

Aplicación CRUD con Python, Flask y MongoDB Atlas

Jorge Alejandro Mejía, Licet Sofia Ladino, Emmanuel Duran, Juan Carlos Guzman

Universidad Tecnológica de Pereira

7 de mayo de 2024

Introducción

En esta documentación se detalla el desarrollo de una aplicación CRUD (Crear, Leer, Actualizar, Eliminar) utilizando Python con el framework Flask y MongoDB Atlas como base de datos en la nube. Esta aplicación permite gestionar una colección de productos a través de una interfaz RESTful.

1. Aplicación Web

Aplicación principal

1.1. Función 'home()'

1. Función 'home()' es responsable de renderizar la página de inicio de la aplicación, mostrando todos los productos almacenados en la base de datos.

- Esta función recupera todos los productos de la base de datos y los pasa a la plantilla HTML 'index.html' para ser renderizados en el navegador.

- Retorna: *item,emplatelaplantillarenderizada*

```
1 #Rutas de la aplicaci n
2 @app.route('/')
3 def home():
4     products = db['products']
5     productsReceived = products.find()
6     return render_template('index.html', products=productsReceived)
```

1.2. Función addProduct()

Esta función recibe los datos del nuevo producto (nombre, precio y cantidad) desde el formulario web y los valida.

- Si el producto no existe previamente en la base de datos, se crea un nuevo documento de producto y se inserta en la colección correspondiente.
- En caso de éxito, devuelve un mensaje JSON indicando que el producto ha sido guardado.

```

1  @app.route('/products', methods=['POST'])
2  def addProduct():
3      products = db['products']
4      name = request.form['name']
5      price = request.form['price']
6      quantity = request.form['quantity']
7
8      product = products.find_one({'name': name})
9      if product:
10         # Si existe, devuelve un mensaje de error
11         return jsonify({'message': 'Ya existe un producto con ese
nombre.', 'success': False})

```

1.3. Función delete(product name)

La función delete() permite eliminar un producto de la base de datos mediante una solicitud DELETE.

- Esta función recibe el nombre del producto a eliminar como parámetro de la URL.
- Busca y elimina el producto correspondiente de la colección de productos en la base de datos.
- Luego redirige al usuario a la página de inicio.

```

1  @app.route('/delete/<string:product_name>')
2  def delete(product_name):
3      products = db['products']
4      products.delete_one({'name' : product_name})
5      return redirect(url_for('home'))

```

2. Data Base

Utiliza el paquete pymongo para consumir la propiedad mongo cliente y conectarse a la base de datos que se tiene de forma remota

```

1  from pymongo import MongoClient
2  import certifi

```

- from pymongo import MongoClient: Importa la clase MongoClient de la biblioteca pymongo. Esta clase se utiliza para establecer la conexión con la base de datos MongoDB.

- `import certifi`: Importa el módulo `certifi`, que proporciona una forma de acceder a la ubicación del archivo de certificado CA.

```

1  def dbConnection():
2  try:
3      client = MongoClient(MONGO_URI, tlsCAFile=ca)
4      db = client["dbb_products_app"]
5  except ConnectionError:
6      print('Error de conexión con la bdd')
7  return db

```

- `def dbConnection()`:: Define una función llamada `dbConnection` que establece una conexión con la base de datos MongoDB Atlas.
- `try`:: Inicia un bloque `try-except` para manejar posibles errores de conexión.
- `client = MongoClient (MONGO URI, tlsCAFile=ca)`: Crea una instancia de `MongoClient` utilizando la URI de conexión y el archivo de certificado CA.
- `db = client ["dbb products app"]`: Selecciona la base de datos llamada "dbb products app" y la asigna a la variable `db`.
- `except ConnectionError`:: Captura cualquier excepción de conexión y muestra un mensaje de error si ocurre un problema al conectar con la base de datos.
- `return db`: Devuelve la conexión a la base de datos "dbb products app" para su uso en otras partes del código.

3. Producto

Contiene una clase `Producto` la cual describe la estructura del documento de la Base de Datos

```

1  class Product:
2  def __init__(self, name, price, quantity):
3      self.name = name
4      self.price = price
5      self.quantity = quantity

```

- Representa un producto con nombre, precio y cantidad.
Esta clase proporciona una estructura para representar productos en la aplicación. Cada instancia de la clase 'Product' tiene atributos para almacenar el nombre, precio y cantidad del producto.
Attributes: `name (str)`: El nombre del producto. `price (float)`: El precio del producto. `quantity (int)`: La cantidad disponible del producto.

```

1     def toDBCollection(self):
2         return{
3             'name': self.name,
4             'price': self.price,
5             'quantity': self.quantity
6         }

```

- Convierte el objeto Producto en un diccionario para su almacenamiento en la base de datos.

Returns: dict: Un diccionario que representa el producto con las claves 'name', 'price' y 'quantity'.

4. Desarrollando un CRUD de Productos

El CRUD es una operación básica en la mayoría de las aplicaciones web, ya que permite a los usuarios realizar operaciones fundamentales sobre los datos. En este caso, desarrollaremos un CRUD para gestionar productos, lo que nos permitirá agregar, ver, editar y eliminar productos de una base de datos.

4.1. Paso 1: Configuración de la conexión a la Base de Datos

```

1 from flask import Flask, jsonify, request
2 from pymongo import MongoClient
3 import certifi
4 import uuid
5
6 app = Flask(__name__)
7
8 uri = 'mongodb+srv://<usuario>:<contrase a>@<cluster>.mongodb.net/?
9
10 retryWrites=true&w=majority&appName=Cluster0'
11
12 client = MongoClient(uri)
13
14 # Verificación de la conexión exitosa
15 try:
16     client.admin.command('ping')
17     print("Conexión exitosa con MongoDB!")
18 except Exception as e:
19     print(e)
20
21 db = client.get_database("miInventario")
22 collection = db.productos

```

4.2. Paso 2: Obtener todos los productos

```

1 @app.route('/allProductos', methods=['GET'])
2 def get_productos():
3     productos = list(collection.find({}, {'_id': 0}))
4     return jsonify(productos)

```

4.3. Paso 3: Obtener un producto por su nombre

```

1 @app.route('/producto', methods=['GET'])
2 def get_producto_por_nombre():
3     nombre = request.args.get('nombre')
4     if nombre:
5         producto = collection.find_one({'nombre': nombre}, {'_id': 0})
6         if producto:
7             return jsonify(producto)
8         return jsonify({"error": "Producto no encontrado"}), 404
9     return jsonify({"error": "Se requiere un nombre de producto"}), 400

```

4.4. Paso 4: Crear un nuevo producto

```

1 @app.route('/newProducto', methods=['POST'])
2 def crear_producto():
3     nuevo_producto = request.get_json()
4     nuevo_producto['_id'] = str(uuid.uuid4())
5     resultado = collection.insert_one(nuevo_producto)
6     return jsonify(nuevo_producto), 201

```

4.5. Paso 5: Actualizar un producto

```

1 @app.route('/editProducto/<string:nombre>', methods=['PUT'])
2 def actualizar_producto(nombre):
3     producto_data = request.get_json()
4     resultado = collection.update_one({'nombre': nombre}, {'$set':
    producto_data})
5     if resultado.modified_count > 0:
6         return jsonify(producto_data)
7     return jsonify({"error": "Producto no encontrado"}), 404

```

4.6. Paso 6: Eliminar un Producto

```

1 @app.route('/deleteProducto/<string:nombre>', methods=['DELETE'])
2 def eliminar_producto(nombre):
3     resultado = collection.delete_one({'nombre': nombre})
4     if resultado.deleted_count > 0:
5         return jsonify({"mensaje": f"El producto {nombre} ha sido
    eliminado"})
6     return jsonify({"error": "Producto no encontrado"}), 404

```

4.7. Ejecutar la aplicación

```
1 if __name__ == '__main__':  
2     app.run(debug=True)
```

5. Conclusión

En este artículo, hemos desarrollado un CRUD de productos utilizando Python, Flask y MongoDB Atlas. Este sistema nos permite realizar operaciones básicas sobre una lista de productos almacenados en una base de datos MongoDB en la nube. Además, hemos explorado cada paso del proceso de desarrollo, desde la configuración de la conexión a la base de datos hasta la implementación de las operaciones CRUD.