IES AUGUSTO GONZALES DE LINARES

CFGS DAM2º

APLICACIÓN DE CONSULTA A API REST

Programación De Servicios Y Procesos

Daniel Espinosa García

CAFÉ CON PALITO

Desarrollo de Aplicaciones

2023/2024

Índice

1. Elección de API

Siguiendo los criterios requeridos para la práctica y despues de investigar dos repositorios de repositorios de API's gratuitos RapidApi y PublicApi me decante por la utilización de la API DeeplL/ Translator que permite traducir texto a varios idiomas. En su versión gratuita permite un uso de 500.000 caracteres.

Me decante de un diseño sencillo en el cual el usuario solo necesita introducir el texto y el idioma de destino, la propia API es capaz de detectar el idioma de origen. Replicando el diseño que utiliza DeepL en la web.

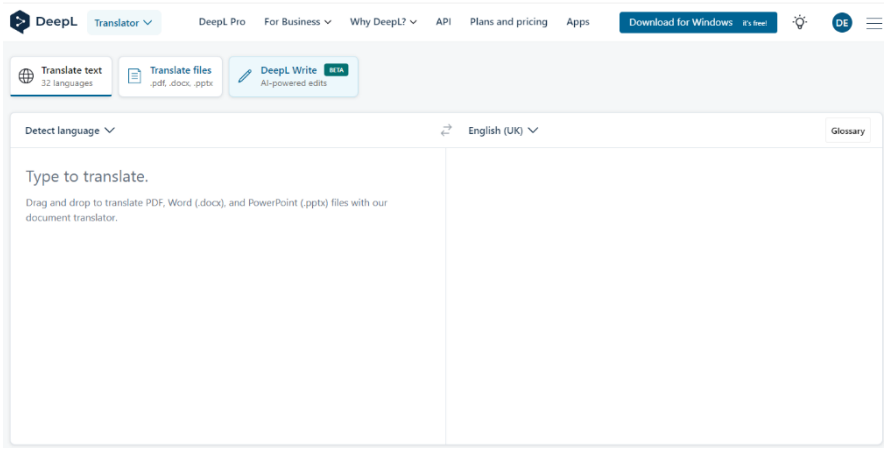

Ilustración 1 Interfaz web de DeepL / Translator

1.1. DeepL / Translator

Lo primero que realice fue el registro en la Web para tener acceso a la API, la cual necesita de una ApiKey para autentificar al usuario.

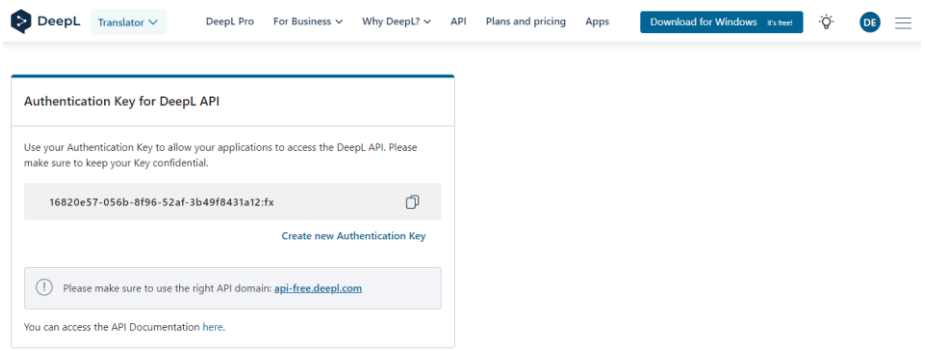

Ilustración 2 DeepL ApiKey

La API de DeepL cuenta con una documentación que explica su funcionamiento, en ella encontré los detalles de cómo utilizar los End Points que utilizo en la práctica.

End Point /Translate

Permite enviando un texto y un lenguaje de objetivo que la API lo traduzca. No es necesario introducir el lenguaje original del texto ya que ella misma se ocupa de averiguarlo.

Aun asi entre las opciones que permite la API se incluye:

- source_lang: Indicar el idioma de origen del texto
- target_lang: Indica el idioma al que tiene que ser traducido el texto
- formality: La formalidad del texto traducido, la cual tiene niveles de ajuste.

Devuelve la información en un JSON Array que contiene dentro un JSON Object con la traducción y los parámetros solicitados.

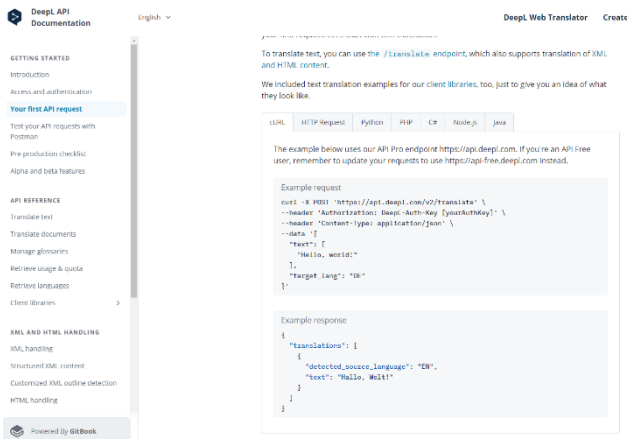

Ilustración 3 Documentación End Point Translate

End Point /Languages

Este end point nos da acceso a todos los idiomas que permite la aplicación traducir.

Dándonos su nombre, una abreviatura que se utilizara para designar el idioma de destino u origen del texto a traducir y si permite que pueda ser pasado a un lenguaje más formal o no en modo de booleano.

Nos entrega la información en modo de un JSON Array que contiene múltiples JSON Objects uno por cada idioma.

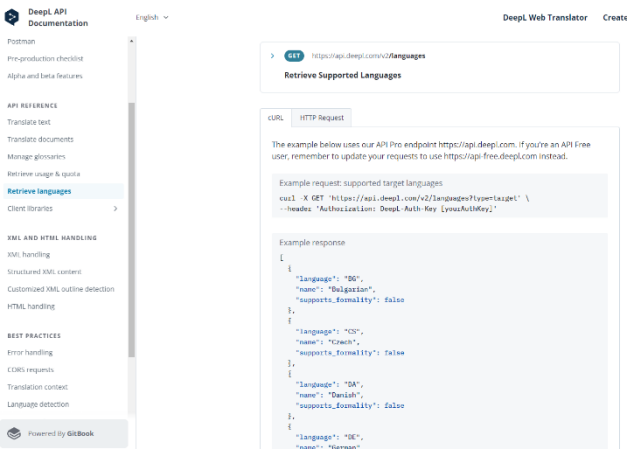

Ilustración 4 Documentación End Point Languages

End Point /Usage

Permite saber el uso de la aplicación y el límite que tiene el usuario, en el modo gratuito cuenta con 500.000 caracteres al mes para traducir.

Este end point solo requiere que se le envié la ApiKey del usuario y devolverá el total de caracteres utilizados y el límite de caracteres que tenga la cuenta.

Entrega la información en modo de un JSON Object.

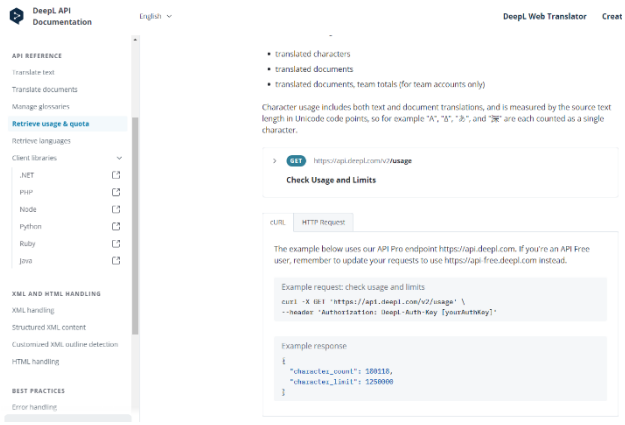

Ilustración 5 Documentación End Point Usage

2. Dependencias Cliente API (org.json)

Para realizar la práctica seleccione la dependencia org.json que consta de distintas clases con sus metodos que permiten realizar el manejo de la información recibida en formato JSON, de los cuales caben mencionar:

JSONArray: nos permite crear objetos de esta clase para almacenar la información recibida en un Array de Objetos y poder recorrerlo rescatando la información almacenada.

JSONObject: esta clase tambien nos permite crear objetos de esta, y almacena la información en mapas de Clave Valor, que es la manera que utiliza JSON para tratar la información, permitiendo la búsqueda de los datos de manera rápida utilizando los metodos que la clase contiene.

Utilizaremos estas clases y algunos de sus metodos para realizar la práctica.

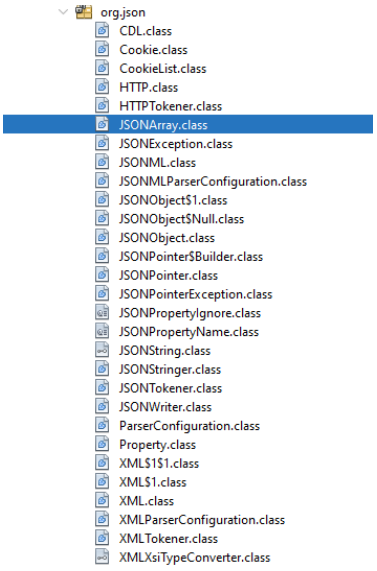

Ilustración 6 Clases de la Dependencia org.json

3. Métodos Relevantes en la Aplicación

La aplicación consta de cuatro metodos relevantes los cuales atacan a los End Points antes mencionados.

Método traducir()

Utilizará el End Point /Translate al cual enviara la información al servidor y devolverá un JSON con el texto traducido y el idioma de origen, trataremos la información utilizando la dependencia org.json para mostrarla en la interfaz gráfica.

```
/**
 * Metodo que envia la peticion POST para que retorne el texto traducido y
 * mostrarlo por pantalla.
 */
private void traducir() {

    try {
        String texto = jTextAreaInText.getText();

        HttpRequest request = HttpRequest.newBuilder()
                .uri(uri: URI.create(str: "https://api-free.deepl.com/v2/translate"))
                .header(name:"Authorization", "DeepL-Auth-Key " + apyKeyTranslate)
                .header(name:"content-type", value:"application/x-www-form-urlencoded")
                .timeout(duration:Duration.ofSeconds(seconds: 5))
                .method(method: "POST", bodyPublisher:HttpRequest.BodyPublishers.ofString("text=" + texto + "&" + "target_lang=" + languageSelected()))
                .build();
        HttpResponse<String> response = HttpClient.newHttpClient().send(request, responseBodyHandler: HttpResponse.BodyHandlers.ofString());

        JSONObject traduc = new JSONObject(source: response.body());

        jTextAreaTraduction.setText(t: traduc.getJSONArray(key: "translations").getJSONObject(index: 0).getString(key: "text"));

    } catch (IOException ex) {
        Logger.getLogger(name:Traductor.class.getName()).log(level:Level.SEVERE, msg: null, thrown: ex);
    } catch (InterruptedException ex) {
        Logger.getLogger(name:Traductor.class.getName()).log(level:Level.SEVERE, msg: null, thrown: ex);
    }

}
```

Ilustración 7 Método traducir

Método obtainLanguages()

Este método utiliza el End Point /Languages el cual tras realizar la petición el servidor nos envía un JSONArray que contiene JSONObject con los lenguajes que permite traducir. Recojo la información que necesito, en este caso, name y language. Para almacenarlo en un mapa de Clave Valor. Mostrando por la interfaz gráfica la Clave para que el usuario seleccione el idioma al que quiere traducir el texto.

```
/**
 * Metodo que obtiene los lenguajes de la api y los almacena en un HashMap.
 */
private void obtainLanguages() {

    try {
        //Envio la peticion a la API
        HttpRequest getLanguages = HttpRequest.newBuilder()
                .uri(uri: URI.create(str: "https://api-free.deepl.com/v2/languages?type=target"))
                .header(name:"Authorization", "DeepL-Auth-Key " + apyKeyTranslate)
                .timeout(duration:Duration.ofSeconds(seconds: 5))
                .GET()
                .build();
        //Recibo la respuesta de la API COMO UN STRING
        HttpResponse<String> responseLang = HttpClient.newHttpClient().send(request: getLanguages, responseBodyHandler: HttpResponse.BodyHandlers.ofString());

        //Cargo la respuesta de la apy en un JSONARRAY
        JSONArray jArrayLang = new JSONArray(source: responseLang.body());

        //Por cada objeto obtenido de la respuesta que lleve al Array lo cargo en un hash map para obtener un par clave
        //valor y poder luego seleccionar el idioma a traducir.
        for (int i = 0; i < jArrayLang.length(); i++) {
            JSONObject aux = jArrayLang.getJSONObject(index: i);
            languages.put(key: aux.getString(key: "name"), value: aux.getString(key: "language"));
        }

    } catch (IOException ex) {
        Logger.getLogger(name:Traductor.class.getName()).log(level:Level.SEVERE, msg: null, thrown: ex);
    } catch (InterruptedException ex) {
        Logger.getLogger(name:Traductor.class.getName()).log(level:Level.SEVERE, msg: null, thrown: ex);
    }
}
```

Ilustración 8 Método obtainLanguages

Método usage()

Ataca al End Point /Usage recibiendo como respuesta del servidor un JSONObject con un par de claves valor uno para el numero de caracteres utilizados y otro para el límite de caracteres mensual de la cuenta. Esta información se muestra en la interfaz gráfica.

```java
/**
 * Metodo que actualiza el uso que tiene la aplicacion.
 */
private void usage() {

    try {
        HttpRequest getUsage = HttpRequest.newBuilder()
                .uri(uri: URI.create(str: "https://api-free.deepl.com/v2/usage"))
                .header(name:"Authorization", "DeepL-Auth-Key " + apyKeyTranslate)
                .timeout(duration:Duration.ofSeconds(seconds: 5))
                .GET()
                .build();
        //Recibo la respuesta de la API COMO UN STRING
        HttpResponse<String> responseUsage = HttpClient.newHttpClient().send(request: getUsage, responseBodyHandler: HttpResponse.BodyHandlers.ofString());

        JSONObject joUsage = new JSONObject(source: responseUsage.body());

        jLabelCaracteresUsados.setText("Caracteres usados: " + joUsage.getInt(key: "character_count"));
        jLabelLimiteCaracteres.setText("Limite de caracteres: " + joUsage.getInt(key: "character_limit"));

    } catch (IOException ex) {
        Logger.getLogger(name:Traductor.class.getName()).log(level:Level.SEVERE, msg: null, thrown: ex);
    } catch (InterruptedException ex) {
        Logger.getLogger(name:Traductor.class.getName()).log(level:Level.SEVERE, msg: null, thrown: ex);
    }

}
```

Ilustración 9 Método usage

Método statusService()

Para comprobar que el servicio está activo al iniciar la aplicación ataco dos End Points de DeepL si todo es correcto devuelve True en caso contrario False. Tanto este como los metodos anteriores utilizan el método .timeout() que permite establecer un tiempo de espera de respuesta del servidor y si falla devuelve una excepción.

```java
/**
 * Metodo que comprube el status de los servicios de la api. si todos son
 * correctos devuelve true;
 */
private boolean statusService() {

    try {
        HttpRequest getUsage = HttpRequest.newBuilder()
                .uri(uri: URI.create(str: "https://api-free.deepl.com/v2/usage"))
                .header(name:"Authorization", "DeepL-Auth-Key " + apyKeyTranslate)
                .timeout(duration:Duration.ofSeconds(seconds: 5)) // Tiempo de espera para comprobar
                .GET()
                .build();
        //Recibo la respuesta de la API COMO UN STRING
        HttpResponse<String> responseUsage = HttpClient.newHttpClient().send(request: getUsage, responseBodyHandler: HttpResponse.BodyHandlers.ofString());

        HttpRequest getLanguages = HttpRequest.newBuilder()
                .uri(uri: URI.create(str: "https://api-free.deepl.com/v2/languages?type=target"))
                .header(name:"Authorization", "DeepL-Auth-Key " + apyKeyTranslate)
                .timeout(duration:Duration.ofSeconds(seconds: 5)) // Tiempo de espera para comprobar
                .GET()
                .build();
        HttpResponse<String> responseLang = HttpClient.newHttpClient().send(request: getLanguages, responseBodyHandler: HttpResponse.BodyHandlers.ofString());

        return true;

    } catch (IOException | InterruptedException ex) {
        return false;
    }
}
```

Ilustración 10 Método statusService

4. Demostración de funcionamiento

Mediante la interfaz gráfica se nos permite:

- Introducir un texto en el TextArea para su traducción.
- Seleccionar mediante un ComboBox el lenguaje al que se quiere traducir el texto, obtenidos desde la API
- Utilizando el botón de Traducir traduce el texto introducido al idioma seleccionado.
- El botón de limpiar limpia los TextArea.

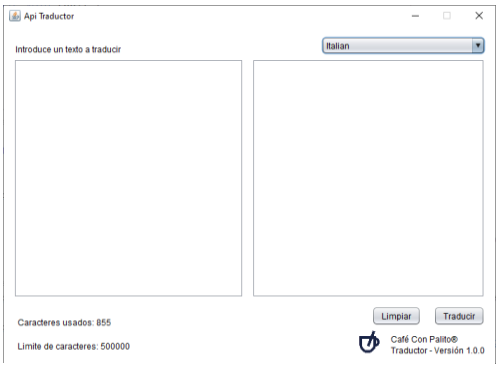

Ilustración 11 Aplicación Traductor

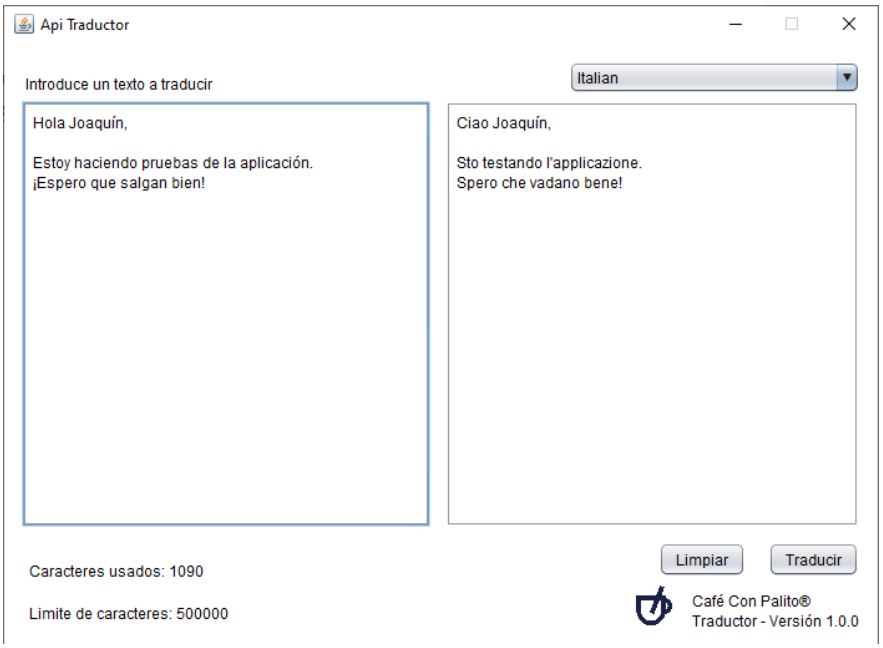

Ilustración 12 Demostración de uso de la aplicación

5. Investigación sobre la creación de servidores API REST

Existen diversas maneras de crear Api Rest y cada lenguaje tiene su propia manera de trabajar con ello, investigue los tres lenguajes más usados Java, Python y C# en el siguiente apartado detallare un poco más su funcionamiento, pero todos permiten las conexiones concurrentes.

5.1. ¿Qué lenguaje, framework o librería se pueden usar?

JAVA

- Framework: Spring
- Dependencias: Spring Boot, Spring Data, Spring Web, Swagger

- Ventaja: al ser un lenguaje muy estricto y muy estructurado el IDE detectara cualquier problema.
- Desventaja: la configuración inicial es muy larga y es necesario tener una buena base de programacion.

PYTHON

- Framework: FastApi
- Dependencias: SQLAlchemy, Alembic
- Ventaja: configuración inicial fácil y fácil de programar
- Desventaja: como no es un lenguaje fuertemente tipado y estructurado es mucho más facil cometer errores

C#

- Framework: .net
- Dependencias: asp.net, Swagger
- Ventaja: la configuración inicial es facil, lenguaje estructurado y tipado
- Desventaja: es necesario tener una buena base de programacion

5.2. ¿Qué elementos hardware y software necesitarías para desplegar la API REST?

Como diría Jose Francisco Márquez, nuestro profesor de DB de 1º, ¡depende!

A nivel de hardware dependerá del número de conexiones concurrentes que recibirá el servicio ademas de tener en cuenta la cantidad de consultas y/o datos que tendrá que gestionar.

Estos servicios se pueden desplegar tanto en “local”, si se dispone de un servidor, o en la nube, esta última es la más utilizada en la actualidad, empresas como Google con Google Cloud, Amazon con AWS o Microsoft con Azure, permiten su despliegue de manera rápida y relativamente sencilla.

Según el tipo de lenguaje necesitaremos como requisitos mínimos de software el compilador o interprete y las dependencias en la máquina que correrá el servicio.

6. Bibliografía

ASP.net. (s.f.). Obtenido de Microsoft: https://dotnet.microsoft.com/es-es/apps/aspnet

DeepL API Documentación. (s.f.). Obtenido de DeepL: https://developers.deepl.com/docs/getting-started/readme

Oracle. (s.f.). *Oracle JavaDoc 17*. Obtenido de https://docs.oracle.com/en/java/javase/17/docs/api/index.html

org.json. (s.f.). Obtenido de Maven Repository: https://mvnrepository.com/artifact/org.json/json/20231013

org.json. (s.f.). Obtenido de Git Hub: https://github.com/stleary/JSON-java

Public Apis. (s.f.). Obtenido de https://github.com/public-apis/public-apis

Rapid Api. (s.f.). Obtenido de https://rapidapi.com/hub

Spring.io. (s.f.). Obtenido de https://spring.io/

SQLAlchemy. (s.f.). Obtenido de SQLAlchemy: https://www.sqlalchemy.org/

7. Tabla de ilustraciones