

# Prueba Técnica

25 de noviembre de 2021

## Descripción

Basado en el diagrama Entidad Relación Proporcionado por Antonio Padrón, Encargado de IT.  
Realizar tres Servicios Restful donde:

1. Registrar Customers.
2. Consultar Customers por DNI o Email.
3. Eliminar Customers del sistema.

## Metas

1. Al registrar, asociar la commune y region del cliente, hacer todas las validaciones pertinentes. Ejm no permitir registro de customer en commune y regiones que no estén relacionadas o no existan.
2. La consulta debe hacerse solo con customer activos (A), no con desactivo (I) o eliminados (trash), adicionalmente deberá retornar name, last\_name, address (de no tener address retorna null en el campo), description region y commune. Realizar validaciones pertinentes.
3. El customer a eliminar debe de estar activo (A) o desactivo (I). En el caso de estar ya eliminado (trash) retornar "Registro no existe". Hacer validaciones pertinentes.
4. . El código debe de estar protegido para sql Injection y con un key de autenticación (Debe implementarse a través de middlewares).

## Requerimientos mínimos

1. Entorno de desarrollo y gestión de base de datos: **XAMPP**.
2. Manejador de paquetes PHP: **Composer**.
3. Editor de texto: **Visual Studio Code**.
4. Framework para PHP: **Laravel**.
5. Sistema de control de versiones: **GIT**.
6. Cliente HTTP: **Postman**

## Instalación

### *Paso 1 Clonar repositorio*

Como primer paso debemos clonar el repositorio del proyecto. Para ello nos dirigimos a la carpeta donde deseamos que se clone el repositorio y abrimos una terminal de git.

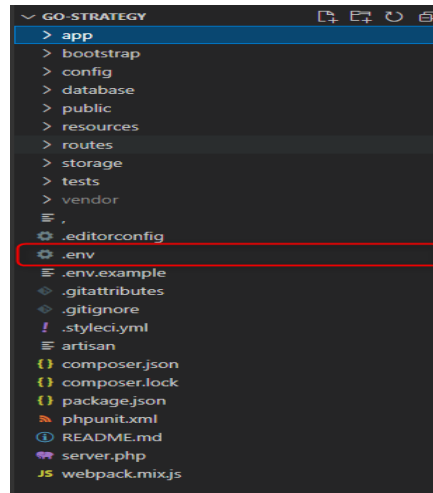
Posteriormente escribimos el siguiente comando:

```
git clone https://github.com/Juan-Carlos-San-Lpz/Go\_Strategy
```

Esperamos a que se clone correctamente todo el repositorio, y abrimos la carpeta con el editor de texto.

### *Paso 2 Configurar .env*

Una vez clonado el repositorio y estando abierto con el editor de texto nos dirigimos al archivo .env que se encuentra en la carpeta principal.



Abrimos el archivo y nos aseguramos que este los datos correctos en la conexión a mysql (línea 9-10 aproximadamente), nos aseguramos que estén todos nuestros datos correctos tomando en cuenta que el username y el password pueden variar segun la configuracion que tenga cada equipo. Verificar que el nombre de la base sea mydb.

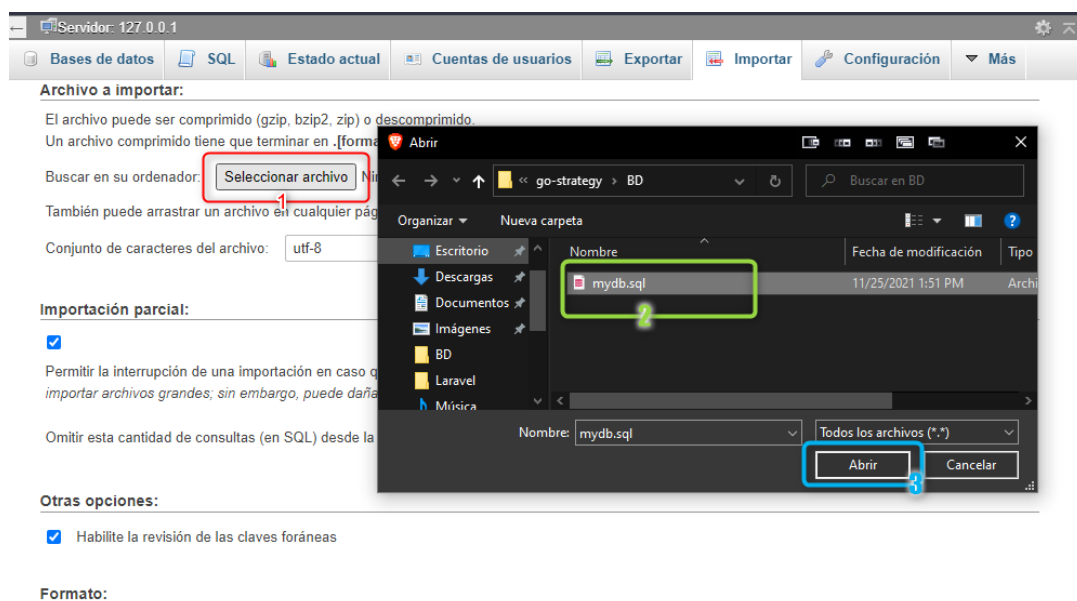
```
9 DB_CONNECTION=mysql
10 DB_HOST=127.0.0.1
11 DB_PORT=3306
12 DB_DATABASE=mydb
13 DB_USERNAME=root
14 DB_PASSWORD=
```

### ***Paso 3 Importación de la base de datos***

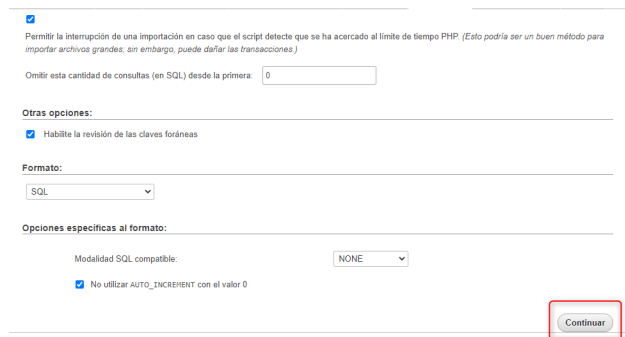
Como tercer paso debemos crear la base de datos correspondiente a la prueba para ello abriremos nuestra administrador de base de datos y damos clic en importar.



Mostrará una página para seleccionar nuestro archivo sql para la importación, damos clic en seleccionar archivo, buscamos el archivo llamado mydb.sql que se encuentra dentro de nuestra carpeta del proyecto en una carpeta llamada BD, seleccionamos el archivo y damos clic en abrir.



Nos desplazamos un poco más abajo de la página y damos clic en continuar.



The screenshot shows a configuration form for a database import. It includes a checkbox to allow script interruption, a text input for the number of queries to skip (set to 0), a section for other options with a checkbox to enable foreign key checks, a format dropdown menu (set to SQL), and a section for specific format options with a dropdown for SQL compatibility (set to NONE) and a checkbox to not use AUTO\_INCREMENT (checked). A red box highlights the 'Continuar' button at the bottom right.

Una vez realizados estos pasos estamos listos para comenzar a probar las API'S.

#### ***Paso 4 Correr nuestro programa de laravel***

Un vez realizado los pasos anteriores estamos listo para levantar nuestro servidor de laravel. para ellos es necesario escribir el siguiente comando en la terminal de nuestro editor de texto:

```
php artisan serve
```

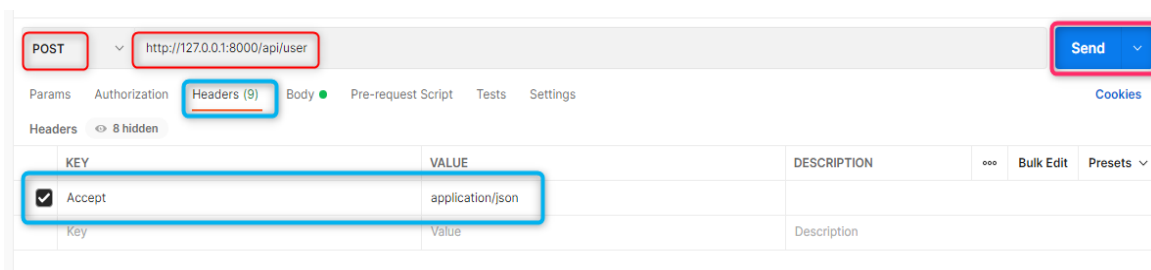
Esperamos un momento y el servidor debe estar arriba

```
linki@DESKTOP-K4TE2NV MINGW64 ~/Documents/Laravel/go-strategy (main)
$ php artisan serve
Laravel development server started: http://127.0.0.1:8000
[Thu Nov 25 14:08:37 2021] PHP 8.0.12 Development Server (http://127.0.0.1:8000) started
```

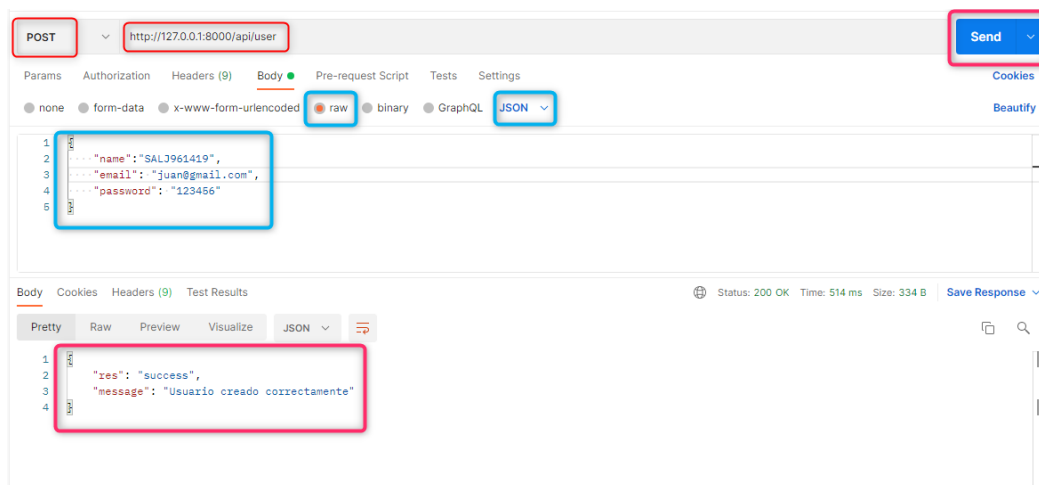
#### ***Paso 5 Realizar pruebas HTTP***

Por último abrimos abrimos nuestro cliente HTTP en este caso la aplicación de Postman para la realizar las pruebas correspondientes.

Como primera prueba es necesario registrar un nuevo usuario ya que el sistema está protegido mediante la authorization y requerimos como primer paso registrar un usuario.

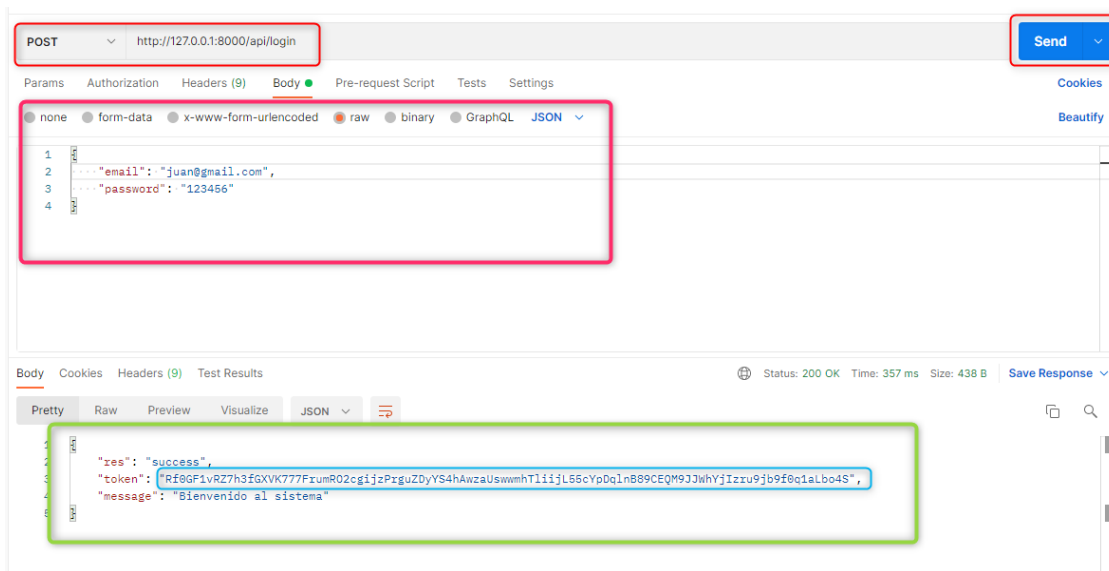


Comprobar que estamos haciendo una petición POST, y verificar que la url sea la correspondiente a la de nuestra aplicación, una vez comprobado los pasos anteriores asegurarnos que acepte archivos JSON Como lo muestra la imagen de arriba.

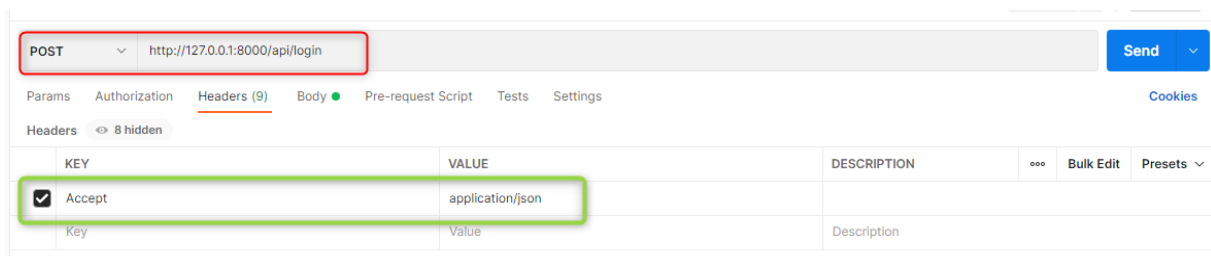


Como siguiente paso dar clic a Body, seleccionar el radio de raw y elegir JSON, Enviar un JSON Con el name, email y password una vez teniendo eso damos clic en Send y nos retornará un mensaje de succes donde indica que el usuario a sido registrado correctamente.

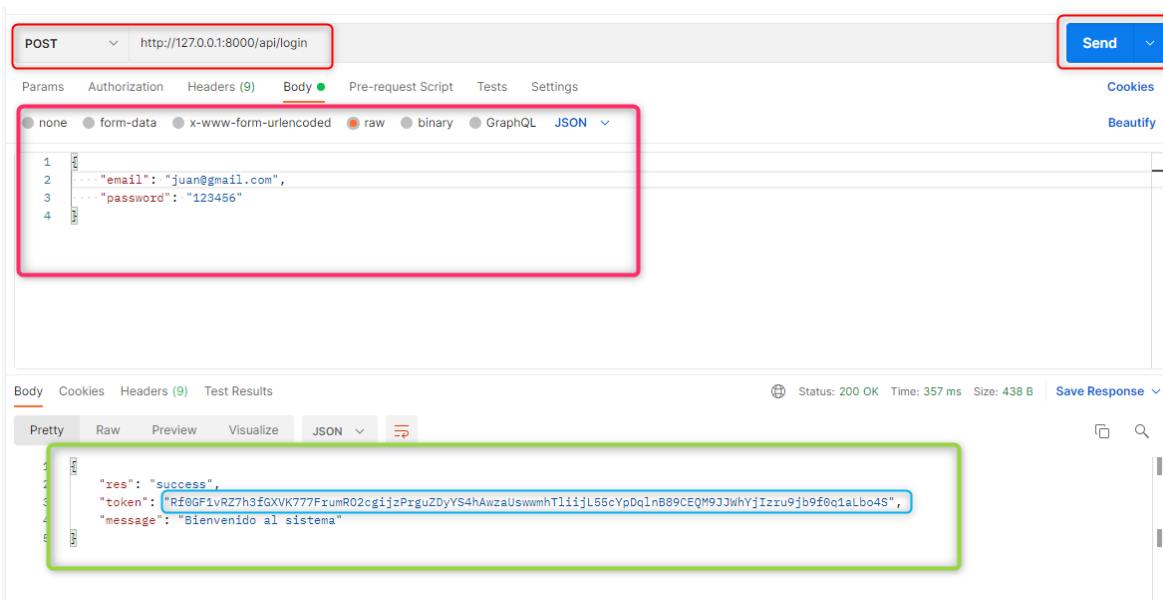
Una vez tengamos el nuevo usuario registrado es necesario hacer login para poder obtener el token de verificación. Realizando la siguiente prueba:



Al igual que la prueba anterior primero debemos verificar que la petición que realizamos sea mediante POST y la url sea la indicada para el login, al igual que acepte archivos JSON.

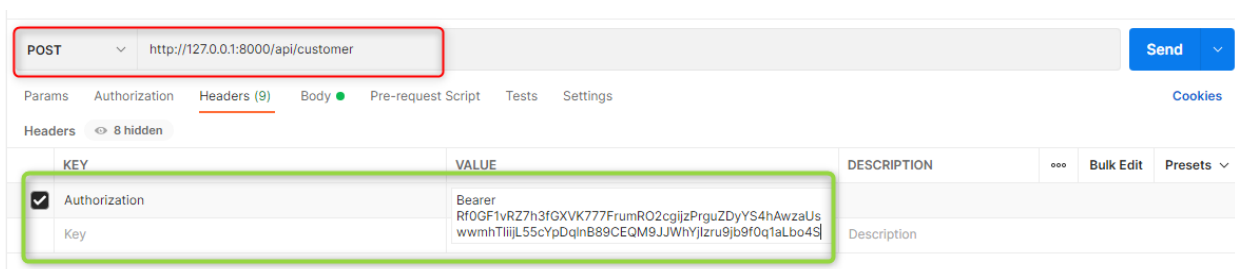


Una vez Verificado los datos anteriores dar clic en body y marcar la casilla de raw, escribir el json con el email y la contraseña y enviar la petición.



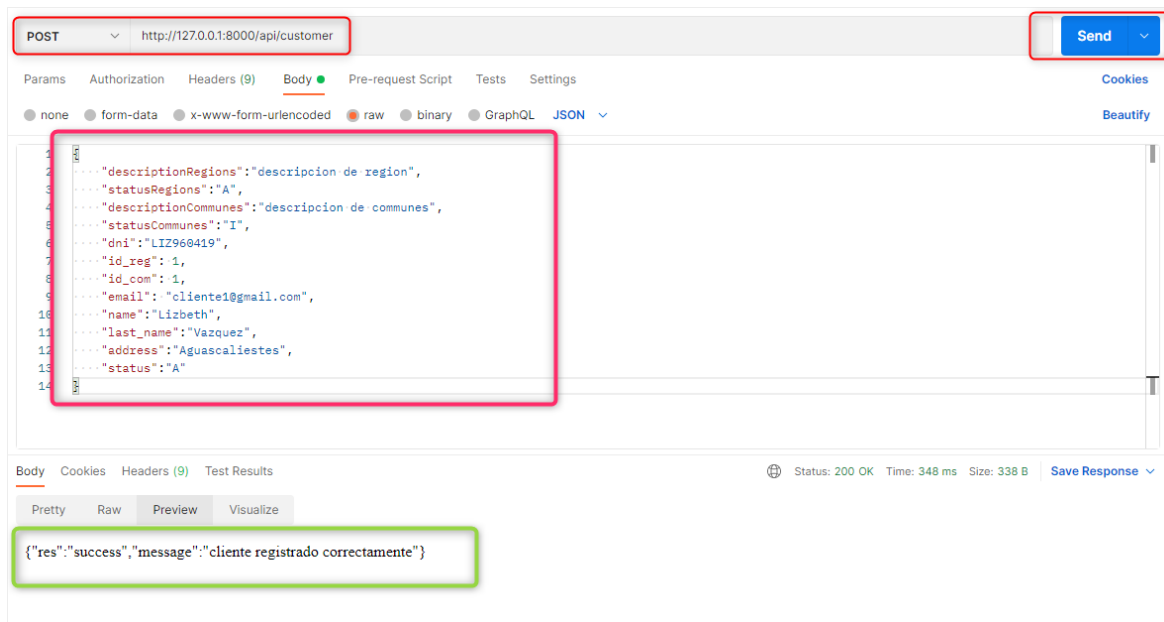
Un vez enviada la petición si los datos son correctos nos retornará un succes con el token que será necesario para la identificación de las rutas siguientes, al igual de un mensaje de bienvenida.

Una vez tengamos nuestro token podemos acceder a la ruta de customer para poder hacer el registro del mismo y así realizar las pruebas siguientes. Al igual que las pruebas anteriores debemos de asegurarnos que estamos haciendo una petición mediante POST y verificar la ruta correspondiente a customer, a diferencia de las pruebas anteriores debemos estar autorizados para acceder a la ruta para ello damos clic a Headers Seleccionamos Authorization y en la casilla de value es importante poner la palabra Bearer seguido del token que nos arrojó anteriormente al momento de loguearnos.



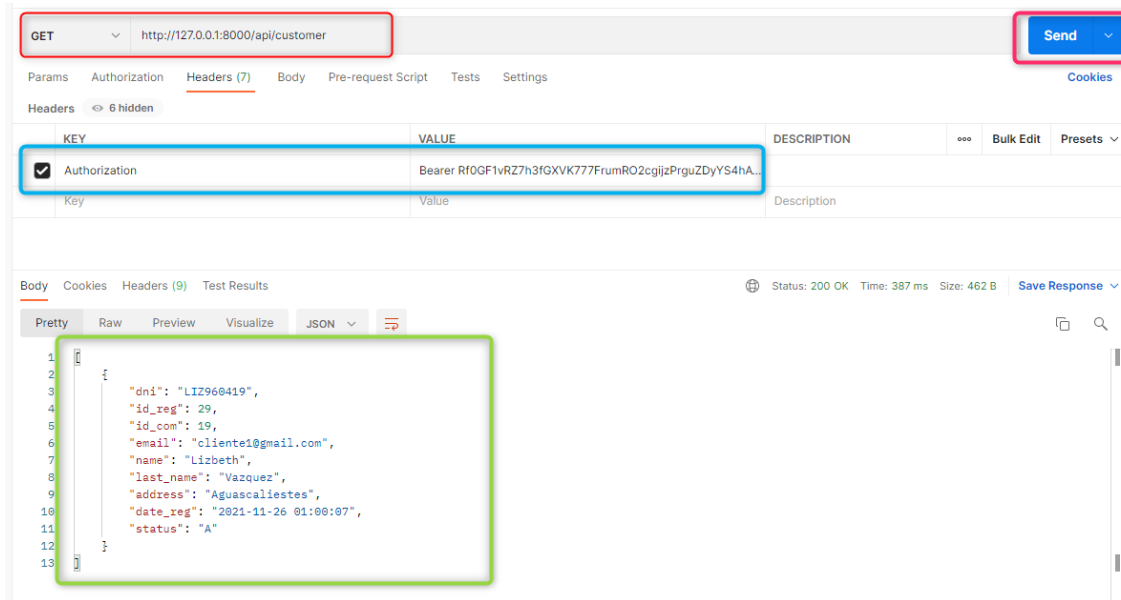


Una vez realizado lo anterior mencionado dar clic en body y seleccionar json, realizar un json con los datos de region, communes y customer y enviar la petición.



Si los datos son correctos se retornara un succes con un mensaje de cliente registrado correctamente.

Para poder consultar los clientes registrados debemos realizar una petición mediante el método GET, asegurándonos que seguimos autenticados con el token.



Si la autenticación y la petición fue enviada con la url correcta y el método GET, Se retornará los datos de todos los clientes registrados en la base de datos.

Para una búsqueda más específica podemos realizar la misma petición, con el mismo método GET y la misma URL poniendo al final de la línea un signo de interrogación cerrado la palabra search signo de igual y el dni o email a buscar. Ejem <http://127.0.0.1:8000/api/customer?search=LIZ>

GET  Send

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Headers ☐ 6 hidden

KEY	VALUE	DESCRIPTION	...	Bulk Edit	Presets
<input checked="" type="checkbox"/> Authorization	Bearer Rf0GF1vRZ7h3fGXVK777FrumRO2cgijzPrguZDyYS4hA...				
Key	Value	Description			

Body Cookies Headers (9) Test Results Status: 200 OK Time: 256 ms Size: 489 B Save Response

Pretty Raw Preview Visualize JSON ☐

```

1  {
2    "res": "success",
3    "customer": {
4      "dni": "LIZ960419",
5      "id_reg": 29,
6      "id_com": 19,
7      "email": "cliente1@gmail.com",
8      "name": "Lizbeth",
9      "last_name": "Vazquez",
10     "address": "Aguascalientes",
11     "date_reg": "2021-11-26 01:00:07",
12     "status": "A"
13   }
14 }

```

Como se puede observar no es necesario poner todo el dni ya que la api con ayuda de like en la sentencia sql buscan los usuarios que tengan coincidencias con el dni, de igual forma la api solo puede realizar búsquedas donde el status del cliente sea "A", (Activo). Si el usuario no encuentra el usuario o el satus de diferente de "A" nos retornará un mensaje de error como el siguiente:

GET  Send

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Headers ☐ 6 hidden

KEY	VALUE	DESCRIPTION	...	Bulk Edit	Presets
<input checked="" type="checkbox"/> Authorization	Bearer Rf0GF1vRZ7h3fGXVK777FrumRO2cgijzPrguZDyYS4hA...				
Key	Value	Description			

Body Cookies Headers (9) Test Results Status: 200 OK Time: 331 ms Size: 330 B Save Response

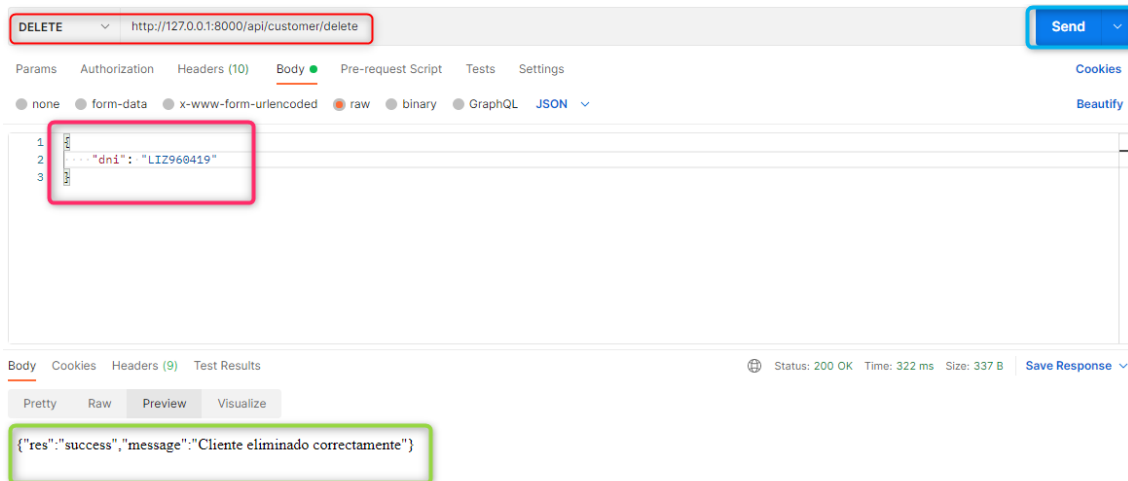
Pretty Raw Preview Visualize JSON ☐

```

1  {
2    "res": "error",
3    "message": "Error al buscar el cliente"
4  }

```

Como última prueba eliminaremos un cliente, Este solo se puede eliminar si su status es Activo "A" o Inactivo "I", para ello necesitamos realizar una petición DELETE, enviando en un JSON el dni del cliente que se desea eliminar, asegurarnos que se realice la petición de manera correcta poniendo la autenticación y enviarlo con el método correcto como a la url correcta.



Si la petición fue correcta nos retornará un success con un mensaje del cliente eliminado correctamente.

## Definición de funciones

Para esta prueba desarrollada con Laravel. Se desarrollaron distintas funciones, cada una de ellas dedicadas a realizar una tarea específica. Para un mejor entendimiento del programa el código fue comentado de una manera breve, clara y concisa. A Continuación presento un ejemplo del código comentado para el desarrollo de la prueba técnica.

```

/**
 * Esta función registra valida toda la información de Region, commune, customer. Registra y asocia region, commune, customer
 * @param Request Objeto sobre el que podemos consultar información sobre el cliente que realiza la solicitud
 * @return Customer Retorna success y mensaje de cliente registrado correctamente
 */
public function store(Request $request)
{
    // Obtenemos la información enviada desde Post como un archivo Json y Hacemos la validaciones correspondientes a cada campo
    $validate = Validator::make($request->all(), [
        'descriptionRegions' => 'required',
        'statusRegions' => 'required',
        'descriptionCommunes' => 'required',
        'statusCommunes' => 'required',
        'dni' => 'required|unique:customers,dni',
        'id_reg' => 'required',
        'id_com' => 'required',
        'email' => 'required|unique:customers,email',
        'name' => 'required',
        'last_name' => 'required',
        'status' => 'required',
    ]);

    // Validamos que no ocurra ningún fallo con las validaciones de los campos
    if ($validate->fails()) {
        return response()->json([
            'res' => 'error',
            'message' => 'Error al registrar el cliente',
            'errors' => $validate->errors()
        ], status:500);
    } else {
        // Creamos una instancia de Region Para poder guardar en la base de datos con los campos ya validados
        $region = new Region();
        $region->description = $request->descriptionRegions;
        $region->status = $request->statusRegions;
        $region->save();
    }
}

```

Para más información del funcionamiento de cada función por favor revisar el código fuente, en el encontrará la descripción y un poco más de información de cada uno de ellos.