1 Node.js y Telegram

1.1 Sumario

2 Node.js y Telegram

En este tutorial, crearemos un **bot básico de Telegram** empleando **Node.js** y un framework llamado **Telegraf**.

2.1 Introducción a los Bots de Telegram

- Los bots, que ahora están muy de moda, son una funcionalidad de Telegram que nos permite crear una aplicación en Telegram que interactúe con nuestro servidor a través de mensajes de texto que se envían al chat del Bot.
- Para buscar bots solamente hay que entrar en la app de Telegram y buscarlos en el cajón de búsqueda. A través de web se puede hacer tecleando **https://telegram.me/** y añadiendo después de la última barra el nombre del bot que queremos localizar.
- Una de las ventajas de los Bots de Telegram es que Telegram proporciona de forma pública y gratuita su API para interactuar con todos sus servicios. Salvo aplicaciones como WhatsApp o similares que son cerradas y no permiten la realización de ninguna de estas tareas.
- Además **recomiendo Telegram encarecidamente** por que va muy por delante en seguridad y servicios disponibles en comparación a WhatsApp o similares.

2.2 Documentación oficial del framework Telegraf de Node

Podemos encontrar la **documentación oficial del framework Telegraf** para Node.js en:

- **https://telegraf.js.org/#/**

2.3 Registro de Bot en BotFather

- Lo primero que tenemos que hacer es **indicar a Telegram de que queremos crear un nuevo Bot**. Para ello tenemos que mandar un mensaje al **BotFather (@BotFather)**, en concreto al comando de «/newbot».
- Posteriormente BotFather nos preguntará por un **name** para el Bot (será el Título del Bot) y a continuación nos pregunta un **username**. Importante, el **username** tiene que **terminar en bot**. Ejemplo: mi_nuevobot, soytubot, etc...
- Si todo está correcto, nos confirmará la creación del Bot con un texto similar al mostrado más abajo, indicando el TOKEN que usaremos para hacer peticiones a la **API de Bots de Telegram**: **https://core.telegram.org/bots/api**
- **Importantísimo: NO COMPARTIR NUNCA NUESTRO TOKEN por que es la referencia a nuestro Bot.**

```
Done! Congratulations on your new bot. You will find it at t.me/veigarobot.

You can now add a description, about section and profile picture for your bot, see /help for a list of commands. By the way, when yo

Use this token to access the HTTP API:
1041226772:AAERTGXt3_oPG1ceVYKnz1KlFKKFLV7ykd2g

Keep your token secure and store it safely, it can be used by anyone to control your bot.

For a description of the Bot API, see this page: https://core.telegram.org/bots/api
```

- **Comandos disponibles en BotFather**:

```
/newbot ? Crea un nuevo bot, te pedirá el nombre y lo creará.
/token ? Te da el Token HTTP para una API que quieras hacer, por ejemplo.
/revoke ? Elimina el acceso mediante el Token a tu Bot.
/setname ? Cambia el nombre de tu bot
/setdescription ? Cambia la descripción de tu bot, muy útil para presentarlo y explicar lo que hace
/setabouttext ? Modifica la información sobre tu bot.
/setuserpic ? Cambia la imagen de perfil de tu bot.
/setinline ? Permite modificar los permisos con respecto a los bots integrados, si los entiende o no.
/setinlinefeedback ? Configura las respuestas a los mensajes con bot integrados.
/setcommands ? Cambia los comandos con los que se podrá interactuar con tu bot
/setjoingroups ? Habilita o deshabilita la opción de añadir tu bot a grupos
/setprivacy ? Ajustes de privacidad, si el bot puede ver todos los mensajes o sólo los que lo mencionen
/deletebot ? Eliminar un bot.
/cancel ? Cancelar la operación vigente con el BotFather
```

2.4 Configuración inicial del servidor de Node.js

1.- Primeramente **crearemos una carpeta para nuestro bot** en el servidor. **2.-** Realizaremos la inicialización del proyecto con **npm init**:

```
npm init
```

3.- Instalamos los siguientes **módulos necesarios** para programar el **bot**:

```
npm install --save telegraf dotenv fs http https express
```

4.- Creamos un **fichero de configuración** del entorno con el nombre **.env**. Ejemplo de contenido de un fichero: **.env**

```
# .env
# Bot Telegram: veigarobot

PUERTO = 8080
BOT_NOMBRE = veigarobot
BOT_TOKEN = 1041226772:AAERTGXt3_oPG1ceVYKnz1KlFKKFLV7ykd2g
```

2.4.1 Creación de certificado SSL para el servidor de Node.js usando LetsEncrypt (solamente cuando no tengamos certificado SSL)

1. Para que Telegram pueda enviar las solicitudes que le lleguen al Bot, a nuestro servidor de Node.js, nuestro servidor deberá tener instalado un **certificado SSL**.
2. Para ello podemos **crearlo de forma gratuita** utilizando el servicio **Let'sEncrypt**: **https://letsencrypt.org/es/**
3. Tendremos que realizar lo siguiente:
4. Tutorial seguido: **https://itnext.io/node-express-letsencrypt-generate-a-free-ssl-certificate-and-run-an-https-server-in-5-minutes-a730fbe528ca**
5. Requerimientos previos:

- Servidor accesible por SSH.
- NodeJS
- Express
- CertBot

- Primero instalaremos **CertBot en Debian 10**:

```
sudo su

# Editamos el fichero /etc/apt/sources.list:
nano /etc/apt/sources.list

# Añadimos este contenido:
deb http://deb.debian.org/debian buster-backports main
deb-src http://deb.debian.org/debian buster-backports main

# Actualizamos las sources.list
apt update

# Instalamos los siguientes paquetes:
apt-get install certbot

# Generamos un certificado SSL manualmente:
certbot certonly --manual

# Introduciremos el nombre de nuestro dominio sin http://
# Seguiremos las instrucciones indicadas...
# En mi caso para confirmar el dominio tuve que crear las siguientes carpetas, dentro de la carpeta de mi bot de telegram:
mkdir -p .well-known/acme-challenge

# Crear el siguiente archivo XdVyxWPEcX_A8HLC3aOJrhY7x8T3o1w0vPyYHG1EVYw dentro de la carpeta acme-challenge:

# Con el contenido que me indicaba.
XdVyxWPEcX_A8HLC3aOJrhY7x8T3o1w0vPyYHG1EVYw.xZzVoAmDSo0mh7XFWJo1tPScYVRdthQSRcRDjffG6B8

# Y a continuación pulsar en Y (y en ese momento letsencrypt se conecta a mi dominio y se crearán los certificados en:

# Cadena completa del certificado en:
/etc/letsencrypt/live/veiga.dynu.net-0001/fullchain.pem

# Clave privada en:
/etc/letsencrypt/live/veiga.dynu.net-0001/privkey.pem
```

- **Secuencia original completa de creación en la terminal:**

```
/var/www/veiga.dynu.net/public/nodejs/bottelegram# certbot certonly --manual

Saving debug log to /var/log/letsencrypt/letsencrypt.log
Plugins selected: Authenticator manual, Installer None
Please enter in your domain name(s) (comma and/or space separated)  (Enter 'c'
to cancel): veiga.dynu.net
Obtaining a new certificate
Performing the following challenges:
http-01 challenge for veiga.dynu.net

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
NOTE: The IP of this machine will be publicly logged as having requested this
certificate. If you're running certbot in manual mode on a machine that is not
your server, please ensure you're okay with that.

Are you OK with your IP being logged?
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
(Y)es/(N)o: Y

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Create a file containing just this data:

XdVyxWPEcX_A8HLC3aOJrhY7x8T3o1w0vPyYHG1EVYw.xZzVoAmDSo0mh7XFWJo1tPScYVRdthQSRcRDjffG6B8

And make it available on your web server at this URL:

http://veiga.dynu.net/.well-known/acme-challenge/XdVyxWPEcX_A8HLC3aOJrhY7x8T3o1w0vPyYHG1EVYw

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Press Enter to Continue
Waiting for verification...
Cleaning up challenges

IMPORTANT NOTES:
 - Congratulations! Your certificate and chain have been saved at:
   /etc/letsencrypt/live/veiga.dynu.net-0001/fullchain.pem
   Your key file has been saved at:
   /etc/letsencrypt/live/veiga.dynu.net-0001/privkey.pem
   Your cert will expire on 2020-04-13. To obtain a new or tweaked
   version of this certificate in the future, simply run certbot
   again. To non-interactively renew *all* of your certificates, run
   "certbot renew"
 - If you like Certbot, please consider supporting our work by:

   Donating to ISRG / Let's Encrypt:   https://letsencrypt.org/donate
   Donating to EFF:                    https://eff.org/donate-le
```

2.4.2 Configuración del certificado SSL para el servidor de Node.js (cuando ya dispongamos de un certificado SSL)

- **Si ya tenemos un certificado SSL** por que lo hemos creado en el paso anterior, o bien por que ya teníamos alguno de nuestro servidor de Nginx, podemos aprovecharlo para este servidor también.
- **La carpeta dónde buscar los certificados de CertBot** sería en:

```
# Directorio de todos los certificados de LetsEncrypt
/etc/letsencrypt/live

# Ahí dentro tendremos carpetas para cada dominio.

# Fichero del certificado, por ejemplo en:
/etc/letsencrypt/live/veiga.dynu.net-0001/cert.pem

# Fichero de la clave privada del certificado, por ejemplo en:
/etc/letsencrypt/live/veiga.dynu.net-0001/privkey.pem

# Fichero de certification authority:
/etc/letsencrypt/live/veiga.dynu.net-0001/chain.pem
```

- **Ejemplo de aplicación de Node.js que hace uso de los certificados SSL**:

```
// index.js de ejemplo de servidor con certificado SSL incluído.

// Módulos necesarios
const fs = require('fs');
const http = require('http');
const https = require('https');
const express = require('express');

const app = express();

// Configuración de los certificados
const privateKey = fs.readFileSync('/etc/letsencrypt/live/veiga.dynu.net/privkey.pem', 'utf8');
const certificate = fs.readFileSync('/etc/letsencrypt/live/veiga.dynu.net/cert.pem', 'utf8');
const ca = fs.readFileSync('/etc/letsencrypt/live/veiga.dynu.net/chain.pem', 'utf8');

const credentials = {
    key: privateKey,
    cert: certificate,
    ca: ca
```

```
};

app.use((req, res) => {
    res.send('Servidor web funcionando tanto en http y https !');
});

// Arrancamos tanto el servidor http como https:
const httpServer = http.createServer(app);
const httpsServer = https.createServer(credentials, app);

httpServer.listen(80, () => {
    console.log('Servidor http funcionando en puerto 80');
});

httpsServer.listen(443, () => {
    console.log('Servidor https funcionando en puerto 443');
});
```

- Podemos probar el servidor desde la línea de comandos.
- **ATENCION: hacerlo como root, ya que necesita tener permisos para poder acceder a las carpetas de los certificados.**

```
# Atención, antes de arrancar nuestro servidor de Node en el puerto 80 y 443, tendremos que parar el servidor de Nginx o cualquier o
sudo su

# Paramos nuestro servidor de Nginx que tenemos funcionando en el VPS de Google.
service nginx stop

# Arrancamos el servidor de Node.js
node index.js

Servidor http funcionando en puerto 80
Servidor https funcionando en puerto 443

# Nos conectaremos con un navegador cliente a la dirección de nuestro dominio por http y https y comprobamos que funciona correctame
http://veiga.dynu.net/
https://veiga.dynu.net/

# Resultado:
Servidor web funcionando tanto en http y https !
```

2.5 Configuración del Webhook de Telegram

- En un bot de Telegram cuando mandamos un comando, el proceso que se ejecuta es el siguiente:

1. El comando se envía desde nuestro teléfono o aplicación cliente de Telegram, a la red de Telegram.
2. Telegram reenvía ese comando al servidor dónde tenemos programado el Bot. En nuestro caso será el servidor en Google (para ello necesita saber la **URL HTTPS** (**Webhook**) del servidor que gestionará dicha petición de Telegram.
3. Nuestro servidor de Google responderá a dicha petición enviando una respuesta procesada, a la red de Telegram.
4. Telegram enviará finalmente al teléfono o app del cliente la respuesta recibida desde el Bot.

- **Fichero de variables de entorno .env completo**:

```
# .env
# Bot Telegram: veigarobot

BOT_NOMBRE = veigarobot
BOT_TOKEN = 1041226772:AAERTGXt3_oPG1ceVYKnz1KlFKKFLV7ykd2g
WEBHOOK_URL = https://veiga.dynu.net
WEBHOOK_CARPETA_SECRETA = /botty
PUERTO = 8443
```

- Ejemplo de **configuración del Webhook** de nuestro **bot**:

```
// index.js

// Cargamos el módulo que nos permite leer variables adicionales de entorno:
const dotenv = require('dotenv');

const fs = require('fs');
```

```
const Telegraf = require('telegraf');
const express = require('express');
const expressApp = express();

// Leemos el fichero ".env" que contiene las variables de configuración y se añadirá a la lista de variables que tenemos en process.
dotenv.config();

// Ejemplo de muestra de la variable de entorno BOT_TOKEN
// console.log(process.env.BOT_TOKEN);

// Constante que referencia a nuestro Bot.
const bot = new Telegraf(process.env.BOT_TOKEN)

// Configuración de los certificados
const privateKey = fs.readFileSync('/etc/letsencrypt/live/veiga.dynu.net/privkey.pem', 'utf8');
const certificate = fs.readFileSync('/etc/letsencrypt/live/veiga.dynu.net/cert.pem', 'utf8');
const ca = fs.readFileSync('/etc/letsencrypt/live/veiga.dynu.net/chain.pem', 'utf8');

const credenciales = {
    key: privateKey,
    cert: certificate,
    ca: ca
};

// Ajuste del webhook
// El webhook en Telegram solamente se puede configurar para los siguientes puertos: 80, 88, 443 or 8443
// Como queremos seguir teniendo nuestro servidor Nginx funcionando (para http 80 y https 443), vamos a trabajar en este caso con el

// Acordarse de abrir en el firewall de Google el puerto TCP 8443 a nuestro servidor.

// Configuraremos en nuestro fichero .env una variable de entorno para la URL del Webhook
// Configuraremos en nuestro fichero .env una variable para el puerto de escucha del servidor Node.js
// Y además configuraremos una ruta con el nombre que queramos que será dónde se recibirán las peticiones que lleguen desde la Red d

// WEBHOOK_URL = https://veiga.dynu.net
// PUERTO = 8443
// WEBHOOK_CARPETA_SECRETA = /botty

// Ajustamos el webhook de nuestro bot
expressApp.use(bot.webhookCallback(process.env.WEBHOOK_CARPETA_SECRETA));
bot.telegram.setWebhook(process.env.WEBHOOK_URL+':'+process.env.PUERTO+process.env.WEBHOOK_CARPETA_SECRETA);

expressApp.get('/', (req, res) => {
  res.send('Hola que tal ?')
})

expressApp.listen(process.env.PUERTO, () => {
  console.log(`Escuchando en el puerto ${process.env.PUERTO} !`);
})
```

2.6 Más conceptos sobre el framework Telegraf

2.6.1 Introducción

- En el framework de Node.js **Telegraf** que vamos a utilizar, un **bot es un objeto que contiene un array de middlewares** que se ejecutan en cada petición.
- Un **Middleware** es una parte esencial de cualquier framework moderno. Te permitirá modificar las peticiones y respuestas que se producen entre Telegram y tu bot.
- Podemos pensar en un middleware como una cadena lógica de conexión entre el bot y la petición de Telegram.
- Un Middleware normalmente acepta **2 parámetros (ctx,next)**. **CTX** es el contexto de la actualización de Telegram, **NEXT** es una función que se ejecutará para pasar al siguiente middleware.
- Retornará una **Promise** con una función **then** que se ejecutará cuando se termine la ejecución de la petición.
- Lista de **Middlewares** disponibles para el **framework Telegraf**: **https://telegraf.js.org/#/?id=known-middleware**

2.6.2 Gestión de errores en Telegraf

- Por defecto Telegraf imprimirá todos los errores a stderr.

2.6.3 Contexto

- Un contexto de Telegraf encapsula una actualización de Telegram.
- Un contexto se crea en cada petición y contiene las siguientes propiedades:
- **https://telegraf.js.org/#/?id=context**

2.6.4 Tipos de actualizaciones soportadas por Telegraf

- Cuando se produce un evento en Telegram los tipos y sub-tipos de actualizaciones disponibles, que podemos gestionar con el evento **.on** son:
- **https://telegraf.js.org/#/?id=update-types**

2.7 Ejemplo de Bot programado con Telegraf y Node.js

- Vamos a programar el bot para que haga lo siguiente:
    - Cuando **entremos nos salude con un mensaje**.
    - Cuando **le enviemos un texto nos responda con otro texto**.
    - Cuando **le enviemos nuestra localización nos devuelva las coordenadas**.
    - Cuando **le enviemos un sticker nos devuelva un icono**.
    - Cuando **le enviemos un comando /fecha nos devuelva la fecha y hora**.
    - Y alguna cosilla más (ver el código).

- Véase la **explicación en la parte de código fuente** //// **Interactuando con nuestro Bot de Telegram** ////
- Si con nodemon nos genera algún error no detectado,.se recomienda parar con CTRL+C y ejecutar manualmente con node index.js.

Código del fichero **.env**:

```
#.env
# Bot Telegram. Datos de configuración:
# Nombre del bot: veigarobot

BOT_NOMBRE = veigarobot
BOT_TOKEN = 1087578423234:AAwer23049afiasdf_lasiwer234sadf21KlFKg
WEBHOOK_URL = https://veiga.dynu.net
WEBHOOK_CARPETA_SECRETA = /newbotxxx

# Puertos validos de conexion desde Telegram a servidores: 80, 88, 443 y 8443
PUERTO = 8443
```

- Crear la carpeta /**descargas** dentro del bot de Node:
- Inicialización de package.json e instalación de todos los módulos necesarios:

```
npm init

npm install --save fs https express dotenv telegraf request unique-filename
```

- Código del fichero **index.js**:

```
// index.js
// Cargamos módulos
const fs = require('fs');
const https = require ('https');
const express = require ('express');
const dotenv = require ('dotenv');
const Telegraf = require ('telegraf');
const Extra = require('telegraf/extra')
const Markup = require('telegraf/markup')
const request = require('request')
const uniqueFilename = require ('unique-filename');

const app= express();

// Configuramos los certificados SSL para HTTPS
const privateKey = fs.readFileSync('/etc/letsencrypt/live/veiga.dynu.net/privkey.pem', 'utf8');
const certificate = fs.readFileSync('/etc/letsencrypt/live/veiga.dynu.net/cert.pem', 'utf8');
const ca = fs.readFileSync('/etc/letsencrypt/live/veiga.dynu.net/chain.pem', 'utf8');
```

```
const credentials = {
    key: privateKey,
    cert: certificate,
    ca: ca
};

// Leemos el fichero de entorno .env
dotenv.config();

// Creamos objeto Telegraf
const bot = new Telegraf(process.env.BOT_TOKEN);

// Ajustamos el Webhook (necesario hacer solamente 1 vez o cuando cambie la URL/puerto de nuestro servidor)
// Esta línea la ejecutamos 1 vez y la comentamos:
// bot.telegram.setWebhook(process.env.WEBHOOK_URL+':'+process.env.PUERTO+process.env.WEBHOOK_CARPETA_SECRETA);

// Ajustamos la ruta en Express, que gestionará las peticiones que recibimos desde Telegram en el bot.
app.use (bot.webhookCallback(process.env.WEBHOOK_CARPETA_SECRETA));

// Arrancamos el servidor https en el puerto 8443
const httpsServer = https.createServer(credentials,app);
var server= httpsServer.listen(process.env.PUERTO, () => {
    console.log('Servidor https funcionando en puerto 8443');
});

// Prueba de servicio web, comentar despues...
// Nos conectamos a https://veiga.dynu.net:8443 para ver si funciona todo OK.
// app.get('/',(req,res)=>{
//     res.send('Hola que tal');
// });

// Función que nos permitirá descargar un fichero desde una URL determinada.
var descargarFichero =function(url,nombrefichero,callback)
{
    request.head(url, function(err,res,body)
    {
        // Consulta de headers.. Ejemplo:
        //console.log('content-type:',res.headers['content-type']);
        //console.log('content-length:',res.headers['content-length']);
        request(url).pipe(fs.createWriteStream(nombrefichero)).on('close',callback);
    });
}


////////////////////////////////////////////////
//// Interactuando con nuestro Bot de Telegram ////
////////////////////////////////////////////////

// Mensaje de bienvenida.
bot.start((ctx)=>{
    //console.log(ctx);
    ctx.reply('Bienvenido/a a mi Bot. Si quieres volver a ver este mensaje, pulsa en /start')
});

// Si escribo hola en el chat, que responda con Hola que tal?
//bot.hears('Hola',(ctx)=>ctx.reply(`Hola ${ctx.from.first_name}, ¿qué tal estás?`));

// Versión mejorada....
bot.hears('Hola',(ctx)=>ctx.replyWithHTML(`Hola <b>${ctx.from.first_name}</b>, ¿qué tal estás?. Tu ID en Telegram es: <b>${ctx.messa

// Cuando alguien escribe "calor"... le decimos algo
bot.hears(/calor/i,(ctx)=>ctx.reply('Pues si, parece que hacer calor por aqui....'));

// Al poner el comando /help... que nos mande un texto.
bot.help((ctx)=>ctx.reply('Aquí se muestra la ayuda del bot... Puedes por ejemplo mandar un sticker y te mando un simbolito...'));

// Programamos el evento de cuando recibimos un sticker nos devuelva la mano de OK
bot.on('sticker',(ctx)=>ctx.reply('?'));

// CUando recibimos unas coordenadas de localización...
bot.on('location',(ctx)=>{
    ctx.replyWithHTML("Hemos recibido tus coordenadas en el Bot:").then(()=>ctx.replyWithHTML(`<b>Latitud</b>: ${ctx.message.locatio
```

```
});

// Programamos un comando específico /fecha
bot.command('fecha', (ctx) =>
{
  let fecha = new Date();
  let dia = ("0" + fecha.getDate()).slice(-2);
  let mes = ("0" + (fecha.getMonth() + 1)).slice(-2);
  let anio = fecha.getFullYear();
  let horas = fecha.getHours();
  let minutos = fecha.getMinutes();
  let segundos = fecha.getSeconds();

  ctx.replyWithHTML("Fecha y hora en el sistema:<b> "+dia + "/" + mes + "/" + anio+" "+horas+ ":" +minutos+ ":" +segundos+"</b>");
});

// Si recibimos una fotografía la almacena en la carpeta descargas de mi bot.
bot.on('photo',(ctx)=>
{
    // Mostramos en consola para averiguar los datos de la foto que recibimos.
    //console.log(ctx.message);
    /*
  photo: [
    {
      file_id: 'AgADBAADG7QxG_-SQFFH64BLw3Mki40othsABAEAAwIAA20AA6ENAgABFgQ',
      file_unique_id: 'AQADjSi2GwAEoQ0CAAE',
      file_size: 14561,
      width: 240,
      height: 320
    },
    {
      file_id: 'AgADBAADG7QxG_-SQFFH64BLw3Mki40othsABAEAAwIAA3gAA6INAgABFgQ',
      file_unique_id: 'AQADjSi2GwAEog0CAAE',
      file_size: 54461,
      width: 600,
      height: 800
    },
    {
      file_id: 'AgADBAADG7QxG_-SQFFH64BLw3Mki40othsABAEAAwIAA3kAA58NAgABFgQ',
      file_unique_id: 'AQADjSi2GwAEnw0CAAE',
      file_size: 132642,
      width: 960,
      height: 1280
    }
  ]
}
    */
    // Necesitamos averiguar la carpeta dónde se almacenó en Telegram la foto enviada por el cliente. Cogemos la de mayor calidad, l
    let urlInfoFichero = `https://api.telegram.org/bot${process.env.BOT_TOKEN}/getFile?file_id=${ctx.message.photo[ctx.message.photo

    // Nos conectamos a Telegram para averiguar la carpeta dónde se almacenó la foto.
    request(urlInfoFichero,function(err,response,body)
    {
        body= JSON.parse(body);

        let urlDescargaFichero = `https://api.telegram.org/file/bot${process.env.BOT_TOKEN}/${body.result.file_path}`;

        // Generamos un nombre único para ese fichero, para poder guardar en /descargas
        let nombreFinalFichero = uniqueFilename('./descargas/')+'.'+body.result.file_path.split('.').pop();

        descargarFichero(urlDescargaFichero,nombreFinalFichero, function()
        {
            ctx.replyWithHTML('Imagen recibida correctamente en el servidor. Le mandamos una copia');

            // Esta foto ya está en TElegram, con lo que si ponemos el file_id es suficiente.
            ctx.replyWithPhoto(body.result.file_id);

            // Si queremos enviar una foto sacada de Internet....
            // ctx.replyWithPhoto('http://www.discovery-8.com/wp-content/uploads/2017/07/parapente-algodonales.jpeg');
        })
    });
});
```

```
// Ejemplo de teclado con un botón que apunta a una URL y un botón que borra la copia del mensaje.
const teclado = Markup.inlineKeyboard([
  Markup.urlButton('??', 'https://www.iessanclemente.net'),
  Markup.callbackButton('Borrar', 'accionborrar')
])

//Programación de accionborrar. Borra el mensaje y el teclado asociado.
bot.action('accionborrar', ({ deleteMessage }) => deleteMessage());

// Cuando reciba un mensaje que no está programado anteriormente entonces, responde con un teclado de ejemplo, que muestra un enlace
bot.on('message', (ctx) => ctx.telegram.sendCopy(ctx.chat.id, ctx.message, Extra.markup(teclado)));

// Por último arrancamos el Bot.
bot.launch();
```

- **ATENCION**: Acordarse de **ejecutarlo como root** con sudo ya que tiene que leer los certificados SSL de nuestro dominio.

2.8 Ejecución permanente del servidor de Node.JS

- Cuando ya tenemos el servidor en **producción** funcionando correctamente, podemos emplear una herramienta que se llama **forever**, que se encarga de monitorizar la aplicación y si ésta se para por alguna razón, volver a arrancarla de forma automática.
- Atención **forever no monitoriza los cambios en la aplicación**, solamente se encarga de mantenerla en funcionamiento permanente.
- Más información sobre **forever** en: **https://www.npmjs.com/package/forever**

Podemos **instalar forever de forma global** con el siguiente comando:

```
npm install -g forever

# Para arrancar de forma continua el fichero index.js haremos:
forever start index.js

# Para ver la lista de procesos controlados por forever:
forever list

# Para parar un proceso:
forever stop uid_del_proceso

# Para reiniciar todos los procesos:
forever restartall

# Si queremos que nuestro script de Node, se arranque cuando se inicia la máquina de Linux en Debian, haremos por ejemplo un script
nano arrancarservidornode.sh

# Con el siguiente contenido, por ejemplo:
-----------------------------------------------------------------

#!/bin/bash
clear
export PATH=/usr/bin:$PATH
RUTA="/var/www/veiga.dynu.net/nodejs/bottelegram/"
LOG="server.log"

cd $ruta
# Inicializamos el log al arrancar el equipo:
echo "" > $RUTA$LOG

forever start -o $RUTA$LOG --command node --sourceDir $RUTA index.js

-----------------------------------------------------------------

# Guardamos el fichero arrancarservidornode.sh
# Le aplicamos permisos:
chmod 755 arrancarservidornode.sh

# A continuación metemos el script bash en el arranque del sistema:
crontab -e

# Añadimos las 2 siguientes líneas:

# Arranque de servidor de node con forever
```

```
@reboot /ruta_del_script_/arrancarservidornode.sh

# Salimos guardando los cambios.

# Probamos a reiniciar el servidor:
reboot

# Y comprobaremos que el proceso de node está funcionando al reiniciar el sistema:
sudo su

forever list
```

Veiga (discusión) 12:59 22 ene 2020 (CET)