



**COORDENADORIA DE ENGENHARIA DA
COMPUTAÇÃO**

**ALEX SANDER GARCIA FELICIO JUNIOR - RA 180016
FELIPE MARTINS DE ALMEIDA – RA 171703
MARCOS VALÉRIO BUENO FILHO – RA 180565
VINICIUS GONZAGA BONATTI – RA 180184**

Marcos Fábio Jardimi

**Turma CP406TIN2
PROGRAMAÇÃO DISTRIBUÍDA**

Contratos e IDL

**SOROCABA,
SP
19/10/2023**

INTRODUÇÃO

Web services e web apps são componentes fundamentais na arquitetura da web moderna, mas servem a propósitos distintos dentro dessa arquitetura. Um web service é uma forma padronizada de comunicação entre diferentes sistemas de software sobre a Internet. Utiliza uma linguagem comum, geralmente baseada em XML ou JSON, para permitir a interoperabilidade entre aplicações construídas em diferentes plataformas de programação. A comunicação é feita através de protocolos web como HTTP ou HTTPS, e os *webs services* são frequentemente implementados utilizando tecnologias como SOAP (*Simple Object Access Protocol*), REST (*Representational State Transfer*) ou GraphQL (M. P. Papazoglou, *Web Services: Principles and Technology*, 2008).

Por outro lado, uma web app é uma aplicação que é acessada via navegador de internet e interage com o usuário final. Web apps são construídas usando tecnologias de *front-end* como HTML, CSS e JavaScript, que são interpretadas pelo navegador, e podem também comunicar-se com *web services* para realizar operações como consultar ou atualizar dados. O desenvolvimento de web apps muitas vezes segue o modelo cliente-servidor, onde o cliente (navegador) interage com um servidor de aplicação que pode utilizar *web services* como parte de sua lógica de negócio (L. Rosenfeld, P. Morville, *Information Architecture for the World Wide Web*, 2006).

Ambos, *web services* e web apps, são essenciais para a funcionalidade da Internet como a conhecemos, possibilitando desde transações de e-commerce até redes sociais e aplicações empresariais.

APRESENTAÇÃO

O *WebApp* selecionado para desenvolvimento é uma plataforma dedicada ao gerenciamento de eventos, que facilita o processo de organização, promoção e venda de ingressos para eventos diversos. Este aplicativo funciona como um intermediário entre organizadores de eventos e participantes potenciais, oferecendo um ponto centralizado para o acesso a informações atualizadas sobre eventos, locais de realização e disponibilidade de ingressos.

Os principais objetos manipulados pelo *WebApp* incluem:

Eventos: São as entidades centrais do aplicativo e incluem informações como título, descrição, data, hora e detalhes do local onde ocorrerão.

Locais: Correspondem aos espaços físicos onde os eventos são realizados, contendo dados como nome, endereço, capacidade e informações para contato.

Ingressos: Representam a admissão aos eventos, com detalhes como preço, quantidade disponível e o evento específico ao qual estão associados.

Dentro da aplicação, os usuários podem realizar diversas operações relacionadas a estes objetos:

Para **Eventos:**

- Listagem de todos os eventos (GET)
- Criação de um novo evento (POST)
- Atualização de um evento existente (PUT)
- Exclusão de um evento (DELETE)

Para **Locais:**

- Listagem de todos os locais (GET)
- Adição de um novo local (POST)
- Modificação de um local existente (PUT)
- Remoção de um local (DELETE)

Para **Ingressos:**

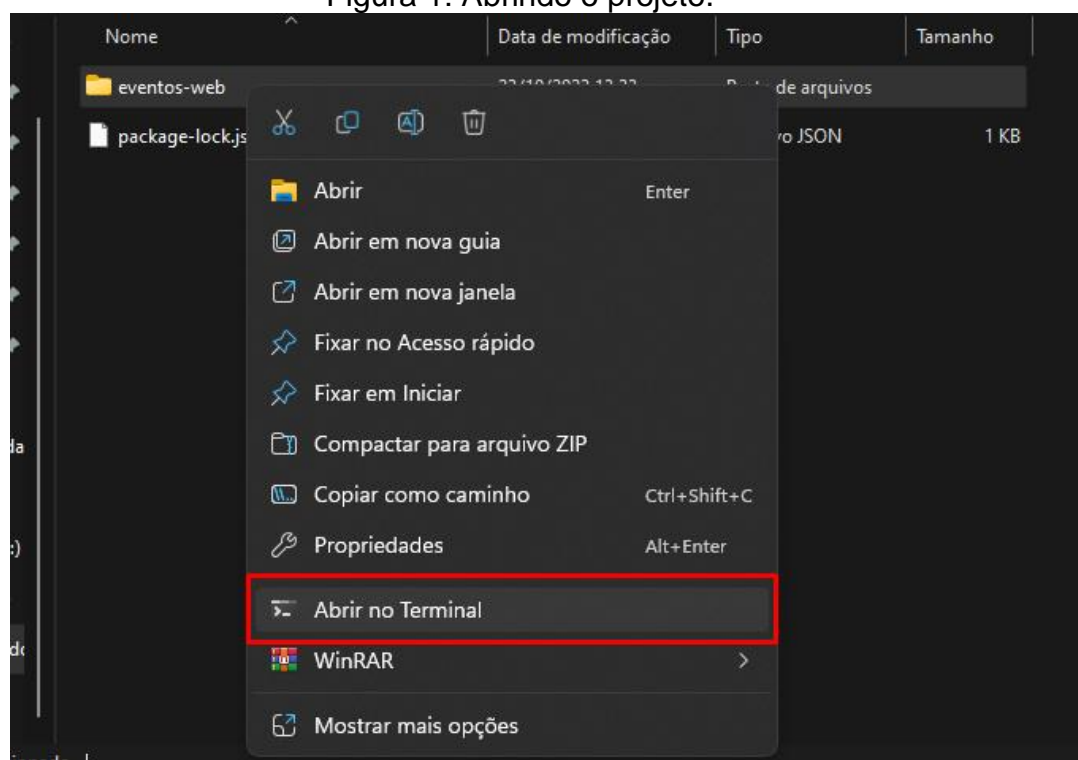
- Listagem de todos os ingressos disponíveis (GET)
- Emissão de um novo lote de ingressos (POST)
- Atualização da quantidade ou informações de ingressos existentes (PUT)
- Cancelamento de um lote de ingressos (DELETE)

A plataforma é construída utilizando a linguagem de programação JavaScript no ambiente do Node.js (Versão 18.17.1), com o auxílio do framework Express para o

manejo das rotas e da funcionalidade da API. A persistência de dados é garantida pelo uso do sistema de banco de dados MongoDB. A API implementada segue os princípios RESTful, garantindo uma comunicação eficiente e padronizada através de métodos HTTP. A documentação e a interface de interação com a API são providenciadas pelo *Swagger*, permitindo uma fácil compreensão e teste dos endpoints.

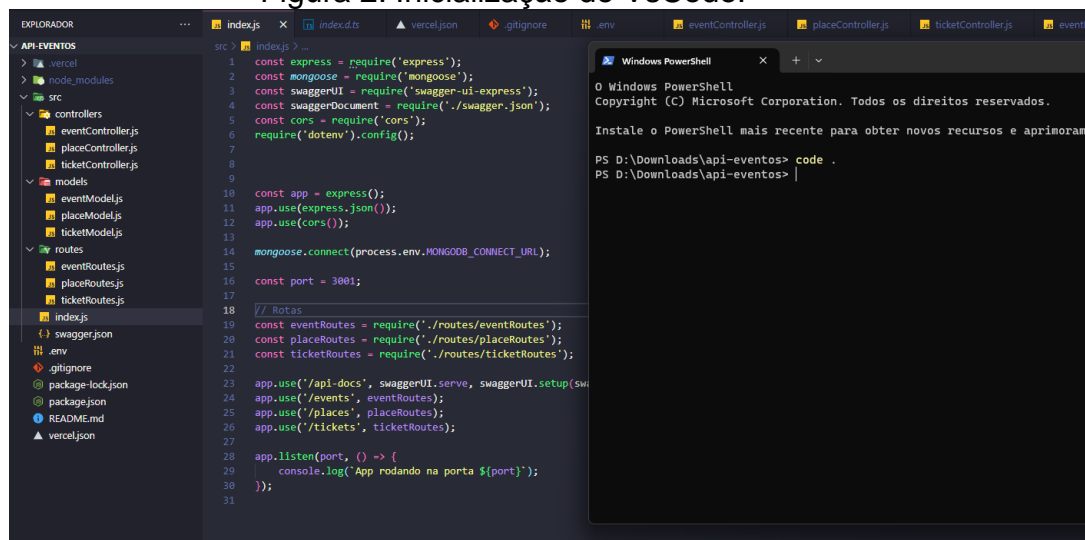
Para ter acesso a API basta ir na pasta do projeto clicar com o botão direito, seleccionar a opção de abrir terminal e dentro do terminal digitar o comando “code .” para ser aberto o visual studio code como mostra a figura a seguir:

Figura 1: Abrindo o projeto.



Fonte: Autoria Própria.

Figura 2: Inicialização do VsCode.



Fonte: Autoria Própria.

Para iniciar a api é necessário abrir o terminal do Visual Studio Code, ir na aba de terminal, escolher a opção de novo terminal e em seguida escrever “*npm install*” e teclar a tecla enter para ser instalado as dependências necessárias para rodar a API.

Figura 3: Abrindo terminal do VsCode.

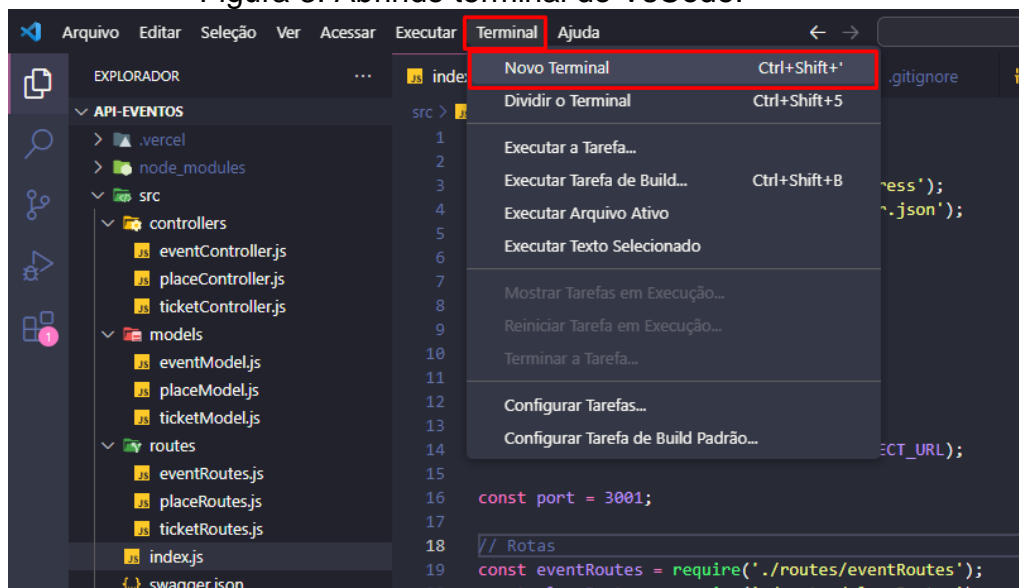
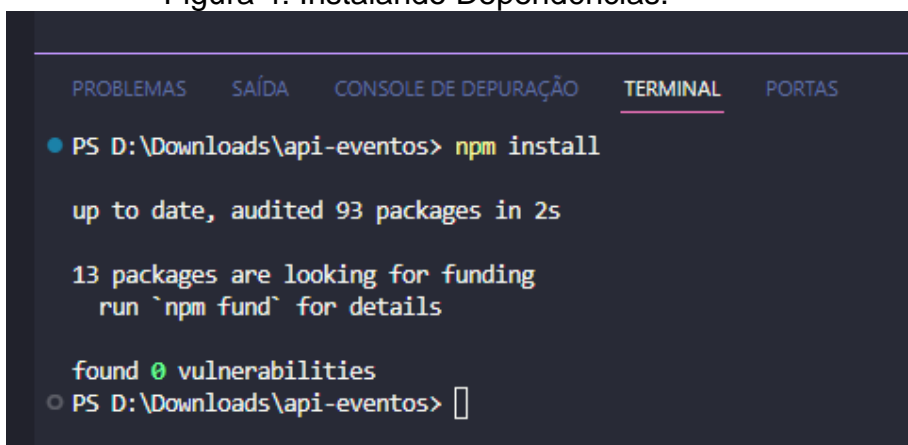


Figura 4: Instalando Dependências.



```
PROBLEMAS  SAÍDA  CONSOLE DE DEPURAÇÃO  TERMINAL  PORTAS

● PS D:\Downloads\api-eventos> npm install

up to date, audited 93 packages in 2s

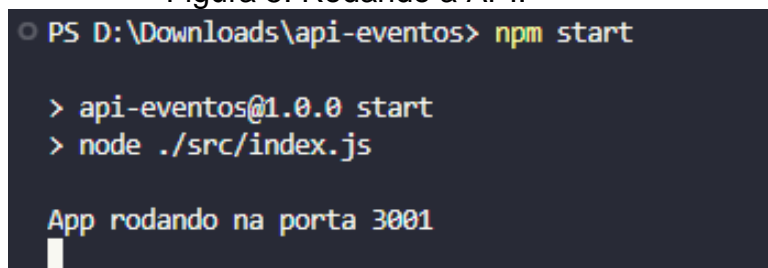
13 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
○ PS D:\Downloads\api-eventos> 
```

Fonte: Autoria Própria.

Feito os passos anteriores basta digitar “*npm start*” para iniciar a api, vale lembrar que por padrão a porta vem na 3001, caso esteja rodando alguma aplicação nessa porta é necessário alterar a porta da API que está no arquivo index.js conforme mostra a Figura 6.

Figura 5: Rodando a API.



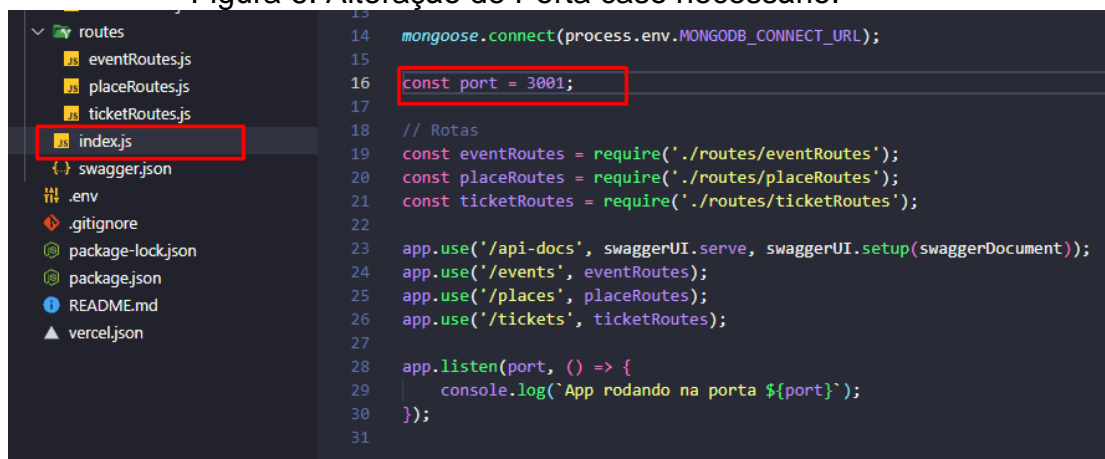
```
○ PS D:\Downloads\api-eventos> npm start

> api-eventos@1.0.0 start
> node ./src/index.js

App rodando na porta 3001
|
```

Fonte: Autoria Própria.

Figura 6: Alteração de Porta caso necessário.



```
13
14 mongoose.connect(process.env.MONGODB_CONNECT_URL);
15
16 const port = 3001;
17
18 // Rotas
19 const eventRoutes = require('./routes/eventRoutes');
20 const placeRoutes = require('./routes/placeRoutes');
21 const ticketRoutes = require('./routes/ticketRoutes');
22
23 app.use('/api-docs', swaggerUI.serve, swaggerUI.setup(swaggerDocument));
24 app.use('/events', eventRoutes);
25 app.use('/places', placeRoutes);
26 app.use('/tickets', ticketRoutes);
27
28 app.listen(port, () => {
29   console.log(`App rodando na porta ${port}`);
30 });
31
```

Fonte: Autoria Própria.

Após a API estar rodando é possível acessar ela através do endereço `http://localhost:3001`. As rotas possível da API são:

Para **eventos**:

`http://localhost:3001/events`

Para **Lugares**:

`http://localhost:3001/places`

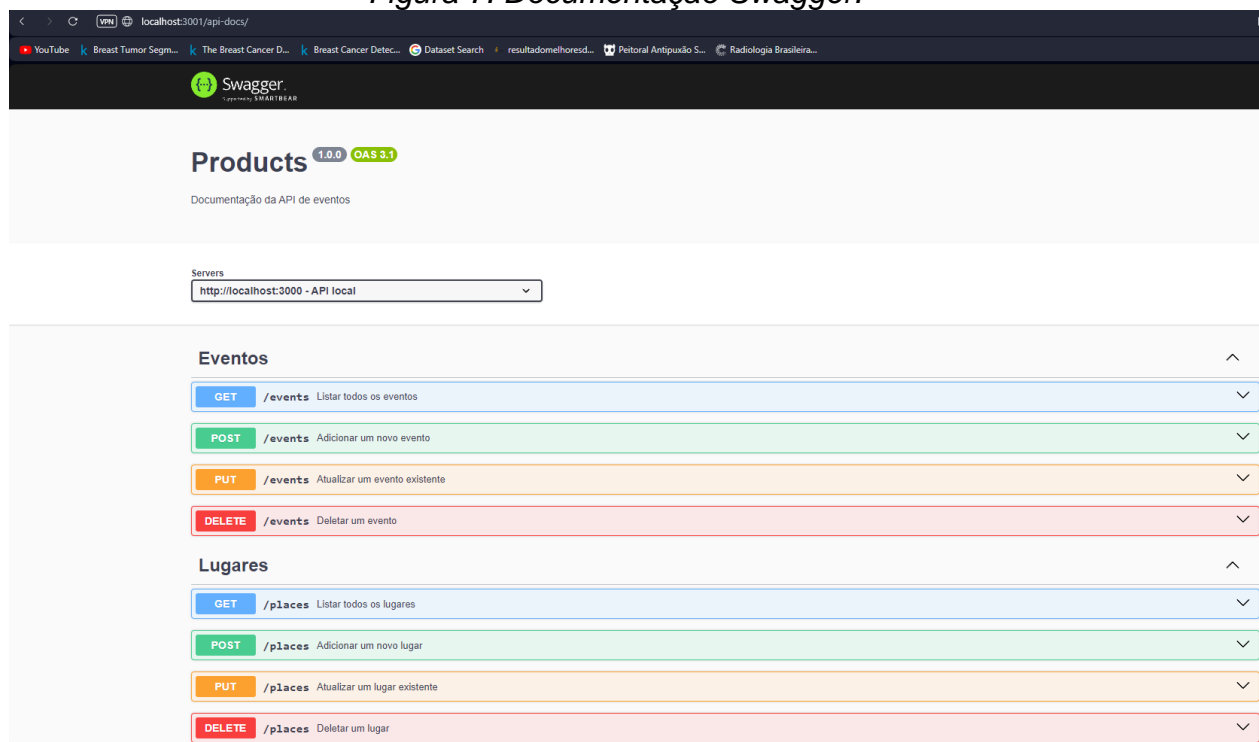
Para **Ingressos**:

`http://localhost:3001/tickets`

Caso haja dúvidas referente a rota é possível acessar a documentação delas pelo

`http://localhost:3001/api-docs`

Figura 7: Documentação Swagger.



Fonte: Autoria Própria.

Implantação da API na Vercel

Para garantir a disponibilidade e o acesso contínuo à API de gerenciamento de eventos, procedeu-se com a implantação da mesma na Vercel, uma plataforma cloud que oferece soluções eficientes para a hospedagem de aplicações web. A Vercel é conhecida por sua capacidade de escalonamento automático, performance otimizada e facilidade no processo de deploy.

Através desta plataforma, a API foi configurada para execução em um ambiente de produção, o que dispensa a necessidade de configuração e execução local do serviço.

Após a implantação na Vercel, a API está agora disponível online e pode ser acessada diretamente através dos seguintes endpoints:

Para visualizar e gerenciar **eventos**:

<https://api-eventos-allexfelicio.vercel.app/events>

Para consultas e operações relacionadas a **locais**:

<https://api-eventos-allexfelicio.vercel.app/places>

Para gerenciamento de **ingressos**:

<https://api-eventos-allexfelicio.vercel.app/tickets>

Os links acima levam diretamente aos recursos da API, facilitando testes rápidos, integração com outras aplicações e demonstrações em tempo real da funcionalidade do sistema.