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 DE IPC2

201900716 – Kelly Mischel Herrera Espino

Resumen

En el presente ensayo se explica como se realizo el tercer proyecto de IPC2 usando expresiones regulares esta sirvió para poder manejar los errores que pudieran presentar un archivo con extensión xml, al aplicar expresiones regulares se obtenía la información importante que viniese en el dicho archivo para posteriormente guardarlos. Para almacenar estos datos se hizo uso de programación orientada objetos creando un arreglo de objetos en el cual se estarían guardando para luego poder hacer uso de ellos en diferentes procedimientos. Para el backend se hizo uso del framework Flask siendo este el servidor 1 el cual hacia todo el proceso de almacenar y retornar los datos para que fuesen usados en el frontend. Para el frontend la aplicación web se utilizó el framework Django siendo este el servidor 2. En el frontend solo se mostraba el resultado final según lo que el cliente deseara hacer como cargar un archivo, consultar los datos, mostrar gráficamente la cantidad de mensajes que un empleado reportaba en cierta fecha especifica o la cantidad de errores que se reportaron según esa fecha.

Palabras clave

- Frontend: consiste en la conversión de datos en una interfaz gráfica para que el usuario pueda ver e interactuar con la información de forma digital usando HTML, CSS y JavaScript.
- Backend: Consiste en un servidor, una aplicación y una base de datos. Se toman los datos, se procesa la información y se envía al usuario.
- Framework: Es un conjunto de archivos y pautas que definen la estructura y metodología, sobre como hace el desarrollo de un proyecto software.
- Flask: Es un framework escrito en Python que permite crear aplicaciones web.
- Django: es un framework de desarrollo web de código abierto, escrito en Python que respeta el patrón de diseño conocido como modelo vista controlador.

Abstract

In this essay it is explained how the third IPC2 project was carried out using regular expressions,

which were used to handle the errors that a file with an xml extension could present, when applying regular expressions the important information that came in the said file was obtained for later save them. To store this data, object-oriented programming was used, creating an array of objects in which they would be saved and then being able to make use of them in different procedures. For the backend, the Flask framework was used, this being the server 1 which did the whole process of storing and returning the data so that it could be used in the frontend. For the frontend, the web application was used the Django framework, this being server 2. In the frontend, only the final result was shown according to what the client wanted to do, such as uploading a file, querying the data, graphically displaying the number of messages that a employee reported on a specific date or the number of errors that were reported based on that date.

Keywords

- *Frontend: consists of converting data into a graphical interface so that the user can see and interact with the information digitally using HTML, CSS and JavaScript.*
- Framework: It is a set of files and guidelines that define the structure and methodology, on how to develop a software project.
- Backend: It consists of a server, an application and a database. The data is collected, the information is processed and sent to the user.
- Flask: It is a framework written in Python that allows you to create web applications.
- Django: it is an open source web development framework, written in Python that respects the design pattern known as the controller view model.

Introducción

Para la creación de este proyecto se usaron expresiones regulares y un autómata para poder ignorar los errores y los datos irrelevantes de archivo xml, al obtener los datos se procedió a guardarlos en una lista de objetos de la clase Datos, ya que luego se haría uso de estos datos para poder mostrarlos en la pagina web cuando el usuario requiriera de dicha información. Con flask se realizo la parte del backent y con Django la parte del frontend. Para Django se tuvo que usar forms en la parte del archivo html y jinja para poder mostrar los datos que se mandaba desde el backend en formato json por medio de diccionarios.

Desarrollo del tema

Expresiones regulares

Como se menciono en el resumen se hizo uso de las expresiones regulares como se muestra en la siguiente imagen:

```
eventos = texto.split("\n")

#Patron para la fecha
patronF=re.compile(r'[0-3]\d/[0-1]\d/\d+')
#Patron para el correo
patronN=re.compile(r'\w+@[\w\.\w]+')
#Patron para codigo de error
patronE=re.compile(r'Error: [\d]+')
#Patron de descripción
patronD=re.compile("-\s+\w[\w\s]+|-\w[\w]+|\s+\w[\w\s],")
```

Figura 1. Expresiones regulares usadas en el programa.

Fuente: elaboración propia, 2021.

```
print("Fecha:",fecha.group())
print("Correo:",correo1.group())
#Correo de los afectados
correosAfectados=patronN.findall(eventos[i])
print("Correos afectados:",correosAfectados)
listaC=[]
```

Figura 2. Expresiones regulares usadas en el programa.

Fuente: elaboración propia, 2021.

- Re.compile: este sirve para colocar un patrón que debe seguir la cadena para ser aceptada como se puede ver en la figura 1 se puede usar las veces que uno desee con diferentes patrones. Por ejemplo, la variable patronF almacenaba el patrón que debe de tener una fecha para ser aceptada en este caso debe de ser un numero que comience 0 y solo puede llegar hasta 3, luego el segundo numero que lo acompañe debe de ser un numero de 1 a 9. Seguido de este debe de haber una diagonal luego dos dígitos juntos de 1 a 12, después otra diagonal y por último el año. La variable patronN guarda el patrón que debe de tener un correo, la variable patronE el patrón que debe de tener el código de error siendo este solo números y por último la variable patronD almacena el patrón que debe de tener una descripción.
- Findal(): En la figura 2 podemos observar la expresión patronN.findal, es que busca en una cadena todo los datos que cumplan con el patrón y los va guardando en una lista.
- re.search(): Por ultimo este método busca en toda la cadena lo que cumpla con el patrón hasta que encuentre uno.

Autómata

Se hizo uso de un autómata para poder obtener los valores de la cadena que cumplirían con ciertas características y así solo devolviendo los datos aceptados.

```
def automata(self, lista):
    pos = 0
    estado = 0
    tamaño = len(lista)
    error = ""
    cadena1 = ""

    while pos < tamaño:
        if estado == 0:
            if lista[pos] == "<EVENTOS>":
                estado = 1
                pos += 1
            else:
                error += lista[pos]
                pos += 1
        elif estado == 1:
            if lista[pos] !="\t<EVENTO>" and lista[pos] != "\t</EVENTO>
                estado = 2
                cadena1 += lista[pos]+"$"
                pos += 1
            else:
                error += lista[pos]
                pos += 1
        elif estado == 2:
            if lista[pos] != "\t</EVENTO>":
                cadena1 += lista[pos]
```

Figura 3. Autómata parte1.

Fuente: elaboración propia, 2021.

```
            pos += 1
        elif lista[pos] == "\t</EVENTO>":
            estado = 1
            cadena1 += "\n"
            pos += 1
        else:
            pos += 1
    elif estado == 3:
        if lista[pos] == "/<EVENTO>":

            estado = 4
            pos += 1
        else:
            cadena1 += lita[pos]
            error += i
    elif estado == 4:
        pos += 1
    else:
        error+=","
#print(cadena1)
return cadena1
```

Figura 4. Autómata parte2.

Fuente: elaboración propia, 2021

Programación Orientada a Objetos POO

Se uso programación orientada a objetos ya que se hizo uso de varias clases y la creación de un arreglo de objetos.

Clases utilizadas:

Datos:

Ya que un clase es una plantilla a la en con la que los objetos deben de cumplir.

```
class Datos:

    def __init__(self,fecha,correo,codigo,descripcion):
        self.fecha=fecha
        self.correo=correo
        self.correos=[]
        self.codigo=codigo
        self.descripcion=descripcion
```

Figura 5 Clase Datos.

Fuente: elaboración propia, 2021

Como se puede observar en la figura 5 se creo un constructor con los siguientes atributos:

- fecha: se almacena la fecha.
- Correo: se almacena el correo del empleado que reporta.
- Correos: se guardan en una lista los usuarios afectados.
- Código: el código de error del problema.
- Descripción: la descripción del problema.

Clase GuardarDatos:

En esta clase se crea un lista de forma global en la cual se van a ir almacenando los objetos de la clase Datos.

```
class GuardarDatos:
    datos=[]
    def guardarDatos(self,fecha,correo,lista,codigo,descripcion):
        d=Datos(fecha, correo, codigo, descripcion)
        self.datos.append(d)
        d.correos=lista
        #for i in lista:
            #d.correos.append(i)
```

Figura 6. Clase GuardarDatos

Fuente: elaboración propia, 2021

Como se puede observar en la figura 6 en la clase Guardar datos se crea un método que recibe de parámetros: fecha, correo, lista, código y descripción. Estos se van a ir colocando en los parámetros del constructor.

Método retornarEstadistica:

```
def retornarEstadistica(self):


    cadena=""
    cadena="<ESTADISTICAS>\n"
#-------
    fechas=[]
    fecha=[]
    for i in self.datos:
        fechas.append(i.fecha)
        #print(fechas)

    for j in fechas:
        if j  not in fecha:
            fecha.append(j)
    #print(fecha)
    for i in fecha:
        #Lista para usuarios
        listaU=[]
        listaU2=[]
        #Listas de errores
        listaR=[]
        listaR2=[]
```

Figura 7 método retornarEstadisitca de la clase Datos.

Fuente: elaboración propia, 2021

Como se puede observar en la figura 7 el método retornarEstadisitcca de la clase GuardarDatos este método lo que hace es retornar en una cadena el xml de las estadísticas. Como se puede observar se usaron varios for para comparar las fechas y poderlas obtener de modo que estas no estuvieran repetidas. Luego de esto según la fecha se obtuvo la cantidad de mensajes que se enviaron en dicha fecha. Luego estos datos se fueron guardando en otras listas para hacer las demás comparaciones.

Método fecha

```
def fecha(self):

#--------
    fechas=[]
    fecha=[]
    for i in self.datos:
        fechas.append(i.fecha)
        #print(fechas)

    for j in fechas:
        if j  not in fecha:
            fecha.append(j)
    #print(fecha)
    return fecha
```

Figura 8 método fecha de la clase Datos.

Fuente: elaboración propia, 2021

Como se puede observar en la figura 8 el método usa dos listas en la primera guarda todas las fechas, luego con un for recorre dicha lista y con un if compara los datos de segunda lista y almacena las fechas sin que esta se repitan. Por ultimo retorna una lista con las fechas.

Cargar Archivo

La clase CargarArchivo es en donde se aplican todas las expresiones regulares y se mandan a llamar los métodos de el resto de clases.

En el caso de inclusión de figuras, deben ser nítidas, legibles en blanco y negro. Se denomina figuras a gráficas, esquemas, fotografías u otros elementos gráficos.

Figura 1. Título o descripción breve de la figura.

Fuente: elaboración propia, o citar al autor, año y página.

Flask

En esta parte se estará explicando lo que se hizo con flask. Para ello se creo una clase con el nombre app.py como se muestra en la figura 9.

```
from flask import Flask, request, jsonify
from flask_cors import CORS
import xml.etree.ElementTree as ET
import xmltodict
from CargarArchivo import CargarArchivo
from GuardarDatos import GuardarDatos
app=Flask(__name__)
CORS(app)

cargarA=CargarArchivo()
guardar=GuardarDatos()
archivo_xml=""
#Recorrer xml
@app.route("/cargar",methods=['POST'])
def archivo():
    content_dict=xmltodict.parse(request.data)
    print(content_dict["EVENTO"])
    return content_dict

#Metodo para cargar Archivo
@app.route("/cargarArchivo",methods=['POST'])
```

Figura 9 clase app.py.

Fuente: elaboración propia, 2021

Como se ve en la figura 9 se importaron varias librerías para el funcionamiento adecuado del programa.

```python
@app.route("/resetear",methods=['GET'])
def resetear():
    guardar.datos.clear()
    mensaje={
        'mensaeje':'Estado actual de la base datos: Vacia'
    }
    return jsonify(mensaje)

#
@app.route("/")
def index():
    return "En linea"

if __name__=="__main__":
    app.run(threaded=True,port=7000,debug=True)
```

Figura 10 clase app.py.

Fuente: elaboración propia, 2021

Frontend

```python
from django.urls import path
from . import views

urlpatterns = [
    path('',views.index, name='myapp-index'),
    path('pagina/',views.pagina,name='pagina'),
    path('pagina2/',views.pagina2,name='pagina2'),
    path('recibir/',views.recibir,name='recibir'),
    path('file/',views.cargarArchivo,name='file'),
    path('fecha/',views.retornar_fechas,name='myapp-Reporte'),
    path('probando/',views.probando,name='probando'),
    path('filtro2/',views.filtro2,name='fitro2'),
    path('ayuda/',views.mostrarAyuda,name='ayuda'),
    path('informacion/',views.mostrar_informacion,name='informacion'),
    path('resetear/',views.vaciar_lista,name='resetear'),
]
```

Figura 11 urls

Fuente: elaboración propia, 2021

Como se puede observar en la figura 11 se usaron varias urls para poder navegar entre las paginas.

El título de la tabla debe ser corto y conciso.

Referencias bibliográficas

Máximo 5 referencias en orden alfabético.

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

Anexos

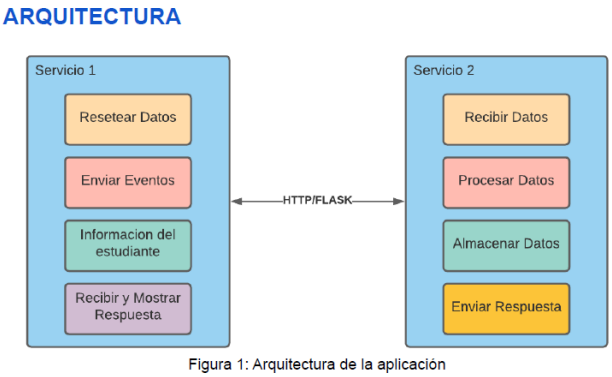

Figura 1: Arquitectura de la aplicación

Figura 12 Arquitectura de a aplicación

Fuente: elaboración propia, 2021

Página principal

Esta es la pagina principal que se le muestra al usuario.

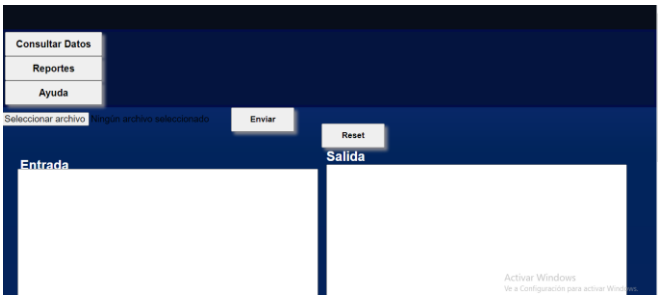

Figura 13 Pagina principal

Fuente: elaboración propia, 2021

Reportes:

En esta parte se muestran las gráficas por fecha.

Figura 14 Reportes

Fuente: elaboración propia, 2021