

Taller 2 – Modelos en Django

1. Objetivo general

Implementar paso a paso una aplicación Django y explorar el diseño de los modelos que definen la estructura de los datos almacenados.

2. Contexto

En este taller, se va a tomar como ejemplo el caso de estudio “Confort Térmico” cuyo enunciado se puede encontrar en el siguiente enlace.

https://uniandes.sharepoint.com/:w:/s/isis2503/EU255aDxROBKvCIDKUz0n_0BVwNpvVG3vRkkyGqzuun9gA?e=LKukp6

En el siguiente repositorio puede encontrar el código fuente que es el punto de partida de este taller:

<https://github.com/ISIS2503/TallerDjango-Modelos.git>

Este repositorio no contiene la base de datos del proyecto. Descargue el código fuente y realice las siguientes instrucciones para crear la base de datos y aplicar las migraciones:

- Desde la consola, ubíquese en el directorio base del proyecto y ejecute las siguientes instrucciones:

```
$ python manage.py makemigrations  
$ python manage.py migrate
```

3. Actividades

NOTA 1: Se espera que el estudiante realice estas actividades en su computador personal o en algún computador que cuente con una interfaz gráfica.

NOTA 2: Se sugiere utilizar algún editor de texto para Python como PyCharm, el cual ofrece una licencia para estudiantes. Puede descargarlo desde el sitio oficial en el siguiente enlace:

<https://www.jetbrains.com/pycharm/>

3.1 Definición de Modelos

Una aplicación web en Django administra los datos a través de objetos de Python llamados modelos. Los modelos definen la estructura de los datos almacenados, atributos, tipos de datos, valores predeterminados, etc. El diseño de un modelo en Django no depende de la base de datos que utilice, es decir, los modelos de su aplicación son iguales para cualquier base de datos como SQLite, PostgreSQL, MySQL, etc. Al final, Django se encargará de comunicarse con la base de datos.

Cuando definimos los modelos para un proyecto Django, uno de las cosas más importantes es crear las relaciones entre estos. Los tres tipos más comunes de relaciones de base de datos son:

- Many-to-one: para definir este tipo de relación en Django, se sigue la siguiente nomenclatura:

```
attribute = models.ForeignKey(  
    model,  
    on_delete=models.CASCADE  
)
```

- Many-to-many: para definir este tipo de relación en Django, se sigue la siguiente nomenclatura:

```
attribute = models.ManyToManyField(  
    model  
)
```

- One-to-one: para definir este tipo de relación en Django, se sigue la siguiente nomenclatura:

```
attribute = models.OneToOneField(  
    model,  
    on_delete=models.CASCADE,  
    primary_key=True  
)
```

En el siguiente enlace puede encontrar la documentación de Django, referente a la creación de modelos, tipos de datos, relaciones, y algunos ejemplos.

<https://docs.djangoproject.com/es/2.2/topics/db/models/>

Analizando el modelo de datos para el caso de estudio “Termal Confort” (Figura 1), vamos a crear dos modelos, uno para **Measurement** y otro para **Variable**, cada uno con sus atributos y tipos de datos respectivos. La relación que

existe entre los dos modelos es *Many-to-one*, donde una instancia de **Variable** se compone de cero o muchas instancias de **Measurement**.

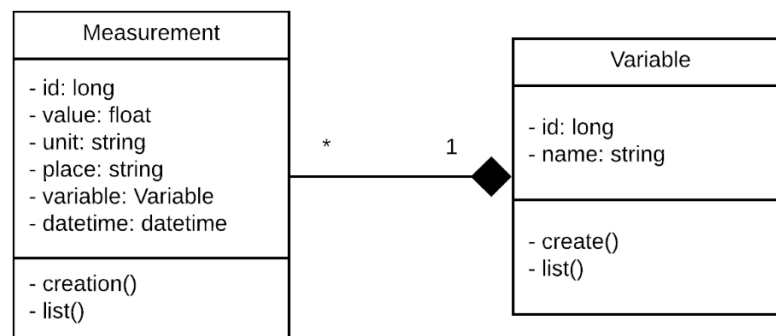


Fig 1. Modelo de datos

El código fuente que se le ha proporcionado para este taller, contiene un proyecto Django nombrado **monitoring**, con dos aplicaciones, **measurements** y **variables**. A continuación, vamos a crear un modelo para cada aplicación.

Modelo “*Variable*”

Cada aplicación contiene un archivo *models.py* para definir los modelos. Vamos a comenzar por definir el modelo para la aplicación **variables**. Diríjase al directorio de la aplicación “*monitoring/variables*” y edite el archivo *models.py*. Debe crear las clases necesarias para modelar las entidades relevantes de la aplicación, en este caso, la clase **Variable** con sus respectivos atributos (**name** de tipo Char). Tenga en cuenta que las clases que defina en el archivo *models.py* deben heredar de la clase *models.Model* que ya se encuentra importado en este archivo por defecto. El contenido de *models.py* para la aplicación **variables**, debe lucir como sigue:

```
1  from django.db import models
2
3  class Variable(models.Model):
4      name = models.CharField(max_length=50)
5
6  def __str__(self):
7      return '{}'.format(self.name)
```

Para el modelo **Variable**, únicamente definimos el atributo **name** de tipo Char. Las líneas 6 y 7, contienen el método de representación textual del modelo, y lo definimos principalmente para obtener la lista de variables por nombre desde la vista de administrador de Django (esto lo podrá ver más adelante en esta guía)

NOTA 1: Los atributos de las clases deben ser de algún tipo definido en *models*. Por ejemplo, atributos que normalmente modelaría como String, puede modelarlo con *models.CharField()* o *models.TextField()*. Para consultar la lista de tipos de campos en la documentación oficial de Django:

<https://docs.djangoproject.com/en/2.2/ref/models/fields/#model-field-types>

NOTA 2: Note que los métodos de *models* que definen el tipo, como el *CharField()* a veces reciben parámetros como *null=True* o *max_length*. Estos parámetros pueden entenderse como las *constraints* de la base de datos. Para ver la lista completa puede consultar la documentación oficial de Django en

<https://docs.djangoproject.com/en/2.2/ref/models/fields/>

Modelo “Measurement”

A continuación, definimos el modelo para la aplicación **measurements**. Diríjase al directorio de la aplicación “monitoring/measurements” y edite el archivo *models.py*. El contenido de este archivo debe lucir como sigue:

```
1 from django.db import models
2 from variables.models import Variable
3
4 class Measurement(models.Model):
5     variable = models.ForeignKey(Variable, on_delete=models.CASCADE)
6     value = models.FloatField(null=True, blank=True, default=None)
7     unit = models.CharField(max_length=50)
8     place = models.CharField(max_length=50)
9     dateTime = models.DateTimeField(auto_now_add=True)
10
11     def __str__(self):
12         return '%s %s' % (self.value, self.unit)
```

El atributo *variable* es una llave foránea que representa la relación muchos a uno que existe entre los modelos **Measurement** y **Variable**. Note que es necesario importar el modelo **Variable** (línea 2) de la aplicación **variables**, para poder definir la relación (línea 5).

Siempre que haga cambios en los modelos del proyecto, debe crear las migraciones y aplicarlas a la base de datos. Para esto, desde la consola ubíquese en el directorio raíz del proyecto, y ejecute los siguientes comandos:

```
$ python manage.py makemigrations
$ python manage.py migrate
```

3.2 Interfaz de Administrador Django

Luego de aplicar las migraciones a la base de datos (SQLite por defecto), vamos a comprobar los modelos y la creación de las tablas desde la vista de administrador de Django. Django ofrece una interfaz gráfica de usuario (GUI) interactiva que se encarga de todas las actividades administrativas. Cada proyecto web requiere una interfaz de administración lista para usar y Django provee una interfaz generada automáticamente para administración de acuerdo con los modelos de proyecto proporcionados.

A continuación, configure el superusuario de Django y agregue los modelos a la vista de administrador.

- Para crear el superusuario, desde la consola ubíquese en el directorio raíz del proyecto, y ejecute el siguiente comando:

```
$ python manage.py createsuperuser
```

Ingresa la información para el usuario administrador.

- Los modelos que ha definido anteriormente debe registrarlos en el archivo *admin.py* de la aplicación que los contiene. Es decir, para registrar el modelo **Measurement**, edite el archivo *admin.py* de la aplicación **measurements** que se encuentra en el directorio “monitoring/measurements”. El contenido de este archivo debe lucir como sigue:

```
1 from django.contrib import admin
2 from .models import Measurement
3
4 admin.site.register(Measurement)
```

Note que es necesario importar el modelo **Measurement** (línea 2)

- Registre el modelo **Variable** al igual que lo hizo en el paso anterior. Pero esta vez edite el archivo *admin.py* de la aplicación **variables** que se encuentra en el directorio “monitoring/variables”. El contenido de este archivo debe lucir como sigue:

```
1 from django.contrib import admin
2 from .models import Variable
3
4 admin.site.register(Variable)
```

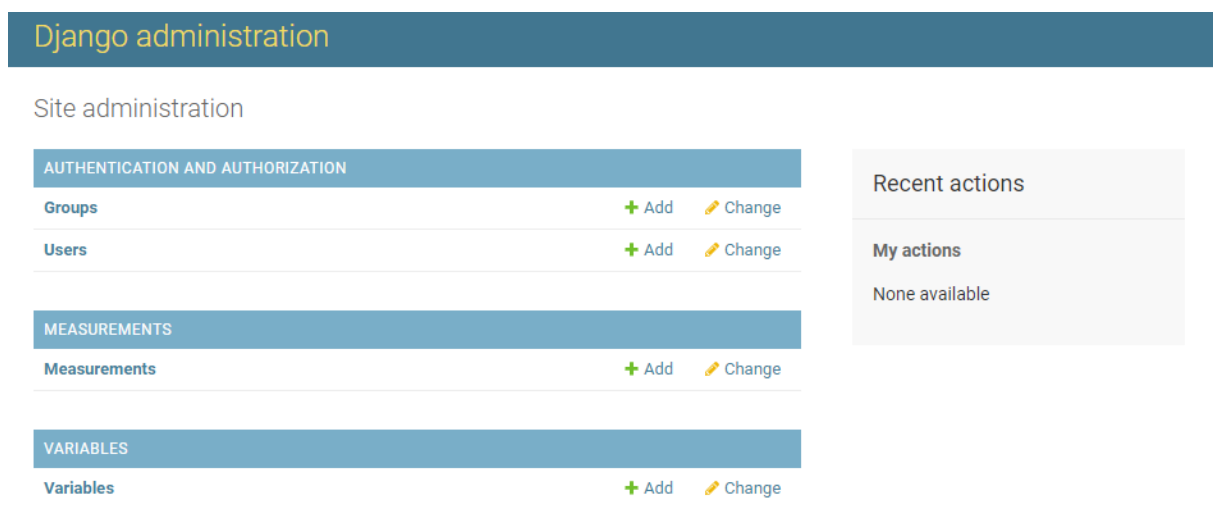
3.3 Ejecutar Aplicación Web y abrir el administrador Django

A continuación, ejecute la aplicación web, y verifique desde la vista de administrador de Django, la creación de los modelos.

- Desde la consola, ubíquese en el directorio base del proyecto y ejecute la aplicación con el siguiente comando:

```
$ python manage.py runserver
```

- Desde el navegador ingrese a la dirección “localhost:8000/admin” y auténtíquese con los datos que ha definido anteriormente para el superusuario. La vista de administrador deberá mostrarle los modelos **Measurements** y **Variables** como sigue:





- Si hace clic en una de las entidades, podrá acceder a los datos persistidos en la base de datos, editarlos, y borrarlos desde la interfaz gráfica del administrador. Ahora, cree algunas variables como Temperature, Moisture y Oxygen.
- A continuación, cree algunas medidas (measurements). Puede notar que, al momento de crear un registro de una medida, debe seleccionar una única variable (debido a la relación Many-to-one o ForeignKey que configuro en el modelo), como se observa en la siguiente imagen:

Django administration

Home > Measurements > Measurements > Add measurement

Add measurement

Variable:  

Value:

Unit:

Place:

Temperature
Moisture
Oxygen

4. Entregables

En el proyecto Django, cree una nueva aplicación llamada **alarms**. Suponga que la entidad **alarm** tiene una relación muchos-a-muchos con la entidad **measurement**. A continuación, cree un modelo para la aplicación **alarms** y defina por lo menos un atributo del tipo ManyToMany que represente la relación con **measurements**.

Haga los cambios pertinentes para agregar la aplicación **alarms** al administrador de Django (sección 3.2 de esta guía).

- En un documento registre los pasos necesarios con tomas de pantalla, para realizar el siguiente procedimiento:

Ejecute la aplicación, y desde la vista del administrador de Django, cree varios registros de **measurements**. Luego cree un registro de **alarm** y compruebe que la relación ManyToMany que ha definido en el modelo, funciona correctamente.