

Sistema Acadêmico



Danilo Azevedo | Adriano Dias | Eduardo Juan | Rafael Botti | João Acerbi



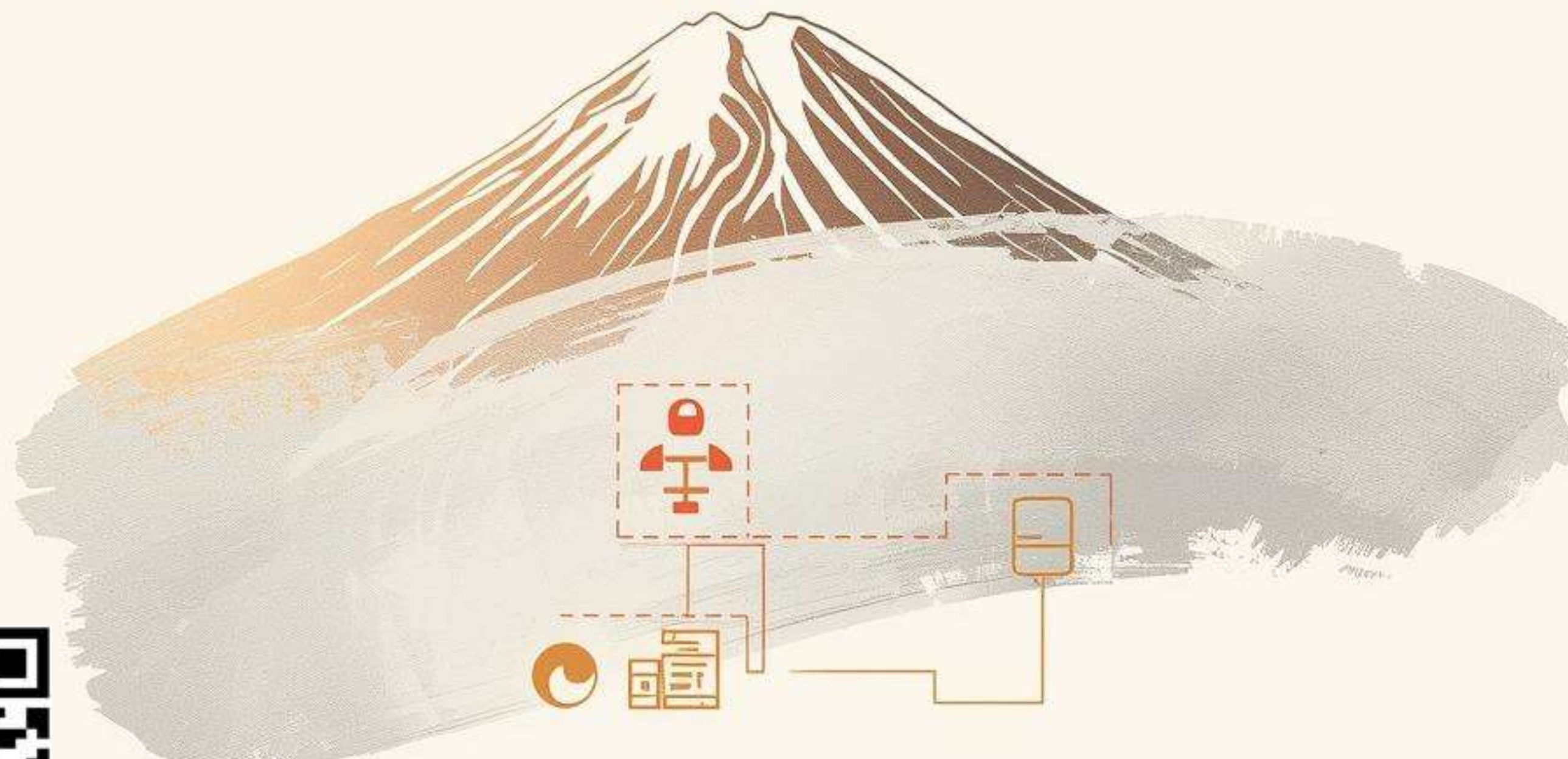
Link para o código



Link para o Github:

https://github.com/ejuanoli/UNIP_ADS.git

* Observação: o Manual do Usuário está nos arquivos do github, assim como os outros arquivos solicitados.



UNIP ADS



TEMA

Uma instituição de ensino necessita de um sistema colaborativo para apoiar professores e alunos no gerenciamento de turmas, aulas e atividades. Atualmente, controles são realizados de forma descentralizada (planilhas, e-mails, mensagens em aplicativos). O sistema deve permitir cadastro de turmas e alunos, registro de aulas e diário eletrônico, upload e consulta de atividades, e módulos distribuídos em uma rede. Um dos objetivos é a eliminação do uso de papel pelos professores como medidas sustentáveis.



SUMÁRIO

- **Biblioteca utilizadas**
- **Arquitetura do Sistema**
- **Detalhamento das funções**
- **Conexão entre Servidor e Cliente**
- **Banco de dados**
- **Design do Programa**
- **Diferenciais**
- **Créditos**

BIBLIOTECAS UTILIZADAS NO SISTEMA

Gerenciamento de Dados e Sistema

- **os:** Gerencia arquivos e pastas do sistema operacional.
- **ctypes:** Permite que o Python use o banco de dados feito em linguagem C.
- **json:** Salva e lê configurações e dados de usuários de forma organizada.
- **struct:** Grava e lê dados binários, como notas e presenças.
- **logging:** Cria registros (logs) para depurar e monitorar o sistema.

Interface e Comunicação

- **tkinter:** Constrói toda a interface gráfica do sistema (janelas, botões, menus).
- **socket:** Permite a comunicação de rede entre a aplicação e o servidor.
- **threading:** Executa tarefas simultâneas para que a interface não trave e o servidor atenda vários clientes.
- **multiprocessing:** Executa o servidor em um processo separado da interface gráfica.

Execução e Utilitários

- **time:** Fornece funções de tempo, como a criação de timestamps.
- **calendar:** Oferece funções de calendário, como a configuração da semana.
- **re:** Valida formatos de texto, como e-mails e senhas (através de expressões regulares).
- **datetime:** Manipula datas e horas para registrar eventos como o login.

```
import tkinter as tk
from tkinter import ttk, filedialog, messagebox
import tkinter.font as tkFont
import socket
import os
import ctypes
import threading
import multiprocessing
import time
import struct
import calendar
import re
import json
from datetime import datetime
import logging
from logging.handlers import RotatingFileHandler
```

ARQUITETURA DO SISTEMA

Frontend (Interface Gráfica):

- Construída em Python com a biblioteca tkinter.
- Dividida em duas partes principais: uma tela de Login e a Aplicação Principal (dashboard).
- Possui um design flexível com suporte a temas claro e escuro.

Backend (Servidor):

- Executa em um processo separado com multiprocessing para não travar a interface do usuário.
- Usa threading para conseguir atender múltiplos usuários ao mesmo tempo sem lentidão.
- Recebe os comandos, processa a lógica e gerencia o acesso aos dados.

Comunicação Cliente-Servidor:

- A interface (cliente) envia e recebe dados do backend (servidor) através de socket.
- Possui um modo offline: caso o servidor não responda, o sistema utiliza dados salvos localmente para continuar funcionando.

Camada de Dados (Como as informações são guardadas):

- Banco de Dados Principal (em C): O núcleo com turmas e alunos é gerenciado pela biblioteca em C (.dll ou .so), acessada pelo Python com ctypes.
- Dados de Usuários (users.json): Informações de login, senhas, perfis e permissões são salvas em um arquivo JSON.
- Dados Binários (.dat): Notas e presenças são armazenadas em arquivos binários para maior performance e integridade, lidos com struct.
- Arquivos de Upload: Atividades e outros documentos são salvos diretamente em pastas no sistema de arquivos.

FUNCIONALIDADES

Autenticação e Perfis:

- **Login Seguro:** Sistema de autenticação com cadastro, recuperação de senha e perfis de usuário (Aluno, Professor, Administrador).
- **Aprovação de Cadastros:** Novos usuários ficam pendentes até que um administrador aprove o acesso, garantindo controle.

Recursos e Ferramentas:

- **Gerenciador de Arquivos:** Professores podem fazer upload de materiais e atividades, que podem ser baixados pelos usuários da turma.
- **Anotações Pessoais:** Um bloco de notas integrado para que cada usuário possa criar, salvar e gerenciar suas próprias anotações.
- **Interface Personalizável:** Suporte a temas claro e escuro para maior conforto visual do usuário.
- **Relatórios e Diagnósticos:** Ferramentas para administradores visualizarem o estado do sistema e exportarem dados.

Gerenciamento Acadêmico:

- **Gestão de Turmas e Alunos:** Permite criar, editar, listar e excluir turmas, além de gerenciar os alunos dentro delas.
- **Lançamento de Notas:** Interface para professores lançarem as notas (NP1, NP2, PIM) e calcularem a média automaticamente, assim como um status de aprovado/reprovado com lógica para a nota de exame.
- **Controle de Presença:** Ferramenta para registrar a presença dos alunos em uma data específica, utilizando um calendário visual.
- **Calendário de Provas:** Permite agendar e consultar as datas das avaliações de cada turma.

CONEXÃO ENTRE SERVIDOR E CLIENTE

Interface do Usuário (Cliente):

- A Interface Gráfica (cliente) se conecta ao servidor usando o endereço de IP e a porta definidos no código (ex: 10.101.108.58, porta 65432).
- Cada ação do usuário (como "Listar Turmas" ou "Salvar Notas") é enviada como um comando de texto via socket.
- A interface aguarda uma resposta do servidor para exibir os dados na tela ou confirmar uma ação.

Plano de Contingência (Fallback):

- O sistema primeiro tenta se conectar ao servidor na rede. Se a conexão falhar, ele ativa um modo de fallback.
- Nesse modo, a interface para de tentar se conectar e passa a ler e salvar os dados em arquivos locais (turmas_local.json, alunos_local.json), permitindo o uso offline.



Servidor (Backend):

- O Servidor fica constantemente "escutando" na rede local, esperando por conexões de qualquer interface.
- Ao receber uma conexão, o servidor cria uma thread dedicada para atender aquele usuário, permitindo que vários se conectem ao mesmo tempo.
- Ele interpreta o comando recebido (ex: "ADD_TURMA..."), executa a ação correspondente e acessa os bancos de dados para buscar ou salvar as informações.



Acesso aos Bancos de Dados:

- Banco de Dados C (.dll/.so): Apenas o servidor tem acesso direto a este banco. Ele usa ctypes para chamar as funções que gerenciam turmas e alunos.
- Arquivos JSON e DAT: O servidor também é o responsável por ler e escrever nos arquivos de usuários (users.json), notas (notas.dat), presenças e outros, garantindo que os dados fiquem centralizados.
- A interface do usuário nunca acessa os bancos de dados diretamente; ela apenas solicita e recebe os dados prontos do servidor.

BANCO DE DADOS

Banco de Dados em C:

- É o núcleo que gerencia os dados principais (turmas e alunos), salvando-os diretamente em arquivos binários.
- Usa structs (estruturas) para definir como os dados são organizados na memória e nos arquivos.
- Contém toda a lógica para criar, ler, atualizar e deletar os registros acadêmicos de forma eficiente.

O Arquivo DLL(.dll):

- É o arquivo final, o resultado do código C compilado, que o Python consegue executar.
- A interface em Python não lê o código C; ela chama as funções que estão prontas dentro deste arquivo.
- Funciona como o "motor" do banco de dados, enquanto o Python (com ctypes) atua como o "painel de controle" que dá os comandos.

O Arquivo H (.h):

- É o "manual de instruções" do arquivo DLL.
- Ele descreve as structs e as funções que existem no código C, mas não contém a lógica delas.
- O Python usa essa "descrição" para saber exatamente como se comunicar com o DLL, entendendo quais dados enviar e o que esperar de volta.

```
# 1. Define o caminho do arquivo .dll (Windows) ou .so (Linux)
lib_path = "./libdatabase.so" if os.name != 'nt' else "./database.dll"

# 2. Carrega a biblioteca C na memória (esta é a "conexão")
lib = ctypes.CDLL(lib_path)

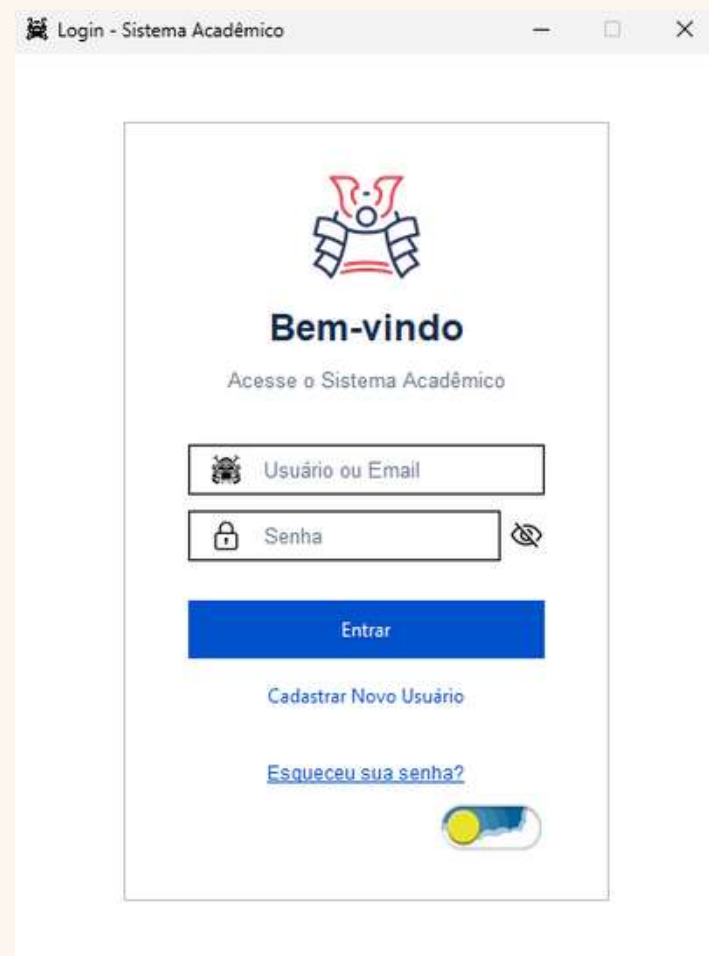
# 3. "Ensina" o Python a usar as funções do C (define os tipos de argumentos e retorno)
lib.salvar_turma.argtypes = [ctypes.POINTER(Turma)]
lib.salvar_turma.restype = None

lib.salvar_aluno.argtypes = [ctypes.POINTER(Aluno)]
lib.salvar_aluno.restype = None

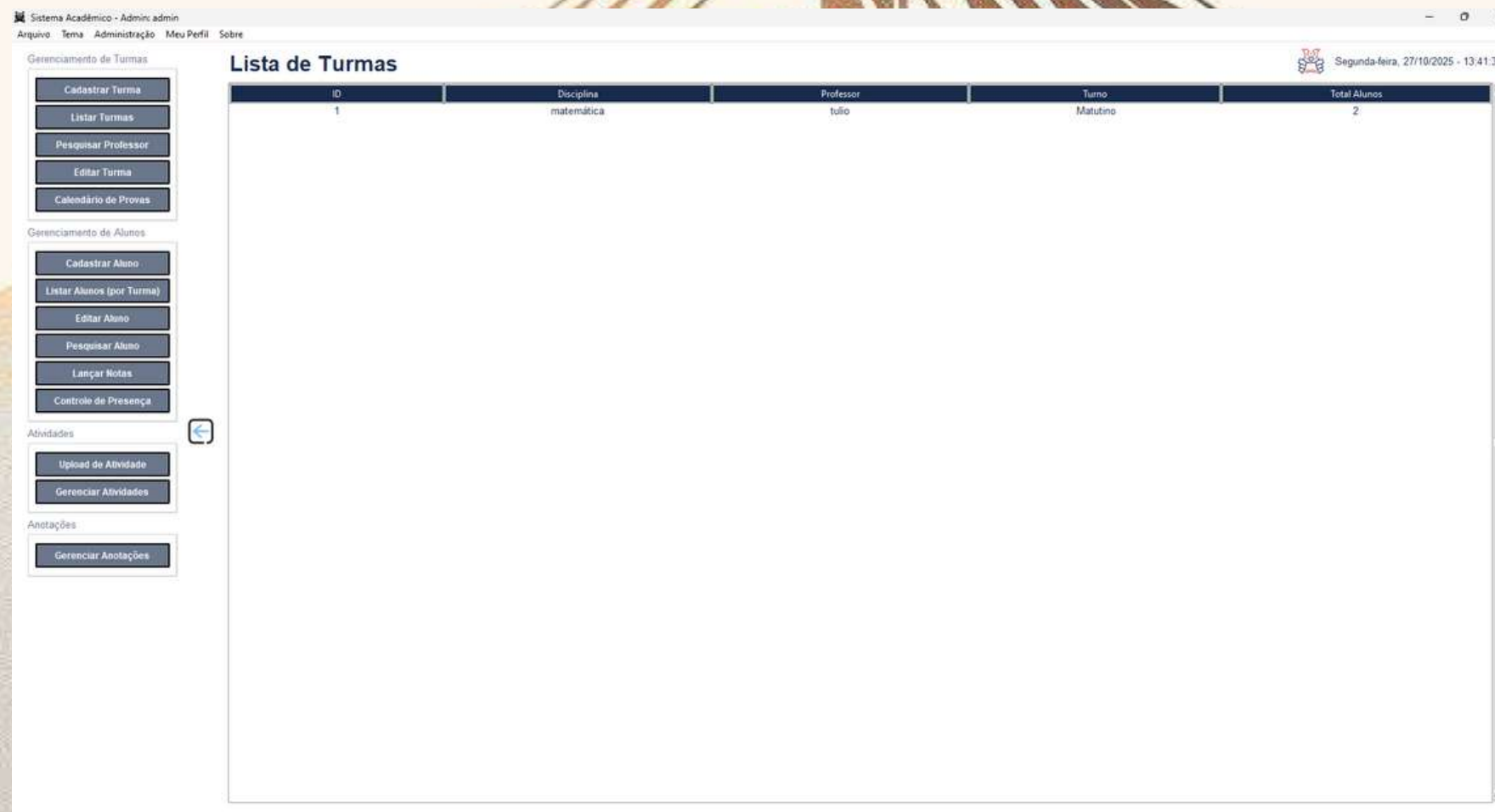
lib.listar_turmas.argtypes = [ctypes.POINTER(Turma), ctypes.c_int]
lib.listar_turmas.restype = ctypes.c_int
```

DESIGN DO PROGRAMA

Interface do menu de login



Interface principal, com as funcionalidades



Ícones dinâmicos e amigáveis, com estilização moderna.



Design baseado no tema do projeto do primeiro semestre,



Pim I:



DIFERENCIAIS DO PROGRAMA

Arquitetura Híbrida e Resiliente:

- Combina um backend de alta performance em C com uma interface flexível em Python.
- Possui um modo offline que ativa automaticamente se o servidor falhar, garantindo o funcionamento contínuo.
- O servidor é paralelo e multithreaded, permitindo múltiplos acessos simultâneos sem travar a interface.

Interface Moderna e Flexível:

- Suporte a temas claro e escuro com componentes visuais que se adaptam.
- Barra lateral (sidebar) retrátil para maximizar o espaço de visualização de dados.
- Componentes de interface customizados (janelas, botões) que se integram perfeitamente aos temas.

Segurança e Controle Administrativo:

- Fluxo de aprovação de cadastros: novos usuários precisam da liberação de um administrador para acessar o sistema.
- Senhas criptografadas (hash) para proteger as contas de usuário.

Diagrama de Atividades

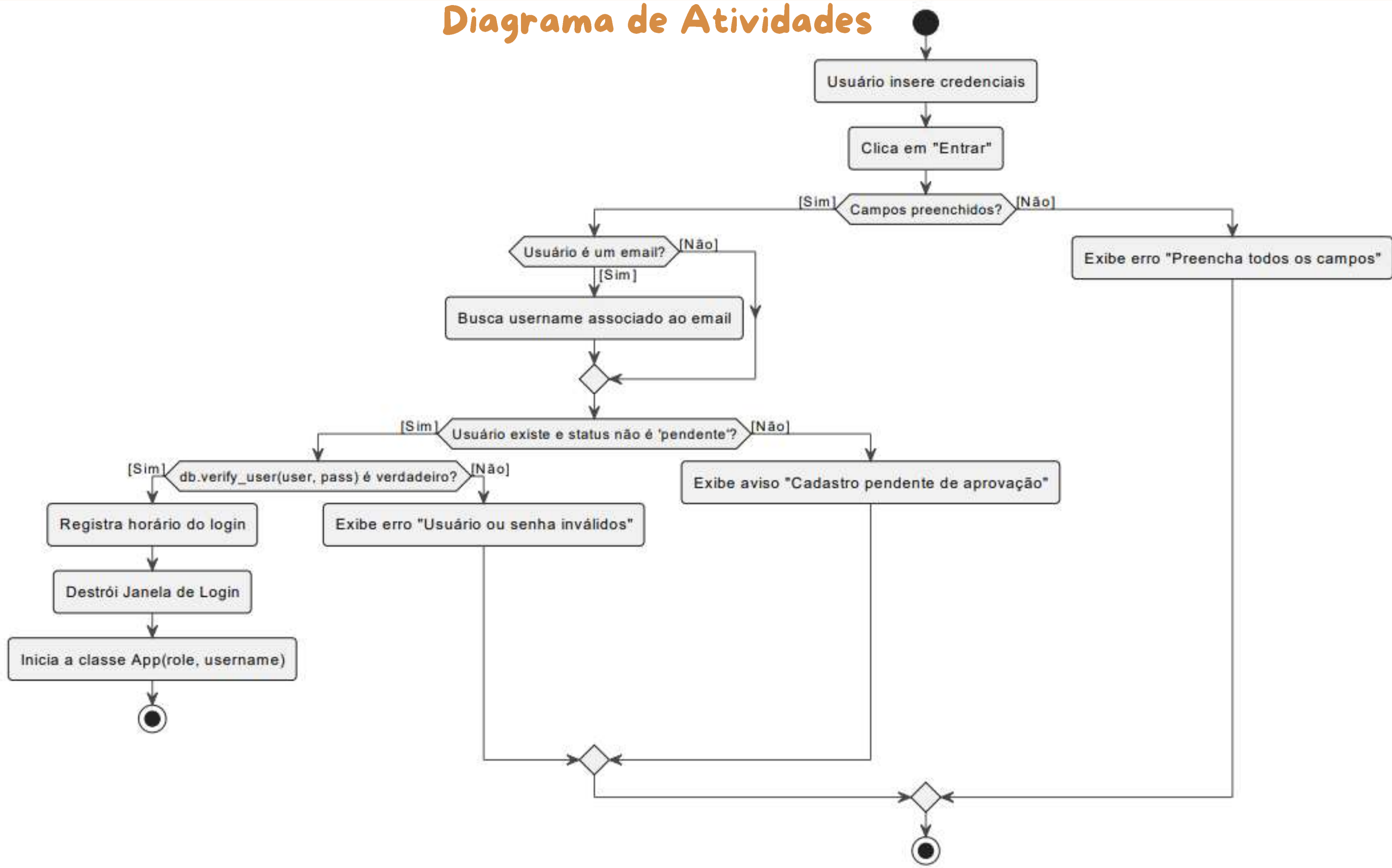


Diagrama de Classes

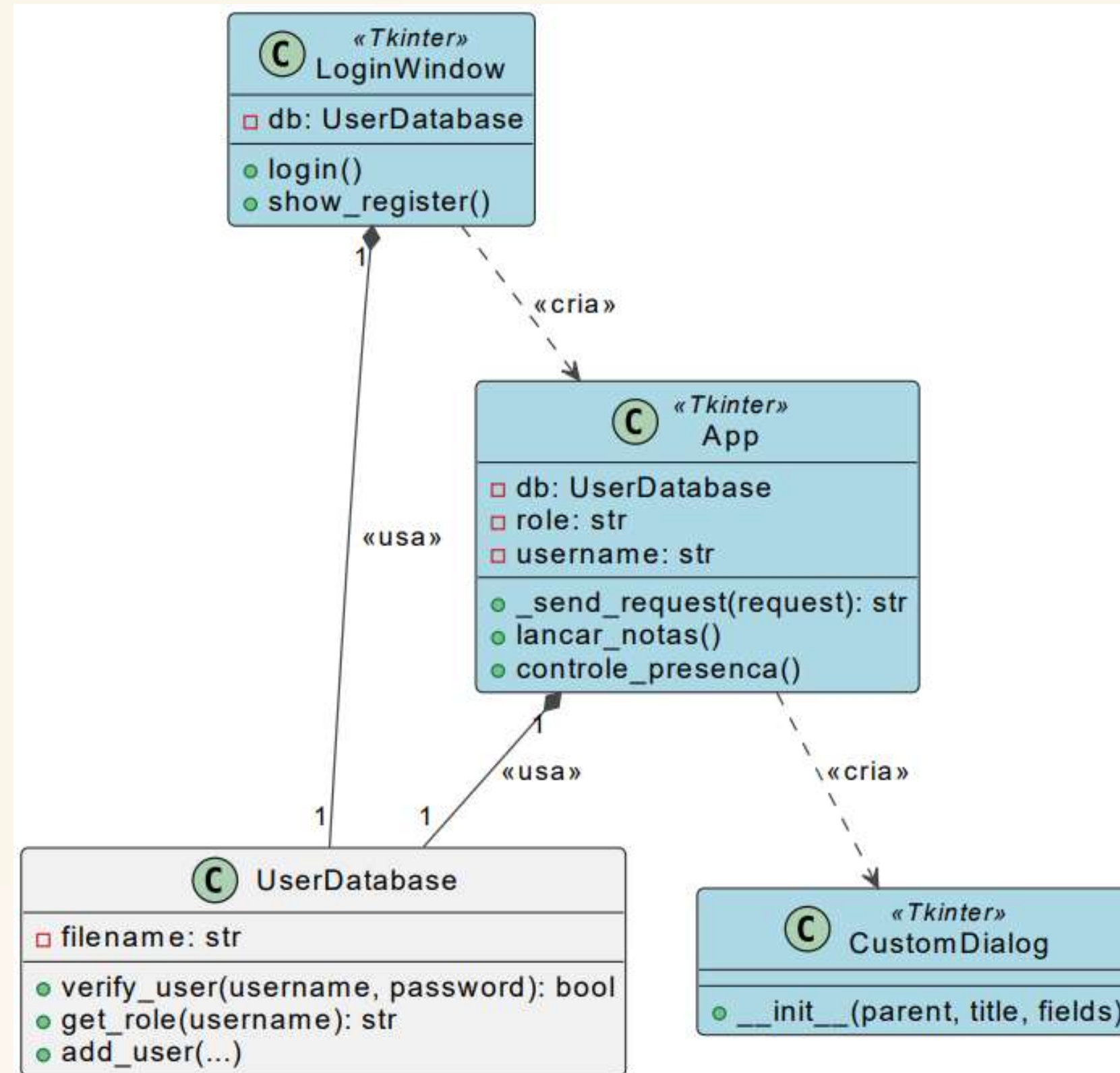


Diagrama de Sequencias

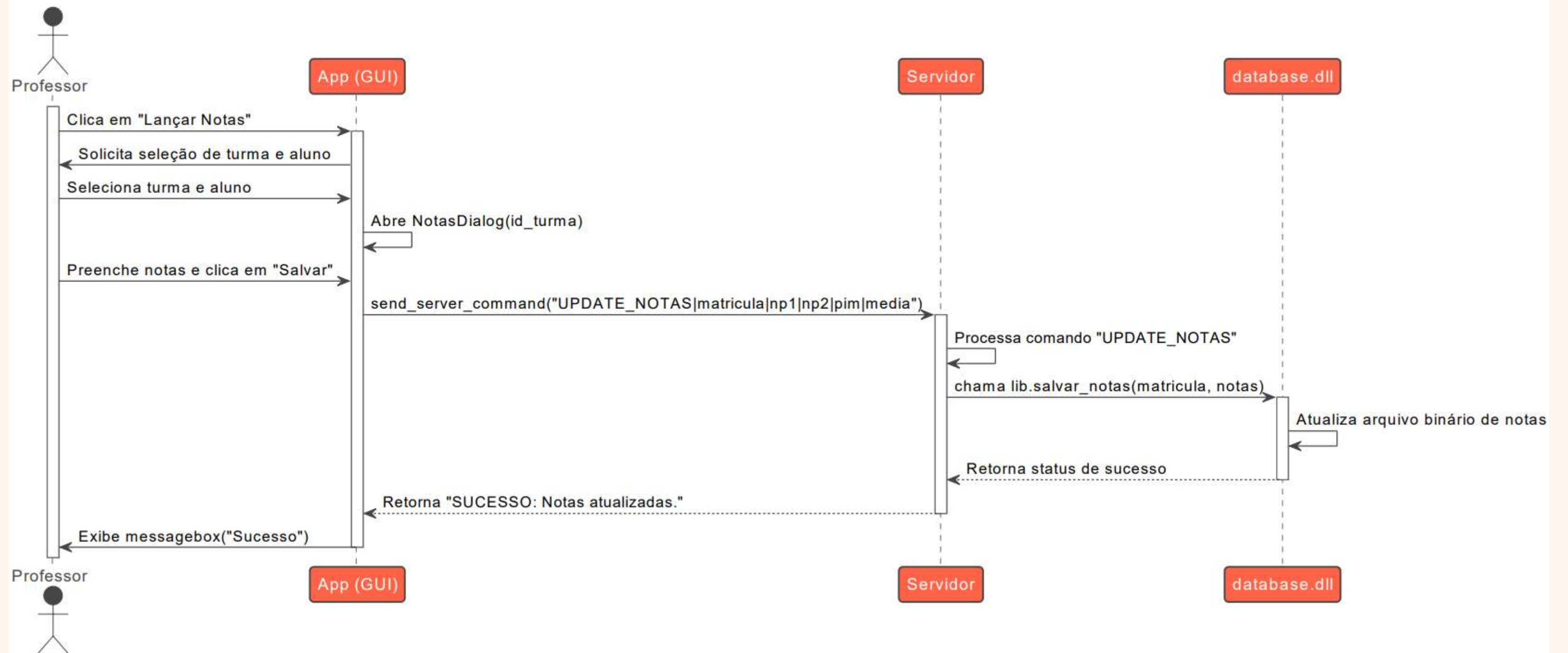


Diagrama de Casos de Uso

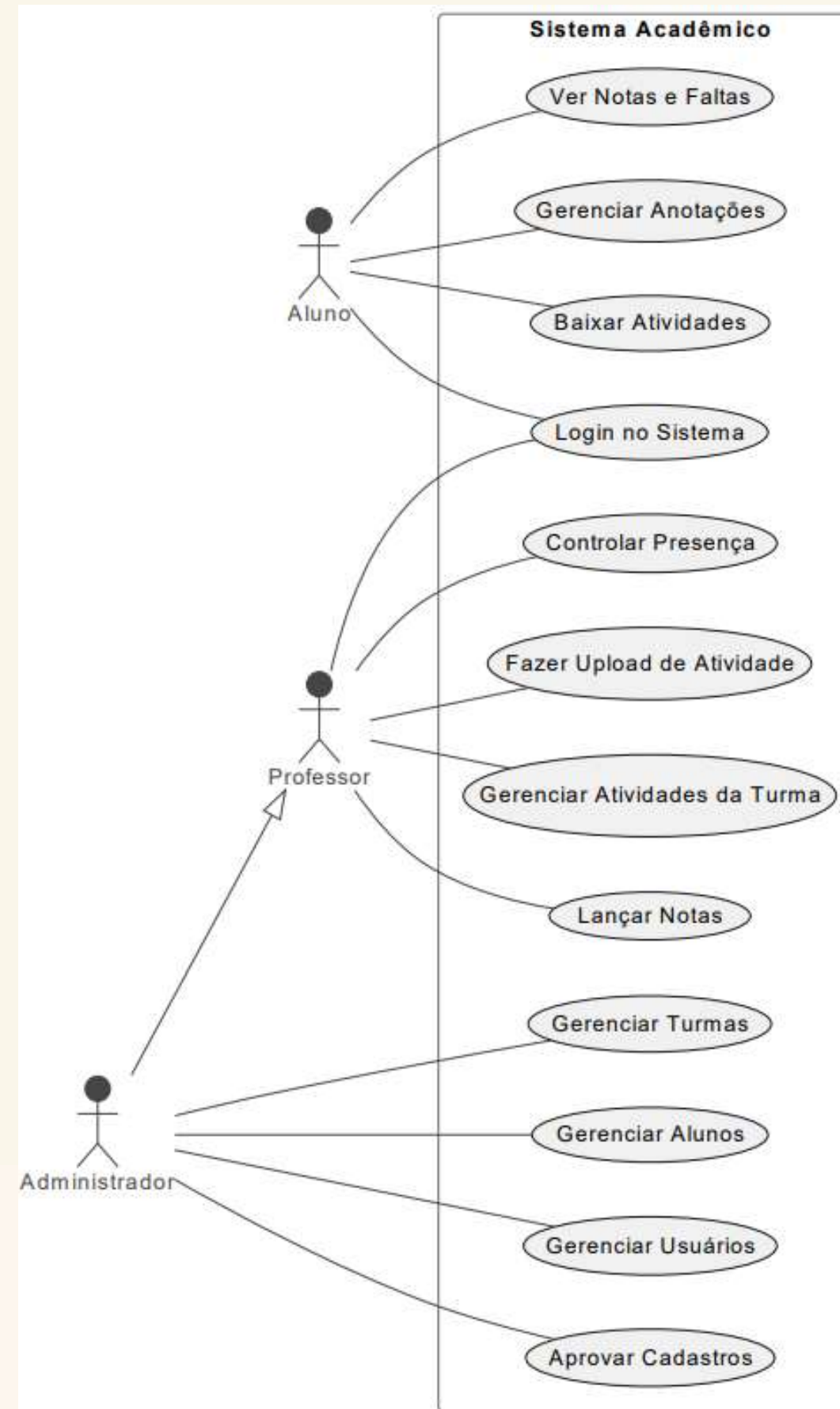
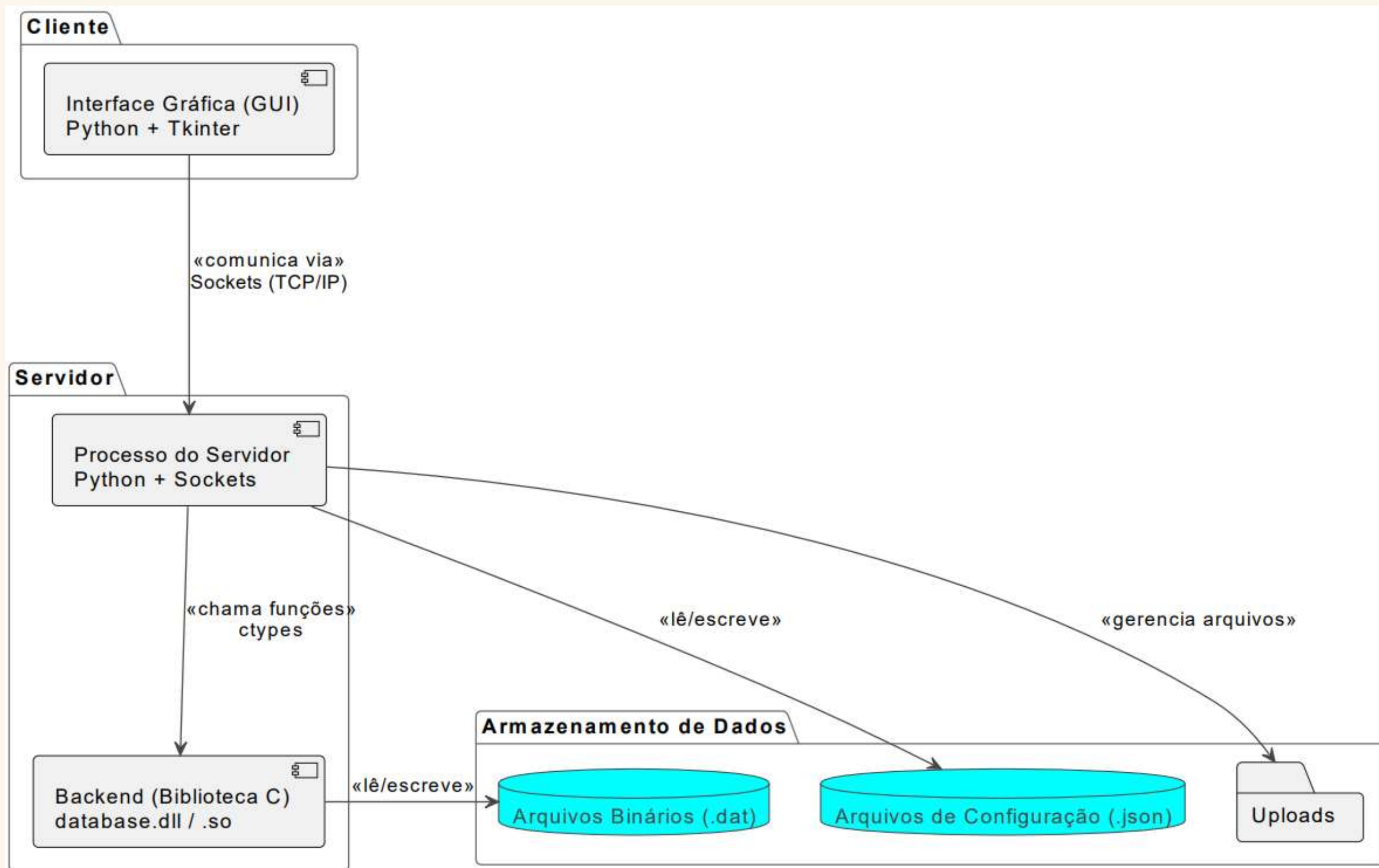


Diagrama de Arquitetura



Utilização do Claude Sonnet

File

Edit

Selection

View

Go

Run

Terminal

Help

cliente_gui.py

SISTEMA_ACADEMICO_CLI

> __pycache__

> .vscode

> img

> uploads

cliente_gui.py

database.c

database.dll

database.h

users.json

10344

10345

10346

10347

10348

10349

10350

10351

10352

10353

10354

10355

10356

10357

10358

10359

10360

10361

10362

10363

10364

10365

10366

10367

10368

10369

10370

if not anotacao_encontrada:

messagebox.showerror("Erro", "Anotação não encontrada!")

else:

messagebox.showerror("Erro", "Arquivo de anotações não encontrado!")

except Exception as e:

messagebox.showerror("Erro", f"Erro ao exportar anotação: {e}")

Botões em uma linha compacta para evitar cortes

ttk.Button(buttons_frame, text="Salvar", command=salvar_anotacao, style='Dialog.TButton').pack(side=tk.LEFT, padx=(

ttk.Button(buttons_frame, text="Excluir", command=excluir_anotacao, style='Danger.TButton').pack(side=tk.LEFT, padx=(

ttk.Button(buttons_frame, text="Exportar", command=exportar_anotacao, style='Secondary.TButton').pack(side=tk.LEFT,

ttk.Button(buttons_frame, text="Atualizar", command=carregar_anotacoes, style='Secondary.TButton').pack(side=tk.LEFT,

Bind duplo clique para editar anotação

notes_tree.bind("<Double-1>", lambda e: editar_anotacao_selecionada())

Carregar anotações iniciais

carregar_anotacoes()

if __name__ == "__main__":

if os.name != 'posix': multiprocessing.freeze_support(); multiprocessing.set_start_method('spawn', True)

print("[MAIN] Iniciando servidor..."); server_p = multiprocessing.Process(target=run_server, daemon=True); server_p.start()

if not server_p.is_alive(): print("[MAIN-ERRO] Falha ao iniciar o servidor.")

else: print("[MAIN] Iniciando GUI..."); login=LoginWindow(); login.mainloop()

if server_p.is_alive(): print("[MAIN] Encerrando servidor..."); server_p.terminate(); server_p.join()

adapte o código para trabalhar apenas

adapte o código para trabalhar apenas com o servidor, não utilize ou armazene dados locais.

To-dos 6

Adicionar gerenciamento de usuários no servidor (UserDatabase integrado)

Adicionar comandos de servidor para operações de usuários (login, criação, atualização)

Adicionar gerenciamento de provas, turnos e exames no servidor

Adicionar gerenciamento de anotações no servidor

Remover todas as funções de carregamento/salvamento local do cliente

Adaptar o cliente para fazer todas as requisições via socket

Read cliente_gui.py L1-100

Thought for 1s

Read cliente_gui.py L260-349

Read cliente_gui.py L300-374

Grepped load_provas|save_provas|load_turnos|save_t...

Thought for 1s

Grepped get_nota_exame|set_nota_exame|get_turno_...

Read cliente_gui.py L9670-9769

Thought for 1s

@ 1 Tab + Browser 34.8%

Plan, @ for context, / for commands

Ctrl+I Auto

Utilização do Claude Sonnet

The image shows a screenshot of a Visual Studio Code editor window. The main editor area displays a Python file named `cliente_gui.py`. The code is written in a dark theme and includes several sections: `# Role`, `# Turno`, `# Matrícula (se aluno)`, `# Campos editáveis`, and `# Email`. The code uses Tkinter for GUI elements like labels and entry fields, and includes logic for displaying roles, selecting turns, and handling student registration.

On the right side of the editor, there is a chat window titled "CHAT". It contains a message from Claude Sonnet 3.5 asking for a dark mode implementation. The message reads: "agora eu apenas preciso que o modo dark seja implementado nas janelas pop up que aparecem no aplicativo, e que o modo escuro ou claro seja lembrado no Tema ao lado do perfil, caso inicie o login no modo escuro, ele já deve aparecer marcado, senão desmarcado". Below the message, there is a search bar and a list of results for the search query "def _create_dialog_window". The results show that the function is defined in `cliente_gui.py` at lines 1168 to 1308.

The chat window also includes a section for "Add context (#), extensions (@), commands (/)" and a dropdown menu for the agent, currently set to "Claude Sonnet 3.5".

Utilização Gemini Pro

Google AI Studio

Home

Chat

Diagrama UML Do Código

GitHub README with Code Lines

Swap Largest and Smallest Ele...

Excel Update Guide For TASL

Observações no Power BI: Tool...

View all history →

Build

Dashboard

Documentation

Google AI models may make mistakes, so double-check outputs.

Get API key

Settings

lucky36616@gmail.c...

Samurai and Datetime Widget Placement270.590 tokens

Agora, o ícone da seta no seu `ToggleButton` ficará perfeitamente centrado na vertical, não importa o quão alto ou baixo a janela do seu aplicativo seja.

User

OK, agora arrume o calendario: no modo escuro, ele está ficando preto. `import tkinter as tk`
`from tkinter import ttk, filedialog, messagebox`
`import tkinter.font as tkFont`
`import socket`
`import os`
`import ctypes`
`import threading`
`import multiprocessing`
`import time`
`import struct`
`import calendar`
`import re`
`import json`
`from datetime import datetime`

`# Presence persistence lock`
`presenca_lock = threading.Lock()`

`LIGHT_THEME = {`
`# Cor de Fundo e Estrutura`

Start typing a prompt

Run settings

Gemini 2.5 Pro

gemini-2.5-pro

Our most powerful reasoning model, which excels at coding and complex reasoning tasks.

System instructions

Optional tone and style instructions for the model

Temperature

Media resolution

Default

Thinking

Thinking mode

Set thinking budget

Tools

Structured output

CRÉDITOS

Obrigado pela atenção!

Responsáveis:

Adriano Dias Junior

João Acerbi

Eduardo Juan

Rafael Botti

Danilo Azevedo

Matrícula:

H7451E8

R855DD4

R590AJ8

R8497A1

R660648

Alunos do segundo semestre de Análise e
Desenvolvimento de Sistemas na UNIP, em Jundiaí.

Tecnologias Utilizadas:

Linguagens:

- Python 3.x: Linguagem principal para a interface, lógica do cliente e orquestração do sistema.
- C (DLL/SO): Linguagem usada para o backend de alta performance, gerenciando o banco de dados principal.

Armazenamento de dados:

- JSON: Formato utilizado para armazenar dados de usuários, configurações e anotações de forma legível.
- Arquivos Binários (.dat): Formato eficiente para salvar dados estruturados como notas e presenças, otimizando performance e espaço.

Inteligência Artificial (Desenvolvimento):

- Cursor: Editor de código com IA, utilizado para acelerar o desenvolvimento, refatorar e depurar o código.
- Gemini: Modelo de IA generativa, usado para auxiliar na criação da arquitetura, documentação e na resolução de problemas complexos de lógica.



Link para o Github:

https://github.com/ejuanoli/UNIP_ADS.git

E-mail de contato:

danilo.azevedo2002@gmail.com