Documentación de la API de propiedades para la prueba técnica de desarrollador back end de Propital.

28/06/2023

API LUIS EDUARDO CONDE PINTO

PROPITAL

DESARROLLADA EN C# .NET CORE 6.0

Base de datos: MongoDb en Atlas

La API de Consulta de Propiedades de Propital es una solución diseñada para proporcionar a los usuarios acceso a una amplia base de datos de propiedades inmobiliarias. Esta API permite a los desarrolladores y usuarios obtener información detallada sobre propiedades residenciales y comerciales, incluyendo características, ubicación, precios y más.

Con la API de Consulta de Propiedades, los usuarios pueden realizar consultas personalizadas y obtener resultados precisos y actualizados. Esta API es especialmente útil para aplicaciones y sitios web relacionados con el mercado inmobiliario, agentes de bienes raíces y empresas que desean integrar información inmobiliaria en sus servicios.

Características principales:

Búsqueda de propiedades por ubicación, tipo, precio, tamaño y otras características.
Acceso a detalles completos de propiedades, incluyendo descripciones,etc….
Filtros avanzados para refinar y personalizar las consultas de búsqueda.
La API de Consulta de Propiedades está diseñada para ser fácilmente integrada en aplicaciones y sitios web mediante el uso de solicitudes HTTP y respuestas en formato JSON. Se proporcionan ejemplos de código y documentación detallada para facilitar la implementación de la API.

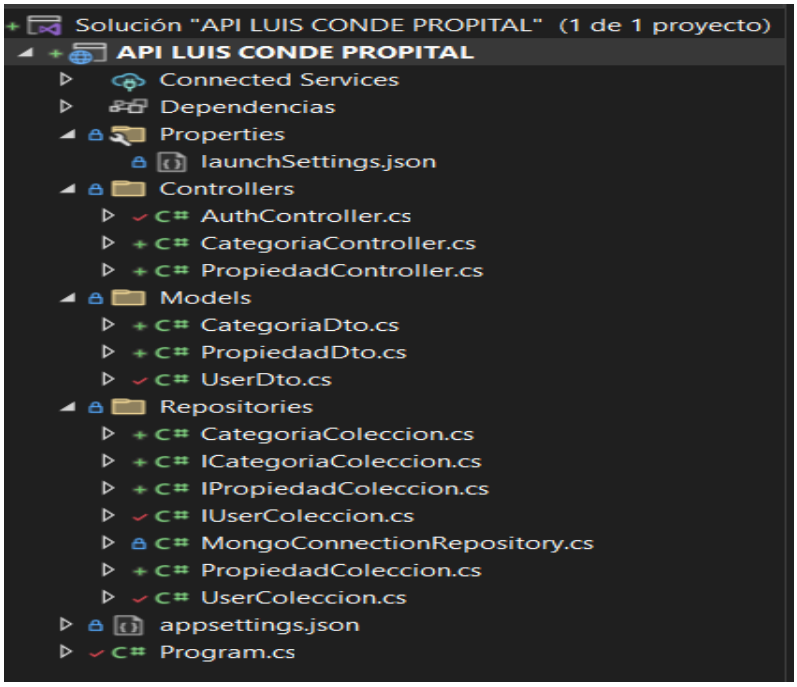
Audiencia:

La API de Consulta de Propiedades de Propital está dirigida a desarrolladores de software, equipos de desarrollo de aplicaciones y empresas relacionadas con el sector inmobiliario. Los usuarios objetivo incluyen:

Desarrolladores interesados en integrar información inmobiliaria en aplicaciones y sitios web.
Agentes de bienes raíces y profesionales del sector inmobiliario que buscan acceder a una amplia base de datos de propiedades.
Empresas de tecnología relacionadas con el mercado inmobiliario que desean enriquecer sus productos y servicios con datos inmobiliarios actualizados.
La API de Consulta de Propiedades de Propital proporciona una solución potente y eficiente para acceder y aprovechar información valiosa sobre propiedades inmobiliarias, brindando a los usuarios la capacidad de ofrecer servicios más completos y personalizados a sus clientes.

Estructura de la API

Como podemos ver en la siguiente imagen,usamos un patrón MVC.



Repositorio MongoConnectionRepository

En esta clase es donde realizamos la conexión con la base de datos de mongo DB:

```
7 referencias
public class MongoConnectionRepository
{

    public MongoClient clienteMongo;


    public IMongoDatabase db;

    3 referencias
    public MongoConnectionRepository()
    {

        try
        {
            clienteMongo = new MongoClient("mongodb+srv://luisado:Condeeiemlda123@cluster0.abcj6sj.mongodb.net/?retryWrites=true&w=majority");
            db = clienteMongo.GetDatabase("Luisconde");

            // Resto del código para trabajar con la base de datos

            Console.WriteLine("Conexión exitosa a la base de datos");
        }
        catch (Exception ex)
        {
            Console.WriteLine("Error en la conexión a la base de datos: " + ex.Message);
            // Aquí puedes realizar alguna acción adicional en caso de que la conexión falle, como mostrar un mensaje de error al usuario o real
        }

    }
}
```

Métodos de la API

(esta API la he creado con varios métodos que están como un añadido extra como por ejemplo el de crear usuarios,crear categorías y crear propiedades,estos métodos funcionan perfectamente y los pueden consumir desde postman.
Sin embargo los principales que son consumidos por el front end son GetAllPropiedades y GetAllCategorias pero repito ,todos los métodos se pueden usar y para finalidades practicas,todos los métodos irán documentados en este documento).
(otro extra,esta api tiene implementada autenticación con JWT usando el token en bearer junto a validación de rol,ya que no se pedía esto en el enunciado del problema técnico,el data notation [Authorize] esta comentado en los métodos para que no sea necesario validar token]

Métodos de la API en uso(más importantes)

tener en cuenta el el localhost y el puerto,adaptar para la maquina donde se desee ejecutar
tener en cuenta el el localhost y el puerto,adaptar para la maquina donde se desee ejecutar
tener en cuenta el el localhost y el puerto,adaptar para la maquina donde se desee ejecutar
tener en cuenta el el localhost y el puerto,adaptar para la maquina donde se desee ejecutar

G**etAllPropiedades()**

la primera parte de la url depende de cada equipo local donde se ejecute y su puerto correspondiente

Descripción

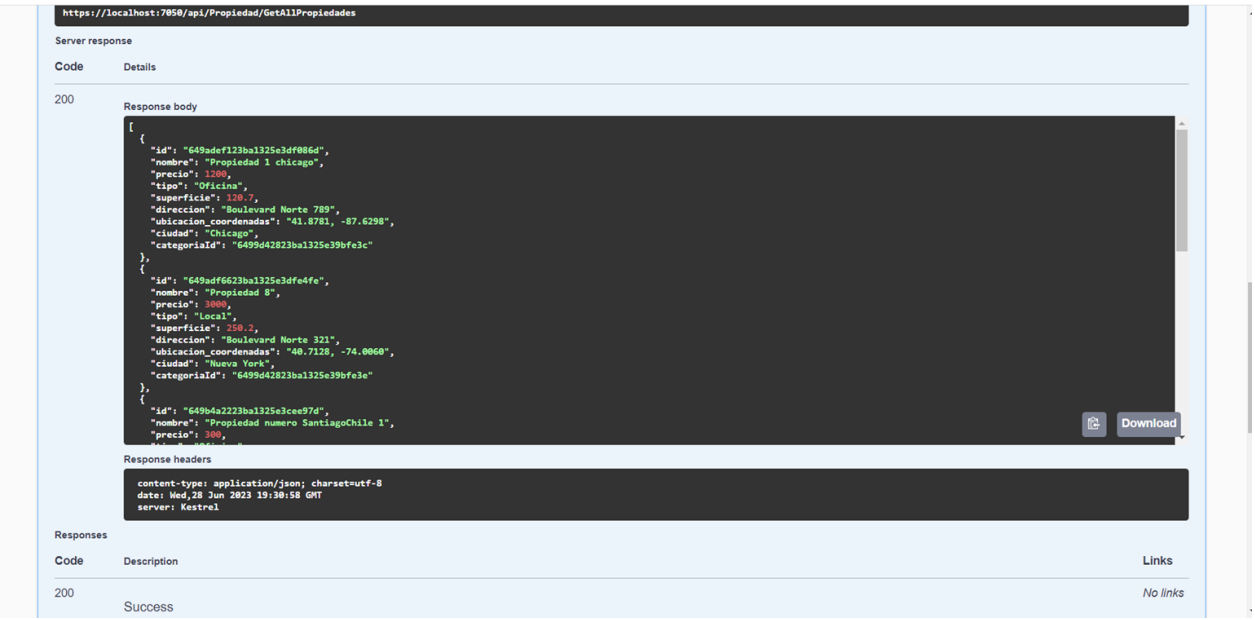
Consulta todas las propiedades existentes en la base de datos.

1. Endpoint: https://localhost:7050/api/Propiedad/GetAllPropiedades

2. **MetodoHttp:** GET

3. Parametros: no requiere parámetros

4. Retorna:



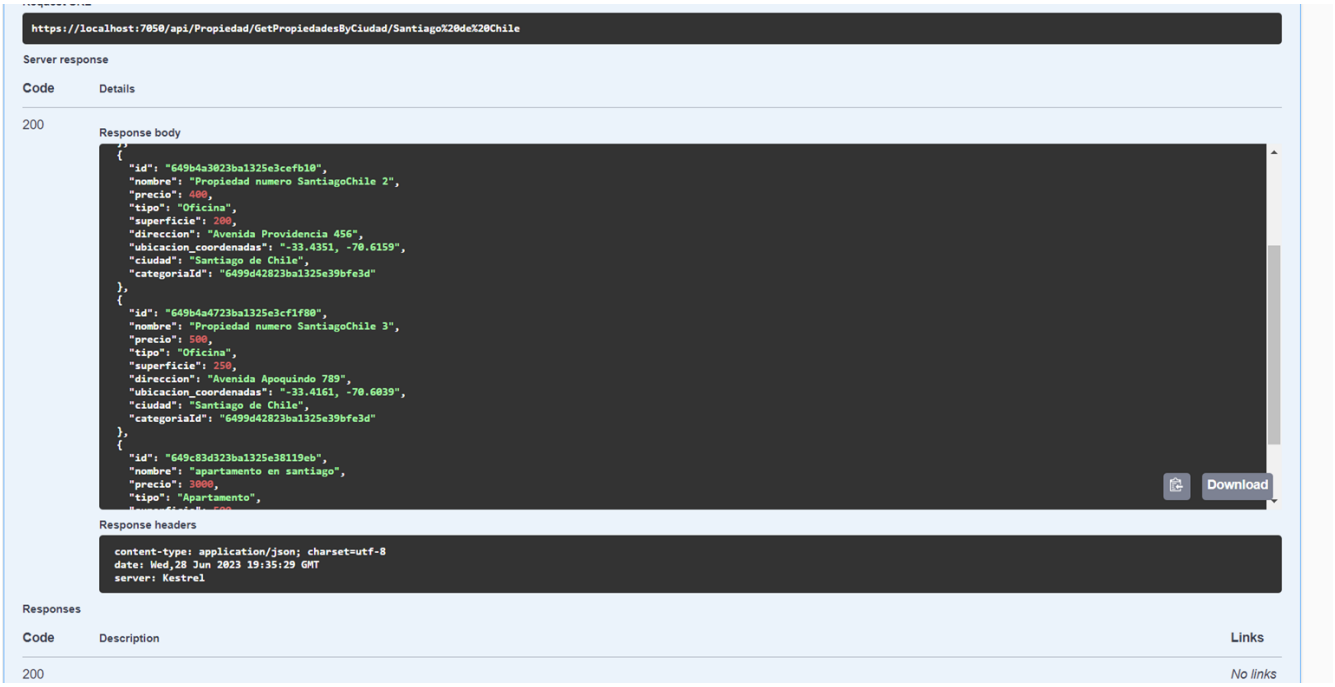
GetAllPropiedadesByCiudad()

1.Endpoint: https://localhost:7050/api/Propiedad/GetPropiedadesByCiudad

2.Metodo Http: GET

3.Parametros: {nombreCiudad} pdta: se deben respetar los espacios:
ejemplo:
https://localhost:7050/api/Propiedad/GetPropiedadesByCiudad/Chicago

4. Retorna:



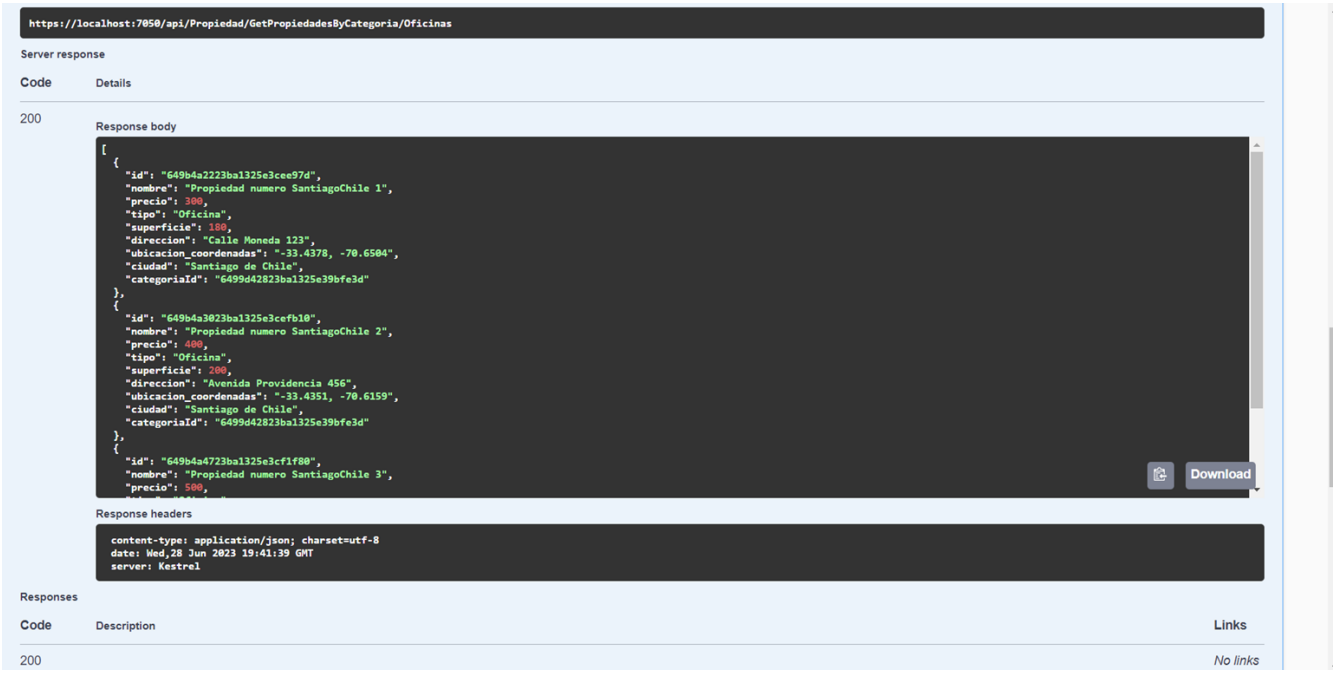
GetPropiedadesByCategoria()

1.Endpoint: https://localhost:7050/api/Propiedad/GetPropiedadesByCategoria/

2.Metodo Http: GET

3.Parametros: {nombreOficina}
ejemplo: https://localhost:7050/api/Propiedad/GetPropiedadesByCategoria/Oficinas

4. Retorna:



Métodos CRUD Propiedades
(el Read o consultar es el GetAllPropiedades que esta arriba)
Estos métodos son un aporte de valor por mi parte para facilitar la creación ,edición y eliminación de propiedades puesto que la base de datos esta en atlas y es privada

Crear-→ CreatePropiedad

1. EndPoint: https://localhost:7050/api/Propiedad/CreatePropiedad
2. Método Http: POST
3. **Parametros:**

Example Value | Schema


```
{
  "id": "string",
  "nombre": "string",
  "precio": 0,
  "tipo": "string",
  "superficie": 0,
  "direccion": "string",
  "ubicacion_coordenadas": "string",
  "ciudad": "string",
  "categoriaId": "string"
}
```

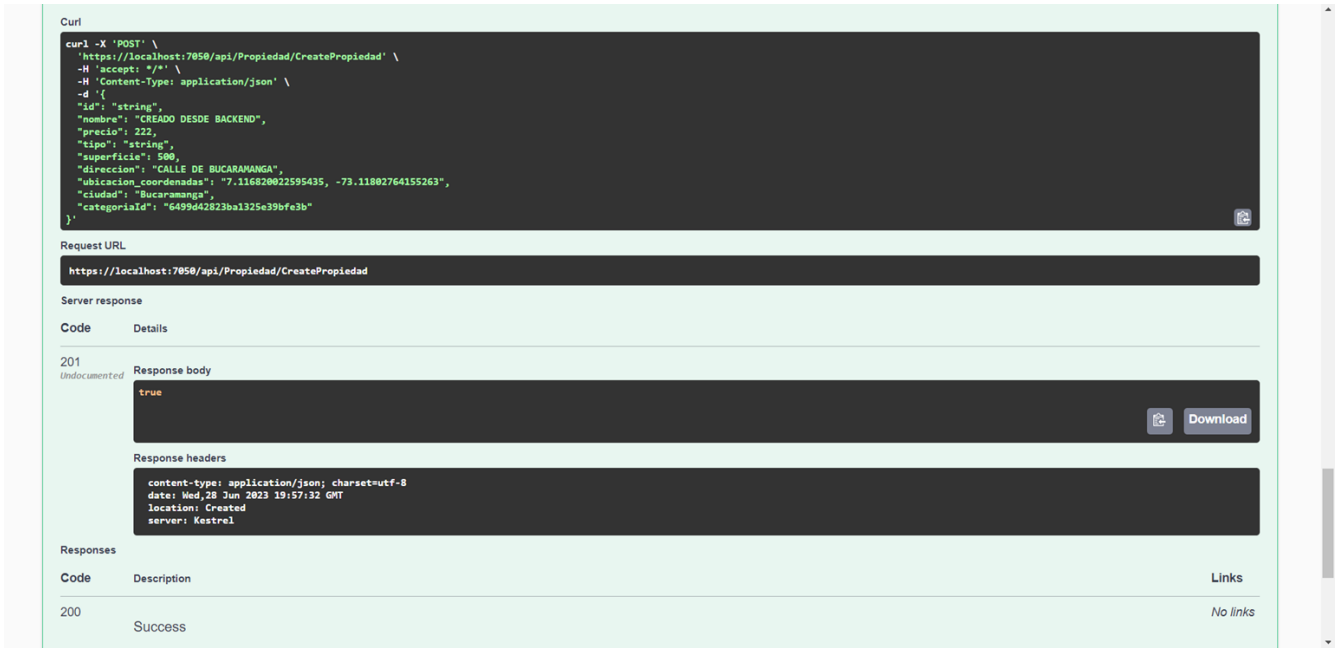

el valor de la llave “id” puede ir con cualquier valor de cadena,es indiferente ya que en el la lógica del método le estoy creando un nuevo id de mongo automáticamente

4: Prueba de creación

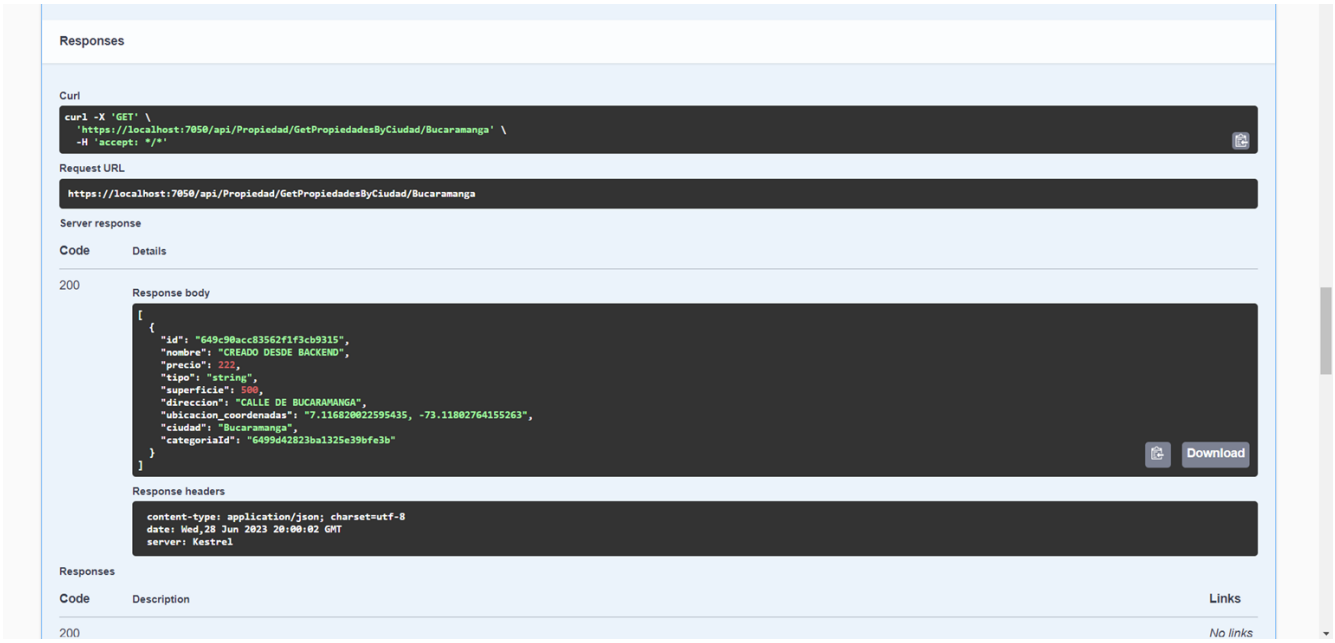
JSON:


```
{
  "id": "string",
  "nombre": "CREADO DESDE BACKEND",
  "precio": 222,
  "tipo": "string",
  "superficie": 500,
  "direccion": "CALLE DE BUCARAMANGA",
  "ubicacion_coordenadas": "7.116820022595435, -73.11802764155263",
  "ciudad": "Bucaramanga",
  "categoriaId": "6499d42823ba1325e39bfe3b"
}
```


Retorna:



Así se ve el documento creado :



Eliminar-→DeletePropiedad()

1. Endpoint: https://localhost:7050/api/Propiedad/DeletePropiedadad/

2.**Metodo Http:** POST

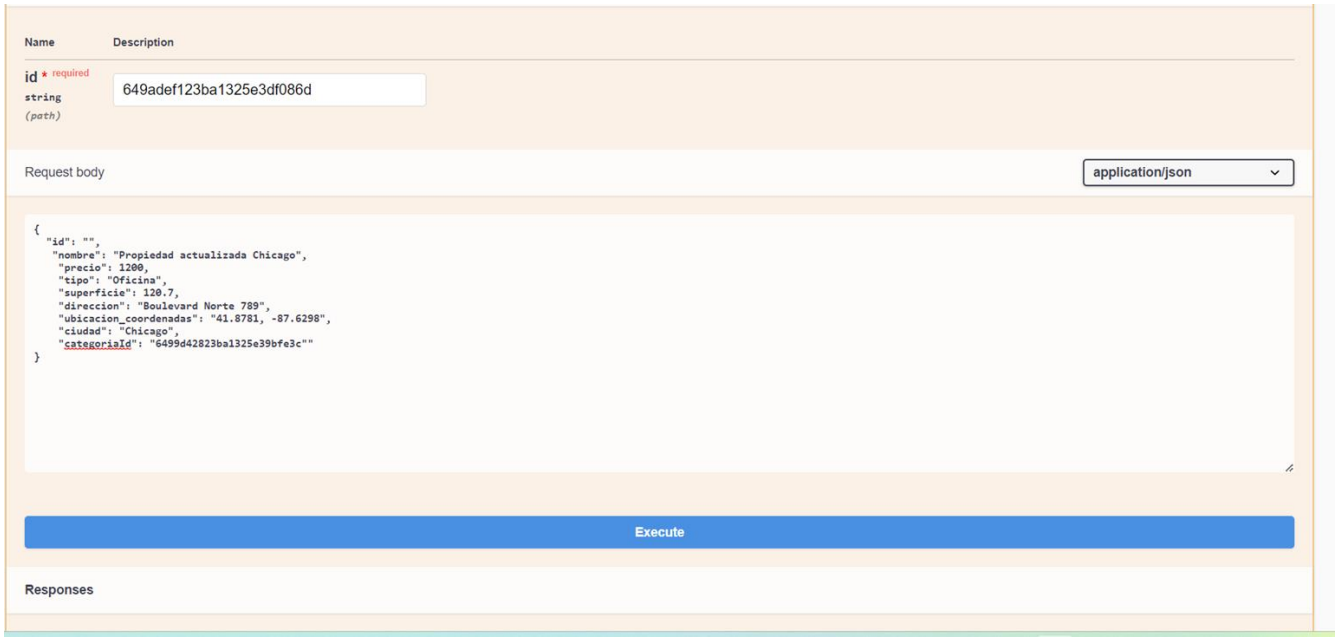
3.Parámetros: {idPropiedad}

4. Prueba Eliminación

Actualizar→ UpdatePropiedad();

1. Endpoint: https://localhost:7050/api/Propiedad/UpdatePropiedadad/
2. Método Http: PUT
3. Parámetros:

En este método es necesario especificar el del documento de dicha propiedad(lo pueden obtener usando el método GetAllPropiedades) o verificando en la consola de google el siguiente arreglo: GestionPropiedades.arregloPropiedadesFiltradas(Se explica a fondo en la documentación del front end).
No existe un método que consulte por nombre puesto que la inmensa mayoría de la lógica ya hago del lado del Front End para evitar la sobrecarga de peticiones al servidor.



Ejemplo Actualizar:

Actualizare la propiedad de la ciudad de chicago

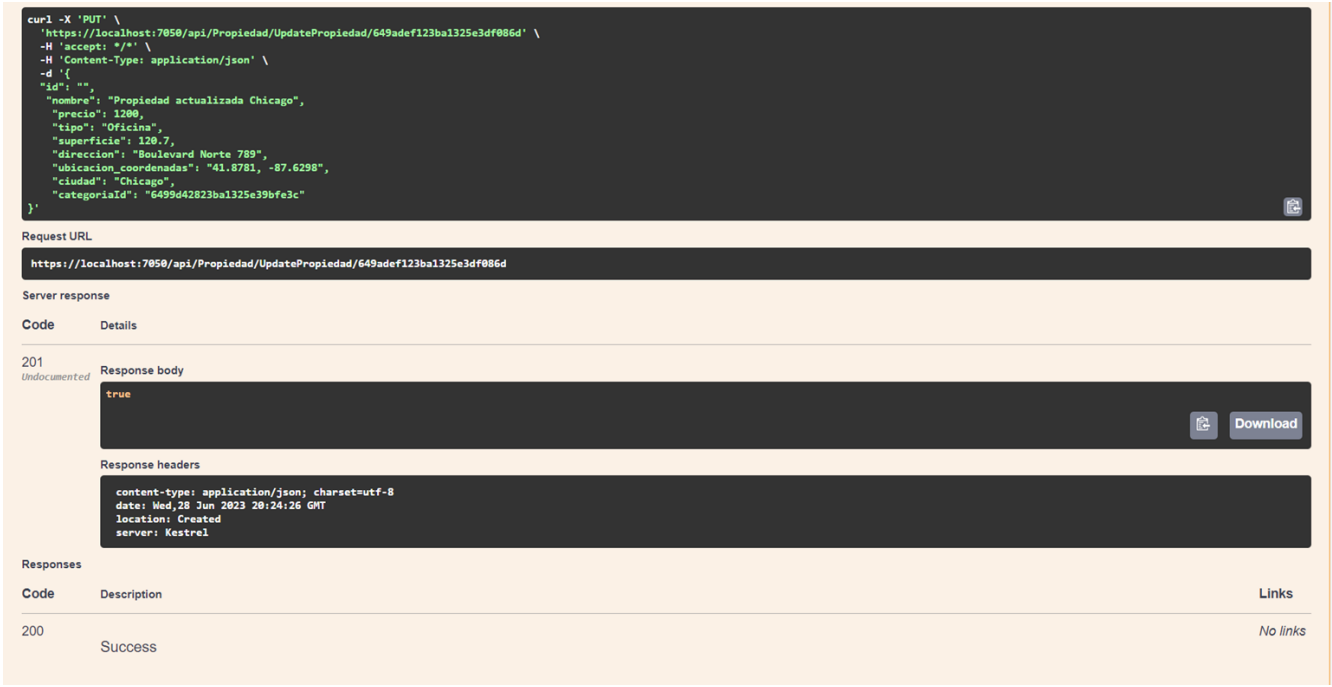
Response body


```
[
  {
    "id": "649adef123ba1325e3df086d",
    "nombre": "Propiedad 1 chicago",
    "precio": 1200,
    "tipo": "Oficina",
    "superficie": 120.7,
    "direccion": "Boulevard Norte 789",
    "ubicacion_coordenadas": "41.8781, -87.6298",
    "ciudad": "Chicago",
    "categoriaId": "6499d42823ba1325e39bfe3c"
  },
  {
    "id": "649adf6623ba1325e3dfe4fe",
    "nombre": "Propiedad 8",
    "precio": 3000,
    "tipo": "Local",
    "superficie": 250.2,
    "direccion": "Boulevard Norte 321",
    "ubicacion_coordenadas": "40.7128, -74.0060",
    "ciudad": "Nueva York",
    "categoriaId": "6499d42823ba1325e39bfe3e"
  },
  {
    "id": "649b4a2223ba1325e3cee97d",
    "nombre": "Propiedad numero SantiagoChile 1",
    "precio": 300,
```


Este es el cuerpo del JSON

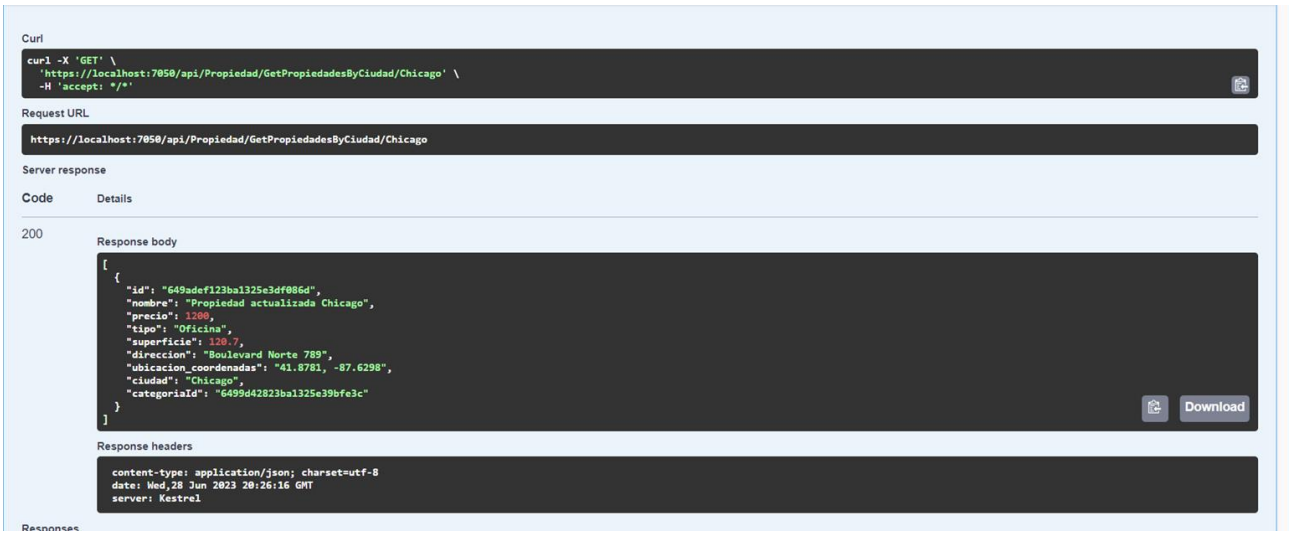

```
{
  "id": "",
   "nombre": "Propiedad actualizada Chicago",
    "precio": 1200,
    "tipo": "Oficina",
    "superficie": 120.7,
    "direccion": "Boulevard Norte 789",
    "ubicacion_coordenadas": "41.8781, -87.6298",
    "ciudad": "Chicago",
    "categoriaId": "6499d42823ba1325e39bfe3c""
}
```


Esto es lo que enviamos:



como podemos ver si o si debe ir el id en el request url

El resultado final es:



Métodos Categorías

para la colección categorías he creado 3 métodos de consulta:

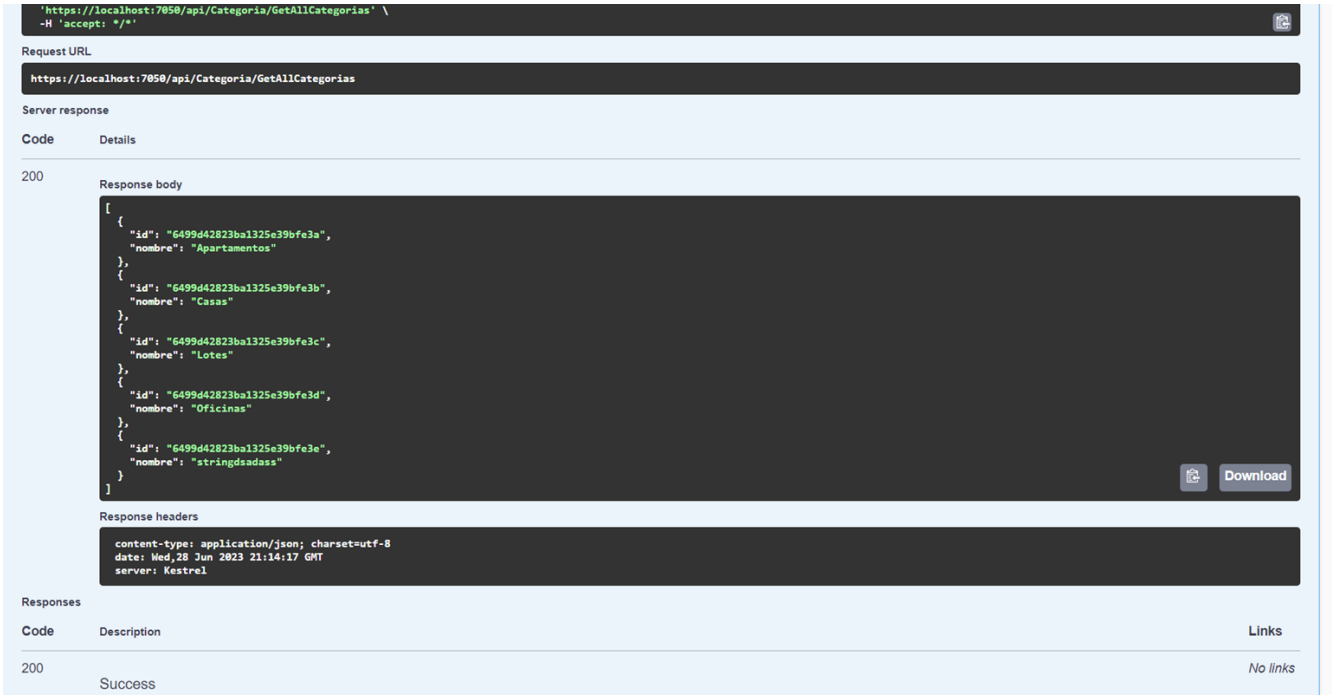
- **GetAllCategorias()-–→obtiene todas las categorías**

1. Endpoint: https://localhost:7050/api/Categoria/GetAllCategorias/

2. Método Http: GET

3. Parámetros : Ninguno

4. Retorna:



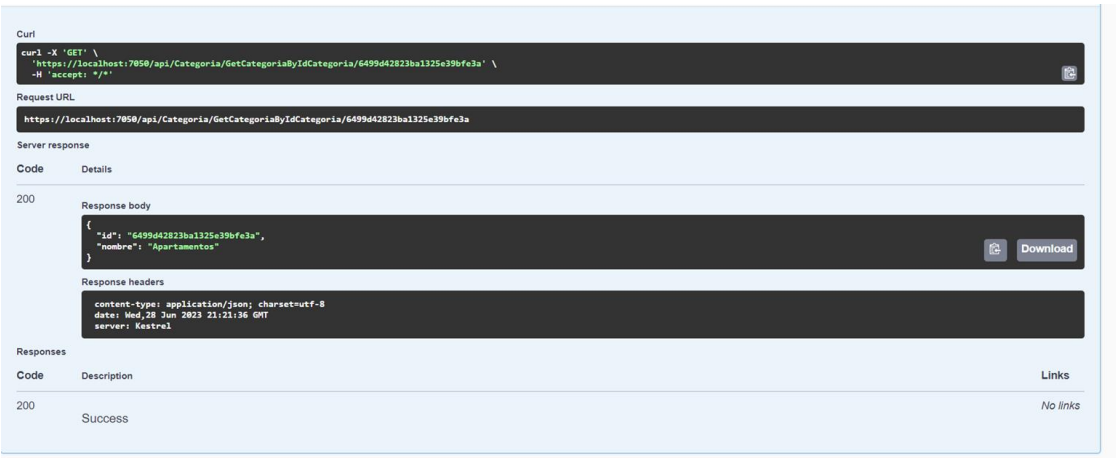
- **GetCategoriaByIdCategoria()**

1. Endpoint : https://localhost:7050/api/Categoria/GetCategoriaByIdCategoria/

2.Método Http: GET

3.Parámetros: {idCategoria}

4.Retorna:

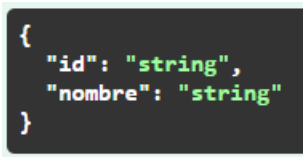


A continuación veremos los métodos que he creado como añadido personal para que puedan crear categorías mas fácilmente.

Métodos CRUD Categorías

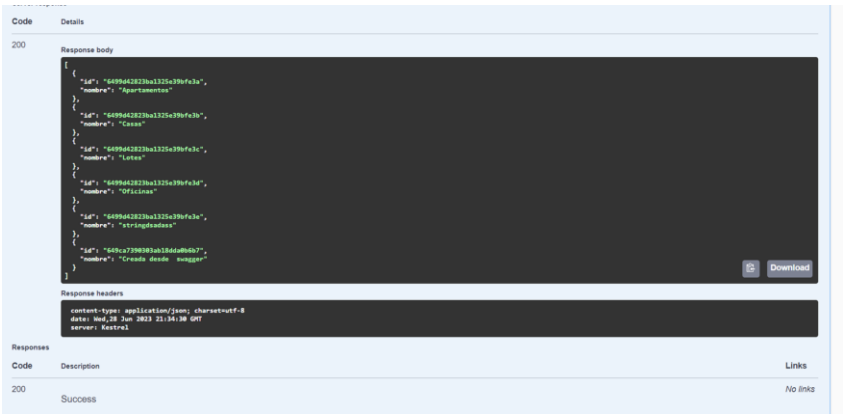
.Crear-→ CreateCategoria

1.Endpoint:https://localhost:7050/api/Categoria/CreateCategoria/
2.Método Http: POST
3.Parámetros: el campo "id" puede ir con cualquier texto,no es importante ya que en el back end le asigno un object id automaticamente



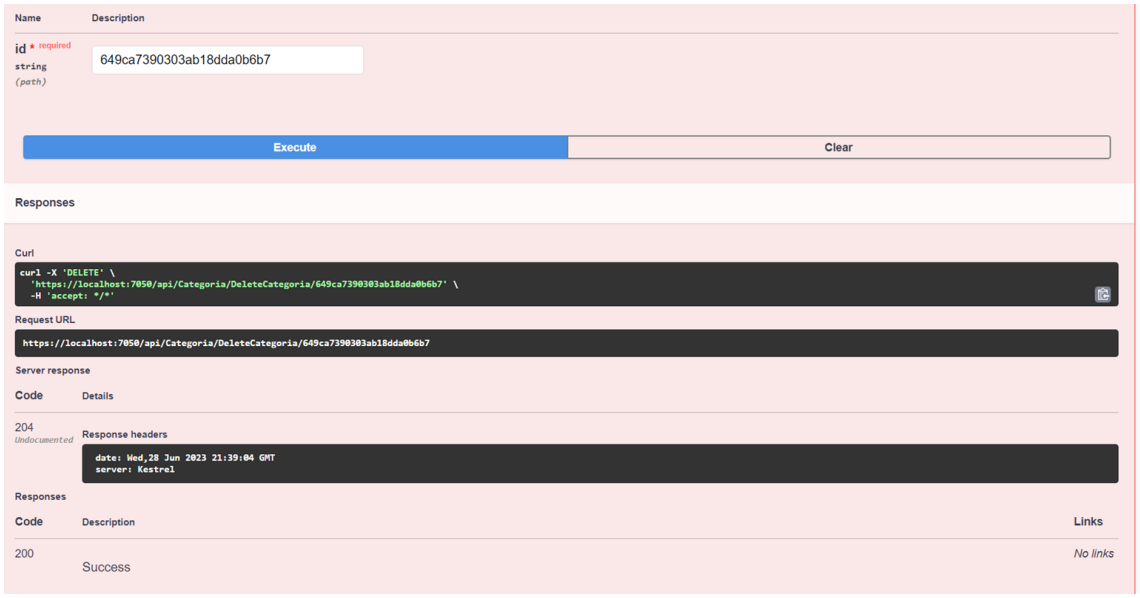
4.Retorna:

Consulto categorias y veo la nueva:



Eliminar-→DeleteCategoria()

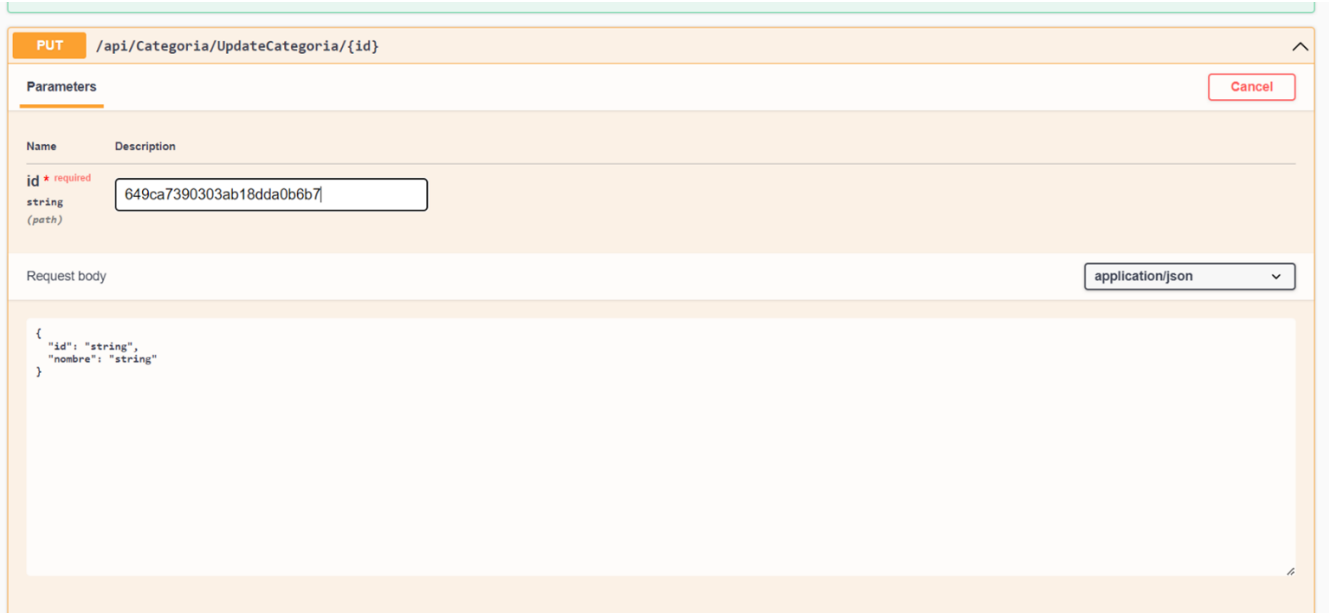
1. Endpoint: https://localhost:7050/api/Categoria/DeleteCategoria/
2. Metodo Http: POST
3. Parámetros: {idCategoria}
4.Retorna: aca borramos la que acabamos de crear



Actualizar→UpdateCategoria()

1. **Endpoint:** https://localhost:7050/api/Categoria/UpdateCategoria/
2. **Metodo Http: PUT**
3. **Parametros:**

En este método es necesario especificar el del documento de dicha Categoria(lo pueden obtener usando el método GetAllCategorias) o verificando en la consola de google el siguiente arreglo: GestionCategorias.arregloCategorias**(Se explica a fondo en la documentación del front end).**



Actualizaremos la categoria llamada "Creada desde Swagger":

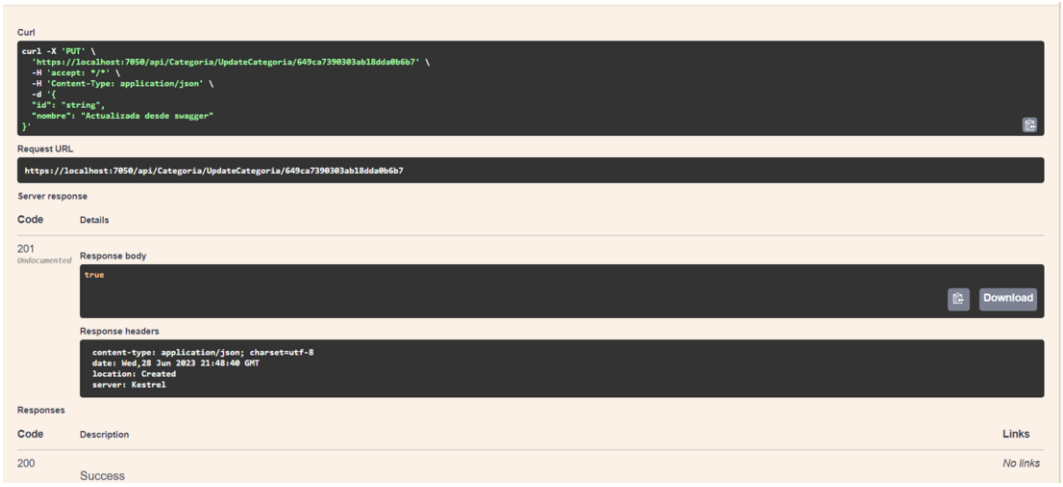
Response body


```
[
  {
    "id": "6499d42823ba1325e39bfe3a",
    "nombre": "Apartamentos"
  },
  {
    "id": "6499d42823ba1325e39bfe3b",
    "nombre": "Casas"
  },
  {
    "id": "6499d42823ba1325e39bfe3c",
    "nombre": "Lotes"
  },
  {
    "id": "6499d42823ba1325e39bfe3d",
    "nombre": "Oficinas"
  },
  {
    "id": "6499d42823ba1325e39bfe3e",
    "nombre": "stringdsadass"
  },
  {
    "id": "649ca7390303ab18dda0b6b7",
    "nombre": "Creada desde  swagger"
  }
]
```


Este es el cuerpo de la solicitud:


```
{
  "id": "string",
  "nombre": "Actualizada desde swagger"
}
```

nos Retorna esto:



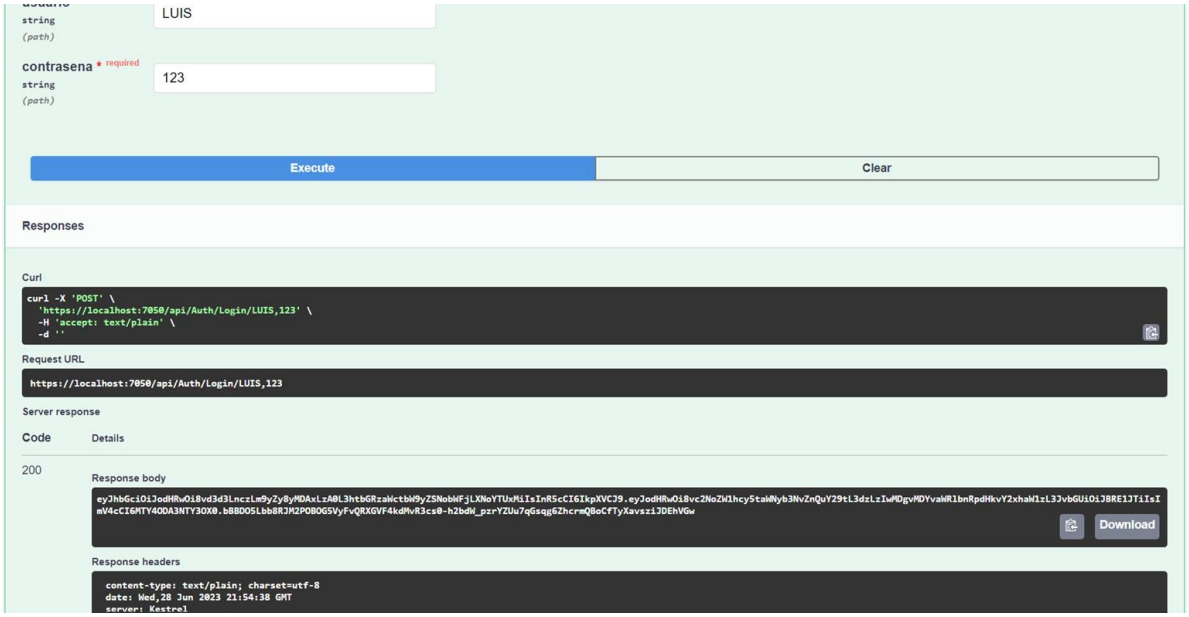
Verificamos nuevamente:


```
[
  {
    "id": "6499d42823ba1325e39bfe3a",
    "nombre": "Apartamentos"
  },
  {
    "id": "6499d42823ba1325e39bfe3b",
    "nombre": "Casas"
  },
  {
    "id": "6499d42823ba1325e39bfe3c",
    "nombre": "Lotes"
  },
  {
    "id": "6499d42823ba1325e39bfe3d",
    "nombre": "Oficinas"
  },
  {
    "id": "6499d42823ba1325e39bfe3e",
    "nombre": "stringdsadass"
  },
  {
    "id": "649ca9f50303ab18dda0b6b8",
    "nombre": "Actualizada desde swagger"
  }
]
```


Autenticación

Como he comentado anteriormente en este documento,esta API tiene autenticacion via token usando JWT en .net core 6.0,para no hacer mas largo este documento y ya que esto no se pedia en el enunciado,procedo a dar el endpoint de el metodo que consulta los usuarios y de el metodo que Loguea los usuarios

- **https://localhost:7050/api/Auth/Login/**



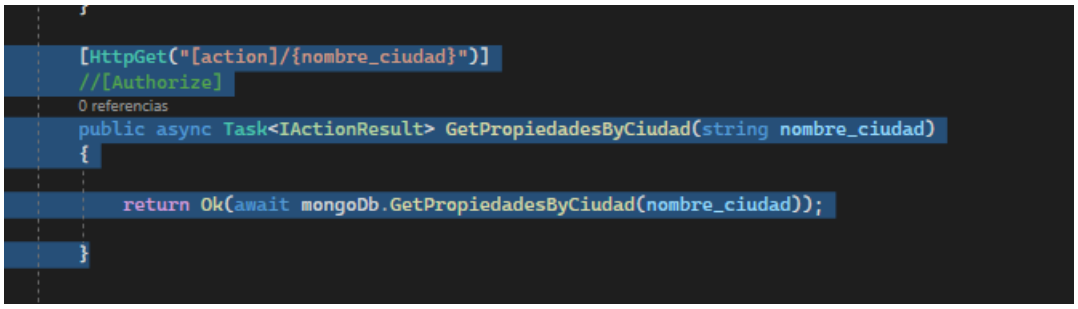
Los parametros son el usuario y la contraseña,esto devuelve el token

.https://localhost:7050/api/Auth/GetAllUser

Como pueden observar esto devuelve los usuarios

Debo aclarar que la autenticación SOLO funciona en la api,es decir,en el front esta autenticación no esta implementada.

Por ejemplo si



quisiéramos activar la autenticación por rol,deberíamos poner [Authorize(Roles = "ADMIN")] en la parte donde esta el authorize comentado.

Esto fue un añadido extra pero debo insistir en que si se activa esta opción,el front end no podrá hacer peticiones ya que no tiene implementada la autenticación

(Puedo sustentar esto en la entrevista técnica si es necesario ya que la autenticación no se solicitaba en el enunciado del problema)

EL DESPLIEGUE LO EXPLICO EN EL ARCHIVO "MANUAL DE DESPLIEGUE"