

Hacemos un proyecto Big Data: Gestión de la web y Base de Datos de la Inmobiliaria MJ

Estudiantes María Correas Crespo, Judith Urbina Córdoba
Profesora Marta Martín Moreno
Asignatura Análisis de Datos en entornos Big Data

Índice

Ejercicio 1. Descripción y justificación de los servicios de AWS	3
1. Amazon S3 (Simple Storage Service)	3
2. Amazon RDS (Relational Database Service)	4
3. AWS Lambda	4
4. Amazon API Gateway	4
5. Amazon Virtual Private Cloud (VPC)	4
6. AWS Identity and Access Management (IAM)	4
Ejercicio 2. Pruebas de procesamiento de datos	5
1. ETL	5
1.1. Prueba: Carga de las viviendas en el bucket	5
1.2. Prueba: Carga de las viviendas en la base de datos Amazon RDS	5
2. Simulación actualización precios	6
2.1. Prueba: Generación de una muestra adecuada	6
2.2. Prueba: Modificación de los parámetros	6
3. Pruebas de Procesamiento de Datos Relacionadas con el correcto funcionamiento del API	7
3.1. Prueba: Correcta creación de usuarios	7
3.2. Prueba: Correcta obtención de usuarios	7
3.3. Prueba: Correcta actualización de usuarios	8
3.4. Prueba: Correcta eliminación de usuarios	8
3.5. Prueba: Correcta obtención de viviendas	9
3.6. Prueba: Correcta obtención de viviendas mediante filtros	9
3.7. Prueba: Correcta eliminación de viviendas	9
3.8. Prueba: Correcta creación de vivienda favorita de usuario	11
3.9. Prueba: Correcta obtención de viviendas favoritas de usuario	11
3.10. Prueba: Correcta eliminación de vivienda favorita de usuario	12
Ejercicio 3. Validación de los KPIs	13
1. KPIs Relacionados con el correcto funcionamiento del API	13
1.1. KPI: Tasa de éxito de las operaciones del API	13
2. KPIs Relacionados con el correcto funcionamiento de la base de Datos	13
2.1. KPI: Tasa de incidentes de datos	13
2.2. Tasa de disponibilidad de los datos	13
Ejercicio 4. Implementación de scripts o código relevante	14
1. Preprocesamiento de datos	14
2. Proceso ETL y actualización de precios	14
3. Política y roles de IAM	15
3.1. AWS Glue	16
3.2. AWS RDS	16
3.3. AWS CloudWatch Logs	16
3.4. AWS Glue	16
3.1. AWS RDS	17
3.2. AWS CloudWatch Logs	17
3.3. AWS API Gateway	17
4. SQL script de generación de tablas	17
5. Código de la función lambda	19
6. SWAGGER de la API	19

Introducción

Nuestro proyecto simula que somos una Inmobiliaria ubicada en Londres cuya infraestructura en cuanto a tecnología está migrando a la nube a través de los servicios de Amazon.

Ejercicio 1 Descripción y justificación de los servicios de AWS

El proyecto parte de los datos inmobiliarios de Londres almacenados en el archivo CSV de kaggle y consiste en procesarlos e integrarlos en una base de datos relacional RDS para consultas posteriores a través de funciones Lambda. Como subobjetivo, tenemos la necesidad del uso de PySpark durante el curso en el entorno de los servicios de Amazon. Esto lo conseguimos mediante la implementación de un trabajo en Amazon Glue Studio.

Evolución de la arquitectura

A continuación, mostramos las arquitecturas inicial > y final >, de nuestro proyecto y el modelo físico >, que hemos construido para la base de datos relacional. Después describimos los servicios seleccionados y ofrecemos su justificación.

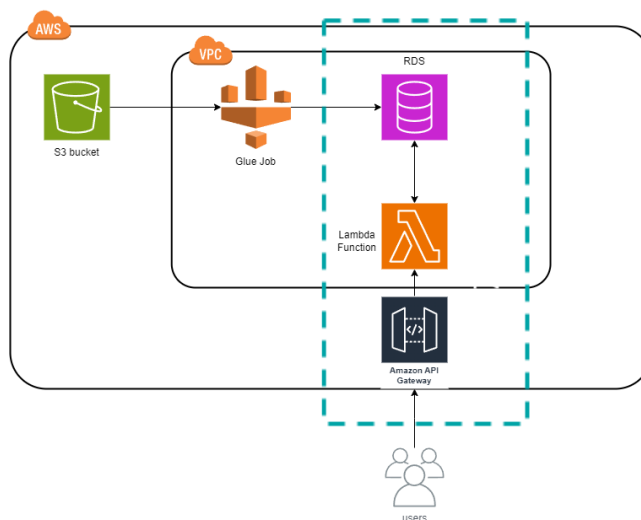


Figura 1: Arquitectura de la inmobiliaria de Londres final.

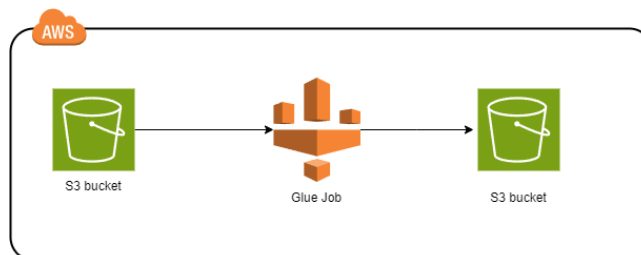


Figura 2: Arquitectura de la inmobiliaria de Londres inicial.

1. Amazon S3 (Simple Storage Service)

Justificación: Amazon S3 es ideal para almacenar grandes cantidades de datos estructurados o no estructurados de manera económica y escalable. En este caso, se utiliza para almacenar el archivo CSV inicial con los datos de las casas.

Rol en el flujo: Almacena el archivo london_houses.csv con los datos de propiedades inmobiliarias. Actúa como origen de datos para AWS Glue.

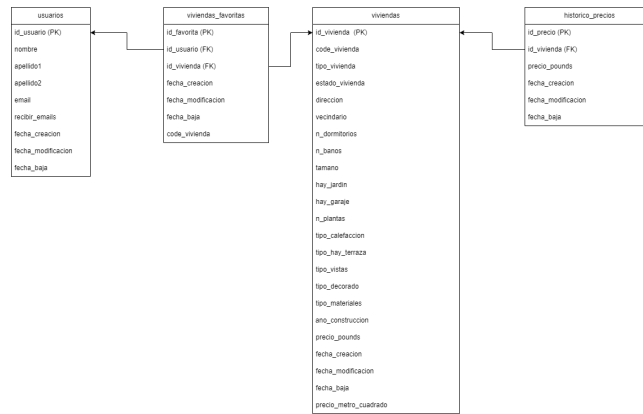


Figura 3: Diseño físico de la base de datos de la inmobiliaria.

2. Amazon RDS (Relational Database Service)

Justificación: Amazon RDS ofrece una base de datos gestionada que es confiable, escalable y fácil de usar. Es perfecta para alojar los datos transformados y permitir consultas rápidas y eficientes. En este caso, se elige MySQL como motor de base de datos debido a la buena integración con AWS Lambda, herramientas de análisis de datos y la capacidad de manejar datos estructurados como los requeridos en este proyecto.

Rol en el flujo: Almacena los datos inmobiliarios procesados de forma relacional. Sirve como origen para las consultas realizadas por la función Lambda.

3. AWS Lambda

Justificación: AWS Lambda permite ejecutar código en respuesta a eventos y es ideal para construir aplicaciones backend sin servidor. Es eficiente en costos y escalable.

Rol en el flujo: La lambda funciona como una REST-API con diferentes métodos: crear, actualizar, obtener y eliminar usuarios, obtener viviendas de acuerdo a filtros (por ejemplo, precio, número de habitaciones, ubicación), dar de baja viviendas o crear la relación vivienda favorita entre usuario y vivienda.

4. Amazon API Gateway

Justificación: API Gateway actúa como puerta de enlace para exponer las funciones Lambda como endpoints HTTP, permitiendo que los usuarios interactúen con el backend.


Rol en el flujo: Proporciona endpoints RESTful para las funcionalidades: búsqueda de casas, gestión de usuarios y casas favoritas. Maneja la autenticación y autorización para garantizar la seguridad por medio del uso de un API Key.

5. Amazon Virtual Private Cloud (VPC)

Justificación: Amazon VPC proporciona una red privada y aislada dentro de la nube de AWS, lo que garantiza un entorno seguro para servicios como RDS.

Rol en el flujo: En este proyecto, la VPC se utiliza para proteger la base de datos y limitar el acceso desde Internet.

6. AWS Identity and Access Management (IAM)

Justificación: Aunque no aparezca explícitamente en la arquitectura antes mostrada , AWS IAM actúa de manera subyacente en los servicios de Amazon. Esta proporciona control de acceso granular a los recursos y servicios de AWS. En este proyecto, IAM se utiliza para asignar los permisos mínimos necesarios a cada recurso, asegurando la seguridad del flujo de trabajo.

Ejercicio 2 Pruebas de procesamiento de datos

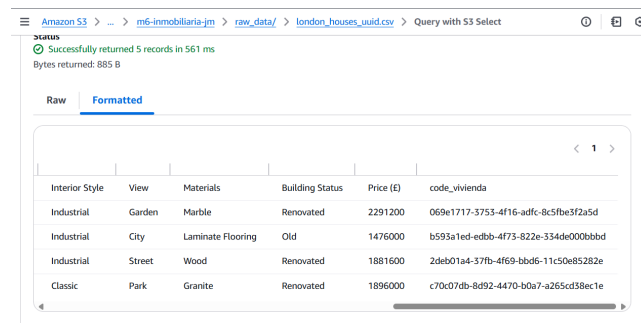
Adjuntamos los resultados de las pruebas de procesamiento de datos que hemos hecho.

1. ETL

A continuación mostramos las pruebas de procesamiento de datos relacionadas con la ejecución de la extracción, transformación y carga inicial de los registros de las viviendas.

1.1. Prueba: Carga de las viviendas en el bucket

Esta prueba mediante un S3 Query sobre el fichero en el Bucket elegido demuestra que se ha preprocesado correctamente el archivo y que en efecto está en el entorno CLOUD de Amazon.



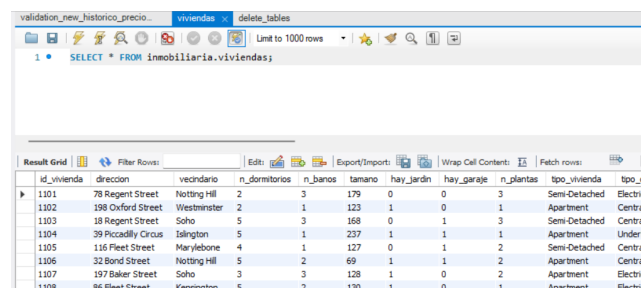
The screenshot shows the Amazon S3 Query interface. At the top, it indicates 'Successfully returned 5 records in 561 ms' and 'Bytes returned: 885 B'. Below this, there are tabs for 'Raw' and 'Formatted'. The 'Formatted' tab is selected, displaying a table with the following data:

Interior Style	View	Materials	Building Status	Price (£)	code_vivienda
Industrial	Garden	Marble	Renovated	2291200	069e1717-3753-4f16-adfc-8c5fbc3f2a5d
Industrial	City	Laminate Flooring	Old	1476000	b593a1ed-edbb-4f73-822e-334de000bbdd
Industrial	Street	Wood	Renovated	1881600	2deb01a4-37fb-4f69-bbd6-11c50e85282e
Classic	Park	Granite	Renovated	1896000	c70c07db-8d92-4470-b0a7-a265cd38ec1e

Figura 4: CSV de los datos preprocesados en el bucket.

1.2. Prueba: Carga de las viviendas en la base de datos Amazon RDS

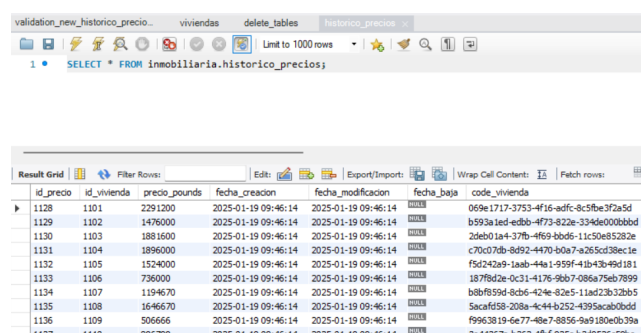
Mediante dos queries en el entorno MySQL Workbench se demuestra que las viviendas se han procesado y han hecho la carga inicial correctamente en la base de datos RDS en las tablas viviendas e historico precios.



The screenshot shows the MySQL Workbench interface with a query window containing the query: `SELECT * FROM inmobiliaria.viviendas;`. The 'Result Grid' tab is selected, displaying the following data:

id_vivienda	direccion	vecindario	n_dormitorios	n_banos	tamano	hay_jardin	hay_garaje	n_plantas	tipo_vivienda	tipo_c
1101	78 Regent Street	Notting Hill	2	3	179	0	0	3	Semi-Detached	Electric
1102	198 Oxford Street	Westminster	2	1	123	1	0	1	Apartment	Centra
1103	18 Regent Street	Soho	5	3	168	0	1	3	Semi-Detached	Centra
1104	39 Piccadilly Circus	Islington	5	1	237	1	1	1	Apartment	Underf
1105	116 Fleet Street	Maylebone	4	1	127	0	1	2	Semi-Detached	Centra
1106	32 Bond Street	Notting Hill	5	2	69	1	1	2	Apartment	Centra
1107	197 Baker Street	Soho	3	3	128	1	0	2	Apartment	Electric
1108	86 Fleet Street	Kensington	5	2	130	1	0	1	Apartment	Electric

Figura 5: Resultado de la query de consulta SELECT sobre la tabla viviendas.



The screenshot shows the MySQL Workbench interface with a query window containing the query: `SELECT * FROM inmobiliaria.historico_precios;`. The 'Result Grid' tab is selected, displaying the following data:

id_precio	id_vivienda	precio_pounds	fecha_creacion	fecha_modificacion	fecha_baja	code_vivienda
1128	1101	2291200	2025-01-19 09:46:14	2025-01-19 09:46:14		069e1717-3753-4f16-adfc-8c5fbc3f2a5d
1129	1102	1476000	2025-01-19 09:46:14	2025-01-19 09:46:14		b593a1ed-edbb-4f73-822e-334de000bbdd
1130	1103	1881600	2025-01-19 09:46:14	2025-01-19 09:46:14		2deb01a4-37fb-4f69-bbd6-11c50e85282e
1131	1104	1896000	2025-01-19 09:46:14	2025-01-19 09:46:14		c70c07db-8d92-4470-b0a7-a265cd38ec1e
1132	1105	1524000	2025-01-19 09:46:14	2025-01-19 09:46:14		f5d242a9-1a4b-44e1-959f-41b43b49d181
1133	1106	736000	2025-01-19 09:46:14	2025-01-19 09:46:14		1878b02e-0c31-4176-9b07-086a75eb7699
1134	1107	1194670	2025-01-19 09:46:14	2025-01-19 09:46:14		b0b7859d-8db6-424e-83a5-11a2d2b32b05
1135	1108	1646670	2025-01-19 09:46:14	2025-01-19 09:46:14		5acaf658-208a-4c44-b252-4395acab0b0d
1136	1109	505666	2025-01-19 09:46:14	2025-01-19 09:46:14		f9963819-6e77-4ae7-8856-9a9180eb39a
1137	1110	804700	2025-01-19 09:46:14	2025-01-19 09:46:14		7a4d3c7a-3767-48f7-875c-374097c6c0b9

Figura 6: Resultado de la query de consulta SELECT sobre la tabla historico precios.

2. Simulación actualización precios

Esta prueba consiste en la simulación de una nueva inserción de datos de casas a la base de datos en la que se incluía la actualización del precio de una de las casas que anteriormente ya estaban registradas en la inmobiliaria.

2.1. Prueba: Generación de una muestra adecuada

Mediante el remuestreo de las viviendas originales con un script de python y la posterior modificación manual de un registro, se ha generado una muestra de nuevas casas del tamaño de una décima parte del archivo original.

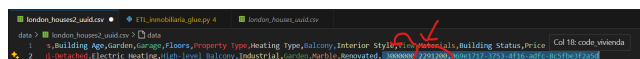


Figura 7: Muestra del valor original y al valor modificado del precio de la vivienda ya existente en la base de datos RDS.

2.2. Prueba: Modificación de los parámetros

Mediante la corrección del job parameter correspondiente al nombre del fichero y la ejecución del script del ETL en el entorno Amazon Glue Studio, se ha actualizado el precio en la base de datos inmobiliaria.

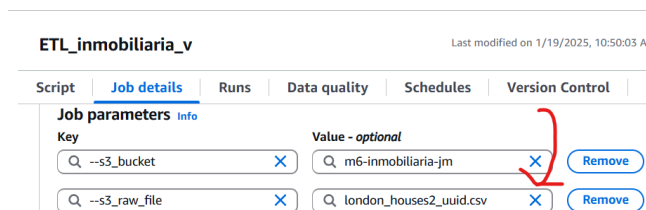


Figura 8: Parámetro modificado en los parámetros del trabajo del Amazon Glue Studio.

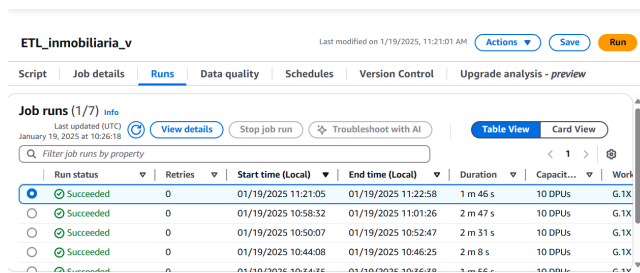


Figura 9: Visualización de las ejecuciones correctas del job.

Mostramos los resultados de las queries sobre la base de datos en MySQL Workbench.

lunds	fecha_creacion	fecha_modificacion	fecha_baja	ano_construccion	precio_metro_cuadrado	code_vivienda
1128	2025-01-19 09:46:08	2025-01-19 10:22:46	null	1953	16759.8	069e1717-3753-4f16-adfc-8c5fbc3f2a5d

Figura 10: Registro de la vivienda escogida después de la actualización en la tabla viviendas.

id_precio	id_vivienda	precio_pounds	fecha_creacion	fecha_modificacion	fecha_baja	code_vivienda
1128	1101	2291200	2025-01-19 09:46:14	2025-01-19 09:46:14	null	069e1717-3753-4f16-adfc-8c5fbc3f2a5d
2278	1101	3000000	2025-01-19 10:22:46	2025-01-19 10:22:46	null	069e1717-3753-4f16-adfc-8c5fbc3f2a5d

Figura 11: Registro de la vivienda escogida después de la actualización en la tabla historico precios.

3. Pruebas de Procesamiento de Datos Relacionadas con el correcto funcionamiento del API

Todas las pruebas están recogidas en la colección de Postman: Inmobiliaria.postman_collection.

3.1. Prueba: Correcta creación de usuarios

Esta prueba verifica que al enviar una solicitud POST al endpoint /all_users con datos válidos, el usuario es creado correctamente en la base de datos.

- **Entrada esperada:** Un cuerpo JSON con campos como nombre, apellido1, apellido2, email, recibir_emails.
- **Salida esperada:** Código de estado 201 y un JSON con la información del usuario creado.

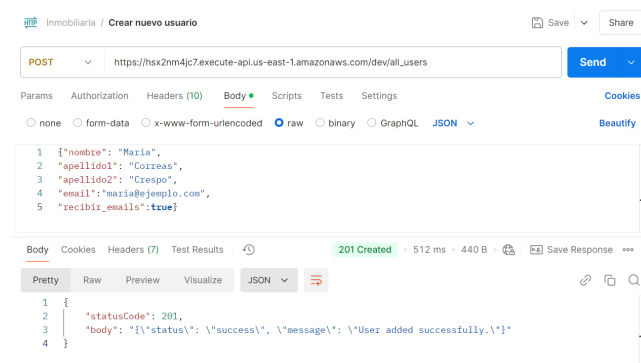


Figura 12: Petición para crear usuario.

3.2. Prueba: Correcta obtención de usuarios

Se prueba que al enviar una solicitud GET al endpoint /all_users o /user con un user_id válido, la API devuelve la información del usuario correspondiente.

- **Entrada esperada:** user_id (en parámetros de la ruta o consulta).
- **Salida esperada:** Código de estado 200 y un JSON con los datos del usuario.

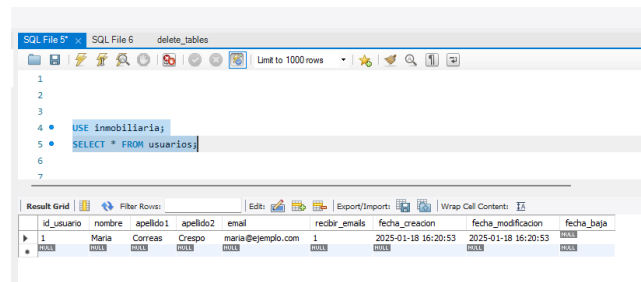


Figura 13: Usuario creado en la base de datos.

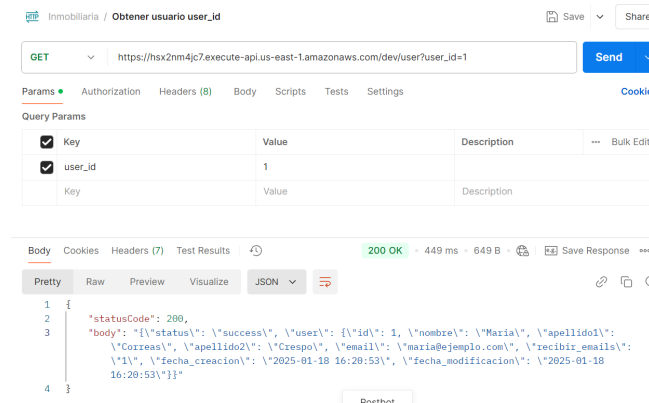


Figura 14: Petición para obtener usuario.

3.3. Prueba: Correcta actualización de usuarios

Esta prueba verifica que al enviar una solicitud PUT al endpoint /user con un user_id válido y datos actualizados, los cambios se reflejan correctamente en la base de datos.

- **Entrada esperada:** user_id y un cuerpo JSON con los datos a actualizar.
- **Salida esperada:** Código de estado 200 y un JSON confirmando los cambios realizados.

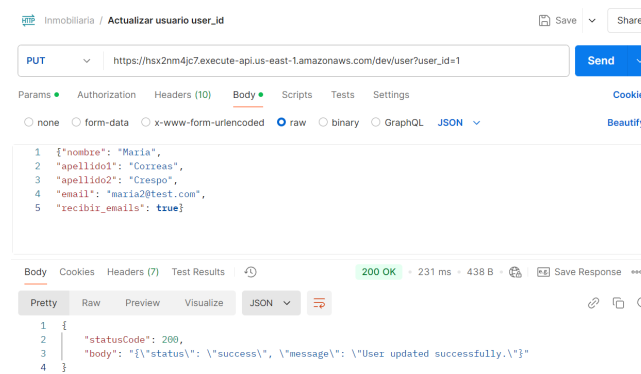


Figura 15: Petición para actualizar los datos del usuario.

3.4. Prueba: Correcta eliminación de usuarios

Se verifica que al enviar una solicitud DELETE al endpoint /user con un user_id válido, el usuario es eliminado correctamente.

- **Entrada esperada:** user_id.
- **Salida esperada:** Código de estado 200 y un JSON confirmando la eliminación.


```

3
4 • USE inmobiliaria;
5 • SELECT * FROM usuarios;
6
7

```

	id_usuario	nombre	apellido1	apellido2	email	recibir_email	fecha_creacion	fecha_modificacion	fecha_baja
1	1	Maria	Correas	Crespo	maria2@test.com	1	2025-01-18 16:20:53	2025-01-18 16:22:34	

Figura 16: Registro actualizado en la base de datos.

Inmobiliaria / Eliminar usuario user_id

DELETE https://hsx2nm4jc7.execute-api.us-east-1.amazonaws.com/dev/user?id=1 Send

Params Authorization Headers (8) Body Scripts Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
user_id	1		

Body Cookies Headers (7) Test Results 200 OK - 178 ms - 438 B Save Response

Pretty Raw Preview Visualize JSON

```

1 {
2   "statusCode": 200,
3   "body": {"status": "success", "message": "User deleted successfully."}
4 }

```

Figura 17: Petición para dar de bajo el usuario.

3.5. Prueba: Correcta obtención de viviendas

Esta prueba asegura que el endpoint `/all_properties` devuelve todas las viviendas almacenadas en la base de datos.

- **Entrada esperada:** Ninguna entrada específica requerida.
- **Salida esperada:** Código de estado 200 y un JSON con la lista completa de viviendas.

3.6. Prueba: Correcta obtención de viviendas mediante filtros

Se verifica que al usar el endpoint `/properties` con filtros (e.g., ubicación, rango de precios), los resultados devueltos cumplan con los criterios.

- **Entrada esperada:** Parámetros de filtro en la solicitud (e.g., tamaño, precio).
- **Salida esperada:** Código de estado 200 y un JSON con las viviendas que cumplen los filtros.

3.7. Prueba: Correcta eliminación de viviendas

Esta prueba asegura que al enviar una solicitud DELETE al endpoint `/properties` con un `property_id` válido, la vivienda es eliminada de la base de datos.

- **Entrada esperada:** `id_vivienda`.
- **Salida esperada:** Código de estado 200 y un JSON confirmando la eliminación.

```

3
4 • USE inmobiliaria;
5 • SELECT * FROM usuarios;
6
7

```

	id_usuario	nombre	apellido1	apellido2	email	recibir_email	fecha_creacion	fecha_modificacion	fecha_baja
1	1	Maria	Correas	Crespo	maria2@test.com	1	2025-01-18 16:20:53	2025-01-18 16:22:25	2025-01-18 16:23:25

Figura 18: Registro actualizado en la base de datos.

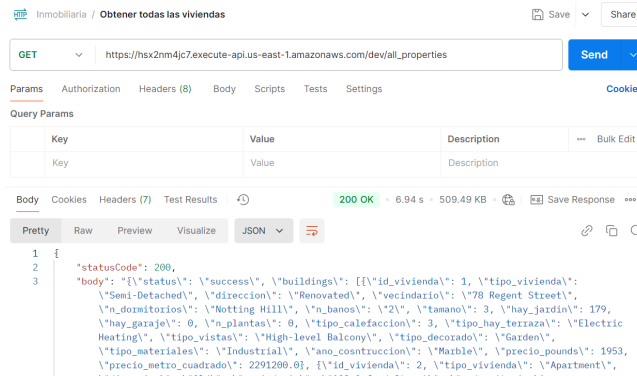


Figura 19: Petición para obtener las viviendas.

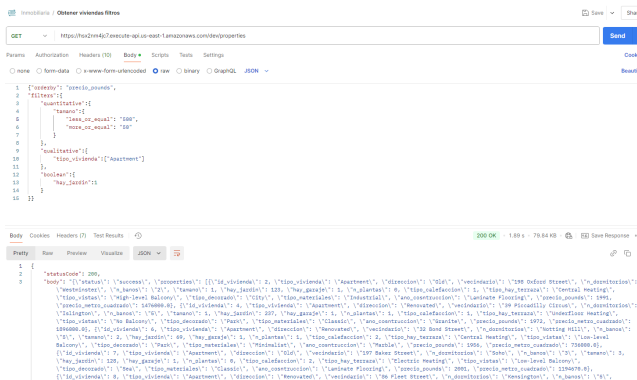
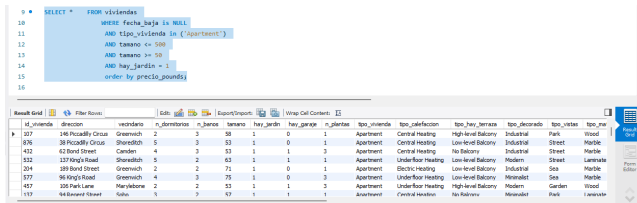


Figura 20: Petición para obtener las viviendas con filtros.



3.8. Prueba: Correcta creación de vivienda favorita de usuario

Se prueba que al usar el endpoint /favorite_properties para agregar una vivienda favorita, la API la registra correctamente.

- **Entrada esperada:** user_id y id_vivienda.
- **Salida esperada:** Código de estado 201 y un JSON confirmando la creación.

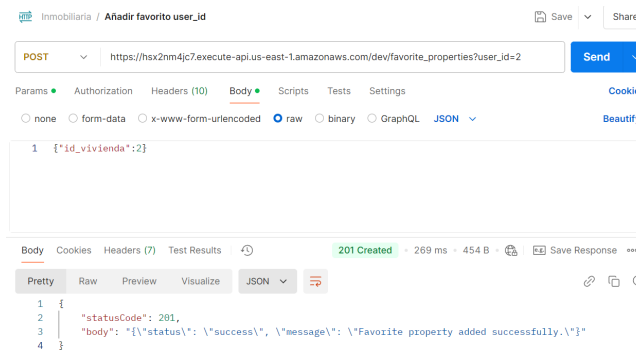


Figura 24: Petición para crear vivienda favorita para usuario.

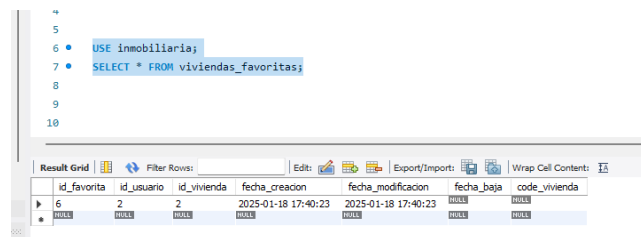


Figura 25: Registro en la base de datos de las viviendas favoritas

3.9. Prueba: Correcta obtención de viviendas favoritas de usuario

Esta prueba verifica que al enviar una solicitud GET al endpoint /favorite_properties, se devuelven correctamente las viviendas favoritas de un usuario.

- **Entrada esperada:** user_id.
- **Salida esperada:** Código de estado 200 y un JSON con las viviendas favoritas del usuario.

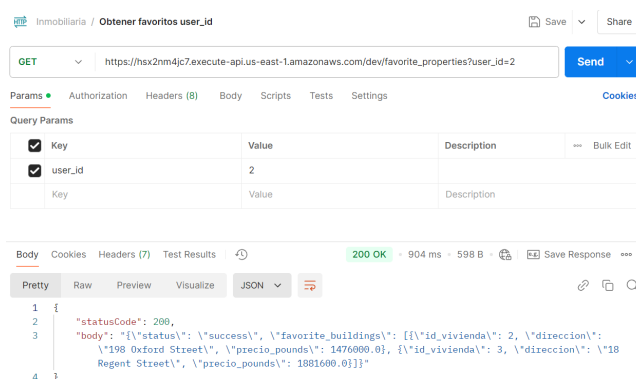


Figura 26: Petición para obtener las viviendas favoritas.

3.10. Prueba: Correcta eliminación de vivienda favorita de usuario

Se verifica que al enviar una solicitud PUT al endpoint `/favorite_properties` con un `user_id` y `id_vivienda`, la vivienda es eliminada correctamente de las favoritas del usuario.

- **Entrada esperada:** `user_id` y `id_vivienda`.
- **Salida esperada:** Código de estado 200 y un JSON confirmando la eliminación.

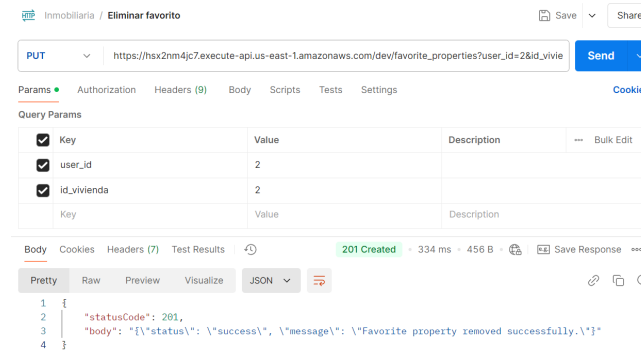


Figura 27: Petición para eliminar las vivienda favorita.

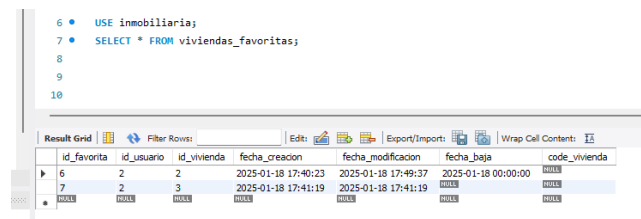


Figura 28: Registro en la base de datos con la vivienda favorita dada de baja.

Ejercicio 3 Validación de los KPIs

1. KPIs Relacionados con el correcto funcionamiento del API

1.1. KPI: Tasa de éxito de las operaciones del API

- **Definición:** Porcentaje de solicitudes que devuelven un código de estado exitoso (2xx) respecto al total de solicitudes realizadas.

- **Fórmula:**

$$\text{Tasa de éxito} = \left(\frac{\text{Número de respuestas 2xx}}{\text{Número total de solicitudes}} \right) \times 100$$

- **Objetivo:** Mantener una tasa de éxito superior al 95 %.
- **Objetivo Cumplido:** Sí.
- **Definición Informal:** Este KPI mide qué tan bien responde el API a las solicitudes que recibe. Es como si estuvieras revisando cuántas veces un cajero en una tienda atiende correctamente a los clientes sin errores ni retrasos. Una alta tasa de éxito indica que el API está funcionando como se espera y atendiendo las solicitudes de manera confiable.

2. KPIs Relacionados con el correcto funcionamiento de la base de Datos

2.1. KPI: Tasa de incidentes de datos

- **Definición:** Porcentaje de incidencias relacionadas con inexactitudes, violaciones o pérdida de datos respecto al número total de datos procesados. Por ejemplo, la recepción de un fichero CSV donde los códigos de las viviendas no estén correctamente señalizados o campos que no deberían estar vacíos aparezcan vacíos.

- **Fórmula:**

$$\text{Tasa de incidentes} = \left(\frac{\text{Número de incidentes reportados}}{\text{Número total de datos procesados}} \right) \times 100$$

- **Objetivo:** Mantener una tasa de incidentes inferior al 5 %.
- **Objetivo Cumplido:** No (se han reportado incidencias superiores al objetivo).
- **Definición Informal:** Este KPI mide cuántos problemas o errores han ocurrido con los datos en relación al total de datos procesados. Es como si una tienda revisara cuántos productos tienen etiquetas incorrectas o datos faltantes en comparación con el inventario total. Una baja tasa de incidentes indica que los datos son fiables y precisos.

2.2. Tasa de disponibilidad de los datos

- **Definición:** Porcentaje de tiempo en que el servidor de Amazon RDS está disponible y operativo para la API, garantizando que no haya interrupciones en el acceso a los datos.

- **Fórmula:**

$$\text{Tasa de disponibilidad} = \left(\frac{\text{Tiempo de actividad del servidor RDS}}{\text{Tiempo total monitorizado}} \right) \times 100$$

- **Métrica asociada:** Este KPI se calcula utilizando la métrica RDS Availability"de Amazon CloudWatch, que mide el tiempo de actividad del servidor RDS durante un período específico.
- **Objetivo:** Mantener una tasa de disponibilidad superior al 99.9 %.
- **Objetivo Cumplido:** Sí (el servicio ha mantenido una disponibilidad de 99.95 % en el último mes).
- **Definición Informal:** Este KPI mide si el servidor de datos está siempre disponible cuando la API lo necesita. Es como asegurarse de que la despensa de una tienda siempre esté abierta y lista para reponer productos en los estantes. Si el servidor falla, la API no puede acceder a los datos, lo que afecta directamente la experiencia de los usuarios.

Ejercicio 4 Incorporación de scripts o código relevante

A continuación detallaremos fragmentos de scripts relevantes para nuestro proyecto. Si se desea consultar todo el código en el siguiente repositorio privado se puede encontrar.

1. Preprocesamiento de datos

Tras descargar el fichero de datos original de kaggle mediante el siguiente enlace decidimos incorporar una nueva columna con identificadores universales. Estos simulan códigos de viviendas que ya vendrían predefinidos.

Aquí se visualiza parte del script en python.

Listing 1: Script de generación de identificadores únicos de las viviendas.

```
# Add UUIDs to the dataset
import uuid
import pandas as pd

# Read the original dataset
data = pd.read_csv('data\\london_houses.csv')

# Generate unique identifiers (UUIDs) for all rows
data['code_vivienda'] = [uuid.uuid4() for _ in range(len(data))]

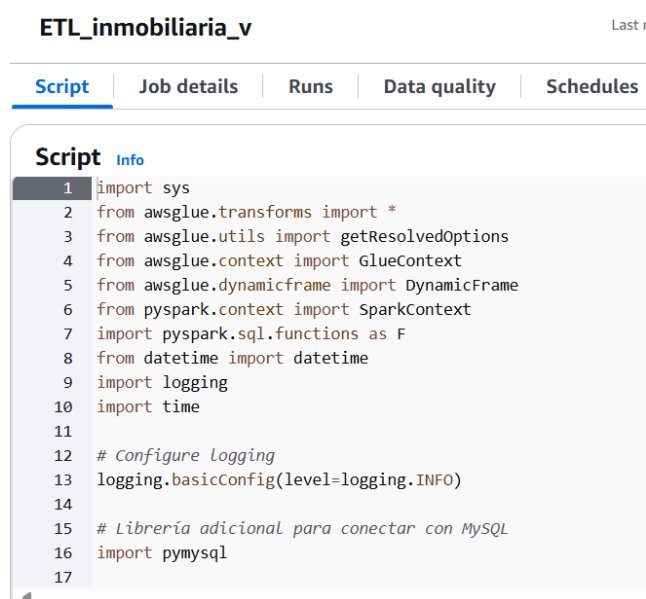
# Save the dataset with UUIDs to a CSV file
data.to_csv('data\\london_houses_uuid.csv', index=False)

print("The original data has been prepared with UUIDs.")
```

2. Proceso ETL y actualización de precios

Destacamos el archivo python que ejecuta desde la lectura, la transformación y la carga de los datos de las viviendas inicial a la base de datos RDS hasta la inserción de nuevos ficheros de datos tras la carga inicial y un registro de la actualización de precios en la tabla historico viviendas.

Las librerías necesarias para esta funcionalidad se muestran a continuación. Aparte, cabe destacar la



```
ETL_inmobiliaria_v Last r

Script Job details Runs Data quality Schedules

Script Info

1 import sys
2 from awsglue.transforms import *
3 from awsglue.utils import getResolvedOptions
4 from awsglue.context import GlueContext
5 from awsglue.dynamicframe import DynamicFrame
6 from pyspark.context import SparkContext
7 import pyspark.sql.functions as F
8 from datetime import datetime
9 import logging
10 import time
11
12 # Configure logging
13 logging.basicConfig(level=logging.INFO)
14
15 # Librería adicional para conectar con MySQL
16 import pymysql
17
```

Figura 29: Fragmento de código de la ETL y la importación agrupada.

incorporación de la librería pymysql como fichero comprimido en los parámetros del job en el entorno

de Amazon Glue Studio. Sin este, no podríamos conectar la base de datos RDS con el script de Glue en python.

The screenshot shows the 'Libraries' configuration section in Amazon Glue Studio. It includes a title 'Libraries' with an 'Info' link. Below the title are three input fields: 'Python library path' containing 's3://m6-inmobiliaria-jm/dependencies/pymysql_layer.zip', 'Dependent JARs path' which is empty, and 'Referenced files path' which is also empty.

Figura 30: Configuración de la librería adicional de python pymysql.

3. Política y roles de IAM

Recalcamos los dos documentos .json creados que describen la política y roles de IAM para los servicios de Amazon usados. Sin estos dos documentos, los servicios no reciben los permisos necesarios para poder ejecutar funciones como el job del ETL para Amazon Glue o poner en marcha la API de la inmobiliaria, para Amazon Lambda.

Listing 2: Primer fichero json, los permisos y roles de Glue

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3::<nombre-del-bucket-s3>",
        "arn:aws:s3:::<nombre-del-bucket-s3>/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "rds:DescribeDBInstances",
        "rds:Connect"
      ],
      "Resource": "arn:aws:rds:<region>:<account-id>:db:<nombre-de-la-base-de-datos>"
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "arn:aws:logs:<region>:<account-id>:log-group:/aws/glue/*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "glue:CreateJob",
        "glue>DeleteJob",

```

```

        "glue:UpdateJob",
        "glue:StartJobRun",
        "glue:GetJob",
        "glue:GetJobRun",
        "glue:GetTable",
        "glue:GetTables",
        "glue:BatchCreatePartition",
        "glue:BatchDeletePartition",
        "glue:GetPartition",
        "glue:GetPartitions",
        "glue:UpdateTable"
    ],
    "Resource": "*"
}
]
}

```

3.1. AWS Glue

En la política de IAM que se ha asociado a AWS Bucket se otorgan permisos de lectura (s3:GetObject) y de listado de objetos (s3:ListBucket) sobre el bucket S3 específico. Reemplaza <nombre-del-bucket-s3> con el nombre de tu bucket. escribir logs en CloudWatch.

3.2. AWS RDS

Respecto al acceso a RDS, se otorgan permisos para describir las instancias de RDS (rds:DescribeDBInstances) y conectar a la base de datos (rds:Connect). Reemplaza <nombre-de-la-base-de-datos> con el nombre de tu base de datos y ajusta la región y el ID de cuenta de AWS.

3.3. AWS CloudWatch Logs

Respecto al CloudWatch Logs, se permiten acciones necesarias para crear log groups, crear log streams y escribir eventos de log en CloudWatch (logs:CreateLogGroup, logs:CreateLogStream, logs:PutLogEvents). Esto es útil para que los Glue Jobs generen logs.

3.4. AWS Glue

Respecto al Glue, se permiten varias acciones, como crear, actualizar, ejecutar y obtener detalles de los Glue Jobs, así como trabajar con las tablas y particiones en el Glue Data Catalog.

Listing 3: Segundo fichero json, permisos y roles de Lambda

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "rds:DescribeDBInstances",
        "rds:Connect",
        "rds:ExecuteStatement"
      ],
      "Resource": "arn:aws:rds:<region>:<account-id>:db:<nombre-de-la-base-de-datos>"
    },
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue"
      ],

```



```

    "Resource": "arn:aws:secretsmanager:<region>:<account-id>:secret:<nombre-del-secret>/*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "logs:CreateLogGroup",
      "logs:CreateLogStream",
      "logs:PutLogEvents"
    ],
    "Resource": "arn:aws:logs:<region>:<account-id>:log-group:/aws/lambda/*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "apigateway:GET",
      "apigateway:POST",
      "apigateway:PUT",
      "apigateway:DELETE"
    ],
    "Resource": "arn:aws:apigateway:<region>::/restapis/*"
  }
]
}

```

3.1. AWS RDS

Respecto al acceso a RDS, se otorgan permisos para interactuar con RDS: Se otorgan permisos de descripción (rds:DescribeDBInstances), conexión (rds:Connect) y ejecución de sentencias SQL (rds:ExecuteStatement) para interactuar con la base de datos. Asegúrate de reemplazar <nombre-de-la-base-de-datos> con el nombre de tu base de datos y ajustar la región y el ID de cuenta..

3.2. AWS CloudWatch Logs

Respecto al CloudWatch Logs, se dan permisos para las acciones necesarias para crear un log group, crear un log stream y escribir eventos de log en CloudWatch. Estos permisos son necesarios para que la función Lambda registre las métricas de ejecución. Se usa la ruta /aws/lambda/* para indicar que se aplica a todas las funciones Lambda en la cuenta.

3.3. AWS API Gateway

Respecto a la API Gateway, se permiten las acciones para hacer peticiones HTTP (GET, POST, PUT, DELETE) a cualquier API de API Gateway en la región. Esto es útil si la Lambda va a realizar alguna operación relacionada con una API Gateway (por ejemplo, invocar API Gateway o interactuar con recursos de una API).

4. SQL script de generación de tablas

Incorporamos el script de generación de las tablas en la base de datos inmobiliaria. Este script se ejecutaría en modo local. En particular, recomendamos MySQL Workbench, y tras haber hecho la conexión con la base de datos de Amazon RDS. Su relevancia ya es autodescrita, pero en resumen sirve para entender configuraciones específicas de los atributos de las dimensiones y las relaciones entre los hechos y dimensiones más allá de lo que se puede visualizar en el modelo físico. Por ejemplo, podemos ver que el id de las viviendas es autoincremental.

Listing 4: SQL script de generación de las tablas.

```

# Script que crea las tablas de la base de datos
USE inmobiliaria;

```

```

# Creamos la tabla usuarios
CREATE TABLE IF NOT EXISTS usuarios (
    id_usuario INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(255),
    apellido1 VARCHAR(255),
    apellido2 VARCHAR(255) NULL,
    email VARCHAR(255),
    recibir_emails BOOLEAN,
    fecha_creacion DATETIME DEFAULT CURRENT_TIMESTAMP,
    fecha_modificacion DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    fecha_baja DATETIME NULL
);

# Creamos la tabla viviendas
CREATE TABLE IF NOT EXISTS viviendas (
    id_vivienda INT AUTO_INCREMENT PRIMARY KEY,
    direccion VARCHAR(255),
    vecindario VARCHAR(255),
    n_dormitorios INT,
    n_banos INT,
    tamano INT,
    hay_jardin BOOLEAN,
    hay_garaje BOOLEAN,
    n_plantas INT,
    tipo_vivienda VARCHAR(50),
    tipo calefaccion VARCHAR(50),
    tipo_hay_terraza VARCHAR(50) NULL,
    tipo_decorado VARCHAR(50) NULL,
    tipo_vistas VARCHAR(50) NULL,
    tipo_materiales VARCHAR(50),
    estado_vivienda VARCHAR(50),
    precio_pounds FLOAT,
    fecha_creacion DATETIME DEFAULT CURRENT_TIMESTAMP,
    fecha_modificacion DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    fecha_baja DATETIME NULL,
    ano_construccion INT,
    precio_metro_cuadrado FLOAT,
    code_vivienda VARCHAR(255)
);

# Creamos la tabla viviendas_favoritas
CREATE TABLE IF NOT EXISTS viviendas_favoritas (
    id_favorita INT AUTO_INCREMENT PRIMARY KEY,
    id_usuario INT,
    id_vivienda INT,
    fecha_creacion DATETIME DEFAULT CURRENT_TIMESTAMP,
    fecha_modificacion DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    fecha_baja DATETIME NULL,
    code_vivienda VARCHAR(255),
    FOREIGN KEY (id_usuario) REFERENCES usuarios(id_usuario) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (id_vivienda) REFERENCES viviendas(id_vivienda) ON DELETE CASCADE ON UPDATE CASCADE
);

# Creamos la tabla historico_precios
CREATE TABLE IF NOT EXISTS historico_precios (
    id_precio INT AUTO_INCREMENT PRIMARY KEY,
    id_vivienda INT,

```

```

precio_pounds FLOAT,
fecha_creacion DATETIME DEFAULT CURRENT_TIMESTAMP,
fecha_modificacion DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
fecha_baja DATETIME NULL,
code_vivienda VARCHAR(255),
FOREIGN KEY (id_vivienda) REFERENCES viviendas(id_vivienda) ON DELETE CASCADE ON UPDATE CASCADE
);

```

5. Código de la función lambda

El código de la función lambda viene recogido en cuatro ficheros distintos:

- `lambda_function.py`: gestiona la recepción de eventos por parte de la lambda
- `user_functions.py`: en este script se recogen todas las funciones de los métodos relacionados con la gestión de usuarios
- `properties_functions.py`: en este script se recogen todas las funciones de los métodos relacionados con la gestión de las viviendas.
- `favorite_functions.py`: en este script se recogen todas las funciones de los métodos relacionados con la gestión de las viviendas favoritas.

Dichos ficheros de código se alojan en la ruta: `code/lambda/` Además, para el correcto funcionamiento de la lambda fue necesario crear una layer con la librería `pymysql`.



Figura 31: Layer para lambda con librería python pymysql.

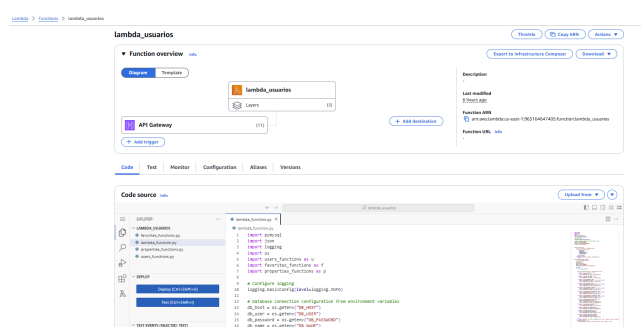


Figura 32: Función lambda creada.

6. SWAGGER de la API

Añadimos una sección del fichero Swagger de la API que hemos creado mediante el servicio Lambda. Su importancia radica en que sirve de documentación técnica para las interacciones de los usuarios con la página web de la inmobiliaria y que respeta las especificaciones estandarizadas de las APIs. Este fichero se encuentra en la ruta `doc/User and Properties API-dev-swagger.yaml`

Listing 5: Swagger de la API en un fichero yaml

```

openapi: "3.0.1"
info:

```

```

  title: "User and Properties API"
  description: "REST API for managing users, favorite properties, and property listings."
  version: "1.0.0"
servers:
- url: "https://hsx2nm4jc7.execute-api.us-east-1.amazonaws.com/{basePath}"
  variables:
    basePath:
      default: "dev"
paths:
  /all_properties:
    get:
      responses:
        "200":
          description: "200 response"
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/ArrayOfMODEL94b96c"
  /user:
    get:
      parameters:
        - name: "user_id"
          in: "query"
          required: true
          schema:
            type: "string"
      responses:
        "200":
          description: "200 response"
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/MODEL76e69"
    put:
      parameters:
        - name: "user_id"
          in: "query"
          required: true
          schema:
            type: "string"
      requestBody:
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/MODELbdad0d"
          required: true
      responses:
        "200":
          description: "200 response"
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/MODELd1898c"
    delete:
      parameters:
        - name: "user_id"
          in: "query"
          required: true

```

```

    schema:
      type: "string"
  responses:
    "200":
      description: "200 response"
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/MODEL20678"
/all_users:
  get:
    responses:
      "200":
        description: "200 response"
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/ArrayOfMODELcf5cf4"
  post:
    requestBody:
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/MODEL2dd992"
      required: true
    responses:
      "201":
        description: "201 response"
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/MODEL126cc6"
/properties:
  get:
    requestBody:
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/MODEL3fdbf7"
      required: true
    responses:
      "200":
        description: "200 response"
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/ArrayOfMODEL34e674"
/property:
  get:
    parameters:
      - name: "property_id"
        in: "query"
        required: true
        schema:
          type: "string"
  delete:
    parameters:
      - name: "property_id"

```

```

      in: "query"
      required: true
      schema:
        type: "string"
/ favorite_properties:
get:
  parameters:
  - name: "user_id"
    in: "query"
    required: true
    schema:
      type: "string"
  responses:
    "200":
      description: "200 response"
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/ArrayOfMODELd6bcb1"
put:
  parameters:
  - name: "user_id"
    in: "query"
    required: true
    schema:
      type: "string"
  - name: "id_vivienda"
    in: "query"
    required: true
    schema:
      type: "string"
  responses:
    "201":
      description: "201 response"
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/MODELb7cb55"
post:
  parameters:
  - name: "user_id"
    in: "query"
    required: true
    schema:
      type: "string"
  requestBody:
    content:
      application/json:
        schema:
          $ref: "#/components/schemas/MODEL7de55e"
    required: true
  responses:
    "201":
      description: "201 response"
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/MODEL87ba46"

```

```
components:
  schemas:
    MODEL7de55e:
      type: "object"
    MODELb7cb55:
      type: "object"
    MODELe20678:
      type: "object"
    MODEL2dd992:
      type: "object"
    MODEL126cc6:
      type: "object"
    MODELe76e69:
      type: "object"
    ArrayOfMODELd6bcb1:
      type: "array"
      items:
        type: "object"
    ArrayOfMODEL94b96c:
      type: "array"
      items:
        type: "object"
    MODEL3fdbf7:
      type: "object"
    MODELbdad0d:
      type: "object"
    ArrayOfMODEL34e674:
      type: "array"
      items:
        type: "object"
    MODEL87ba46:
      type: "object"
    ArrayOfMODELcf5cf4:
      type: "array"
      items:
        type: "object"
    MODELd1898c:
      type: "object"
```