Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
Introducción a la programación y computación 2, 2do. Semestre 2022.

PROYECTO 3 – PROGRAMA ADMINISTRACIÓN DE RECURSOS EN LA NUBE – "TECNOLOGÍAS CHAPINAS S.A"

202110206 – Julio Alejandro Zaldaña Ríos

Resumen

En el presente proyecto, se da a conocer una forma para manejar y administrar recursos, instancias, configuraciones y más que todo servicios de infraestructura en la nube. Donde se hace un uso primordial del framework Flask del lenguaje de programación Python. Donde gracias a ella, se desarrolló la parte del Backend, elaborando diferentes peticiones, desarrollando una API etc. Se instaló de igual manera, algunas configuraciones con el framework Django, para el Frontend.

Palabras clave

Python

Flask

Backend

Frontend

Django

Abstract

In the present project, it is shown a way to manage and administrate resources, instances and configurations, and overall infrastructure services using the Cloud. Where it was used a framework in the programming language Python, named Flask. By using it, it helped to create the major parts of the Backend, and also some requests creating the API. By the way, the framework Django was also installed for the Frontend.

Keywords

Python

Flask

Backend

Frontend

Django

Introducción

A continuación, se presentará la parte del backend de la arquitectura de trabajo, propuesta por "Tecnologías Chapinas S.A", con el fin de crear una aplicación para crear, recursos y servicios de infraestructura de la nube.

Desarrollo del tema

Se trabaja para poder crear instancias definidas por los clientes que podrán registrarse en los servicios del programa.

1. **Clientes:**

Endpoint: /crearCliente

El objetivo es que el cliente pueda registrarse, y obtener un usuario, una clave y tener todos sus datos. De la misma forma este pueda tener la oportunidad de crear instancias e infraestructuras en la nube.

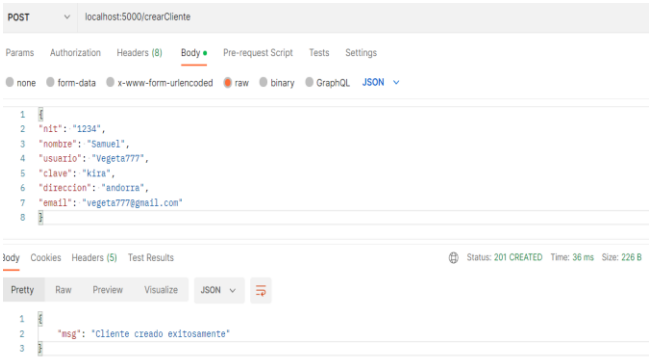


Figura 1. Método para crear cliente, en Postman

Fuente: elaboración propia, 2022

2. **Instancias:**

Endpoint: /crearInstancia

El objetivo de las instancias es definir el estado de esta, si está en vigencia o ya está cancelado. Son importantes para definir las configuraciones de las infraestructuras.

3. **Recursos:**

Endpoint: /crearRecurso

El objetivo de los recursos es definir características de las distintas herramientas y servicios que se pueden utilizar en la aplicación

4. **Categorías:**

Endpoint: /crearCategoria

Se pueden crear categorías para las distintas configuraciones que se pueden crear en la aplicación.

5. **Configuraciones:**

Endpoint: /crearConfiguracion

Las configuraciones detallan y definen la cantidad de recursos que se estarán manejando en las instancias.

6. Facturas:

Endpoint: /generarFactura

Se podrá generar facturas, y después se podrá crear y generar un reporte con un código único para la factura, con el nit de los clientes, y el monto a pagar.

7. Carga de archivos

Endpoint: /cargarArchivo

Como funcionalidad importante de la aplicación se puede cargar un archivo XML en la base de datos.

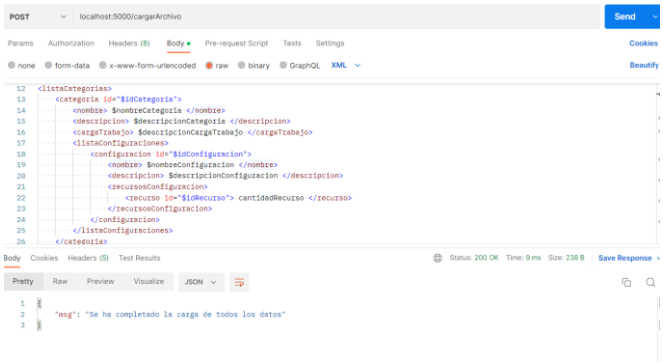


Figura 2. Carga de archivos XML, usando Postman

Fuente: elaboración propia, 2022

8. Consulta de Datos:

Para la consulta de datos, se utilizó el endpoint y los diferentes objetos que se requiere para obtener los datos guardados en la base de datos.

Endpoint: /consultarDatos/objeto

Entre los objetos se tienen a los:

- Recursos
- Categorias
- Configuraciones
- Clientes
- Instancias

Estructuras de métodos:

```
@app.route('/crearCliente', methods = ['POST'])
def crearCliente():
    body = request.get_json()
    try:
        if("nit" in body and "nombre" in body and "usuario" in  body and "clave" in body and "direccion" in body and "email" in body):
            cliente = Cliente(body["nit"], body["nombre"],body["usuario"],body["clave"],body["direccion"],body["email"])
            if(gestor.agregar_cliente(cliente)):
                return{'msg': "Cliente creado exitosamente"}, 201 #created
            else:
                return{'msg': 'El NIT ya se encuentra registrado.'}, 406 #not acceptable
        else:
            return{'msg': 'Faltan campos por rellenar'},400 #bad request
    except:
        return {'msg': 'Ocurrió un error en el servidor'},500
```

Figura 3. Estructura de un método POST

Fuente: elaboración propia, 2022

```
@app.route('/consultarDatos/cliente/<nit>', methods=['GET'])
def ver(nit):
    try:
        cliente = gestor.obtener_clientes(nit)
        if (cliente != None):
            return jsonify(cliente.getData()),200
        else:
            return {'msg': 'No se encontró el cliente'}, 404
    except:
        return {'msg': 'Ocurrió un error en el servidor'}, 500
```

Figura 4. Estructura de un método GET

Fuente: elaboración propia, 2022

```
#CARGA DE ARCHIVO DE CONSUMOS
@app.route('/cargarConsumos', methods = ['POST'])
def cargarConsumos():
    xml = request.data.decode('utf-8')
    raiz = ET.XML(xml)
    for consumo in raiz:
        nitcliente =  consumo.attrib['nitCliente']
        idinstancia = consumo.attrib['idInstancia']
        for params in consumo:
            if(params.tag == "tiempo"):
                tiempo = params.text.strip()
            elif(params.tag == "fechaHora"):
                fechaHora = params.text.strip()
        nuevoconsumo = Consumo(nitcliente, idinstancia, tiempo, fechaHora
        gestor.crear_consumo(nuevoconsumo)
    return {'msg': 'Se ha completado la carga de los datos'}, 200
```

Figura 5. Estructura para cargar datos de un archivo XML

Fuente: elaboración propia, 2022

Documentación Postman:

Para un resumen de la mayoría de los métodos y peticiones más a fondo, se creó una pequeña recopilación con sus funcionamientos:

https://documenter.getpostman.com/view/20302235/2s8YRqjVoB

Para trabajar con el framework de Flask, se trabajó de la siguiente forma:

- **Clase gestor:** Funciona como un gestor de base de datos y de funcionamientos.
- **Archivo main.py**, servirá como la app principal donde inicializa el servidor; en la dirección: ***localhost:5000***
- Se puede notar que se utilizó un concepto de models-routes para cada clase de objeto: Categoría, cliente, configuración, instancia, recurso etc.

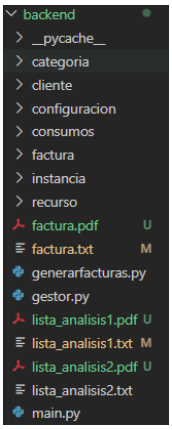
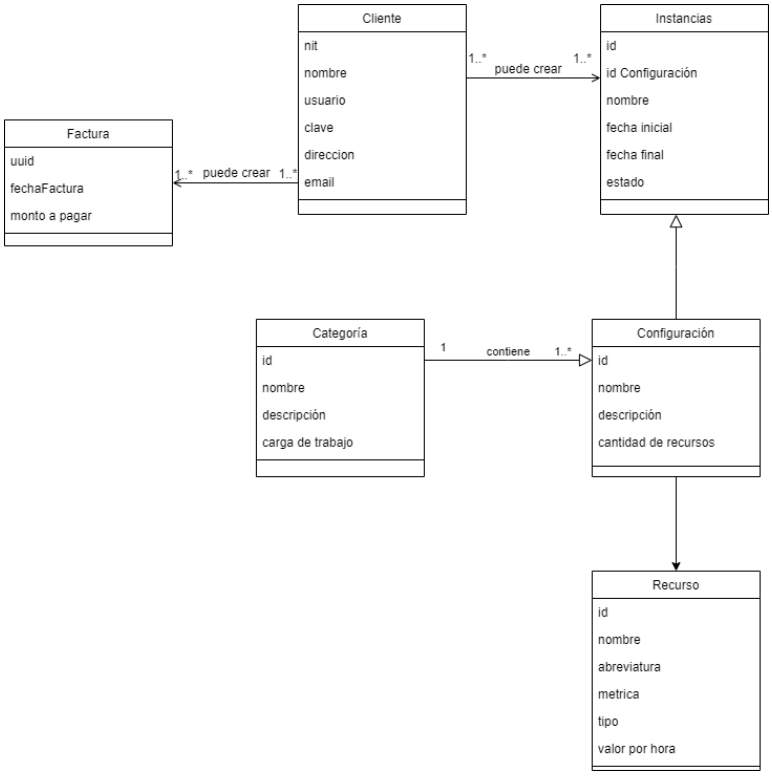


Figura 5. Ejemplo de administración de archivos para el funcionamiento del proyecto

Fuente: elaboración propia, 2022

Diagramas de Clases



Conclusiones

Es muy útil saber cómo funciona y tener conocimientos sobre como se puede manejar el backend, ya que es relevante saber el detrás de cada servicio o estructura de una página o aplicación web, ya que tiene mucho uso y es tendencia en la actualidad.

Al igual que es importante reconocer el concepto y funcionamiento de los diferentes frameworks del lenguaje de programación de Python, ya que tienen muchas aplicaciones y son muy útiles para crear aplicaciones web.

Referencias bibliográficas

Documentacion Flask, (27, octubre,2022).
User's Guide
https://flask.palletsprojects.com/en/2.2.x/