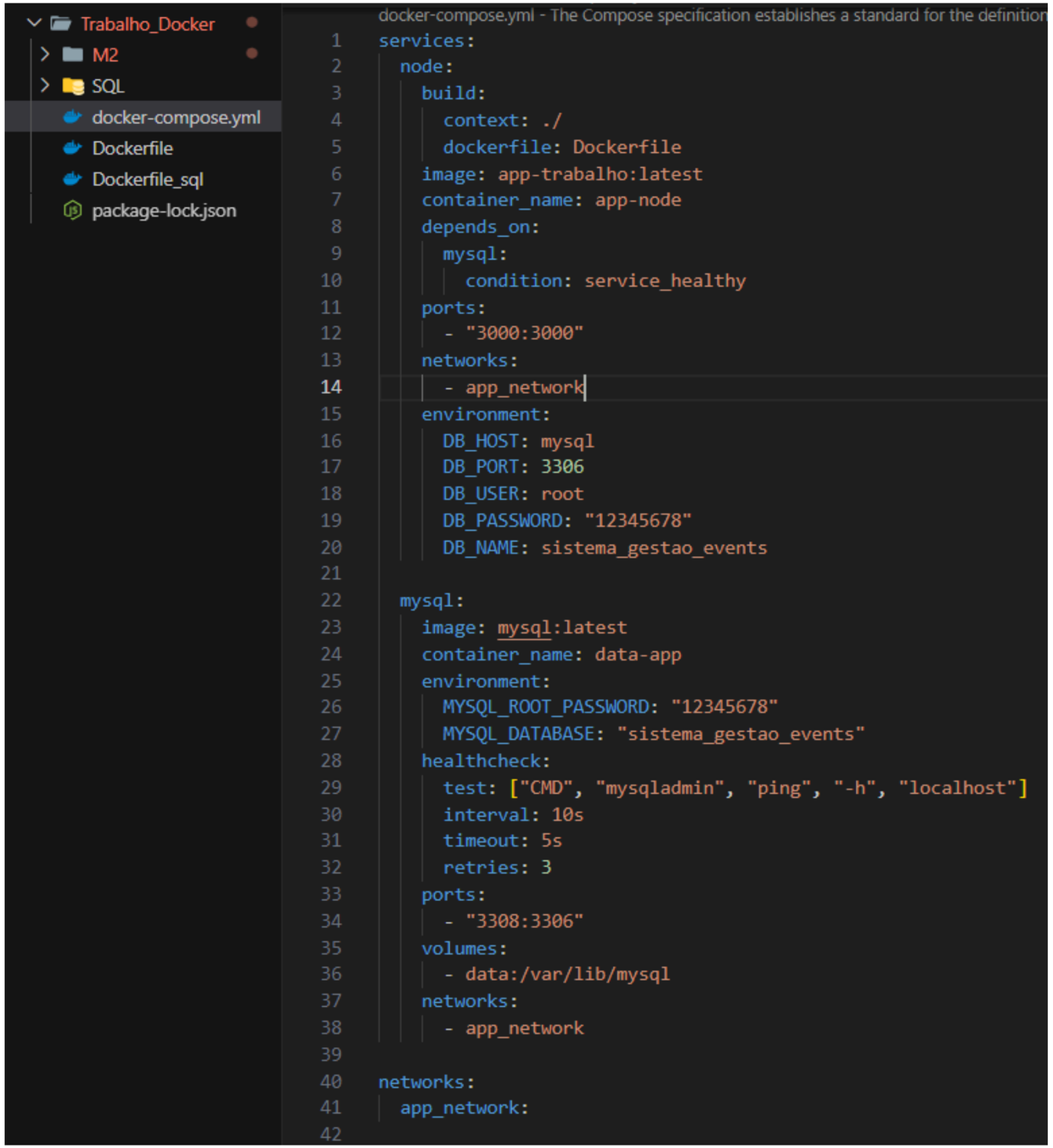
A044897

BIA

A044416

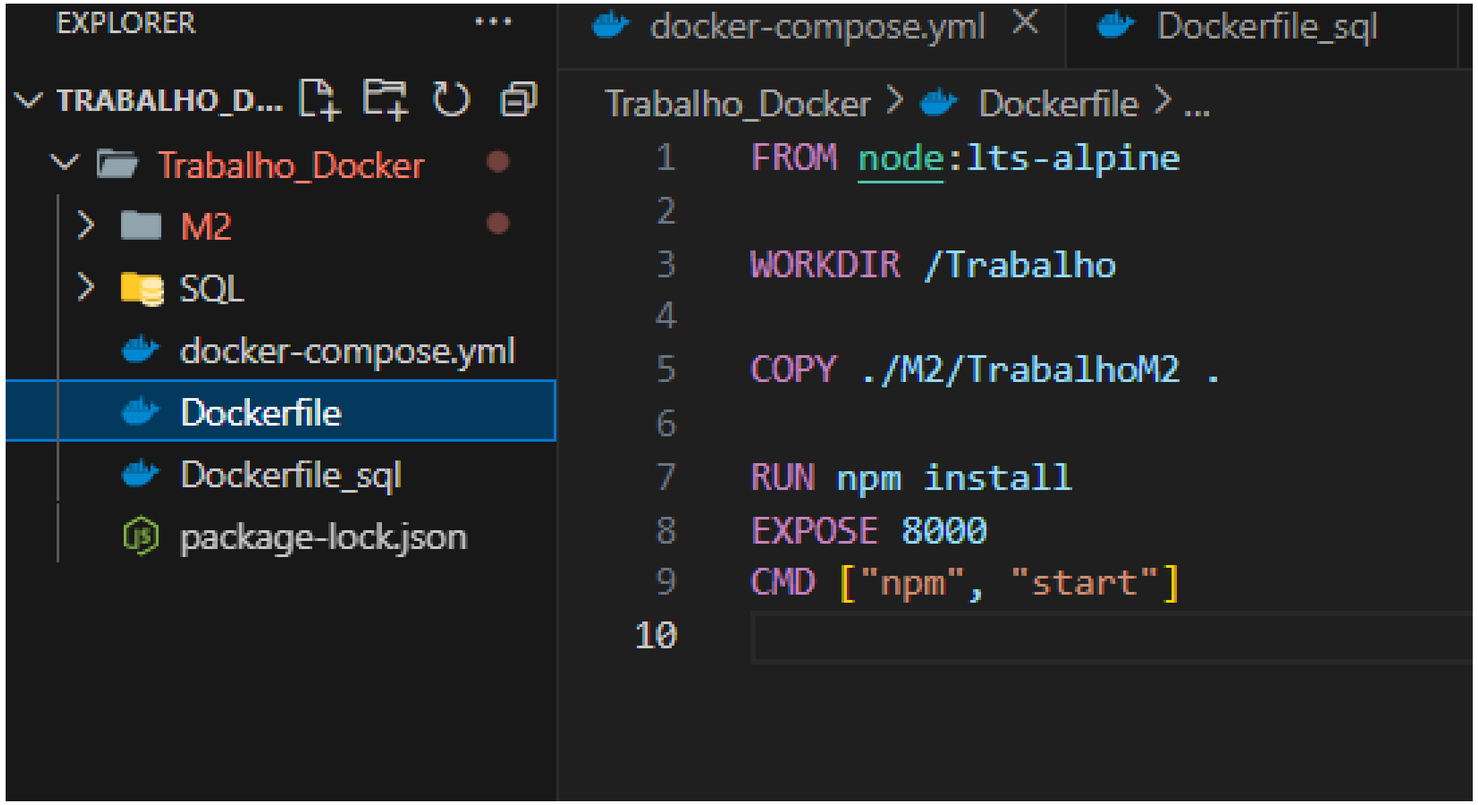
DOCKER

DOCKER-COMPOSE



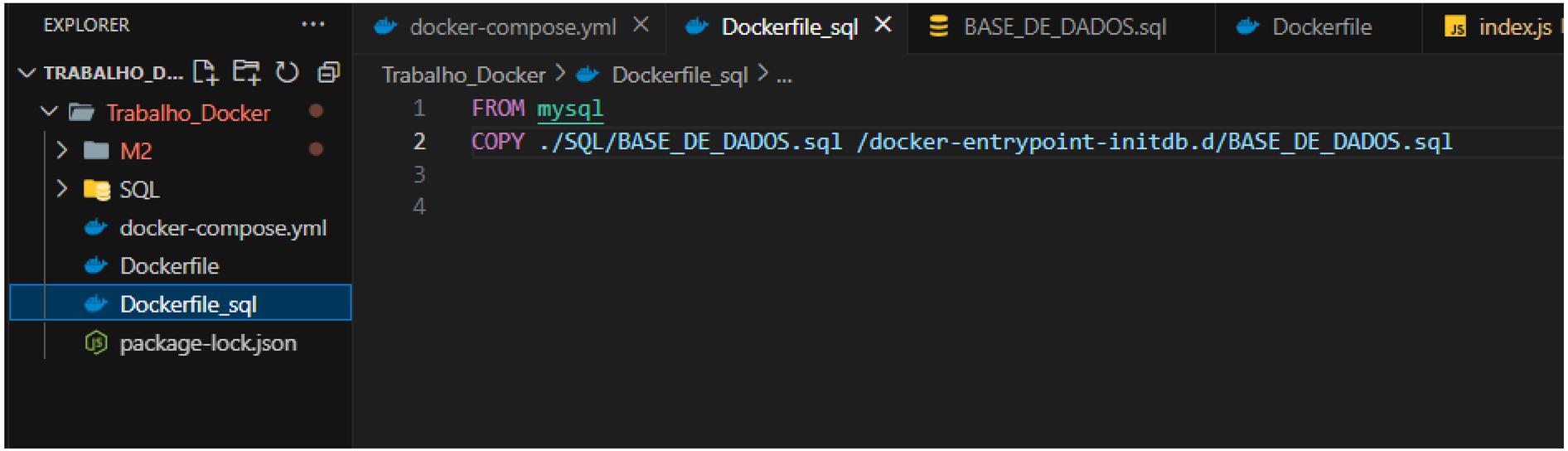
DOCKER

DOCKERFILE



DOCKER

DOCKERFILE_SQL



DOCKER

```
PS C:\Users\beatr\Downloads\Trabalho_Docker> docker builder prune
```

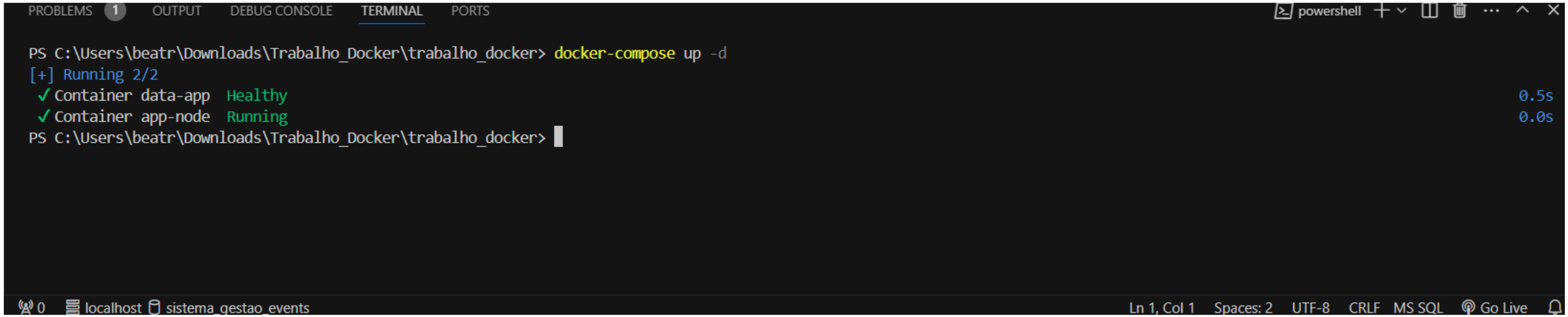
```
PS C:\Users\beatr\Downloads\Trabalho_Docker> docker builder -t app-trabalho
```

```
PS C:\Users\beatr\Downloads\Trabalho_Docker> docker build -f Dockerfile_sql -t mysql:latest .
```

Comando de criação das imagens para o docker.

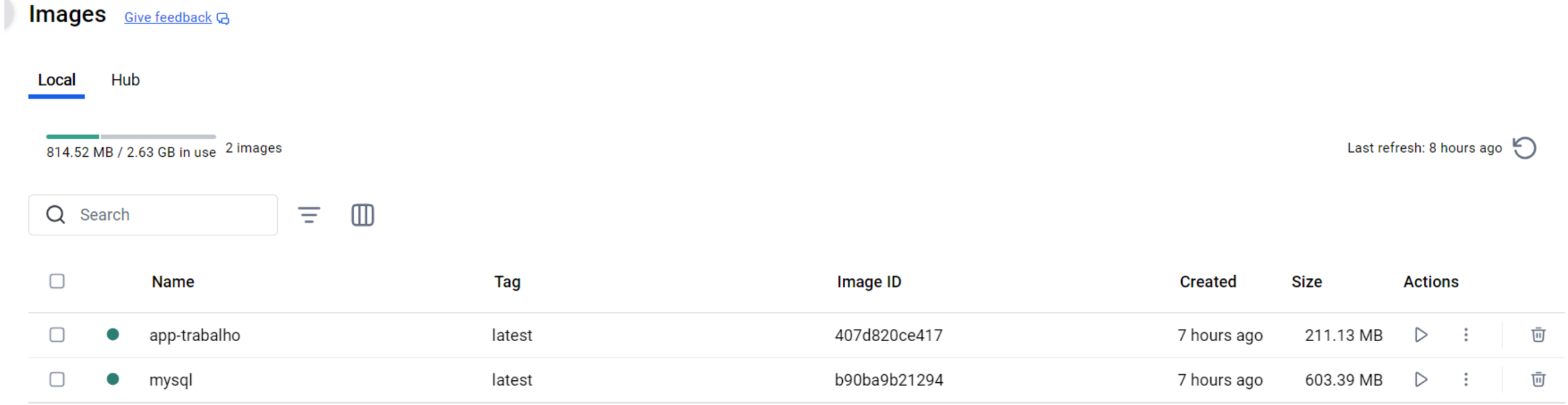
DOCKER

Criação do container



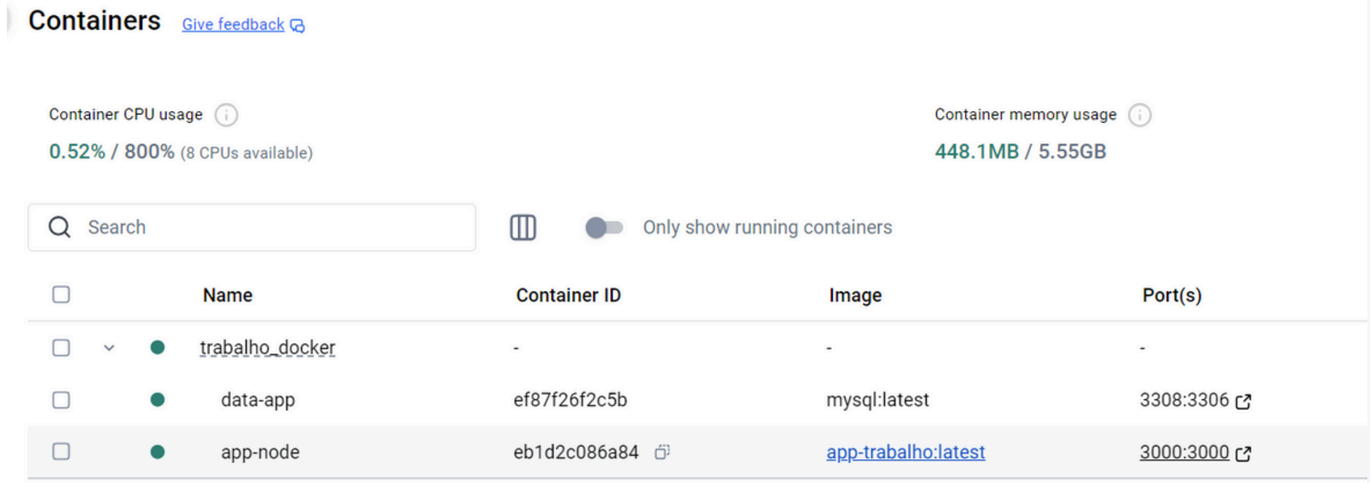
DOCKER

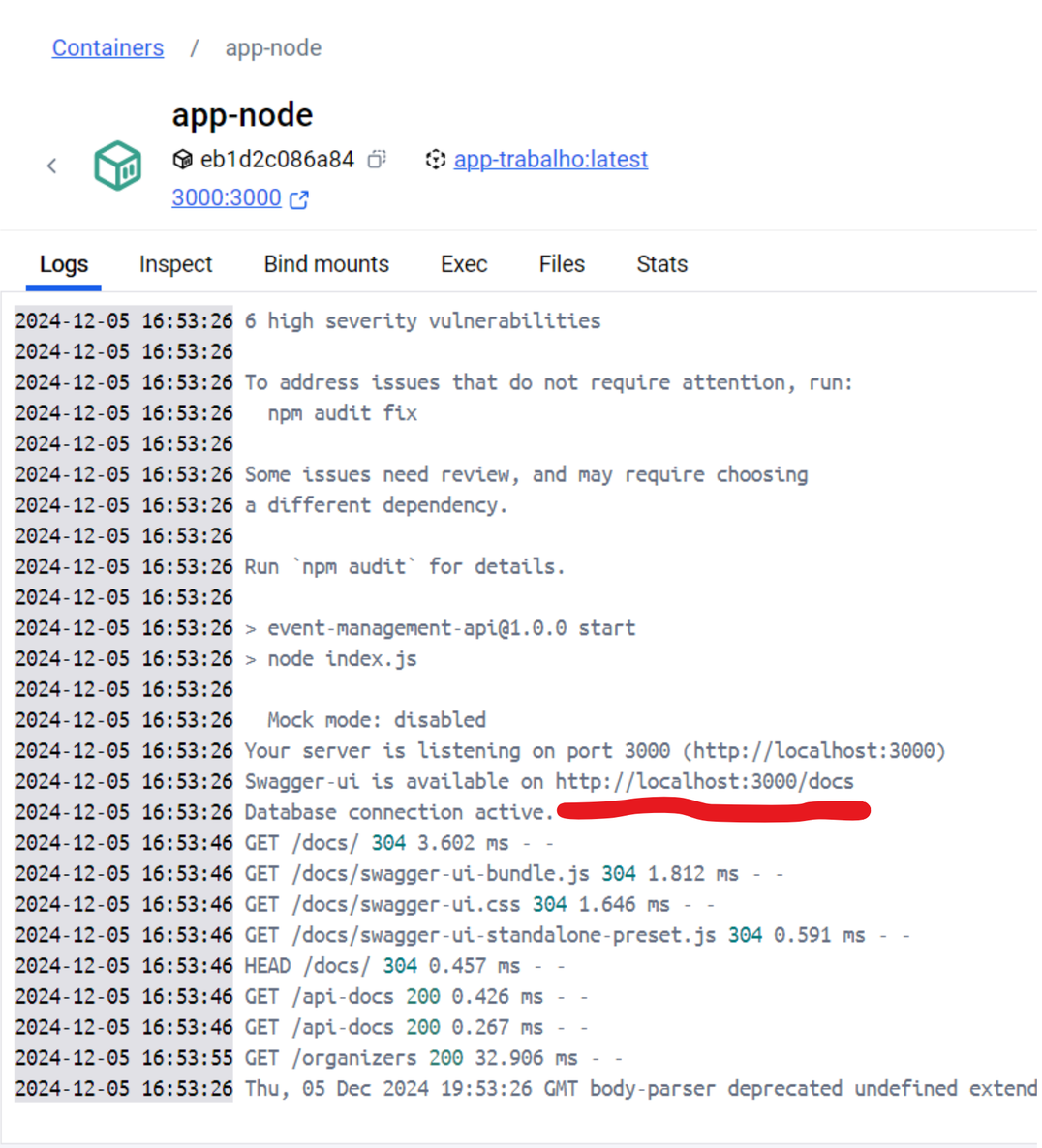
Imagens criadas no docker



DOCKER

Container criado na docker. Ao clicar na app-node conseguimos abria o http://localhost:3000/docs e ir para nossa api.





Obrigado !

Beatriz Almeida
(A044416)
Carolina Fernandes
(A044897)
Pedro Venda
(A045464)