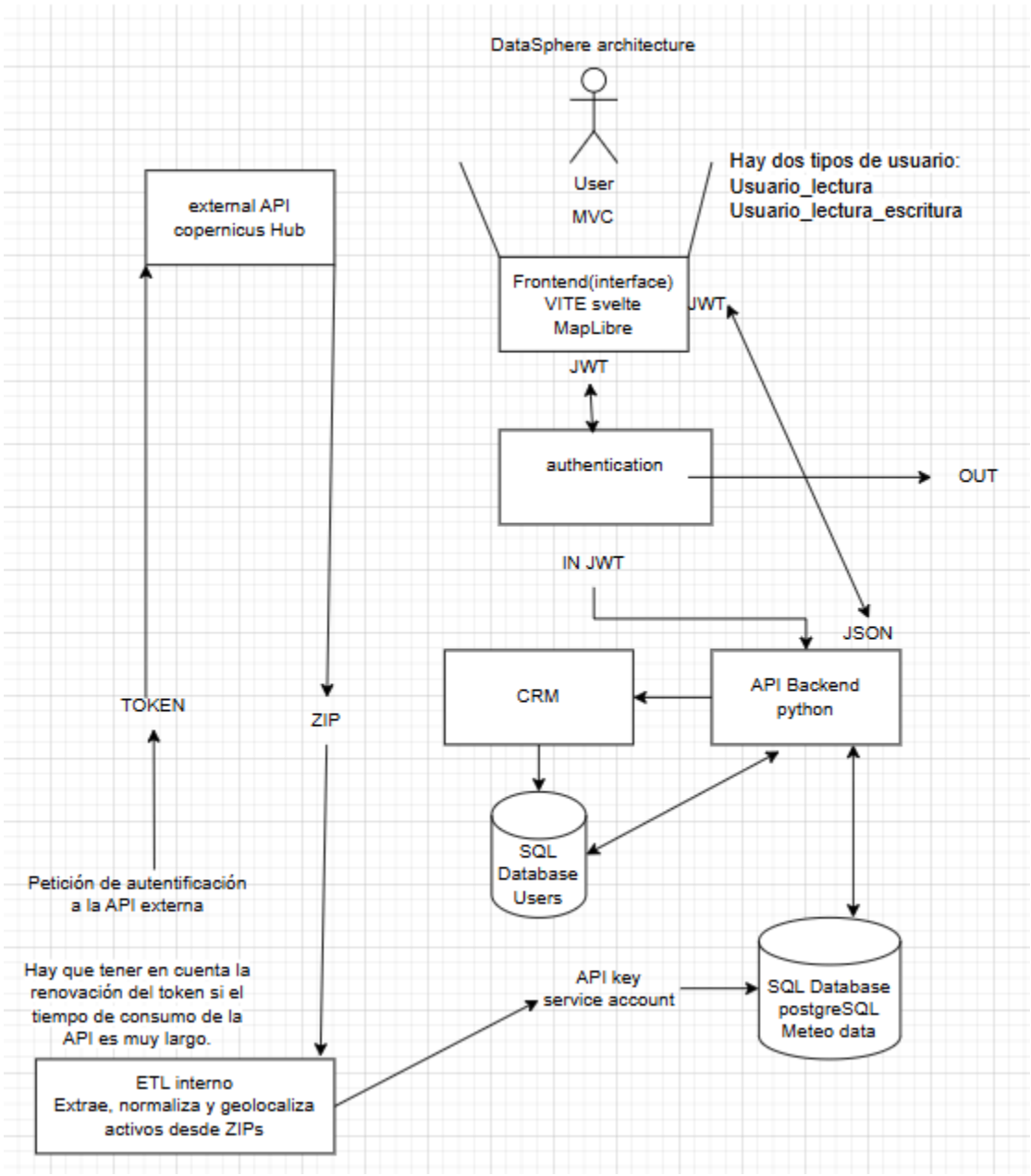
Actividad 1: Arquitectura de Aplicaciones Web - Caso Práctico
Alumno: Antonio López García

TAREA 1 — DIAGRAMA DE ARQUITECTURA (Draw.io)



TAREA 2 — DIAGRAMA DE AUTENTICACIÓN

Autenticación humano:

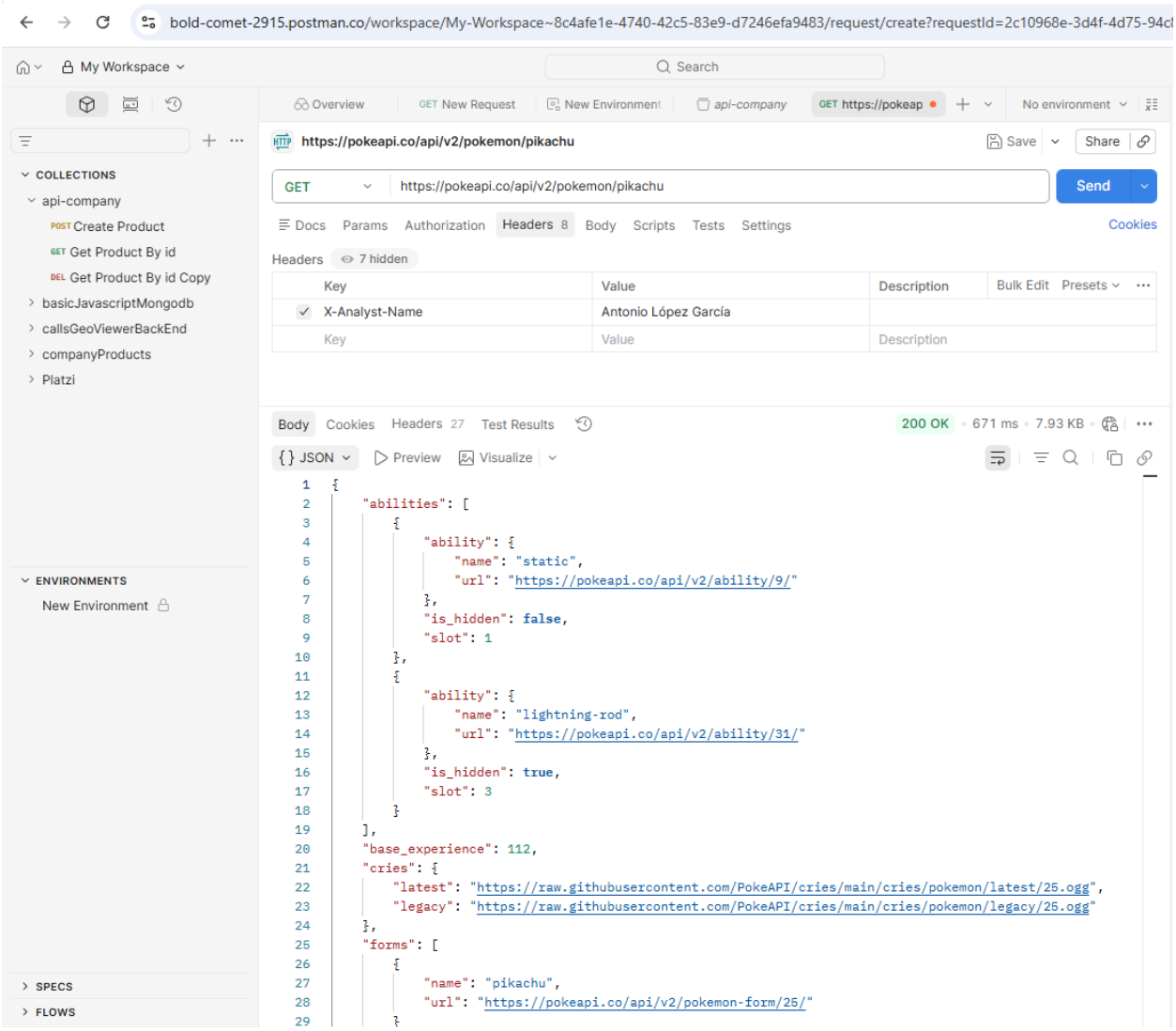
1.El usuario interactúa con el Frontend (Svelte) e introduce sus credenciales. Estas se envían al servicio de autenticación.
2.El servicio de autenticación verifica la identidad y devuelve al Frontend un token JWT. Dentro de la de este token, va el rol del usuario.
3.Para cargar los datos, el Frontend hace peticiones Backend adjuntando el JWT en las cabeceras HTTP .
4.El Backend verifica el token, extrae el rol. Si el rol tiene los permisos adecuados, autoriza la operación y devuelve la información en formato JSON (200 OK). Si no tiene permisos para esa acción, rechaza la petición (403 Forbidden).

Autenticación no humano:

Credenciales externas / Token de acceso temporal (Comunicación Machine-to-Machine).
Para descargar los pesados ficheros de datos meteorológicos (ZIP), necesita acceder a la API externa de Copernicus Hub.

1.El script ETL realiza una petición inicial a la API externa proporcionando sus propias credenciales o API Key.
2.Copernicus Hub valida la solicitud y devuelve un Token temporal. El ETL inyecta este Token en las cabeceras de sus peticiones HTTP posteriores para autorizar la descarga continua de los ficheros ZIP.
3.La descarga y procesamiento de grandes volúmenes de datos puede ser un proceso prolongado, el proceso ETL está diseñado para monitorizar el tiempo de vida de este token y gestionar su renovación de forma transparente, evitando así cortes de conexión o respuestas.

TAREA 3 — TRAZA HTTP REAL (POSTMAN)
GET 200
Al pedir la información de `/pokemon/pikachu`, el servidor la encuentra y nos responde con un código 200 OK, entregándonos los datos solicitados.
Esto demuestra una comunicación correcta entre quien pide la información (el cliente) y quien la tiene (el servidor). Si llevamos este ejemplo a DataSphere, esta es la respuesta que queremos ver cuando nuestro sistema consulta datos del clima a un proveedor externo. Significa que el proveedor ha entendido la petición y nos envía la información (como predicciones o coordenadas) para que podamos seguir analizando los riesgos sin problemas.

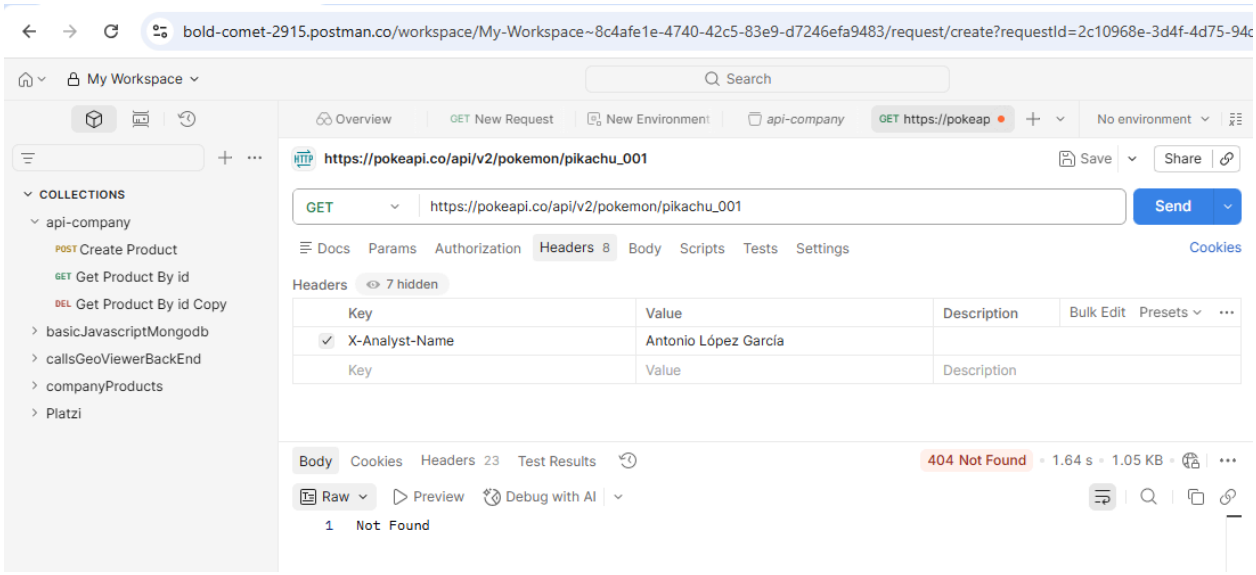


GET 404

Al cambiar la dirección hacia algo que no existe (`/pokemon/pikachu_001`), el servidor nos contesta con un código 404 Not Found.

Lo importante aquí es entender que esto no es un fallo de internet ni del propio servidor; la comunicación ha funcionado perfectamente, pero el servidor nos está diciendo "no tengo lo que buscas".

En DataSphere, nuestro sistema tiene que estar preparado para estas situaciones. Si le pedimos datos externos sobre un huracán y recibimos un 404, nuestra aplicación no debe bloquearse. Simplemente debe tomar nota de que ese dato no está disponible y continuar trabajando con el siguiente, para no detener de golpe todo el análisis.



TAREA 4 — GIT STRATEGY + HISTORIAL

```
PS C:\Users\a.lopez.g\Documents\trabajos\programming\entrega-arquitectura-datasphere> git log --graph --pretty
* bf5ac04 - docs(release): compile and add final deliverable for activity 1 (63 seconds ago) <Lopez>
* 8fe6e33 - Revert "chore: add environment configuration file" (12 hours ago) <Lopez>
* 613d12d - chore: add environment configuration file (12 hours ago) <Lopez>
* cca607d - docs(strategy): define version control guidelines and folder structure (12 hours ago) <Lopez>
* fe18314 - docs(auth): add JWT and API Key authentication flow (12 hours ago) <Lopez>
* a647406 - docs(architecture): add system diagram and component description (12 hours ago) <Lopez>
PS C:\Users\a.lopez.g\Documents\trabajos\programming\entrega-arquitectura-datasphere>
```

La gestión del control de versiones de este proyecto se ha ejecutado aplicando buenas prácticas y estándares profesionales. Tal y como refleja el árbol del historial adjunto, el flujo de trabajo se basa en los siguientes puntos:

Commits Atómicos y Semánticos: Se ha registrado en transacciones independientes utilizando *Conventional Commits* (por ejemplo, `docs(architecture): add system diagram`). Esto genera un historial predecible y fácil de auditar.

Se simuló la subida errónea de un archivo con información sensible y se solucionó aplicando el comando `git revert`. Esta práctica revierte los cambios de forma segura sin alterar ni destruir la historia previa del repositorio.

Se implementó un flujo de trabajo basado en ramas. Se creó una rama secundaria (`main_dev_pro`) donde se versionó el entregable definitivo de forma aislada. Posteriormente, este trabajo se integró en la rama principal (`main`) mediante un `merge` .

Repositorio online:
https://github.com/tonilogardev/entrega-arquitectura-datasphere