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 AUTOMATIZADA PARA INFRAESTRUCTURA DE NUBE

202303204 – Christian Javier Rivas Arreaga

Resumen

El presente ensayo expone el desarrollo de un sistema de facturación automatizada para servicios de infraestructura en la nube, implementado mediante una arquitectura cliente-servidor utilizando tecnologías web modernas.

El sistema permite la gestión integral de recursos computacionales, configuraciones de máquinas virtuales, registro de clientes e instancias, así como el cálculo y facturación detallada del consumo de recursos. La solución implementa una API RESTful desarrollada en Flask para el backend, una aplicación web en Django para el frontend, y utiliza archivos XML como mecanismo de persistencia de datos.

Se destacan los beneficios de la arquitectura desacoplada, la importancia de las expresiones regulares para validación de datos, y la generación automatizada de reportes PDF. El sistema representa una solución práctica y escalable para empresas proveedoras de servicios cloud que requieren facturación detallada y transparente de recursos aprovisionados, contribuyendo a la eficiencia operativa y la satisfacción del cliente mediante reportes precisos y auditables.

Palabras clave: Facturación cloud, API REST, Django, Flask, Persistencia XML.

Abstract

This essay presents the development of an automated billing system for cloud infrastructure services, implemented through a client-server architecture using modern web technologies.

The system enables comprehensive management of computational resources, virtual machine configurations, client registration and instances, as well as detailed calculation and billing of resource consumption. The solution implements a RESTful API developed in Flask for the backend, a web application in Django for the frontend, and uses XML files as a data persistence mechanism. The benefits of decoupled architecture, the importance of regular expressions for data validation, and automated PDF report generation are highlighted.

The system represents a practical and scalable solution for cloud service providers requiring detailed and transparent billing of provisioned resources, contributing to operational efficiency and customer satisfaction through accurate and auditable reports.

Keywords: *Cloud billing, REST API, Django, Flask, XML Persistence*

Introducción

La computación en la nube ha transformado radicalmente la forma en que las organizaciones consumen recursos tecnológicos, permitiendo el aprovisionamiento dinámico y escalable de infraestructura computacional.

Este paradigma exige sistemas de facturación precisos que calculen los costos según el uso real de recursos como procesadores, memoria RAM y almacenamiento. El presente ensayo documenta el diseño e implementación de un sistema integral de facturación para servicios cloud, desarrollado como parte del proyecto académico del curso de Introducción a la Programación y Computación 2. Se explora la arquitectura REST, los patrones de diseño aplicados, las tecnologías web utilizadas y los desafíos técnicos superados.

El objetivo es demostrar cómo las tecnologías modernas permiten construir soluciones empresariales robustas, escalables y mantenibles que resuelven problemas reales del sector tecnológico.

Desarrollo del tema

1. Arquitectura del Sistema

El sistema implementa una arquitectura cliente-servidor desacoplada, separando claramente las responsabilidades entre presentación (frontend) y lógica de negocio (backend). Esta separación permite desarrollo independiente, escalabilidad horizontal y facilita el mantenimiento evolutivo.

Backend - API RESTful en Flask:

El servidor backend, desarrollado en Flask (Grinberg, 2018), expone 22 endpoints REST que implementan operaciones CRUD (Create, Read, Update, Delete) sobre las entidades del sistema. Flask fue seleccionado por su simplicidad, flexibilidad y excelente soporte para desarrollo de APIs. La arquitectura sigue el patrón MVC (Modelo-Vista-Controlador) adaptado al contexto de servicios web.

Frontend - Aplicación Web en Django:

La interfaz de usuario utiliza Django siguiendo el patrón MVT (Modelo-Vista-Template). Django proporciona un framework completo con sistema de plantillas robusto, enrutamiento URL elegante y middleware para seguridad CSRF. Las vistas consumen la API REST del backend mediante la librería requests, actuando como cliente HTTP.

La interfaz fue diseñada utilizando Tailwind CSS, un framework utility-first que permite crear interfaces modernas y responsivas sin escribir CSS personalizado. Se implementó un tema oscuro para reducir la fatiga visual y mejorar la experiencia del usuario durante sesiones prolongadas.

2. Modelo de Datos y Persistencia XML

Decisión de Arquitectura:

A diferencia de bases de datos relacionales tradicionales (PostgreSQL, MySQL) o NoSQL (MongoDB), el proyecto utiliza archivos XML como mecanismo de persistencia. Esta decisión se justifica por:

- **Portabilidad:** Los archivos XML son autocontenidos y fácilmente transportables
- **Legibilidad humana:** Facilita la depuración y auditoría manual
- **Estándar abierto:** XML es un estándar W3C ampliamente soportado
- **Validación mediante XSD:** Permite definir esquemas que garantizan integridad estructural

3. Validación de Datos con Expresiones Regulares

La integridad de los datos es crítica en sistemas de facturación. Se implementaron expresiones regulares para validar.

Estas expresiones regulares garantizan que solo datos con formato válido sean procesados por el sistema,

previniendo errores de cálculo y asegurando la trazabilidad de las transacciones.

4. Proceso de Facturación

El algoritmo de facturación constituye el núcleo del sistema y sigue estos pasos:

1. **Filtrado temporal:** Selecciona consumos dentro del rango de fechas especificado
2. **Agrupación por cliente:** Organiza consumos pendientes por NIT
3. **Cálculo de montos:** Aplica la fórmula de costo total por configuración
4. **Generación de factura:** Crea registro con número único secuencial (formato: FAC-XXXXXX)
5. **Marcado de consumos:** Actualiza el estado invoiced para evitar facturación duplicada

Fórmula de Cálculo:

El monto total de una factura se calcula mediante la suma de los costos de todos los recursos consumidos:

$$\text{Monto} = \Sigma(\text{Cantidad}_i \times \text{CostoHora}_i \times \text{Horas}_i)$$

donde:

- **Cantidad_i** = cantidad del recurso i en la configuración (ej: 4 GiB de RAM)
- **CostoHora_i** = costo por hora del recurso i (ej: $0.75 por GiB)
- **Horas_i** = tiempo total de consumo en horas
- **i** = índice del recurso (1 hasta n recursos en la configuración)

Por ejemplo, una configuración con 4 GiB de RAM ($0.75/hora) y 2 núcleos ($1.20/hora) consumida durante 2.5 horas genera:

$$\text{Monto} = (4 \times 0.75 \times 2.5) + (2 \times 1.20 \times 2.5)$$
$$= 7.5 + 6.0$$
$$= \$13.50$$

5. Generación de Reportes PDF

Se implementaron tres tipos de reportes utilizando la librería ReportLab, que permite la creación programática de documentos PDF con control total sobre el diseño y contenido.

A. Detalle de Factura:

Presenta la descomposición completa de una factura específica, incluyendo:

- Datos del cliente (nombre, NIT, dirección, email)
- Información de la factura (número, fecha, monto total)
- Listado de consumos individuales con fecha/hora exacta
- Detalle de recursos utilizados por instancia
- Costo unitario y subtotal por cada recurso
- Subtotales por instancia y total general

Este reporte es crucial para la transparencia y permite a los clientes auditar exactamente qué recursos consumieron y cuánto les costó cada uno.

B. Análisis de Ventas por Categorías:

Identifica qué configuraciones generan más ingresos para la empresa, calculando:

- Ingresos totales por categoría/configuración
- Porcentaje de contribución al ingreso total
- Ranking de configuraciones más rentables

Este análisis permite a la empresa enfocar esfuerzos de marketing y desarrollo en las configuraciones más demandadas.

C. Análisis de Ventas por Recursos:

Determina qué recursos individuales son más rentables:

- Desglose de ingresos por tipo de recurso (RAM, CPU, almacenamiento, etc.)
- Identificación de recursos de mayor margen
- Tendencias de consumo

Esta información guía decisiones sobre precios y aprovisionamiento de recursos físicos.

6. Programación Orientada a Objetos

El sistema utiliza el paradigma de Programación Orientada a Objetos (POO) mediante clases y dataclasses de Python para modelar las entidades del dominio.

Conclusiones

El desarrollo del sistema de facturación para infraestructura cloud demuestra la viabilidad de

construir soluciones empresariales robustas utilizando tecnologías web modernas y arquitecturas desacopladas. La separación entre frontend (Django) y backend (Flask) permite escalabilidad independiente y facilita el mantenimiento evolutivo, siguiendo principios de diseño ampliamente aceptados en la industria del software.

El uso de XML como mecanismo de persistencia, aunque poco convencional en sistemas modernos que prefieren bases de datos relacionales o NoSQL, resultó adecuado para el alcance académico del proyecto. Esta decisión facilitó la comprensión de conceptos fundamentales de almacenamiento estructurado, validación de esquemas mediante XSD, y navegación de datos jerárquicos mediante XPath.

Las expresiones regulares demostraron ser herramientas esenciales para validación y extracción de datos en formatos flexibles. La capacidad de extraer fechas de textos arbitrarios ("Guatemala, 15/01/2025 se emitió") garantiza robustez ante variaciones en los datos de entrada, un requisito común en sistemas de integración empresarial.

La implementación de Programación Orientada a Objetos mediante clases Python con encapsulación de datos y comportamiento facilita el mantenimiento y extensibilidad del código. La separación de responsabilidades en módulos distintos (models, services, validators) sigue el principio de responsabilidad única de SOLID.

Referencias bibliográficas

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine.

Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python* (2nd ed.). O'Reilly Media.

Percival, H., & Gregory, B. (2020). *Architecture Patterns with Python: Enabling Test-Driven Development, Domain-Driven Design, and Event-Driven Microservices*. O'Reilly Media.

Ronacher, A. (2024). *Flask Documentation* (Version 3.0). https://flask.palletsprojects.com/

The Django Software Foundation. (2024). *Django Documentation* (Version 4.2). https://docs.djangoproject.com/