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 2021.

PROYECTO 3 – INTRODUCCION A LA PROGRAMACION Y COMPUTO 2

201700375 – Bryan Steve Montepeque Santos

Resumen

Se requiere que el programador desarrolle una solución de Software Web diseñada para poder realizar la gestión de errores en la página web de la empresa, el servidor de la empresa está hecho en Flask debido a la simplicidad de su forma de usarse, pero todo el front end del proyecto está diseñado con Django, entonces se le solicita que utilice el Front End con Django y que solo le envie los datos al servidor de Flask mediante requests, esto es porque todo el sistema de la empresa no funciona con Java Script entonces se debe trabajar todo mediante la librería requests de Python para poder manejar de forma correcta las peticiones de HTTP entre un servidor y otro, es un sistema complejo pues tenemos dos servidores, uno de Django y uno de Flask pero para simplificar la tarea puede ver el servidor de Django como el controlador de la vista en HTML y de ahí comunicarse con el servidor real que es el de Flask.

Palabras clave

1. Vista
2. Front End
3. Controlador
4. Back End
5. Pertición HTTP

Abstract

We require the programmer to develop a web software solution designed to be able to manage error reports on the Enterpise's Web Page, the Web Page's server is made entirely on Flask because of its simplicity when you are working with it, but the whole Front End is designed on a Django server, so we require you to work the Front End on Django and only send the data to the Flask server using requests, this is because the whole enterprise's system doesn't work with Java Script so you must work everything using Python's requests library in order to handle all de HTTP Requests correctly between one server and the other, it is a complex system since we have two servers, a Django one and a Flask one, but to simplify the task you can see de Django server as just the controller for the HTML view and from there just communicate with the real server which is the one made on Flask.

Keywords

1. View
2. Front End
3. Controller
4. Back End
5. HTTP Request

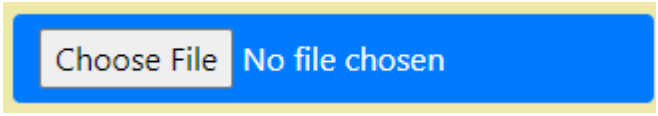
Introducción

Se ha solicitado un Administrador de Errores de una Página Web que debe contar con la opción de cargar un archivo XML a un servidor de Django y enviárselo a el servidor de Flask con una Petición HTTP en donde será procesado, los datos necesarios se identifican y se genera un nuevo XML donde se ordenarán los incidentes según sus fechas y sus errores, mostrará cuantas incidencias hay de cada error así como la cantidad de reportes de cada fecha y enviará el resultado de vuelta al servidor de Django para mostrar su código en la etiqueta del Django, y una vez ahí desde el Front End se podrá solicitarle estadísticas al servidor de Flask, y este devolverá la estadística requerida de forma gráfica para mostrarla en la vista del Proyecto.

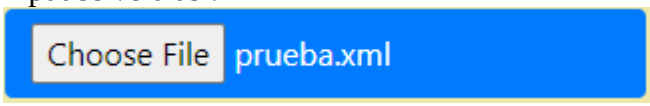
Desarrollo del tema

Para poder desarrollar este proyecto, fue necesario dividirlo en varias partes más pequeñas:

1. **Front End:**
   Para crear la Interfaz Gráfica fue necesario utilizar las diversas etiquetas que ofrece HTML, así como la CDN de Bootstrap para poder darle una mejor vista a la Página, para ingresar el archivo fue necesario usar una etiqueta input de tipo file para poder seleccionar el archivo que deseamos subir,

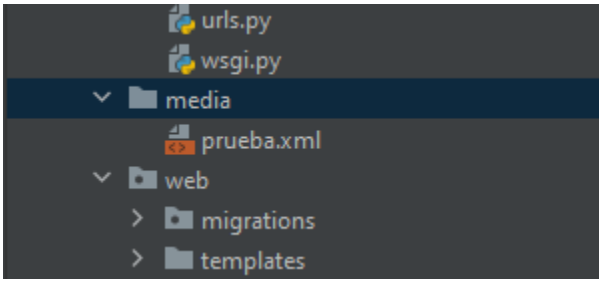


Pero esta etiqueta solamente abre el File Chooser y selecciona el archivo a subir, por lo que después de seleccionar el archivo la etiqueta input se verá así:

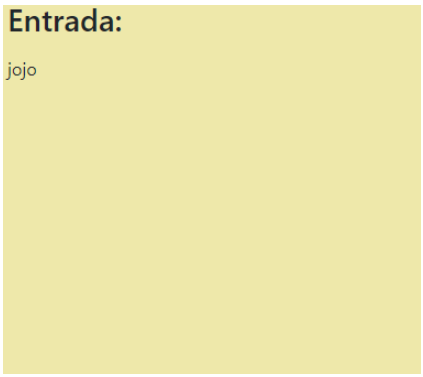


Pues ya está selecionado el archivo, pero para poder subirlo al servidor necesitamos otro botón, el botón Subir Archivo:

Lo que hace es el botón es subir el archivo al servidor al directorio “/media/”



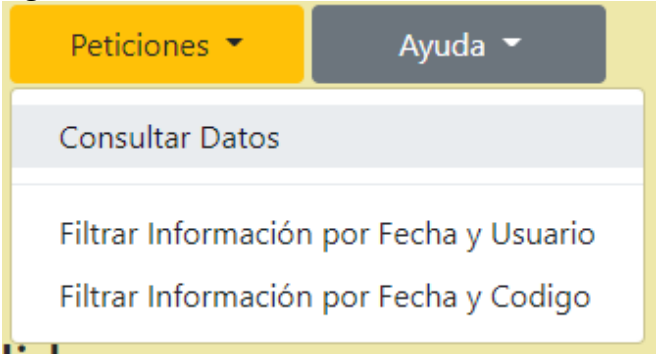
Se pudo realizar esta función en un solo botón, es decir, que el mismo botón “Subir Archivo” abriera el FileChooser y lo subiera, pero eso complicaría mucho la programación del botón debido a que el mismo botón tiene que realizar dos acciones al mismo tiempo, entonces mejor se optó por tener una etiqueta input para seleccionar el archivo y un botón de Subir Archivo para subir el archivo al servidor, el cual también lee el archivo y lo muestra en la etiqueta de “Entrada” del Front End



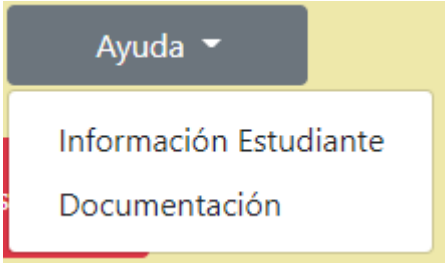
Y ahí hubiera mostrado mi entrada si hubiera logrado actualizar el texto del label con el botón.jpg

Después van los botones dropdown de Peticiones y Ayuda

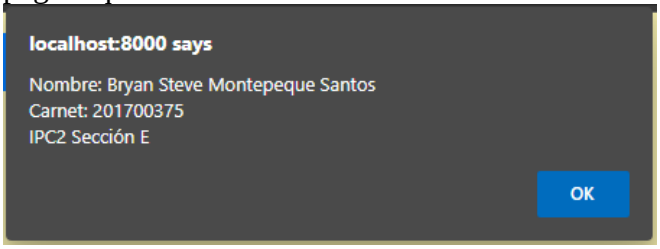
Estos lo que hacen es darnos acceso a las diferentes opciones dentro del programa, en el caso del botón Peticiones al hacer click en el nos muestra un menú con todas las posibles opciones que podemos pedirle al servidor que nos muestre tales como ver los datos almacenados en el XML ingresado, la gráfica de las estadísticas Fecha – Usuario o la gráfica de las estadísticas Fecha – Errores al hacer click en él nos muestra el siguiente menú:



Luego está el botón Ayuda, este contiene opciones mucho más simples que el botón Peticiones que no están relacionadas con el funcionamiento del programa sino que son más como extras del programa tales como visualizar los datos del estudiante y abrir la documentación, este es el menú que muestra el botón Ayuda al hacer click en él:

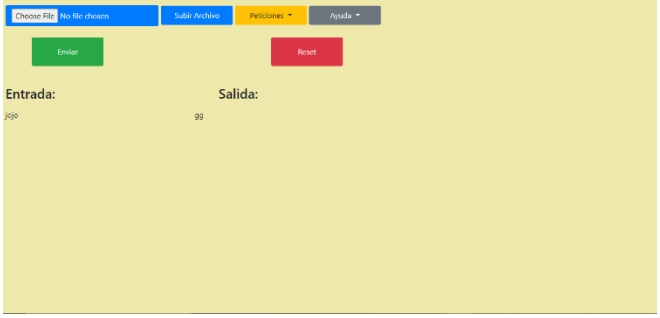


Y si hacemos click en Información Estudiante muestra una alerta en la parte superior de la página que se ve así:



Ahora están los botones de Enviar y Reset, el Botón Enviar lo que hace es mandarle los datos cargados en el XML del servidor de Django al servidor de Flask para que sea procesado mediante una petición POST, y el botón Reset solo vuelve toda la página a su estado inicial, estos s son los botones Enviar y Reset:

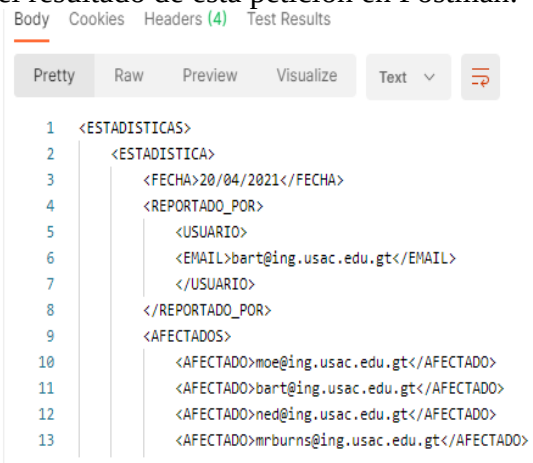
Al juntar todos estos componentes lo que obtenemos es esta vista:



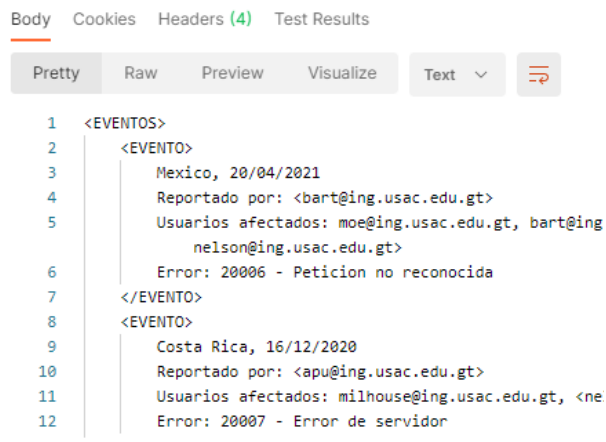
Fueron elegidos estos colores con el afán de crear una página que resulte estéticamente placentera para cualquier persona que utilice nuestro sistema. Eso es el fin del Front End ahora veremos el Back End

2. **Back End:**
   En esta parte veremos cómo funciona el programa en sí, específicamente veremos la parte en la que se analiza el XML, entonces en el lado del servidor de Flask tenemos dos peticiones en la dirección inicial de nuestra Página, la petición GET que lo que hace es

procesar el XML y genera el nuevo archivo de XML llamado Estadísticas y nos devuelve el archivo Estadísticas abierto como se ve en el resultado de esta petición en Postman:



Mientras que la petición POST solo nos muestra el archivo ingresado abierto:



Ahora, el análisis del XML es una de las partes más difíciles debido a que no tiene etiquetas, sí es un XML y eso nos obliga a usar Element Tree, ¿no? Pues no, porque solo tiene etiquetas que identifican a cada evento no a cada elemento dentro de cada evento que son las cosas que necesitamos saber, la idea original fue hacer un analizador léxico carácter por carácter, pero eso hubiera sido demasiado largo y tedioso, y al final del día no hubiera servido bien porque eso nos limita muchísimo la cantidad de cadenas que podemos ingresar, en especial con los correos, entonces, al final lo que se decidió fue meter todo el texto del archivo en un string y tratarlo como texto plano y usar Split para irnos deshaciendo de todas las partes inútiles del código, esta es solo una pequeña muestra del código necesario para analizar el XML:

```
#En cada linea hay un "\n", ahora ignoramos ese "\n"
for LineaInfoReal in range(1, len(EventosPurosPartidosPorLineas) - 1):


    #Quitamos el "/t"
    LineasDeInformacionValida = EventosPurosPartidosPorLineas[LineaInfoRea

    if LineaInfoReal == 1:
        #Partir Linea Pais, Fecha
        LineaActual = LineasDeInformacionValida[1].split(",")
        PatronFecha = re.search(r'(\d+/\d+/\d+)', LineaActual[1])
        FechaActual = PatronFecha.group()

    elif LineaInfoReal == 2:
        #Partir Linea Reportado por: Usuario
        LineaActual = LineasDeInformacionValida[1].split(":")
        PatronReportador = re.search(r'(\d+|\w+)[@](\d+|\w+)(\.|\d+|\w+)+
        ReportadorActual = PatronReportador.group()
```

Esto al final se almacena en listas e imprime en consola cada parte que fue analizada correctamente:

```
Esta es la Fecha Actual: 16/01/2021
Este es el Reportador Actual: user1@ing.usac.edu.gt
Esta es la lista de Afectados Actuales:
['user2@ing.usac.edu.gt', 'user4@ing.usac.edu.gt', 'coordinador@ing.usac.edu.gt']
Este es el Codigo del Error Actual: 30001
Este es el error actual:  30001 - Servicio no encontrado
```

Y al final se genera el nuevo XML con un string que se va concatenando según se necesiten los componentes, esto se logra solo recorriendo las listas almacenadas:

```
for i in range(len(ListaEventos)):
    StringArchivoXML = StringArchivoXML + "\t<FECHA>" + str(ListaEventos[i].Fech

    #Reportado Por
    StringArchivoXML = StringArchivoXML + "\t\t<REPORTADO_POR>\n"
    StringArchivoXML = StringArchivoXML + "\t\t\t<USUARIO>\n"
    StringArchivoXML = StringArchivoXML + "\t\t\t\t<EMAIL>" + ListaEventos[i].Repo
    StringArchivoXML = StringArchivoXML + "\t\t\t</USUARIO>\n"
    StringArchivoXML = StringArchivoXML + "\t\t</REPORTADO_POR>\n"

    #Afectados
    StringArchivoXML = StringArchivoXML + "\t\t<AFECTADOS>\n"
    for j in range(len(ListaEventos[i].Afectados)):
        StringArchivoXML = StringArchivoXML + "\t\t\t<AFECTADO>"+ ListaEventos[i
    StringArchivoXML = StringArchivoXML + "\t\t</AFECTADOS>\n"

    #Errores
```

y luego se genera un archivo al cual solo se le escribe el string concatenado y esto genera el

archivo estadísticas.xml:

```
#Generar el Archivo
with open("estadistica.xml", 'w') as EstadisticaXML:
    EstadisticaXML.write(StringArchivoXML)
EstadisticaXML.close()
```

Finalmente se abre el archivo para poder enviar solo texto en la respuesta del servidor:

```
#LeerElArchivoParaLaResponse
LeerNuevoXML = open("estadistica.xml", "r")
RespuestaNuevoXML = LeerNuevoXML.read()
return Response(response=RespuestaNuevoXML, mimetype='text/
```

Conclusiones

El programa fue dividido en varios "Mini Programas" o módulos para poder simplificarnos la tarea de programarlo y también para poder repartirnos el trabajo entre varias personas si fuera necesario hacerlo así, esta dedicación y atención al detalle se ve reflejada a lo largo de todo el programa, como al enfocarnos en asegurarnos que cada etiqueta del XML fue identificada de manera correcta e incluso se intentó recuperar los errores que vienen partidos en dos líneas, es algo muy complicado por lo que lo más recomendado es que solo se usen si es completamente necesario de lo contrario es mejor indicarle al encargado de hacer el archivo de entrada que lo haga bien.

Referencias bibliográficas

1. Chowdhury, T. A. (2020, February 5). *What is a view in Web Application? - Frontend Weekly*. Medium. https://medium.com/front-end-weekly/what-is-a-view-in-web-application-6a2836eed4eb#:%7E:text=To%20sum%20it%20all%20up,application%20development%20so%20much%20fun.

2. *Frontend Definition*. (2018, April 18). Tech Terms. https://techterms.com/definition/frontend#:%7E:text=The%20frontend%20of%20a%20software,that%20makes%20the%20interface%20function.

3. colaboradores de Wikipedia. (2021, May 5). *Modelo–vista–controlador*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador

4. N. (2020, September 30). *¿Qué es Back-End, Front-End y Back Office y por qué es importante para tu web?* Agencia Inbound Marketing Madrid. https://nestrategia.com/desarrollo-web-back-end-front-end/#:%7E:text=En%20otras%20palabras%2C%20el%20Back,la%20comunicaci%C3%B3n%20con%20el%20servidor.

5. *Mensajes HTTP - HTTP | MDN*. (2021, May 10). MDN Web Docs. https://developer.mozilla.org/es/docs/Web/HTTP/Messages