Documentación del Proyecto PY\_TEST\_APISG

# **Estructura de carpetas y archivos**

## **.pytest\_cache**

Carpeta interna de pytest que guarda cache de ejecuciones.

## **.vscode**

Configuraciones de Visual Studio Code, contiene settings.json.

## **assets**

## Define la apariencia visual (colores, tamaños de fuente, márgenes, etc.) del reporte HTML.

## **config**

Configuración del proyecto, incluye endpoints.py y \_\_init\_\_.py.

## **data**

Archivos JSON con datos de prueba para los endpoints. Cada JSON contiene casos de entrada o validaciones específicas.

## **reportes**

Carpeta para almacenar resultados de pruebas o reportes

## **tests\_v2.0.3**

Contiene la suite de pruebas automatizadas organizadas por módulos de endpoints. Incluye subcarpetas: adyacente, callejero, datos\_utiles, geocoder, nombre\_calle, reverse, sitios\_interes, ubicaciones\_tecnicas.

## **conftest.py**

Archivo de configuración de Pytest, define fixtures y configuraciones globales.

## **README**

Archivo de documentación inicial del proyecto.

# **Detalle de carpetas clave**

## **config**

Contiene scripts de configuración:

- endpoints.py: Define los endpoints de la API a probar.

- \_\_init\_\_.py: Indica que la carpeta es un paquete de Python.

**Nota:** Contar con un archivo de configuración separado de los tests permite centralizar la definición de los endpoints de la API.

* **Reutilización**: todos los tests consumen una única fuente de verdad para las URLs.
* **Mantenibilidad**: si cambia un path o la BASE\_URL, se modifica en un solo lugar.
* **Escalabilidad**: facilita agregar nuevos endpoints o manejar distintos entornos (DEV, QA, PROD).
* **Consistencia**: evita errores por URLs hardcodeadas y asegura que todos los tests usen la misma configuración.
* **Claridad**: permite ver de forma rápida y ordenada qué servicios se están probando, funcionando además como documentación viva de la API.

## **data**

Incluye múltiples archivos JSON para validar distintos casos de prueba de los endpoints

**Nota:** Contar con una carpeta data aislada de los tests, donde se almacenan los archivos **JSON de entrada y validación**:

* **Separación de responsabilidades**: los tests se enfocan en la lógica de validación, mientras que los datos de prueba se mantienen aparte.
* **Reutilización**: los mismos archivos JSON pueden usarse en múltiples tests sin duplicación.
* **Mantenibilidad**: si cambian los datos de prueba, basta con actualizar el archivo correspondiente sin modificar la lógica del test.
* **Escalabilidad**: permite organizar los casos de prueba en distintos archivos según el endpoint o escenario.
* **Claridad y trazabilidad**: facilita identificar qué datos alimentan cada prueba y documenta los casos de manera explícita.

. Ejemplos:

- calles\_dada\_de\_baja.json: Casos de calles eliminadas.

- direccion\_adyacente.json: Validación de direcciones adyacentes.

- lugares\_reverse.json y lugares\_reverse\_errores.json: Casos positivos y negativos para /lugares/reverse.

- reverse\_coordenadas.json: Datos de prueba para /coordenadas/reverse.

- ubicaciones\_tecnicas\_direcciones.json: Validaciones de ubicaciones técnicas.

- Varios archivos 'metodo\_\*': Pruebas específicas de cada método de geocodificación.

## **tests\_v2.0.3**

Carpeta principal de pruebas automatizadas con pytest, estructurada por módulos de endpoints

* **Organización:** cada endpoint tiene su propio conjunto de tests, lo que facilita localizar y entender las pruebas relacionadas.
* **Escalabilidad:** al crecer la cantidad de endpoints o versiones de la API, se pueden añadir nuevos módulos sin desordenar el proyecto.
* **Mantenimiento:** si cambia la lógica de un endpoint, solo se ajustan las pruebas de su módulo correspondiente.
* **Colaboración:** distintos miembros del equipo pueden trabajar en pruebas de módulos separados sin interferir entre sí.
* **Trazabilidad:** es más sencillo relacionar cada conjunto de pruebas con el endpoint que validan.

- **adyacente:** Validación de calles adyacentes esperadas.

- **callejero:** Validaciones del callejero, estatus code 200 y la cantidad de datos existentes en la respuesta.

- **datos\_utiles**:

* Validación de atributos con y sin el parámetro v2,
* Compara respuesta con el parámetro dirección y coordenadas,
* Valida el valor correcto en el atributo polígono itinerario y distrito económico tanto para el parámetro direcciones como para el parámetro coordenadas y contempla errores ya corregidos.

- **geocoder:**

* Validación del correcto funcionamiento de ambiguar con el v2=true, v2=false y sin el v2 igual para el parámetro ambiguar.
* Validación para los métodos de geocodificacion correcto y solo en presencia del parámetro v2=true, este script interactua con todo los json en la carpeta data que contengan el nombre metodo al principio.
* Valida que la respuesta del servicio coincida con los datos del JSON para los campos: comuna, barrio, id\_via e id\_via\_usig (sin v2)(**esta validación es importante ya que ayuda no solo a tener una certeza de un buen funcionamiento sino de poder ubicar las puertas mal asignadas**)

y que NO incluya 'metodo\_geocodificacion

- **nombre\_calle:**

* Validación de calles apagadas tanto por el nombre como por el código de calle
* Validación de estructura de la respuesta para calles, calles con altura, cruces y calles atípicas; y respuesta de mensajes de error (alturas invalidas, consultas vacías, consultas con menos de 3 caracteres, consultas con signos)

- **reverse:** Validaciones de respuesta esperada y errores conocidos que generan status code 500.

- **sitios\_interes:**

Reverse:

* Valida mensajes de error cuando se coloca parámetros inválidos
* Valida filtrado de sitios de interés por id de sitio

- **ubicaciones\_tecnicas:**

* Validaciones de json esperado, calle normalizada, comuna, barrio correcto y nombre normalizado