# Taller de Proyecto II - año 2018

## Facultad de Informática

## Universidad Nacional de La Plata

## Práctica 1

Fecha de entrega: 30 de agosto de 2018

Autor: Dibene Santiago

# Ejercicios

Realizar un fork del template que se encuentra en el repositorio https://github.com/gmaron/tp2_template y comenzar a desarrollar las siguientes consignas:

1) 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 (process.py) 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 cada 1 segundo.

b) El servidor flask (app.py) tendrá que servir páginas HTML para mostrar:

i) Promedio de las últimas 10 muestras de cada variable

ii) La última muestra de cada variable tomada por la placa

iii) Selección de período de muestreo. El usuario puede elegir entre un conjunto predefinido de períodos de muestreo (ej: 1, 2, 5, 10, 30, 60 segundos). Este cambio es sólo para el refresco de la página web. NO DE LA PLACA .

Aclaración : Se deberá detallar todo el proceso de adquisición de datos, cómo se ejecutan ambos procesos, el esquema general y la interacción del usuario. El documento HTML generado debe ser accesible, responsivo y realizado con Bootstrap 4.

2) 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.
3) ¿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.

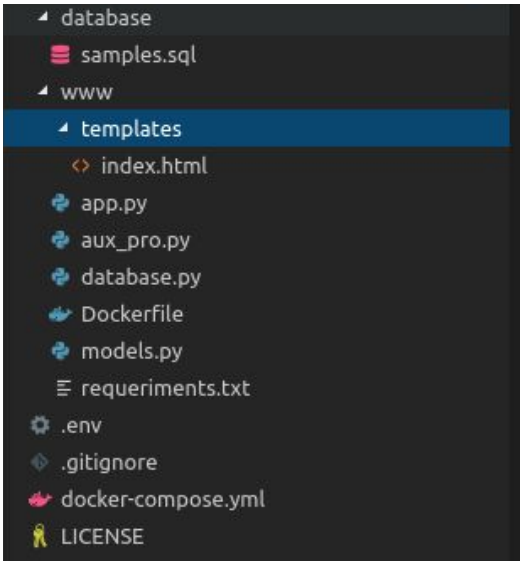
# Resolución

## Introducción

Template: el proyecto basado en el siguiente template, está constituido por las siguientes tecnologías:

**Docker**, es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización, en otras palabras es el que se encarga de darle soporte y alojar el proyecto con su conjunto de aplicaciones.

**Python** con librería **sqlalchemy**, el proyecto estará basado en el lenguaje python, es un lenguaje interpretado, usa tipado dinámico y es multiplataforma, como filosofía hace hincapié en una sintaxis que favorezca un código legible.
**MySQL,** se usará como motor de base de datos, ya que con python se trabajará con objetos se hará uso de la librería SQLAlchemy, un ORM, que nos brinda todas las herramientas para almacenar y manipular los objetos en la base de datos.
**Flask** que incluye **Jinja2**, Flask es un framework minimalista escrito en Python que permite crear aplicaciones web rápidamente, y Jinja2 es el motor que usa este framework para renderizar las vistas.
**phpMyAdmin**, es una herramienta escrita en PHP con la intención de manejar la administración de MySQL a través de páginas web.
**Bootstrap**, es una librería que brinda herramientas para el diseño de paginas web, estas son estilos y elementos javascript.
Estructura del template:




database: en esta carpeta se almacena la consulta para crear la base de datos
www: punto de acceso web de la aplicación, dentro se almacenan las vistas en templates, luego se creará una carpeta llamada static donde se almacenarán los recursos, además se almacenan los archivos .py que son utilizados para controlar toda la aplicación.

## Ejercicio 1)

a) Para simular el proceso que almacenará los valores meteorológicos simulados mediante una función random, en una base de datos primero, se creo un objeto, Samples que cuenta con los atributos meteorológicos.
La clase Sample se almacena en el archivo models.py, en este archivo se mapeara el objeto con la DB mediante sqlalchemy.

Luego el archivo process.py cuenta con solo 2 metodos:
- Main: para instanciar el proceso y que en un bucle infinito cree y almacene las muestras en la base de datos cada 1 seg, mediante una conexión creada a la base de datos almacenada en la variable session.
Está conexión se genera mediante la libreria sqlalchemy y el metodo se almacena en el archivo database.py junto con metodos para acceder al modelo, haciendo de repositorio.
- GracefulKiller: es el otro método, este sirve para cerrar el proceso cuando se cierre la aplicación.

b) El archivo app.py se usara de ruteador, cada método agarra una petición http y devuelve lo pedido

i) El método lastTenAction es llamado cuando se accede a la página '/lastten', con petición tipo GET.
Este método pide las últimas 10 muestras del repositorio, para luego genera, formatear y devuelve en formato json las 10 muestras.
Ademas de este metodo, en app.py se encuentran 3 métodos más, el primero index(), es para renderizar la aplicación, está será una app SinglePage.
luego los otros dos métodos, start() y stop() son para simular la recolección de datos del sensor, para que el proceso anteriormente mencionado, no esté todo el tiempo guardando datos en la base de datos indefinidamente, ya que es una aplicación de prueba.
ii) La última muestra, se toma conjunto las últimas 10, mediante una petición ajax a la ruta /lastten, enviada desde el cliente, para luego, encargarse mediante javascript de calcular el promedio de estas 10 y mostrarlo además de la última.
iii) utilizando los 2 primeros incisos se configura mediante javascript, que por defecto, envie la peticion ajax, cada 1 segundo, y que cada vez que el usuario elija otro periodo de muestreo T mediante un input select, la petición se mande cada ese T periodo de muestreo.

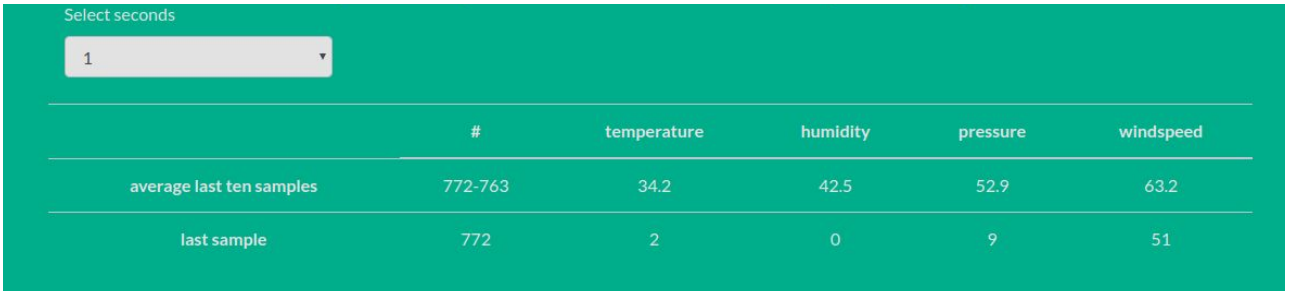
## Proceso de adquisición de datos.

### Frontend

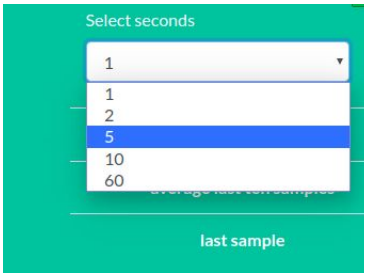
Para las vistas se utilizó el siguiente template de boostrap 4:
https://startbootstrap.com/template-overviews/freelancer/
1ro se presiona en start sensor, esto enviará una petición http y se ejecutará el process.py y se comenzará a simular cómo se generan datos.


2do se presiona en start samples, para empezar a enviar las peticiones ajax que traerá y mostrará los resultados


3ro automáticamente se empezaran a mostrar las muestras, de sensado, por defecto cada 1 segundo.




4to el usuario puede elegir cada cuánto se muestran los resultados, esto hace que se modifique cada cuanto se manda la petición ajax.



## Backend

Cuando se ingresa en localhost:8888

```
@app.route('/')
def index():
    return render_template('index.html')
```

Para iniciar o detener el proceso de sensado

```
@app.route('/stop')
def stopAction():
    pro.stop_process()
    return render_template('index.html')

@app.route('/start')
def startAction():
    pro.start_process()
    return render_template('index.html')
```

Este metodo inicia o detiene el proceso que simula el sensado:

```
def main(session):
    killer = GracefulKiller()
    while(1):
        sample = Sample()
        sample.humidity = random.randint(0,100)
        sample.pressure = random.randint(0,100)
        sample.temperature = random.randint(0,100)
        sample.windspeed = random.randint(0,100)

        session.add(sample)
        session.commit()

        print("Add new sample: %s %s %s %s" % (sample
        sys.stdout.flush()

        time.sleep(1)
        if killer.kill_now:
            session.close()
            break
```

Luego la función ajax que envía la petición para procesar y mostrar los datos.

```
$(document).ready(function(){
  sec = 1000; //default
  intervalId=0; // id timeout
  $( "#selectSeconds" ).change(function() {
     sec =  $( "#selectSeconds" ).val();
   });
  $( "#startSamples" ).click(function() {
    sendRequest();
  });
  $( "#stopSamples" ).click(function() {
    clearInterval(intervalId);
  });
  function sendRequest(){
     $.ajax({
      url: "/lastten",
      success:
       function(data){
          //process data
          average=Array();
          average["temperature"]=0;
          average["humidity"]=0;
          average["pressure"]=0;
          average["windspeed"]=0;

          data.forEach(element => {
             average["temperature"]+=element.temperature;
             average["humidity"]+=element.humidity;
             average["pressure"]+=element.pressure;
             average["windspeed"]+=element.windspeed;
          });

          average["temperature"]=average["temperature"]/10;
          average["humidity"]=average["humidity"]/10;
          average["pressure"]=average["pressure"]/10;
          average["windspeed"]=average["windspeed"]/10;

          $( "#average" ).text(data[0].id+'-'+data[9].id);
          $( "#average_temperature" ).text(average["temperature"]);
          $( "#average_humidity" ).text(average["humidity"]);
          $( "#average_pressure" ).text(average["pressure"]);
          $( "#average_windspeed" ).text(average["windspeed"]);
          $( "#last" ).text(data[0].id);
          $( "#temperature" ).text(data[0].temperature);
          $( "#humidity" ).text(data[0].humidity);
          $( "#pressure" ).text(data[0].pressure);
          $( "#windspeed" ).text(data[0].windspeed);
      },
      complete: function() {
        clearInterval(intervalId);
        intervalId = setInterval(sendRequest, sec);
    }
   });
  };
});
```

Está función pide los datos a /lastten

```
@app.route('/lastten', methods = ["GET"])
def lastTenAction():
    samples = db.getSamples()
    print('[%s]' % ', '.join(map(str, samples)))
    sys.stdout.flush()
    jsonSamples=[]
    for s in samples:
        jsonSamples.append(s.serialize())
    print('[%s]' % ', '.join(map(str, jsonSamples)))

    sys.stdout.flush()
    return jsonify(jsonSamples)
```

lastTenAction toma los datos del repositorio y los formatea para devolver una respuesta en json.

Para obtener los datos, la función en el repositorio, mediante la session creada con sqlalchemy , usando la función query para obtener los datos de la DB.

```
def getSamples(self):
    session = self.get_session()
    samples = session.query(Sample).order_by(Sample.id.desc()).limit(10).all()
    session.close()
    return samples
```

## Ejercicio 2)

Este problema es un clásico problema de concurrencia productor consumidor, en este caso el productor sería el proceso que sensa la información y el consumidor el proceso que pide los datos. Ambos procesos lo piden mediante la  session de sqlalchemy.
Acá entran los requerimientos y el problema de tiempo real, hay que definir cuánto se tarda / cuánto se puede tardar, entre que el productor deja la muestra y se consume por el cliente.

En cuanto sqlalchemy y como maneja la session según la documentación:
is the session thread safe? no recomiendan que dos procesos compartan la session, y que si lo hacen que se resuelva la concurrencia cuando se inserta y actualiza el modelo.
En el ambiente real, puede suceder que se tarde más en insertar en la base de datos, o obtener de la base de datos, se tienen que considerar las variables de conexión y retardos que pueden llegar a ocurrir.
Además de estos problemas, se pueden producir otros en cuanto al muestreo de uno y otro proceso, hay que planificar bien las tareas de ambos procesos para que no se superpongan y que cada una obtenga su respuesta en tiempo y forma

## Ejercicio 3)

En la simulación planteada, se define un modelo en el que no tiene ningún problema de tiempos, se generan valores random y se insertan cada 1 segundo según el proceso. Como se dijo anteriormente en un sistema real, pueden existir retardos, ruidos de medición, ruidos en la comunicación, variables que hacen al modelo más difícil de plantear y calcular.
Según se quiera, se puede agregar más parecido al real, con la desventaja en dificultar el modelo, y con la ventaja de que sea más preciso.