

# **SEGUNDO EXAMEN DE TECNOLOGÍAS WEB 2**

**Universidad Católica Boliviana “San Pablo”**



**Nombre: Huascar Camilo Durán Avendaño**

**Fecha: 3/5/2025**

**Carrera: Ingeniería en Sistemas**

**Docente: Miguel Ángel Pacheco Artega**

## 1. Análisis del Proyecto Integrador:

### 1.1 Identificación del Endopint

- **POST /api/login**
  - Devuelve un token JWT autenticando con usuario y contraseña.
- **GET /api/users**
  - Devuelve una lista de usuarios registrados. Requiere token en header.

### 1.2 Justificación de la Elección

- El endpoint /api/users (o /api/students) fue elegido porque representa datos claves en el proyecto integrador.
- Es útil para construir dashboards, paneles administrativos o listados.
- Al encapsularlo en GraphQL, podemos seleccionar sobre los campos necesarios y reducir carga de red.
- GraphQL facilita la integración con frontends modernos como React.

### 1.3 Descripción Técnica del Endpoint

Característica	Valor
Método	POST (login), GET (users)
URL	/api/login, /api/users
Autenticación	JWT (Authorization: Bearer <token>)
Formato de datos	JSON (usuario/contraseña para login)
Formato de respuesta	JSON (array de objetos con campos como id, name, etc.)

## 2. Diseño del Microservicio:

### 2.1 Objetivo del microservicio

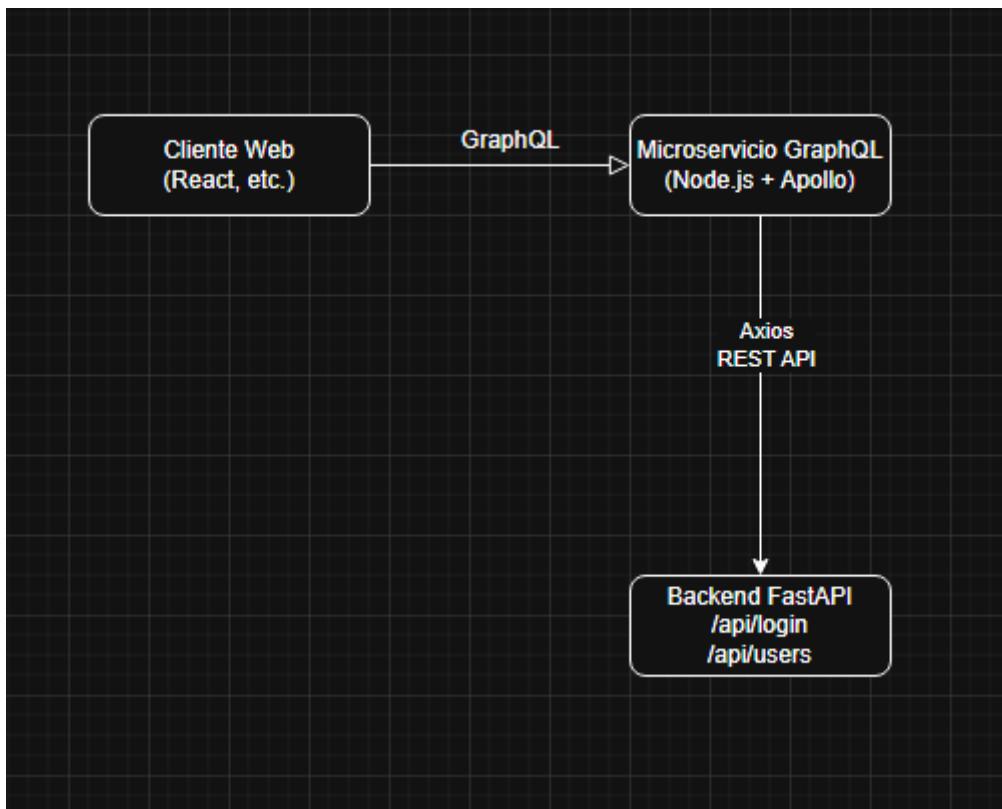
- Construir un microservicio independiente usando GraphQL que consuma el endpoint REST /api/users (o /api/students) de nuestro backend FastAPI. Este microservicio manejará la autenticación automáticamente mediante login y expondrá una consulta users para obtener datos desde un nuevo endpoint /graphql.

### 2.2 Elección y Justificación de la Tecnología de Comunicación

Se elige GraphQL como tecnología de comunicación por las siguientes razones:

- Permite consultar solo los campos necesarios (ej. name, email), mejorando el rendimiento.
- Es ideal para agregar una capa flexible sobre APIs REST ya existentes.
- Simplifica la integración con clientes como React.
- Permite extender fácilmente nuevas funcionalidades o relaciones entre recursos.

## 2.3 Diagrama del Flujo de Integración



## 3. Implementación técnica

### Estructura del proyecto

/MICROSERVICIO-GRAPHQL

```
|── src/  
|   ├── resolvers.js  
|   ├── schema.graphql  
|   ├── apiClient.js  
|   └── server.js  
├── .env  
└── package.json  
└── README.md
```

## Código base:

### 1. schema.graphql

```
type User {  
    id: ID  
    email: String  
    is_active: Boolean  
}
```

```
type Query {
```

```
    users: [User]  
}
```

### 2. resolvers.js

```
const { getUsers } = require('./apiClient');
```

```
const resolvers = {
```

```
    Query: {
```

```
        users: async () => {  
            return await getUsers();  
        }  
    }  
};
```

```
module.exports = resolvers;
```

### 3. apiClient.js

```
const axios = require('axios');  
require('dotenv').config();
```

```
const API_URL = process.env.API_URL;  
const USERNAME = process.env.USERNAME;
```

```
const PASSWORD = process.env.PASSWORD;

let token = null;

async function login() {
  const response = await axios.post(`${API_URL}/users/login`, {
    email: USERNAME,
    password: PASSWORD
  });
  token = response.data.access_token;
}

async function getUsers() {
  if (!token) {
    await login();
  }

  const response = await axios.get(`${API_URL}/users`, {
    headers: {
      Authorization: `Bearer ${token}`
    }
  });
}

return response.data;
}

module.exports = { getUsers };
```

#### **4. server.js**

```
const { ApolloServer } = require('apollo-server');
const fs = require('fs');
const path = require('path');
const resolvers = require('./resolvers');

const typeDefs = fs.readFileSync(path.join(__dirname, 'schema.graphql'), 'utf8');

const server = new ApolloServer({
  typeDefs,
  resolvers,
});

server.listen({ port: 4000 }).then(({ url }) => {
  console.log(`🚀 Microservicio GraphQL activo en: ${url}`);
});
```

#### **5. .env**

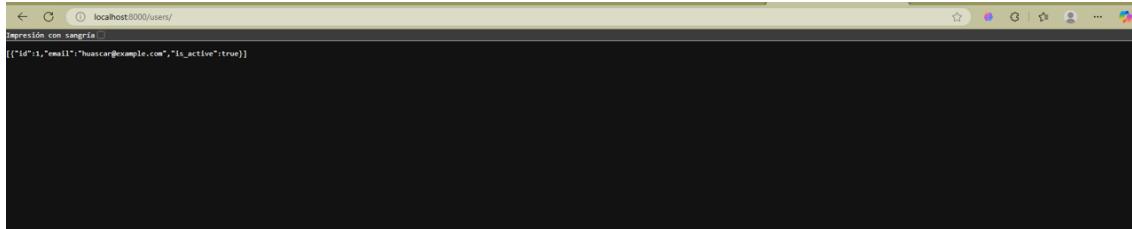
```
API_URL=http://localhost:8000
USERNAME=graphql@test.com
PASSWORD=123456
```

#### **Librerias necesarias**

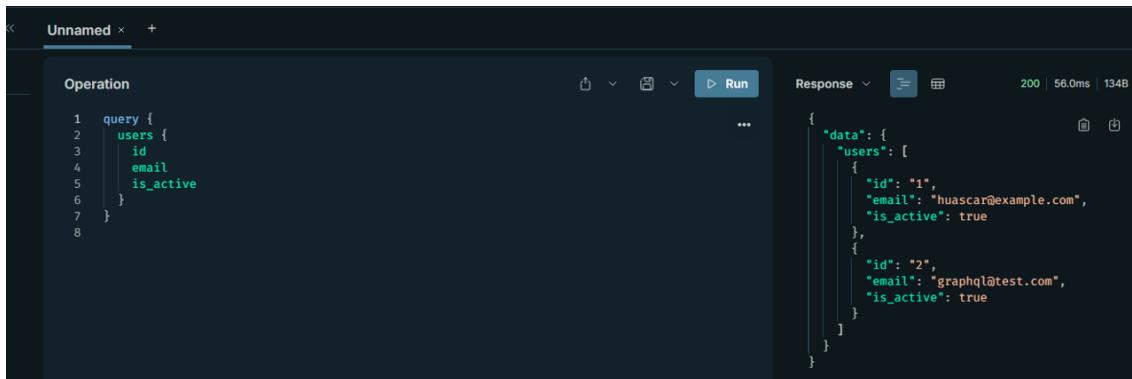
```
npm install apollo-server graphql axios dotenv
```

## 4. Pruebas y documentación

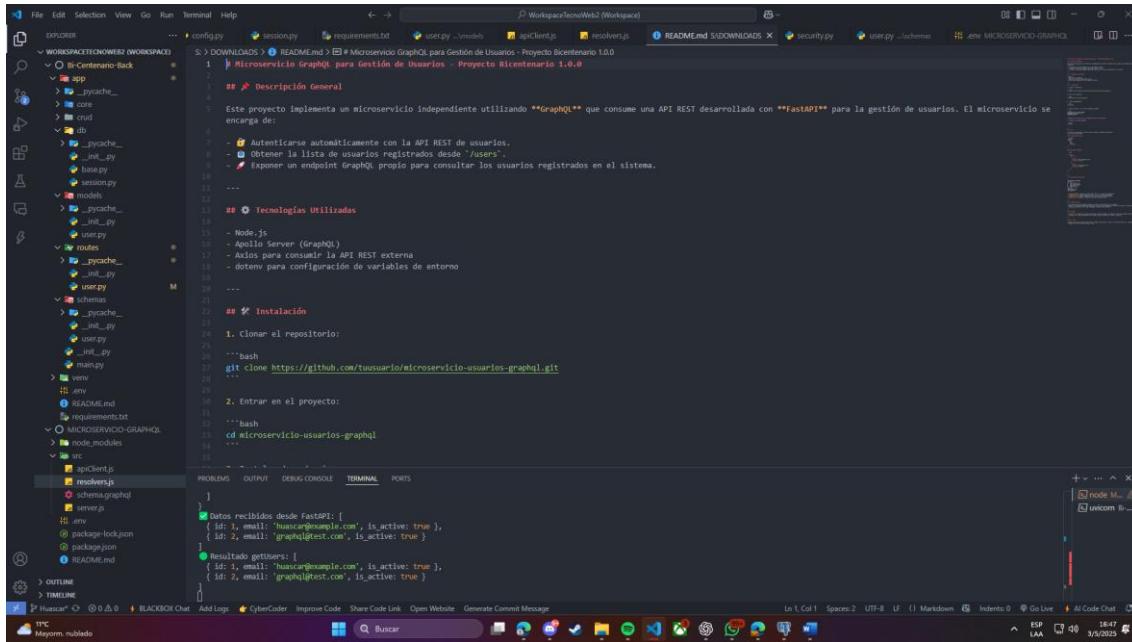
- Backend FastAPI respondiendo correctamente con JSON limpio.



- Consulta exitosa en Apollo Studio: GraphQL muestra usuarios desde FastAPI



Se creó un Workspace con los dos proyectos, para poder trabajar con el backend del proyecto fundador.



## Pruebas Manuales

### 4.1. Probar sin token: validar login automático

- Condición inicial: Al iniciar la aplicación, no hay token cargado.

- Prueba realizada: Se realizó una consulta users desde Apollo Studio sin autenticarse manualmente.
- Resultado esperado: El microservicio ejecuta login() automáticamente y obtiene un token desde el backend FastAPI.
- Resultado obtenido: El token se obtiene correctamente y se usa en la cabecera Authorization para acceder al endpoint /users.

## 4.2 Probar consulta de usuarios: validar datos correctos

- **Consulta usada:**

```
query {
  users {
    id
    email
    is_active
  }
}
```

- **Resultado esperado:** Se recibe una lista con los datos de usuarios registrados en el backend.

- **Resultado obtenido:** Se mostraron correctamente usuarios como:

```
[
  {
    "id": 1,
    "email": "huascar@example.com",
    "is_active": true
  },
  {
    "id": 2,
    "email": "graphql@test.com",
    "is_active": true
  }
]
```

- **Validación:** Los datos fueron verificados directamente desde la base de datos y desde la respuesta del endpoint /users en FastAPI.

## 4.3 README.MD

Se adjuntará el archivo README.MD con todas las instrucciones dadas.

## **5. Presentación final:**

### **Justificación técnica del uso de GraphQL**

Se optó por implementar un microservicio GraphQL como intermediario entre el cliente y el backend REST desarrollado con FastAPI, con el objetivo de desacoplar responsabilidades y exponer una capa más flexible de consulta de datos. GraphQL permite al cliente definir exactamente los campos que necesita, reduciendo el overfetching típico de APIs REST. Esta arquitectura permite escalar funcionalidades futuras agregando nuevas consultas o mutaciones sin alterar la API original, manteniendo una integración desacoplada y eficiente.