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 2025

Proyecto 3

Software Solution GT

201503846 – Mario Noel Giron Maldonado

Resumen

En este proyecto se planteó para desarrollar una API con la que modificar, consultar y facturar un archivo XML, cada etiqueta del xml representa los servicios que puede ofreces una empresa a sus clientes, con esto se busca poder facilitar la administración y facturación de los clientes que también se encuentran dentro del XML

Palabras clave
TDA, POO, XML, Frontend, Backend

Abstract

In this project, it was proposed to develop an API with which to modify, consult and invoice an XML file, each tag of the XML represents the services that a company can offer to its clients, with this it is sought to facilitate the administration and billing of clients who are also within the XML

Keywords
TDA, OOP, XML, Frontend, Backend

Introducción

En el desarrollo web facilita el uso de aplicaciones, usando cualquier dispositivo que se puede conectar a internet puede fácilmente acceder a paginas para consumir contenido, acceder a cuentas personales, pagar servicios, tramites, etc.
Es por esto que se eligió el desarrollo de una API para administrar la base de datos de una empresa (en formato de xml) para tener un control de sus clientes y los servicios que ellos consumen facilitando la facturación de sus servicios usando Flask y Python para el Backend y Django para crear una interfaz para el navegador

Desarrollo del tema

Las estructuras principales con las que se manejaron este proyecto esta divida en 2 partes el Backend y Frontend

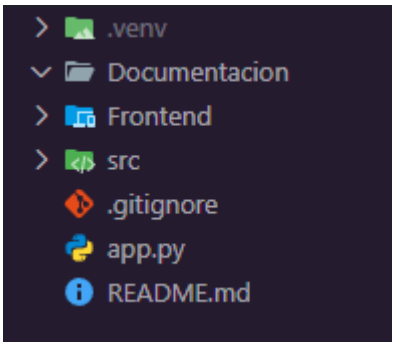

Figura 1. Carpetas del proyecto.
Fuente: elaboración propia.

Backend

En esta parte utilizaremos Python con Flask para crear las clases y funciones necesarias para manipular el XML nuestra clase principal que estará encargada de leer y manipular el XML será xmlReader

```python
import xml.etree.ElementTree as ET

from .categoria import categoria
from .cliente import cliente
from .configuracion import configuracion
from .instancia import instancia
from .recurso import recurso
You, 1 hour ago | 1 author (You)
class xmlReader:
    def __init__(self):
        self.recursos = []
        self.categorias = []
        self.clientes = []
        self.nombre_archivo = None
        self.datos_cargados = False

    def leer_xml(self, archivo):
        try:
            tree = ET.parse(archivo)
            root = tree.getroot()

            self.leer_recursos(root.find('listaRecursos'))

            self.leer_categorias(root.find('listaCategorias'))

            self.leer_clientes(root.find('listaClientes'))

            self.datos_cargados = True

            return True

        except Exception as e:
            print(f"Error al leer el archivo: {e}")
            return False
```

Figura 2. Clase xmlReader.
Fuente: elaboración propia.

Esta clase posee 3 listas para guardar las sub categorías que tiene el xml recursos, categoría y clientes estos a su vez tendrán sus propias sub listas que veremos mas adelante, usando la libreriar ElementTree para parsear el archivo y acceder a cada etiqueta recórrela capturar cada elemento y agregarlo es su respectiva lista

```python
def leer_xml(self, archivo):
    try:
        tree = ET.parse(archivo)
        root = tree.getroot()

        self.leer_recursos(root.find('listaRecursos'))

        self.leer_categorias(root.find('listaCategorias'))

        self.leer_clientes(root.find('listaClientes'))

        self.datos_cargados = True

        return True

    except Exception as e:
        print(f"Error al leer el archivo: {e}")
        return False
```

Figura 3. Función leer_xml.
Fuente: elaboración propia.

```python
def leer_recursos(self, lista_recursos_element):

    if lista_recursos_element is None:
        return

    for recurso_element in lista_recursos_element.findall('recurso'):
        recurso_id = recurso_element.get('id')
        nombre = self.obtener_texto(recurso_element.find('nombre'))
        abreviatura = self.obtener_texto(recurso_element.find('abreviatura'))
        metrica = self.obtener_texto(recurso_element.find('metrica'))
        tipo = self.obtener_texto(recurso_element.find('tipo'))
        valor_hora = self.obtener_texto(recurso_element.find('valorXhora'))
        nombre = self.obtener_texto(recurso_element.find('nombre'))
        abreviatura = self.obtener_texto(recurso_element.find('abreviatura'))
        metrica = self.obtener_texto(recurso_element.find('metrica'))
        tipo = self.obtener_texto(recurso_element.find('tipo'))
        valor_hora = self.obtener_texto(recurso_element.find('valorXhora'))

        try:
            valor_hora = float(valor_hora) if valor_hora else 0.0
        except ValueError:
            valor_hora = 0.0

        nuevo_recurso = recurso(recurso_id, nombre, abreviatura, metrica, tipo, valor_hora)

        self.recursos.append(nuevo_recurso)
```

Figura 4. Captura de datos de la etiqueta lista recursos.
Fuente: elaboración propia.

Esto se repite con todos los elementos del archivo (categoría y clientes) y sus subelementos (configuración, recursos configuración, instancias) estas ultimas se guardaran en las clases que poseen una sub lista

```python
class cliente:
    def __init__(self, nit, nombre, usuario, clave, direccion, correo):
        self.nit = nit
        self.nombre = nombre
        self.usuario = usuario
        self.clave = clave
        self.direccion = direccion
        self.correo = correo
        self.lista_instancias = []

    def to_dict(self):

        return {      You, 2 days ago • se agregaron a las clases categoria, cliente, c…
            'nit': self.nit,
            'nombre': self.nombre,
            'usuario': self.usuario,
            'clave': self.clave,
            'direccion': self.direccion,
            'correo': self.correo,
            'instancias': [instancia.to_dict() for instancia in self.lista_instancias]
        }
```

Figura 4. Clase cliente.
Fuente: elaboración propia.

Si nuestro primero dato de la lista resulta no ser None crea una variable temporal y entrará en un bucle en donde pasara al siguiente nodo y revisar su puntero siguiente hasta encontrar uno que se None y allí insertar el nuevo nodo que se creó al principio con esta se crea la estructura de una lista con los nodos

```python
from nodo_le import nodo_le

You, 1 hour ago | 1 author (You)
class lista_enlazada:
    def __init__(self):
        self.primero = None
        self.size = 0

    def lista_vacia(self):
        return self.primero == None

    def insertar(self, dato):
        nuevo = nodo_le(dato)

        if self.primero == None:
            self.primero = nuevo

        else:
            actual = self.primero
            while actual.siguiente != None:
                actual = actual.siguiente
            actual.siguiente = nuevo
        self.size += 1
```

Figura 3. Lista enlazada.
Fuente: elaboración propia.

Las siguientes estructuras son muy parecidas, ya que salen de estas mismas clases la lista doble y su nodo doble.

```python
You, 7 hours ago | 1 author (You)
class nodo_ld:
    def __init__(self, dato):
        self.dato = dato
        self.siguiente = None
        self.anterior = None
```

Figura 4. Nodo doble
Fuente: elaboración propia.

En este el nodo de una lista doble, su diferencia es que ahora tiene otro apuntador. En la lista simple, los datos solo apuntaban a un dato al que le seguía . En este, puede apuntar tanto al dato que le sigue como al que le precede

```python
import os
from nodo_ld import nodo_ld
You, 2 hours ago | 1 author (You)
class lista_doble:
    def __init__(self):
        self.primero = None
        self.size = 0

    def lista_vacia(self):
        return self.primero == None

    def insertar(self, dato):
        nuevo = nodo_ld(dato)

        if self.primero == None:
            self.primero = nuevo

        else:
            actual = self.primero
            while actual.siguiente != None:
                actual = actual.siguiente
            actual.siguiente = nuevo
            nuevo.anterior = actual
        self.size += 1

    def agregar(self, dato):
        nuevo = nodo_ld(dato)

        if self.primero == None:
            self.primero = nuevo

        else:
            nuevo.siguiente = self.primero
            self.primero.anterior = nuevo
            self.primero = nuevo

        self.size += 1
```

Figura 5. Lista doble
Fuente: elaboración propia.

La estructura de la lista doble es idéntica a la lista enlazada, tanto los métodos como lista vacía. El cambio que vemos en el de insertar es que ahora, al insertar un nuevo dato, también tenemos que indicarle al apuntador anterior qué dato viene antes que él.

La función agregar es un método que nos permite ingresar un dato al inicio de nuestra lista. Lo que hace es que el apuntador del nuevo nodo apunte al primer nodo de nuestra lista y ahora él apuntador anterior del primer nodo apunta al nuevo.

```
from lista_doble import lista_doble
from celda import celda
You, 2 hours ago | 1 author (You)
class campo_agricola:
    def __init__(self, id_campo, nombre_campo):
        self.id_campo = id_campo
        self.nombre_campo = nombre_campo
        self.matriz = lista_doble()

    def matriz_frecuencias(self, estaciones, sensores):
        for y in range(estaciones):
            encabezado = self.matriz.insertar(celda(n = 0, s = 1))
            for x in range(sensores):
                encabezado.fila.insertar(celda(n = x, s = y))      You, 7 ho
        print("Se creo el campo agricola")
```

Figura 6. Clase campo
Fuente: elaboración propia.

La clase campó agrícola contendrá la información de los campos, cada campo tiene un, id y un nombre único también cada campo tendrá un atributo matriz que es una lista doble está lista contendrá en cada nodo una lista simple hace es como crearemos la matriz cada lista simple será una fila y la lista dobles eran las columnas

La función matriz frecuencias creará la matriz de cada campo usando como parámetros la cantidad de estaciones y sensores que posee cada uno, para esto nuestro primer for recorrerá el número de estaciones e insertará en el atributo matriz este atributo al ser una lista doble tiene todos sus métodos y luego entrará en otro for para insertar los sensores como definimos en los nodos de la lista doble cada uno tiene una lista simple en el atributo fila

```
class celda:
    def __init__(self, frecuencia="0", s = -1, n = -1):
        self.frecuencia = frecuencia
        self.s = s
        self.n = n      You, 7 hours ago • se agrego a la
```

Figura . Clase celda
Fuente: elaboración propia.

La clase celda es llamada en la función matriz y nos ayuda a guardar las coordenadas de la matriz usando el contador x y en los for así cada celda tendrá su posición en la matriz. Cuando busquemos un dato, lo haremos con las coordenadas.

```
def cargarArchivo():
    ruta = input("Ingrese la ruta del archivo: ")
    leerArchivo(ruta)

def leerArchivo(rutaArchivo):
    tree = ET.parse(rutaArchivo)
    root = tree.getroot()

    for elementos_campo in root.findall('campo'):
        id = elementos_campo.get('id')
        nombre = elementos_campo.get('nombre')
        campo = campo_agricola(id, nombre)

        for elementos_sensor in elementos_campo.find('estacionesBase').findall('estacion'):
            id = elementos_sensor.get('id')
            nombre = elementos_sensor.get('nombre')
            estacion = nodo_ld(nombre)
            campo.fila.insertar(estacion)

        columnasXml.insertar(campo)
    columnasXml.imprimir()      You, yesterday • se agrego un metodo str a la clase campo para i…
    campo.fila.imprimir()
```

Figura 6. Función para leer un archivo
Fuente: elaboración propia.

En el archivo principal en donde se ejecutan todas las clases y funciones, tendremos una función para leer un archivo XML de donde extraeremos todos los datos de los campos.

Esto lo haremos con la librería de XML element tree para leer y escribir archivos, le pediremos la ruta del archivo al usuario y recorreremos cada elemento del archivo por medio de sus etiquetas. todos estos valores los capturamos en todas las clases vistas anteriormente.

Conclusiones

Aunque se lograron plantear algunas modelos y clases para el proyecto, debido al tiempo no se lograron implementar todo lo que el proyecto solicitaba, como el uso de Graphviz para representar

los campos de cultivo y la resta de filas de la matriz del campo

Referencias bibliográficas

Larman, C. (2004). *Applying UML and patterns: An introduction to object-oriented analysis and design and iterative development* (3rd ed.). Prentice Hall.

Python aplicado a la POO (Muy claro y práctico):
*Phillips, D. (2014). *Python 3 object-oriented programming* (2nd ed.). Packt Publishing.*