Taller de Proyecto II

Práctica 1

Facultad de Informática
Universidad Nacional de La Plata

Fecha de entrega: 30 de agosto de 2018

Duranti, Máximo (993/9)
Poch, Gonzalo Julián (1050/1)

Punto 1

Adquisición de datos

La adquisición de los datos se encuentra principalmente definida en el archivo *database.py* . Se basa en la definición de un único método: *'get_samples'*, que resulta suficientes para resolver los tres items que pide el enunciado, previa la configuración correcta de la base de datos y la conexión de la misma con python y flask.

El método *'get_samples'* recibe un parámetro que determina la cantidad de valores que se piden de la base de datos. Devolviendo así las últimas N muestras que existen en la base de datos.

Su ejecución corresponde a la siguiente serie de pasos:

1- Obtener la sesión, en el caso de que exista, o crear una nueva para conectarse con la base de datos.

2- Ejecutar la consulta para obtener los datos desde la base de datos.

3- Cerrar la sesión, y así la conexión con la base de datos.

4- Devolver la serialización de los valores resultantes de la consulta

De este modo para resolver el primer ítem, al ingresar en la ruta *'/promedios',* se ejecuta el método con valor de parámetro '10'. Obteniendo así las diez últimas muestras para que se procesen, calculando los promedios, antes de renderizar el html correspondiente, al cual se le envían los datos necesarios para poder ser mostrados.

En el caso de los otros dos ítems, se ejecuta de la misma manera, ejecutando el método *'get_samples'* pero esta vez enviando '1' como parámetro. De esta forma, se obtiene la última muestra y se envía como parámetro para ser renderizada en el html.

Particularmente para la resolución del tercer ítem, se crea una ruta *'/last-sample'* que utiliza el mismo método *'get_samples'* con parámetro igual a '1'. Dicha ruta ejecuta un método get que devuelve en formato json el resultado del método, y es utilizada dentro de *'vivo.js'* para actualizar los datos en vivo en el html.

Dicho javascript se encarga de, una vez definido un intervalo, ejecutar periódicamente una función que realiza la consulta get al sistema. Recibiendo así los datos actualizados para reemplazar los datos antiguos de la página.

Ejecución de procesos

En cuanto a la simulación de la placa o de la estación meteorológica se utiliza un proceso llamado *process.py*, que se encarga de calcular valores aleatorios para cada medición.

Dichos valores se crean cada un segundo en un loop infinito que se ejecuta durante la ejecución del proceso.

Para el manejo del proceso de simulación existe otro, llamado *aux_pro.py*, que define métodos para comenzar, parar y conocer el estado del proceso. Estos procesos son utilizados desde una ruta, llamada *'/toggle-process'*, para realizar lo siguiente:

Si el proceso está corriendo, lo detiene y si no está corriendo lo enciende.

Además dentro del proceso de simulación se declara el llamado *'Graceful Killer'* que logra que el proceso se detenga de la manera correcta cuando recibe la señal de hacerlo.

Esquema general

El sistema consta en una base de datos, que contiene una única tabla llamada Samples, que en sus columnas contiene la información de cada muestra (id, temperatura, humedad, presión y velocidad del viento).

Dicha base de datos puede ser manejada por medio de phpmyadmin, que además permite ver la información contenida en la misma.

El webserver, desarrollado en python y flask provee las rutas de acceso a la página web, desde donde se accede a los datos dispuestos en la base de datos.

Interacción del usuario

Al ingresar a la ruta correspondiente a la página principal "/", se muestra en la parte superior izquierda un panel compuesto por tres botones funcionales, los cuales permiten al usuario dirigirse a distintas rutas de la aplicación.

- Promedios: al presionar el botón de 'Promedios' provisto en la página principal, se accede a una lista que detalla las últimas diez muestras al momento de ingresar (en el caso de que aún no se hayan generado diez, se muestran las existentes) y en la última fila se muestra el resultado de calcular el promedio de cada uno de los parámetros solicitados (temperatura, humedad, presión y viento).
- Última muestra: al presionar el botón de 'Última muestra', se puede visualizar de forma asincrónica la última muestra almacenada al momento de presionarlo.
- En vivo: al presionar el botón de 'En vivo', la aplicación se dirige a la ruta de '/vivo' en donde se puede observar de forma sincrónica la última muestra obtenida. En este caso, los datos se actualizan según el periodo de muestreo seleccionado por el usuario, el cual se inicializa por defecto en 1 segundo.

Por otra parte, en la página principal de la aplicación, existe un botón cuyo funcionamiento es de tipo *toggle,* que permite accionar y detener el proceso que genera muestras aleatorias en *background.*

Punto 2

Con respecto a la concurrencia, podemos observar que hay varios ‘clientes’ de la base de datos. Podemos identificar el servidor web y el proceso que corre en background generando las muestras. Además, aumentando la frecuencia de muestreo, el volumen de solicitudes que tiene que manejar la base de datos crece.

De este modo, la base de datos debe responder a varias solicitudes que llegan de manera simultánea, siendo necesario que logren ser resueltas de manera satisfactoria y se reciban los datos pedidos por parte del servidor web y persistan los datos enviados por el proceso.

En el ambiente real podría producirse la misma situación, potenciada quizás al incrementar la cantidad de placas que agregan muestras a la base de datos junto con varios usuarios accediendo a los datos en simultáneo.

Los problemas que logramos identificar, creemos que resultan simulables en el ambiente utilizado.

Punto 3

A partir de la simulación planteada, se detectan diferencias respecto del funcionamiento del sistema real. A la hora de definir si la solución propuesta es correcta o no, se debe tener en cuenta su potencial aplicación en el sistema completo.

En primer lugar, se puede apreciar que las muestras obtenidas pueden ser fácilmente diferenciadas entre sí ya que los valores son generados de manera aleatoria. En un caso real, es posible que, por ejemplo, el valor de la temperatura se mantenga estable durante un período extenso de tiempo. De este modo, a la vista del usuario del sistema, sería dificultoso detectar si se trata de un problema en los sensores o si efectivamente la temperatura medida se encuentra estabilizada.

Por otra parte, teniendo en cuenta un sistema real, podría ser que el período de muestreo seleccionado sea menor que el tiempo que se demora el sistema en obtener nuevas mediciones y transmitirlas al servidor. Es así que el programador debería contemplar estas situaciones limitando los posibles valores seleccionables para el período de muestreo o, en su defecto, implementando una comunicación de parte de la aplicación hacia al usuario sobre lo sucedido.

Por último, dentro de los parámetros reales, sería posible encontrarse en una situación en la que se pierde la conexión entre la placa y el servidor. De la misma forma que se expresó anteriormente, se destaca la importancia de que exista una comunicación con el usuario del sistema para que éste pueda visualizar el estado actual de la conexión mencionada.