PharmaSUG China 2025 - Paper AP-161

API Speed Dating: SAS vs Python

Yanze Zhang, BioNTech

ABSTRACT

In the era of connected data ecosystems, the ability to efficiently interact with web-based APIs (Application Programming Interfaces) is becoming an essential skill for data professionals. This paper presents a practical and comparative exploration of API integration using two widely adopted programming environments: SAS and Python. The fundamental concepts of APIs, including HTTP methods, REST architecture, and SDKs are introduced first to provide a concise foundation for understanding how web services exchange information. To illustrate real-world applications, API calls using Google Cloud Translate and DeepSeek Chat API are demonstrated, showcasing how similar tasks can be executed in both Python and SAS. The paper explores the differences in coding complexity, highlighting Python's use of high-level SDKs versus SAS's reliance on PROC HTTP and manual payload and response handling. The convenience, flexibility and performance of the two approaches are assessed and compared, especially from the perspective of statistical programmers and analysts. A key focus of this study is a benchmarking experiment that evaluates the performance of SAS and Python in making repeated API calls over common datasets. Execution time and scalability under different loads are measured to offer insight into their suitability for batch API interactions. This paper aims to bridge the gap for SAS users venturing into web-based integrations and provides a practical reference for making informed decisions in hybrid programming environments.

INTRODUCTION

In today's data-driven landscape, the ability to seamlessly access, integrate, and automate data workflows across platforms is critical. As cloud platforms, third-party services, and internal tools increasingly expose their functionality via APIs, API integration becomes a foundational skill for data professionals, including statistical programmers. In the 2024 Gartner API Strategy Survey, 82% of respondents reported that their organizations use APIs internally, while 71% also use APIs provided by third parties such as SaaS vendors. As for statistical programmers in the pharmaceutical industry, SAS and Python are two major programming languages to analyze data. While both SAS and Python can complete the same API tasks, their underlying execution models, network handling, and payload management differ. It is interesting to explore the differences between the two approaches and perform some benchmark comparisons. These insights can help us make informed design decisions when integrating APIs into production workflows.

The structure of this paper is as follows:

1. Basic introduction to Web APIs
2. Overview of REST API Calling in SAS and Python
3. Case Study 1: Google Cloud Translate API
4. Case Study 2: DeepSeek Chat API
5. Benchmark Experiment and Comparison
6. Conclusion

BASIC INTRODUCTION TO WEB APIS

An API is a set of rules and protocols that allow different software applications to communicate and interact with each other. It acts as an intermediary that defines how requests and responses are structured between a client (the application making the request) and a server (the application providing the service). Much like a matchmaker at a speed dating event, these intermediaries help manage, facilitate, or enhance the communication between two parties - often without the client or server even realizing it. APIs enable software components to exchange data, access features, or perform functions

without needing to know the internal workings of each other. For example, a weather app on our cell phone uses an API to request weather data from a remote server and display it to the user.

REST API ARCHITECTURE

APIs includes Web APIs, Operating System APIs, Framework APIs and Hardware APIs. In this paper we only focus on Web APIs which are dominating the modern software development. Among different Web APIs the most widely used architectural style is called REST (Representational State Transfer) or RESTful APIs. REST is a set of architectural constraints, which are mostly applied using HTTP protocol. HTTP (Hypertext Transfer Protocol) is a protocol used for communication on the World Wide Web. It is the foundation of any data exchange on the Web and is an application layer protocol that facilitates the transfer of data between clients and servers. REST APIs use standard HTTP methods (GET, POST, PUT, DELETE) to manipulate resources. The operation of the four HTTP methods is illustrated in Table 1.

HTTP Method	Operation
GET	Retrieves data from a remote server. It can be a single resource or a list of resources.
POST	Creates a new resource on the remote server.
PUT	Updates the data on the remote server.
DELETE	Deletes data from the remote server.

Table 1. HTTP Methods and Operations

Resources, such as data or services, are uniquely identified by URL (Uniform Resource Locator) endpoint. For example, DeepSeek API uses the following URL endpoint:

```
https://api.deepseek.com/chat/completions
```

The server and client must communicate data in a format that the other will understand. The most common formats are JSON (JavaScript Object Notation) and XML (Extensible Markup Language). Many APIs have adopted the newer JSON representation because it's built on the popular JavaScript programming language, and it is a very simple format that is expressed using a combination of punctuation marks and real, readable words. Each object in JSON - set off between curly brackets - contains two pieces, keys and values, each of which are contained within quotation marks and separated by a colon. Below is an example of a JSON body used in Google Cloud Translate API request payload:

```
{
  "q": "The Great Pyramid of Giza (also known as the Pyramid of Khufu or
the Pyramid of Cheops) is the oldest and largest of the three pyramids in
the Giza pyramid complex.",
  "source": "en",
  "target": "es",
  "format": "text"
}
```

There are four attributes (keys) in this JSON body: “q” is the text to be translated, “source” is the original language code, “target” is the language code to translated into, and “format” is the format of the input text. After this request payload is sent to the server using HTTP method POST, the response payload is returned. The payload is the actual data being transmitted in an HTTP request or response body.

Think of an HTTP message as an envelope:

- Headers are like the label on the envelope - they describe the contents.
- Payload is the letter inside - the meaningful content.

The REST API architecture is illustrated in Figure 1. More details on REST API calling are introduced later in this paper.

API AUTHENTICATION

There are basically two authentication schemes: Basic Authentication and API key authentication. Basic Authentication only requires a username and password. The client takes these two credentials, smooshes them together to form a single value, and passes that along in the request in an HTTP header called Authorization. API key authentication is a technique that overcomes the weakness of using shared credentials by requiring the API to be accessed with a unique key. In this scheme, the key is usually a long series of letters and numbers that is distinct from the account owner's login password. One common approach is to have the client put the key in the Authorization header in lieu of a username and password. When the client authenticates with the API key, the server gives the client access to data, but now has the option to limit administrative functions, like changing passwords or deleting accounts. The flexibility is there with API key authentication to limit control as well as protect user passwords.

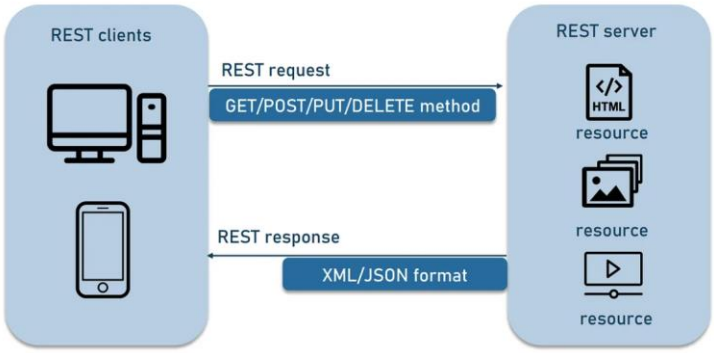

Figure 1. REST API Architecture

OVERVIEW OF REST API CALLING IN SAS AND PYTHON

PROC HTTP IN SAS

PROC HTTP is the primary procedure in SAS used to make HTTP requests to REST APIs. It allows SAS programmers to send data to web services and retrieve responses, making it possible to integrate external systems such as AI models, translation services into SAS workflows. A typical PROC HTTP block includes:

```
proc http
  url="https://api.example.com/endpoint"
  method="POST" /* or GET, PUT, DELETE */
  in=myPayload
  out=myResponse
  headerout=myHeader
  ct="application/json";
  headers
    "Authorization"="Bearer <api_key>"
    "Custom-Header"="Value";
run;
```

The purpose of each option is illustrated in Table 2. Bearer authorization is an HTTP authentication scheme commonly used with OAuth 2.0 (Open Authorization 2.0). In this approach, the client includes an access token in the "Authorization" header using the "Bearer" scheme, granting permission to access protected resources. APIs often require (or optionally support) custom headers beyond the standard ones like Authorization or Content-Type. "Custom-Header"="Value" would be replaced with actual headers defined by the API documentation you're using.

Option	Purpose
url=	The API endpoint you're calling
method=	HTTP verb (GET, POST, PUT, DELETE)
in=	File containing the request payload (e.g., JSON)

Option	Purpose
out=	File to store the response body
headerout=	File to store response headers (e.g., for debugging)
ct=	Content type (e.g., application/json)
headers	Allows setting custom HTTP headers

Table 2. SAS PROC HTTP Options

Since SAS is not natively object-oriented or JSON-first, the request body must be:

- Written to a file (using DATA _NULL_)
- Or generated from a dataset and exported as a text file

Example:

```
filename myPayload temp;

data _null_;
  file myPayload;
  put '{ "q": "Hello", "target": "fr", "source": "en" }';
run;
```

The response from an API is often in JSON format. SAS can parse this using:

```
libname respjson json fileref=myResponse;

data translations;
    set respjson.translations;
run;
```

This converts the JSON into SAS datasets, although complex nesting can be challenging to handle without manual restructuring.

Strengths of PROC HTTP for calling REST APIs:

- Native support in Base SAS (no need for additional components)
- Secure (supports HTTPS, bearer tokens, custom headers)
- Easily used in clinical reporting pipelines or batch processes

Limitations:

- Requires manual payload construction (limited JSON manipulation)
- JSON parsing can be confusing for nested structures
- Lacks the ecosystem and convenience of Python SDKs

SDKS AND STANDARD LIBRARIES IN PYTHON

Python offers a flexible and developer-friendly ecosystem for API integration, with tools ranging from low-level HTTP clients to high-level SDKs that abstract away much of the complexity.

Software Development Kits (SDKs) are official client libraries provided by API vendors (e.g., Google, AWS, Azure) to simplify authentication, request formatting, error handling, and response parsing.
Example:

```
from google.cloud import translate_v2 as translate

client = translate.Client()

result = client.translate(
    "Hello world",
```

```
    target_language="es"
)

print(result['translatedText'])  # Output: Hola Mundo
```

Behind the Scenes:

- Auth handled via environment variable (GOOGLE_APPLICATION_CREDENTIALS) or config file
- POST request with payload automatically generated
- JSON response parsed into Python dictionary

Besides SDKs, the requests package is one of the most popular third-party libraries for making HTTP calls manually.

Example:

```
import requests

url = "https://api.deepseek.com/chat/completions"
headers = {
    "Authorization": "Bearer <api_key>",
    "Content-Type": "application/json"
}
payload = {
    "model": "deepseek-chat",
    "messages": [{"role": "user", "content": "Tell me a joke"}]
}

response = requests.post(url, headers=headers, json=payload)
print(response.json())
```

Python has both SDKs with high-level of abstraction and requests package at medium level of abstraction. The benefits of SDKs are integrated authentication, low setup complexity, whereas requests package presents flexibility to be used for any REST APIs.

CASE STUDY 1: GOOGLE CLOUD TRANSLATE API

Google Cloud Translation API uses Google's neural machine translation technology to dynamically translate text through the API using a Google pre-trained, custom model, or a translation specialized large language model (LLMs). It comes in Basic and Advanced editions. The advanced offers customization features, such as domain-specific translation, formatted document translation, and batch translation. To start using Google Cloud Translation you should first set up credentials and authentication. Due to length constraints this paper won't cover this part.

PERFORMING TRANSLATION TASKS IN SAS

The use case involves translating character variables in a dataset from Chinese (or another source language) to a target language (e.g., English), with optional use of glossary-based translations provided by Google Cloud.

Workflow Overview (simplified workflow in Figure 2):

1. Dataset Preparation: The input dataset is preprocessed and sorted by key variables. Input variables are filtered for non-empty strings and split into chunks based on character count to comply with API limits.
2. Payload Construction: For each chunk, SAS dynamically creates a JSON payload using DATA _NULL_, matching Google's expected format (including "sourceLanguageCode",

"targetLanguageCode", "contents", and optionally a "glossaryConfig" block).

3. Access Token Handling: The macro retrieves an OAuth 2.0 access token from the system using the command-line tool gcloud and stores it in a macro variable.
4. API Call via PROC HTTP: Each chunk is submitted to the Google API endpoint using PROC HTTP, with the necessary headers:
   - Authorization: Bearer <access_token>
   - x-goog-user-project: <project_number>
   - Content-Type: application/json; charset=utf-8
5. Response Handling: The response is parsed using libname respjson json, and the translatedText values are extracted and merged back to the source data using ordinal mapping.
6. Glossary Support: If glossary-based translation is enabled (glossary_yn=Y), the request includes the glossary resource identifier, allowing consistent domain-specific translation.
7. Output: Translated variables are renamed and merged back into the original dataset. If all variables are selected (input_vars=ALL), labels are also translated and reapplied.

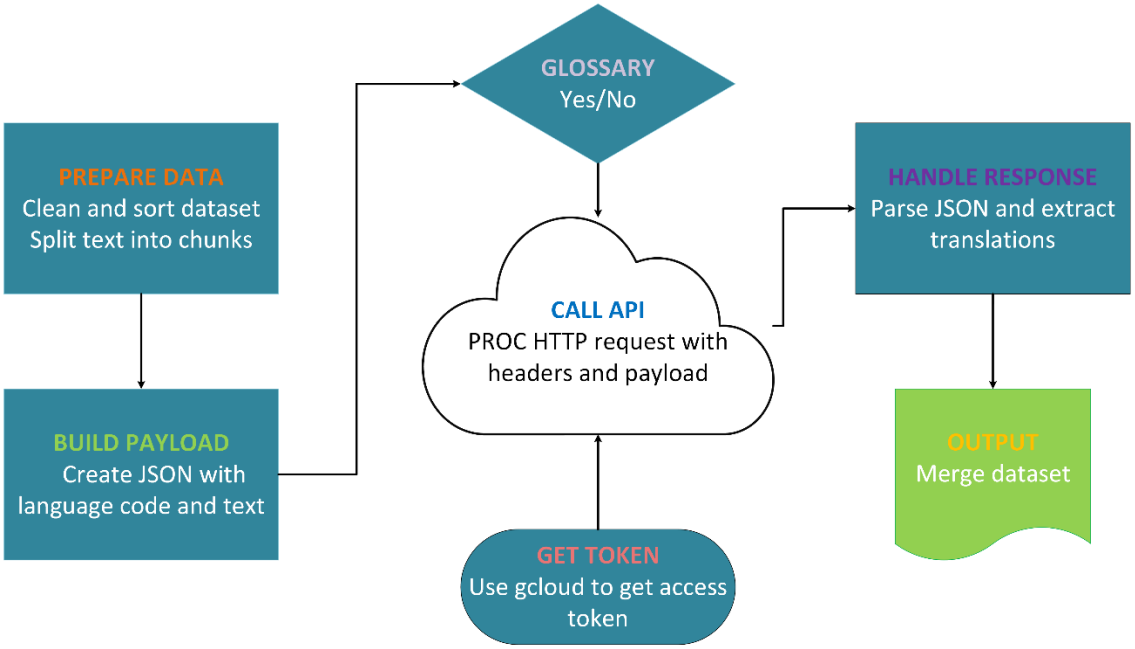

Figure 2. Workflow of Calling Google Cloud Translate API in a SAS macro

This macro demonstrates how SAS, despite being procedural and file-based, can fully integrate with modern cloud APIs using PROC HTTP.

PERFORMING TRANSLATION TASKS IN PYTHON

As a parallel to the SAS implementation, a Python-based solution was developed using the official Google Cloud Translation API v3 client library. The goal is to read a .txt file line by line, translate each line from a source language to a target language using the Google Translate API, and write the results to a new file. The solution is built around a Batch Translator Function:

- Reads all lines from an input .txt file.
- Splits the lines into batches based on a codepoint threshold (default: 30,720).
- For each batch, it sends a request to the Google Translate API using the google-cloud-translate

SDK.

- Collects and writes translated lines to a new .txt file.

The code to call google-cloud-translate SDK:

```
client = translate_v3.TranslationServiceClient()

PROJECT_ID = os.environ.get("GOOGLE_CLOUD_PROJECT")
location = "us-central1"
parent = f"projects/{PROJECT_ID}/locations/{location}"
mime_type = "text/plain"

response = client.translate_text(
    contents=batch,
    parent=parent,
    mime_type="text/plain",
    source_language_code="en-US",
    target_language_code="zh"
)
```

The advantages over SAS implementation are:

- Minimal setup: Uses the SDK with built-in authentication via the GOOGLE_CLOUD_PROJECT environment variable. But to authenticate Cloud Translation, Application Default Credentials (ADC) must be set up first.
- No manual JSON or header construction: All handled by the SDK.

CASE STUDY 2: DEEPSEEK CHAT API

DeepSeek Chat is a large language model (LLM) API developed by DeepSeek, offering capabilities similar to OpenAI's ChatGPT. DeepSeek's API follows a chat completion paradigm inspired by OpenAI's GPT-style APIs. This means users send a list of structured messages in the form of:

```
[
  {"role": "user", "content": "Your question here"}
]
```

and receive a response generated by the model. The API follows a standard RESTful design, accessed via: https://api.deepseek.com/chat/completions.

SAS IMPLEMENTATION

The SAS implementation showcases how to perform line-by-line question-answering tasks using DeepSeek's API. Key steps include:

1. Reading an input .txt file of user questions line-by-line.
2. For each question, manually crafting a JSON request payload inside a DATA _NULL_ step.
3. Sending the POST request using PROC HTTP with authorization headers.
4. Parsing the JSON response using the libname JSON engine to extract the answer.
5. Saving all responses in a numbered output .txt file

The process is macro-driven and JSON creation and response handling require careful formatting.

PYTHON IMPLEMENTATION 1: USING REQUESTS AND RAW API

This version mirrors the SAS approach but in Python using the requests library and manual JSON payloads:

```
import requests
headers = {"Authorization": "Bearer <api_key>", ...}
```

```
payload = {"model": "deepseek-chat", "messages": [{"role": "user",
"content": question}]}
requests.post(API_URL, headers=HEADERS, json=payload)
```

PYTHON IMPLEMENTATION 2: USING OPENAI SDK WITH DEEPSEEK ENDPOINT

This version takes advantage of DeepSeek's OpenAI-compatible interface by using the OpenAI Python SDK, with a custom base_url as DeepSeek's endpoint:

```
from openai import OpenAI
client = OpenAI(api_key="<api_key>", base_url="https://api.deepseek.com")
```

The actual API call is made by the following:

```
response = client.chat.completions.create(
    model="deepseek-chat",
    messages=messages,
    temperature=0.7,
    stream=False
)
content = response.choices[0].message.content.strip()
```

As Table 3 shows, there are several parameters you can use to fine-tune behavior, control output, and optimize performance.

Parameter	Type	Description
model	string	Model name. For example, "deepseek-chat"
messages	array	List of message objects (role + content) forming the chat history
temperature	float	Controls randomness. Range: 0.0–2.0. Higher = more creative
n	integer	Number of completions to generate for each input
stream	boolean	Whether to stream tokens back in real time
stop	string or array	Sequences where the model should stop generating further tokens
max_tokens	integer	Maximum number of tokens to generate in the response

Table 3 Common Parameters for Chat Completion APIs

Compared with the other two approaches, using OpenAI SDK to call DeepSeek API is simple and intuitive, no need to handle raw HTTP requests, and allows for more scalable, maintainable, and extendable API workflows.

BENCHMARK EXPERIMENT AND COMPARISON

This section presents a systematic benchmark experiment designed to evaluate and compare the performance and scalability of SAS and Python in interacting with modern REST APIs. The benchmarks are conducted using two representative APIs: Google Cloud Translate API for translation tasks and DeepSeek Chat API for language generation tasks. The comparison is structured around several key criteria, including execution time, ease of implementation, and ability to scale with data volume. To simulate real-world usage, English-language questions were randomly sampled from the SQuAD 2.0 (Stanford Question Answering Dataset version 2.0) training set (130319 questions). These questions were saved as a .txt file, with one question per line. This format allows for direct line-by-line processing and is compatible with both SAS and Python implementations. All experiments were conducted on a Windows laptop connected to the internet via home Wi-Fi. Access to Google Cloud Translate API requires an active VPN connection due to regional restrictions. VPN is not used for DeepSeek API test. Since there could be fluctuations in network speed, each benchmark test was repeated three times in both SAS and Python, and the average execution time was used in the final comparison to ensure relative reliability.

GOOGLE CLOUD TRANSLATE API TEST

Goal: Translate the 100/1,000/5,000/10,000 questions from English to Chinese.

Execution time was recorded in SAS using:

```
%let launchTime = %sysfunc(time());
<other code>
%let timeTaken = %sysevalf(&landTime.-&launchTime.);
%let timeTakenFmt = %sysfunc(putn(&timeTaken, 8.2));
%put *****     TIME TAKEN (seconds): &timeTakenFmt     *****;
```

In Python:

```
import time
start_time = time.time()
<other code>
end_time = time.time()
elapsed_time = end_time - start_time
print(f"Translation completed in {elapsed_time:.2f} seconds.")
```

	SAS		Python	
	Time (s)	Avg.	Time (s)	Avg.
100 Questions (5957 Characters)	1.78 1.81 2.07	1.89	4.07 6.08 4.30	4.82
1,000 Questions (58,925 Characters)	6.28 5.06 5.30	5.55	6.32 5.69 6.25	6.09
5,000 Questions (295,890 Characters)	28.17 26.13 25.75	26.68	14.30 11.16 14.39	13.28
10,000 Questions (593,394 Characters)	54.50 51.71 52.11	52.77	19.97 22.71 20.49	21.06

Table 4 Translation Execution Time (in seconds)

As Table 4 shows:

- At smaller sizes, SAS wins due to lower start-up overhead - Python loads more libraries and authenticates in more complex ways.
- Python still holds edge in large jobs, likely due to:
  - Internal efficiencies in the google-cloud-translate client.
  - Better I/O handling and faster JSON processing.

One of the key challenges when calling APIs from SAS is the manual construction of JSON payloads - particularly when dealing with special characters such as quotation marks ("). In JSON, quotation marks within string values must be escaped using a backslash (\") to prevent syntax errors. This process is not handled automatically in SAS, so the programmer must explicitly encode any embedded quotation marks

in strings. By contrast, Python handles JSON serialization automatically and securely via libraries like json or SDK methods

DEEPSEEK CHAT API TEST

Goal: Submit each question to DeepSeek's LLM and record the response.

Due to the inherently longer response time of generative models and the computational cost of multi-turn completions, the test was limited to 10/50/100 questions randomly selected from the SQuAD 2.0 training set. The 50 questions randomly picked are listed in the Appendix for reader's interest.

The following three implementations were compared:

1. SAS with PROC HTTP
2. Python with requests library and manual JSON payloads
3. Python with OpenAI SDK

	SAS		Python Requests		Python SDK	
	Time (s)	Avg.	Time (s)	Avg.	Time (s)	Avg.
10 Questions	130.19 139.00 127.75	132.31	132.35 132.29 140.99	135.21	122.51 128.20 129.38	126.70
50 Questions	684.72 700.06 671.28	685.35	640.78 655.46 612.91	636.38	631.87 669.70 640.26	647.28
100 Questions	1304.92 1338.22 1388.76	1343.97	1363.00 1356.62 1400.95	1373.52	1234.68 1287.46 1241.07	1254.40

Table 5. DeepSeek Chat API Execution Time (in seconds)

As Table 5 shows:

- All three methods show relatively consistent performance across repeated runs.
- Python requests approach is close in performance to SAS.
- Python with OpenAI SDK demonstrates the best average performance.

CONCLUSION

This study conducted a side-by-side "speed dating" comparison between SAS and Python in the context of modern API integration, specifically through two case studies: Google Cloud Translate API for text translation and DeepSeek Chat API for LLM-based question answering. Both tools were tested for performance, scalability, and ease of integration in real-world programming environments.

In the translation use case, both SAS and Python successfully interfaced with Google Cloud's Translate API. The SAS implementation, though functionally robust, involved more complex JSON generation and manual handling of escape characters (e.g., quotes) in payload construction. In contrast, Python offered streamlined development via google-cloud-translate SDK and achieved significantly faster runtime, particularly as the number of translation requests scaled up to 10,000. The use of codepoint-aware batching (up to 30,720 characters) further enhanced Python's efficiency.

For the DeepSeek Chat API, the LLM task was inherently slower due to model response time. We tested three methods: SAS using PROC HTTP, Python using the requests module, and Python using the

OpenAI SDK targeting DeepSeek's endpoint. All three were able to return coherent answers, but the Python SDK approach consistently demonstrated the best performance and least code complexity.

Ultimately, this speed dating session revealed that Python outperforms SAS in both tasks, especially under high-volume loads. This advantage is attributed to Python's optimized SDKs, more efficient string and I/O handling, and easier batch processing logic. Meanwhile, SAS, while capable, is better suited for small-to-medium scale tasks or environments where it is already embedded in enterprise workflows.

REFERENCES

Google. "Cloud Translation Guides" Available at https://cloud.google.com/translate/docs/overview

DeepSeek. "DeepSeek API Docs" Available at https://api-docs.deepseek.com

Cooksey, Brian. "An introduction to APIs" Zapier. Accessed January 2024. Available at https://zapier.com/resources/guides/apis

Pilawka, Olga. "What Are RESTful Web APIs" Stackademic. Accessed Nov 19, 2023. Available at https://medium.com/stackademic/what-is-restful-web-apis-part-1-the-definition-of-api-68b4f422519e.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Yanze Zhang
BioNTech Shanghai Pharmaceuticals Co. Ltd.
Unit 3401-3402, Level 34, Tower 1
No 288 Shimen Yi Road
Shanghai, China
Email: yanze.zhang@biontech.com

APPENDIX

Randomly picked 50 SQuAD 2.0 questions:

Who wrote about their opponents in the 200's

In what time period did the Devangari script become prevalent?

When did Russian language spread to the peasants?

What type of iron could pig iron be converted into?

What gave rise to a distinct Canaanite ethnic group?

When did eudicots replace conifers as dominant trees?

What started this practice in 2003?

What is the longest pier in Somerset?

What program is being installed in madrasahs outside of Singapore?

Because of their heat resistance, glass-ceramics are especially suitable for what?

What hard rock supergroup contained members of Soundgarden and Rage Against the Machine?

How much in stimulus money was announced in 2008?

Coercion of a representative or a state itself will result in what happening to its consent to a treaty?

Who offered many Africans a chance to study in their country?

Who said "This extrapolation, however, is impossible... atoms are lots of things"

The abbey was without towers following the destruction by whom?

Each Brigade contains how many regiments?

What class of considerations did Popper believe were not important in scientific measurement?

How long does each cycle of birth and destruction of the universe last according to Ancient Indian texts?

Who weeps around the body of the King?

What is an example of a dielectric relaxation process?

Which dynasty supported the Deccani Urdu literature movement?

What term did Irenaeus use to describe the Christian community's ideologies?

Who was the minister of the Interior for Thuringia in 1930?

What English mathematician had no interest in philosophy?

What is in the Fayetteville business district?

Where is additional genetic material found in pathogenic microbes?

What is it called when a monarch has a share of executive powers?

Who designed the Praça dos Três Poderes?

Where does the Permanent Secretary serve as the senior civil servant?

How long does basic training last?

What is the size of non-repetitive DNA divided by to get the proportion of non-repetitive DNA?

What church uses mainly the Russian language?

What border does I-19 come close to?

Rabbi Yehuda Hachasid led a group of how many Jews into Jerusalem?

In what year did "the first genocide of the Twentieth Century" end?

Who protested for their compatriots who were in Xinjiang?

Why has defining pain been a challenge?

Did rising efficiency in production and communication lower or raise the prices of consumer goods?

Who were the two most powerful nations in the 1860's?

What century did Songhai become independent from Mali?

On what river is London situated?

Who rejected the plan known as "the Grand Model"?

How much did clubs replay each other after the 1990s?

What did the Prussian Army hold in Paris on 17 February?

ABC Records acquired what Nashville music publisher?

What did the Top Hoodlum Program gather information on?

How was the committee of ministers divided?

Etch-back removes glass fibers and what other material?

On what day did both the Julian and Gregorian calendars add leap day?