

ART WITH DRONES

Práctica 2 Sistemas Distribuidos 23 - 24

Antonio Díaz-Parreño Lajara 50505021B
Eric Muñoz Rouillion 48768596V



Índice

Introducción.....	3
Guía de lanzamiento.....	3
Resultados.....	5
OpenWeather.....	8
Front.....	10
API_REST.....	12
Mecanismos de seguridad.....	14

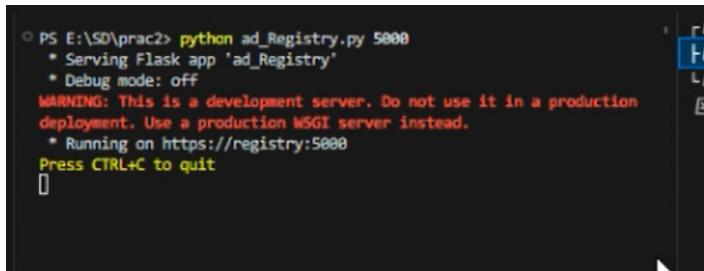


Introducción

El objetivo de esta práctica es, a partir de la práctica anterior, implementar la tecnología de comunicación basada en servicios REST además de otros principios de seguridad vistos en teoría y finalmente un frontend.

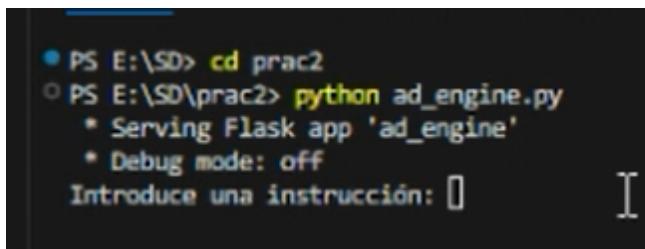
Guía de lanzamiento

Para lanzar la aplicación seguimos los siguientes pasos. Primero instalamos las dependencias necesarias.



```
PS E:\SD\prac2> python ad_Registry.py 5000
 * Serving Flask app 'ad_Registry'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production
deployment. Use a production WSGI server instead.
 * Running on https://registry:5000
Press CTRL+C to quit
```

Ejecutamos nuestro servidor AD_REGISTRY



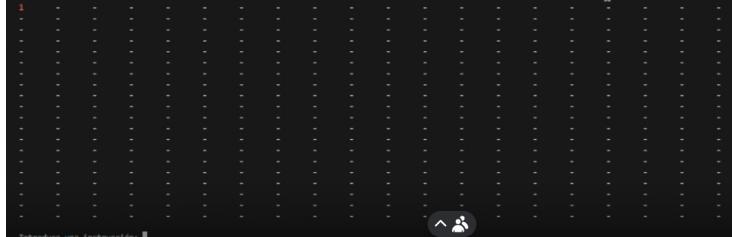
```
PS E:\SD> cd prac2
PS E:\SD\prac2> python ad_engine.py
 * Serving Flask app 'ad_engine'
 * Debug mode: off
Introduce una instrucción: [ ]
```

A continuación lanzamos nuestro AD_ENGINE el cual nos pedirá una instrucción, START para empezar el espectáculo, una vez lanzado engine los drones pueden ir registrándose. Una vez registrados nos pedirá introducir un archivo de figuras.

```

● PS E:\SD> cd prac2
○ PS E:\SD\prac2> python ad_engine.py
  * Serving Flask app 'ad_engine'
  * Debug mode: off
Introduce una instrucción: start
Introduce el fichero JSON: figuras\figuras2

```



Este es el resultado dentro de nuestra consola del engine

```

C:\Users\emoky\Desktop\Universidad\1º Cuatrimestre\SD\v9>python API_ENGINE.py 2500
  * Serving Flask app 'API_ENGINE'
  * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
  * Running on http://apiengine:2500
Press CTRL+C to quit

```

Podemos ir lanzando nuestro API_ENGINE en el puerto 2500 por ejemplo, aunque este paso se puede hacer cuando queramos ya que es independiente de nuestro programa y solo accede a la base de datos para recopilar información.

Lo mismo ocurre con el frontend:

```

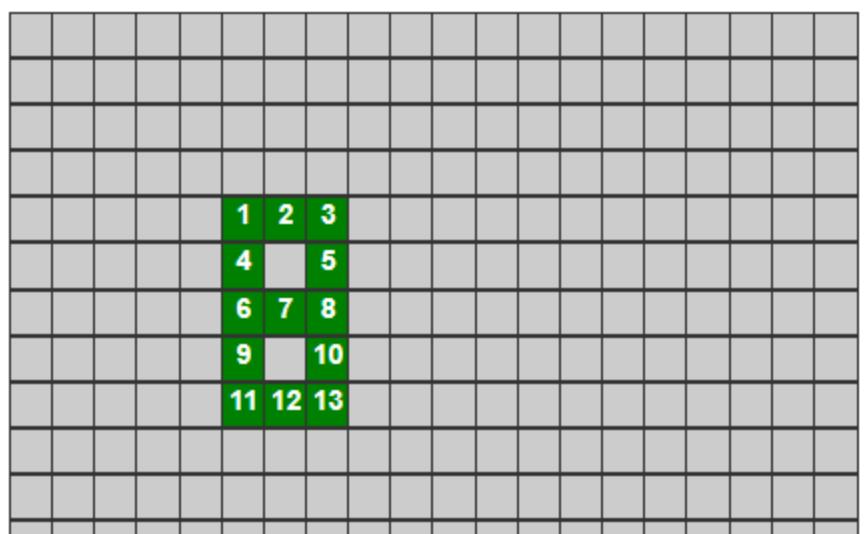
C:\Users\emoky\Desktop\Universidad\1º Cuatrimestre\SD\version11\prac2>python Front.py 1500 2500
  * Serving Flask app 'Front'
  * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
  * Running on http://front:1500
Press CTRL+C to quit
  * Restarting with stat
  * Debugger is active!
  * Debugger PIN: 256-112-373
127.0.0.1 - - [17/Dec/2023 23:32:08] "GET /socket.io/?EIO=4&transport=polling&t=Onvw74- HTTP/1.1" 200 -
Cliente conectado
127.0.0.1 - - [17/Dec/2023 23:32:08] "POST /socket.io/?EIO=4&transport=polling&t=Onvw7su&sid=9Ek-BUm1jc0kPI9HAAAA HTTP/1.1" 200 -
127.0.0.1 - - [17/Dec/2023 23:32:08] "GET /socket.io/?EIO=4&transport=polling&t=Onvw7sv&sid=9Ek-BUm1jc0kPI9HAAAA HTTP/1.1" 200 -
127.0.0.1 - - [17/Dec/2023 23:32:08] "GET /socket.io/?EIO=4&transport=polling&t=Onvw7t0&sid=9Ek-BUm1jc0kPI9HAAAA HTTP/1.1" 200 -

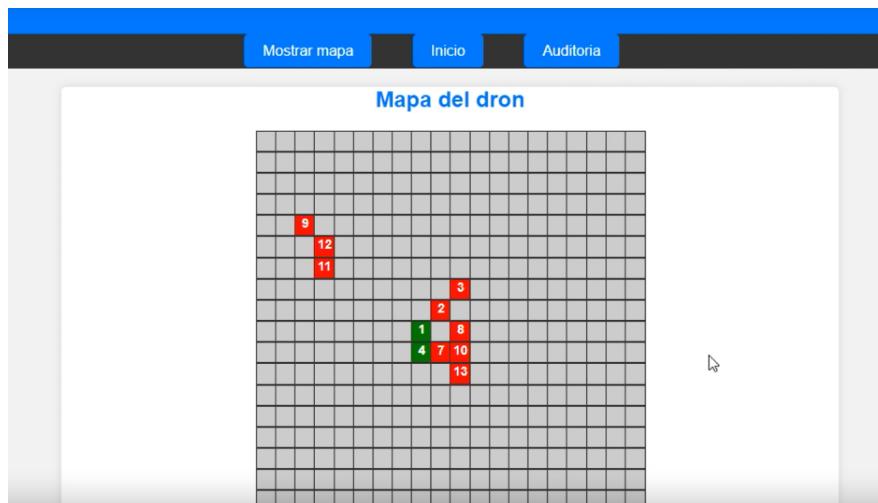
```

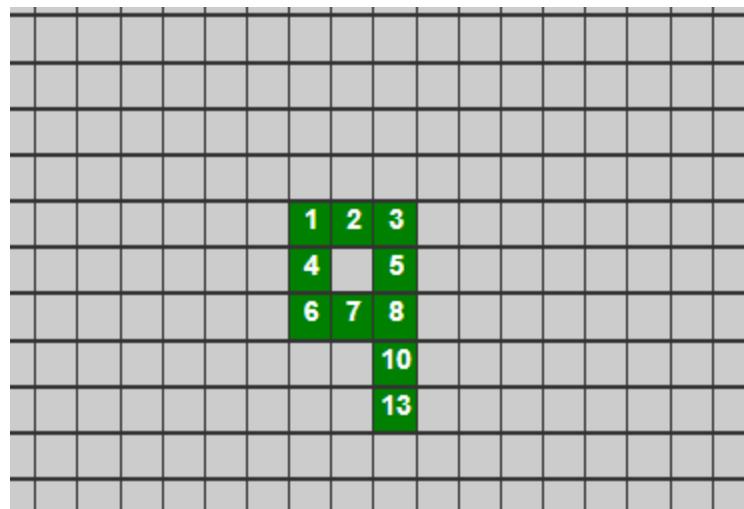
Aquí lo ejecutamos y enseguida empieza a consumir los datos por websockets del mapa

Resultados

Aquí mostramos un ejemplo de ejecución de un fichero JSON en nuestro frontend:





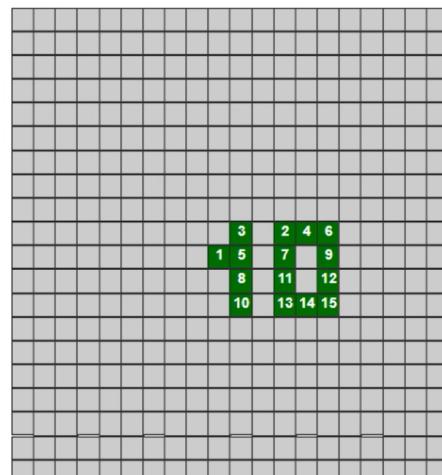


[Mostrar mapa](#)

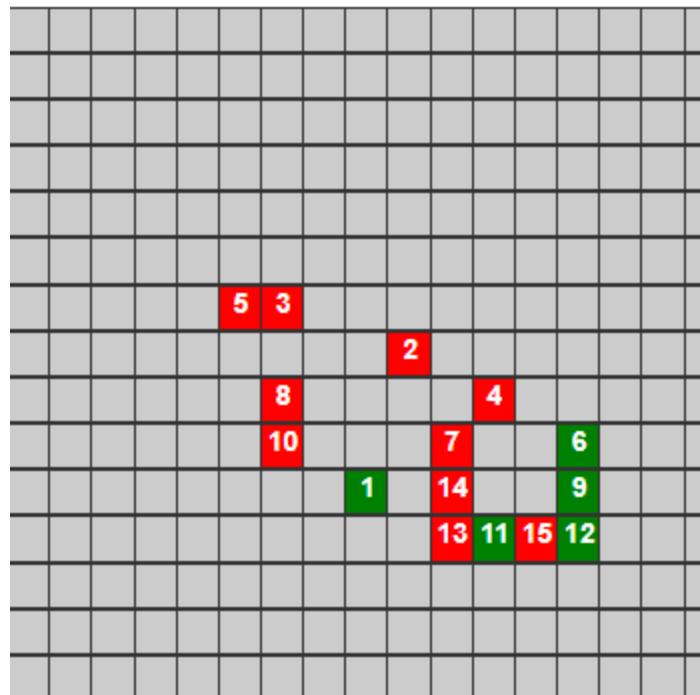
Inicio

Auditoria

Mapa del dron



Mapa del dron



Podemos ver en los pantallazos la secuencia de cómo se forman las figuras, y tras formar la última figura (el número 10) mostramos como todos los drones se van a base

OpenWeather

En cuanto a AD_WEATHER que era lo que teníamos en la práctica anterior, lo hemos sustituido completamente añadiendo en nuestro AD_ENGINE un módulo capaz de hacer consultas a la API OpenWeather. Para ello hemos requerido de una api_key que nos proporciona la propia API al registrarnos. Este módulo lee de un fichero de texto la ciudad y de otro fichero la api_key, en caso de cambiar el contenido de estos ficheros podemos obtener la temperatura de otra ciudad o hacer que no funcione la petición al cambiar la api_key.

```

133
134     class WeatherAPI:
135         def __init__(self, api_key):
136             self.api_key = api_key
137             self.base_url = "https://api.openweathermap.org/data/2.5/weather"
138
139         def get_weather(self, city):
140             url = f"{self.base_url}?q={city}&appid={self.api_key}"
141             response = requests.get(url)
142
143             if response.status_code == 200:
144                 data = response.json()
145                 return data
146             else:
147                 print(f"Error {response.status_code}: Unable to fetch weather data.")
148                 return None
149

```

```

724     def consulta_weather(self):
725         while True:
726             if self.start:
727                 if self.jsonIntroducido:
728                     self.city = read_file("ciudad.txt")
729                     self.api_key = read_file("key.txt")
730                     try:
731                         weather_api = WeatherAPI(api_key)
732                         weather_data = weather_api.get_weather(self.city)
733                         if weather_data:
734                             temperature_kelvin = weather_data['main']['temp']
735                             self.temperatura = kelvinCelsius(temperature_kelvin)
736                         except:
737                             self.temperatura = 50
738
739                         if self.temperatura < 0 or self.temperatura == 100 or self.stop:
740                             if not self.stop:
741                                 print("CONDICIONES CLIMÁTICAS ADVERSAS. ESPECTÁCULO FINALIZADO.")
742                                 fecha, hora = obtenerFecha()
743                                 evento = AD_EVENT(fecha,hora,HOST,"CONDICIONES CLIMÁTICAS ADVERSAS", f"Se ha finalizado el espectáculo por temperatura")
744                                 self.semaforo.adquiere()
745                                 escribir(evento)
746                                 self.semaforo.libera()
747                         if self.stop:
748                             self.temp = "S"
749                             mapastr = self.mapa.mapToString()
750                             mensaje = self.temp + mapastr
751                             mensaje = mensaje.encode(FORMAT)
752                             self.producirMapa.send(self.datos_kafka[1], value=mensaje)
753                         else:
754                             self.terminado = True
755                             self.temp = "W"
756                             mapastr = self.mapa.mapToString()
757                             mensaje = self.temp + mapastr
758                             mensaje = mensaje.encode(FORMAT)
759                             self.producirMapa.send(self.datos_kafka[1], value=mensaje)
760             time.sleep(5)

```

Al igual que en la práctica anterior consultamos el clima cada 5 segundos

```

95 @app.route('/mapa')
96 def mapa():
97
98     ciudad = read_city_name()
99     key= read_api_key()
100
101    if ciudad is None:
102        ciudad = "Ciudad Desconocida" # Valor predeterminado si no se puede leer la ciudad
103
104    try:
105        weather_url = f"http://api.openweathermap.org/data/2.5/weather?q={ciudad}&appid={key}&units=metric"
106        response = requests.get(weather_url)
107
108        if response.status_code == 200:
109            weather_data = response.json()
110            temp = weather_data['main']['temp']
111        else:
112            temp = "N/A"
113
114    except Exception as e:
115        print(f"Error: {e}")
116        temp = "No se ha podido leer la temperatura"
117
118    try:
119        e=None
120        mapa_global= coger_mapa_inicial()
121
122    except Exception as e:
123        print(f"Error: {e}")
124
125    return render_template('mapa.html', mapa_del_dron=mapa_global, temperatura=temp, ciudad=ciudad, drone_error_message=e)
126

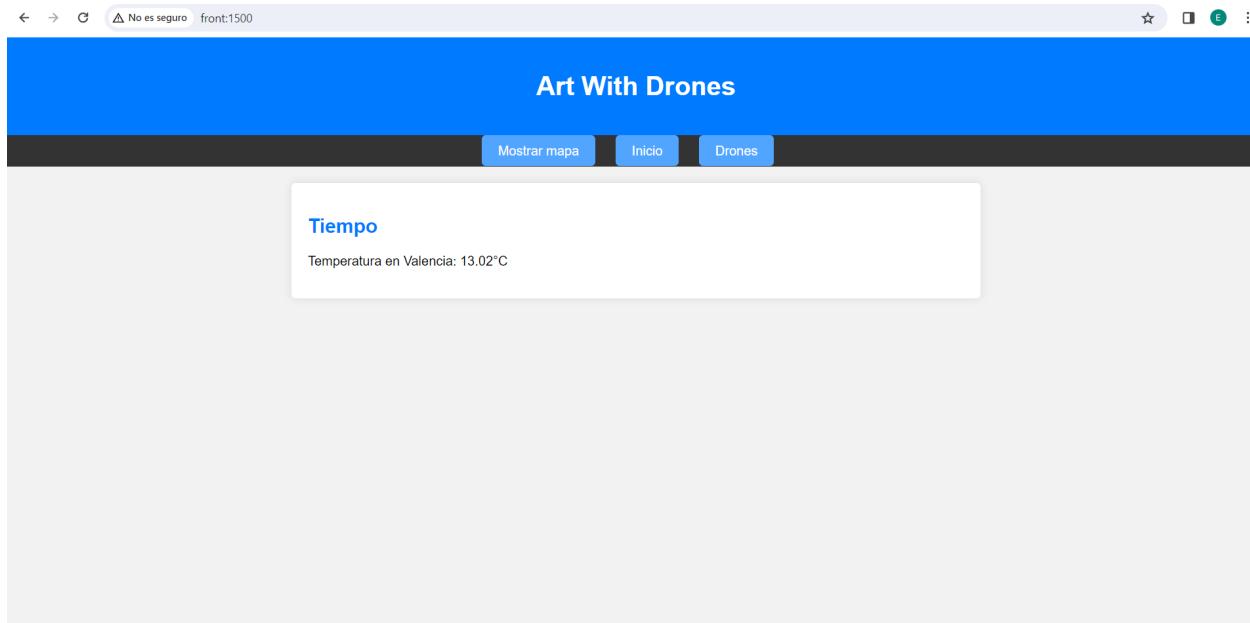
```

Aquí se muestra un extracto de código de front.py en el que hacemos la misma llamada a openweather leyendo desde los mismos ficheros de datos que AD_ENGINE, por lo tanto en nuestra página web se mostrar la misma temperatura que estamos usando en la aplicación entera

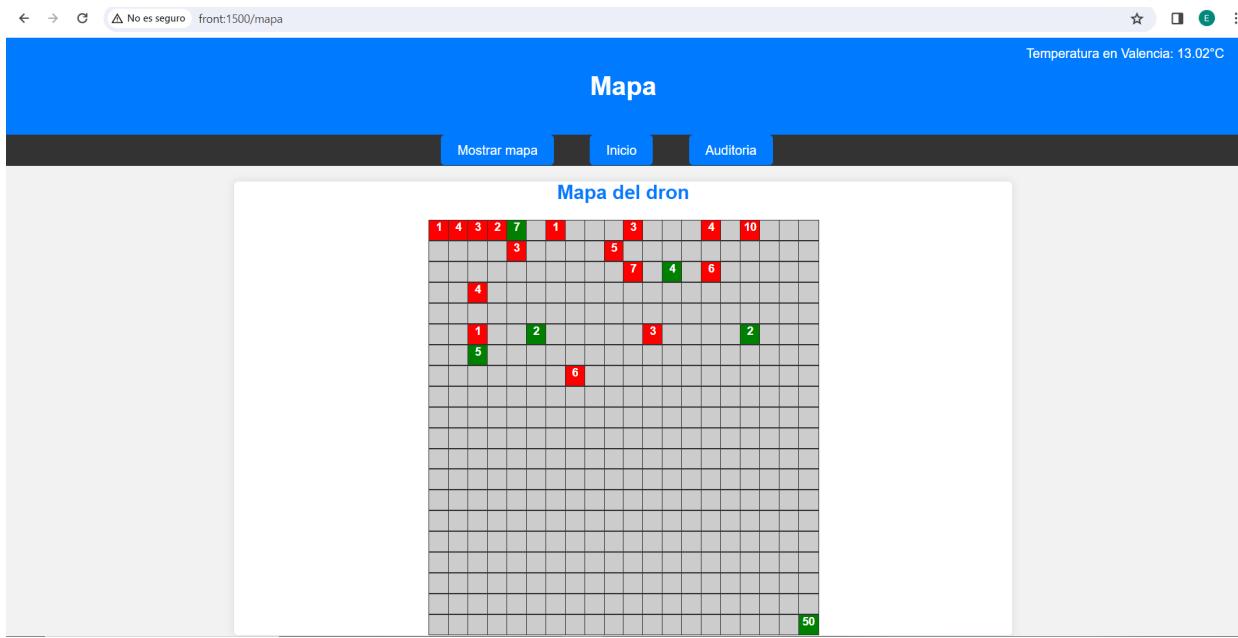
Front

El frontend que se ha implementado es una aplicación con la librería de flask, se comunica con API ENGINE para obtener los datos necesarios que quiero mostrar por pantalla al usuario. Permite abrir más de un frontend a la vez para que más de 1 usuario pueda consumirlo.

En caso de fallo del API ENGINE, no se podrá mostrar la página de mapa pero se podrá seguir consultando la página principal:



Y aquí una muestra de cómo se vería un mapa con drones colocados aleatoriamente:



Como podemos ver en el pantallazo, las casillas de los drones se pintan en función de si están en su posición final (**verde**) o están moviéndose todavía (**rojo**).

Esto se ha hecho teniendo en cuenta el valor del dron en cada posición de la matriz del mapa, por ejemplo: las 4 primeras casillas que vemos en la captura vemos 1 4 3 2 en rojo, en la base de datos estaría guardado de la siguiente forma:

```

1 _id: ObjectId('655125c95d76485251f68c2f')
2 ID: 1
3 Alias: "Drone_1"
4 POS: "1.5,4"
5 Token: "ivpuezqvs"
6 Estado: "N"
7 Mapa: "R1 R4 R3 R2 F7 - R1 - - - R3 - - - R4 - R10 - - - "
      - - - R3 - - - R5 - - - -
      - - - - - R7 - F4 - R6 - - - -
      - R4 - - - - -
      - - - - - - - -
      - - R1 - - F2 - - - - R3 - - - F2 - - -
      - - F5 - - - - - - - -
      - - - - - R6 - - - - -
      - - - - - - - - - - -

```

Aquí vemos la información del dron 1 qué es el que usamos para leer el mapa ya que todos los drones contienen el mapa actualizado, simplemente usamos el mapa del dron número 1.

La parte del mapa en el frontend se actualiza constantemente, es decir que en cuanto se modifique la base de datos tardará muy poco en plasmar la información en pantalla sin necesidad de recargar la página.

API_REST

1. API ENGINE

En cuanto a la API ENGINE, según el enunciado únicamente sirve como enlace entre el servidor frontend y nuestra base de datos, he añadido algunas peticiones GET más en caso de querer usarlas más adelante para tener un frontend con más información para el usuario, tal como puede ser la auditoría, que es un medio de ver por pantalla los eventos que van ocurriendo durante la ejecución del programa.

Esta API también está securizada como las demás.

```

63  @app.route("/mapa", methods=['GET'])
64  def get_map():
65      try:
66          collection = obtenerBaseDatos("REGISTRY")
67          registro = collection.find({"ID": 1}, {'_id': False, 'Mapa': True})
68          busqueda = list(registro)
69          response = {
70              'data': busqueda,
71              'error': False,
72              'message': 'Objetos cogidos satisfactoriamente'
73          }
74      return jsonify(response), 200
75
76  except Exception as e:
77      print(e)
78      return jsonify(error='Error del servidor'), 500
79
80

```

Aquí vemos la función GET para obtener el mapa dentro del dron con ID=1, la ruta mapa es a la que se llama desde nuestro frontend

2. API expuesta por Registry

Se ha implementado dentro de AD_REGISTRY.py un API, securizada mediante https, que permite las siguientes funcionalidades:

- Registrar drones en la base de datos.
- Proporcionar tokens a los drones que quieran autenticarse en el espectáculo, estos tokens tendrán una duración de 20 segundos ya que se borrarán de la base de datos una vez haya cumplido el tiempo especificado, dejando la base de datos limpia de tokens.
- Obtener los datos de todos los drones registrados en la base de datos.

3. API expuesta por Engine

Se ha implementado dentro de AD_ENGINE.py un API, securizada mediante https, que permite las siguientes funciones:

- Autenticar drones comprobando los drones comparando el token de la base de datos con el token que nos proporciona el dron a través del método POST del API REST que expone Engine. Cuando los drones se autentican, se realiza a su vez el intercambio de claves entre Engine y Dron.
- Permitir obtener los datos de los drones registrados en la base de datos mediante el método GET.

Mecanismos de seguridad

1. Autenticación segura

Para la autenticación segura entre Registry y drones hemos cifrado el canal de comunicación, en esta parte API_REST, con https, por lo que la información que se comparte por el canal está cifrada. Respecto a los tokens, como he dicho anteriormente, tienen una validez de 20 segundos y, una vez ha expirado este tiempo, se elimina de la base de datos.

2. Auditoría de eventos

Para realizar esta auditoría se ha facilitado una clase que representa a un evento y que va añadiendo al fichero .csv en cada línea, esta clase dispone de cinco atributos: 'fecha', 'hora', 'ipEvento', 'acción' y 'descripción'.

```

149
150     def obtenerFecha():
151         fecha_hora = datetime.datetime.now()
152         fecha = fecha_hora.strftime("%Y-%m-%d")
153         hora = fecha_hora.strftime("%H:%M:%S")
154         return fecha,hora
155
156     claveEngine = AES()
157     claveMapa = AES()
158     claveRSA = RSA()
159
160     def escribir(evento : AD_EVENT):
161         with open('auditoria_engine.csv', 'a', newline='') as csvfile:
162             spamwriter = csv.writer(csvfile, delimiter='\t')
163             spamwriter.writerow([evento.fecha, evento.hora, evento.ipEvento, evento.accion, evento.descripcion])
164
165     def vaciar_csv(archivo_csv):
166         # Abre el archivo CSV en modo escritura ('w') para vaciar su contenido
167         with open(archivo_csv, 'w', newline='') as csvfile:
168             # Crea un objeto escritor de CSV
169             csv_writer = csv.writer(csvfile)
170
171             # Escribe una fila vacía para vaciar el contenido
172             csv_writer.writerow([])
173

```

La función escribir coge un objeto evento de tipo AD_EVENT como entrada y escribe sus atributos en un archivo CSV llamado 'auditoria_engine.csv'. Los atributos que se escriben en el archivo CSV son: evento.fecha, evento.hora, evento.ipEvento, evento.accion, evento.descripcion.

La función vaciar toma el nombre de un archivo CSV como entrada (archivo_csv) y lo abre en modo escritura ('w') con la opción newline="" para asegurarse de que no se añaden líneas en blanco adicionales. Luego, crea un objeto escritor de CSV y escribe una fila vacía en el archivo. Esto hace que se vacíe el contenido del archivo CSV especificado.

```
851
852     def registraEventos(self,evento):
853         MONGO_URI = DATOS_DATABASE
854         cliente = MongoClient(MONGO_URI)
855         database = cliente["dbREGISTRY"]
856         collection = database["AUDITORIA"]
857
858         insertar = {
859             "Fecha" : evento.fecha,
860             "Hora" : evento.hora,
861             "IP" : evento.ipEvento,
862             "Accion" : evento.accion,
863             "Descripcion" : evento.descripcion
864         }
865         collection.insert_one(insertar)
866
867     def limpiaAuditoria(self):
868         MONGO_URI = DATOS_DATABASE
869         cliente = MongoClient(MONGO_URI)
870         database = cliente["dbREGISTRY"]
871         collection = database[ "AUDITORIA" ]
872         collection.drop()
873
874
```

3. Cifrado de datos entre Engine y Drones

El cifrado entre Engine y Drone se ha realizado de forma simétrica para la comunicación por Kafka y asimétrica para la comunicación por API_REST.

- Cifrado simétrico: El algoritmo que hemos elegido ha sido AES, por su sencilla implementación utilizando la librería cryptography de python.
- Cifrado asimétrico: Se ha realizado la securización por https.