Universidad del País Vasco
Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE MASTER

Anexo II: Descripción de la solución realizada: Diseño de bajo nivel

profesiolan

Empresa apoyada por:

Estudiante: Hospital Gómez, Ander

Director: Prieto Agujeta, Gorka

Curso: 2023-2024

Fecha: Bilbao, a 23 de enero del 2024

Intencionadamente en blanco.

Introducción

El presente anexo está englobado en el Trabajo Fin de Máster Dimensionamiento, desarrollo, despliegue y análisis de rendimiento de una api graphql con una arquitectura de microservicios y cluster de base de datos altamente escalable en aws para producción.

Durante el mismo, se presentan los aspectos técnicos de todos y cada uno de los componentes que componen el proyecto realizado.

Contenido

Lista de tablas

Lista de ilustraciones

1. Data Layer

Como se ha referenciado durante el documento, la capa de datos del proyecto Profesiolan está basada en un motor MySQL desplegado sobre el servicio AWS Aurora. A continuación, se expone tanto la configuración desarrollada para Aurora, como el diseño de la base de datos. Además, se incluye información sobre la población de datos *dummy* inicial.

La parte referente a la configuración del servicio, así como la configuración de la instancia se ha desarrollado en el *Anexo I. Descripción de la solución realizada: Diseño de alto nivel* primero del presente Trabajo Fin de Máster.

1.1.1. Diseño de la base de datos

El diseño de la base de datos se basa en 15 tablas, y se detalla en la *Tabla 1: Definición de las tablas de la base de datos.*

Nombre de la tabla	Atributo	Clave	Tipo	Información extra
user	user_id	**PK**	binary(16)	Información básica de los usuarios.
	user_name		varchar(80)	
	first_name		varchar(80)	
	last_name		varchar(80)	
	email		varchar(100)	
	password		varchar(100)	
	registered		datetime	
	slug		varchar(50)	
	status		tinyint(4)	
	is_verified		tinyint(4)	
	count		int(11)	
	role		varchar(45)	
	suscription_number		int(11)	
	activation_key		varchar(70)	

	latitude		double	
	longitude		double	
	user_billing_id	**FK**	int(11)	
	user_meta_id	**FK**	binary(16)	
	user_address_id	**FK**	binary(16)	
	email_verification_token_id	**FK**	int(11)	
user_meta	user_meta_id	**PK**	binary(16)	Información extra de los usuarios.
	website		varchar(100)	
	linkedin		varchar(100)	
	facebook		varchar(100)	
	twitter		varchar(100)	
	biography		varchar(1000)	
	supplier		tinyint(4)	
	producer		tinyint(4)	
	ocasional		tinyint(4)	
	realestate		tinyint(4)	
	rating		int(11)	
	total_opinions		int(11)	
user_billing	user_billing_id	**PK**	int(11)	Información de los datos de facturación de los usuarios.
	address		varchar(200)	
	company_name		varchar(100)	
	tax_id		varchar(15)	
post	post_id	**PK**	binary(16)	Información básica de los activos.
	title		varchar(100)	
	content		varchar(5000)	
	published		datetime	
	modified		datetime	
	slug		varchar(200)	
	specific_condition		varchar(80)	
	use_condition		varchar(80)	
	description_condition		varchar(45)	

	latitude		varchar(20)	
	longitude		varchar(20)	
	status		varchar(45)	
	internal_pn		varchar(16)	
	external_pn		varchar(16)	
	stock		int(11)	
	price		double	
	price_insight		varchar(45)	
	has_price_insight		tinyint(1)	
	post_meta_id	**FK**	binary(16)	
	user_id	**FK**	binary(16)	
	isProfessional		tinyint(1)	
post_meta	post_meta_id	**PK**	binary(16)	Información extra de los activos publicados. Campos de metadata.
	hours		int(11)	
	km		int(11)	
	height		double	
	width		double	
	depth		double	
	weight		double	
category	category_id	**PK**	int(11)	Categorías del marketplace.
	count		int(11)	
	name		varchar(100)	
	meta_desc		varchar(2000)	
	parent_id		int(11)	
	has_child		tinyint(4)	
	slug		varchar(100)	
location	location_id	**PK**	int(11)	Localizaciones en las que opera el marketplace.
	count		int(11)	
	name		varchar(100)	
	meta_desc		varchar(1000)	
	parent_id		int(11)	

	has_child		tinyint(4)	
	slug		varchar(100)	
brand	brand_id	**PK**	int(11)	Información de todas las marcas disponibles asociadas a activos a la venta.
	name		varchar(100)	
	count		int(11)	
	slug		varchar(100)	
	meta_desc		varchar(1000)	
model	model_id	**PK**	int(11)	Información de todos los modelos asociados a activos a la venta.
	name		varchar(45)	
	count		int(11)	
	slug		varchar(45)	
	brand_id	**FK**	int(11)	
roles	roles_id	**PK**	int(11)	Información de todos los roles.
	name		varchar(45)	
category_has_brand	category_id	**PK**	int(11)	Asociación MxN.
	brand_id	**PK**	int(11)	
category_has_model	category_id	**PK**	int(11)	Asociación MxN.
	model_id	**PK**	int(11)	
post_has_categories	post_id	**PK**	binary(16)	Asociación MxN.
	category_id	**PK**	int(11)	
post_has_locations	post_id	**PK**	binary(16)	Asociación MxN.
	location_id	**PK**	int(11)	
user_has_roles	user_id	**PK**	binary(16)	Asociación MxN.
	roles_id	**PK**	int(11)	

Tabla 1: Definición de las tablas de la base de datos.

1.1.2. Población inicial de la base de datos

Gracias a la disponibilidad de una base de datos previa, se ha podido realizar una rápida población inicial para la fase de desarrollo de la solución. A continuación, en la *Tabla 2: Ejemplo del contenido dummy inicial de la base de datos,* se muestra una parte reducida de la información de muestra embebida en, por ejemplo, la tabla de usuarios. En el mencionado primer volcado de información, se han incluído, como mínimo, 15 entradas para cada tabla. Además, como se puede observar en la Tabla 2, el contenido *dummy* es totalmente coherente, de cara a facilitar el desarrollo y proceso de *debugging* de la aplicación.

user_id	0x17c6fccc054a11eeaffb2eb5a363657c	0x17c6ffa6054a11eeba812eb5a363657c
user_name	Vehículos Industriales Euskadi S.L.	Iberdrola S.A.
first_name	Aitana	Pedro
last_name	Herrera	Pérez
email	maria.gomez@gmail.com	pedro.perez@gmail.com
password	$2a$12$RvhOFRFI5Hm0kN5KHER4UePbxjGvYG4HLUM.R5RgHw1Ks6MiCI/Ve	$2a$12$RvhOFRFI5Hm0kN5KHER4UePbxjGvYG4HLUM.R5RgHw1Ks6MiCI/Ve
registered	2022-07-04 0:00:00	2023-04-02 10:16:58
slug	vehículos-industriales-euskadi-s.l.	iberdrola-s.a.
status	0	0
is_verified	0	1
count	5	5
role	client-tier-1	client-tier-2
suscription_number	200	20
activation_key	NULL	NULL
latitude	413.811.388.052.469	414.239.862.383.944
longitude	217.708.988.742.087	219.128.714.754.974
user_billing_id	1	2
user_meta_id	0x17c6fccc054a11eeaffb2eb5a36365ab	0x17c6fccc054a11eeaffb2eb5a36365cd
user_address_id	0x17c6fccc054a11eeaffb2eb5a36365ef	0x17c6fccc054a11eeaffb2eb5a36365gh

Tabla 2: Ejemplo del contenido dummy inicial de la base de datos.

2. Core Layer

Como se ha referenciado durante el documento, la capa core del proyecto Profesiolan se basa en una arquitectura distribuida y modular de microservicios basados en aplicaciones Java Spring Boot, desplegadas en contenedores AWS Lambda.

En este apartado se define el alcance de operación de cada uno de los servicios, además de presentar la estructura de los proyectos Spring desarrollados.

Una vez más, la parte referente a la configuración del servicio, así como la configuración de las instancias se ha desarrollado en el *Anexo I. Descripción de la solución realizada: Diseño de alto nivel* primero del presente Trabajo Fin de Máster.

2.1. Servicio Mgmt

El primer microservicio a presentar de la solución desarrollada es el denominado Servicio MGMT. En él, se centraliza gran parte de las tareas de mantenimiento y gestión del producto.

El servicio de Mgmt resuelve consultas relativas a la gestión de usuarios, de stock, de categorías y de filtros, entre otros. Por supuesto y tal y como se verá a lo largo de este apartado, se tratan de gestiones exclusivas a usuarios con permisos de administración.

En esta sección, se presentan tanto la lista de funcionalidades a cubrir como la estructura del proyecto Spring Boot. El grueso del análisis de rendimiento relativo a este microservicio en concreto se desarrolla durante en el *Anexo III. Análisis de rendimiento*.

2.1.1. Funcionalidades a cubrir

	Llamada	Uso
MgmtService	adminCreateBrand	MGMT
	adminUpdateBrand	MGMT
	adminDeleteBrand	MGMT

	adminCreateCategory	MGMT
	adminUpdateCategory	MGMT
	adminDeleteCategory	MGMT
	adminCreateLocation	MGMT
	adminUpdateLocation	MGMT
	adminDeleteLocation	MGMT
	adminCreateModel	MGMT
	adminUpdateModel	MGMT
	adminDeleteModel	MGMT
	getCategoryById	MGMT
	getLocationById	MGMT
	getBrandById	MGMT

Tabla 3: Funcionalidades cubiertas por el microservicio de gestión general.

2.1.2. Estructura del proyecto

2.1.2.1. Fichero de configuración pom.xml

En el proyecto Lan se trabaja con Apache Maven para la gestión de dependencias. Esta potente herramienta es de gran utilidad para compilar el código de manera automática, para instalar los .jar, gestionar las distintas fases del ciclo de vida de cada microservicio, y hasta generar documentación a partir del código fuente.

En concreto, se presenta el fichero de configuración pom.xml para el microsercivio MGMT en *Tabla 4: Fichero de configuración pom.xml para el microservicio MGMT.*

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>2.7.3</version>
                <relativePath/>
        </parent>
        <groupId>com.lan.mgmt</groupId>
        <artifactId>mgmt</artifactId>
        <version>0.0.1-SNAPSHOT</version>
```

```
<name>Mgmt Service</name>
<description>Management Service on AWS Lambda for LAN Project</description>
<properties>
	<java.version>11</java.version>
	<maven.compiler.source>11</maven.compiler.source>
	<maven.compiler.target>11</maven.compiler.target>
	<aws.java.sdk.version>2.17.66</aws.java.sdk.version>
</properties>

<dependencyManagement>
	<dependencies>
		<dependency>
			<groupId>software.amazon.awssdk</groupId>
			<artifactId>bom</artifactId>
			<version>${aws.java.sdk.version}</version>
			<type>pom</type>
			<scope>import</scope>
		</dependency>
	</dependencies>
</dependencyManagement>
<dependencies>
	<dependency>
		<groupId>com.fasterxml.uuid</groupId>
		<artifactId>java-uuid-generator</artifactId>
		<version>4.1.1</version>
	</dependency>
	<dependency>
		<groupId>com.amazonaws</groupId>
		<artifactId>aws-lambda-java-core</artifactId>
		<version>1.2.2</version>
	</dependency>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-web</artifactId>
		<exclusions>
			<exclusion>
				<groupId>org.springframework.boot</groupId>
				<artifactId>spring-boot-starter-tomcat</artifactId>
			</exclusion>
		</exclusions>
	</dependency>
	<dependency>
		<groupId>org.projectlombok</groupId>
		<artifactId>lombok</artifactId>
	</dependency>
	<dependency>
		<groupId>com.amazonaws</groupId>
		<artifactId>aws-lambda-java-events</artifactId>
		<version>3.11.0</version>
	</dependency>
	<dependency>
		<groupId>software.amazon.awssdk</groupId>
		<artifactId>url-connection-client</artifactId>
	</dependency>
	<dependency>
		<groupId>org.springframework.graphql</groupId>
		<artifactId>spring-graphql</artifactId>
	</dependency>
	<dependency>
```

```
                        <groupId>software.amazon.awssdk</groupId>
                        <artifactId>rdsdata</artifactId>
                        <version>2.17.66</version>
                        <exclusions>
                                <exclusion>
                                        <groupId>software.amazon.awssdk</groupId>
                                        <artifactId>netty-nio-client</artifactId>
                                </exclusion>
                                <exclusion>
                                        <groupId>software.amazon.awssdk</groupId>
                                        <artifactId>apache-client</artifactId>
                                </exclusion>
                        </exclusions>
                </dependency>
        </dependencies>
        <build>
                <plugins>
                        <plugin>
                                <groupId>org.apache.maven.plugins</groupId>
                                <artifactId>maven-shade-plugin</artifactId>
                                <version>3.2.4</version>
                                <configuration>

<createDependencyReducedPom>false</createDependencyReducedPom>
                                </configuration>
                                <executions>
                                        <execution>
                                                <phase>package</phase>
                                                <goals>
                                                        <goal>shade</goal>
                                                </goals>
                                        </execution>
                                </executions>
                        </plugin>
                </plugins>
        </build>
</project>
```

Tabla 4: Fichero de configuración pom.xml para el microservicio MGMT.

2.1.2.2. Aplicación principal

La aplicación principal del proyecto es una clase MgmtServiceApplication anotada con la directiva @SpringBootApplication.

La clase contiene un único método main con la llamada SpringApplication.run.

2.1.2.3. Handler del microservicio

Las clases Handler son la interfaz de entrada y salida del microservicio, y el único punto de comunicación con la API en la parte de backend.

En la arquitectura AWS Lambda, es necesario declarar el método handleRequest, que toma como entrada el contexto de la aplicación y el evento GraphQL, y devuelve la respuesta GraphQL para ser parseada por parte del servidor GraphQL en la capa API.

Finalmente, se incluye un switch con todos los métodos posibles a resolver, en un método llamado executeOperation. Este método toma el campo field y los argumentos en un Map, y llama al servicio correspondiente para su resolución. La lista de métodos aquí incluidos es la misma que se presenta en la *Tabla 3: Funcionalidades cubiertas por el microservicio de gestión general.*

2.1.2.4. Paquete Helper

El paquete Helper por su parte incluye clases de gran utilidad para el correcto funcionamiento de la aplicación. Por ejemplo, aquí se ha desarrollado la clase ConnectionHelper, que hace posible la comunicación con el cluster MySQL.

Entre otras cosas, se encarga de poder ofrecer operaciones transaccionales con la base de datos, gestionar las excepciones en tiempo real, parsear la información y, por supuesto, gestionar clientes de la base de datos.

2.1.2.5. Paquete Services

El paquete Services atiende a las buenas prácticas de desarrollo de proyecto Java Spring, y está compuesto de los siguientes servicios.

- BrandService
- CategoryService
- LocationService
- ModelService

2.1.2.6. Paquete Repositories

El paquete Repositories incluye las cuatro interfaces Java (relativas a resolver los cuatro servicios definidos), además de cuatro clases Java que implementan esos métodos.

2.1.2.7. Paquete Models

El paquete Models por su parte incluye las cuatro clases Java que contienen las cuatro entidades (Java Entities). Estas entidades son BrandEntity, CategoryEntity, LocationEntity, y ModelEntity.

2.1.2.8. Paquete DTOs

El paquete DTOs incluye las definiciones de objetos que no son entidades como tal, pero que son de alta importancia debido a que son los objetos que realmente son transferidos durante la resolución de las operaciones.

Aquí se encuentran tanto las clases AppSyncEvent y AppSyncResponse (objetos de entrada y salida GraphQL) como los objetos Output e Input definidos en el esquema GraphQL.

2.2. Servicio Stock

2.2.1. Funcionalidades a cubrir

	Llamada	Uso
StockService	adminCreatePost	MGMT
	adminUpdatePost	MGMT
	adminDeletePost	MGMT
	adminApprovePost	MGMT
	publishPost	Producto
	updatePost	Producto
	deletePost	Producto

Tabla 5: Funcionalidades cubiertas por el microservicio de gestión del stock.

2.2.2. Estructura del proyecto

2.2.2.1. Fichero de configuración pom.xml

A continuación, se presenta el fichero de configuración pom.xml para el microsercivio STOCK en

Tabla 6: Fichero de configuración pom.xml para el microservicio STOCK.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>2.7.3</version>
		<relativePath/>
	</parent>
	<groupId>com.lan.stock</groupId>
	<artifactId>stock</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<name>Stock Service</name>
	<description>Stock Management Service on AWS Lambda for LAN Project</description>
	<properties>
		<java.version>11</java.version>
		<maven.compiler.source>11</maven.compiler.source>
		<maven.compiler.target>11</maven.compiler.target>
		<aws.java.sdk.version>2.17.66</aws.java.sdk.version>
	</properties>

	<dependencyManagement>
		<dependencies>
			<dependency>
				<groupId>software.amazon.awssdk</groupId>
				<artifactId>bom</artifactId>
				<version>${aws.java.sdk.version}</version>
				<type>pom</type>
				<scope>import</scope>
			</dependency>
		</dependencies>
	</dependencyManagement>
	<dependencies>
		<dependency>
			<groupId>com.fasterxml.uuid</groupId>
			<artifactId>java-uuid-generator</artifactId>
			<version>4.1.1</version>
		</dependency>
		<dependency>
			<groupId>com.amazonaws</groupId>
```

```
                <artifactId>aws-lambda-java-core</artifactId>
                <version>1.2.2</version>
            </dependency>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-web</artifactId>
                <exclusions>
                    <exclusion>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-tomcat</artifactId>
                    </exclusion>
                </exclusions>
            </dependency>
            <dependency>
                <groupId>org.projectlombok</groupId>
                <artifactId>lombok</artifactId>
            </dependency>
            <dependency>
                <groupId>com.amazonaws</groupId>
                <artifactId>aws-lambda-java-events</artifactId>
                <version>3.11.0</version>
            </dependency>
            <dependency>
                <groupId>software.amazon.awssdk</groupId>
                <artifactId>url-connection-client</artifactId>
            </dependency>
            <dependency>
                <groupId>org.springframework.graphql</groupId>
                <artifactId>spring-graphql</artifactId>
            </dependency>
            <dependency>
                <groupId>software.amazon.awssdk</groupId>
                <artifactId>rdsdata</artifactId>
                <version>2.17.66</version>
                <exclusions>
                    <exclusion>
                        <groupId>software.amazon.awssdk</groupId>
                        <artifactId>netty-nio-client</artifactId>
                    </exclusion>
                    <exclusion>
                        <groupId>software.amazon.awssdk</groupId>
                        <artifactId>apache-client</artifactId>
                    </exclusion>
                </exclusions>
            </dependency>
        </dependencies>
        <build>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-shade-plugin</artifactId>
                    <version>3.2.4</version>
                    <configuration>

<createDependencyReducedPom>false</createDependencyReducedPom>
                    </configuration>
                    <executions>
                        <execution>
                            <phase>package</phase>
```

```
                                        <goals>
                                            <goal>shade</goal>
                                        </goals>
                                    </execution>
                                </executions>
                            </plugin>
                        </plugins>
                    </build>
</project>
```

Tabla 6: Fichero de configuración pom.xml para el microservicio STOCK.

2.2.2.2. Aplicación principal

La aplicación principal del proyecto es una clase StockServiceApplication anotada con la directiva @SpringBootApplication.

La clase contiene un único método main con la llamada SpringApplication.run.

2.2.2.3. Handler del microservicio

Las clases Handler son identicas entre sí, con la excepción del método executeOperation, que en este caso incluye, en su switch, la lista de métodos presentados en la *Tabla 5: Funcionalidades cubiertas por el microservicio de gestión del stock.*

2.2.2.4. Paquete Helper

La reusabilidad del código permite que el paquete Helper sea identico para cada uno de los microservicios.

2.2.2.5. Paquete Services

En este caso, el paquete de servicios incluye una única clase java como servicio, llamada StockService. Esta clase es la encargada de toda la gestión sobre el stock de Profesiolan.

2.2.2.6. Paquete Repositories

En este caso, el paquete Repositories incluye una única interfaz Java (StockRepository), además de la clase Java StockRepositoryImpl con la directiva implements StockRepository.

2.2.2.7. Paquete Models

El paquete Models por su parte incluye únicamente la clase Java StockEntity.

2.2.2.8. Paquete DTOs

Además de las clases AppSyncEvent y AppSyncResponse (objetos de entrada y salida GraphQL) se han incluido las entidades PostDataOutput, ReducedPost, etc.

2.3. Servicio User

2.3.1. Funcionalidades a cubrir

	Llamada	Uso
UserService	adminCreateUserBilling	MGMT
	adminUpdateUserBilling	MGMT
	adminDeleteUserBilling	MGMT
	adminCreateUserMeta	MGMT
	adminUpdateUserMeta	MGMT
	adminDeleteUserMeta	MGMT
	adminCreateUserAddress	MGMT
	adminUpdateUserAddress	MGMT
	adminDeleteUserAddress	MGMT
	adminUpdateUser	MGMT
	updateUser	Producto
	adminGetUser	MGMT

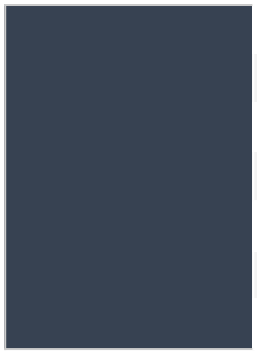	getUserMetaByUserId	MGMT
	getUserAddressByUserId	MGMT
	getUserBillingByUserId	MGMT
	hasUserMeta	Producto
	hasUserAddress	Producto
	hasUserBilling	Producto
	getUserByUserId	Producto

Tabla 7: Funcionalidades cubiertas por el microservicio de gestión de usuarios.

2.3.2. Estructura del proyecto

2.3.2.1. Fichero de configuración pom.xml

A continuación, se presenta el fichero de configuración pom.xml para el microsercivio User en *Tabla 8: Fichero de configuración pom.xml para el microservicio User.*

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>2.7.3</version>
		<relativePath/>
	</parent>
	<groupId>com.lan.user</groupId>
	<artifactId>user</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<name>User Service</name>
	<description>User Service on AWS Lambda for LAN Project</description>
	<properties>
		<java.version>11</java.version>
		<maven.compiler.source>11</maven.compiler.source>
		<maven.compiler.target>11</maven.compiler.target>
		<aws.java.sdk.version>2.17.66</aws.java.sdk.version>
	</properties>

	<dependencyManagement>
		<dependencies>
			<dependency>
				<groupId>software.amazon.awssdk</groupId>
				<artifactId>bom</artifactId>
				<version>${aws.java.sdk.version}</version>
				<type>pom</type>
				<scope>import</scope>
```

```
                </dependency>
        </dependencies>
</dependencyManagement>
<dependencies>
        <dependency>
                <groupId>com.fasterxml.uuid</groupId>
                <artifactId>java-uuid-generator</artifactId>
                <version>4.1.1</version>
        </dependency>
        <dependency>
                <groupId>com.amazonaws</groupId>
                <artifactId>aws-lambda-java-core</artifactId>
                <version>1.2.2</version>
        </dependency>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-web</artifactId>
                <exclusions>
                        <exclusion>
                                <groupId>org.springframework.boot</groupId>
                                <artifactId>spring-boot-starter-tomcat</artifactId>
                        </exclusion>
                </exclusions>
        </dependency>
        <dependency>
                <groupId>org.projectlombok</groupId>
                <artifactId>lombok</artifactId>
        </dependency>
        <dependency>
                <groupId>com.amazonaws</groupId>
                <artifactId>aws-lambda-java-events</artifactId>
                <version>3.11.0</version>
        </dependency>
        <dependency>
                <groupId>software.amazon.awssdk</groupId>
                <artifactId>url-connection-client</artifactId>
        </dependency>
        <dependency>
                <groupId>org.springframework.graphql</groupId>
                <artifactId>spring-graphql</artifactId>
        </dependency>
        <dependency>
                <groupId>software.amazon.awssdk</groupId>
                <artifactId>rdsdata</artifactId>
                <version>2.17.66</version>
                <exclusions>
                        <exclusion>
                                <groupId>software.amazon.awssdk</groupId>
                                <artifactId>netty-nio-client</artifactId>
                        </exclusion>
                        <exclusion>
                                <groupId>software.amazon.awssdk</groupId>
                                <artifactId>apache-client</artifactId>
                        </exclusion>
                </exclusions>
        </dependency>
</dependencies>
<build>
        <plugins>
```

```
                        <plugin>
                                <groupId>org.apache.maven.plugins</groupId>
                                <artifactId>maven-shade-plugin</artifactId>
                                <version>3.2.4</version>
                                <configuration>

<createDependencyReducedPom>false</createDependencyReducedPom>
                                </configuration>
                                <executions>
                                        <execution>
                                                <phase>package</phase>
                                                <goals>
                                                        <goal>shade</goal>
                                                </goals>
                                        </execution>
                                </executions>
                        </plugin>
                </plugins>
        </build>
</project>
```

Tabla 8: Fichero de configuración pom.xml para el microservicio STOCK.

2.3.2.2. Aplicación principal

La aplicación principal del proyecto es una clase UserServiceApplication anotada con la directiva @SpringBootApplication.

La clase contiene un único método main con la llamada SpringApplication.run.

2.3.2.3. Handler del microservicio

Las clases Handler son identicas entre sí, con la excepción del método executeOperation, que en este caso incluye, en su switch, la lista de métodos presentados en la *Tabla 7: Funcionalidades cubiertas por el microservicio de gestión de usuarios.*

2.3.2.4. Paquete Helper

La reusabilidad del código permite que el paquete Helper sea identico para cada uno de los microservicios.

2.3.2.5. Paquete Services

En este caso, el paquete de servicios incluye las siguientes clases, encargadas de la gestión integral de todos los usuarios, independientemente del plan contratado.

- UserService
- UserMetaService
- UserAddressService
- UserBillingService

2.3.2.6. Paquete Repositories

En este caso, el paquete Repositories incluye cuatro interfaces (para definir la resolución de los cuatro servicios), además de las cuatro clases Java con la directiva implements.

2.3.2.7. Paquete Models

El paquete Models por su parte incluye las entidades UserEntity, UserMetaEntity, UserAddressEntity y UserBillingEntity.

2.3.2.8. Paquete DTOs

Además de las clases AppSyncEvent y AppSyncResponse (objetos de entrada y salida GraphQL) se ha incluido la entidad UserDataOutput.

2.4. Servicio Search

2.4.1. Funcionalidades a cubrir

	Llamada	Uso
SearchService	getBrandById	MGMT
	getBrandsByCategoryId	MGMT
	getModelById	MGMT
	getModelsByBrandAndCategory	Producto
	getCategoryById	Producto
	getCategoriesByPostId	Producto
	adminGetPostById	Producto
	adminGetPostsByUser	MGMT
	getMyPosts	Producto
	getPostsByFilters	Producto
	getPostById	Producto

Tabla 9: Funcionalidades cubiertas por el microservicio de búsqueda.

2.4.2. Estructura del proyecto

2.4.2.1. Fichero de configuración pom.xml

A continuación, se presenta el fichero de configuración pom.xml para el microsercivio SEARCH en

Tabla 10: Fichero de configuración pom.xml para el microservicio Search.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
          <modelVersion>4.0.0</modelVersion>
          <parent>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-starter-parent</artifactId>
                    <version>2.7.3</version>
                    <relativePath/>
          </parent>
          <groupId>com.lan.search</groupId>
          <artifactId>search</artifactId>
          <version>0.0.1-SNAPSHOT</version>
          <name>Search Service</name>
```

```
<description>Search Service on AWS Lambda for LAN Project</description>
<properties>
	<java.version>11</java.version>
	<maven.compiler.source>11</maven.compiler.source>
	<maven.compiler.target>11</maven.compiler.target>
	<aws.java.sdk.version>2.17.66</aws.java.sdk.version>
</properties>
<dependencyManagement>
	<dependencies>
		<dependency>
			<groupId>software.amazon.awssdk</groupId>
			<artifactId>bom</artifactId>
			<version>${aws.java.sdk.version}</version>
			<type>pom</type>
			<scope>import</scope>
		</dependency>
	</dependencies>
</dependencyManagement>
<dependencies>
	<dependency>
		<groupId>com.fasterxml.uuid</groupId>
		<artifactId>java-uuid-generator</artifactId>
		<version>4.1.1</version>
	</dependency>
	<dependency>
		<groupId>com.amazonaws</groupId>
		<artifactId>aws-lambda-java-core</artifactId>
		<version>1.2.2</version>
	</dependency>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-web</artifactId>
		<exclusions>
			<exclusion>
				<groupId>org.springframework.boot</groupId>
				<artifactId>spring-boot-starter-tomcat</artifactId>
			</exclusion>
		</exclusions>
	</dependency>
	<dependency>
		<groupId>org.projectlombok</groupId>
		<artifactId>lombok</artifactId>
	</dependency>
	<dependency>
		<groupId>com.amazonaws</groupId>
		<artifactId>aws-lambda-java-events</artifactId>
		<version>3.11.0</version>
	</dependency>
	<dependency>
		<groupId>software.amazon.awssdk</groupId>
		<artifactId>url-connection-client</artifactId>
	</dependency>
	<dependency>
		<groupId>org.springframework.graphql</groupId>
		<artifactId>spring-graphql</artifactId>
	</dependency>
	<dependency>
		<groupId>software.amazon.awssdk</groupId>
		<artifactId>rdsdata</artifactId>
```

```
                        <version>2.17.66</version>
                        <exclusions>
                                <exclusion>
                                        <groupId>software.amazon.awssdk</groupId>
                                        <artifactId>netty-nio-client</artifactId>
                                </exclusion>
                                <exclusion>
                                        <groupId>software.amazon.awssdk</groupId>
                                        <artifactId>apache-client</artifactId>
                                </exclusion>
                        </exclusions>
                </dependency>
        </dependencies>
        <build>
                <plugins>
                        <plugin>
                                <groupId>org.apache.maven.plugins</groupId>
                                <artifactId>maven-shade-plugin</artifactId>
                                <version>3.2.4</version>
                                <configuration>

<createDependencyReducedPom>false</createDependencyReducedPom>
                                </configuration>
                                <executions>
                                        <execution>
                                                <phase>package</phase>
                                                <goals>
                                                        <goal>shade</goal>
                                                </goals>
                                        </execution>
                                </executions>
                        </plugin>
                </plugins>
        </build>
</project>
```

Tabla 10: Fichero de configuración pom.xml para el microservicio SEARCH.

2.4.2.2. Aplicación principal

La aplicación principal del proyecto es una clase SearchServiceApplication anotada con la directiva @SpringBootApplication.

La clase contiene un único método main con la llamada SpringApplication.run.

2.4.2.3. Handler del microservicio

Las clases Handler son identicas entre sí, con la excepción del método executeOperation, que en este caso incluye, en su switch, la lista de métodos presentados en la *Tabla 9: Funcionalidades cubiertas por el microservicio de búsqueda.*

2.4.2.4. Paquete Helper

La reusabilidad del código permite que el paquete Helper sea identico para cada uno de los microservicios.

2.4.2.5. Paquete Services

En este caso, el paquete de servicios incluye únicamente el servicio SearchService. Clase responsable de resolver todas las consultas de búsqueda y de navegación.

2.4.2.6. Paquete Repositories

De igual forma, el paquete Repositories incluye una única interfaz SearchRepository, y la respectiva clase de implementación, SearchRepositoryImpl.

2.4.2.7. Paquete Models

Al tratarse de un microservicio que agrega la totalidad de la necesidad de filtrado y búsqueda, en el paquete Models encontramos las definiciones de Brand, Category, Location, Model y Post.

2.4.2.8. Paquete DTOs

Además de las clases AppSyncEvent y AppSyncResponse (objetos de entrada y salida GraphQL) se han incluido los DTO PostDataOutput, PublicPost, ReducedModel, ReducedCategory, ReducedPost y ReducedLocation.

3. API Layer

Como se ha referenciado durante el documento, la capa API del proyecto Profesiolan está basada en la tecnología GraphQL, y ha sido desplegada sobre el servicio AWS AppSync. A continuación, se expone tanto la configuración desarrollada para AppSync, como el esquema Graph público de la API. Además, se definen todas las consultas desarrolladas y servibles desplegadas.

3.1. Esquema público de la API

A continuación, se detalla el esquema GraphQL de la API desarrollado.

Main schema: Query and Mutations definition.

```
schema {
	query: Query
	mutation: Mutation
}

type Mutation {
	adminCreateBrand(input: AdminCreateBrandInput!): brand
		@aws_auth(cognito_groups: ["admin"])
	adminUpdateBrand(input: AdminUpdateBrandInput!): brand
		@aws_auth(cognito_groups: ["admin"])
	adminDeleteBrand(input: AdminDeleteBrandInput!): String
		@aws_auth(cognito_groups: ["admin"])
	adminCreateCategory(input: AdminCreateCategoryInput!): category
		@aws_auth(cognito_groups: ["admin"])
	adminUpdateCategory(input: AdminUpdateCategoryInput!): category
		@aws_auth(cognito_groups: ["admin"])
	adminDeleteCategory(input: AdminDeleteCategoryInput!): String
		@aws_auth(cognito_groups: ["admin"])
	adminCreateLocation(input: AdminCreateLocationInput!): location
		@aws_auth(cognito_groups: ["admin"])
	adminUpdateLocation(input: AdminUpdateLocationInput!): location
		@aws_auth(cognito_groups: ["admin"])
	adminDeleteLocation(input: AdminDeleteLocationInput!): String
		@aws_auth(cognito_groups: ["admin"])
	adminCreateModel(input: AdminCreateModelInput!): model
		@aws_auth(cognito_groups: ["admin"])
	adminUpdateModel(input: AdminUpdateModelInput!): model
		@aws_auth(cognito_groups: ["admin"])
	adminDeleteModel(input: AdminDeleteModelInput!): String
		@aws_auth(cognito_groups: ["admin"])
	adminCreateUserBilling(input: AdminCreateUserBillingInput!): user_billing
		@aws_auth(cognito_groups: ["admin"])
	adminUpdateUserBilling(input: AdminUpdateUserBillingInput!): user_billing
		@aws_auth(cognito_groups: ["admin"])
	adminDeleteUserBilling(input: AdminDeleteUserBillingInput!): String
		@aws_auth(cognito_groups: ["admin"])
	adminCreateUserMeta(input: AdminCreateUserMetaInput!): user_meta
```

```
        @aws_auth(cognito_groups: ["admin"])
    adminUpdateUserMeta(input: AdminUpdateUserMetaInput!): user_meta
        @aws_auth(cognito_groups: ["admin"])
    adminDeleteUserMeta(input: AdminDeleteUserMetaInput!): String
        @aws_auth(cognito_groups: ["admin"])
    adminCreateUserAddress(input: AdminCreateModelInput!): user_address
        @aws_auth(cognito_groups: ["admin"])
    adminUpdateUserAddress(input: AdminUpdateUserAddressInput!):
user_address
        @aws_auth(cognito_groups: ["admin"])
    adminDeleteUserAddress(input: AdminDeleteUserAddressInput!): String
        @aws_auth(cognito_groups: ["admin"])
    adminUpdateUser(input: AdminUpdateUserInput!): user
        @aws_auth(cognito_groups: ["admin"])
    adminCreatePost(input: AdminCreatePostInput): post
        @aws_auth(cognito_groups: ["admin"])
    adminUpdatePost(input: AdminUpdatePostInput): post
        @aws_auth(cognito_groups: ["admin"])
    adminDeletePost(input: AdminDeletePostInput): String
        @aws_auth(cognito_groups: ["admin"])
    updateUser(input: UpdateUserDataInput!): AdminUpdateUserOutput
        @aws_auth(cognito_groups:
["seller_tier_occasional","seller_tier_pro_1","seller_tier_pro_2"])
    publishPost(input: PublishPostInput!): PostDataOutput
        @aws_auth(cognito_groups:
["seller_tier_occasional","seller_tier_pro_1","seller_tier_pro_2"])
    updatePost(input: UpdatePostInput!): PostDataOutput
        @aws_auth(cognito_groups:
["seller_tier_occasional","seller_tier_pro_1","seller_tier_pro_2"])
    deletePost(input: DeletePostInput!): String
        @aws_auth(cognito_groups:
["seller_tier_occasional","seller_tier_pro_1","seller_tier_pro_2"])
    adminApprovePost(input: AdminApprovePostInput!): String
        @aws_auth(cognito_groups: ["admin"])
}

type Query {
    getBrandById(brand_id: Int!): brand
        @aws_api_key
    getBrandsByCategoryId(category_id: Int!): [brand]
        @aws_api_key
    getModelById(model_id: Int!): model
        @aws_api_key
    getModelsByBrandAndCategory(category_id: Int!, brand_id: Int!):
[ReducedModel]
        @aws_api_key
    getCategoryById(category_id: Int!): ReducedCategory
        @aws_api_key
    getCategoriesByPostId(post_id: ID!): [ReducedCategory]
        @aws_api_key
    adminGetPostById(post_id: ID!): PostDataOutput
        @aws_auth(cognito_groups: ["admin"])
    adminGetPostsByUser(user_id: ID!): ReducedPost
        @aws_auth(cognito_groups: ["admin"])
    getMyPosts(user_id: ID!): [ReducedPost]
        @aws_cognito_user_pools(cognito_groups:
["admin","seller_tier_occasional","seller_tier_pro_1","seller_tier_pro_2"])
@aws_auth(cognito_groups:
["admin","seller_tier_occasional","seller_tier_pro_1","seller_tier_pro_2"])
```

```
        getPostsByFilters(
                pageNumber: Int,
                pageSize: Int,
                sortDirection: String,
                sortBy: String,
                category_id: Int,
                professional: Boolean
        ): [ReducedPost]
                @aws_api_key
        getPostById(post_id: ID!): PublicPost
                @aws_api_key
@aws_cognito_user_pools(cognito_groups:
["admin","seller_tier_occasional","seller_tier_pro_1","seller_tier_pro_2"])
@aws_auth(cognito_groups:
["admin","seller_tier_occasional","seller_tier_pro_1","seller_tier_pro_2"])
        adminGetUser(user_id: ID!): AdminUpdateUserOutput
                @aws_auth(cognito_groups: ["admin"])
        hasUserMeta(user_id: ID!): Boolean
                @aws_api_key
        hasUserAddress(user_id: ID!): Boolean
                @aws_api_key
        hasUserBilling(user_id: ID!): Boolean
                @aws_api_key
        getUserMetaByUserId(user_id: ID!): user_meta
@aws_cognito_user_pools(cognito_groups:
["admin","seller_tier_occasional","seller_tier_pro_1","seller_tier_pro_2"])
@aws_auth(cognito_groups:
["admin","seller_tier_occasional","seller_tier_pro_1","seller_tier_pro_2"])
        getUserAddressByUserId(user_id: ID!): user_address
@aws_cognito_user_pools(cognito_groups:
["admin","seller_tier_occasional","seller_tier_pro_1","seller_tier_pro_2"])
@aws_auth(cognito_groups:
["admin","seller_tier_occasional","seller_tier_pro_1","seller_tier_pro_2"])
        getUserBillingByUserId(user_id: ID!): user_billing
@aws_cognito_user_pools(cognito_groups:
["admin","seller_tier_occasional","seller_tier_pro_1","seller_tier_pro_2"])
@aws_auth(cognito_groups:
["admin","seller_tier_occasional","seller_tier_pro_1","seller_tier_pro_2"])
        getUserByUserId(user_id: ID!): UserDataOutput
@aws_cognito_user_pools(cognito_groups:
["admin","seller_tier_occasional","seller_tier_pro_1","seller_tier_pro_2"])
@aws_auth(cognito_groups:
["admin","seller_tier_occasional","seller_tier_pro_1","seller_tier_pro_2"])
}
```

Main schema: Graph type-entities definition.

```
type user {
        user_id: ID!
        user_name: String
        first_name: String
        last_name: String
        email: AWSEmail
        password: String
        registered: AWSDateTime
        slug: String
        status: Boolean
        is_verified: Boolean
        count: Int
        role: String
        suscription_number: Int
```

```
        latitude: Float
        longitude: Float
        userAddress: user_address
}

type user_address {
        user_address_id: ID!
        address_line: String!
        city: String!
        postal_code: String!
        district: String!
        country: String!
        phone: String!
}

type user_billing {
        user_billing_id: ID!
        address: String!
        company_name: String!
        tax_id: String!
}

type user_meta {
        user_meta_id: ID!
        website: String
        linkedin: String
        facebook: String
        twitter: String
        biography: String!
        supplier: Boolean
        producer: Boolean
        ocasional: Boolean
        realestate: Boolean
        rating: Float!
        total_opinions: Int!
}

type post {
        post_id: ID!
        title: String!
        content: String!
        published: String
        modified: String
        slug: String!
        specific_condition: String
        use_condition: String
        description_condition: String
        latitude: String!
        longitude: String!
        status: String
        stock: Int
        price: Float
        price_insight: String
        has_price_insight: Boolean
        postMeta: post_meta
        user: user
        postCategory: [category]
        postLocation: [location]
}
```

```
type post_meta {
        post_meta_id: ID!
        hours: Int
        km: Int
        height: Float
        width: Float
        depth: Float
}

type brand {
        brand_id: ID
        name: String
        count: Int
        slug: String
        meta_desc: String
}

type category {
        category_id: ID
        count: Int!
        name: String!
        meta_desc: String!
        parent: category
        has_child: Boolean
        slug: String!
        brands: [brand]
        models: [model]
}

type location {
        location_id: ID!
        count: Int!
        name: String!
        meta_desc: String!
        parent: location
        has_child: Boolean!
        slug: String!
}

type model {
        model_id: ID!
        name: String!
        count: Int!
        slug: String!
        categories: [ReducedCategory]
        brand: brand
}

type category_has_brand {
        category_id: ID!
        brand_id: ID!
}

type category_has_model {
        category_id: ID!
        model_id: ID!
}
```

Main schema: Graph input-entities definition.

```
type post_has_categories {
	post_id: ID!
	category_id: ID!
}

type post_has_locations {
	post_id: ID!
	locations_id: ID!
}

type roles {
	roles_id: ID!
	name: String
}

type user_has_roles {
	user_id: ID!
	roles_id: ID!
}
```

```
input AdminCreateBrandInput {
	name: String!
	slug: String!
	meta_desc: String!
}

input AdminCreateCategoryInput {
	name: String!
	slug: String!
	meta_desc: String!
	parent_id: Int!
}

input AdminCreateLocationInput {
	name: String!
	slug: String!
	meta_desc: String!
	parent_id: Int!
}

input AdminCreateModelInput {
	name: String!
	slug: String!
	brand_id: Int!
}

input AdminCreatePostInput {
	user_id: ID!
	title: String!
	content: String!
	use_condition: String!
	description_condition: String!
	latitude: Float!
	longitude: Float!
	price: Float!
	category_id: [Int!]!
}

input AdminCreateUserAddressInput {
```

```
        address_line: String!
        city: String!
        postal_code: String!
        district: String!
        country: String!
        phone: String!
        user_id: ID!
}

input AdminCreateUserBillingInput {
        address: String!
        company_name: String!
        tax_id: String!
        user_id: ID!
}

input AdminCreateUserMetaInput {
        website: String!
        biography: String!
        supplier: Boolean!
        producer: Boolean!
        ocasional: Boolean!
        realestate: Boolean!
        user_id: ID!
}

input AdminDeleteBrandInput {
        brand_id: Int!
}

input AdminDeleteCategoryInput {
        category_id: Int!
}

input AdminDeleteLocationInput {
        location_id: Int!
}

input AdminDeleteModelInput {
        model_id: Int!
}

input AdminDeletePostInput {
        post_id: ID!
}

input AdminDeleteUserAddressInput {
        user_address_id: ID!
}

input AdminDeleteUserBillingInput {
        user_billing_id: Int!
}

input AdminDeleteUserMetaInput {
        user_meta_id: ID!
}

input AdminUpdateBrandInput {
```

```
        brand_id: Int!
        name: String!
        slug: String!
        meta_desc: String!
}

input AdminUpdateCategoryInput {
        category_id: Int!
        name: String!
        slug: String!
        meta_desc: String!
}

input AdminUpdateLocationInput {
        location_id: Int!
        name: String!
        slug: String!
        meta_desc: String!
}

input AdminUpdateModelInput {
        model_id: ID!
        name: String
        count: Int
        slug: String
}

input AdminUpdatePostInput {
        post_id: ID!
        title: String!
        content: String!
        slug: String!
        use_condition: String!
        description_condition: String!
        latitude: Float!
        longitude: Float!
        price: Float!
        category_id: [Int!]!
}

input AdminUpdateUserAddressInput {
        user_address_id: ID!
        address_line: String!
        city: String!
        postal_code: String!
        district: String!
        country: String!
        phone: String!
}

input AdminUpdateUserBillingInput {
        user_billing_id: Int!
        address: String!
        company_name: String!
        tax_id: String!
}

input AdminUpdateUserInput {
        user_id: ID!
```

```
        first_name: String!
        last_name: String!
        latitude: Float!
        longitude: Float!
}

input AdminUpdateUserMetaInput {
        user_meta_id: ID!
        website: String!
        biography: String!
        supplier: Boolean!
        producer: Boolean!
        ocasional: Boolean!
        realestate: Boolean!
}

input CategoryListInput {
        category_id: Int!
}

input DeletePostInput {
        post_id: ID!
        user_id: ID!
}

input FindAllPostsWithFiltersInput {
        postPageInput: PostPageInput
        postSearchCriteriaInput: PostSearchCriteriaInput
}

input PostPageInput {
        pageNumber: Int
        pageSize: Int
        sortDirection: String
        sortBy: String
}

input PostSearchCriteriaInput {
        isProfessional: Boolean
        category_id: ID
}

input PublishPostInput {
        user_id: ID!
        title: String!
        content: String!
        use_condition: String!
        description_condition: String!
        latitude: Float!
        longitude: Float!
        price: Float!
        category_id: [Int!]!
}

input UpdatePostInput {
        post_id: ID!
        user_id: ID!
        title: String!
        content: String!
```

```
        use_condition: String!
        description_condition: String!
        latitude: Float!
        longitude: Float!
        price: Float!
        category_id: [Int!]!
}

input UpdateUserDataInput {
        user_id: ID!
        first_name: String!
        last_name: String!
        latitude: Float
        longitude: Float
}
```

Main schema: Graph output-entities definition.

```
type PostSimplifiedOutput {
        post_id: ID!
        title: String!
        specific_condition: String
        price: Float!
        price_insight: String
        has_price_insight: Boolean
        professional: Boolean
        slug: String
        postCategory: [category]
}

type PublishPostOutput {
        user_id: ID!
        title: String!
        content: String!
        use_condition: String!
        description_condition: String!
        latitude: Float!
        longitude: Float!
        price: Float!
        category_id: [Int!]!
}

type UpdateUserDataOutput {
        user_id: ID!
        first_name: String!
        last_name: String!
        latitude: Float
        longitude: Float
}

type UserDataOutput {
        user_id: ID!
        user_name: String
        first_name: String
        last_name: String
        email: AWSEmail
        slug: String
        count: Int
        role: String
        suscription_number: Int
        latitude: Float
```

```
        longitude: Float
        is_verified: Boolean
        user_address: user_address
        user_billing: user_billing
        user_meta: user_meta
}

type ReducedCategory {
        category_id: ID
        count: Int!
        name: String!
        meta_desc: String!
        has_child: Boolean
        slug: String!
}

type ReducedModel {
        model_id: ID!
        name: String!
        count: Int!
        slug: String!
}

type ReducedPost {
        post_id: ID!
        title: String!
        slug: String!
        status: String!
        price: Float!
        professional: Boolean!
}

type PublicPost {
        post_id: ID!
        title: String!
        content: String!
        published: String!
        slug: String!
        use_condition: String!
        description_condition: String!
        latitude: Float!
        longitude: Float!
        stock: Int!
        price: Float!
        has_price_insight: Boolean!
        user_id: ID!
        professional: Boolean!
}
```

Tabla 11: Esquema público de la API, organizado por grupos lógicos de objetos.

3.2. Conjunto de Queries & Mutations de Administración

A continuación, se procede a definir en detalle la totalidad de métodos disponibles de administración. Para cada uno, también se presenta una tabla resumen con la información básica más práctica, que incluye además de un identificador y las definiciones de entradas y salidas, el rol requerido para ejecución, además del microservicio del *Core Layer* donde se resuelve.

3.2.1. adminCreateBrand()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	M.M.1
Tipo	Mutation
Llamada	adminCreateBrand()

Recibe	AdminCreateBrandInput - Obligatorio {
	name: String!
	slug: String!
	meta_desc: String!
	}

Devuelve	brand - Obligatorio {
	brand_id: ID
	name: String
	count: Int
	slug: String
	meta_desc: String
	}

Autorización	Grupo administrador

Tabla 12: Resumen del método adminCreateBrand.

Método de administración que sirve para incluir una nueva marca en la base de datos. Desde un principio, la base de datos cuenta con más de 500 marcas, importadas con su jerarquía y metadatos desde la base de datos del MVP. No obstante, un marketplace se encuentra en un estado de continuo crecimiento e incorporación de nuevo stock.

3.2.2. adminUpdateBrand()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	M.M.2
Tipo	Query
Llamada	adminUpdateBrand()

Recibe	AdminUpdateBrandInput - Obligatorio {
	brand_id: Int!
	name: String!
	slug: String!
	meta_desc: String!
	}

Devuelve	brand - Obligatorio {
	brand_id: ID
	name: String
	count: Int
	slug: String
	meta_desc: String
	}

Autorización	Grupo administrador

Tabla 13: Resumen del método adminUpdateBrand.

Método de administración que sirve para actualizar algún campo de alguna marca en la base de datos. Por ejemplo, esto puede ser para actualizar alguna jerarquía, para cambiar el slug de cara a la mejora del SEO, o incluso para modificar el valor del recuento de activos que se encuentran asociados a una marca en concreto.

3.2.3. adminDeleteBrand()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	M.M.3
Tipo	Mutation
Llamada	adminDeleteBrand()

Recibe	AdminDeleteBrandInput - Obligatorio {
	brand_id: Int!
	}

Devuelve	String - Obligatorio
Autorización	Grupo administrador

Tabla 14: Resumen del método adminDeleteBrand.

Método de administración que sirve para eliminar la referencia de alguna marca en la base de datos. Comúnmente, se trata de un método necesario ante la posibilidad de creación errónea de alguna marca.

3.2.4. adminCreateCategory()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	M.M.4
Tipo	Mutation
Llamada	adminCreateCategory()

Recibe	AdminCreateCategoryInput - Obligatorio {
	name: String!
	slug: String!
	meta_desc: String!
	parent_id: Int!
	}

Devuelve	category - Obligatorio {
	category_id: ID!
	count: Int!
	name: String!
	meta_desc: String!
	parent: category
	has_child: Boolean
	slug: String!
	brands: [brand]
	models: [model]
	}

Autorización	Grupo administrador

Tabla 15: Resumen del método adminCreateCategory.

Método de administración que sirve para incluir una nueva categoría en la base de datos. Desde un principio, la base de datos cuenta con más de 1000 categorías, importadas con su jerarquía y metadatos desde la base de datos del MVP. No obstante, un marketplace se encuentra en un estado de continuo crecimiento e incorporación de nuevo stock, que requiere a su vez de nuevas categorías para un correcto etiquetado y filtrado.

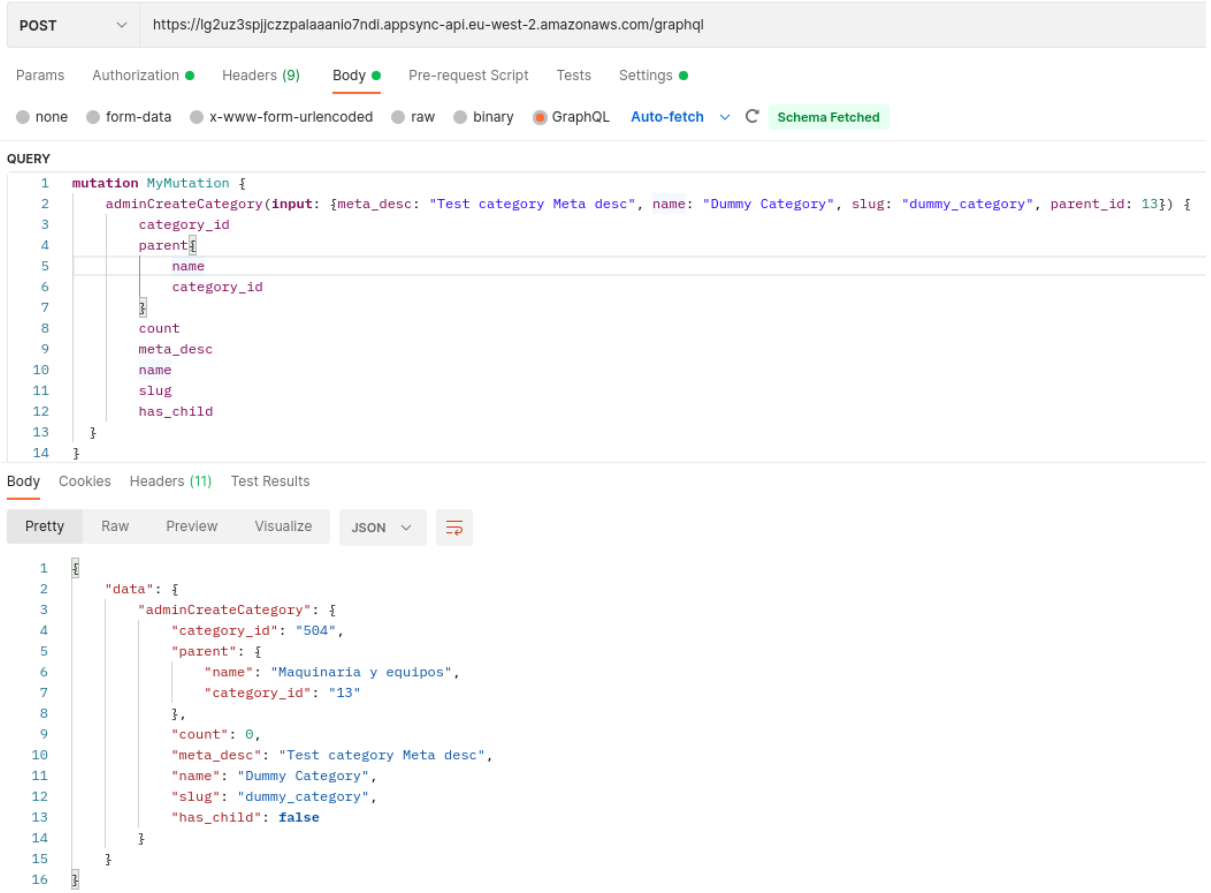

Figura 1: Ejecución adminCreateCategory.

3.2.5. adminUpdateCategory()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	M.M.5
Tipo	Mutation
Llamada	adminUpdateCategory()

Recibe	AdminUpdateCategoryInput - Obligatorio {
	category_id: Int!
	name: String!
	slug: String!
	meta_desc: String!
	}

Devuelve	category - Obligatorio type category { category_id: ID! count: Int! name: String! meta_desc: String!

	parent: category has_child: Boolean slug: String! brands: [brand] models: [model] }
Autorización	Grupo administrador

Tabla 16: Resumen del método adminUpdateCategory.

Método de administración que sirve para actualizar algún campo de alguna categoría en la base de datos. Por ejemplo, esto puede ser para actualizar alguna jerarquía, para cambiar el slug de cara a la mejora del SEO, o incluso para modificar el valor del recuento de activos que se encuentran asociados a una categoría en concreto.

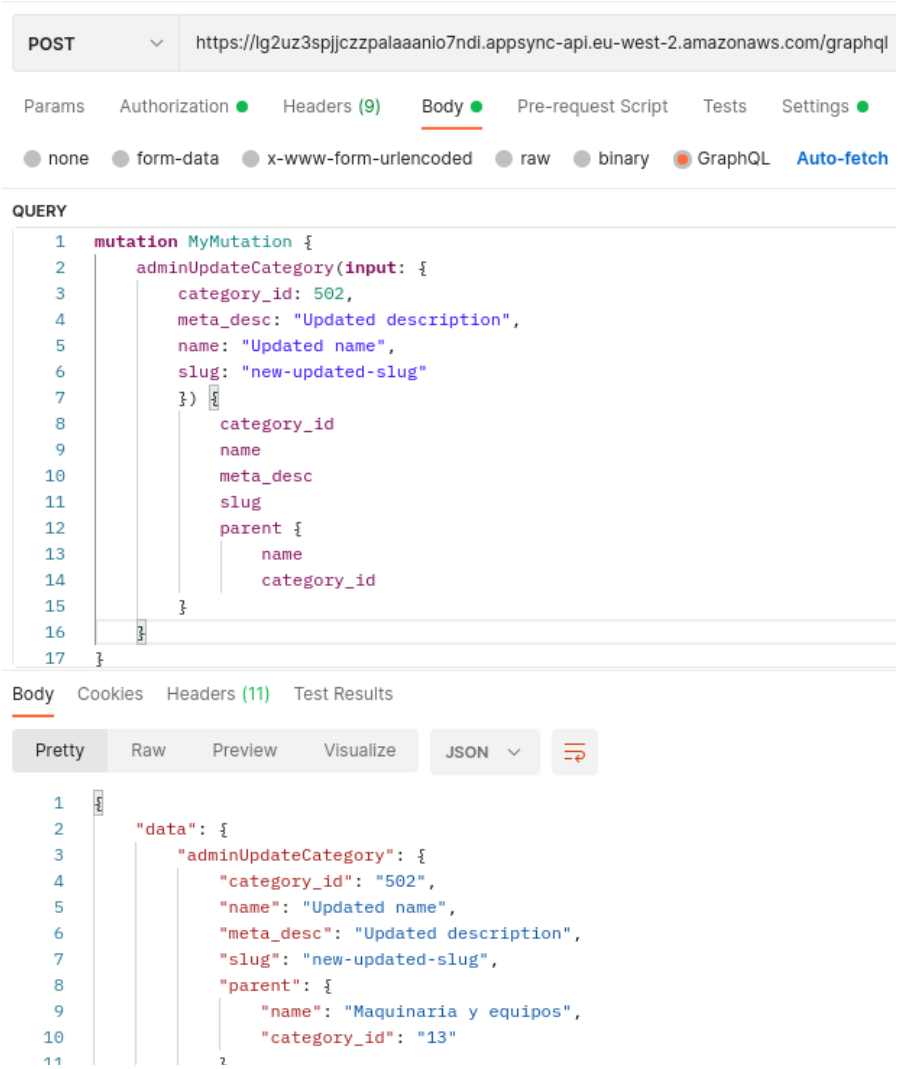

Figura 2: Ejecución adminUpdateCategory.

3.2.6. adminDeleteCategory()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	M.M.6
Tipo	Mutation
Llamada	adminDeleteCategory()

Recibe	AdminDeleteCategoryInput - Obligatorio {
	category_id: Int!
	}

Devuelve	String - Obligatorio
Autorización	Grupo administrador

Tabla 17: Resumen del método adminDeleteCategory.

Método de administración que sirve para eliminar la referencia de alguna categoría en la base de datos. Comúnmente, se trata de un método necesario ante la posibilidad de creación errónea de alguna categoría.

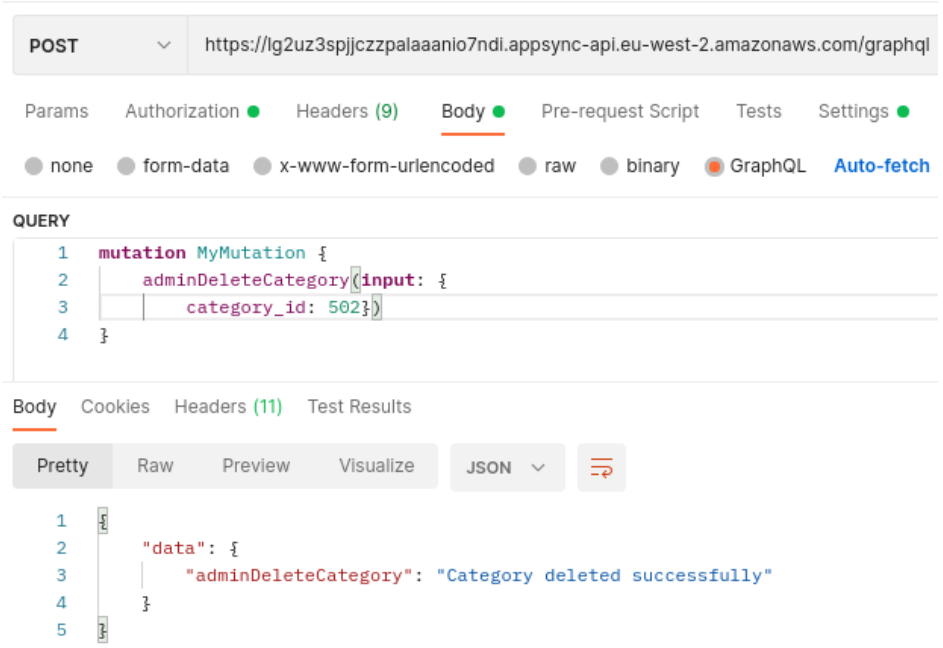

Figura 3: Ejecución adminDeleteCategory.

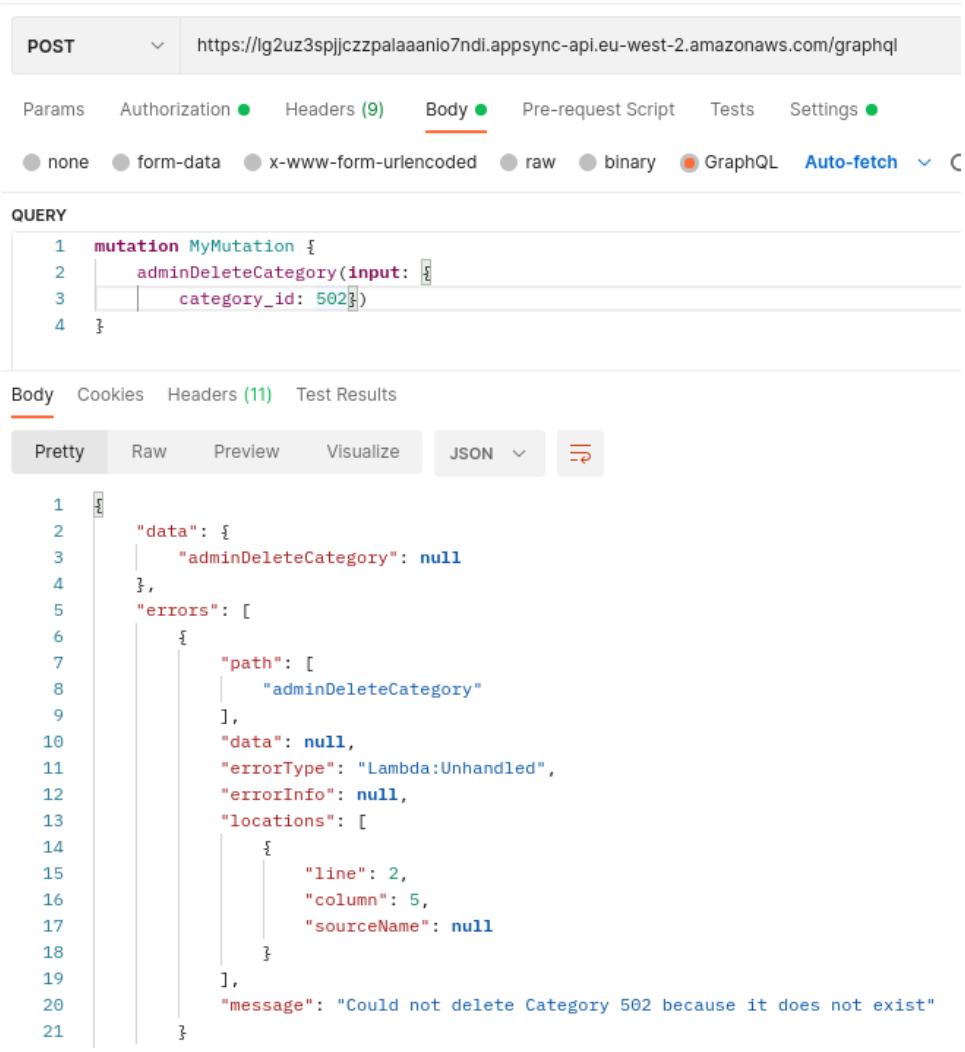

Figura 4: Ejecución adminCreateCategory con respuesta ante error.

3.2.7. adminCreateLocation()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	M.M.7
Tipo	Mutation
Llamada	adminCreateLocation()

Recibe	AdminCreateLocationInput - Obligatorio {
	name: String!
	slug: String!
	meta_desc: String!
	parent_id: Int!
	}

Devuelve	location - Obligatorio { location_id: ID!

	count: Int! name: String! meta_desc: String! parent: location has_child: Boolean! slug: String! }
Autorización	Grupo administrador

Tabla 18: Resumen del método adminCreateLocation.

Método de administración que sirve para incluir una nueva localización en la base de datos. Desde un principio, la base de datos cuenta con más de 3.000 referencias de localidades, importadas con su jerarquía y metadatos desde la base de datos del MVP, con una jerarquía de árbol que llega a presentar una granularidad de municipio. De hecho, de los más de 8.100 municipios existentes en España, Profesiolan presenta activos en más de 3.000. No obstante, un marketplace se encuentra en un estado de continuo crecimiento e incorporación de nuevo stock, que requiere a su vez de nuevas localidades para un correcto etiquetado y filtrado.

En cualquier caso, el diseño de esta tabla permite la generación de jerarquías, pudiendo en un futuro delegar la totalidad de la granularidad de ubicación a una API como Google Maps, dejando a Profesiolan únicamente la responsabilidad de mapear los usuarios a una jerarquía más holgada como puede ser la de identificación provincial.

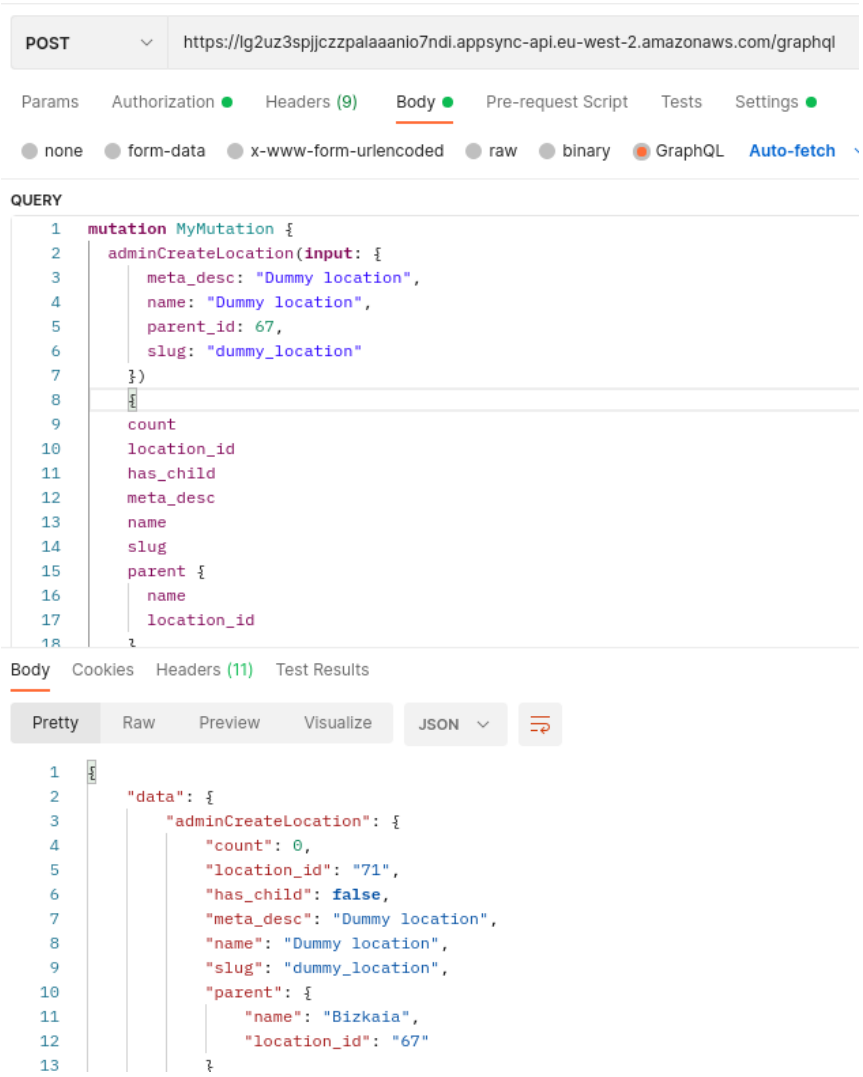

Figura 5: Ejecución adminCreateLocation.

3.2.8. adminUpdateLocation()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	M.M.8
Tipo	Mutation
Llamada	adminUpdateLocation()

Recibe	AdminUpdateLocationInput - Obligatorio {
	location_id: Int!
	name: String!
	slug: String!
	meta_desc: String!
	}

Devuelve	location - Obligatorio {

	location_id: ID! count: Int! name: String! meta_desc: String! parent: location has_child: Boolean! slug: String! }
Autorización	Grupo administrador

Tabla 19: Resumen del método adminUpdateLocation.

Método de administración que sirve para actualizar algún campo de alguna localización en la base de datos. Por ejemplo, esto puede ser para actualizar alguna jerarquía, para cambiar el slug de cara a la mejora del SEO, o incluso para modificar el valor del recuento de activos que se encuentran asociados a una localización en concreto.

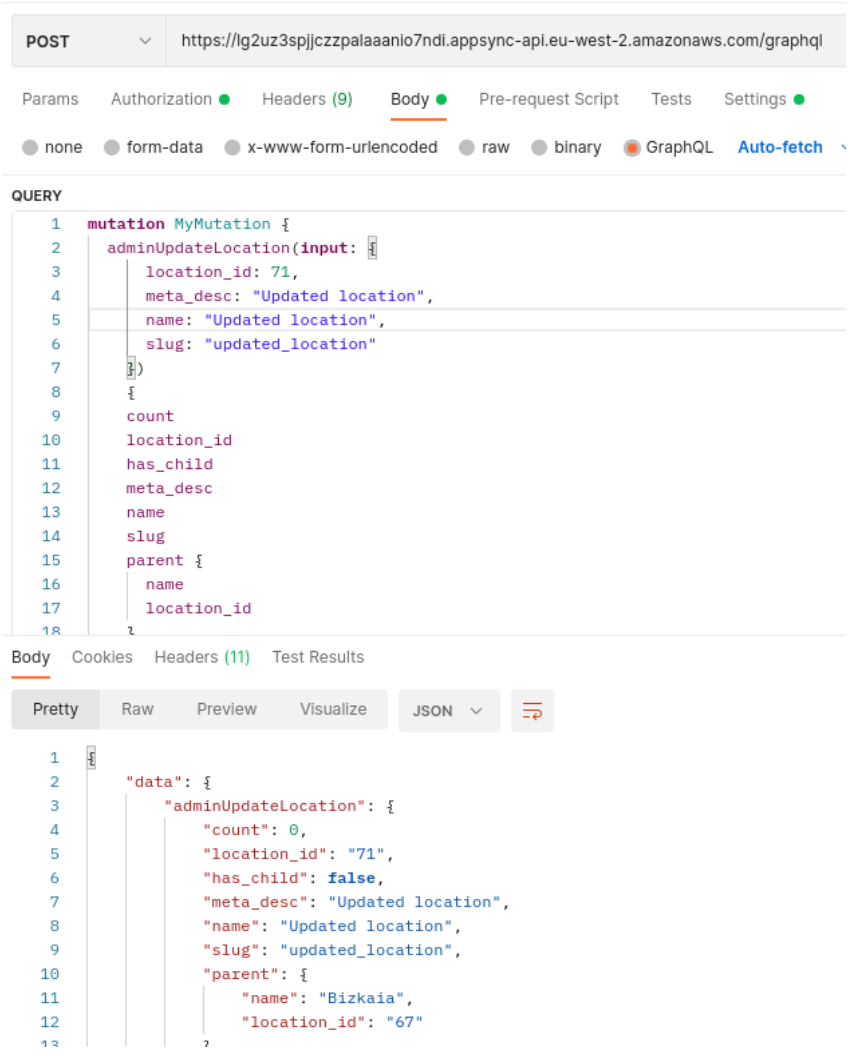

Figura 6: Ejecución adminUpdateLocation.

3.2.9. adminDeleteLocation()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	M.M.9
Tipo	Mutation
Llamada	adminDeleteLocation()

Recibe	AdminDeleteLocationInput - Obligatorio {
	location_id: Int!
	}

Devuelve	String - Obligatorio
Autorización	Grupo administrador

Tabla 20: Resumen del método adminDeleteLocation.

Método de administración que sirve para eliminar la referencia de alguna localización en la base de datos. Comúnmente, se trata de un método necesario ante la posibilidad de creación errónea de alguna localización.

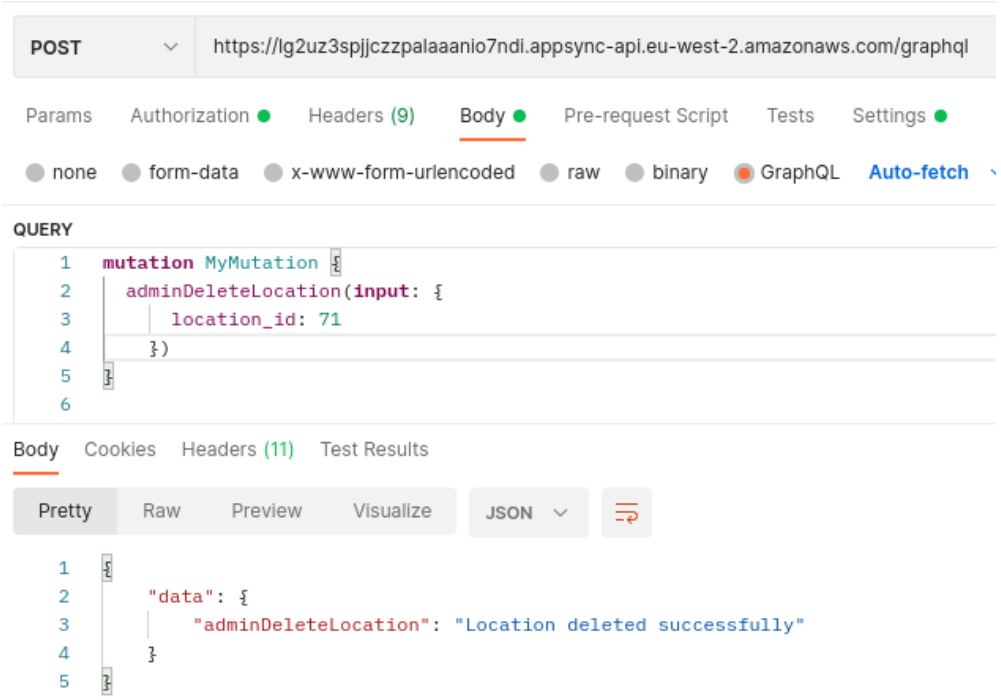

Figura 7: Ejecución adminDeleteLocation.

3.2.10. adminCreateModel()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	M.M.10
Tipo	Mutation
Llamada	adminCreateModel()

Recibe	AdminCreateModelInput - Obligatorio {
	name: String
	count: Int
	slug: String
	brand_id: Int
	category_id: [Int!]!
	}

Devuelve	model - Obligatorio {
	model_id: ID!
	name: String!
	count: Int!
	slug: String!
	categories: [category]!
	brand: brand
	}

Autorización	Grupo administrador

Tabla 21: Resumen del método adminCreateModel.

Método de administración que sirve para incluir una nueva referencia de un modelo en la base de datos. Desde un principio, la base de datos cuenta con más de 2.000 referencias de modelos, importadas con su jerarquía y metadatos desde la base de datos del MVP, que a su vez guarda una relación con la marca fabricante. No obstante, un marketplace se encuentra en un estado de continuo crecimiento e incorporación de nuevo stock, que requiere a su vez de nuevas marcas y modelos para un correcto etiquetado y filtrado.

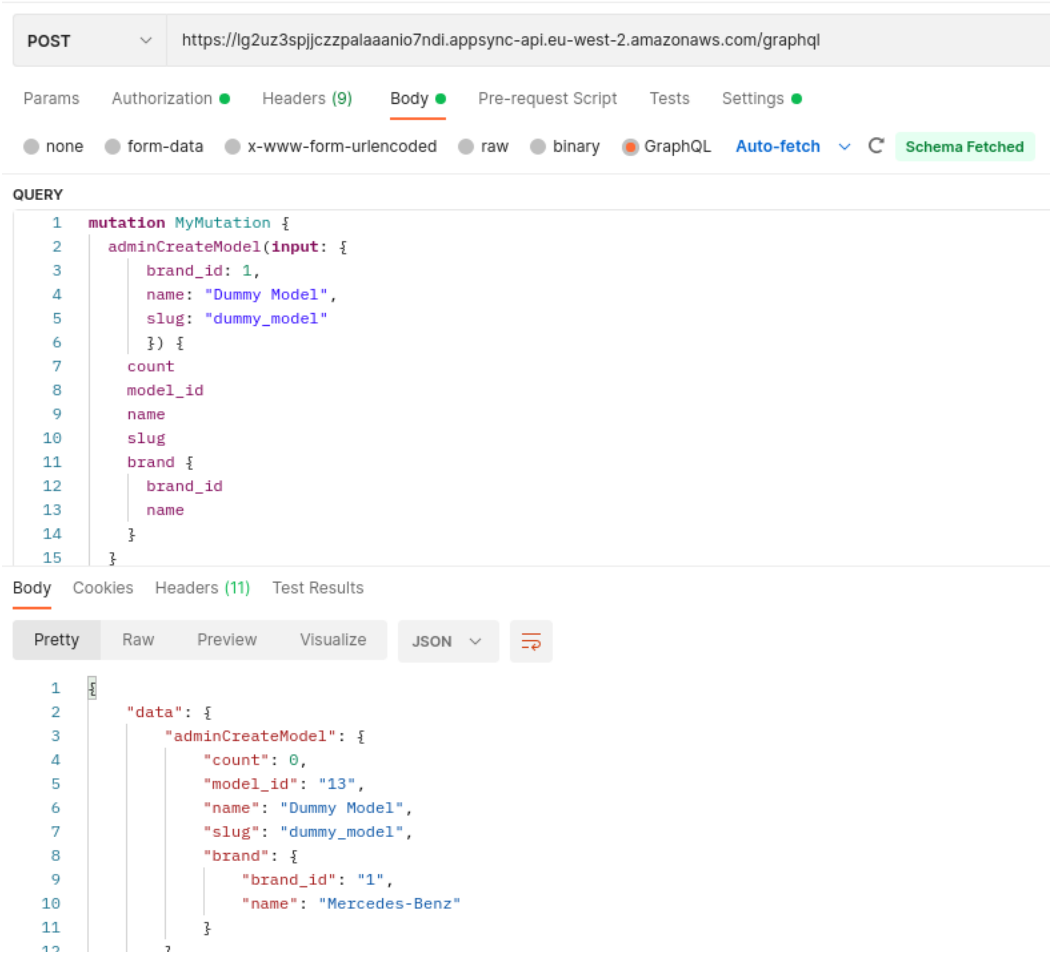

Figura 8: Ejecución adminCreateModel.

3.2.11. adminUpdateModel()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	M.M.11
Tipo	Mutation
Llamada	adminUpdateModel()

Recibe	AdminUpdateModelInput - Obligatorio {
	model_id: ID!
	name: String
	count: Int
	slug: String
	}

Devuelve	model - Obligatorio { model_id: ID! name: String! count: Int!

	slug: String! categories: [category]! brand: brand }
Autorización	Grupo administrador

Tabla 22: Resumen del método adminUpdateModel.

Método de administración que sirve para actualizar algún campo de algún modelo en la base de datos. Esto puede ser principalmente para cambiar el slug de cara a la mejora del SEO, o incluso para modificar el valor del recuento de activos que se encuentran asociados a ese modelo en concreto.

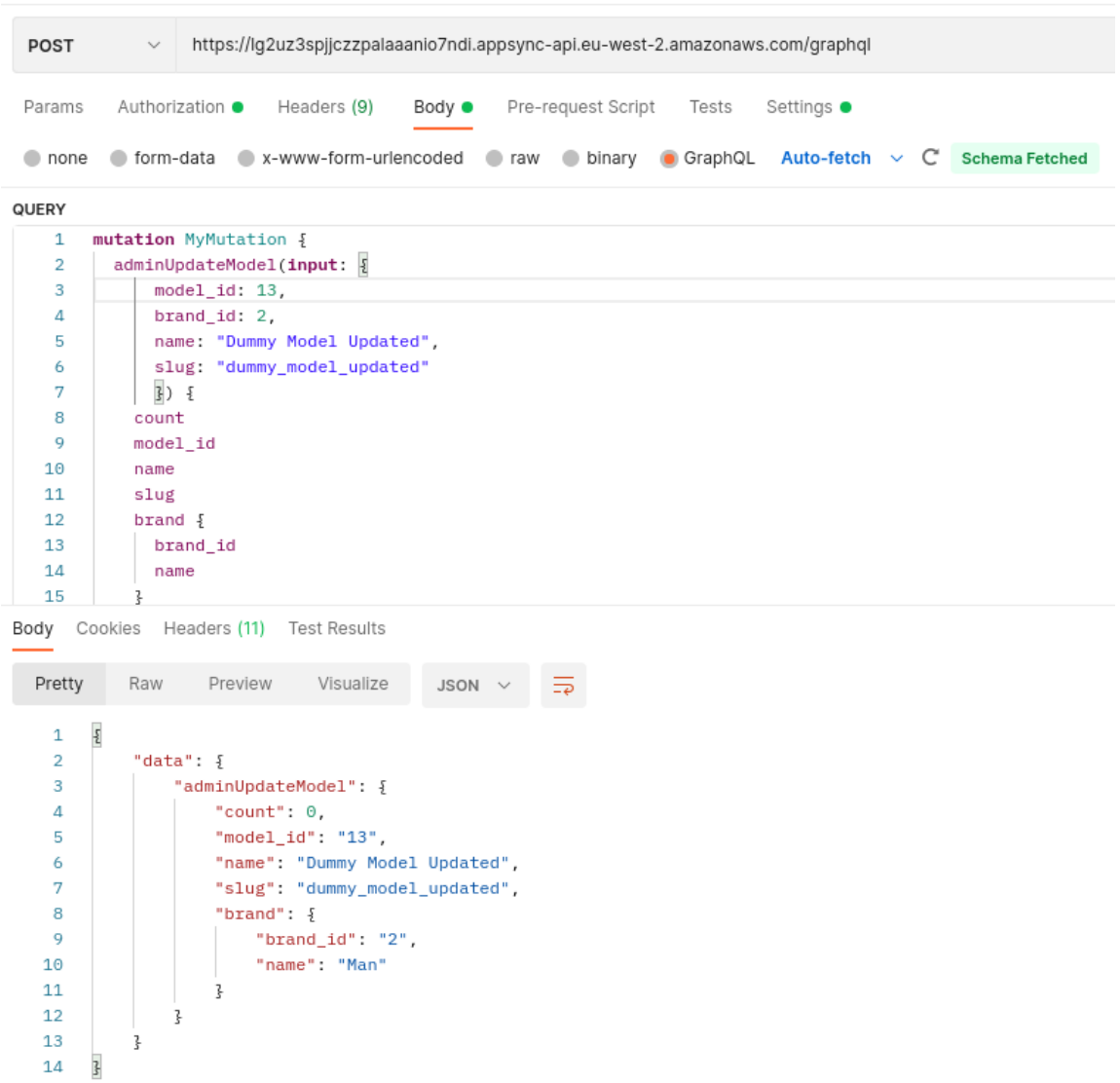

Figura 9: Ejecución adminUpdateModel.

3.2.12. adminDeleteModel()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	M.M.12
Tipo	Mutation
Llamada	adminDeleteModel()

Recibe	AdminDeleteModelInput - Obligatorio {
	model_id: ID!
	}

Devuelve	String - Obligatorio
Autorización	Grupo administrador

Tabla 23: Resumen del método adminDeleteModel.

Método de administración que sirve para eliminar la referencia de algún modelo en la base de datos. Comúnmente, se trata de un método necesario ante la posibilidad de creación errónea de algún modelo.

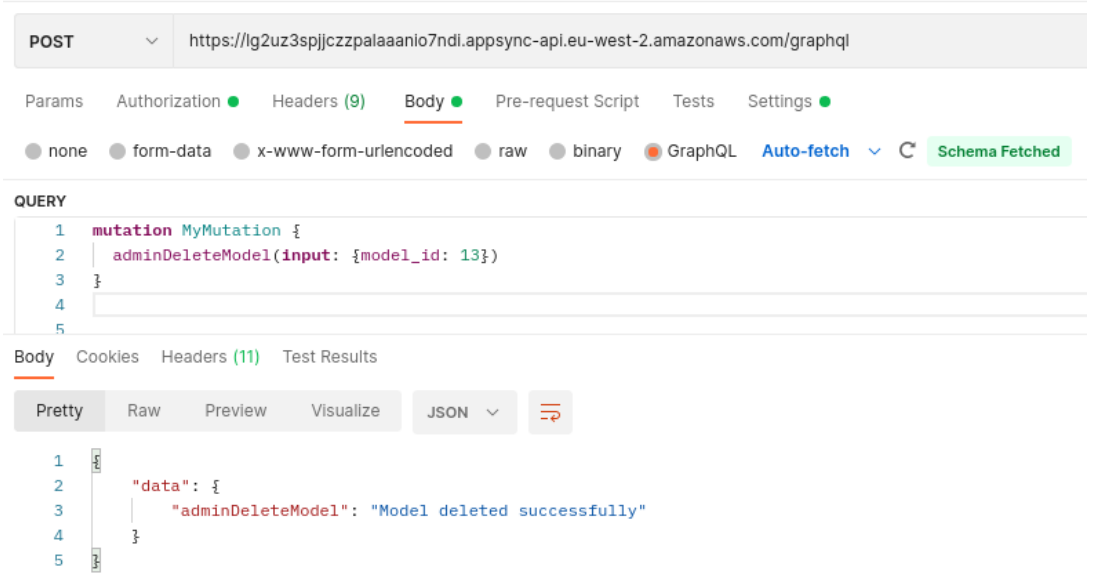

Figura 10: Ejecución adminDeleteModel.

3.2.13. adminUpdateUser()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	M.U.1
Tipo	Mutation
Llamada	updateUser()

Recibe	AdminUpdateUserInput - Obligatorio {
	user_id: ID!
	first_name: String!
	last_name: String!
	latitude: Float
	longitude: Float
	}

Devuelve	user - Obligatorio {
	user_id: ID!
	user_name: String
	first_name: String
	last_name: String
	email: AWSEmail
	password: String
	registered: AWSDateTime
	slug: String
	status: Boolean
	is_verified: Boolean
	count: Int
	role: String
	suscription_number: Int
	activation_key: String
	latitude: Float
	longitude: Float
	userAddress: user_address
	userBilling: user_billing
	userMeta: user_meta
	emailVerificationToken: email_verification_token
	roles: [roles]
	}

Autorización	Grupo administrador

Tabla 24: Resumen del método adminUpdateUser.

Método de administración que sirve para actualizar la información de un usuario en la base de datos. Se trata de un método que permite modificar la información incluída en la tabla *user* de un usuario concreto, no permite la modificación de campos pertenecientes a tablas como *user_billing*, *user_address*, *user_meta* o *user_roles*.

La razón detrás de la no existencia de un método de administración para la creación y el borrado de un usuario, se debe a que son tareas que deben ejecutarse desde el panel de AWS, en el entorno de AWS Cognito. Por aspectos de seguridad, la creación de un usuario está únicamente llevada a cabo en el entorno Cognito, así como el borrado de toda la información relativa a un usuario.

Esto a su vez es así debido a que AWS Cognito, como se ha desarrollado a lo largo de este Anexo I y el documento principal de la Memoría Trabajo Fin de Máster, es una potente herramienta completa que gestiona de inicio a fin las tareas de autenticación, autorización, gestión de certificados, almacenamiento en el pool de usuarios de forma segura y garantizando el cumplimiento de todas las regulaciones. Además, a la hora de la creación de un usuario a través del panel de AWS, se cumplen los best-practices relativos a la solicitud de MFA, verificación de correo electrónico mediante una confirmación basada en clic, posibilidad de verificación del teléfono mediante código de seguridad, y obligación de cambio de contraseña con un periodo de vencimiento prudente.

Es por todo ello que se ha concluído en el uso único y completo de AWS Cognito para la creación y el borrado de usuarios Profesiolan.

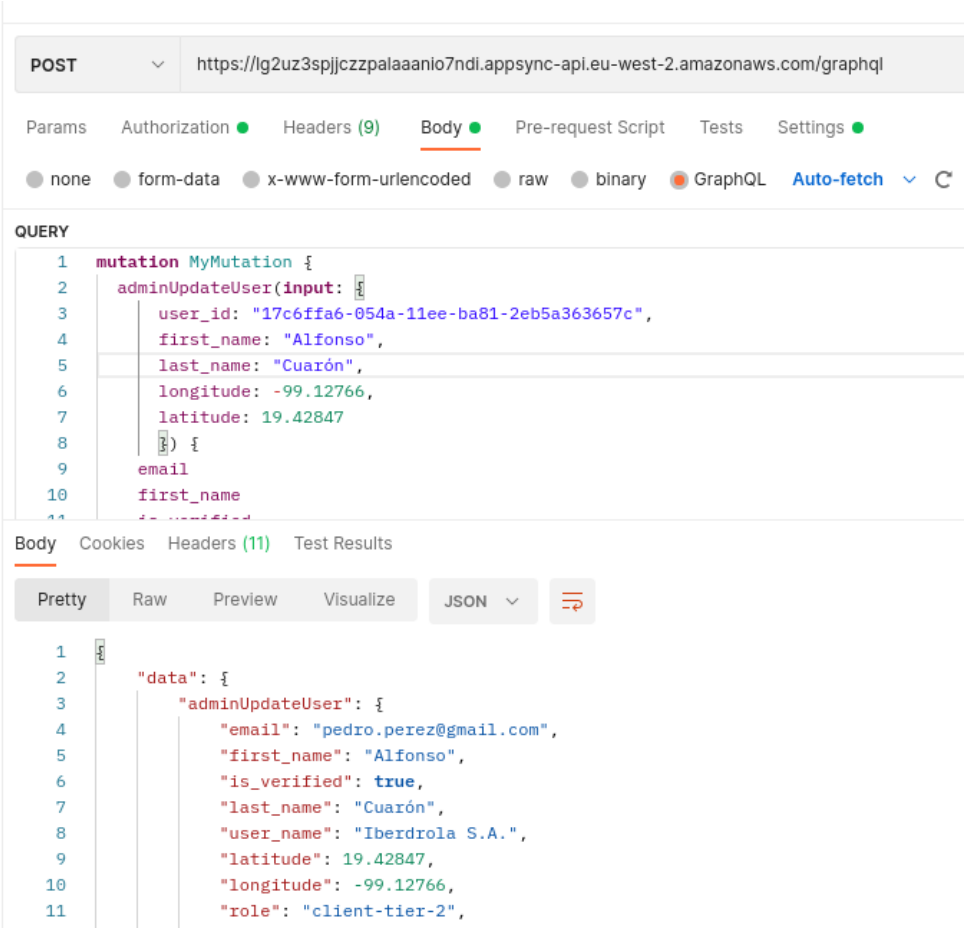

Figura 11: Ejecución adminUpdateUser.

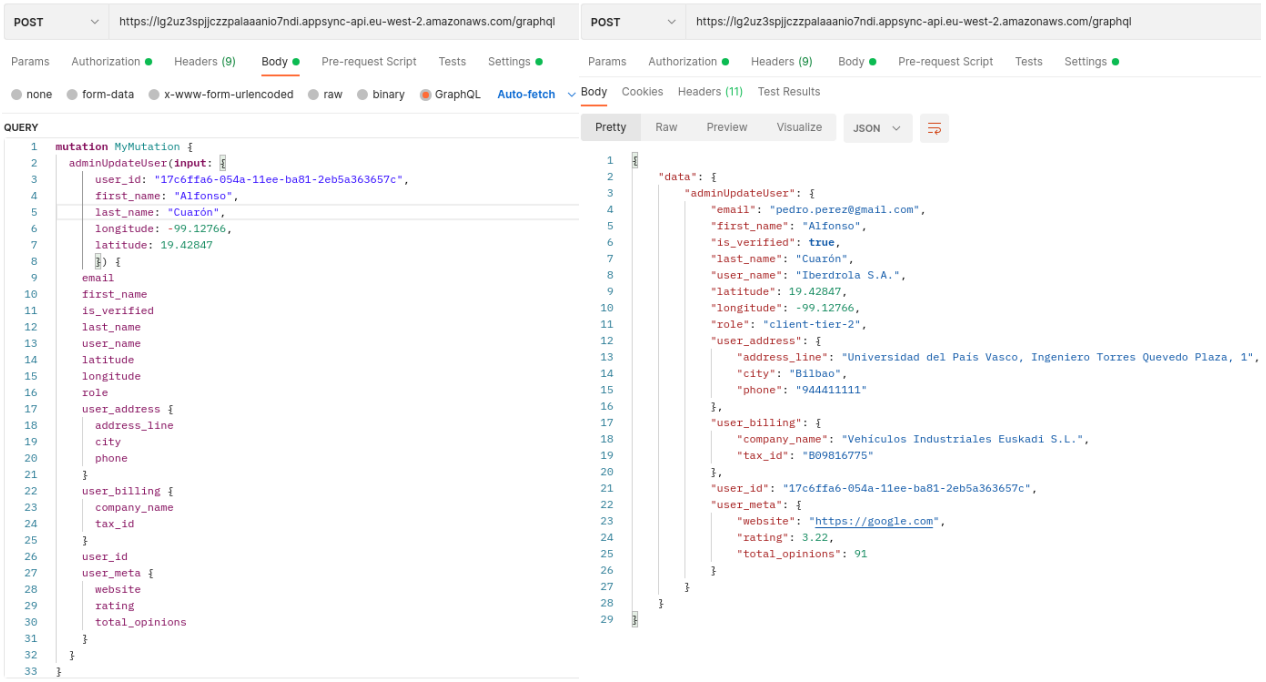

Figura 12: Ejecución adminUpdateUser descripción completa.

3.2.14. adminCreateUserBilling()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	M.U.2
Tipo	Mutation
Llamada	adminCreateUserBilling()

Recibe	AdminCreateUserBillingInput - Obligatorio {
	address: String!
	company_name: String!
	tax_id: String!
	}

Devuelve	user_billing - Obligatorio {
	user_billing_id: ID!
	address: String!
	company_name: String!
	tax_id: String!
	}

Autorización	Grupo administrador

Tabla 25: Resumen del método adminCreateUserBilling.

Método de administración que sirve para crear la información de facturación de un usuario en la base de datos.

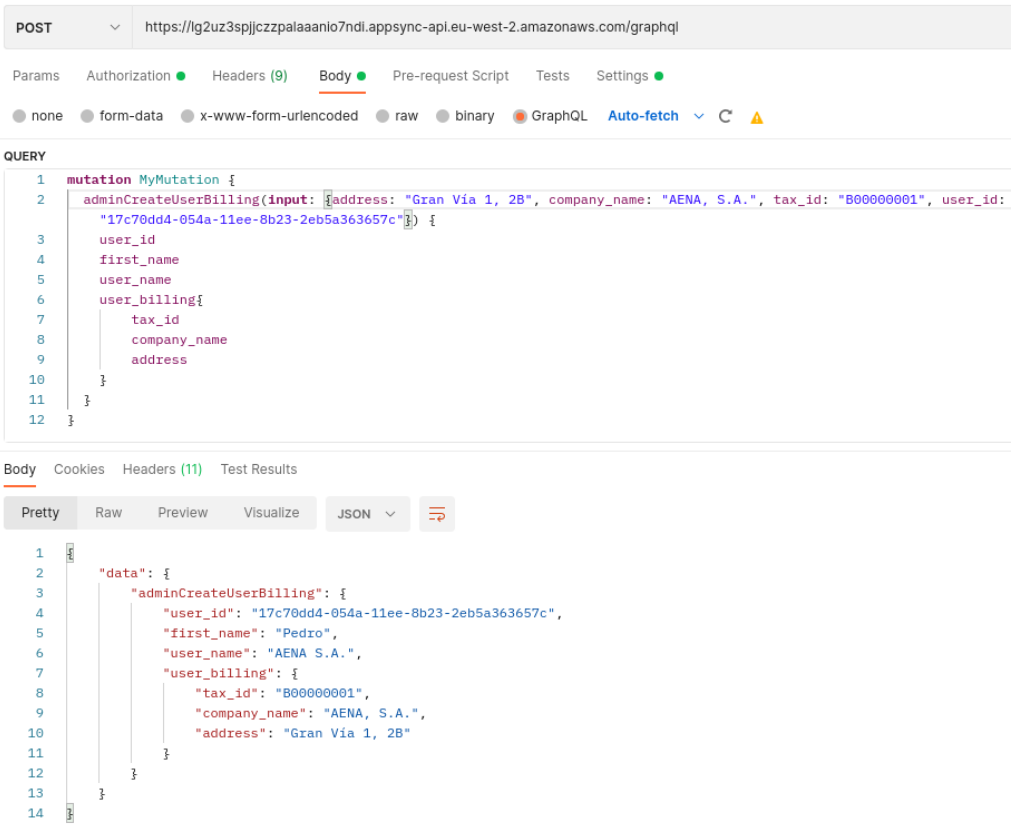

Figura 13: Ejecución adminCreateUserBilling.

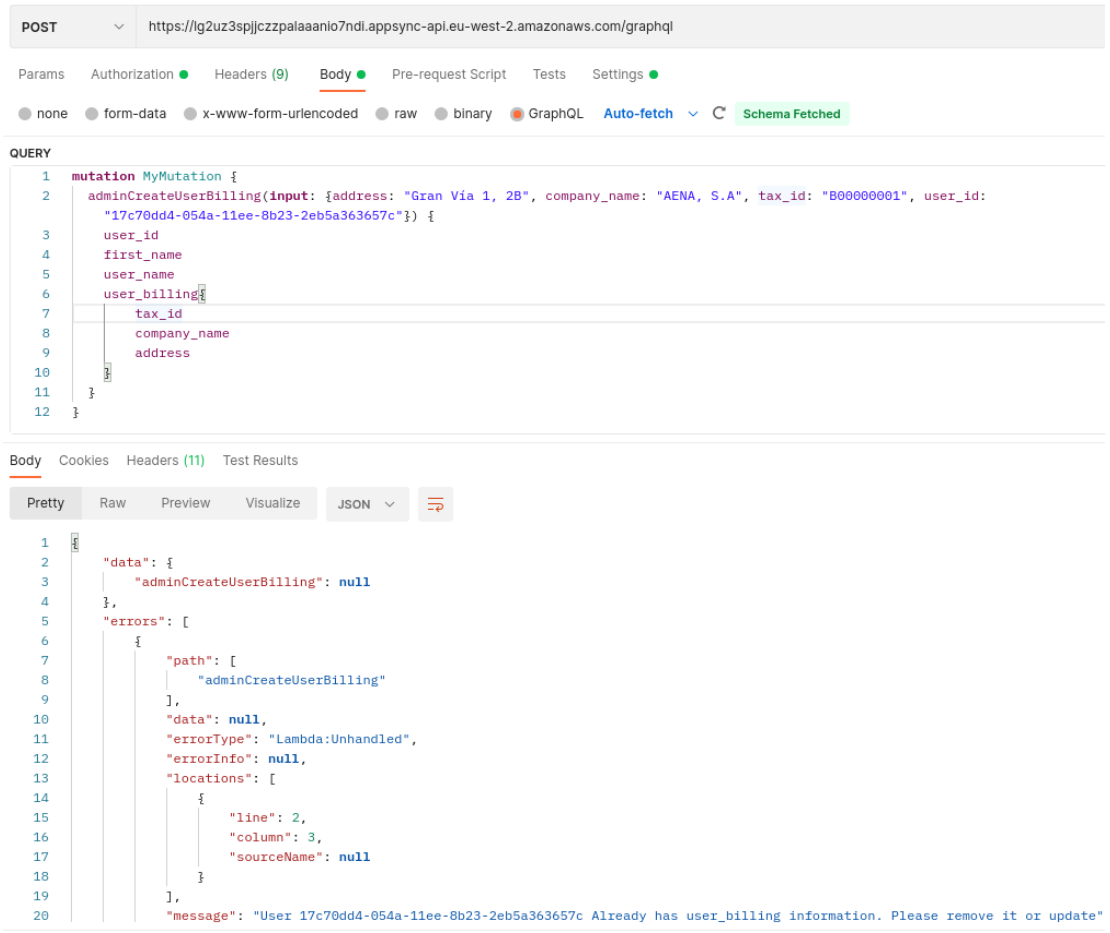

Figura 14: Ejecución adminCreateUserBilling con respuesta ante error.

3.2.15. adminUpdateUserBilling()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	M.U.3
Tipo	Mutation
Llamada	adminUpdateUserBilling()

Recibe	AdminUpdateUserBillingInput - Obligatorio {
	user_billing_id: Int!
	address: String!
	company_name: String!
	tax_id: String!
	}

Devuelve	user_billing - Obligatorio { user_billing_id: ID! address: String! company_name: String! tax_id: String!

	}
Autorización	Grupo administrador

Tabla 26: Resumen del método adminUpdateUserBilling.

Método de administración que sirve para actualizar la información de facturación de un usuario en la base de datos.

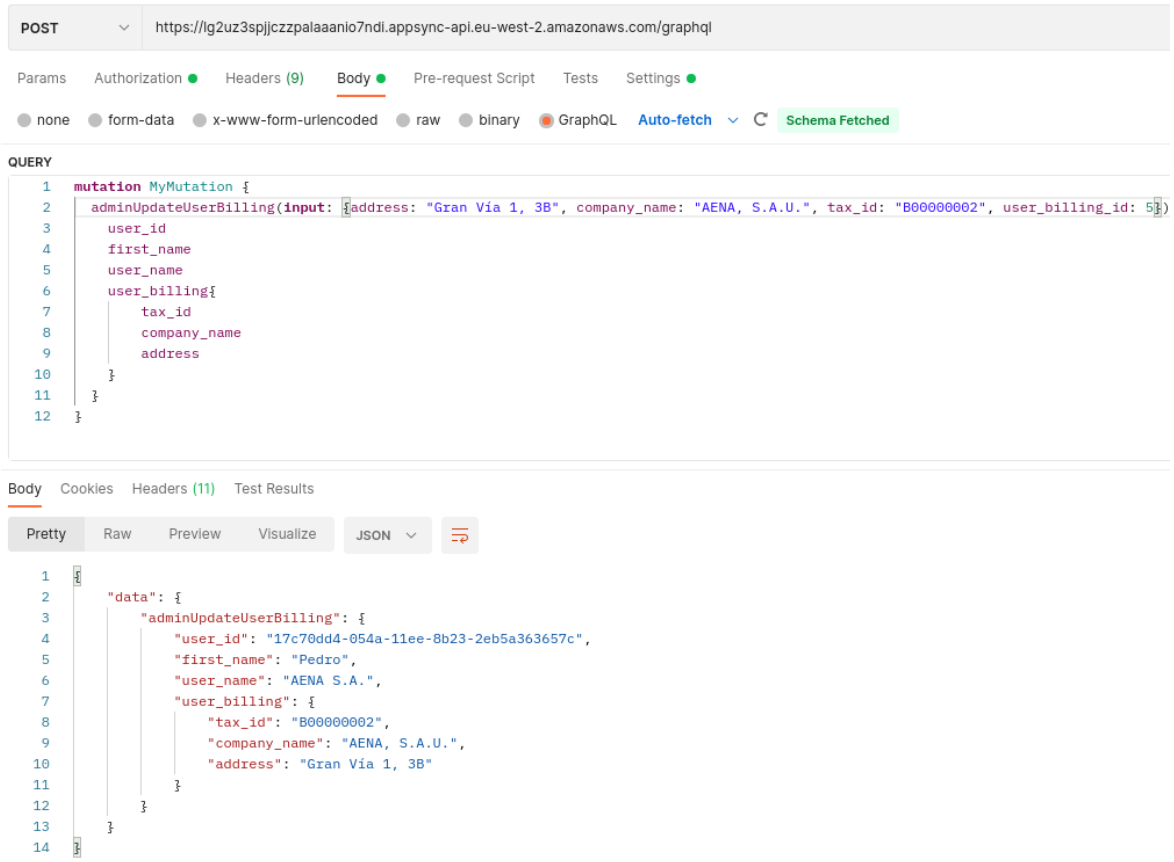

Figura 15: Ejecución adminUpdateUserBilling.

3.2.16. adminDeleteUserBilling()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	M.U.4
Tipo	Mutation
Llamada	adminDeleteUserBilling()
Recibe	AdminDeleteUserBillingInput - Obligatorio { user_billing_id: Int!

	}
Devuelve	String - Obligatorio
Autorización	Grupo administrador

Tabla 27: Resumen del método adminDeleteUserBilling.

Método de administración que sirve para borrar permanentemente la información de facturación de un usuario en la base de datos.

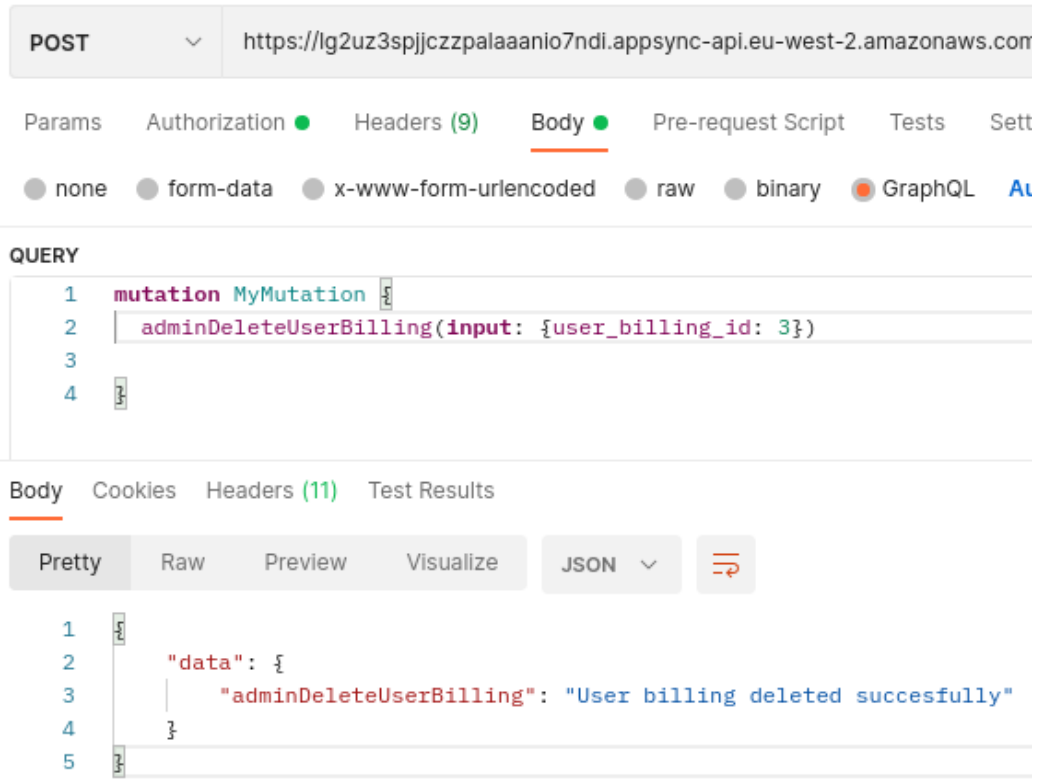

Figura 16: Ejecución adminDeleteUserBilling.

3.2.17. adminCreateUserMeta()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	M.U.5
Tipo	Mutation
Llamada	adminCreateUserMeta()
Recibe	AdminCreateUserMetaInput - Obligatorio { website: String

	linkedin: String facebook: String twitter: String biography: String! supplier: Boolean producer: Boolean ocasional: Boolean realestate: Boolean rating: Int! total_opinions: Int! }

Devuelve	user_meta - Obligatorio {
	user_meta_id: ID!
	website: String
	linkedin: String
	facebook: String
	twitter: String
	biography: String!
	supplier: Boolean
	producer: Boolean
	ocasional: Boolean
	realestate: Boolean
	rating: Int!
	total_opinions: Int!
	}

Autorización	Grupo administrador

Tabla 28: Resumen del método adminCreateUserMeta.

Método de administración que sirve para crear información adicional de un usuario en la base de datos.

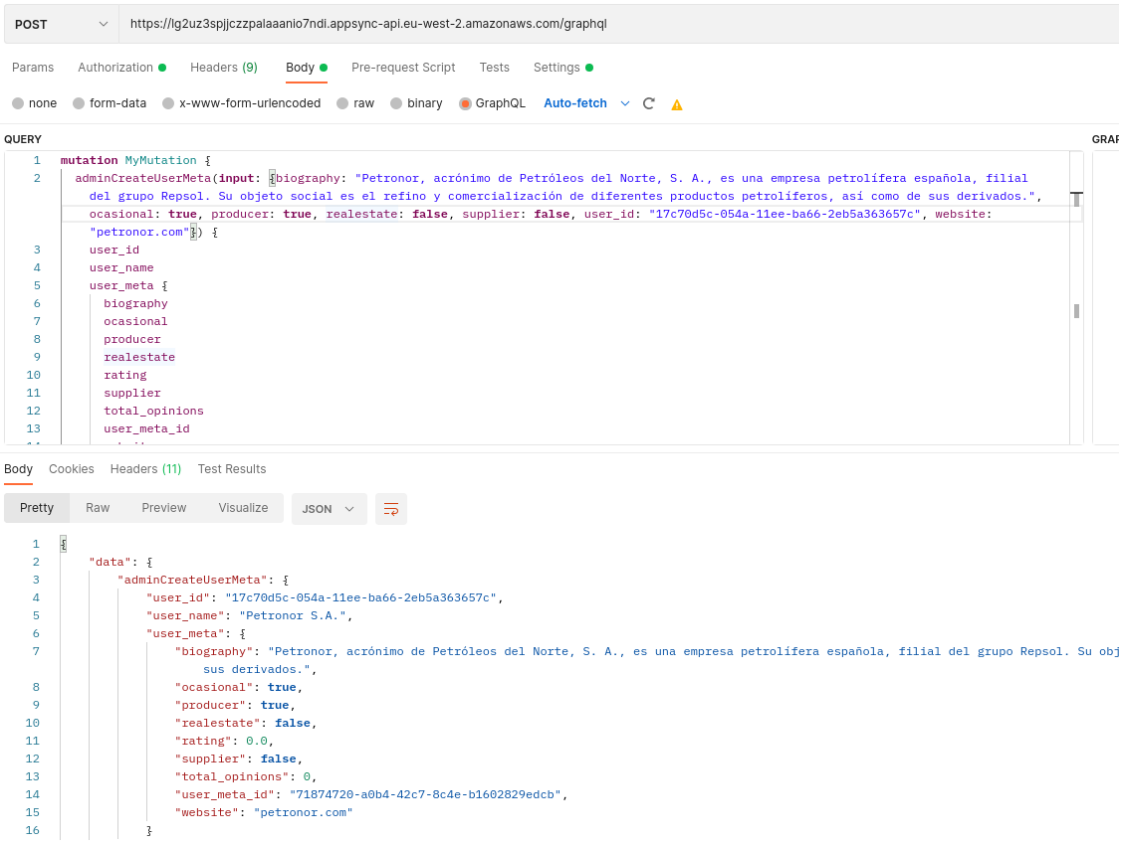

Figura 17: Ejecución adminCreateUserMeta.

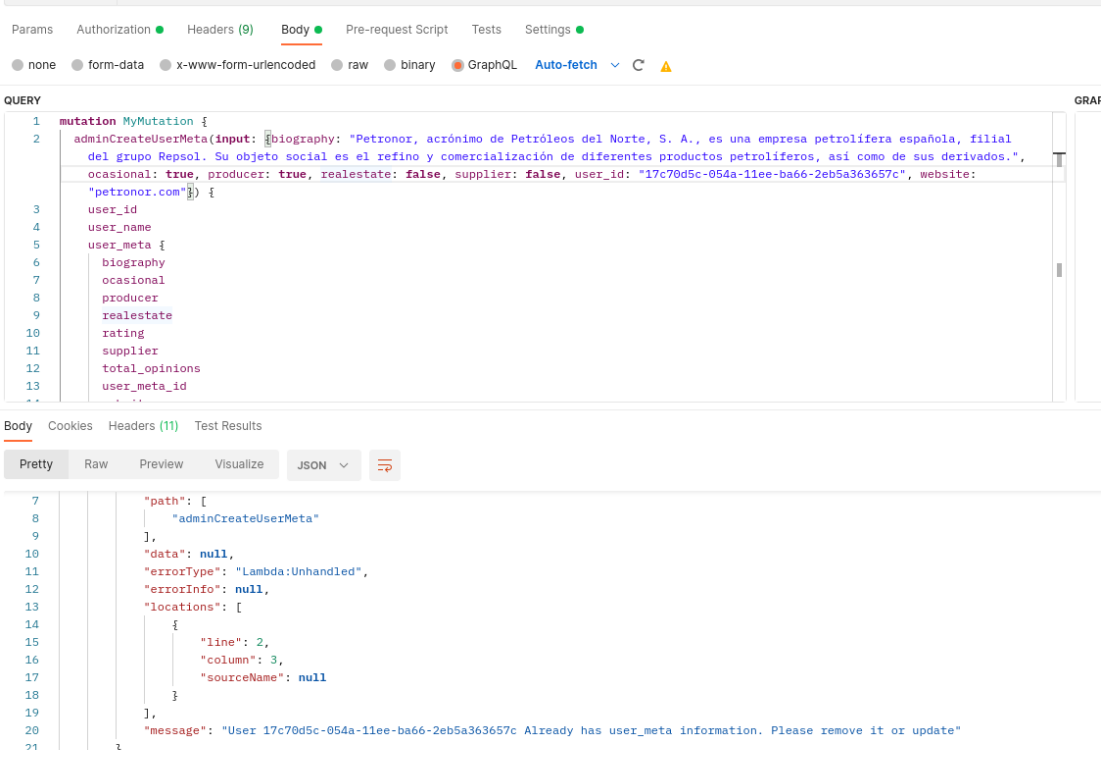

Figura 18: Ejecución adminCreateUserMeta con respuesta ante error.

3.2.18. adminUpdateUserMeta()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	M.U.6
Tipo	Mutation
Llamada	adminUpdateUserMeta()

Recibe	AdminUpdateUserMetaInput - Obligatorio {
	user_meta_id: ID!
	website: String
	linkedin: String
	facebook: String
	twitter: String
	biography: String!
	supplier: Boolean
	producer: Boolean
	ocasional: Boolean
	realestate: Boolean
	rating: Int!
	total_opinions: Int!
	}

Devuelve	user_meta - Obligatorio {
	user_meta_id: ID!
	website: String
	linkedin: String
	facebook: String
	twitter: String
	biography: String!
	supplier: Boolean
	producer: Boolean
	ocasional: Boolean
	realestate: Boolean
	rating: Int!
	total_opinions: Int!
	}

Autorización	Grupo administrador

Tabla 29: Resumen del método adminUpdateUserMeta.

Método de administración que sirve para actualizar algún campo de información adicional de un usuario en la base de datos.

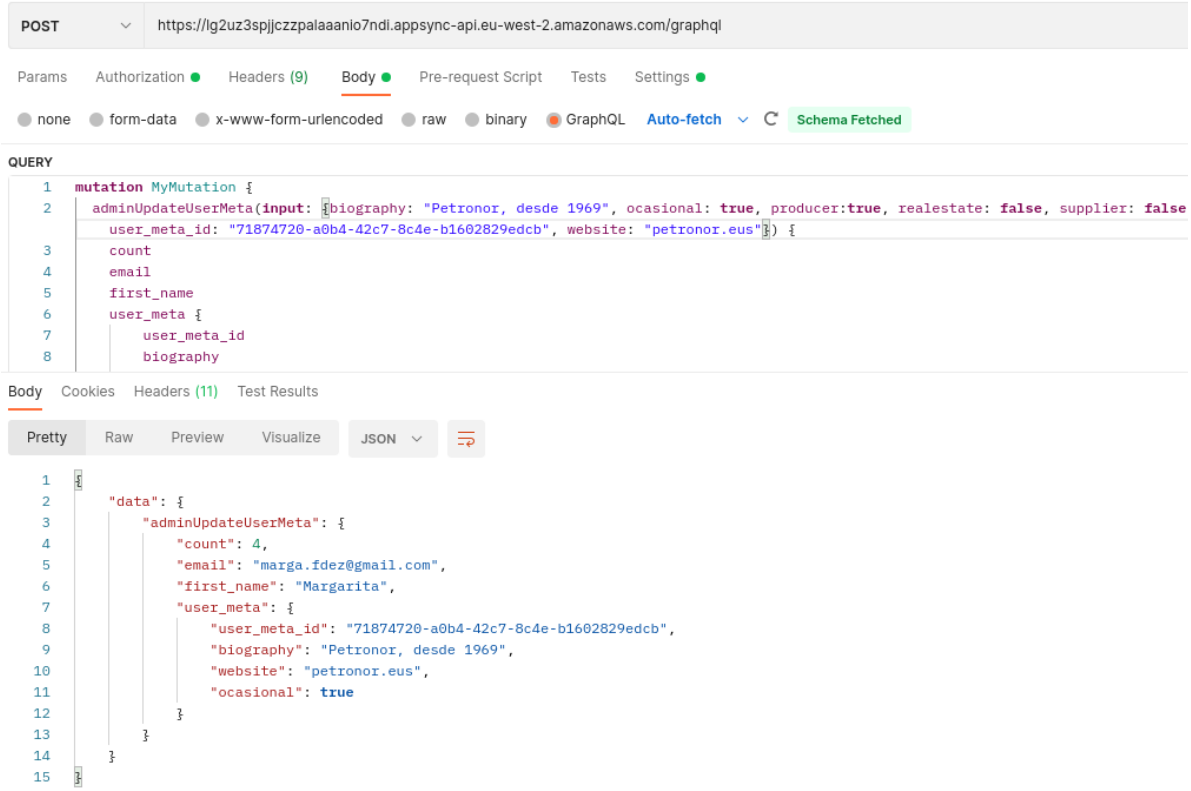

Figura 19: Ejecución adminUpdateUserMeta.

3.2.19. adminDeleteUserMeta()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	M.U.7
Tipo	Mutation
Llamada	adminDeleteUserMeta()

Recibe	AdminDeleteUserMetaInput - Obligatorio {
	user_meta_id: ID!
	}

Devuelve	String - Obligatorio
Autorización	Grupo administrador

Tabla 30: Resumen del método adminDeleteUserMeta.

Método de administración que sirve para eliminar información adicional de un usuario en la base de datos.

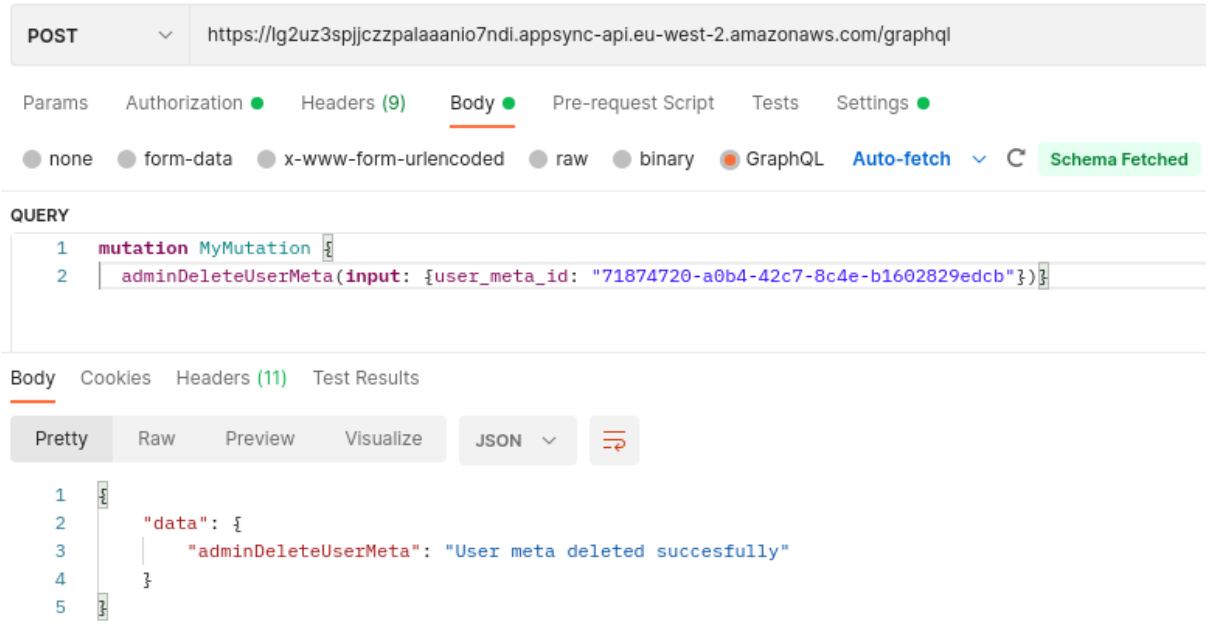

Figura 20: Ejecución adminCreateUserMeta.

3.2.20. adminCreateUserAddress()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	M.U.8
Tipo	Mutation
Llamada	adminCreateUserAddress()

Recibe	AdminCreateUserAddressInput - Obligatorio {
	address_line: String!
	address_ine2: String
	city: String!
	postal_code: String!
	district: String!
	country: String!
	phone: String!
	phone2: String
	}

Devuelve	user_address - Obligatorio { user_address_id: ID! address_line: String! address_ine2: String city: String! postal_code: String! district: String! country: String! phone: String!

	phone2: String }
Autorización	Grupo administrador

Tabla 31: Resumen del método adminCreateUserAddress.

Método de administración que sirve para crear información sobre direcciones de un usuario en la base de datos.

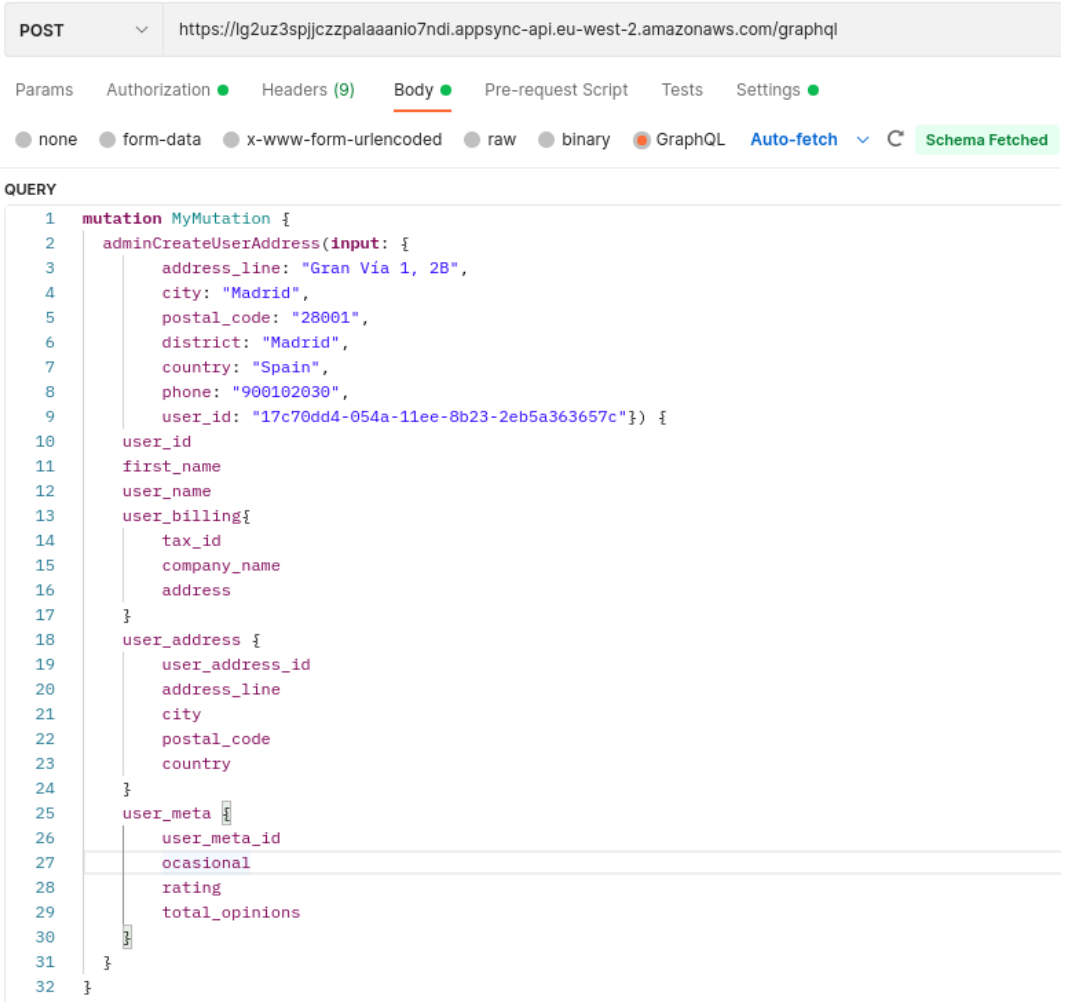

Figura 21: Ejecución adminCreateUserAddress I.

```
{
    "data": {
        "adminCreateUserAddress": {
            "user_id": "17c70dd4-054a-11ee-8b23-2eb5a363657c",
            "first_name": "Pedro",
            "user_name": "AENA S.A.",
            "user_billing": {
                "tax_id": "B00000002",
                "company_name": "AENA, S.A.U.",
                "address": "Gran Vía 1, 3B"
            },
            "user_address": {
                "user_address_id": "df671a8a-07bd-4d6f-8d72-d1bb7a8dcc5b",
                "address_line": "Gran Vía 1, 2B",
                "city": "Madrid",
                "postal_code": "28001",
                "country": "Spain"
            },
            "user_meta": null
        }
    }
}
```


Figura 22: Ejecución adminCreateUserAddress II.

3.2.21. adminUpdateUserAddress()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	M.U.9
Tipo	Mutation
Llamada	adminUpdateUserAdress()

Recibe	AdminUpdateUserAddressInput - Obligatorio {
	user_address_id: ID!
	address_line: String!
	address_ine2: String
	city: String!
	postal_code: String!
	district: String!
	country: String!
	phone: String!
	phone2: String
	}

Devuelve	user_address - Obligatorio {
	user_address_id: ID!
	address_line: String!
	address_ine2: String
	city: String!
	postal_code: String!
	district: String!
	country: String!
	phone: String!
	phone2: String
	}

Autorización	Grupo administrador

Tabla 32: Resumen del método adminUpdateUserAddress.

Método de administración que sirve para actualizar la información sobre direcciones de un usuario en la base de datos.

```
mutation MyMutation {
  adminUpdateUserAddress(input: {
      address_line: "Gran Vía 1, 5B",
      city: "Madrid",
      postal_code: "28001",
      district: "Madrid",
      country: "Spain",
      phone: "900102030",
      user_address_id: "df671a8a-07bd-4d6f-8d72-d1bb7a8dcc5b"}) {
    user_id
    first_name
    user_name
    user_billing{
      tax_id
```

Body Cookies Headers (11) Test Results

Pretty Raw Preview Visualize JSON


```
{
    "data": {
        "adminUpdateUserAddress": {
            "user_id": "17c70dd4-054a-11ee-8b23-2eb5a363657c",
            "first_name": "Pedro",
            "user_name": "AENA S.A.",
            "user_billing": {
                "tax_id": "B00000002",
                "company_name": "AENA, S.A.U.",
                "address": "Gran Vía 1, 3B"
            },
            "user_address": {
                "user_address_id": "df671a8a-07bd-4d6f-8d72-d1bb7a8dcc5b",
                "address_line": "Gran Vía 1, 5B",
                "city": "Madrid",
                "postal_code": "28001",
                "country": "Spain"
            },
            "user_meta": null
```


Figura 23: Ejecución adminUpdateUserMeta.

3.2.22. adminDeleteUserAddress()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	M.U.10
Tipo	Mutation
Llamada	adminDeleteUserAddress()

Recibe	AdminDeleteUserAddressInput - Obligatorio {
	user_address_id: ID!
	}

Devuelve	String - Obligatorio
Autorización	Grupo administrador

Tabla 33: Resumen del método adminDeleteUserAddress.

Método de administración que sirve para eliminar permanentemente la información sobre direcciones de un usuario en la base de datos.

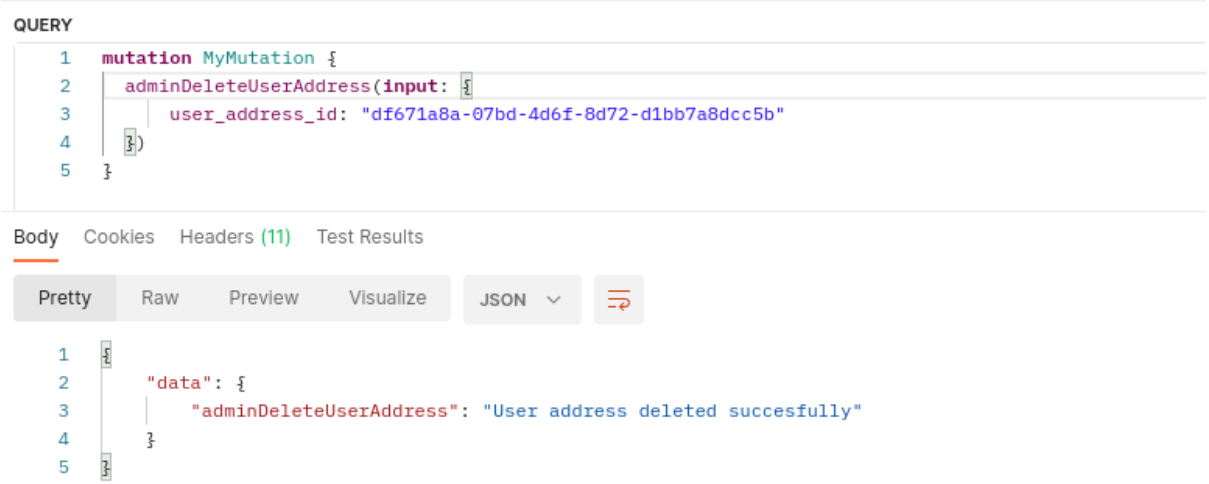

Figura 24: Ejecución adminDeleteUserMeta.

3.2.23. adminCreatePost()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	M.K.1
Tipo	Mutation
Llamada	adminCreatePost()
Recibe	AdminCreatePostInput - Obligatorio { user_id: ID! title: String! content: String! use_condition: String! description_condition: String! latitude: Float! longitude: Float!

	price: Float! category_id: [Int!]! }

Devuelve	PostDataOutput - Obligatorio {
	post_id: ID!
	title: String!
	content: String!
	published: String!
	slug: String!
	use_condition: String!
	description_condition: String!
	latitude: Float!
	longitude: Float!
	status: String!
	stock: Int!
	price: Float!
	has_price_insight: Boolean!
	user: UserDataOutput
	professional: Boolean!
	categories: [category]
	}

Autorización	Grupo administrador

Tabla 34: Resumen del método adminCreatePost.

QUERY

```
mutation MyMutation {
  adminCreatePost(
      input: {
          title: "Dummy Post Updated",
          content: "Dummy content text area that will be parsed and saved in a centralized manner",
          description_condition: "Prácticamente nuevo",
          latitude: 41.3811388,
          longitude: 2.1770898,
          price: 99999,
          use_condition: "Prácticamente nuevo",
          user_id: "921f2bd6-d04a-4ffc-af85-03b90963574b",
          category_id: [437, 439]
          }
          )
        {
    title
    content
    price
    published
    description_condition
    has_price_insight
    latitude
    longitude
    post_id
    user{
        user_id
        first_name
    }
    categories{
        category_id
        name
    }
  }
```

Body Cookies Headers (11) Test Results

Figura 25: Ejecución adminCreatePost I.

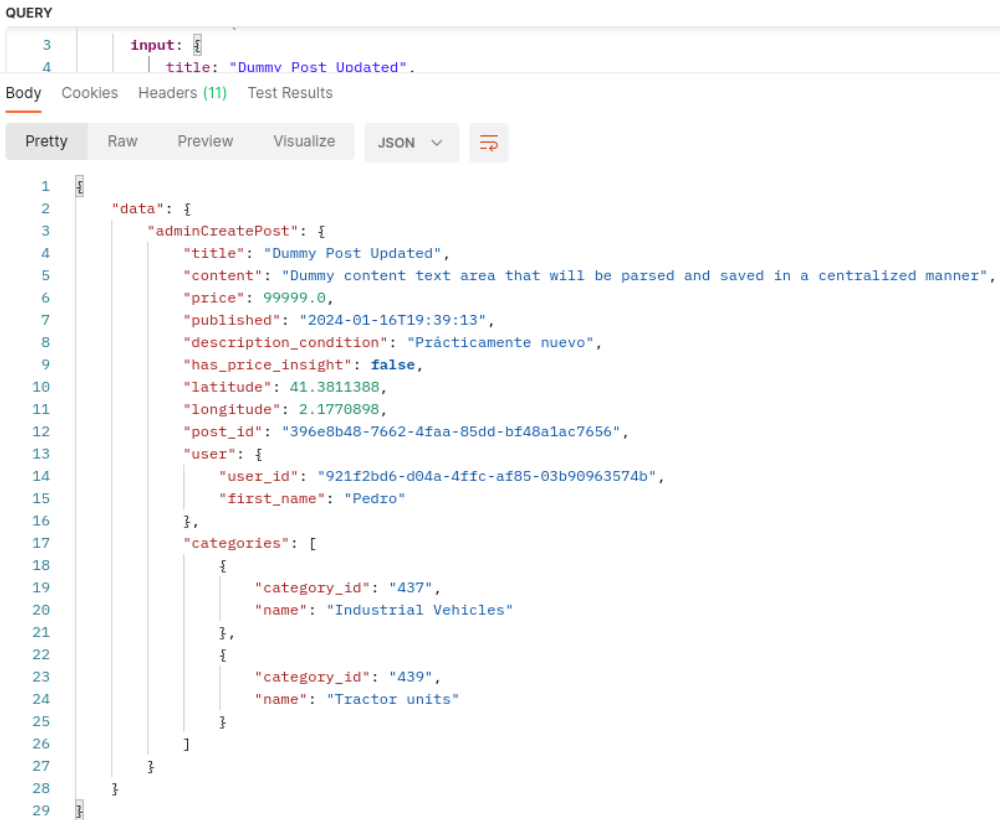

Figura 26: Ejecución adminCreatePost II.

3.2.24. adminUpdatePost()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	M.K.2
Tipo	Mutation
Llamada	adminUpdatePost()

Recibe	AdminUpdatePostInput - Obligatorio {
	post_id: ID!
	title: String!
	content: String!
	slug: String!
	use_condition: String!
	description_condition: String!
	latitude: Float!
	longitude: Float!
	price: Float!
	category_id: [Int!]!
	}

Devuelve	PostDataOutput - Obligatorio { post_id: ID! title: String!

	content: String! published: String! slug: String! use_condition: String! description_condition: String! latitude: Float! longitude: Float! status: String! stock: Int! price: Float! has_price_insight: Boolean! user: UserDataOutput professional: Boolean! categories: [category] }
Autorización	Grupo administrador

Tabla 35: Resumen del método adminUpdatePost.

QUERY

```
post_id: "430d4b39-fdd6-4b70-8ee2-fa0ce0fd1eb1",
title: "Last test post",
content: "Dummy content text area that will be parsed and saved
description_condition: "Prácticamente nuevo",
slug: "last-test-post",
latitude: 41.3811388,
longitude: 2.1770898,
price: 99999,
use_condition: "Prácticamente nuevo",
category_id: [438, 440]
}
```

Body Cookies Headers (11) Test Results

Pretty Raw Preview Visualize JSON


```
"price": 99999.0,
"published": "2024-01-17T10:30:10",
"description_condition": "Prácticamente nuevo",
"has_price_insight": false,
"latitude": 41.3811388,
"longitude": 2.1770898,
"post_id": "430d4b39-fdd6-4b70-8ee2-fa0ce0fd1eb1",
"user": {
    "user_id": "921f2bd6-d04a-4ffc-af85-03b90963574b",
    "first_name": "Pedro"
},
"categories": [
    {
        "category_id": "438",
        "name": "Buses"
    },
    {
        "category_id": "440",
        "name": "Trucks"
    }
```

Figura 27: Ejecución adminUpdatePost.

3.2.25. adminDeletePost()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	M.K.3
Tipo	Mutation
Llamada	adminDeletePost()

Recibe	AdminDeletePostInput - Obligatorio {
	post_id: ID!
	}

Devuelve	String - Obligatorio
Autorización	Grupo administrador

Tabla 36: Resumen del método adminDeletePost.

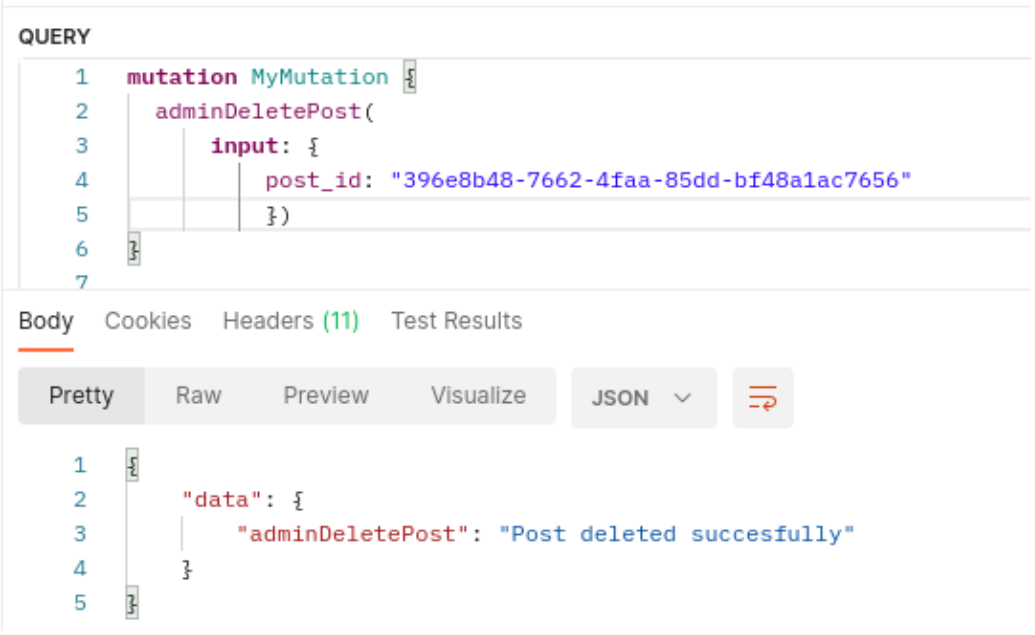

Figura 28: Ejecución adminDeletePost.

3.2.26. adminApprovePost()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	M.K.4
Tipo	Mutation

Llamada	adminApprovePost()

Recibe	AdminDeletePostInput - Obligatorio {
	post_id: ID!
	}

Devuelve	String - Obligatorio
Autorización	Grupo administrador

Tabla 37: Resumen del método adminApprovePost.

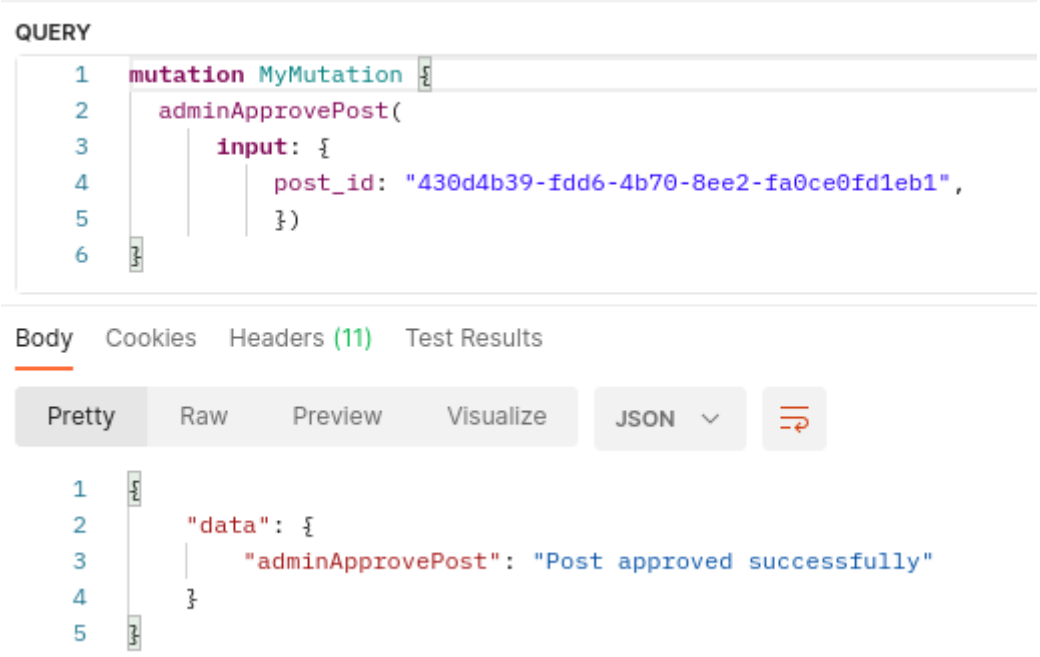

Figura 29: Ejecución adminApprovePost.

3.2.27. adminGetPostById()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	Q.S.4
Tipo	Query
Llamada	adminGetPostById()
Recibe	post_id - Obligatorio
Devuelve	PostDataOutput - Obligatorio { post_id: ID! title: String! content: String! published: String! slug: String!

	use_condition: String! description_condition: String! latitude: Float! longitude: Float! status: String! stock: Int! price: Float! has_price_insight: Boolean! user: UserDataOutput professional: Boolean! categories: [category] }
Autorización	Grupo administrador

Tabla 38: Resumen del método adminGetPostById.

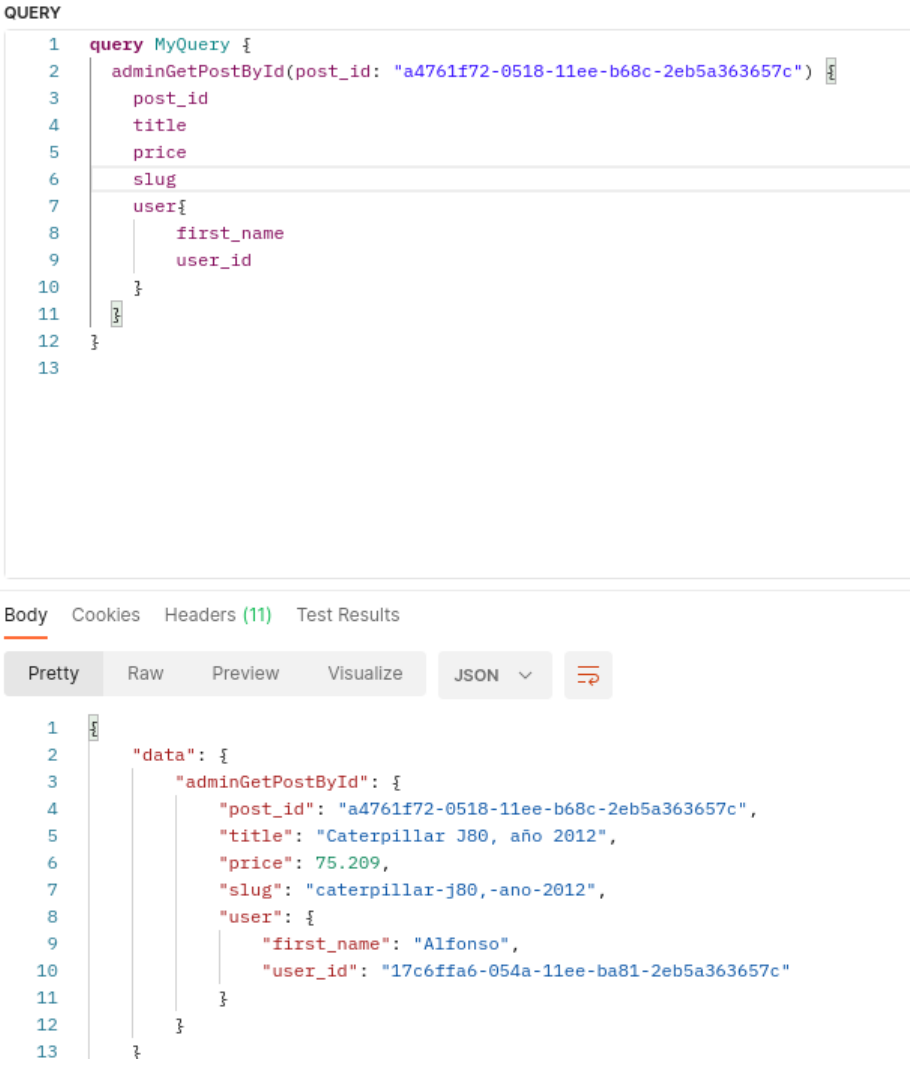

Figura 30: Ejecución adminGetPostById.

3.3. Conjunto de Queries & Mutations de producto

A continuación, se lista y desarrolla la totalidad de funcionalidades públicas definidas en la API. Al contrario de las descritas en el apartado inmediatamente anterior, estas están pensadas para ser ejecutadas directamente desde cualquiera de los clientes de Profesiolan, ya sea web o app.

En este apartado, se debe prestar especial atención a la granularidad de la política de accesos para cada método. De esta forma, se dejan atrás métodos definidos para su ejecución por parte de usuarios con rol de administrador, para dejar paso a nuevos grupos de usuarios.

La política de usuarios, al completo, está definida en el *Apartado 6: Política de accesos y gestión de usuarios.*

3.3.1. updateUser()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	M.U.11
Tipo	Mutation
Llamada	updateUser()

Recibe	UpdateUserDataInput - Obligatorio {
	user_id: ID!
	first_name: String!
	last_name: String!
	latitude: Float
	longitude: Float
	}

Devuelve	UpdateUserDataOutput - Obligatorio {
	user_id: ID!
	first_name: String!
	last_name: String!
	latitude: Float
	longitude: Float
	}

Autorización	Usuario autenticado

Tabla 39: Resumen del método updateUser.

Mutación perteneciente al microservicio de usuario. Permite la actualización de la información básica de un usuario registrado y autenticado.

Figura 31: Ejecución updateUser.

3.3.2. publishPost()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	M.K.5
Tipo	Mutation
Llamada	publishPost()

Recibe	PublishPostInput - Obligatorio {
	user_id: ID!
	title: String!
	content: String!
	use_condition: String!
	description_condition: String!
	latitude: Float!
	longitude: Float!
	price: Float!
	category_id: [Int!]!
	}

Devuelve	PublishPostOutput - Obligatorio { post_id: ID! title: String! content: String! published: String! slug: String! use_condition: String! description_condition: String! latitude: Float! longitude: Float! status: String! stock: Int! price: Float!

	has_price_insight: Boolean! user_id: ID! professional: Boolean! postCategory: [category]! }
Autorización	Usuario autenticado

Tabla 40: Resumen del método PublishPost.

Mutación para la publicación de un activo asignado a una serie de categorías, con toda la información necesaria. El método genera el activo, lo asocia al usuario, a las categorías, se le añade toda la metadata, y se guarda en la base de datos, con un estado "draft", a falta de pasar por el proceso de fotoverificación y peritación que actualicen, en un futuro, ese estado a "published".

QUERY

```
mutation MyMutation {
  publishPost(
    input: {
      title: "Caterpillar F80, año 2022",
```

Body Cookies Headers (11) Test Results

Pretty Raw Preview Visualize JSON


```
{
    "data": {
        "publishPost": {
            "title": "Caterpillar F80, año 2022",
            "content": "Dummy content text area that will be parsed and saved in a centralized manner",
            "price": 99999.0,
            "published": "2023-07-07T13:07:29",
            "user_id": "17c6fccc-054a-11ee-affb-2eb5a363657c",
            "description_condition": "Prácticamente nuevo",
            "has_price_insight": false,
            "latitude": 41.3811388,
            "longitude": 2.1770898,
            "post_id": "11ca963e-6984-4cd2-8b0c-83a8cc20dffb",
            "postCategory": [
                {
                    "category_id": "437",
                    "name": "Industrial Vehicles",
                    "count": 0,
                    "meta_desc": "Industrial Vehicles"
                },
                {
                    "category_id": "439",
                    "name": "Tractor units",
                    "count": 0,
                    "meta_desc": "Tractor units"
                }
            ]
        }
    }
}
```

Figura 32: Ejecución publishPost.

Este módulo sí implementa una mayor granularidad en las excepciones. Por ejemplo, para una categoría inexistente, la solicitud devuelta es la siguiente:

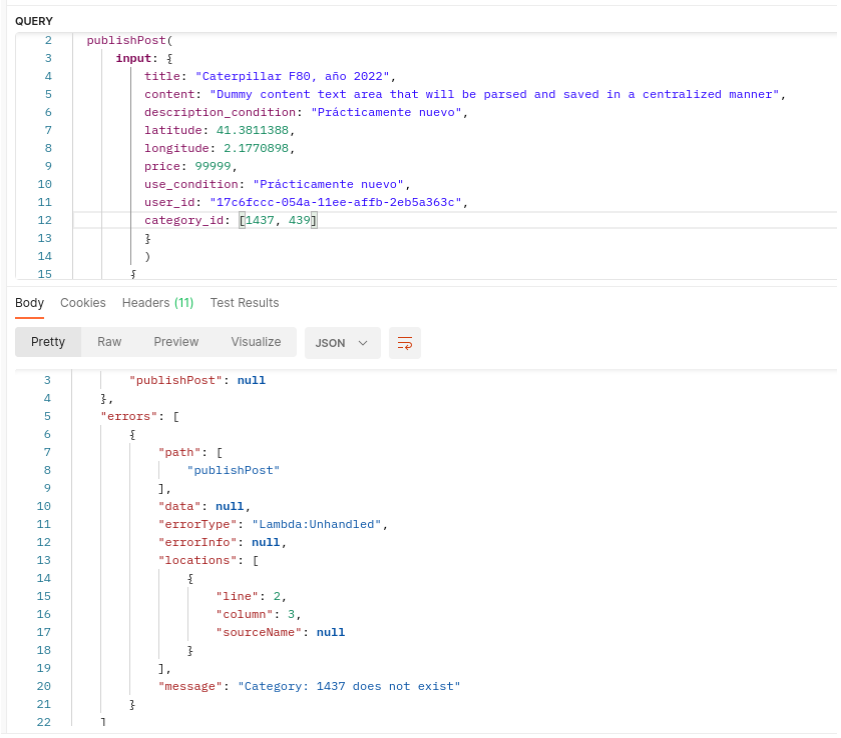

Figura 33: Ejecución publishPost con respuesta ante error I.

También devuelve una excepción notificando cuando algún valor requerido no ha sido introducido.

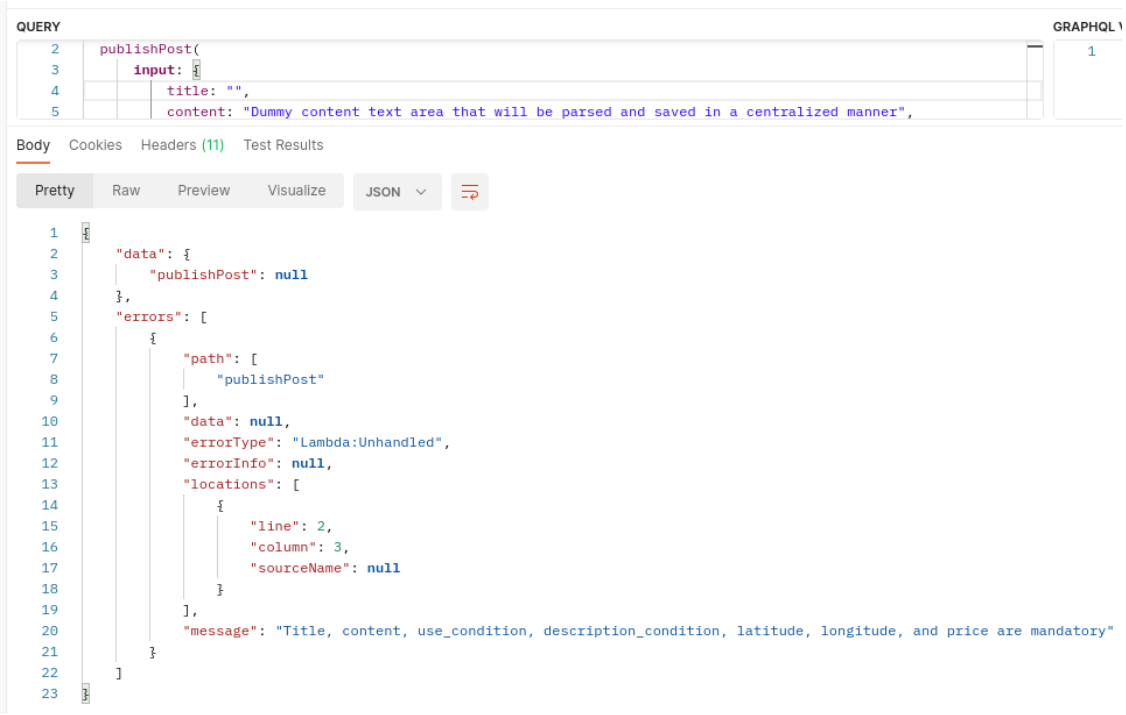

Figura 34: Ejecución publishPost con respuesta ante error II.

También se ha testeado el caso de uso en el que el usuario autenticado no es el propietario del activo a gestionar, que devuelve un código 401: Unauthorized.

QUERY

```
mutation MyMutation {
  publishPost(
    input: {
      title: "Published post by authenticated non-admin user",
      content: "Lorem Ipsum",
      description_condition: "Prácticamente nuevo",
      latitude: 41.3811388,
      longitude: 2.1770898,
      price: 99999,
      use_condition: "Prácticamente nuevo",
      user_id: "921f2bd6-d04a-4ffc-af85-03b90163574b",
      category_id: [240, 241]
      }
```

Body Cookies Headers (11) Test Results

Pretty Raw Preview Visualize JSON


```
{
  "data": {
    "publishPost": null
  },
  "errors": [
    {
      "path": [
        "publishPost"
      ],
      "data": null,
      "errorType": "Lambda:Unhandled",
      "errorInfo": null,
      "locations": [
        {
          "line": 2,
          "column": 3,
          "sourceName": null
        }
      ],
      "message": "401: Unauthorized"
```


Figura 35: Ejecución publishPost con respuesta ante error III.

En la práctica totalidad de las veces, estas excepciones no son capturadas, ya que es el propio cliente el que realiza las comprobaciones previo envío de las consultas a la API. No obstante, en las APIs públicas, es una práctica más que obligatoria mapear e inspeccionar correctamente todos los argumentos de entrada.

3.3.3. UpdatePost()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	M.K.6
Tipo	Mutation
Llamada	updatePost()
Recibe	UpdatePostInput - Obligatorio { post_id: ID!

	user_id: ID! title: String! content: String! use_condition: String! description_condition: String! latitude: Float! longitude: Float! price: Float! category_id: [Int!]! }

Devuelve	UpdatePostOutput - Obligatorio {
	post_id: ID!
	title: String!
	content: String!
	published: String!
	slug: String!
	use_condition: String!
	description_condition: String!
	latitude: Float!
	longitude: Float!
	status: String!
	stock: Int!
	price: Float!
	has_price_insight: Boolean!
	user_id: ID!
	professional: Boolean!
	postCategory: [category]!
	}

Autorización	Usuario autenticado

Tabla 41: Resumen del método UpdatePost.

QUERY

```
mutation MyMutation {
  updatePost(
    input: {
      post_id: "920a89bd-dc03-47da-a94b-a1fbf78933fc",
      title: "Updated post by authenticated non-admin user",
      content: "Lorem Ipsum",
      description_condition: "Con vida útil",
      latitude: 41.3811388,
      longitude: 2.1770898,
      price: 5000,
      use_condition: "Con vida útil",
      user_id: "921f2bd6-d04a-4ffc-af85-03b90963574b",
      category_id: [437, 439]
    }
```

Body Cookies Headers (11) Test Results

Pretty Raw Preview Visualize JSON


```
    "updatePost": {
      "title": "Updated post by authenticated non-admin user",
      "content": "Lorem Ipsum",
      "status": "draft",
      "price": 5000.0,
      "published": "2024-01-17T11:53:47",
      "description_condition": "Con vida útil",
      "has_price_insight": false,
      "latitude": 41.3811388,
      "longitude": 2.1770898,
      "post_id": "920a89bd-dc03-47da-a94b-a1fbf78933fc",
      "user": {
        "user_id": "921f2bd6-d04a-4ffc-af85-03b90963574b",
        "first_name": "Pedro"
      },
      "categories": [
        {
          "category_id": "437",
          "name": "Industrial Vehicles"
```


Figura 36: Ejecución updatePost.

3.3.4. deletePost()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	M.K.7
Tipo	Mutation
Llamada	deletePost()
Recibe	DeletePostInput - Obligatorio
Devuelve	String - Obligatorio
Autorización	Usuario autenticado

Tabla 42: Resumen del método deletePost.

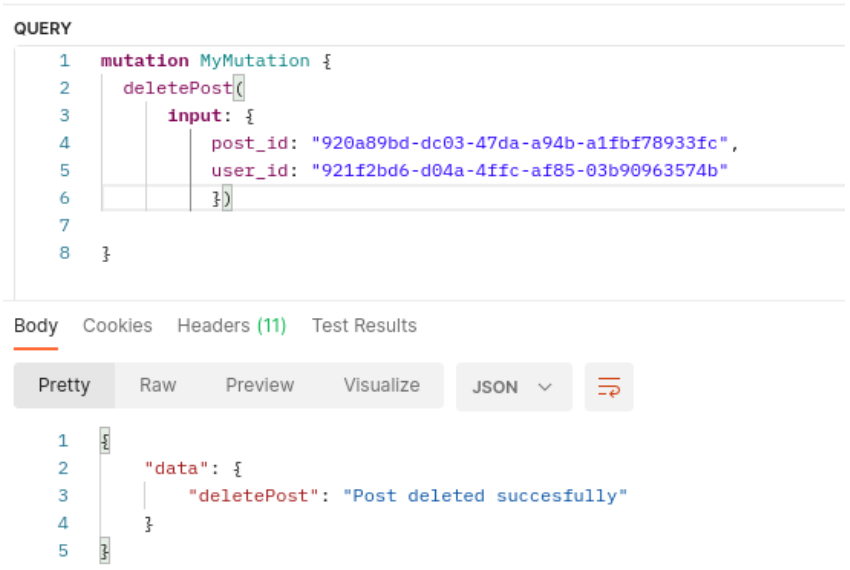

Figura 37: Ejecución deletePost.

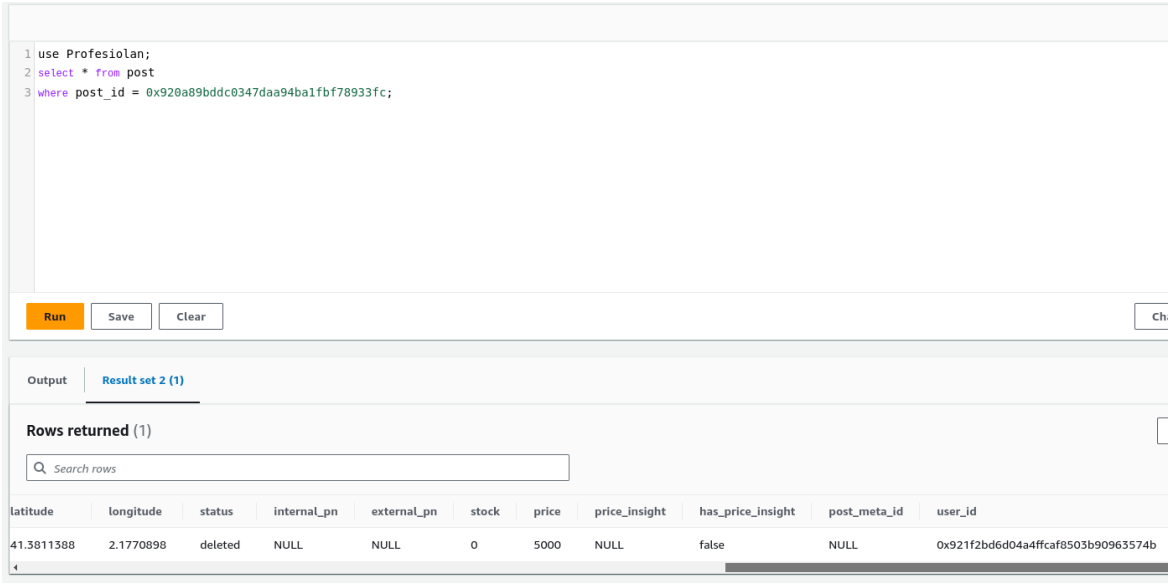

Figura 38: Comprobación previa a ejecución deletePost.

Por supuesto, al igual que todas las operaciones permitidas por la API desarrollada, se mantiene la coherencia en el sentido de que únicamente el usuario concreto dueño del activo puede borrarlo.

3.3.5. getMyPosts()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	Q.S.5
Tipo	Query
Llamada	getMyPosts()
Recibe	user_id - Obligatorio

Devuelve	Array of ReducedPost - Obligatorio {
	post_id: ID!
	title: String!
	slug: String!
	status: String!
	price: Float!
	professional: Boolean!
	}

Autorización	Usuario autenticado

Tabla 43: Resumen del método getMyPosts.

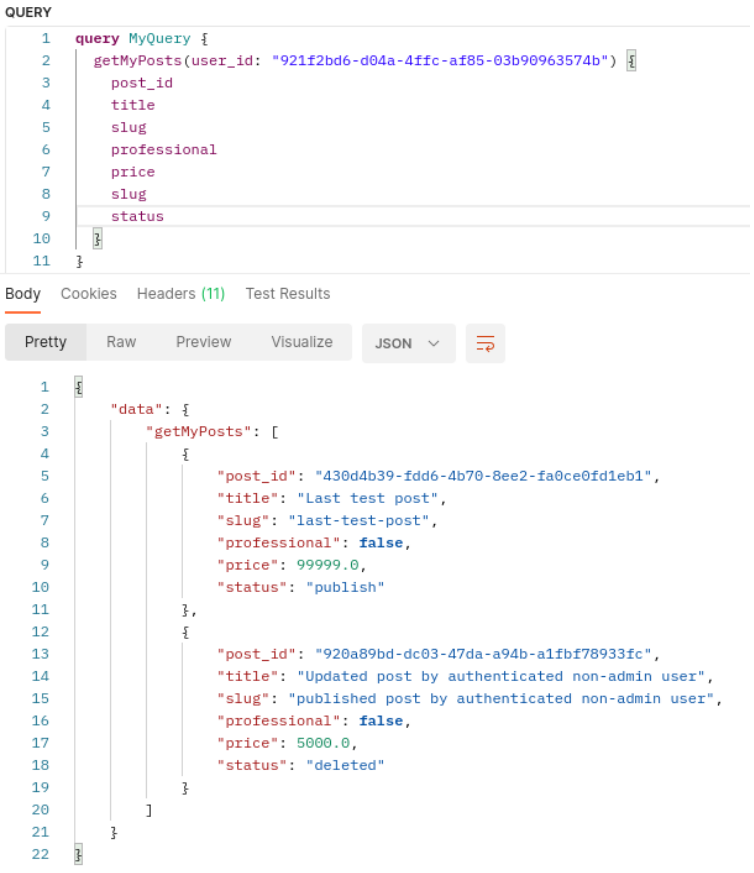

Figura 39: Ejecución getMyPosts.

3.3.6. getBrandsByCategoryId()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	Q.S.1
Tipo	Query
Llamada	getBrandsByCategoryId()
Recibe	category_id - Obligatorio
Devuelve	[brand]
Autorización	Public

Tabla 44: Resumen del método getBrandsByCategoryId.

Método contemplado en el módulo o servicio Search, la solicitud resuelve la búsqueda de marcas para una categoría.

En el proceso de filtrado, es necesario mostrar la lista de marcas aplicantes a los activos referentes a la categoría de búsqueda seleccionada. Por ejemplo, para una búsqueda de Vehículos Industriales > Camiones Rígidos > Camiones Basculantes, Profesiolan muestra una lista de los principales proveedores o marcas para los que filtrar activos.

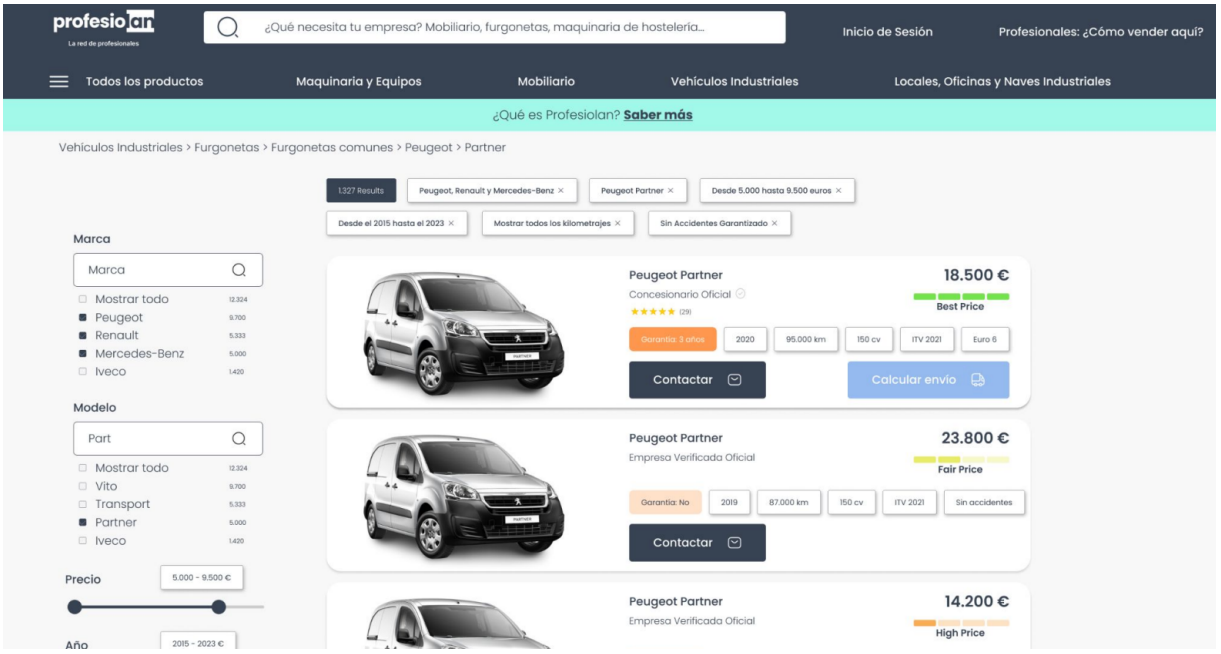

Figura 40: Ejemplo UI/UX del producto para getBrandsByCategoryId.

En este caso, una solicitud que se puede realizar es la siguiente:

```
query MyQuery {
 getBrandsByCategoryId(category_id: 437){
   name
   slug
   meta_desc
 }
}
```

Tabla 45: Ejemplo ejecución getBrandsByCategoryId.

Se comprueba que la categoría 437 es la referente a Vehículos Industriales.

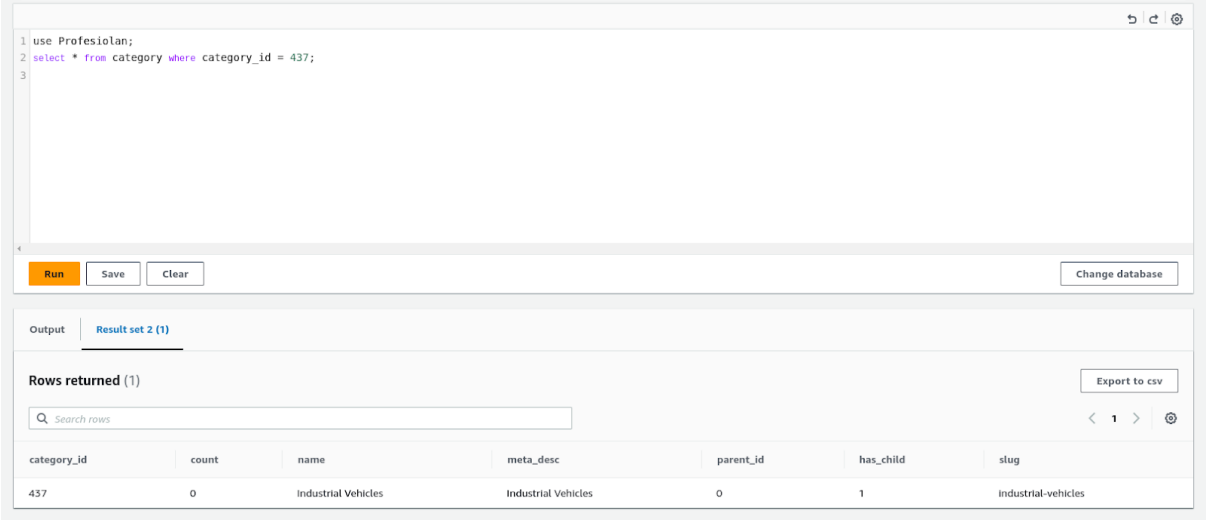

Figura 41: Comprobación previa a ejecución getBrandsByCategoryId.

Como es de esperar, la respuesta tiene la siguiente forma.


```
{
    "data": {
        "getBrandsByCategoryId": [
            {
                "name": "Mercedes-Benz",
                "slug": "mercedes-benz",
                "meta_desc": "Mercedes Benz"
            },
            {
                "name": "Man",
                "slug": "man",
                "meta_desc": "Man"
            }
        ]
    }
}
```

Figura 42: Ejecución getBrandsByCategoryId.

Por supuesto, para los casos en el que directamente no se ha incluído el parámetro obligatorio category_id, la respuesta es una excepción de tipo MalformedHttpRequestException, con el siguiente mensaje de error "Unable to parse graphQL query".

3.3.7. getModelsByBrandAndCategory()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	Q.S.10
Tipo	Query
Llamada	getModelsByBrandAndCategory()
Recibe	category_id - Obligatorio, brand_id - Obligatorio
Devuelve	[model]
Autorización	Public

Tabla 46: Resumen del método getModelsByBrandAndCategory.

Contemplado en el módulo o servicio Search, la solicitud resuelve la búsqueda de Modelos para una Marca y que aplica en una Categoría.

Profesiolan cuenta con miles de vehículos industriales y maquinaria. En este contexto, muchos fabricantes (Brand) tienen productos en diferentes categorías (Vehículos Industriales > Cabezas tractoras, Vehículos Industriales > Camiones Rígidos > Camiones Basculantes, Maquinaria y Equipos > Maquinaria de Construcción y Obra Pública > Excavadoras…).

Esta solicitud se ejecuta a diario en prácticamente todas las sesiones, ya que es un filtro esencial en el proceso de búsqueda.

QUERY

```
query MyQuery {
  getModelsByBrandAndCategory(brand_id: 1, category_id: 437) {
    model_id
    count
```

Body Cookies Headers (11) Test Results

Pretty Raw Preview Visualize JSON


```
{
    "data": {
        "getModelsByBrandAndCategory": [
            {
                "model_id": "1",
                "count": 0,
                "name": "Actros",
                "brand": {
                    "name": "Mercedes-Benz",
                    "brand_id": "1"
                }
            },
            {
                "model_id": "2",
                "count": 0,
                "name": "Atego",
                "brand": {
                    "name": "Mercedes-Benz",
                    "brand_id": "1"
                }
            },
            {
                "model_id": "3",
                "count": 0,
                "name": "Sprinter",
                "brand": {
                    "name": "Mercedes-Benz",
                    "brand_id": "1"
```


Figura 43: Ejecución getModelsByBrandAndCategory.

3.3.8. getCategoriesByPostId()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	Q.S.6
Tipo	Query
Llamada	getCategoriesByPostId()
Recibe	post_id - Obligatorio
Devuelve	Array of category - Obligatorio { category_id: ID count: Int! name: String! meta_desc: String!

	parent: category has_child: Boolean slug: String! brands: [brand] models: [model] }
Autorización	No

Tabla 47: Resumen del método getCategoriesByPostId.

Método de cliente para la obtención del arbol de categorías para las migas de pan en la representación de los productos. Se trata de un método púramente de navegación dinámica.

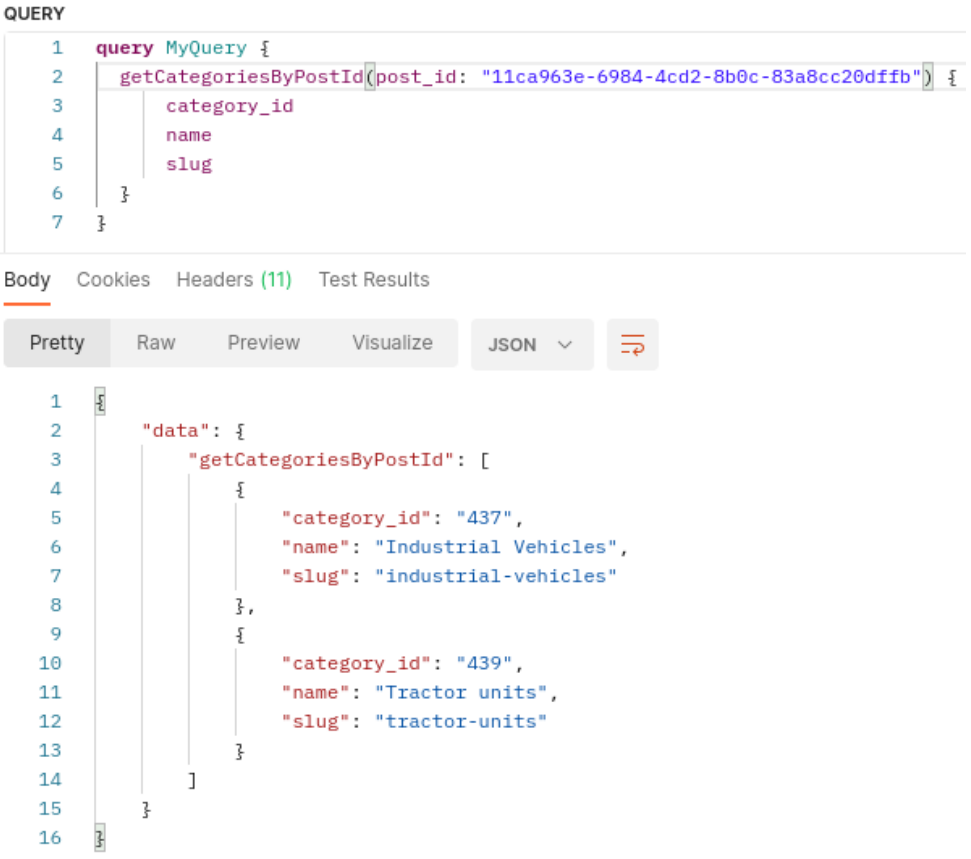

Figura 44: Ejecución getCategoriesByPostId.

3.3.9. getModelById()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	Q.S.7

Tipo	Query
Llamada	getModelById()
Recibe	model_id - Obligatorio

Devuelve	model - Obligatorio {
	model_id: ID!
	name: String!
	count: Int!
	slug: String!
	categories: [ReducedCategory]
	brand: brand
	}

Autorización	No

Tabla 48: Resumen del método getCategoriesByPostId.

Método de cliente para la obtención de la información del modelo para más opciones de navegación durante la representación de los productos. Se trata de un método púramente de navegación dinámica.

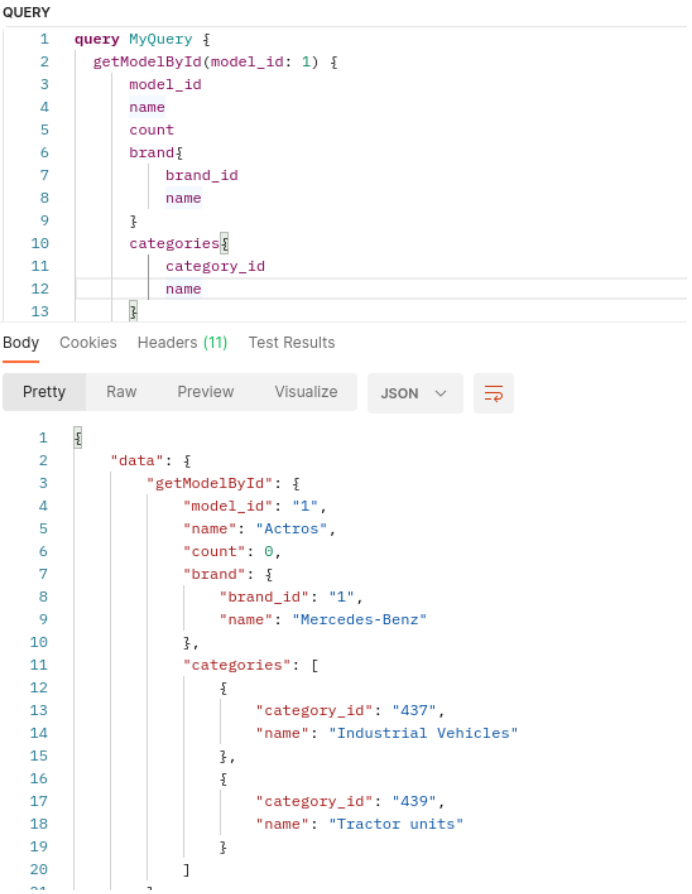

Figura 45: Ejecución getModelById.

3.3.10. getCategoryById()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	Q.S.8
Tipo	Query
Llamada	getCategoryById()
Recibe	category_id - Obligatorio

Devuelve	category - Obligatorio {
	category_id: ID
	count: Int!
	name: String!
	meta_desc: String!
	parent: category
	has_child: Boolean
	slug: String!
	brands: [brand]
	models: [model]
	}

Autorización	No

Tabla 49: Resumen del método getCategoryById.

Método de cliente para la obtención de la información de la categoría para más opciones de navegación durante la representación de los productos. Se trata de un método púramente de navegación dinámica.

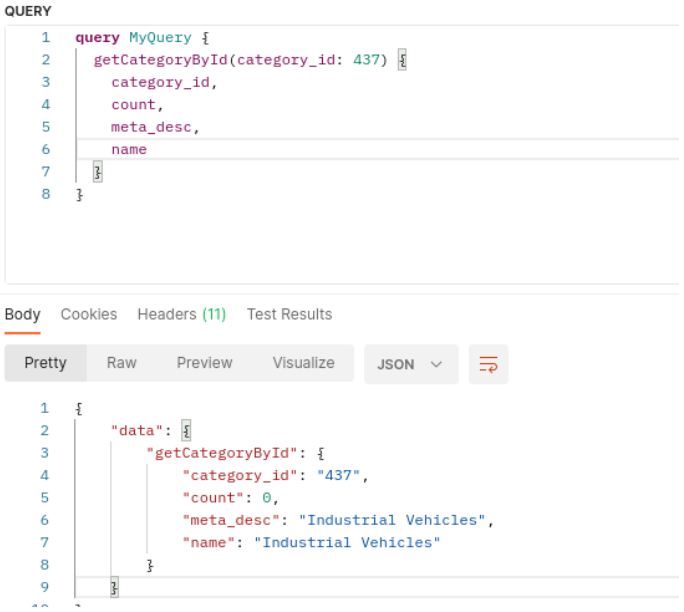

Figura 46: Ejecución getCategoryById.

3.3.11. getBrandById()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	Q.S.9
Tipo	Query
Llamada	getBrandById()
Recibe	brand_id - Obligatorio

Devuelve	brand - Obligatorio {
	brand_id: ID
	name: String
	count: Int
	slug: String
	meta_desc: String
	}

Autorización	No

Tabla 50: Resumen del método getBrandById.

Método de cliente para la obtención de la información de la marca para más opciones de navegación durante la representación de los productos. Se trata de un método púramente de navegación dinámica.

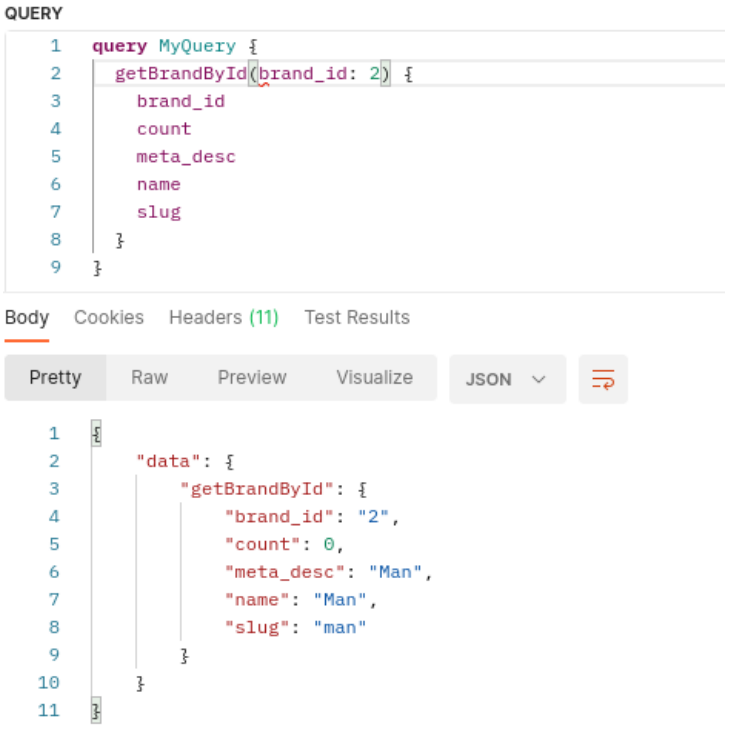

Figura 47: Ejecución getBrandById.

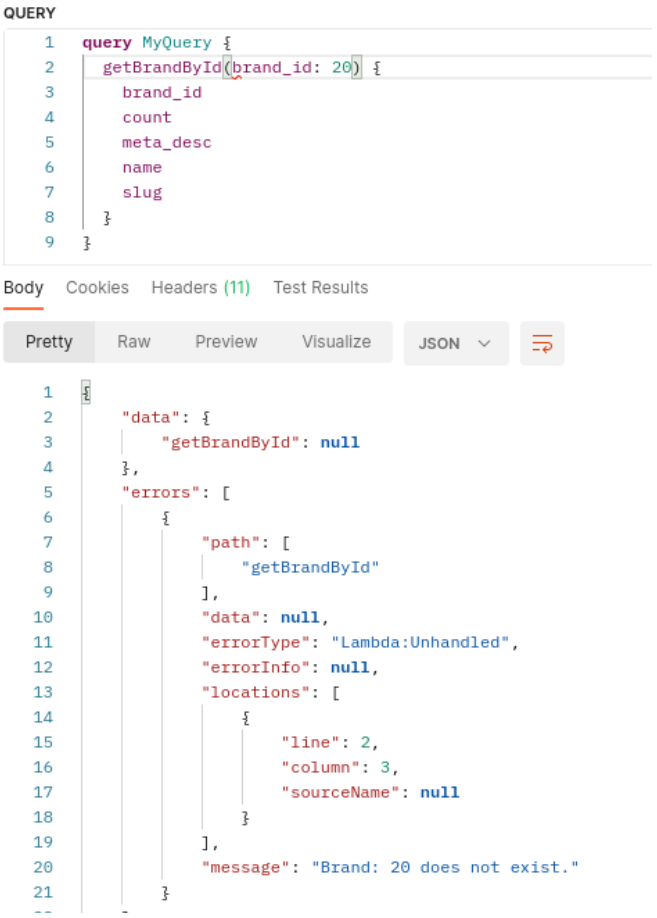

Figura 48: Ejecución getBrandById no correcta.

3.3.12. getPostById()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	Q.S.2
Tipo	Query
Llamada	getPostById()
Recibe	post_id - Obligatorio
Devuelve	PublicPost - Obligatorio { post_id: ID! title: String! content: String! published: String! slug: String! use_condition: String! description_condition: String! latitude: Float! longitude: Float!

	stock: Int! price: Float! has_price_insight: Boolean! user_id: ID! professional: Boolean! }
Autorización	No

Tabla 51: Resumen del método getPostById.

Operación fundamental en la infraestructura, perteneciente al módulo Search. Devuelve toda la información necesaria para la generación de la página de compra de un activo.

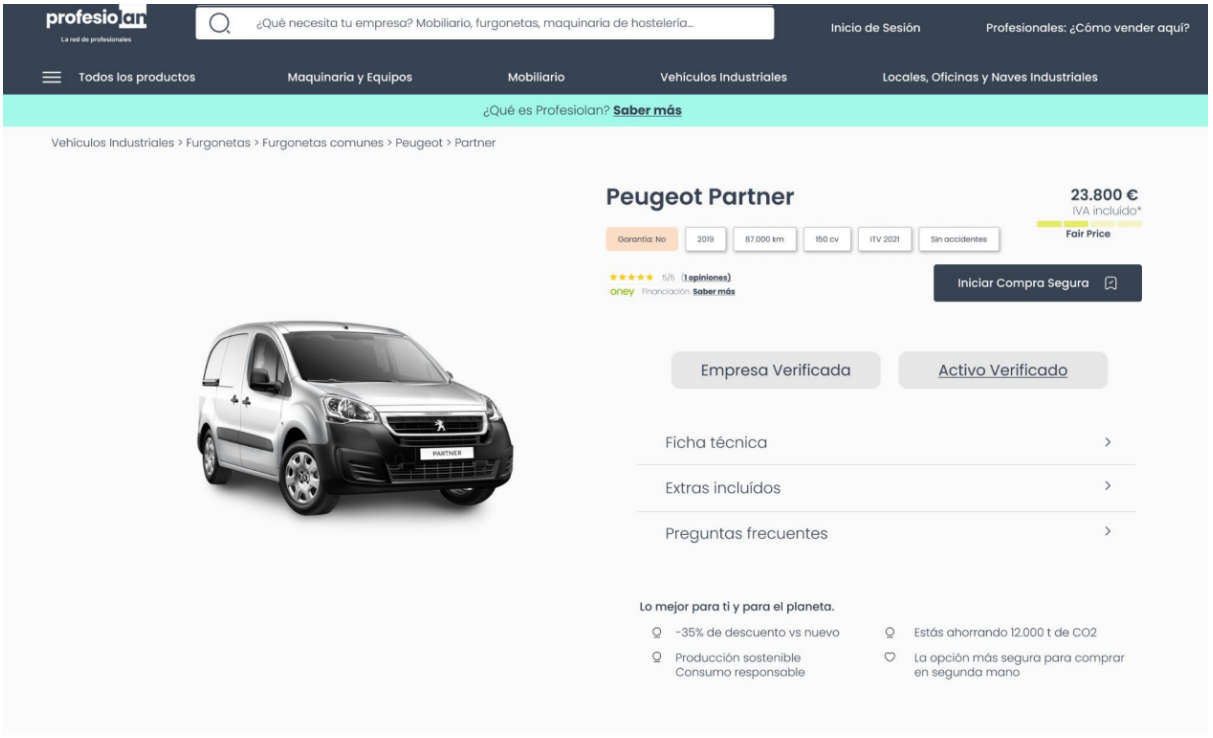

Figura 49: Ejemplo UI/UX del producto para getPostById.

A continuación, se presenta una query con un post_id existente y referente a un activo dummy insertado. La base de datos dummy cuenta con 90 activos de diferentes categorías.

```
{
    "data": {
        "getPostById": {
            "post_id": "a4761db0-0518-11ee-838f-2eb5a363657c",
            "published": null,
            "modified": null,
            "specific_condition": "2531 h",
            "use_condition": "Buen estado general",
            "description_condition": "Buen estado general",
            "status": "draft",
            "stock": 0,
            "price": 83.455,
            "price_insight": null,
            "has_price_insight": true,
            "postCategory": [
                {
                    "category_id": "13",
                    "name": "Maquinaria y equipos",
                    "slug": "maquinaria-y-equipos"
                },
                {
                    "category_id": "56",
                    "name": "Construcción y obra pública",
                    "slug": "construccion-obra-publica-y-mineria"
                },
                {
                    "category_id": "66",
                    "name": "Excavadoras",
                    "slug": "excavadoras"
                }
            ]
        }
    }
}
```


Figura 50: Ejecución getPostById.

En este caso, preguntar por la categoría es especialmente útil para la funcionalidad de navegación mediante migas de pan. Por ello, se devuelve el slug, el nombre, y el número de la categoría.

3.3.13. getPostsByFilters()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	Q.S.3
Tipo	Query

Llamada	getPostsByFilters()
Recibe	pageNumber - Obligatorio, pageSize - Obligatorio, sortDirection - Obligatorio, sortBy - Obligatorio, category_id - Obligatorio, professional - Obligatorio

Devuelve	PublicPost - Obligatorio {
	post_id: ID!
	title: String!
	content: String!
	published: String!
	slug: String!
	use_condition: String!
	description_condition: String!
	latitude: Float!
	longitude: Float!
	stock: Int!
	price: Float!
	has_price_insight: Boolean!
	user_id: ID!
	professional: Boolean!
	}

Autorización	No

Tabla 52: Resumen del método getPostById.

Una vez aplicados los filtros, esta solicitud resuelve la búsqueda de activos de forma unificada.

Además del filtrado por categoría y por tipo de empresa, también soporta paginación y ordenación (por ahora en orden ascendente o descendente pero solo en función de las fechas de publicación).

En este caso, se ha aplicado un concepto de best-practice consistente en la generación de un objeto de datos tipo Output, orientado al caso de uso, para la optimización del tiempo. La realidad es que, al devolver muchos resultados y paginados, no tendría sentido pedir más datos que los estrictamente necesarios para mostrar en las tarjetas del front-end.

```
{
    "data": {
        "getPostByFilters": [
            {
                "post_id": "a4761f72-0518-11ee-b68c-2eb5a363657c",
                "title": "Caterpillar J80, año 2012",
                "price": 75.209,
                "slug": "caterpillar-j80,-ano-2012",
                "has_price_insight": true,
                "price_insight": "Buen Precio",
                "professional": false,
                "postCategory": [
                    {
                        "name": "Maquinaria y equipos",
                        "category_id": "13",
                        "slug": "maquinaria-y-equipos"
                    },
                    {
                        "name": "Construcción y obra pública",
                        "category_id": "56",
                        "slug": "construccion-obra-publica-y-mineria"
                    },
                    {
                        "name": "Excavadoras",
                        "category_id": "66",
                        "slug": "excavadoras"
                    }
                ]
            },
            {
                "post_id": "a47624e0-0518-11ee-8743-2eb5a363657c",
                "title": "JCB 220 W Hydralig, año 2021",
                "price": 23.591,
                "slug": "jcb-220-w-hydralig,-ano-2021",
                "has_price_insight": true,
                "price_insight": "Buen Precio",
                "professional": false,
                "postCategory": [
```


Figura 51: Ejecución getPostsByFilters.

En este momento, tanto la paginación como la ordenación son funcionalidades totalmente operativas, además del propio filtro de categoría y tipo de empresa.

3.3.14. hasUserMeta()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	Q.U.1
Tipo	Query
Llamada	hasUserMeta()
Recibe	user_id - Obligatorio

Devuelve	Boolean
Autorización	No

Tabla 53: Resumen del método hasUserMeta.

Estos métodos son rápidas consultas a realizar por el cliente para el rendedirzado dinámico del contenido de cada cliente en su escritorio de usuario.

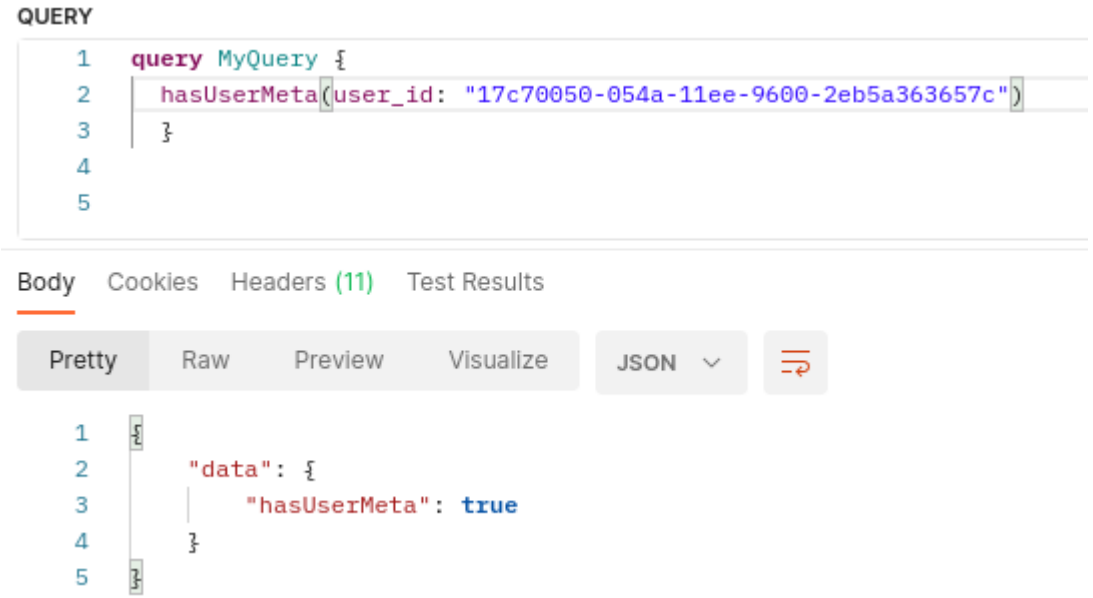

Figura 52: Ejecución hasUserMeta.

Además, se comprueba a continuación cómo el mencionado usuario tiene realmente asociado un objeto de metainformación.

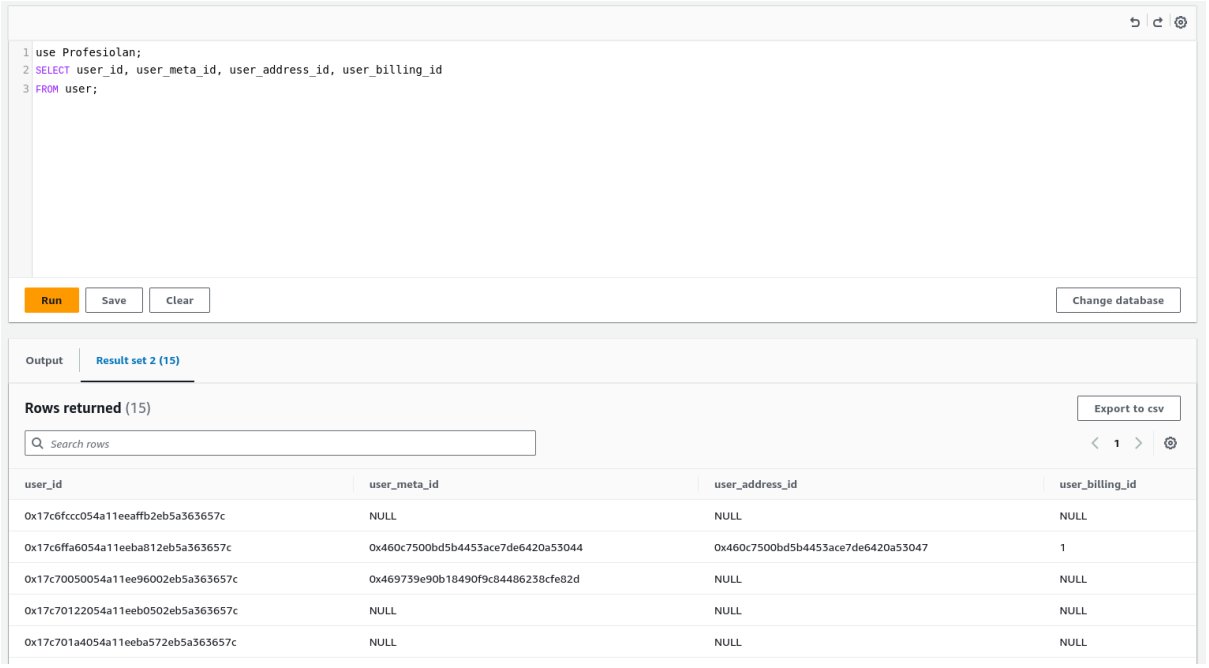

Figura 53: Comprobación posterior a ejecución hasUserMeta.

3.3.15. hasUserAddress()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	Q.U.2
Tipo	Query
Llamada	hasUserAddress()
Recibe	user_id - Obligatorio
Devuelve	Boolean
Autorización	No

Tabla 54: Resumen del método hasUserAddress.

Método idéntico para la comprobación del objeto de direcciones y opciones de contacto para un usuario.

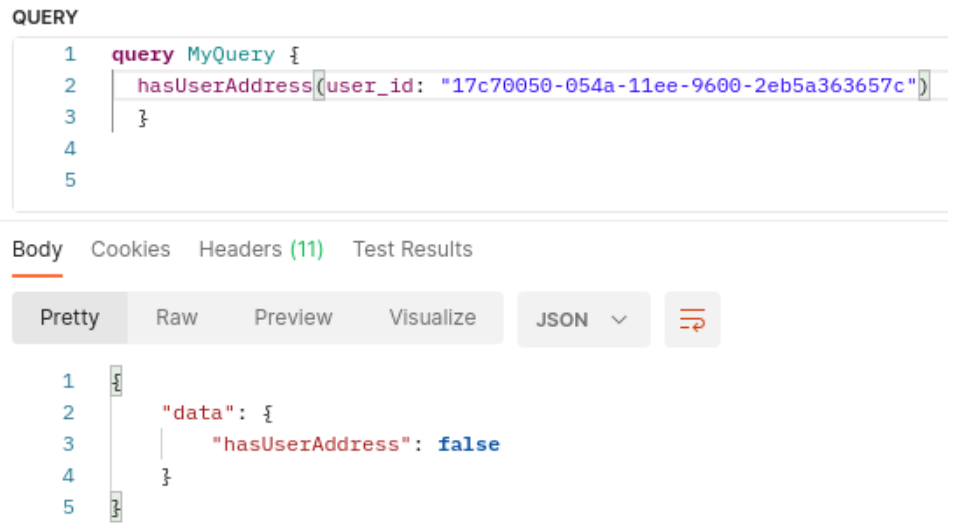

Figura 54: Ejecución hasUserAddress para usuario sin objeto.

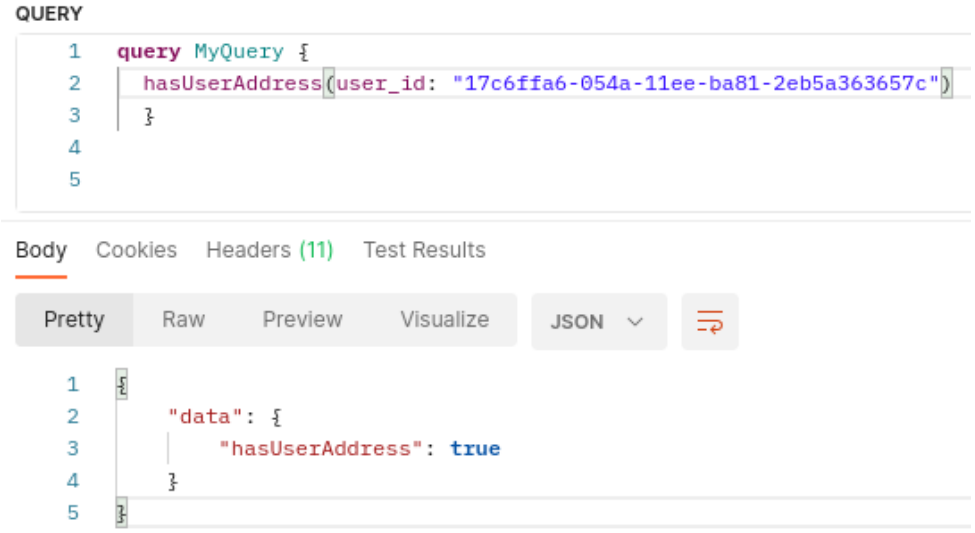

Figura 55: Ejecución hasUserAddress para usuario con objeto.

3.3.16. hasUserBilling()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	Q.U.3
Tipo	Query
Llamada	hasUserBilling()
Recibe	user_id - Obligatorio
Devuelve	Boolean
Autorización	No

Tabla 55: Resumen del método hasUserBilling.

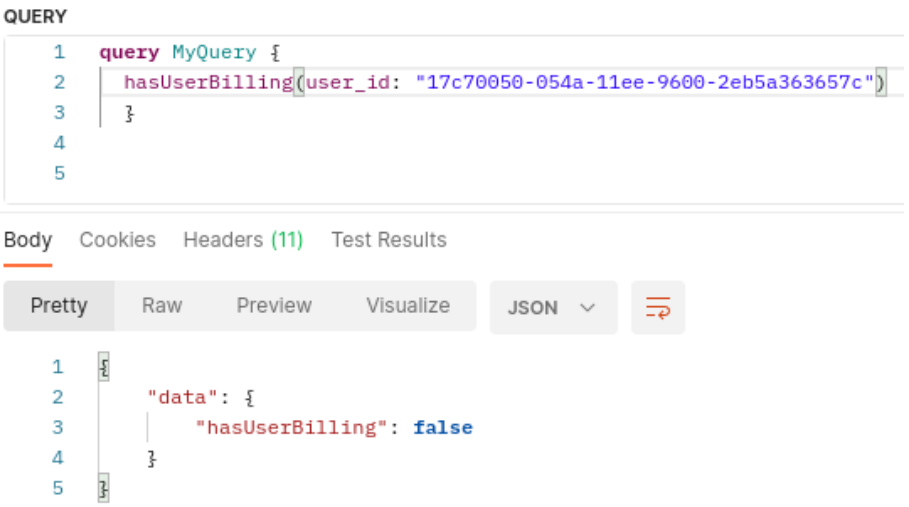

Figura 56: Ejecución hasUserBilling.

3.3.17. getUserMetaByUserId()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	Q.U.4
Tipo	Query
Llamada	getUserMetaByUserId()
Recibe	user_id - Obligatorio

Devuelve	user_meta - Obligatorio {
	user_meta_id: ID!
	website: String
	linkedin: String
	facebook: String
	twitter: String
	biography: String!
	supplier: Boolean
	producer: Boolean
	ocasional: Boolean
	realestate: Boolean
	rating: Float!
	total_opinions: Int!
	
	}

Autorización	Usuario autenticado y propietario de los datos

Tabla 56: Resumen del método getUserMetaByUserId.

QUERY

```
query MyQuery {
  getUserMetaByUserId(user_id: "17c6ffa6-054a-11ee-ba81-2eb5a363657c") {
    user_meta_id
    website
    biography
    supplier
    producer
    ocasional
    realestate
    rating
    total_opinions
  }
}
```

Body Cookies Headers (11) Test Results

Pretty Raw Preview Visualize JSON


```
{
    "data": {
        "getUserMetaByUserId": {
            "user_meta_id": "460c7500-bd5b-4453-ace7-de6420a53044",
            "website": "https://google.com",
            "biography": "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc ia
            "supplier": true,
            "producer": false,
            "ocasional": false,
            "realestate": false,
            "rating": 3.22,
            "total_opinions": 91
        }
    }
}
```


Figura 57: Ejecución getUserMetaByUserId.

3.3.18. getUserAddressByUserId()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	Q.U.5
Tipo	Query
Llamada	getUserAddressByUserId()
Recibe	user_id - Obligatorio
Devuelve	user_address - Obligatorio { user_address_id: ID! address_line: String! city: String! postal_code: String! district: String! country: String!

	phone: String! }
Autorización	Usuario autenticado y propietario de los datos

Tabla 57: Resumen del método getUserAddressByUserId.

QUERY

```
query MyQuery {
  getUserAddressByUserId(user_id: "17c6ffa6-054a-11ee-ba81-2eb5a363657c") {
    user_address_id
    address_line
    city
    postal_code
    district
    country
    phone
  }
}
```

Body Cookies Headers (11) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "data": {
        "getUserAddressByUserId": {
            "user_address_id": "460c7500-bd5b-4453-ace7-de6420a53047",
            "address_line": "Universidad del País Vasco, Ingeniero Torres Quevedo Plaza, 1",
            "city": "Bilbao",
            "postal_code": "48010",
            "district": "Bizkaia",
            "country": "Spain",
            "phone": "944411111"
        }
    }
}
```

Figura 58: Ejecución getUserAddressByUserId.

3.3.19. getUserBillingByUserId()

Conjunto de consultas y peticiones API disponibles	
Identificador del recurso	Q.U.6
Tipo	Query
Llamada	getUserBillingByUserId()
Recibe	user_id - Obligatorio
Devuelve	user_billing - Obligatorio { user_billing_id: ID!

	address: String! company_name: String! tax_id: String! }
Autorización	Usuario autenticado y propietario de los datos

Tabla 58: Resumen del método getUserBillingByUserId.

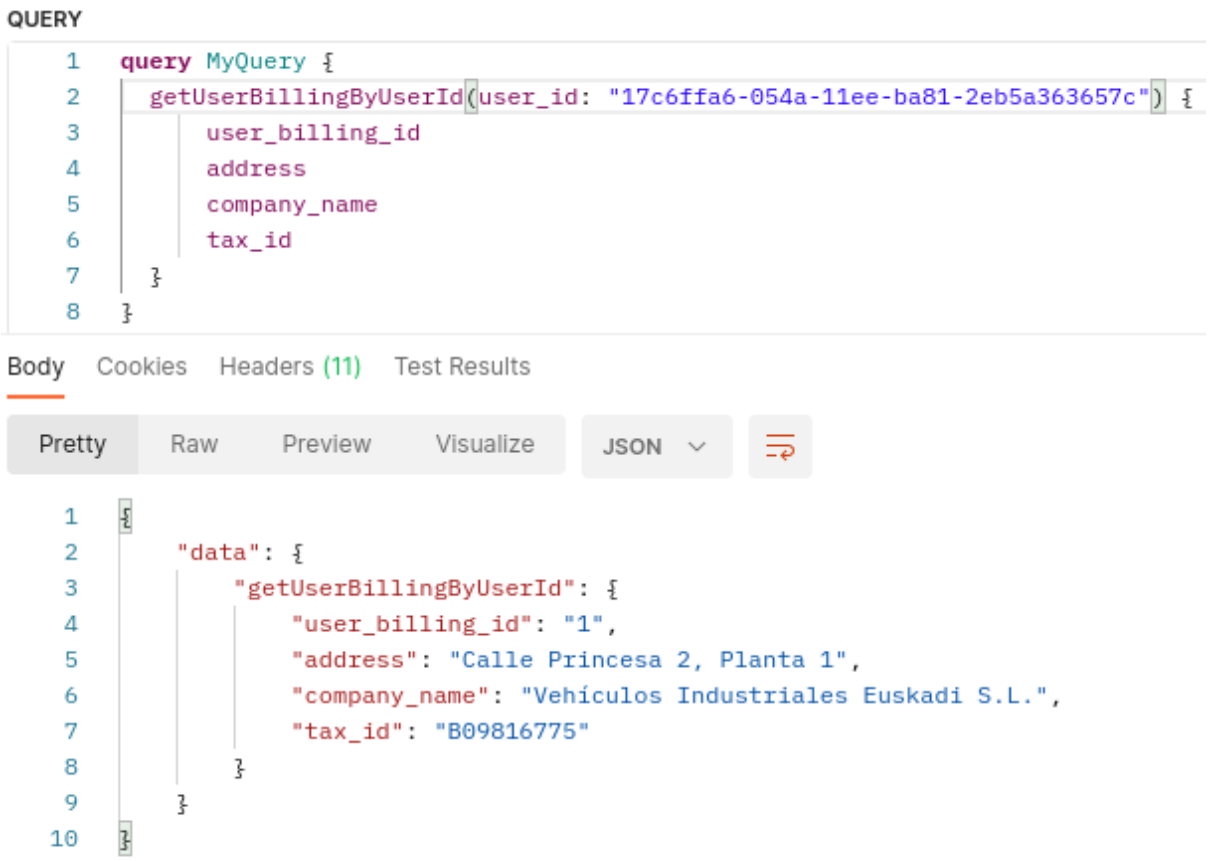

Figura 59: Ejecución getUserBillingByUserId.

3.4. Gestión de la Caché

Una correcta implementación de una memoria Caché a nivel API es indispensable en cualquier producto en producción. En concreto, la habilitación de la Caché, facilita que la información pueda ser obtenida en alta disponibilidad, optimizando mayúsculamente el rendimiento, y de igual forma reduciendo las latencias asociadas al tiempo de respuesta.

AWS AppSync ofrece una granularidad de Caché excelente por los siguientes motivos. En primer lugar, ofrece hasta 8 niveles diferentes de tamaño de Caché, batiéndose entre 1 vCPU, con 1.5 GiB de RAM, hasta la posibilidad de un ancho de banda de 10 GiB de rendimiento de red, con una RAM de 317.77 GiB distribuida en 48 CPUs virtuales.

Por supuesto, se debe hacer un coherente análisis de las especificaciones para poder seleccionar la Caché óptima para cada solución. Las conclusiones de este análisis, junto a los resultados, pueden ser análizados en el *Apartado 7: Análisis de rendimiento.*

En segundo lugar, AWS AppSync ofrece la posibilidad de activación de la Caché a dos niveles. La primera opción, llamada Full Request Caching, permite que todas las peticiones de cliente sean cacheadas, de forma que todas las llamadas a la API devuelven respuestas de la memoria caché. La segunda opción, lógicamente más liviana y por tanto más económica, permite la limitación de cachear únicamente unas determinadas operaciones o campos definidos en alguno de los resolvers. Como se puede observar, esta opción es ideal para ofrecer una granularidad de actuación cercana a la perfección para cada caso de uso.

En último lugar, AWS Appsync ofrece la posibilidad de activar diferentes niveles de caché (planes de vCPUs) y diferentes enfoques (Full Request o Per-resolver) para cada uno de los data sources (puntos de interfaz a los microservicios).

En resumen, la correcta gestión y optimización de la caché son, per sé, un campo de continua mejora y de profundo análisis, con la que se puede llegar a niveles de rendimiento superiores a cualquier competencia, sin incurrir en facturas demasiado elevadas.

4. Política de accesos y gestión de usuarios

Tal y como se ha desarrollado en el *Apartado 2.4: Amazon Cognito* del presente Anexo, como también en el *Apartado 1.6.4: Problemática IV: Enfoque de la seguridad* del *Documento Principal* del presente Trabajo Fin de Máster, la política de accesos y la gestión de los usuarios, que está a su vez ligado con la seguridad de la propia API, se ha llevado a cabo mediante un enfoque de Identidad Federada, a través del servicio AWS Cognito.

4.1. Definición de niveles de usuario

Tal y como ocurre en cualquier SaaS, y sin olvidar de que se trata de el producto de una empresa, existen diferentes planes a contratar por parte de los clientes. Diferentes planes que incluyen diferentes posibilidades y funcionalidades. Además, al tratarse de un marketplace de economía circular empresarial, las empresas compradoras de activos fijos pueden también ser puntualmente vendedoras. Además, dada la naturaleza del mercado donde se opera, existen entidades que son vendedoras en masa; por ejemplo, entidades bancarias que tras procedimientos concursales disponen de cientos de activos fijos a los que deben dar salida, y que disponen de planes acorde.

Teniendo todo ello en cuenta, desde dirección se plantea la agrupación de usuarios en seis diferentes posibilidades.

- Grupo_admin: Orientado a tareas de administración, tiene acceso a los métodos de administración, además de a todos los métodos de usuario.
- Grupo_buyer_tier_0: Es el principal grupo otorgado a todo usuario en el momento de la creación de la cuenta. Permite generar transacciones con funcionalidades limitadas. En principio se trata de un grupo freemium.
- Grupo buyer_tier_1: Upsell del grupo previo y orientado bajo una suscripción.
- Grupo seller_tier_ocasional: Es el principal grupo otorgado a todo usuario en el momento de publicación de un activo. Para la empresa ocasional, la operativa está basada en una comisión tras venta, los datos de contacto son los de un agente Profesiolan, y el activo se publica de forma blanqueada.
- Grupo_seller_tier_pro_1: Orientado a aquellas empresas que se dedican activamente a la venta de activos fijos de ocasión, pero que disponen de un volúmen de activos menor a 50.
- Grupo_seller_tier_pro_2: Ídem al grupo inmediatamente anterior, con un volúmen de activos superior y capacidades extras como sincronizaciones de stock y posibilidad de destacar.

A continuación, se muestran listados los grupos de usuarios mencionados, desde la consola de

administración provista por AWS, en la *Figura 60: Resumen de los grupos de usuario.*

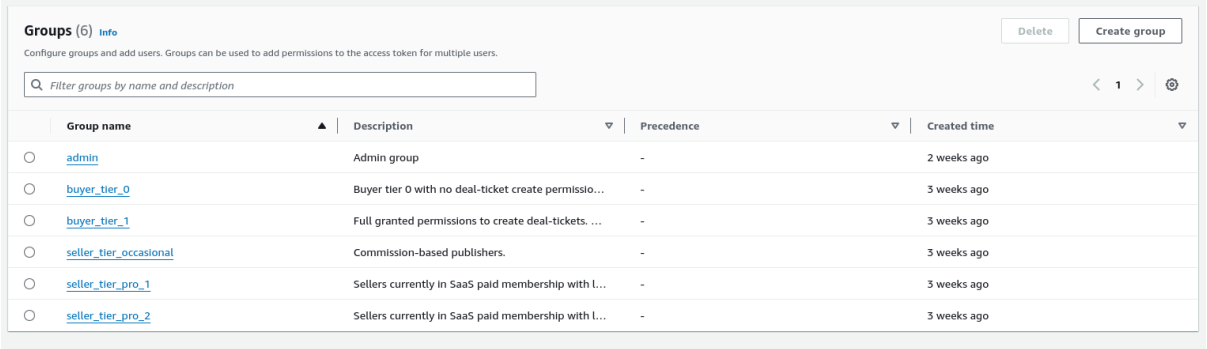

Figura 60: Resumen de los grupos de usuario.

Como resulta lógico, un mismo usuario puede pertenecer a varios grupos. De hecho, por defecto, tras la creación de una cuenta gratuita, la empresa cliente puede publicar activos de forma ocasional, además de poder filtrar anuncios y navegar por el marketplace con las capacidades de comprador freemium.

A continuación, se muestra la información más básica de un usuario registrado, junto a su perteneciencia a dos de los grupos mencionados en la *Figura 61: Información básica de un usuario dummy en el User Pool.*

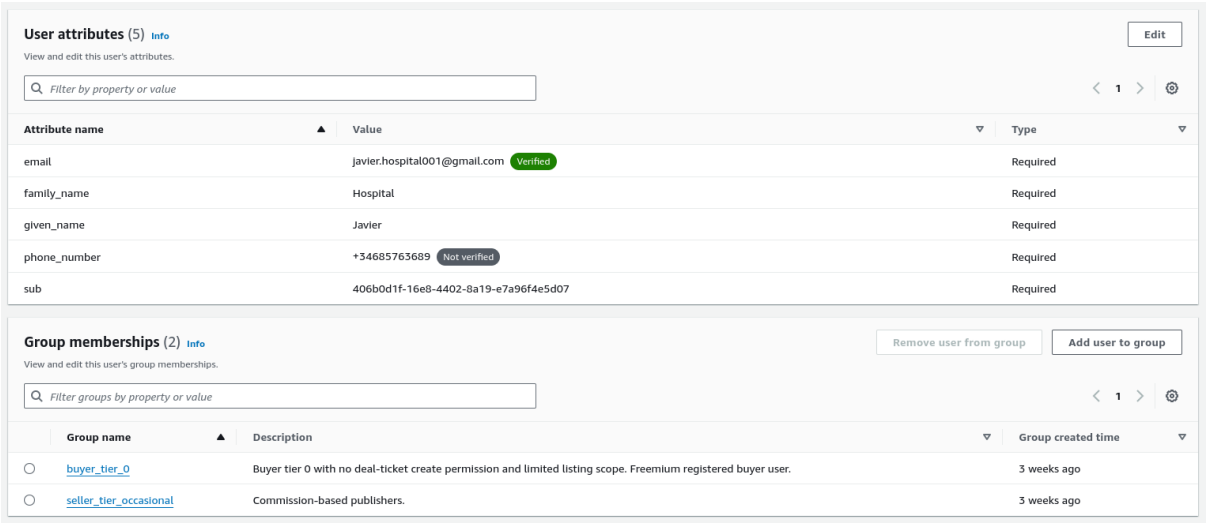

Figura 61: Información básica de un usuario dummy en el User Pool.

4.2. Proceso de creación de cuenta

Tal y como se ha desarrollado en el *Apartado 5.3.13: updateUser()*, la creación de usuarios ha sido desarrollada sobre el servicio Cognito. En este apartado se va a presentar, el proceso de creación de una cuenta de usuario.

Para ello, se ha desarrollado un formulario sencillo en un dominio de la propiedad Profesiolan, accesible a través del siguiente enlace:

https://profesiolan.auth.eu-west-2.amazoncognito.com/signup?client_id=5a9rj9a8g116vvl4kef9e9gqe1&response_type=code&scope=email+openid+phone&redirect_uri=https%3A%2F%2Fprofesiolan.com

Una vez generado el usuario, se deberá verificar la cuenta de correo electrónico, completando el siguiente paso del formulario, con el código de 6 dígitos envíado a la bandeja de entrada del correo electrónico a registrar, tal y como se presenta en la *Figura 62: Email de confirmación de correo electrónico tras registro.*

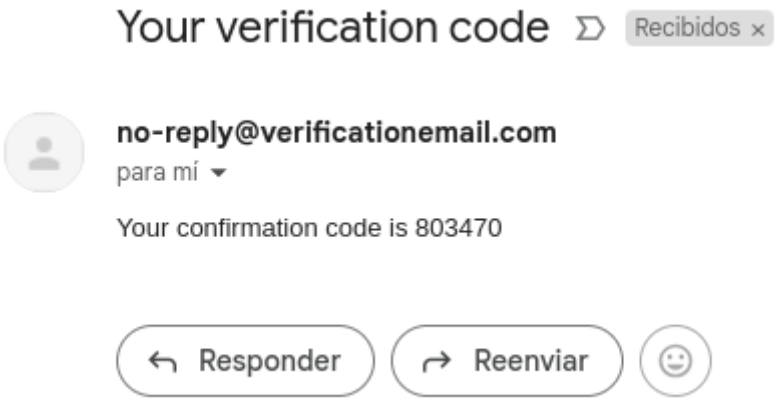

Figura 62: Email de confirmación de correo electrónico tras registro.

En el momento en el que el usuario completa el formulario, es dirigido directamente a un callback a la Landing de profesiolan.com, completando así el proceso de registro de usuario. En ese momento, el usuario pasa a pertenecer, de forma automática, al grupo Grupo *buyer_tier_0* y al grupo Grupo *seller_tier_ocasional*. Además, dada la naturaleza B2B de la solución, y debido a que el contacto

telefónico es parte indispensable en la operativa, Profesiolan también puede exigir la verificación telefónica por medio de un SMS.

El estado del usuario, en este preciso instante, sería el que se puede ver en la *Figura 61: Información básica de un usuario dummy en el User Pool,* con el flag de verificado en el correo electrónico, pero no en el número de teléfono, además de la perteneciencia a los dos grupos mencionados.

4.3. Proceso de inicio de sesión

En este apartado, se va a desarrollar el proceso de inicio de sesión para un usuario registrado. Para ello, se puede acceder directamente al mismo formulario habilitado en la siguiente dirección:

https://profesiolan.auth.eu-west-2.amazoncognito.com/login?client_id=5a9rj9a8g116vvl4kef9e9gqe1&response_type=code&scope=email+openid+phone&redirect_uri=https%3A%2F%2Fprofesiolan.com

No obstante, dado que se quiere analizar el flujo de tokens en detalle, el proceso de login se va a realizar a través de la herramienta de testeo de APIs Postman. Para ello, en el entorno preparado y para el cual se presenta una guía en el *Apartado XXX: Entorno Postman*, se establece la siguiente configuración:

- Tipo: OAuth 2.0
- Configuración de nuevo Token
    - Nombre del Token: Indiferente.
    - Tipo de grant: Código de autorización.
    - Callback: Indiferente.
    - URL de autorización: https://profesiolan.auth.eu-west-2.amazoncognito.com/oauth2/authorize
    - URL de acceso del Token: https://profesiolan.auth.eu-west-2.amazoncognito.com/oauth2/token
    - ID del cliente: 5a9rj9a8g116vvl4kef9e9gqe1
    - Secreto del cliente: Privado.
    - Alcance, Estado: Indiferente.

- Autenticación de cliente: Envío de las credenciales en el cuerpo.

De igual forma, se puede ver la configuración en la *Figura 63: Configuración del entorno Postman para iniciar el proceso de inicio de sesión de un usuario registrado.*

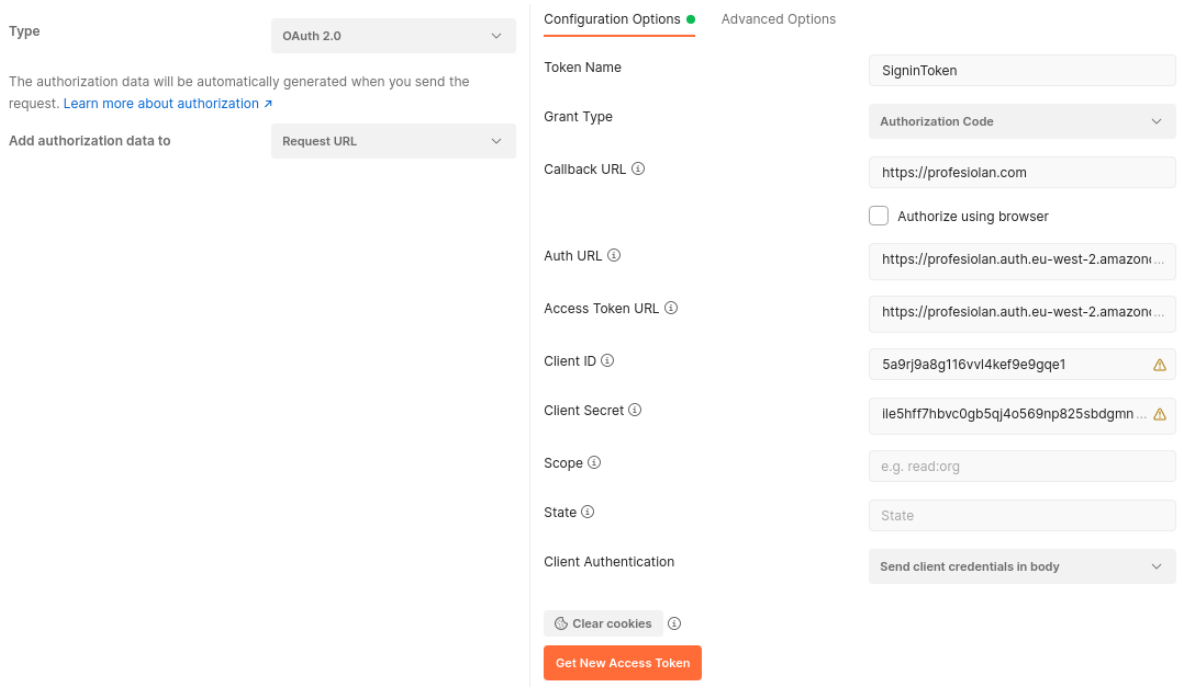

Figura 63: Configuración del entorno Postman para iniciar el proceso de inicio de sesión de un usuario registrado.

Una vez hecho clic en el botón *Get New Access Token*, el propio Postman abrirá, si tiene los permisos, una ventana de navegador, con la URL precisa que permite visualizar el formulario de inicio de sesión, tal y como se puede ver en la *Figura 64: Redirección de Postman al formulario de inicio de sesión.*

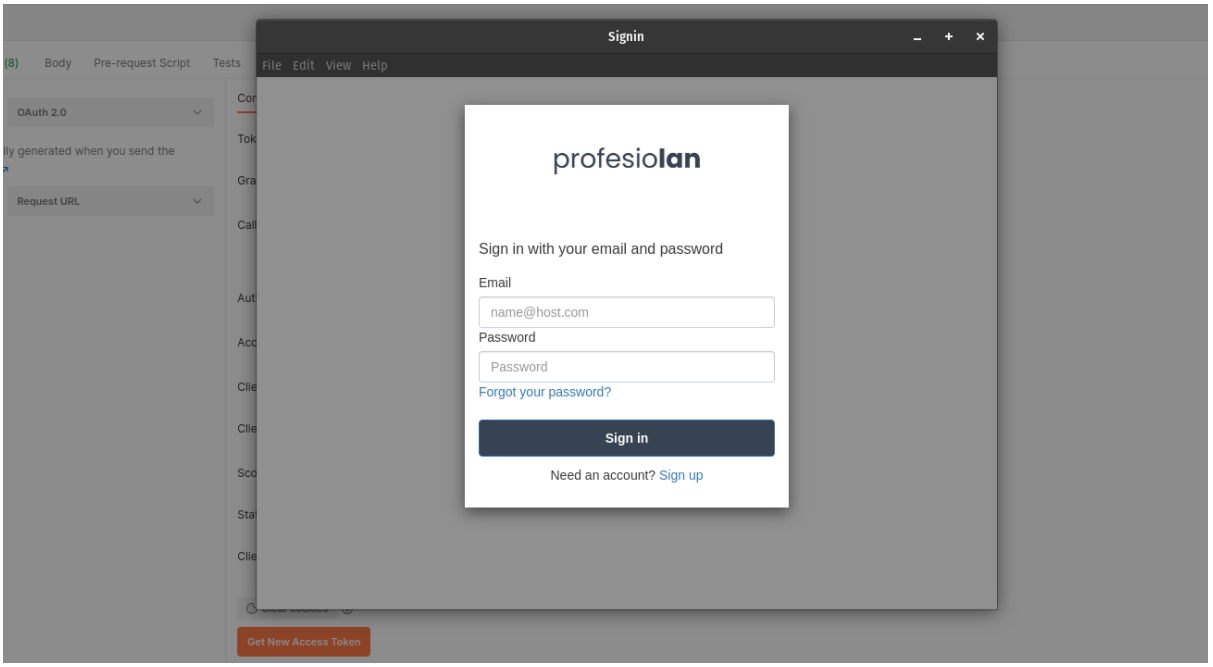

Figura 64: Redirección de Postman al formulario de inicio de sesión.

Se procede a rellenar la información correo electrónico y contraseña del usuario registrado, tal y como se puede observar en la *Figura 65: Formulario de inicio de sesión completado.*

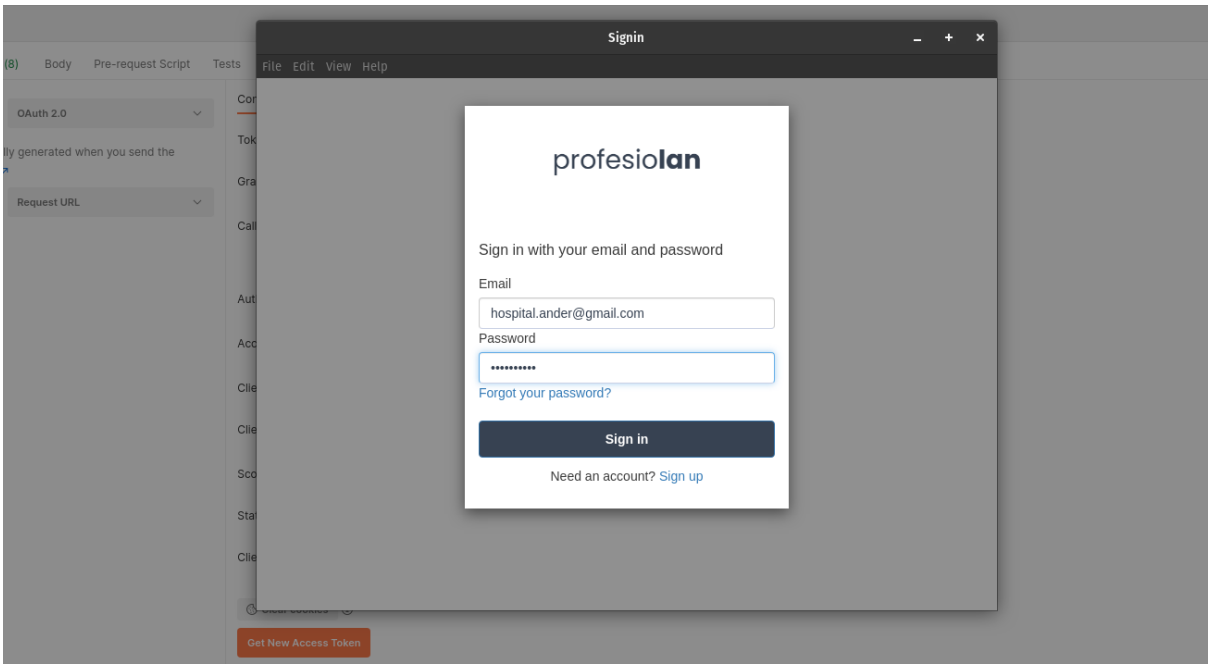

Figura 65: Formulario de inicio de sesión completado.

Tras un inicio de sesión válido, tal y como se puede observar en la *Figura 66: Respuesta del servidor tras login exitoso*, el backend responderá una confirmación que será interceptada por el propio cliente Postman y que, además, contendrá el token JWT con la información de usuario y de sesión. Está anotado el problema CSS relativo a la elección de colores de la notificación, debería corregirse en un futuro para que pueda ser más legible.

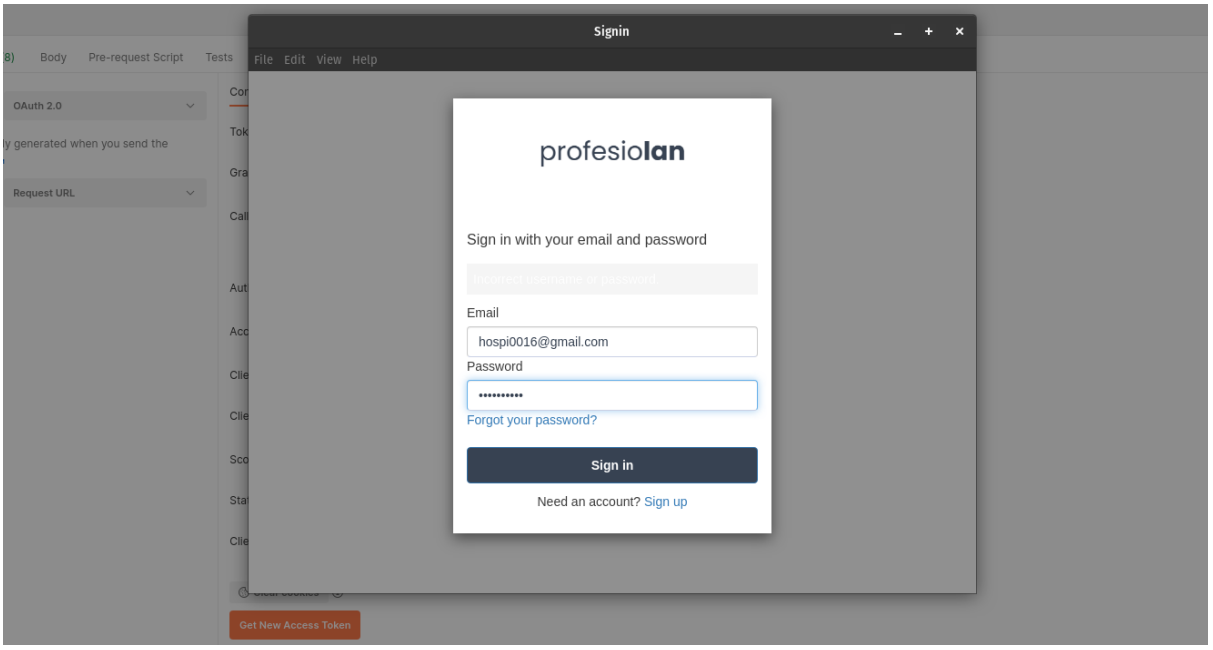

Figura 66: Respuesta del servidor tras login exitoso.

Automáticamente, tras la recepción del token, Postman eliminará la ventana del navegador, y ofrecerá a través de su entorno, el JWT devuelto por la API Profesiolan, tal y como se puede observar en la *Figura 67: Postman finaliza el proceso de autenticación.*

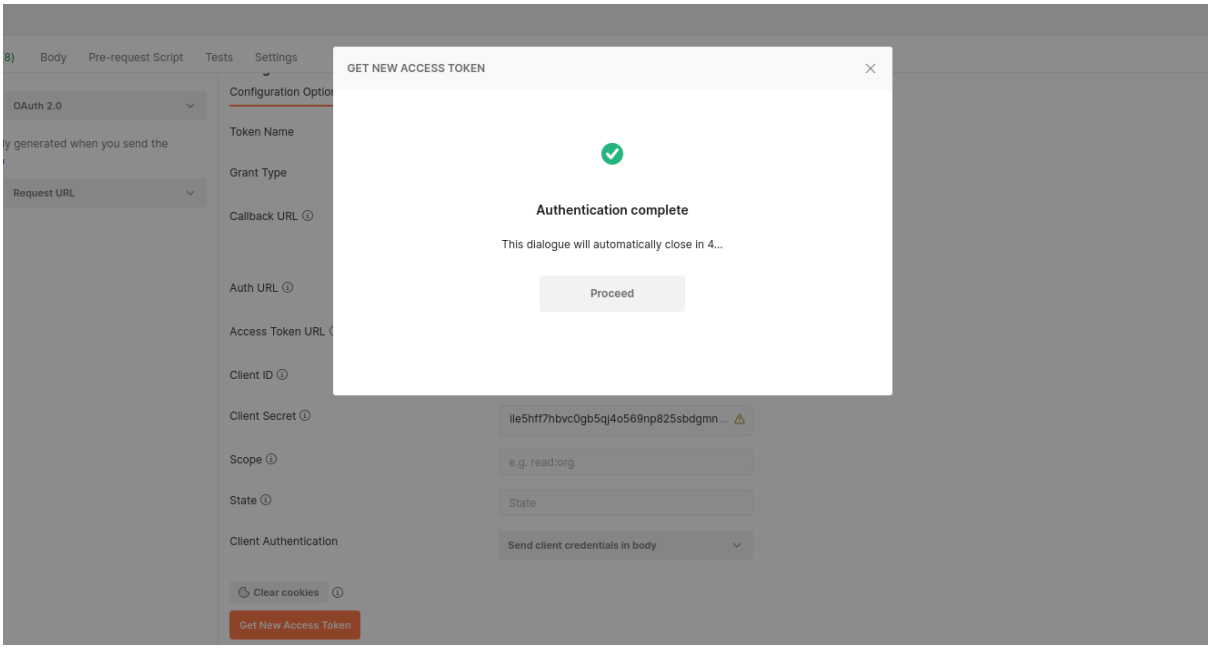

Figura 67: Postman finaliza el proceso de autenticación.

A partir de este momento, podremos acceder a este token en el mismo entorno, tal y como se puede observar en la *Figura 68: Postman almacena y muestra el token de sesión del usuario autenticado.*

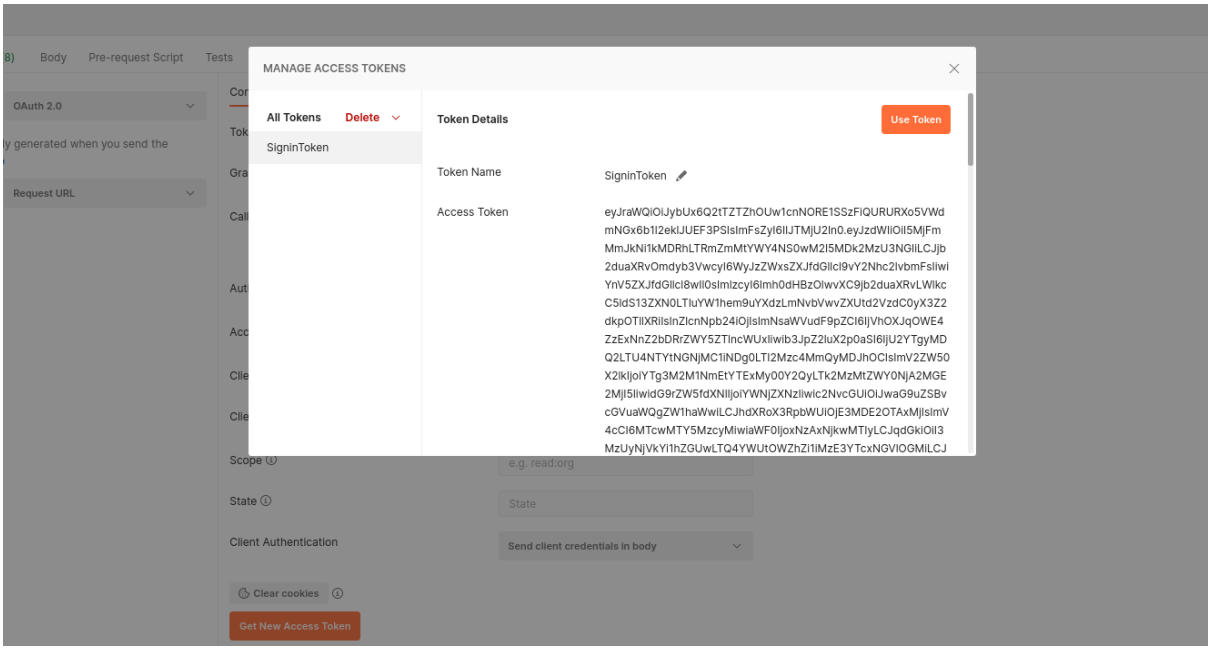

Figura 68: Postman almacena y muestra el token de sesión del usuario autenticado.

A continuación, resulta de especial interés analizar, en profundidad, toda la información que transporta el mencionado token. Para ello, se hace uso de una herramienta de decodificación de

tokens JWT como es https://jwt.io/, y como se puede observar en la *Figura 69: Decodificación del token JWT de sesión del usuario autenticado.*

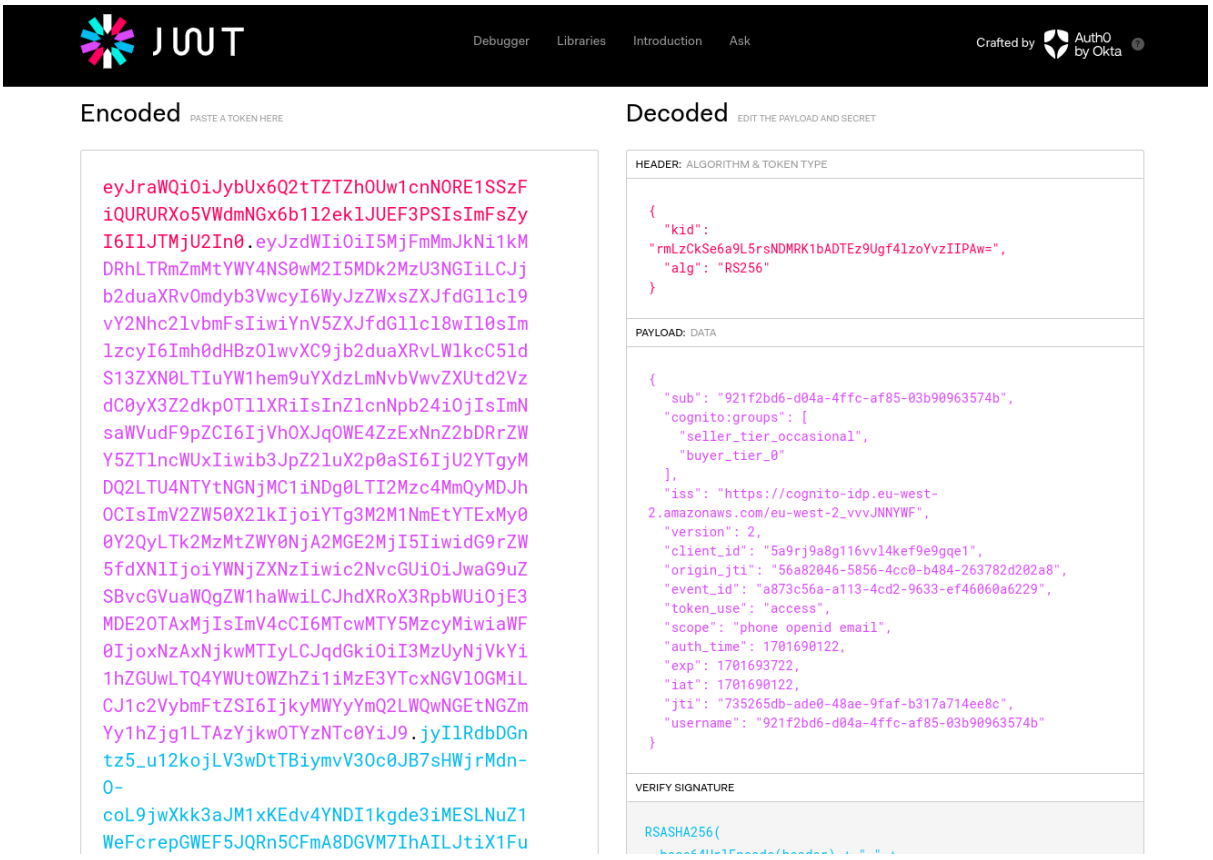

Figura 69: Decodificación del token JWT de sesión del usuario autenticado.

En este momento, se va a proceder a analizar todos y cada uno de los campos de información embebidos en el mencionado token.

- Cabecera:
    - kid: Key ID. Indica la clave que ha sido utilizada para la emisión del token.
    - alg: Algoritmo criptográfico utilizado para securizar el token de acceso. En este caso, se está utilizando un RS256, que es un RSA con SHA-256.
- Payload:
    - sub: Identificador UUID del usuario registrado en el User Pool de Profesiolan.
    - cognito groups: array de los grupos a los que el usuario registrado pertenece.
    - iss, version: Información del proveedor de identidad y su correspondiente versión.
    - client_id: Identificador del cliente Cognito de Profesiolan.
    - auth_time: Momento en el que el usuario completa la autenticación, en tiempo Unix.

- exp: Momento en el que el token expira, en tiempo Unix.
- iat: Momento en el que se ha expedido el token, en tiempo Unix.
- jti: Identificador único del JWT expedido.
- username: Nombre de usuario, coincide siempre con el sub.

- Verify Signature:
  - ID token signature: calculada en función de la cabezera y el cuerpo del JWT. Antes de aceptar tokens, es recomendable verificar la firma.

Una vez obtenido el token de acceso, se procede a probar uno de los métodos securizados, primero con un token incorrecto para, posteriormente, realizar la misma acción HTTP Post con el token correcto.

Como se puede observar en la *Figura 70: HTTP Post de publicación de activo con un token inválido I* y *Figura 71: HTTP Post de publicación de activo con un token inválido II*

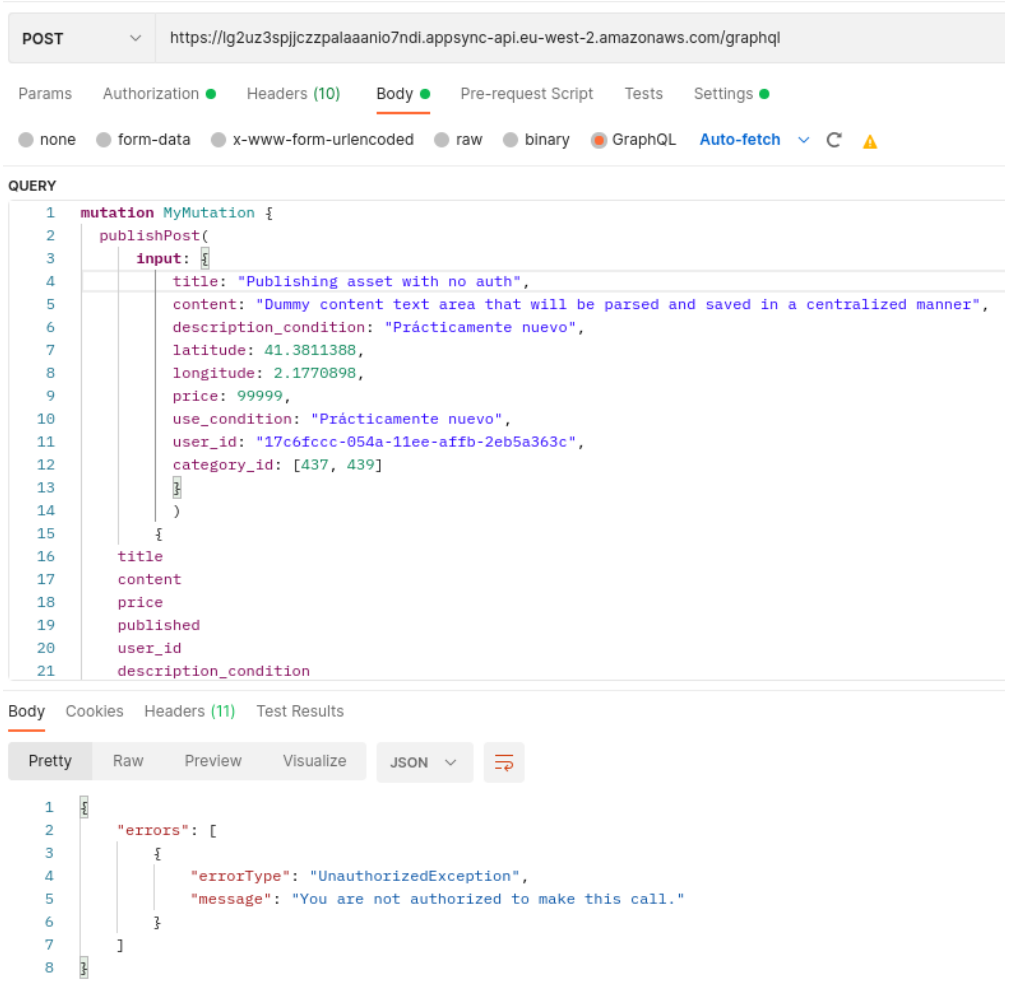

Figura 70: HTTP Post de publicación de activo con un token inválido I.

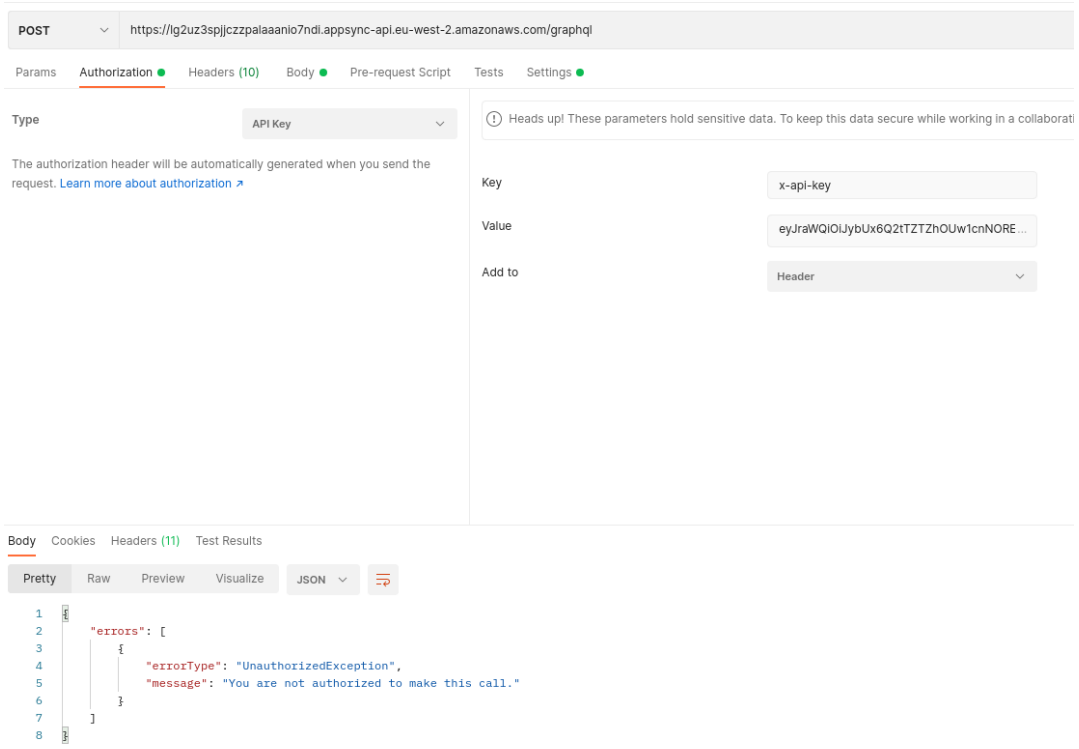

Figura 71: HTTP Post de publicación de activo con un token inválido II.

Como se puede observar, la API devuelve una respuesta de Error HTTP 401, con una descripción de mensaje en la que se traduce el mismo mensaje.

A continuación, se procede a realizar la misma mutación a la API, esta vez con el token de acceso obtenido para el usuario registrado y autenticado. Tal y como se puede observar en la *Figura 72: Publicación exitosa de activo con método protegido para usuario autenticado,* la API devuelve un mensaje de éxito HTTP 200, con el cuerpo de mensaje descrito en la API GraphQL bajo la propiedad PublishPostOutput.

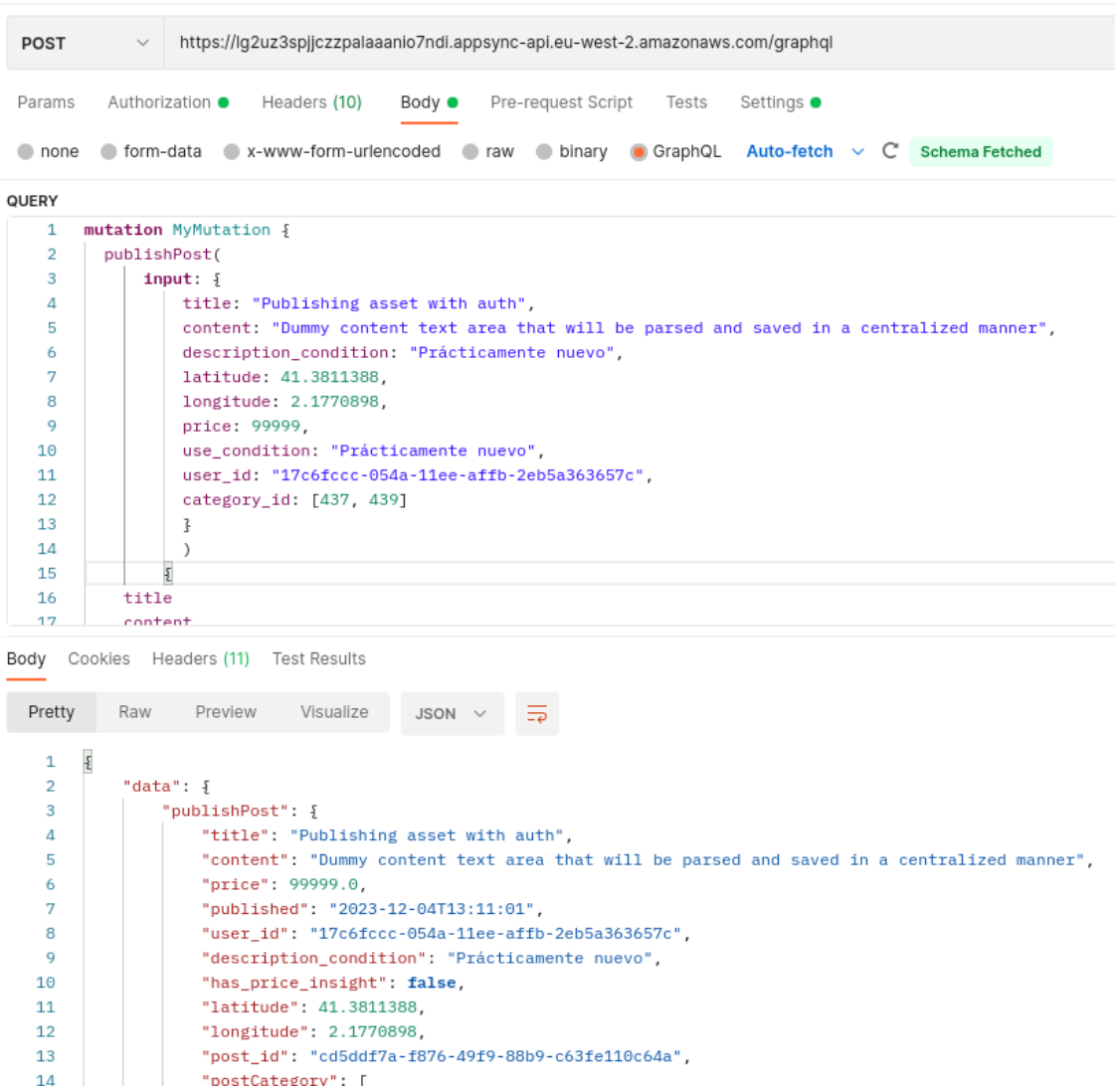

Figura 72: Publicación exitosa de activo con método protegido para usuario autenticado.