

Universidad Nacional de La Plata  
Facultad de Informática



Taller de Proyecto II

Resolución de Trabajo Práctico N°1

Autores: \_\_\_\_\_ Nro Alumno:

Castillo Amarís

712/4

Risso Agustín

541/1

Año Lectivo: \_\_\_\_\_

2017

Fecha de entrega:

14/09

Índice:

Objetivos.....	2
Levantamiento de un servidor Flask.....	3
Captura de paquetes HTTP.....	5
Simulación de una placa de sensado.....	9
Intervalo de muestreo modificable.....	11
Problemática de concurrencia.....	12
Diferencias respecto a una placa de sensado real.....	13

# Objetivos

El propósito del presente informe es la familiarización con Python, lenguaje que se destaca por su sencillez, legibilidad y precisión de sintaxis. A su vez, se aprovecha el hecho de que mediante Flask, framework escrito en Python, se pueden crear aplicaciones web rápidamente y con un mínimo número de líneas. Por ende, con tales herramientas, levantar un servidor se vuelve una tarea mucho más simple y rápida.

Entre otros objetivos se encuentra el poder hacer una introducción a los conceptos de aplicaciones web, la comunicación entre el servidor y los clientes, manejo de HTML, Bootstrap, Javascripts, protocolos de comunicación, etc.

También se hace un hincapié en conocer los problemas de concurrencia que se pueden presentar y la diferencia que existe entre las simulaciones y los sistemas implementados físicamente.

## **Inciso N°1: Levantamiento de un servidor Flask.**

Resumen del enunciado:

Se desea poder levantar un servidor con Flask, generando el archivo requirements.txt que contenga las dependencias necesarias para realizar dicha tarea.

Resolución:

Para desarrollar la aplicación se decidió utilizar como IDE el Visual Studio 2017 y se descargo el desarrollador de Python para la edición, depuración, desarrollo y control de código fuente Python.

También, se configuró el IDE para que todo los proyectos tengan por defecto Python 2.7, que es la versión del lenguaje que será utilizada para el desarrollo. Luego, se creó un proyecto web de Python genérico nombrado "TP1E1". Una vez creado, a partir del gestor de paquetes de Python PyPi(Python Package Index), se instaló Flask desde un archivo "requirements.txt" usando el comando "pip install flask".

Los paquetes instalados fueron los siguientes:

- Flask: Framework minimalista escrito en python.
- Itsdangerous: Módulo que provee un puerto para el registro de Django
- Click: Paquete para crear líneas de comando amigables
- Werkzeug: Libreria WSGI(Web Service Gateway Interface o Interfaz de puerta de enlace del servidor web) para python. Cumple el rol de comunicador o enrutador entre la aplicación Python y el servidor tanto en ambientes de desarrollo como de producción.
- Jinja2: Es un lenguaje de plantillas para Python, basado en las plantillas de Django
- Virtualenv: Utilidad para aislar recursos como librerías y entorno de ejecución, del sistema principal o de otros entornos virtuales.

La realización los pasos mencionados permitió que se pudiese levantar el servidor Flask.

Una vez configurado Visual Studio e instalado todos paquetes y dependencias necesarias para correr flask dentro del proyecto "TP1E1", se creó un archivo Python index.py configurado como archivo de inicio del proyecto, que importa la clase Flask.

Luego, se creó una instancia de la clase, a la que se le pasa como argumento el nombre del módulo (index). En este caso, como solamente hay un módulo, se le pasa "\_\_name\_\_" como parámetro. Después, se usa la función route() para saber la URL (en este caso '/') que dispara la función, para este ejemplo, hello\_world(), que imprime en pantalla "Hello World!". Finalmente, se configura al localhost como servidor, utilizando el puerto 80.

Secuencia de acciones (Figura 1):

Cuando un usuario ejecuta una petición desde un navegador, se activa un archivo de configuración de URLs (viene por defecto con Flask) y lo redirecciona a una determinada vista. Esta puede interaccionar, ya sea con una base de datos, un archivo, u otra fuente de datos para obtenerlo. También puede hacer el llamado a una plantilla(template), que

renderiza la respuesta a la solicitud del navegador, o simplemente se comunica con la plantilla para renderizarla.

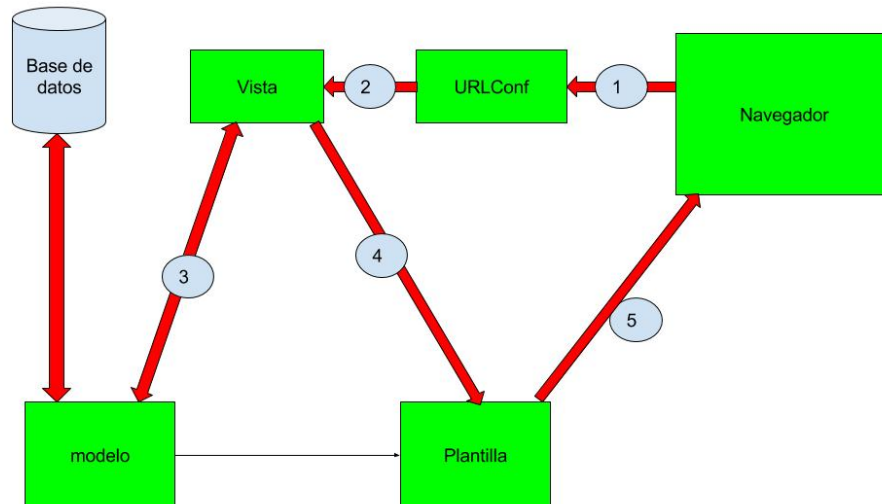


Figura 1. Diagrama de acciones y procesos desde que un usuario inicia una petición hasta que el servidor responde.

Se indicó que la ruta “ / ” es la que contiene la función "hello\_world" que devuelve el mensaje escrito.

Dicha aplicación es ejecutada en el localhost, indicando la IP y el puerto, lo que crea el servidor web instantáneamente.

Al ejecutar el programa se abre una nueva pestaña en el navegador predeterminado (en este caso, Chrome), pudiéndose visualizar el mensaje “Hello, World!”. (Figura 2).

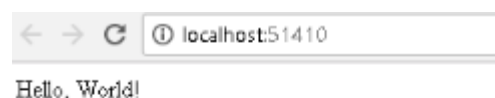


Figura 2. Visualización de la aplicación hello world en ejecución.

## Inciso N°2: Captura de paquetes HTTP.

Resumen del enunciado:

Se desea poder capturar el mensaje de respuesta que llega al lado del cliente al nivel HTTP, para verificar si el servidor agrega o quita información al generar un programa Python.

Resolución:

Se utilizaron los siguientes programas para la realización de la captura de los paquetes HTTP:

- RawCap: es un Sniffer portable de red que usa sockets raw, el cual permite realizar el sniffado de diferentes interfaces, en este caso, el localhost. Como único requerimiento, se necesita tener instalado el Framework 2.0 de .NET. Una vez descargado el .exe, se ejecuta, pasándole como parámetros la IP del localhost (127.0.0.1) y el fichero destino.  
Link de descarga: <http://www.netresec.com/?page=RawCap>.
- Wireshark: Analizador de paquetes de red que muestra los datos en un entorno gráfico, de forma amigable. A su vez, solicita para su correcto funcionamiento, la instalación de WinPcap. Dicha herramienta permite acceder a la conexión entre capas de red en entornos Windows.  
Link de descarga: <https://www.wireshark.org/download.html>.

Descripción de la aplicación:

Se creó una aplicación que permite sumar dos números entre 1 y 100.

Al igual que en el inciso 1, se importó Flask y en la ruta “ / ” se definió la función “suma()” que permite realizar dicha operación.

A parte, se creó la carpeta “templates” que contiene la vista “suma.html” y la carpeta scripts, que contiene todo los archivos para el manejo de bootstrap y javascript (utilizadas para el desarrollo de interfaces). La aplicación suma() es la encargada de realizar la redirección hacia dicha vista.

“Suma.html” consta de un formulario con dos campos tipo number y un botón que envía los datos seteados en los campos por el método POST, de manera que no pueden ser vistos en la URL, disparando la acción form\_action() dentro del archivo app.py que realiza la suma.(Figura 3).

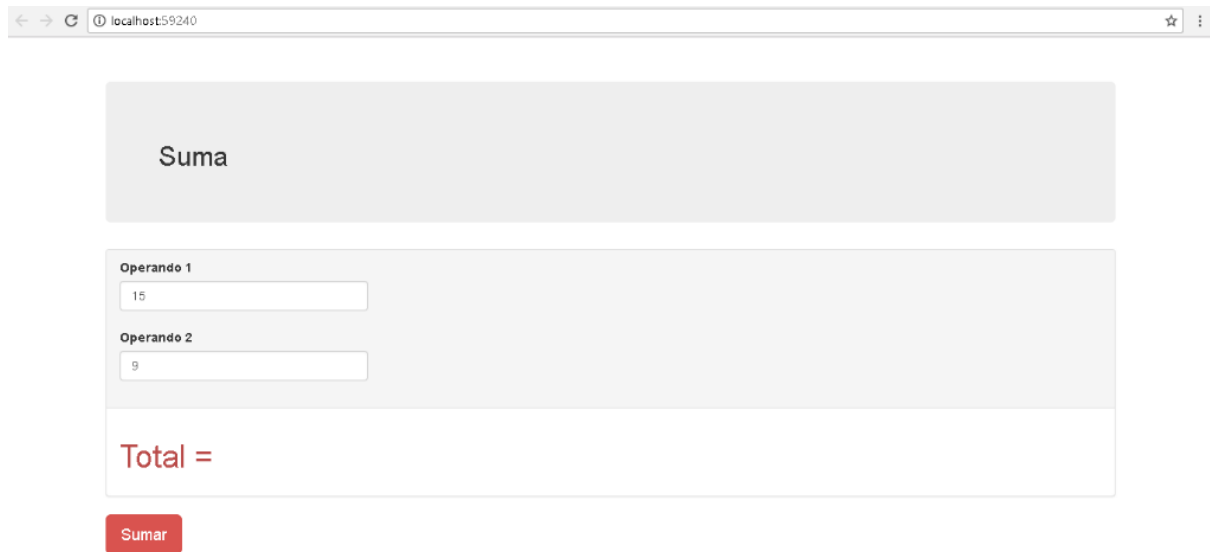


Figura 3. Visualización de la aplicación suma en ejecución.

Al ejecutar el programa Python, RawCap captura los paquetes generados a través de la interfaz loopback y guarda el resultado en el fichero especificado.

Cabe aclarar que se decide utilizar RawCap, ya que Wireshark no captura el tráfico de paquetes entre la máquina y ella misma, debido a que dicho tráfico no es enviado por una interfaz de red real.

El fichero que contiene los paquetes capturados, es luego abierto por el Wireshark, para su análisis.

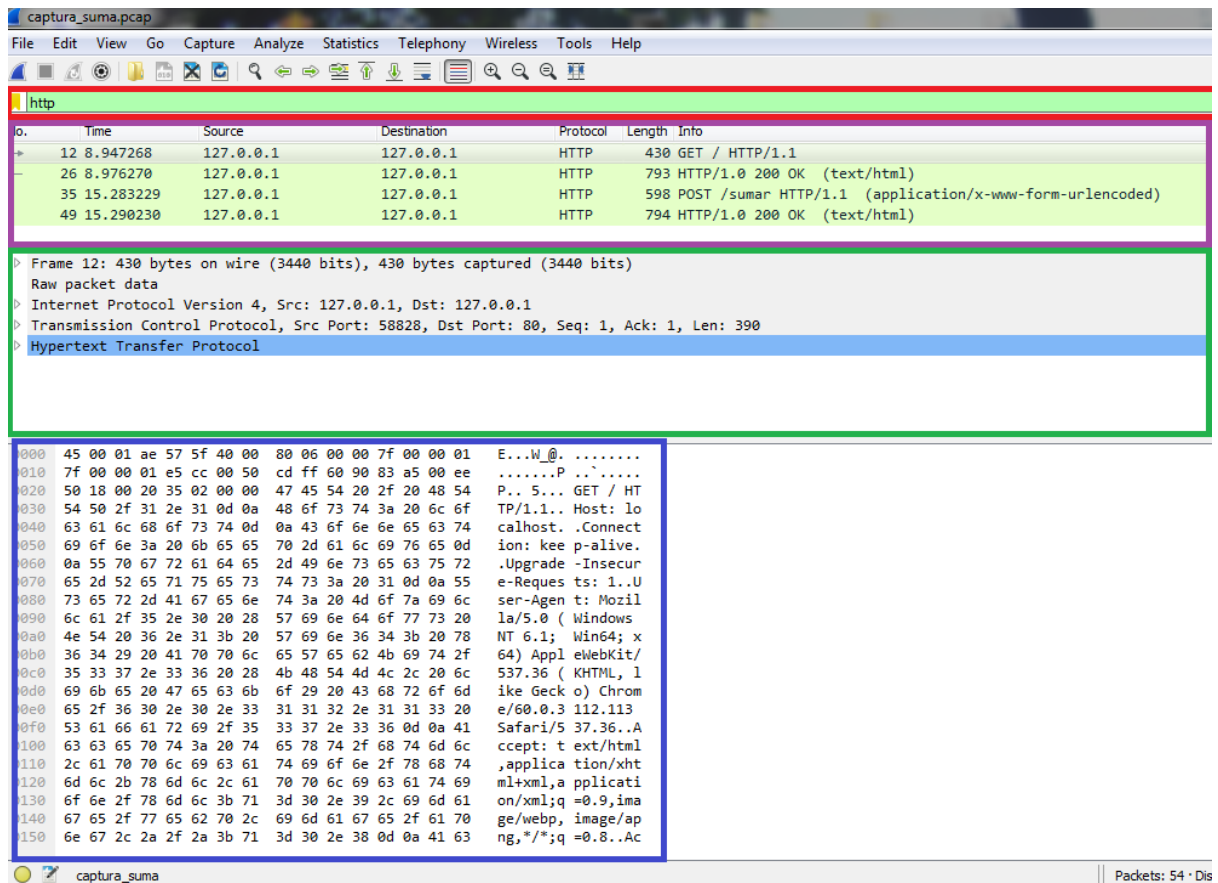


Figura 4. Paquetes capturados de la interfaz loopback.

En la Figura 4 se detallan las diferentes áreas de interés de Wireshark.

- Área roja: Representa el tipo de filtro aplicado a los paquetes, que puede ser por protocolo, IP, dirección MAC, etc. Para la resolución de la consigna se utilizó el de protocolo HTTP.
- Área morada: Representa la lista de paquetes filtrados, con sus datos correspondientes. Por ejemplo: tipo de protocolo, números de secuencia, flags, marcas de tiempo, puertos, etc.
- Área verde: Desglosa por capas cada una de las cabeceras de los paquetes seleccionados de forma detallada.
- Área azul: Representa en formato hexadecimal, el paquete tal y como fue capturado por la tarjeta de red.

Al seleccionar un paquete de respuesta del servidor, se muestra en detalle la información que llega del lado del cliente, como se aprecia en la figura 4.



```

# Frame 26: 793 bytes on wire (6344 bits), 793 bytes captured (6344 bits)
  Encapsulation type: Raw IP (7)
  Arrival Time: Sep 12, 2017 13:36:20.281047000 Hora estándar de Argentina
  [Time shift for this packet: 0.000000000 seconds]
  Epoch Time: 1505234180.281047000 seconds
  [Time delta from previous captured frame: 0.000000000 seconds]
  [Time delta from previous displayed frame: 0.029002000 seconds]
  [Time since reference or first frame: 8.976270000 seconds]
  Frame Number: 26
  Frame Length: 793 bytes (6344 bits)
  Capture Length: 793 bytes (6344 bits)
  [Frame is marked: False]
  [Frame is ignored: False]
  [Protocols in frame: raw:ip:tcp:http:data-text-lines]
  [Coloring Rule Name: HTTP]
  [Coloring Rule String: http || tcp.port == 80 || http2]
  Raw packet data
# Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ▷ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 793
  Identification: 0x576d (22381)
  ▷ Flags: 0x02 (Don't Fragment)
  Fragment offset: 0
  Time to live: 128
  Protocol: TCP (6)
  Header checksum: 0x0000 [validation disabled]
  [Header checksum status: Unverified]
  Source: 127.0.0.1
  Destination: 127.0.0.1
  [Source GeoIP: Unknown]
  [Destination GeoIP: Unknown]
# Transmission Control Protocol, Src Port: 80, Dst Port: 58828, Seq: 156, Ack: 391, Len: 753
  Source Port: 80
  Destination Port: 58828
  [Stream index: 0]
  [TCP Segment Len: 753]
  Sequence number: 156 (relative sequence number)
  [Next sequence number: 909 (relative sequence number)]

```

Figura 5. Cabeceras de los paquetes capturados

Se puede observar que el paquete tiene datos como la hora de llegada, la cantidad de bytes que ocupa el header, el TTL, el checksum, los puertos y las IP de origen y destino , el protocolo, etc.

### Inciso N°3: Simulación de una placa de sensado.

Resumen del enunciado:

Se desea poder simular el funcionamiento de un microcontrolador que permite sensar periódicamente temperatura, humedad, presión atmosférica y velocidad del viento. Dicha información será mostrada en un documento HTML (de carácter accesible y responsivo) junto al promedio y la frecuencia de muestreo elegida.

Resolución:

Se planteó el simulador de acceso a valores de temperatura, humedad, presión y viento de la siguiente manera:

Los datos son provistos por un proceso productor, el cual escribe los cuatro valores en un archivo de texto. Un proceso consumidor toma los valores del archivo de texto y los guardará en variables de la aplicación. La aplicación entrega los datos a la vista del simulador y la renderiza. La vista posee un botón para finalizar la simulación y volver al inicio.

La implementación de la aplicación fue la siguiente:

- Archivo raíz `Index.py`: Contiene toda la lógica y configuración del puerto por el cual se comunica al servidor, funciones con la lógica para el funcionamiento de la aplicación y el renderizado de las plantillas.
  - Archivo de texto "`datos.txt`": almacena los datos generados por el productor.
  - Carpeta `templates`: Guarda todas las plantillas, es decir, los archivos HTML utilizados en la aplicación. Para la resolución de la consigna se utilizará tres archivos:
    - `layout.html`: Contiene el encabezado y el pie de página de la aplicación.
    - `index.html`: Contiene una breve descripción de lo que es la aplicación y un botón para iniciar la simulación.
- A parte de flask se importaron los siguientes módulos:
- `random`: Permite la generación aleatoria de valores.
  - `time`: Permite obtener la fecha del sistema.
  - `threading`: Permite generar los threads.
  - `os`: Permite leer y escribir en archivos.
  - `simulador.html`: en él se renderizan los datos tomados del archivo de texto de datos.
- Carpeta `script`: Guarda todos los archivos para el uso de Bootstrap y Javascript.

La aplicación comienza dando la bienvenida al simulador. Cuando se presiona el botón de ir al simulador, empieza la ejecución de dos hilos.

El hilo productor ejecuta la función `producir()` y es el encargado de escribir los datos de temperatura, humedad, presión atmosférica y velocidad del viento, generados de forma aleatoria, en un archivo de texto cada 2 segundos.

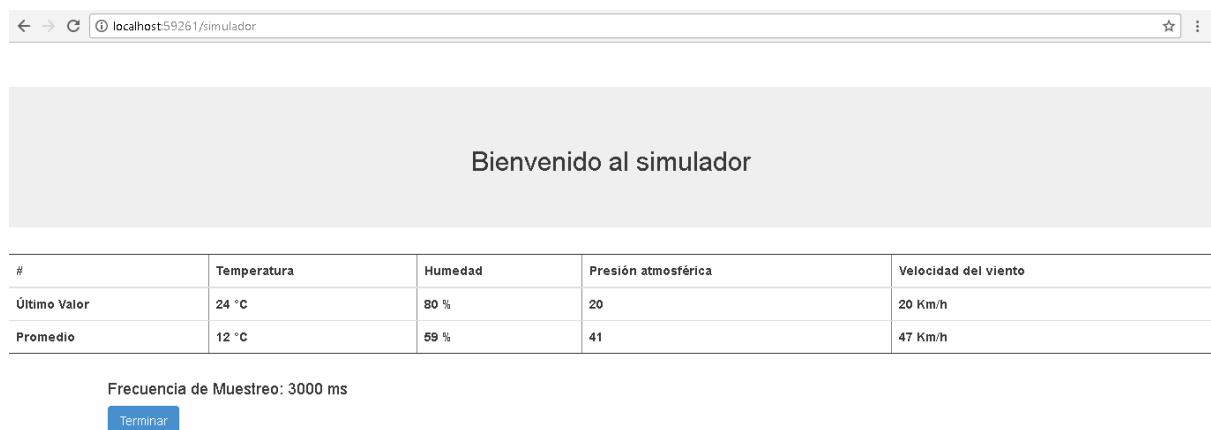
Antes de iniciar la ejecución del hilo, se setea el daemon como `true`, para que los dos threads se puedan ejecutar en el mismo proceso.

El hilo consumidor ejecuta la función `consumir()`, donde toma la información contenida en el archivo de texto y los guarda en variables, que luego son presentadas en la vista.

Finalmente, la función `simulador()` toma los datos sensados, los envía y hace el renderizado de la plantilla `simulador.html`, que posee una tabla que muestra la información obtenida y un botón para concluir la simulación y termina la ejecución de los threads (Figura 6).

Las tres funciones mencionadas se encuentran contenidas dentro del archivo `index.py`.

Cabe destacar que el hilo encargado de consumir datos del archivo de texto nunca puede ejecutarse más rápido de lo que escribe el productor de datos, puesto que si sucede eso, se mostrarían datos erróneos. Para garantizar la consistencia de los datos, en el ámbito de la simulación se estableció consumir los datos a una frecuencia mayor que la frecuencia del microcontrolador simulado.



#	Temperatura	Humedad	Presión atmosférica	Velocidad del viento
Último Valor	24 °C	80 %	20	20 Km/h
Promedio	12 °C	59 %	41	47 Km/h

Frecuencia de Muestreo: 3000 ms

Terminar

Figura 6. Vista del simulador, con los últimos valores sensados, el promedio y la frecuencia de muestreo.

#### Inciso N°4: Intervalo de muestreo modificable.

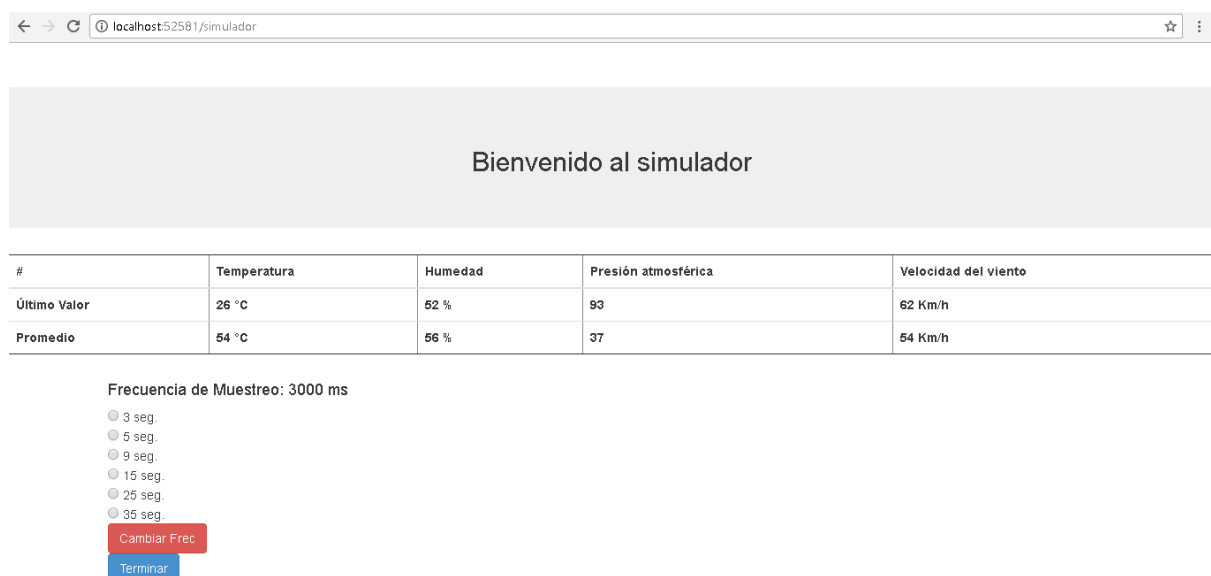
Resumen del enunciado:

Se desea agregar a la simulación de la placa, la posibilidad de que el usuario sea el que seleccione el intervalo de muestreo desde un conjunto predeterminado.

Resolución:

En la vista que contiene el simulador(Figura 7) se agregó un formulario con un menú basado en botones de radio que consta de seis diferentes velocidades de 3, 5, 9, 15, 25, 35 segundos a las cuales se van a tomar las muestras, junto con un botón para el envío del dato que representa la velocidad de toma de muestras. Dicho dato es enviado desde la vista hacia el servidor por el método POST, es decir el dato no se puede visualizar en la URL. Cuando el dato llega al servidor, se verifica que efectivamente haya llegado por POST, para luego enviarlo nuevamente a la vista y actualizar la vista del servidor con el nuevo tiempo de toma de datos.

Dado que la frecuencia en la que se producen los datos es de 2 segundos, se seleccionaron frecuencias para tomar datos que no sean múltiplos de 2 ni menores a 2, de tal manera que la lectura se realice de manera adecuada, sin que se lean datos erróneos u obsoletos (Figura 7).



#	Temperatura	Humedad	Presión atmosférica	Velocidad del viento
Último Valor	26 °C	52 %	93	62 Km/h
Promedio	54 °C	56 %	37	54 Km/h

Frecuencia de Muestreo: 3000 ms

☐ 3 seg.  
☐ 5 seg.  
☐ 9 seg.  
☐ 15 seg.  
☐ 25 seg.  
☐ 35 seg.

Figura 7. Vista del simulador, con los últimos valores sensados, el promedio, la frecuencia de muestreo y el menú para setear diferentes frecuencias de muestreo.

## **Inciso N°5: Problemática de concurrencia.**

Resumen del enunciado:

Se desea poder abarcar la problemática de la concurrencia de la simulación hecha, sobre todo al haber agregado la posibilidad de cambiar el período de muestreo, haciendo énfasis en lo que puede llegar a pasar en un ambiente real.

Resolución:

Como sucede habitualmente en la aplicaciones web cliente/servidor, se soporta la conexión de varios clientes concurrente a un mismo servidor. El servidor es el programa que ofrece los servicios a los cuales se puede acceder a través de la red, teniendo una IP permanente. Los clientes son programas que envían solicitudes al servidor y esperan una respuesta, conectándose esporádicamente al servidor. El servidor inicia la ejecución antes que comience la interacción y generalmente continúa sin fin. El cliente envía un request y espera una respuesta y generalmente finaliza después de utilizar al servidor un número finito de veces.

El servidor espera por los pedidos en un port “well-known” que fue reservado por el servicio que ofrece. El cliente reserva y utiliza un port arbitrario (efímero), no reservado para la comunicación.

Las vistas y plantillas se ejecutan dentro de cada uno de los clientes. Por lo tanto, en la aplicación hecha anteriormente, cuando un usuario ingresa un valor de frecuencia de muestreo, este se renderiza en su navegador y no existe problema de que varios clientes ejecuten el simulador al mismo tiempo.

Dicho comportamiento no sería igual en un microcontrolador físico. Normalmente, la frecuencia de muestreo de datos se configura dentro del mismo, soportando una única configuración. En otras palabras, si existen varios “clientes” queriendo cambiar la frecuencia de muestreo, dichas configuraciones se pisarían, quedando la última ingresada.

También se destaca de la simulación que al usarse como comunicación el protocolo HTTP, al ser no orientado a conexión, el servidor responde las peticiones cuando puede. Más allá de que cada petición tiene un cierto tiempo de vida para ser respondida, y que los tiempos que maneja no son los mismos de una persona, no se pueden simular aplicaciones de tiempo real, puesto que las mismas requieren un tiempo de respuesta muy preciso, cosa que el protocolo HTTP no puede garantizar.

## **Inciso 6: Diferencias respecto a una placa de sensado real.**

Resumen del enunciado:

Se desea mencionar las diferencias que hay entre la simulación planteada y el sistema real.

Resolución:

Como principal diferencia entre la simulación planteada y el sistema real es que la simulación no es susceptible a los males que sufre el hardware. No se desgasta, no se debe preocupar por los voltajes y corrientes de alimentación, de temperatura o de humedad para su correcto funcionamiento; su ciclo de vida no pasa por la cantidad de uso que se le dé y tampoco existe interferencia en los valores sensados, entre otras cosas, que sí podría sufrir el sistema de tiempo real. En otras palabras, se podría decir que en una simulación se está en un entorno “ideal”, cosa que en el sistema no es así.

Como conclusión, se puede afirmar que el simulador da una aproximación de cómo un microcontrolador realiza el muestreo de diferentes variables, en este caso, la temperatura, humedad, presión atmosférica y velocidad del viento. También simula las operaciones que se pueden realizar con las muestras y si dependen de otros factores externos a las propias mediciones, etc.

Al ejecutar diferentes experimentos en la simulación, se podría llegar a tener una noción o idea de cómo sería el sistema, las cosas que tiene que tener, como debería funcionar.

Lo que la simulación no puede lograr es igualar sus resultados a los del sistema real, debido a que, como se mencionó anteriormente, el simulador está en un entorno ideal, sin estar sometido a diferentes situaciones que no pueden ser simuladas.