"Playing" con diferentes clasificadores y datasets

Haessler Joan Ortiz Moncada

Universidad Distrital Francisco José de Caldas
Facultad de Ingeniería
Curso: Big Data

12 de septiembre de 2025

1. Introducción

Este documento presenta el desarrollo del taller #2 del curso de Construcción de Pruebas de Software. Su propósito es documentar los hallazgos de la implementación de la recursión en métodos de suma, multiplicación y cálculo del factorial, así como su integración, utilizando el lenguaje de programación **Java**. Además, se aplican pruebas unitarias a estos métodos, se publica el proyecto en **GitHub** y se automatiza la ejecución de dichas pruebas ante cada *push* y *pull request*. Por último, el informe explica brevemente qué es Postman, para qué sirve y muestra algunos ejemplos de uso.

2. Metodología

El proyecto Java con la implementación recursiva se desarrolló en el Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés) **IntelliJ IDEA**, aprovechando que ya contaba con este software y con el Kit de Desarrollo de Java (JDK) instalado en la máquina local. Para el diseño, ejecución y documentación de las pruebas se empleó el *framework* **JUnit**. El control de versiones y la publicación del proyecto en un repositorio remoto se realizaron con **Git** y **GitHub**. Finalmente, la automatización de las pruebas unitarias se configuró mediante un archivo `.yml` para que la sección **Actions** de GitHub ejecute las pruebas en cada *push* y *pull request*.

Respecto a Postman se probaron los métodos *http* clásicos (*GET, POST, PUT, DELETE*) mediante solicitudes a una Interfaz de Programación de Aplicaciones (API, por sus siglas en inglés) pública.

3. Desarrollo

3.1. Recursión

Se diseñó una clase sencilla con tres funciones recursivas: una que suma, otra que multiplica apoyándose en la suma, y la del factorial que usa la multiplicación. Se impuso

un tope claro: sólo se calcula hasta 10!, si se excede o se ingresan negativos, se avisa con una excepción. La idea fue mantener la "recursividad en capas" sin depender de un operador de multiplicación directo dentro del factorial.

3.2. JUnit

Se craron pruebas unitarias con JUnit 5 y, para no repetir la invocación de las preubas, se usaron pruebas parametrizadas: se escribe una sola prueba y se pasan varias filas de datos. Se probaron los sigueintes aspectos:

1. **Casos correctos**: valores típicos y casos base (por ejemplo, factoriales de 0 a 10).
2. **Propiedades**: se comprobó que la multiplicación no depende del orden de los factores (conmutatividad).
3. **Errores esperados**: se verificó que, cuando toca, se lance la excepción correcta (por ejemplo, si $n > 10$ o si hay negativos).

La Figura 1 muestra como luce el proyecto Java que se construyó.

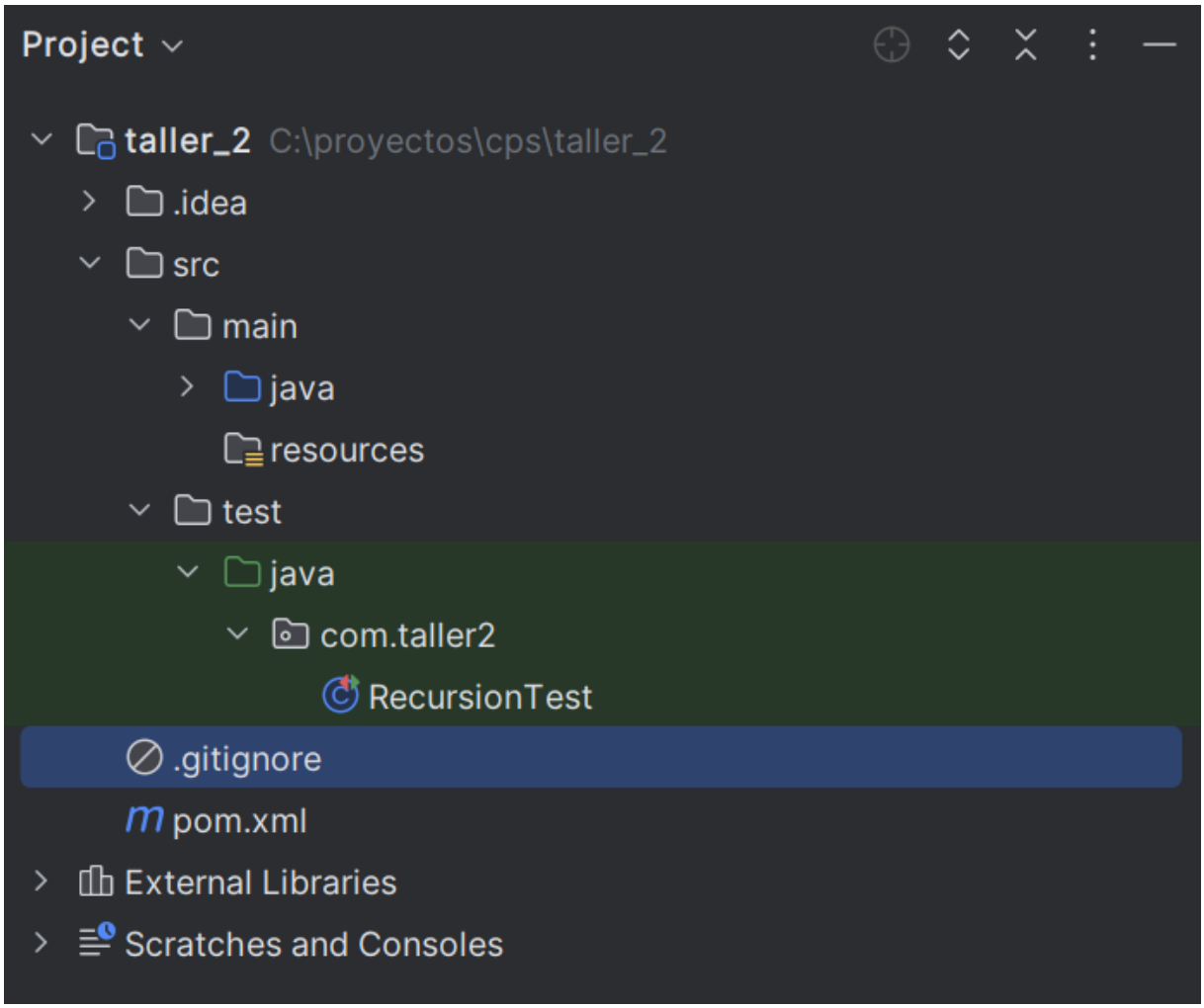

Figura 1: Estructura proyecto Java.

3.3. GitHub Actions

Una vez se hizo *push* del proyecto Java al repositorio de GitHub cps_repositorio, se agregó un archivo `workflow` en la ruta `.github/workflows/ci.yml` en la raiz del repositorio, esto con la intención de generar la automatización de las pruebas al proyecto Java realizado. Con la creación de este archivo se logró lo siguiente:

1. Se dispara solo en cada *push* y en cada *pull request*.
2. Prepara y ejecuta los tests de `taller_2`.
3. Publica los reportes de pruebas como artefactos en la pestaña *Actions*.

La Figura 2 muestra como luce la estructura del proyecto cargado y el archivo `.yml`.

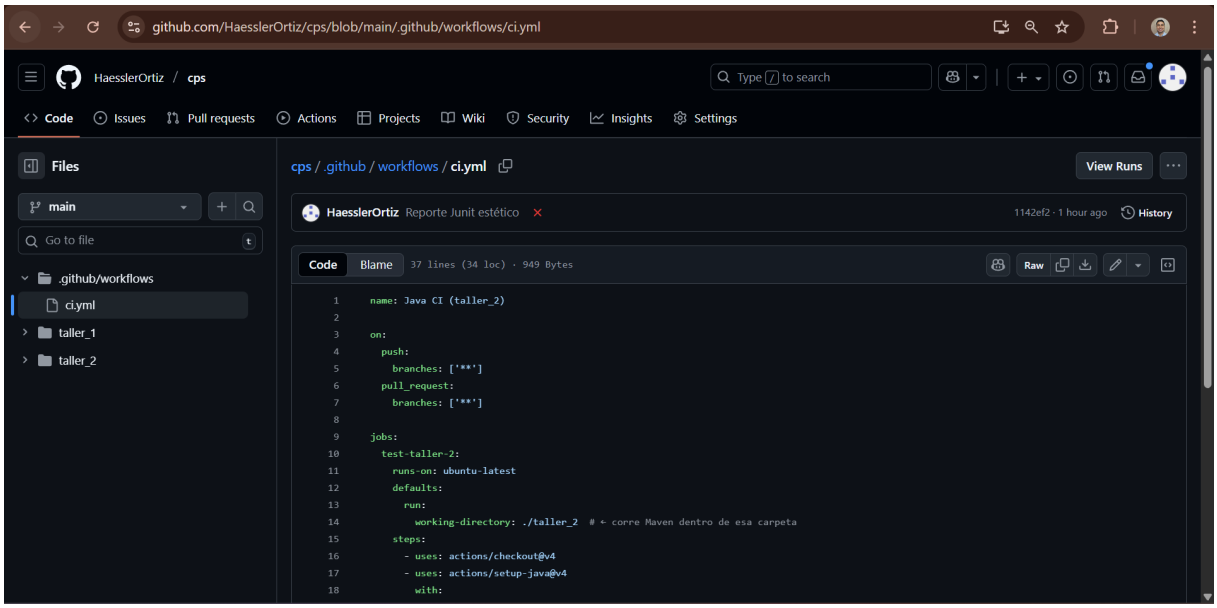

Figura 2: Estructura proyecto Java.

En cuanto a los resultados, se detectó el fallo de algunos *tests* (aunque la mayoría de las pruebas pasaron). A partir de los *logs*, la causa probable es una profundidad de recursión excesiva al calcular el factorial para ciertos valores de entrada, lo que deriva en errores de ejecución (p. ej., *stack overflow*). Como acciones de mejora, conviene simplificar la implementación (por ejemplo, adoptar una versión iterativa), validar el dominio de entrada (solo $n \geq 0$) y limitar el tamaño máximo permitido para n. La Figura 3 muestra el reporte generado por `GitHub Actions`.

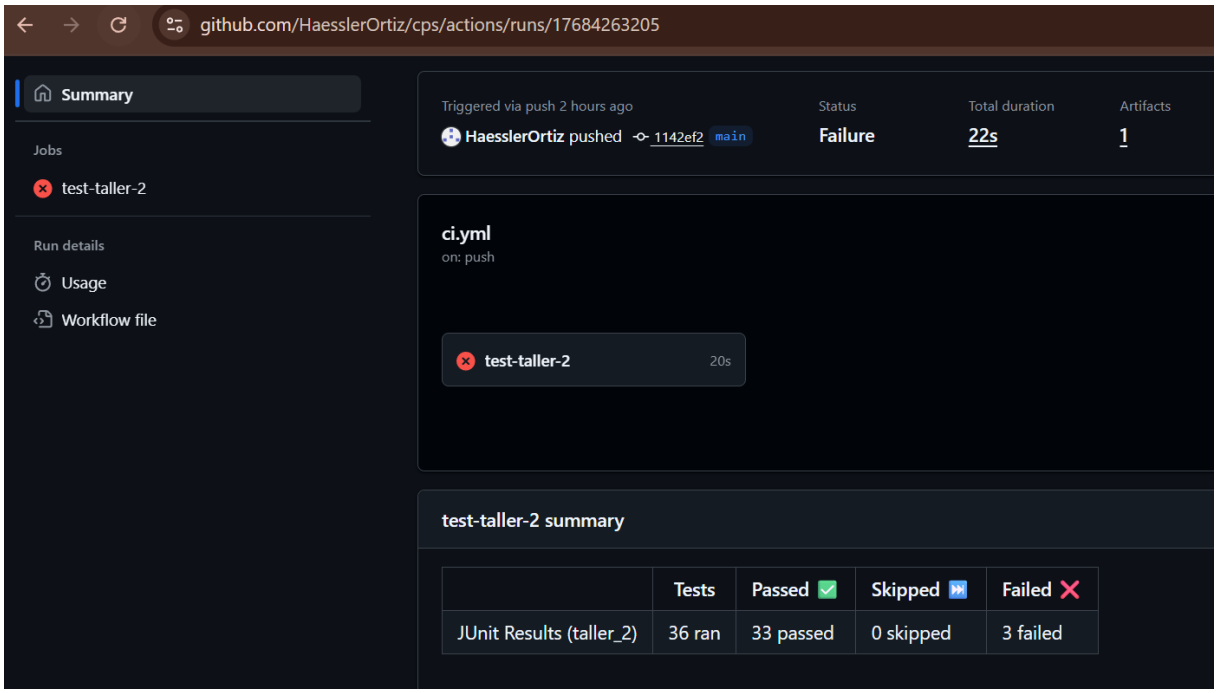

Figura 3: Reporte GitHub Actions.

El archivo `.yml` desarrollado, fue sugerida por la Inteligencia Artificial (IA) **ChatGPT**.

3.4. Postman

Postman es un programa que permite enviar preguntas a un servicio en internet y ver lo que ese servicio retorna. Es como una ventana sencilla para conversar con esos servicios y entender qué devuelven. Postman permite lo siguiente:

1. Comprobar rápidamente si un servicio en línea está funcionando.
2. Probar cómo responde cuando se envías datos.
3. Guardar y repetir pruebas sin volver a escribir todo cada vez.
4. Compartir esas pruebas con otras personas.
5. Detectar errores de un servicio.

Después de crear una cuenta (con mi correo Gmail) e iniciar una sesión en Postman, me apareció la siguiente interfaz:

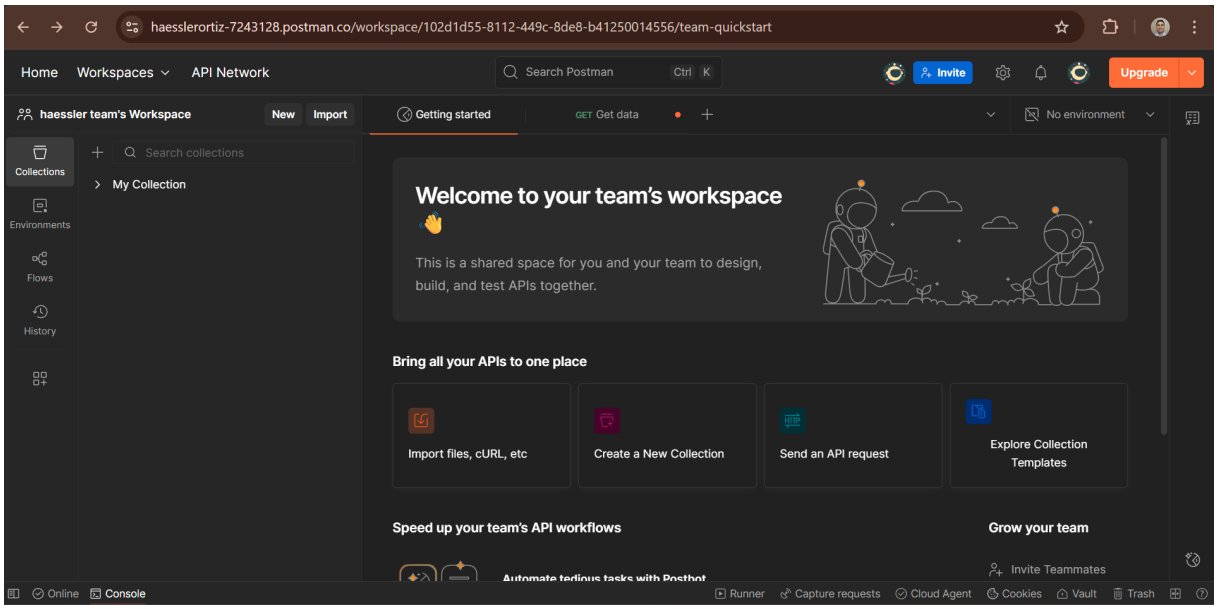

Figura 4: Reporte GitHub Actions.

Una vez allí probé los métodos clásicos *http*, en la API pública . Se obtuvieron los sigueintes resultados:

- **GET.** Diligenciando el *endpoint* se obtuvo el siguiente resultado:

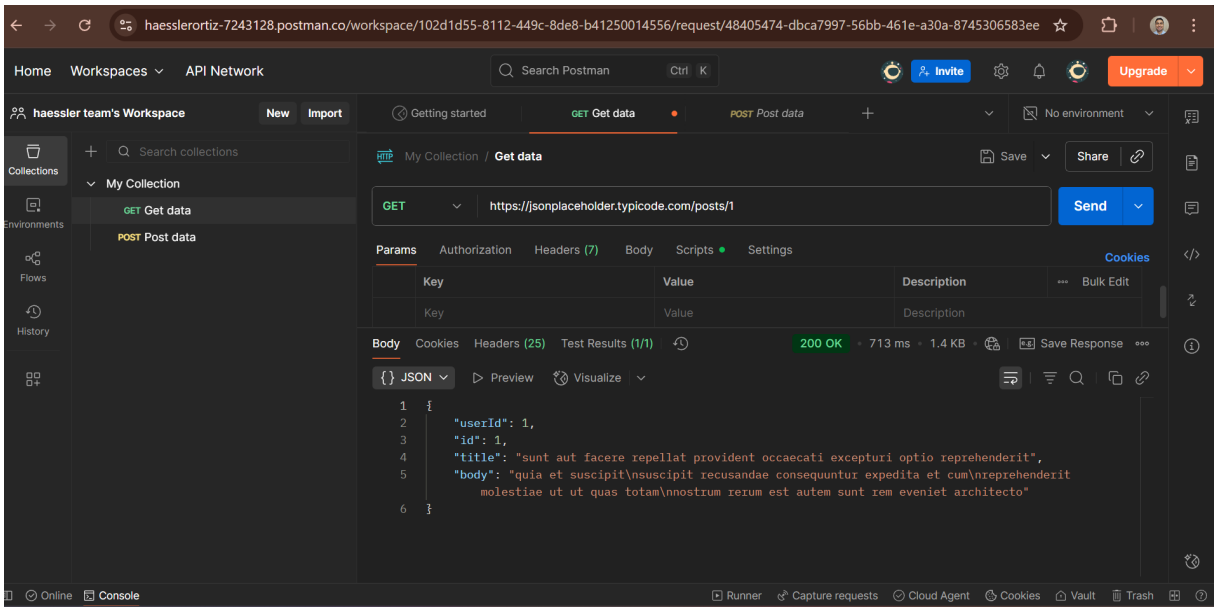

Figura 5: Solicitud GET.

La Figura 5 muestra el estado de la Transferencia de Estado Representacional (REST, por sus siglas en inglés) y el contenido del Objeto con Notación *JavaScript* (JSON, por sus siglas en inglés).

- **POST.** Diligenciando el *endpoint* y configurando el *body* del JSON (resaltado en amarillo en la Figura 6) a enviar, se obtuvo lo siguiente:

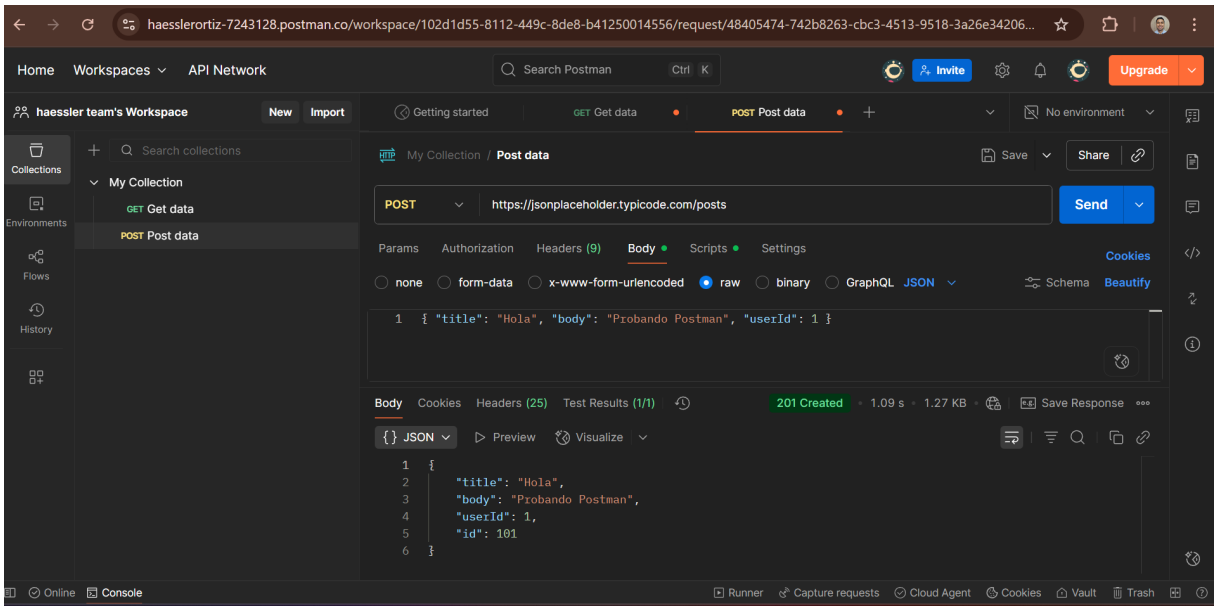

Figura 6: Solicitud POST.

- **PUT.** Con un procedimiento similar al anterior, se actualizó el JSON que en la solicitud anterior se había generado. La Figura 7 muestra el resultado:

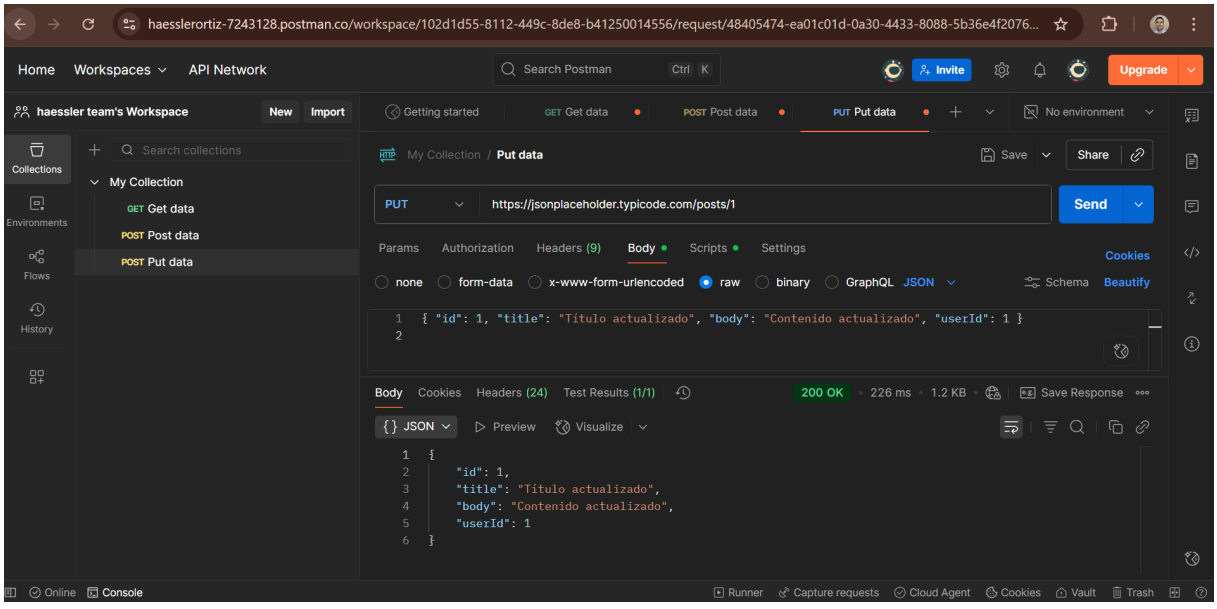

Figura 7: Solicitud PUT.

- **DELETE.** Aquí lo que se hizo fue borrar el registro que reciéntemente modificamos. Se tuvo el siguiente resultado:

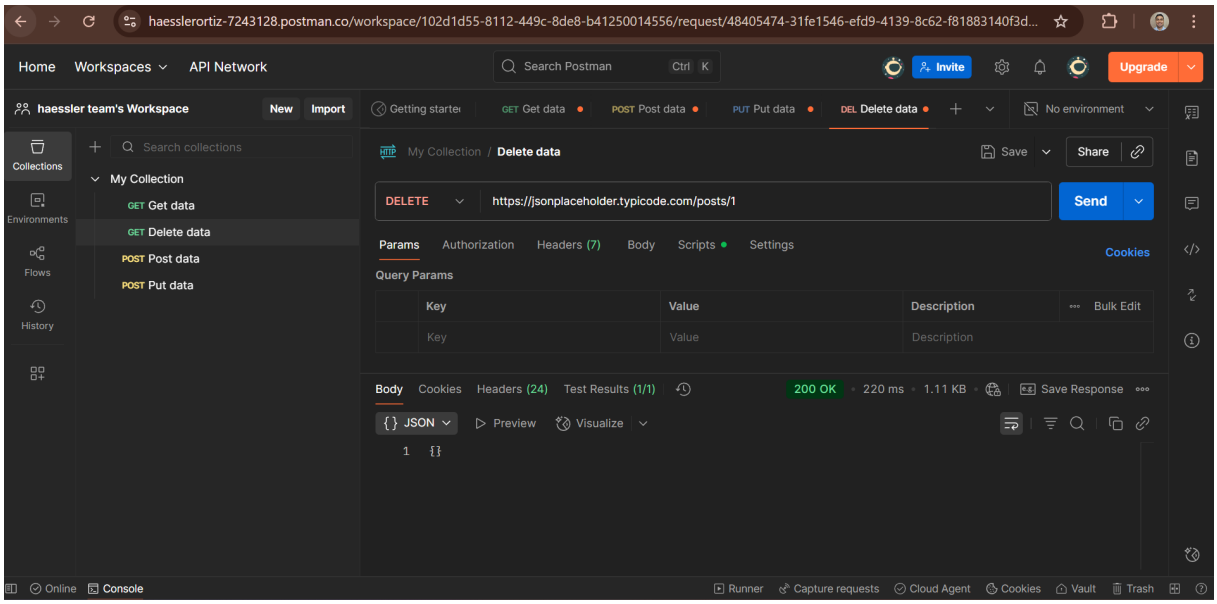

Figura 8: Solicitud DELETE.

La API aquí utilizada, fue sugerida por la Inteligencia Artificial (IA) **Gemini**.