Aplicación CRUD con Python, Flask y MongoDB Atlas

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

Universidad Tecnológica de Pereira

10 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. Marco teórico

1.1. MongoDB Atlas

MongoDB Atlas es un servicio de base de datos en la nube que ofrece MongoDB, una base de datos NoSQL altamente escalable y flexible. MongoDB es una base de datos orientada a documentos, lo que significa que almacena datos en documentos JSON (JavaScript Object Notation), lo que permite una estructura de datos dinámica y adaptable.

MongoDB Atlas se utiliza para gestionar bases de datos MongoDB en la nube de manera eficiente y segura. Algunas de las características y funcionalidades clave de MongoDB Atlas incluyen su disponibilidad global, escalabilidad, respaldo y recuperacion, seguridad y su capacidad de integracion con otras herramientas.

1.2. Flask

Flask es un framework de desarrollo web en Python conocido por su minimalismo y facilidad de uso. Al ser un microframework, ofrece herramientas básicas sin imponer una estructura rígida, lo que permite a los desarrolladores tener un mayor control sobre el diseño de sus aplicaciones. Con extensiones disponibles para agregar funcionalidades como autenticación y manejo de bases de datos, Flask utiliza Jinja2 para plantillas y proporciona un enrutador simple para asociar URLs a funciones específicas. Es ideal para proyectos pequeños a medianos donde la agilidad y la flexibilidad son clave, con un servidor de desarrollo integrado para pruebas locales durante el proceso de desarrollo.

1.3. RESTFUL Api

Una RESTFUL API es una interfaz que dos sistemas de computación utilizan para intercambiar información de manera segura a través de internet, por medio de un estilo de arquitectura de software que se basa en el protocolo HTTP y los principios del modelo de transferencia de estado representacional (REST). Se utilizan comúnmente en el desarrollo de aplicaciones web y móviles, ya que, por lo general, las aplicaciones para empresas deben comunicarse con otras aplicaciones internas o de terceros para llevar a cabo varias tareas.

1.4. Postman

Es una herramienta de colaboración y desarrollo que permite a los desarrolladore interactuar y probar el funcionameitno de servicios web y aplicaciones. Este entorno ofrece una GUI que facilita el envió de solicitudes HTTP y HTTPS a una API y a gestionar las respustas recibidas. Proporciona características como la capacidad de guardar solicitudes y respuestas para referencia futura, automatización de pruebas, colaboración en equipo y documentación de APIs.

2. Aplicación Web

Aplicación principal

2.1. 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$_t emplatelaplantillarenderizada$

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

2.2. 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.

```
    @app.route('/products', methods=['POST'])
def addProduct():
    products = db['products']
    name = request.form['name']
    price = request.form['price']
    quantity = request.form['quantity']

    product = products.find_one({'name': name})
    if product:
        # Si existe, devuelve un mensaje de error
        return jsonify({'message': 'Ya existe un producto con ese
   nombre.', 'success': False})
```

2.3. 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.

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

3. 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

```
    from pymongo import MongoClient
    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.

```
def dbConnection():
try:
    client = MongoClient(MONGO_URI, tlsCAFile=ca)
    db = client["dbb_products_app"]
except ConnectionError:
    print('Error de conexi n con la bdd')
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 appz 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.

4. Producto

Contiene una clase producto la cuál describe la estructura del documento de la Base de Datos

```
class Product:
def __init__(self, name, price, quantity):
    self.name = name
    self.price = price
    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.

```
def toDBCollection(self):
    return{
        'name': self.name,
        'price': self.price,
        'quantity': self.quantity
    }
```

- 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'. "

5. 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.

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

```
from flask import Flask, jsonify, request
from pymongo import MongoClient
import certifi
import uuid

app = Flask(__name__)

uri  = 'mongodb+srv://<usuario>:<contrase a>@<cluster>.mongodb.net/?

retryWrites=true&w=majority&appName=Cluster0'

client = MongoClient(uri)

# Verificaci n de la conexi n exitosa
try:
    client.admin.command('ping')
    print("  Conexin   exitosa con MongoDB!")
except Exception as e:
    print(e)

db = client.get_database("miInventario")
collection = db.productos
```

5.2. Paso 2: Obtener todos los productos

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

5.3. Paso 3: Obtener un producto por su nombre

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

5.4. Paso 4: Crear un nuevo producto

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

5.5. Paso 5: Actualizar un producto

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

5.6. Paso 6: Eliminar un Producto

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

5.7. Ejecutar la aplicación

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

6. 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.