



Facultad de
INFORMÁTICA



UNIVERSIDAD
NACIONAL
DE LA PLATA

Práctica 1

Taller de Proyecto II - 2017

Integrantes:

Kelly Martín

Pryszlak Mauricio

Sanjuan Diego

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.

El archivo 'requirements.txt' es una lista de argumentos de instalación de pip colocados en un archivo. Considerando que no hemos instalado Flask mediante el comando 'pip install Flask', en el archivo 'requirements.txt' debemos especificarle a pip que lo instale, para ello colocamos en el archivo 'Flask==0.12.0'.

Un pequeño ejemplo de uso puede verse de la siguiente forma:

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'
```

Veamos qué hace éste código:

1. Primero importamos la clase Flask. Una instancia de esta clase será nuestra Web Server Gateway Interface.
2. A continuación, creamos una instancia de esta clase. El primer argumento es el nombre del módulo o paquete de la aplicación. Si está utilizando un solo módulo (como en este ejemplo), debe usar __name__ porque dependiendo de si se ha iniciado como aplicación o importado como módulo, el nombre será diferente. Esto es necesario para que Flask sepa dónde buscar plantillas, archivos estáticos, etc.
3. A continuación, usamos el comentario route () para decirle a Flask qué URL debe activar nuestra función.
4. A la función se le da un nombre que también se utiliza para generar URLs para esa función en particular y devuelve el mensaje que queremos mostrar en el navegador.

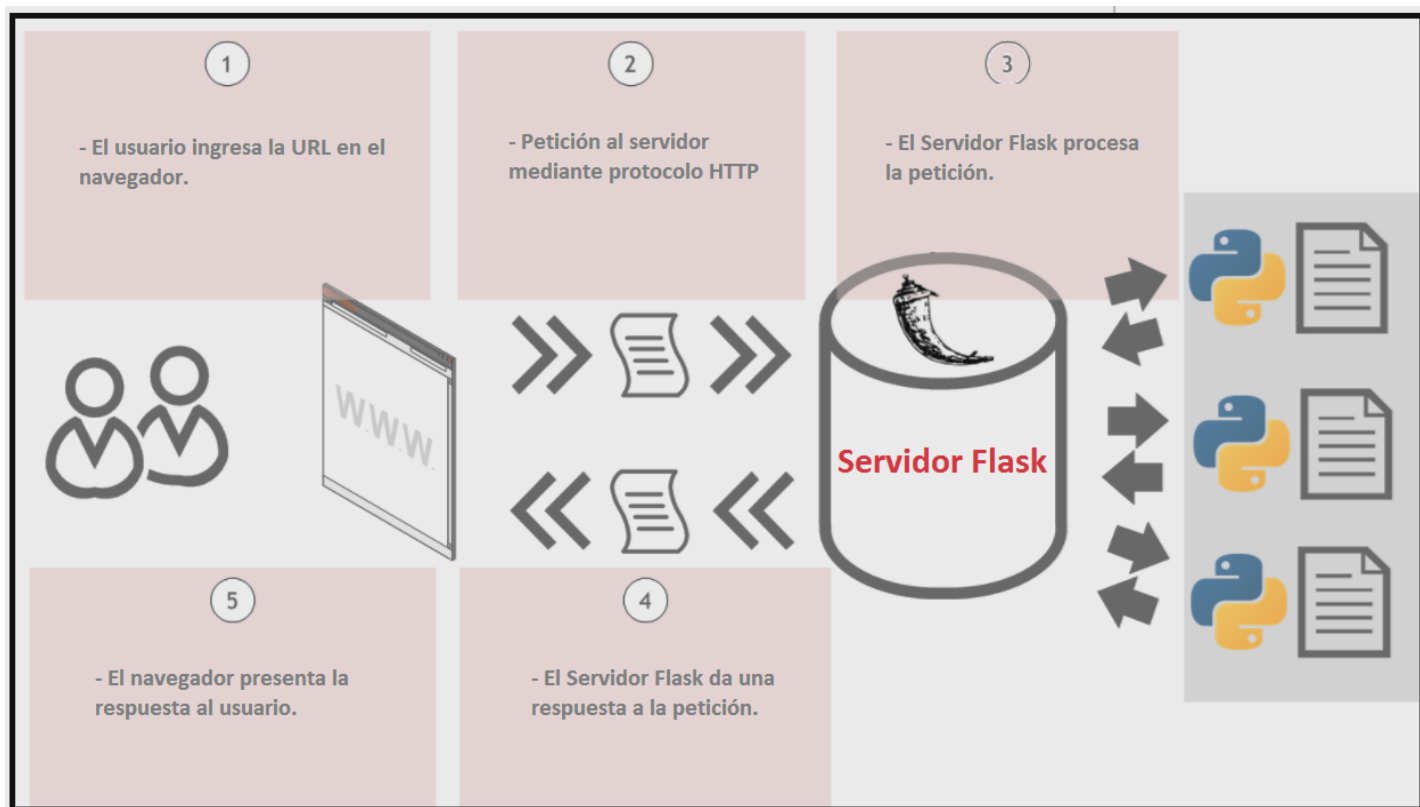


Imagen 1: Interacción del servidor Flask con el usuario.

2) Desarrollar un experimento que muestre si el servidor HTTP agrega o quita información que la genera 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).

Para éste ejercicio se adaptó uno de los ejemplos de aplicación python brindados por la cátedra. El mismo puede encontrarse en el repositorio del grupo como practica1ejercicio2.

Para la captura de los paquetes se utilizó el programa RawCap el cual puede descargarse gratuitamente de la página del fabricante. Al ejecutar el programa se nos presentará una consola como la de la **Imagen 2**.

```
Interfaces:
0. 10.3.8.89 Ethernet Ethernet
1. 169.254.138.98 Wi-Fi Wireless80211
2. 169.254.41.76 Conexión de área local* 2 Wireless80211
3. 127.0.0.1 Loopback Pseudo-Interface 1 Loopback
Select interface to sniff [default '0']: _
```

Imagen 2: interfaces disponibles para analizar.

En nuestro caso seleccionamos la interface 3, ya que nos encontramos trabajando en localhost. Una vez seleccionada la misma, nos pedirá un directorio o

nombre de archivo para el resultado del análisis, nosotros lo dejemos con el que viene por defecto.

Una vez indicado todo lo anterior el programa comienza con el análisis, a partir de ese momento ingresamos a través del navegador a nuestro proyecto de prueba, el programa nos indicará los paquetes capturados. Una vez realizadas las acciones que queremos analizar en nuestra aplicación, cerramos la consola generada por RawCap y el archivo con el resultado del análisis se generará automáticamente.

Para el análisis de los paquetes capturados optamos por utilizar la herramienta Wireshark. Una vez instalado el mismo, ejecutamos el programa y nos dirigimos a File -> Open -> Seleccionamos el archivo generado en los pasos anteriores por el programa RawCap.

A partir de lo explicado anteriormente podemos ver los paquetes enviados por nuestra aplicación y las respuestas por parte del servidor.

El navegador pide en el servidor web correspondiente los datos necesarios para mostrar la página. Esta consulta tiene lugar mediante HTTP en la forma de un paquete de datos que contiene toda la información que el servidor web necesita para entregar los datos de la página web (**Imagen 2**). El servidor web evalúa esta información y emite un código de estado HTTP como puede verse en la **Imagen 3**. Si la solicitud tiene éxito, el servidor envía un paquete de datos al navegador con toda la información necesaria para la visualización de la página web. Si, por el contrario, el servidor no encuentra la página web en la dirección solicitada, emite un código de error 404.

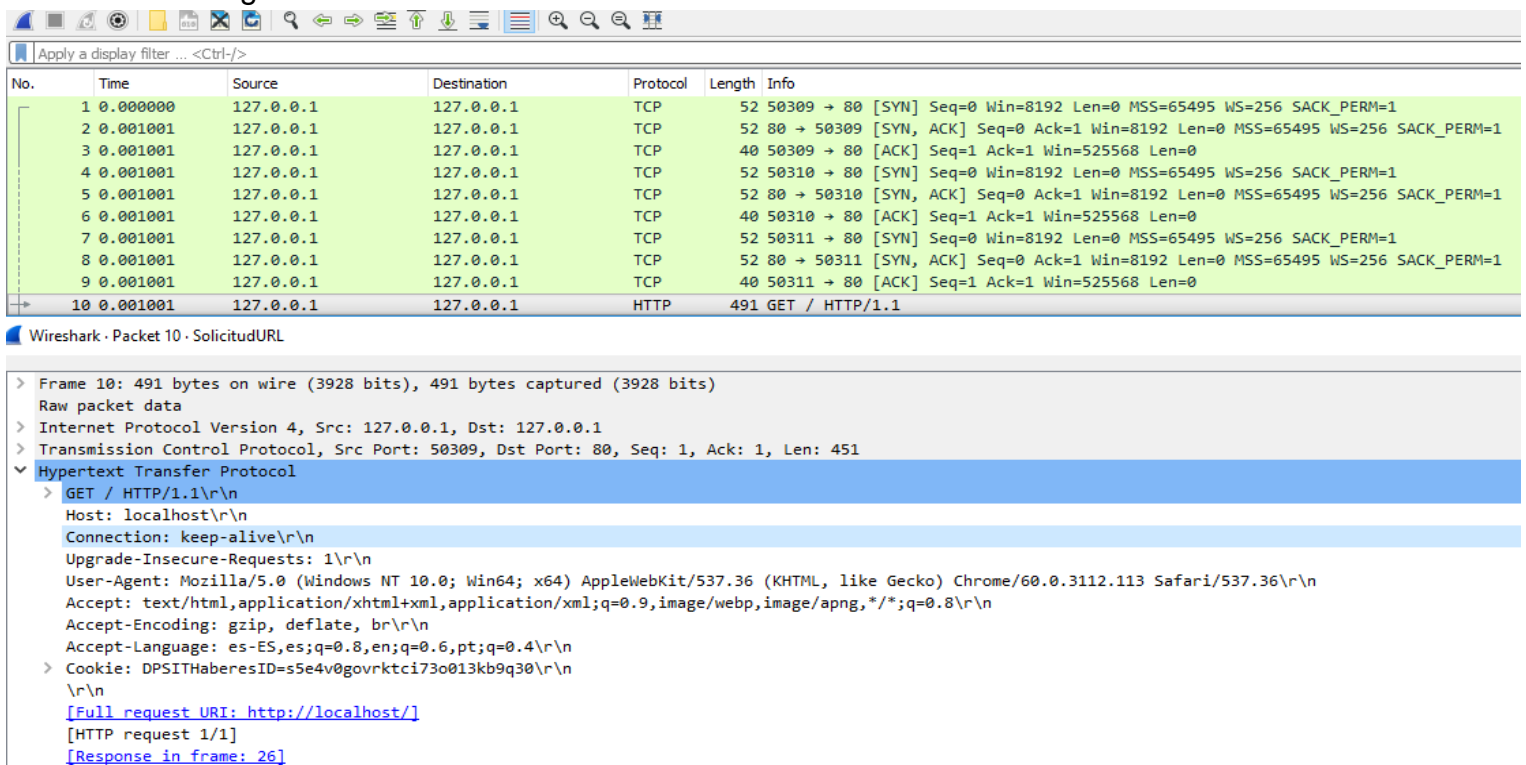


Imagen 3: Solicitud de la página al servidor mediante GET.

No.	Time	Source	Destination	Protocol	Length	Info
19	0.012017	127.0.0.1	127.0.0.1	TCP	40	50309 → 80 [ACK] Seq=452 Ack=118 Win=525312 Len=0
20	0.012017	127.0.0.1	127.0.0.1	TCP	77	80 → 50309 [PSH, ACK] Seq=118 Ack=452 Win=525568 Len=37 [TCP segment of a reassembled PDU]
21	0.012017	127.0.0.1	127.0.0.1	TCP	40	50309 → 80 [ACK] Seq=452 Ack=155 Win=525312 Len=0
22	0.012017	127.0.0.1	127.0.0.1	TCP	42	80 → 50309 [PSH, ACK] Seq=155 Ack=452 Win=525568 Len=2 [TCP segment of a reassembled PDU]
23	0.012017	127.0.0.1	127.0.0.1	TCP	40	50309 → 80 [ACK] Seq=452 Ack=157 Win=525312 Len=0
24	0.012017	127.0.0.1	127.0.0.1	TCP	1500	80 → 50309 [ACK] Seq=157 Ack=452 Win=525568 Len=1460 [TCP segment of a reassembled PDU]
25	0.012017	127.0.0.1	127.0.0.1	TCP	1500	80 → 50309 [ACK] Seq=1617 Ack=452 Win=525568 Len=1460 [TCP segment of a reassembled PDU]
26	0.012017	127.0.0.1	127.0.0.1	HTTP	734	HTTP/1.0 200 OK (text/html)

Wireshark · Packet 26 · SolicitudURL

```
> Frame 26: 734 bytes on wire (5872 bits), 734 bytes captured (5872 bits)
Raw packet data
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 80, Dst Port: 50309, Seq: 3077, Ack: 452, Len: 694
> [9 Reassembled TCP Segments (3770 bytes): #12(17), #14(40), #16(22), #18(38), #20(37), #22(2), #24(1460), #25(1460), #26(694)]
▼ Hypertext Transfer Protocol
  > HTTP/1.0 200 OK\r\n
    Content-Type: text/html; charset=utf-8\r\n
    Content-Length: 3614\r\n
    Server: Werkzeug/0.12.2 Python/2.7.6\r\n
    Date: Thu, 07 Sep 2017 18:33:08 GMT\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.011016000 seconds]
    [Request in frame: 10]
    File Data: 3614 bytes
▼ Line-based text data: text/html
  <html>\n
    <head>\n
      \n
      <title>TP1EJ2</title>\n
      <meta name="viewport" content="width=device-width, initial-scale=1">\n
      <!-- JQuery -->\n
      <script src="/static/jquery/jquery-3.2.1.min.js"></script>\n
      <!-- Bootstrap CSS -->\n
      <link rel="stylesheet" href="/static/bootstrap/css/bootstrap.min.css" />\n
      <!-- Bootstrap JS -->\n
      <script src="/static/bootstrap/js/popper.min.js"></script>\n
      <script src="/static/bootstrap/js/bootstrap.min.js"></script>\n
    \n
  </head>\n
  <body>\n
```

Imagen 4: respuesta HTTP del servidor.

La aplicación de prueba contiene un formulario, el mismo se envía al servidor a través de una petición POST, como puede observarse en la Imagen 5.

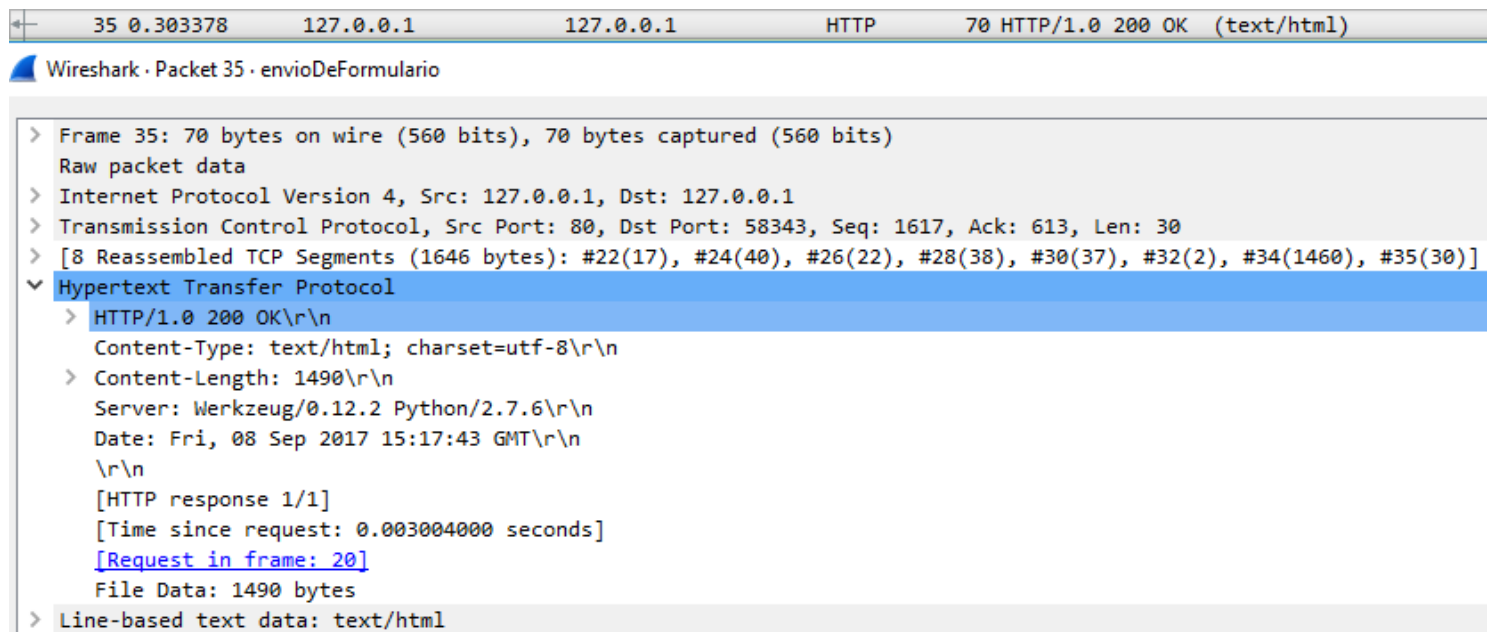
20	0.300374	127.0.0.1	127.0.0.1	HTTP	652	POST /form HTTP/1.1 (application/x-www-form-urlencoded)
----	----------	-----------	-----------	------	-----	---

Wireshark · Packet 20 · envioDeFormulario

```
> Frame 20: 652 bytes on wire (5216 bits), 652 bytes captured (5216 bits)
Raw packet data
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 58343, Dst Port: 80, Seq: 1, Ack: 1, Len: 612
> Hypertext Transfer Protocol
▼ HTML Form URL Encoded: application/x-www-form-urlencoded
  > Form item: "nombre" = "Martin"
  > Form item: "apellido" = "Kelly"
  > Form item: "email" = "m@gmail.com"
  > Form item: "sexo" = "masc"
```

Imagen 5: envío del formulario al servidor mediante POST.

Si el contenido enviado es correcto, el servidor nos responde con un **HTTP 200 OK** como puede observarse en la Imagen 6, y con el contenido HTML que corresponda.



```
35 0.303378 127.0.0.1 127.0.0.1 HTTP 70 HTTP/1.0 200 OK (text/html)
Wireshark · Packet 35 · envíoDeFormulario

> Frame 35: 70 bytes on wire (560 bits), 70 bytes captured (560 bits)
Raw packet data
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 80, Dst Port: 58343, Seq: 1617, Ack: 613, Len: 30
> [8 Reassembled TCP Segments (1646 bytes): #22(17), #24(40), #26(22), #28(38), #30(37), #32(2), #34(1460), #35(30)]
Hypertext Transfer Protocol
> HTTP/1.0 200 OK\r\n
Content-Type: text/html; charset=utf-8\r\n
> Content-Length: 1490\r\n
Server: Werkzeug/0.12.2 Python/2.7.6\r\n
Date: Fri, 08 Sep 2017 15:17:43 GMT\r\n
\r\n
[HTTP response 1/1]
[Time since request: 0.003004000 seconds]
[Request in frame: 20]
File Data: 1490 bytes
> Line-based text data: text/html
```

Imagen 6: respuesta del servidor al envío del formulario.

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.

La adquisición de los datos se realizó mediante la función `random.uniform` la cual arroja valores float, estos mismos con la función `round()` son truncados a dos decimales de representación, en la placa real estos datos serían adquiridos de los ADC de la placa.

```

valor_temp = random.uniform(0, 100)
valor_vient = random.uniform(0, 80)
valor_hum = random.uniform(0, 20)
valor_pres = random.uniform(0, 50)
valor_temp = round(valor_temp, 2)
valor_vient = round(valor_vient, 2)
valor_hum = round(valor_hum, 2)
valor_pres = round(valor_pres, 2)

```

Imagen 7: simulación de adquisición de valores.

En este caso se supuso una placa con 4 ADC para poder leer los valores, cada sensor estará conectado a un canal del periférico ADC, los cuales recibirán diferentes niveles de tensiones analógicas que corresponderán a diferentes mediciones. Luego, serán convertidos a niveles de tensiones digitales para poder ser procesados. Otro punto que se tendría en cuenta sería el de un posible circuito de adaptación para los sensores, por ejemplo, qué funciones a una tensión diferente a la entregada por la placa. El proyecto simularía el esquema que puede verse en la *Imagen 7*.

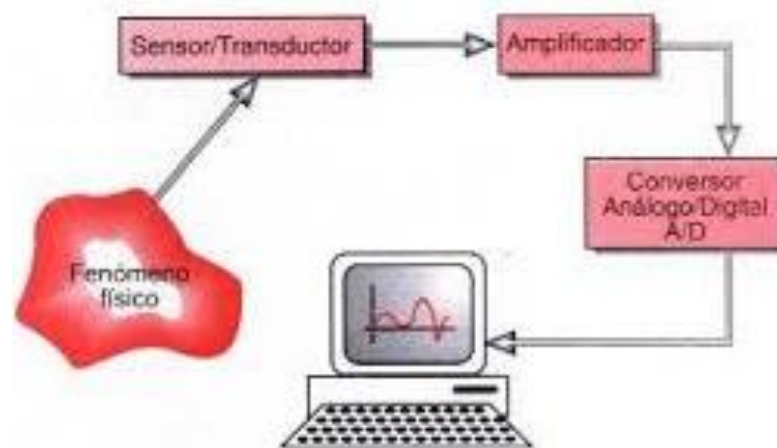


Imagen 8: representación del sistema a simular.

Se decidió por realizar dos procesos diferentes, los cuales se comunican mediante una base de datos, que gestionamos con phpMyAdmin a través de XAMPP, cabe aclarar que en el puerto de los procesos y el puerto de XAMPP para el módulo MySQL deben ser los mismos. En nuestro caso utilizamos el puerto 3306.

XAMPP Control Panel v3.2.2					
Modules					
Service	Module	PID(s)	Port(s)	Actions	
<input type="checkbox"/>	Apache	9140 11772	80, 443	<input type="button" value="Stop"/>	Admin Config Logs
<input type="checkbox"/>	MySQL	11272	3306	<input type="button" value="Stop"/>	Admin Config Logs

Imagen 9: puerto configurado para el módulo MySQL

Una vez configurado el puerto, se procede a crear la base de datos, para lo cual se utilizó phpMyAdmin como se mencionó anteriormente. La estructura de la base de datos puede observarse en la *Imagen 10*.

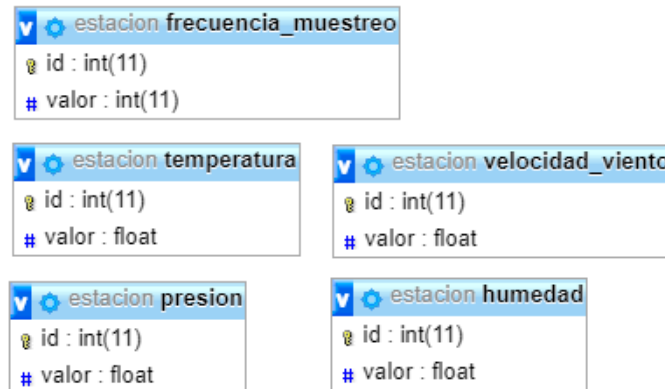


Table Name	Column Name	Column Type
estacion frecuencia_muestreo	id	int(11)
	valor	int(11)
estacion temperatura	id	int(11)
	valor	float
estacion velocidad_viento	id	int(11)
	valor	float
estacion presion	id	int(11)
	valor	float
estacion humedad	id	int(11)
	valor	float

Imagen 10: tablas de la base de datos utilizada.

La ejecución de los procesos debe realizarse por separado. En primera instancia el módulo productor genera los valores y almacena en la base de datos, realizado esto se pone a dormir durante 60 segundos para simular la frecuencia de muestreo.

Por otro lado, el módulo consumidor refresca sus datos a la frecuencia dada tomando el último valor de cada una de las magnitudes, los últimos diez valores con los que genera el promedio, la frecuencia de muestreo y presenta todos estos datos para que el usuario pueda visualizarlos. La actualización de los valores lo hace a través de la actualización de la propia página en el navegador para lo cual se utilizan los metas tags que se muestran en la *Imagen 11*.

```
<meta http-equiv="Pragma" content="no-cache">
<meta http-equiv="refresh" content="{ { frecuencia } }>
```

Imagen 11: tags utilizados para actualizar la página.

Con respecto a la concurrencia, la base de datos se encarga de manejarla, aunque en éste caso no va a tener mayores dificultades ya que uno de los procesos escribe, mientras que el otro se encarga de leer dichos valores.

SIMULACIÓN Estación meteorológica

Frecuencia de muestreo

60

Modificar

Última muestra obtenida

Temperatura	Humedad	Presión atmosférica	Velocidad del viento
87.61	7.88	19.21	53.78

Promedio Últimas 10 muestras

Temperatura	Humedad	Presión atmosférica	Velocidad del viento
68.836	12.304	34.656	49.462

Imagen 12: datos presentados al usuario.

Con respecto al promedio de los últimos 10 valores obtenidos se contempló el caso en que se tengan menos de 10 muestras, se resolvió realizando el promedio de las muestras obtenidas hasta el momento y en caso de no poseer muestras esto se indica con un “cero”.

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.

Para agregar la nueva funcionalidad solicitada, se optó por utilizar un formulario seleccionable a partir del cual el usuario puede modificar la frecuencia de muestreo.

```
<div class="form-group">
  <label>Frecuencia de muestreo</label>
  <select class="form-control select2" id="frec" name="frec" style="width: 100%;">
    <option selected="selected">{{ frecuencia }}</option>
    <option>1</option>
    <option>2</option>
    <option>5</option>
    <option>10</option>
    <option>30</option>
    <option>60</option>
  </select>
</div>
<button type="submit" class="btn btn-primary" style="float: right;">Modificar</button>
</form>
```

Imagen 13: formulario HTML para la selección de la frecuencia.

Si el usuario decide modificar la frecuencia, la misma se actualiza con el valor seleccionado en la base de datos a partir de donde el proceso productor la toma y cambia su frecuencia de muestreo, así como también el proceso consumidor toma el nuevo valor y de esta forma modifica la frecuencia de actualización de los datos presentados al usuario.

A nivel HTTP vamos a tener una petición del tipo POST para el envío de los datos del formulario al servidor, el cual los procesa y de ser válidos los datos enviados, actualiza la frecuencia en la base de datos.

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.

La problemática de la concurrencia de la simulación se da entre los dos procesos existentes. Uno de ellos, productor, simula los valores correspondientes a diversos sensores con cierto período de muestreo y los almacena en un medio (base de datos); por otro lado, el consumidor lee el medio compartido en simultáneo y presenta en la web la información solicitada. El problema radica en que el productor no escriba más datos que los que el consumidor lee o que, éste último, no lea valores antes de que sean escritos dando lugar a pérdidas de datos. Específicamente, la posibilidad de cambiar el período de muestreo podría ocasionar tanto una sobrecarga (interrupción de E/S) que impida que los procesos actúen en el tiempo adecuado como así también, provocar un aumento en la coordinación dado que los plazos determinísticos a cumplir se reducen.

En el ambiente real, atendiendo al nivel de exigencia temporal, es decir, dependiendo el tipo de sistema de tiempo real en el que se utilicen las mediciones ya mencionadas, las tolerancias a fallos serán de mayor o menor importancia. En el caso que sea un sistema crítico, el tiempo de respuesta debe garantizarse en absoluto, caso contrario, pueden generarse consecuencias fatales. Otro caso es el de los sistemas “soft real time” en donde la pérdida de un dato no produce perjuicios catastróficos, pero sí un deterioro del funcionamiento global.

Posibles problemas en sistemas de tiempo real:

- Sincronización y coordinación de tareas: se debe proveer de un mecanismo por el cual los procesos se transmitan datos de unos a otros con cierto sincronismo (semáforos, por ejemplo).

- Procesamiento de interrupciones y manejo de E/S sin pérdida de datos: evitar interbloqueos, manejo de interrupciones anidadas y correctitud temporal tras el cambio de contexto y latencia que conlleva la atención.
- Tiempo de respuesta crítico: cumplimiento de los plazos temporales.
- Mantenimiento: ante cualquier cambio se debe verificar tanto la correctitud temporal como funcional, debido a que, un pequeño cambio puede afectar al resto del sistema.

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.*

Entre la simulación planteada y el sistema real se suponen las siguientes diferencias:

- Los valores random provistos por el sistema sensores-microcontrolador en la simulación se suponen exactos y sin errores, lo que nos puede llevar a conclusiones falsas. Por lo tanto, en el sistema real para acercarse lo más posible a los mismos valores se deberá contar con etapas de acondicionamiento de la señal que arrojen medidas correctas y limpias de ruido. Todo ello, puede implicar cálculos adicionales; como, por ejemplo, la conversión del valor de salida de un ADC a un valor comprensible para el humano.
- Es posible que la simulación, no haya tenido en cuenta ciertos valores que se encuentran fuera de los límites para el cual fue construido el sistema real.