Sistema de Gerenciamento de Compras para Repúblicas Estudantis

1st Arthur Pires
Engenharia de Controle e Automação
Universidade Federal de Minas Gerais
Belo Horizonte, Brasil
arthurfgp@ufmg.br

3rd Gabriel Groppo
Engenharia de Controle e Automação
Universidade Federal de Minas Gerais
Belo Horizonte, Brasil
gabrielgroppo@ufmg.br

2nd Caio Veras
Engenharia de Controle e Automação
Universidade Federal de Minas Gerais
Belo Horizonte, Brasil
caiocv@ufmg.br

4th Maycon Oliveira
Engenharia de Controle e Automação
Universidade Federal de Minas Gerais
Belo Horizonte, Brasil
maycon2014@ufmg.br

***Resumo*—O artigo apresenta um sistema de gerenciamento de compras para repúblicas estudantis, utilizando conceitos de programação orientada à objetos e integração com banco de dados, garantindo simplicidade e eficiência na organização financeira residencial, e conformidade nos interesses coletivos e individuais dos moradores. A aplicação desenvolvida permite gerenciar listas de compras e controle de estoque, com monitoramento de dívidas e administração de usuários.**

I. Introdução

Nos últimos anos, a escolha de um local para morar durante a vida universitária tem se mostrado um desafio significativo para muitos estudantes, especialmente para aqueles que se mudam de suas cidades natais. As repúblicas estudantis surgem como uma solução popular, oferecendo um ambiente compartilhado onde indivíduos de diferentes regiões e contextos culturais convivem. Este ambiente diversificado, embora enriquecedor, traz consigo uma série de desafios, particularmente no que diz respeito à organização das responsabilidades domésticas e financeiras.

Em muitas repúblicas, a administração financeira é frequentemente informal e desorganizada. Cada morador pode ter diferentes níveis de comprometimento e hábitos financeiros, o que pode levar a desentendimentos e confusões. Por exemplo, é comum que um único morador assuma a responsabilidade de realizar compras coletivas para a casa, enquanto outros podem precisar de compras individuais em momentos diferentes. Sem um sistema claro e eficiente, essas tarefas podem resultar em desorganização, falta de transparência e até mesmo em conflitos entre os moradores.

Para enfrentar esses desafios, desenvolvemos um sistema de gerenciamento de compras para repúblicas estudantis, com foco na organização e eficiência das compras coletivas e individuais. Este sistema utiliza conceitos de programação orientada a objetos e integração com bancos de dados, proporcionando um sistema colaborativo de fácil usabilidade.

Os códigos utilizados estão disponíveis em um repositório do GitHub [3] que pode ser acessado por meio deste link: Stock Caro. O mesmo inclui arquivo "README.md"com orientações de uso para teste do projeto.

A. Principais Contribuições

A partir do exposto acima, as principais contribuições deste estudo são:

- Desenvolvimento de um modelo de gestão financeira para repúblicas estudantis utilizando conceitos avançados de programação orientada a objetos e integração com banco de dados;
- Implementação de uma aplicação em uma interface amigável que permite que moradores, sem conhecimento em programação, utilizem a ferramenta de forma eficiente;

B. Organização do Trabalho

O restante deste trabalho está organizado da seguinte forma. A seção 2 apresenta um referencial teórico revisando conceitos utilizados no projeto. Já na seção 3 é apresentada uma metodologia empregada e ferramentas utilizadas ao desenvolver o sistema, bem como a estrutura geral dos códigos. Em seguida, a seção 4 apresenta detalhes do banco de dados, que é central no projeto, além de ilustrar a interface utilizada e sua construção. Finalmente, a seção 6 traz uma conclusão com testes empregados para demonstrar a completude do trabalho.

II. Referencial Teórico

A. Programação Orientada à Objetos

A Programação Orientada à Objetos (POO) é um paradigma de desenvolvimento de software que organiza código em torno de objetos, que encapsulam dados e comportamentos. Seus pilares são herança, encapsulamento, polimorfismo e abstração. A herança promove a reutilização de código, o encapsulamento protege dados, o polimorfismo permite flexibilidade e

a abstração simplifica a complexidade. A POO é amplamente utilizada em diversas áreas da computação, desde sistemas complexos até aplicações móveis, devido à sua capacidade de criar software robusto, modular e fácil de manter. Essa abordagem facilita a colaboração entre desenvolvedores e a adaptação às mudanças de requisitos.

B. Banco de Dados

Os bancos de dados são sistemas fundamentais para o armazenamento, organização e recuperação eficiente de dados. Eles permitem a gestão estruturada de grandes volumes de informações, essenciais para o funcionamento de aplicações modernas. Entre os tipos mais comuns estão os bancos de dados relacionais, que utilizam tabelas para representar dados e suas relações, como foi utilizado neste projeto. Os bancos de dados relacionais utilizam a linguagem SQL (Structured Query Language) para a definição e manipulação dos dados, garantindo integridade e consistência através de transações e constraints. Ademais, são essenciais para o suporte de operações críticas em diversas áreas, como sistemas financeiros, por exemplo, pois se trata de uma ferramenta que garante acesso rápido e confiável às informações.

C. Interface de Usuário (GUI)

As Interfaces Gráficas de Usuário (GUI) são componentes fundamentais no desenvolvimento de software, proporcionando uma interação intuitiva e visual entre os usuários e os sistemas computacionais. Elas são projetadas para facilitar a usabilidade e melhorar a experiência do usuário, oferecendo elementos visuais como janelas, ícones, menus e botões. As GUIs permitem que os usuários executem tarefas complexas de maneira eficiente e com uma curva de aprendizado reduzida, eliminando a necessidade de memorizar comandos textuais.

D. Trabalhos Correlatos

Dado a eficiência dos conceitos supracitados, sobretudo em fatores como aplicabilidade e modularização, é válida a integração desses elementos para soluções tecnológicas simples à avançadas, que atendem às necessidades de armazenamento, manipulação e visualização de dados. Para contextualizar, no trabalho desenvolvido em [1] discute a arquitetura necessária para integrar sistemas de gerenciamento de banco de dados (DBMS) com linguagens de programação orientada a objetos (OOP). Os autores abordam os desafios e as soluções para gerenciar a persistência e a recuperação de dados em ambientes orientados a objetos. A pesquisa destaca a importância de um design que facilite a interoperabilidade entre o armazenamento de dados e a manipulação de objetos, permitindo que dados persistentes possam ser facilmente integrados e manipulados em aplicações desenvolvidas com OOP. Os autores propuseram uma arquitetura que utiliza conceitos avançados de POO para criar um sistema de banco de dados orientado a objetos que suporta herança, encapsulamento e polimorfismo. Eles implementaram uma camada de abstração que permite que os objetos sejam armazenados e recuperados diretamente do banco de dados, eliminando a necessidade de conversão entre objetos e registros de banco de dados. Essa abordagem inovadora demonstrou melhorias significativas na eficiência e na manutenção de sistemas complexos, proporcionando uma integração mais natural entre a lógica de aplicação e a persistência de dados.

Além disso, no trabalho evidenciado em [2] os autores exploram os avanços modernos nos sistemas de gerenciamento de banco de dados (DBMS) e enfatizam a eficiência e a eficácia desses sistemas quando integrados com o desenvolvimento de aplicações. A pesquisa aborda como as interfaces gráficas de usuário (GUIs) podem melhorar significativamente a interação dos usuários com sistemas complexos de banco de dados.

Os pesquisadores desenvolveram um sistema que combina um DBMS robusto com uma GUI intuitiva, permitindo que usuários executem consultas complexas e gerenciem dados com facilidade. A interface gráfica foi projetada para ser altamente responsiva e personalizável, suportando arrastar e soltar, visualizações em tempo real e feedback imediato. Essa integração resultou em um aumento significativo na produtividade dos usuários e na satisfação com o sistema, demonstrando como a combinação de DBMS e GUI pode simplificar a gestão de dados e melhorar a experiência do usuário.

III. Metodologia

Inicialmente, a ideia do projeto seguia o propósito de materializar uma forma de gerenciamento de compras, menos rudimentar como costuma ocorrer na maioria das repúblicas estudantis. No entanto, após uma análise contextual da realidade relatada de muitos moradores, concluiu-se que uma abordagem mais ampla fosse necessária, como administração de usuários e da própria residência, visto que é de costume repúblicas de proporções maiores terem hierarquias estruturadas. Bem como, que fosse possível haver uma integração nos interesses coletivos da residência, que envolvam materiais ou alimentos compartilhados, com interesses individuais de cada morador, ou seja, compras para uso pessoal. Sendo assim, de forma que seja possível integrar um estoque, registrado em um banco de dados, de fácil acesso para ambos casos. Dessa forma,

A. Ferramentas

As ferramentas utilizadas para o desenvolvimento do sistema foram as seguintes:

- **Python 3.12**, como linguagem de programação utilizada no desenvolvimento dos códigos disponíveis em [2].
- **Git e Github**, para controle de versão e colaboração no desenvolvimento do código, utilizado como referência para consulta.
- **SQLite**, utilizado para construção de um banco de dados relacional leve, com a finalidade de armazenar os dados da execução.
- **Tkinter**, como uma das bibliotecas Python para modelar a interface gráfica (GUI) utilizada.

B. Estrutura geral do código

O código foi desenvolvido com bibliotecas padrões do Python para auxiliar no desenvolvimento além da própria Tkinter, como a ”mysql.connector”que permite a interação entre o Python e o Banco de Dados MySQL. Bem como bibliotecas presentes no módulo ”typing”. *Toda a arquitetura dos códigos é acessada em [3] juntamente com o diagrama UML (Unified Modeling Language) ilustrando as classes e seus métodos.*

1) Scripts:

- **usuario.py**, que implementa tanto manipulação de dados pessoais do usuário, relação com produtos, sobretudo os enquadrados em compras individuais, como também manejamento de dívidas;
- **residencia.py**, que envolve integração de dispensa, compras, criação e exclusão de residência da interface;
- **produto.py**, compõe o manejo de produtos com métodos como criar produto e deletar produto, além do detalhamento de informações como id, categoria, quantidade e registro em estoque da residência;
- **lista.py**, maneja os produtos registrados em lista de compras;
- **dispensa.py**, registra justamente os produtos presentes em estoque, não avaliados como itens individuais, sendo possível que um administrador altere suas definições;
- **repository.py**, importa a biblioteca ”mysql.connector”por meio da classe Database para conexão com o banco de dados, o que permite a manipulação de dados presentes no mesmo.
- **GUI.py**, script responsável por integrar a interface gráfica, construindo frames visuais das telas, e configurações de botões e blocos com visualização de informações.

2) Classes Principais:

- **Database**: Classe base responsável pela conexão com o banco de dados;
- **UserRepository**: Classe base para repositórios, utilizando a funcionalidade de classe abstrata, verifica diversas condições booleanas e trata informações a serem apresentadas na GUI;
- **Usuario**: Também instanciável, determina, por meio de seus métodos, dados do usuário e suas compras, registros em listas, ou dívidas.
- **Administrador**: Em sua implementação há, além de dados do administrador, métodos específicos como adicionar ou remover moradores de determinada residência.
- **GUI**: Justamente a classe responsável pela interface, é nela que os ajustes de cada frame gráfico, sendo possível através dela, visualizar também tratamento de erros.

C. Modelagem do banco de dados

Para o desenvolvimento do sistema de gerenciamento, foi adotado um modelo de banco de dados relacional utilizando **MySQL**. A modelagem física foi elaborada com base nos requisitos funcionais e estruturais do sistema. As principais entidades e relacionamentos são:

*1) Entidade '**residencias**':* Armazena informações sobre as residências dos usuários do sistema, incluindo o administrador de cada residência.

*2) Entidade '**usuarios**':* Armazena dados dos usuários do sistema, incluindo nome, email e senha.

*3) Entidade '**produtos_residencia**':* Armazena informações sobre os produtos presentes em cada residência, incluindo quantidade existente, unidade de medida e preço médio.

*4) Entidade '**produtos_listados_pessoais**':* Armazena dados dos usuários do sistema, incluindo nome, email e senha.

*5) Entidade '**transferencias**':* Armazena dados de transferências entre devedores e credores.

A modelagem do banco de dados foi desenvolvida com o objetivo de garantir a integridade referencial dos dados, utilizando chaves estrangeiras para manter a consistência das informações entre as tabelas. Além disso, foram aplicadas restrições para garantir a exclusividade dos emails na tabela '**usuarios**'.

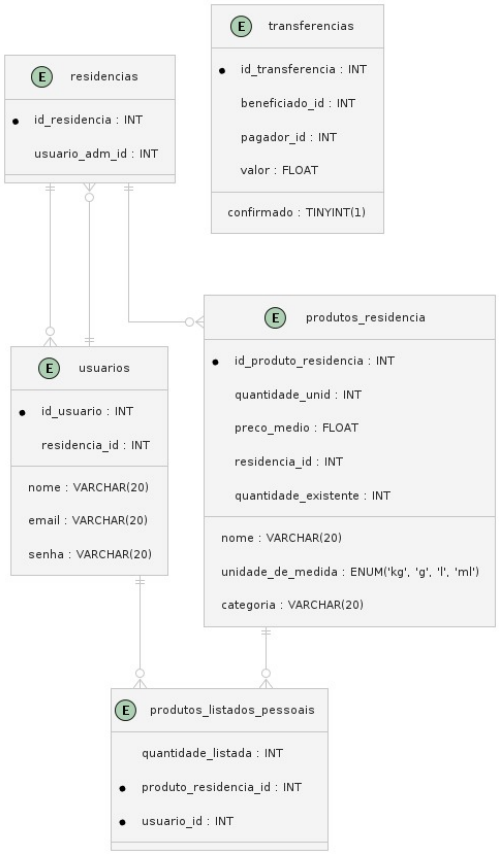

Figura 1. Diagrama do Banco de Dados

D. Integração

A integração entre os diferentes componentes do sistema foi um dos aspectos mais desafiadores do projeto. A comunicação entre o backend, responsável pelo gerenciamento de dados e lógica de negócios, e o frontend, que oferece a interface gráfica

ao usuário, foi facilitada pelo uso de Tkinter e a estruturação do código em módulos bem definidos.

A interação com o banco de dados foi garantida através do repository.py, que centraliza as operações de CRUD (Create, Read, Update, Delete) necessárias para a manutenção dos dados. A integração das funcionalidades permitiu criar uma aplicação coesa, onde ações realizadas na GUI são refletidas diretamente no banco de dados, assegurando que todos os dados estejam sincronizados e atualizados em tempo real.

IV. Resultados

Ao acessar [3] e seguir os passos descritos no 'README.md', ao executar o arquivo 'main.py', observa-se que a interface inicia imediatamente na tela de login, solicitando email e senha ou cadastro a ser feito, como ilustra:

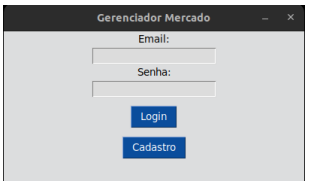

Figura 2. Tela de login

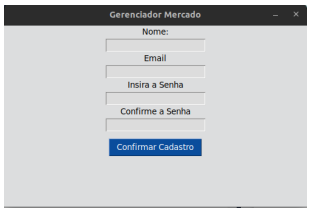

Figura 3. Cadastro

Em seguida, após efetuar possível cadastro e login com as novas credenciais, a nova tela principal se dispõe, com menu interativo para as diferentes necessidades.

Figura 4. Página Inicial

Primeiramente, na seção 'Configurações', ao entrar nela é possível acompanhar detalhes da própria residência, como ID, e criar uma nova gerando novo ID. Bem como, alterar senha, ou consultar ID e email individual do usuário.

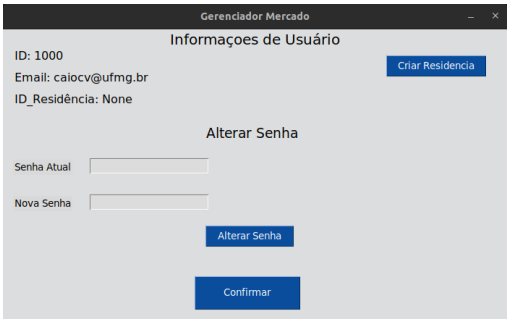

Figura 5. Configurações

Vale resaltar que, uma vez que é criada uma nova residência, e efetuado novamente o login, a tela de configurações é alterada, contendo agora opção de excluir residência e o novo ID dela.

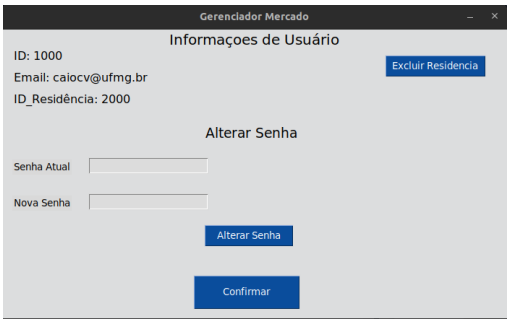

Figura 6. Configurações com ID residencia

Ademais, logo a direita da Figura 3, há a seção de lista pessoal, que na adição de itens à lista, determinando nome, categoria e quantidade, irá justamente resultar na lista de compras pessoal do usuário.

Figura 7. Lista de Compras

Em 'Realizar Compra' na Figura 3, agora é possível visualizar a lista compartilhada dos moradores, sendo possível efetuar uma compra, deixando os itens registrados na dispensa, bem como atualizar suas quantidades ou remover itens que não são mais necessários.

Figura 8. Realizar Compra

Figura 9. Dispensa

Na seção 'Adicionar Morador é possível que adicionar novos ou remover atuais com base no seu id gerado, apresentado na seção 'Configurações'.

Figura 10. Adicionar morador

Por fim, a seção 'Dívidas' permite o controle de débitos e créditos entre os moradores da república. O usuário pode registrar novas dívidas ou marcar dívidas como pagas, além de visualizar o saldo atual de cada morador.

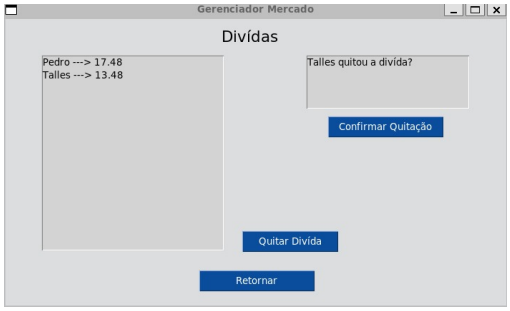

Figura 11. Verificar Dívidas

V. Conclusão

O desenvolvimento deste sistema de gerenciamento de compras para repúblicas estudantis demonstrou como a aplicação de conceitos de programação orientada a objetos e integração com banco de dados pode resolver desafios práticos enfrentados em ambientes compartilhados. A criação de uma ferramenta colaborativa que integra tanto interesses coletivos quanto individuais dos moradores mostra-se válida para melhorar a organização e eficiência no gerenciamento das finanças domésticas.

Ao longo do projeto, a escolha de Python como linguagem de programação e a utilização de SQLite para o banco de dados proporcionaram uma base robusta e flexível para o sistema. A interface gráfica desenvolvida com Tkinter possibilitou uma GUI mais intuitiva e de simplicidade durante o uso.

A arquitetura modular do código e a integração eficiente entre frontend e backend asseguram que as operações realizadas na interface gráfica sejam refletidas imediatamente no banco de dados, garantindo a sincronização e atualização dos dados em tempo real, bem como não foi perceptível erros aparentes, evidenciando o tratamento realizado desde o início do projeto. Os testes realizados confirmaram a eficácia do sistema em proporcionar uma solução prática, o que atende às necessidades do projeto.

O projeto está disponível em repositório anexado em [2], onde pode ser acessado e testado, ao seguir as orientações de criação do banco de dados e conexão do mesmo. Em suma, este trabalho apresentou uma solução prática para o gerenciamento de compras e finanças em repúblicas estudantis, destacando a importância da tecnologia na simplificação e melhoria da vida cotidiana dos estudantes.

Referências

[1] Riegel, S., Mellender, F., and Straw, A. (1988). Integration of database management with an object-oriented programming language. In: Dittrich, K.R. (eds) Advances in Object-Oriented Database Systems. OODBS 1988. Lecture Notes in Computer Science, vol 334. Springer, Berlin, Heidelberg.

[2] Sharma, A., et al. Database Management Systems—An Efficient, Effective, and Augmented Approach for Organizations. In: ICT with Intelligent Applications conference.

[3] Caracas, Caio (2024). StockCaro. Disponível em: https://github.com/caiocaracas/StockCaro/