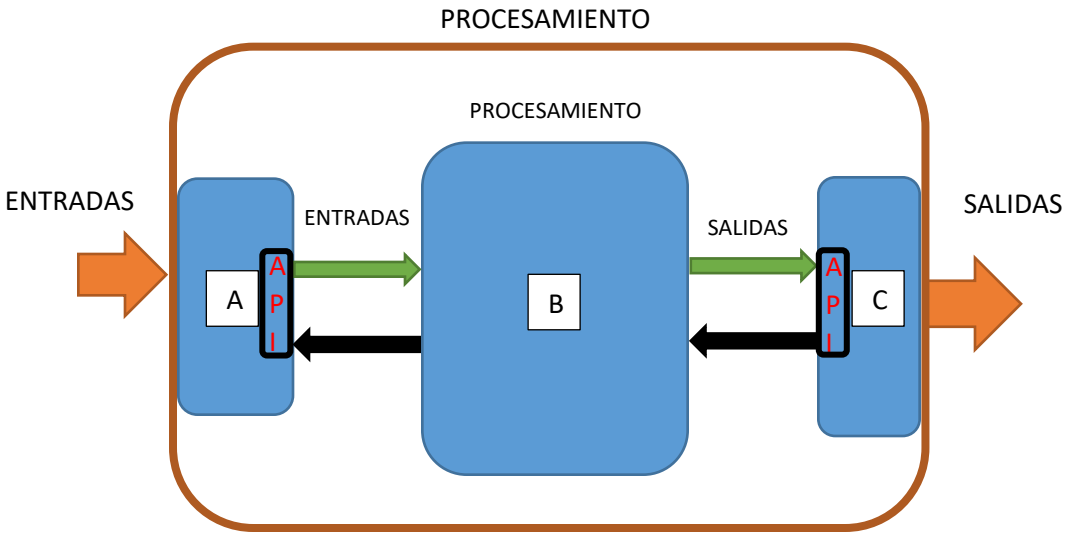
Automatización de procesos e integración con otras aplicaciones a través del uso de API's.

Introducción

Conceptualmente, a todo sistema electrónico se lo puede representar con un diagrama en bloques donde hay tres partes principales: las entradas, el procesamiento de la información capturada, y las salidas. Las entradas pueden por ejemplo ser señales provenientes de sensores de temperatura que suministran al sistema información que es procesada luego de diferentes formas a través de circuitería dedicada para tal fin o de forma genérica a través de un sistema basado en microprocesador; como consecuencia de ese procesamiento se alcanzan resultados que se pueden visualizar a través de las salidas como por ejemplo diodos emisores de luz (LEDs). Este enfoque sistémico puede aplicarse también a las sub-partes que constituyen el sistema principal y por lo tanto tendremos un diagrama en bloques consistente de varios diagramas en bloques más pequeños e interconectados tal como lo muestra la figura siguiente.



Si suponemos ahora que el bloque A maneja sensores de temperatura, pero en forma remota, es decir que A puede enviar el valor de la temperatura del sitio que se le solicite y físicamente el bloque A no se encuentra en el mismo lugar que B. Las flechas verde y negra reflejan esta interacción entre A y B que es por ejemplo a través de internet. Para facilitar esta interacción, A ofrece una serie de "comandos", "funciones" o "servicios" que pueden entregarse ante el requerimiento de B y como son a través de internet se los lleva a cabo por medio del protocolo HTTP que es lo natural en la web; a este conjunto de rutinas, protocolos y herramientas para crear aplicaciones de software se las denomina API (**A**pplication **P**rogramming **I**nterfaces). Por lo general, se utilizan para exponer datos o servicios que otras aplicaciones pueden consumir. Dependiendo de su propósito, las API pueden usarse internamente, con socios o ponerse a disposición del público en general.

Del mismo modo que con la entrada, el bloque B que lleva a cabo el procesamiento de los datos capturados, desea visualizar los resultados alcanzados enviando por ejemplo un mensaje al celular del

cliente en lugar de o además de encender los leds. Con el mismo concepto desarrollado anteriormente podemos seleccionar una aplicación de mensajería tipo Telegram, la cual ofrece un conjunto de servicios tal como enviar un mensaje a través del uso de una de sus API.

Lo que sigue a continuación en este documento, es una descripción detallada de cómo acceder a diferentes sitios web que suministran APIs y hacer uso de las mismas efectuando los requerimientos necesarios y obteniendo los resultados deseados desde un programa escrito en el lenguaje de programación C.

¿Cómo acceder a las API´s?

Existen innumerables sitios web que ofrecen una variada cantidad de API's de diferentes temáticas tanto de carácter público como privadas. Las primeras no necesitan ningún trámite previo para accederlas mientras que las segundas por lo general necesitan una registración y obtención de una clave propia del cliente registrado que debe incluirse en todos los requerimientos que se efectúen para obtener satisfactoriamente los datos buscados. Esta registración suele tener acceso gratuito, aunque limitado en algunos casos e ilimitado en otros y en la mayoría de los sitios web es con un costo que varía de acuerdo a los beneficios que se obtienen.

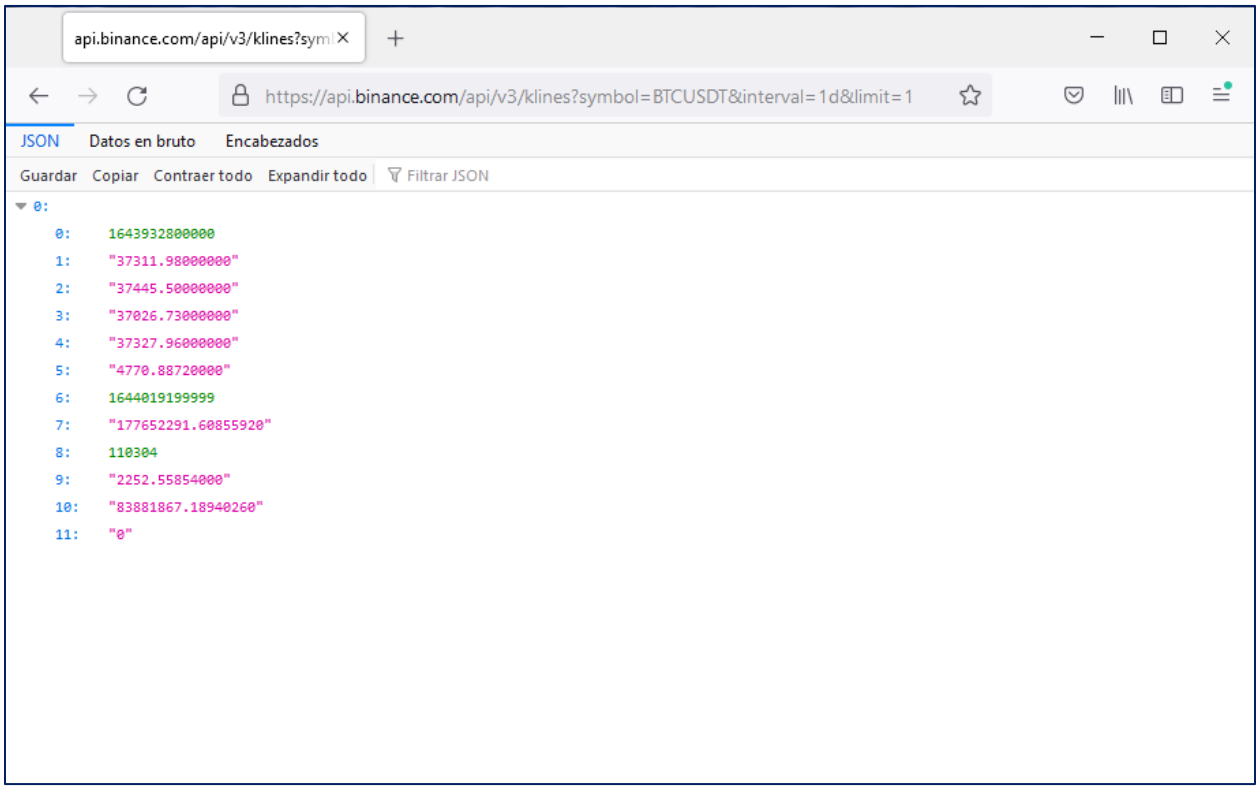
Si queremos por ejemplo acceder al precio del bitcoin, el Exchange Binance ofrece una API sin ningún costo ni registración previa en la que devuelve, ante el requerimiento, la información que se necesita para trazar un gráfico de velas [FIG1]. Estos datos suministrados por el sitio web vienen empaquetados en el formato JSON [2]. Toda la interacción entre la computadora desde donde se solicita el pedido y el sitio que la provee es a través de la red y con el protocolo HTTP. Una principal herramienta que gestiona este tipo de transacciones es el navegador web; existen varios entre los que se destaca el Mozilla Firefox. A través de ellos, como usuarios podemos invocar un sitio web indicándolo en la barra de direcciones y

recibiendo en el área de trabajo el contenido del sitio en formato HTML que el navegador se encargará de visualizar. Hasta ahí lo más normal que realizamos diariamente. El navegador o browser se encargará básicamente de enviar un requerimiento GET por HTTP y a través del mismo protocolo recibir la respuesta.

Cuando se accede a un sitio que suministra API's, el requerimiento necesita información adicional a la dirección web del sitio, la cual se suministra en la misma barra de direcciones del navegador, a continuación de la dirección web y en el formato que lo requiera la API correspondiente. Veamos a título de ejemplo el acceso al precio del bitcoin en Binance. Con la siguiente línea en el navegador

https://api.binance.com/api/v3/klines?symbol=BTCUSDT&interval=1d&limit=1

se obtiene:



Según la documentación oficial de Binance que se encuentra en:

https://github.com/binance/binance-spot-api-docs/blob/master/rest-api.md

seleccionando Kline/Candlestick data obtenemos la explicación del formato JSON (formato de texto sencillo para el intercambio de datos) regresado:

```
[
  [
    1499040000000,      // Open time (Tiempo de apertura en milisegundos)
    "0.01634790",       // Open (Precio de apertura del Mercado de ese día)
    "0.80000000",       // High (Precio máximo obtenido)
    "0.01575800",       // Low  (precio mínimo obtenido)
    "0.01577100",       // Close (Precio de cierre del Mercado de ese día)
```

```
  "148976.11427815",  // Volume
  1499644799999,      // Close time
  "2434.19055334",    // Quote asset volume
  308,                // Number of trades
  "1756.87402397",    // Taker buy base asset volume
  "28.46694368",      // Taker buy quote asset volume
  "17928899.62484339" // Ignore.
 ]
]
```

Si observamos la información adicional, estamos solicitando información del precio del bitcoin(BTC) en dólares (USDT), lo que registró en el día (interval=1d) y con limit=1 le pedimos que sea solo el último. De no hacerlo nos suministrará los últimos 500 por defecto y como máximo 1000. Por supuesto que toda esta información puede cambiarse para conseguir el par Ethereum/bitcoin (symbol=ETHBTC) y cada 5 minutos (interval=5m) y así sucesivamente para lo que se desee obtener y de ese modo poder trazar el gráfico de velas correspondiente donde se muestra la tendencia del precio.

[FIG1] Gráfico de velas.

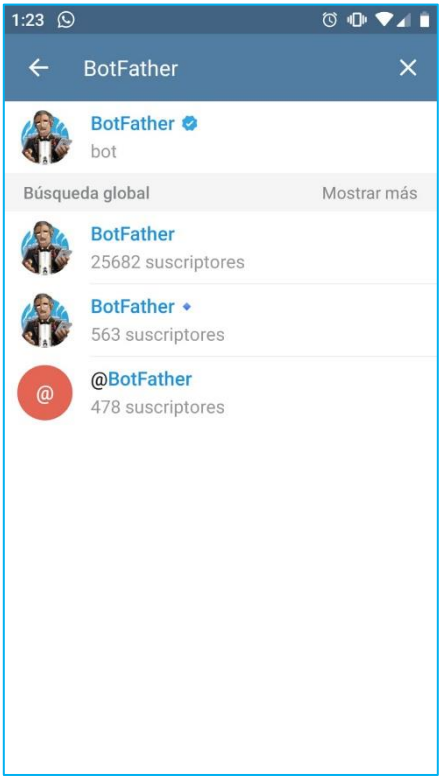
¿Cómo acceder a Telegram?

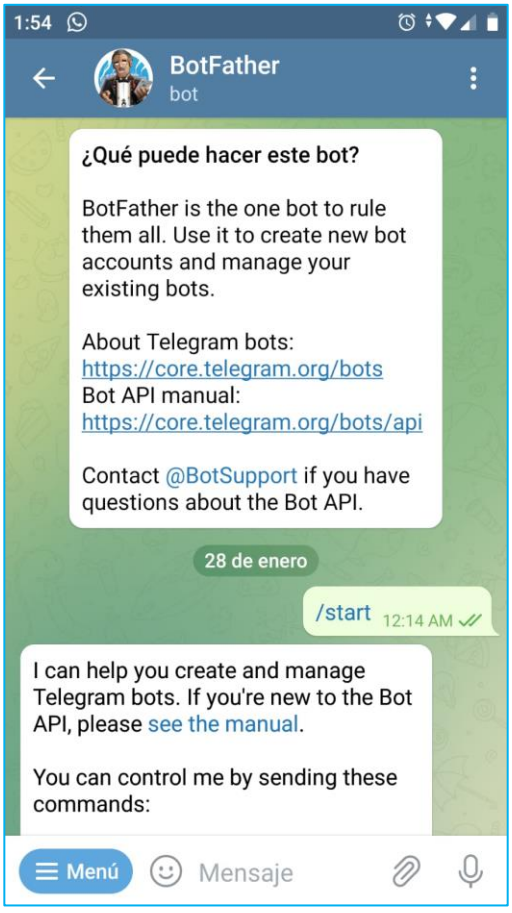
Ahora que hemos experimentado nuestra primera interacción con una API totalmente libre, veamos cómo podemos hacerlo con Telegram, una API pensada para desarrolladores completamente documentada y gratuita cuya descripción podemos consultar del siguiente sitio oficial web:

https://core.telegram.org/bots/api

Para enviar un mensaje por intermedio de la API de Telegram requiere, desde una cuenta de la aplicación en el teléfono, la generación y configuración de un bot el cual será el encargado de gestionar y administrar los accesos a los servicios ofrecidos por las API´s. Este proceso de creación del bot dará como resultado un bot_token que será la identificación que este tendrá en Telegram. Luego, desde la cuenta de Telegram instalada en el teléfono, se envía un mensaje al bot a fin de obtener el propio chat_ID, que es la identificación del usuario de la cuenta instalada en el teléfono (reemplaza indirectamente al número telefónico en la identificación del destinatario al enviarle un mensaje). Posteriormente se podrá enviar el mensaje usando el comando sendMessage e incorporando en el formato del mismo, el bot_token (indicará quien es el remitente) y el chat_ID (indicará quien es el destinatario) tal como lo veremos a continuación. Podemos entonces enunciar e ilustrar los pasos necesarios para poder realizar lo descripto:

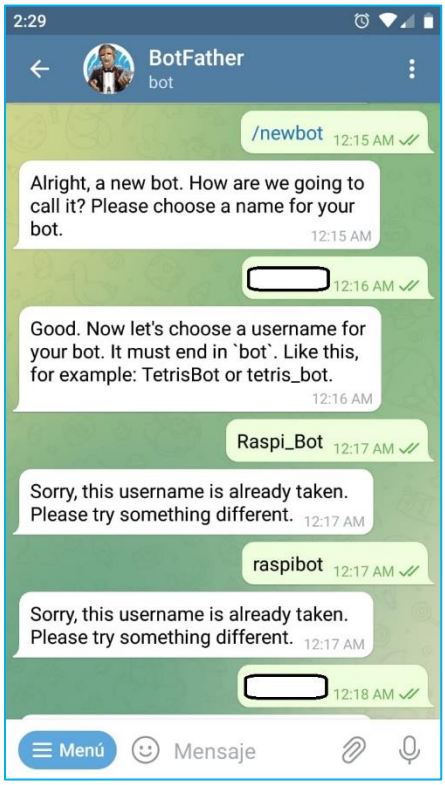
1) A partir de la tienda de descarga (google play store) instalar la aplicación de Telegram.
2) En Telegram buscar el contacto del bot denominado BotFather y abrir el chat para enviar el mensaje

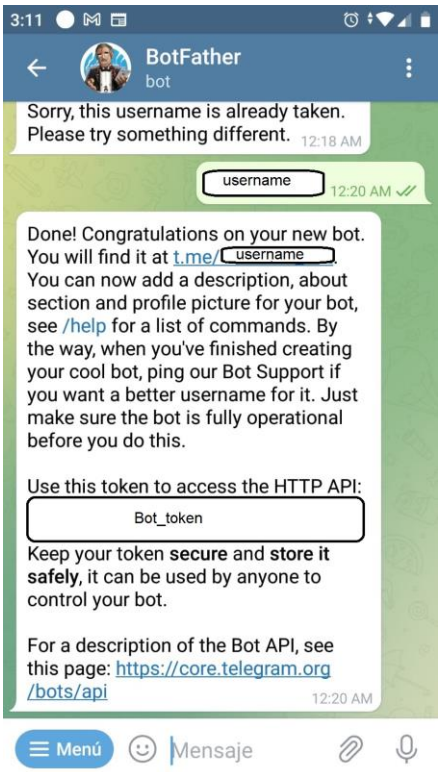




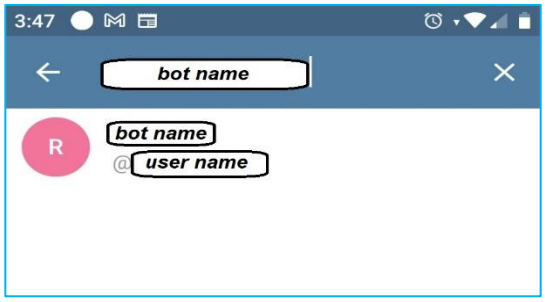
/start a lo cual BotFather responderá de acuerdo a lo que muestran las imágenes anteriores y suministrando una lista de comandos.

3) Crear un nuevo bot requiere enviar a continuación el mensaje: /newbot y responder satisfactoriamente a los pedidos de nombre tanto del bot como del usuario.





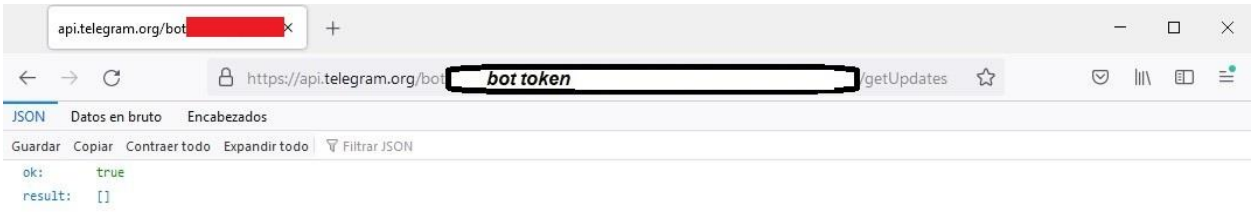
4) En Telegram buscar el contacto del bot creado denominado <bot name> con < user name> y abrir el chat para enviar un simple primer mensaje como “hola” a fin de obtener el chat_ID.



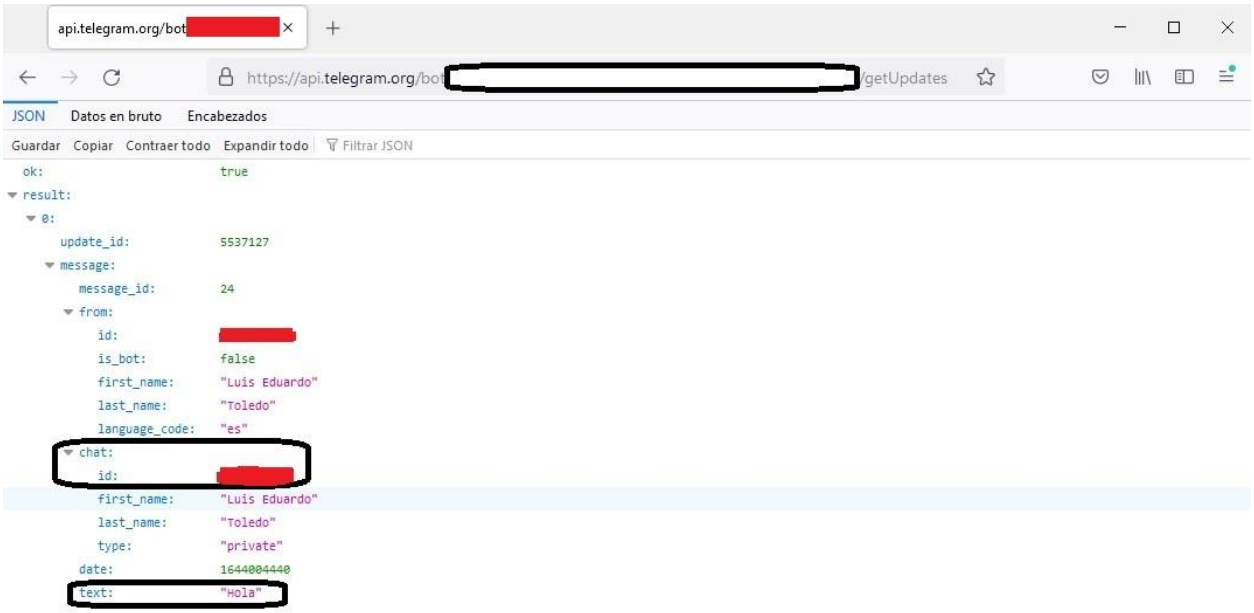
Para leer mensajes desde el bot, usamos nuevamente el navegador ingresando en la barra de direcciones la dirección web del sitio e incorporando la identificación obtenida del bot (bot token) seguido del nombre del comando getUpdates:

https://api.telegram.org/bot<bot_token>/getUpdates

Si no hay mensajes disponibles, se obtiene un resultado vacío



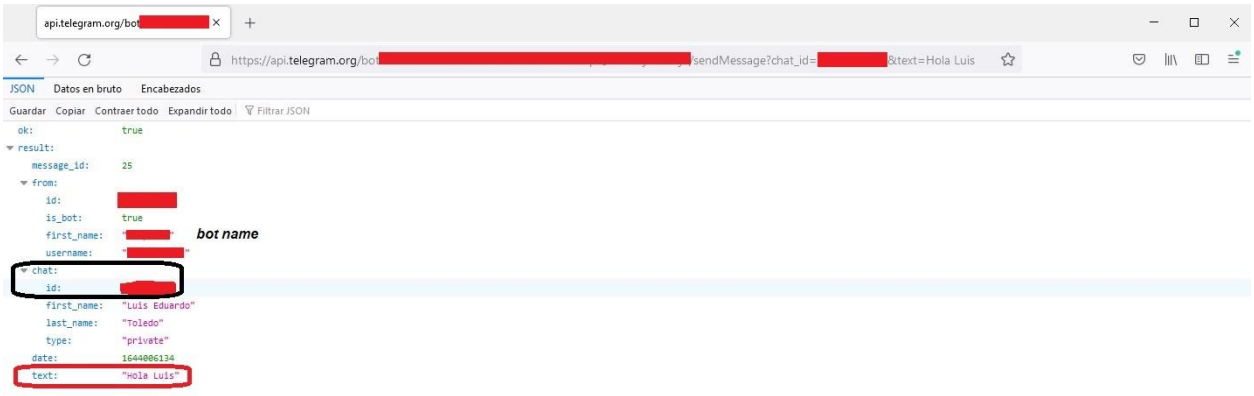
y si hemos enviado el hola al bot, recibiremos



de donde podemos extraer el chat_ID.

5) Ahora que tenemos la información tanto del bot_token como del chat_ID estamos en condiciones de enviar un mensaje como respuesta a través del navegador web usando el comando sendMessage:

https://api.telegram.org/bot<bot _token>/sendMessage?chat_id=<chat_ID>&text=Hola Luis



El mensaje “Hola Luis” llegará a la cuenta de Telegram del usuario seleccionado a través del chat_ID como se mostró anteriormente y como confirmación de que el mensaje ha sido enviado llegará al navegador en el formato JSON una serie de datos como: quien fue el remitente (sección From) con el id cuyo origen es un bot de determinado nombre y con el nombre de usuario indicado; a quien fue dirigido el mensaje (sección chat) con su id y nombre completo del usuario de la cuenta de Telegram y finalmente el texto enviado.

Sucesivos mensajes enviados al mismo usuario de Telegram seguirán usando el mismo chat_ID.

Como acceder a las API´s sin el navegador web usando un programa escrito en C

Todos los requerimientos de API´s que hasta aquí hemos realizado han sido gestionados con el navegador web; por lo tanto, podríamos decir que lo que necesitamos es poder realizar un GET de HTTP sin el browser.

Para poder llevar a cabo esta tarea, existe una biblioteca de funciones denominada CURL que nos permite realizarla desde un programa en C. CURL (client-side URL transfer library) es una biblioteca gratuita y fácil de usar para transferencias URL desde el lado del cliente en el típico modelo cliente-servidor, el cual describe el proceso de interacción entre la computadora local (el cliente) y la remota (el servidor). El cliente le hace peticiones (requests, solicitudes, requerimientos) al servidor, el cual procesa dicho requerimiento y retorna los resultados al cliente. Toda esta interacción puede llevarse a cabo transparentemente con CURL sin necesidad de entrar en detalle de los protocolos HTPP utilizados.

Para consultar la documentación e innumerables ejemplos de cómo se puede usar, se puede visitar el siguiente sitio oficial web: https://curl.se/libcurl

Antes de proceder a instalar CURL, primero actualice la lista de paquetes disponibles y sus versiones con:
$ sudo apt update
y luego actualice dichos paquetes usando el comando:
$ sudo apt upgrade

Ahora si proceda a instalar la biblioteca curl con el commando:

$ sudo apt-get install libcurl4-openssl-dev

El código en C del anexo, llamado httpget.c, hace uso de la biblioteca CURL y permite llevar a cabo los mismos requerimientos efectuados con el navegador, ingresando por línea de comando la dirección del sitio web con la información para solicitar el servicio a la API.

Para compilar (gcc/linux):
$ gcc httpget.c -lcurl -o httpget

Para probar:
$./httpget "https://api.telegram.org/bot<bot_token>/sendMessage?chat_id=<chat_ID>&text=Hola Luis "

Recuerde que <bot_token> y <chat_id> deben reemplazarse por los identificadores respectivos obtenidos en los pasos previos. Este simple programa C puede transformarse en una función para uso en programas más elaborados y complejos.

ANEXO: código C del programa httpget.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <curl/curl.h>
typedef struct string_buffer_s
{
   char * ptr;
   size_t len;
} string_buffer_t;

static void string_buffer_initialize( string_buffer_t * sb )
{
   sb->len = 0;
   sb->ptr = malloc(sb->len+1); /* will be grown as needed by the realloc belove */
   sb->ptr[0] = '\0';          /* no data at this point */
}

static void string_buffer_finish( string_buffer_t * sb )
{
   free(sb->ptr);
   sb->len = 0;
   sb->ptr = NULL;
}
static size_t string_buffer_callback( void * buf, size_t size, size_t nmemb, void * data )
{
   string_buffer_t * sb = data;
   size_t new_len = sb->len + size * nmemb;
   sb->ptr = realloc( sb->ptr, new_len + 1 );
   memcpy( sb->ptr + sb->len, buf, size * nmemb );
   sb->ptr[ new_len ] = '\0';
   sb->len = new_len;
   return size * nmemb;
}

static size_t header_callback(char * buf, size_t size, size_t nmemb, void * data )
{
   return string_buffer_callback( buf, size, nmemb, data );
}

static size_t write_callback( void * buf, size_t size, size_t nmemb, void * data )
{
   return string_buffer_callback( buf, size, nmemb, data );
}
```

```
int main( int argc, char * argv[] )
{
   CURL * curl;
   CURLcode res;
   string_buffer_t strbuf;

   char * myurl = argv[1];
//"https://api.telegram.org/bot .....bot_token...../sendMessage?chat_id=....&text=Hola%20Luis";

   string_buffer_initialize( &strbuf );

   /* init the curl session */
   curl = curl_easy_init();

   if(!curl)
   {
      fprintf(stderr, "Fatal: curl_easy_init() error.\n");
      string_buffer_finish( &strbuf );
      return EXIT_FAILURE;
   }
   /* specify URL to get */
   curl_easy_setopt(curl, CURLOPT_URL, myurl );

   curl_easy_setopt(curl, CURLOPT_FOLLOWLOCATION, 1L );

   /* send all data to this function  */
   curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, write_callback );
   curl_easy_setopt(curl, CURLOPT_HEADERFUNCTION, header_callback );

   /* we pass our 'strbuf' struct to the callback function */
   curl_easy_setopt(curl, CURLOPT_WRITEDATA, &strbuf );
   curl_easy_setopt(curl, CURLOPT_HEADERDATA, &strbuf );

   //curl_easy_setopt(curl, CURLOPT_VERBOSE, 1L);
   //curl_easy_setopt(curl, CURLOPT_HEADER, 1L);

   /* get it! */
   res = curl_easy_perform(curl);

   /* check for errors */
   if( res != CURLE_OK )
   {
      fprintf( stderr, "Request failed: curl_easy_perform(): %s\n", curl_easy_strerror(res) );
```

```
      curl_easy_cleanup( curl );
      string_buffer_finish( &strbuf );

      return EXIT_FAILURE;
   }
   /*
    * Now, our strbuf.ptr points to a memory block that is strbuf.len
    * bytes big and contains the remote resource.
    *
    * Do something nice with it!
    */
   printf( "%s\n\n", strbuf.ptr );
   printf("%lu bytes retrieved\n", (unsigned long)strbuf.len);

   /* cleanup curl stuff */
   curl_easy_cleanup( curl );
   string_buffer_finish( &strbuf );

   return EXIT_SUCCESS;
}

/* eof */
```