Proyecto Fin de Ciclo - Flight Tracker

Ana Sersic

Diciembre 2020

Contents

1 Introducción

El objetivo de este proyecto es desarrollar una aplicación que muestre los vuelos activos dentro del espacio aéreo español en el momento en que el usuario se conecta, pudiendo ver los detalles de los mismos, así como filtrar los vuelos entrantes y salientes por aeropuerto.

2 Tecnologías escogidas y justificación

La aplicación está desarrollada en Python, utilizando como framework Flask. Esta elección se debe principalmente a cuatro razones.

1. Su sencillez. Flask es lo que se conoce como microframework. Es decir, en su instalación nos proporciona solo las herramientas imprescindibles para el desarrollo de una aplicación, pudiendo ir añadiendo funcionalidades según las vayamos necesitando y evitando así llenar la aplicación de elementos innecesarios.
2. Este framework está diseñado para trabajar con el patrón MVC, aunque no es obligatorio, por lo que podemos separar fácilmente la lógica de la visualización.
3. Flask nos proporciona tanto un servidor de desarrollo como un depurador.
4. Cuenta con una documentación extensa y detallada en http://flask.palletsprojects.com

Dado que la idea es que la aplicación muestre visualmente el tráfico aéreo se usará la librería Folium, diseñada para visualizar datos de tipo geoespacial.

3 Diseño

La aplicación está dividida en dos secciones

1. Pantalla de vuelos activos.

   Aquí se genera un mapa con el que el usuario puede visualizar la localización de los vuelos que en ese momento se encuentran atravesando el espacio aéreo español.

2. Pantalla de aeropuertos. En esta sección también se genera un mapa que, en este caso, muestra la localización de los aeropuertos españoles.

   El usuario tiene la opción de elegir un aeropuerto en particular. Una vez seleccionado, se crea un mapa que contiene tres capas.

   (a) Capa 1. Se visualizan todos los vuelos con destino u origen en dicho aeropuerto y que han estado activos durante la última hora.

   (b) Capa 2. Llegadas. El usuario puede filtrar solo los vuelos que se dirigen al aeropuerto.

   (c) Capa 3. Salidas. El usuario puede filtrar solos los vuelos que tienen como origen dicho aeropuerto.

4 Estructura de la aplicación

A continuación se muestra cómo se ha estructurado la aplicación. Primero las tres partes comunes a toda aplicación desarrollada en Flask (routing, templates, static), para seguir con las específicas de este proyecto, las cuales han sido divididas acorde a su funcionalidad (utils y maps).

4.1 Routing

El decorador route() asocia URL y métodos. La aplicación tiene dos URL.

1. / : mostrará los vuelos activos
2. /airports: mostrará los aeropuertos españoles para ver su tráfico aéreo específico

4.2 Templates

Flask utiliza Jinja2 como motor de renderizado de plantillas. Podemos definir plantillas básicas que estructuren las vistas de nuestra aplicación y de las cuales heredarán otras en las que definiremos su comportamiento específico.

Para renderizar las plantillas se utiliza el método render_template() al cual le pasamos el nombre de la plantilla y las variables que esta necesite en forma de parámetros clave-valor. Flask busca estas plantillas en el directorio templates.

Las vistas de esta aplicación están estructuradas en cinco bloques básicos

1. Barra de navegación
2. Formulario de selección
3. Mapa que contendrá una representación visual de los vuelos
4. Modal para mostrar la información detallada de los vuelos

5. Scripts

Estos bloques se han incluido en la plantilla básica index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="./../static/css/style.css">
    <meta http-equiv="Content-Type" content="text/html;charset=UTF-8">
</head>
<body>
    <div class="nav">
        <ul>
            <li><a href="{{ url_for('index') }}">Flights</a></li>
            <li><a href="{{ url_for('national') }}">Airports</a></li>
        </ul>
    </div>
    <div class="content">
        {% block form %}{% endblock %}
        {% block map %}{% endblock %}
    </div>
    {% block modal %}{% endblock %}
    {% block script %}{% endblock %}
</body>
</html>
```

templates/index.html

```
{% extends "index.html" %}

{% block form %}
    <form action="/national" method="POST">
        <select name="airport_name" id="airport_name" onchange="update()" >
            <option name="airport_name_op" value="all">All airports</option>
            {% for airport in airports %}
            <option name="airport_name_op" value={{ airport.attributes.OBJECTID }}>
            {{ airport.attributes.Texto }} </option>
        {% endfor %}
        </select>
    </form>
{% endblock %}
```

```
14
15 {% block map %}
16     {% if source %}
17         <iframe id="iframe" class="iframe-airports" scrolling="no" srcdoc="{{ source }}">
18         </iframe>
19     {% endif %}
20 {% endblock %}
21
22 {% block modal %}
23     <div id="modal-background" class="modal-background" onclick="close()"></div>
24     <div id="modal" class="modal">
25         <button class="close" onclick="close()">X</button>
26     </div>
27 {% endblock %}
```

Definición específica de bloques en templates/national.html

4.3 Static

Dentro de static se incluye un archivo json que incluye un listado con los datos necesarios para obtener la información del tráfico aéreo de los distintos aeropuertos españoles, tales como sus diferentes códigos identificativos y su localización geográfica.

4.4 Utils

Aquí se definen todas las funciones relacionadas con la obtención y procesamiento de datos.

Por ejemplo, para obtener los vuelos activos, se hará uso de la siguiente función:

```
1 def get_active_flights():
2     flights = []
3     response = requests.get(url).json()
4     for flight in response:
5         if flight["flight"]["iataNumber"] or flight["flight"]["icaoNumber"]:
6             flights.append(flight)
7     return flights
```

Los vuelos activos serán filtrados según tengan o no al menos una identificación válida.

4.5 Maps

En Maps incorporan las funciones encargadas de la creación de mapas y marcadores.

```
def create_base_map():
    if os.path.exists('./app/templates/map.html'):
        os.remove('./app/templates/map.html')
    response = utils.get_active_flights()

    map = folium.Map(location=[40.463667, -3.74922], zoom_start=6)
    tooltip = "See flight information"

    logoIcon = folium.features.CustomIcon('./app/static/img/paperplane.png',
    icon_size=(50, 50))

    for flight in response:
        flight_id = flight["flight"]["iataNumber"]
        marker=folium.Marker(location=[flight["geography"]["latitude"],
        flight["geography"]["longitude"]], popup=f'<span>Flight: {flight_id}</span>
        <button id="flight_button" class="show" value="{flight_id}" onclick="func()">
        Show flight details</button>', tooltip=tooltip, icon=folium.features.CustomIcon(
        './app/static/img/paperplane.png', icon_size=(30, 30))).add_to(map)

    map.save('./app/templates/map.html')
```

Debido a que Folium no cuenta con la funcionalidad necesaria para actualizar periódicamente el mapa de forma dinámica los mapas generados serán guardados y sobrescritos al solicitar nueva información.