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: ANALIZADOR TECNOLOGIAS CHAPINAS, S.A.

**202001570 – Nataly Saraí Guzmán Duarte**

**Resumen**

En este ensayo se presenta la creación y desarrollo técnico del proyecto No. 3 del curso de Introducción a la Computación 2, del primer semestre de 2022. Este proyecto se realizó haciendo uso de la arquitectura cliente servidor, el servidor o también conocido como backend será el encargado de recibir y procesar la información, el cliente o también conocido como frontend será el encargado de mostrarle todas las opciones al cliente. Para la elaboración de este proyecto usamos dos Framework, Django para el Frontend y Flask para el Backend.

El programa desarrollo proporciona al usuario una gestión de datos de forma dinámica, para lo cual la información se almacenará en un archivo XML, cuya información será manejada por el servidor, permitiendo la posibilidad de realizar un conjunto de funciones indicadas en el documento de requerimiento. El conjunto de datos obtenido en el sistema se puede visualizar a través de la página web del cliente, proporcionando así la posibilidad de generar reportes.

**Palabras clave**

Protocolo de Internet, Servidor, Cliente, Petición y puerto.

***Abstract***

*This essay presents the creation and technical development of project No. 3 of the Introduction to Computing 2 course, from the first semester of 2022. This project was carried out using the client server architecture, the server or also known as backend will be the one in charge of receiving and processing the information, the client or also known as the frontend, will be in charge of showing all the options to the client. For the elaboration of this project we use two Framework, Django for the Frontend and Flask for the Backend*

*The development program provides the user with dynamic data management, for which the information will be stored in an XML file, whose information will be handled by the server, allowing the possibility of performing a set of functions indicated in the requirement document. The set of data obtained in the system can be viewed through the client's website, thus providing the possibility of generating reports..*

***Keywords***

*Internet Protocol, Server, Client, Request and port.*

## Introducción

La construcción del proyecto se fundamenta en el uso de la programación orientado a objetos, este paradigma nos permitió generar métodos y funciones para el procesamiento de la información que el documento de entrada tiene, así como las diferentes opciones que el usuario puede generar en la información del sistema. Para el obtener la información de los archivos se utilizó una librería cuyo propósito se basa en el procesamiento de archivo con extensión XML, esta librería permitió la lectura de datos así también como la construcción de archivos de salida con la extensión anteriormente mencionada.

El desarrollo del proyecto consiste en aplicar funcionalidades en los datos proporcionados en los archivos, la función principal es realizar un análisis de los sentimientos que se reciben de un mensaje y buscar a que empresa y servicio perteneces, así también la fecha de envió, que el archivo de entrada proporciona, cabe mencionar que toda la información dada en los archivos pasa por un riguroso proceso para saber si es válida o no. El proyecto proporciona al estudiante poner en práctica sus conocimientos y su lógica para encontrar la solución del problema propuesto en el documento de requerimiento.

## Desarrollo del tema

El problema propuesto en el documento de especificaciones y requerimientos especifica que la empresa Tecnologías Chapinas, S.A desea implementar un sistema que sea capaz de verificar la información de archivos con extensión XML, para dicha verificación es necesario que la información obtenida pase por un flujo de revisión que descarta la información que no cumple con los requerimientos solicitados. La información valida será aprobada por el sistema y se genera un archivo de salida con la misma extensión, llamado Respuesta.xml, este archivo contiene toda la información necesaria para generar reportes y mostrárselo al cliente.

Para encontrar la solución al problema anteriormente mencionado, debemos saber que el programa se desarrolló con las siguientes especificaciones: contamos con un servidor (Backend) encargado de realizar todas las funciones en el flujo de entrada y este es capaz de enviar respuestas al cliente (Frontend) encargado de mostrar una interfaz gráfica creado con Django y HTML cuyo propósito es generar la información y recibir para ser mostrada. El sistema es capaz de hacer una vista a la información por medio de filtros como la fecha, empresa y servicio, así como también de generar reportes en extensión PDF. Para implementar la solución utilizamos ciclo de vida del software, donde fase con fase se logró solventar el problema, las fases son:

a. Análisis
b. Diseño
c. Codificación
d. Prueba y corrección

A continuación, se explicará cada una de las etapas.

**Etapa de Análisis:**
En esta etapa se consideró cada una de las especificaciones dadas y propuestas en el documento de especificaciones y requerimientos y se logró determinar que lo más eficiente para solución sería utilizar el paradigma de programación orientada a

objetos, ya que este nos permite el manejo de clases y objetos. Las clases son una plantilla abstracta de los individuos que interactúan en el problema.

Tomando como base los individuos que interactúan en el sistema de autenticaciones, se logró determinar las siguientes clases: clase MAIN, MANAGER, EMPRESA, MENSAJES, NEGATIVOS, POSITIVOS, RESPUESTA, SERVICIOS estas clases ayudaran a generar las estructuras del backend.

Tabla I. *Listado de clases.*

| CLASES | FUNCION |
|---|---|
| MAIN | |
| MANAGER | APPI PROCESOS |
| EMPRESA | OBJETO |
| MENSAJES | OBJETO |
| NEGATIVOS | OBJETO |
| POSITIVOS | OBJETO |
| RESPUESTA | OBJETO |
| SERVICIOS | OBJETO |

Fuente: elaboración propia.

**Etapa de Diseño:**

Obteniendo la información de la etapa de análisis, se procedió a crear el diseño de las clases propuestas en la Tabla I y la relación entre ellas. Para esta etapa se utilizó el diagrama de clases que nos proporciona UML. Para el diagrama utilizamos relaciones de asociación y dependencia para relacionar las clases, también cuenta con la multiplicidad en las relaciones

La parte de la estructuración de proyecto, se decidió utilizar el modelo Cliente Servidor ya que hacemos unos de una API que gestiona todo el proceso del backend esta será el servidor, y el cliente este proceso lo gestiona el frontend para ello implementamos este modelo para el funcionamiento de nuestra aplicación en el siguiente diagrama mostraremos el planteamiento de la arquitectura utilizado y planteada en el proyecto.

El diagrama de la arquitectura es la siguiente:

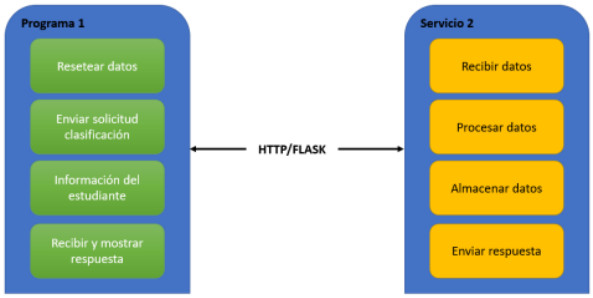


Figura 1. *Arquitectura Cliente-Servidor.*

Fuente: enunciado proyecto.

El diagrama de clases ya finalizado es el siguiente:

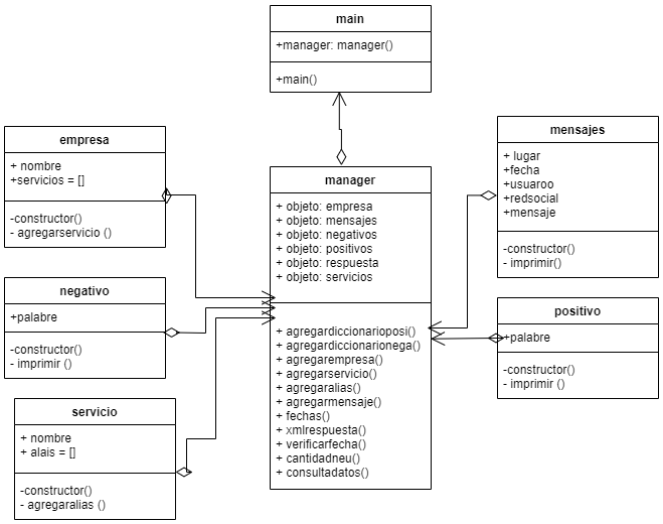


Figura 2. *Diagrama de clases UML.*

Fuente: elaboración propia.

**Etapa de Codificación:**

Para esta etapa del proyecto utilizamos los siguientes componentes:

a. Sistema operativo: Windows 10

b. Editor de código: Visual Studio Code

c. Lenguaje de Programación: Python 3.9.6

d. Framework Backend: Flask

e. Framework Frontend: Django

Debemos saber que la codificación del backend va de la mano con el diagrama mostrado en la figura 1, se explicara la funcionalidad de cada una de las clases y los métodos utilizados. Se comenzó instalando los paquetes de Flask y Django para poder hacer uso de métodos y características.

Primero codificó el backend ya que es la parte más fundamental del proyecto, ya que este es el encargado de hacer todo el procesamiento de los datos obtenidos del archivo de entrada. Flask cuenta con una sintaxis muy amigable y sencilla de utilizar se van generando endpoint y cada uno de estos cuenta con una función, el primer archivo a crear fue el DTE.py el cual es la clase objeto utilizada para verificar la información, a pesar que el funcionamiento del backend va en el archivo App.py, el archivo que contiene toda la lógica de la programación orientada a objetos la llega el archivo manager.py, este es el encargado de implementar los métodos necesarios para poder generar la verificación de los datos así como la lógica al manejarlos, debemos saber que para ello necesitamos usar la librería Element Tree para poder manejar la información XML que ingresaba por medio de los archivos, el archivo main.py se encarga de generar, un objeto de tipo manager para poder hacer uso de los distintos métodos que esta clase contiene.

Como nuestra aplicación es web debemos saber y conocer de los métodos del protocolo HTTP para poder gestionar información, estos métodos son el POST, GET, DELETE y PUT, existen varios pero los más conocidos son estos. Cada endpoint creado en la API debe especificar qué tipo de método HTTP utiliza porque de esto depende la petición que va a generar el frontend.

Para la codificación del cliente, esta parte es el frontend es toda interfaz gráfica que el usuario ve y utiliza para gestionar el sistema, para el frontend utilizamos el framework Django, su sintaxis utiliza Python con la diferencia que debemos configurarlo y manejar todo por vistas y enlaces, para esta parte usamos una plantilla HTML obtenido de Bootstrap luego la implementamos como templates dentro de nuestra app creado con Django, luego se configuración las url y las vistas para renderizar las templates necesarias, en las vistas haciendo uso de la librería request se procedió a generar las peticiones, para cada una de las vistas, esto para hacer la conexión con la API ya que para las peticiones POST se envía la información a la API para que esta se encargue de manipularla, en lugar que las peticiones GET estas reciben información a través de las respuestas de los endpoints de la API, cabe mencionar que la comunicación entre petición y endpoint son respuestas en estructura Response, toda petición generada debe recibir una respuesta, obligatoriamente.

**Etapa de Prueba y Corrección:**

En esta etapa se realizaron las pruebas de cada una de las funciones del proyecto, en especial las generadas en la API ya que utilizamos un software que genera peticiones HTTP para gestionar las pruebas necesarias en este caso utilizamos POSTMAN, cabe mencionar que esta etapa se hice en conjunto con la etapa de codificación ya que era necesario probar cada una de las funciones creadas en la API.

## Conclusiones

La arquitectura de cliente servidor es muy utilizada, ya que en la actualidad cualquier servicio en la red se necesario un servidor potente dependiendo que tipo de servicio se presta, existen muchos framework y sistemas que gestionan los servidores de mejor manera todo depende del tipo de servicio y la cantidad de gestión que el servidor llevara.

La implementación de framework facilita la creación de servidores Flask y Django se caracterizan por tener una sintaxis muy amigable y sencilla para que las personas puedan crear y configurar los servidores.

## Referencias bibliográficas

M.A. Grinberg, (2018). *Flask Web Development Second Edition*. O'Reailly, Inc.

A. J. Holovaty,(2009). *The Book Django Framework.* Prentice-Hall

M. A. Grinberg, (2018). *The New and Improved Flask Mega*. O'Reailly, Inc.

## Anexos

La interfaz gráfica es amistosa y dinámica para su interacción con el usuario, cuenta con distintas opciones como podemos ver tiene inicio, cargar archivo, realizar peticiones, mostrar información del estudiante y documentación, todas las opciones cuentan con una interfaz limpia y sencilla para su fácil entendimiento para el cliente.

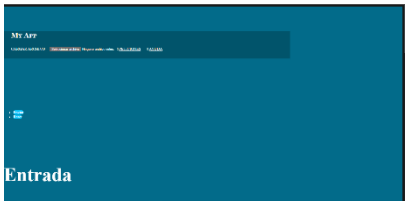


Figura 3. *Interfaz gráfica creada en Django.*

Fuente: elaboración propia.

Archivo de entrada para la maquina tiene una estructura especifica:

```xml
<?xml version="1.0"?>
<solicitud_clasificacion>
    <DICCIONARIO>
        <sentimientos_positivos>
            <palabra> bueno </palabra>
            <palabra> excelente </palabra>
            <palabra> cool </palabra>
            <palabra> satisfecho </palabra>
        </sentimientos_positivos>
        <sentimientos_negativos>
            <palabra> malo </palabra>
            <palabra> pésimo </palabra>
            <palabra> triste </palabra>
            <palabra> molesto </palabra>
            <palabra> decepcionado </palabra>
            <palabra> enojo </palabra>
        </sentimientos_negativos>
        <empresas_analizar>
            <empresa>
                <nombre> USAC </nombre>
                <servicio nombre="inscripción">
```

Figura 4. *Archivo de solicitudes.*

Fuente: elaboración propia.

Codificación del objeto empresa para la construcción de la lista de información de solicitudes, cuenta con sus atributos.

```python
from servicios import servicios
class Empresa():
    def __init__(self, nombre):
        self.nombre = nombre
        self.servicios = []

    def getNombre (self):
        return self.nombre

    def agregarservicio(self,servicio):
        new= servicios(servicio)
        self.servicios.append(new)

    def getServicios(self):
        return self.servicios
```

Figura 5. *Clase empresa.*

Fuente: elaboración propia.

Codificación del objeto mensajes para la construcción de la lista de información de solicitudes, cuenta con sus atributos.

```python
class mensajes():
    def __init__(self, lugar,fecha,usuario,redsocial,mensaje):
        self.mensaje= mensaje
        self.lugar = lugar
        self.fecha = fecha
        self.usuario = usuario
        self.redsocial = redsocial

    def getmensaje (self):
        return self.mensaje

    def setmensaje (self, mensaje):
        return self.mensaje == mensaje

    def impri(self):
        print("mensaje:" +self.mensaje)
```

Figura 6. *Clase mensajes.*

Fuente: elaboración propia.

Codificación del objeto negativos, positivos y servicios para la construcción de la lista de información de solicitudes, cuenta con sus atributos.

```python
class negativos():
    def __init__(self, palabra):
        self.palabra= palabra

    def getPalabra (self):
        return self.palabra

    def setPalabra (self, palabra):
        return self.palabra == palabra

    def impri(self):
        print("Palabra:" +self.palabra)
```

Figura 7. *Clase negativos.*

Fuente: elaboración propia.

```python
class positivos():
    def __init__(self, palabra):
        self.palabra= palabra

    def getPalabra (self):
        return self.palabra

    def setPalabra (self, palabra):
        return self.palabra == palabra

    def impri(self):
        print("Palabra:" +self.palabra)
```

Figura 8. *Clase positivos.*

Fuente: elaboración propia.

```python
class servicios():
    def __init__(self, nombre):
        self.nombre = nombre
        self.alias= []

    def agregaralias(self, alias):
        self.alias.append(alias)

    def getAlias(self):
        return self.alias
```

Figura 9. *Clase servicios.*

Fuente: elaboración propia.

Codificación de la manager con sus métodos y constructor que será implementado para la lógica del programa.

```python
import xml.etree.ElementTree as ET
import re
import json
from servicios import servicios
from positivos import positivos
from negativos import negativos
from mensajes import mensajes
from empresa import Empresa
import webbrowser
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
from sys import flags
class manager():
    def __init__(self):...

    def agregardiccioposi(self, palabra):...

    def agregardiccionega(self, palabra):...

    def agregarempresa(self,nombreemp):...
```

Figura 10. *Clase manager.*

Fuente: elaboración propia.

Codificación de la clase app donde se emplea la sintaxis de Flask y se implementan los distintos endpoint.

```python
@app.route('/consulta')
def consulta():
    manager.consultaDatos()
    return jsonify({'ok' : True, 'msg':'Consulta de datos correcta'})

@app.route('/resumenfecha/<fecha>', methods=['GET'])
def resumen(fecha):
    fech=str(fecha).replace("-","/")
    ob=manager.resumenFecha(fech)
    return jsonify({'ok' : True, 'msg':'Todo correcto'})

@app.route('/resumenfecha/<fecha>/<empresa>', methods=['GET'])
def resumenempre(fecha,empresa):
    fech=str(fecha).replace("-","/")
    ob=manager.resumenFechaEmp(fech,empresa)
    return jsonify({'ok' : True, 'msg':'Todo correcto'})

@app.route('/ayudaop1', methods=['GET'])
def ayudaop1():
    return 'NOMBRE:Nataly Saraí Guzmán Duarte \n CARNÉ:202001570'
```

Figura 11. *Clase main.*

Fuente: elaboración propia.

Codificación y clases que se utilizan para el frontend implementado con Django, la carpeta core contiene la aplicación principal de Django mientras la parte del frontend es la encargada de proporcionar los ajustes y las rutas que generaran las vistas.

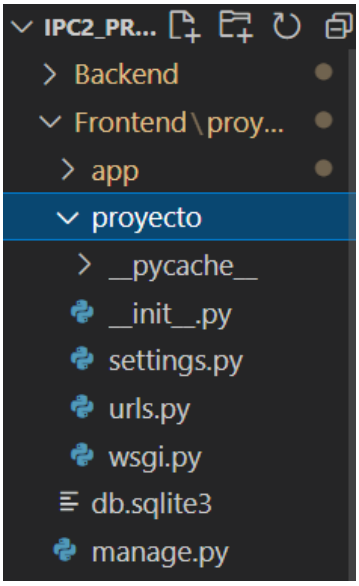


Figura 12. *Aplicación Django.*

Fuente: elaboración propia.

Archivo de salida con extensión XML que se usara como base de datos, llamado Resultado.xml

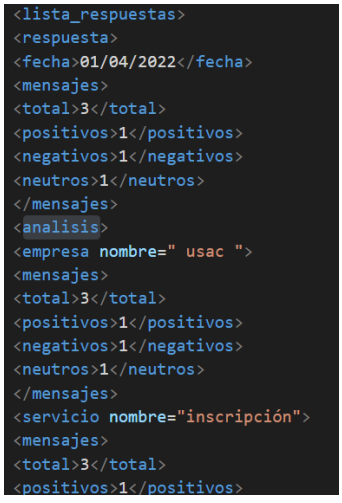


Figura 12. *Archivo Resultado.xml.*

Fuente: elaboración propia.