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

Carnet 202004822 – Marco Sebastian Solares España

Resumen

La empresa Tecnologías Chapinas, S.A. está desarrollando una herramienta que sea capaz de facturar detalladamente los servicios de infraestructura de nube que aprovisiona a sus clientes.

La estructura de la tecnología de nube desarrollada por Tecnologías Chapinas, S.A.
consiste en crear configuraciones de infraestructura que agrupan recursos necesarios para que una empresa pueda construir las arquitecturas de despliegue de aplicaciones que requiera.

Palabras clave

1. ***Factura***
2. ***Clientes***
3. ***Precio***
4. ***Servicios***
5. ***Nube***

Abstract

The company Tecnologias Chapinas, S.A. is developing a tool that is capable of detailed billing for the cloud infrastructure services it provides to its customers.

The structure of the cloud technology developed by Tecnologías Chapinas, S.A. It consists of creating infrastructure configurations that group the necessary resources so that a company can build the architectures for the use of the applications that they require.

Keywords

1. *Inovice*
2. *Customers*
3. *Price*
4. *Services*
5. *Cloud*

Introducción

El proyecto se basa en desarrollar una API que brinde servicios usando HTTP, bajo el conocimiento de POO y también de Django y Flask, para este proyecto se esta desarrollando una herramienta que sea capaz de facturar detalladamente los servicio en la nube para sus clientes.

Desarrollo del tema

Para el desarrollo de este proyecto separe por 2 carpetas el backend y el frontend.

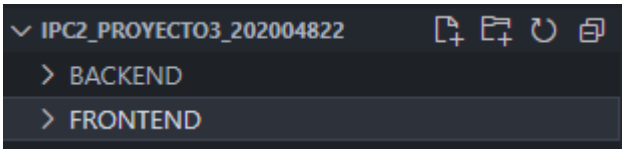


Figura 1. Carpetas.

Fuente: elaboración propia

BACKEND: En el apartado de backend utilice un archivo py. Llamado “main” en el cual en este apartado estarán todas las rutas de mi proyecto.

Y en otro llamado archivo manage se creo para ir haciendo toda la lógica del programa.

También utilice mis clases para ir almacenando las listas de configuración, recursos y clientes etc.

```
class Categorias():
    def __init__(self,idCategoria,nombre,descripcion,cargaTrabajo):
        self.nombre=nombre
        self.idCategoria=idCategoria
        self.descripcion=descripcion
        self.cargaTrabajo=cargaTrabajo
        self.lista_configuraciones= []
```

Figura 2. Class Categoria.

Fuente: elaboración propia

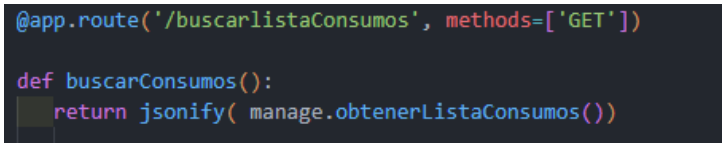
Una vez teniendo todos los archivos necesarios para almacenar la info, en el main se encuentra la ruta de agregar xml, esta ruta sivre para obtener desde el frontend o postman el xml para luego manipular esa información desde manage e ir agregando la información necesaria a mis listas.

```
@app.route('/add', methods=['POST'])

def AgregarDatos():
    xmlGood =True
    try:

        manage.listaRecursos =[]
        manage.listaCategorias = []
        manage.listaClientes = []
        manage.listaConfiguraciones = []
        count= 0

        xml = request.get_data().decode('utf-8')

        root = ET.XML(xml)

        for p in root:
            if p.tag== "listaRecursos":
                for subp in p:
                    manage.addListaRecursos(subp.attrib["id"], subp[0].text, subp[1].text, subp[2].text, subp[3].text, subp[4].text)

            elif p.tag =="listaCategorias":
                for subp2 in p:
                    for subsubp2 in subp2:
                        for subsubsubp2 in subsubp2:
                            for subsubsubsubp2 in subsubsubsubp2:
                                for subsubsubsubsubp2 in subsubsubsubsubp2:
                                    manage.addRecursosConfig(subsubsubsubsubp2.attrib["id"],subsubsubsubsubp2.text)
                                    count +1
```

Figura 3: agregar XML.

Fuente: elaboración propia

Una vez agregada la información se creo varias rutas para poder obtener esa información con un método “GET”, y así sucesivamente con la demás informacion.



```
@app.route('/buscarlistaConsumos', methods=['GET'])

def buscarConsumos():
    return jsonify( manage.obtenerListaConsumos())
```

Figura 4: Buscando datos.

Fuente: elaboración propia

También se utiliza un método “DELETE” para poder borrar toda la información almacenada den las listas.

```python
@app.route('/reset', methods=['DELETE'])

def reset():
    manage.reset()
    return jsonify({'msg':'borrado'})
```

Figura 5: borrando información .

Fuente: elaboración propia

```python
def reset(self):        You, 2 weeks ag
    self.listaRecursos = []
    self.listaCategorias = []
    self.listaClientes = []
    self.listaConfiguraciones = []
    self.listaRecursosConfig = []
    self.listaInstanci = []
```

Figura 5: Borrando informacion.

Fuente: elaboración propia

FRONTEND: Para realizar el frontend utilice el framewok llamado Django.

Django es un framework de desarrollo web de código abierto, escrito en Python. Este framework nos sirvira para poder unir el backend realizado en flask con el frontend

*Figura 6:*Django.

Fuente: elaboración propia

Para utilizar flask se deben hacer configuraciones, levantar el proyecto etc.

En django existe una carpeta llamada templates, el cual contenera todos los archivos html que iremos a utilizar en la Aplicación web y a la vez una carpeta llamada static que contiene todas las imágenes y archivos css.

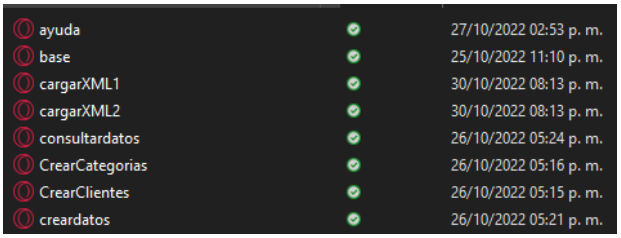


Figura 7: Templates.

Fuente: elaboración propia

En django esta el archivo views que esto sirve para conectar el frontend con flask, en este apartado podemos recibir o mandar informacion al backend.

```python
endpoint = 'http://127.0.0.1:4000/'


def index(request):
    context = {
        'datos': [],
        'title': 'Home'
    }

    try:
        response = requests.get(endpoint + 'buscardatos')
        buscardatos = response.json()

        context['datos'] = buscardatos

    except:
        print("API no esta corriendo")

    return render(request, "index.html", context=context)
```

Figura 8: views.

Fuente: elaboración propia

Para que funcionen las views debemos confirurar el archivo llamado “urls” en este archivo estarán todas las rutas de nuestros archivos html.

```
urlpatterns =[
    path('',views.index,name ='index'),
    path('signUp/',views.signUp,name='signUp'),
    path('login/',views.login,name='login'),
    path('home/',views.home,name='home'),
    path('operaciones/',views.operaciones,name='operaciones'),
```

Figura 9: urls.

Fuente: elaboración propia

Para poder mandar informacion del backend al frontend como por ejemplo un campo de texto en un html que diga nombre o contraseña, para poder mandar esta informacion django brinda un apartado llamado forms, que esto sirve para poder conectar los formularios del html y poder mandar esta info a las views.

```
class FileFormRecursos(forms.Form):
    idRecurso= forms.CharField(label="idRecurso")
    nombre= forms.CharField(label="nombre")
    abreviatura = forms.CharField(label="abreviatura")
    metrica= forms.CharField(label="metrica")
    tipo = forms.CharField(label="tipo")
    valorxhora= forms.CharField(label="valorxhora")
```

Figura 10: Forms.

Fuente: elaboración propia

Cuando hice el apartado de crear categorías, recursos, clientes manualmente en mi html se crearon los campos para que el usuario pueda mandar esa información y procesarla es por esto que era necesario crear el FileForm que se ve arriba

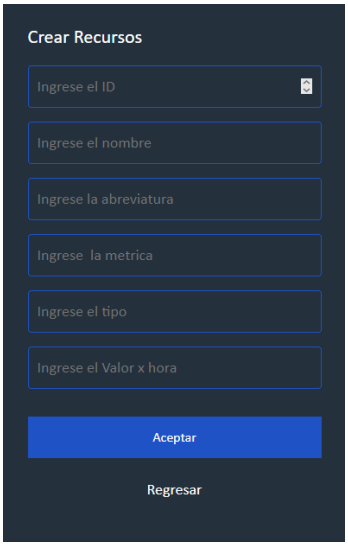


Figura 11: crear datos.

Fuente: elaboración propia

Conclusiones

Los módulos de lectura y escritura de archivos XML que son Minidom y ElementTree son 2 modulos muy buenos a la hora de realizar este proceso ya que su sintaxis es fácil de manejarla, cualquier novato en la programación podrá entender, la recomendación es usar elementTree ya que es más fácil de entender que Minidom.

Python es un lenguaje de programación fantástico para principiantes ya que su sintaxis es fácil de aprender y estos tipos de proyectos funcionan muy bien en el lenguaje.

Flask es fácil de entender, es una ventaja para los principiantes en estos temas también incluye un propio servidor web y cuenta con una documentación extensa.

Los beneficios de Django son el desarrollo rápido, el procesamiento rápido y la escalabilidad, mientras que las desventajas giran en torno a su naturaleza monolítica y la incapacidad de crear proyectos

más pequeños. También que no existe mucha documentación detallada

Bibliografía

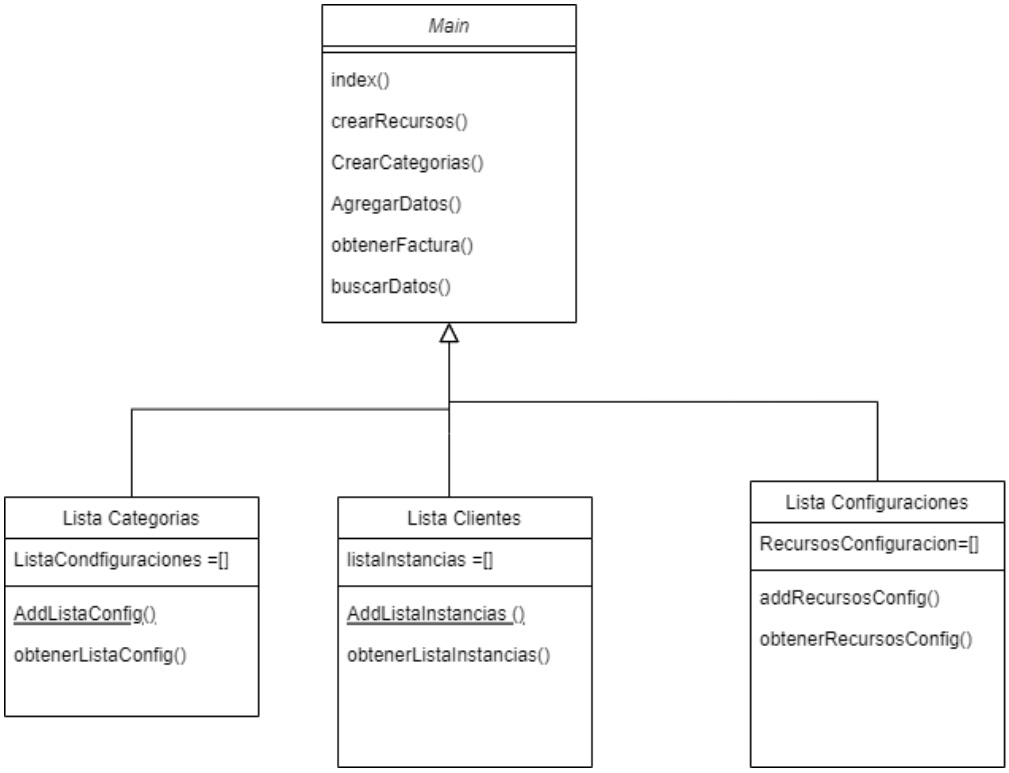
Django. (s.f.). *Django documentation*. Obtenido de https://docs.djangoproject.com/en/4.1/

ilimit. (s.f.). Obtenido de https://www.ilimit.com/blog/flask-vs-django/#:~:text=Ventajas%20de%20Flask,para%20el%20desarrollo%20de%20aplicaciones.

Kemel. (s.f.). *exabyteinformatica*. Obtenido de https://www.exabyteinformatica.com/tienda/foro/django-definicion-ventajas-y-desventajas-t3308.html#:~:text=Los%20beneficios%20de%20Django%20son,de%20crear%20proyectos%20más%20pequeños.

mdn web docs. (s.f.). Obtenido de https://developer.mozilla.org/en-US/docs/Web/CSS

Python. (s.f.). *The ElementTree XML API*. Obtenido de https://docs.python.org/3/library/xml.etree.elementtree.html



Modelos de datos