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 2024

SISTEMA DE GESTION DE CLIENTES, BANCOS Y TRANSACCIONES ATRAVES DE UNA PAGINA WEB COMBINANDO FLASK Y DJANGO

202100543 – Harold Benjamin Oxlaj Mangandi

Resumen

La empresa “Industria Típica Guatemalteca, S.A.” (ITGSA) ha decidido expandir su negocio creando un canal de distribución denominado “On-Line”. En el canal de distribución On-Line, la empresa ha creado una solución web denominado “Tienda virtual” para que los clientes adquieran productos y servicios que ITGSA ofrece. Una vez que el cliente ha seleccionado los productos y servicios que desea adquirir, se debe generar una factura por el monto total que el cliente debe pagar.

Adicionalmente, ITGSA ha realizado un convenio con los bancos de Guatemala, de tal manera que los clientes, utilizando la banca en línea de su banco favorito, pueden realizar pagos que se trasladan directamente a las cuentas de ITGSA.
El proyecto se basa en una página web consumiendo una API para la realización de las acciones que desee la empresa, por ejemplo, cargar clientes, bancos, sus respectivas transacciones, ver los estados de cuenta de los clientes activos, así como también generar en formato .pdf reportes de las tablas y gráficos de los pagos.

Palabras clave
Web, servicios, api, consumir, solicitudes

Abstract

The company "Industria Típica Guatemalteca, S.A." (ITGSA) has decided to expand its business by creating a distribution channel called "On-Line". In the On-Line distribution channel, the company has created a web solution called "Virtual Store" for customers to purchase products and services that ITGSA offers. Once the customer has selected the products and services they wish to purchase, an invoice must be generated for the total amount the customer needs to pay.

Additionally, ITGSA has entered into an agreement with banks in Guatemala, so that customers, using the online banking of their favorite bank, can make payments that are transferred directly to ITGSA's accounts. The project is based on a web page consuming an API to perform actions desired by the company, such as loading clients, banks, their respective transactions, viewing account statements of active clients, as well as generating reports in .pdf format of tables and graphs of the payments.

Keywords
Web, services, API, consume, requests

Introducción

Los servicios en línea se han vuelto una necesidad para los vendedores y empresas para expandir la cantidad de posibles clientes interesados en sus servicios.

El siguiente proyecto es una aplicación que consta de un Frontend desarrollado en Django, y un Backend desarrollado en Flask. Esta combinación permite que ambos frameworks trabajen al mismo tiempo.

Mediante el Backend del sistema se podrá llevar control de las facturas generadas por el sitio web y de los pagos que los clientes realicen en las distintas bancas en línea, Si un cliente paga más de lo que debe, podría quedar con un saldo a su favor, el cual se aplicará automáticamente cuando el cliente realice otra compra en el sitio web.

Por otro lado, el Frontend se utilizará para crear los clientes y bancos con que trabaja ITGSA, además de gestionar las operaciones de facturación y pagos por medio de tablas y graficas generadas en el sitio web disponibles para su descarga.

Desarrollo del tema

La Estructura de la aplicación se divide en dos, el Backend y en Frontend anteriormente dicho

Backend

Flask está configurado para usar el puerto: 8880

Por el lado del Backend se utilizó el framework flask, para la realización de los endpoints, el manejo y extracción de datos de la base de datos simulada por dos archivos xml, “db.clientes” y “db.transacciones” ambas registran los clientes agregados en la aplicación

Las funciones del archivo sin contar los endpoints son: agregar la información de los clientes y transacciones de la base de datos, verificar cualquier error en los datos de las facturas y pagos y obtener los meses anteriores dada una fecha, utilizada para el Resumen de pagos.

```python
#Leer de La db los clientes
def agregar_info_clientes(ruta_archivo_clientes):
    global ClientesRegistrados
    global BancosRegistrados
    tree = ET.parse(ruta_archivo_clientes)
    raiz = tree.getroot()
    lista_clientes = raiz.find("clientes")
    lista_bancos = raiz.find("bancos")
    if lista_clientes is not None:
        for cliente in lista_clientes:
            nit = cliente.find("NIT").text.strip()
            nombre = cliente.find("nombre").text.strip()
            nuevo_cliente = Cliente(nombre, nit)
            ClientesRegistrados.append(nuevo_cliente)
    if lista_bancos is not None:
        for banco in lista_bancos:
            codigo = banco.find("codigo").text.strip()
            nombre = banco.find("nombre").text.strip()
            nuevo_banco = Banco(nombre, codigo)
            BancosRegistrados.append(nuevo_banco)
            CodigosBanco.append(nuevo_banco.codigo)
```

Figura 1. Recuperación de clientes desde la Base de Datos
Fuente: Elaboración propia 2024

Las Clases utilizadas en este proyecto son las siguientes:

```python
class Banco:
    def __init__(self, nombre, codigo):
        self.nombre = nombre
        self.codigo = codigo
```

Figura 2. Clase Banco
Fuente: Elaboración propia 2024

```python
class Cliente:
    def __init__(self, nombre, nit):
        self.nombre = nombre
        self.nit = nit
        self.saldo = 0
        self.transacciones = []
        self.pagos = []

    def parseDiccionario(self):
        return {
            'nit': self.nit,
            'nombre': self.nombre,
            'saldo': self.saldo,
            'transacciones': [transaccion.parseDiccionario() for transaccion in self.transacci
            'pagos': [pago.parseDiccionario() for pago in self.pagos]
        }
```

Figura 3. Clase Cliente
Fuente: Elaboración propia 2024

```python
class Factura:
    def __init__(self, numeroFactura, NITcliente, fecha, valor):
        self.numeroFactura = numeroFactura
        self.NITcliente = NITcliente
        self.fecha = fecha
        self.valor = valor

    def parseDiccionario(self):
        return {
            'numeroFactura': self.numeroFactura,
            'NITcliente': self.NITcliente,
            'fecha': self.fecha,
            'valor': self.valor,
        }
```

Figura 4. Clase Factura
Fuente: Elaboración propia 2024

```python
class Pago:
    def __init__(self, codigoBanco, fecha, NITcliente, valor):
        self.codigoBanco = codigoBanco
        self.fecha = fecha
        self.NITcliente = NITcliente
        self.valor = valor

    def parseDiccionario(self):
        return {
            'codigoBanco': self.codigoBanco,
            'fecha': self.fecha,
            'NITcliente': self.NITcliente,
            'valor': self.valor,
        }
```

Figura 5. Clase Pago
Fuente: Elaboración propia 2024

Las clases Cliente, Factura y Pago tienen un método que permite formatearlas a un diccionario y así ser manejadas de mejor manera al mandar la respuesta desde el endpoint.

Los endpoints están contenidos en el archivo main.py en la carpeta backend estos procesan todas las solicitudes que se hagan desde el frontend o en una aplicación externa como postman

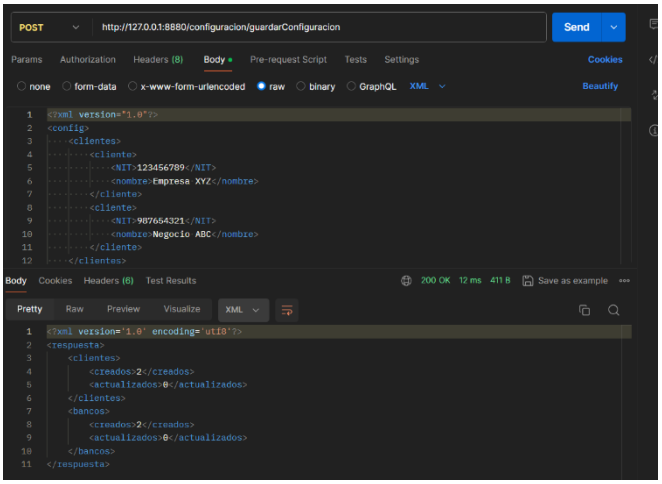

Figura 6. Solicitud para guardar clientes y bancos desde postman
Fuente: Elaboración propia 2024

Los endpoints creados para el manejo de los datos son:

- Limpiar datos [/limpiarDatos]
- Guardar Transacciones [/transaccion/guardarTransaccion]
- Guardar Configuraciones [/configuracion/guardarConfiguracion]
- Devolver Estado de Cuenta [/estado_cuenta/<nit>]
- Devolver Estados de Cuenta (Todos) [/EstadosCuentas]
- Devolver el Resumen de Pago [/estado_cuenta/<nit>/ResumenPago/<fecha>]

El endpoint para devolver el estado de cuenta de un cliente en específico necesitan de un nit valido para funcionar si no se mostrará un mensaje de error en formato JSON, la misma lógica se aplica al endpoint para devolver el resumen de pagos, se necesita de un

nit de un cliente disponible y de una fecha en el formato correcto “dd/yyy” si no retonará un mensaje de error en formato JSON

```python
@app.route("/estado_cuenta/<nit>/ResumenPago/<fecha>", methods=["GET"])
def devolver_resumen_pagos(fecha, nit):
    global ClientesRegistrados
    fecha_formateada = fecha.lower()
    fecha_formateada = fecha_formateada.replace("-", "/")
    mes, anio = fecha_formateada.split('/')
    fecha_formateada = f"{meses_español[mes]}/{anio}"
    fecha_formateada = datetime.strptime(fecha_formateada, '%B/%Y')
    meses_a_verificar = obtener_meses_anteriores(fecha_formateada, 3)
    pagos_por_mes = {f"{mes}/{anio}": [] for mes, anio in meses_a_verificar}
    try:
        for cliente in ClientesRegistrados:
            if cliente.nit == nit:
                for pago in cliente.pagos:
                    fecha_pago_formateada = datetime.strptime(pago.fecha, '%d/%m/%Y')
                    clave_mes_anio = f"{fecha_pago_formateada.month}/{fecha_pago_formateada.ye
                    if (fecha_pago_formateada.month, fecha_pago_formateada.year) in meses_a_ve
                        pagos_por_mes[clave_mes_anio].append(float(pago.valor))
                totales_por_mes = {mes_anio: sum(montos) if montos else 0 for mes_anio, montos
        return jsonify(totales_por_mes)
    except ValueError:
        respuesta = {"msg": "El formato de la fecha es Incorrecto tiene que ser: mm/yyyy",
                     "status_code": 400}
        return jsonify(respuesta)
```

Figura 7. Endpoint para devolver el Resumen de Pagos
Fuente: Elaboración propia 2024

Para la funcionalidad de datos persistentes que se mencionó anteriormente se crean y utilizan archivos XML con nombres “db.clientes” y “db.transacciones” estos se actualizan en cada endpoint dependiendo si es un cliente nuevo, una actualización de un cliente ya existente o la creación de una factura o pago, las facturas y pagos duplicadas y con errores son ignoradas por los archivos y por último, la función para limpiar los archivos, dejándolos sin datos. Las funciones relacionadas a estos archivos se encuentran en el archivo db.py en la carpeta “backend”

```python
def agregar_cliente_DB(cliente, ruta_archivo):
    tree = ET.parse(ruta_archivo)
    root = tree.getroot()

    clientes = root.find('clientes')

    cliente_element = ET.SubElement(clientes, 'cliente')

    ET.SubElement(cliente_element, 'NIT').text = cliente.nit
    ET.SubElement(cliente_element, 'nombre').text = cliente.nombre

    tree.write(ruta_archivo, encoding='utf-8', xml_declaration=True)
```

Figura 8. Función para agregar cliente a la base de datos
Fuente: Elaboración propia 2024

Frontend

Por el lado del frontend se utilizó Django creando un proyecto llamado “frontend” y una aplicación llamada “ITGSA” que es el nombre de la empresa.

Django está configurado para usar el puerto predeterminado: 8000

Se manejaron las URLs disponibles de la página web en el archivo urls.py, conectándolas con las funciones contenidas en el archivo views.py en la carpeta “ITGSA”

```python
urlpatterns = [
    path('', views.home, name='home'),
    path('limpiarDatos', views.reiniciar_datos, name='reiniciar_datos'),
    path('configuracion', views.configuracion, name='configuracion'),
    path('transaccion', views.transaccion, name='transaccion'),
    path('configuracion/guardarConfiguracion', views.guardar_configuracion, name='guardar_configuracion'),
    path('transaccion/guardarTransaccion', views.guardar_transaccion, name='guardar_transaccion'),
    path('Ayuda', views.ayuda, name='ayuda'),
    path('clientes', views.clientes, name='clientes'),
    path('estado_cuenta/', views.estado_cuenta, name='estado_cuenta'),
    path('estado_cuenta/<str:nit>/ResumenPago', views.resumen_pagos, name='resumen_pagos'),
    path('EstadosCuenta', views.EstadosCuenta, name='EstadosCuenta'),
]
```

Figura 9. URLs en el archivo views.py
Fuente: Elaboración propia 2024

Para que el proyecto reconociera la carpeta “ITSGA” se modificó el archivo settings.py de la carpeta principal, añadiendo el nombre de la aplicación:

```python
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'ITGSA',
]
```

Figura 10. Modificación de archivo settings.py
Fuente: Elaboración propia 2024

En la parte de las views.py se crearon funciones en las cuales se nombran en el archivo urls.py cada para manejar una solicitud distinta, algunas simplemente para rendedizar un template, y otras para mandar la

solicitud al backend y obtener una respuesta y dependiendo la respuesta seguir una url u otra

```python
@csrf_exempt
def clientes(request):
    return render(request, 'ITGSA/clientes.html')

@csrf_exempt
def ayuda(request):
    return render(request, 'ITGSA/ayuda.html')
```

Figura 11. Funciones que rendizan templates
Fuente: Elaboración propia 2024

```python
@csrf_exempt
def guardar_configuracion(request):
    if request.method == 'POST':
        archivo = request.FILES['Archivo_configuracion']
        # Enviar la solicitud al backend con el contenido del archivo
        flask_endpoint = 'http://localhost:8880/configuracion/guardarConfiguracion'
        response = requests.post(flask_endpoint, data=archivo.read(), headers={'Content-Type': 'te
        parseado_string = parseString(response.content)
        xml_formateado = parseado_string.toprettyxml(indent= "      ")
        return render(request,'ITGSA/configuracion.html',{
            "respuesta_xml": xml_formateado
        } )
    else:
        # Método no permitido
        return JsonResponse({'error': 'Método no permitido', 'status': 405})
```

Figura 12. Función para guardar configuración
Fuente: Elaboración propia 2024

Dependiendo la respuesta recibida en el backend se procesarán los datos serán enviados al template para mostrarlos dependiendo el caso. En algunos casos se tendrá que formatear los datos del formato XML que es la respuesta que envia el backend a JSON para poder trabajar correctamente sobre ellos en los templates y scripts necesarios.

En la parte de los templates se crearon archivos html para cada dirección web, algunos de cuales tienen también scripts que son partes de código en javascript para el funcionamiento de algunos botones o acciones, por ejemplo, la funcionalidad para imprimir el estado de cuenta en formato .pdf o para generar la gráfica fue realizado en los scripts de los respectivos template con la ayuda de varias librerías como HTML2canvas y chart.js. También se utilizaron CSS frameworks que son plantillas de objetos para darle un mejor aspecto visual a la página.

Conclusiones

La utilización de aplicaciones externas para procesar las solicitudes a modo de prueba es bastante efectiva si se empieza a desarrollar por el lado del backend, ya que solo se necesita la URL del proyecto, el método y, si fuera necesario, información raw que es la información que en este caso se envía en formato XML.

La combinación de dos frameworks puede ser útil no solo para adquirir experiencia en dichos frameworks, sino también para separar, en este caso, Backend y Frontend de una manera más efectiva. Utilizar frameworks específicos para cada parte de la aplicación, como Flask para el backend y Django para el frontend, optimiza el desarrollo al permitir a los desarrolladores trabajar de manera más especializada.

La utilización del formato XML para enviar las respuestas desde el backend al servidor no suele ser la mejor opción, ya que el formato JSON es más fácil de entender y manejar al momento de procesar y mostrar los datos.

Referencias bibliográficas

Pallets. (2024). *Flask Documentation (versión 3.0.x)*. Recuperado de https://flask.palletsprojects.com/en/3.0.x/

Chart.js. (2024). *Chart.js | Open source HTML5 Charts for your website*. Recuperado de https://www.chartjs.org/

Django Software Foundation. (2024). The web framework for perfectionists with deadlines. Recuperado de https://www.djangoproject.com/

npm, Inc. (2022). jsPDF. Recuperado de https://www.npmjs.com/package/jspdf

Anexos

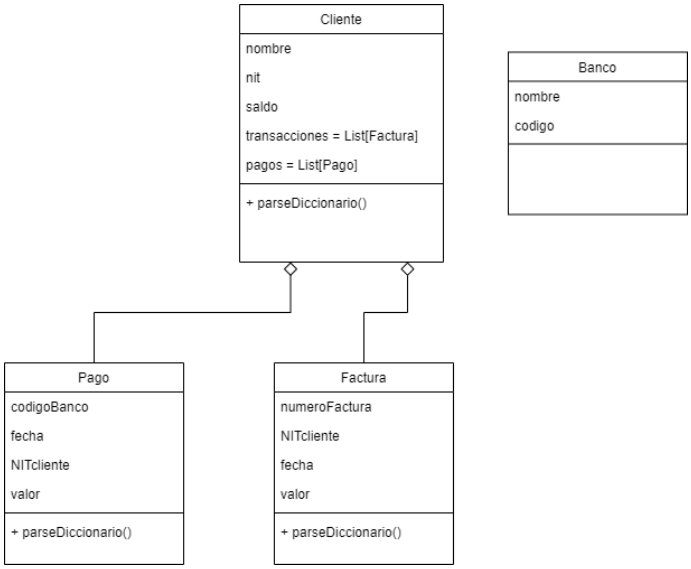

Figura 13. Diagrama de Clases del proyecto
Fuente: Elaboración propia 2024

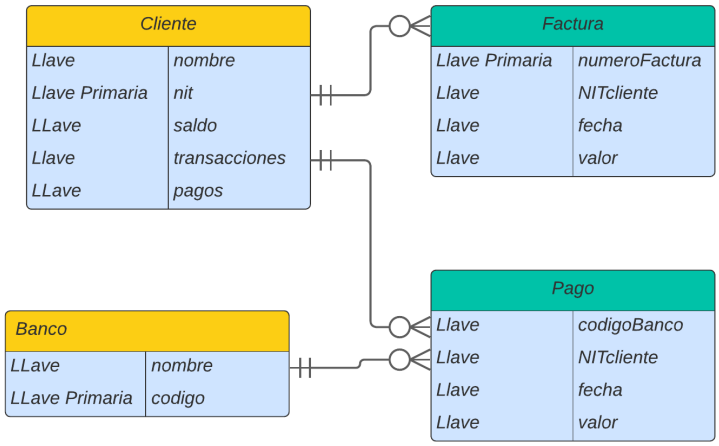

Figura 14. Diagrama Entidad - Relación del proyecto
Fuente: Elaboración propia 2024