

2017

SERVIDOR DE APLICACIÓN FLASK

PROGRAMACIÓN DE UNA APLICACIÓN
CON SERVICIOS WEB

MIGUEL ÁNGEL GARCÍA ROMERAL

ÍNDICE

INTRODUCCIÓN	2
ANÁLISIS	3
INSTALACIÓN DE LA APLICACIÓN	5
Instalación de Flask.....	5
Instalación de MongoDB.....	7
Registro y configuración de ThingSpeak	9
ESTRUCTURA	11
app.py.....	11
hilo.py.....	12
modelo.py.....	12
util.py	12
web_interface.py	12
PROBLEMAS ENCONTRADOS	13
Instalación y manejo de MongoDB	13
Conexión a internet para la máquina virtual (servidor).....	13
Recarga de la página cada dos minutos	13
Doble ejecución del programa y módulo consultor	14
Inconsistencia entre bases de datos.....	14
MANUALES.....	15
Para el cliente	15
Para el administrador	19

INTRODUCCIÓN

El presente documento tiene como fin explicar de manera detallada la elaboración de una aplicación con servicios web. El primer apartado detalla el análisis realizado para el desarrollo de la aplicación según se especificaba en el enunciado de la práctica. El siguiente apartado detalla la instalación de los correspondientes componentes necesarios para llevar a cabo dicha instalación. En tercer lugar, se detalla la estructura de la aplicación con la funcionalidad de los diferentes módulos Python.

También se reserva un epígrafe en el que se comentan los problemas más importantes que se han encontrado durante el desarrollo de la práctica. Finalmente, se encuentra un manual en el que se encuentran dos manuales, uno para el usuario de la aplicación y otro para el administrador que se encarga de poner en funcionamiento el módulo principal y gestionar las bases de datos.

ANÁLISIS

El primer paso para crear nuestra aplicación será estudiar qué tecnología debemos usar que se adapte a nuestras necesidades. Por ello, se ha utilizado el framework Flask para la creación del servidor web, de este modo, el lenguaje utilizado para la codificación de la lógica de la aplicación es Python. Para poder conectar Flask con un módulo de Python será necesario importar los métodos necesarios desde el módulo llamado "flask". La aplicación será desarrollada en un sistema operativo Linux (concretamente, distribución Ubuntu 14.04) en una máquina virtual.

Además del propio servidor, será necesario conectar dicho servidor a dos bases de datos, una local y otra externa.

El sistema gestor de la base de datos local será MongoDB, uno de los sistemas más conocidos que utiliza el lenguaje NoSQL para la modificación de sus bases de datos. Además de MongoDB, usaremos el programa Robomongo con el fin de poder acceder a los datos de manera más cómoda y, sobre todo, para la depuración de la aplicación puesto que el usuario final no usará Robomongo. También será necesario contar con el módulo Pymongo instalado, puesto que se trata de una API para la modificación y consulta de las bases de datos que se encuentran en MongoDB.

Para poder replicar los datos en una base de datos remota (alojada en internet y no en nuestra máquina) hemos utilizado la API de ThingSpeak para la inserción de los datos que la aplicación vaya generando. ThingSpeak se utiliza para monitorizar diferentes parámetros en dispositivos, tales como la temperatura, uso de la CPU o memoria entre otros, aunque la funcionalidad para nuestra aplicación consistirá únicamente en guardar números generados aleatoriamente.

Para navegar en ambas bases de datos debemos tener en cuenta que los datos que insertemos y posteriormente recuperemos, son documentos JSON, por lo que la aplicación debe poder no solo conectarse a ambas bases de datos, sino también manejar el uso de estos documentos para las inserciones y consultas.

La aplicación debe poder realizar diferentes acciones, las cuales se enumeran a continuación:

- Acceso a una página web que muestre el último número aleatorio registrado en la base de datos. Este número aleatorio será obtenido solicitando una página web que los genera y, mediante expresiones regulares, obtener uno de ellos para registrar, junto a la fecha y hora en el que se llevó a cabo dicha inserción.
- Solicitar un número aleatorio tal y como se ha especificado en el punto anterior cuando se inicia la aplicación web y que este proceso se repita cada dos minutos.
- Recargar la página web solicitada por el usuario cada dos minutos para que pueda ver el último número registrado.
- Filtrar los números según un determinado umbral, así como obtener el último número registrado en la base de datos que cumple la condición de ser menor que el umbral especificado, indicando el momento exacto en el que fue registrado. Además, la aplicación mostrará al usuario la lista completa de números filtrados.
- Calcular la media de todos los números almacenados en ambas bases de datos.

- Mostrar una gráfica con los últimos números registrados en la base de datos externa, ayudándonos de la API de ThingSpeak.

Una vez que tenemos la lógica de negocio bien definida, necesitamos una interfaz que conecte esta lógica con el usuario final. Dado que nuestra aplicación tiene una arquitectura de N-niveles, la capa de presentación estará basada en una web, por lo que debemos crear una página web a través de documentos HTML.

La página principal de la web mostrará el último número aleatorio obtenido por el servidor y ofrecerá al cliente la opción que desee mediante un formulario. Las opciones serán introducir un número que sirva de umbral para poder ver la lista de números que se sitúan por debajo de dicho umbral, obtener la media de los números de las bases de datos y por último acceder a la gráfica de los últimos números de la base de datos externa. Cada una de estas opciones devolverá otro documento HTML con los datos requeridos por el cliente.

La página que muestra la lista de valores filtrados por el umbral mostrará además el último número que cumple la condición antes de mostrar dicha lista. También indicará el umbral que el cliente ha seleccionado. Cabe destacar, que aparecerán los registros de ambas bases de datos, tanto la local, como la externa.

La tercera página diseñada en la aplicación servirá para calcular la media de las dos bases de datos. Para una mayor trazabilidad sobre los cálculos, se indica la lista completa de números registrados.

La última página, mostrará la gráfica de los últimos números registrados en la base de datos remota.

Las tres páginas web secundarias cuentan con un botón que servirá para volver a la página principal y solicitar nuevamente un número aleatorio.

También será necesario incluir una página que indique al usuario que algo salió mal durante la solicitud. Únicamente sirve para evitar que aparezcan los diferentes errores HTTP tales como 400 ó 500. Normalmente el cliente las recibe cuando no se ha podido conectar con la página de números aleatorios.

Una vez definido el análisis de la aplicación, procedemos a detallar su estructura.

INSTALACIÓN DE LA APLICACIÓN

Para poder desarrollar la aplicación será necesario haber instalado algunos componentes tales como Flask o MongoDB. En este epígrafe se detalla la instalación de los diferentes componentes.

Instalación de Flask

Los pasos realizados son los que se nos indica en la documentación oficial de Flask (<http://flask.pocoo.org/docs/0.10/installation/>) Lo primero que debemos hacer es situarnos en el directorio en el que desarrollaremos la aplicación.

El siguiente paso es instalar Python en nuestra máquina, lo cual se realiza mediante los siguientes comandos:

```
$ sudo apt-get install python-dev
```

```
$ sudo apt-get install python-virtualenv
```

De esta manera, ya tendremos el entorno virtual instalado.

El siguiente paso es crear una carpeta para nuestro proyecto y situarnos en ésta.

```
$ mkdir iroom
```

```
$ cd iroom
```

Una vez situados, debemos activar el entorno correspondiente a Flask:

```
$ virtualenv flask
```

A partir de ese momento, es entonces cuando debemos activar Flask:

```
$ . flask/bin/activate
```

Por último, tenemos que instalar los módulos de Flask para poder trabajar en Python con ello.

```
$ pip install flask
```

```
$ pip install flask-script
```

De este modo, ya podremos ejecutar un módulo Python en el directorio del proyecto:

```
$ python app.py
```

Instalación de MongoDB

Ahora, gracias a la documentación que nos proporciona MongoDB (<https://docs.mongodb.com/v3.0/tutorial/install-mongodb-on-ubuntu/>) podemos instalar el sistema gestor de la base de datos.

Lo primero que debemos hacer es importar la clave publica que es usada por el paquete de gestión del sistema:

```
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10
```

Lo siguiente será crear una lista de archivos para MongoDB (este es el comando para el sistema operativo Ubuntu 14.04):

```
$ echo "deb http://repo.mongodb.org/apt/ubuntu trusty/mongodb-org/3.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.0.list
```

A continuación, será recargar los paquetes.

```
$ sudo apt-get update
```

Después será necesario instalar los paquetes de MongoDB, instalando una versión estable del software:

```
$ sudo apt-get install -y mongodb-org
```

Una vez instalado, debemos arrancar el servicio de MongoDB:

```
$ sudo service mongod start
```

Comprobamos que está arrancando haciendo un "ps" o viendo el log del servicio que se encuentra en "/var/log/mongodb/mongod.log":

Por último, para tener acceso a los datos que vamos registrando, descargamos el software Robomongo desde su página web (<https://robomongo.org/download>).

Como ya se mencionó con anterioridad, Robomongo solo servirá al administrador del servidor para poder manipular los datos según sea conveniente.

db.getCollection('numeros').find({})

numeros 0.028 sec.

Key	Value	Type
(1) Objectid("5a32cb309385a14c951a6baa")	{ 3 fields }	Object
_id	Objectid("5a32cb309385a14c951a6baa")	Objectid
fecha	Thu Dec 14 20:04:16 2017	String
numero	1.56	String
(2) Objectid("5a32cb409385a14c951a6bab")	{ 3 fields }	Object
_id	Objectid("5a32cb409385a14c951a6bab")	Objectid
fecha	Thu Dec 14 20:04:32 2017	String
numero	73.27	String
(3) Objectid("5a32cb509385a14c951a6bac")	{ 3 fields }	Object
_id	Objectid("5a32cb509385a14c951a6bac")	Objectid
fecha	Thu Dec 14 20:04:48 2017	String
numero	9.40	String
(4) Objectid("5a32cb7b9385a14c951a6bad")	{ 3 fields }	Object
_id	Objectid("5a32cb7b9385a14c951a6bad")	Objectid
fecha	Thu Dec 14 20:05:31 2017	String
numero	44.76	String
(5) Objectid("5a32cb879385a14c951a6bae")	{ 3 fields }	Object
_id	Objectid("5a32cb879385a14c951a6bae")	Objectid
fecha	Thu Dec 14 20:05:43 2017	String
numero	48.47	String
(6) Objectid("5a32cb969385a14c951a6baf")	{ 3 fields }	Object
_id	Objectid("5a32cb969385a14c951a6baf")	Objectid
fecha	Thu Dec 14 20:05:58 2017	String
numero	38.23	String
(7) Objectid("5a32cba69385a14c951a6bb0")	{ 3 fields }	Object
_id	Objectid("5a32cba69385a14c951a6bb0")	Objectid
fecha	Thu Dec 14 20:06:14 2017	String
numero	33.31	String
(8) Objectid("5a32cbb79385a14c951a6bb1")	{ 3 fields }	Object
_id	Objectid("5a32cbb79385a14c951a6bb1")	Objectid
fecha	Thu Dec 14 20:06:31 2017	String
numero	35.79	String

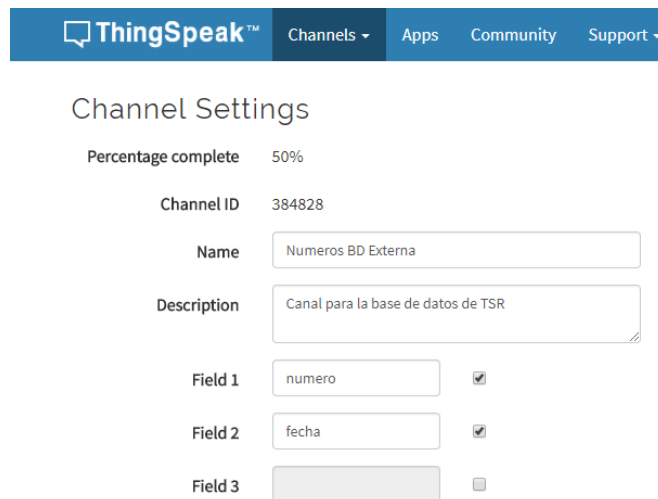
Registro y configuración de ThingSpeak

Para la base de datos externa, en la nube, la opción seleccionada es ThingSpeak. Para poder relacionar dispositivos y bases de datos, debemos crear una cuenta en su portal web.

Nuestro nombre de usuario será “miguelromeral” y crearemos un canal para registrar los datos de la aplicación. El canal se llama “Numeros BD Externa” y lo haremos público para tener acceso desde el servidor.

Name	Created	Updated At
Numeros BD Externa Private Public Settings Sharing API Keys Data Import / Export	2017-12-14	2017-12-30 17:44

En los ajustes del canal, creamos dos tipos de datos, uno que se llamará numero y otro fecha.



ThingSpeak™ Channels Apps Community Support

Channel Settings

Percentage complete 50%

Channel ID 384828

Name Numeros BD Externa

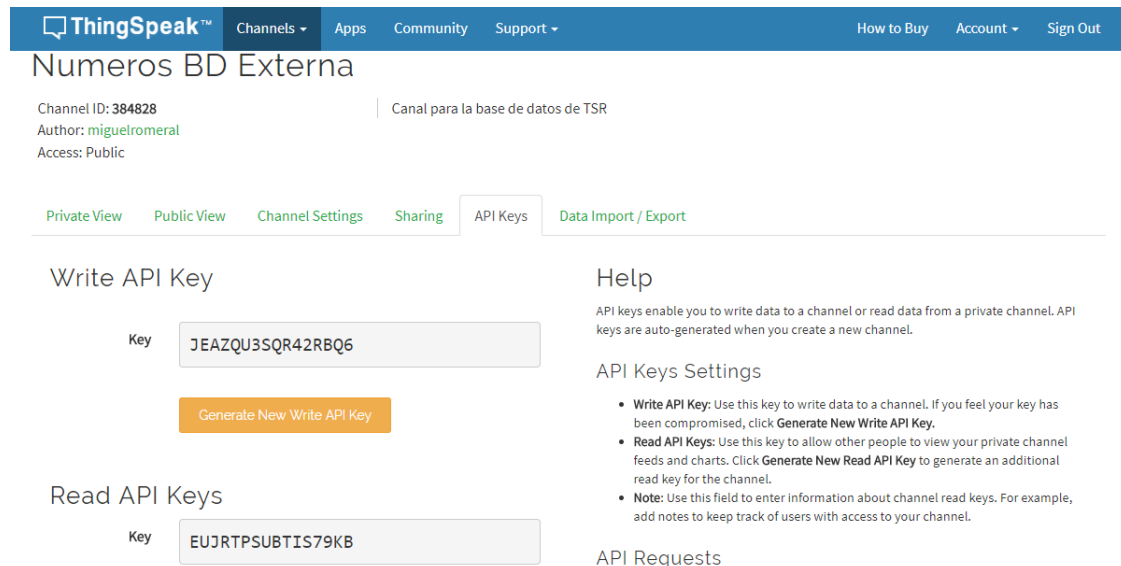
Description Canal para la base de datos de TSR

Field 1 numero ☒

Field 2 fecha ☒

Field 3 ☐

Por último, debemos obtener las claves de lectura y escritura del canal, además del número identificativo del canal. Todos estos datos serán utilizados por la aplicación por un módulo que se encarga de comunicarse con el canal para obtener los números y poder insertarlos.



ThingSpeak™ Channels Apps Community Support How to Buy Account Sign Out

Numeros BD Externa

Channel ID: 384828 | Canal para la base de datos de TSR

Author: miguelromeral

Access: Public

Private View Public View Channel Settings Sharing API Keys Data Import / Export

Write API Key

Key JEAZQU3SQR42RBQ6

Generate New Write API Key

Read API Keys

Key EUJRTPSUBTIS79KB

Help

API keys enable you to write data to a channel or read data from a private channel. API keys are auto-generated when you create a new channel.

API Keys Settings

- Write API Key:** Use this key to write data to a channel. If you feel your key has been compromised, click **Generate New Write API Key**.
- Read API Keys:** Use this key to allow other people to view your private channel feeds and charts. Click **Generate New Read API Key** to generate an additional read key for the channel.
- Note:** Use this field to enter information about channel read keys. For example, add notes to keep track of users with access to your channel.

API Requests

En la siguiente imagen se observa como en el módulo de Python se han incluido las claves correspondientes.

```
# Base de datos externa (ThingSpeak)

#Obtenemos las claves que proporciona ThingSpeak para escribir y leer en la BD respectivamente.
key = 'JEAZQU3SQR42RBQ6'
r_key = 'EUJRTPSUBTIS79KB'
# Identificador del repositorio en el que está la BD.
ch_id = '384828'
# Abrimos la conexión con la API de ThingSpeak
conn_ts = httplib.HTTPConnection("api.thingspeak.com:80")
print 'Abierta la conexión con la base de datos de ThingSpeak'
```

ESTRUCTURA

La estructura de la aplicación se encuentra en el directorio “iroom” de la máquina virtual, que es la encargada de ejecutar dicha aplicación. El árbol de directorios está dividido en “./iroom /flask” que es donde se guardan todos los archivos referidos al framework, “./iroom /static” almacena los ficheros HTML, y por último, los ficheros escritos en Python se encuentran en “./iroom”.

El directorio completo de los ficheros es el siguiente (se omiten los ficheros y directorios que se generan cuando se instala Flask):

```

./ iroom
├── app.py
├── flask
│   └── (...)
├── hilo.py
├── modelo.py
├── static
│   ├── error.html
│   ├── estilos.css
│   ├── main.html
│   ├── media.html
│   └── view_umbral.html
├── util.py
└── web_interface.py

```

El directorio “static” contiene las páginas web que son devueltas al navegador del cliente cuando son solicitadas y una hoja de estilos CSS. Desde el módulo principal de la aplicación en Python debemos indicar dónde se sitúan los documentos HTML para que pueda devolverlas al navegador del cliente al realizar una solicitud.

Se puede apreciar que los módulos Python son app.py, hilo.py, modelo.py, util.py y web_interface.py, los cuales son detallados a continuación.

app.py

Es el módulo principal. Se encarga gestionar las peticiones de los usuarios a los diferentes GET y POST que realicen. Devuelve el contenido de las páginas web con la información necesaria.

En cada solicitud, recupera los valores que son necesarios, tales como la lista de registros en la base de datos filtrado por un umbral o la media de la lista total de los números que se encuentran en la base de datos, pasándoselos a los documentos HTML con el fin de poder observar las instancias actuales.

Cuando se para la aplicación de manera manual, cierra las conexiones de manera segura con las bases de datos que el modelo de la aplicación abrió al ser iniciado.

hilo.py

Este módulo inicializa un hilo que se encarga de recoger datos de la página web de números aleatorios cada dos minutos desde que se inicializó la aplicación.

En el momento en el que se para la ejecución del módulo principal de la aplicación, el hilo es parado de manera automática.

Es importante destacar que esta funcionalidad no se encontraba en la aplicación que fue defendida el pasado 20 de diciembre en el laboratorio (anteriormente los números eran insertados en la base de datos en cada solicitud de los clientes). De esta manera, nos aseguramos de que los datos únicamente son insertados cada dos minutos, sean cuales sean las solicitudes que reciba el servidor.

modelo.py

Es el módulo que se encarga de obtener los datos de las bases de datos. Cuando la aplicación es arrancada, inicia la conexión tanto con MongoDB (a través del módulo pymongo, instalado en el equipo) como con ThingSpeak.

Para conectarse a MongoDB antes debe estar ejecutándose el proceso “mongod”. Si el proceso de MongoDB está correctamente iniciado, entonces escuchará en el puerto 27017 (que está establecido por defecto) todas las peticiones que realice la aplicación con la base de datos.

Para poder hacerlo con ThingSpeak necesitamos, además de tener conexión a internet, conocer las claves tanto de escritura como de lectura que nos ofrece la API cuando el canal (o la base de datos) es creado. También es necesario, antes de poder crear ningún canal, haberse registrado y validado en su página web, tal y como se ha especificado anteriormente.

Este módulo también tiene los diferentes métodos que operan las consultas sobre las bases de datos, tales como obtener la lista de los registros, filtrarlos por un determinado umbral o insertar los números en ambas bases de datos.

util.py

Módulo auxiliar que tiene métodos para sumar los elementos de una lista o imprimir la salida de una determinada manera para facilitar la trazabilidad de la aplicación al administrador.

Estos métodos no tienen cumplen con una funcionalidad específica de la lógica de aplicación que deseamos, pero contiene utilidades para su desarrollo.

web_interface.py

Gestiona las conexiones con el exterior de la aplicación y que no modifican la base de datos, ya que de eso se encarga modulo.py.

Tiene dos métodos, uno para obtener los valores aleatorios de la página web www.numeroalazar.com.ar mediante expresiones regulares y otro método para recuperar las gráficas de la base de datos externas cuando el cliente lo solicite.

PROBLEMAS ENCONTRADOS

En esta sección se detallan los problemas más importantes que han sido encontrados durante la realización de la presente práctica.

Instalación y manejo de MongoDB

Para poder crear la base de datos local, se instaló MongoDB. Se procedió según se especifica en la documentación en su página oficial, pero tras la instalación, el servicio “mongod” (que se encarga de escuchar en el puerto 27017 las conexiones con los clientes) no arrancaba. Observando los logs de la aplicación se determinó que había un problema de espacio en el disco (como se trataba de una máquina virtual, tenía mucho menos espacio de lo habitual). Se añadió al archivo de configuración una opción para ejecutar la aplicación en lugar de la manera que viene por defecto. En esta línea se indicaba que se utilizan archivos pequeños (una opción que por defecto está deshabilitada y hay que introducirla). Tras ello, al ejecutar `sudo service mongod start` ya arrancaba MongoDB de manera correcta, pudiendo así crear bases de datos, tablas y poder conectar MongoDB con la aplicación.

Conexión a internet para la máquina virtual (servidor)

El servidor inicialmente se alojaba en una máquina virtual Ubuntu mediante el programa Oracle VirtualBox. Este programa tiene diferentes opciones de red, pero para la aplicación era necesario una conexión tanto con internet (más allá del Gateway de la máquina local) como con la máquina local en la que se hicieron las solicitudes de prueba durante el desarrollo de la aplicación. Se experimentaron problemas con la configuración en modo puente para poder compartir misma red que la máquina local. Finalmente, la máquina virtual tuvo que ser exportada y se importó desde el programa VMware con la configuración de red en modo NAT, de esta forma, la aplicación se puede conectar a la máquina local (para las solicitudes) y a internet (para la base de datos externa y la obtención de números aleatorios) de manera simultánea. Sin embargo, no es posible, desde otra máquina diferente, acceder a la máquina virtual de VMware, por lo que las solicitudes a la página solo pueden ser desde la máquina local o desde sí misma mediante loopback.

Recarga de la página cada dos minutos

Para que la página que fue devuelta al cliente obtuviera diferentes números aleatorios cada dos minutos desde que se solicitó se usó la tecnología AJAX que utiliza JavaScript de manera asíncrona. De este modo, existe una función definida en JavaScript únicamente en el documento HTML perteneciente a la página principal, que se encarga de recargar la página cuando se cumplen los ciento veinte segundos desde que fuese recargada. Otra opción hubiera sido incluir un elemento en los metadatos del documento HTML con un atributo que indicase el tiempo en segundos que deben transcurrir para que se recargue la página.

Finalmente, se ha incluido un módulo `hilo.py` que se encarga de obtener números aleatorios cada dos minutos, por lo que la página web únicamente muestra el último número obtenido en la base de datos.

Doble ejecución del programa y módulo consultor

Si se activaba el modo de depuración en el módulo principal de la aplicación (insertando la línea `app.debug = True` en el main), la aplicación se ejecutaba de manera dual obteniendo dos valores aleatorios para una misma solicitud y devolviendo la página dos veces al cliente, por lo que el desarrollo fue llevado a cabo sin esa línea de código.

El hilo que recoge los números aleatorios es tratado como un daemon para que a la hora de parar el módulo principal, éste finalice de manera automática. Esto se ha conseguido añadiendo la línea de código `thread.daemon = True`, donde `thread` es la instancia del hilo consultor.

Inconsistencia entre bases de datos

Al generar números aleatorios, éstos son registrados en las bases de datos, sin embargo, no todos los números son registrados en las dos. Puede ocurrir que no se añadan por un motivo en concreto, por ejemplo, en la local, podría no insertarse si la conexión no fue abierta porque el servicio `"mongod"` no fue arrancado o, en la externa, porque no disponemos de internet. Inicialmente, la aplicación obtenía los números aleatorios en cada solicitud de los clientes, sin embargo, la API en uso gratuito de ThingSpeak no permite la inserción de datos sin que haya pasado un intervalo de aproximadamente veinte segundos. Finalmente, se ha introducido un hilo consultor que hace la inserción de datos cada dos minutos, por lo que cada solicitud ya no inserta datos en las bases de datos.

MANUALES

Para el cliente

Cuando el usuario solicita la dirección de la máquina servidora, debe especificar el puerto por defecto de Flask, que es 5000, en el navegador. Una vez hecho, la primera pantalla que aparece es la siguiente.

RandoMnize

192.168.196.129:5000

#Aleatorio = ; 89.34 !

Elija la opción que desee:

Escriba un número del 0 al 100 (incluidos) para saber cual es el último valor aleatorio obtenido menor que dicho número:

50 UMBRAL

Obtener la media de los numeros: MEDIA

Obtener graficas sobre la base de datos externa: GRAFICAS

En la página se aprecia en negrita y con una fuente más grande de lo normal el último número que ha sido obtenido de manera aleatoria por la aplicación.

Además de aparecer el número aleatorio, se encuentran las tres opciones de la aplicación, filtrar los registros de la base de datos por un número (menores que el umbral especificado), ver la media de los números y las gráficas de la base de datos externa. Al pulsar en cualquiera de los botones (en el primero habrá que introducir un número entre 0 y 100 anteriormente) el servidor nos devolverá la página para cada una de las funciones.

Si introducimos un valor para el umbral que no se encuentra entre 1 y 100, el elemento del formulario nos avisará de que corriamos el error antes de redirigirnos a la página del umbral.

Si cumplimos el requisito y pulsamos en el botón "UMBRAL" nos aparece la siguiente página.

150 UMBRAL

El valor debe ser inferior o igual a 100

Obtener graficas sobre la base de datos externa: GRAFICAS

100 (incluidos) para saber cual es el último valor aleatorio

-20 UMBRAL

El valor debe ser superior o igual a 0

Obtener graficas sobre la base de datos externa: GRAFICAS

Umbral de 2

192.168.196.129:5000

Umbral seleccionado: 2

[Ir a la BD local](#)
[Ir a la BD externa](#)
[Ir al final de la página](#)

BASE DE DATOS LOCAL

Existen un total de 7 / 272 numeros que son menores que 2 en la base de datos local.

El último número registrado menor que 2 es 1.35, que fue añadido el día 16/12/2017 - 15:40:55

Los demás números que cumplen dicha condición son los siguientes:

NÚMERO	FECHA
1.56	14/12/2017 - 20:4:16
0.94	14/12/2017 - 20:12:6
1.3	14/12/2017 - 20:14:13
1.46	15/12/2017 - 20:25:13
1.64	15/12/2017 - 20:46:59

La página muestra al usuario el umbral que ha seleccionado así como algunos datos como la cantidad de números que sean menores que el umbral o el último número registrado que pertenezca al rango especificado. Además, es incluye la lista de valores filtrados junto a sus fechas exactas.

Además de los registros pertenecientes a la base de datos local, también se muestran los registros de la base de datos externa, indicando las estadísticas así como la lista de números para ésta.

BASE DE DATOS EXTERNA

Existen un total de **6 / 237** numeros que son menores que 2 en la base de datos local.

El último número registrado menor que 2 es **1.35**, que fue añadido el día **2017-12-16T14:40:58Z**

Los demás números que cumplen dicha condición son los siguientes:

NÚMERO	FECHA
1.56	2017-12-14T19:04:53Z
0.94	2017-12-14T19:12:44Z
1.3	2017-12-14T19:14:52Z
1.64	2017-12-15T19:47:00Z
1.63	2017-12-16T11:47:59Z
1.35	2017-12-16T14:40:58Z

[Ir a la BD local](#)
[Ir a la BD externa](#)

VOLVER AL MENÚ PRINCIPAL

La página dispone además de enlaces para facilitar la navegación en esta, puesto que, si las tablas son demasiado grandes, se pierde dicha facilidad de navegación.

También dispone de un botón que nos lleva nuevamente a la página principal de la aplicación.

Si la opción elegida en el menú principal se trata de la media de valores, la página que es devuelta es esta:

Medida de valores

BASE DE DATOS LOCAL

Existen un total de 273 numeros en la base de datos local.

Los valores son los siguientes: [1.56, 73.27, 9.4, 44.76, 48.47, 38.23, 33.31, 35.79, 51.72, 24.63, 52.75, 86.25, 80.79, 83.81, 53.06, 97.95, 54.03, 82.51, 8.23, 25.58, 76.54, 46.62, 7.49, 34.48, 28.19, 29.2, 92.01, 13.21, 0.94, 82.55, 79.57, 7.47, 27.37, 27.66, 90.37, 97.62, 1.3, 95.3, 84.74, 78.29, 2.7, 96.79, 78.92, 94.28, 56.15, 36.0, 84.63, 93.96, 13.26, 50.01, 45.06, 76.54, 72.84, 11.05, 17.94, 5.94, 39.29, 14.68, 6.05, 28.72, 39.29, 14.68, 80.05, 6.05, 57.27, 5.46, 8.98, 96.25, 28.72, 91.14, 10.15, 30.61, 12.26, 73.0, 16.89, 51.26, 71.28, 51.42, 86.01, 16.71, 62.78, 13.92, 93.7, 55.07, 21.33, 13.92, 1.46, 5.94, 34.08, 56.88, 30.48, 23.93, 52.43, 19.96, 79.74, 99.0, 73.33, 50.44, 1.64, 70.18, 37.18, 95.39, 69.27, 89.65, 89.15, 70.58, 66.85, 82.52, 93.31, 74.42, 86.62, 86.5, 35.64, 50.91, 67.62, 3.24, 4.44, 10.32, 14.3, 2.9, 57.95, 85.42, 6.28, 18.24, 51.0, 73.12, 77.17, 50.52, 79.82, 31.63, 73.28, 40.12, 79.41, 12.91, 41.54, 50.99, 88.32, 19.73, 11.48, 51.51, 8.49, 77.4, 37.45, 10.46, 7.19, 12.38, 9.43, 38.04, 84.27, 57.47, 77.62, 58.93, 76.57, 3.51, 62.44, 91.44, 1.63, 66.52, 12.01, 41.54, 67.6, 36.19, 5.11, 79.72, 71.13, 45.91, 9.23, 34.67, 77.4, 97.68, 50.4, 32.45, 23.32, 70.51, 25.24, 47.67, 94.8, 35.26, 46.19, 10.49, 3.75, 45.6, 6.33, 62.98, 48.16, 70.18, 1.35, 80.48, 18.62, 17.81, 80.05, 77.14, 39.57, 14.05, 23.25, 10.67, 81.96, 9.69, 34.96, 87.79, 77.84, 70.61, 95.62, 97.4, 66.36, 6.46, 39.13, 29.25, 65.55, 14.71, 98.79, 9.99, 40.09, 36.37, 85.44, 85.02, 87.17, 19.61, 68.69, 69.15, 54.08, 20.55, 73.92, 93.21, 9.89, 52.31, 9.53, 82.52, 56.13, 47.64, 25.23, 25.85, 71.54, 96.22, 6.29, 51.94, 72.19, 50.26, 2.08, 8.07, 63.0, 8.06, 81.2, 20.78, 16.58, 99.06, 50.27, 17.89, 51.03, 7.62, 58.37, 72.91, 23.08, 11.01, 61.63, 83.98, 72.64, 82.06, 11.37, 12.39, 59.03, 6.07, 41.91, 72.45, 91.43, 89.34, 9.28, 27.52, 76.67, 58.7, 15.94, 90.82, 49.81, 99.54, 45.59]

Sumando en total la cantidad de 13224.88

Por lo que la media total de la base de datos local es: **48.4427838828**

BASE DE DATOS EXTERNA

Existen un total de 238 numeros en la base de datos externa.

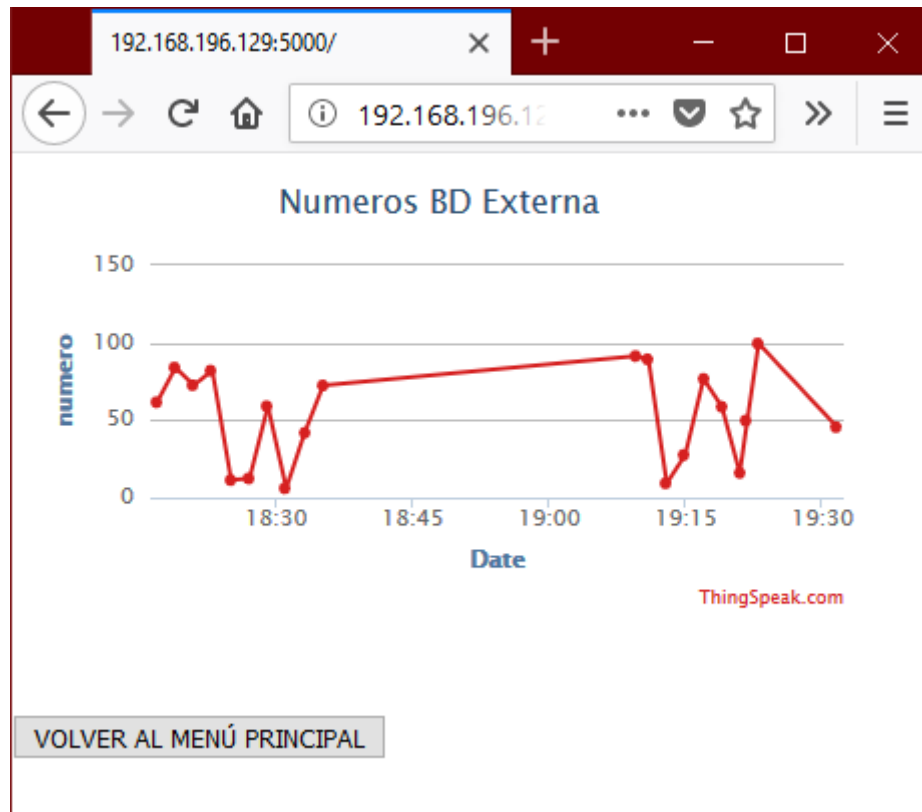
Los valores son los siguientes: [57.27, 5.46, 1.56, 73.27, 9.4, 44.76, 38.23, 33.31, 35.79, 51.72, 24.63, 52.75, 86.25, 80.79, 83.81, 53.06, 97.95, 54.03, 82.51, 8.23, 25.58, 76.54, 46.62, 7.49, 34.48, 28.19, 29.2, 92.01, 13.21, 0.94, 82.55, 79.57, 7.47, 27.37, 27.66, 90.37, 97.62, 1.3, 95.3, 84.74, 78.29, 2.7, 96.79, 78.92, 94.28, 56.15, 36.0, 84.63, 50.01, 45.06, 76.54, 72.84, 11.05, 17.94, 5.94, 39.29, 14.68, 6.05, 28.72, 91.14, 10.15, 12.26, 73.0, 51.26, 71.28, 51.42, 86.01, 16.71, 13.92, 93.7, 55.07, 21.33, 5.94, 34.08, 56.88, 30.48, 23.93, 52.43, 19.96, 79.74, 99.0, 50.44, 1.64, 70.18, 37.18, 69.27, 89.65, 66.85, 82.52, 74.42, 86.5, 35.64, 50.91, 67.62, 3.24, 4.44, 14.3, 85.42, 18.24, 77.17, 50.52, 31.63, 40.12, 79.41, 12.91, 41.54, 50.99, 88.32, 19.73, 11.48, 77.4, 37.45, 10.46, 7.19, 12.38, 38.04, 84.27, 57.47, 77.62, 76.57, 3.51, 62.44, 91.44, 1.63, 66.52, 12.01, 41.54, 67.6, 36.19, 5.11, 79.72, 71.13, 45.91, 9.23, 34.67, 77.4, 97.68, 50.4, 32.45, 23.32, 70.51, 25.24, 47.67, 94.8, 35.26, 46.19, 10.49, 3.75, 45.6, 6.33, 62.98, 48.16, 70.18, 1.35, 80.48, 18.62, 17.81, 80.05, 77.14, 39.57, 14.05, 23.25, 10.67, 81.96, 9.69, 34.96, 87.79, 77.84, 70.61, 95.62, 97.4, 66.36, 6.46, 39.13, 29.25, 65.55, 14.71, 98.79, 9.99, 40.09, 36.37, 85.44, 85.02, 87.17, 19.61, 68.69, 69.15, 54.08, 20.55, 73.92, 93.21, 9.89, 52.31, 9.53, 82.52, 56.13, 47.64, 25.85, 71.54, 6.29, 72.19, 50.26, 2.08, 8.07, 63.0, 8.06, 81.2, 20.78, 16.58, 99.06, 50.27, 17.89, 51.03, 7.62, 58.37, 72.91, 23.08, 11.01, 61.63, 83.98, 72.64, 82.06, 11.37, 12.39, 59.03, 6.07, 41.91, 72.45, 91.43, 89.34, 9.28, 27.52, 76.67, 58.7, 15.94, 49.81, 99.54, 45.59]

Sumando en total la cantidad de 11361.66

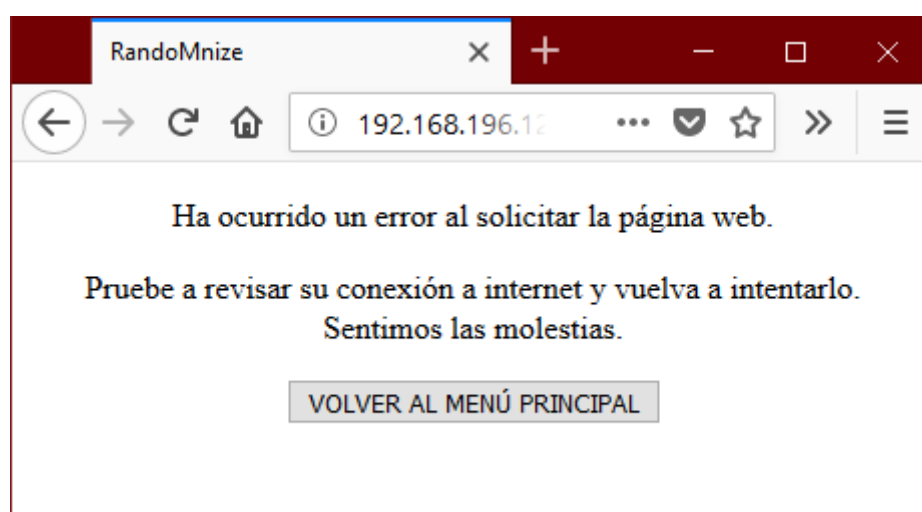
Por lo que la media total de la base de datos externa es: **47.7380672269**

Aquí aparecen todos los registros (sin la fecha) que fueron creados, todo ello para indicar al cliente en el último párrafo de cada sección, cuál es la media total de los números en las bases de datos.

La última opción del menú principal nos muestra el gráfico de los últimos números registrados en la base de datos externa, gracias a la API que proporciona ThingSpeak.



Si la aplicación no pudiese funcionar de manera correcta, al usuario le devolvería una página de error como ésta, junto a la posibilidad de volver al menú principal en caso de que los problemas fuesen resueltos.



Para el administrador

El administrador de la aplicación puede seguir el funcionamiento de ésta. Esto se consigue observando la salida de la ejecución de la aplicación en la terminal en que se ha ejecutado el comando. Situándonos en la carpeta "iroom", ejecutamos el módulo principal y será entonces cuando esa terminal nos avise por pantalla del uso de la aplicación, que al comienzo muestra el siguiente aviso:

```
servidor@servidormr: ~/iroom
(flask)servidor@servidormr:~/iroom$ python app.py
Abierta la conexion con la base de datos "dblocal" de mongodb://localhost:27017/.
Abierta la conexión con la base de datos de ThingSpeak
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

Los primeros avisos indican que las conexiones con las bases de datos han sido abiertas de manera satisfactoria. A partir de entonces, permanece en espera hasta que lleguen solicitudes. En la siguiente imagen se encuentra una solicitud de la máquina local (y no la máquina virtual que ejecuta la aplicación) a la página principal.

```
servidor@servidormr: ~/iroom
(flask)servidor@servidormr:~/iroom$ python app.py
Abierta la conexion con la base de datos "dblocal" de mongodb://localhost:27017/.
Abierta la conexión con la base de datos de ThingSpeak
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
Insertado a la BD LOCAL el número 27.76 (Sun Dec 17 19:47:48 2017)
Insertado a la BD EXTERNA el número 27.76 (Sun Dec 17 19:47:48 2017)
192.168.196.1 - - [17/Dec/2017 19:47:49] "GET / HTTP/1.1" 200 -
```

Se observa que, de manera remarcada, hay dos avisos, uno por cada base de datos, indicando que se ha podido registrar el número en la base de datos correspondiente.

También aparece la cabecera HTTP con el método (GET en este caso), la versión del protocolo y el código de repuesta (200, por lo que se ha devuelto de manera correcta).

Pasados dos minutos, sin que el cliente haya solicitado nuevamente la página web, ésta se recarga. Además, el hilo consultor solicita un nuevo número aleatorio y se añade a las bases de datos.

```
servidor@servidormr: ~/iroom
(flask)servidor@servidormr:~/iroom$ python app.py
Abierta la conexion con la base de datos "dblocal" de mongodb://localhost:27017/.
Abierta la conexión con la base de datos de ThingSpeak
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
Insertado a la BD LOCAL el número 27.76 (Sun Dec 17 19:47:48 2017)
Insertado a la BD EXTERNA el número 27.76 (Sun Dec 17 19:47:48 2017)
192.168.196.1 - - [17/Dec/2017 19:47:49] "GET / HTTP/1.1" 200 -
Insertado a la BD LOCAL el número 65.99 (Sun Dec 17 19:49:49 2017)
Insertado a la BD EXTERNA el número 65.99 (Sun Dec 17 19:49:49 2017)
192.168.196.1 - - [17/Dec/2017 19:49:52] "GET / HTTP/1.1" 200 -
```

Observe que el tiempo que ha pasado de una inserción a otra es de dos minutos más el tiempo que haya tardado la base de datos externa en realizar el registro (la base de datos local es bastante más rápida, ya que no necesita conectarse a internet).

Si existiera un problema con la solicitud, tal como no poder acceder a la página de la que se recogen los números aleatorios, el administrador será consciente de ello.

```

servidor@servidormr: ~/iroom
(flask)servidor@servidormr:~/iroom$ python app.py
Abierta la conexión con la base de datos "dblocal" de mongodb://localhost:27017/.
Abierta la conexión con la base de datos de ThingSpeak
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
ERROR: Ha ocurrido un error al intentar solicitar la página http://www.numeroalazar.com.ar/.
192.168.196.1 - - [17/Dec/2017 19:54:50] "GET / HTTP/1.1" 200 -

```

En este caso el código 200 indica que la página de error (vista en el manual de usuario para cliente) ha sido enviada al cliente, en lugar de la principal.

Otros errores pueden aparecer en la terminal como que no se pudo acceder a la base de datos local (si el servicio de MongoDB no se encuentra activo) o a la externa (si no tiene conexión a internet) o si el número no ha podido registrarse en una de éstas.

```

servidor@servidormr: ~/iroom
(flask)servidor@servidormr:~/iroom$ python app.py
Abierta la conexión con la base de datos "dblocal" de mongodb://localhost:27017/.
Abierta la conexión con la base de datos de ThingSpeak
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
Insertado a la BD LOCAL el número 29.17 (Sun Dec 17 20:01:59 2017)
Insertado a la BD EXTERNA el número 29.17 (Sun Dec 17 20:01:59 2017)
192.168.196.1 - - [17/Dec/2017 20:01:59] "GET / HTTP/1.1" 200 -
Insertado a la BD LOCAL el número 76.83 (Sun Dec 17 20:02:02 2017)
ERROR: no se pudo registrar 76.83 en la BD externa
192.168.196.1 - - [17/Dec/2017 20:02:02] "GET / HTTP/1.1" 200 -

```

Al finalizar la ejecución de la aplicación (ejecutando CTRL+C), aparecen dos mensajes indicando que las conexiones a ambas bases de datos han sido cerradas con éxito y de manera segura.

```

servidor@servidormr: ~/iroom
(flask)servidor@servidormr:~/iroom$ python app.py
Abierta la conexión con la base de datos "dblocal" de mongodb://localhost:27017/.
Abierta la conexión con la base de datos de ThingSpeak
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
Insertado a la BD LOCAL el número 36.54 (Sun Dec 17 20:04:52 2017)
Insertado a la BD EXTERNA el número 36.54 (Sun Dec 17 20:04:52 2017)
192.168.196.1 - - [17/Dec/2017 20:04:53] "GET / HTTP/1.1" 200 -
Insertado a la BD LOCAL el número 22.85 (Sun Dec 17 20:06:53 2017)
Insertado a la BD EXTERNA el número 22.85 (Sun Dec 17 20:06:53 2017)
192.168.196.1 - - [17/Dec/2017 20:06:54] "GET / HTTP/1.1" 200 -
Insertado a la BD LOCAL el número 4.48 (Sun Dec 17 20:08:54 2017)
Insertado a la BD EXTERNA el número 4.48 (Sun Dec 17 20:08:54 2017)
192.168.196.1 - - [17/Dec/2017 20:08:55] "GET / HTTP/1.1" 200 -
^C
Cerrada la conexión con mongodb://localhost:27017/
Cerrada la conexión con ThingSpeak
(flask)servidor@servidormr:~/iroom$

```