Taller de Proyecto II – año 2018

Práctica N° 1

Facultad de Informática
Universidad Nacional de La Plata

.

Krasowski & Madou

CONSIGNA GENERAL

Se propone generar un proyecto de simulación de acceso a valores de temperatura, humedad, presión atmosférica y velocidad del viento. Para su resolución, se involucran dos procesos: uno (desde ahora, *process.py*) simulando una placa con microcontrolador y sensores, generando y almacenando muestras periódicamente, el otro (*servidor*, en *app.py*) sirviendo páginas web que exponen procesados algunos de los datos almacenados por el primero, y permiten cierta interacción con el usuario.

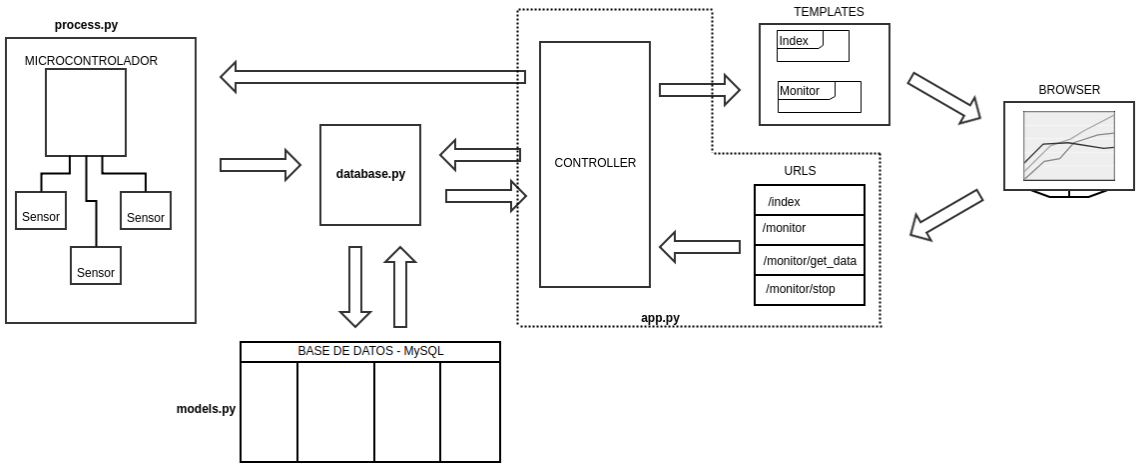
TECNOLOGÍAS

Para la implementación del proyecto se utiliza un contenedor Docker, provisto por la cátedra, que genera un ambiente donde se instalan todas las dependencias del proyecto general, permitiendo la ejecución automática de los procesos sin necesidad de instalarlas localmente. La base de datos corre bajo *MySQL 5.7*, la simulación de la placa y el *back-end* del servidor en *Python 3* (es un servidor *flask*, que se relaciona con la base de datos mediante la biblioteca *sqlalchemy*), mientras que la interacción con el usuario y el comportamiento de la página es una integración de *HTML*, *CSS* (con *Bootstrap*) y *Javascript* (utilizando *JQuery*).

ESQUEMA GENERAL

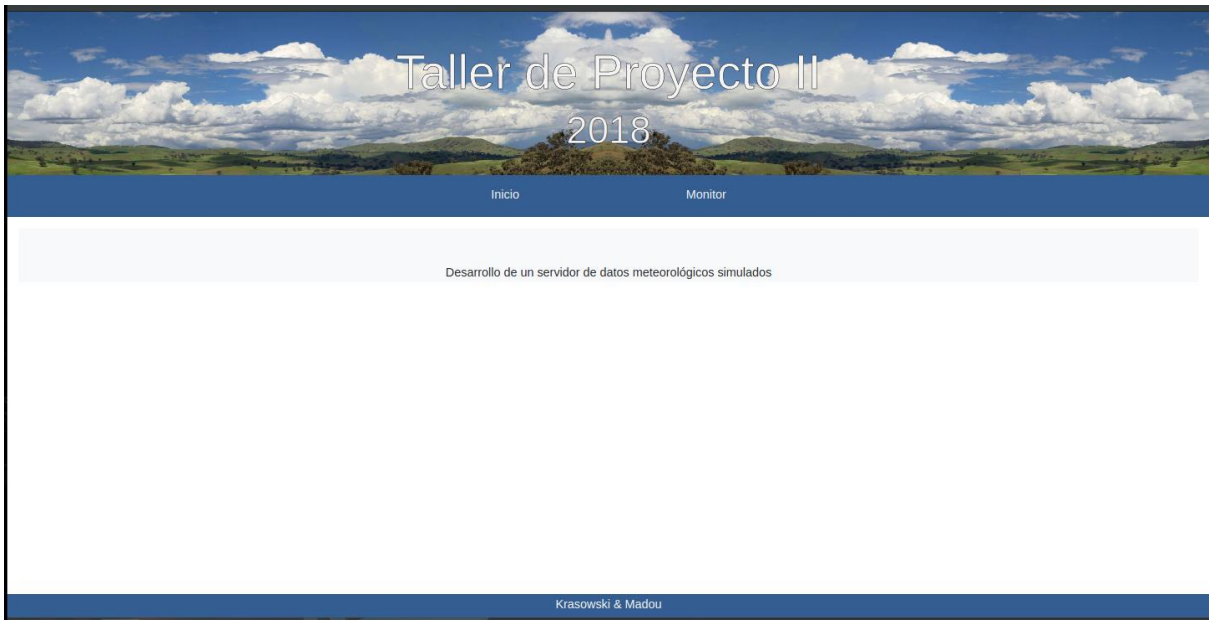
El sistema es un ejemplo claro del paradigma “productor-consumidor”, donde el recurso en común es la base de datos, el productor es el simulador de la placa adquisidora con sus sensores, y el consumidor es la aplicación web que recupera la información de la base de datos para hacer uso de ella (procesarla y mostrársela al usuario). En extensión, la aplicación web sólo actúa como consumidora cuando el usuario hace pedidos que requieren de algún dato almacenado; para ello, cada pedido es recibido en una “ruta” que refiere a un controlador particular, encargado de hacer las peticiones a la base de datos y devolver los resultados.

En resumen:

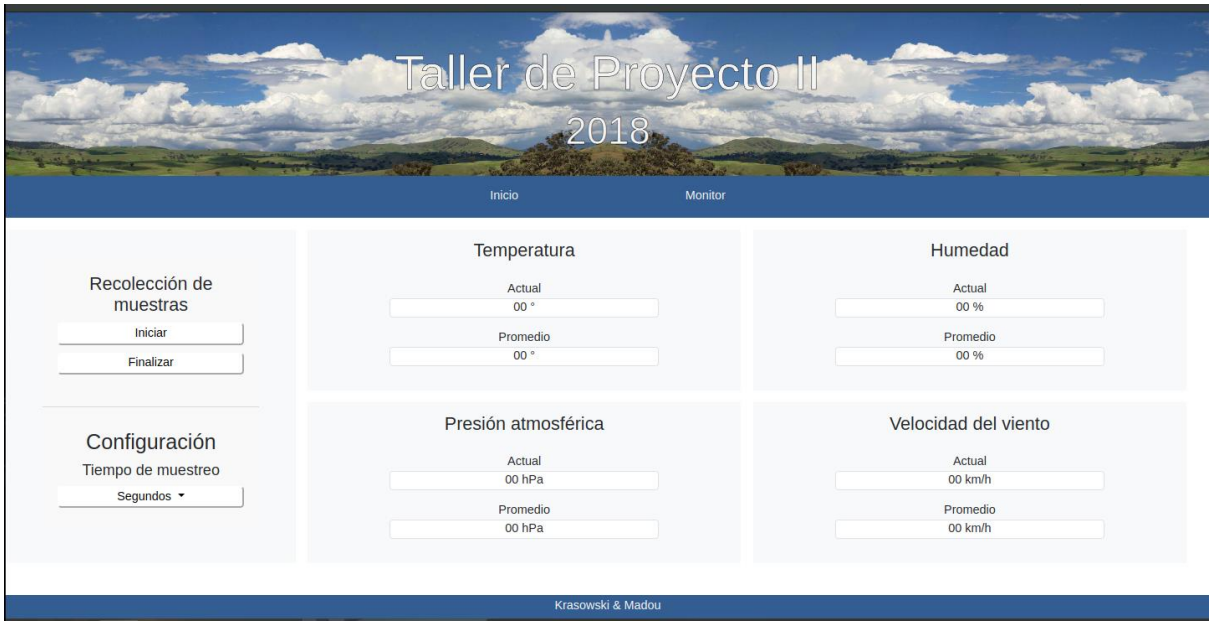


INTERACCIÓN CON EL USUARIO

El sistema es amigable e intuitivo de usar. Al acceder al sitio, uno se encuentra con una página de bienvenida y dos botones para acceder a sus dos secciones (la misma página de bienvenida y el *monitor*, la simulación propiamente dicha):



Una vez en la segunda pantalla, ya se puede ver cómo se muestran los requisitos de la consigna: un recuadro para cada una de las variables atmosféricas, con su correspondiente entrada para el último valor sensado (“Actual”) y el promedio de las últimas diez muestras (“Promedio”); botones para comenzar a actualizar los valores de dichas variables y dejar de hacerlo, y; una lista desplegable para, cuando se están actualizando las magnitudes, poder elegir cada cuánto tiempo hacerlo (1, 2, 5, 10, 30 o 60 segundos).



Profundizando en las tecnologías que provocan que esta interacción con el usuario sea exitosa, hay tres grandes categorías para analizar: HTML, CSS y Javascript (JS).

HTML es el encargado del maquetado general de las páginas. Es el que define lo que ve el usuario. Entre sus etiquetas <head> se definen reglas de estilo, metadatos, el título de la página. En el <body> se presenta el cuerpo principal de lo que se muestra (se incluyen otras etiquetas variadas, de párrafos, enlaces, navegación, etc), y se incluyen los archivos JS que se desean ejecutar entre etiquetas <script>. Cada elemento (definido entre etiquetas) puede tener atributos; aquí se puntualiza en las *clases* ("*class=.<class_name>"*, que agrupa elementos con características generales en común) y los *ids* ("*id=<id_tag>*", que diferencia elementos que deben adquirir una característica o comportamiento singular). Las etiquetas, las clases y los ids permiten relacionar el HTML con los archivos de estilo (CSS) y de comportamiento (JS)

CSS permite definir cómo se ven las páginas del sitio. Mediante referencias a los elementos (*p, a, div*), a las clases (*.<class_name>*) y a los ids (*#<id_tag>*) del HTML (solos o combinados), admite la modificación de su diseño y posicionamiento en la pantalla. Se utiliza Bootstrap para acondicionar los elementos de las páginas principales, aprovechando la flexibilidad de sus contenedores y lo terminado de sus diseños para los distintos elementos. Se logra, mediante su uso, una adaptabilidad (*responsiveness*) al sitio, de modo de que los componentes no se distorsionen ante un redimensionado de la pantalla. En el proyecto en cuestión, además, se adiciona un archivo propio de diseño, para poder adecuar las vistas a lo deseado.

Javascript es el principal responsable de que los elementos HTML adquieran un comportamiento dinámico. Así, cuando el usuario decide comenzar a actualizar las variables atmosféricas en pantalla, presiona el botón *Iniciar*; el mismo, en el HTML tiene como atributo el *id="start-sampling"*, el cual lo relaciona directamente con el método en *"/static/js/monitor.js"* que invoca a la JQuery *"$.(#start-sampling).click(...);* y ejecuta lo que se indica en ella. Lo mismo para interrumpir la actualización (botón *Finalizar*, atributo *id="stop-sampling"*, JQuery *"$.(#stop-sampling).click(...);*) o actualizar sus tiempos (misma idea). Para lograr todo este funcionamiento, al final del HTML (en el archivo *"/www/templates/base.html"*), se incluyen líneas para los scripts de JQuery y el *monitor.js* local.

BASE DE DATOS

Se utiliza una base de datos MySQL, generada en conjunto con el contenedor Docker, y administrable a través de un PHPMyAdmin.

Al momento de configurar el Docker, se tienen en cuenta algunos aspectos que invloucran la base de datos:

- se establece un binding del puerto en el docker para MySQL al puerto típico (esto es, 3309:3306). Lo mismo para PHPMyAdmin (8080:80) y para el servidor web (el *www*, 8888:8888)
- se indica en *database.py* (contiene toda la programación para la interacción con la base de datos) que la MySQL será accedido por el puerto 3306.
- se configura el archivo *.env* con las variables de entorno necesarias (MYSQL_USER, MYSQL_PASSWORD y MYSQL_DATABASE)

- se corrigen detalles del dump provisto */database/samples.sql*, corrigiendo el nombre de creación de la base de datos (de 'samples' a 'tp2') para que coincida con las configuraciones del proyecto.

En cuanto a la manipulación de la misma, se utilizan funciones ejecutables sobre un objeto *Database*, definido en el archivo *database.py*, que establece una sesión y permite añadir una nueva muestra (objeto *Samples*) o consultar las últimas diez de la base de datos. Para ello, se vale de los recursos de la biblioteca *sqlalchemy*.

PLACA SIMULADA

La parte "productora" del proyecto es la simulación de la placa con sus sensores. La misma está implementada en el archivo */www/process.py*, cuya función *main* es convocada cuando el usuario accede por primera vez a la simulación. Al hacerlo, el gestor de la ruta involucrada llama a la función *start_process* del archivo */www/aux_pro.py*, que se encarga de realizar las verificaciones pertinentes y convocar efectivamente al *main* del productor.

La tarea de este proceso es resumible en el siguiente pseudocódigo:

```
setear variables climáticas por defecto
generar Muestra a partir de las variables climáticas por defecto
mientras (no hay una señal de terminado):
      insertar Muestra en la base de datos
      "muestrear" (generar variables climáticas aleatoriamente)
      generar Muestra a partir de las variables climáticas
```

La señal de terminado referida es generada por alguno de estos dos eventos:

- el botón *Inicio* es presionado, con lo que se redirige al usuario a la pantalla de bienvenida y se llama a *stop_process* en *aux_pro.py*
- se cambia/sale de la ventana, con lo que se convoca al JQuery *$(window).on(unload, …);* que tiene el mismo efecto que el punto anterior.

SERVIDOR

Éste es el encargado de permitir la carga (imagen y comportamiento) de las páginas, y responder a los *requests* del usuario interactuando, al mismo tiempo, como consumidor de la base de datos. Está expuesto en el puerto 8888 (esto es, si se monta el servidor en *localhost*, para acceder a la aplicación se debe invocar la URL *"localhost:8888"*.

Se definen "rutas" a partir de los decoradores de *flask* "*@app.route(<dirección>, methods = [<métodos>]* en */www/app.py*. A partir de esto, se habilitan los accesos a las distintas secciones de la aplicación: las dos páginas visibles ("*/"* y "*/monitor*"), una más para solicitar la recuperación y tratamiento de datos muestreados ("*/monitor/get_data*"), y otra para detener el proceso de muestreo ("*/monitor/stop*"), estas dos últimas no renderizadas.

PROBLEMAS DE CONCURRENCIA Y TIEMPO REAL

Durante el desarrollo del proyecto de recolección de datos de tiempo real, es necesario considerar las limitaciones que éste presenta. Al trabajar en paralelo con recursos compartidos, se establece una coordinación implícita entre los mismos con el

fin de lograr el correcto funcionamiento sin presentar fallas dadas, por ejemplo, por un acceso simultáneo a la base de datos. La metodología de trabajo productor-consumidor conlleva la necesidad de crear una sección crítica donde el productor tiene acceso exclusivo, mientras que los consumidores pueden hacer uso simultáneo de la misma. De esta forma, se utiliza una metodología de trabajo con exclusión mutua. Al añadir la opción al usuario de configurar el tiempo de muestreo, el problema se hace más evidente y se añade un factor de complejidad adicional.

La referencia al tiempo real impone un régimen estático de trabajo sin posibilidad de flexibilidad ante la generación de errores: un dato recibido fuera de su rango temporal lo convierte en basura y necesita ser reemplazado por uno nuevo. A esto se le suman las fluctuaciones de los tiempos físicos de la comunicación entre componentes que, siendo aleatorias, no pueden ser reemplazadas por una simulación estática. Para ello es necesario preparar al sistema para que funcione dentro de límites temporales aproximados, para luego adjuntarle el bloque de adquisición real.