

UNIVERSIDAD NACIONAL DE LA PLATA
MAESTRÍA EN INGENIERÍA DE SOFTWARE

Trabajo Práctico Final

Asignatura: Tópicos de Ingeniería de Software II

Año: 2023

Título: Trabajo Integrador: Servicio Web (API) –
Predicción de Riesgo Cardíaco

Autor: Karina Policano

Docentes:

Prof. Andrés Díaz Pace

Prof. Claudia Pons

Prof. Gabriela Pérez

Prof. Matías Urbietta

Contenido

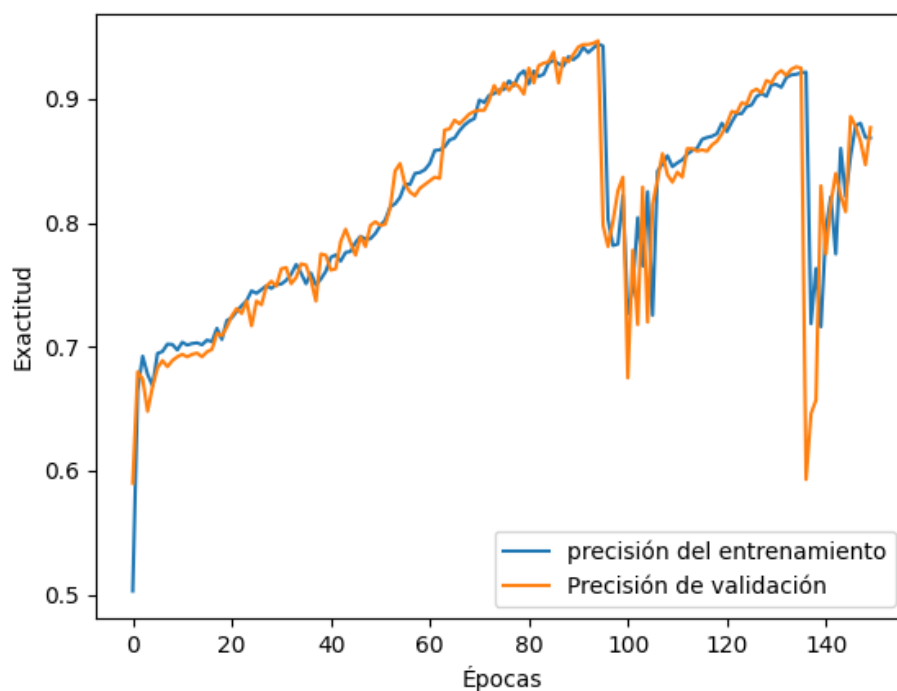
Requisitos Obligatorios para Ejecutar el Proyecto	3
Pasos del Proyecto.....	3
Servicios ofrecidos:.....	4
Comunicación entre servicios:	5
Decisiones de Diseño	5
Uso de Redis:.....	5
Funciones Implementadas	5
Servicio <i>predictor</i>	5
validartiempo:	5
predecir_conTiempo:	5
validarParametros:	5
Programas Python del Proyecto	5
Para la ejecución:	5
Caso de prueba.....	10
Servicio altaUsuario.....	11
Incorporación de usuarios.....	11
Errores probables	13
Servicio predictor	16
Servicio guardarBitacora	16
Manejo de consultas con Redis	18
Errores posibles	20
Servicio autorizacion	21
Errores posibles	22

Requisitos Obligatorios para Ejecutar el Proyecto

1. Acceso al Repositorio: <https://github.com/polka2005/TP2RiesgoCardiaco>
2. Python:
 - Tener Python instalado en tu sistema.
 - Trabajo realizado con versión 3.10
3. Visual Studio Code:
 - Necesita tener instalado Visual Studio Code, un entorno de desarrollo de código abierto.
4. Redis:
 - La aplicación depende de Redis como sistema de almacenamiento en caché.
 - Se adjunta archivos en carpeta Redis
5. Postman:
 - Se recomienda tener Postman para probar y documentar las API.

Pasos del Proyecto

1. Mediante el archivo `C:\RiesgoCardiaco\RiesgoCardiaco_ML.py`, se realizó el entrenamiento utilizando 3 capas, de 50, 25 y 35, 1 neurona y 150 épocas. Con lo que se obtuvo el modelo extensión keras y su respectivo gráfico de precisión.



```

21
22 model.add(Dense(50, input_shape=(6,), activation='relu', kernel_initializer='uniform'))
23 model.add(Dense(25, activation='relu', kernel_initializer='random_normal'))
24 model.add(Dense(35, activation='relu', kernel_initializer='random_normal'))
25 model.add(Dense(1, activation='relu'))
26
27 model.save('modeloRiesgoC.keras')
28
29 model.compile(optimizer='adam',
30               loss='binary_crossentropy',
31               metrics=['accuracy'])
32
33
34
35 history = model.fit(X_train, y_train, validation_data=(X_test, y_test), verbose=2, batch_size = 1000, epochs=150)
36
37 plt.plot(history.history['accuracy'], label='precisión del entrenamiento')
38 plt.plot(history.history['val_accuracy'], label='Precisión de validación')
39 plt.xlabel('Épocas')
40 plt.ylabel('Exactitud')
41 plt.legend()

```

OUTPUT TERMINAL PORTS DEBUG CONSOLE

```

Epoch 138/150
4/4 - 0s - loss: 1.7154 - accuracy: 0.7188 - val_loss: 2.6638 - val_accuracy: 0.6460 - 278ms/epoch - 70ms/step
Epoch 139/150
4/4 - 0s - loss: 1.0423 - accuracy: 0.7632 - val_loss: 0.8897 - val_accuracy: 0.6570 - 281ms/epoch - 70ms/step
Epoch 140/150
4/4 - 0s - loss: 0.6459 - accuracy: 0.7163 - val_loss: 0.3513 - val_accuracy: 0.8300 - 233ms/epoch - 58ms/step
Epoch 141/150
4/4 - 0s - loss: 0.5361 - accuracy: 0.7947 - val_loss: 0.5885 - val_accuracy: 0.7750 - 163ms/epoch - 41ms/step
Epoch 142/150
4/4 - 0s - loss: 0.4183 - accuracy: 0.8210 - val_loss: 0.3993 - val_accuracy: 0.8160 - 190ms/epoch - 48ms/step
Epoch 143/150
4/4 - 0s - loss: 0.4319 - accuracy: 0.7750 - val_loss: 0.3764 - val_accuracy: 0.8400 - 177ms/epoch - 44ms/step
Epoch 144/150
4/4 - 0s - loss: 0.3523 - accuracy: 0.8602 - val_loss: 0.3390 - val_accuracy: 0.8220 - 317ms/epoch - 79ms/step
Epoch 145/150
4/4 - 1s - loss: 0.3648 - accuracy: 0.8215 - val_loss: 0.3491 - val_accuracy: 0.8090 - 603ms/epoch - 151ms/step
Epoch 146/150
4/4 - 0s - loss: 0.3306 - accuracy: 0.8543 - val_loss: 0.3212 - val_accuracy: 0.8860 - 172ms/epoch - 43ms/step
Epoch 147/150
4/4 - 0s - loss: 0.3287 - accuracy: 0.8785 - val_loss: 0.3289 - val_accuracy: 0.8780 - 209ms/epoch - 52ms/step
Epoch 148/150
4/4 - 0s - loss: 0.3161 - accuracy: 0.8805 - val_loss: 0.3076 - val_accuracy: 0.8660 - 186ms/epoch - 46ms/step
Epoch 149/150
4/4 - 0s - loss: 0.3066 - accuracy: 0.8690 - val_loss: 0.3115 - val_accuracy: 0.8470 - 381ms/epoch - 95ms/step
Epoch 150/150
4/4 - 1s - loss: 0.3026 - accuracy: 0.8687 - val_loss: 0.3069 - val_accuracy: 0.8770 - 553ms/epoch - 138ms/step

```

Una vez que se obtuvo el modelo para poder realizar las predicciones se procede a:

2. Ejecutar los servicios alojados en C:\RiesgoCardiaco\1_flask\app1\flaskr

Servicios ofrecidos

Los servicios utilizan **methods de tipo POST**:

Desarrollados en app.py:

altaUsuario: servicio para la incorporación de usuarios para el uso de la predicción

predictor: servicio que predice si se posee o no riesgo cardiaco

Desarrollado en guardarBitacora.py:

guardarBitacora: servicio de carga en la bitácora

Desarrollado en autorizacion.py:

autorizacion: servicio de autenticación

Comunicación entre servicios

El servicio <http://127.0.0.1:5000/predictor> requiere de los servicios <http://127.0.0.1:5001/autorizacion> a fin de validar usuarios que al devolver el valor 200, indica que ha validado correctamente al usuario.

Luego de una *correcta predicción*, evaluándose cantidad de consultas según tipo de usuario, cantidad y calidad de los parámetros para dicha medición, este resultado se almacena utilizando el servicio <http://127.0.0.1:5005/guardarBitacora>

Decisiones de Diseño

Con respecto al Conjunto de API Keys Válidas:

Actualmente, se utiliza un conjunto predefinido de API keys válidas, por ejemplo, "karina", "karina2"

Es dable mencionar que la llamada en la Base de Datos demora más que lo previsto para acceder a la nube y predecir, que utilizando una base de datos de manera local.

Uso de Redis:

Redis se utiliza para administrar la cantidad de consultas en el tiempo según el tipo de usuario que accede a la predicción.

Funciones Implementadas

Servicio *predictor*

validartiempo:

Objetivo: La función validarTiempo tiene como objetivo evaluar diferentes condiciones relacionadas con el tiempo y el usuario para determinar si deben permitirse más consultas, enviando respectivos mensajes:

Cuando un usuario FREEMIUM ha superado 5 consultas.

Cuando un usuario PREMIUM ha superado 50 consultas.

predecir_conTiempo:

Objetivo: La función predecir_conTiempo valida las condiciones del usuario según su tipo considerando lo registrado en Redis.

validarParametros:

Objetivo: La función que valida los parámetros, considerando la cantidad de parámetros enviados para la predicción como así también que estos se encuentren dentro de un rango correcto pre establecido, dependiendo del parámetro.

Programas Python del Proyecto

- db.py: Define el acceso a la base de datos.

Para la ejecución:

- 1) Se debe ejecutar redis-server

```
C:\redis\redis-server.exe
[3224] 14 Apr 07:34:02.842 # Warning: no config file specified, using the default config. In order to specify a config file use C:\redis\redis-server.exe /path/to/redis.conf

Redis 3.0.504 (00000000/0) 64 bit

Running in standalone mode
Port: 6379
PID: 3224

http://redis.io

[3224] 14 Apr 07:34:02.842 # Server started, Redis version 3.0.504
[3224] 14 Apr 07:34:02.858 * DB loaded from disk: 0.002 seconds
[3224] 14 Apr 07:34:02.858 * The server is now ready to accept connections on port 6379
```

2) Se debe ejecutar redis-cli

```
C:\redis\redis-cli.exe
127.0.0.1:6379>
```

3) En la consola, se pone cada uno de los servicios: “app.py”; “autorizacion.py”; “guardarBitacora.py”.

4) Ejecutar **Postman**, la herramienta que utilizará para realizar solicitudes a la aplicación. En la sección de encabezados (Headers), ingresa en el campo Authorization una de las siguientes Api_keys válidas: "karina", "karina2".

```
pp.py  autorizacion.py
RiesgoCardiaco > 1_flask > app2 > flaskr > autorizaci
1 import os
2 from flask import Flask, jsonify, request
3 from db import get_db
4 conjunto_api = {
5     "karina", "karina2"
6 }
7 app = Flask(__name__)
```

Import GET /hello POST /hello POST http://127.0.0.1:5000/all POST http://127.0.0.1:5000/pr POST http://127.0.0.1:5000/pr POST http://127.0.0.1:5000/all PC

HTTP http://127.0.0.1:5000/predictor

POST http://127.0.0.1:5000/altaUsuario?

Params Authorization Headers (9) Body Pre-request Script Tests Settings

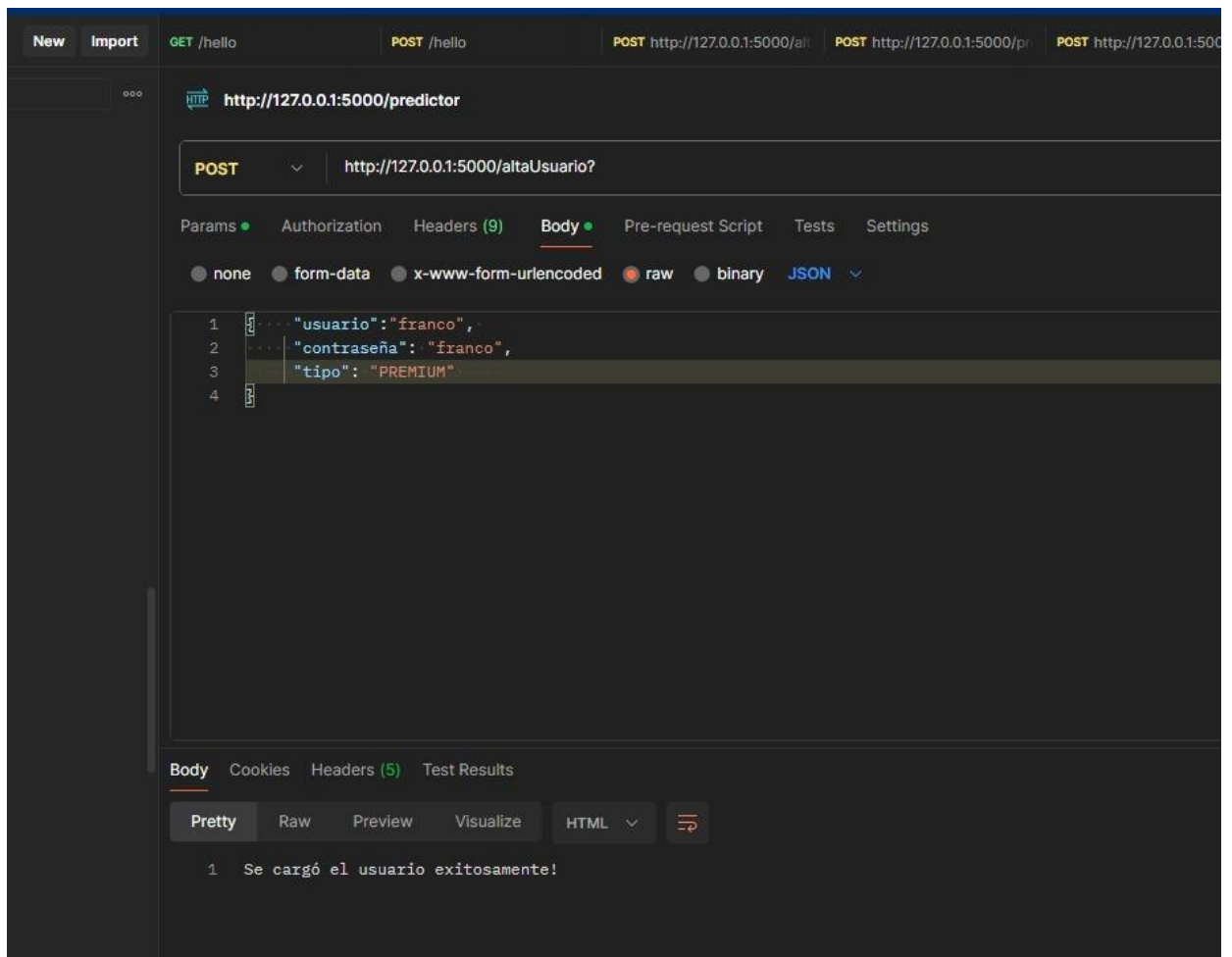
Headers Hide auto-generated headers

Key	Value
<input checked="" type="checkbox"/> Postman-Token ①	<calculated when request is sent>
<input checked="" type="checkbox"/> Content-Type ①	application/json
<input checked="" type="checkbox"/> Content-Length ①	<calculated when request is sent>
<input checked="" type="checkbox"/> Host ①	<calculated when request is sent>
<input checked="" type="checkbox"/> User-Agent ①	PostmanRuntime/7.37.3
<input checked="" type="checkbox"/> Accept ①	*/*
<input checked="" type="checkbox"/> Accept-Encoding ①	gzip, deflate, br
<input checked="" type="checkbox"/> Connection ①	keep-alive
<input checked="" type="checkbox"/> Authorization	karina

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize HTML

1 Mal tipo



5) Teniendo la siguiente Base de Datos `topicos2_bd`:

Ingresando a la BBDD alojada en **Atlas**

Usuario: kpolicano@hotmail.com

Contraseña: PracticoT2

Donde en la estructura *usuario* se guardan los datos de los usuarios, considerando para ser cargados: *appy_key*, *usuario*, *contraseña* y *tipo* de usuario (FREEMIUM-PREMIUM)

Karina's Org... Access Manager Billing

Data Services App Services Charts

+ Create Database

Search Namespaces

sample_mflix

topicos2_bd

request_log

usuario

topicos2_bd.usuario

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 523B TOTAL DOCUMENTS: 5 INDEXES TOTAL SIZE: 36KB

Find Indexes Schema Anti-Patterns Aggregation Search Indexes

Filter Type a query: { field: 'value' }

QUERY RESULTS: 1-2 OF 2

```

_id: ObjectId('661987ba5eac42c05768f1e9')
api_key: "karina2"
usuario: "Franco"
contraseña: "Franco"
tipo: "FREEMIUM"

_id: ObjectId('6619e07e1b17667e2fe2e451')
api_key: "karina"
usuario: "franco"
contraseña: "franco"
tipo: "PREMIUM"

```

Asimismo, cuenta con una bitácora, *request_log*, donde se registra cada consulta exitosa, dejando un historial de las mismas.

Karina's Org... Access Manager Billing

Data Services App Services Charts

KARINA'S ORG - 2024-04-12 > PROJECT 0 > DATABASES

ClusterO

Overview Real Time Metrics Collections Atlas Search Query Insights Profiler Performance Advis

DATABASES: 2 COLLECTIONS: 8

+ Create Database

Search Namespaces

sample_mflix

topicos2_bd

request_log

usuario

topicos2_bd.request_log

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 5,98KB TOTAL DOCUMENTS: 28 INDEXES TOTAL SIZE: 36KB

Find Indexes Schema Anti-Patterns Aggregation Search Indexes

Filter Type a query: { field: 'value' }

```

_id: ObjectId('6619ddd0e8e7f934836972e4')
timestamp: "2024-04-12T22:20:16.399735"
params: Array (7)
response: ": Por lo tanto, el paciente no tiene riesgo cardíaco"

_id: ObjectId('6619ddeae8e7f934836972e6')
timestamp: "2024-04-12T22:20:42.358601"
params: Array (7)
response: ": Por lo tanto, el paciente no tiene riesgo cardíaco"

_id: ObjectId('6619de0ae8e7f934836972e8')
timestamp: "2024-04-12T22:21:14.562910"
params: Array (7)
response: ": Por lo tanto, el paciente tiene riesgo cardíaco"

```

Caso de prueba

Activación de los servicios

Autorizacion.py utilizando el puerto: 5001

```
106 | else:
OUTPUT TERMINAL PORTS DEBUG CONSOLE
PS C:\RiesgoCardiaco\1_flask\app2\flaskr> py autorizacion.py
* Serving Flask app 'autorizacion'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5001
Press CTRL+C to quit
127.0.0.1 - - [13/Apr/2024 17:56:38] "POST /autorizacion HTTP/1.1" 200 -
127.0.0.1 - - [13/Apr/2024 17:57:03] "POST /autorizacion HTTP/1.1" 200 -
127.0.0.1 - - [13/Apr/2024 17:57:51] "POST /autorizacion HTTP/1.1" 200 -
127.0.0.1 - - [13/Apr/2024 17:58:23] "POST /autorizacion HTTP/1.1" 200 -
127.0.0.1 - - [13/Apr/2024 17:59:04] "POST /autorizacion HTTP/1.1" 200 -
127.0.0.1 - - [13/Apr/2024 17:59:25] "POST /autorizacion HTTP/1.1" 200 -
127.0.0.1 - - [13/Apr/2024 17:59:45] "POST /autorizacion HTTP/1.1" 200 -
```

app.py utilizando el puerto: 5000

```
● PS C:\RiesgoCardiaco> cd .\1_flask\
● PS C:\RiesgoCardiaco\1_flask> cd .\app2\
● PS C:\RiesgoCardiaco\1_flask\app2> cd .\flaskr\
○ PS C:\RiesgoCardiaco\1_flask\app2\flaskr> py app.py
2024-04-14 08:51:58.364671: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly d
nt computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
WARNING:tensorflow:From C:\Users\User\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\src\losses.py:297:
f.compat.v1.losses.sparse_softmax_cross_entropy instead.

* Serving Flask app 'app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
■
```

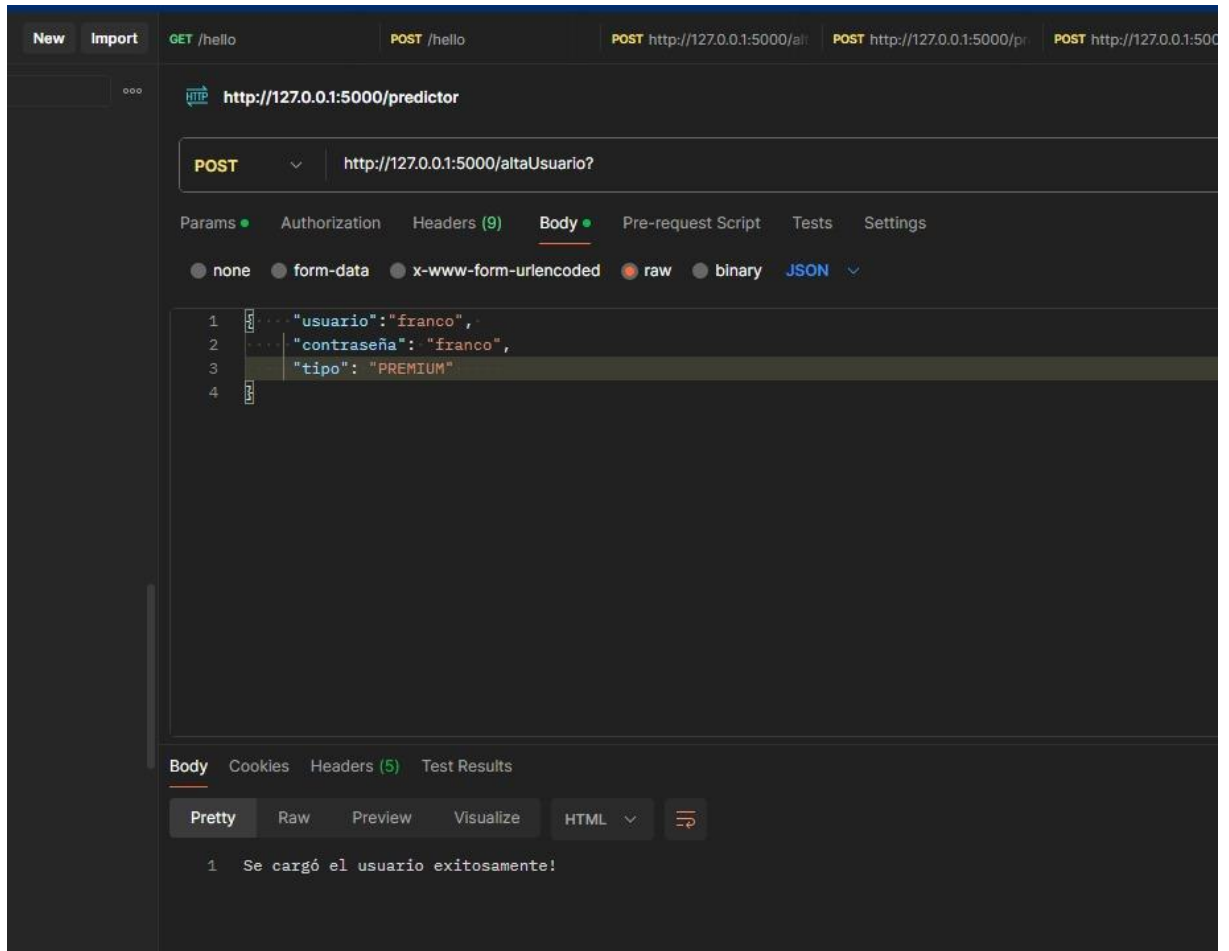
guardarBitacora.py utilizando el puerto: 5005

```
OUTPUT TERMINAL PORTS DEBUG CONSOLE
PS C:\RiesgoCardiaco\1_flask\app2> cd .\flaskr\
PS C:\RiesgoCardiaco\1_flask\app2\flaskr> py guardarBitacora.py
* Serving Flask app 'guardarBitacora'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5005
```

Servicio altaUsuario

Incorporación de usuarios

Se procede a insertar los usuarios de ambos tipos para la predicción, utilizándose para el usuario: franco de tipo PREMIUM la appy_key karina



Se procede a intentar insertar un usuario ya registrado, por lo que se obtiene el siguiente mensaje

POST

▼

http://127.0.0.1:5000/altaUsuario

ParamsAuthorizationHeaders (9)Body ●Pre-request ScriptTestsSettings

Headers🔗 Hide auto-generated headers

	Key	Value
<input checked="" type="checkbox"/>	Postman-Token ⓘ	<calculated when request is sent>
<input checked="" type="checkbox"/>	Content-Type ⓘ	application/json
<input checked="" type="checkbox"/>	Content-Length ⓘ	<calculated when request is sent>
<input checked="" type="checkbox"/>	Host ⓘ	<calculated when request is sent>
<input checked="" type="checkbox"/>	User-Agent ⓘ	PostmanRuntime/7.37.3
<input checked="" type="checkbox"/>	Accept ⓘ	*/*
<input checked="" type="checkbox"/>	Accept-Encoding ⓘ	gzip, deflate, br
<input checked="" type="checkbox"/>	Connection ⓘ	keep-alive
<input checked="" type="checkbox"/>	Authorization	karina2

The screenshot displays a REST client interface with a dark theme. At the top, a dropdown menu shows 'POST' and a URL 'http://127.0.0.1:5000/altaUsuario'. Below this, a horizontal tab bar includes 'Params', 'Authorization', 'Headers (9)', 'Body' (which is selected and underlined), 'Pre-request Script', 'Tests', and 'Settings'. Under the 'Body' tab, a radio button menu shows 'none', 'form-data', 'x-www-form-urlencoded', 'raw', 'binary', and 'JSON' (which is selected). The main area shows a JSON body with four lines:

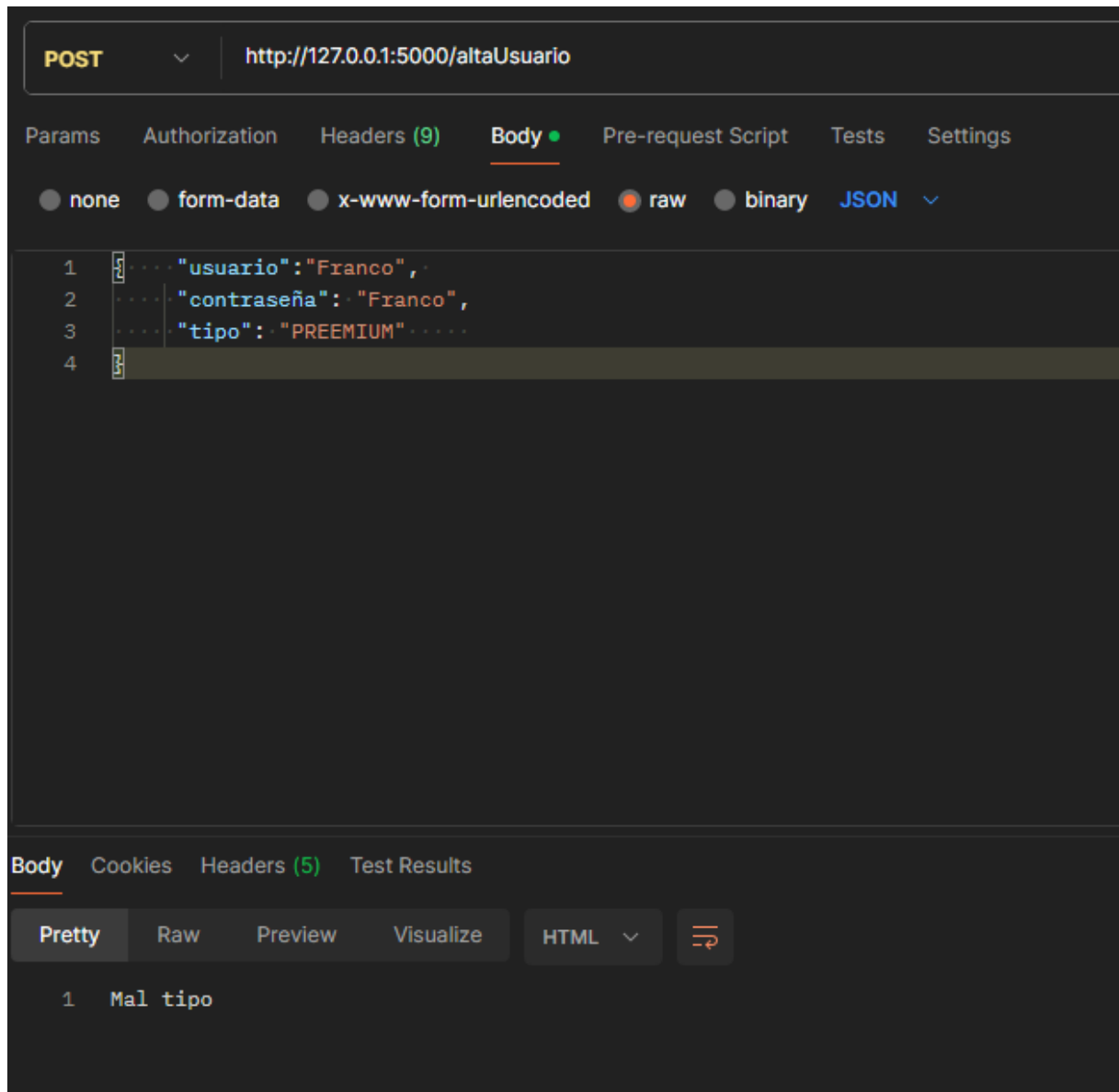
```
1 {
2   "usuario": "Franco",
3   "contraseña": "Franco",
4   "tipo": "FREEMIUM"
}
```

 At the bottom, another horizontal tab bar shows 'Body' (selected), 'Cookies', 'Headers (5)', and 'Test Results'. Below this, a row of buttons includes 'Pretty', 'Raw', 'Preview', 'Visualize', 'HTML' (with a dropdown arrow), and a button with a circular arrow icon. The bottom-most section shows a single line of text:

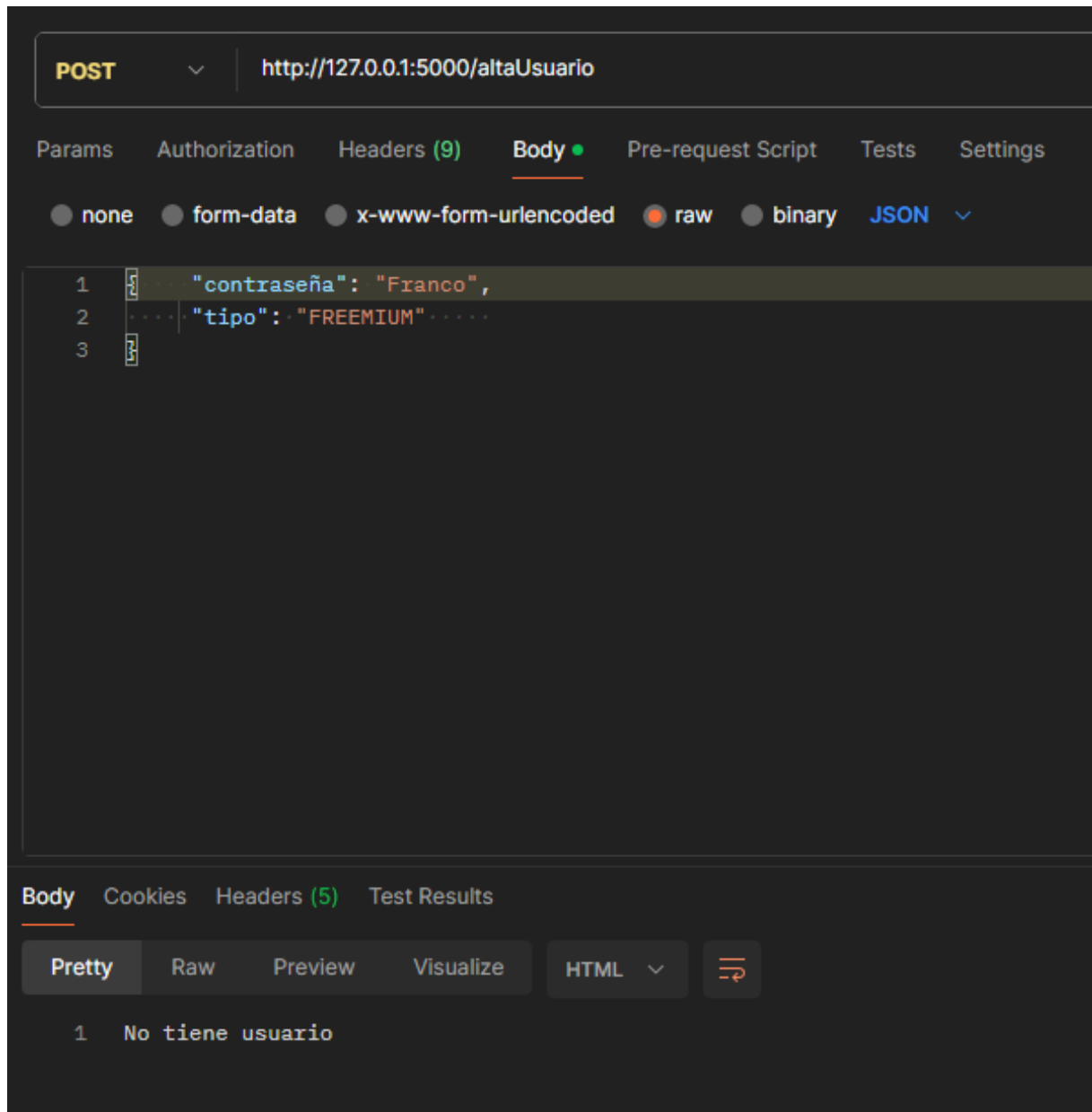
```
1 Existe el Api_key. Cada Api_key se asigna a un usuario
```

Errores probables

En caso que haya un error en el tipo, en este caso se utilizó el tipo *PREMIUM* de usuario a ingresar, muestra el mensaje respectivo:



Caso de intentar insertar un usuario sin uno de los parámetros como ser usuario, muestra el mensaje respectivo:




POST ▼ http://127.0.0.1:5000/altaUsuario

Params Authorization Headers (9) **Body** ● Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary JSON ▼

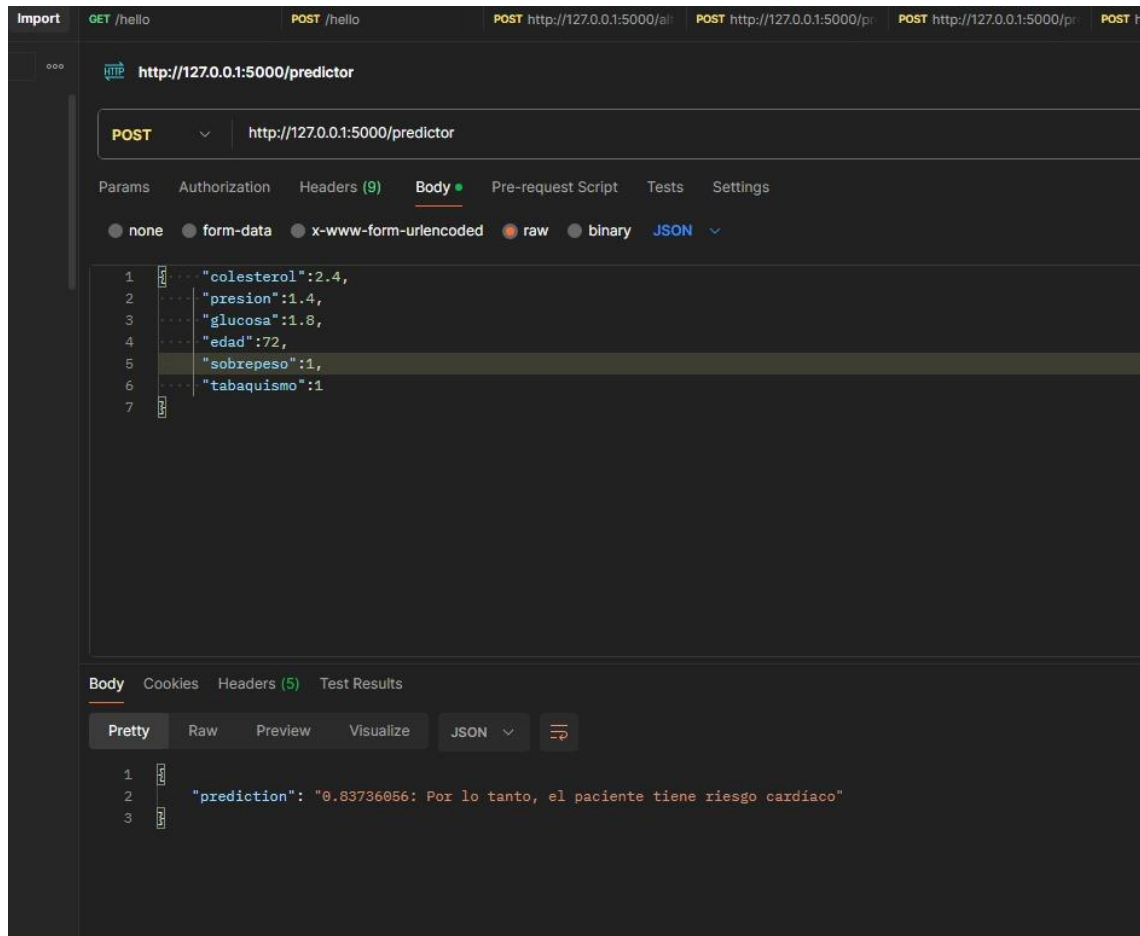
```
1 {  "usuario": "Franco",  
2  
3   "tipo": "FREEMIUM"  
4 }
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize HTML ▼ 

```
1 No tiene contraseña
```

Servicio predictor

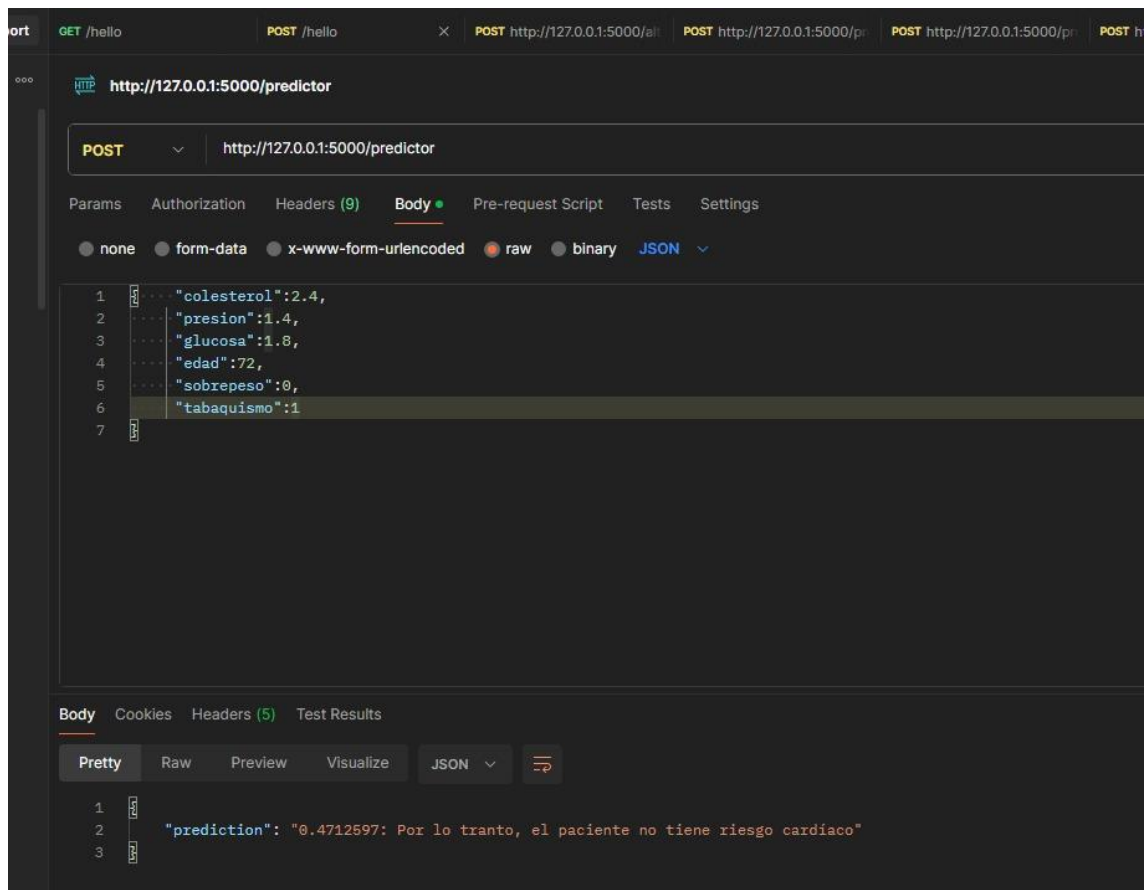


Servicio guardarBitacora

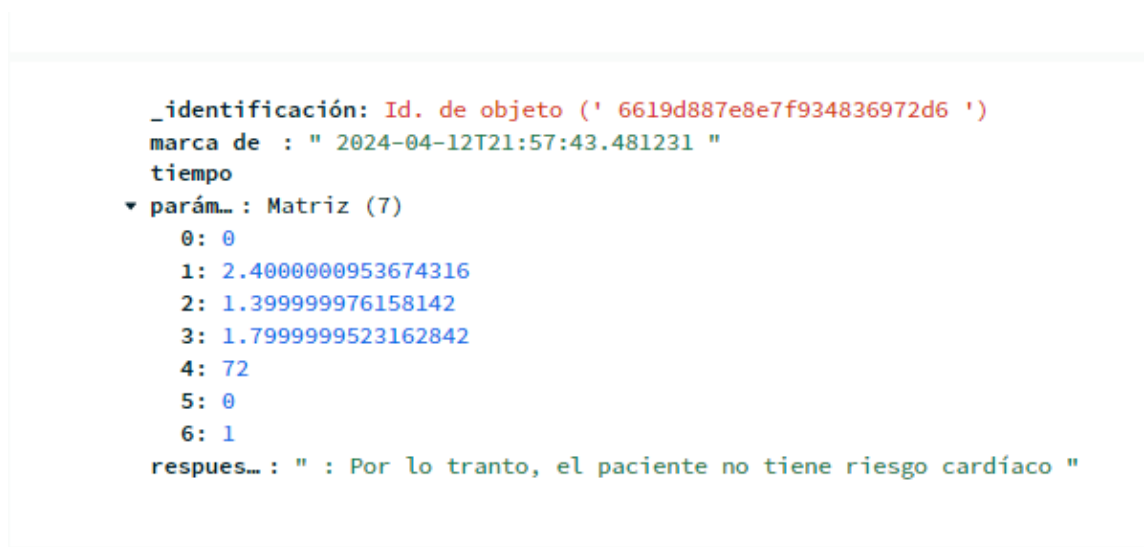
`http://127.0.0.1:5005/guardarBitacora`

Cuyo guardado en la colección `request_log` es:

```
_identificación: Id. de objeto(' 6619d7ffe8e7f934836972d4 ')  
marca de : " 2024-04-12T21:55:27.509578 "  
tiempo  
▼ parám... : Matriz (7)  
  0: 0  
  1: 2.40000000953674316  
  2: 1.3999999976158142  
  3: 1.79999999523162842  
  4: 72  
  5: 1  
  6: 1  
respues... : " : Por lo tanto, el paciente tiene riesgo cardíaco "
```

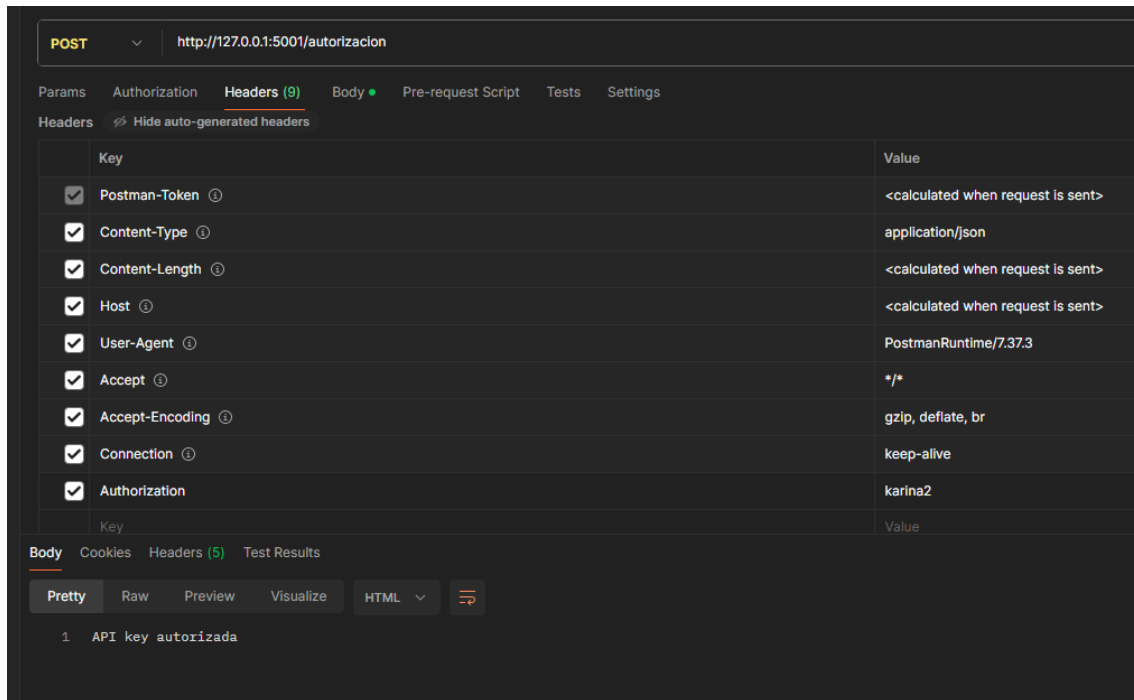



Cuyo guardado en bitácora es:



Manejo de consultas con Redis

En este caso de usuario tipo **FREEMIUM**, el servicio dejara de ser brindado a las 5 consultas, dato proporcionado por Redis.



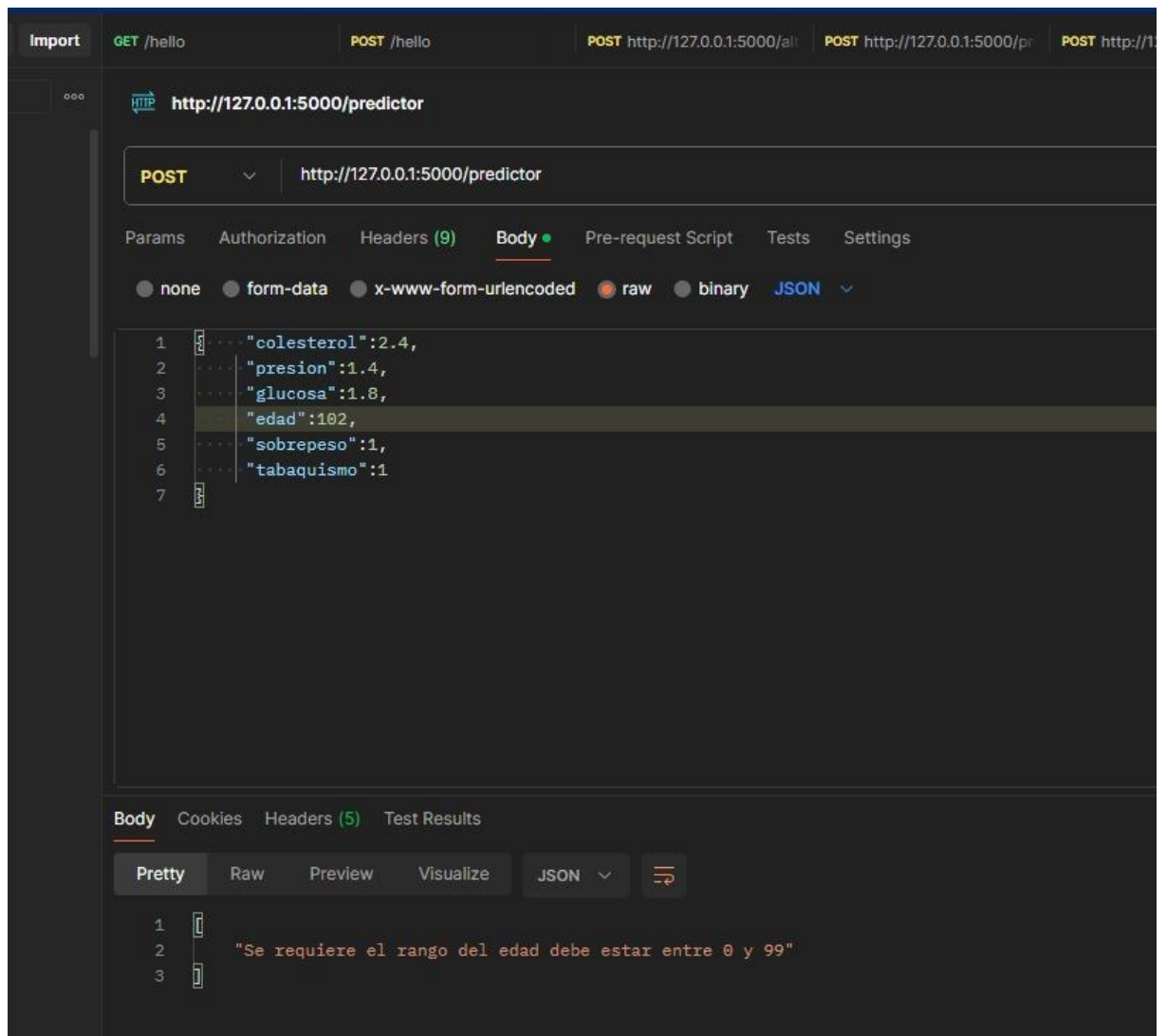
HTTP client interface showing a POST request to `http://127.0.0.1:5000/predict` with a JSON body. The body contains patient data: `{ "colesterol": 2.4, "presion": 1.4, "glucosa": 1.8, "edad": 52, "sobrepeso": 1, "tabaquismo": 1 }`. The response is a 400 status code with the message: `Excedio el limite de consultas el usuario`.

Terminal output (C:\redis\redis-cli.exe) showing Redis commands and responses:

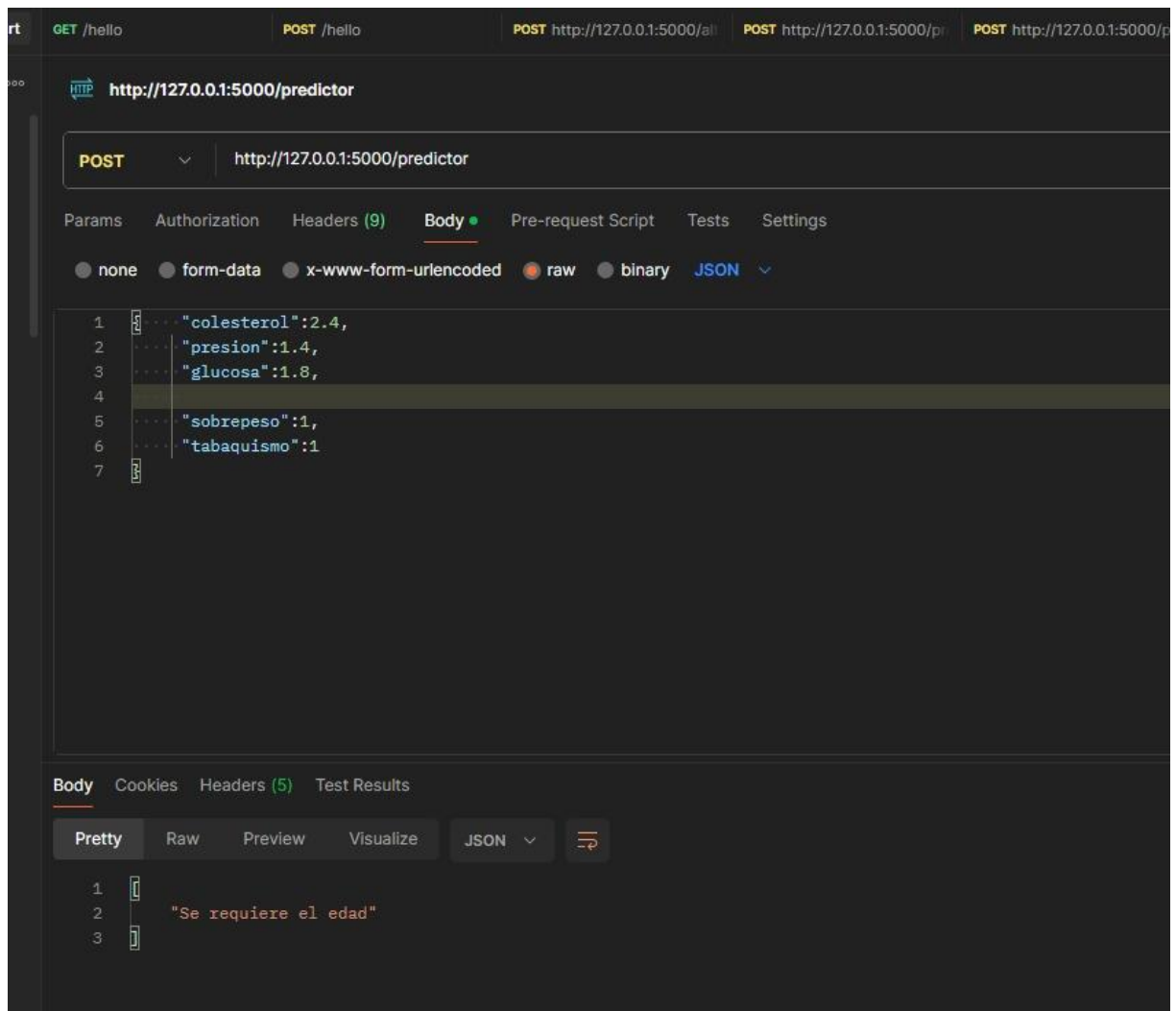
```
127.0.0.1:6379> HGETALL rate_limit:karina2
1) "count"
2) "4"
127.0.0.1:6379> HGETALL rate_limit:karina2
1) "count"
2) "5"
127.0.0.1:6379> HGETALL rate_limit:karina2
1) "count"
2) "5"
127.0.0.1:6379>
```

Errores posibles

En este caso emite el mensaje de parámetro fuera de rango



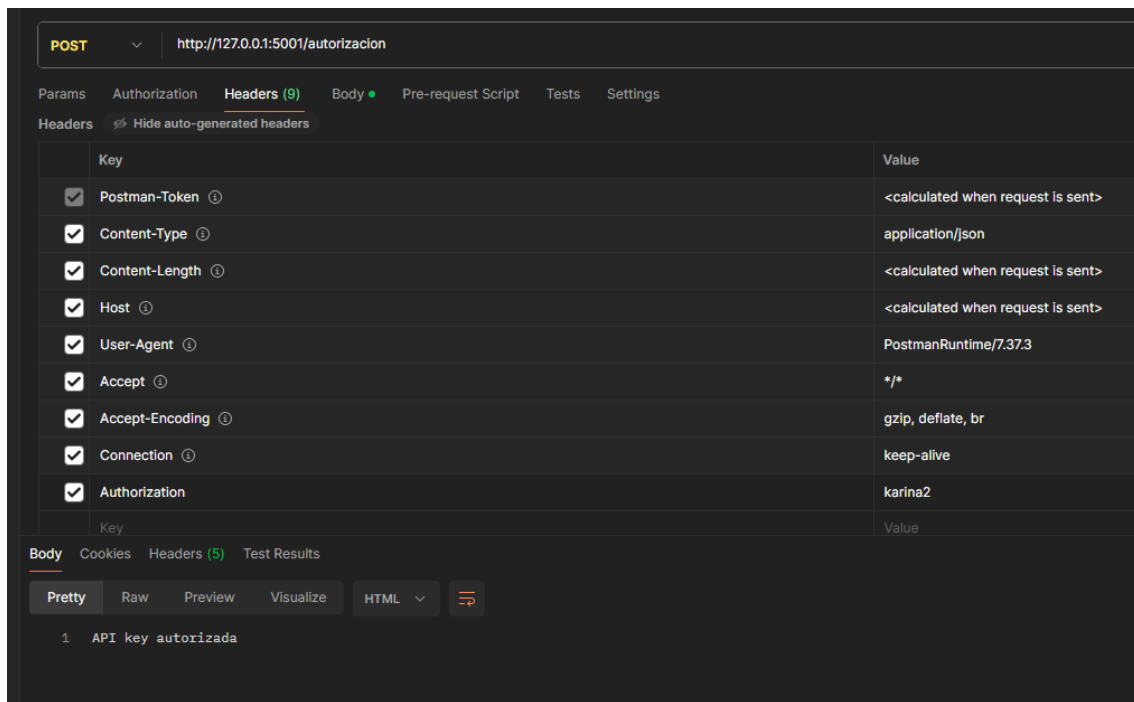
Falta de algún parámetro



Servicio autorizacion

<http://127.0.0.1:5001/autorizacion>

En este caso se ingresa la `apy_key` karina2 registrada



Errores posibles

En el caso de ingresar una `apy_key` no registrada tendremos el siguiente mensaje de error 401

