Universidade Católica de Brasília
Ciência da Computação
Atividade Avaliativa - Novas Tecnologias

Sistema de Gestão de Biblioteca Digital

Artur Vivacqua, Felipe Porto, Adenilson Alves, Flávio de Araujo e João Gabriel Brito
Professor: William Malvezzi

Brasília
2025

ARTUR VIVACQUA, FELIPE PORTO, ADENILSON ALVES, FLÁVIO DE ARAUJO E JOÃO GABRIEL BRITO

SISTEMA DE GESTÃO DE BIBLIOTECA DIGITAL

Artigo apresentado ao curso de graduação em ciência da computação/engenharia de software da Universidade Católica de Brasília, como requisito parcial para a obtenção do título de bacharel em computação.

Orientador: William Malvezzi

Brasília
2024

RESUMO

O objetivo do grupo é desenvolver um sistema em Python que permita o gerenciamento de uma biblioteca digital simples, com cadastro de livros, usuários e empréstimos. O sistema usa estruturas de dados e funcionalidades baseadas nos notebooks da disciplina. O contexto do projeto é o seguinte:

"A Universidade está implementando uma Biblioteca Digital para auxiliar no empréstimo e gestão de livros em formato digital. Seu grupo foi contratado para criar a primeira versão do sistema, utilizando Python e boas práticas de programação. A aplicação será usada inicialmente para testes internos."

SUMÁRIO

1 INTRODUÇÃO

A evolução da tecnologia tem permitido a digitalização de diversos processos, incluindo a gestão de bibliotecas. Com o aumento do consumo de materiais digitais e a necessidade de organização eficiente de acervos, surgiu a necessidade de desenvolver um Sistema de Biblioteca Digital que permita o gerenciamento adequado de livros eletrônicos, proporcionando acesso facilitado a estudantes, pesquisadores e demais leitores.

O Sistema de Biblioteca Digital foi concebido para atender às demandas de uma biblioteca moderna, oferecendo um ambiente intuitivo para cadastro, consulta e empréstimo de livros. Além disso, o sistema busca otimizar o fluxo de trabalho dos administradores, permitindo a gestão eficiente do acervo e o controle de acessos de usuários.

Este relatório detalha o desenvolvimento do sistema, incluindo seus objetivos, arquitetura, tecnologias utilizadas e metodologias de implementação. Com essa documentação, espera-se fornecer uma visão abrangente das soluções adotadas e das melhorias que podem ser realizadas no futuro.

2 OBJETIVOS DO SISTEMA E ARQUITETURA DO SISTEMA

2.1 Objetivos do sistema

O Sistema de Biblioteca Digital foi desenvolvido com os seguintes objetivos:

- Permitir o cadastro e gerenciamento de livros digitais.
- Facilitar a busca e consulta de títulos disponíveis.
- Gerenciar usuários e seus acessos ao acervo.
- Implementar um sistema de empréstimo e devolução de livros.

2.2 Arquitetura do sistema

- A arquitetura é baseada em princípios de programação orientada a objetos, onde as entidades principais (livros, usuários, biblioteca) são representadas como classes.
- O código é organizado em módulos, com cada classe responsável por uma parte específica da lógica da aplicação, como gerenciamento de livros, usuários e interface gráfica.
- A aplicação utiliza arquivos JSON para armazenar dados, permitindo que informações sobre livros e usuários sejam salvas e carregadas entre as execuções.

- A interface gráfica construída com Tkinter permite que os usuários interajam com a aplicação de forma intuitiva, facilitando operações como cadastro, busca e empréstimos de livros.

Essa combinação de tecnologias e arquitetura proporciona uma base sólida para o desenvolvimento e a manutenção da aplicação, permitindo futuras expansões e melhorias.

3 TECNOLOGIAS UTILIZADAS

O desenvolvimento do sistema utilizou as seguintes tecnologias:

- Linguagem de programação: Python
- Interface gráfica: Tkinter (para a construção da interface do usuário)
- Persistência de dados: JSON (para armazenamento de informações sobre livros e usuários)

4 FUNCIONALIDADES E METODOLOGIAS DE IMPLEMENTAÇÃO

O sistema foi desenvolvido seguindo princípios de boas práticas em programação orientada a objetos. Os principais métodos utilizados incluem:

- Cadastro de livros: Implementado através de um formulário na interface gráfica que coleta dados do usuário e os armazena em um arquivo JSON.
- Pesquisa de livros: Utiliza filtragem em listas de dicionários para encontrar livros com base nos critérios inseridos pelo usuário.
- Gerenciamento de usuários:
    - Cadastro: Os usuários são cadastrados com validação para evitar e-mails duplicados.
- Empréstimo e devolução:
    - Registra livros emprestados e devolvidos, atualizando o status do livro e o histórico de empréstimos.
    - Notificações são exibidas na interface gráfica para informar o sucesso ou falha nas operações de empréstimo e devolução.

5 ALGORÍTMO

5.1 Importação de Bibliotecas

```
import json
from datetime import datetime
import tkinter as tk
from tkinter import messagebox
from tkinter import ttk
```

O código importa quatro bibliotecas padrão do Python:

- **json**
    - Usada para manipulação de dados no formato JSON (JavaScript Object Notation).
    - Possível uso: leitura e gravação de arquivos JSON para armazenar dados do sistema.
- **datetime**
    - Fornece funções para trabalhar com data e hora.
    - Possível uso: registro de datas em logs, marcação de tempo de eventos, controle de empréstimos e devoluções.
- **tkinter (tk)**
    - Biblioteca padrão do Python para criar interfaces gráficas (GUI).
    - Possível uso: desenvolvimento de janelas, botões, formulários e interações visuais.
- **tkinter.messagebox**
    - Módulo que fornece caixas de diálogo para exibição de mensagens ao usuário.
    - Possível uso: exibição de alertas, confirmações e mensagens de erro.

- **tkinter.ttk (ttk)**
    - Extensão do tkinter que fornece widgets com aparência mais moderna.
    - Possível uso: criação de botões, caixas de entrada, tabelas e outros elementos gráficos.

5.2 Classes

1. Classe Livro

```python
# Classe que representa um livro na biblioteca
class Livro:
    def __init__(self, titulo, autor, publicacao, isbn, categoria, id_exemplar):
        # Inicializa os atributos do livro
        self.titulo = titulo  # Título do livro
        self.autor = autor  # Autor do livro
        self.publicacao = publicacao  # Data de publicação do livro
        self.isbn = isbn  # ISBN do livro
        self.categoria = categoria  # Categoria do livro
        self.id_exemplar = id_exemplar  # ID único do exemplar do livro
        self.emprestado = False  # Indica se o livro está emprestado
        self.emprestimos_count = 0  # Contador de quantas vezes o livro foi emprestado
```

A classe Livro representa um exemplar de livro na biblioteca.

Atributos:

- **titulo**: (str) – O título do livro.
- **autor**: (str) – O nome do autor do livro.
- **publicacao**: (str ou datetime) – A data de publicação do livro.
- **isbn**: (str) – O código ISBN (International Standard Book Number), que identifica o livro.
- **categoria**: (str) – A categoria ou gênero do livro (ex: ficção, ciência, história).
- **id_exemplar**: (str ou int) – Um identificador único para cada exemplar do livro.
- **emprestado**: (bool) – Indica se o livro está emprestado (True) ou disponível (False).
- **emprestimos_count**: (int) – Conta quantas vezes o livro já foi emprestado.

Uso:

- Criar objetos representando livros.
- Controlar a disponibilidade e o histórico de empréstimos de cada exemplar.

2. Classe Usuario

```python
# Classe que representa um usuário da biblioteca
class Usuario:
    def __init__(self, nome, email, tipo):
        # Inicializa os atributos do usuário
        self.nome = nome  # Nome do usuário
        self.email = email  # Email do usuário
        self.tipo = tipo  # Tipo de usuário (ex: aluno, professor)
```

A classe Usuario representa um usuário da biblioteca, que pode ser um aluno, professor ou outro tipo de leitor.

Atributos:

- **nome**: (str) – O nome completo do usuário.
- **email**: (str) – O endereço de e-mail do usuário.
- **tipo**: (str) – O tipo de usuário (ex: aluno, professor), o que pode influenciar regras de empréstimo.

Uso:

- Criar perfis de usuários para registrar quem está emprestando os livros.
- Diferenciar usuários de acordo com seus privilégios na biblioteca (ex: alunos podem pegar menos livros que professores).

3. Classe Biblioteca

```python
# Classe que gerencia a biblioteca, incluindo livros, usuários e empréstimos
class Biblioteca:
    def __init__(self, livros_arquivo="livros.txt", usuarios_arquivo="usuarios.txt", emprestimos_arquivo="emprestimos.txt"):
        # Inicializa os arquivos que armazenam os dados da biblioteca
        self.livros_arquivo = livros_arquivo  # Caminho do arquivo de livros
        self.usuarios_arquivo = usuarios_arquivo  # Caminho do arquivo de usuários
        self.emprestimos_arquivo = emprestimos_arquivo  # Caminho do arquivo de empréstimos
```

A classe Biblioteca gerencia a coleção de livros, os usuários e os empréstimos.

Atributos:

- **livros_arquivo**: (str) – Caminho do arquivo onde os dados dos livros são armazenados (livros.txt).

- **usuarios_arquivo**: (str) – Caminho do arquivo onde os dados dos usuários são armazenados (usuarios.txt).

- **emprestimos_arquivo**: (str) – Caminho do arquivo onde os registros de empréstimos são armazenados (emprestimos.txt).

Uso:

- Carregar e salvar informações de livros, usuários e empréstimos.

- Gerenciar operações da biblioteca, como adicionar livros, registrar usuários e controlar empréstimos.

5.3 Métodos

1. carregar_dados(self, arquivo)

```python
# Método para carregar dados de um arquivo JSON
def carregar_dados(self, arquivo):
    try:
        with open(arquivo, "r") as f:
            return json.load(f)  # Retorna os dados carregados do arquivo
    except (FileNotFoundError, json.JSONDecodeError):
        return []  # Retorna uma lista vazia se o arquivo não for encontrado ou estiver vazio
```

Descrição:
Este método lê os dados de um arquivo JSON e retorna o conteúdo. Se o arquivo não existir ou estiver corrompido, retorna uma lista vazia.

Funcionamento:

- Tenta abrir o arquivo especificado (arquivo) no modo leitura ("r").

- Usa json.load(f) para carregar os dados e retorná-los.

- Se o arquivo não for encontrado (FileNotFoundError) ou houver erro na formatação JSON (json.JSONDecodeError), retorna uma lista vazia [].

2. salvar_dados(self, arquivo, dados)

```
# Método para salvar dados em um arquivo JSON
def salvar_dados(self, arquivo, dados):
    with open(arquivo, "w") as f:
        json.dump(dados, f, indent=4)  # Salva os dados no arquivo com formatação
```

Descrição:
Salva os dados fornecidos no arquivo JSON, garantindo formatação adequada.

Funcionamento:

- Abre o arquivo especificado (arquivo) no modo escrita ("w").
- Usa json.dump(dados, f, indent=4) para gravar os dados no arquivo com indentação para melhor legibilidade.

3. get_next_id(self)

```
# Método para obter o próximo ID disponível para um livro
def get_next_id(self):
    livros = self.carregar_dados(self.livros_arquivo)  # Carrega a lista de livros
    if livros:
        max_id = max(livro["id_exemplar"] for livro in livros)  # Encontra o maior ID existente
        return max_id + 1  # Retorna o próximo ID
    return 1  # Retorna 1 se não houver livros cadastrados
```

Descrição:
Retorna o próximo ID disponível para um novo livro.

Funcionamento:

- Carrega a lista de livros do arquivo JSON.
- Se houver livros cadastrados, busca o maior id_exemplar e retorna o próximo número (max_id + 1).
- Se não houver livros, retorna 1.

4. listar_todos_livros(self)

```
# Método para listar todos os livros cadastrados
def listar_todos_livros(self):
    return self.carregar_dados(self.livros_arquivo)  # Retorna a lista de livros
```

Descrição:
Retorna a lista de todos os livros cadastrados.

Funcionamento:

- Simplesmente carrega os dados do arquivo de livros (self.livros_arquivo).

5. cadastra_livro(self, livro)

```
# Método para cadastrar um novo livro
def cadastra_livro(self, livro):
    livros = self.carregar_dados(self.livros_arquivo)  # Carrega a lista de livros
    livros.append(livro.__dict__)  # Adiciona o novo livro à lista
    self.salvar_dados(self.livros_arquivo, livros)  # Salva a lista atualizada
    messagebox.showinfo("Sucesso", "Livro cadastrado com sucesso!")  # Exibe mensagem de sucesso
```

Descrição:
Adiciona um novo livro ao sistema.

Funcionamento:

- Carrega a lista de livros.
- Adiciona o novo livro (convertido para dicionário com livro.__dict__).
- Salva a lista atualizada no arquivo JSON.
- Exibe uma mensagem de sucesso.

6. cadastra_usuario(self, usuario)

```python
# Método para cadastrar um novo usuário
def cadastra_usuario(self, usuario):
    usuarios = self.carregar_dados(self.usuarios_arquivo)  # Carrega a lista de usuários
    # Verifica se o email já está cadastrado
    if any(u["email"] == usuario.email for u in usuarios):
        messagebox.showwarning("Erro", "Usuário já cadastrado com este e-mail.")  # Exibe aviso se o email já existir
        return
    usuarios.append(usuario.__dict__)  # Adiciona o novo usuário à lista
    self.salvar_dados(self.usuarios_arquivo, usuarios)  # Salva a lista atualizada
    messagebox.showinfo("Sucesso", "Usuário cadastrado com sucesso!")  # Exibe mensagem de sucesso
```

Descrição:
Adiciona um novo usuário ao sistema.

Funcionamento:

- Carrega a lista de usuários.
- Verifica se o e-mail já está cadastrado.
- Se o e-mail já existir, exibe um aviso e interrompe o cadastro.
- Se o e-mail for novo, adiciona o usuário (usuario.__dict__).
- Salva os dados atualizados e exibe uma mensagem de sucesso.

7. cadastra_emprestimo(self, id_exemplar, usuario_email)

```python
def cadastra_emprestimo(self, id_exemplar, usuario_email):
    livros = self.carregar_dados(self.livros_arquivo)  # Carrega a lista de livros
    usuarios = self.carregar_dados(self.usuarios_arquivo)  # Carrega a lista de usuários
    emprestimos = self.carregar_dados(self.emprestimos_arquivo)  # Carrega a lista de empréstimos

    # Busca o livro que será emprestado
    livro = next((l for l in livros if l["id_exemplar"] == id_exemplar and not l["emprestado"]), None)
    # Busca o usuário que está solicitando o empréstimo
    usuario = next((u for u in usuarios if u["email"] == usuario_email), None)

    # Verifica se o livro não foi encontrado ou já está emprestado
    if not livro:
        messagebox.showerror("Erro", "Livro não encontrado ou já emprestado!")  # Exibe mensagem de erro
        return
    # Verifica se o usuário não foi encontrado
    if not usuario:
        messagebox.showerror("Erro", "Usuário não encontrado!")  # Exibe mensagem de erro
        return

    livro["emprestado"] = True  # Marca o livro como emprestado
    livro["emprestimos_count"] = livro.get("emprestimos_count", 0) + 1  # Incrementa o contador de empréstimos
    self.salvar_dados(self.livros_arquivo, livros)  # Salva a lista de livros atualizada

    # Cria um novo registro de empréstimo
    novo_emprestimo = {
        "id_exemplar": id_exemplar,
        "usuario_email": usuario_email,
        "data_emprestimo": datetime.now().strftime("%Y-%m-%d %H:%M:%S")  # Data e hora do empréstimo
    }
    emprestimos.append(novo_emprestimo)  # Adiciona o novo empréstimo à lista
    self.salvar_dados(self.emprestimos_arquivo, emprestimos)  # Salva a lista de empréstimos atualizada
    messagebox.showinfo("Sucesso", "Empréstimo registrado com sucesso!")  # Exibe mensagem de sucesso
```

Descrição:
Registra um empréstimo de livro para um usuário.

Funcionamento:

- Carrega a lista de livros, usuários e empréstimos.
- Verifica se o livro está disponível (não emprestado).
- Verifica se o usuário existe no sistema.
- Se houver problemas (livro já emprestado ou usuário inexistente), exibe erro.
- Se estiver tudo certo, atualiza o livro como emprestado e incrementa o contador de empréstimos.
- Adiciona o empréstimo à lista e salva os dados atualizados.
- Exibe uma mensagem de sucesso.

8. lista_emprestimos(self)

```python
# Método para listar todos os empréstimos ativos
def lista_emprestimos(self):
    emprestimos = self.carregar_dados(self.emprestimos_arquivo)  # Carrega a lista de empréstimos
    if not emprestimos:
        messagebox.showinfo("Info", "Nenhum empréstimo ativo.")  # Exibe mensagem se não houver empréstimos
        return
    # Cria uma mensagem com os detalhes dos empréstimos
    msg = "\n".join([f"Livro ID: {emp['id_exemplar']}, Usuário: {emp['usuario_email']}, Data: {emp['data_emprestimo']}" for emp in emprestimos])
    messagebox.showinfo("Empréstimos Ativos", msg)  # Exibe a lista de empréstimos ativos
```

Descrição:
Lista todos os empréstimos ativos.

Funcionamento:

- Carrega a lista de empréstimos.
- Se não houver empréstimos, exibe um aviso.
- Caso contrário, formata a lista de empréstimos em uma mensagem e a exibe.

9. devolve_livro(self, id_exemplar)

```python
# Método para devolver um livro
def devolve_livro(self, id_exemplar):
    livros = self.carregar_dados(self.livros_arquivo)  # Carrega a lista de livros
    emprestimos = self.carregar_dados(self.emprestimos_arquivo)  # Carrega a lista de empréstimos

    # Busca o livro que será devolvido
    livro = next((l for l in livros if l["id_exemplar"] == id_exemplar), None)
    if livro:
        livro["emprestado"] = False  # Marca o livro como não emprestado

    self.salvar_dados(self.livros_arquivo, livros)  # Salva a lista de livros atualizada

    # Remove o registro do empréstimo da lista
    emprestimos = [emp for emp in emprestimos if emp["id_exemplar"] != id_exemplar]
    self.salvar_dados(self.emprestimos_arquivo, emprestimos)  # Salva a lista de empréstimos atualizada

    messagebox.showinfo("Sucesso", "Livro devolvido com sucesso!")  # Exibe mensagem de sucesso
```

Descrição:
Marca um livro como devolvido e remove seu registro de empréstimo.

Funcionamento:

- Carrega os livros e empréstimos.

- Busca o livro pelo id_exemplar e o marca como não emprestado.
- Atualiza a lista de empréstimos removendo aquele livro.
- Salva os dados e exibe uma mensagem de sucesso.

10. busca_livros(self, criterio, valor)

```python
# Método para buscar livros com base em um critério
def busca_livros(self, criterio, valor):
    livros = self.carregar_dados(self.livros_arquivo)  # Carrega a lista de livros
    # Filtra os livros que atendem ao critério de busca
    resultados = [livro for livro in livros if valor.lower() in livro[criterio].lower()]
    if resultados:
        # Cria uma mensagem com os resultados da busca
        msg = "\n".join([f"{livro['titulo']} - {livro['autor']} - {livro['categoria']}" for livro in resultados])
        messagebox.showinfo("Resultado da Busca", msg)  # Exibe os resultados da busca
    else:
        messagebox.showinfo("Resultado da Busca", "Nenhum livro encontrado.")  # Exibe mensagem se não houver resultados
```

Descrição:
Busca livros que correspondam a um critério específico (ex: título, autor, categoria).

Funcionamento:

- Carrega a lista de livros.
- Filtra os livros que contêm o valor na chave correspondente ao criterio.
- Se houver resultados, exibe os livros encontrados.
- Se não houver, exibe uma mensagem informando que nada foi encontrado.
-

11. livros_por_categoria(self)

```python
# Método para contar livros por categoria
def livros_por_categoria(self):
    livros = self.carregar_dados(self.livros_arquivo)  # Carrega a lista de livros
    categoria_count = {}  # Dicionário para contar livros por categoria
    for livro in livros:
        categoria = livro["categoria"]  # Obtém a categoria do livro
        categoria_count[categoria] = categoria_count.get(categoria, 0) + 1  # Incrementa o contador da categoria
    if categoria_count:
        # Cria uma mensagem com a contagem de livros por categoria
        msg = "\n".join([f"{cat}: {count}" for cat, count in categoria_count.items()])
        messagebox.showinfo("Livros por Categoria", msg)  # Exibe a contagem de livros por categoria
    else:
        messagebox.showinfo("Livros por Categoria", "Nenhum livro cadastrado.")  # Exibe mensagem se não houver livros
```

Descrição:
Conta quantos livros há em cada categoria.

Funcionamento:

- Carrega a lista de livros.
- Percorre os livros e conta quantos pertencem a cada categoria.
- Exibe a contagem para cada categoria.

12. emprestimos_por_usuario(self)

```python
# Método para contar empréstimos por tipo de usuário
def emprestimos_por_usuario(self):
    emprestimos = self.carregar_dados(self.emprestimos_arquivo)  # Carrega a lista de empréstimos
    usuarios = self.carregar_dados(self.usuarios_arquivo)  # Carrega a lista de usuários
    type_count = {}  # Dicionário para contar empréstimos por tipo de usuário
    for emp in emprestimos:
        email = emp["usuario_email"]  # Obtém o email do usuário do empréstimo
        user = next((u for u in usuarios if u["email"] == email), None)  # Busca o usuário correspondente
        if user:
            tipo = user["tipo"]  # Obtém o tipo do usuário
            type_count[tipo] = type_count.get(tipo, 0) + 1  # Incrementa o contador do tipo de usuário
    if type_count:
        # Cria uma mensagem com a contagem de empréstimos por tipo de usuário
        msg = "\n".join([f"{tipo}: {count}" for tipo, count in type_count.items()])
        messagebox.showinfo("Empréstimos por Tipo de Usuário", msg)  # Exibe a contagem de empréstimos por tipo de usuário
    else:
        messagebox.showinfo("Empréstimos por Tipo de Usuário", "Nenhum empréstimo registrado.")  # Exibe mensagem se não houver empréstimos
```

Descrição:
Conta quantos empréstimos foram feitos por tipo de usuário (ex: aluno, professor).

Funcionamento:

- Carrega a lista de empréstimos e usuários.
- Associa cada empréstimo ao tipo de usuário correspondente.
- Conta quantos empréstimos cada tipo de usuário fez.
- Exibe os resultados.

13. livros_mais_emprestados(self)

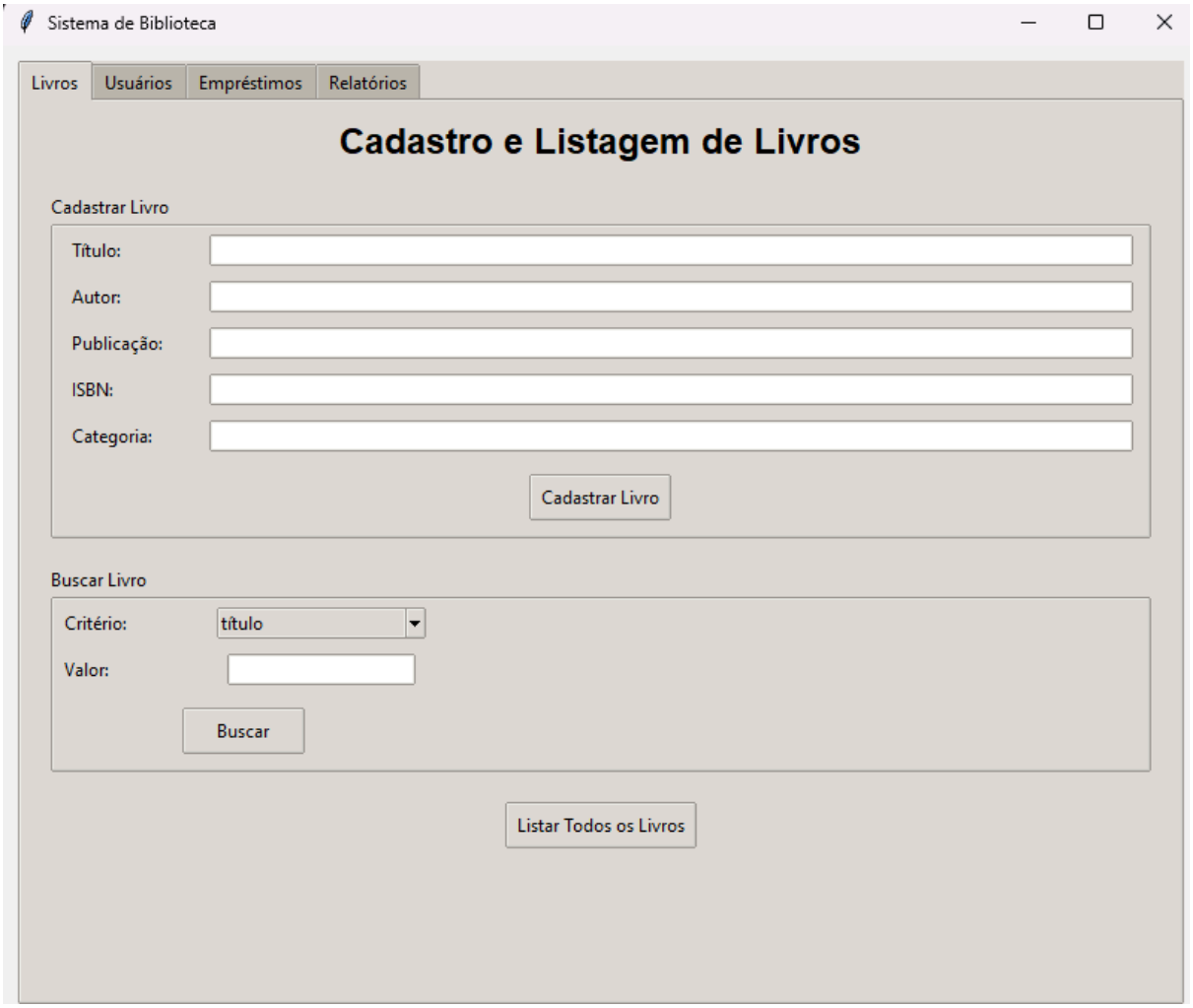
```python
# Método para listar os livros mais emprestados
def livros_mais_emprestados(self):
    livros = self.carregar_dados(self.livros_arquivo)  # Carrega a lista de livros
    if not livros:
        messagebox.showinfo("Livros mais Emprestados", "Nenhum livro cadastrado.")  # Exibe mensagem se não houver livros
        return

    # Garante que todos os livros tenham um contador de empréstimos
    for livro in livros:
        if "emprestimos_count" not in livro:
            livro["emprestimos_count"] = 0
    # Ordena os livros pelo número de empréstimos em ordem decrescente
    sorted_books = sorted(livros, key=lambda l: l["emprestimos_count"], reverse=True)
    top_books = sorted_books[:3]  # Seleciona os 3 livros mais emprestados
    # Cria uma mensagem com os livros mais emprestados
    msg = "\n".join([f"{livro['titulo']} - {livro['emprestimos_count']} empréstimos" for livro in top_books])
    messagebox.showinfo("Livros Mais Emprestados", msg)  # Exibe os livros mais emprestados
```

Descrição:
Lista os três livros mais emprestados.

Funcionamento:

- Carrega a lista de livros.
- Garante que cada livro tenha um contador de empréstimos (emprestimos_count).
- Ordena os livros pelo número de empréstimos (do maior para o menor).
- Seleciona os três mais emprestados e os exibe.

5.4 Interface Gráfica



A interface gráfica do sistema da biblioteca digital foi projetada para ser intuitiva, eficiente e acessível, garantindo uma experiência fluida para os usuários. O design segue princípios de usabilidade, com menus organizados e funcionalidades bem distribuídas para facilitar a navegação.

Principais Elementos da Interface

1. **Tela de Login e Cadastro**
    - Permite que os usuários acessem o sistema com credenciais seguras.
    - Opção para novos usuários realizarem o cadastro.
2. **Tela Inicial (Dashboard)**
    - Exibe um resumo das principais funcionalidades do sistema.

- Acesso rápido a empréstimos, reservas e consultas de livros.

3. **Seção de Consulta e Pesquisa de Livros**
    - Barra de pesquisa dinâmica com filtros avançados.
    - Exibição detalhada das informações dos livros, incluindo disponibilidade.
4. **Gerenciamento de Empréstimos e Devoluções**
    - Interface simplificada para registrar e visualizar empréstimos ativos.
    - Opção para renovar ou devolver livros de maneira rápida.
5. **Área do Administrador**
    - Controle sobre cadastros de usuários e livros.
    - Gerenciamento de permissões e monitoramento de atividades no sistema.

7 CONCLUSÃO

O desenvolvimento do sistema de biblioteca digital trouxe uma solução eficiente e moderna para a gestão de acervos bibliográficos, proporcionando maior acessibilidade, controle e organização. A implementação de tecnologias como bancos de dados otimizados, interfaces intuitivas e automação de processos contribui significativamente para a redução do tempo de execução de tarefas, melhorando a experiência tanto para os administradores quanto para os usuários finais.

A digitalização do catálogo e a possibilidade de busca avançada permitem que os usuários encontrem livros de forma rápida e precisa, eliminando a necessidade de consultas manuais demoradas. Além disso, a automação de processos como cadastro, empréstimo e devolução reduz significativamente a carga de trabalho dos funcionários da biblioteca, minimizando erros e aumentando a eficiência operacional.

Outro aspecto relevante é a implementação de notificações automatizadas e sistemas de controle de prazos, que ajudam a evitar atrasos na devolução dos materiais e facilitam a gestão de usuários. O uso de tecnologia também permite um monitoramento mais detalhado do uso dos recursos, fornecendo relatórios que auxiliam na tomada de decisões para melhorar continuamente o serviço.

Em suma, a adoção de um sistema digital não só moderniza a administração da biblioteca, mas também otimiza a experiência dos usuários, garantindo maior agilidade, segurança e praticidade nos processos. O impacto positivo dessas tecnologias reafirma a importância da inovação na gestão de acervos, tornando o acesso ao conhecimento mais eficiente e democrático.