

Documento practicas Focus Soft -Diego Barboteo Edo

Se pide realizar un proyecto en DJANGO el cual realice la tarea de obtener el precio del kilovatio hora a través de la web OMIE, a parte, hay que comparar el precio para a modo de un semáforo, el cual indique un color respecto al precio. A parte, hay que crear una API que conecte con una aplicación para Android (.APK) para poder consultar lo anteriormente mencionado.

1—Scrapper (Realizado por mi cuenta)

a- Obtener el precio del kWh



```
def obtenerPrecio_kwh(fecha):
    hoy = fecha.strftime("%Y%m%d")
    periodo = fecha.hour + 1
    filename = f"marginalpdbc_{hoy}.1"
    url = f"https://www.omie.es/es/file-download?filename={filename}&parents=marginalpdbc"
    headers = {"User-Agent": "Mozilla/5.0"}

    resp = requests.get(url, headers=headers, timeout=10)
    resp.raise_for_status()

    for linea in resp.text.splitlines():
        partes = linea.split(";")
        if len(partes) >= 6 and partes[0].isdigit():
            try:
                p = int(partes[3])
            except ValueError:
                continue
            if p == periodo:
                precio_mwh = float(partes[4].replace(",", "."))
                return round(precio_mwh / 1000, 5)
    raise ValueError("No se encontró el periodo en el archivo de precios.")
```

El código descarga un archivo .CSV el cual contiene el precio del kWh, el cual filtramos por hora exacta.

b- Obtener el precio de los peajes y cargos

```
def obtenerPeaje(fecha):
    hora = fecha.hour
    if fecha.weekday() >= 5:
        return PEAJE_P3
    elif 10 <= hora < 14 or 18 <= hora < 22:
        return PEAJE_P1
    elif 8 <= hora < 10 or 14 <= hora < 18 or 22 <= hora < 24:
        return PEAJE_P2
    return PEAJE_P3

def obtenerCargo(fecha):
    hora = fecha.hour
    if fecha.weekday() >= 5:
        return CARGO_P3
    elif 10 <= hora < 14 or 18 <= hora < 22:
        return CARGO_P1
    elif 8 <= hora < 10 or 14 <= hora < 18 or 22 <= hora < 24:
        return CARGO_P2
    return CARGO_P3
```

En base a la hora del día o el día (si es festivo) evalúa los precios de los peajes

c- Escribir a la base de datos

```
def guardarConsumo():
    fecha = timezone.now()
    try:
        precio = obtener_precio_kwh(fecha)
        peaje = obtenerPeaje(fecha)
        cargo = obtenerCargo(fecha)

        ConsumoDiario.objects.create(
            fecha_exacta=fecha,
            precio=precio,
            peaje=peaje,
            cargo=cargo
        )
        print(f"Guardado: Precio={precio}, Peaje={peaje}, Cargo={cargo}")
    except Exception as e:
        print(f"Error al guardar en la base de datos: {e}")
```

Desde DJANGO escribimos en la base de datos los precios por separado tanto del peaje, cargo y del kWh

d- Leer desde la base de datos

```
def obtenerConsumoTotal():
    try:
        consumo = ConsumoDiario.objects.latest('fecha_exacta')
        total = consumo.precio + consumo.peaje + consumo.cargo
        return total
    except ConsumoDiario.DoesNotExist:
        print("No hay registros en la base de datos.")
        return None
    except Exception as e:
        print(f"Error al obtener total desde la base de datos: {e}")
        return None
```

A pesar de poder cargar los datos directamente desde el propio programa, es conveniente leer los valores desde la base de datos, ya que si obtuviéramos el precio del kWh por cada vez que se ejecuta el programa en un dispositivo podríamos tener problemas a la hora de ser detectados como robots por múltiples peticiones

2—App consumo (Realizado por mi cuenta)

a- Models.py

```
from django.db import models

class ConsumoDiario(models.Model):
    fecha_exacta = models.DateTimeField()
    precio = models.FloatField()
    peaje = models.FloatField()
    cargo = models.FloatField()

    @property
    def color_alerta(self):
        total = self.precio + self.peaje + self.cargo
        if total <= 0.10:
            return "verde"
        elif total <= 0.20:
            return "amarillo"
        else:
            return "rojo"
```

Evaluamos el precio (del kWh) y en función del valor devolvemos un 'string' con el correspondiente color

b- Views.py

```
from django.http import HttpResponse
from consumo.models import ConsumoDiario

def test_view(request):
    try:
        consumo = ConsumoDiario.objects.latest('fecha_exacta')
        total = consumo.precio + consumo.peaje + consumo.cargo
        color = consumo.color_alerta
        return HttpResponse(f"Total del último consumo: {total:.4f} – Color alerta: {color}")
    except ConsumoDiario.DoesNotExist:
        return HttpResponse("No hay datos disponibles.")
    except Exception as e:
        return HttpResponse(f"Error: {e}")
```

Creamos la vista a la que accedemos al entrar en <http://127.0.0.1:8000/>.

Esto nos devolverá lo que vemos al entrar, tanto el precio del kWh (con 4 decimales) y el color

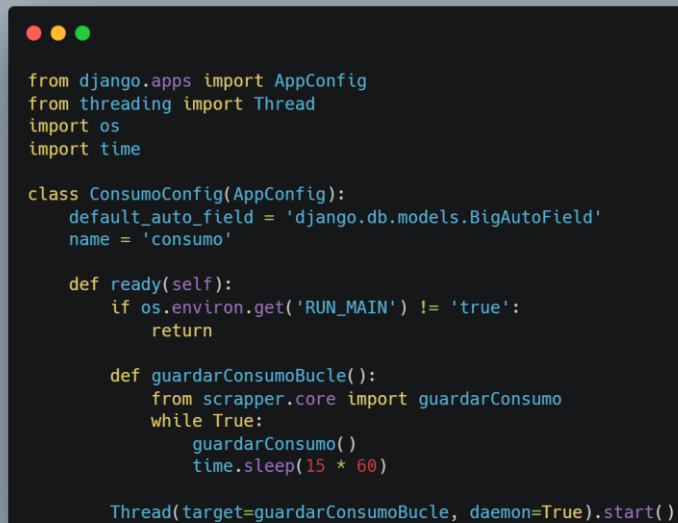
c- Urls.py

```
from django.urls import path
from .views import test_view

urlpatterns = [
    path('test/', test_view, name='test_view'),
]
```

Asigna la vista a la URL

d- Apps.py



```
from django.apps import AppConfig
from threading import Thread
import os
import time

class ConsumoConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'consumo'

    def ready(self):
        if os.environ.get('RUN_MAIN') != 'true':
            return

        def guardarConsumoBucle():
            from scrapper.core import guardarConsumo
            while True:
                guardarConsumo()
                time.sleep(15 * 60)

        Thread(target=guardarConsumoBucle, daemon=True).start()
```

La funcion para escribir en la base de datos cada 15 minutos, esta se activa automáticamente. Además, crea un hilo (o subproceso) a parte para evitar congelar todo el programa.

3—Proyecto

a- Settings.py

En settings.py solo he modificado un par de cosas respecto al base

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'consumo',  
]
```

Añadir la APP consumo, la cual es la responsable de la view, app, y urls asignadas

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.postgresql',  
        'NAME': 'consumo_bd',  
        'USER': 'postgres',  
        'PASSWORD': 'password',  
        'HOST': 'localhost',  
        'PORT': '5432',  
    }  
}
```

Configurar la base de datos (LOCAL) postgre

4—API (Realizado por parte de mis compañeros, no he participado)

5—APK (Realizado por parte de mis compañeros, no he participado)

6—Docker (Realizado por parte de mis compañeros, no he participado)