

Ingeniería en Redes Inteligentes y Ciberseguridad

Unidad III Interfaces de Programación de Aplicaciones en la automatización de redes

Grupo:

GRIC3091

Tema:

Instrumento de evaluación

Alumno:

Reyes Morales Salvador

Docente:

Gabriel Barrón Rodríguez

Fecha:

17/08/2023

Crear una imagen y contenedor de una aplicación Python en Docker

Para crear la imagen con Docker se creó primero el Docker file en el cual se asignaron los siguientes parámetros:

```
# syntax=docker/dockerfile:1

# Comments are provided throughout this file to help you get started.
# If you need more help, visit the Dockerfile reference guide at
# https://docs.docker.com/engine/reference/builder/

ARG PYTHON_VERSION=3.11.4
FROM python:${PYTHON_VERSION}-slim as base

# Prevents Python from writing pyc files.
ENV PYTHONDONTWRITEBYTECODE=1

# Keeps Python from buffering stdout and stderr to avoid situations where
# the application crashes without emitting any logs due to buffering.
ENV PYTHONUNBUFFERED=1

WORKDIR /app

# Create a non-privileged user that the app will run under.
# See https://docs.docker.com/develop/develop-images/dockerfile\_best-practices/#user
ARG UID=10001
RUN adduser \
    --disabled-password \
    --gecos "" \
    --home "/nonexistent" \
    --shell "/sbin/nologin" \
    --no-create-home \
    --uid "${UID}" \
    appuser

# Download dependencies as a separate step to take advantage of Docker's caching.
# Leverage a cache mount to /root/.cache/pip to speed up subsequent builds.
# Leverage a bind mount to requirements.txt to avoid having to copy them into
# into this layer.
RUN --mount=type=cache,target=/root/.cache/pip \
    --mount=type=bind,source=requirements.txt,target=requirements.txt \
    python -m pip install -r requirements.txt
RUN pip install bcrypt
RUN pip install Flask mysql-connector-python

# Switch to the non-privileged user to run the application.
USER appuser

# Copy the source code into the container.
COPY . .

# Expose the port that the application listens on.
EXPOSE 5000

# Run the application.
CMD flask --app instrumento_evaluacion_microservicios run
```

En las imágenes anteriores se configuran los usuarios, además se descargan las dependencias necesarias para ejecutar la aplicación con flask, después se expone el puerto 5000, y con CMD flask --app instrumento_evaluacion_microservicios run se mantiene la aplicación en ejecución

Construir Imagen

Para construir la imagen se utilizó el comando Docker build con la opción --tag para asignarle el nombre a la imagen

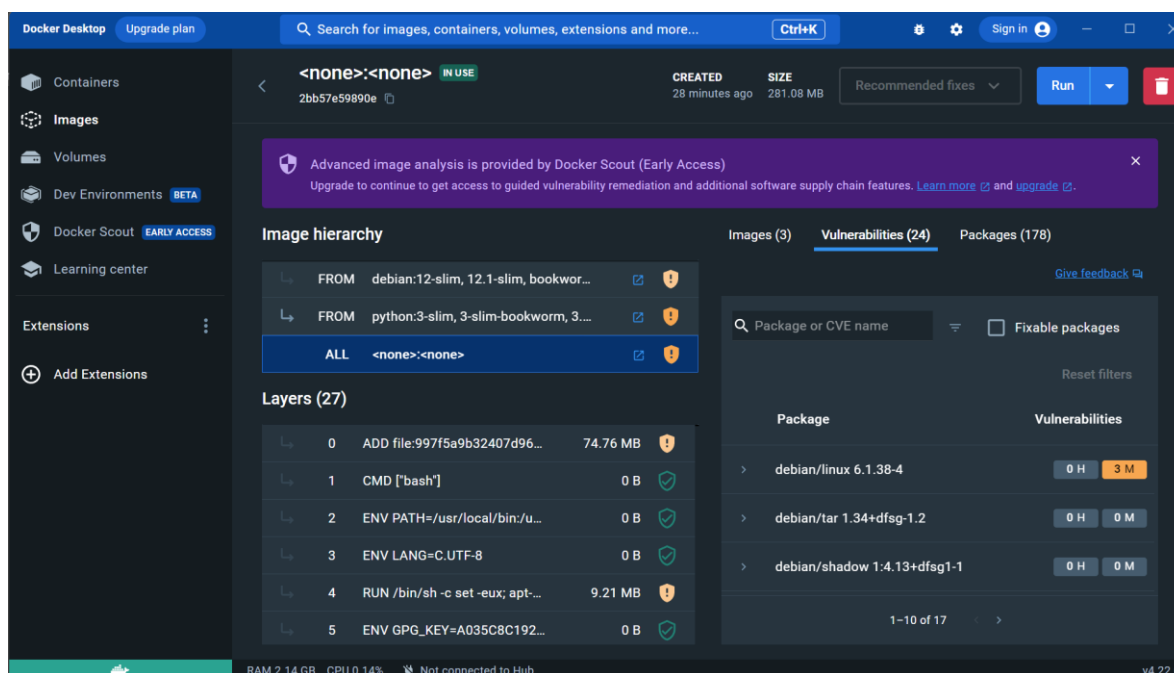
```

PS C:\Users\salsa\Documents\automatiza\flask\python-docker> docker build --tag docker_app .
[+] Building 8.9s (14/14) FINISHED
=> [internal] load .dockerignore                                0.1s
=> => transferring context: 680B                                0.0s
=> [internal] load build definition from Dockerfile              0.1s
=> => transferring dockerfile: 1.68kB                            0.0s
=> resolve image config for docker.io/docker/dockerfile:1      3.3s
=> CACHED docker-image://docker.io/docker/dockerfile:1@sha256:ac85f380a63b13dfcefa89046420e1781752bab202122f8f50032edf31be002 0.0s
=> [internal] load metadata for docker.io/library/python:3.11.4-slim 4.4s
=> [base 1/7] FROM docker.io/library/python:3.11.4-slim@sha256:17d62d681d9ecef20aae6c6605e9cf83b0ba3dc247013e2f43e1b5a045ad49 0.0s
=> [internal] load build context                                0.0s
=> => transferring context: 1.18kB                                0.0s
=> CACHED [base 2/7] WORKDIR /app                               0.0s
=> CACHED [base 3/7] RUN adduser --disabled-password --gecos "" --home "/nonexistent" --shell "/sbin/nologin" 0.0s
=> CACHED [base 4/7] RUN --mount=type=cache,target=/root/.cache/pip --mount=type=bind,source=requirements.txt,target=requirements.txt 0.0s
=> CACHED [base 5/7] RUN pip install bcrypt                      0.0s
=> CACHED [base 6/7] RUN pip install Flask mysql-connector-python 0.0s
=> [base 7/7] COPY . .                                          0.1s
=> exporting to image                                           0.1s
=> => exporting layers                                           0.1s
=> => writing image sha256:4440dd8aa2cf7593889a5bfeb668f515b97fad237a67df17b3495e4c35f334b 0.0s
=> => naming to docker.io/library/docker_app                    0.0s

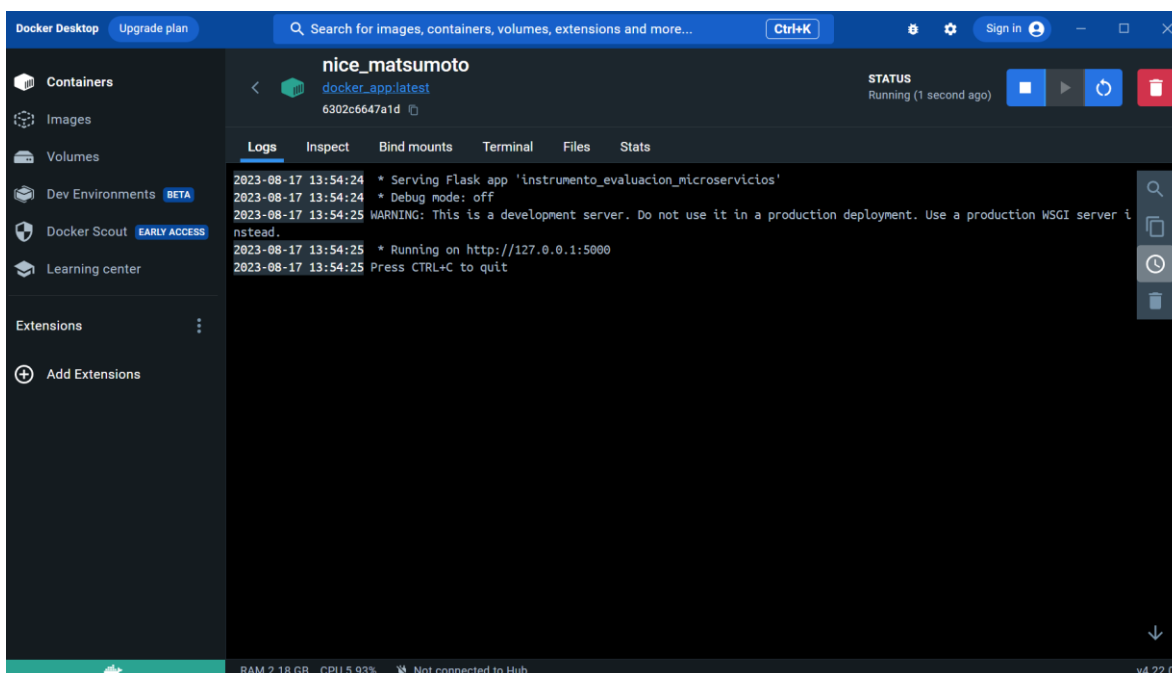
What's Next?
View summary of image vulnerabilities and recommendations → docker scout quickview
PS C:\Users\salsa\Documents\automatiza\flask\python-docker>
  
```

Correr Imagen

Para correr la imagen se ejecutó con Docker Desktop para ver los logs que aparecían en la aplicación



Como se puede ver en la siguiente imagen la ejecución del Docker se hizo de forma exitosa y ya está la aplicación de Python corriendo sobre el puerto 5000



Desarrollar el microservicio

Importación de módulos y configuración de la base de datos:

Se importan los módulos necesarios: Flask para el desarrollo web, jsonify para convertir datos en formato JSON, request para manejar las solicitudes HTTP, bcrypt para el hashing de contraseñas y mysql.connector para interactuar con la base de datos MySQL.

Se establece una conexión con la base de datos MySQL utilizando las credenciales proporcionadas.

```

...
    Autor: Salvador Reyes Morales
    Fecha: 31-07-2023
    Descripcion: Instrumento de Evaluacion
...

# Se importan las librerías necesarias para el código
from flask import Flask, jsonify, request
import bcrypt
import re
import mysql.connector

app = Flask(__name__)
#Se realiza la conexión a la base de datos
students_db = mysql.connector.connect(
    host="192.168.0.115",
    user="sreyes",
    password="Srm75231033",
    database="students_db"
)
  
```

Definición de las rutas y funciones asociadas:

@app.get('/estudiantes'): Esta ruta permite obtener la lista de estudiantes desde la base de datos.

def get_estudiantes(): Esta función ejecuta una consulta SQL para obtener todos los registros de la tabla 'estudiantes' y luego devuelve estos registros en formato JSON.

```
@app.get('/estudiantes')
def get_estudiantes():
    # Verificar si la conexión está abierta
    if not students_db.is_connected():
        students_db.reconnect()
    # Envía la petición para mostrar todo los estudiantes
    cursor.execute("SELECT * FROM estudiantes")
    estudiantes = cursor.fetchall()
    # Regresa y muestra todos los estudiantes
    return jsonify(estudiantes), 200
```

Validación de contraseñas:

def validar_contraseña(contraseña): Esta función toma una contraseña como entrada y verifica si cumple con ciertos criterios: longitud mínima de 8 caracteres, al menos una letra mayúscula, al menos una letra minúscula y al menos un carácter especial.

```
#Se utiliza una funcion para validar la contraseña
def validar_contraseña(contraseña):
    # Debe tener al menos 8 caracteres
    if len(contraseña) < 8:
        return False

    # Debe contener al menos una letra mayúscula
    if not any(c.isupper() for c in contraseña):
        return False

    # Debe contener al menos una letra minúscula
    if not any(c.islower() for c in contraseña):
        return False

    # Debe contener al menos un carácter especial
    if not re.search(r"[!@#$%^&*(),.?\"':{}|<>]", contraseña):
        return False

    return True
```

Agregar estudiantes:

@app.post('/agregar'): Esta ruta permite agregar un nuevo estudiante a la base de datos.

def add_estudiantes(): Esta función recibe los datos del estudiante en formato JSON desde la solicitud, verifica si ya existe un estudiante con el mismo número de control o nombre de usuario, valida la contraseña llamando a la función validar_contraseña, realiza el hash de la contraseña utilizando bcrypt y luego inserta el registro en la base de datos.

```
# Se utiliza el metodo post para agregar estudiantes
@app.post('/agregar')
def add_estudiantes():
    # Verificar si la conexión está abierta
    if not students_db.is_connected():
        students_db.reconnect()
    # Obtiene Los datos de postman
    datos = request.get_json()
    # Verifica si los datos existen
    cursor.execute("SELECT * FROM estudiantes WHERE numero_control = %s OR username = %s", (datos['numero_control'], datos['username']))
    existing_student = cursor.fetchone()
    # Si los datos existen regresa; Este usuario ya se encuentra registrado
    if existing_student:
        return {'error': 'Este usuario ya se encuentra registrado'}, 201
    # Si la contraseña no cumple los requisitos envía un mensaje
    if not validar_contraseña(datos['contraseña']):
        return {'error': 'La contraseña no cumple con los requisitos, deben ser mayúsculas, minúsculas, mínimo 8 caracteres, caracteres especiales'}, 400
    # Encripta la contraseña
    hashed_password = bcrypt.hashpw(datos['contraseña'].encode('utf-8'), bcrypt.gensalt())
    datos['contraseña'] = hashed_password.decode('utf-8')
    # Inserta Los valores a la base de datos
    insert_query = "INSERT INTO estudiantes (numero_control, username, contraseña, nombre) VALUES (%s, %s, %s, %s)"
    cursor.execute(insert_query, (datos['numero_control'], datos['username'], datos['contraseña'], datos['nombre']))
    students_db.commit()
    # Regresa un mensaje exitoso
    return {'success': 'Registro agregado con éxito'}, 201
# Se utiliza el metodo post para Logear estudiantes
```

Iniciar sesión:

@app.post('/login'): Esta ruta permite que un estudiante inicie sesión.

def iniciar_sesion(): Esta función verifica si el usuario proporcionado existe en la base de datos. Si existe, verifica la contraseña proporcionada con la contraseña almacenada en la base de datos utilizando bcrypt.

```
@app.post('/login')
def iniciar_sesion():
    # Verificar si la conexión está abierta
    if not students_db.is_connected():
        students_db.reconnect()
    # Obtiene Los datos de postman
    datos = request.get_json()
    # Comprueba que los datos son iguales
    cursor.execute("SELECT * FROM estudiantes WHERE username = %s", (datos['username'],))
    estudiante = cursor.fetchone()

    if estudiante:
        # Desencripta las contraseñas
        hashed_password = estudiante['contraseña'].encode('utf-8')
        provided_password = datos['contraseña'].encode('utf-8')
        # Compara las contraseñas y si coinciden envía un mensaje de éxito
        if bcrypt.checkpw(provided_password, hashed_password):
            return {'success': 'Inicio de sesión correcto'}, 200
        # En caso contrario envía un mensaje de error
        else:
            return {'error': 'Usuario y contraseña incorrectos'}, 401
    # Si no se encuentra ningún usuario envía un mensaje de error
    else:
        return {'error': 'Usuario no encontrado'}, 404
```

Operaciones con maestros:

@app.get('/maestro'): Esta ruta obtiene la lista de maestros desde la base de datos.

def get_maestro(): Esta función ejecuta una consulta SQL para obtener todos los registros de la tabla 'maestro' y devuelve los registros en formato JSON.

```

'''----- Microservicio Maestro -----'''
'''-----'''

# Con La funcion get se obtienen Los maestros
@app.get('/maestro')
def get_maestro():
    # Verificar si La conexión está abierta
    if not students_db.is_connected():
        students_db.reconnect()
    # Realiza La petición para mostrar todos Los maestros
    cursor.execute("SELECT * FROM maestro")
    maestro = cursor.fetchall()
    # Regresa en json a todos Los maestros
    return jsonify(maestro), 200
    
```

@app.post('/maestro/agregar'): Esta ruta permite agregar un nuevo maestro a la base de datos.

def add_maestro(): Similar a la función add_estudiantes, esta función agrega un nuevo maestro a la base de datos.

```

@app.post('/maestro/agregar')
def add_maestro():
    # Verificar si La conexión está abierta
    if not students_db.is_connected():
        students_db.reconnect()
    # Obtiene Los datos de postman
    datos = request.get_json()
    # Try except sirve para atrapar errores y evitar que el código truee
    try:
        # Compara si existe cve_maestro y nombre en La base de datos
        cursor.execute("SELECT * FROM maestro WHERE cve_maestro = %s OR nombre = %s", (datos['cve_maestro'], datos['nombre']))
        existing_maestro = cursor.fetchone()
        # Si el maestro existe envia un mensaje de error
        if existing_maestro:
            return {'error': 'Este maestro ya se encuentra registrado'}, 201
        # Inserta Los valores a La base de datos
        insert_query = "INSERT INTO maestro (cve_maestro, nombre, apellido, correo, edificio, telefono, cubiculo, direccion) VALUES (%s, %s, %s, %s, %s, %s, %s, %s)"
        cursor.execute(insert_query,
            (datos['cve_maestro'], datos['nombre'], datos['apellido'], datos['correo'], datos['edificio'], datos['telefono'], datos['cubiculo'], datos['direccion']))
        students_db.commit()
        # Regresa un mensaje de éxito
        return {'success': 'Maestro agregado con éxito'}, 201
    except:
        # Envía un mensaje de error
        return {'error': 'Por favor verifica los datos'}
    
```

Operaciones con materias:

@app.route('/materias', methods=['GET']): Esta ruta permite obtener la lista de materias desde la base de datos.

def get_materias(): Similar a las funciones get_estudiantes y get_maestro, esta función obtiene los registros de la tabla 'materias'.


```

'''-----Microservicio Materias-----'''

# Se utiliza La funcion get para obtener Las materias
@app.get('/materias')
def get_materias():
    # Verificar si La conexión está abierta
    if not students_db.is_connected():
        students_db.reconnect()
    # Realiza La petición para mostrar todas Las materias
    cursor.execute("SELECT * FROM materias")
    materias = cursor.fetchall()
    # Regresa Las materias en json
    return jsonify(materias), 200
  
```

@app.post('/materias/agregar'): Esta ruta permite agregar una nueva materia a la base de datos.

def add_materias(): Similar a las funciones add_estudiantes y add_maestro, esta función agrega una nueva materia a la base de datos.

```

@app.post('/materias/agregar')
def add_materias():
    # Verificar si La conexión está abierta
    if not students_db.is_connected():
        students_db.reconnect()
    # Recibe Los datos del postman
    datos = request.get_json()
    # Try except sirve para atrapar errores y evitar que el código truee
    try:
        # Verifica se cve_mat y nombre existen en La base de datos
        cursor.execute("SELECT * FROM materias WHERE cve_mat = %s OR nombre = %s", (datos['cve_mat'], datos['nombre']))
        existing_materias = cursor.fetchone()
        # Si existen envía un mensaje de error
        if existing_materias:
            return {'error': 'Esta materia ya se encuentra registrado'}, 201
        # Si no existen envía Los valores a La base de datos
        insert_query = "INSERT INTO materias (cve_mat, nombre, horas_practicas, horas_teoricas, carrera, unidades, cve_maestro, numero_control) VALUES (%s, %s, %s, %s, %s, %s, %s, %s)"
        cursor.execute(insert_query, (datos['cve_mat'], datos['nombre'], datos['horas_practicas'], datos['horas_teoricas'], datos['carrera'], datos['unidades'], datos['cve_maest'], datos['numero_control']))
        students_db.commit()
        # Regresa un mensaje de éxito
        return {'success': 'Materias agregada con éxito'}, 201
    except:
        # Si encuentra un error envía un mensaje
        return {'error': 'Por favor verifica los datos'}
  
```

En general, este código define una API web que proporciona endpoints para realizar operaciones (Crear, Leer, Obtener) en una base de datos MySQL relacionada con estudiantes, maestros y materias. También incluye funciones de validación de contraseñas y uso de hashing para mantener la seguridad de las contraseñas almacenadas en la base de datos.

Base de datos

Para crear la base de datos funcional para los microservicio, primero se creo y se utilizo la base de datos, se eliminan tablas en caso de que existan para evitar errores de ejecución.

```

create database if not exists students_db;
use students_db;
drop TABLE if exists materias;
drop TABLE if exists estudiantes;
drop table if exists maestro;
  
```


Se crea la tabla de estudiantes con sus respectivos valores

```
CREATE TABLE estudiantes (  
    numero_control INT PRIMARY KEY,  
    nombre VARCHAR (1000) NULL,  
    username VARCHAR (1000) NULL,  
    contraseña VARCHAR(1000) NULL  
);
```

Después se crea la tabla maestro con sus respectivos valores

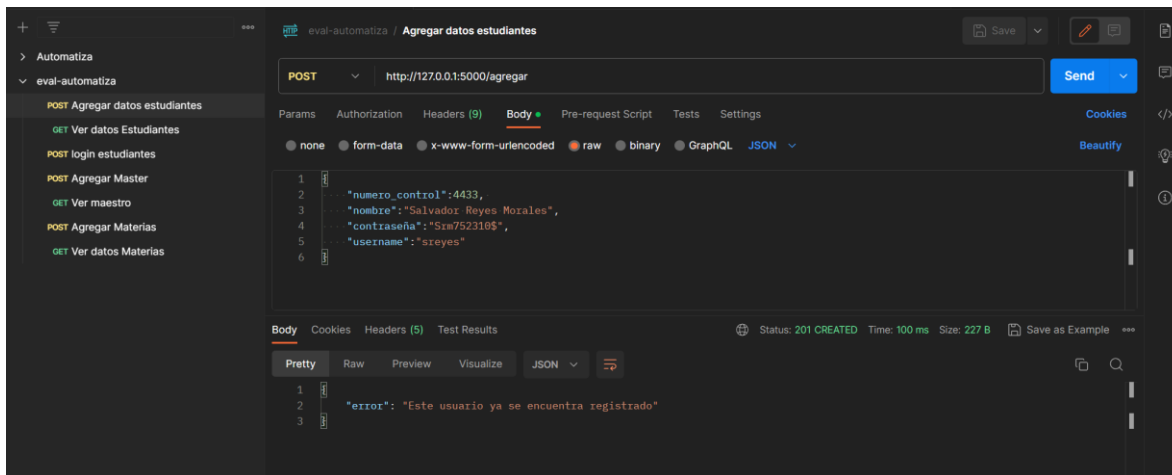
```
CREATE TABLE maestro (  
    cve_maestro INT PRIMARY KEY,  
    nombre VARCHAR (1000) NULL,  
    apellido VARCHAR (1000) NULL,  
    correo varchar (100) null,  
    edificio varchar (100) null,  
    telefono varchar (100) null,  
    cubiculo INT null,  
    direccion varchar (100) null  
);
```

Por ultimo se crea la tabla materias asignando llaves foráneas de las tablas creadas anteriormente

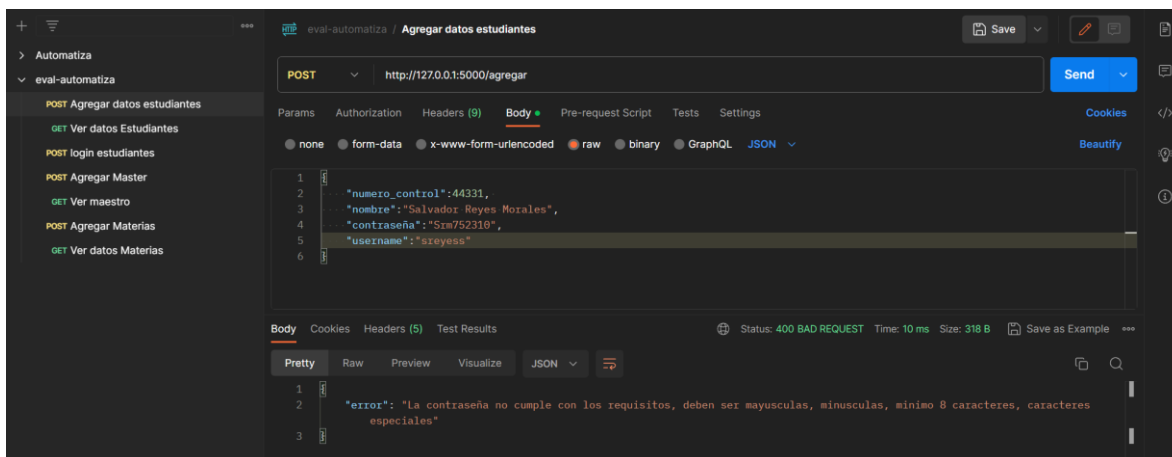
```
CREATE TABLE materias (  
    cve_mat INT PRIMARY KEY,  
    nombre varchar (50) not null,  
    horas_practicas varchar (100) null,  
    horas_teoricas varchar (100) null,  
    carrera varchar (100) null,  
    unidades INT null,  
    cve_maestro int not NULL,  
    numero_control int not null,  
    foreign key (numero_control)  
    references estudiantes (numero_control),  
    foreign key (cve_maestro)  
    references maestro (cve_maestro)  
);
```

Pruebas de Postman

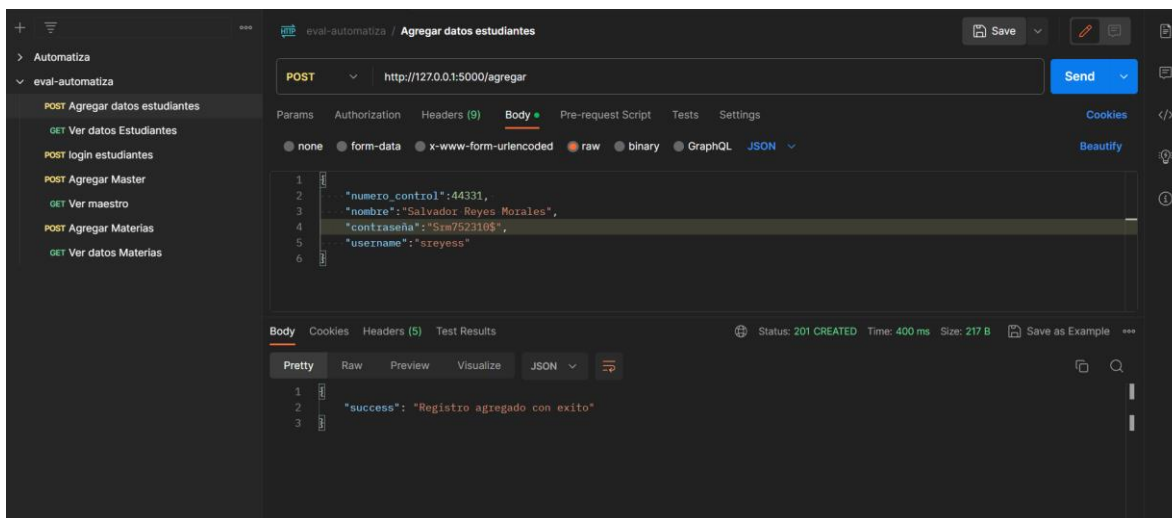
Pantalla "Este usuario se encuentra registrado" de agregar estudiantes



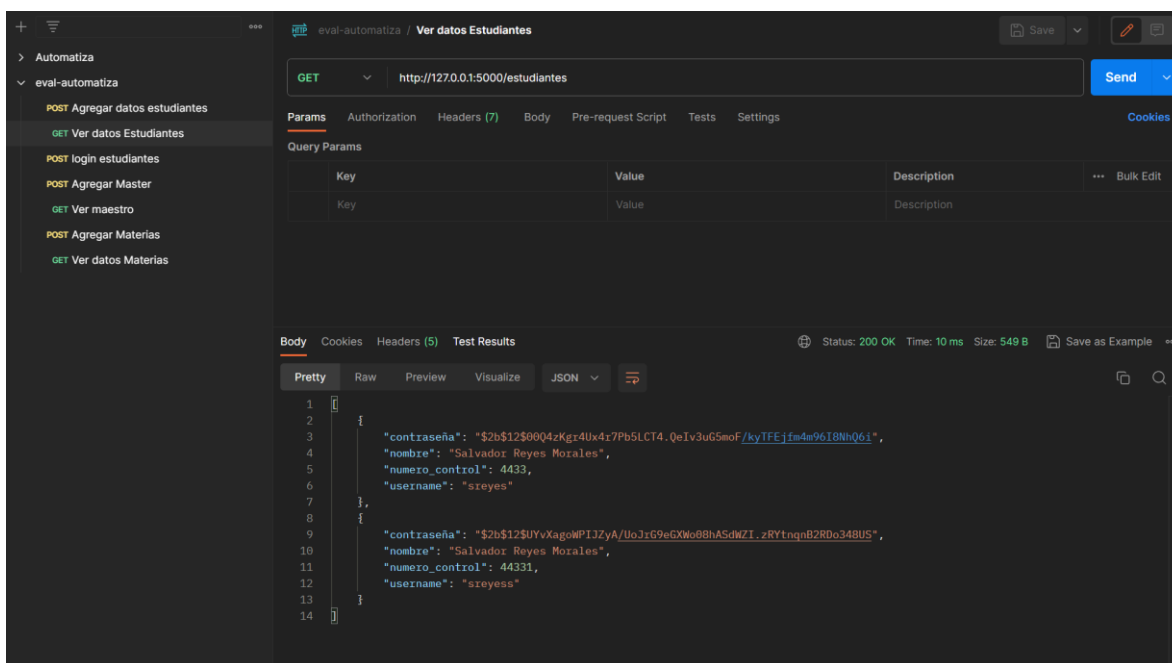
Pantalla "La contraseña no cumple con los requisitos, deben ser mayusculas, minusculas, minimo 8 caracteres, caracteres especiales" de agregar estudiantes



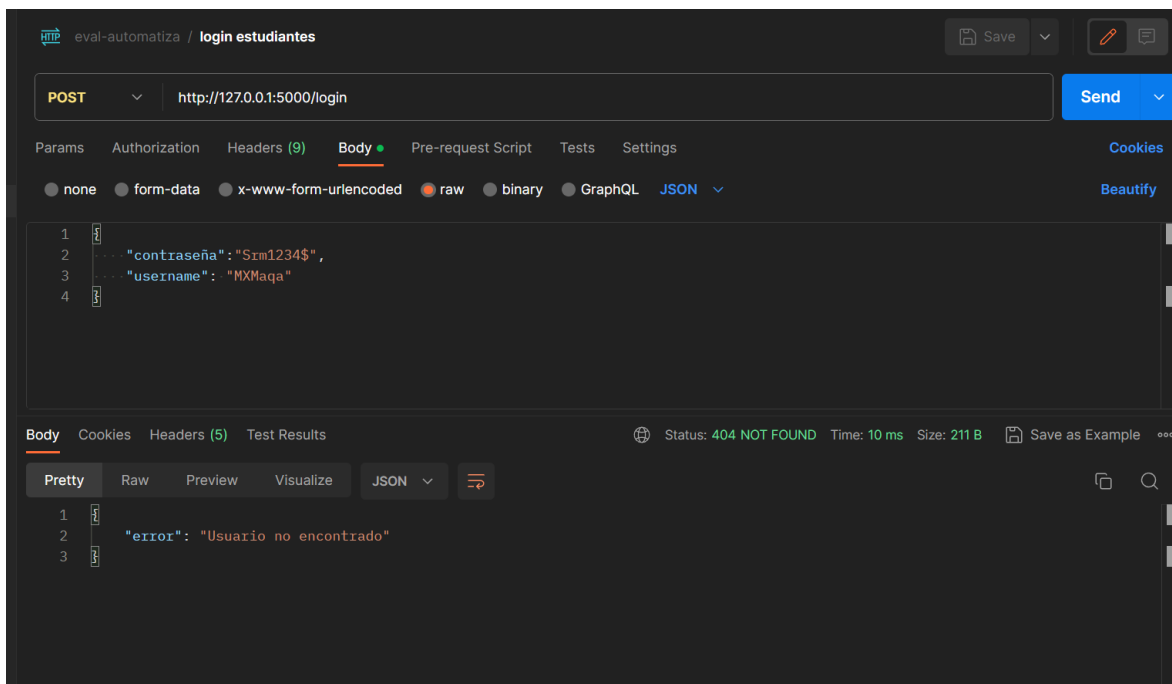
Pantalla "Registro agregado con exito", después se enviar el mensaje se envían los datos a la base de datos



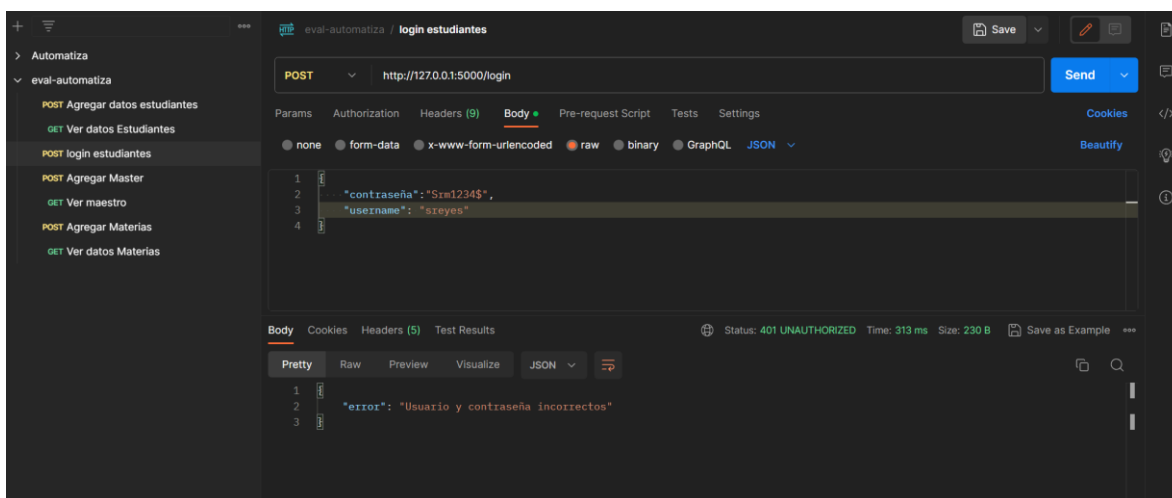
Despliegue de los estudiantes de la base de datos con el método GET



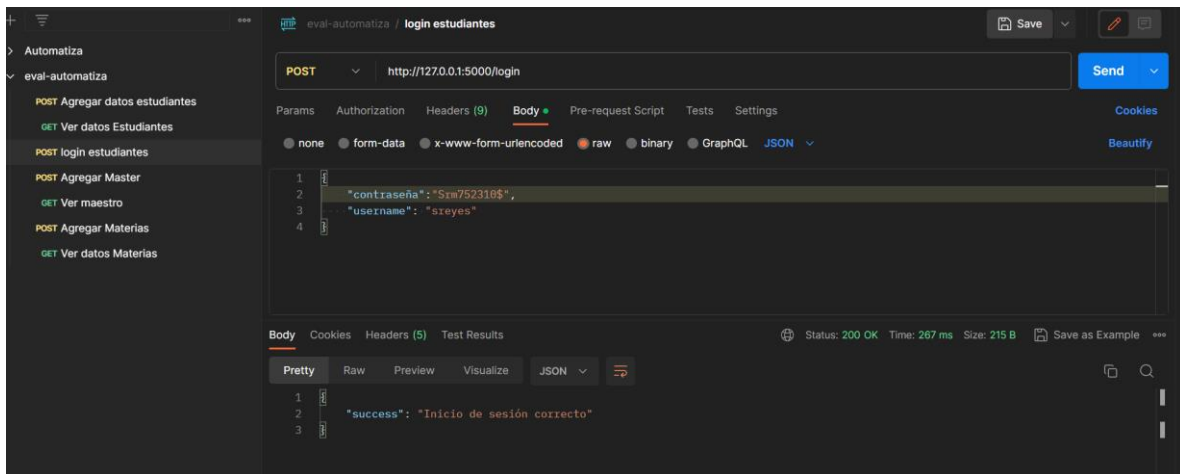
Pantalla de usuario no encontrado, ya que no encontró el usuario en la base de datos.



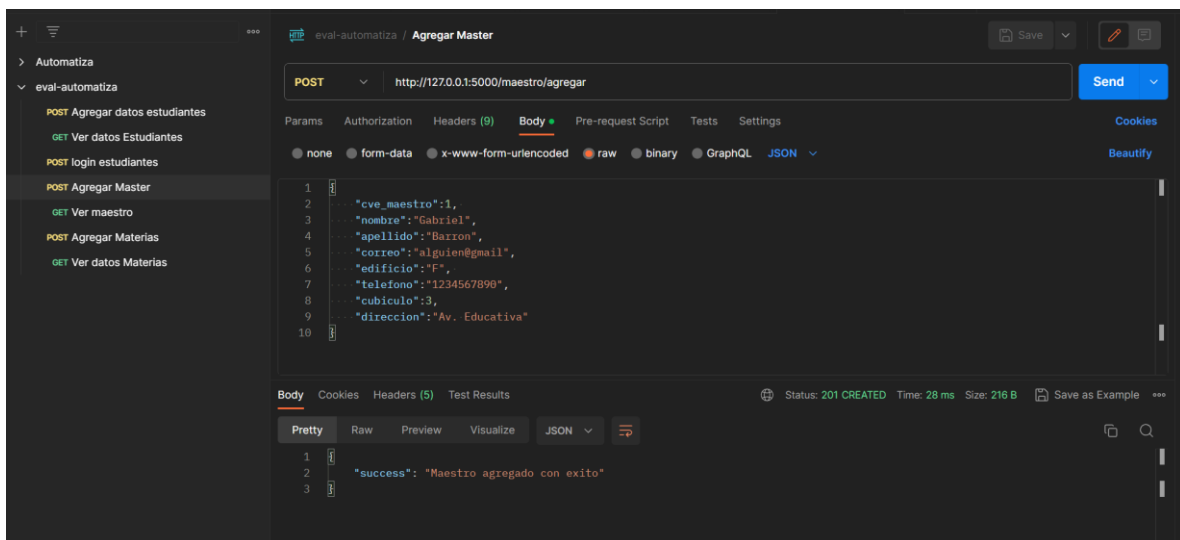
Pantalla de usuario y contraseña incorrectos



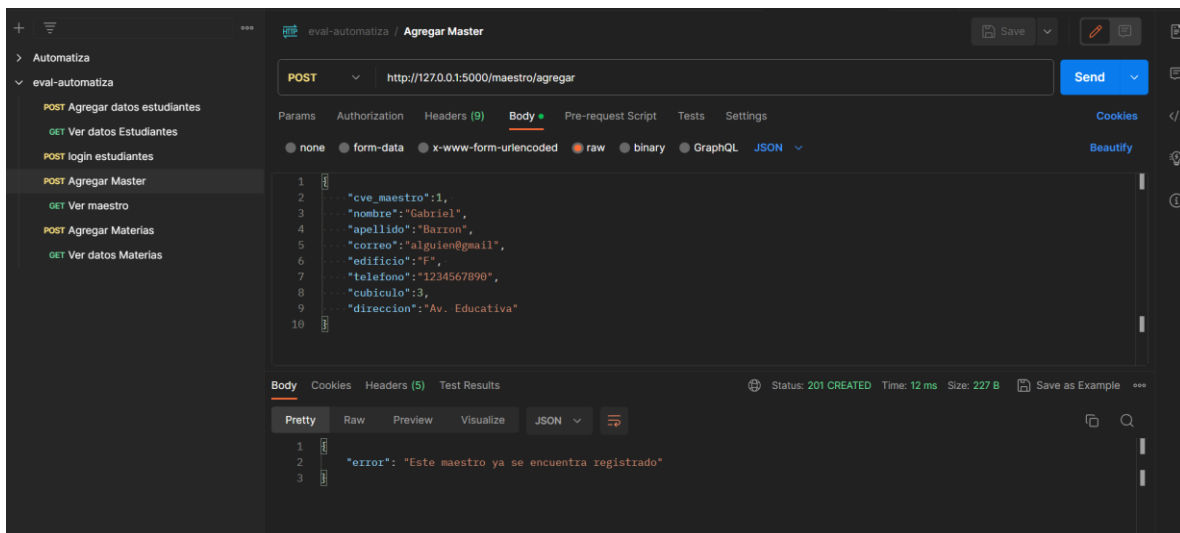
Pantalla de inicio de sesión exitoso



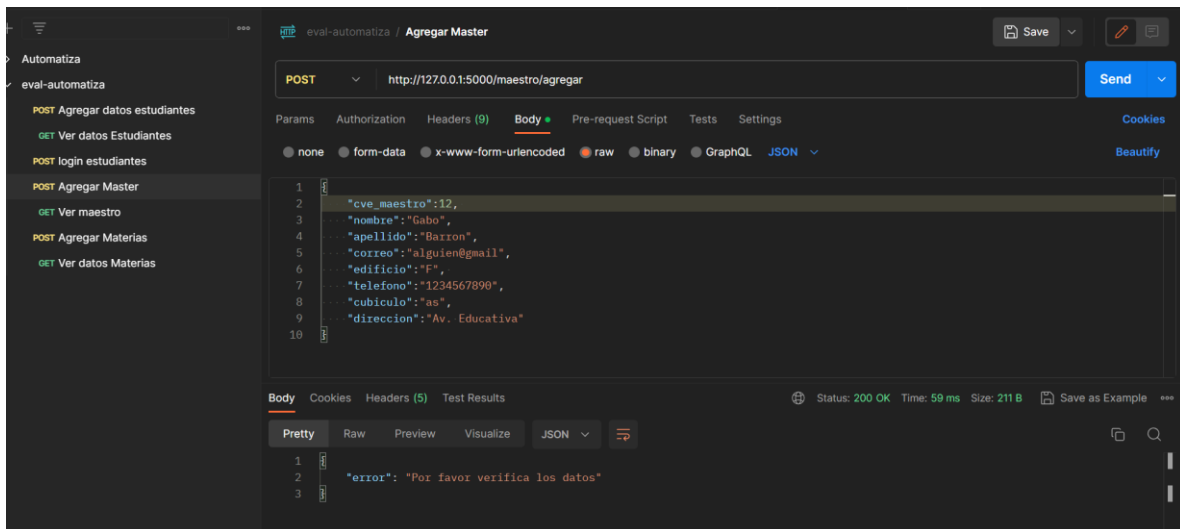
Pantalla de “maestro agregado con éxito” de agregar maestro.



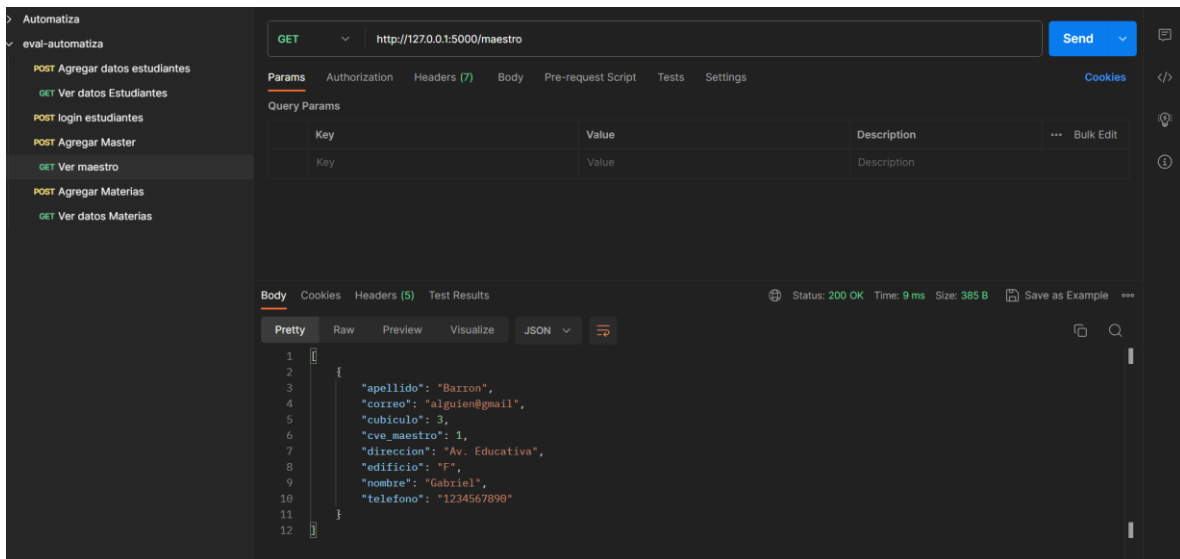
Pantalla de “Este maestro ya se encuentra registrado” de agregar maestro.



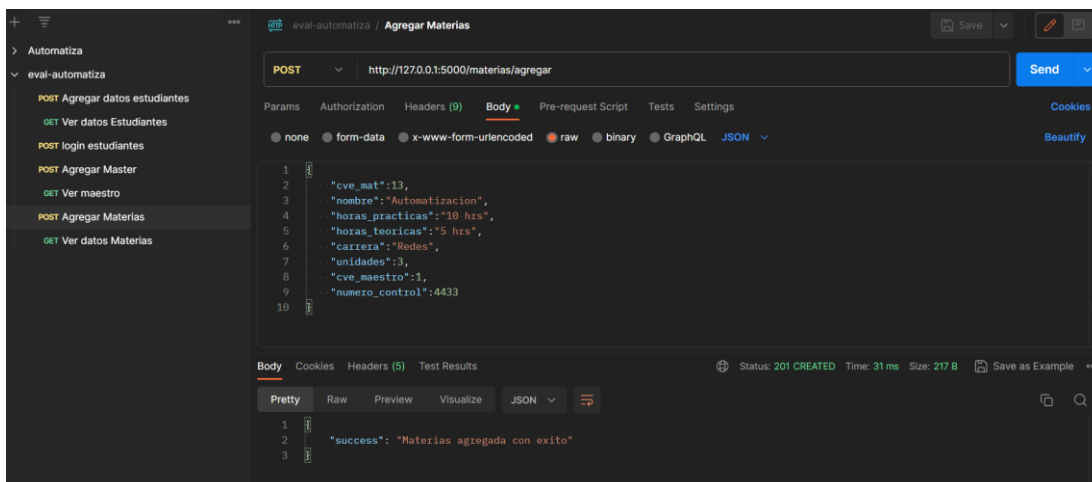
Pantalla de verificación de datos para evitar que la aplicación trueque por ingresar datos mal de la base de datos, en este caso cubículo esta registrado como int pero se puso como cadena por lo que envía un mensaje de “Por favor verifica los datos”



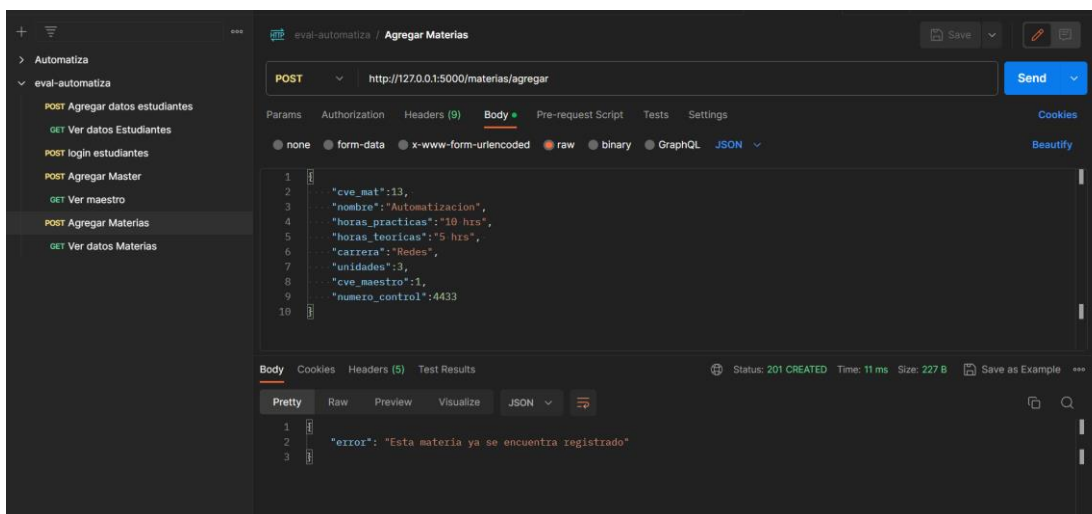
Pantalla para listar a los maestros registrados en la base de datos



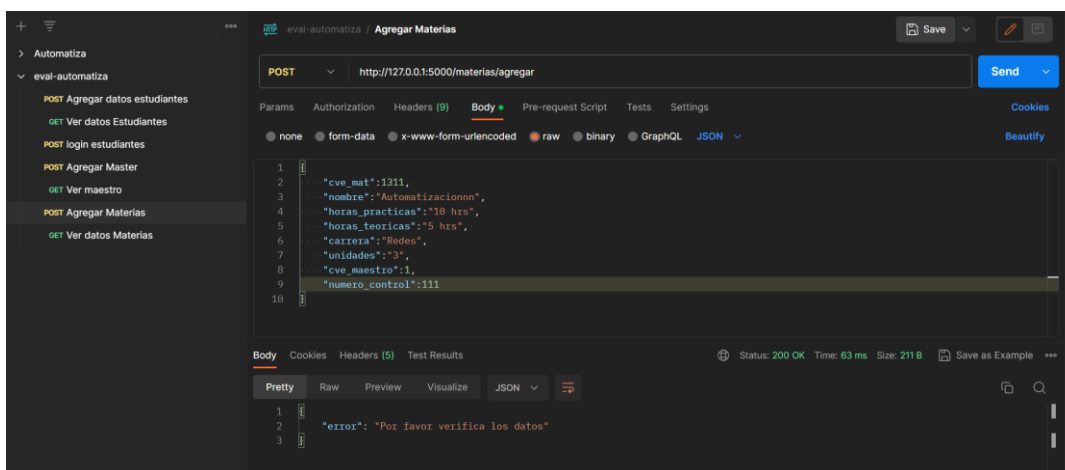
Pantalla de agregar maestrías de forma exitosa



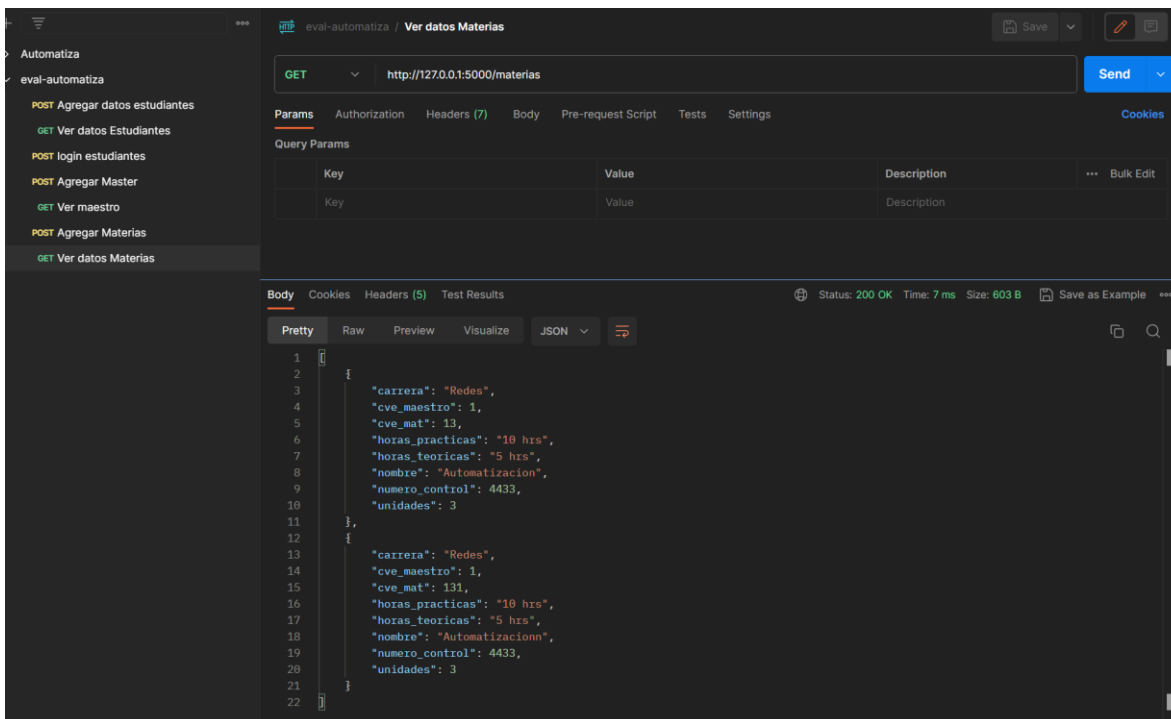
Pantalla de esta materia ya se encuentra registrada



Pantalla de verificación de datos para evitar que la aplicación truene por ingresar datos mal de la base de datos, en este caso numero_control no está registrado en estudiantes por lo que mande ese mensaje



Pantalla que muestra todas las materias registradas



The screenshot shows a REST client interface with the following components:

- Left Sidebar:** A list of API endpoints under the "eval-automatiza" collection:
 - POST Agregar datos estudiantes
 - GET Ver datos Estudiantes
 - POST login estudiantes
 - POST Agregar Master
 - GET Ver maestro
 - POST Agregar Materias
 - GET Ver datos Materias (selected)
- Request Editor:**
 - Method: GET
 - URL: http://127.0.0.1:5000/materias
 - Buttons: Save, Send
 - Tabs: Params, Authorization, Headers (7), Body, Pre-request Script, Tests, Settings
 - Query Params table:

Key	Value	Description
Key	Value	Description
- Response Viewer:**
 - Status: 200 OK, Time: 7 ms, Size: 603 B
 - Buttons: Save as Example
 - Tabs: Body, Cookies, Headers (5), Test Results
 - Body tab selected, showing JSON in "Pretty" format:


```

1  {
2    {
3      "carrera": "Redes",
4      "cve_maestro": 1,
5      "cve_mat": 13,
6      "horas_practicas": "10 hrs",
7      "horas_teoricas": "5 hrs",
8      "nombre": "Automatizacion",
9      "numero_control": 4433,
10     "unidades": 3
11   },
12   {
13     "carrera": "Redes",
14     "cve_maestro": 1,
15     "cve_mat": 131,
16     "horas_practicas": "10 hrs",
17     "horas_teoricas": "5 hrs",
18     "nombre": "Automatizacionn",
19     "numero_control": 4433,
20     "unidades": 3
21   }
22 }
```