

Taller 3 -Arquisoft

Felipe Rueda- 202010903

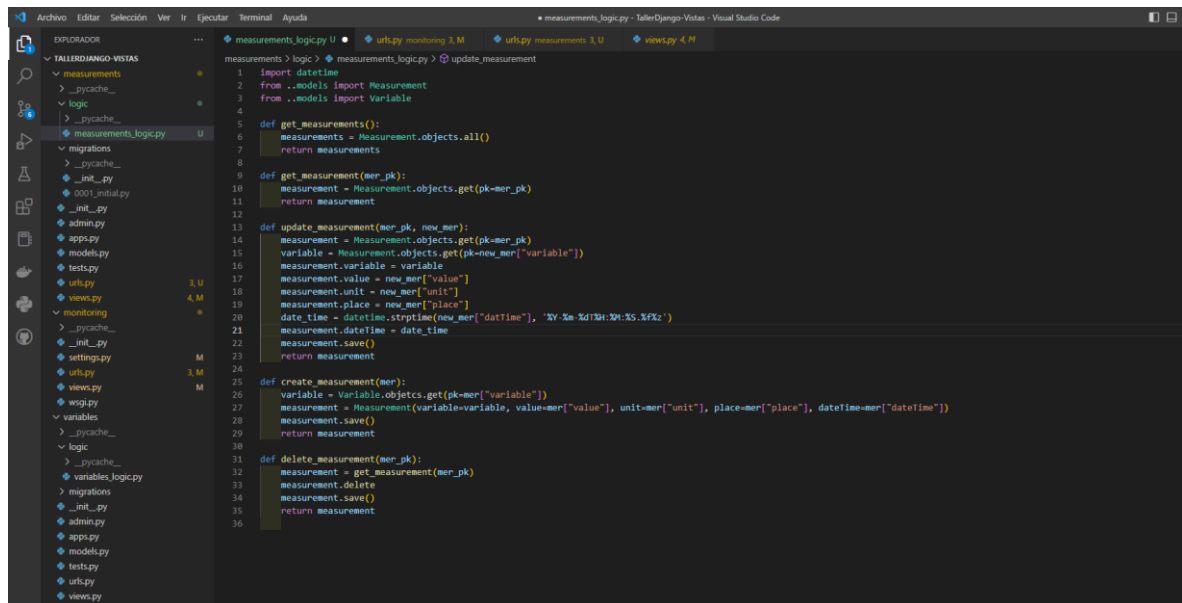
Repositorio: <https://github.com/Feru34/TallerDjango-Vistas.git>

Evidencias del código creado para Measurements

Evidencia 1: Dentro de l carpeta variables, se crea el archivo que se encarga de las actualizaciones, Measurements_logic.py.

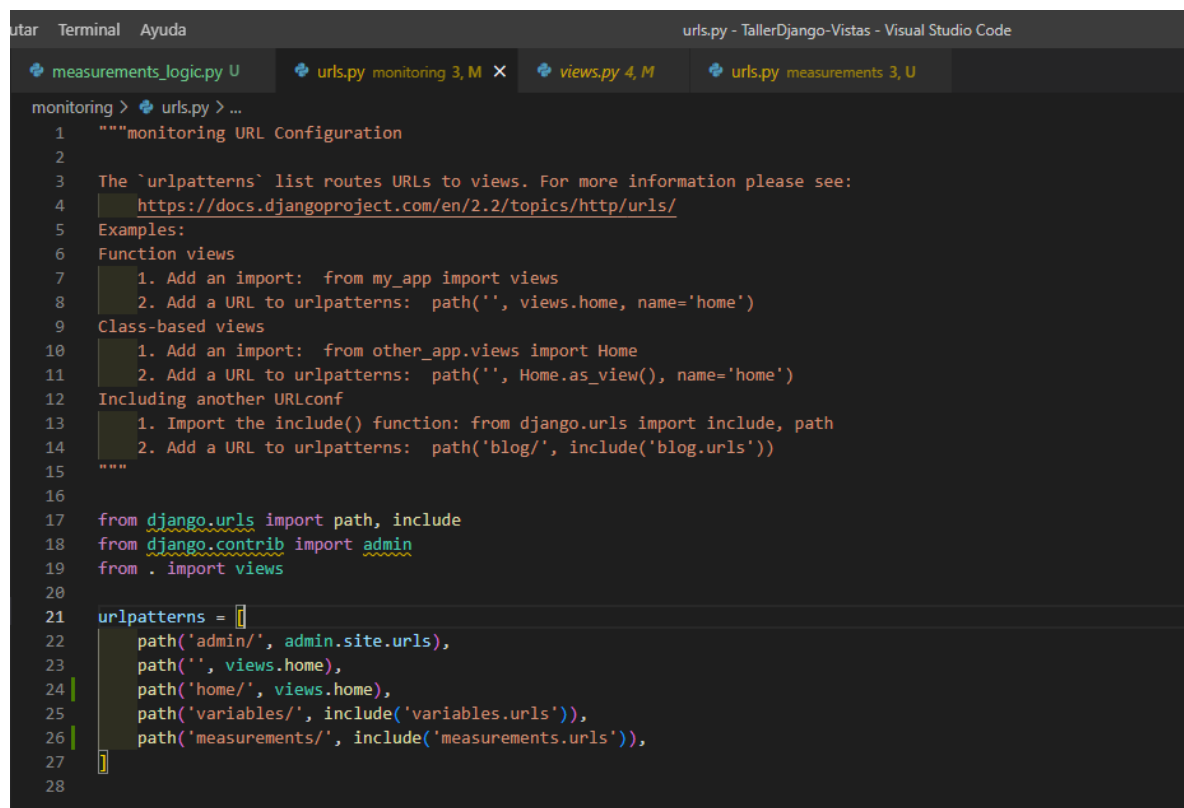
- Se crea la función get_measurements() Consultar lista de todas las medidas, se utiliza .objects.all() para extraer todas las Measurements.
- Se crea la función get_measurement() para tomar una en específico requiere una PK de objeto al quiere consultar.
- Se crea la función create_measurement() el cual crea una medida con una de la creación de otra variable cuya llave primaria corresponde al parámetro.
- Se crea la función update_measurement() la cual cambia una medida común identificador en específico, esta también incluye una llave foránea de Variable.
- Se crea la función delete_measurement() () la cual elimina una medida común identificador en específico.

Todas llamando los respectivos atributos del objeto para su consulta o modifción. Así mismo, se respeta la llave foránea que tiene con la clase variables como vemos en la línea 15 y 26.



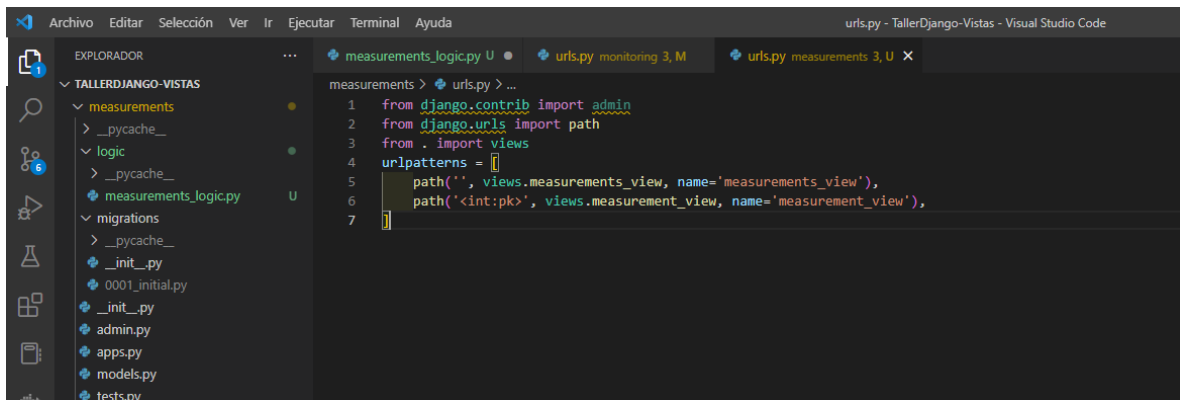
```
1 import datetime
2 from ..models import Measurement
3 from ..models import Variable
4
5 def get_measurements():
6     measurements = Measurement.objects.all()
7     return measurements
8
9 def get_measurement(mer_pk):
10     measurement = Measurement.objects.get(pk=mer_pk)
11     return measurement
12
13 def update_measurement(mer_pk, new_mer):
14     measurement = Measurement.objects.get(pk=mer_pk)
15     variable = Measurement.objects.get(pk=new_mer["variable"])
16     measurement.variable = variable
17     measurement.value = new_mer["value"]
18     measurement.unit = new_mer["unit"]
19     measurement.place = new_mer["place"]
20     date_time = datetime.strptime(new_mer["datetime"], "%Y-%m-%dT%H:%M:%S.%fZ")
21     measurement.datetime = date_time
22     measurement.save()
23     return measurement
24
25 def create_measurement(mer):
26     variable = Variable.objects.get(pk=mer["variable"])
27     measurement = Measurement(variable=variable, value=mer["value"], unit=mer["unit"], place=mer["place"], datetime=mer["datetime"])
28     measurement.save()
29     return measurement
30
31 def delete_measurement(mer_pk):
32     measurement = get_measurement(mer_pk)
33     measurement.delete()
34     measurement.save()
35     return measurement
36
```

Evidencia 2: Se crea el path con Measurements junto con la directiva include, el cual marca como comenzarán la base de las direcciones URLs del objeto



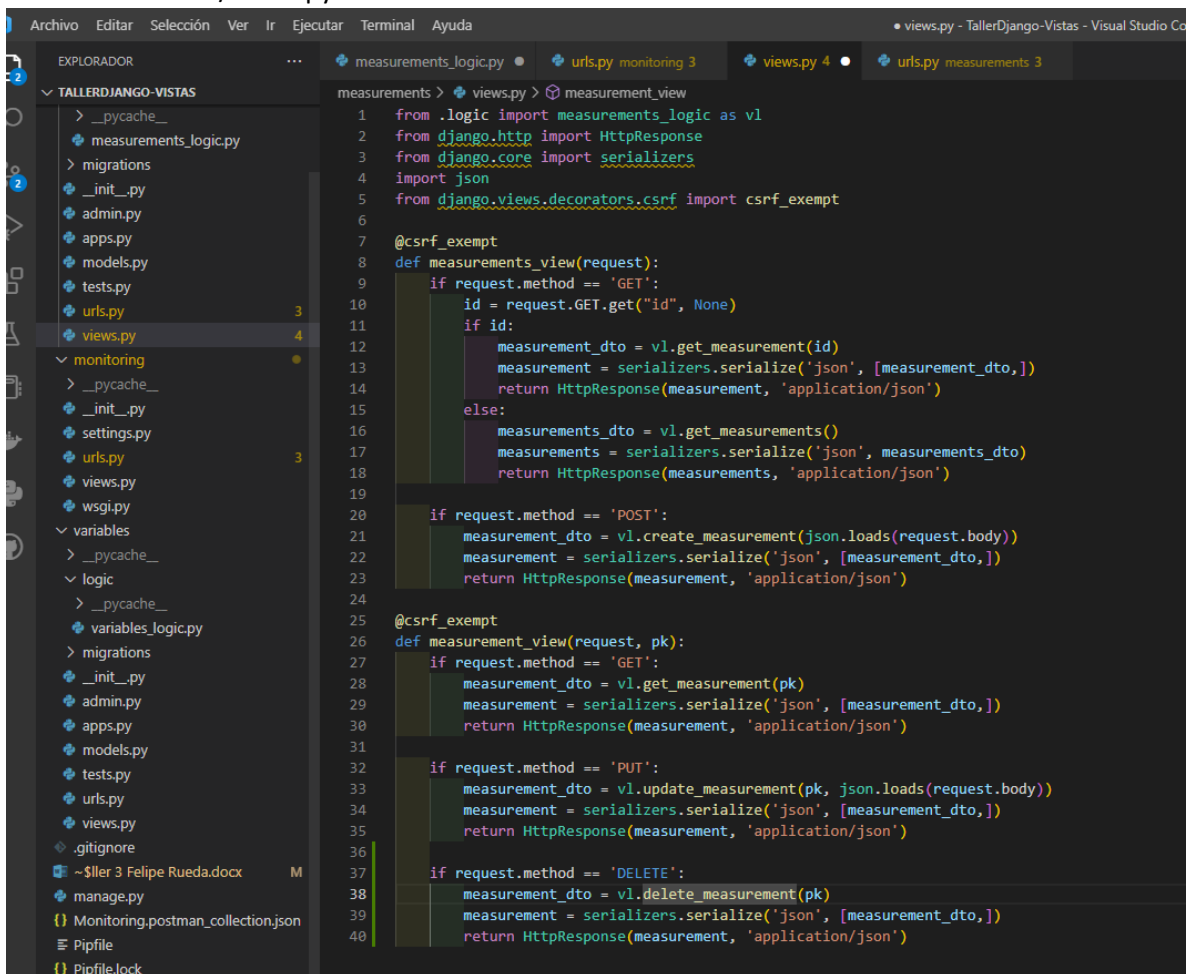
```
monitoring > urls.py > ...
1  """monitoring URL Configuration
2
3  The `urlpatterns` list routes URLs to views. For more information please see:
4  https://docs.djangoproject.com/en/2.2/topics/http/urls/
5  Examples:
6  Function views
7      1. Add an import: from my_app import views
8      2. Add a URL to urlpatterns: path('', views.home, name='home')
9  Class-based views
10     1. Add an import: from other_app.views import Home
11     2. Add a URL to urlpatterns: path('', Home.as_view(), name='home')
12  Including another URLconf
13     1. Import the include() function: from django.urls import include, path
14     2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
15  """
16
17  from django.urls import path, include
18  from django.contrib import admin
19  from . import views
20
21  urlpatterns = [
22      path('admin/', admin.site.urls),
23      path('', views.home),
24      path('home/', views.home),
25      path('variables/', include('variables.urls')),
26      path('measurements/', include('measurements.urls')),
27  ]
28
```

Evidencia 3: Se crean las URLs que se usarán en la página para measurements, son bastante parecidas a las de variables pues tienen la misma lógica.



```
1 from django.contrib import admin
2 from django.urls import path
3 from . import views
4 urlpatterns = [
5     path('', views.measurements_view, name='measurements_view'),
6     path('<int:pk>', views.measurement_view, name='measurement_view'),
7 ]
```

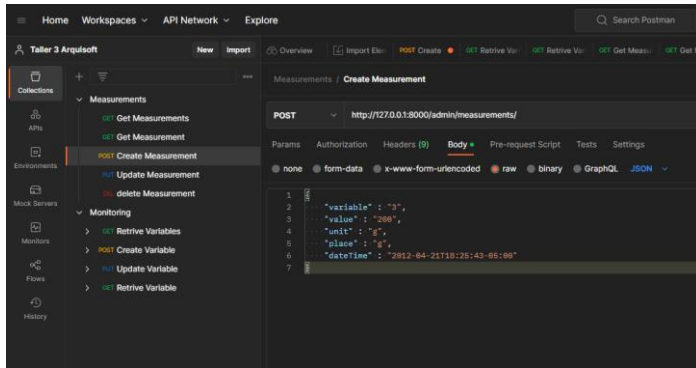
Evidencia 3: Posteriormente, dentro del archivo que se encarga de las consultas, views.py, se agrega el decorador `@csrf:exempt` para poder realizar las pruebas postman sin necesidad de un token de seguridad. En `measurements/views.py` se crean las vistas de forma casi iguales a como se hizo con `variables/views.py`



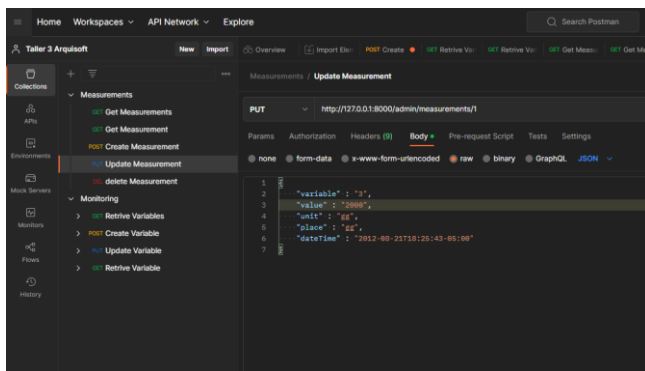
```
1 from .logic import measurements_logic as vl
2 from django.http import HttpResponse
3 from django.core import serializers
4 import json
5 from django.views.decorators.csrf import csrf_exempt
6
7 @csrf_exempt
8 def measurements_view(request):
9     if request.method == 'GET':
10         id = request.GET.get("id", None)
11         if id:
12             measurement_dto = vl.get_measurement(id)
13             measurement = serializers.serialize('json', [measurement_dto,])
14             return HttpResponse(measurement, 'application/json')
15         else:
16             measurements_dto = vl.get_measurements()
17             measurements = serializers.serialize('json', measurements_dto)
18             return HttpResponse(measurements, 'application/json')
19
20     if request.method == 'POST':
21         measurement_dto = vl.create_measurement(json.loads(request.body))
22         measurement = serializers.serialize('json', [measurement_dto,])
23         return HttpResponse(measurement, 'application/json')
24
25 @csrf_exempt
26 def measurement_view(request, pk):
27     if request.method == 'GET':
28         measurement_dto = vl.get_measurement(pk)
29         measurement = serializers.serialize('json', [measurement_dto,])
30         return HttpResponse(measurement, 'application/json')
31
32     if request.method == 'PUT':
33         measurement_dto = vl.update_measurement(pk, json.loads(request.body))
34         measurement = serializers.serialize('json', [measurement_dto,])
35         return HttpResponse(measurement, 'application/json')
36
37     if request.method == 'DELETE':
38         measurement_dto = vl.delete_measurement(pk)
39         measurement = serializers.serialize('json', [measurement_dto,])
40         return HttpResponse(measurement, 'application/json')
```

Evidencia Pruebas Postman:

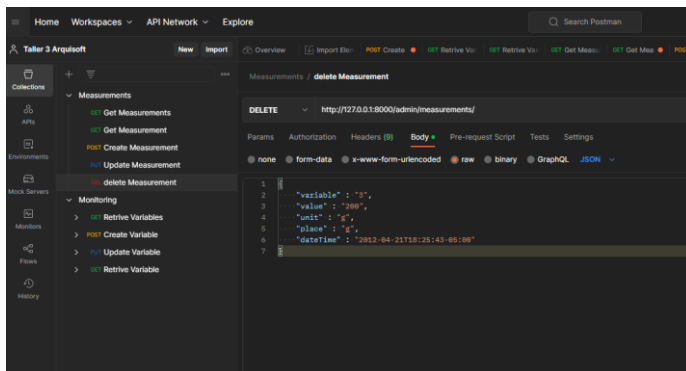
Evidencia 1: Create Measurement



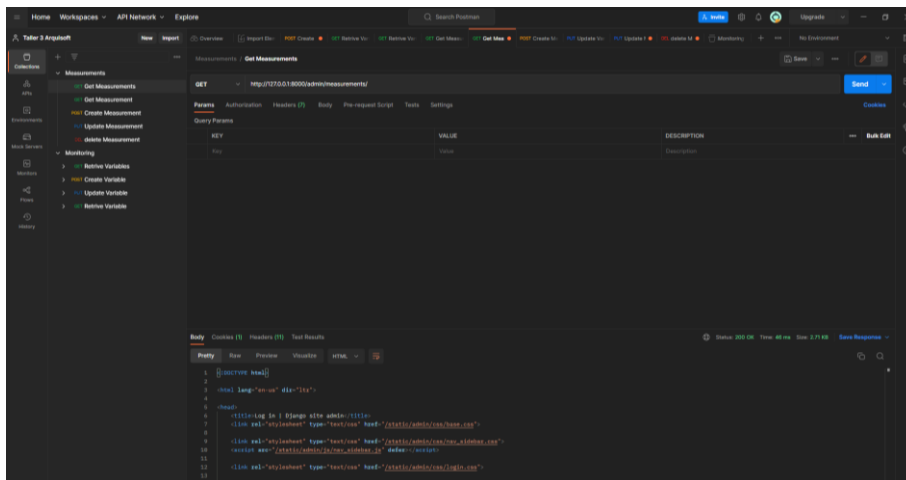
Evidencia 2: Update Measurement



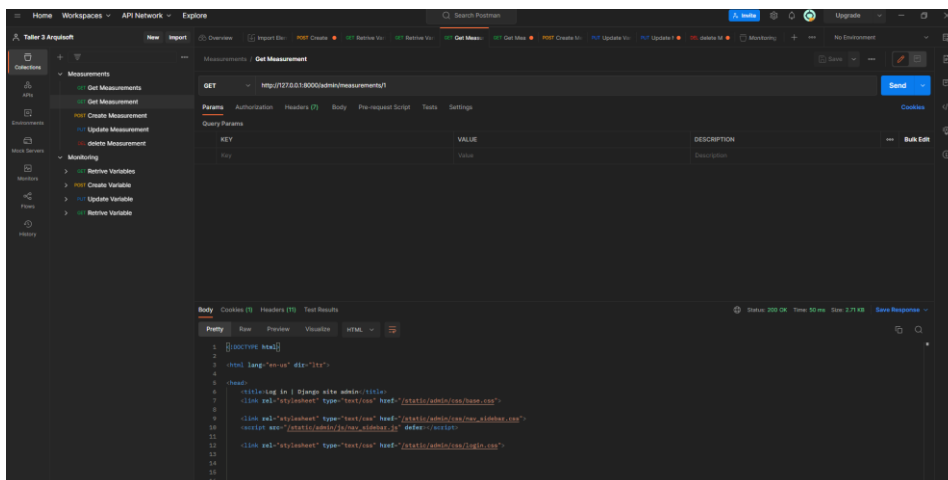
Evidencia 3: Delete Measurement



Evidencia 4: Get Measurements



Evidencia 1: Get Measurement



Desafortunadamente las pruebas Postman no funcionaron, aunque el código y la lógica estuviera correcta al criterio del redactor, aparecía en consola un error inesperado el cual no se pudo solucionar, este marcaba que el token de seguridad era incorrecto. Algo extraño pues se estaba usando el decorador `@csrf:exempt` justamente para evitar este problema. A continuación, el error en cuestión:

```
2/Feb/2023 19:39:38] "DELETE /admin/measurements/ HTTP/1.1" 403 2519
2/Feb/2023 19:39:58] "GET /variables?id=3 HTTP/1.1" 301 0
2/Feb/2023 19:39:58] "GET /variables/?id=3 HTTP/1.1" 200 73
rbidden (CSRF token missing or incorrect.): /admin/measurements/
2/Feb/2023 19:40:05] "POST /admin/measurements/ HTTP/1.1" 403 2519
rbidden (CSRF token missing or incorrect.): /admin/measurements/
2/Feb/2023 19:40:06] "POST /admin/measurements/ HTTP/1.1" 403 2519
rbidden (CSRF token missing or incorrect.): /admin/measurements/
2/Feb/2023 20:10:11] "POST /admin/measurements/ HTTP/1.1" 403 2519
rbidden (CSRF token missing or incorrect.): /admin/measurements/
2/Feb/2023 20:11:15] "POST /admin/measurements/ HTTP/1.1" 403 2519
2/Feb/2023 20:11:15] "POST /admin/measurements/ HTTP/1.1" 403 2519
```

Todas las pruebas están en el repositorio