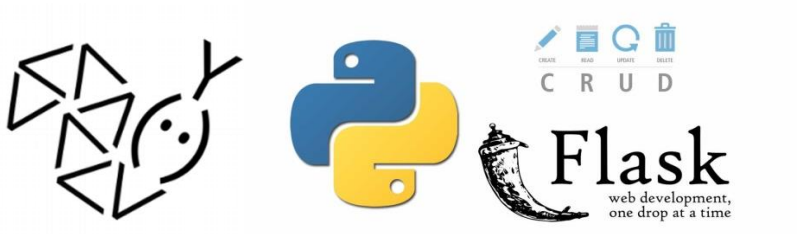
Documentación Técnica de la API



API elaborada para ejecutar el servidor backend del trabajo final integrador del Curso Desarrollo Full Stack Python Codo a Codo 4.0 eventos. Realizada por los estudiantes Adrian Nicolas Bonfanti; D.N.I. N° 36.017.999 y Gonzalo Ismael Candia Gonzalez; D.N.I. N° 36.017.999.

Esta API proporciona servicios para administrar funciones, grupos, productores y usuarios, ofreciendo una solución integral para la planificación y organización de eventos artísticos.

Repositorio

https://github.com/gonzalocandia92/backend-teatro

Características Principales

- Gestión de Usuarios y Roles: La API permite la creación y gestión de usuarios con roles específicos, como administradores y usuarios estándar. La autenticación se realiza mediante JSON Web Tokens (JWT), asegurando un acceso seguro a las funcionalidades.
- Administración de Funciones: La plataforma facilita la gestión de funciones artísticas, incluyendo detalles como título, fecha, hora e imágenes asociadas. Cada función está vinculada a un grupo y un productor.
- Catálogo de Grupos y Productores: La API ofrece endpoints para obtener listas de grupos y productores, permitiendo una administración eficiente de los artistas y profesionales involucrados en cada evento.
- Registro de ventas: Suministra un modelo para el registro de ventas de funciones. Cada una vinculada a Funciones, Grupos y Productores, junto a la fecha y al monto de cada una de ellas.

Tecnologías Utilizadas

La API está desarrollada utilizando el framework Flask de Python, proporcionando un entorno ágil y eficiente para la creación de aplicaciones web. A continuación, se detallan algunas de las tecnologías clave utilizadas:

- Flask: Un framework web ligero y modular que facilita el desarrollo rápido y la creación de APIs RESTful de manera sencilla.
- SQLAlchemy: Un ORM (Object-Relational Mapping) que simplifica la interacción con la base de datos y proporciona una capa de abstracción sobre el sistema de gestión de bases de datos.
- Flask-Security: Proporciona herramientas para gestionar la autenticación, autorización y otros aspectos de seguridad en la aplicación.
- Flask-JWT-Extended: Se utiliza para la generación y verificación de JSON Web Tokens, asegurando la autenticación segura de los usuarios.
- Marshmallow: Una biblioteca de serialización y validación que simplifica la conversión entre objetos Python y representaciones JSON.

Base URL

La API está hosteada en el siguiente dominio:

```
https://gcandia1992.pythonanywhere.com
```

Autenticación

La API utiliza JSON Web Tokens (JWT) para la autenticación. Algunas rutas requieren el rol “administrador” para acceder. Asegúrate de incluir el token en la cabecera de la solicitud como sigue:

```
Authorization: Bearer <tu_token_jwt>
```

Rutas Disponibles

Usuarios

- GET /usuarios: Obtiene la lista de usuarios.
- POST /register: Registra un nuevo usuario.
- POST /login: Inicia sesión y devuelve un token JWT.
- POST /logout: Cierra la sesión del usuario actual.

Funciones

- GET /funciones: Obtiene la lista de funciones.
- GET /funciones/{funcion_id}: Obtiene detalles de una función específica.
- POST /dashboard/funciones/crear: Crea una nueva función.
- GET/PUT/DELETE /dashboard/funciones/{funcion_id}: Gestiona una función específica.

Grupos

- GET /grupos: Obtiene la lista de grupos.
- GET /grupos/{grupo_id}: Obtiene detalles de un grupo específico.
- POST /dashboard/grupos: Crea un nuevo grupo.
- GET/PUT/DELETE /dashboard/grupos/{grupo_id}: Gestiona un grupo específico.

Productores

- GET /productores: Obtiene la lista de productores.
- GET /productores/{productor_id}: Obtiene detalles de un productor específico.
- POST /dashboard/productores: Crea un nuevo productor.
- GET/PUT/DELETE /dashboard/productores/{productor_id}: Gestiona un productor específico.

Funciones

- POST /ventas: Requiere token JWT, permite el registro de ventas de los clientes logueados
- GET /dashboard/ventas Obtiene lista de ventas
- GET, PUT, DELETE /dashboard/ventas/{id}: Gestiona una venta

Roles

La API utiliza roles para gestionar el acceso a ciertas rutas. Los roles disponibles son:

- administrador: Acceso completo a todas las funcionalidades.
- usuario: Acceso limitado.

Modelos y Relaciones

1. **Funcion:**
    - Representa una función o evento en la plataforma.
    - Atributos:
        - **id**: Identificador único de la función.
        - **titulo**: Título de la función.
        - **fecha**: Fecha de la función.
        - **hora**: Hora de la función.
        - **imagen**: Imagen asociada a la función.
    - Relaciones:
        - Pertenencia a un **Grupo** mediante **grupo_id**.
        - Pertenencia a un **Productor** mediante **productor_id**.
        - Relación inversa: **grupo** y **productor** mediante **backref='funciones'**.
2. **Grupo:**
    - Representa un grupo asociado a funciones.
    - Atributos:
        - **id**: Identificador único del grupo.
        - **nombre**: Nombre del grupo.
        - **integrantes**: Número de integrantes del grupo.
    - Relaciones:
        - Muchas funciones pueden pertenecer a un grupo a través de **backref='funciones'**.
3. **Productor:**
    - Representa un productor asociado a funciones.
    - Atributos:
        - **id**: Identificador único del productor.
        - **nombre**: Nombre del productor.

- Relaciones:
    - Muchas funciones pueden pertenecer a un productor a través de **backref='funciones'**.

4. **Role:**
    - Representa un rol asociado a los usuarios.
    - Atributos:
        - **id**: Identificador único del rol.
        - **name**: Nombre del rol.
        - **description**: Descripción del rol.
5. **User:**
    - Representa un usuario registrado en la plataforma.
    - Atributos:
        - **id**: Identificador único del usuario.
        - **email**: Correo electrónico del usuario.
        - **username**: Nombre de usuario del usuario.
        - **password**: Contraseña del usuario.
        - **active**: Estado activo/inactivo del usuario.
        - **fs_uniquifier**: Identificador único para Flask-Security.
    - Relaciones:
        - Muchos roles pueden pertenecer a un usuario a través de **backref='user_roles'**.
        - Muchas funciones pueden pertenecer a un usuario a través de la relación inversa **funciones**.
        - Muchos grupos pueden pertenecer a un usuario a través de la relación inversa **grupos**.
        - Muchos productores pueden pertenecer a un usuario a través de la relación inversa **productores**.
6. **UserRoles:**
    - Tabla de asociación que representa la relación muchos a muchos entre usuarios y roles.
    - Atributos:
        - **id**: Identificador único de la asociación.
        - **user_id**: Clave foránea que referencia al usuario.
        - **role_id**: Clave foránea que referencia al rol.

7. **Ventas:**
    - Representa una venta de funciones
    - Atributos:
        - **id**: Identificador único de la venta.
        - **fecha_venta**: Fecha de la venta
        - **hora_venta**: Hora de la venta
        - **monto**: Valor en $ de la venta
    - Relaciones:
        - Muchos funciones, grupos, productores y usuarios pueden pertenecer a una venta a través de **backref='ventas'**.

Ejemplos de Solicitudes

Obtener Lista de usuario https://gcandia1992.pythonanywhere.com/usuarios


```
{
  "usuario": {
    "email": "administrador@administrador.com",
    "id": 6,
    "roles": [
      {
        "description": "administrador",
        "id": 1,
        "name": "administrador"
      }
    ]
  }
}
```


Obtener lista de funciones https://gcandia1992.pythonanywhere.com/funciones


```
{
  "id": 1,
  "titulo": "Concierto",
  "fecha": "2023-12-01",
  "hora": "18:00",
  "imagen": "url_imagen",
  "grupo": {
    "id": 1,
    "nombre": "Grupo Musical",
    "integrantes": 5
  },
  "productor": {
    "id": 1,
    "nombre": "Productor Musical"
  }
}
```

Obtener productores https://gcandia1992.pythonanywhere.com/productores


```
{
  "productores": [
    {
      "id": 1,
      "nombre": "Productor A"
    },
    {
      "id": 2,
      "nombre": "Productor B"
    }
  ]
}
```


Obtener lista de grupos

https://gcandia1992.pythonanywhere.com/grupos


```
{
  "grupos": [
    {
      "id": 1,
      "nombre": "Grupo X",
      "integrantes": 5
    },
    {
      "id": 2,
      "nombre": "Grupo Y",
      "integrantes": 3
    }
  ]
}
```