Informe prácticas

1. Repositorio online

Se ha desarrollado un repositorio online con el propósito de almacenar y clasificar imágenes de plantas. El repositorio tiene dos objetivos principales:

- Crear un *dataset* para poder ser utilizado en el entrenamiento de redes neuronales que detecten enfermedades en fotos tomadas en el campo.
- Organizar el material en formato físico del que se dispone en la Escuela Técnica Superior de Ingeniería Agronómica y del Medio Natural.

El repositorio está diseñado para permitir la inserción de nuevos campos de forma dinámica. Todas operaciones que se realizan sobre el repositorio están administradas por una API.

1.1 Tecnologías

En la base de datos se ha utilizado MongoDB. Se ha optado esta tecnología por los siguientes motivos:

- Flexibilidad para almacenar los datos.
- Posibilidad de almacenar los datos en múltiples formatos.
- Compatibilidad con Python. Los documentos se almacenan en formato JSON, que se traduce de forma natural a diccionarios en Python.

Para la API se ha utilizado Python. Las librerías principales que se han empleado han sido Flask y Flask_Pymongo para crear la API y permitir la comunicación con la base de datos, respectivamente. Además, se han empleado otras librerías de forma puntual como apoyo (base64, Image, uuid, pandas, zipfile...).

Dada la flexibilidad que permite mongo, todas las comprobaciones para garantizar la integridad de los datos se han realizado en la propia API.

2. Aplicación móvil

Se ha desarrollado una aplicación móvil para poder poblar la base de datos con imágenes tomadas en el campo.

La aplicación contempla tres roles principales: usuario, validador y administrador.

Los administradores pueden verificar la información de los usuarios y bloquear las cuentas.

El rol de usuario únicamente tiene la opción de subir imágenes al repositorio, además les puede dar una clasificación provisional, estas imágenes se suben como no validadas.

Los validadores tienen varias funciones:

- Comprobar las imágenes sin validar para cambiar las etiquetas provisionales si se considera oportuno y validarlas.
- Añadir nuevos campos a la base de datos. También deben poder añadir nuevas etiquetas a los campos de este tipo, así como modificarlas. Únicamente se permiten eliminar los campos cuando no exista alguna imagen que lo posea.
- Modificar las imágenes validadas. Añadiendo nuevos datos, modificando los que estén permitidos e incluso eliminar imágenes.

La aplicación se ha desarrollado casi en su totalidad. Falta por cambiar algunos aspectos de la lógica de aplicación para que funcione correctamente con la versión más reciente de la API. También falta por migrar el sistema de usuarios junto con la API y finalmente añadir las vistas del rol de validador para permitir que modifiquen los campos, etiquetas y editen la información de las imágenes.

2. API y repositorio

El repositorio ha quedado casi finalizado, faltaría realizar más pruebas para validar su correcto funcionamiento y realizar el despliegue con los cambios que esto pueda conllevar. Al final de este documento se adjunta la documentación a fecha de hoy.

Documentación de la API Flask y Repositorio de Plantas

Contents

1 Descripción

La API descrita a continuación define las operaciones sobre un repositorio destinado a almacenar y etiquetar imágenes de plantas. Además, este repositorio se ha planteado para funcionar conjuntamente con una aplicación externa que permita subir nuevas imágenes tomadas en dispositivos móviles. La estructura y operaciones planteadas para el repositorio permiten añadir campos de forma dinámica.

El repositorio está formado originalmente por las siguientes colecciones:

- `Docs`: Colección en la que se almacenan los datos sobre las imágenes.
- `Campos`: Registro de los campos que pueden contener las imágenes.
- `Etiquetas`: Colección con las etiquetas principales de las imágenes.

Las imágenes están identificadas por los siguientes campos:

- `_id` (str): Cadena generada con uuid3 a partir de la representación en base64 de la imagen al momento de subirla. Como esta cadena es única para cada imagen, no se permiten duplicidades.
- `imagen_rgb` (str): URL que corresponde a la ruta definida en la API para recuperar la imagen en base64.
- `validada` (bool): Este campo identifica las imágenes que han sido verificadas.
- `clase` (int): Identificador de las etiquetas principales.

Al no permitirse la modificación de los campos principales estos no están registrados en la colección de `Campos`. A excepción de la `clase`, que requiere estar registrada para poder manejar la estructura y colección donde se almacenan las etiquetas.

Para estructurar el añadido de campos de forma dinámica se utiliza un sistema de códigos para identificar su tipo:

- **0**: Código exclusivo del campo de Etiquetas principal, se utiliza para prevenir operaciones no deseadas, como la eliminación.
- **1**: Datos numéricos, incluye tanto enteros como números en coma flotante.
- **2**: Datos de texto.
- **3**: Imágenes adicionales.
- **4**: Campo para etiquetas.
- **5**: Valores booleanos.

Un campo registrado está definido por los siguientes datos:

- `_id` (ObjectID): Identificador autogenerado por mongoDB, no se utiliza en las operaciones.
- `nombre` (str): Nombre del campo, funciona como identificador y se corresponde con el nombre que tendrá en la colección de `Docs` al asignarse a una imagen.
- `cod` (int): Código del tipo de dato.

- `coleccion` (str, opcional): Exclusivo para los campos del tipo Etiquetas, identifica en nombre de la colección en la que se almacenarán.
- `campos_etiqueta` (JSON, opcional): Documento JSON embedido que define la estructura que tendrán las etiquetas insertadas. Exclusivo de los campos del tipo Etiquetas. En este documento se debe indicar el nombre de los campos de las etiquetas como clave y el tipo de los campos en formato cadena, los tipos admitidos son `str`, `bool`, `int` o `float`

La información que se muestra en cada función o ruta descrita en este documento es la siguiente:

- Parámetros: estos pueden ser tanto parámetros de entrada de las funciones, de una petición POST o parámetros en la URL.
- Descripción: una explicación breve del funcionamiento de la función.
- Retorno: valor de retorno de la función.
- Errores posibles: un listado con los casos que pueden generar errores. Estos van acompañados de un código de error 400 en el caso de las rutas. Como aclaración, los errores descritos son aquellos casos que son comprobados explícitamente durante la ejecución de la función y que por tanto incluyen descripciones de su causa, NO se describen errores imprevistos (por ejemplo aquellos causados por fallos en la configuración de la base de datos).

2 Inicialización

En esta sección se describen los pasos para incializar y lanzar la API en un entorno local. Se asume que ya se tienen instaladas las dependencias de Python y MongoDB, así como la incialización del servicio de mongo en local. Los pasos seguidos a continuación se han realizado considerando el código base del repositorio, si se desea cambiar el nombre de algún elemento o el puerto donde se aloja el servicio de mongo se requerirán aplicar los cambios pertinentes al código.

1. Acceder al shell de mongo.
2. Crear la base de datos: `use Repositorio_Plantas`
3. Dentro de la base de datos, registrar el campo de etiquetas principales: `db.Campos.insertOne({nombre: 'clase', cod: 0, campos_etiqueta: {clasificacion: 'str', nombre: 'str', nombre_cientifico: 'str'}, coleccion: 'Etiquetas'})`

   Se puede cambiar el nombre del campo, colección o campos de la etiqueta sin que afecte al funcionamiento de la API o se requiera cambiar el código. Es obligatorio que el código de esta etiqueta sea 0 y que no se añada manualmente otro campo con el mismo código ya que podría causar problemas. La API no permite la inserción de campos con código 0. Las eiquetas asociadas a este campo se deben añadir, eliminar o modificar utilizando las funciones de la API, nunca manualmente.
4. Lanzar la API, desde la raíz del proyecto ejecutar: `python3 main.py`

3 Funciones Utilitarias

validar_imagen_base64(cadena_base64)

Parámetros:

- `cadena_base64` (str): Cadena codificada en base64 que representa una imagen.

Descripción: Verifica que la cadena corresponda a una imagen. Se decodifica la imagen en bytes y se intenta cargar utilizando Image. Este proceso se encapsula en un try-except para capturar los posibles errores.

Retorno:
Tupla `(bool, bytes/error)` donde el booleano indica si la imagen es válida. El segundo elemento de la tupla se corresponde a la imagen en bytes si no ha habido error o una cadena con la excepción capturada.

Errores posibles:

- Excepciones lanzadas al decodificar la cadena o intentar la carga de la imagen.

guardar_imagen(imagen_bytes, ruta_salida)

Parámetros:

- `imagen_bytes` (bytes): Imagen en formato bytes.
- `ruta_salida` (str): Ruta relativa donde guardar la imagen.

Descripción: Guarda la imagen en disco en la ruta especificada. Esta ruta ha sido configurada en la API para ser `/imagenes`

Retorno:
Ninguno.

Errores posibles: Ninguno.

comprobar_tipo(tipo_etq, tipo_esperado)

Parámetros:

- `tipo_etq` (type): Tipo real del dato.
- `tipo_esperado` (str): Tipo esperado del dato.

Descripción: Verifica si el tipo real del dato coincide con el tipo esperado. Se contemplan los siguientes tipos: `str`, `bool`, `int` o `float`.

Retorno:
`True`/`False`.

Errores posibles: Ninguno.

4 Rutas API - Campos

GET /recuperar_campos

Descripción: Devuelve todos los campos definidos en el repositorio.

Retorno:
JSON con el formato {`campos=[...]`}.

Errores posibles: Ninguno.

POST /add_campo

Parámetros JSON:

- `nombre` (str): Nombre del campo, deberá coincidir con el nombre del campo que se utilice en las imágenes.
- `cod` (int): Código del tipo de campo.
- `campos_etiqueta` (dict, opcional): Parámetro opcional que debe contener el nombre de los campos de las etiquetas como clave y el tipo como valor. Solo es necesario si se introduce un campo que contenga etiquetas. Solo se contemplan los tipos: `str`, `bool`, `int` o `float`.
- `coleccion` (str, opcional): Parámetro opcional que contiene el nombre de la colección donde se almacenarán las etiquetas. Únicamente es necesario si se introduce un campo que contenga etiquetas.

Descripción: Registra un nuevo campo en el repositorio con la información proporcionada, no se permite la inserción de campos con el código 0.

Retorno:
JSON con confirmación: {`success=True`}.

Errores posibles:

- Nombre del campo proporcionado reservado (`_id`, `imagen_rgb` o `validada`).
- No se ha introducido el parámetro con los tipos de la etiqueta o el nombre de la colección cuando se introducía un campo con etiquetas.
- Se ha especificado algún tipo no válido en el parámetro de los campos de las etiquetas.
- Se ha intentado introducir un campo con el código 0.
- Se ha intentado introducir un campo con un código incorrecto.

Ejemplos:

Listing 1: Ejemplo de JSON para inserción de campo

```
{
  nombre: "campo-prueba",
  cod: 2,
}
```

Listing 2: Ejemplo de JSON para inserción de campo con etiquetas

```
{
  nombre: "campo-prueba-etiquetas",
  cod: 4,
  coleccion: "Coleccion-prueba",
  campos_etiqueta: {
        campo_etiqueta_1: "str",
        campo_etiqueta_2: "int"
  }
}
```

DELETE /eliminar_campos

Parámetros JSON:

- `nombre` (str): Nombre del campo que a eliminar.

Descripción: Elimina un campo del registro si no existe ninguna imagen que contenga un valor para el campo indicado.

Retorno:
JSON con confirmación: {`success=True`}.

Errores posibles:

- Campo en uso.
- Campo no encontrado.
- No se permita eliminar el campo, código 0.

Ejemplos:

Listing 3: Ejemplo de JSON para eliminación de campo

```
{
  nombre: "campo-prueba",
}
```

GET /devolver_campos_etiqueta

Parámetro URL:

- `nombre_campo` (str): Nombre del campo.

Descripción: Devuelve los campos definidos para una etiqueta junto a su tipo.

Retorno:
Lista JSON con subcampos: {`campos_etiqueta=[...]`}.

Errores posibles:

- Etiquetas no encontradas.
- No se han introducido los parámetros URL.

5 Rutas API - Etiquetas

GET /recuperar_etiquetas

Parámetro URL:

- `nombre_campo` (str): Nombre del campo del tipo etiquetas.

Descripción: Devuelve un JSON con todas las etiquetas del campo proporcionado.

Retorno:
Lista JSON con etiquetas: {`etiquetas=[...]`}.

Errores posibles:

- Colección no encontrada.
- El campo proporcionado no es de tipo Etiquetas.
- No se han enviado los parámetros URL.

POST /add_etiqueta

Parámetros JSON:

- `nombre_campo` (str): Nombre del campo al que añadir una etiqueta
- `etiqueta` (JSON): JSON con la etiqueta a insertar.

Descripción: Inserta una nueva etiqueta en la colección correspondiente. La etiqueta insertada debe corresponderse con la estructura proporcionada al insertar el campo, es decir, debe tener el mismo número d campos y tipos. Las etiquetas se añaden con un identificador (`_id`) entero autoincremental.

Retorno:
JSON con confirmación o error.

Errores posibles:

- Campo no encontrado.
- El campo no es del tipo Etiquetas.
- Los campos de la etiqueta no se corresponden con los indicados.
- Los tipos de los datos de la etiqueta o se corresponden con los indicados.

Ejemplos:

Listing 4: Ejemplo de JSON para inserción de etiquetas

```
{
  nombre_campo: "campo-prueba-etiquetas",
  etiqueta: {
        campo_etiqueta_1: "Prueba 1",
        campo_etiqueta_2: 33
  }
}
```

POST /modificar_etiqueta

Parámetros JSON:

- `nombre_campo` (str): Nombre del campo al que modificar una etiqueta
- `etiqueta` (JSON): JSON con la etiqueta, se debe incluir el `_id` de la etiqueta que se reemplaza.

Descripción: Reemplaza una etiqueta existente para el campo proporcionado

Retorno:
JSON con confirmación: {`success=True`}.

Errores posibles:

- Campo no existente.
- Campo no es del tipo Etiqueta.
- La etiqueta proporcionada no sigue la estructura definida en el campo.
- No se ha encontrado ninguna etiqueta con el identificador proporcionado.

Ejemplos:

Listing 5: Ejemplo de JSON para inserción de etiquetas

```
{
  nombre_campo: "campo-prueba-etiquetas",
  etiqueta: {
        _id: 0,
        campo_etiqueta_1: "Prueba 2",
        campo_etiqueta_2: 34
  }
}
```

6 Rutas API - Documentos

POST /add_dato

Parámetros JSON:

- `_id` (str): Identificador del documento al que añadir nuevos datos
- `campos` (JSON): JSON con los datos que añadir al documento.

Descripción: Agrega nuevos datos a un documento existente. Se comprueba la correspondencia de tipos entre los datos proporcionados y los indicados en el registro de campos.

Si el tipo de campo es una imagen se debe proporcionar como una cadena codificada en `base64`. Si el campo contiene etiquetas se inserta el entero correspondiente al identificador de la etiqueta.

Si el identificador del documento es incorrecto no ocurrirá ningún error pero tampoco se alterará la base de datos.

Retorno:
JSON con confirmación: {`success=True`}.

Errores posibles:

- No existe algún campo proporcionado en el registro.
- El tipo de algún campo no es correcto.
- No se ha podido verificar la imagen proporcionada.
- No existe ninguna etiqueta con el identificador proporcionado.

Listing 6: Ejemplo de JSON para inserción de datos en documento

```
{
  _id: "aaaaaaaaaaaaaa",
  campos: {
        campo-1: "Hola",
        campo-2: 321,
        campo-3: False
  }
}
```

POST /eliminar_dato

Parámetros JSON:

- `_id` (str): Identificador de la imagen a la que eliminar un campo.
- `nombre_campo` (str): Nombre del campo a eliminar de la imagen.

Descripción: Elimina un campo asociado a una imagen, el campo a eliminar debe estar registrado. No se permite eliminar el campo de la etiqueta principal.

Retorno:
JSON con confirmación: {`success=True`}.

Errores posibles:

- No se ha encontrado el campo proporcionado.
- Se ha intentado eliminar el campo de la etiqueta principal.
- El identificador proporcionado no se corresponde a ninguna imagen.

Ejemplos:

Listing 7: Ejemplo de JSON para eliminación de datos

```
{
  _id: " aaaaaaaaaaaaaa"
  nombre_campo: "campo-prueba"
}
```

POST /subir_imagen

Parámetros JSON:

- `imagen_b64` (str): Imagen en cadena b64.

- `clase` (int): Identificador de la etiqueta principal.
- `campos_extra` (JSON, opcional): JSON con campos adicionales.

Descripción: Añade una nueva imagen al repositorio y se registra como no validada. El identificador para las imágenes es una cadena generada con *uuid3* a partir del parámetro `imagen_b64`. Si se introducen imágenes duplicadas se sustituirán los datos por la inserción más reciente, sin embargo, si la imagen que se sustituye ha sido validada lanzará un error.

Retorno:
JSON con confirmación: {`success=True`}.

Errores posibles:

- Fallo al procesar la imagen.
- Un campo proporcionado en los campos adicionales no está registrado.
- Error de tipos con un campo proporcionado en los campos adicionales.
- No existe una etiqueta con el identificador proporcionado.
- Se ha intentado subir una imagen validada.

Ejemplos:

Listing 8: Ejemplo de JSON para inserción de imágenes

```
{
  imagen_b64: "(cadena b64)"
  clase: 10
}
```

Listing 9: Ejemplo 2 de JSON para inserción de imágenes

```
{
  imagen_b64: "(cadena b64)"
  clase: 9
  campos_extra: {
        campo-prueba: 1
        campo-prueba-2: "hola"
  }
}
```

POST /subir_imagen_validada

Parámetros JSON:

- `imagen_b64` (str): Imagen en cadena b64.
- `clase` (int): Identificador de la etiqueta principal.
- `campos_extra` (JSON, opcional): JSON con campos adicionales.

Descripción: Funciona de forma análoga a la ruta `/subir_imagen`. Se registran directamente las imágenes como validadas y en caso de duplicados siempre se reemplazan los datos por la versión más reciente.

Retorno:
JSON con confirmación: {`success=True`}.

Errores posibles:

- Fallo al procesar la imagen.
- Un campo proporcionado en los campos adicionales no está registrado.
- Error de tipos con un campo proporcionado en los campos adicionales.
- No existe una etiqueta con el identificador proporcionado.

POST /eliminar_imagen

Parámetros JSON:

- `_id` (str): Identificador de la imagen a eliminar.

Descripción: Elimina el archivo físico de imagen y su entrada en la base de datos. Si el identificador de la imagen es incorrecto no se realiza ninguna operación.

Retorno:
JSON con confirmación: {`success=True`}.

Errores posibles: Ninguno

GET /docs

Parámetros URL:

- `validados` (bool, opcional): Identificador de la imagen a eliminar.

Descripción: Devuelve todas las entradas de la colección de imágenes.

Retorno:
JSON con lista de imágenes: {`docs=[...]`}.

Errores posibles: Ninguno

POST /clasificar

Parámetros JSON:

- `_id` (str): Identificador de la imagen para clasificar.
- `etiqueta` (int): Identificador de la etiqueta principal.

Descripción: Cambia la etiqueta principal de una imagen (puede ser la misma, pero debe proporcionarse el identificador otra vez) y cambia su estado a validada.

Retorno:
JSON con confirmación: {`success=True`}.

Errores posibles:

- No se ha encontrado ningún documento que se corresponda con el identificador.

- No se ha encontrado ninguna etiqueta con el identificador proporcionado.

Listing 10: Ejemplo de JSON para clasificar imágenes

```
{
  _id: "aaaaaaaaaaa"
  etiqueta: 10
}
```

GET /exportar_datos

Descripción: Exporta todos los datos como un archivo zip (imágenes y etiquetas en un archivo CSV).

Retorno:
Archivo .zip descargable.

Errores posibles: Ninguno

Estructura de los archivos:

Listing 11: Estructura del zip

```
/
--/imagenes
----ejemplo-1.jpg
----ejemplo-2.png
----...
--etiquetas.csv
```

7 Otras Rutas

GET /imagen_base64/<nombre>

Parámetro URL:

- `nombre` (str): Nombre del archivo de imagen, se debe incluir la extensión.

Descripción: Devuelve la imagen codificada en base64.

Retorno:
Cadena base64 de la imagen.

Errores posibles: Ninguno

GET /servir_n_archivos

Parámetro URL:

- `inicio` (int): Índice de la base de datos a partir del que se devolverán los archivos.
- `n_archivos` (int): Número de archivos que devolver de la base de datos.

Descripción: Devuelve una cantidad N de imágenes validadas de la base de datos desde la posición indicada. Esta ruta se utiliza para iterar sobre las imágenes por una aplicación externa.

Retorno:
JSON con lista de imágenes: {`docs=[...]`}.

Errores posibles: Ninguno

GET /servir_n_archivos_sin_validar

Parámetro URL:

- `inicio` (int): Índice de la base de datos a partir del que se devolverán los archivos.
- `n_archivos` (int): Número de archivos que devolver de la base de datos.

Descripción: Devuelve una cantidad N de imágenes sin validar de la base de datos desde la posición indicada. Esta ruta se utiliza para iterar sobre las imágenes por una aplicación externa.

Retorno:
JSON con lista de imágenes: {`docs=[...]`}.

Errores posibles: Ninguno