Gerenciador de tarefas

🎲 ESTRUTURA DE DADOS

EQUIPE :

1. Jose Mayckon Guedes Morais.
2. Mikael Noberto do nascimento.
3. Matheus Cabral Queiroz.

Introdução

- O **Gerenciador de Tarefas** é um projeto desenvolvido em Python que oferece uma solução simples para organizar e acompanhar suas tarefas diárias. Com essa aplicação, você pode adicionar, remover e visualizar suas tarefas de forma conveniente.

Objetivos

- **Facilitar o Gerenciamento de Tarefas**: Desenvolver uma aplicação que permita aos usuários organizarem e acompanharem suas tarefas diárias de forma eficiente.
- **Promover a Produtividade**: Ajudar os usuários a melhorar sua produtividade pessoal, fornecendo uma ferramenta simples para monitorar suas atividades pendentes e concluídas.
- As metas (no caso as funcionalidades) principais, foram alcançadas e executadas de forma eficiente.

Descrição do Sistema

- O Gerenciador de Tarefas é uma aplicação Python que ajuda os usuários a organizar e acompanhar suas tarefas diárias. A aplicação permite adicionar,

editar, remover e listar tarefas mediante uma interface de linha de comando simples e intuitiva.

- **Funcionalidade Principais**
  - **Adicionar Tarefas**: Permitir a inclusão de novas tarefas com descrições e nível de prioridade.
  - **Remover Tarefas**: Proporcionar a exclusão de tarefas não mais necessárias.
  - **Listar Tarefas**: Exibir todas as tarefas com indicação de prioridade.

diagrama de arquitetura:

```
+-----------------------------------------+
|          Gerenciador de Tarefas         |
|-----------------------------------------|
| - tasks: List[Task]                     |
|-----------------------------------------|
| + adicionar_tarefa(titulo: str, descricao: str, prioridade:
| + remover_tarefa()                      |
| + listar_tarefas()                      |
+--------------------+--------------------+
                     |
                     |
                     v
+--------------------+--------------------+
|        Tarefa                           |
|-----------------------------------------|
| - titulo: str                           |
| - descricao: str                        |
| - prioridade: int                       |
|-----------------------------------------|
| + __init__(titulo: str, descricao: str, prioridade: int) |
| + __str__()                             |
| + __lt__(other)                         |
+-----------------------------------------+
```

Metodologia

- A metodologia **Ágil**, especificamente o framework Scrum, foi escolhida para o desenvolvimento do Gerenciador de Tarefas. Essa abordagem é ideal para projetos de software que requerem flexibilidade e rápida adaptação às mudanças.
- **Planejamento**
  - **Definição de Requisitos**: Coleta dos requisitos funcionais e não funcionais.
  - **Criação do Backlog**: Listagem das funcionalidades desejadas.
- **Design**
  - **Estrutura do Sistema**: Definição da arquitetura do sistema e das classes principais (`Task` e `PriorityQueue`).
  - **Diagrama de Arquitetura**: Visualização da estrutura e interações entre componentes.
- **Implementação**
  - **Desenvolvimento Incremental**: Implementação das funcionalidades principais em ciclos curtos (sprints).
  - **Revisões Regulares**: Revisões e refinamentos após cada sprint.
- Justificativa da metodologia utilizada : A metodologia ágil é uma abordagem flexível para gerir projetos, permitindo ajustes e melhorias contínuas ao longo do desenvolvimento. Em vez de seguir um plano rígido, como na gestão tradicional, os métodos ágeis abraçam mudanças e focam na colaboração, autonomia das equipes e entrega de valor. Além disso, ela gera eficiência, identifica problemas rapidamente e promove um ambiente de trabalho colaborativo.

Ferramentas Utilizadas

- Ambiente de desenvolvimento : VScode
- Linguagem de programação : Python

- Bibliotecas e frameworks : **heapq :** Biblioteca para manipulação da fila de prioridade; **TKinter : Biblioteca para criar a interface grafica.**
- Ferramentas de controle de versão: GitHub
- Trello, Notion

Desenvolvimento do Projeto

- **Semana 1:**
  - Planejamento dos requisitos
  - Polir os requisitos mais importantes
  - Implementação inicial do código
- **Semana 2:**
  - Planejamento da introdução de funções complexas
  - Procura de bibliotecas que ajudam na implementação
  - Aplicar em código
- **Semana 3:**
  - Corrigir bugs
  - Aprender TKinter e suas funcionalidades
  - Aplicar em código
- **Semana 4:**
  - Polir o código de forma geral
  - Implementar boa práticas
  - fazer teste de funcionalidade.

Estado Atual do Projeto

- O projeto está completo, mas seria interessante realizar testes mais extensivos para validar todas as funcionalidades.
- Não foram realizados testes extensivos.

- Nenhuma funcionalidade adicionais.

Resultados e Discussão

- O desempenho foi bom e atendeu às expectativas.
- Algumas funcionalidades previstas no protótipo foram deixadas de lado para melhorar o desempenho do projeto.

Conclusão

- O sistema de gerenciamento de tarefas em python foi bem implementada, os resultados foram satisfatorios.
- As ideias inicias foram implementadas de forma eficiente.
- Sistema de gerenciador de despesas.

Anexos (Bonus):

- Códigos-fonte relevantes

```
import heapq

class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class Task:
    def __init__(self, title, description, priority):
        self.title = title
        self.description = description
```

```
        self.priority = priority

    def __lt__(self, other):
        return self.priority < other.priority

class PriorityQueue:
    def __init__(self):
        self.tasks = []

    def add_task(self, task):
        heapq.heappush(self.tasks, (task.priority, task))

    def remove_task(self):
        if self.tasks:
            return heapq.heappop(self.tasks)[1]
        else:
            return None

class LinkedList:
    def __init__(self):
        self.head = None
        self.tail = None

    def append(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
        else:
            self.tail.next = new_node
        self.tail = new_node

    def print_list(self):
        current_node = self.head
        while current_node:
            print(current_node.data, end=" -> ")
            current_node = current_node.next
        print("None")
```

```
    def add_task(self, data):
        self.append(data)

    def remove_task(self, data):
        current_node = self.head
        previous_node = None
        while current_node:
            if current_node.data == data:
                if previous_node:
                    previous_node.next = current_node.next
                else:
                    self.head = current_node.next
                if current_node == self.tail:
                    self.tail = previous_node
                return
            previous_node = current_node
            current_node = current_node.next
```

```
import tkinter as tk
from tkinter import messagebox
import Gerenciar

priority_queue = Gerenciar.PriorityQueue()

def add_task():
    title = title_entry.get()
    description = description_entry.get()
    priority = priority_entry.get()

    if title and description and priority:
        try:
            priority = int(priority)
            if priority in [1, 2, 3, 4]:
                task = Gerenciar.Task(title, description, pri
                priority_queue.add_task(task)
                title_entry.delete(0, tk.END)
                description_entry.delete(0, tk.END)
                priority_entry.delete(0, tk.END)
```

```
                messagebox.showinfo("Sucesso", "Tarefa adicio
            else:
                messagebox.showerror("Erro", "A prioridade de
        except ValueError:
            messagebox.showerror("Erro", "A prioridade deve s
    else:
        messagebox.showerror("Erro", "Preencha todos os campo

def remove_task():
    removed_task = priority_queue.remove_task()
    if removed_task:
        messagebox.showinfo("Sucesso", f"Tarefa removida: {re
    else:
        messagebox.showinfo("Erro", "Não há tarefas na fila."


def show_tasks():
    tasks = priority_queue.tasks
    task_list.delete(1.0, tk.END)
    if tasks:
        for _, task in tasks:
            task_list.insert(tk.END,
                             f"Título: {task.title}, Descriçã
    else:
        task_list.insert(tk.END, "Não há tarefas na fila.")

root = tk.Tk()
root.title("Gerenciador de Tarefas")

tk.Label(root, text="Título da Tarefa").grid(row=0, column=0)
title_entry = tk.Entry(root)
title_entry.grid(row=0, column=1)

tk.Label(root, text="Descrição da Tarefa").grid(row=1, column
description_entry = tk.Entry(root)
description_entry.grid(row=1, column=1)

tk.Label(root, text="Prioridade da Tarefa (1-Crítica, 2-Alta,
```

```
priority_entry = tk.Entry(root)
priority_entry.grid(row=2, column=1)

add_task_button = tk.Button(root, text="Adicionar Tarefa", co
add_task_button.grid(row=3, column=0, columnspan=2)

remove_task_button = tk.Button(root, text="Remover Tarefa", c
remove_task_button.grid(row=4, column=0, columnspan=2)

show_tasks_button = tk.Button(root, text="Mostrar Tarefas", c
show_tasks_button.grid(row=5, column=0, columnspan=2)

task_list = tk.Text(root, height=10, width=50)
task_list.grid(row=6, column=0, columnspan=2)

root.mainloop()
```

- Diagramas adicionais

```
+-----------------+
|      Node       |
+-----------------+
| - data: any     |
| - next: Node    |
+-----------------+
| + __init__(data)|
+-----------------+

+----------------------------+
|           Task             |
+----------------------------+
| - titulo: str              |
| - descricao: str           |
| - prioridade: int          |
+----------------------------+
| + __init__(titulo, descricao, prioridade) |
| + __lt__(other: Task): bool |
+----------------------------+
```

```
+-----------------------------+
|       PriorityQueue         |
+-----------------------------+
| - tasks: list[Task]         |
+-----------------------------+
| + __init__()                |
| + add_task(task: Task)      |
| + remove_task(): Task       |
+-----------------------------+

+-----------------------------+
|        LinkedList           |
+-----------------------------+
| - head: Node                |
| - tail: Node                |
+-----------------------------+
| + __init__()                |
| + append(data: any)         |
| + print_list()              |
| + add_task(data: any)       |
| + remove_task(data: any)    |
+-----------------------------+
```

- Documentação técnica

Estrutura do Projeto

O projeto do Gerenciador de Tarefas consiste em três classes principais:

1. **Node**: Representa um nó em uma lista encadeada. Cada nó armazena um elemento de dados e uma referência para o próximo nó na lista.
2. **Task**: Representa uma tarefa a ser gerenciada pelo sistema. Cada tarefa possui um título, descrição e prioridade. Esta classe também implementa o método `__lt__` para permitir a comparação de tarefas com base em suas prioridades.
3. **PriorityQueue**: implementa uma fila de prioridade usando a biblioteca `heapq` do Python. Esta classe mantém uma lista de tarefas ordenadas por

prioridade, permitindo adicionar tarefas com eficiência e remover a tarefa de maior prioridade.

Funcionalidades Principais

Adicionar Tarefa

Permite adicionar uma nova tarefa ao sistema, especificando seu título, descrição e prioridade.

Remover Tarefa

Permite remover uma tarefa existente do sistema.

Listar Tarefas

Exibe todas as tarefas atualmente registradas no sistema, incluindo detalhes como título, descrição e prioridade.

Utilização

Uso na Interface Gráfica

Execute o script principal para iniciar o Gerenciador de Tarefas com a interface gráfica:

```
import tkinter as tk
from tkinter import messagebox
import Gerenciar
```

Exemplo de Uso

```
# Importar classes
from gerenciador_de_tarefas import Task, PriorityQueue

# Criar uma nova tarefa
task1 = Task("Estudar para o exame", "Revisar os capítulos
1-5 do livro", 2)

# Criar uma fila de prioridade
queue = PriorityQueue()
```

```
# Adicionar tarefa à fila de prioridade
queue.add_task(task1)

# Remover tarefa da fila de prioridade
task_removida = queue.remove_task()

# Imprimir título da tarefa removida
print("Tarefa Removida:", task_removida.title)
```

Conclusão

O Gerenciador de Tarefas oferece uma solução simples e eficiente para organizar e acompanhar tarefas diárias. Com sua estrutura modular e funcionalidades essenciais, é uma ferramenta versátil que pode ser facilmente integrada em diversos projetos e ambientes de desenvolvimento.