Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
Introducción a la programación y computación 2, 1er. Semestre 2022.

PROYECTO 3, LECTOR Y PROCESADOR DE MENSAJES XML

202010040 – Evelio Marcos Josué Cruz Soliz

Resumen

Se construyó una aplicación web con modelo cliente-servidor capaz de procesar archivos en formato ".xml" utilizando como front-end una aplicación con el módulo de Python "Django" y se usó flask para recibir y procesar las peticiones, es decir el back-end. Estos módulos se conectaron a través de envíos y repuestas en formatos json, para luego ser renderizados al usuario en formato html.

Palabras clave

Framework, front-end, back-end, respuesta, html, flask, xml, Django, petición.

Abstract

A web application with a client-server model capable of processing files in ".xml" format was built using an application with the Python module "Django" as a front-end and flask was used to receive and process the requests, that is, the back - end. These modules were connected through sending and responding in json formats, to later be rendered to the user in html format.

Keywords

Framework, front-end, back-end, response, html, flask, xml, Django, request.

Introducción

Una aplicación web es un conjunto de herramientas especializadas que tienen como fin permitir al usuario acceder a una base de datos o servidor, retornando de manera intuitiva mediante páginas web las solicitudes que el usuario haga. En este caso se solicitó una aplicación que tenga como fin principal el procesamiento y análisis de archivos xml, retornando las respuestas y dando la posibilidad de imprimir reportes de las mismas, para ello se implementaron dos frameworks, Django como front-end y Flask como back-end, estos se comunicaban mediante archivos json en rutas y métodos específicos.

Desarrollo del tema

Para obtener una mejor comprensión del tema, se ha dividido el contenido en varias secciones con el fin de dejar claros todos los puntos posibles

a. ¿Qué es una aplicación web?

Una aplicación web es un sistema que consta de varias capas que permiten a los usuarios clientes hacer varias peticiones al ordenador, por lo general estas aplicaciones funcionan bien en cualquier sistema, dispositivo y navegador que cuente con conexión a internet y capacidad de procesamiento de html5 y css.

Por lo general una aplicación web suele contener tres capas:

1. Una capa superficial, esta es la que se muestra en el navegador y por lo general, no suele utilizar lenguajes de programación, sino que solamente lenguajes de diseño de formato, como html y css.
2. Una capa interna, donde se ejecuta la "Lógica del negocio", es decir todos aquellos programas y métodos necesarios para el correcto procesamiento de los datos, envíos y requisitos que el usuario necesite
3. Una subcapa, que contiene los datos que la capa interna necesita para ejecutarse

Estos son los componentes principales de una aplicación web y su funcionamiento en general, en este caso en específico se trabajaron ambas capas con frameworks.

b. Que es un Framework

Framework o entorno se trabajo es una estructura creado para elaborar proyectos con fines específicos, se componen de distintos métodos y practicas a seguir dependiendo de cada proyecto y sus objetivos para proporcionar un molde o estructura a seguir en dicho proyecto

c. Flask

Flask es un Framework escrito en Python que se utiliza para facilitar el desarrollo de aplicaciones web, mediante un MVC (Modelo Vista Controlador), este es un inicio es algo limitado, pero podemos ir añadiéndole plugins para darle más funcionalidades, otra ventaja que posee es su curva estabilizada de aprendizaje

d. Django

Es un framework de alto nivel utilizado para desarrollar aplicaciones web rápidas, seguras y mantenibles, fundado en 2004, hoy en día posee una comunidad bastante grande y prospera que cada día se encuentran creando nuevos proyectos y expandiendo las posibilidades que este ofrece

Django también es un marco de trabajo que permite la creación de entornos web seguros en sí mismo, es decir, sin agregar funcionalidades, proveyendo una forma segura de administrar datos sensibles como las cuentas de usuario, contraseñas e información personal

Al estar escrito en Python es bastante fácil de leer, aunque posea una curva de aprendizaje algo elevada, se puede ejecutar en casi cualquier sistema operativo, media vez tenga un intérprete de Python

Ahora que hemos hablado de algunos conceptos básicos fundamentales, se ha de hablar acerca de que necesidades había en la aplicación y que métodos se usaron para solventarlas

1. Union de front-end y back-end

Una de las cosas más complicadas de entender al principio es como se va a unir dos entornos de trabajo diferentes, sin embargo, esto es bastante sencillo de entender:

- Como flask es el api que gestiona las peticiones correspondientes al back-end debemos de importarlo y crear una ruta inicial.

```
from flask import Flask, make_response, render_template, request
from flask.json import jsonify

app = Flask(__name__)
app.config["DEBUG"]=True

CORS(app)

@app.route("/", methods = ['GET'])
def MostrarAyuda():
    return jsonify({'creador':202010040, "documentacion" : False})
```

- Ahora ejecutamos y nos debe dar una instrucción con nuestra IP local como la siguiente:

```
t.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
 * Restarting with stat
C:\Users\Compu Fire\Documents\IPC2LAB\IPC2_PROYECTO3\Ejemplo Lab 11\IPC
3\backend\main.py:1: DeprecationWarning: The distutils package is depre
slated for removal in Python 3.12. Use setuptools or check PEP 632 for
alternatives
  from distutils.log import error
 * Debugger is active!
 * Debugger PIN: 743-186-873
```

- Copiamos la direccion local (localhost que aparece ahí y nos vamos a Django
- En django, creamos una ruta en el modulo “path” y creamos una variable global que guarde nuestro localhost

```
endpoint= 'http://127.0.0.1:5000/'

from io import BytesIO

from django.http import HttpResponse
from django.template.loader import get_template
from xhtml2pdf import pisa


#funcion de rendelizar el reporte
def render_to_pdf(template_src, context_dict):
    template = get_template(template_src)
    html  = template.render(context_dict)
    result = BytesIO()
    pdf = pisa.pisaDocument(BytesIO(html.encode("utf-8")), result)
    if not pdf.err:
        return HttpResponse(result.getvalue(), content_type='application/pdf')
```

Luego de eso ya tendriamos nuestra primer pagina con su back-end y front-end conectados

2. Carga de archivos

De igual manera, creamos una nueva view en Django, con la unica diferencia que le enviaremosa a la api un archivo recien cargado por el usuario, esto lo asignamos al parametro “data”

```
def cargaMasiva(request):
    ctx = {
        'content':None,
        'response':None
    }
    if request.method == 'POST':
        form = FileForm(request.POST, request.FILES)
        if form.is_valid():
            f = request.FILES['file']
            xml_binary = f.read()
            xml = xml_binary.decode('utf-8')
            ctx['content'] = xml
            response = requests.post(endpoint + 'CargaMasiva', data=xml_binary)
            if response.ok:
                ctx['response'] = xml
                response = response.json()
                return render(request, 'carga.html', response)
            else:
                ctx['response'] = 'El archivo se envio, pero hubo un error en el servidor'
                response = response.json()
                return render(request, 'carga.html', response)
        response = response.json()
        return render(request, 'carga.html', response)
```

3. Ordenamiento y clasificación de clases

Lo primero que se hace al recién recibir el archivo es guardarlo internamente, es decir, en el servidor, posteriormente se convierte de XML a un diccionario en el que es as fácil de recorrer y usar POO

Se ha definido una clase para cada objeto a utilizar en el reporte, es decir, mensajes, empresas y servicios, también se crean arreglos de estos objetos y con ayuda de ciclos, condicionales y variables temporales, todo a través de más de 200 líneas de código en las que se clasifican y ordenan el contenido del mensaje, es decir, fecha, lugar, empresa, servicio, etc.

```
class Mensaje:
    def __init__(self,lugar,fecha,hora,user,red, contenido):
        self.lugar=lugar
        self.fecha=fecha
        self.hora = hora
        self.user=user
        self.red =red
        self.contenido = contenido
        #Las primeras son variables de declaracion normal, las otras se declaran con funciones especiales
        self.texto = ""
        self.empresa = 'None'
        self.servicio = 'Desconocido'
        self.positivos = 0
        self.negativos = 0
        self.tipo = 'None'
        self.balancePositivo = 0.0
        self.balanceNegativo = 0.0
        self.Contener()
```

4. Salidas y reportes

Toda salida en el back-en se hacía iterando entre clases, objetos y atributos y se retorna siempre con un json, conteniendo este todo lo necesario para que en el front-end se renderize la respuesta o las respuestas necesitadas

Conclusiones

Django y Flask son entornos de trabajo muy diferentes entre sí, Flask es más sencillo e intuitivo de entender, mientras que Django es mucho más robusto y permite más configuraciones al momento de lanzar una aplicación web, sin embargo, sabiendo alternar bien entre estos dos frameworks se pueden crear cosas increíbles.

Lo más importante a la hora de crear una conexión entre el front-end posiblemente sean los archivos json a través de los cuales se dan peticiones y respuestas, sino que muchas veces también son parte de la base de datos

Crear una API es una labor que no debe tomarse a la ligera, cuando las personas ven la aplicación web desplegada, muchas veces se tiende a menospreciar el trabajo que hay detrás

Enfatizando, lo importante es destacar las principales posturas fundamentadas del autor, que desea transmitir a los lectores.

Adicionalmente, pueden incluirse preguntas abiertas a la reflexión y debate, temas concatenados con el tema expuesto o recomendaciones para profundizar en la temática expuesta.

Referencias bibliográficas

C. J. Date, (1991). *An introduction to Database Systems.* Addison-Wesley Publishing Company, Inc.

(Django, 2021)

.