

DESENVOLVIMENTO DE UM SISTEMA DE GESTÃO COMERCIAL EM PYTHON : APLICAÇÃO DE ARQUITETURA MODULAR E BANCO DE DADOS RELACIONAL

Gustavo de Lima
Fernando Bordin

Resumo

O presente artigo apresenta o desenvolvimento de um **Sistema de Gestão Comercial** voltado ao controle de **ordens de serviço (OS)** em assistências técnicas de computadores, utilizando a linguagem **Python**, interface gráfica com **Tkinter** e banco de dados **PostgreSQL**. O sistema foi projetado segundo uma **arquitetura modular**, permitindo a separação de responsabilidades entre as camadas de interface, lógica de negócio, persistência e utilitários, o que facilita a manutenção e a expansão futura do software. A metodologia de desenvolvimento adotou princípios incrementais e iterativos, com foco na legibilidade, testabilidade e integração entre os componentes.

O banco de dados foi modelado sob o paradigma relacional, aplicando **funções PL/pgSQL** e **triggers** para automação de processos, como a geração sequencial de identificadores de OS e validação de consistência dos registros. Além disso, a integração com a biblioteca **FPDF2** possibilitou a emissão automatizada de relatórios e recibos em formato PDF, garantindo padronização e rastreabilidade documental.

Os resultados obtidos demonstraram ganhos significativos de **eficiência operacional, segurança da informação e confiabilidade administrativa**, validando a aplicação prática da arquitetura modular proposta. Conclui-se que a simplicidade tecnológica adotada oferece uma base sólida para evolução do sistema, permitindo a futura implementação de módulos complementares, como controle de estoque e gestão financeira.

Palavras-chave: Automação comercial. Python. PostgreSQL. Arquitetura modular. Sistema de gestão.

1. INTRODUÇÃO

A informatização dos processos empresariais consolidou-se como um pilar essencial no cenário organizacional contemporâneo, transcendendo a mera digitalização de registros e tornando-se um **diferencial estratégico**. Em ambientes de alta competitividade, a dependência de métodos manuais, planilhas dispersas e controles não integrados compromete a **precisão administrativa** e a **eficiência operacional**. Por outro lado, a adoção de sistemas automatizados permite a **integração de setores cruciais** — como vendas, estoque, finanças e atendimento

— resultando na centralização da informação e em uma gestão em tempo real. Essa automação, além de otimizar a produtividade e reduzir custos, aprimora a **tomada de decisão gerencial**, apoiada na geração de relatórios e indicadores de desempenho (KPIs).

Diante dessa realidade, as tecnologias de desenvolvimento de software assumem papel preponderante. A linguagem **Python** destaca-se por sua **sintaxe simples, ampla comunidade** e vasta coleção de bibliotecas que facilitam a integração com **interfaces gráficas** e **bancos de dados relacionais**. A utilização do **PostgreSQL**, por sua vez, assegura a **integridade, segurança e consistência dos dados empresariais**, oferecendo recursos avançados de automação por meio da linguagem **PL/pgSQL**. A combinação entre a flexibilidade do Python e a robustez do PostgreSQL constitui uma base técnica sólida para o desenvolvimento de sistemas de gestão comerciais modernos e escaláveis.

Entretanto, muitas **pequenas e médias empresas** ainda enfrentam o desafio de implementar soluções personalizadas, confiáveis e sustentáveis, sem depender de sistemas monolíticos e inflexíveis. Surge, assim, a necessidade de desenvolver **aplicações modulares**, que possam evoluir de forma independente e organizada, adaptando-se facilmente às demandas específicas de cada negócio.

Diante desse contexto, o presente artigo propõe o **desenvolvimento de um Sistema de Gestão Comercial em Python**, com aplicação de **arquitetura modular e banco de dados relacional**. O objetivo central é projetar e implementar uma aplicação desktop funcional que automatize os processos de **cadastro de clientes, controle de ordens de serviço e geração de relatórios administrativos**, garantindo escalabilidade, desempenho e facilidade de manutenção.

A justificativa do estudo está na **viabilidade técnica e didática** da aplicação da arquitetura modular em Python, demonstrando como é possível criar soluções de alto desempenho com baixo custo de implementação e manutenção. A adoção do banco de dados relacional reforça a segurança e a integridade dos dados, assegurando a confiabilidade necessária para o ambiente comercial.

O artigo está estruturado da seguinte forma: a **Seção 2** apresenta a metodologia de desenvolvimento; a **Seção 3** detalha a arquitetura de persistência e o banco de dados; a **Seção 4** discute a automação de relatórios e logs; a **Seção 5** apresenta os resultados e benefícios observados; e, por fim, a **Seção 6** traz as considerações finais e perspectivas futuras.

2. METODOLOGIA DE DESENVOLVIMENTO

O desenvolvimento do sistema, denominado **GF Informática**, baseou-se em uma metodologia que combina os princípios de **design modular** com a filosofia de

simplicidade e legibilidade da linguagem **Python**. Essa abordagem visou garantir não apenas a funcionalidade do software, mas também sua **escalabilidade, manutenibilidade e transparência do código**.

2.1 Características da Metodologia com Python

A linguagem **Python** foi escolhida por sua aderência aos princípios de **simplicidade, clareza e modularidade**, características que favorecem o desenvolvimento incremental e colaborativo. As principais diretrizes adotadas foram:

- **Desenvolvimento incremental e iterativo:** o sistema foi construído por partes independentes (módulos e funções), que puderam ser testadas isoladamente, minimizando riscos de regressão.
- **Legibilidade e simplicidade do código:** a sintaxe de alto nível facilitou a compreensão e a manutenção do projeto em equipe.
- **Supporte à modularização:** a estrutura da linguagem incentiva a separação lógica de responsabilidades por meio de pacotes e módulos, promovendo o reuso de código.
- **Capacidade de integração tecnológica:** o ecossistema de bibliotecas Python permitiu integração eficiente com o **PostgreSQL**, além de possibilitar o uso de ferramentas como **Tkinter** para a interface e **FPDF2** para relatórios.
- **Testabilidade e manutenção:** a arquitetura estruturada facilitou a criação de testes unitários e o processo de depuração (debug).

2.2 Arquitetura Modular e Organização do Projeto

A organização do projeto foi definida pela **arquitetura modular**, com separação clara de responsabilidades (Separation of Concerns), conforme ilustrado a seguir:

Módulo / Diretório	Função no Sistema	Benefício Arquitetural
--------------------	-------------------	------------------------

database/	Scripts de conexão e queries SQL para PostgreSQL	Facilita a manutenção da camada de persistência e permite substituição do SGBD
services/	Lógica de negócio (CRUD de ordens de serviço e clientes)	Mantém a lógica de negócio isolada da interface gráfica
ui/	Interface gráfica desenvolvida com Tkinter	Permite alterar o layout sem afetar o núcleo lógico
utils/	Funções auxiliares (validação, formatação, geração de relatórios PDF)	Favorece o reuso e a padronização do código

3. ARQUITETURA DE PERSISTÊNCIA E BANCO DE DADOS RELACIONAL

O sistema utiliza o **PostgreSQL** como SGBD principal, devido à sua **maturidade, segurança e suporte ao padrão SQL**.

3.1 Estrutura e Integridade dos Dados

O banco foi modelado com base no paradigma **relacional**, seguindo normas de **normalização**. As principais entidades são **usuários, clientes e ordens de serviço**, interligadas por chaves estrangeiras para garantir a integridade referencial.

Constraints e índices otimizam o desempenho e a segurança, enquanto os campos de **timestamp** oferecem rastreabilidade e auditoria automática.

3.2 Automação com PL/pgSQL e Triggers

Foram implementadas **funções PL/pgSQL e triggers** para automatizar tarefas de persistência.

A função `gerar_numero_os()` utiliza um **objeto SEQUENCE** para gerar identificadores únicos no formato “OS0001”, acionada antes da inserção de novos registros — garantindo controle e consistência automatizada.

4. AUTOMAÇÃO E RELATÓRIOS

A capacidade de transformar dados em informações úteis foi um dos pilares do sistema.

4.1 Relatórios em PDF com FPDF2

A biblioteca **FPDF2** é utilizada para gerar relatórios (ordens de serviço, recibos) de forma programática, centralizada na classe `OSPDFGenerator`.

Os relatórios incluem cabeçalho, metadados, corpo formatado e rodapé com dados de auditoria.

4.2 Log e Auditoria de Eventos

O sistema também implementa **registros de log** na pasta `logs/`, armazenando informações sobre login, criação e atualização de ordens de serviço, e eventuais erros.

Esse histórico funciona como **ferramenta de diagnóstico e controle de segurança**, garantindo rastreabilidade das ações administrativas.

5. RESULTADOS E BENEFÍCIOS

A aplicação apresentou ganhos expressivos em **eficiência, confiabilidade e organização**, conforme resumido:

Área de Resultado	Resultado Prático	Benefício Principal
Confiabilidade Operacional	Padronização e rastreabilidade das OSs	Redução de erros e aumento da agilidade
Automação de Documentos	Emissão automática de relatórios PDF	Profissionalização e transparência dos processos
Segurança da Informação	Validação e logs integrados ao PostgreSQL	Mitigação de falhas e inconsistências
Manutenção e Expansão	Estrutura modular	Facilidade de atualização e evolução futura
Gestão Estratégica	Métricas e relatórios gerenciais	Apoio à tomada de decisão

6. CONCLUSÃO

O presente estudo atingiu seu objetivo ao demonstrar a viabilidade de um **sistema desktop modular** em Python com **integração relacional via PostgreSQL**.

A arquitetura modular e o uso de triggers PL/pgSQL garantiram **consistência e automação de processos**, enquanto a geração de relatórios e o sistema de logs consolidaram o caráter profissional da aplicação.

Conclui-se que a **simplicidade tecnológica adotada** não representa limitação, mas uma base sólida para evolução. O modelo proposto permite futuras expansões, como **módulos de estoque e financeiro**, além da migração para frameworks modernos de interface, ampliando a acessibilidade e a usabilidade do sistema.

7. REFERÊNCIAS

SOUZA, Luan R.; MENDES, Gabriel F. *Desenvolvimento de sistemas de gestão comercial para microempresas utilizando Python e banco de dados relacional*. *Revista Científica Multidisciplinar Núcleo do Conhecimento*, v. 7, n. 10, p. 157–170, 2023.

PYTHON SOFTWARE FOUNDATION. *The Python Language Reference*. Disponível em: <https://docs.python.org/3/>. Acesso em: 9 out. 2025.

POSTGRESQL GLOBAL DEVELOPMENT GROUP. *PostgreSQL 15 Documentation*. Disponível em: <https://www.postgresql.org/docs/>. Acesso em: 9 out. 2025.

REPORTLAB DEVELOPERS. *ReportLab User Guide*. Disponível em: <https://www.reportlab.com/docs/reportlab-userguide.pdf>. Acesso em: 9 out. 2025.

PYFPDF COMMUNITY. *FPDF2 Documentation*. Disponível em: <https://pyfpdf.github.io/fpdf2/>. Acesso em: 9 out. 2025.

PYTHON SOFTWARE FOUNDATION. *Tkinter — Python Interface to Tcl/Tk*. Disponível em: <https://docs.python.org/3/library/tkinter.html>. Acesso em: 9 out. 2025.

Abstract

This paper presents the development of a **Commercial Management System** designed to control **service orders (OS)** in computer technical assistance environments, using the **Python** programming language, graphical interface with **Tkinter**, and **PostgreSQL** relational database. The system was designed following a **modular architecture**, ensuring a clear separation between the interface, business logic, data persistence, and utility layers, which facilitates maintenance and future scalability. The development methodology followed incremental and iterative principles, emphasizing readability, testability, and integration among components.

The database was modeled under the relational paradigm, implementing **PL/pgSQL functions** and **triggers** to automate processes such as sequential OS number generation and data consistency validation. In addition, the integration with the **FPDF2** library enabled automated generation of reports and receipts in PDF format, ensuring standardization and document traceability.

The results demonstrated significant improvements in **operational efficiency, data security, and administrative reliability**, validating the practical application of the proposed modular architecture. The study concludes that the technological simplicity adopted provides a solid foundation for future system evolution, allowing for the implementation of additional modules such as inventory and financial management.

Keywords: Commercial automation. Python. PostgreSQL. Modular architecture. Management system.