📚Sistema de Registro de Leituras

1. Visão Geral do Projeto

O **Sistema de Registro de Leituras** é uma aplicação com interface gráfica desenvolvida em **Python**, utilizando a biblioteca **CustomTkinter (ctkinter)**, com o objetivo de permitir ao usuário registrar e gerenciar informações sobre livros já lidos.

O sistema não utilizará banco de dados. Todos os dados serão armazenados e manipulados por meio de **arquivos de texto**, garantindo simplicidade, portabilidade e fácil manutenção.

---

2. Objetivos

2.1 Objetivo Geral

Desenvolver uma aplicação gráfica intuitiva que permita ao usuário armazenar e consultar dados de livros lidos.

2.2 Objetivos Específicos

- Criar uma interface gráfica amigável e organizada
- Permitir o cadastro de livros
- Permitir a remoção de livros cadastrados
- Implementar funcionalidade de busca/pesquisa
- Armazenar dados de forma persistente em arquivos de texto
- Garantir validação básica de dados inseridos pelo usuário

---

3. Funcionalidades do Sistema

3.1 Cadastro de Livros

O sistema deverá permitir o cadastro de livros contendo os seguintes campos obrigatórios:

- **Título do livro**
- **Autor**
- **Quantidade de páginas**
- **Data de início da leitura**
- **Data de término da leitura**

3.2 Remoção de Livros

- Permitir excluir um livro previamente cadastrado
- A remoção deve refletir diretamente no arquivo de armazenamento

3.3 Pesquisa de Livros

- Buscar livros por título
- Buscar livros por autor
- Exibir os dados completos do livro encontrado

3.4 Visualização de Dados

- Listagem de todos os livros cadastrados
- Exibição clara e organizada das informações

---

4. Requisitos do Sistema

4.1 Requisitos Funcionais

- O sistema deve permitir adicionar livros
- O sistema deve permitir remover livros
- O sistema deve permitir pesquisar livros
- O sistema deve salvar os dados em arquivo de texto
- O sistema deve carregar os dados salvos ao iniciar

4.2 Requisitos Não Funcionais

- Interface gráfica desenvolvida exclusivamente com **CustomTkinter**
- Linguagem de programação: **Python 3.x**
- Não utilizar banco de dados
- Código organizado em módulos e classes
- Boa performance para pequenas e médias quantidades de registros

---

5. Tecnologias Utilizadas

- **Python 3.x**
- **CustomTkinter (ctkinter)**
- **Manipulação de arquivos (.txt)**
- **Paradigma de Programação Orientada a Objetos (POO)**

---

6. Estrutura de Armazenamento de Dados

6.1 Arquivo de Texto

Os dados dos livros serão armazenados em um arquivo de texto, por exemplo:

```
livros.txt
```

6.2 Formato dos Dados (Exemplo)

Cada livro pode ser armazenado em uma linha, usando um delimitador:

```
Título|Autor|Páginas|Data_Início|Data_Término
```

Exemplo:

```
Dom Casmurro|Machado de Assis|256|01/01/2024|15/01/2024
```

---

7. Estrutura do Projeto

Sugestão de organização dos arquivos:

```
projeto_leituras/
│
├── main.py              # Arquivo principal
├── interface.py         # Interface gráfica
├── livro.py             # Classe Livro
├── gerenciador.py       # Lógica de manipulação dos dados
├── livros.txt           # Arquivo de armazenamento
└── README.md            # Documentação do projeto
```

---

8. Boas Práticas de Desenvolvimento

8.1 Organização do Código

- Utilizar **classes** para representar entidades (ex: Livro)
- Separar lógica de negócio da interface gráfica
- Evitar código repetido

8.2 Programação Orientada a Objetos

- Criar uma classe `Livro` com atributos bem definidos
- Criar uma classe responsável pelo gerenciamento dos livros

8.3 Manipulação de Arquivos

- Sempre verificar se o arquivo existe antes de ler
- Utilizar tratamento de exceções (`try/except`)
- Garantir fechamento correto do arquivo (`with open()`)

8.4 Interface Gráfica

- Usar layouts organizados (frames)
- Evitar excesso de widgets na mesma tela
- Manter consistência de cores e fontes

8.5 Validação de Dados

- Verificar se campos obrigatórios foram preenchidos
- Garantir que páginas sejam números
- Validar formato de datas

---

9. Tratamento de Erros

O sistema deve tratar erros comuns, como:

- Arquivo inexistente
- Campos vazios
- Dados inválidos
- Erros de leitura/escrita em arquivo

Exibir mensagens amigáveis ao usuário utilizando caixas de diálogo.

---

10. Possíveis Melhorias Futuras

- Exportação dos dados para CSV
- Estatísticas de leitura (total de páginas lidas)
- Filtros avançados
- Ordenação por data ou autor
- Tema claro/escuro

---

11. Criação de Executável (Aplicativo Desktop)

11.1 Objetivo

Permitir que o projeto seja executado como um **aplicativo desktop**, sem a necessidade de o usuário instalar Python ou bibliotecas adicionais.

11.2 Ferramenta Utilizada

Será utilizada a ferramenta **PyInstaller**, que permite empacotar aplicações Python em um único arquivo executável.

11.3 Requisitos para Geração do Executável

- Python instalado no ambiente de desenvolvimento
- Biblioteca `customtkinter` instalada
- Projeto funcionando corretamente antes da conversão
- Sistema operacional compatível (Windows, Linux ou macOS)

11.4 Instalação do PyInstaller

A instalação deve ser feita via terminal ou prompt de comando:

```
pip install pyinstaller
```

11.5 Geração do Executável

No diretório raiz do projeto, executar o comando:

```
pyinstaller --onefile --windowed main.py
```

Parâmetros utilizados:

- `--onefile`: gera um único arquivo executável
- `--windowed`: impede a abertura do terminal (modo gráfico)
- `main.py`: arquivo principal da aplicação

11.6 Estrutura Gerada

Após a execução do comando, serão criadas as seguintes pastas:

- `dist/` → contém o executável final
- `build/` → arquivos temporários
- `main.spec` → arquivo de configuração do PyInstaller

O executável final estará localizado em:

```
dist/main.exe
```

11.7 Arquivos de Dados Externos

Como o sistema utiliza arquivos de texto para armazenar os livros, é importante:

- Garantir que o arquivo `livros.txt` esteja no mesmo diretório do executável

• Ou configurar o código para criar o arquivo automaticamente caso não exista

---

11.8 Boas Práticas para Executáveis

• Usar caminhos relativos para arquivos
• Evitar caminhos absolutos no código
• Tratar exceções ao acessar arquivos externos
• Testar o executável em outro computador

11.9 Limitações Conhecidas

• O executável funciona apenas no sistema operacional em que foi gerado
• O tamanho do arquivo pode ser elevado
• Antivírus podem sinalizar falsos positivos

---

12. Considerações Finais

Este documento serve como base para o desenvolvimento do projeto, auxiliando na organização, padronização e boas práticas. Ele pode ser expandido conforme novas funcionalidades forem adicionadas ao sistema.

✍️**Autor:** Tiago Vieira 🟫**Projeto em desenvolvimento**