



CENTRE D'ESTUDIS AULA CAMPUS
CICLO FORMATIVO DE GRADO SUPERIOR
DESARROLLO DE APLICACIONES WEB

Proyecto de Fin de Ciclo

Desarrollo de un BOT traductor de Telegram en C#

Marcos Sant Hernández

Tutor: Mari Cardells Cifres

Curso: 2020/2022

Contents

1. Introducción:.....	3
2. Justificación del proyecto:.....	3
3. Objetivos:.....	4
4. Desarrollo:.....	5
Análisis previo de mercado y necesidades:.....	5
Diagrama de Gantt (planificación del proyecto):.....	6
Herramientas o tecnologías (para qué la he usado y cómo se conecta con los objetivos):.....	6
Capturas de código y cómo se refleja en la aplicación:	7
Claves.cs:	7
Translator.cs:.....	8
Program.cs:.....	10
¿Cómo usar y acceder al bot?	14
5. Conclusiones:	15
6. Líneas de investigación futuras:.....	16
7. Webgrafía:	17

1. Introducción:

Siempre he visto la programación como algo más propio del mundo intangible que de la realidad, ya que es muy complicado explicar o visualizar lo que se hace si no se conoce el campo.

Por ejemplo, un carpintero hace una mesa, y está ahí, puedes sentarte o poner comida en ella. Pero cuando programamos a veces el resultado es que una web en lugar de 10 segundos, tarde solamente 1 segundo en cargar. O que ahora al hacer clic en los botones cambian de color o te muestran un pequeño texto indicándote qué es lo que ocurrirá si pulsas en él.

Esto es mucho más difícil de apreciar, y es por ello que quiero que mi PFG sea algo real, que pueda usarse y que tenga impacto en el día a día.

En esta misma línea he querido también utilizarlo como una oportunidad para aprender más, utilizando tecnologías actuales, con aplicaciones en varios campos y que me supusiera un reto.

Por último, he querido también desarrollar este proyecto con una mentalidad Open Source. Es decir, que cualquiera pueda replicarlo, adaptarlo a sus necesidades o simplemente utilizarlo como base para desarrollar otro tipo de bots o cualquier otra aplicación que se le encuentre.

Al comenzar este proyecto no conocía ni había usado C# o cualquiera de las tecnologías que menciono a continuación.

Las principales tecnologías que usaré en el proyecto son las siguientes:

- C#
- .NET Core
- API REST
- Telegram.Bot
- The BotFather
- JSON

2. Justificación del proyecto:

Desde que vine a vivir a Tenerife en 2021 he estado involucrado en diferentes iniciativas de voluntariado y proyectos en la isla. En todos ellos la mezcla cultural era muy grande, con gente de todas partes del mundo, y esto daba lugar a un gran problema. La comunicación.

Mi objetivo es solucionar este problema, desarrollando para ello una herramienta que lo haga posible.

Durante el estudio previo a la realización del proyecto encontré un inconveniente, y es que este problema ya está solucionado.

Con aplicaciones tan sencillas como google Translator, Microsoft Translator, etc... puedes traducir un texto a otros idiomas en segundos, y además las traducciones son muy fiables y optimizadas (de hecho, usaré la API de Microsoft Translator para el proyecto).

Aun así, quería algo más directo, más sencillo, óptimo y rápido. Y para ello pensé en incluir la traducción no en una app externa, sino dentro de las propias apps que utilizamos para comunicarnos a diario, como WhatsApp o Telegram.

Por último, queda decidir el idioma, y teniendo en cuenta el contexto global en el que nos movemos actualmente, ha sido relativamente sencillo. Y es que el inglés desde hace varios años es el lenguaje “por defecto” a aprender como segunda lengua, tanto para comunicarnos con desconocidos, como para trabajar en grandes empresas e incluso para viajar al extranjero.

Mi objetivo final es, por tanto, el desarrollo de un bot de Telegram que traduzca de español a inglés, aprendiendo en el proceso el lenguaje C# y ayudando a solucionar los problemas de comunicación entre personas.

Considero que este proyecto puede ser de gran utilidad tanto en un entorno social, como en un entorno laboral, y sus aplicaciones son infinitas.

Desde agilizar el contacto con alguien que has conocido en un viaje, a hablar con tu jefe alemán con el que solo hablas en inglés o incluso para ayudar a la inserción de inmigrantes en un nuevo país del cual no conocen la lengua.

3. Objetivos:

- Desarrollar una aplicación funcional y útil:

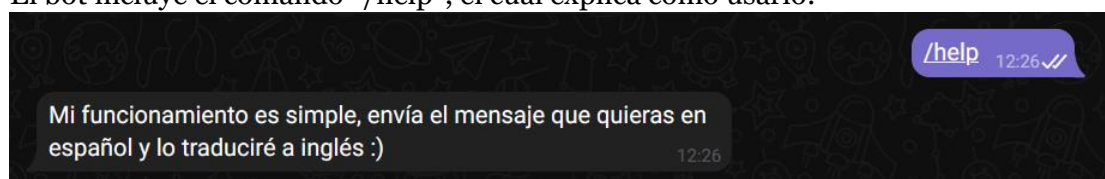
Desarrollar una aplicación capaz de traducir textos de diferentes longitudes e idiomas de manera eficiente y precisa. Utilizando además para ello recursos gratuitos a los que cualquiera tenga acceso.

La opción elegida es la API de Microsoft Translator. Más información sobre cómo funciona [aquí](#).

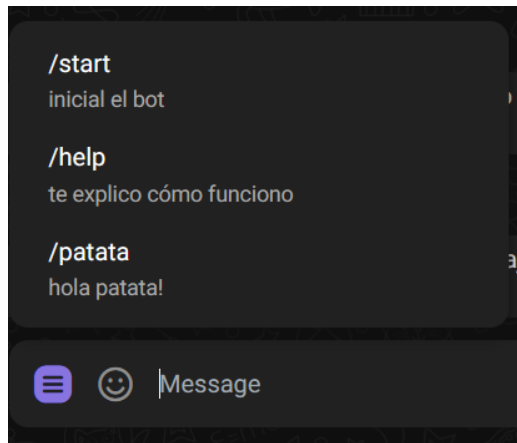
- Implementarla de manera que cualquiera pueda utilizarla:

Facilitar su uso a través de una interfaz sencilla que permita utilizarla a cualquiera. Para este propósito utilizaré Telegram como interfaz del bot, haciendo su uso lo más fácil posible.

El bot incluye el comando “/help”, el cual explica cómo usarlo:



Además, el bot incluye un panel lateral con los diferentes comandos posibles, se despliega al hacer clic:



- Solucionar un problema real:

Dar facilidades para la comunicación entre personas que hablan diferente lengua y que pueda aplicarse en diferentes situaciones (laboral, social...)

- Desarrollar en código abierto y reutilizable:

Desarrollar la aplicación con herramientas y tecnologías públicas y gratuitas para que cualquier pueda tanto replicarlo, como modificarlo o usarlo sin ningún coste, alcanzando así al mayor número de personas y consiguiendo el mayor impacto posible.

- Publicar el proyecto para su uso:

Otro de los objetivos del proyecto es publicar el código en [GitHub](#), ya que no existe nada similar en internet y creo que puede ser de gran utilidad no solo para facilitar la comunicación entre personas, sino también en entornos como escuelas o institutos para aprender nuevos idiomas u otros propósitos que yo no he tenido en cuenta.

De esta manera el código es *público* y su uso *gratuito*.

- Establecer unas bases sólidas en un nuevo lenguaje de programación:

Aprovechar el desarrollo de este proyecto para aplicar los conocimientos de C# y otras tecnologías aprendidas durante las FCTs.

4. Desarrollo:

Análisis previo de mercado y necesidades:

Actualmente existen varias opciones para traducción de texto, la más usada y conocida es Google Translate. Este servicio y aplicación de Google es gratuito para los usuarios y funciona notablemente bien.

A pesar de ello considero que podría ser aún más simple, y ese es el objetivo de este proyecto. Incluir en una app de mensajería, una opción simple, directa e instantánea de traducción de texto. Sin necesidad de abrir una app externa.

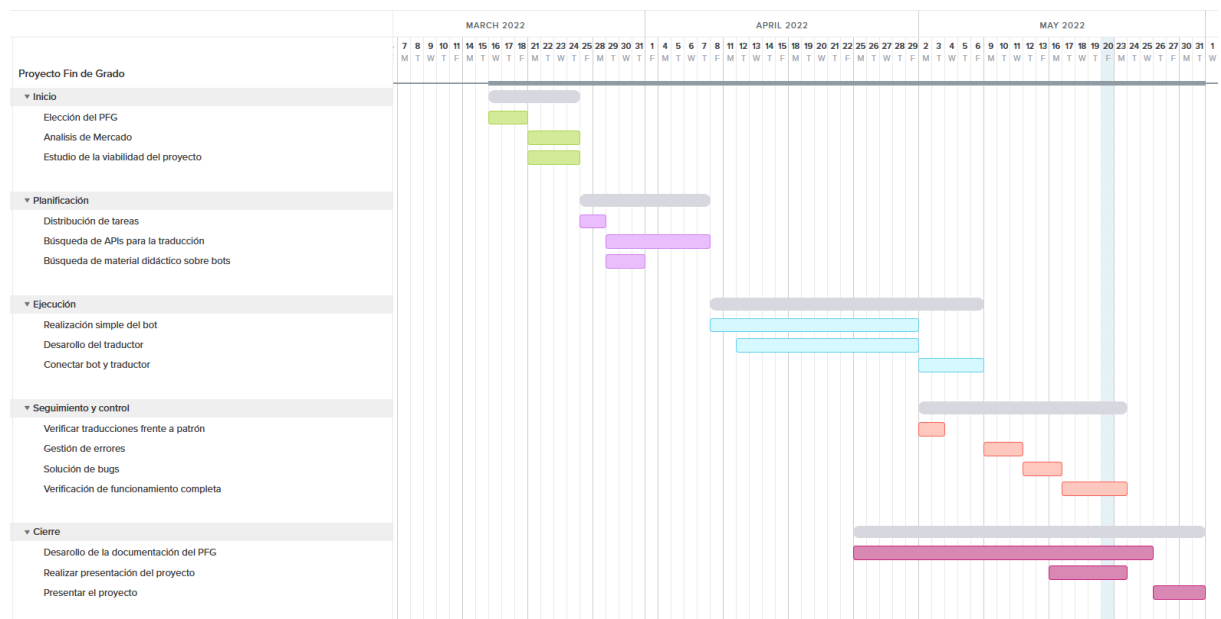
He valorado las apps de mensajería más utilizadas actualmente, datos obtenidos de [informabl](#):

- WhatsApp: esta aplicación es la más usada actualmente, con más de 1.500 millones de usuarios. A pesar de ello cuenta con una API muy restrictiva, de poco uso y encima muy complejo. Además, no cuenta con ningún tipo de apoyo a la hora de construir bots o interactuar con ellos.

- Messenger: 1.300 millones de usuarios. Esta aplicación de chat incluida en la plataforma de Facebook es la segunda más usada en el mundo. Sigue la misma línea que WhatsApp (pertenecen a la misma compañía) y no existen APIs o librerías para facilitar el desarrollo de bot externos en la misma, de hecho, están prohibidos según sus propios términos de uso.
- Telegram: Comparada con las anteriores, los 200 millones de usuarios hacen parecer a Telegram pequeña. Pero esta aplicación de mensajería además de proteger los datos de los usuarios es muy amigable (friendly) con los desarrolladores, dando facilidades en varios aspectos. No solo para implementar bots, sino para interactuar con ella en general, pudiendo enviar no solo mensajes de texto, sino también gifts, imágenes, vídeos y notas de voz desde el propio código de la app... De hecho, es por estas facilidades, junto con su seguridad y su buen funcionamiento, lo que la convierte en la app por defecto en la comunicación en el mundo de la informática.
- Otras: Existen otras aplicaciones más usadas que Telegram, como son Skype, Snapchat, Instagram... Pero he considerado que su uso no se ajusta al propósito del proyecto y por tanto las he descartado.

Diagrama de Gantt (planificación del proyecto):

He desarrollado el diagrama en la web <https://www.teamgantt.com/> y lo he ido adaptando a medida que el proyecto avanzaba, se adjunta en PDF.



El proyecto ha requerido más de 100h para su desarrollo, distribuidas a lo largo de dos meses y medio. Este diagrama representa el tiempo real destinado a cada uno de los diferentes apartados. Creía que iba a ser un proyecto más sencillo, pero la complejidad de los diferentes apartados ha resultado ser mucho mayor de lo esperado.

Herramientas o tecnologías (para qué la he usado y cómo se conecta con los objetivos):

Tecnologías utilizadas:

- C#
- .NET Core
- API REST
- Telegram.Bot
- The BotFather
- JSON

C#, junto con .NET Core han sido el lenguaje a utilizar para el desarrollo de la aplicación. He utilizado el IDE Microsoft Visual Studio 2022 para ello.

Por otro lado, la librería Telegram.bot junto con The BotFather, el bot para crear otros bots en Telegram, han sido necesarias para configurar mi propio bot y para poder comunicarme con el usuario y recibir instrucciones (de otra manera habría que hacerlo por la consola).

Por último, la API (application programming interface) de Microsoft Translate, es el medio por el cual se consigue traducir el texto de un idioma a otro y capturar dicha respuesta para devolvérsela al usuario en forma de texto. JSON es el formato en el que se gestionan los datos entre diferentes aplicaciones o partes de la aplicación, que se traducen posteriormente a variables con las que sí se puede trabajar (por ejemplo, un string) para devolverlas al usuario.

Capturas de código y cómo se refleja en la aplicación:

La aplicación consta de 3 clases:

Claves.cs:

Esta clase se encarga de la lectura de las claves almacenadas en archivos de texto (.txt).

```
public class Claves
{
    2 references
    public string botToken
    { get; private set; }
    2 references
    public string translatorKey
    { get; private set; }
    2 references
    public string translatorEndpoint
    { get; private set; }

    3 references
    public Claves()
    {
        botToken = File.ReadAllText(@"SecretBotToken.txt");
        translatorKey = File.ReadAllText(@"Key.txt");
        translatorEndpoint = File.ReadAllText(@"Endpoint.txt");
    }
}
```

He creído necesario gestionar de esta manera las claves y tokens ya que son únicos, es decir: Tanto el token para iniciar el bot de Telegram, como la key para el uso de la API

de Microsoft Translator están asociados conmigo, por tanto, son personales y su uso me corresponde a mí.

Al subir este código a internet si hubiera incluido en el propio código del programa dichas claves cualquiera tendría acceso a ellas, pero de esta manera puedo mantener dichos archivos en privado y que nadie tenga acceso, pero manteniendo el resto del código como público.

Translator.cs:

Esta clase se encarga de la traducción del texto.

Lo primero es autorizar el uso de la API, para ello facilitamos las claves desde la clase Clase.cs en su construcción:

```
1 reference
internal class Translator
{
    private static readonly string location = "westeurope";
    private static readonly string key = new Claves().translatorKey;
    private static readonly string endpoint = new Claves().translatorEndpoint;

    public string textToTranslate;
    public string route;
```

Esta clase sólo tiene un método, el cual recibe tres parámetros de entrada. El primero es el texto a traducir (*textToTranslate*), el segundo indica el idioma desde el cual se hará la traducción (*languageFrom*), y por último el idioma al que se va a traducir el texto (*languageTo*):

```
1 reference
public async Task<string> translate(string textToTranslate, string languageFrom = "es", string languageTo = "en")
{
    try
    {
        this.textToTranslate = textToTranslate;
```

Como se puede observar en el código, los idiomas a los que se va a traducir y desde el que se va a traducir son español (es) e inglés (en).

Estos son los valores por defecto, pero el código ya está estructurado con la intención de que en un futuro se pueda implementar la traducción desde otros idiomas que no sean el español a otros idiomas que tampoco sean el inglés.

Además, como se ve en la captura, el método es “async”, esto significa que es asíncrono.

Utilizando esta propiedad de C#, múltiples métodos de traducción pueden ejecutarse a la par en diferentes hilos de ejecución, siendo esta la solución a que varios usuarios usen el bot al mismo tiempo, ya que de otra forma daría error o cada usuario tendría que esperar a que el anterior haya recibido su texto traducido antes de recibir el suyo propio.

Imaginemos utilizar google Translate y tener que esperar a que todas las traducciones que se han hecho antes que la nuestra terminen para que nos toque el turno. Esto llevaría muchísimo tiempo y sería muy ineficiente.

Una vez enviado al método el texto y los idiomas, esta información se manda a la API de traducción de Microsoft:


```
// Input and output languages are defined as parameters.
this.route = $"/translate?api-version=3.0&from={languageFrom}&to={languageTo}";
Console.WriteLine(this.route);

object[] body = new object[] { new { Text = textToTranslate } };
var requestBody = JsonConvert.SerializeObject(body);

using (var client = new HttpClient())
using (var request = new HttpRequestMessage())
{
    // Build the request.
    request.Method = HttpMethod.Post;
    request.RequestUri = new Uri(endpoint + route);
    request.Content = new StringContent(requestBody, Encoding.UTF8, "application/json");
    request.Headers.Add("Ocp-Apim-Subscription-Key", key);
    request.Headers.Add("Ocp-Apim-Subscription-Region", location);

    // Send the request and get response.
    HttpResponseMessage response = await client.SendAsync(request).ConfigureAwait(false);
}
```

Una vez la Api ha validado que nuestra key es correcta, utiliza un servidor interno de Azure (la nube de Microsoft) el cual tiene instalado este traductor para devolver el texto traducido.

Más información sobre cómo funciona este servicio o sobre cómo crear otro servidor [aquí](#).

Existen varias opciones para este servicio, yo he escogido la opción gratuita, que incluye la traducción de un máximo de 2 millones de caracteres al mes.

Si todo este proceso es correcto, leemos el contenido de la variable response para obtener los datos de la traducción:

```
// Read response as a string.
var translationObject = await response.Content.ReadAsStringAsync();
Console.WriteLine(translationObject);
```

Esta variable ahora mismo contiene todos los datos de la traducción, desde los idiomas que hemos elegido (se puede traducir a varios idiomas diferentes también), la cantidad de caracteres que hemos enviado, el texto traducido, etc. Es decir, muchísima información que no nos interesa. De manera que toda esta información se traduce a un JSON para darle estructura y poder acceder a todos estos datos de manera ordenada:

```
// Read response as a string.
var translationObject = await response.Content.ReadAsStringAsync();
Console.WriteLine(translationObject);

var JsonTranslation = JObject.Parse(translationObject);
Console.WriteLine(JsonTranslation);
```

Y por último se devuelve el texto traducido:

```
//Return of the translated text
return (string)JsonTranslation[0]["translations"][0]["text"];
}
```

En caso de que ocurra un error, al estar todo ello dentro de un try catch, ejecutaría el siguiente código:

```
}  
catch (Exception ex)  
{  
    Console.WriteLine(ex);  
    return "Ha ocurrido un error";  
}
```

Mostrando por consola el error (en local) y devolviendo al usuario del bot en Telegram el texto “Ha ocurrido un error”, evitando así que se detenga el programa y dándonos la opción de ver qué error es y más información del mismo para poder solucionarlo.

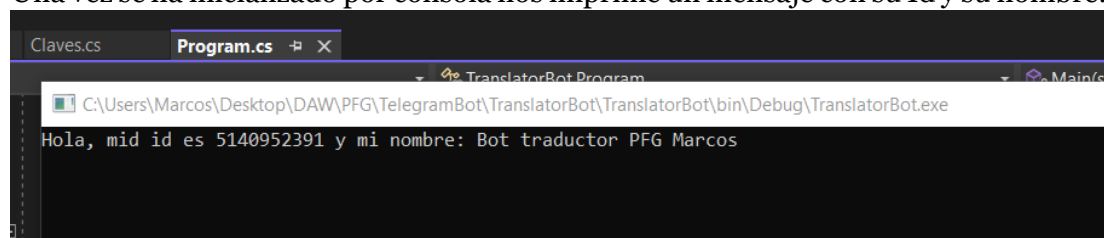
Program.cs:

Esta clase contiene el Main, y es la que ejecuta el programa:

```
static void Main(string[] args)  
{  
    string token = new Claves().botToken;  
  
    //Starting the bot  
    botClient = new TelegramBotClient(token);  
    var meBot = botClient.GetMeAsync().Result;  
  
    Console.WriteLine($"Hola, mid id es {meBot.Id} y mi nombre: {meBot.FirstName}");  
  
    //Checks every new message  
    botClient.OnMessage += BotClient_OnMessage;  
    botClient.StartReceiving();  
    Console.ReadKey();  
    botClient.StopReceiving();  
}
```

En él se crea el cliente del bot de Telegram, utilizando el token único de la clase Claves.cs para inicializarlo.

Una vez se ha inicializado por consola nos imprime un mensaje con su Id y su nombre:



```
Claves.cs Program.cs x  
TranslatorBot Program  
C:\Users\Marcos\Desktop\DAW\PFG\TelegramBot\TranslatorBot\TranslatorBot\bin\Debug\TranslatorBot.exe  
Hola, mid id es 5140952391 y mi nombre: Bot traductor PFG Marcos
```

Esta es una manera sencilla de comprobar que la aplicación se ha iniciado correctamente y que conecta con éxito con el cliente de Telegram (sino no podría devolver su Id ni nombre).

A continuación, comprueba de manera constante si se ha recibido un nuevo mensaje, y en caso de recibirlo se ejecuta el método BotClient_OnMessage:

```
private static void BotClient_OnMessage(object sender, Telegram.Bot.Args.MessageEventArgs e)
{
    var id = e.Message.Chat.Id;
    var text = e.Message.Text;
    var from = e.Message.From.FirstName;

    Console.WriteLine($"He recibido: {text} de {from}");

    string message = "";
}
```

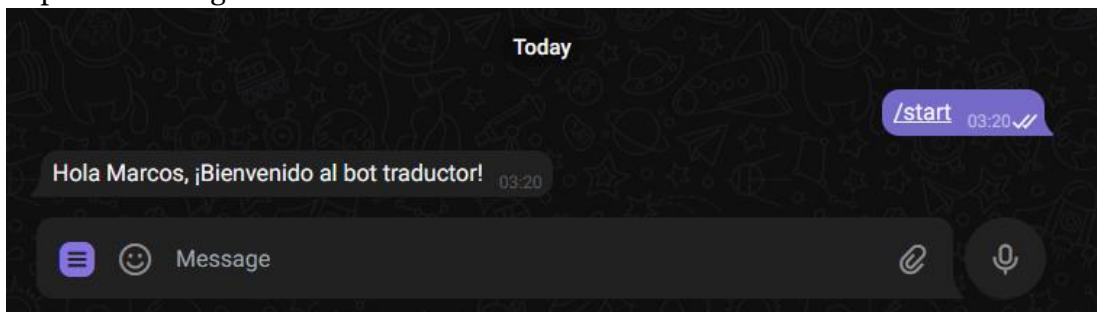
Este método define unas variables comunes a todos los mensajes. Estas son:

- Id del chat del que se ha recibido el mensaje.
- Texto del mensaje.
- Nombre público de Telegram de la persona que ha enviado el mensaje.

Estas variables van a ser comunes a todos los casos posibles que ocurran a continuación. Una vez guardadas se imprime por pantalla el mensaje que se ha recibido y el nombre de la persona que lo ha mandado, así como el mensaje que se envía de vuelta por Telegram. Para este ejemplo he enviado el mensaje “/start” para iniciar el bot:

```
Hola, mid id es 5140952391 y mi nombre: Bot traductor PFG Marcos
He recibido: /start de Marcos
Hola Marcos, ¡Bienvenido al bot traductor!
```

Captura de Telegram:



A continuación, el método contiene un switch, el cual evalúa los diferentes casos posibles para determinar qué acción ejecutar. Dichos casos son:

- Comando “/patata”: Es un comando de prueba para verificar el funcionamiento del bot. El bot devuelve “Hola patata” y no se imprime nada por pantalla.



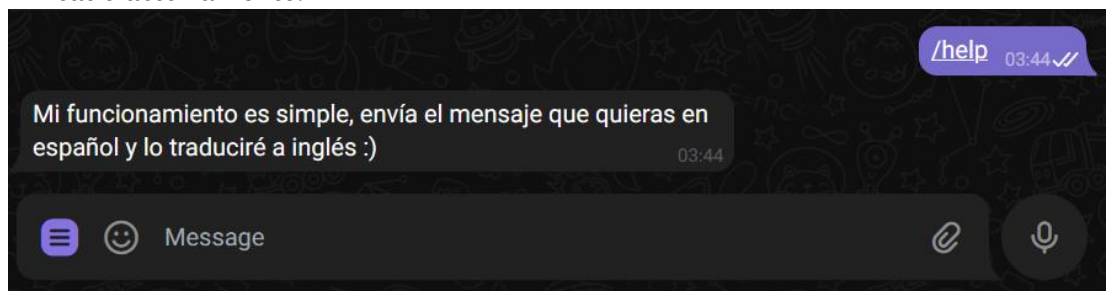
- Comando “/start”: Este comando es **obligatorio** para iniciar el bot, si no se envía el bot no tendrá permisos para comunicarse contigo. Esto es debido a las

medidas de seguridad de Telegram para evitar el spam indeseado de bots a usuarios.

Cuando se recibe este comando el bot devuelve un mensaje de bienvenida, reflejado en la captura anterior, con el nombre de la persona que ha enviado el mensaje.

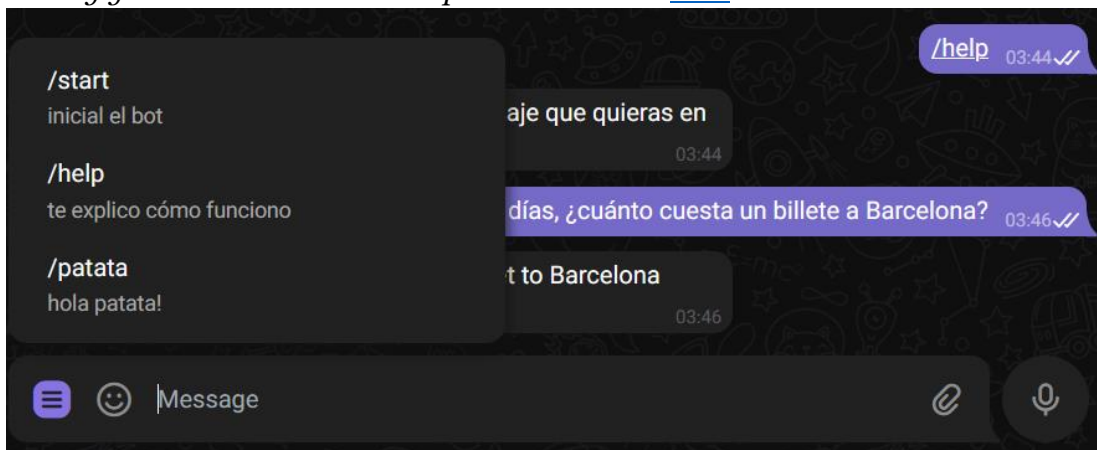
También se imprime en la consola el mensaje enviado.

- Comando “/help”: Este comando, como se intuye por su nombre, ayuda al usuario a entender el funcionamiento del bot. Actualmente envía un mensaje simple, explicando que el texto que envíes será traducido al inglés. En caso de añadir funcionalidades al bot, como otros idiomas, u otras funciones, explicará cómo usar cada una de ellas, ayudando así al usuario a entender el bot y usarlo satisfactoriamente.



Por último, imprime por consola el mensaje enviado al usuario (no incluyo captura porque es redundante).

Nota: para cada uno de estos comandos he configurado su propio texto explicativo dentro de la interfaz de Telegram, se despliega haciendo clic en el botón azul inferior izquierdo con las 3 líneas horizontales. Esto ha sido configurado utilizando la herramienta TheBotFather, que es, por decirlo así, el bot maestro de Telegram para crear y gestionar los bot creados por los usuarios. [Link](#).

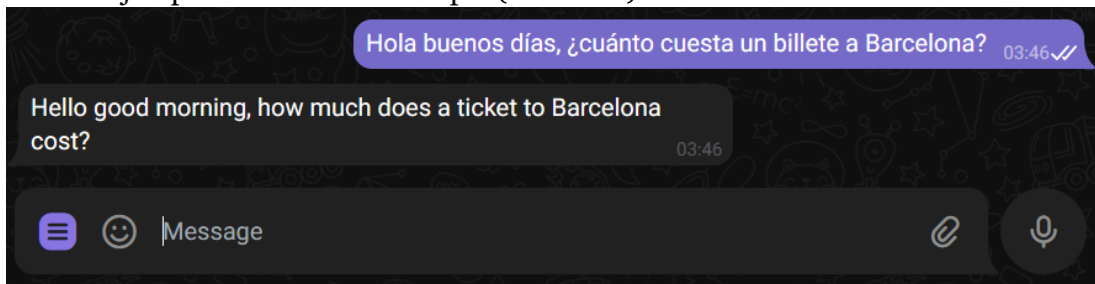


- “Default”: En caso de no recibir ninguno de los comandos anteriores, se ejecuta la sentencia default. En esta situación se crea una instancia de la clase traductor para generar la traducción del texto. Se envía el texto a traducir como parámetro (un string) y se recoge la Task<string> enviada de vuelta cuando esté lista, almacenándola en la variable “traducción”. Recordemos que no sabemos lo que va a tardar la traducción en la API, ya que un usuario puede enviar una frase, y otro, varios párrafos. Para gestionar esta situación se hace uso de la programación asíncrona.

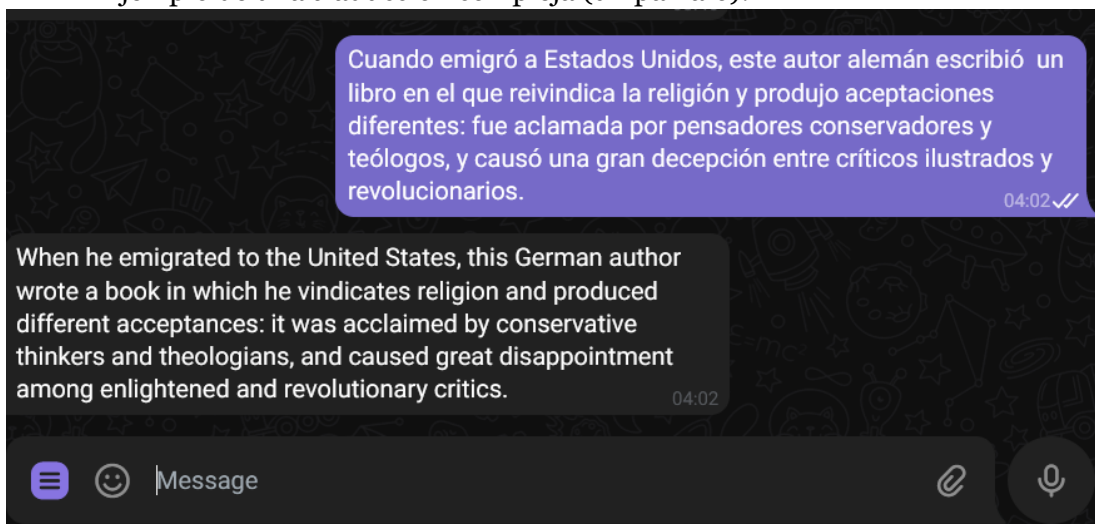
```
//If it is not a command the text is translated and returned
default:
    var traductor = new Translator();
    var translation = traductor.translate(text);
    botClient.SendTextMessageAsync(id, translation.Result);
    break;
}
```

Una vez recibida la traducción se envía de vuelta al usuario.

Ejemplo de traducción simple (una frase):



Ejemplo de una traducción compleja (un párrafo):



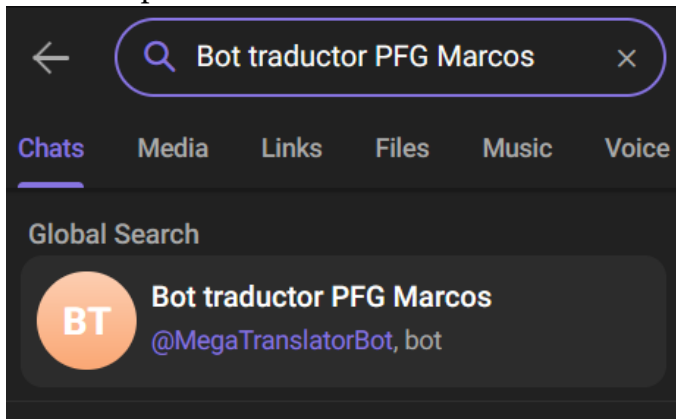
También se imprime por pantalla desde la clase Translator el JSON asociado a dicha traducción, incluyendo datos como el idioma a traducir y el idioma desde el que se traduce, el texto inicial y el final:

```
/translate?api-version=3.0&from=es&to=en
[{"translations":[{"text":"When he emigrated to the United States, this German author wrote a book in which he vindicates religion and produced different acceptances: it was acclaimed by conservative thinkers and theologians, and caused great disappointment among enlightened and revolutionary critics.", "to":"en"}]}]
[
  {
    "translations": [
      {
        "text": "When he emigrated to the United States, this German author wrote a book in which he vindicates religion and produced different acceptances: it was acclaimed by conservative thinkers and theologians, and caused great disappointment among enlightened and revolutionary critics.",
        "to": "en"
      }
    ]
  }
]
```

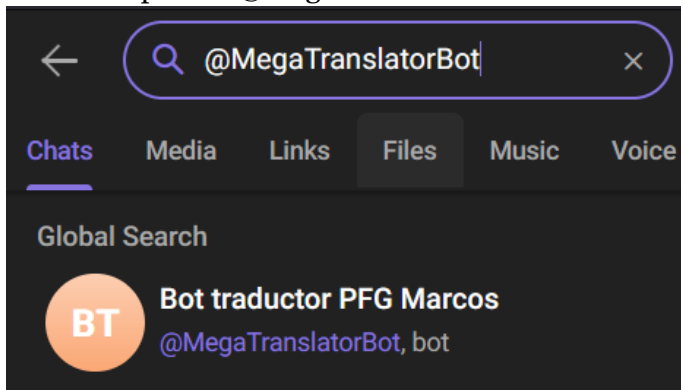
De toda esta información solo el texto final traducido es devuelto al programa principal para su envío al usuario.

¿Cómo usar y acceder al bot?

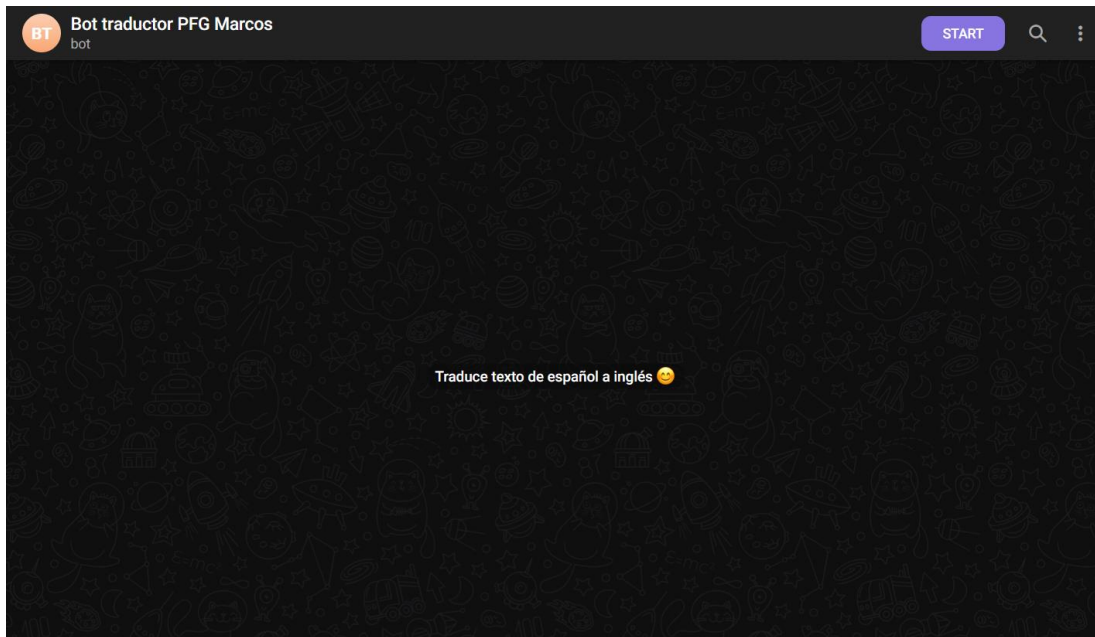
Para hacer uso del bot lo primero es ejecutar el programa. Una vez iniciado sin fallos hay que buscarlo en Telegram. Tanto el nombre como el ID son válidos: Buscando por nombre “Bot traductor PFG Marcos”:



Buscando por id “@MegaTranslatorBot”:



Una vez encontrado se abre y el bot te dará las instrucciones sobre su uso. Si es la primera vez el bot solo da la opción de iniciarlo:



No se puede enviar texto o interactuar con él debido al protocolo de seguridad de Telegram antes mencionado. Una vez hagamos clic en START se envía el comando /start y el bot funciona como ya hemos visto antes.

5. Conclusiones:

El desarrollo de este proyecto me ha enseñado varias cosas. La principal es que a simple vista las aplicaciones y proyectos informáticos parecen simples, pero no tienen por qué serlo. En el caso concreto de este proyecto, al que se supone hay que destinarle 40h, he tenido que dedicarle más de 100, debido a los diferentes problemas encontrados.

En cuanto al proyecto en sí mismo estoy satisfecho con la funcionalidad y utilidad de este, aunque hay varias áreas de mejora, las cuales mencionaré a continuación.

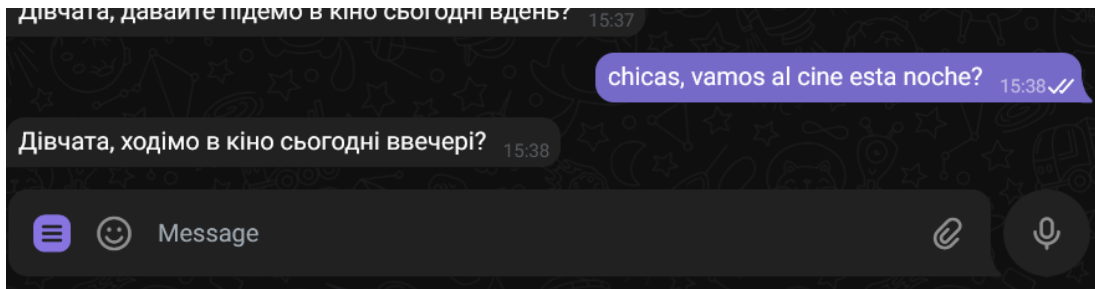
Considero que un traductor en este formato sería algo muy útil de implementarse más profesionalmente (en un servidor).

A nivel personal he creado otro bot idéntico y cambiado los idiomas de inglés a ucraniano, ya que mi familia está hospedando a una familia de refugiadas ucranianas y no hablan ningún otro idioma que no sea ucraniano o ruso.

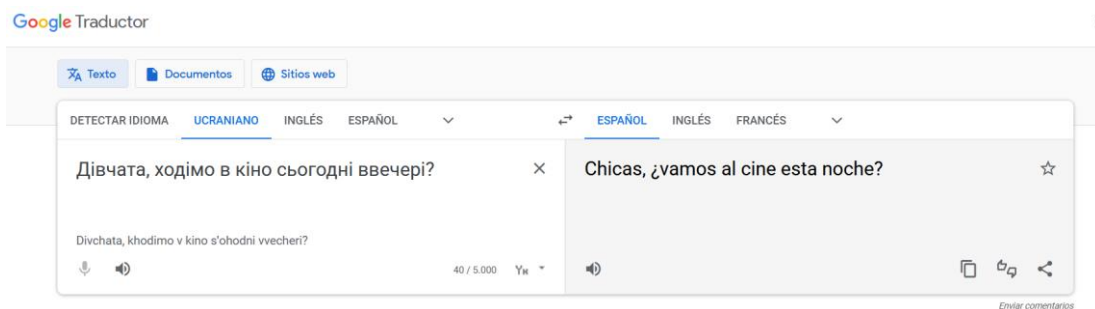
Gracias a lo simple del funcionamiento y que está incluido dentro de la propia app de mensajería (Telegram) la usamos a diario para comunicarnos con ellos y ellos para comunicarse con nosotros (he tenido que crear otro bot que traduzca de ucraniano a español para ellos).

Por tanto, ha probado ser útil en una situación real y ese era el objetivo de la misma.

Captura de la traducción a ucraniano:



Lo he comprobado con [GoogleTranslate](https://www.google.com/traductor) para que se vea el significado:



6. Líneas de investigación futuras:

Considerando la naturaleza del proyecto (un traductor), las posibles mejoras futuras más obvias y relevantes incluyen la traducción:

1. Desde otros lenguajes que no sean el español
2. A otros lenguajes que no sean el inglés

Estas posibles mejoras ya están planeadas en el código para una fácil implementación. Como se ve en la captura a continuación, los parámetros que recibe la clase `Translator` incluyen los strings `lenguajeFrom` y `lenguajeTo`:

```
1 reference
public async Task<string> translate(string textToTranslate, string lenguajeFrom = "es", string lenguajeTo = "en")
{
    try
    {
        this.textToTranslate = textToTranslate;
    }
}
```

Por defecto dichos valores son español e inglés, pero la API admite más de 100 idiomas diferentes (se pueden consultar en detalle [aquí](#)) y se podría preguntar al usuario desde qué idioma se traduce y a qué idioma se desea traducir.

Otra posible línea de mejora es la creación de una base de datos para guardar las traducciones, de manera que las consultas duplicadas no tengan que ser enviadas a la API y traducidas, sino que directamente se envíen desde esta base de datos.

De esta manera se ahorra tiempo y dinero, ya que las traducciones tienen un coste asociado. Actualmente estoy usando una licencia que permite traducir hasta 2 millones de caracteres mensuales de manera gratuita, a partir de ahí pasan a ser de pago.

7. Webgrafia:

- <https://app.pluralsight.com/library/courses/csharp-fundamentals-dev/table-of-contents>
- <https://core.telegram.org/bots>
- <https://github.com/TelegramBots/Telegram.Bot>
- <https://telegrambots.github.io/book/index.html>
- <https://www.delftstack.com/howto/csharp/how-to-parse-json-in-csharp/>
- <https://docs.microsoft.com/en-us/azure/cognitive-services/translator/>
- <https://app.pluralsight.com/library/courses/applying-asynchronous-programming-c-sharp/table-of-contents>
- <https://docs.microsoft.com/en-us/azure/cognitive-services/Translator/language-support>
- <https://core.telegram.org/bots>
- <https://www.teamgantt.com/>