15 DE DICIEMBRE DE 2023

FLIGHT RADAR

PROYECTO FINAL

GRUPO 1
UIE

Contenido

Introducción

Nuestro proyecto trata de hacer una base de datos que se **actualiza en tiempo real** sobre viajes que realizan aviones de diferentes aerolíneas a un destino, el código de avión, el tiempo... Los usuarios podrán entrar en la página web en la que podrán pedir información de la base de datos.

Resumen

Como muchas aplicaciones de información sobre vuelos tienen suscripciones de pago, nuestro objetivo es crear una herramienta que permita la visualización de los vuelos en España de un modo interactivo y **sin** tener que **pagar.**

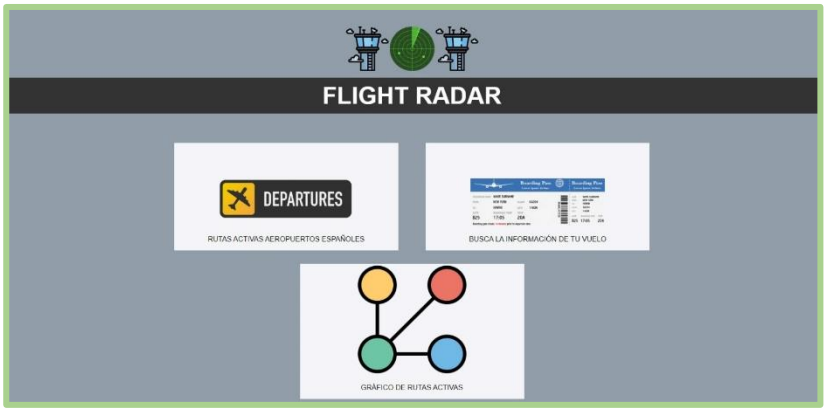
Nuestra base de datos relaciona en tiempo real la información de vuelos y los aeropuertos más importantes de España.

Para hacer esto, usamos cuatro lenguajes de programación, **Python, SQL, HTML y CSS.** Con estos, se obtiene la información de la base de datos que está guardada en un archivo JSON con información como identificadores de rutas, vuelos, destinos, orígenes, aeropuertos y sus nombres y las aerolíneas que operan dentro de estas.

Además, mediante la mezcla de Python y SQL se automatiza tanto la creación de tabla como entrada de datos en la Base, consiguiendo de esta forma una gran base en la que se añaden nuevos datos cada **5 segundos.**

Para facilitar ver la información, utilizamos html para hacer una página web con un menú con tres opciones en la que se puede ver las rutas activas de aeropuertos españoles, buscar la información de un vuelo en concreto, y ver un gráfico de las rutas activas. También hay un colapsable con las tres opciones para poder cambiar lo que se hace sin tener que volver al menú principal.

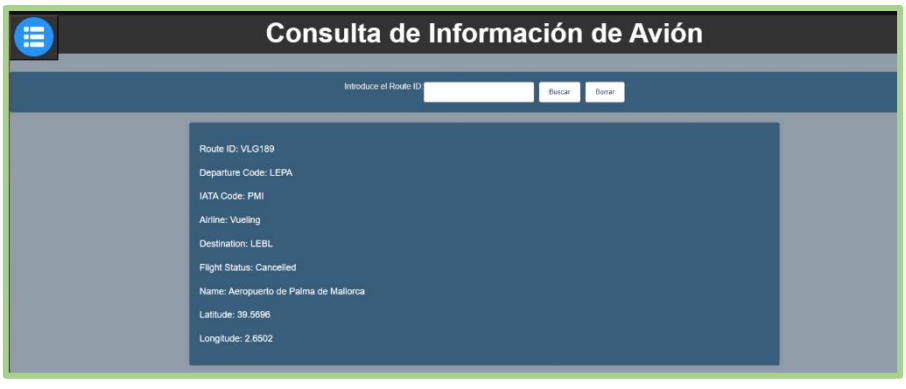
Página Web



Esta sería la interfaz de la página web que se llama Flight radar, desde el inicio se puede seleccionar **tres opciones**, estas son: (Información de vuelo, Información del aeropuerto y gráfico de rutas activas).

En Información del aeropuerto el usuario tiene que poner el nombre del aeropuerto en el que está interesado obtener información, en esta sección muestra los vuelos que salen desde ese aeropuerto y también las líneas.

Por ejemplo, en esta imagen Se puede ver 4 rutas que salen del aeropuerto, su código y sus respectivas aerolíneas.



En la sección de información del vuelo se puede ver información más detallada sobre el vuelo que en la sección del aeropuerto, por ejemplo, muestra el código de la ruta, la aerolínea, el estado del vuelo… También si está retrasado, cancelado o en tiempo, el aeropuerto de destino, altitud y latitud a la que se encuentra el avión.

CÓDIGO

Hicimos **16 ficheros** para la realización de este proyecto, empezando por el fichero **"CreacionDatos.py"**

```python
import random
import time
import mysql.connector


# Conexion a la base de datos
def create_database_connection():
    conn = mysql.connector.connect(
        host="localhost",
        user="root",
        password="root",
        database="aviones"
    )
    c = conn.cursor()

    return conn, c
```

Primero **importamos** los módulos estándar de Python y las bibliotecas necesarias. Definimos la función **"create_database"** para establecer una conexión con una base de datos MySQL, usando el módulo **connect** y especificando los detalles necesarios para conectarse, que son "host", "user", "password" y "database".

Empezamos a **definir** las **tablas** que se van a crear en la base de datos, todas ellas siguiendo una estructura similar excepto para la tabla **"flights"**, la cual contiene las **claves primarias del resto de tablas**. Primero se especifica que se cree la tabla si no existe previamente (en esta caso la claves primarias del resto de tablas tabla "Departures") y se va aumentando en 1 el id de la clave primaria mediante la propiedad auto_increment propia de MySQL. Después se añade la variable departure_code, que es el código del aeropuerto de salida de dicho avión.

```
def create_departure_table(c):
    c.execute('''
        CREATE TABLE IF NOT EXISTS Departures (
            id_departure INTEGER AUTO_INCREMENT PRIMARY KEY,
            departure_code VARCHAR(255)
        )
    ''')
```

Luego se crea la tabla **"flights"**, que contiene la clave primaria (id) del resto de tablas, facilitando hacer las uniones necesarias más adelante. Por tanto, estas claves tienen que ser especificadas como claves foráneas dentro de flights, para que luego no haya errores de **relaciones entre las tablas y las claves.** Para introducirlas correctamente, introducimos cada uno de los id (con valor predeterminado 0, ya que los datos los extrae de un diccionario de python creado a partir del archivo json) y posteriormente establecemos las referencias como se puede observar en la imagen.

Finalmente, esta tabla también tiene una clave primaria llamada **id_flight.**

```
def create_flights_table(c):
    c.execute('''
        CREATE TABLE IF NOT EXISTS Flights (
            id_flight INTEGER AUTO_INCREMENT PRIMARY KEY,
            id_departure INTEGER DEFAULT NULL,
            id_iata INTEGER DEFAULT NULL,
            id_name INTEGER DEFAULT NULL,
            id_air INTEGER DEFAULT NULL,
            id_destination INTEGER DEFAULT NULL,
            id_status INTEGER DEFAULT NULL,
            id_route INTEGER DEFAULT NULL,
            id_longitude INTEGER DEFAULT NULL,
            id_latitude INTEGER DEFAULT NULL,
            FOREIGN KEY (id_departure) REFERENCES Departures (id_departure),
            FOREIGN KEY (id_iata) REFERENCES IataCodes (id_iata),
            FOREIGN KEY (id_name) REFERENCES names (id_name),
            FOREIGN KEY (id_air) REFERENCES Airlines (id_air),
            FOREIGN KEY (id_destination) REFERENCES Destinations (id_destination),
            FOREIGN KEY (id_status) REFERENCES FlightStatuses (id_status),
            FOREIGN KEY (id_route) REFERENCES routeIds (id_route),
            FOREIGN KEY (id_longitude) REFERENCES Longitude (id_longitude),
            FOREIGN KEY (id_latitude) REFERENCES Latitude (id_latitude)

        )
    ''')
```

Esta función genera un avión con los datos que se le dan. Para **randomAirport,** se elige un aeropuerto aleatorio de una lista de aeropuertos generada previamente en un diccionario route , que es el archivo **Json** "traducido" (el término sería "dump") a Python . Posteriormente se sigue el mismo esquema para Route y Status.

Luego se crea un diccionario “plane” que contiene detalles sobre un vuelo, como el aeropuerto de salida, el código IATA, el nombre, la longitud, la latitud, la aerolínea, el destino, el ID de la ruta y el estado del vuelo. Los valores para estos campos se toman del diccionario creado randomAirport. Finalmente, la función devuelve el diccionario “plane”, que representa un vuelo aleatorio específico de un aeropuerto.

```
def get_random_flight(flightStatus, prop, route):
    randomAirport = random.choice(route['airports'])
    randomRoute = random.choice(randomAirport['routes'])
    randomStatus = random.choices(flightStatus, weights=prop, k=1)[0]

    plane = {
        "Departure": randomAirport['departure'],
        "iataCode": randomAirport['iataCode'],
        "name": randomAirport['name'],
        "longitude": randomAirport['longitude'],
        "latitude": randomAirport['latitude'],
        "Airline": randomRoute['airline'],
        "Destination": randomRoute['destination'],
        "Id": randomRoute['routeId'],
        "flightStatus": randomStatus

    }

    return plane
```

Con esta función se introduce en las tablas correspondientes los datos del vuelo a partir de cada segmento correspondiente del diccionario “plane”. En este caso, se inserta en la tabla **“Departures”** el valor de **“departure_code**” obtenido del diccionario “plane” definido anteriormente.

```
def insert_flight_data(plane, c, conn):
    time.sleep(5)

    try:
        c.execute("INSERT INTO Departures (departure_code) VALUES (%s)",
                  (plane['Departure'],))
        id_departure = c.lastrowid

        c.execute("INSERT INTO IataCodes (iataCode) VALUES (%s)",
                  (plane['iataCode'],))
        id_iata = c.lastrowid
```

En el caso de la tabla “flights”, al tratarse de claves foráneas, hay que introducir los id de cada uno de forma diferente al resto de tablas, por tratarse de claves foráneas, avisando sobre si se produjo algún error en este proceso.

```
    c.execute("INSERT INTO Flights (id_departure, id_iata, id_name, id_air, i
              (id_departure, id_iata, id_name, id_air, id_destination, id_sta

except mysql.connector.Error as e:
    print(
        f"Se produjo un error al insertar datos en la base de datos: {e}")
```

Luego creamos el fichero "**main.py"**, que es el fichero principal.

```
import json
import CreacionDatos as cd
i = 0

conn, c = cd.create_database_connection()

def tableCreation():
    cd.create_departure_table(c)
    cd.create_iata_table(c)
    cd.create_names_table(c)
    cd.create_airlines_table(c)
    cd.create_destinations_table(c)
    cd.create_flightstatus_table(c)
    cd.create_longitude_table(c)
    cd.create_latitude_table(c)
    cd.create_routeids_table(c)
    cd.create_flights_table(c)
```

El main es el archivo donde usamos las funciones de creaciondatos.py y el json, por lo que se lo metemos al principio, conectamos a la base de datos y hacemos una nueva funcion donde se crean todas las tablas.

Se crea la variable "i" para hacer 100 iteraciones y se solicita conectarse a la base de datos y crear las tablas. Después, se abre el archivo Json y se carga en una variable route, convirtiendo el fichero Json en un fichero de Python creando el diccionario. Luego se definen unos estados del vuelo con las proporciones correspondientes (por ejemplo hay un 80% de probabilidades que el vuelo esté en hora).Por último, se escoge un vuelo aleatorio, se insertan los datos en ella y se suma uno al contador.

```python
def main(route, i):

    for i in range (100):
        conn, c = cd.create_database_connection()
        tableCreation()

        with open('routes.json') as routes_file:
            route = json.load(routes_file)

        flightStatus = ['Scheduled', 'Cancelled', 'Delayed']
        prop = [0.80, 0.15, 0.05]


        plane = cd.get_random_flight(flightStatus, prop, route)
        print(plane)
        cd.insert_flight_data(plane, c, conn)
        i = i + 1

# Cargar los datos del archivo JSON
with open('routes.json') as routes_file:
    route = json.load(routes_file)

# Llamar a main() con los datos cargados
main(route, i)
```

Por último, se creó una web con Flask, un módulo de Python que permite hacer páginas y aplicaciones pequeñas.

Importamos las librerías necesarias para la creación de la web. Luego se le especifica que se va a usar un formato Sql con una base de datos. Para traducir el Sql de la base a flask y que la aplicación pueda interpretar tanto las tablas como la información de la base de datos, se definen las clases como las tablas correspondientes del fichero Creaci

```python
from flask import Flask, render_template, request, redirect, url_for, jsonify
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy.orm import joinedload
# import pymysql

# pymysql.install_as_MySQLdb()
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql://root:root@localhost:3306/aviones'
db = SQLAlchemy(app)


class Departures(db.Model):
    __tablename__ = 'Departures'
    id_departure = db.Column(db.Integer, primary_key=True)
    departure_code = db.Column(db.String(255))


class IataCodes(db.Model):
    __tablename__ = 'IataCodes'
    id_iata = db.Column(db.Integer, primary_key=True)
    iataCode = db.Column(db.String(255))
```

Para la página de inicio, se crea la página de inicio cargando la plantilla del fichero “menu.html” después se define la sección en la web llamada “Buscar avión” para mostrar la selección de vuelos, el usuario va a tener que meter el id del avión sobre el que desea ver la información, mientras que la función borrar avión redirige al usuario a la ruta correspondiente para poder realizar nuevas búsquedas.

La función borrar avión debería ser más compleja, usando conceptos como las sesiones del usuario para borrar sus consultas. En realidad, con borrar avion no se está borrando

la consulta del usuario, si no que se le está devolviendo a la página anterior pero sin borrar la consulta, como se podría observar en el registro de la página web.

No obstante, esto implicaría usar librerías como sesión y adentrarse en un backend mucho más profundo, el cual coincidimos en que no era necesario para un proyecto enfocado en la base de datos y no en desarrollo web.

```python
@app.route('/', methods=['GET'])
def menu():
    return render_template('menu.html')


@app.route('/buscar_avion', methods=['GET'])
def buscar_avion():
    route_id = request.args.get('route_id')
    error = None
    vuelo = None
    if route_id is not None:
        vuelo = db.session.query(Flights, Departures, IataCodes, Airlines, Destinations, FlightStatuses, routeIds, names, Lati
            Destinations, Flights.id_destination == Destinations.id_destination).join(FlightStatuses, Flights.id_status == Fli
        if vuelo is None:
            error = "No se encontró ningún vuelo con el routeId proporcionado"
    return render_template('SeleccionVuelos.html', vuelo=vuelo, error=error)


@app.route('/borrar_avion', methods=['GET'])
def borrar_avion():
    return redirect(url_for('buscar_avion'))
```

La función información de aeropuertos permite al usuario ver la información sobre los distintos aeropuertos con la condición de que escriba el nombre del aeropuerto sobre el que está interesado en consultar información. En caso de que exista el aeropuerto se muestran las aerolíneas, rutas, y nombre del aeropuerto de destino de los aviones.

Objetivos cumplidos con el proyecto

Con este proyecto hemos sido capaces de adentrarnos en conceptos o **frameworks** nunca vistos como **flask y** su relación con el **desarrollo web y conexión con la base de datos a un servidor.** Además, con la creación de la base se ha aprendido a realizar consultas, tablas y relaciones en lenguaje SQL.

También hemos tenido que realizar arreglos a la base de datos para mantener su integralidad, dado que al principio podían aparecer valores repetidos. Finalmente, se ha profundizado bastante en el funcionamiento de Python y su relación con conceptos de programación muy diferentes, ayudándonos a comprender mejor los conceptos de la asignatura y explorar y mejorar muchos de ellos, intentando hacer algo un poco más complejo de lo que meramente podíamos haber hecho.

Readme

Enlace al repositorio github: https://github.com/VforVitorio/Bases-de-Datos-Final.git

Tener en cuenta:

Tener instalado una base de datos mySQL
usuario: root
contraseña: root o contraseña: 1234

Tener configurada una extensión en Visual Studio Code como Database Client JDBC
https://database-client.com/

Tener instalados los siguientes paquetes:

```
import mysql.connector
from flask import Flask, render_template, request, redirect, url_for,
jsonify
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy.orm import joinedload
```

En caso de error probar poner las siguientes instrucciones justo después de las anteriores´

```
import pymysql
pymysql.install_as_MySQLdb()
```

Asegurarse que:

```
app.config['SQLALCHEMY_DATABASE_URI'] =
'mysql://root:1234@localhost:3306/aviones'
```

El usuario, contraseña, así como el nombre de la base de datos sea el correcto