

Taller de proyecto II

Práctica 1

Fecha: 14/09/2017

Autores:

- Basanta, Sofia 524/1
- Discoli, Tomas 543/4
- Minig Traverso, Marcelo 489/7

Índice

Enunciado	2
Resolución de ejercicios	
Ejercicio 1	3
Ejercicio 2	4
Ejercicio 3	7
Ejercicio 4	8
Ejercicio 5	9
Ejercicio 6	10

Enunciado

- 1) Generar el archivo 'requirements.txt' con las dependencias necesarias para poder levantar un servidor con Flask. Explicar un ejemplo de uso con la secuencia de acciones y procesos involucrados desde el inicio de la interacción con un usuario hasta que el usuario recibe la respuesta.
- 2) Desarrollar un experimento que muestre si el servidor HTTP agrega o quita información al generar un programa Python. Nota: debería programar o utilizar un programa Python para conocer exactamente lo que genera y debería mostrar la información que llega del lado del cliente, a nivel de HTTP o, al menos, a nivel de HTML (preferentemente HTTP).
- 3) Generar un proyecto de simulación de acceso a valores de temperatura, humedad, presión atmosférica y velocidad del viento.
 - a) Un proceso simulará una placa con microcontrolador y sus correspondientes sensor/es o directamente una estación meteorológica proveyendo los valores almacenados en un archivo o en una base de datos. Los valores se generan periódicamente (frecuencia de muestreo).
 - b) Un proceso generará un documento HTML conteniendo:
 - i) Frecuencia de muestreo
 - ii) Promedio de las últimas 10 muestras
 - iii) La última muestra
 - c) El documento HTML generado debe ser accesible y responsivo.

Aclaración: Se deberá detallar todo el proceso de adquisición de datos, cómo se ejecutan ambos procesos (ya sea threads o procesos separados), el esquema general, las decisiones tomadas en el desarrollo de cada proceso y la interacción del usuario.

- 4) Agregar a la simulación anterior la posibilidad de que el usuario elija entre un conjunto predefinido de períodos de muestreo (ej: 1, 2, 5, 10, 30, 60 segundos). Identifique los cambios a nivel de HTML, de HTTP y de la simulación misma.
- 5) Comente la problemática de la concurrencia de la simulación y específicamente al agregar la posibilidad de cambiar el período de muestreo. Comente lo que estima que podría suceder en el ambiente real ¿Podrían producirse problemas de concurrencia muy difíciles o imposibles de simular? Comente brevemente los posibles problemas de tiempo real que podrían producirse en general.
- 6) ¿Qué diferencias supone que habrá entre la simulación planteada y el sistema real? Es importante para planificar un conjunto de experimentos que sean significativos a la hora de incluir los elementos reales del sistema completo.

Ejercicio 1

Primeramente, debíamos instalar *python* y *pip*, que es un sistema de gestión de paquetes utilizado para instalar y administrar paquetes de software escritos en Python. Para lo cual ejecutamos los siguientes comandos en consola:

```
sudo apt-get install python
sudo apt-get install python-pip
```

A partir de estas instalaciones, creamos el archivo “requirements.txt”. Dicho archivo contiene las dependencias necesarias para poder levantar un servidor con Flask. Cuyo contenido se muestra a continuación:

requirements.txt
Flask Flask-mysqldb

Para instalar las dependencias necesarias, ejecutamos la instrucción:

```
pip install -r requirements.txt
```

Asimismo, instalamos para administrar la base de datos *mySQL* a partir de los siguientes comandos:

```
sudo apt-get install mysql-server libmysqlclient-dev
```

Creamos una carpeta con los archivos requeridos para levantar un servidor Flask descargandolos del repositorio. Por último ejecutamos el comando para crear y configurar la base de datos:

```
mysql -u root -p < DB.sql
```

Al ejecutar el siguiente comando iniciamos el servidor y lo podemos ver en el puerto `http://127.0.0.1:8000/`

```
python app.py
```

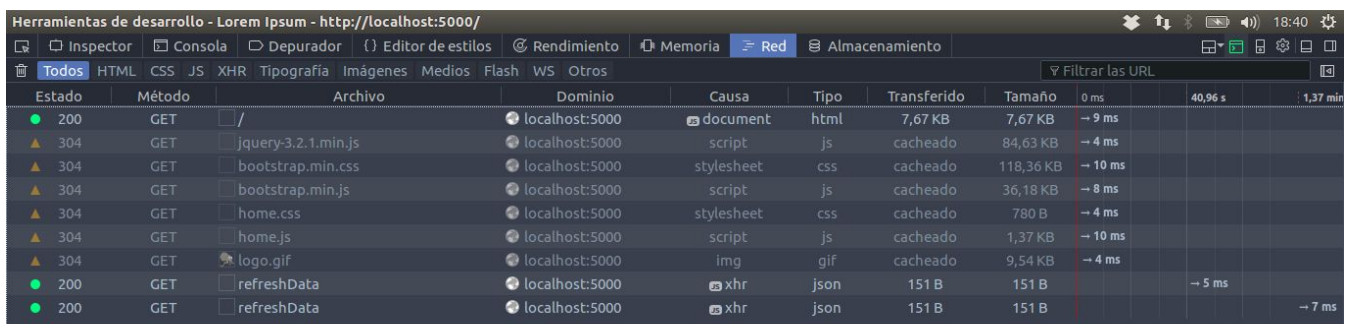
La aplicación que se encuentra en el repositorio ejemplifica la utilización de Flask con un par de rutas y el procesamiento de información recibida por un formulario HTML. La información se procesa y persiste en una base de datos MySQL para poder ser solicitada y expuesta de la forma que se desee.

Cuando el cliente desde el navegador web solicita la url `http://127.0.0.1:8000/` se genera un paquete HTTP con una petición para el servidor. Luego el servidor procesa la petición y devuelve al cliente una respuesta con lo solicitada dentro de otro paquete HTTP. Esto se ve con más detalle en la sección siguiente.

Ejercicio 2

Para resolver este ejercicio se utiliza la aplicación del ejercicio anterior y utilizamos dos programas para analizar y ver la interacción entre el navegador y el servidor de la aplicación. Estos programas son: primero Wireshark que es un analizador de protocolos utilizado para realizar análisis en redes de comunicaciones, para desarrollo de software y protocolos. Y también la herramienta de desarrollador que proporciona el Navegador Web, especialmente el monitor de red.

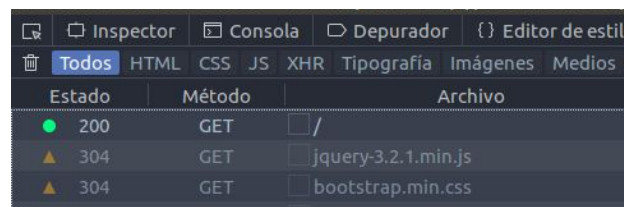
En la siguiente imagen se puede ver todas la peticiones que se hacen al servidor al cargar la página actual, el tiempo total y el tamaño total de todos los elementos al cargarla, así como las cabeceras enviadas y recibidas, códigos de estado y tiempo de respuesta de cada petición.



Estado	Método	Archivo	Dominio	Causa	Tipo	Transferido	Tamaño	0 ms	40,96 s	1,37 min
200	GET	/	localhost:5000	document	html	7,67 KB	7,67 KB	→ 9 ms		
304	GET	jquery-3.2.1.min.js	localhost:5000	script	js	cacheado	84,63 KB	→ 4 ms		
304	GET	bootstrap.min.css	localhost:5000	stylesheet	css	cacheado	118,36 KB	→ 10 ms		
304	GET	bootstrap.min.js	localhost:5000	script	js	cacheado	36,18 KB	→ 8 ms		
304	GET	home.css	localhost:5000	stylesheet	css	cacheado	780 B	→ 4 ms		
304	GET	home.js	localhost:5000	script	js	cacheado	1,37 KB	→ 10 ms		
304	GET	logo.gif	localhost:5000	img	gif	cacheado	9,54 KB	→ 4 ms		
200	GET	refreshData	localhost:5000	xhr	json	151 B	151 B		→ 5 ms	
200	GET	refreshData	localhost:5000	xhr	json	151 B	151 B			→ 7 ms

Imagen 1 - Interacción entre el navegador y el servidor.

En la imagen 2 se puede observar un zoom a la tabla, para observar los dos códigos de respuesta, donde el 200 indica una respuesta correcta y el 304 dice que la respuesta ha sido encontrada en la caché, es decir Indica que la petición a la URL no ha sido modificada desde que fue requerida por última vez.



Estado	Método	Archivo
200	GET	/
304	GET	jquery-3.2.1.min.js
304	GET	bootstrap.min.css

Imagen 2 - Detalle de códigos de respuesta

Si nos colocamos sobre una de estas entradas podremos ver en detalle las cabeceras tanto de la petición como de su respuesta. En este punto vemos como el navegador y el servidor agregan información a los paquetes HTTP sobre la transacción en curso. Podemos ver que dicha información se muestra mediante la sintaxis “Clave: Valor”.

Algunas de ellas son:

- Host: indica el nombre del dominio o IP.
- User Agent: contiene información como navegador web, sistema operativo, etc.
- Accept: tipos de contenido que acepta, ej: texto plano, HTML.
- Accept-Language: indica el o los idiomas que acepta.
- Accept-Encoding: indica los tipos de codificación que acepta.
- Referer: indica la dirección URL de donde proviene.
- Cookie: muestra cookies usadas previamente.

- Connection: especifica el tipo de conexión.
- Content-Type: tipo de contenido de la petición en POST.
- Content-Length: el tamaño del contenido de la petición en bytes.
- Date: fecha y hora de la petición.

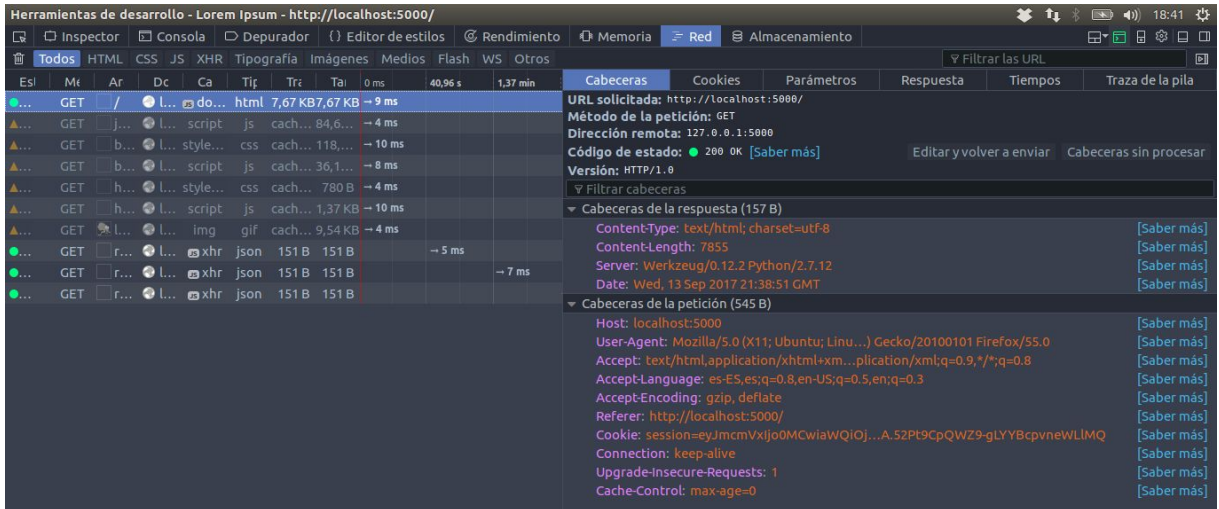


Imagen 3 - Cabeceras de la primera petición.

Al abrir la respuesta a la primera petición, podemos observar que se obtiene un documento HTML. El mismo coincide con el archivo layout.html, al cual se le insertó el header, el body y el footer que forman el contenido de la página solicitada.

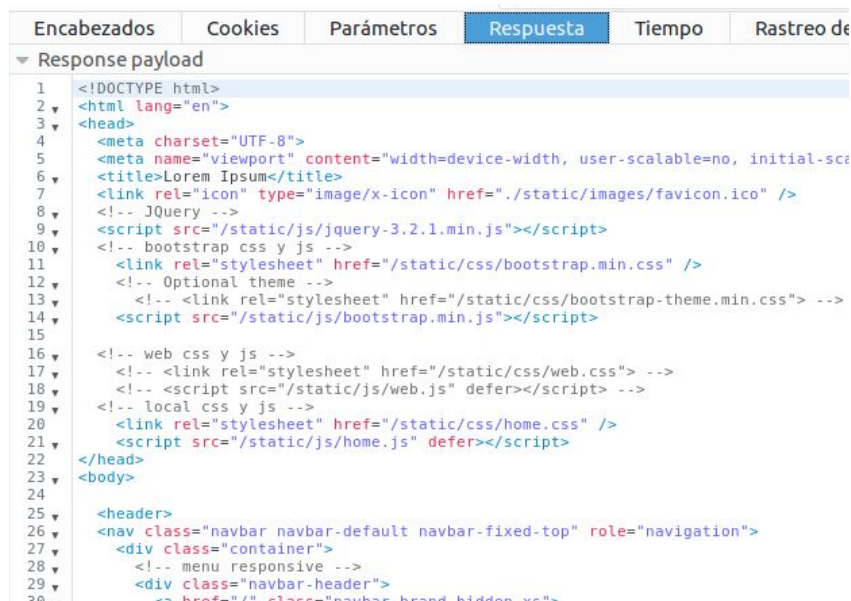


Imagen 4 - Detalle de la respuesta a la petición.

Lo dicho hasta ahora explica como se establece la comunicación entre el servidor y el cliente mediante el protocolo HTTP, es decir dentro de lo que se conoce como la Capa de Aplicación.

Se puede analizar un poco más en profundidad la comunicación yendo a la Capa de Transporte y ver particularmente el protocolo TCP como en la siguiente imagen.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	53934→8000 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 S
2	0.000023000	127.0.0.1	127.0.0.1	TCP	74	8000→53934 [SYN, ACK] Seq=0 Ack=1 Win=43690 Len=0
3	0.000038000	127.0.0.1	127.0.0.1	TCP	66	53934→8000 [ACK] Seq=1 Ack=1 Win=43776 Len=0 TSval
4	0.000166000	127.0.0.1	127.0.0.1	HTTP	417	GET / HTTP/1.1
5	0.000195000	127.0.0.1	127.0.0.1	TCP	66	8000→53934 [ACK] Seq=1 Ack=352 Win=44800 Len=0 TSv
6	0.020479000	127.0.0.1	127.0.0.1	TCP	83	[TCP segment of a reassembled PDU]
7	0.020499000	127.0.0.1	127.0.0.1	TCP	66	53934→8000 [ACK] Seq=352 Ack=18 Win=43776 Len=0 TS
8	0.020514000	127.0.0.1	127.0.0.1	TCP	106	[TCP segment of a reassembled PDU]
9	0.020527000	127.0.0.1	127.0.0.1	TCP	66	53934→8000 [ACK] Seq=352 Ack=58 Win=43776 Len=0 TS
10	0.020535000	127.0.0.1	127.0.0.1	TCP	88	[TCP segment of a reassembled PDU]
11	0.020539000	127.0.0.1	127.0.0.1	TCP	66	53934→8000 [ACK] Seq=352 Ack=80 Win=43776 Len=0 TS
12	0.020555000	127.0.0.1	127.0.0.1	TCP	104	[TCP segment of a reassembled PDU]
13	0.020559000	127.0.0.1	127.0.0.1	TCP	66	53934→8000 [ACK] Seq=352 Ack=118 Win=43776 Len=0 T
14	0.020581000	127.0.0.1	127.0.0.1	TCP	103	[TCP segment of a reassembled PDU]
15	0.020586000	127.0.0.1	127.0.0.1	TCP	66	53934→8000 [ACK] Seq=352 Ack=155 Win=43776 Len=0 T
16	0.020593000	127.0.0.1	127.0.0.1	TCP	68	[TCP segment of a reassembled PDU]
17	0.020602000	127.0.0.1	127.0.0.1	TCP	66	53934→8000 [ACK] Seq=352 Ack=157 Win=43776 Len=0 T
18	0.020607000	127.0.0.1	127.0.0.1	HTTP	6794	HTTP/1.0 200 OK (text/html)
19	0.020624000	127.0.0.1	127.0.0.1	TCP	66	53934→8000 [ACK] Seq=352 Ack=6885 Win=174720 Len=0
20	0.020639000	127.0.0.1	127.0.0.1	TCP	66	8000→53934 [FIN, ACK] Seq=6885 Ack=352 Win=44800 L
21	0.020712000	127.0.0.1	127.0.0.1	TCP	66	53934→8000 [FIN, ACK] Seq=352 Ack=6886 Win=174720
22	0.020724000	127.0.0.1	127.0.0.1	TCP	66	8000→53934 [ACK] Seq=6886 Ack=353 Win=44800 Len=0

Se puede observar perfectamente, en estos 22 paquetes capturados con wireshark, como es que el cliente (puerto 53934) y el servidor (puerto 8000) establecen una conexión TCP del paquete 1 al 3 mediante el pedido de conexión del cliente con el paquete de sincronización [SYN], la respuesta [SYN, ACK] del servidor y la confirmación final del cliente [ACK].

Luego el cliente manda la petición GET / HTTP/1.1 solicitando por HTTP el recurso "/" al servidor. El cual es transmitido por partes en los paquetes del 6 al 18 [TCP segment of a reassembled PDU] llegando en su totalidad al cliente en el paquete 18 donde se ve que lo que recibió fue un HTTP/1.0 200 OK con un contenido de texto HTML.

Ante cada parte que recibe el cliente manda la correspondiente confirmación [ACK] para avisarle al servidor que le están llegando los datos.

Por último el servidor como ya envió todos los datos procede a finalizar la conexión con el paquete [FIN, ACK] y el cliente le responde con [FIN, ACK] y [ACK].

Lo siguiente que el cliente hace es analizar el archivo HTML y repetir el proceso para solicitar al servidor el resto de los recursos necesarios para cargar la página como archivos CSS, JS, imágenes, etc.

Las capturas de pantalla y otra información se puede ver con más detalle en la carpeta Info del repositorio.

Ejercicio 3

Para este ejercicio realizamos dos aplicaciones. La primera se encarga de la simulación de un microcontrolador. La cual se puede encontrar en el repositorio con el nombre de **micro_sim.py**. En dicha aplicación se generan periódicamente muestras de temperatura, presión, humedad y velocidad del viento para ser almacenadas en una base de datos junto con el tiempo en que fueron generadas.

Por otra parte, una segunda aplicación denominada **app.py**. Se encarga de tomar esas muestras, procesarlas y exponerlas mediante una interfaz web a medida que se van generando.

De esta forma el sistema completo simula la interacción entre una aplicación con servicios web y un microcontrolador con acceso a la misma base de datos.

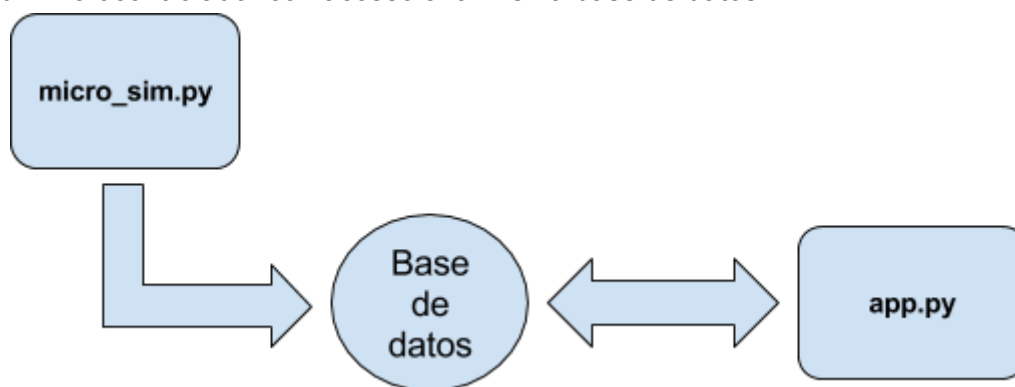


Imagen 5 - Diagrama del sistema completo.

Microcontrolador:

El archivo **micro_sim.py** se encarga de simular el microcontrolador, generando las muestras cada una frecuencia fija de 5 segundos a partir de unos valores iniciales preseleccionados y la utilización de Perlin Noise para crear cambios pseudo aleatorios más realistas. Luego de guardar cada muestra en la base de datos, el proceso se duerme los segundos necesarios hasta tener que generar la próxima “medición”.

La función Perlin Noise es una función matemática que genera números de forma pseudo-aleatoria. Lo que nos llevó a utilizar esta función fue que al programar comportamientos realistas la aleatoriedad no es necesariamente natural. Perlin Noise tiene un aspecto más orgánico porque produce una secuencia naturalmente ordenada (“suave”) de números pseudo-aleatorios. De esta manera podemos simular un microcontrolador donde, por ejemplo, la temperatura varía gradualmente.

Servidor:

El servidor web expone las muestras que el microcontrolador genera en una página web que recarga automáticamente cada 5 segundos para mostrar los nuevos datos.

Ejercicio 4

Es una mejora de las aplicaciones del ejercicio anterior donde se le da al usuario la posibilidad de cambiar la frecuencia de muestreo del microcontrolador como también de apagarlo. Al mismo tiempo se admite múltiples usuarios con distintas frecuencias y la posibilidad de estar logueado o no. Para esto se necesita que el microcontrolador sepa manejar los posibles inconvenientes que se puedan generar y que el servidor web haga un manejo de los usuarios registrados. El sistema completo continúa simulando la interacción entre una aplicación con servicios web y un microcontrolador con acceso a la misma BD.

Microcontrolador:

El archivo **micro_sim.py** se encarga de simular el microcontrolador, generando las muestras a partir de unos valores iniciales preseteados y la utilización de Perlin Noise para crear cambios pseudo aleatorios más realistas. La cantidad de muestras por segundo que se genera varía entre los valores 1, 5, 10, 20 y 40. Como este valor es elegido por los usuarios, el microcontrolador busca en la BD entre los datos de los usuarios y como necesita satisfacer las peticiones de todos, genera las muestras a la mayor velocidad solicitada. Luego de guardar cada muestra en la BD el proceso se duerme los segundos necesarios hasta tener que generar la próxima.

El microcontrolador también cuenta con la posibilidad de estar apagado, ya sea porque todos los usuarios decidieron apagarlo o porque no hay ninguno logueado. Mientras un usuario lo mantenga encendido el microcontrolador va a seguir generando las muestras.

El servidor web:

La forma más fácil de entender como fue implementado el servidor es a partir de las rutas que lo conforman.

- **/** : Es la página principal donde el usuario puede hacer login o logout, contiene los formularios para que el usuario logueado encienda y apague el micro, como para que elija la frecuencia de muestreo. También expone al usuario las últimas muestras realizadas por el microcontrolador, actualizándose mediante AJAX cada la frecuencia elegida. Hay que aclarar que si varios usuarios registrados tiene seteadas distintas frecuencias, cada uno ve las muestras de la frecuencia que eligió. Si el usuario **A** eligió 1 muestra por segundo y el **B** una cada 5 segundos, el **A** va a ver las muestras 1, 2, 3, 4, 5, 6, 7, 8, 9 y 10 mientras que el **B** sólo las 5 y 10.
- **/refreshData** : Esta ruta se usa para solicitar los datos de las últimas muestras al servidor. Funciona como una API, recibe una solicitud GET, busca la información en la BD y la devuelve en formato JSON al usuario.
- **/form_power** : Recibe las solicitudes del usuario de encender o apagar el microcontrolador. Esta informacion se guarda en la BD con los datos del usuario para que el microcontrolador pueda hacer uso de ella.
- **/form_sense/** : Se encarga de procesar el formulario en el que un usuario solicita cambiar su frecuencia de muestreo. Dicha frecuencia es actualizada en la BD junto con los datos de ese usuario.
- **/develop** : Página sin contenido.

- **/login** : Es la ruta a través de la cual el usuario se loguea y se le crea automáticamente un usuario para la sesión actual con un nombre y frecuencia de muestreo fijas. Luego se persiste esta información en la BD para que el microcontrolador pueda hacer uso de ella y se redirecciona a **/**.
- **/logout** : En esta ruta se elimina el usuario actual tanto de la session como de la BD y se redirecciona hacia **/** a la espera de un nuevo logueo.

Base de Datos:

El archivo **BD.sql** contiene todos los órdenes MySQL necesarias para crear y utilizar la BD del sistema.

La BD cuenta con dos tablas *samples* donde **micro_sim.py** guarda las muestras generadas para que **app.py** las lea y *config* donde se almacenan los usuarios y su información de control sobre el microcontrolador. Para una mejor administración de la BD cada aplicación tiene su propio usuario con los permisos necesarios para hacer uso de las tablas. Los usuarios son *flaskapp1_3_mic* y *flaskapp1_3_app*.

Ejercicio 5

En la simulación anterior se necesita que dos procesos que tienen acceso a una misma bases de datos cooperen. El problema del ejercicio se debe a que se necesitaba modelar un proceso que produce datos (**micro_sim.py** genera periódicamente muestras de presión, temperatura, humedad y velocidad del viento); y otro que debe tomar (simultáneamente) esos datos y los muestra al usuario en una página web. A esta problemática se la denomina: problema del productor-consumidor. Para solucionar esto se debe proporcionar:

- Mecanismos de sincronización y comunicación entre procesos.
- Ejecución ordenada.

El problema consiste en que el productor no produzca un dato después de que el consumidor haya tomado un dato, ya que se estaría mostrando una medición anterior a la última disponible. Pero tampoco que el consumidor tome un dato antes de que se haya producido un dato nuevo.

Como existe la posibilidad de que haya múltiples usuarios, se tiene que idear un mecanismo para que el microcontrolador pueda satisfacer la frecuencia de muestra solicitada por todos. Esto podría llegar a generar algún problema si no se tiene en cuenta.

Ejercicio 6

La diferencia principal entre la simulación planteada y el sistema real es que estos últimos no son perfectos. Los mismos tienen un tiempo de arranque hasta que están operativos, mientras que el simulador del microcontrolador, tiene un inicio más rápido. El sistema real es susceptible a ruidos y a diversas condiciones exteriores que el proceso simulado no tiene en principio, aunque algunas se podrían simular. Por ejemplo, algunas mediciones pueden tener saltos o picos, que distorsionan los promedios. Puede haber problemas externos como falta de electricidad o en el medio de comunicación, por ejemplo: que el Wifi no funcione correctamente, por lo que el productor no tiene dónde informar nuevas muestras.

Básicamente la importancia de planificar un conjunto de experimentos que sean significativos a la hora de incluir elementos reales nos permite anticiparnos a los problemas, que intentar resolverlos en una etapa de implementación puede resultar más tedioso y más costoso. Además, nos permite proponer diferentes mecanismos para solucionar estos problemas, analizando la eficiencia de cada uno, comparando sus ventajas y limitaciones.

En los archivos README.md del repositorio se encuentra toda la documentación sobre las aplicaciones, su implementación, funcionamiento y cómo ejecutarlas.