UNIVERSIDAD TECNOLÓGICA DE PEREIRA

Facultad de Ingenierías

Sistemas Distribuidos

CRUD - API RESTful - MongoDB

Alumnos:
Brandon Castaño G.
Juan P. Cataño O.
Gerwin Lambraño T.
Sebastián Velásquez M.

Profesor:
Felipe Gutierrez Isaza

30 de abril de 2024

Índice

1. Introducción

1.1. ¿Qué es API RESTful?

API RESTful (Representational State Transfer) es un estilo arquitectónico para diseñar servicios web basado en el protocolo HTTP. La intención es crear una interfaz de programación de aplicaciones (API) que sea coherente, fácil de entender y que siga los principios REST.

A continuación se describen algunos de los principios arquitectónico de REST de acuerdo con AWS:

- **Interfaz uniforme** La interfaz uniforme es esencial en el diseño de servicios web RESTful. Esto implica que el servidor transfiere información en un formato estándar, conocido como representación en REST. Esta representación puede diferir de la estructura interna del recurso en el servidor. Por ejemplo, los datos pueden almacenarse en formato de texto en el servidor, pero enviarse como HTML al cliente.
- **Tecnología sin estado** En la arquitectura REST, la tecnología sin estado implica que el servidor no guarda ningún estado de la sesión del cliente entre solicitudes. Cada solicitud del cliente se procesa de forma independiente, sin referencia a las solicitudes anteriores. Esto significa que el servidor puede entender y responder completamente a cada solicitud sin depender del contexto de las solicitudes anteriores. Esta característica permite una mayor escalabilidad y confiabilidad en las aplicaciones RESTful.
- **Código bajo demanda** Al completar un formulario en un sitio web, el servidor puede enviar código al navegador del cliente para resaltar errores inmediatamente, como un número de teléfono incorrecto. Esto mejora la experiencia del usuario al proporcionar retroalimentación instantánea durante la interacción.

Las API RESTful requieren que las solicitudes contengan los siguientes componentes:

1. **Identificador único de recursos:** Los servidores proporcionan el recurso deseado cuando el usuario con ayuda de una URL accede a estos.
2. **Métodos:** los métodos en una API RESTful son acciones que se pueden realizar sobre los recursos, los métodos mas utilizados son:
   - **Get** Se usa para recuperar datos de un recurso específico en el servidor. No modifica el estado del servidor ni de los recursos, solo obtiene información.
   - **Post** Se utiliza para enviar datos al servidor para crear un nuevo recurso. Los datos enviados pueden incluir información para ser procesada por el servidor, que puede luego responder con un recurso creado.
   - **Put** Se utiliza para actualizar un recurso existente en el servidor. El cliente envía los datos actualizados al servidor y especifica la ubicación del recurso que debe ser modificado.
   - **Delete** Se usa para eliminar un recurso específico en el servidor. El cliente envía una solicitud DELETE junto con la ubicación del recurso que debe ser eliminado.

3. **Datos:** Las solicitudes de una API RESTful pueden necesitar datos para que métodos como POST, PUT y otros funcionen correctamente, por lo general se brindan en formato JSON

1.2. Descripción del proyecto

En este proyecto, se desea desarrollar una API que cumpla con los principios de RESTful. Para ello, se ha elegido Flask como framework y Postman para verificar su funcionamiento. La API debe gestionar un inventario, en este caso, se opta por almacenar toda la información necesaria sobre libros, simulando el inventario de una biblioteca. Además, se planea utilizar una base de datos no relacional. Se observará su comportamiento durante la creación de la base de datos, incluyendo sus colecciones y réplicas. Para este propósito, se ha seleccionado MongoDB Atlas.

1.3. Herramientas utilizadas

1.3.1. Postman

Postman es una herramienta popular utilizada por desarrolladores para probar, desarrollar y documentar APIs de forma eficiente. Proporciona una interfaz gráfica de usuario que facilita la creación y envío de solicitudes HTTP a servidores web y la visualización de las respuestas correspondientes.

En el proyecto se usa para probar el funcionamiento de la API y sus diferentes métodos, ya que con ayuda de la URL proporcionada por flask (se describe mas adelante), es posible enviar peticiones y observar el comportamiento.

Figura 1: Logo Postman

1.3.2. MongoDB

MongoDB Atlas es un servicio de base de datos en la nube completamente administrado por MongoDB, Inc. que permite a los desarrolladores implementar, escalar y administrar bases de datos MongoDB o no relacionales de manera sencilla y eficiente sin preocuparse por la infraestructura subyacente.

En el proyecto se utiliza para almacenar la base de datos donde se encuentran las colecciones y información de los registros, se conecta a ella por medio de una URI (Se describe en la sección de desarrollo).

Figura 2: Logo MongoDB

1.3.3. Flask

Flask es un popular framework de desarrollo web en Python que permite a los desarrolladores crear aplicaciones web de forma rápida y sencilla. Es minimalista y flexible, lo que significa que proporciona las herramientas necesarias para construir aplicaciones web sin imponer una estructura rígida.

En el proyecto se utiliza para obtener las rutas de los métodos posteriormente consumidos en postman, en un desarrollo futuro serían las rutas a consumir por la interfaz de usuario para realizar las peticiones al servidor.

Figura 3: Logo Flask

2. Desarrollo

En esta sección, se presentará el código de forma segmentada para facilitar su explicación. Si se desea utilizar, se recomienda acceder al repositorio GitHub del proyecto para visualizarlo de manera más clara: GitHub del proyecto.

```
import os
from bson import json_util, ObjectId
from bson.errors import InvalidId

from flask import Flask, jsonify, request, Response
from dotenv import load_dotenv
from flask_pymongo import PyMongo
```

Las librerías utilizadas incluyen Flask para la creación de la aplicación web, PyMongo para interactuar con la base de datos MongoDB, y otras librerías como os, jsonify, request, Response y dotenv para diversas funcionalidades relacionadas con la manipulación de archivos, la manipulación de solicitudes HTTP y la gestión de variables de entorno.

Listing 1: Librerias o dependencias

```
load_dotenv()

app = Flask(__name__)
app.config['MONGO_URI'] = os.getenv('MONGO_URI')

mongo = PyMongo()
mongo.init_app(app)
```

Esta parte del código carga las variables de entorno desde un archivo .env, configura la URI de la base de datos MongoDB la cual es proporcionada por MongoDB Atlas y donde tiene la siguiente estructura:

```
MONGO_URI="mongodb+srv://USER:PASS@cluster0.zqi8aps.mongodb.net
```

donde 'USER' es un usuario con acceso a la base de datos previamente configurado y 'PASS' su correspondiente contraseña. En adición se establece la conexión con la base de datos MongoDB a través de la aplicación Flask utilizando PyMongo.

Listing 2: Inicialización

```python
@app.route('/books', methods=['POST'])
def create_user():
    user_data = request.get_json()
    user_keys = user_data.keys()
    if {'title', 'author', 'genre', 'amount',
    'price', 'release_date', 'editorial'} == set(user_keys):
        new_user = mongo.db.books.insert_one({
            'title': user_data.get('title', None),
            'author': user_data.get('author', None),
            'genre': user_data.get('genre', None),
            'amount': user_data.get('amount', None),
            'price': user_data.get('price', None),
            'release_date': user_data.get('release_date', None),
            'editorial': user_data.get('editorial', None),
        })
        response = {'_id': str(new_user.inserted_id)}
        response.update(user_data)
        return jsonify(response)
    else:
        return jsonify({'message': 'Missing some fields
        in payload or payload is invalid'}), 400
```

Formato JSON necesario para su funcionamiento:

```json
{
    "title": "El amor en los tiempos del cólera",
    "author": "Gabriel García Márquez",
    "genre": "Realismo mágico",
    "amount": 8,
    "price": 12.50,
    "release_date": "1967-05-30",
    "editorial": "Sudamericana"
}
```

Esta función maneja solicitudes POST en la ruta /books. Verifica si se proporcionan todas las claves requeridas en los datos JSON del usuario y, si es así, inserta un nuevo documento en la colección books de la base de datos MongoDB. Retorna una respuesta JSON con el ID del nuevo documento si la inserción es exitosa, o un mensaje de error si faltan campos o el formato del payload es inválido.

Listing 3: Agregar un nuevo libro, método POST

```
@app.route('/books', methods=['GET'])
def get_all_users():
    users = mongo.db.books.find()
    return Response(json_util.dumps(users), mimetype='application/json')
```

Esta función maneja solicitudes GET en la ruta /books. Recupera todos los documentos de la colección books de la base de datos MongoDB y devuelve una respuesta HTTP con los documentos serializados como JSON.

Listing 4: Obtener lista de todos los libros, método GET

```
@app.route('/books/<identifier>', methods=['GET'])
def get_user(identifier):
    try:
        user_id = ObjectId(identifier)
    except InvalidId:
        return jsonify({'message': 'User not found'}), 404

    user = mongo.db.books.find_one({'_id': user_id})
    if user:
        return Response(json_util.dumps(user), mimetype='application/json')
    else:
        return jsonify({'message': 'User not found'}), 404
```

Esta función maneja solicitudes GET en la ruta /books/<identifier>. Intenta convertir el identificador de usuario en un objeto ObjectId. Si el identificador no es válido, devuelve un mensaje de error con un código de estado HTTP 404 (Not Found). Luego, busca un documento en la colección books de la base de datos MongoDB que coincida con el identificador proporcionado. Si encuentra el documento, devuelve una respuesta HTTP con el documento serializado como JSON. Si no encuentra el documento, devuelve un mensaje de error con un código de estado HTTP 404.

Listing 5: Obtener un solo libro, método GET

```python
@app.route('/books/<identifier>', methods=['PUT'])
def update_user(identifier):
    try:
        user_id = ObjectId(identifier)
    except InvalidId:
        return jsonify({'message': 'User not found'}), 404

    data = request.get_json()
    if not data:
        return jsonify({'message': 'Invalid payload: payload is empty'}), 400

    response = mongo.db.books.update_one({'_id': user_id}, {'$set': data})

    if response.matched_count == 0:
        return jsonify({'message': 'User not found'}), 404
    elif response.modified_count > 0:
        return jsonify({'message': 'User updated successfully'})
    else:
        return jsonify({'message': 'Nothing to update'})
```

Formato JSON necesario para su funcionamiento:

```
{
    "KEY" : "VALUE"
    "price" : 10.00
}
```

Esta función maneja solicitudes PUT en la ruta /books/<identifier>. Primero, intenta convertir el identificador de usuario en un objeto ObjectId. Si el identificador no es válido, devuelve un mensaje de error con un código de estado HTTP 404 (Not Found). Luego, obtiene los datos JSON de la solicitud. Si los datos están vacíos, devuelve un mensaje de error con un código de estado HTTP 400 (Bad Request). A continuación, actualiza el documento en la colección books de la base de datos MongoDB que coincide con el identificador proporcionado con los datos proporcionados en la solicitud. Si el documento no se encuentra, devuelve un mensaje de error con un código de estado HTTP 404. Si el documento se actualiza correctamente, devuelve un mensaje de éxito. Si no hay nada que actualizar, devuelve un mensaje indicando que no hay cambios.

Listing 6: Actualizar información de un libro, método PUT

```python
@app.route('/books/<identifier>', methods=['DELETE'])
def delete_user(identifier):
    try:
        user_id = ObjectId(identifier)
    except InvalidId:
        return jsonify({'message': 'User not found'}), 404

    response = mongo.db.books.delete_one({'_id': user_id})

    if response.deleted_count > 0:
        return jsonify({'message': 'User deleted successfully'})
    else:
        return jsonify({'message': 'User not found'}), 404
```

Esta función maneja solicitudes DELETE en la ruta /books/<identifier>. Primero, intenta convertir el identificador de usuario en un objeto ObjectId. Si el identificador no es válido, devuelve un mensaje de error con un código de estado HTTP 404 (Not Found). Luego, elimina el documento de la colección books de la base de datos MongoDB que coincide con el identificador proporcionado. Si se elimina el documento con éxito, devuelve un mensaje de éxito. Si no se encuentra el documento, devuelve un mensaje de error con un código de estado HTTP 404.

Listing 7: Eliminar información de un libro, método DELETE

```json
{
    "title": "El amor en los tiempos del cólera",
    "author": "Gabriel García Márquez",
    "genre": "Realismo mágico",
    "amount": 8,
    "price": 12.50,
    "release_date": "1967-05-30",
    "editorial": "Sudamericana"
}
```

Este fragmento de JSON representa la información que se almacena en la base de datos

Listing 8: Formato JSON

3. Conclusiones

- **Base de Datos** En cuanto a la base de datos MongoDB permitió desarrollar el proyecto de forma relativamente fácil, pues su interfaz amigable y bien explicada permite crear y entender su funcionamiento de manera sencilla.

  Se creó una base de datos 'maindb' y la colección 'books', la cual previamente se explicaron los datos allí almacenados.

  En cuanto a la cantidad de datos, esta base de datos ofrece un servicio gratuito que cuenta con 512MB de almacenamiento, y es soportado por AWS en la región de Virginia, Estados Unidos.

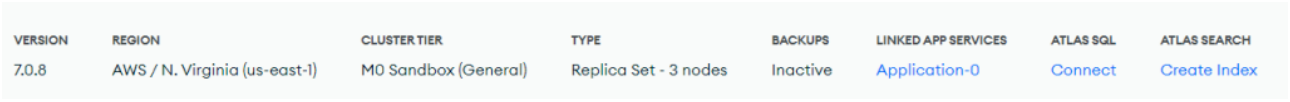

Figura 4: Información almacenamiento

Además, se cuenta con 2 réplicas a las cuales no se tiene acceso. Como su nombre lo indica, estas dependen únicamente de la principal y su deber es replicar lo que ocurre en esta para contar con backups en caso de ser necesario. Es posible que en versiones de pago se cuente con acceso a ellas, pero sería algo innecesario.

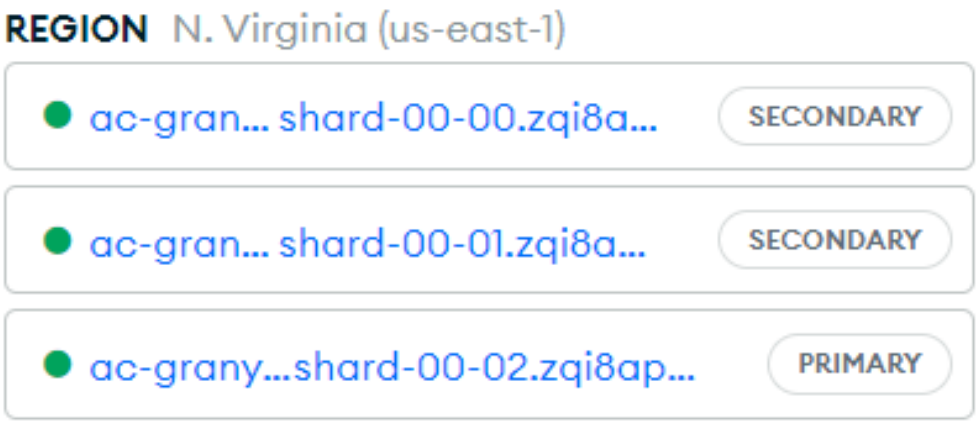

Figura 5: Información replicas

- **Post**
- **Put**
- **Delete**

4. Bibliografía

Referencias

[1] ¿Qué es una API de RESTful? - Explicación de API de RESTful - AWS. (s.f.). Amazon Web Services, Inc. https://aws.amazon.com/es/what-is/restful-api/