Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
Introducción a la programación y computación 2, 2do. Semestre 2025

Sistema de facturación por consumo de recursos en la nube para Tecnologías Chapinas, S.A.

202300502 – Pablo Javier Alvarez Marroquin

Resumen

Este ensayo presenta el diseño e implementación de una solución backend-fronted para gestionar la configuración, uso y facturación de recursos de infraestructura en la nube de Tecnologías Chapinas, S.A. La propuesta procesa mensajes XML de configuración y consumo, almacena datos en archivos XML persistentes, expone una API REST con Flask y ofrece un simulador/cliente con Django para pruebas y reportes. Se emplea programación orientada a objetos para modelar Recurso, Categoría, Configuración, Cliente, Instancia, Consumo y Factura; expresiones regulares extraen y validan fechas y NIT; la facturación agrupa consumos por instancia y recurso para calcular costos por hora. Se generan reportes detallados en PDF y vistas web para operaciones del sistema (inicializar, consultar, crear, facturar). El desarrollo prioriza trazabilidad, validaciones y modularidad, facilitando pruebas con archivos XML de ejemplo y permitiendo auditoría de consumos antes de facturar. Como resultado, la solución satisface los requisitos funcionales del enunciado y proporciona una base extensible para integración futura con bases de datos relacionales.

Palabras clave

Facturación, API REST, XML, POO, validación NIT.

Abstract

This essay describes the design and implementation of a backend-frontend solution to manage configuration, usage and billing of cloud infrastructure resources for Tecnologías Chapinas, S.A. The system processes XML messages for configurations and consumptions, persists data in XML files, exposes a REST API with Flask and provides a Django-based client/simulator for testing and reporting. Object-oriented models represent Resource, Category, Configuration, Client, Instance, Consumption and Invoice; regular expressions extract and validate dates and tax IDs; billing groups consumptions per instance and resource to compute hourly costs. Detailed PDF reports and web views support system operations (reset, query, create, bill). The implementation emphasizes traceability, validation and modularity, enabling testing with sample XML files and auditability before invoicing. The outcome meets the project requirements and

provides an extensible foundation for future relational DB integration.

Keywords
Billing, REST API, XML, OOP, NIT validation

Introducción

La digitalización de servicios en la nube exige soluciones que permitan configurar recursos, medir su uso y facturar con precisión. Este proyecto propone un sistema que automatiza ese ciclo para una pyme local, integrando ingestión de mensajes XML, modelado orientado a objetos y una API que facilita interoperabilidad entre frontend y backend. El objetivo es entregar una implementación reproducible que demuestre buenas prácticas de diseño, validación y persistencia ligera en XML, con énfasis en trazabilidad de consumos y precisión en la facturación. El ensayo documenta el enfoque técnico, decisiones de diseño y resultados funcionales.

Desarrollo del tema

a. Modelado orientado a objetos y persistencia
Se definieron clases que reflejan el dominio: Recurso (id, nombre, métrica, tipo, costo/hora), Configuración (id, recursos y cantidades), Categoría (contiene configuraciones), Cliente (nit, datos, instancias), Instancia (vinculada a configuración, estado y fechas), Consumo (nit, idInstancia, tiempo, fechaHora) y Factura (número, nit, fecha, monto y detalles por instancia). La persistencia seleccionada para el curso fue archivos XML que actúan como "base de datos" por requisitos del enunciado. Se diseñaron formatos XML de salida (db_recursos.xml, db_categorias.xml, db_clientes.xml, db_consumos.xml, db_facturas.xml) y utilidades para leer y escribir manteniendo consistencia de IDs y estructura.

b. Ingesta y parsing de mensajes XML
Se implementó un parser robusto usando xml.etree.ElementTree combinado con expresiones regulares para validar y extraer fechas en formato dd/mm/yyyy y fechas con hora dd/mm/yyyy hh:mm. Reglas implementadas: extraer la primera fecha válida encontrada en un campo libre; aceptar NITs en formato numérico-guion-dígito (0–9 o K); validar tipo de recurso como "Hardware" o "Software"; interpretar consumos en horas con decimales (por ejemplo 1.75). El parser crea objetos de dominio y delega la persistencia a un módulo independiente.

c. API REST y operaciones del sistema
El backend expone endpoints que cubren: recibir configuración (/api/configuracion POST), recibir consumos (/api/consumo POST), inicializar/resetear (/api/reset POST), consultar datos (/api/consultarDatos GET), crear entidades CRUD (recursos, categorías, configuraciones, clientes, instancias) y generar facturas (/api/generarFactura POST). Flask se empleó por su ligereza y la facilidad de montar blueprints para separar rutas. Cada cambio relevante persiste en XML y se registran respuestas claras para facilitar pruebas (p. ej., "3 recursos creados, 1 cliente creado").

d. Lógica de facturación
La facturación agrupa consumos no facturados por cliente y por instancia dentro de un rango de fechas. Para cada instancia se suma el tiempo consumido y se multiplica por el costo por hora de cada recurso según la cantidad definida en la configuración: costo_instancia = sum(recurso.valor_x_hora *

cantidad * horas). Las facturas incluyen un número único generado secuencialmente, NIT del cliente, fecha (último día del rango) y monto total; además se generan detalles por instancia mostrando tiempos, recursos y aportes al monto total. Tras generar la factura, los consumos asociados se marcan como facturados para evitar duplicidad.

e. Frontend, simulación y reportes
El frontend (Django) actúa como simulador y herramienta de pruebas: formularios para subir archivos XML de configuración y consumo, vistas para operaciones del sistema (inicializar, consultar, crear) y secciones para generar/descargar reportes PDF. Los reportes incluyen: detalle de factura (con desglose por instancia y recurso) y análisis de ventas por categoría/configuración o por recurso en un periodo. La generación de PDFs se implementó con ReportLab (o librería equivalente), y se añadió soporte para exportar datos en XML de salida para auditoría.

f. Validaciones, pruebas y casos de ejemplo
Se añadieron validaciones clave (NIT, fechas, estado instancia Vigente/Cancelada, evitar duplicados por ID) y pruebas unitarias básicas para parsers, persistencia y cálculo de facturas. Se incluyeron archivos de ejemplo: archivoConfiguraciones_ejemplo.xml y listadoConsumos_ejemplo.xml para reproducir escenarios de prueba y validar resultados de facturación.

Conclusiones

La solución implementada cumple los requisitos del enunciado: ingestión y validación de mensajes XML, modelado OOP claro, API REST para operaciones del sistema, persistencia en archivos XML y cálculo de facturación con desglose por instancia y recurso. El diseño modular facilita extensión futura (migrar a DB relacional, añadir autenticación, integrar pasarela de pago). Las validaciones implementadas reducen riesgo de facturación errónea y las rutinas para marcar consumos como facturados garantizan idempotencia. Recomendaciones: agregar pruebas automatizadas más completas, añadir control de errores y logging estructurado, y evaluar migración a base de datos relacional para escalabilidad y concurrencia en escenarios reales.

Referencias bibliográficas

Máximo 5 referencias en orden alfabético.

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

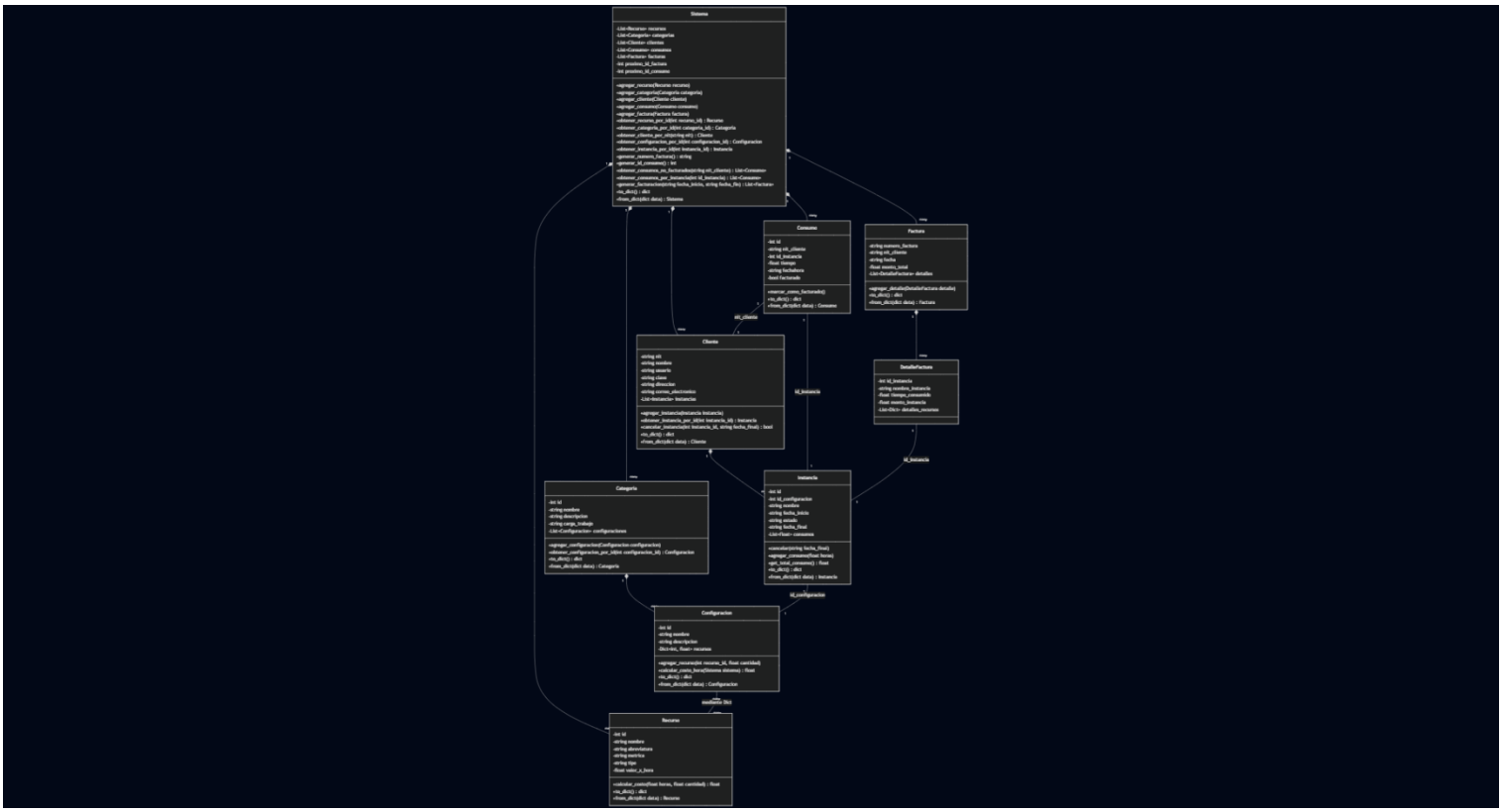
Extensión: de cuatro a siete páginas como máximo

Flask Documentation.

Python xml.etree.ElementTree documentation.

ReportLab User Guide.

C. J. Date (1991). An Introduction to Database Systems.
Addison-Wesley.



```
def simulate():
        return "Invernadero no encontrado", 404
    plan = None
    for p in gh.planes:
        if p.nombre == plan_name:
            plan = p
            break
    if plan is None:
        return "Plan no encontrado", 404
    tiempo_opt, timeline, eficiencia, snapshots = simulate_plan(gh, plan)
    gh.snapshots = snapshots
    html_path = os.path.join(REPORTS_FOLDER, f"reporte_{gh.nombre}_{plan.nombre}.html".replace("
    generate_html_report(gh, plan, tiempo_opt, eficiencia, timeline, html_path)
    output_xml_path = os.path.join(REPORTS_FOLDER, f"salida_{gh.nombre}_{plan.nombre}.xml".replac
    generate_output_xml([gh], [(gh, [(plan, tiempo_opt, eficiencia, timeline)])], output_xml_path
    assignments = []
    for d in gh.drones:
        assignments.append({'nombre': d.nombre, 'hilera': d.hilera, 'posicion': d.posicion})
    return render_template('simulate.html', gh=gh, plan=plan, tiempo=tiempo_opt, eficiencia=efici

@app.route('/tda_form', methods=['GET','POST'])
def tda_form():
    global _loaded_greenhouses
    if not _loaded_greenhouses:
        return redirect(url_for('index'))
```