

Microservicio en GraphQL para el Listado de Usuarios

Adrián Fabricio Ordóñez Paredes

Tecnologías Web II

I - 2025

Universidad Católica Boliviana “San Pablo”

1. Análisis del Proyecto Integrador

- Identificación clara del endpoint

Endpoint que se consumirá:

- **GET** /usuarios/listar

Para obtener la lista de los usuarios registrados

- Justificación de la elección del endpoint

Se eligió este endpoint porque:

- Permite obtener la lista de usuarios registrados en la base de datos para la utilización de la plataforma de podcasts
- Es crucial para la verificación de creación (registro) de los usuarios a la plataforma para que puedan utilizarla
- Es viable de convertirlo a un esquema que GraphQL nos permite consumir como microservicio y de forma más clara y directa

- Descripción técnica del endpoint (estructura, método, formato de datos)

Característica	Descripción
Método	GET (usuarios)
URL	/usuarios/listar
Formato de Datos	JSON
Formato de Respuesta	JSON

2. Diseño del Microservicio

- Objetivo del microservicio claramente definido

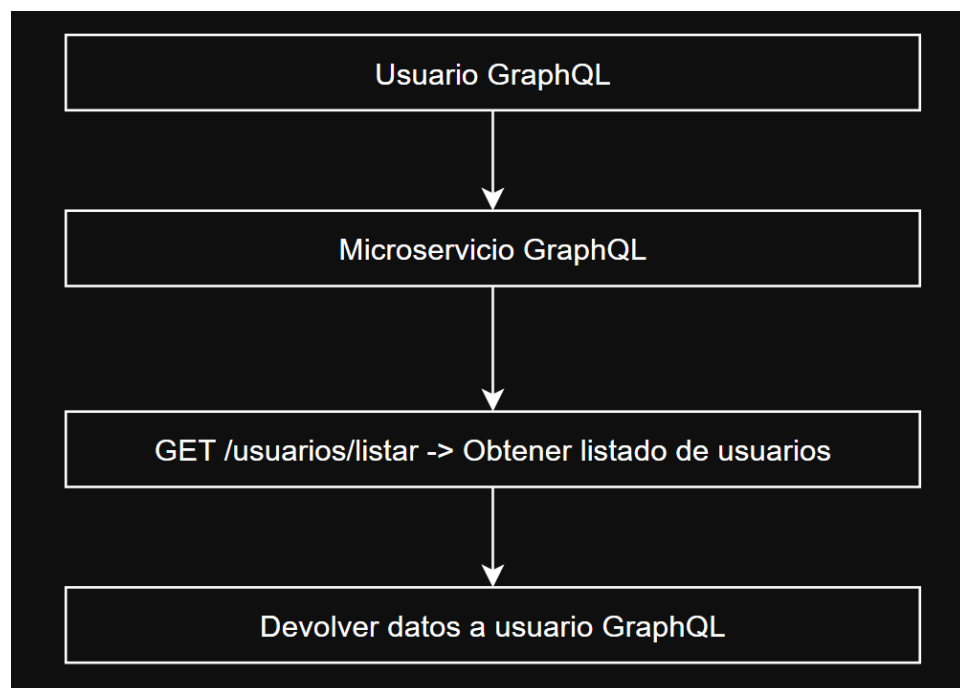
Objetivo: Construir un microservicio en GraphQL que permita a los administradores tener acceso simple, estructurado y rápido para la verificación de creación de usuarios en el sistema.

- Elección y justificación de la tecnología de comunicación

Se eligió GraphQL porque:

- Permite obtener resultados mejor estructurados
- Permite hacer consultas a campos específicos
- Facilita las consultas constantes

- Diagrama de Flujo de Integración



3. Implementación Técnica

Estructura del proyecto:

/Podcast_bicentenario-main

```
|  
|— backend/backend  
| |— models/  
| |— templates/  
| |— settings.py  
| |— urls.py  
| |— .env
```

- |— graphql_service/
 - |— schema.py
- |— manage.py
- |— frontend/
- |— [README.md](#)

Código Base

1. schema.py:

```
import graphene
import requests

class UserType(graphene.ObjectType):
    idusuario = graphene.ID()
    usuario = graphene.String()
    correo = graphene.String()
    fecha_ingreso = graphene.String()
    rol = graphene.String()

class Query(graphene.ObjectType):
    usuarios = graphene.List(UserType)

    def resolve_usuarios(self, info):
        response = requests.get("http://127.0.0.1:8000/usuarios/listar/")
        if response.status_code == 200:
            data = response.json()
            return data.get('usuarios', [])
        return []

schema = graphene.Schema(query=Query)
```

2. backend/backend/urls.py:

```
from django.contrib import admin
from django.urls import path

from django.urls import include, path
```

```

from graphene_django.views import GraphQLView
from graphql_service.schema import schema
from . import views

urlpatterns = [
    path('usuarios/crear/', views.crear_usuario,
name='crear_usuario'), ##con Postman
    path('usuarios/listar/', views.listar_usuarios,
name='listar_usuarios'), ##con Postman

    path('creadores/listar/', views.listar_creadores, name='listar_
_creadores'),
    path('creadores/registrar',
views.mostrar_formulario_registro, name='registrar'),
    path('registro', views.registro, name='registro'),
    path('creadores/mostrar', views.mostrar_creadores,
name='mostrar_creadores'),

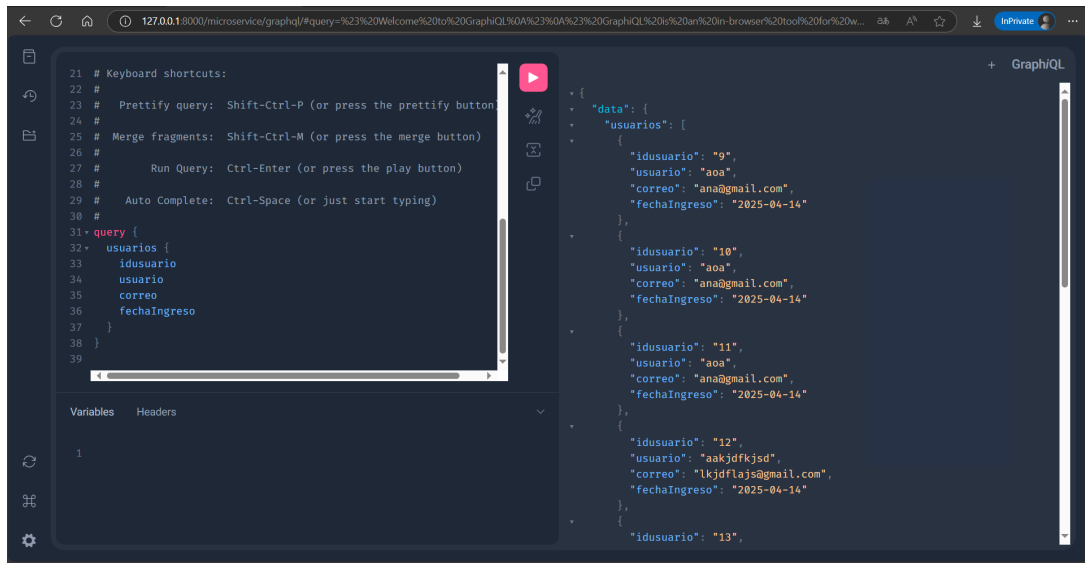
    path('usuarios/seguirCreador', views.seguirCreador, name='segu
ir_creador'), ##con Postman
    path('usuarios/obtenerSeguimientos/',
views.obtenerSeguimientos,
name='obtener_seguimientos'), ##con Postman
    path('microservice/graphql/',
GraphQLView.as_view(graphiql=True)),
]

```

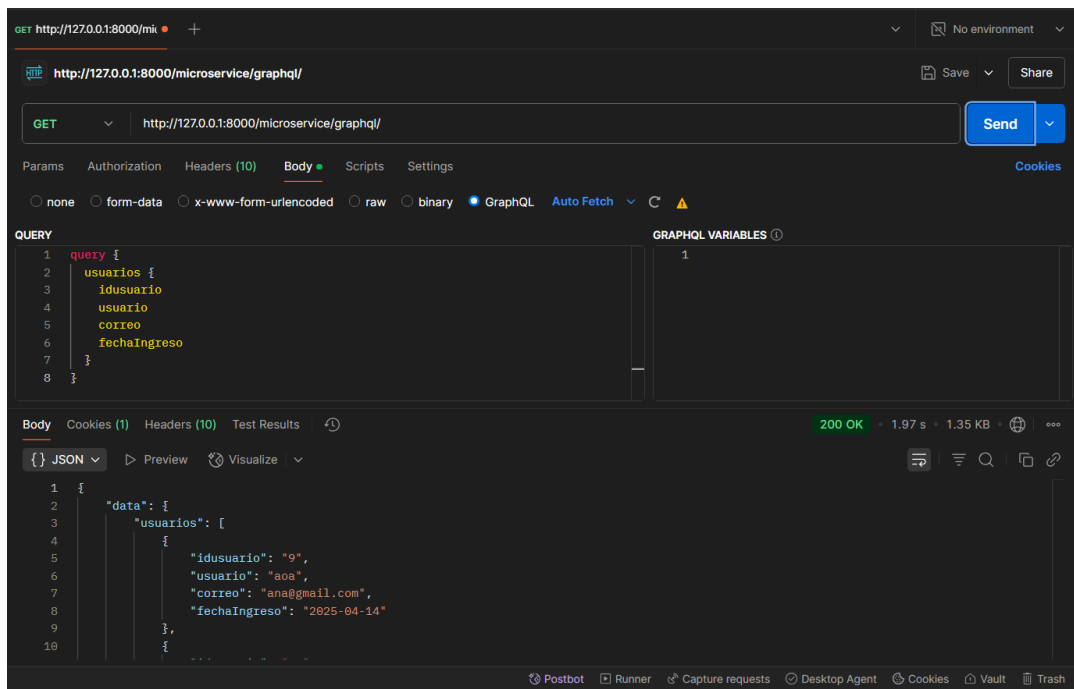
4. Pruebas y Documentación

- **Evidencias Funcionales**

GraphQL Playground:



Postman:



- **Pruebas Unitarias con Pytest:**

```

PS C:\Users\usuario\Documents\GitHub\copiaPodcast\Podcast_bicentenario-main\backend\backend> pytest graphql_service/tests/test_schema.py -v
===== test session starts =====
platform win32 -- Python 3.13.2, pytest-8.3.5, pluggy-1.5.0 -- C:\Users\usuario\AppData\Local\Programs\Python\Python313\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\usuario\Documents\GitHub\copiaPodcast\Podcast_bicentenario-main\backend\backend
plugins: anyio-4.9.0, django-4.11.1, mock-3.14.0, requests-mock-1.12.1
collected 2 items

graphql_service/tests/test_schema.py::test_usuarios_query PASSED [ 50%]
graphql_service/tests/test_schema.py::test_create_user_mutation PASSED [100%]

===== 2 passed in 0.38s =====
PS C:\Users\usuario\Documents\GitHub\copiaPodcast\Podcast_bicentenario-main\backend\backend>

```

Esta prueba simula la llamada a la API usando una función *requests_mock*, luego ejecuta una consulta de GraphQL para los datos de usuarios, finalmente verifica que:

- No ocurran errores
 - Los datos de usuarios retornados coincidan con los datos de simulación
 - La prueba retorna exactamente un usuario
- **Documentación Clara** (Leer archivo [README.md](#))
 - **Ejemplo de Consulta**

```
query {  
  usuarios {  
    idusuario  
    usuario  
    correo  
    fechaIngreso  
  }  
}
```

- **Enlace al repositorio**
https://github.com/AdrianOrdonez2705/AdrianOrdonez_2daEvaluacionBackend