

# Event Management System

Desenvolvimento WEB I

# Event Management 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

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

- Header bar: Body, Cookies, Headers (6), Test Results.
- Toolbox: Pretty (selected), Raw, Preview, Visualize, JSON dropdown, and a copy icon.
- Response body (Pretty printed JSON):

```
1 [  
2 {  
3   "organizerId": 1,  
4   "name": "Festivais Lisboa",  
5   "email": "festivais.lisboa@example.com",  
6   "phone": "+351910000001"  
7 },  
8 {  
9   "organizerId": 2,  
10  "name": "events Porto",  
11  "email": "events.porto@example.com",  
12  "phone": "+351910000002"  
13 }]
```

## CREATE ORGANIZER (POST)

Cria um novo Organizador

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

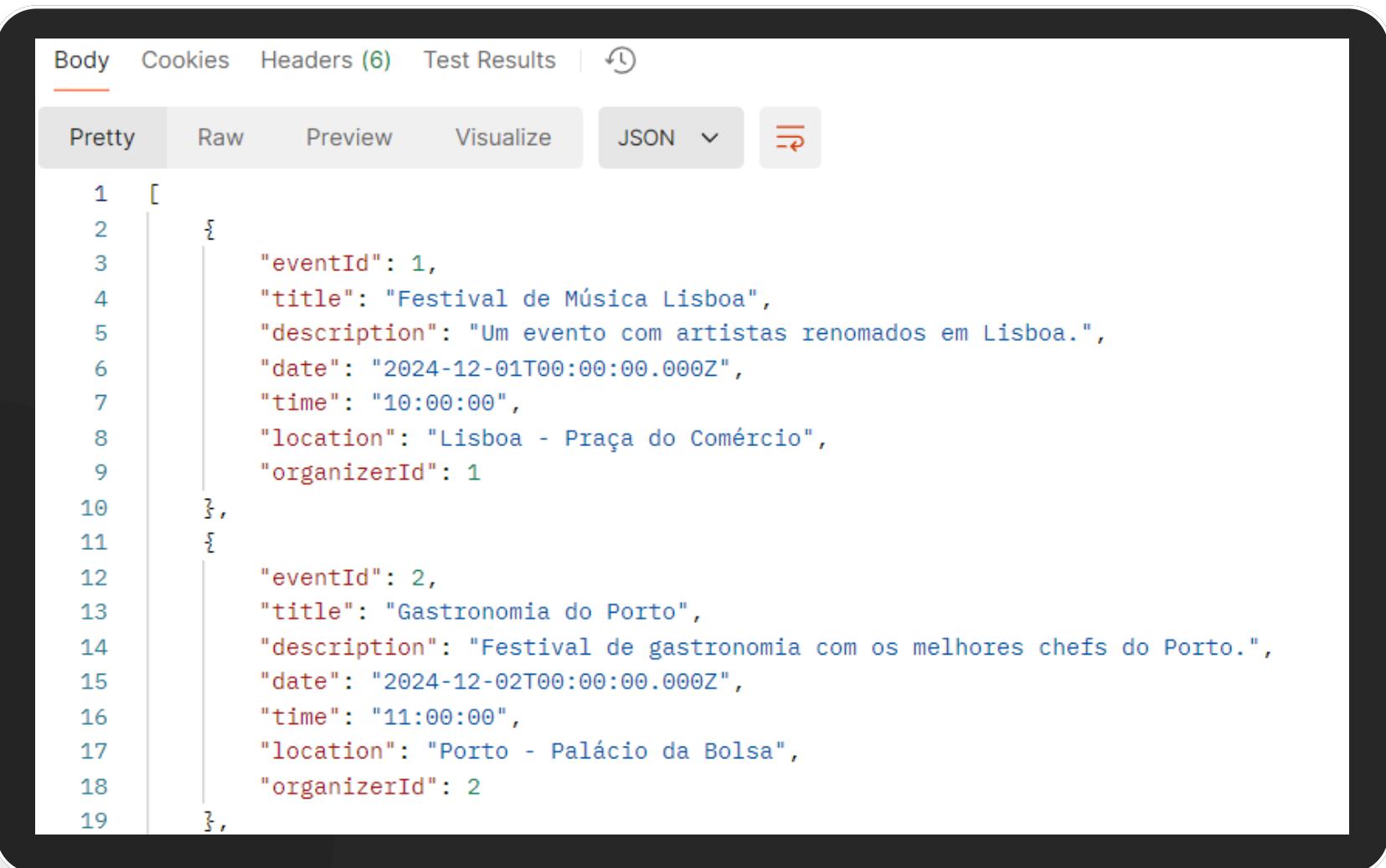
- Header bar: Body, Cookies, Headers (6), Test Results.
- Toolbox: Pretty (selected), Raw, Preview, Visualize, JSON dropdown, and a copy icon.
- Request body (Pretty printed JSON):

```
1 {  
2   "id": 32,  
3   "organizerId": 32,  
4   "name": "Festas Com Cor Setubal",  
5   "email": "festascomcorsetubal@example.com",  
6   "phone": "+351910000000"  
7 }
```

# Events

## EVENTS (GET)

Obtém a Lista de todos os Eventos



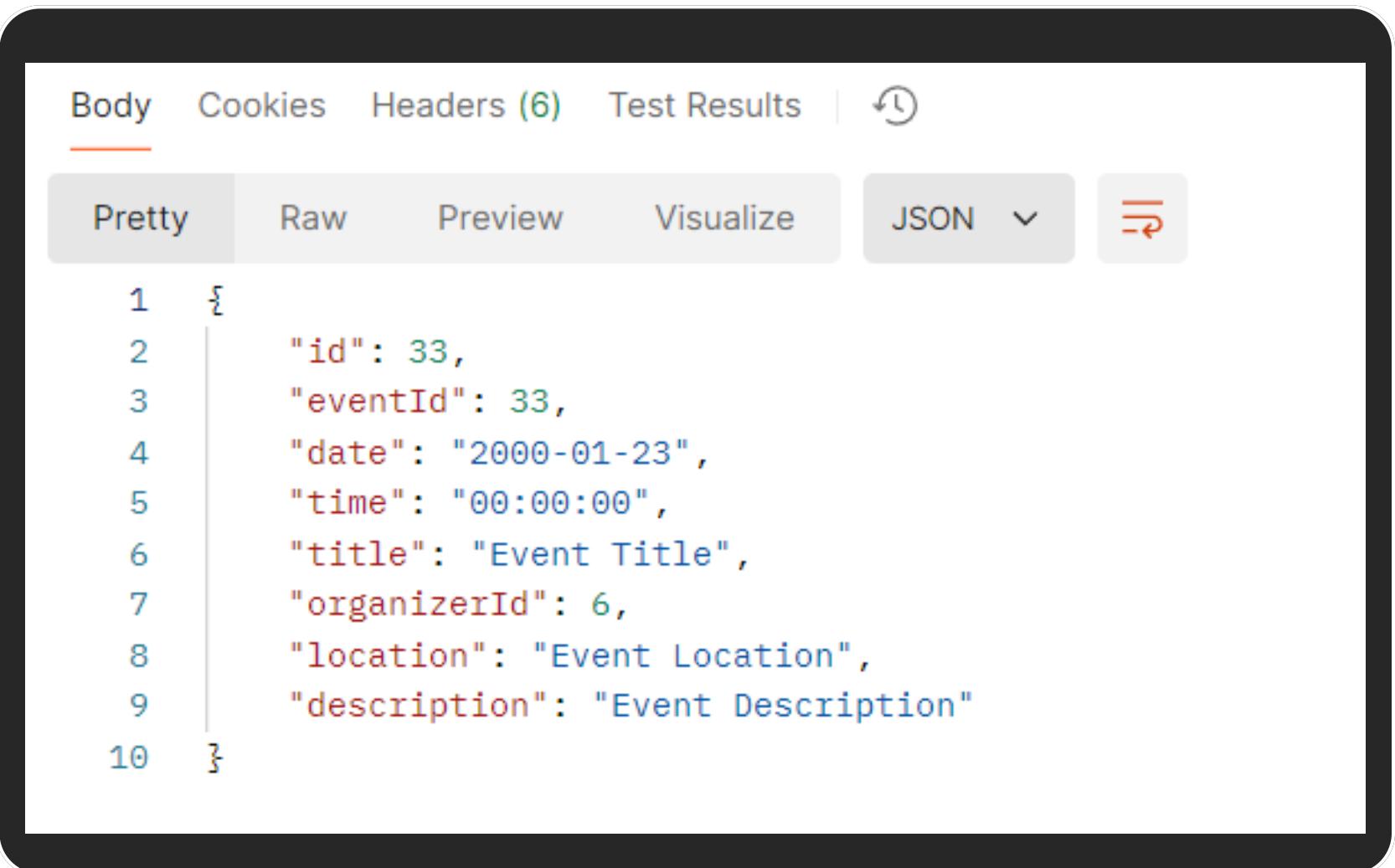
The screenshot shows a REST client interface with the following details:

- Header: Body, Cookies, Headers (6), Test Results.
- Body tab is selected.
- JSON dropdown is set to Pretty.
- Response content:

```
1 [  
2 {  
3   "eventId": 1,  
4   "title": "Festival de Música Lisboa",  
5   "description": "Um evento com artistas renomados em Lisboa.",  
6   "date": "2024-12-01T00:00:00.000Z",  
7   "time": "10:00:00",  
8   "location": "Lisboa - Praça do Comércio",  
9   "organizerId": 1  
10 },  
11 {  
12   "eventId": 2,  
13   "title": "Gastronomia do Porto",  
14   "description": "Festival de gastronomia com os melhores chefs do Porto.",  
15   "date": "2024-12-02T00:00:00.000Z",  
16   "time": "11:00:00",  
17   "location": "Porto - Palácio da Bolsa",  
18   "organizerId": 2  
19 },
```

## CREATE EVENT (POST)

Cria um novo Evento



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

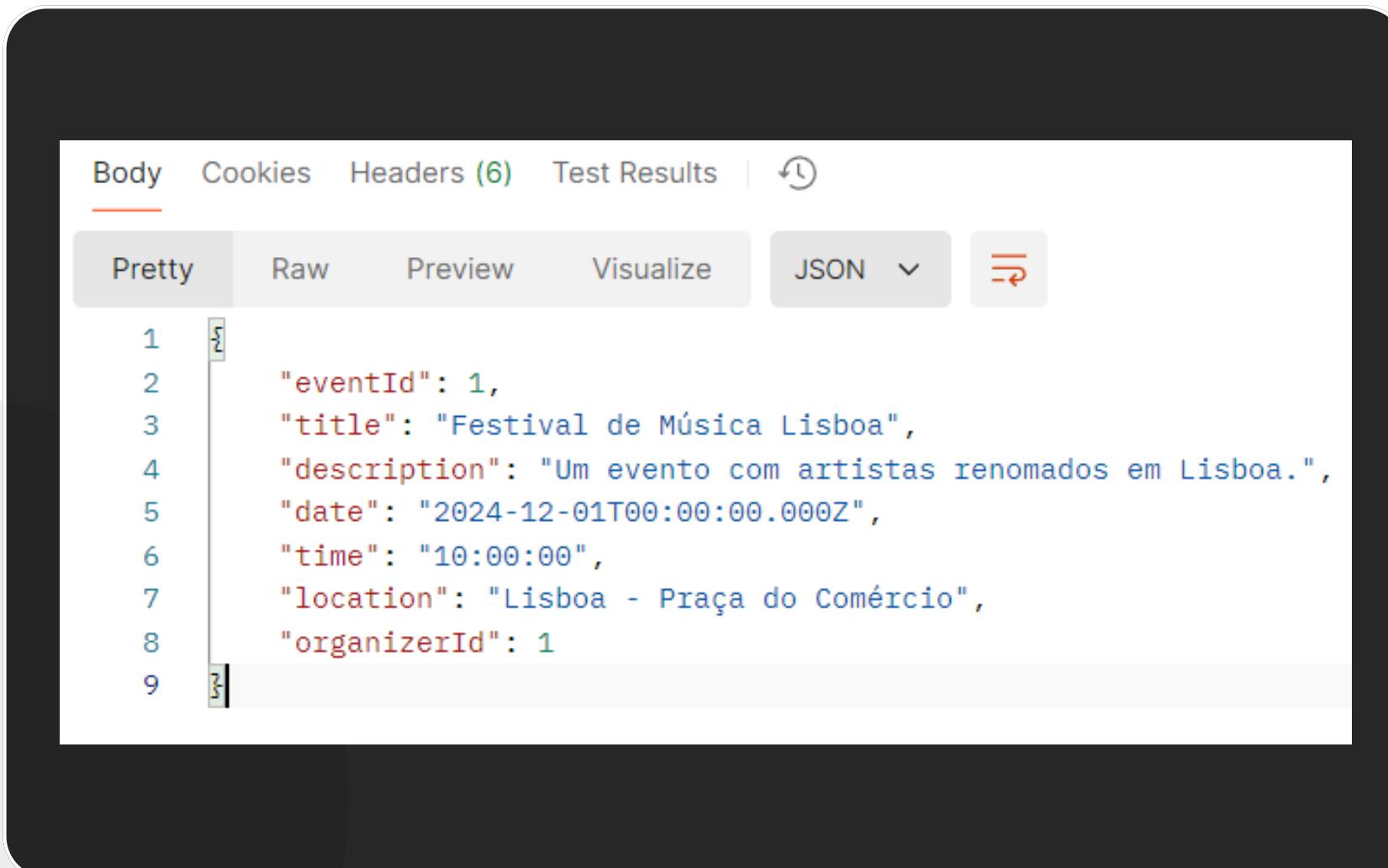
- Header: Body, Cookies, Headers (6), Test Results.
- Body tab is selected.
- JSON dropdown is set to Pretty.
- Request content:

```
1 {  
2   "id": 33,  
3   "eventId": 33,  
4   "date": "2000-01-23",  
5   "time": "00:00:00",  
6   "title": "Event Title",  
7   "organizerId": 6,  
8   "location": "Event Location",  
9   "description": "Event Description"  
10 }
```

# Events

## SPECIFIC EVENTS (GET)

Obtém um Evento Específico (“eventId”)

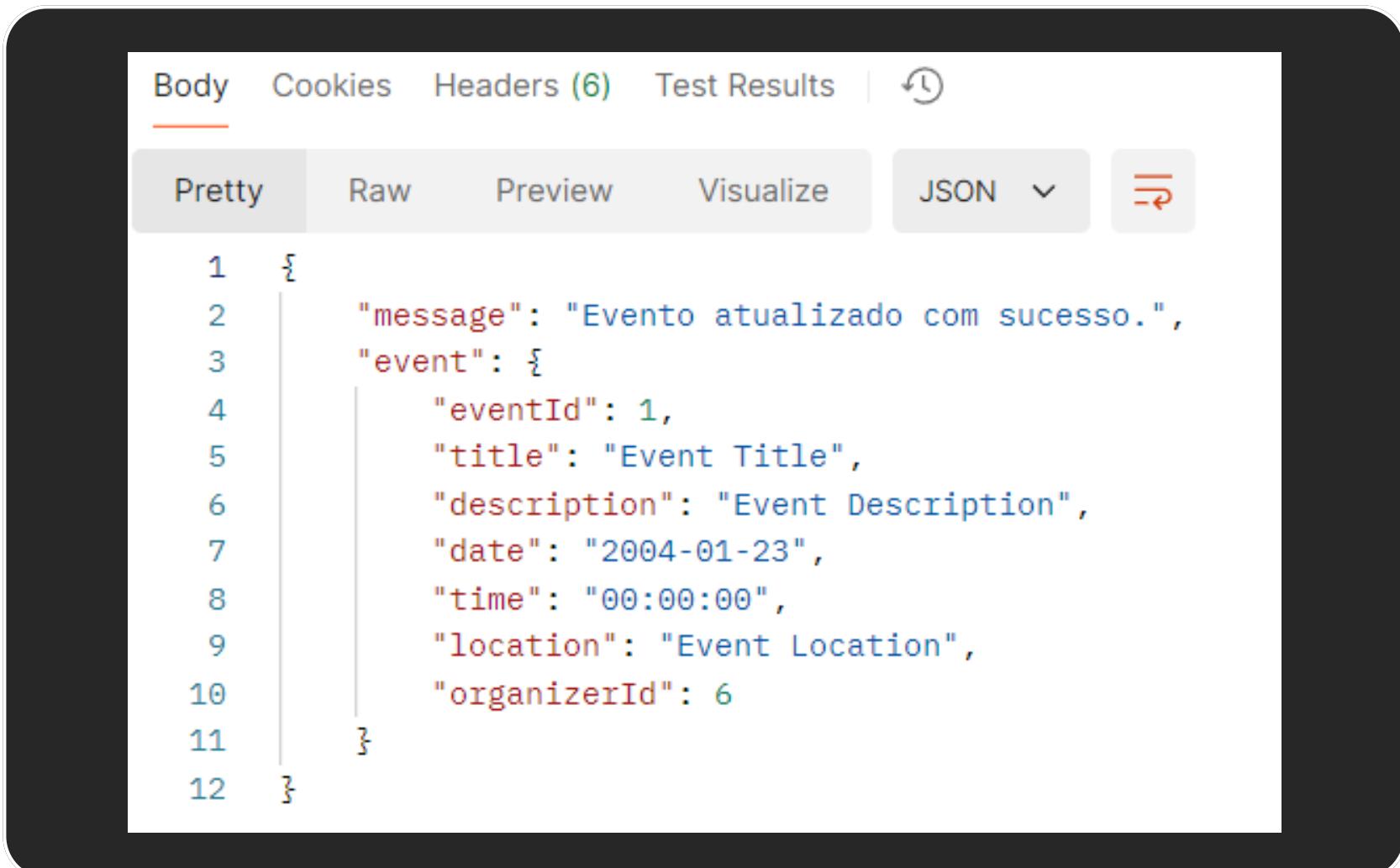


A screenshot of a REST client interface. The top navigation bar includes tabs for Body, Cookies, Headers (6), and Test Results, with Headers being the active tab. Below the tabs is a toolbar with Pretty, Raw, Preview, Visualize, and JSON dropdown buttons. The main area displays a JSON response with line numbers on the left. The JSON data represents an event with fields: eventId, title, description, date, time, location, and organizerId.

```
1 {  
2   "eventId": 1,  
3   "title": "Festival de Música Lisboa",  
4   "description": "Um evento com artistas renomados em Lisboa.",  
5   "date": "2024-12-01T00:00:00.000Z",  
6   "time": "10:00:00",  
7   "location": "Lisboa - Praça do Comércio",  
8   "organizerId": 1  
9 }
```

## UPDATE SPECIFIC EVENT (PUT)

Atualiza informação de um Evento Específico (“eventId”)



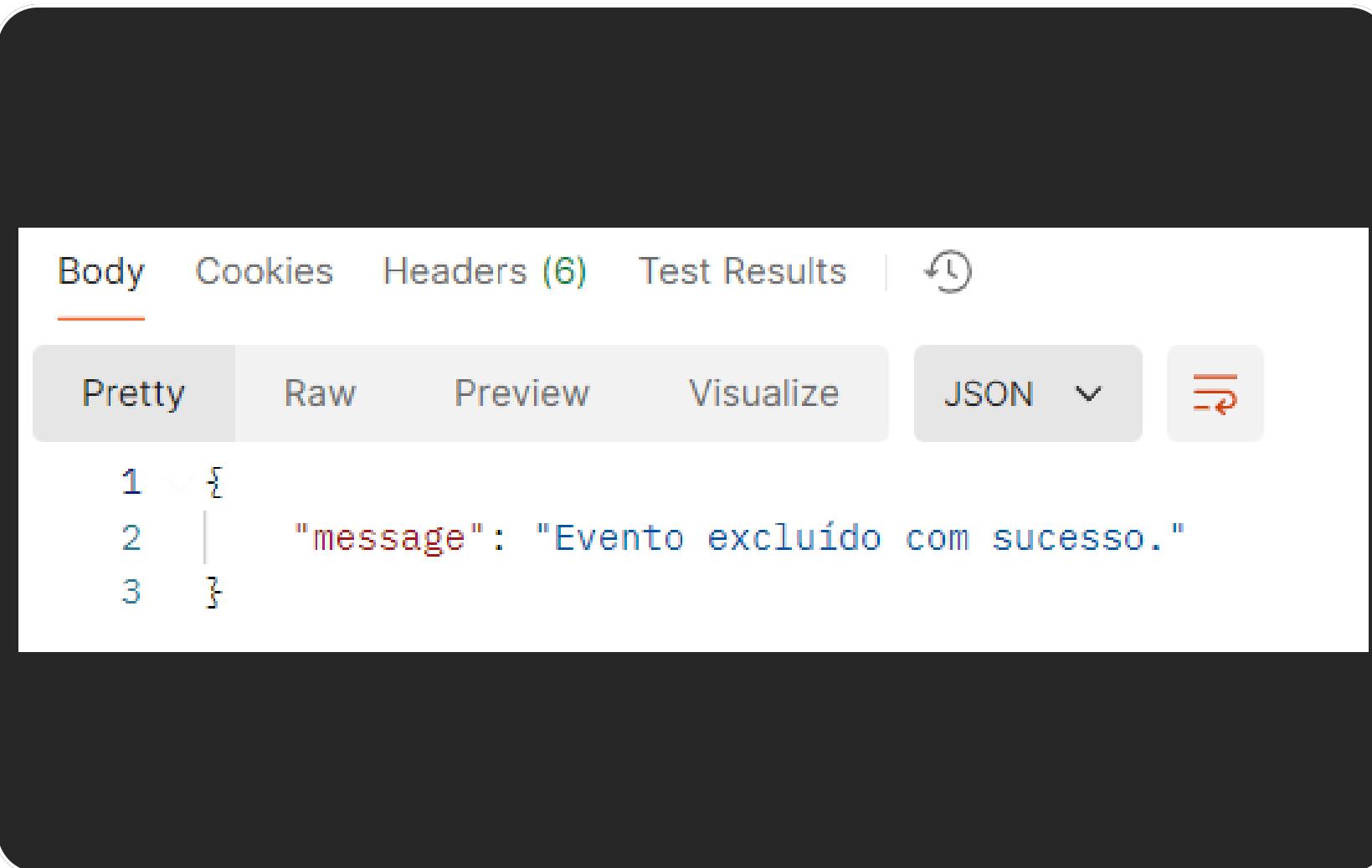
A screenshot of a REST client interface. The top navigation bar includes tabs for Body, Cookies, Headers (6), and Test Results, with Body being the active tab. Below the tabs is a toolbar with Pretty, Raw, Preview, Visualize, and JSON dropdown buttons. The main area displays a JSON request body with line numbers on the left. The JSON data contains a message key and an event object with fields: eventId, title, description, date, time, location, and organizerId.

```
1 {  
2   "message": "Evento atualizado com sucesso.",  
3   "event": {  
4     "eventId": 1,  
5     "title": "Event Title",  
6     "description": "Event Description",  
7     "date": "2004-01-23",  
8     "time": "00:00:00",  
9     "location": "Event Location",  
10    "organizerId": 6  
11  }  
12 }
```

# Events

## ***DELETE SPECIFIC EVENT (DELETE)***

**Apaga um Evento específico (“eventId”)**



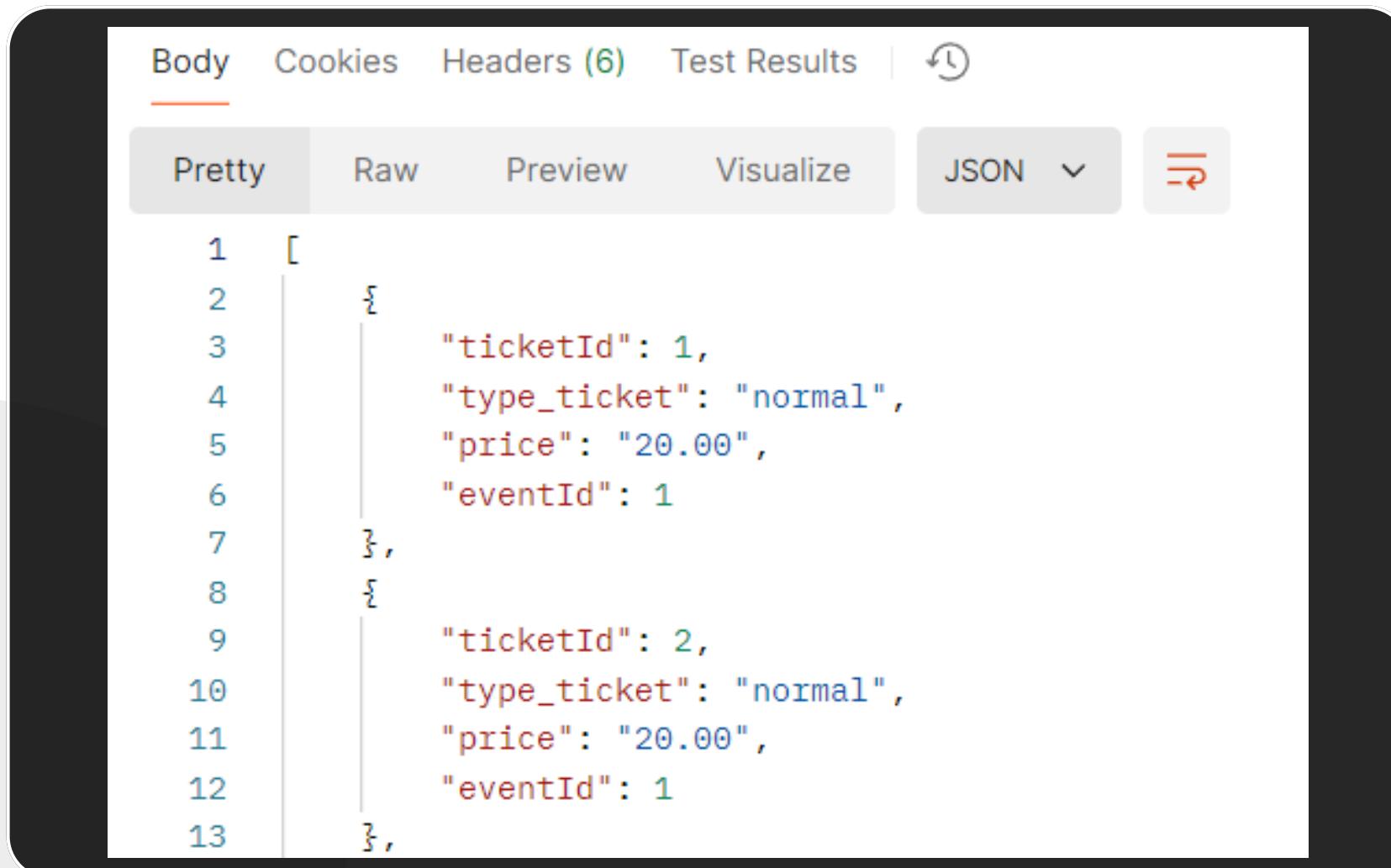
The screenshot shows a user interface for a REST API testing tool. At the top, there are tabs for "Body", "Cookies", "Headers (6)", "Test Results", and a refresh icon. Below the tabs is a toolbar with buttons for "Pretty", "Raw", "Preview", "Visualize", "JSON" (with a dropdown arrow), and a copy icon. The main area displays a JSON response with line numbers:

```
1 {  
2   "message": "Evento excluído com sucesso."  
3 }
```

# Tickets

## **TICKETS (GET)**

**Obtém a Lista de todos os bilhetes**



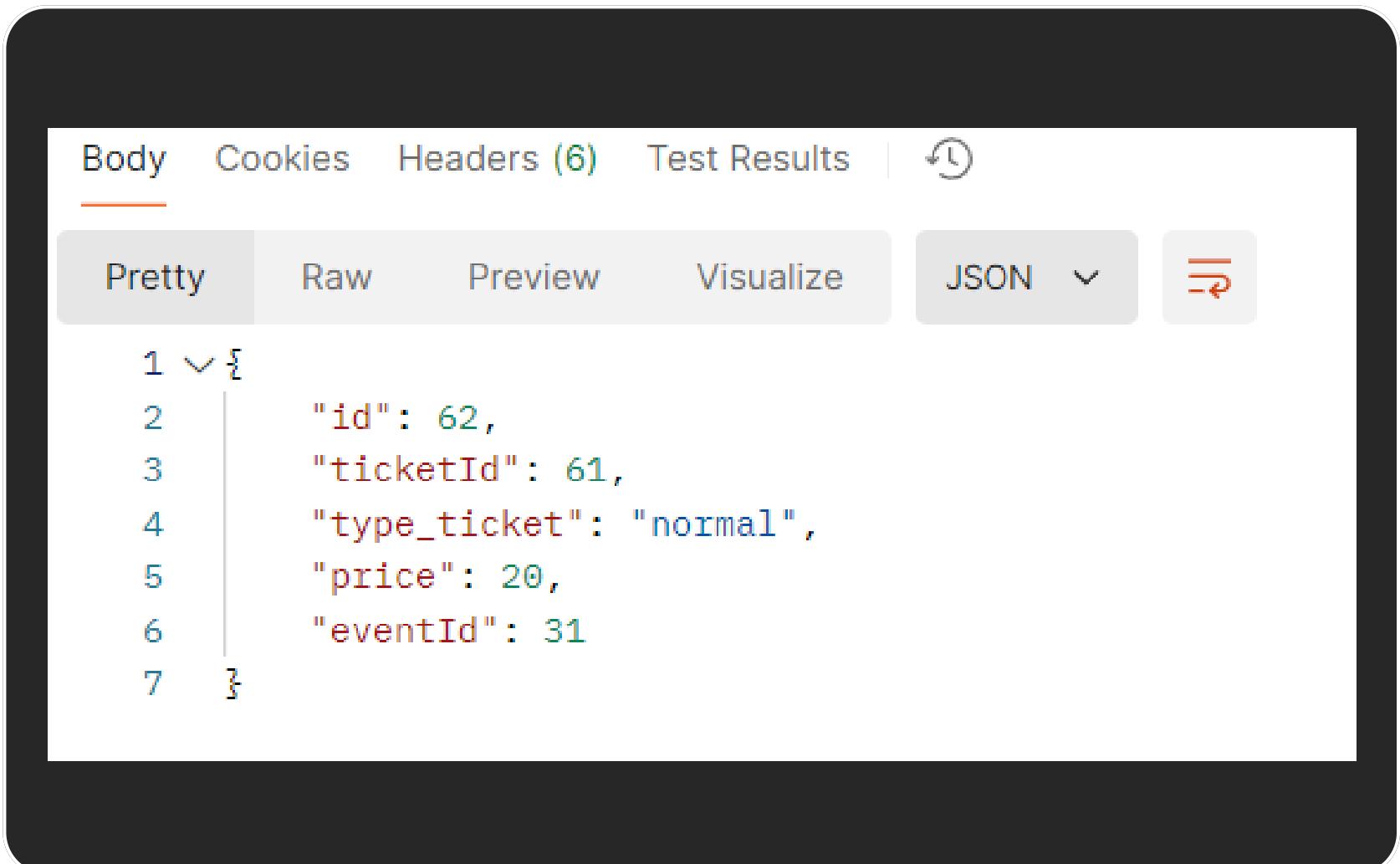
The screenshot shows a REST client interface with the following details:

- Header tabs: Body, Cookies, Headers (6), Test Results.
- Body tab selected.
- Format: JSON (Pretty).
- Response content:

```
1 [  
2 {  
3   "ticketId": 1,  
4   "type_ticket": "normal",  
5   "price": "20.00",  
6   "eventId": 1  
7 },  
8 {  
9   "ticketId": 2,  
10  "type_ticket": "normal",  
11  "price": "20.00",  
12  "eventId": 1  
13 }]
```

## **CREATE TICKET (POST)**

**Cria um novo Bilhete**



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

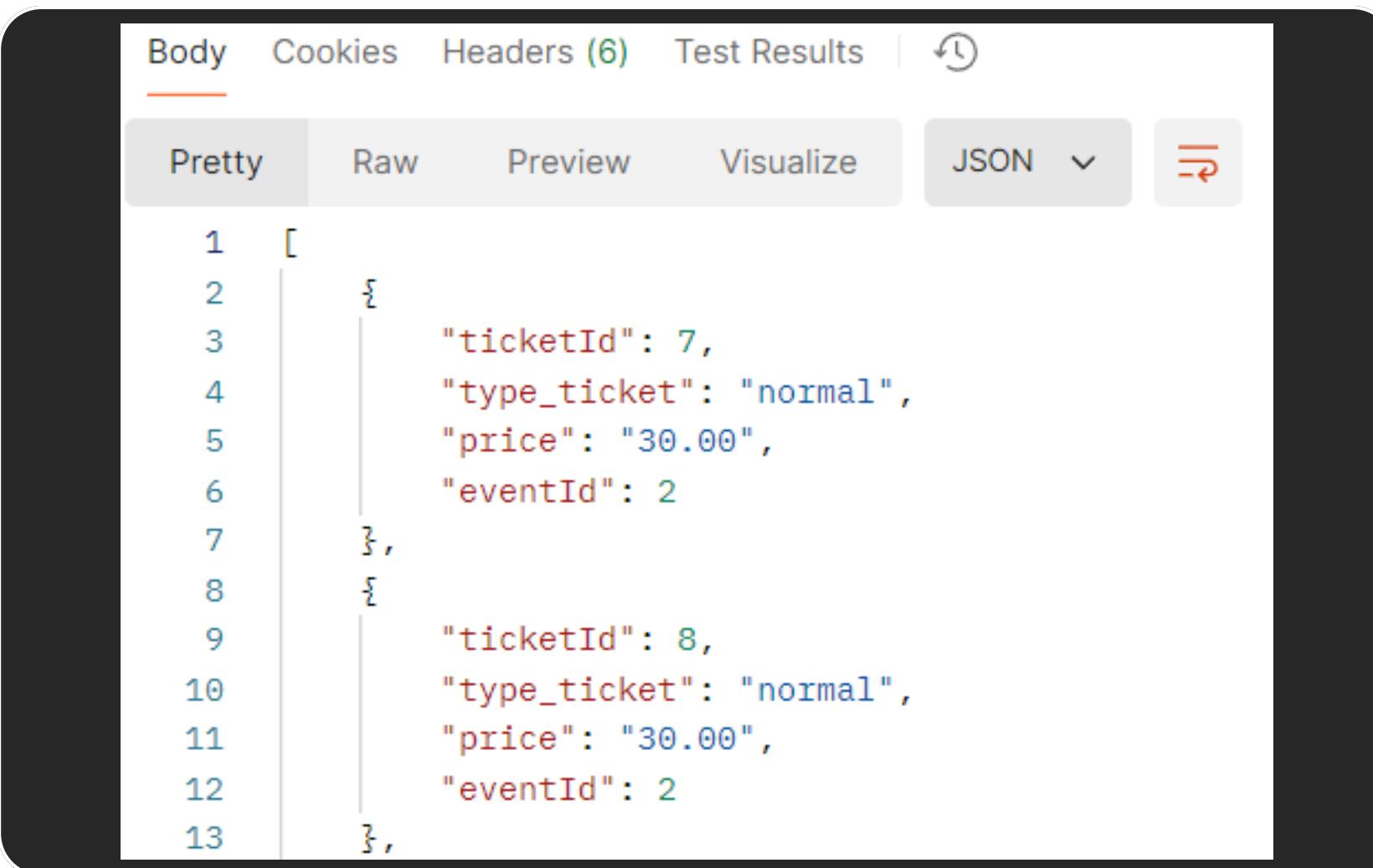
- Header tabs: Body, Cookies, Headers (6), Test Results.
- Body tab selected.
- Format: JSON (Pretty).
- Request content:

```
1 {  
2   "id": 62,  
3   "ticketId": 61,  
4   "type_ticket": "normal",  
5   "price": 20,  
6   "eventId": 31  
7 }
```

# Tickets

## SPECIFIC TICKET (GET)

Obtém Bilhete específico (ticketId)



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

- Header bar: Body, Cookies, Headers (6), Test Results.
- Toolbox: Pretty, Raw, Preview, Visualize, JSON dropdown, and a copy icon.
- JSON response (Pretty printed):

```
1  [
2    {
3      "ticketId": 7,
4      "type_ticket": "normal",
5      "price": "30.00",
6      "eventId": 2
7    },
8    {
9      "ticketId": 8,
10     "type_ticket": "normal",
11     "price": "30.00",
12     "eventId": 2
13   },
```

# Participants

## PARTICIPANTS (GET)

Obtém a Lista de todos os Participantes

The screenshot shows a REST client interface with the 'Body' tab selected. The response is displayed in 'Pretty' JSON format, showing a list of participants. Each participant is represented by an object with properties: participantId, name, email, status, and ticketId.

```
1 [  
2 {  
3   "participantId": 1,  
4   "name": "João Silva",  
5   "email": "joao.silva@example.com",  
6   "status": "confirmado",  
7   "ticketId": 1  
8 },  
9 {  
10  "participantId": 2,  
11  "name": "Maria Oliveira",  
12  "email": "maria.oliveira@example.com",  
13  "status": "pendente",  
14  "ticketId": 2  
15 }]
```

## CREATE PARTICIPANT (POST)

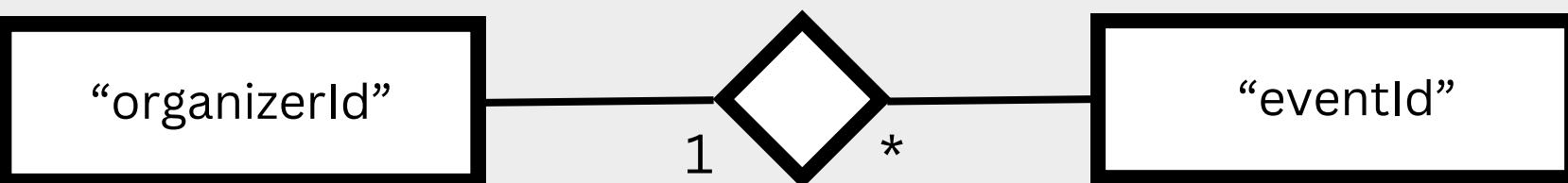
Cria um novo Participante

The screenshot shows a REST client interface with the 'Body' tab selected. The request body is displayed in 'Pretty' JSON format, representing a single participant object with properties: participantId, name, email, status, and ticketId.

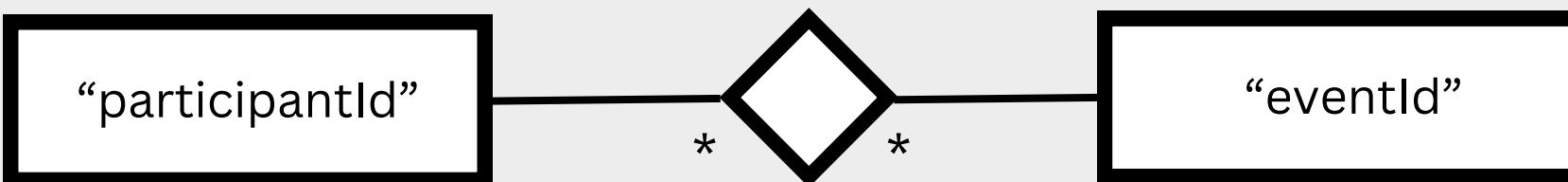
```
1 {  
2   "participantId": 31,  
3   "name": "John Doe",  
4   "email": "johndoe@example.com",  
5   "status": "confirmado",  
6   "ticketId": 62  
7 }
```

# Cardinality Relationship

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.

The screenshot shows the Event Management API documentation in the Swagger Editor. At the top, it says "Event Management API 1.0.0 OAS3". Below that, it says "api-docs" and "API for managing events, organizers, tickets, and participants." It also mentions "Contact Group 04".

At the top left, there's a "Server" dropdown set to "http://localhost:8080 - Localhost API Server". Below it, the "Computed URL" is "http://localhost:8080".

Under "Server variables", the "basePath" is set to "/".

The main content area is organized into sections:

- Organizers**:
  - GET /organizers - Retrieve a list of organizers
  - POST /organizers - Create a new organizer
- Events**:
  - GET /events - Retrieve a list of events
  - POST /events - Create a new event
  - GET /events/{eventId} - Retrieve a specific event
  - PATCH /events/{eventId} - Update an existing event
  - DELETE /events/{eventId} - Deletes an event
- Tickets**:
  - GET /tickets - Retrieve a list of tickets
  - POST /tickets - Create a new ticket
  - GET /tickets/event/{eventId} - Retrieve tickets for a specific event
- Participants**:
  - GET /participants - Retrieve a list of participants
  - POST /participants - Create a new participant
- Schemas**:
  - Organizer >
  - Event >
  - Ticket >
  - Participant >

# Connection to DataBase

## db.js

```
'use strict';

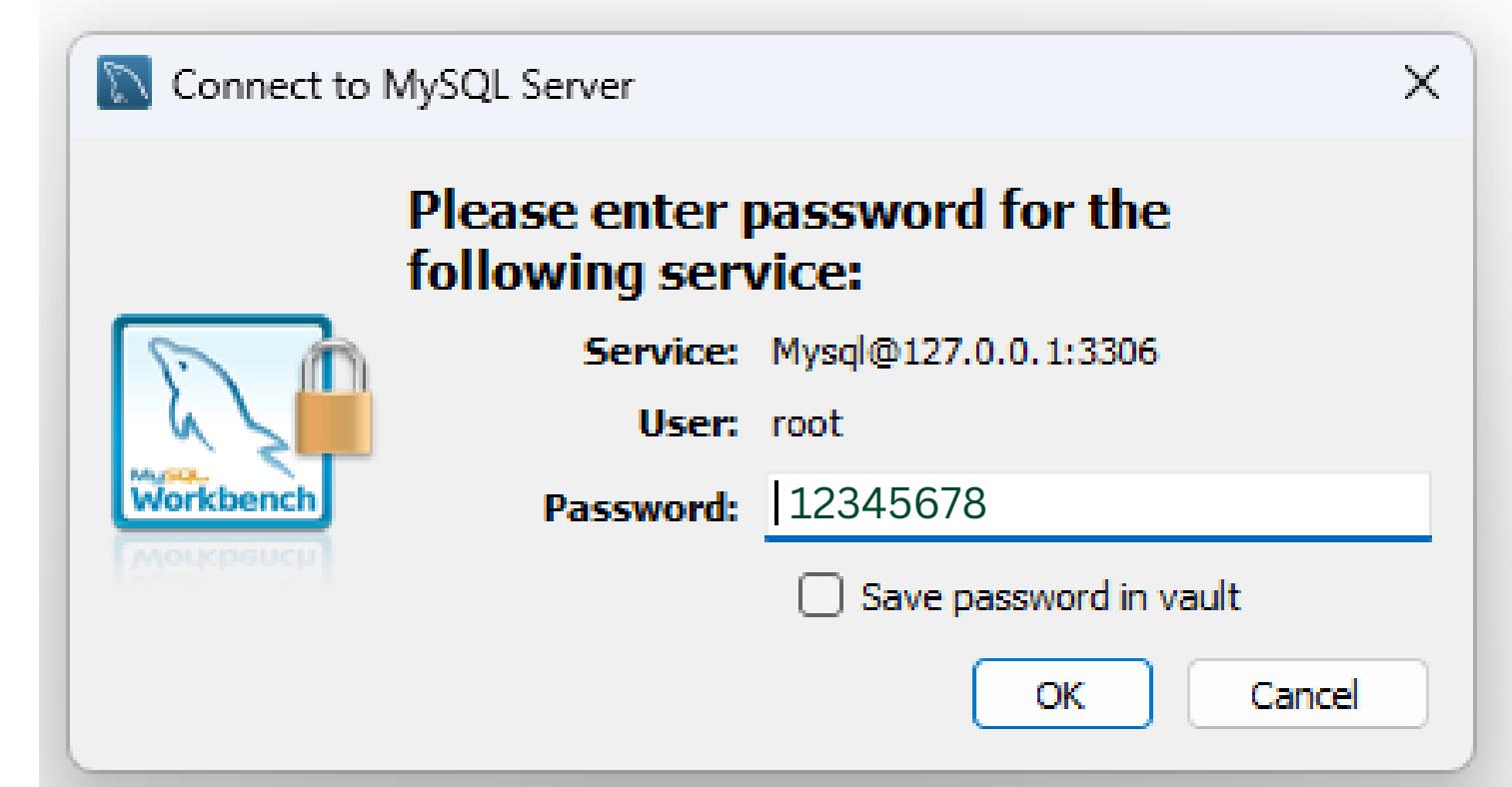
var mysql = require('mysql2');

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)



## 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).
  - ticketId : Referência ao ingresso em que o participante está inscrito.

# STRUCTURE



# Data Base (MYSQL)

```
-- 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)
);

-- 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'),
•••

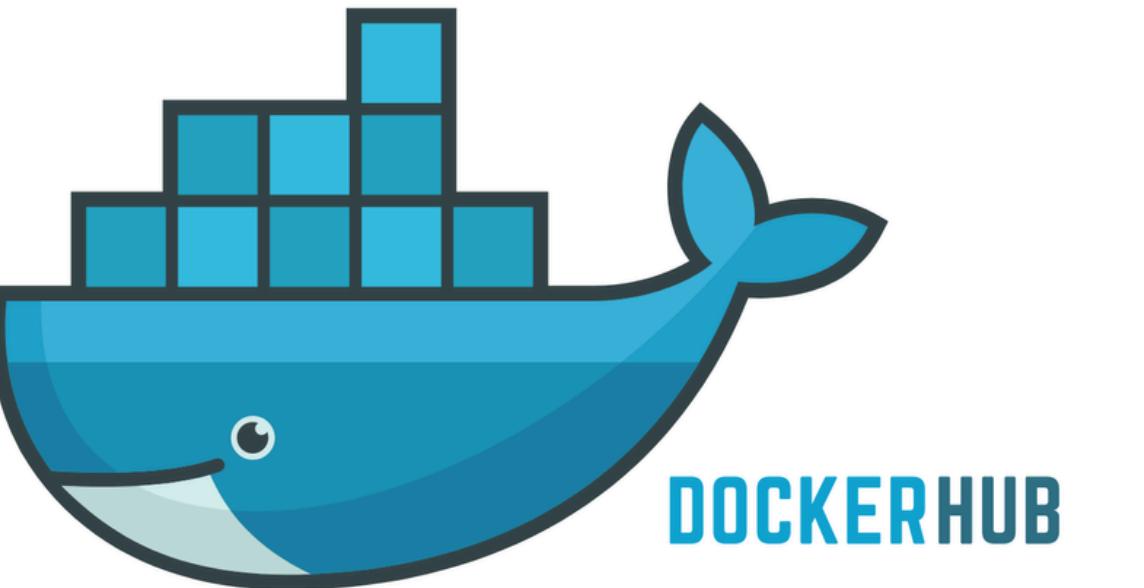
-- Inserindo events
INSERT INTO events (eventId, title, description, date, time, location, organizerId) VALUES
(1, 'Festival de Música Lisboa', 'Um evento com artistas renomados em Lisboa.', '2024-12-01', '10:00:00', 'Lisboa - Praça do Comércio', 1),
(2, 'Gastronomia do Porto', 'Festival de gastronomia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

-- 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),
•••

-- 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

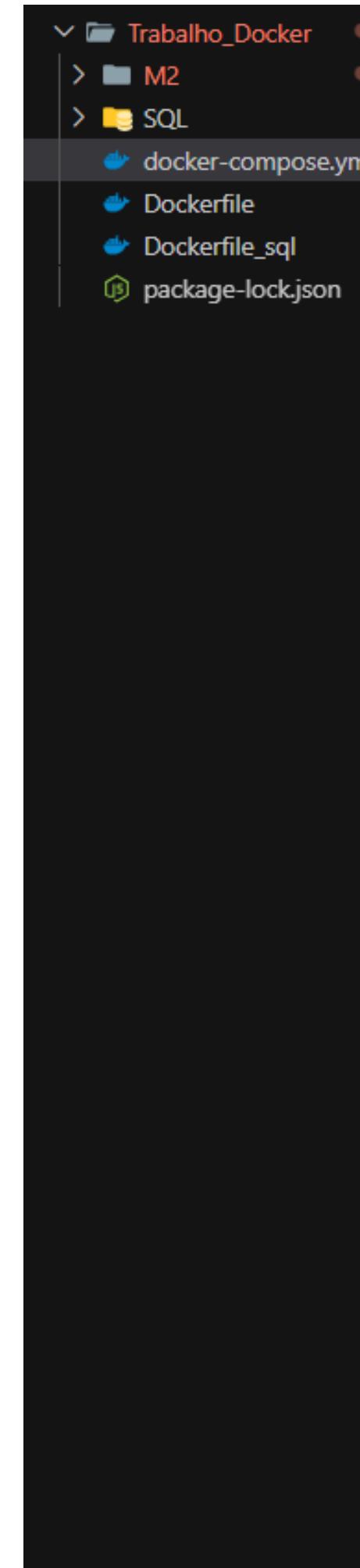
BIA

A044416



# DOCKER

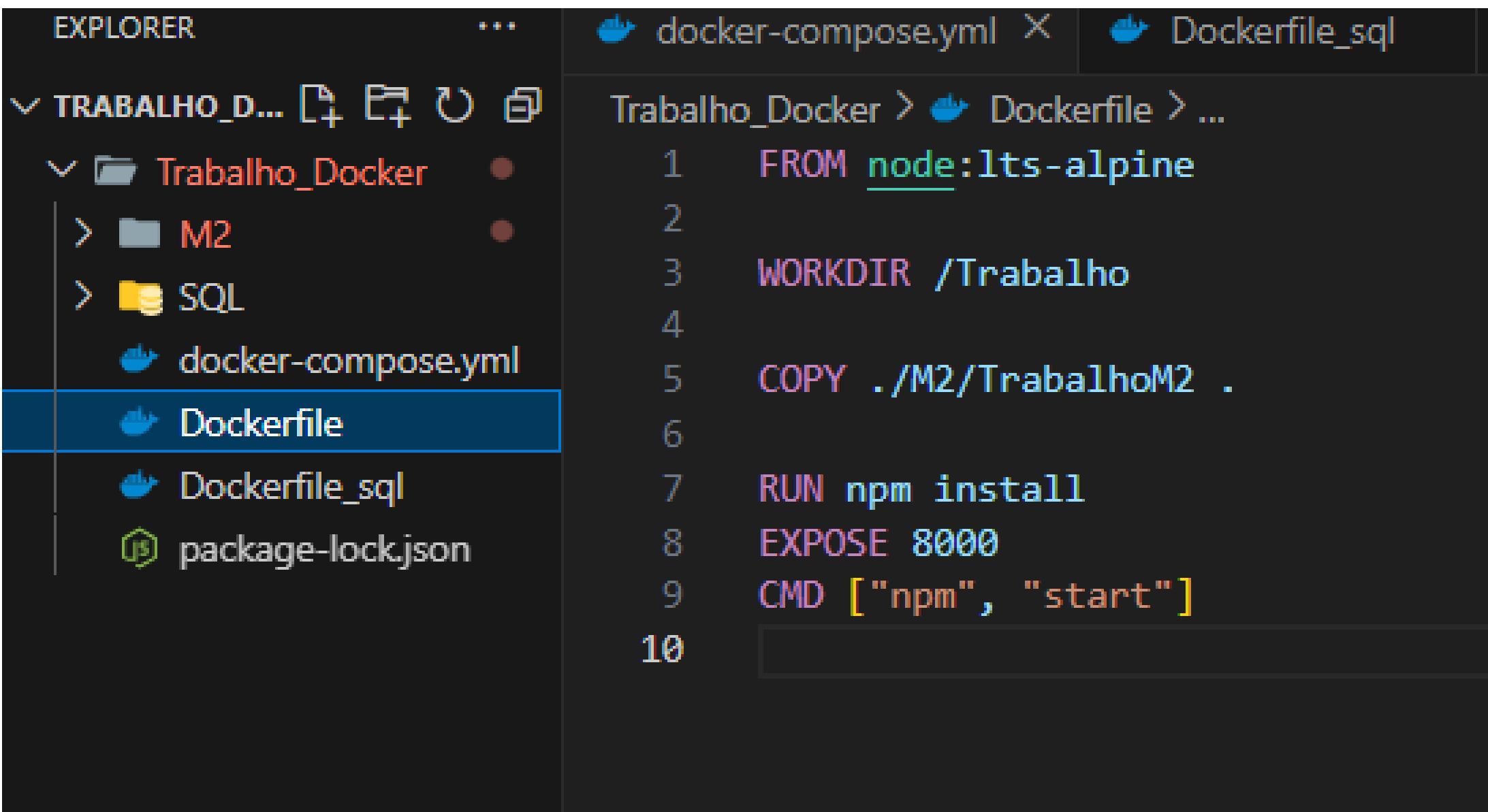
## DOCKER-COMPOSE



```
1 docker-compose.yml - The Compose specification establishes a standard for the definition
2 services:
3   node:
4     build:
5       context: ./
6       dockerfile: Dockerfile
7       image: app-trabalho:latest
8       container_name: app-node
9       depends_on:
10      mysql:
11        condition: service_healthy
12      ports:
13        - "3000:3000"
14      networks:
15        - app_network
16      environment:
17        DB_HOST: mysql
18        DB_PORT: 3306
19        DB_USER: root
20        DB_PASSWORD: "12345678"
21        DB_NAME: sistema_gestao_events
22
23      mysql:
24        image: mysql:latest
25        container_name: data-app
26        environment:
27          MYSQL_ROOT_PASSWORD: "12345678"
28          MYSQL_DATABASE: "sistema_gestao_events"
29        healthcheck:
30          test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
31          interval: 10s
32          timeout: 5s
33          retries: 3
34        ports:
35          - "3308:3306"
36        volumes:
37          - data:/var/lib/mysql
38        networks:
39          - app_network
40
41      networks:
42        app_network:
```

# DOCKER

## DOCKERFILE



The image shows a screenshot of a code editor interface, likely Visual Studio Code, displaying a Dockerfile. The left pane shows a file tree with a folder named 'Trabalho\_Docker' containing subfolders 'M2' and 'SQL', and files 'docker-compose.yml', 'Dockerfile', 'Dockerfile\_sql', and 'package-lock.json'. The 'Dockerfile' file is currently selected and highlighted with a blue background. The right pane displays the contents of the Dockerfile, which defines a Node.js application container.

```
FROM node:lts-alpine
WORKDIR /Trabalho
COPY ./M2/TrabalhoM2 .
RUN npm install
EXPOSE 8000
CMD ["npm", "start"]
```



# DOCKER



## DOCKERFILE\_SQL

The screenshot shows a code editor interface with the following details:

- EXPLORER** sidebar: Shows a project structure under "TRABALHO\_D...".
  - Trabalho\_Docker (selected)
  - M2
  - SQL
    - docker-compose.yml
    - Dockerfile
    - Dockerfile\_SQL** (highlighted in blue)
    - package-lock.json
- CODE** tab bar: docker-compose.yml, Dockerfile\_SQL (highlighted), BASE\_DE\_DADOS.sql, Dockerfile, index.js.
- Content Area:**

```
FROM mysql
COPY ./SQL/BASE_DE_DADOS.sql /docker-entrypoint-initdb.d/BASE_DE_DADOS.sql
```

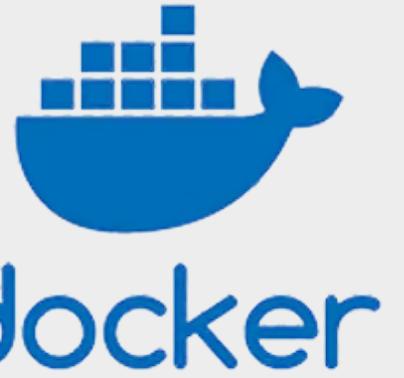
# DOCKER

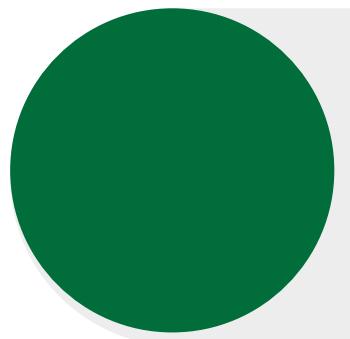
```
PS C:\Users\beatr\Downloads\Trabalho_Docker> docker builder prune
```

```
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

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\beatr\Downloads\Trabalho\_Docker\trabalho\_docker> docker-compose up -d  
[+] Running 2/2  
✓ Container data-app Healthy 0.5s  
✓ Container app-node Running 0.0s  
PS C:\Users\beatr\Downloads\Trabalho\_Docker\trabalho\_docker>

# DOCKER



Imagenes criadas no docker

Images [Give feedback](#)

Local Hub

814.52 MB / 2.63 GB in use 2 images Last refresh: 8 hours ago

Search

<input type="checkbox"/>	Name	Tag	Image ID	Created	Size	Actions
<input type="checkbox"/>	app-trabalho	latest	407d820ce417	7 hours ago	211.13 MB	<a href="#">▶</a> <a href="#">⋮</a> <a href="#">trash</a>
<input type="checkbox"/>	mysql	latest	b90ba9b21294	7 hours ago	603.39 MB	<a href="#">▶</a> <a href="#">⋮</a> <a href="#">trash</a>

# DOCKER

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

Containers [Give feedback](#)

Container CPU usage (1)  
0.52% / 800% (8 CPUs available)

Container memory usage (1)  
448.1MB / 5.55GB

Search

Only show running containers

<input type="checkbox"/>	Name	Container ID	Image	Port(s)
<input type="checkbox"/>	trabalho_docker	-	-	-
<input type="checkbox"/>	data-app	ef87f26f2c5b	mysql:latest	3308:3306 ↗
<input type="checkbox"/>	app-node	eb1d2c086a84 ↗	<a href="#">app-trabalho:latest</a>	<a href="#">3000:3000 ↗</a>

[Containers](#) / app-node

## app-node

<  eb1d2c086a84 ↗  [app-trabalho:latest](#)  
[3000:3000 ↗](#)

[Logs](#) [Inspect](#) [Bind mounts](#) [Exec](#) [Files](#) [Stats](#)

2024-12-05 16:53:26 6 high severity vulnerabilities  
2024-12-05 16:53:26  
2024-12-05 16:53:26 To address issues that do not require attention, run:  
2024-12-05 16:53:26 npm audit fix  
2024-12-05 16:53:26  
2024-12-05 16:53:26 Some issues need review, and may require choosing  
2024-12-05 16:53:26 a different dependency.  
2024-12-05 16:53:26  
2024-12-05 16:53:26 Run `npm audit` for details.  
2024-12-05 16:53:26  
2024-12-05 16:53:26 > event-management-api@1.0.0 start  
2024-12-05 16:53:26 > node index.js  
2024-12-05 16:53:26  
2024-12-05 16:53:26 Mock mode: disabled  
2024-12-05 16:53:26 Your server is listening on port 3000 (<http://localhost:3000>)  
2024-12-05 16:53:26 Swagger-ui is available on <http://localhost:3000/docs>  
2024-12-05 16:53:26 Database connection active. REDACTED  
2024-12-05 16:53:46 GET /docs/ 304 3.602 ms --  
2024-12-05 16:53:46 GET /docs/swagger-ui-bundle.js 304 1.812 ms --  
2024-12-05 16:53:46 GET /docs/swagger-ui.css 304 1.646 ms --  
2024-12-05 16:53:46 GET /docs/swagger-ui-standalone-preset.js 304 0.591 ms --  
2024-12-05 16:53:46 HEAD /docs/ 304 0.457 ms --  
2024-12-05 16:53:46 GET /api-docs 200 0.426 ms --  
2024-12-05 16:53:46 GET /api-docs 200 0.267 ms --  
2024-12-05 16:53:55 GET /organizers 200 32.906 ms --  
2024-12-05 16:53:26 Thu, 05 Dec 2024 19:53:26 GMT body-parser deprecated undefined extend

# Obrigado !

---

**Beatriz Almeida**

**(A044416)**

**Carolina Fernandes**

**(A044897)**

**Pedro Venda**

**(A045464)**