



Procesamiento de información de un parque eólico a través de FastAPI

Grupo 2 - Iñigo Murga, Jon Cañadas y Mikel García



Reto abordado

Este proyecto consiste en la concentración y validación de datos de un parque eólico. Para ello utilizamos FastAPI, Pydantic y Uvicorn.

- Se generan datos sintéticos de 10 molinos de viento de manera simultánea.
- Uvicorn gestiona las conexiones web desde el navegador o el cliente API y permite que FastAPI gestione las solicitudes HTTP.
- Se calculan diferentes promedios de las variables de los molinos.



Simulación de los generadores

Para simular los molinos de viento tenemos un script que genera datos sintéticos con un 15% de error.

Para no tener que abrir 10 diferentes terminales tenemos un archivo "lanzar.bat" que ejecutara 10 veces el archivo "molino.py".

```
def generar_dato(id_generador: int, prob_error: float = 0.15):  
    """Genera un dato sintético para un generador eólico con una probabilidad de error en un solo campo."""  
  
    # Datos correctos  
    potencia = round(random.uniform(500, 3000), 2) # kW  
    velocidad_viento = round(random.uniform(3, 25), 2) # m/s  
    temperatura = round(random.uniform(10, 90), 2) # °C  
    timestamp = datetime.now(timezone.utc).isoformat() # Con zona horaria UTC  
  
    # Introducir error en un solo campo con probabilidad N  
    if random.random() < prob_error:  
        campo_erroneo = random.choice(["potencia_kw", "velocidad_viento", "temperatura_c"])  
        if campo_erroneo == "potencia_kw":  
            potencia = random.choice([-1000, "NaN", 99999])  
        elif campo_erroneo == "velocidad_viento":  
            velocidad_viento = random.choice([-5, "error", 100])  
        elif campo_erroneo == "temperatura_c":  
            temperatura = random.choice(["null", -273, 200])  
  
    return {  
        "id_generador": id_generador,  
        "potencia_kw": potencia,  
        "velocidad_viento": velocidad_viento,  
        "temperatura_c": temperatura,  
        "timestamp": timestamp  
    }
```



Simulación de los generadores

Para simular los molinos de viento tenemos un script que genera datos sintéticos con un 15% de error.

Para no tener que abrir 10 diferentes terminales tenemos un archivo "lanzar.bat" que ejecutara 10 veces el archivo "molino.py".

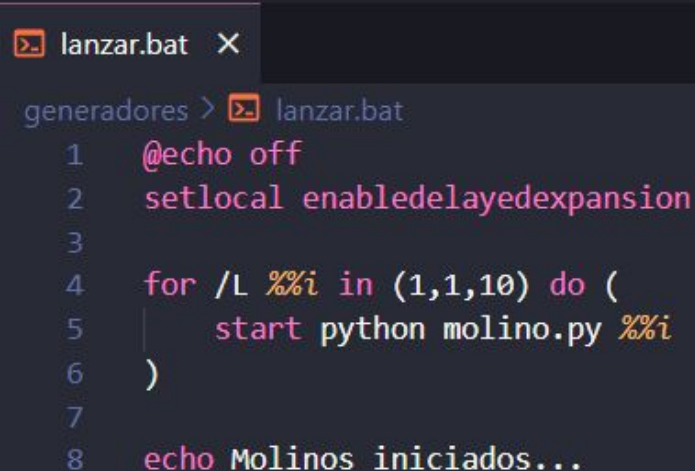
```
def enviar_datos(id_generador):  
    while True:  
        dato = generar_dato(id_generador)  
        print(json.dumps(dato, indent=2)) # Muestra los datos generados en consola  
  
        try:  
            # Enviar datos a FastAPI  
            response = requests.post(API_URL, json=dato, timeout=5)  
            print(f"Respuesta del servidor: {response.status_code} - {response.text}")  
        except requests.exceptions.RequestException as e:  
            print(f"[GEN-{id_generador}] Error de conexión: {e}")  
  
        time.sleep(2) # Generar datos cada 2 segundos
```



Simulación de los generadores

Para simular los molinos de viento tenemos un script que genera datos sintéticos con un 15% de error.

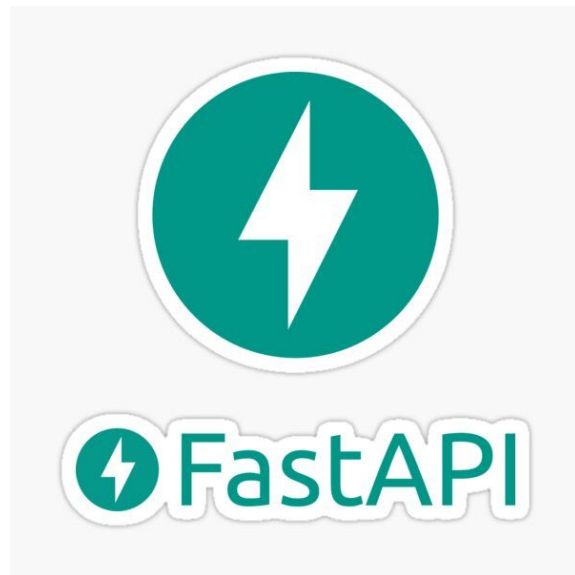
Para no tener que abrir 10 diferentes terminales tenemos un archivo "lanzar.bat" que ejecutara 10 veces el archivo "molino.py".



```
lanzar.bat X
generadores > lanzar.bat
1 @echo off
2 setlocal enabledelayedexpansion
3
4 for /L %%i in (1,1,10) do (
5     start python molino.py %%i
6 )
7
8 echo Molinos iniciados...
```

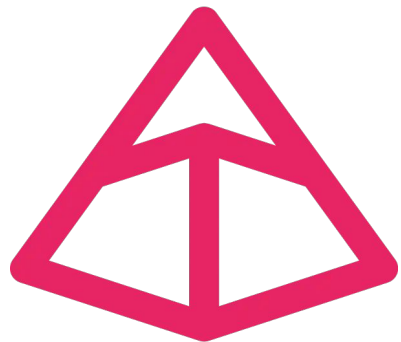
Concentramiento y validación de datos

```
52 datos_generadores: List[DatoGenerador] = []
53 datos_correctos: List[DatoGenerador] = []
54 @app.post("/ingresar_dato")
55 async def recibir_dato(dato: DatoGenerador):
56     try:
57         datos_generadores.append(dato)
58
59         try:
60             validado = DatoGenerador(**dato.dict())
61             datos_correctos.append(validado)
62         except ValueError:
63             pass
64
65         return {"mensaje": "Dato recibido correctamente", "total_datos": len(datos_generadores)}
66
67     except Exception as e:
68         error_msg = f"Error procesando el dato de ID {dato.id_generador}: {str(e)}"
69         logging.error(error_msg)
70         raise HTTPException(status_code=400, detail=error_msg)
71
72 @app.get("/datos")
73 async def obtener_datos():
74     return datos_generadores
75
76 @app.get("/datos_correctos")
77 async def obtener_datos_correctos():
78     return datos_correctos
79
```



Concentramiento y validación de datos

```
18 class DatoGenerador(BaseModel):
19     id generador: int
20     potencia_kw: float
21     velocidad_viento: float
22     temperatura_c: float
23     timestamp: str
24
25     @field_validator("potencia_kw")
26     def validar_potencia(cls, value, values):
27         id_gen = values.data.get("id_generador", "Desconocido")
28         if not isinstance(value, (int, float)) or value < 0 or value > 5000 or math.isnan(value):
29             error_msg = f"[GEN-{id_gen}] Potencia invalida: {value}, debe estar entre 0 y 5000 kw"
30             logging.error(error_msg)
31             raise ValueError(error_msg)
32         return value
33
34     @field_validator("velocidad_viento")
35     def validar_velocidad(cls, value, values):
36         id_gen = values.data.get("id_generador", "Desconocido")
37         if not isinstance(value, (int, float)) or value < 0 or value > 50 or math.isnan(value):
38             error_msg = f"[GEN-{id_gen}] Velocidad del viento invalida: {value}, debe estar entre 0 y 50 m/s"
39             logging.error(error_msg)
40             raise ValueError(error_msg)
41         return value
42
43     @field_validator("temperatura_c")
44     def validar_temperatura(cls, value, values):
45         id_gen = values.data.get("id_generador", "Desconocido")
46         if not isinstance(value, (int, float)) or value < -50 or value > 100 or math.isnan(value):
47             error_msg = f"[GEN-{id_gen}] Temperatura invalida: {value}, debe estar entre -50 C y 100 C"
48             logging.error(error_msg)
49             raise ValueError(error_msg)
50         return value
```



Pydantic



Agregaciones aplicadas

Como agregaciones hemos decidido añadir el promedio de potencia, velocidad y temperatura de los molinos.

```
@app.get("/promedio")
async def calcular_promedios():
    if not datos_correctos:
        raise HTTPException(status_code=404, detail="No hay datos válidos disponibles")

    promedio_potencia = sum(d.potencia_kw for d in datos_correctos) / len(datos_correctos)
    promedio_viento = sum(d.velocidad_viento for d in datos_correctos) / len(datos_correctos)
    promedio_temp = sum(d.temperatura_c for d in datos_correctos) / len(datos_correctos)

    return {
        "promedio_potencia_kw": promedio_potencia,
        "promedio_velocidad_viento": promedio_viento,
        "promedio_temperatura_c": promedio_temp,
        "total_datos_validos": len(datos_correctos)
    }
```




Problemas y Posibles futuras mejoras

- Al principio del reto, planteamos erróneamente los generadores inclumpliendo la parte de que los generadores deben ser programas individuales.
- A su vez, a la hora de desarrollar la persistencia del programa nos han surgido diversos problemas que nos han impedido su implementación exitosamente.
- ✓ Implementar la mejora de persistencia incluyendo una base de datos mongodb.
- ✓ Implementar seguridad en las comunicaciones entre los generadores y el concentrador.