Sprint 2

Universidad Nacional de Colombia

Docente a cargo
Brayan Steven Garcia

Integrantes
Maria Paula Torres
Anderson Daza
Santiago Rodiguez
Laura Patarroyo
Julian Mora

Grupo
P14

Grupo de trabajo
3

Repositorio:
https://github.com/4a-docs/4a-ms1
https://github.com/4a-docs/4a-ms2/tree/master

Microservicio AuthMS

Descripción: Framework Laravel

Modelos:

Models/User.php

En este archivo se definen los atributos del modelo al cual hace referencia a la migración de users, es de resaltar que en este modelo se realiza la asignación masiva que consiste en definir los campos que son requeridos para crear registros de users, estos campos se definen en la variable protegida $fillable.

```php
<?php

namespace App\Models;

use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use Laravel\Sanctum\HasApiTokens;
use Tymon\JWTAuth\Contracts\JWTSubject;
use Spatie\Permission\Traits\HasRoles;

class User extends Authenticatable implements JWTSubject
{
    use HasApiTokens, HasFactory, Notifiable, HasRoles;

    /**
     * The attributes that are mass assignable.
     *
     * @var string[]
     */
    protected $fillable = [
        'name',
        'last_name',
        'phone',
        'eps',
        'identification',
        'birthdate',
        'email',
        'password',
        'admin_id',
    ];

```

```
39      protected $hidden = [
40          'password',
41          'remember_token',
42      ];
43
44      /**
45       * The attributes that should be cast.
46       *
47       * @var array
48       */
49      protected $casts = [
50          'email_verified_at' => 'datetime',
51      ];
52
53      public function getJWTIdentifier()
54      {
55          return $this->getKey();
56      }
57
58      public function getJWTCustomClaims()
59      {
60          return [];
61      }
62
63  }
```

Migraciones:

database/migrations

En esta migración se especifica cada uno de los campos que se van a crear al momento de realizar la migración para crear la tabla users.

```
1   <?php
2
3   use Illuminate\Database\Migrations\Migration;
4   use Illuminate\Database\Schema\Blueprint;
5   use Illuminate\Support\Facades\Schema;
6
7   class CreateUsersTable extends Migration
8   {
9       /**
10       * Run the migrations.
11       *
12       * @return void
13       */
```

```php
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('last_name');
            $table->string('phone');
            $table->string('eps');
            $table->string('identification');
            $table->date('birthdate');
            $table->string('email')->unique();
            $table->timestamp('email_verified_at')->nullable();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
            $table->unsignedBigInteger('admin_id')->nullable();

            $table->foreign('admin_id')
                ->references('id')->on('users')
                ->onDelete('set null');

        });
    }
```

Rutas:

routes/api.php

En las rutas es donde se define cada endpoint de la api, de igual manera se establece el método, el controlador y método a ejecutar para cada una (ruta).

```php
<?php

use App\Http\Controllers\UserController;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;
```

```php

Route::post('register', [UserController::class, 'register']);
Route::post('login', [UserController::class, 'authenticate']);

Route::group(['middleware' => ['jwt.verify']], function() {

    Route::get('user', [UserController::class, 'getAuthenticatedUser']);

});
```

Controladores:

Controllers/UserController.php

En el controlador UserController se definen las diferentes acciones que se pueden llevar a cabo cuando es llamado este controlador por route/api, las acciones o funcionalidades que permite realizar este controlador, son:

- Authenticate: verificar si un usuario al momento de querer iniciar sesión cumple con los requerimientos del sistema.
- GetAuthenticatedUser: devuelve la información del usuario logueado.
- Register: permite el registro de usuarios con rol de pacientes, es de resaltar que este al momento del registro el sistema validará cada dato ingresado.
- Logout: permite que un usuario que está autenticado pueda cerrar sesión.

```php
<?php

namespace App\Http\Controllers;

use App\Models\User;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Facades\Validator;
use Tymon\JWTAuth\Exceptions\JWTException;
use Tymon\JWTAuth\Facades\JWTAuth;

class UserController extends Controller
{
    public function authenticate(Request $request)
    {
        $credentials = $request->only('email', 'password');
        try {
            if (!$token = JWTAuth::attempt($credentials)) {
                return response()->json(['error' => 'invalid_credentials'], 400);
            }
        } catch (JWTException $e) {
            return response()->json(['error' => 'could_not_create_token'], 500);
        }
        return response()->json(compact('token'));
    }

```

```php
    public function getAuthenticatedUser()
    {
        try {
            if (!$user = JWTAuth::parseToken()->authenticate()) {
                return response()->json(['user_not_found'], 404);
            }
        } catch (Tymon\JWTAuth\Exceptions\TokenExpiredException $e) {
            return response()->json(['token_expired'], $e->getStatusCode());
        } catch (Tymon\JWTAuth\Exceptions\TokenInvalidException $e) {
            return response()->json(['token_invalid'], $e->getStatusCode());
        } catch (Tymon\JWTAuth\Exceptions\JWTException $e) {
            return response()->json(['token_absent'], $e->getStatusCode());
        }
        $user = [
            'name' => auth()->user()->name,
            'last_name' => auth()->user()->last_name,
            'email' => auth()->user()->email,
            'eps' => auth()->user()->eps,
            'identification' => auth()->user()->identification,
            'birthdate' => auth()->user()->birthdate,
            'phone' => auth()->user()->phone,
            'role' => auth()->user()->roles[0]['name']
        ];
        return response()->json(compact('user'));
    }

```

```php
    public function register(Request $request)
    {

        $validator = Validator::make($request->all(), [
            'name' => 'required|string|max:40',
            'last_name' => 'required|string|max:40',
            'phone' => 'required|string|max:20',
            'eps' => 'required|string|max:40',
            'identification' => 'required|unique:users|string|max:15',
            'birthdate' => 'required|date',
            'email' => 'required|string|email|max:255|unique:users',
            'password' => 'required|string|min:6|confirmed',
        ]);

        if ($validator->fails()) {
            return response()->json($validator->errors()->toJson(), 400);
        }

        $user = User::create([
            'name' => $request->get('name'),
            'last_name' => $request->get('last_name'),
            'phone' => $request->get('phone'),
            'eps' => $request->get('eps'),
            'identification' => $request->get('identification'),
            'birthdate' => $request->get('birthdate'),
            'email' => $request->get('email'),
            'password' => Hash::make($request->get('password')),
        ])->assignRole('patient');

        $token = JWTAuth::fromUser($user);

        return response()->json(compact('user', 'token'), 201);
    }
```

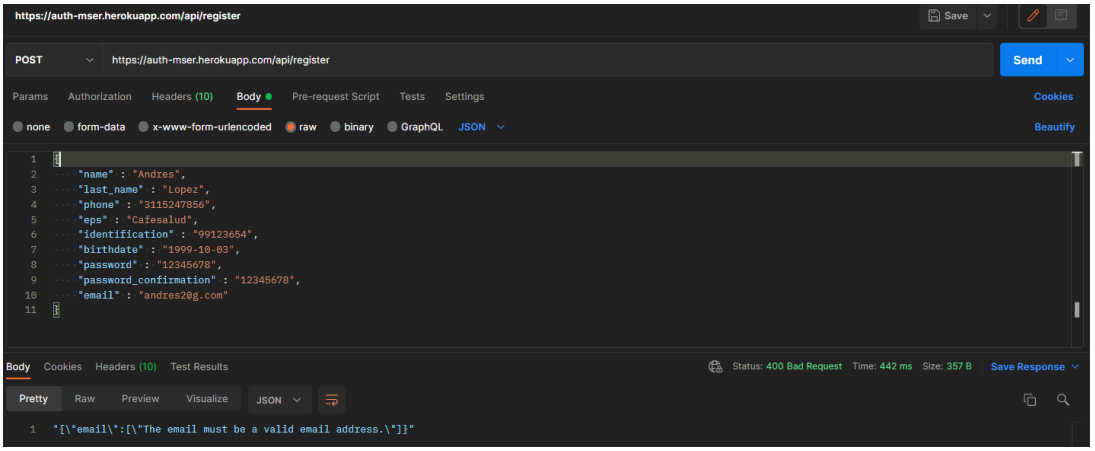
```php
    public function logout() {
        Auth::guard('api')->logout();

        return response()->json([
            'status' => 'success',
            'message' => 'logout'
        ], 200);
    }
}
```

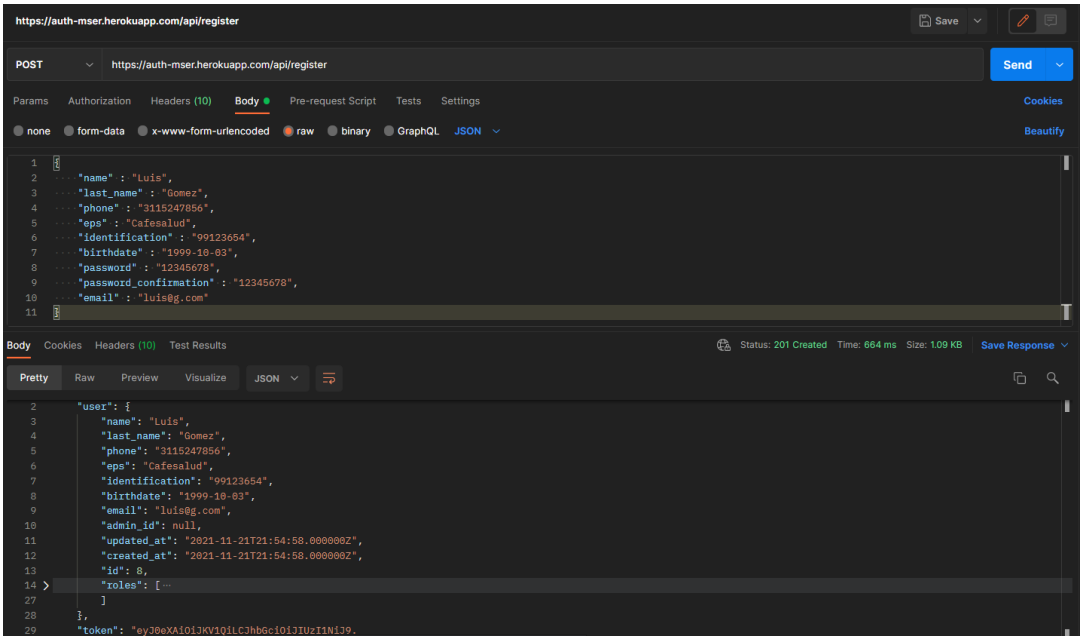
Postman: EndPoints

A continuación se realizó la prueba de cada ruta, para:

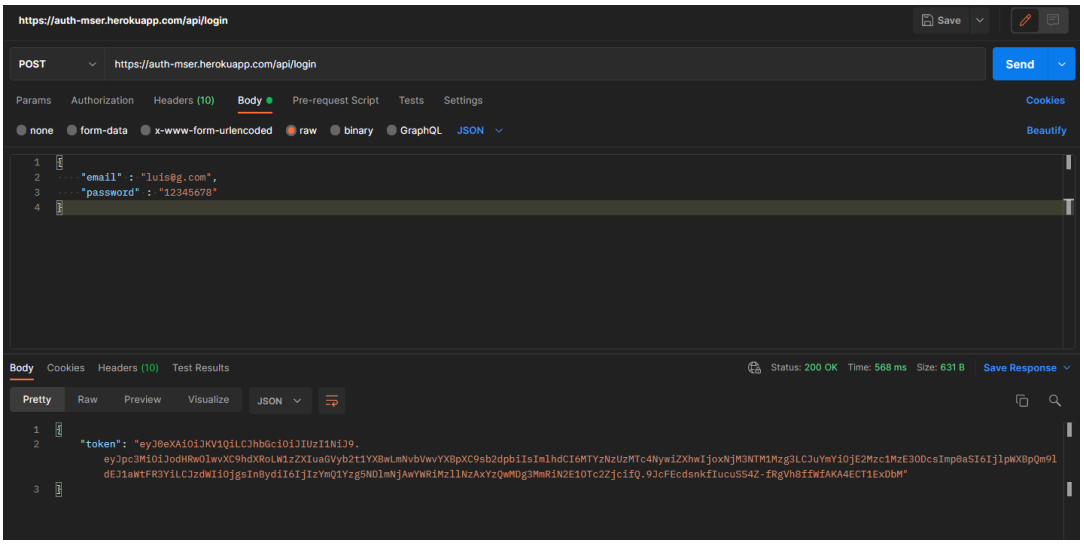
Registro de usuario y validación de campos.



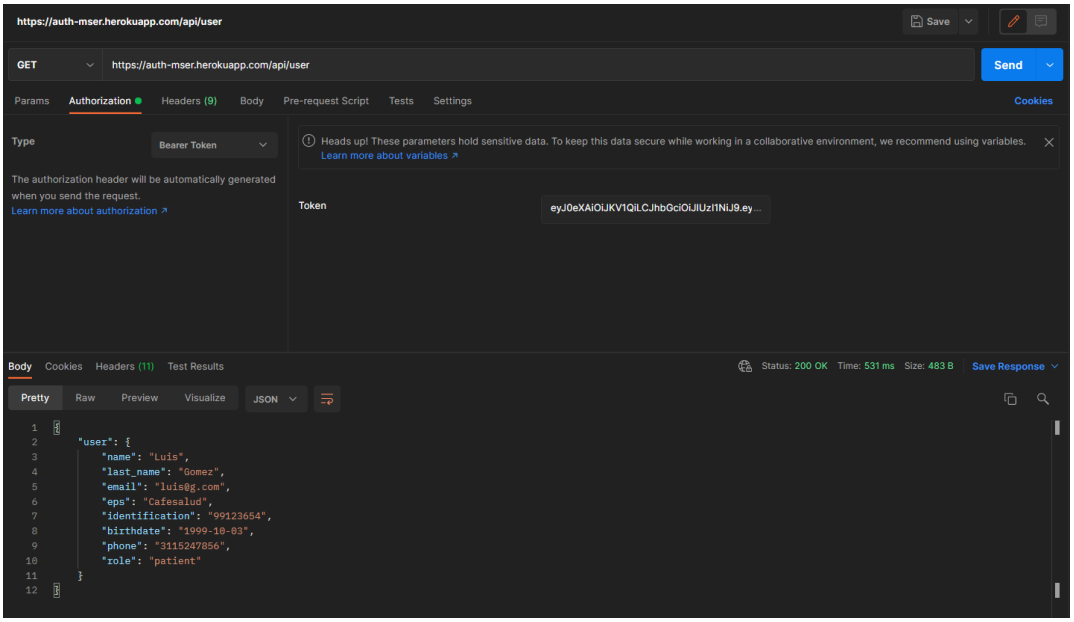
Registro exitoso de usuario



Inicio de sesión para cualquier tipo de usuario.



Conocer información del usuario autenticado .



Microservicio DoctorMS
Descripción: Framework Django

Modelos:
En este archivo se definen los datos del rol Doctor que se van a apreciar en la base de datos.

models.py

ms2_doctor > models.py > Doctor

```python
from django.db import models

# Modelo Doctor.

class Doctor(models.Model):
    id = models.AutoField(primary_key= True)
    nombre = models.CharField(max_length=32)
    numero_documento = models.CharField (max_length=15, unique= True)
    email = models.EmailField (max_length=100, unique = True)
    contraseña = models.CharField(max_length = 20)
```

Serializador:
Se pueden identificar la clase DoctorSerializer que nos permite transformar los datos a JSON.

serializer.py

ms2_doctor > serializer.py > ...

```python
from rest_framework import serializers
from .models import Doctor;

class DoctorSerializer(serializers.ModelSerializer):

    class Meta:
        model=Doctor
        fields='__all__'
```

Vistas:
En vistas .py vamos a realizar el modelo CRUD(create,read,update,delete)

views.py

ms2_doctor > views.py > ...

```python
from django.shortcuts import render
from rest_framework import generics, authentication, permissions
from .models import Doctor
from .serializer import DoctorSerializer

# Create your views here.
class DoctorListCreate(generics.ListCreateAPIView):
    queryset = Doctor.objects.all()
    serializer_class = DoctorSerializer

class DoctorUpdateDelete(generics.RetrieveUpdateDestroyAPIView):
    queryset = Doctor.objects.all()
    serializer_class = DoctorSerializer
```

Urls ms2:
Las urls son las que nos ayudan a poder desplegar los las views creadas para esto se usa django path.

```python
from django.contrib import admin
from django.urls import path, include


import ms2_doctor


urlpatterns = [
    path('admin/', admin.site.urls),
    path('rest-auth/', include('rest_auth.urls')),
    path('rest-auth/registration/', include('rest_auth.registration.urls')),
    path('ms2-doctor/',include('ms2_doctor.urls')),
]
```

doctor > urls.py > ...

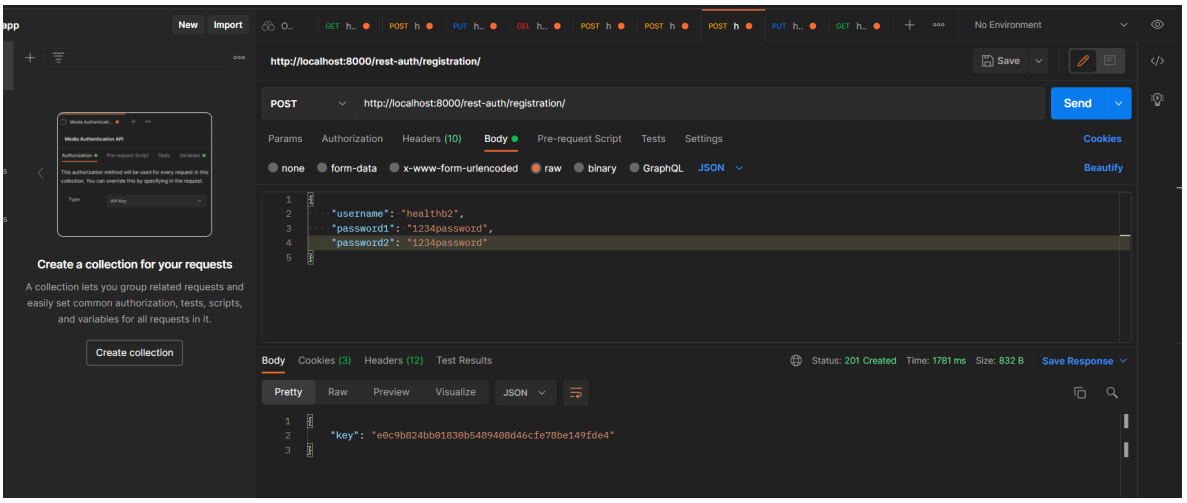
```python
from django.urls import path
from .views import DoctorListCreate,DoctorUpdateDelete

urlpatterns = [
    path('doctor/', DoctorListCreate.as_view()),
    path('doctor/<pk>/', DoctorUpdateDelete.as_view())
]
```

Postman ms2: EndPoints

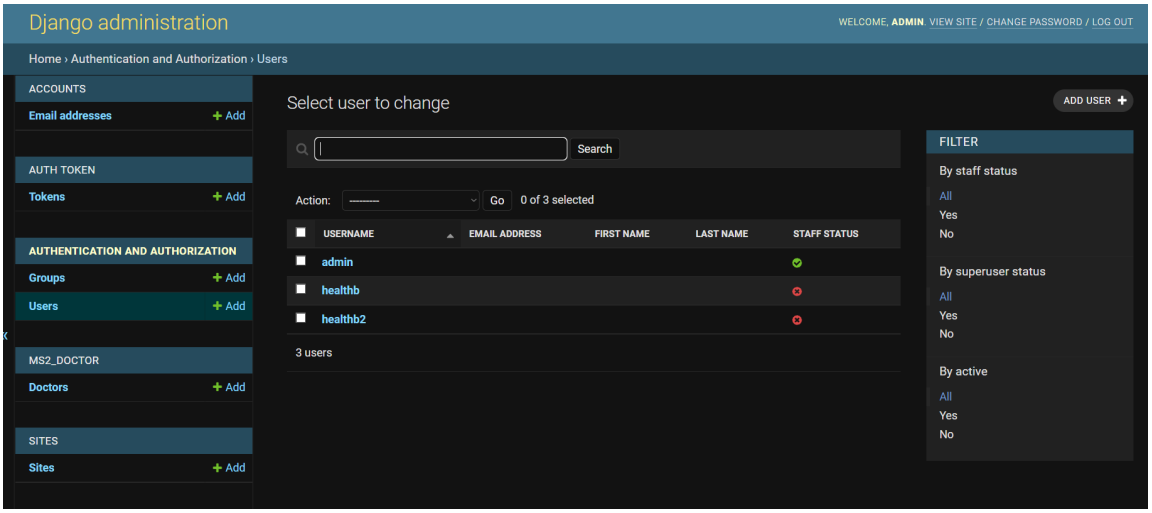
Registro Usuario:

Podemos apreciar el registro de un usuario, al realizarse el registro el programa nos arroja un token de acceso en respuesta.



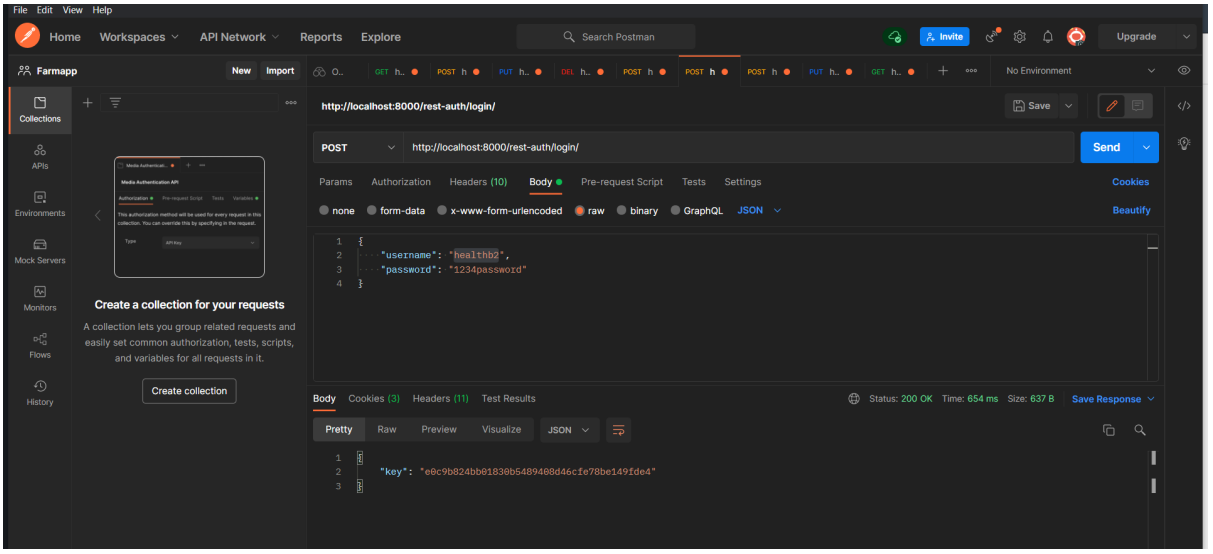
Prueba en el admin de la creación del usuario healthb2:

En el admin podemos observar los usuarios existentes



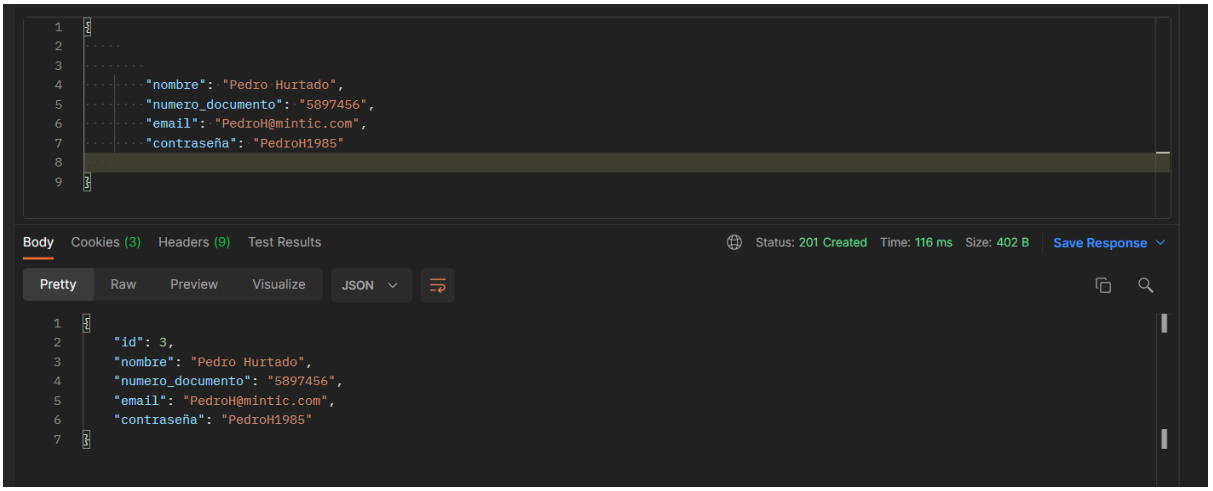
Login Usuario en postman:

Podemos apreciar cómo se envían las credenciales de inicio de sesión mediante un json y el programa en respuesta nos arroja un token de acceso.



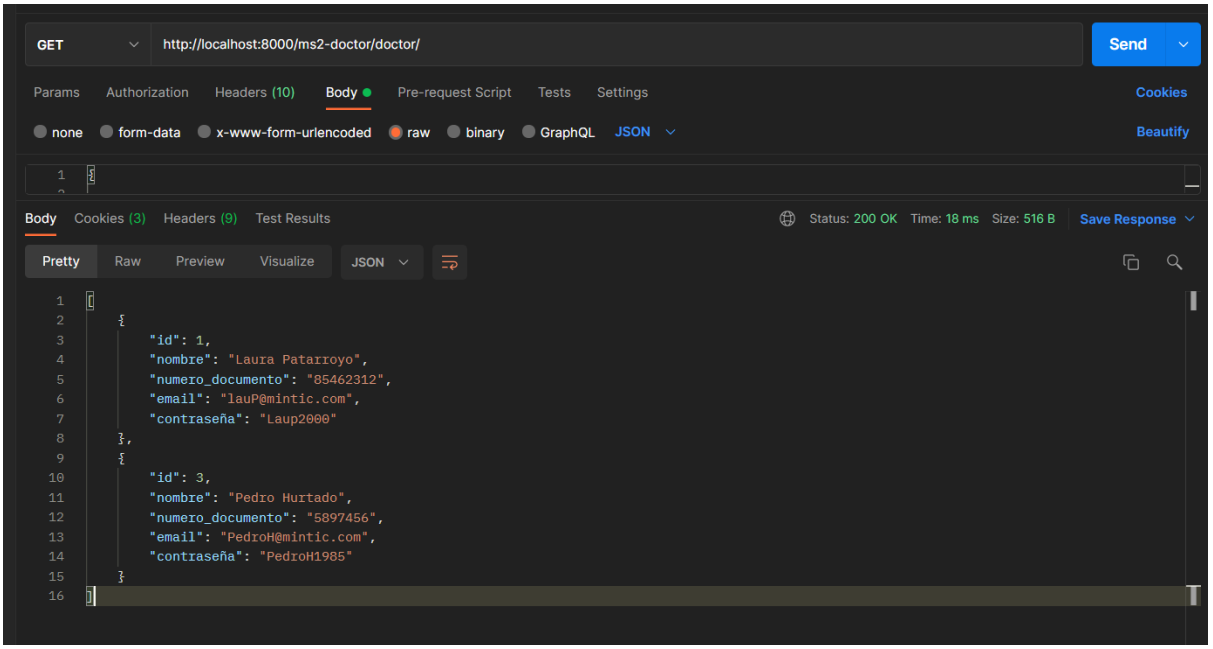
Registro de un Doctor en la base de datos:

Al registrar un doctor se envían los datos correspondientes en el Json y el programa nos arroja estos datos más el ID con el que se va a guardar la información dada.



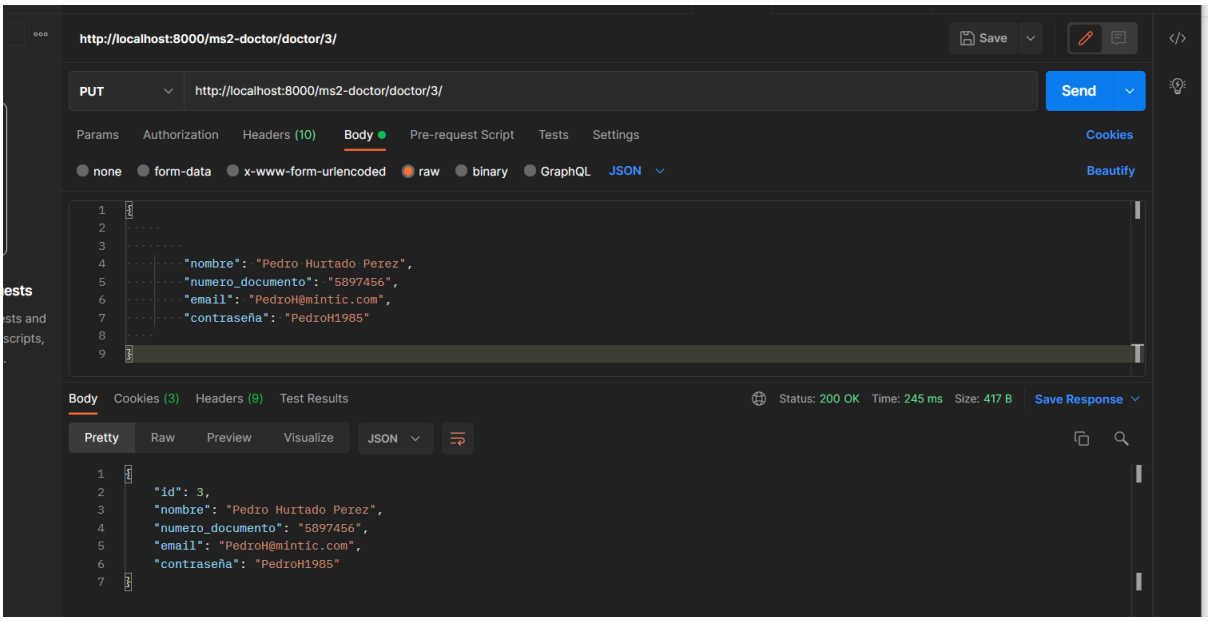
Lista de doctores registrados:

Al hacer un get el programa nos arrojará una lista de los doctores anteriormente registrados.



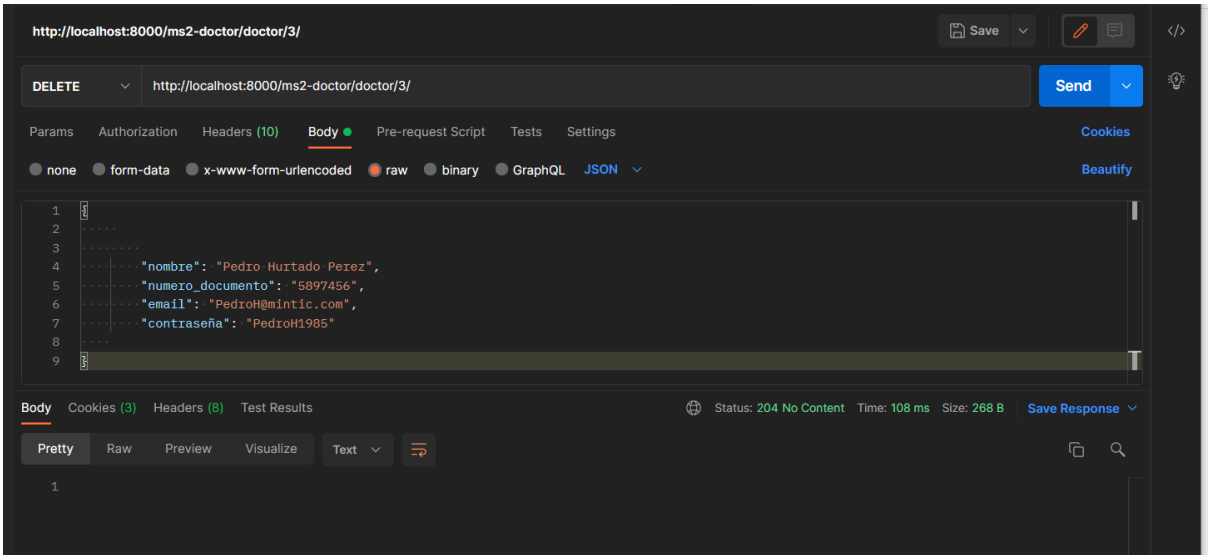
Edición de un registro Doctor:

Para poder editar la información de un doctor tenemos que agregar a la URL el ID del registro, y hacer un PUT con la información a corregir en un JSON, el programa nos arroja la información con las correcciones hechas.



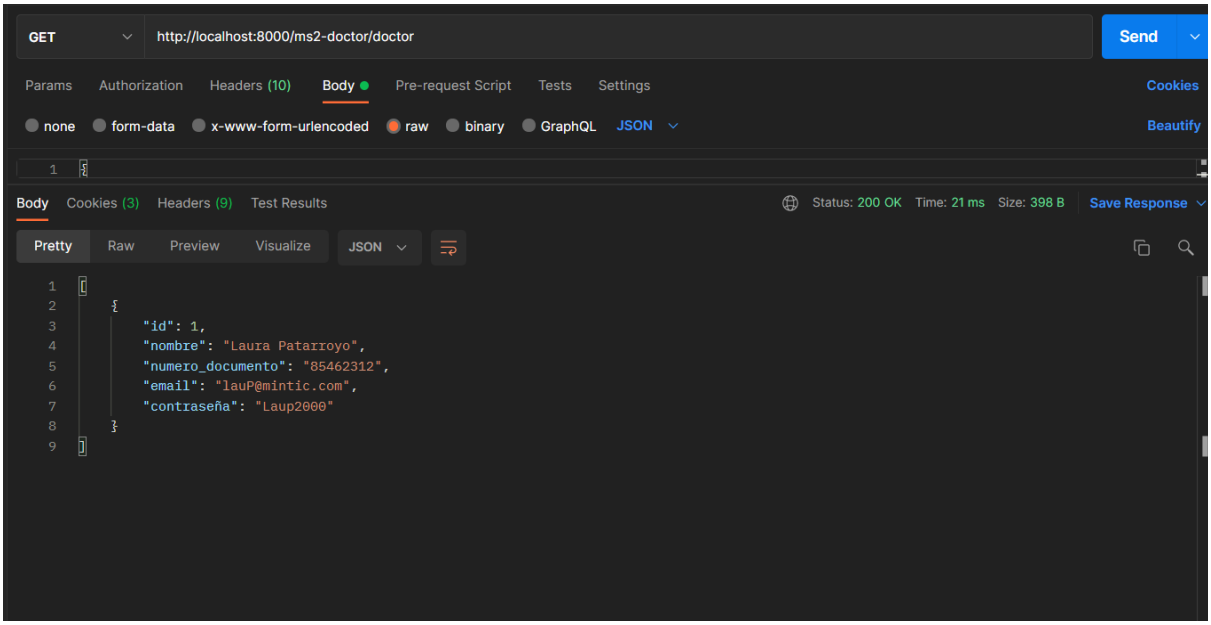
Eliminación de un registro Doctor:

Al hacer un DELETE la información será eliminada para esto tenemos que agregar el ID del registro a eliminar a la URL.



Lista de Doctores actualizada :

Después de eliminar el registro al hacer de nuevo el GET nos arroja la lista de los registros actualizada.



Prueba en el admin de Doctores existentes :

Se puede apreciar en el Django-Admin los doctores existentes con su información.