PROYECTO #3

20201005 – Derek Esquivel Díaz

Resumen

El presente proyecto es un programa construido para la Superintendencia de Administración Tributaria que tiene como finalidad recibir y autorizar Documentos Tributarios Electrónicos (DTE)

Para realizar esta validación y autorización el usuario deberá ingresar un DTE con formato xml en la página web de la SAT, el programa se encargará de verificar si los datos ingresados en el DTE son validos y si el numero de referencia es único en el sistema.

Si el DTE es considerado valido el programa se encargará de darle un numero de autorización único que consiste en la fecha del día y el numero de facturas que se han analizado ese día.

Luego será posible consultar todos los datos que existen actualmente en la base de datos del programa, ya sea exportando un archivo xml con todas los DTE autorizados o presentar los datos en forma de gráficos.

Palabras clave

Django; Python; Base de Datos; HTTP; XML

Abstract

This project is a program built for the Superintendency of Tax Administration (SAT) that aims to receive and authorize Electronic Tax Documents (DTE)

To perform this validation and authorization, the user must enter a DTE in xml format on the SAT website, the program will be verifying if the data entered in the DTE are valid and if the reference number of said document is unique in the system.

If the DTE is considered valid, the program will attach to the document an unique authorization number that consists of the date of the day the document was registered and the number of invoices that have been analyzed said day.

Then it will be possible to consult all the data that currently exists in the program's database, either by exporting an xml file with all the authorized DTEs or presenting the data in the form of graphs.

Keywords

Django; Python; Data Base; HTTP; XML

Introducción

El presente proyecto es un programa construido para la Superintendencia de Administración Tributaria que tiene como finalidad recibir y autorizar Documentos Tributarios Electrónicos (DTE)

El programa consiste en dos partes; el frontend y el backend.

El frontend será toda la interfaz con la que el usuario podrá interactuar, en esta será donde se podrá subir nuevos DTE al sistema, consultar todos los datos actualmente en el sistema y obtener gráficos con resúmenes de los movimientos realizados por los DTE.

El backend será un servicio que se encargará de procesar y almacenar todos los datos recibidos en el frontend. Para esto deberá analizar si la escructura del XML del DTE es correcta, así como verificar si el número de referencia es correcto. Si no detecta errores en el sistema deberá almacenar dicho DTE en su base de datos.

Además, este servicio tendrá la funcionalidad de regresar los datos que han sido almacenados de regreso al frontend para ser mostrados al cliente.

Desarrollo del tema

La primera parte del programa consiste en el fontend, este se dividirá en 3 partes, el área de carga de archivos, el área de grafico por rango de fechas y el área de grafico de IVA por fecha.

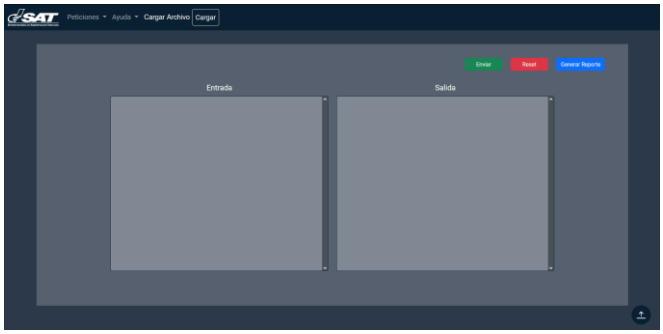

Figura 1: Pagina de carga de datos en el frontend.
Fuente: Elaboración propia (2021)

En el área de carga de datos habran dos áreas de texto, una identificada como “Entrada” y otra como “Salida”. En el área de entrada el usuario podrá ingresar manualmente un DTE o podra cargar un archivo XML con el mismo

```
<?xml version="1.0"?>
- <SOLICITUD_AUTORIZACION>
   - <DTE>
        <TIEMPO> Guatemala, 15/01/2021 15:25 hrs. </TIEMPO>
        <REFERENCIA> A1990 </REFERENCIA>
        <NIT_EMISOR> 737810 </NIT_EMISOR>
        <NIT_RECEPTOR> 8338815 </NIT_RECEPTOR>
        <VALOR> 100.00 </VALOR>
        <IVA> 12.00 </IVA>
        <TOTAL> 112.00 </TOTAL>
    </DTE>
 </SOLICITUD_AUTORIZACION>
```

Figura 2: Estructura del archivo DTE
Fuente: Elaboración propia (2021)

El archivo que se subirá del DTE deberá seguir la estructura de etiquetas de un archivo XML y deberá contener los siguientes datos:

- **Tiempo:** En esta etiqueta el usuario deberá ingresar la fecha en la que fue procesado el DTE en formato dd/mm/yyyy y la hora en la que se procesó en formato de 24 horas
- **Referencia:** Este será un código de hasta 40 caracteres, ya sea números o letras. Este código no se podrá repetir en el programa, de ser así el DTE será rechazado.
- **Nit Emisor:** Este será un numero de 1 a 20 caracteres.
- **Nit Receptor:** Este será un numero de 1 a 20 caracteres.
- **Valor:** Valor sobre el cual se esta realizando el DTE.
- **IVA:** Iva que se aplicara sobre el valor del DTE, este será el 12% sobre el valor.
- **Total:** Esta es la suma del valor y el IVA.

Una sola autorización podrá contener múltiples DTE.

Este archivo de autorizaciones será enviado al backend, donde se utilizara la librería “ElementTree” para leer las etiquetas y valores del archivo para así poder generar con objeto tipo “Autorizacion” por fecha. Este contendrá una lista con todas las facturas analizadas en una fecha especifica.

```python
    def analizarEntrada(self, xml, listaAutorizaciones):
        string = xml
        if string == "":
            return

        tree = ET.ElementTree(ET.fromstring(string))
        root = tree.getroot()

        for dte in  root.findall("DTE"):
            valid = True
            tiempo = str(dte.find("TIEMPO").text).strip()
            referencia = str(dte.find("REFERENCIA").text).strip()
            emisor = str(dte.find("NIT_EMISOR").text).strip()
            receptor = str(dte.find("NIT_RECEPTOR").text).strip()
            valor = str(dte.find("VALOR").text).strip()
            iva = str(dte.find("IVA").text).strip()
            total = str(dte.find("TOTAL").text).strip()
```

Figura 3: Código para leer el archivo XML de la autorizacion.
Fuente: Elaboración propia (2021)

Este código leerá el archivo y buscará errores, si detecta algún error en algún DTE no será agregado a la actualización. Además, se llevará un control de que tipos de errores fueron encontrados durante la lectura de los DTE.

Aquí se deberá validar cada dato ingresado para determinar si se agregara la factura a la lista de autorizaciones. Primero se analizará la fecha de la factura, esto para determinar si es necesario crear una nueva autorización o la factura será agregada a una autorización ya existente.

```python
    def getAutorizacion(self, listaAutorizacion, fecha):
        index = 0
        for autorizacion in listaAutorizacion:
            if autorizacion.fecha.strip() == fecha.strip():
                return index
            index += 1
        error = Errores()
        listaAutorizacion.append(Autorizacion(fecha,0,[],error,0,0,0))
        return index
```

Figura 4: Código para determinar la autorizacion.
Fuente: Elaboración propia (2021)

Una vez se ha determinado el índice de la autorización en la lista se procederá a almacenar los datos de la factura.

La siguiente comprobación que se hará será la de ver si el numero de referencia ya existe en el sistema, si no existe se procederá, si este código de referencia esta repetido la factura se decartara y será tomada como un error.

Luego se hará la validación del NIT, esta seguirá los siguientes pasos:

1. Se elimina el carácter más a la derecha.
2. Se multiplica cada digito por su posición respectiva, contando de derecha a izquierda (El ultimo digito por 1, el penúltimo por 2…)
3. Se suman todos los resultados de las multiplicaciones anteriores.
4. Se obtiene el residuo de dividir esta suma por 11.
5. A 11 se le resta este residuo.
6. Se calcula el residuo de este resultado al ser dividido por 11, si este numero es 11 entonces se le agregara al final del NIT el carácter validador “K”

```python
    def comprobarNIT(self, nit):
        valor = str(nit).strip()
        try:
            int(valor)
        except ValueError:
            return (False, None)

        if len(valor) > 20:
            return (False, None)

        digit_map = map(int, valor)
        nit_list = list(digit_map)

        nit_list.pop()

        nit_list = list(reversed(nit_list))

        total = 0
        aux = 1
        for num in nit_list:
            total += aux*num
            aux+=1


        total = total % 11
        total = 11 - total
        total = total % 11

        if total == 10:
            nit = str(nit).strip()
            nit += "K"

        return (True, nit)
```

Figura 5: Código para determinar el código validador del NIT.
Fuente: Elaboración propia (2021)

Si no se detecta ningún error durante la lectura se creará un objeto de tipo autorización por cada fecha. Este almacenara una lista de facturas leídas durante esta fecha, así como un control de todos los errores que se encontraron. A cada factura también se le agregará un numero de autorización, que será el orden en la que fue leída cada factura durante esa fecha.

```
13      def getCodigo(self, fecha):
14          cero = "00000000"
15          lista = fecha.split("/")
16          lista = reversed(lista)
17          valor = ''.join(map(str, lista))
18          codigo = str(valor) + cero
19          return (codigo.replace(' ',''))
20
```

Figura 6: Código para generar el número de autorizacion.
Fuente: Elaboración propia (2021)

Luego de haber leído y almacenado las autorizaciones el programa deberá crear un archivo de salida, para poder reanudar las operaciones del programa incluso si se llegara a apagar el servidor.

```
<?xml version="1.0"?>
- <LISTAAUTORIZACIONES>
  - <AUTORIZACION>
      <FECHA> 15/01/2021 </FECHA>
      <FACTURAS_RECIBIDAS> 8 </FACTURAS_RECIBIDAS>
    - <ERRORES>
        <NIT_EMISOR> 1 </NIT_EMISOR>
        <NIT_RECEPTOR> 1 </NIT_RECEPTOR>
        <IVA> 0 </IVA>
        <TOTAL> 0 </TOTAL>
        <REFERENCIA_DUPLICADA> 1 </REFERENCIA_DUPLICADA>
      </ERRORES>
      <FACTURAS_CORRECTAS> 4 </FACTURAS_CORRECTAS>
      <CANTIDAD_EMISORES> 2 </CANTIDAD_EMISORES>
      <CANTIDAD_RECEPTORES> 2 </CANTIDAD_RECEPTORES>
    - <LISTADO_AUTORIZACIONES>
      - <APROBACION>
          <NIT_EMISOR ref="A1993"> 7378106k </NIT_EMISOR>
          <CODIGO_APROBACION> 2021011500000001 </CODIGO_APROBACION>
          <TOTAL>639.80</TOTAL>
        </APROBACION>
      - <APROBACION>
          <NIT_EMISOR ref="A1994"> 7378106k </NIT_EMISOR>
          <CODIGO_APROBACION> 2021011500000002 </CODIGO_APROBACION>
          <TOTAL>1053.82</TOTAL>
        </APROBACION>
      - <APROBACION>
          <NIT_EMISOR ref="A1995"> 7378107k </NIT_EMISOR>
          <CODIGO_APROBACION> 2021011500000003 </CODIGO_APROBACION>
          <TOTAL>1759.80</TOTAL>
        </APROBACION>
      - <APROBACION>
          <NIT_EMISOR ref="A1996"> 7378107k </NIT_EMISOR>
          <CODIGO_APROBACION> 2021011500000004 </CODIGO_APROBACION>
          <TOTAL>857.30</TOTAL>
        </APROBACION>
        <TOTAL_APROBACIONES> 4 </TOTAL_APROBACIONES>
      </LISTADO_AUTORIZACIONES>
    </AUTORIZACION>
</LISTAAUTORIZACIONES>
```

Figura 4: Estructura del archivo de autorizaciones.
Fuente: Elaboración propia (2021)

Este archivo se actualizará cada vez que se realiza una consulta al backend para asegurar que los datos están al día y se genera con el siguiente código:

```
71      def generarSalida(self, listaAutorizaciones):
72          root = etree.Element('LISTAAUTORIZACIONES')
73          tree = etree.ElementTree(root)
74
75          for autorizacion in listaAutorizaciones:
76
77              autorizacion_tag = etree.Element('AUTORIZACION')
78              root.append(autorizacion_tag)
79
80              fecha = etree.Element('FECHA')
81              fecha.text = str(autorizacion.fecha)
82              autorizacion_tag.append(fecha)
83
84              recibidas = etree.Element('FACTURAS_RECIBIDAS')
85              recibidas.text = str(autorizacion.noFacturas)
86              autorizacion_tag.append(recibidas)
87
88              errores = etree.Element('ERRORES')
89              autorizacion_tag.append(errores)
```

Figura 5: Código para generar el archivo XML de salida.
Fuente: Elaboración propia (2021)

El usuario podrá consultar la versión mas reciente de este archivo de salida desde el frontend al usar la opción “Consultar Datos” desde la barra de navegación. Esto activara una request de javascript al servidor para obtener los datos del servidor.

```
function getSalida(){
  fetch('http://127.0.0.1:8000/getSalida', {
  method: 'GET',
  })
  .then(res => res.json())
  .catch(err =>{
      console.log("Ha ocurrido un error");
      alert("Ha ocurrido un error")
  })
    .then(response =>{
      document.getElementById('salida').value = response.xml
  })
}
```

Figura 5: Código del request para el archivo de salida.
Fuente: Elaboración propia (2021)

Esto regresara los datos para ser desplegados en el frontend.

Otra parte del frontend permitirá al usuario generar graficas con los datos almacenados en el servidor, estos gráficos podrán ser de dos tipos; un resumen de

IVA por fecha y NIT o un resumen por rango de fechas.

Para el resumen de IVA por fecha y NIT el usuario deberá ingresar la fecha de la cual querrá ver el resumen y se enviará una request con esta al backend.

El backend analizara si esta es una fecha valida que existe en el sistema, si hay autorizaciones en esta fecha regresara al backend una lista con los movimientos realizados por cada NIT, ya sea si este fue el emisor o el receptor.

```
def tablaIva(self, fecha, listaAutorizaciones):
    titulos = []
    lista_emitido = []
    lista_recibido = []
    lista_nits = self.getNits(listaAutorizaciones)

    fecha = fecha.strip()

    for autorizacion in listaAutorizaciones:

        fecha_actual = autorizacion.fecha.strip()

        if fecha == fecha_actual:

            for nit_actual in lista_nits:
                total_emitido = 0
                total_recibido = 0
                titulos.append(nit_actual)

                for factura in autorizacion.listaFacturas:

                    if str(factura.nit_emisor).strip() == str(nit_actual).strip():
                        iva = float(str(factura.iva).strip())
                        total_emitido += iva

                    if str(factura.nit_receptor).strip() == str(nit_actual).strip():
                        iva = float(str(factura.iva).strip())
                        total_recibido += iva

                if total_emitido == 0 and total_recibido == 0:
                    titulos.pop()
                else:
                    lista_emitido.append(float(round(float(total_emitido), 2)))
                    lista_recibido.append(float(round(float(total_recibido), 2)))

            return (titulos, lista_emitido, lista_recibido)

    return(None)
```

Figura 6: Código para obtener el resumen de IVA por fecha y NIT.
Fuente: Elaboración propia (2021)

Una vez el servidor ha enviado estos datos al frontend se utilizará la librería para javascript “Chart JS”, la cual tomara los datos y generará un grafico detallando la cantidad de IVA emitido o recibido por cada NIT.

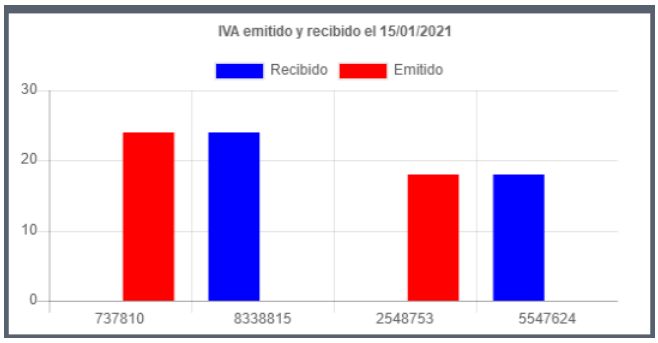

Figura 7: grafico del resumen de IVA por fecha y NIT.
Fuente: Elaboración propia (2021)

Para el resumen por rango de fechas el usuario deberá ingresar dos fechas, una inicial y una final, así mismo deberá seleccionar si desea los montos con o sin IVA. Una vez seleccionados se enviará una request al backend.
El backend analizara si estas fechas son válidas, si son validas analizara si hay autorizaciones dentro de este rango de fechas.

En esta autorización el programa leerá todas las facturas leídas, sumando los totales, ya sea con IVA o sin IVA dependiendo de lo que selecciono el usuario, luego almacenará esto en una lista, donde se irán agregando subsecuentes autorizaciones.

Cuando se han leído todas las autorizaciones dentro del rango de fechas se envía al frontend para generar la gráfica.

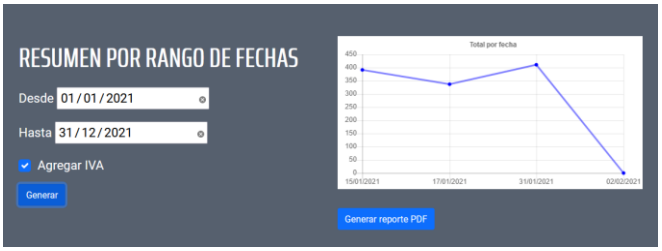

Figura 7: grafico del resumen por rango de fechas.
Fuente: Elaboración propia (2021)

Por último, el usuario podrá reiniciar el servidor borrando todos los datos almacenados, si el usuario decide hacer esto se enviará una request al servidor que borrará la lista de objetos y borrará el contenido del archivo de autorizaciones.

Conclusión

El desarrollo del programa utilizando Django para crear un Modelo-Vista-Template permitió simplificar la creación del frontend, pues permitió utilizar plantillas que fueron utilizadas por varias partes de la página, disminuyendo así la cantidad de código necesario y dando mas control sobre las peticiones al backend.

Utilizar flask en el backend ayudo a crear fácilmente un servidor para procesar las request realizadas en el frontend, pues entrega al usuario herramientas y librarías que permiten manejar todas esas request por nosotros, para que solo fuera necesario recibir la operación y manejarla o analizarla según nuestras necesidades.

Citas

- Django Software Foundation (2021) « Django REST framework», Recuperado de: https://www.django-rest-framework.org/
- Pallets *(2021)* «Flask Documentation», Recuperado de: https://flask.palletsprojects.com/en/2.0.x/