API EVENTOS - NodeJs

Objetivo

Desarrollar una API para explorar los distintos eventos a futuro, la cual permitirá conocer y modificar los eventos que lo componen. También deberá permitir inscribirme en un evento y listar todos los participantes del mismo. Por otro lado, deberá exponer la información para que cualquier frontend pueda consumirla.

👉 Utilizar **Node Js** y **Express**.
👉 No es necesario armar el Frontend.
👉 Las rutas deberán seguir el patrón REST.
👉 Utilizar la librería **pg** para la conexión a base de datos y **dotenv** para almacenar la configuración.

Historial de Cambios

v1.0 a v1.9 (30/03/2024).

- Release inicial del trabajo práctico
- En el punto 5 se aclara el ejemplo de cómo funciona el rating.
- Se agrega el punto 10, rating de un evento.
- Se agregó en el enunciado que se espera la utilización de la biblioteca dotenv.
- Formateado del documento especialmente el anexo y respuestas de JSON.
- Se documentó el endpoint de /api/provinces.
- Se documentaron varios endpoints y se documentaron que deben retornar en cada caso.
- Se agregó la des-registración de un usuario a un evento, con alguna validación.
- Se agregaron los endpoints de location.
- Se documentó el endpoint de Ranking
- Se documentó el endpoint de Event-Location
- Se corrigió la documentación en “Ubicaciones de Eventos”
- Se cambia el endpoint del punto 11 de /event-location a /location.
- Se agregó el endpoint /api/province/**{id}**/locations al province controller.
- Se agregó al punto 11 el endpoint /api/location/**{id}**/event-location
- Se arregla el punto 13.
- Se arreglan problemas de formato en el documento.

v2.0 (13/05/2024).

- Documentación de varios endpoints y sus validaciones.
- Se documentaron los endpoint que necesitan autenticación.
- Se documentaron los siguientes endpoints:
  - /api/user/login
  - /api/user/register

v2.1 (14/05/2024).

- Documentación de punto 8 (Creación, Edición, Eliminación de Eventos (CRUD))

v2.2 (15/05/2024).

- Documentación de punto 10 (Rating de un Evento)

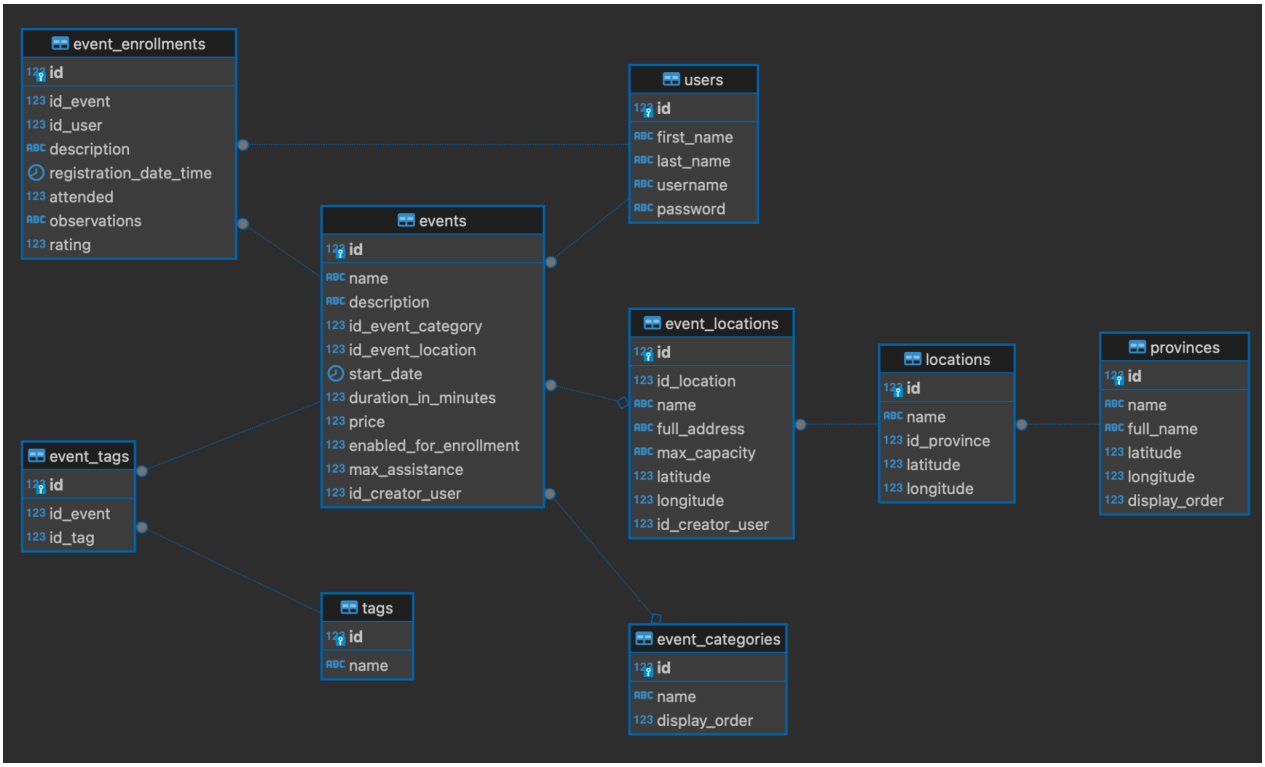
v2.3 (27/05/2024).

- Se corrigieron errores en JSONs de resultados.
- Se agrego la sección final “Para Pensar”.

Requerimientos técnicos

1. Modelado de Base de Datos.

Para esta primera etapa no van a necesitar crear ustedes la base de datos, por lo que se adjunta un DER de la solución para que puedan mockear lo que requieran.



2. Listado de Eventos

Se recuerda que este endpoint debe estar paginado dado a la alta complejidad que se tiene para obtener cada uno de los eventos. Se adjunta un ejemplo de la respuesta que debería devolver el endpoint al final del documento.

El listado deberá mostrar:

- Id
- Nombre.
- Descripcion.
- Fecha del Evento.
- Duración del Evento.
- Precio de la entrada.
- Si está habilitado para inscribirse.
- La capacidad del evento
- El usuario creador del evento (objeto).
- Categoría (objeto).
- Ubicación (objeto).

El endpoint deberá ser:

GET /api/event/

3. Búsqueda de un Evento

Deberá permitir buscar por nombre, categoría, fecha de inicio o un tag determinado.
Para especificar el término de búsqueda o filtros se deberán enviar como parámetros de query.
Se puede filtrar por uno solo o cualquier combinación de los parámetros.
En el anexo (Listado de Eventos) se puede ver un ejemplo del resultado de la búsqueda.

GET /api/event/?name=**{texto}**
GET /api/event/?category=**{texto}**
GET /api/event/?startdate=**{fecha YYYY-MM-DD}**
GET /api/event/?tag=**{texto}**

Ejemplo:
GET /api/event/?name=**taylor**&category=**Musica**&startdate=**2024-03-03**&tag=**Rock**

4. Detalle de un Evento

En el detalle deberán listarse todos los atributos del Evento, como así también su ubicación completa, esto incluye el usuario creador, los tags, el lugar del evento, la localidad y la provincia donde se encuentra la localización.

Ejemplo:

GET /api/event/**{id}**

Retorna un status code **200** (ok) y los datos.

Retorna un status code **404** (not found) en caso de que el **id** sea inexistente.

Ejemplo de respuesta:


```
{
    "id": 8,
    "name": "Toto",
    "description": "La legendaria banda estadounidense se presentará en Buenos Aires.",
    "id_event_category": 3,
    "id_event_location": 2,
    "start_date": "2024-11-22T03:00:00.000Z",
    "duration_in_minutes": 120,
    "price": "150000",
    "enabled_for_enrollment": "1",
    "max_assistance": 12000,
    "id_creator_user": 1,
    "event_location": {
        "id": 2,
        "id_location": 3397,
        "name": "Movistar Arena",
        "full_address": "Humboldt 450, C1414 Cdad. Autónoma de Buenos Aires",
        "max_capacity": "15000",
        "latitude": "-34.593488697344405",
        "longitude": "-58.44735886932156",
        "id_creator_user": 1,
        "location": {
            "id": 3397,
            "name": "Villa Crespo",
            "id_province": 2,
            "latitude": "-34.599876403808594",
            "longitude": "-58.438816070556641",
            "province": {
                "id": 2,
                "name": "Ciudad Autónoma de Buenos Aires",
                "full_name": "Ciudad Autónoma de Buenos Aires",
                "latitude": "-34.61444091796875",
                "longitude": "-58.445877075195312",
                "display_order": null
            }
        },
        "creator_user": {
            "id": 1,
            "first_name": "Pablo",
            "last_name": "Ulman",
            "username": "pablo.ulman@ort.edu.ar",
            "password": "******"
        }
    },
    "tags": [
        {
            "id": 1,
            "name": "rock"
        },
        {
            "id": 2,
            "name": "pop"
        }
    ],
    "creator_user": {
        "id": 1,
        "first_name": "Pablo",
        "last_name": "Ulman",
        "username": "pablo.ulman@ort.edu.ar",
        "password": "******"
    },
    "event_category": {
        "id": 3,
        "name": "Recitales",
        "display_order": 70
    }
}
```

5. Listado de Participantes

En el listado, deberá mostrarse la lista de usuarios inscriptos para un cierto evento. Este endpoint me deberá permitir filtrar por nombre, apellido, nombre de usuario, si asistió al evento y por rating mayor a un determinado valor.

Se puede filtrar por ninguno, uno solo o cualquier combinación de los parámetros.

En el anexo puede encontrar un ejemplo de la respuesta.

GET /api/event/**{id}**/enrollment?first_name=**{texto}**

GET /api/event/**{id}**/enrollment?last_name=**{texto}**

GET /api/event/**{id}**/enrollment?username=**{texto}**

GET /api/event/**{id}**/enrollment?attended=**{boolean}**

GET /api/event/**{id}**/enrollment?rating=**{entero}**

Ejemplo:

GET /api/event/**{id}**/enrollment?username=**gomez**&attended=**1**

GET /api/event/**{id}**/enrollment?attended=**0**


```
{
    "collection": [
        {
            "id": 73,
            "id_event": 6,
            "id_user": 1,
            "user": {
                "id": 1,
                "first_name": "Willy",
                "last_name": "Wonka",
                "username": "guillermito.wonka@chocolat.com",
                "password": "******"
            },
            "description": null,
            "registration_date_time": "2024-05-16T16:34:36.619Z",
            "attended": null,
            "observations": null,
            "rating": null
        },
        {
            "id": 279,
            "id_event": 6,
            "id_user": 66,
            "user": {
                "id": 66,
                "first_name": "Frederick",
                "last_name": "Krueger",
                "username": "freddy.krueger@nightmare.com",
                "password": "*******"
            },
            "description": null,
            "registration_date_time": "2024-05-17T19:14:46.015Z",
            "attended": null,
            "observations": null,
            "rating": null
        },
        {...}
    ],
    "pagination": {
        "limit": 0,
        "offset": 0,
        "nextPage": "",
        "total": 0
    }
}
```

6. Autenticación de Usuarios

Para realizar peticiones a los endpoints subsiguientes el usuario deberá contar con un token que obtendrá al autenticarse. Para ello, se debe desarrollar el endpoint de registración y luego el de login que permita obtener el token.
Vamos a utilizar JWT (JSON Web Token) para la autenticación.

El endpoint encargados de la autenticación y sus respectivos bodys deberán ser:

POST /api/user/login

Se envía la entidad en el body de request.

```
{
    "username": "...",
    "password": "..."
}
```

Retorna un status code **200** (ok) y los datos.

```
{
   "success": true,
   "message": "",
   "token"  : "eyJhbGciOiJIUzI1NCI6IkpXVCJ9.eyJpZCI6MSwiZmlyc3RfbmFtZSI6I...."
}
```

Retorna un status code **400** (bad request) en caso de que el email (**username**) sea sintácticamente inválido.

```
{
   "success": false,
   "message": "El email es invalido.",
   "token"  : ""
}
```

Retorna un status code **401** (unauthorized) en caso de que el usuario sea inexistente.

```
{
   "success": false,
   "message": "Usuario o clave inválida.",
   "token"  : ""
}
```

POST /api/user/register

Se envía la entidad en el body de request.

```
{
    "first_name": "...",
    "last_name" : "...",
    "username"  : "...",
    "password"  : "..."
}
```

Retorna un status code **201** (created).

Retorna un status code **400** (bad request) y un mensaje de error en los siguientes casos:

- Los campos **first_name** o **last_name** están vacíos o tienen menos de tres (3) letras.
- El email (**username**) es sintácticamente inválido.
  Nota: Es conveniente utilizar expresiones regulares.
- El campo **password** está vacío o tiene menos de tres (3) letras.

7. Creación, Edición, Eliminación y Listado de Provincias (CRUD)

Deberán existir las operaciones básicas de creación, edición, eliminación, obtener todos (paginado) y obtener por id de provincias. Además se desea poder obtener todas las localidades de una provincia (paginado)

Ejemplo

GET /api/province

Retorna todas las **provinces**.

Retorna un status code **200** (ok) y un array con los datos.

GET /api/province/**{id}**

Retorna la **province** cuyo id es enviado por parámetro.

Retorna un status code **200** (ok) y un objeto con los datos.

Retorna un status code **404** (not found) en caso de que el **id** sea inexistente.


```
{
    "id": 6,
    "name": "Buenos Aires",
    "full_name": "Provincia de Buenos Aires",
    "latitude": "-36.677391052246094",
    "longitude": "-60.558475494384766",
    "display_order": 30
}
```


GET /api/province/**{id}**/locations

Retorna todas las **locations** de la province **id** enviada por parámetro.

Retorna un status code **200** (ok) y los datos.

Retorna un status code **404** (not found) en caso de que el **id** sea inexistente.


```
{
   "collection": [
      {
          "id": 1,
          "name": "Ciudad de Buenos Aires",
          "id_province": 2,
          "latitude": "-34.608417510986328",
          "longitude": "-58.372135162353516"
      },
      {
          "id": 3351,
          "name": "Constitución",
          "id_province": 2,
          "latitude": "-34.626823425292969",
          "longitude": "-58.387187957763672"
      },
      {...},
      {
          "id": 3398,
          "name": "Villa Ortúzar",
          "id_province": 2,
          "latitude": "-34.578670501708984",
          "longitude": "-58.469482421875"
      }
   ],
   "pagination": {
      "limit": 0,
      "offset": 0,
      "nextPage": "",
      "total": 0
   }
}
```

POST /api/province/

Inserta una **province** que es enviada en el body de request (no lleva id).
Retorna un status code **201** (created).
Retorna un status code **400** (bad request) y un mensaje de error en los siguientes casos:

- El campo **name** está vacío o tiene menos de tres (3) letras.
- Los campos **latitude** y **longitude** no son números.

PUT /api/province/

Actualiza una province que es enviado en el body y retorna un status code **200** (ok).
Retorna un status code **400** (bad request) y un mensaje de error en los siguientes casos:

- El campo **name** está vacío o tiene menos de tres (3) letras.
- Los campos **latitude** y **longitude** no son números.

Retorna un status code **404** (not found) en caso de que el **id** sea inexistente.

DELETE/api/province/**{id}**

Elimina la **province** cuyo id es enviado por parámetro.
Retorna un status code **200** (ok) y los datos.
Retorna un status code **404** (not found) en caso de que el **id** sea inexistente.

8. Creación, Edición, Eliminación de Eventos (CRUD)

Se deberá completar el controller para permitir crear eventos nuevos completando todos los campos y además al usuario creador de cada evento se le debe permitir editar y eliminar sus eventos. El usuario autenticado, no podrá actualizar o eliminar eventos que no sean suyos.

POST /api/event/ **(necesita autenticación)**

Inserta un **evento** que es enviado en el body de request (no lleva id).
Retorna un status code **201** (created).
Retorna un status code **400** (bad request) y un mensaje de error en los siguientes casos:

- El **name** o **description** están vacíos o tienen menos de tres (3) letras.
- El **max_assistance** es mayor que el **max_capacity** del **id_event_location**.
- El **price** o **duration_in_minutes** son menores que cero.

Retorna un status code **401** (Unauthorized) y un mensaje de error en caso de que el usuario no se encuentre autenticado.

PUT /api/event/ **(necesita autenticación)**

Actualiza un **evento** que es enviado en el body y retorna un status code **200** (ok).
Retorna un status code **400** (bad request) y el texto del error, en caso de existir algún error en las reglas de negocio (ver el **POST**).
Retorna un status code **401** (Unauthorized) y un mensaje de error en caso de que el usuario no se encuentre autenticado.
Retorna un status code **404** (not found) y un mensaje de error en caso de que el id del evento no exista, o que el evento no pertenezca al usuario autenticado.

DELETE/api/event/**{id}** **(necesita autenticación)**

Elimina un **event** cuyo id es enviado por parámetro.
Retorna un status code **200** (ok) y los datos.
Retorna un status code **400** (bad request) y un mensaje de error en los siguientes casos:

- Existe al menos un usuario registrado al evento.

Retorna un status code **401** (Unauthorized) y un mensaje de error en caso de que el usuario no se encuentre autenticado.
Retorna un status code **404** (not found) y un mensaje de error en caso de que el id del evento no exista, o que el evento no pertenezca al usuario autenticado.

9. Inscripción a un Evento

Permite la registración de un usuario en un evento, y la eliminación de la suscripción de un usuario a un evento. El usuario que se registra, es el usuario que se autenticó previamente (mediante el login) y tiene un token válido. Es decir que los datos del usuario deben viajar en el header. La fecha y hora de registración (registration_date_time) es la de la computadora.

Ejemplo:

POST /api/event/**{id}**/enrollment/ **(necesita autenticación)**

Registra al **usuario** (autenticado) al evento enviado por parámetro.
Retorna un status code **201** (created), sí se pudo registrar.
Retorna un status code **400** (bad request) y un mensaje de error en los siguientes casos:

- Exceda la capacidad máxima de registrados (**max_assistance**) al evento.
- Intenta registrarse a un evento que ya sucedió (**start_date**), o la fecha del evento es hoy.
- Intenta registrarse a un evento que no está habilitado para la inscripción (**enabled_for_enrollment**).
- El usuario ya se encuentra registrado en el evento.

Retorna un status code **401** (Unauthorized) y un mensaje de error en caso de que el usuario no se encuentre autenticado.
Retorna un status code **404** (not found) en caso de que el **id** sea inexistente.

DELETE/api/event/**{id}**/enrollment/ **(necesita autenticación)**

Remueve al **usuario** (autenticado) del evento enviado por parámetro.
Retorna un status code **200** (ok), sí se pudo remover de la suscripción.
Retorna un status code **400** (bad request) y un mensaje de error en los siguientes casos:

- El usuario no se encuentra registrado al evento.
- Intenta removerse de un evento que ya sucedió (**start_date**), o la fecha del evento es hoy.

Retorna un status code **401** (Unauthorized) y un mensaje de error en caso de que el usuario no se encuentre autenticado.
Retorna un status code **404** (not found) en caso de que el **id** sea inexistente.

10. Rating de un Evento

Luego de que un evento finalice, se le deberá permitir a todos los usuarios que asistieron emitir un rating para dicho evento. El rating consiste en un número entre 1 y 10. Además, deberán poder enviar un feedback para que en los próximos eventos sepan que opinan otros usuarios.

PATCH /api/event/**{id}**/enrollment/**{entero}** **(necesita autenticación)**

Actualiza un **event_enrollment** que es enviado por parámetro con un valor que es enviado por parámetro, para el usuario autenticado. El feedback (**observations**) debe enviarse en el cuerpo del mensaje (puede estar vacío).

Retorna un status code **200** (ok), sí se pudo rankear al evento.

Retorna un status code **400** (bad request) y un mensaje de error en los siguientes casos:

- El usuario no se encuentra registrado al evento.
- El evento no ha finalizado aún, la fecha (**start_date**) tiene que ser mayor a hoy.
- Los valores del rating (**rating**), no se encuentran entre 1 y 10 (inclusives).

Retorna un status code **401** (Unauthorized) y un mensaje de error en caso de que el usuario no se encuentre autenticado.

Retorna un status code **404** (not found) en caso de que el **id** sea inexistente.

Ejemplo:

PATCH /api/event/**6**/enrollment/**9**

Se envía la entidad en el body de request.

```
{
    "observations": "Fue increíble!!. Conviene estacionar lejos!!!!"
}
```

11. Locations

Deberán existir las operaciones para poder obtener una localidad, todas las localidades (paginado). Además, se desea poder obtener todos los event-location de una location (paginado).

Ejemplo

GET /api/location

Retorna todas las **locations**.

Retorna un status code **200** (ok) y los datos.

GET /api/location/**{id}**

Retorna la **location** cuyo id es enviado por parámetro.

Retorna un status code **200** (ok) y los datos.

Retorna un status code **404** (not found) en caso de que el id de la location no exista.

GET /api/location/**{id}**/event-location **(necesita autenticación)**

Retorna todos los **event-location** del location **id** enviado por parámetro, del usuario autenticado.

Retorna un status code **200** (ok) y los datos.

Retorna un status code **401** (Unauthorized) y un mensaje de error en caso de que el usuario no se encuentre autenticado.

Retorna un status code **404** (not found) en caso de que el id de la location sea inexistente.

12. Categorías

Deberán existir las operaciones básicas de creación, edición, eliminación, obtener todos (paginado).

Ejemplo

GET /api/event-category

Retorna todas las **event_categories**.

Retorna un status code **200** (ok) y un array con los datos.

GET /api/event-category/**{id}**

Retorna la **event_category** cuyo id es enviado por parámetro.

Retorna un status code **200** (ok) y un objeto.

Retorna un status code **404** (not found) en caso de que el **id** sea inexistente.

POST /api/event-category/

Inserta una **event_category** que es enviada en el body de request (no lleva id).

Retorna un status code **201** (created).

Retorna un status code **400** (bad request) y un mensaje de error en los siguientes casos:

- El nombre (**name**) está vacío o tiene menos de tres (3) letras.

PUT /api/event-category/

Actualiza un **event_category** que es enviado en el body y retorna un status code **200** (ok).

Retorna un status code **400** (bad request) y un mensaje de error en los siguientes casos:

- El nombre (**name**) está vacío o tiene menos de tres (3) letras.

Retorna un status code **404** (not found) en caso de que el **id** sea inexistente.

DELETE/api/event-category/**{id}**

Elimina la **event_category** cuyo id es enviado por parámetro.

Retorna un status code **200** (ok) y los datos.

Retorna un status code **404** (not found) en caso de que el **id** sea inexistente.

13. Ubicaciones de Eventos

Deberán existir las operaciones para poder insertar, actualizar eliminar, obtener un event-location, todos los event-location (paginado).

Ejemplo

GET /api/event-location **(necesita autenticación)**

Retorna todas las event_locations del usuario autenticado.
Retorna un status code **200** (ok) y los datos.
Retorna un status code **401** (Unauthorized) y un mensaje de error en caso de que el usuario no se encuentre autenticado.

GET /api/event-location/**{id}** **(necesita autenticación)**

Retorna la **event_location** cuyo **id** es enviado por parámetro siempre y cuando sea del usuario autenticado.
Retorna un status code **200** (ok) y los datos.
Retorna un status code **401** (Unauthorized) y un mensaje de error en caso de que el usuario no se encuentre autenticado.
Retorna un status code **404** (not found) en caso de que el id de la event-location no exista, o no sea del usuario autenticado (**id_creator_user**).

POST /api/event-location/ **(necesita autenticación)**

Inserta un **event_location** que es enviado en el body de request (no lleva id). El **id_creator_user** es el usuario autenticado.
Retorna un status code **201** (created).
Retorna un status code **400** (bad request) y el texto del error, en caso de existir algún error en las reglas de negocio por ejemplo:

- El nombre (**name)** o la dirección (**full_address**) están vacíos o tienen menos de tres (3) letras.
- El **id_location** es inexistente.
- El **max_capacity** es el número 0 (cero) o negativo.

Retorna un status code **401** (Unauthorized) y un mensaje de error en caso de que el usuario no se encuentre autenticado.

PUT /api/event-location/ **(necesita autenticación)**

Actualiza un **event_location** que es enviado en el body y retorna un status code **200** (ok).
Retorna un status code **400** (bad request) y el texto del error para las mismas validaciones que el **POST**.
Retorna un status code **401** (Unauthorized) y un mensaje de error en caso de que el usuario no se encuentre autenticado.
Retorna un status code **404** (Not Found) y un mensaje de error en caso de que el **id** del **event_location** sea inexistente o no pertenezca al usuario autenticado.

DELETE/api/event-location/**{id}** **(necesita autenticación)**

Elimina el **event-location** cuyo id es enviado por parámetro.

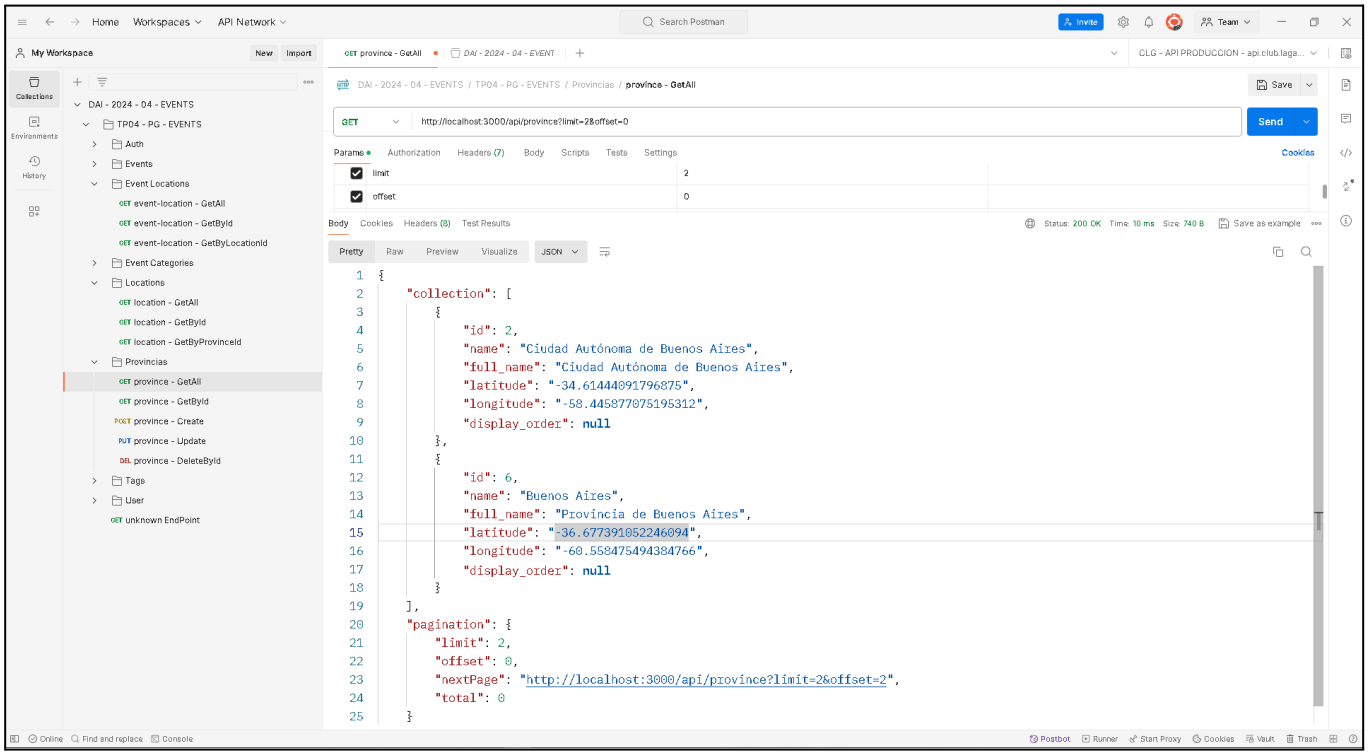
Retorna un status code **200** (ok) y los datos.

Retorna un status code **401** (Unauthorized) y un mensaje de error en caso de que el usuario no se encuentre autenticado.

Retorna un status code **404** (Not Found) y un mensaje de error en caso de que el **id** del **event_location** sea inexistente o no pertenezca al usuario autenticado.

Documentación

Es deseable documentar los endpoints utilizando alguna herramienta como Postman o Swagger.
Y tener una colección **completa** con todos los endPoints.



Anexo Respuesta de los endpoints

Listado de Eventos:


```
{
    "collection": [
        {
            "id": 2,
            "name": "Taylor Swift",
            "description": "Un alto show",
            "event_category": {
                "id": 1,
                "name": "Musica"
            },
            "event_location": {
                "id": 1,
                "name": "Club Atlético River Plate",
                "full_address": "Av. Pres. Figueroa Alcorta 7597",
                "latitude": -34.54454505693356,
                "longitude": -58.4494761175694,
                "max_capacity": "84567",
                "location": {
                    "id": 3391,
                    "name": "Nuñez",
                    "latitude": -34.548805236816406,
                    "longitude": -58.463230133056641,
                    "max_capacity": "84567",
                    "province": {
                        "id": 1,
                        "name": "Ciudad Autónoma de Buenos Aires",
                        "full_name": "Ciudad Autónoma de Buenos Aires",
                        "latitude": -34.61444091796875,
                        "longitude": -58.445877075195312,
                        "display_order": null
                    }
                }
            },
            "start_date": "2024-03-21T03:00:00.000Z",
            "duration_in_minutes": 210,
            "price": "15500",
            "enabled_for_enrollment": true,
            "max_assistance": 120000,
            "creator_user": {
                "id": 3,
                "username": "Jschiffer",
                "first_name": "Julian",
                "last_name": "Schiffer"
            },
            "tags": [
                {
                    "id": 1,
                    "name": "Rock"
                },
                {
                    "id": 2,
                    "name": "Pop"
                }
           ]
        },
        {...}
    ],
    "pagination": {
        "limit": 15,
        "offset": 0,
        "nextPage": null,
        "total": "2"
    }
}
```

Listado de Participantes


```
{
    "collection": [
        {
            "user": {
                "id": 3,
                "username": "Jschiffer",
                "first_name": "Julian",
                "last_name": "Schiffer"
             },
            "attended": false,
            "rating": null,
            "description": null
        },
        {
            "user": {
                "id": 1,
                "username": "Polshetta",
                "first_name": "Pablo",
                "last_name": "Ulman"
            },
            "attended": true,
            "rating": 5,
            "description": "Alto Chow"
        }
    ],
    "pagination": {
        "limit": 15,
        "offset": 0,
        "nextPage": null,
        "total": "2"
    }
}
```

Para pensar:

Luego de comenzado el proyecto, seguramente veamos que hay varias porciones de código similares. Por ejemplo:

- En varios lugares se tiene que validar que el campo tenga al menos X letras y no esté vacío.
- En varios lugares se tiene que validar que el email sea sintácticamente válido.
- En varios lugares se pide que el “id” exista.
- En varios lugares, se tiene que hacer un getById, getAll, etc...

No se les ocurre algo como para no tener que repetir siempre el mismo código en todos lados?