Documentación de los endpoints

EQUIPO 1: Aplicación Especies En Peligro De Extinción

Nuestros endpoints se realizaron en el framework flask e hicimos uso de los 3 tipos de solicitudes http en nuestros endpoints las cuales son **http get, http put, http post**, para iniciar definamos lo que es una petición http:

Hay varios tipos de métodos de petición HTTP, que modifican completamente el tipo de respuesta que obtienes del servidor. Los más comunes son:

- **GET:** Es el método de petición HTTP más utilizado con diferencia. Una petición GET solicita al servidor una información o recurso concreto. Cuando te conectas a un sitio web, tu navegador suele enviar varias peticiones GET para recibir los datos que necesita para cargar la página.
- **HEAD:** Con una petición HEAD, sólo recibes la información de la cabecera de la página que quieres cargar. Puedes utilizar este tipo de petición HTTP para conocer el tamaño de un documento antes de descargarlo mediante GET.
- **POST:** Tu navegador utiliza el método de petición HTTP POST cuando necesita enviar datos al servidor. Por ejemplo, si rellenas un formulario de contacto en un sitio web y lo envías, estás utilizando una petición POST para que el servidor reciba esa información.
- **PUT:** Las peticiones PUT tienen una funcionalidad similar a la del método POST. Sin embargo, en lugar de enviar datos, utilizas las peticiones PUT para actualizar información que ya existe en el servidor final.

Hay otros tipos de peticiones HTTP que puedes utilizar, como los métodos DELETE, PATCH y OPTIONS. Sin embargo, son relativamente poco frecuentes en el uso cotidiano, una vez definido eso podemos comenzar con la documentación de nuestros endpoints, cabe recalcar que se definieron cada una de las solicitudes a partir de funciones.

DOCUMENTACIÓN DE LAS APIS DE MANERA BREVE

- **Configuración General (config.py)**
    - Modo de Desarrollo: Activo (DEBUG = True).
    - Conexión a la Base de Datos MySQL:
    - Host: localhost
    - Usuario: root
    - Contraseña: Su_Contrasena (cambiar a la contraseña real)
    - Nombre de la Base de Datos: Especies

Endpoints (app.py)

- **Listar toda la información de las especies (/Especies)**
    - Método: GET
    - Función: listar_toda_informacion
    - Descripción: Devuelve información detallada de todas las especies en la base de datos.
    - Respuestas:
    - 200 OK: Lista de especies y sus detalles.
    - Tecnología: Flask, Flask_MySQLdb para la conexión a la base de datos.
    - Estructura de Datos: Se realiza una consulta a la base de datos para obtener información de especies, estados y alcaldías.
- **/Especies**
    - Métodos: GET
    - Muestra toda la información de todas las especies (no usuarios); es decir Alcaldias/Municipios, Estados y la información común de estas.

- **/Especies/Estados**
  - Métodos: GET, POST
  - Cuenta con un método GET que nos permite visualizar todos los estados que pertenecen a la Tabla Estados.
  - Y un método POST que nos permite registrar un Estado de la Tabla Estados (por id).

- **/Especies/Estados/<id_estado>**
  - Métodos: GET, DELETE, PUT
  - El método GET nos permite visualizar un estado de la Tabla Estados.
  - El método DELETE nos permite eliminar un estado de la Tabla Estados.
  - El método PUT nos permite actualizar un Estado de la Tabla Estados.
  - (Esto lo hace por medio de la llave primaria que es única).

- **/Especies/Alcaldias**
  - Métodos: GET, POST
  - Cuenta con un método GET que nos permite visualizar todas las Alcaldías.
  - Y un método POST que nos permite registrar una Alcaldía (por id).

- **/Especies/Alcaldias/<id_alcaldia>**
  - Métodos: GET, DELETE, PUT
  - El método GET nos permite visualizar una Alcaldía de la Tabla Alcaldías.
  - El método DELETE nos permite eliminar una Alcaldía de la Tabla Alcaldías.
  - El método PUT nos permite actualizar una Alcaldía de la Tabla Alcaldías.
  - (Esto lo hace por medio de la llave primaria que es única).

- **/Especies/InformacionComun**
  - Métodos: GET, POST
  - Cuenta con un método GET que nos permite visualizar todas las secciones del apartado de la InformacioComun de la Tabla InformacioComun.
  - Y un método POST que nos permite registrar una sección de la InformacionComun (por id).

- **/Especies/InformacionComun/<id_informacion_comun>**
  - Métodos: GET, DELETE, PUT
  - El método GET nos permite visualizar una sección de la InformacioComun de la Tabla InformacioComun.
  - El método DELETE nos permite eliminar una sección de la InformacioComun de la Tabla InformacioComun.
  - El método PUT nos permite actualizar una sección de la InformacioComun de la Tabla InformacioComun.
  - (Esto lo hace por medio de la llave primaria que es única).

- **/Especies/Usuarios**
  - Métodos: POST, GET
  - Cuenta con un método GET que nos permite visualizar todos los Usuarios de la tabla Usuarios (solo los superusuarios lo pueden ver).
  - Y un método POST que nos permite registrar un Usuario (por id, el usuario también puede hacer esto).

- **/Especies/Usuarios/<correo>**
  - Métodos: GET, PUT, DELETE
  - El método GET nos permite visualizar un Usuario de la Tabla Usuarios.
  - El método DELETE nos permite eliminar un Usuario de la Tabla Usuarios (el usuario también puede hacer esto).

- El método PUT nos permite actualizar un Usuario de la Tabla Usuarios (el usuario también puede hacer esto)..
- (Esto lo hace por medio de la llave primaria que es única).

DOCUMENTACIÓN DE LAS APIS DE MANERA EXPLICITA PARA LA GESTIÓN DE ESPECIES:

1. **Documentación de las solicitudes HTTP GET:**

a. **Función listar_toda_informacion()**

Este código Python define un endpoint utilizando el framework Flask para la visualización de datos almacenados en una base de datos. El endpoint está configurado para aceptar solicitudes HTTP GET y se encuentra en la ruta '/Especies'.

- **Definición del Endpoint:**
  - @app.route('/Especies', methods=['GET'])
  - def listar_toda_informacion():
  - Se define una función llamada listar_toda_informacion() asociada al endpoint '/Especies'. Esta función responde solo a solicitudes HTTP GET.

- **Conexión a la Base de Datos:**
  - cursor = mysql.connection.cursor()
  - Se establece una conexión al servidor de la base de datos utilizando un cursor.

- **Consulta a la Base de Datos - Estados:**
  - sql_estados = "SELECT * FROM Estado"
  - cursor.execute(sql_estados)
  - datos_estados = cursor.fetchall()
  - Se realiza una consulta SQL para obtener toda la información de la tabla 'Estado' y se almacenan los resultados en la variable datos_estados.

- **Procesamiento de Datos - Estados:**
  - estados = []
  - for estado in datos_estados:
  - info_estado = {
    - 'id_estado': estado[0],
    - 'nombre_estado': estado[1],
    - 'numero_alcaldias': estado[2],
  - }
  - estados.append(info_estado)
  - Se procesan los datos de la tabla 'Estado' y se crean diccionarios con la información relevante que se almacenan en la lista estados.

- **Consulta a la Base de Datos - Alcaldías:**
  - sql_alcaldias = "SELECT * FROM Alcaldia"
  - cursor.execute(sql_alcaldias)
  - datos_alcaldias = cursor.fetchall()
  - Se realiza una consulta SQL para obtener toda la información de la tabla 'Alcaldia' y se almacenan los resultados en la variable datos_alcaldias.

- **Procesamiento de Datos - Alcaldías:**
  - alcaldias = []
  - for alcaldia in datos_alcaldias:

- info_alcaldia = {
    - 'id_alcaldia': alcaldia[0],
    - 'nombre_alcaldia': alcaldia[1],
- }
- alcaldias.append(info_alcaldia)
- Se procesan los datos de la tabla 'Alcaldia' y se crean diccionarios con la información relevante que se almacenan en la lista alcaldias.

- **Consulta a la Base de Datos - Información Común:**
    - sql_informacion_comun = "SELECT * FROM InformacionComun"
    - cursor.execute(sql_informacion_comun)
    - datos_informacion_comun = cursor.fetchall()
    - Se realiza una consulta SQL para obtener toda la información de la tabla 'InformacionComun' y se almacenan los resultados en la variable datos_informacion_comun.

- **Procesamiento de Datos - Información Común:**
    - informacion_comun = []
    - for fila in datos_informacion_comun:
    - info_comun = {
        - # ... (se crean las claves y valores correspondientes)
    - }
    - informacion_comun.append(info_comun)
    - Se procesan los datos de la tabla 'InformacionComun' y se crean diccionarios con la información relevante que se almacenan en la lista informacion_comun.

- **Respuesta JSON:**
    - return jsonify({
    - 'estados': estados,
    - 'alcaldias': alcaldias,
    - 'informacion_comun': informacion_comun,
    - 'mensaje': 'Toda la información de la base de datos listada.'
    - })
    - Se devuelve una respuesta en formato JSON que contiene la información de los estados, alcaldías e información común, junto con un mensaje indicando el éxito de la operación. En caso de algún error, se devuelve un mensaje de error en formato JSON.

b. Función listar_estados()

Este endpoint devuelve una lista de todos los estados junto con información específica sobre cada uno, como el ID del estado, el nombre y el número de alcaldías.

- **Proceso:**
    - Se establece una conexión a la base de datos.
    - Se ejecuta una consulta SQL para seleccionar todos los datos de la tabla Estado.
    - Se procesan los resultados y se construye una lista de diccionarios donde cada diccionario contiene la información de un estado.
    - Se devuelve una respuesta JSON que contiene la lista de estados y un mensaje indicando que la información de todos los estados ha sido listada.
- **Leer Información de un Estado Específico:**
    - Ruta del Endpoint: /Especies/Estados/<id_estado> con el método GET.
    - Parámetro de Ruta: id_estado es un parámetro de la URL que indica el ID del estado que se desea consultar.

- **Proceso:**
  - Se establece una conexión a la base de datos.
  - Se ejecuta una consulta SQL para seleccionar los datos del estado con el ID proporcionado.
  - Se verifica si la consulta devuelve algún resultado.
  - Si se encuentra información, se construye un diccionario con la información del estado y se devuelve como una respuesta JSON junto con un mensaje indicando que la información ha sido encontrada.
  - Si no se encuentra información, se devuelve un mensaje indicando que la información no fue encontrada.
- **Manejo de Errores:**
  - Ambas funciones (listar_estados y leer_informacion_estado) tienen un bloque except que captura cualquier excepción que pueda ocurrir durante la ejecución y devuelve una respuesta JSON con un mensaje de error en caso de que se produzca una excepción.
  - Se asume que hay una conexión previamente establecida a una base de datos MySQL y que el objeto mysql.connection está configurado adecuadamente

c. Función leer_informacion_estado(id_estado):

- **Definición del Endpoint:**
  - @app.route('/Especies/Estados/<id_estado>', methods=['GET'])
  - @app.route: Esto es un decorador que asocia una función con una ruta específica en la aplicación web. En este caso, la ruta es '/Especies/Estados/<id_estado>'.
  - <id_estado>: Es una parte de la URL que actuará como un parámetro variable. El valor de este parámetro se pasará a la función leer_informacion_estado como argumento.
- **Función leer_informacion_estado:**
  - def leer_informacion_estado(id_estado):
  - Esta función se ejecutará cuando se haga una solicitud GET a la ruta '/Especies/Estados/<id_estado>'.
  - id_estado: Es el parámetro que se toma de la URL y se utiliza para buscar información específica sobre un estado en la base de datos.
- **Bloque Try-Except:**
  - try:
  - Se inicia un bloque try-except para manejar posibles excepciones durante la ejecución del código.
- **Conexión a la Base de Datos:**

- cursorEstado = mysql.connection.cursor()
- Se establece una conexión a la base de datos utilizando un cursor.
- **Consulta SQL:**
  - estado_sql = f"SELECT * FROM Estado WHERE id_estado = {id_estado}"
  - cursorEstado.execute(estado_sql)
  - Se construye una consulta SQL para seleccionar todos los campos de la tabla "Estado" donde el id_estado coincida con el valor proporcionado en la URL.
  - La consulta se ejecuta utilizando el cursor.
- **Obtención de Datos:**
  - datos = cursorEstado.fetchone()
  - Se obtiene la primera fila de resultados de la consulta.
- **Verificación de Datos y Respuestas JSON:**
  - if datos:
  - info_estado = {
  - 'id_estado': datos[0],
  - 'nombre_estado': datos[1],
  - 'numero_alcaldias': datos[2]

- }
- return jsonify({'informacionEstado': info_estado, 'mensaje': 'Información encontrada.'})
- else:
- return jsonify({'Mensaje': 'Información no encontrada'})
- Si se encuentran datos, se construye un diccionario info_estado con la información relevante y se devuelve como una respuesta JSON.
- Si no se encuentran datos, se devuelve un mensaje indicando que la información no fue encontrada.

- **Manejo de Excepciones:**
  - except Exception as ex:
  - return jsonify({'Mensaje': 'Error'})
  - En caso de que ocurra alguna excepción durante la ejecución del código en el bloque try, se captura y se devuelve un mensaje de error en formato JSON. Esto es útil para proporcionar retroalimentación en caso de problemas, por ejemplo, errores de conexión a la base de datos.

d. Función listar_alcadias()

- **Ruta del Endpoint:**
  - @app.route('/Especies/Alcaldias', methods=['GET'])
  - Define la ruta del endpoint. En este caso, el endpoint estará disponible en la URL '/Especies/Alcaldias' y responderá solo a solicitudes HTTP GET.
- **Función listar_alcaldias:**
  - def listar_alcaldias():
  - Define la función que se ejecutará cuando se realice una solicitud al endpoint. Dentro de esta función, se realiza la conexión a la base de datos MySQL y se ejecuta una consulta SQL para obtener información de todas las alcaldías.
- **Consulta SQL:**
  - sql = "SELECT * FROM Alcaldia"
  - cursorAlcaldias.execute(sql)
  - Se define una consulta SQL para seleccionar todas las columnas de la tabla "Alcaldia". Luego, la consulta se ejecuta utilizando el cursor de la conexión a la base de datos.
- **Obtención de Datos:**
  - datos = cursorAlcaldias.fetchall()
  - Se recuperan todos los datos resultantes de la consulta SQL y se almacenan en la variable datos.
- **Creación de Lista de Alcaldías:**
  - alcaldias = []
  - for alcaldia in datos:
  - info_alcaldia = {
  - 'id_alcaldia': alcaldia[0],
  - 'nombre_alcaldia': alcaldia[1]
  - }
  - alcaldias.append(info_alcaldia)
  - Se itera sobre los datos de la consulta y se crea una lista de diccionarios, donde cada diccionario representa la información de una alcaldía. En este caso, se incluye el ID de la alcaldía y su nombre.
- **Respuesta en JSON:**
  - return jsonify({'alcaldias': alcaldias, 'mensaje': 'Información de todas las alcaldías listada.'})
  - Finalmente, la función devuelve una respuesta JSON que incluye la lista de alcaldías y un mensaje indicando que la información de todas las alcaldías ha sido listada con éxito. En caso de algún error, se captura en la sección except y se devuelve un mensaje de error en formato JSON.

e. **Función leer_informacion_alcaldia(id_alcaldia)**
   - @app.route('/Especies/Alcaldias/<id_alcaldia>', methods=['GET']): Esto define una ruta en la aplicación web. El endpoint se encuentra en la URL /Especies/Alcaldias/<id_alcaldia> y acepta solicitudes del tipo GET.
   - def leer_informacion_alcaldia(id_alcaldia): Esta es la función asociada con el endpoint. Toma un parámetro id_alcaldia, que se espera que sea el identificador único de la alcaldía que se desea consultar.
   - Se intenta realizar una conexión a la base de datos utilizando cursorAlcaldia = mysql.connection.cursor().
   - alcaldia_sql = f"SELECT * FROM Alcaldia WHERE id_alcaldia = {id_alcaldia}": Se construye una consulta SQL para seleccionar todos los campos de la tabla "Alcaldia" donde el ID de la alcaldía coincide con el proporcionado en la solicitud.
   - cursorAlcaldia.execute(alcaldia_sql): La consulta se ejecuta en la base de datos.
   - datos = cursorAlcaldia.fetchone(): Se recupera la primera fila de resultados de la consulta.
   - Se verifica si se encontraron datos (if datos:). Si se encontraron, se construye un diccionario info_alcaldia con la información relevante.
   - Se devuelve una respuesta JSON con la información de la alcaldía si se encontraron datos, junto con un mensaje indicando que la información fue encontrada. En caso contrario, se devuelve un mensaje indicando que la información no fue encontrada.
   - En caso de cualquier excepción durante el proceso, se devuelve un mensaje de error.
   - En resumen, este endpoint está diseñado para recuperar información específica de una alcaldía mediante una solicitud GET, consultando una base de datos y devolviendo la información en formato JSON.

f. **Función listar_informacion_comun():**
   - **Ruta del Endpoint:**
     - @app.route('/Especies/InformacionComun', methods=['GET'])
     - Define una ruta para el endpoint. En este caso, la ruta es '/Especies/InformacionComun' y solo acepta solicitudes HTTP GET.
   - **Función del Endpoint:**
     - def listar_informacion_comun():
     - Esta función se ejecutará cuando se realice una solicitud en la ruta especificada. Se encarga de manejar la lógica para recuperar la información común de la base de datos y formatearla para su presentación.
   - **Consulta a la Base de Datos:**
     - with mysql.connection.cursor() as cursor:
     - sql = "SELECT * FROM InformacionComun"
     - cursor.execute(sql)
     - datos = cursor.fetchall()
     - Se establece una conexión a la base de datos MySQL y se ejecuta una consulta SQL para seleccionar todos los registros de la tabla "InformacionComun". Los resultados se almacenan en la variable datos.
   - **Formato de Datos:**
     - informacion = []
     - for fila in datos:
     - info = {
     - # ... (se extraen los campos de la fila y se asignan a un diccionario)
     - }
     - informacion.append(info)
     - Se recorren los resultados de la consulta y se crea una lista de diccionarios, donde cada diccionario representa la información de un registro. Los campos de la tabla se asignan a claves correspondientes en el diccionario.
   - **Respuesta JSON:**

- return jsonify({'informacion_comun': informacion, 'mensaje': 'Información almacenada y listada.'})
- Se devuelve una respuesta JSON que contiene la información formateada y un mensaje indicando que la información ha sido almacenada y listada. En caso de algún error, se devuelve un mensaje de error.

- **Manejo de Excepciones:**
  - except Exception as ex:
  - return jsonify({'mensaje': 'Error'})
  - Si ocurre alguna excepción durante la ejecución (por ejemplo, un error en la conexión a la base de datos), se devuelve un mensaje de error en formato JSON.

g. Función leer_informacion_comun(id_informacion_comun):

- **Ruta del Endpoint:**
  - @app.route('/Especies/InformacionComun/<id_informacion_comun>', methods=['GET'])
  - Define la ruta del endpoint como '/Especies/InformacionComun/<id_informacion_comun>' y especifica que este endpoint solo acepta solicitudes HTTP GET.
- **Función del Endpoint:**
  - def leer_informacion_comun(id_informacion_comun):
  - Define una función llamada leer_informacion_comun que toma el parámetro id_informacion_comun de la URL.
- **Consulta a la Base de Datos:**
  - cursorInformacionComun = mysql.connection.cursor()
  - sql = f"SELECT * FROM InformacionComun WHERE id_informacion_comun = {id_informacion_comun}"
  - cursorInformacionComun.execute(sql)
  - datos = cursorInformacionComun.fetchone()
  - Establece una conexión a la base de datos MySQL, ejecuta una consulta SQL para seleccionar todos los campos de la tabla InformacionComun donde el id_informacion_comun coincide con el valor proporcionado, y recupera la primera fila de resultados.
- **Procesamiento de Resultados:**
  - if datos:
  - # Procesa los datos y los almacena en un diccionario
  - info_comun = {
  - # ... (se enumeran los campos de la tabla y sus respectivos valores)
  - }
  - return jsonify({'informacion_comun': info_comun, 'mensaje': 'Información común encontrada.'})
  - else:
  - return jsonify({'Mensaje': 'Información común no encontrada'})
  - Si se encuentran datos en la base de datos, los procesa y los almacena en un diccionario llamado info_comun. Luego, devuelve este diccionario junto con un mensaje indicando que la información común fue encontrada. Si no hay datos, devuelve un mensaje indicando que la información común no fue encontrada.
- **Manejo de Excepciones:**
  - except Exception as ex:
  - return jsonify({'Mensaje': 'Error'})
  - Captura cualquier excepción que pueda ocurrir durante la ejecución (por ejemplo, errores de conexión a la base de datos) y devuelve un mensaje de error genérico en formato JSON.

2. Documentación de las solicitudes HTTP POST:

a. Función agregar_estado():

- **@app.route('/Especies/Estados', methods=['POST']):** Este decorador especifica la ruta del endpoint ('/Especies/Estados') y los métodos HTTP permitidos (en este caso, solo POST).
- **def agregar_estado()::** Esta función se ejecuta cuando se realiza una solicitud POST a la ruta '/Especies/Estados'.
- **cursorEstados = mysql.connection.cursor():** Se crea un objeto cursor para interactuar con la base de datos. Se asume que mysql es un objeto de conexión a la base de datos MySQL.
- **estado_id = request.json['id_estado']:** Se obtiene el valor del campo 'id_estado' de los datos JSON proporcionados en la solicitud.
- **cursorEstados.execute(f"SELECT * FROM Estado WHERE id_estado = {estado_id}"):** Se realiza una consulta SQL para verificar si ya existe un estado con la misma clave primaria ('id_estado'). El resultado se almacena en existing_estado.
- **if existing_estado:** Se verifica si ya existe un estado con la misma clave primaria. Si es así, se devuelve un mensaje de error.
- **sql = """INSERT INTO Estado (id_estado, nombre_estado, numero_alcaldias) VALUES ({0}, '{1}', {2})""".format(...):** Se construye una consulta SQL de inserción con los datos proporcionados en la solicitud JSON.
- **cursorEstados.execute(sql):** Se ejecuta la consulta de inserción en la base de datos
- **mysql.connection.commit():** Se confirma la acción de inserción, guardando los cambios en la base de datos.
- **return jsonify({'mensaje': "Informacion registrada"}):** Si la inserción es exitosa, se devuelve un mensaje indicando que la información ha sido registrada.
- **except Exception as ex:** Se captura cualquier excepción que ocurra durante la ejecución y se maneja imprimiendo el error y devolviendo un mensaje de error en formato JSON.

b. Función agregar_alcaldia():

- **Ruta y Método HTTP:**
  - La ruta del endpoint es '/Especies/Alcaldias'.
  - El método HTTP permitido es POST.
- **Función agregar_alcaldia:**
  - Esta función se ejecuta cuando se realiza una solicitud POST a la ruta especificada.
  - Utiliza la librería mysql para interactuar con la base de datos.
- **Captura de Datos:**
  - La función obtiene el ID y el nombre de la alcaldía a través de la solicitud POST en formato JSON.
- **Verificación de Existencia:**
  - Se verifica si ya existe una alcaldía con el mismo ID en la base de datos.
- **Inserción en la Base de Datos:**
  - Si no existe una alcaldía con el mismo ID, se realiza la inserción en la base de datos.
- **Respuesta JSON:**
  - Se devuelve una respuesta JSON indicando si la operación fue exitosa o si se encontró un error.
- **Manejo de Errores:**
  - En caso de que ocurra una excepción durante la ejecución, se imprime el error y se devuelve un mensaje de error en la respuesta JSON.

c. Función agregar_informacion_comun():

- **Ruta y Método:** El decorador @app.route('/Especies/InformacionComun', methods=['POST']) indica que esta función manejará solicitudes POST enviadas a la ruta '/Especies/InformacionComun' de la aplicación Flask.
- **Función Principal:** La función agregar_informacion_comun() es ejecutada cuando se recibe una solicitud POST en la ruta mencionada.
- **Conexión a la Base de Datos:** Se establece una conexión con la base de datos utilizando un cursor llamado cursorInfoComun.

- **Consulta SQL y Parámetros:** Se construye una consulta SQL para insertar información en una tabla llamada 'InformacionComun'. Los parámetros de la consulta son obtenidos del cuerpo de la solicitud JSON. La consulta incluye campos como 'entidad_tipo', 'id_geo', 'grupoSNIB', etc.
- **Ejecución de la Consulta:** La consulta SQL es ejecutada utilizando cursorInfoComun.execute(sql) y luego se confirma la transacción con mysql.connection.commit().
- **Respuesta JSON:** Si la inserción en la base de datos es exitosa, la función devuelve un mensaje JSON indicando que la información ha sido registrada. En caso de algún error, se imprime el error y se devuelve un mensaje de error en JSON.

3. Documentación de las solicitudes HTTP DELETE:

a. Función eliminar_estado():

- **Ruta del Endpoint:** El endpoint está asociado a la ruta '/Especies/Estados/<id_estado>', donde <id_estado> es un parámetro variable que representa el identificador único del estado que se desea eliminar.
- **Método DELETE:** Este endpoint está configurado para responder únicamente a solicitudes HTTP DELETE. Esto significa que se espera que los clientes que deseen eliminar un estado envíen una solicitud DELETE a esta ruta.
- **Conexión a la Base de Datos:** Se utiliza una conexión a una base de datos MySQL. El código asume que ya hay una conexión establecida antes de que este endpoint sea invocado.
- **Verificación de Existencia del Estado:** Se realiza una consulta a la base de datos para verificar si el estado con el ID proporcionado (id_estado) existe. Si no se encuentra ningún estado con ese ID, se devuelve un mensaje de error indicando que el estado no existe.
- **Eliminación del Estado:** Si se encuentra el estado, se ejecuta una sentencia SQL DELETE para eliminar el registro correspondiente en la base de datos.
- **Commit en la Base de Datos:** Después de la eliminación, se realiza un commit en la base de datos para confirmar los cambios.
- **Respuestas JSON:** Se devuelven respuestas JSON indicando el resultado de la operación. Si la eliminación es exitosa, se envía un mensaje de éxito. En caso de cualquier error durante el proceso, se devuelve un mensaje de error.
- **Manejo de Excepciones:** Se implementa un bloque try-except para capturar posibles excepciones durante la ejecución del código. En caso de que se produzca una excepción, se imprime un mensaje de error y se devuelve una respuesta JSON indicando un fallo en la eliminación del estado.

b. Función eliminar_alcaldía():

- **Ruta del Endpoint:** El endpoint está definido para la ruta /Especies/Alcaldias/<id_alcaldia>, donde <id_alcaldia> es un parámetro variable que representa el identificador único de la alcaldía que se desea eliminar.
- **Método HTTP:** Este endpoint responde solo a solicitudes HTTP DELETE. Es decir, se espera que los clientes envíen una solicitud DELETE para interactuar con este endpoint.
- **Función de Manejo:** La función asociada con este endpoint comienza creando un cursor para interactuar con la base de datos MySQL.
- **Verificación de Existencia:** Se realiza una consulta para verificar si la alcaldía con el ID proporcionado existe en la base de datos. Si no existe, se devuelve un mensaje de error en formato JSON indicando que la alcaldía no existe.
- **Eliminación:** Si la alcaldía existe, se ejecuta una sentencia SQL DELETE para eliminar la alcaldía de la base de datos. Luego, se realiza la confirmación para aplicar los cambios a la base de datos.
- **Respuestas JSON:** Se devuelven mensajes JSON indicando el resultado de la operación. Si la alcaldía se elimina correctamente, se devuelve un mensaje de éxito. En caso de cualquier error durante el proceso, se captura la excepción y se devuelve un mensaje de error.

c. Función eliminar_información_común():

- **Ruta del Endpoint:** El endpoint está asociado con la ruta '/Especies/InformacionComun/<id_informacion_comun>'. El <id_informacion_comun> es un parámetro de ruta que se espera que sea proporcionado en la URL.
- **Método HTTP:** Este endpoint responde solo a solicitudes HTTP de tipo DELETE.
- **Función asociada al Endpoint:** La función eliminar_informacion_comun se ejecuta cuando se realiza una solicitud DELETE a la ruta especificada. Esta función toma el parámetro id_informacion_comun de la URL.
- **Conexión a la Base de Datos:** El código utiliza un cursor de MySQL para interactuar con una base de datos. Se intenta realizar una conexión y se ejecuta una consulta para verificar si la información común con el ID proporcionado existe en la base de datos.
- **Verificación de Existencia:** Si la información común no existe en la base de datos, se devuelve un mensaje de error indicando que no se puede eliminar porque no existe.
- **Eliminación de Información Común:** Si la información común existe, se ejecuta una consulta DELETE para eliminar la entrada correspondiente en la base de datos.
- **Commit en la Base de Datos:** Después de realizar la eliminación, se realiza un commit en la base de datos para confirmar los cambios.
- **Respuesta JSON:** Se devuelve una respuesta en formato JSON indicando si la eliminación fue exitosa o si hubo algún error durante el proceso.
- **Manejo de Excepciones:** Se utiliza un bloque try-except para manejar excepciones. Si ocurre algún error durante el proceso, se imprime en la consola y se devuelve un mensaje de error en formato JSON.

4. Documentacion de las solicitudes HTTP PUT:

a. Función actualizar_estado():

- **Ruta del Endpoint:** El endpoint está definido en la ruta /Especies/Estados/<id_estado>, donde <id_estado> es un parámetro variable que representa el identificador único del estado que se desea actualizar.
- **Método PUT:** El endpoint está configurado para aceptar solicitudes HTTP con el método PUT. El método PUT se utiliza comúnmente para actualizar recursos en un servidor.
- **Función actualizar_estado:** Esta función se ejecuta cuando se recibe una solicitud PUT en la ruta especificada. Toma el id_estado de la URL como argumento.
- **Verificación de Existencia del Estado:** Antes de intentar actualizar el estado, se realiza una consulta a la base de datos para verificar si el estado con el id_estado proporcionado existe. Si no existe, se devuelve un mensaje de error.

- **Actualización del Estado:** Si el estado existe, se procede a actualizar sus datos. Los nuevos datos se toman de los parámetros proporcionados en la solicitud JSON. Se construye una consulta SQL para actualizar el nombre del estado y el número de alcaldías asociadas.
- **Ejecución de la Consulta SQL:** Se ejecuta la consulta SQL utilizando un cursor de la conexión a la base de datos, y se confirma la transacción con mysql.connection.commit().
- **Respuesta JSON:** Se devuelve una respuesta JSON indicando que la actualización fue exitosa, junto con un mensaje que confirma el éxito y proporciona el ID del estado actualizado.
- **Manejo de Errores:** En caso de cualquier excepción durante la ejecución, se imprime un mensaje de error y se devuelve una respuesta JSON indicando que ocurrió un error durante la actualización del estado.

b. Función actualizar alcaldía():

- **Ruta del Endpoint:** @app.route('/Especies/Alcaldias/<id_alcaldia>', methods=['PUT']): Define la ruta del endpoint. En este caso, el endpoint está ubicado en /Especies/Alcaldias/<id_alcaldia>, donde <id_alcaldia> es un marcador de posición para el identificador único de la alcaldía que se proporcionará en la URL. El método permitido para esta ruta es PUT.
- **Función actualizar_alcaldia:** Esta función se ejecutará cuando se realice una solicitud PUT a la ruta especificada.

- **id_alcaldia:** Se recibe como un parámetro de la URL y se utiliza para identificar la alcaldía que se actualizará.
- **Verificación de Existencia de Alcaldía:** Se realiza una consulta a la base de datos para verificar si la alcaldía con el id_alcaldia proporcionado existe. Si no existe, se devuelve un mensaje de error en formato JSON indicando que la alcaldía no existe.
- **Actualización de la Alcaldía:** Si la alcaldía existe, se procede a actualizarla con los nuevos datos proporcionados en la solicitud PUT. La consulta SQL utiliza el método UPDATE para modificar la información de la alcaldía en la base de datos.
- **Manejo de Excepciones:** Se utiliza un bloque try-except para manejar posibles excepciones que puedan ocurrir durante la ejecución del código. Si hay algún error, se imprime en la consola y se devuelve un mensaje de error en formato JSON.
- **Respuestas JSON:** Se devuelven respuestas en formato JSON que informan sobre el resultado de la operación, ya sea que la alcaldía se haya actualizado correctamente o si se encontró algún error durante el proceso.

c. Función actualizar_informacion_comun(id_informacion_comun):

- **Ruta y Método HTTP:**
  - Ruta: '/Especies/InformacionComun/<id_informacion_comun>' - Indica que este endpoint responderá a las solicitudes que tengan esta ruta, con <id_informacion_comun> siendo un parámetro variable en la URL.
  - Método HTTP: PUT - Indica que este endpoint se utiliza para actualizar recursos en el servidor.
- **Función Asociada al Endpoint:**
  - actualizar_informacion_comun: Esta función se ejecuta cuando se recibe una solicitud PUT en la ruta especificada. El parámetro id_informacion_comun se obtiene de la URL y se utiliza para identificar la información común que se actualizará en la base de datos**.**
- **Manejo de la Base de Datos (MySQL):**
  - Se establece una conexión a la base de datos MySQL.
  - Se verifica si la información común con el ID proporcionado existe en la base de datos. Si no existe, se devuelve un mensaje de error.
  - Si la información común existe, se procede a construir y ejecutar una consulta SQL de actualización para modificar los campos correspondientes en la tabla InformacionComun.
- **Datos de la Solicitud (Request JSON):**
  - Los nuevos datos para la información común se obtienen del cuerpo de la solicitud en formato JSON. Estos datos incluyen valores para los campos como 'entidad_tipo', 'id_geo', 'grupoSNIB', etc.
- **Commit en la Base de Datos:**
  - Después de ejecutar la consulta de actualización, se realiza un commit en la base de datos para confirmar los cambios.
- **Respuesta JSON:**
  - Se devuelve una respuesta en formato JSON indicando si la actualización fue exitosa o si se encontró algún error durante el proceso.
- **Manejo de Excepciones:**
  - Se utiliza un bloque try-except para manejar posibles excepciones durante la ejecución del código. Si se produce una excepción, se imprime un mensaje de error y se devuelve una respuesta JSON indicando el problema.

5. Registro de usuario:

a. HTTP POST:

a. Función registrar_usuario():

- **Ruta del Endpoint:** @app.route('/Especies/Usuarios', methods=['POST']) Este código define una ruta para el endpoint /Especies/Usuarios que acepta solicitudes HTTP del tipo POST. En Flask, @app.route es un decorador que vincula una función a una ruta específica.
- **Función registrar_usuario():** Esta función se ejecutará cuando se realice una solicitud POST a la ruta /Especies/Usuarios. Comienza intentando conectarse a una base de datos MySQL (presumiblemente configurada en otra parte de la aplicación).
- **Verificación de Correo Duplicado:** Verifica si el correo proporcionado en la solicitud ya está registrado en la base de datos. Realiza una consulta SQL para buscar un usuario con el mismo correo. Si encuentra un usuario con el mismo correo, devuelve un mensaje JSON indicando que ya está registrado.
- **Registro de Nuevo Usuario:** Si no se encuentra un usuario con el mismo correo, procede a insertar un nuevo usuario en la base de datos. Utiliza la información proporcionada en la solicitud JSON, que debería incluir campos como correo y contraseña. Realiza una consulta SQL de inserción para agregar el nuevo usuario a la tabla Usuario.
- Después de la inserción, confirma los cambios en la base de datos con mysql.connection.commit(). Devuelve un mensaje JSON indicando que el usuario se registró correctamente.
- **Manejo de Errores:** Si ocurre alguna excepción durante el proceso (por ejemplo, problemas de conexión a la base de datos), captura la excepción, imprime un mensaje de error y devuelve un mensaje JSON indicando que hubo un error al intentar registrar al usuario.

b. HTTP GET:

a. Función obtener_usuarios():

- **Decorador @app.route('/Especies/Usuarios', methods=['GET']):** Este decorador especifica la ruta del endpoint ("/Especies/Usuarios") y los métodos HTTP que puede manejar (en este caso, solo GET).
- **Función obtener_usuarios():** Esta función es la que se ejecutará cuando se realice una solicitud GET a la ruta especificada. Se utiliza un bloque try-except para manejar posibles excepciones durante la ejecución.
- **Conexión a la base de datos:** Se intenta establecer una conexión a la base de datos utilizando un cursor de MySQL.
- **Consulta SQL y obtención de resultados:** Se ejecuta la consulta SQL "SELECT * FROM Usuario" para obtener todos los registros de la tabla "Usuario". Los resultados se almacenan en la variable usuarios.
- **Formato de la respuesta:** Se crea una lista llamada lista_usuarios que contendrá la información de cada usuario en forma de diccionario. Por cada usuario en la variable usuarios, se crea un diccionario info_usuario con las claves 'id_usuario', 'correo' y 'contrasena'. Los diccionarios individuales se agregan a la lista lista_usuarios.
- **Respuesta JSON:** Finalmente, se utiliza la función jsonify de Flask para formatear la respuesta como un objeto JSON. La respuesta incluye la lista de usuarios y un mensaje indicando el éxito de la operación o un mensaje de error en caso de excepción.
- **Manejo de excepciones:** Si ocurre alguna excepción durante la ejecución (por ejemplo, un error en la conexión a la base de datos), se captura en el bloque except. En este caso, se imprime el mensaje de error y se devuelve una respuesta JSON indicando el problema.

b. Función obtener_usuario_por_correo(correo):

- **@app.route('/Especies/Usuarios/<correo>', methods=['GET']):** Esto establece la ruta del endpoint como /Especies/Usuarios/<correo> y especifica que solo se deben manejar solicitudes HTTP GET.
- **def obtener_usuario_por_correo(correo)::** Define la función asociada con este endpoint, tomando el parámetro correo de la URL.
- **En el bloque try, se realiza lo siguiente:**

- Se establece una conexión con la base de datos (presumiblemente MySQL) usando mysql.connection.cursor().
- Se ejecuta una consulta SQL para seleccionar todos los campos de la tabla Usuario donde la columna correo coincide con el valor proporcionado en el parámetro.
- Se recupera la primera fila de los resultados de la consulta utilizando cursor.fetchone().
- Si se encuentra un usuario, se crea un diccionario info_usuario con los detalles del usuario.
- Se devuelve una respuesta JSON con la información del usuario y un mensaje indicando que la información se obtuvo correctamente.
- Si no se encuentra ningún usuario, se devuelve una respuesta JSON indicando que el usuario no fue encontrado.

- **En el bloque except:** se manejan excepciones. Si ocurre algún error durante la ejecución del bloque try, se captura la excepción, se imprime el mensaje de error y se devuelve una respuesta JSON indicando que hubo un error al intentar obtener la información del usuario.

c. HTTP PUT:

a. Función actualizar_usuario(correo):

- **@app.route('/Especies/Usuarios/<correo>', methods=['PUT']):** Define una ruta para el endpoint. El <correo> indica que se espera un parámetro llamado "correo" en la URL. Este endpoint solo responde a solicitudes HTTP de tipo PUT.
- **def actualizar_usuario(correo)::** Define la función que se ejecutará cuando se reciba una solicitud en este endpoint. Toma el parámetro "correo" de la URL.
- Se inicia un bloque try para manejar posibles excepciones.
- **cursor = mysql.connection.cursor():** Crea un objeto de cursor para interactuar con la base de datos.
- **cursor.execute(f"SELECT * FROM Usuario WHERE correo = '{correo}'"):** Ejecuta una consulta SQL para verificar si el usuario con el correo proporcionado existe en la base de datos.
- existing_user = cursor.fetchone(): Obtiene el resultado de la consulta.
- **if not existing_user::** Verifica si el usuario no existe. Si es así, devuelve un mensaje de error.
- Luego, se procede a construir y ejecutar una consulta SQL de actualización para cambiar la contraseña del usuario con el correo proporcionado.
- **request.json['contrasena']:** Obtiene la nueva contraseña del cuerpo de la solicitud PUT.
- **sql = """UPDATE Usuario SET contrasena = '{0}' WHERE correo = '{1}'""".format(request.json['contrasena'], correo):** Construye la consulta SQL de actualización con la nueva contraseña y el correo del usuario.
- **cursor.execute(sql):** Ejecuta la consulta SQL de actualización.
- **mysql.connection.commit():** Confirma los cambios en la base de datos.
- Devuelve un mensaje JSON indicando que la información del usuario ha sido actualizada correctamente.
- En caso de excepción, imprime el error y devuelve un mensaje JSON indicando un error al intentar actualizar la información del usuario.

d. HTTP DELETE:

a. Función eliminar_usuario(correo):

- **Ruta y Método HTTP:**
  - La ruta del endpoint es '/Especies/Usuarios/<correo>', donde <correo> es un parámetro variable que representa la dirección de correo electrónico de un usuario.
  - El método HTTP asociado es DELETE, lo que significa que este endpoint se utiliza para eliminar recursos, en este caso, un usuario.
- **Función eliminar_usuario:**
  - Esta función se ejecutará cuando se reciba una solicitud DELETE en la ruta mencionada.

- El parámetro correo es capturado de la URL y se utiliza para identificar el usuario que se eliminará.
- **Conexión a la Base de Datos:**
  - Se utiliza una conexión a una base de datos MySQL para realizar operaciones relacionadas con el usuario.
- **Verificación de Existencia del Usuario:**
  - Antes de intentar eliminar al usuario, se ejecuta una consulta SQL para verificar si el usuario con la dirección de correo proporcionada ya existe en la base de datos.
  - Si no se encuentra ningún usuario, se devuelve un mensaje JSON indicando que el usuario no existe.
- **Eliminación del Usuario:**
  - Si el usuario existe, se procede a ejecutar otra consulta SQL para eliminar al usuario de la base de datos.
  - Después de ejecutar la consulta, se realiza un commit en la base de datos para confirmar los cambios.
- **Respuestas JSON:**
  - Se devuelven respuestas JSON indicando el resultado de la operación.
  - Si todo va bien, se envía un mensaje de éxito con la confirmación de que el usuario ha sido eliminado correctamente.
  - En caso de errores durante el proceso, se envía un mensaje de error junto con información sobre la excepción que se produjo.
- **Manejo de Excepciones:**
  - Se implementa un bloque try-except para manejar cualquier excepción que pueda ocurrir durante la ejecución de la función.
  - Si se produce una excepción, se imprime un mensaje de error y se devuelve una respuesta JSON indicando que hubo un error al intentar eliminar al usuario.

6. **Pagina no encontrada y main**
   - **Función pagina_no_encontrada:** Esta función es un manejador de errores para el código de estado HTTP 404 (página no encontrada). Toma un argumento error (aunque no lo utiliza en el código), y devuelve una respuesta HTML que indica que la página que se intenta buscar no existe, junto con el código de estado 404.
   - **Configuración de la aplicación (app.config.from_object(config['development'])):** Parece que la aplicación Flask está siendo configurada utilizando una configuración específica para el entorno de desarrollo. La configuración se obtiene de un objeto llamado config y se selecciona la configuración específica para el entorno de desarrollo.
   - **Registro del manejador de errores:** Se registra la función pagina_no_encontrada como manejador de errores para el código de estado 404. Esto significa que cuando la aplicación encuentra una página no encontrada, Flask llamará a esta función para manejar el error y devolver la respuesta correspondiente.
   - **Ejecución de la aplicación (app.run(debug=True)):** Si el script se está ejecutando directamente (no siendo importado como módulo), la aplicación Flask se ejecuta en modo de depuración (debug=True). Esto facilita la identificación y corrección de errores durante el desarrollo.

Este código establece una configuración para el entorno de desarrollo, registra un manejador de errores para las páginas no encontradas, y luego ejecuta la aplicación Flask en modo de depuración. La función pagina_no_encontrada se encarga de devolver una página HTML personalizada cuando se produce un error 404.