# PROJETO TASK FLOW

Equipe: Flávio dos Santos, Eduarda Lopes, Jenifer Barreto

## Resumo do Tema

O desenvolvimento de um sistema organizador de tarefas visa proporcionar uma solução eficaz para otimizar a gestão das atividades diárias do usuário, promovendo uma abordagem estruturada e funcional para o planejamento e acompanhamento de suas responsabilidades.

Este sistema atua diretamente no combate à procrastinação, à falha de memorização e à desorganização, ao proporcionar um ambiente intuitivo e de fácil acesso, onde o usuário pode não apenas criar e gerenciar suas tarefas, mas também priorizá-las, estabelecer prazos e monitorar seu progresso.

Ao integrar ferramentas de organização e lembretes, o sistema busca aumentar a produtividade, reduzir a sobrecarga cognitiva e, consequentemente, melhorar a eficiência nas tarefas cotidianas, favorecendo o alcance de metas e o desenvolvimento de hábitos positivos.

## Público-Alvo

O público-alvo do sistema é, primordialmente, composto por estudantes universitários, um grupo que frequentemente enfrenta dificuldades com a gestão do tempo e organização das suas atividades diárias. O sistema foi projetado para atender às necessidades específicas desse público, visando aumentar sua produtividade e eficácia no cumprimento das tarefas acadêmicas. Além disso, a ferramenta pode se estender a outros indivíduos que busquem uma maneira eficiente de organizar suas responsabilidades cotidianas, como profissionais e professores.

## Método de Pesquisa Utilizado

Para identificar as funcionalidades mais relevantes para o público-alvo, foi adotada a metodologia de pesquisa baseada em questionários online. Este método se mostrou eficaz para alcançar um público amplo e diversificado de estudantes universitários.

Utilizamos a plataforma Google Forms para facilitar a criação, distribuição e análise dos dados. Através dessa abordagem, foi possível coletar informações diretas sobre as preferências, dificuldades e expectativas dos usuários em relação à gestão de suas tarefas cotidianas. Os dados obtidos permitiram um entendimento aprofundado

das necessidades do público-alvo, contribuindo para o desenvolvimento de funcionalidades que garantem a eficiência e a personalização do sistema.

## Requisitos Identificados

Com base nas respostas obtidas nos questionários e nas validações dos protótipos, foram identificadas as funcionalidades principais que o sistema deveria oferecer:

- Adicionar e Editar Tarefas: Permitir ao usuário criar, modificar e excluir tarefas com facilidade.
- Classificação por Categoria e Status: A possibilidade de organizar as tarefas de acordo com sua natureza (por exemplo, acadêmicas, pessoais) e status (pendente, em progresso, concluída).
- Lembretes e Notificações: Alertas automáticos para garantir que o usuário não esqueça de seus compromissos e prazos.
- Interface Amigável e Intuitiva: Um design que facilite a navegação e a interação com o sistema, com um layout claro e simples.

## Processo de Elicitação

No processo de elicitação, o método escolhido foi a prototipação, com a criação de protótipos rápidos para validação com o público-alvo. Após a análise das respostas dos questionários online, optamos por apresentar modelos de interface simplificados, com funcionalidades-chave, para obter feedback direto dos usuários.

Essa abordagem de prototipação permitiu a iteração rápida e a adaptação do design e das funcionalidades, garantindo que o sistema atendesse melhor às necessidades reais dos estudantes. A prototipação também possibilitou a identificação de ajustes e melhorias antes do desenvolvimento completo, minimizando riscos e otimizando o processo de criação.

## Especificação Final dos Requisitos

A partir dos dados coletados e da validação dos protótipos, os requisitos finais foram definidos, levando em consideração as principais necessidades dos usuários:

- Cadastro de Tarefas: O sistema deve permitir que o usuário crie, edite e apague tarefas, atribuindo a elas categorias e prazos.
- Gestão de Prioridades: O usuário poderá marcar as tarefas com diferentes níveis de prioridade, facilitando a organização das atividades mais urgentes.
- Notificações e Alertas: Lembretes automáticos serão enviados antes dos prazos das tarefas, para garantir que o usuário esteja ciente de suas responsabilidades.

- Interface Simples e Clara: A interface gráfica deve ser intuitiva, com fácil navegação e recursos visuais que ajudem o usuário a compreender e utilizar o sistema de forma eficiente.

## Tecnologias

**Python**: Linguagem de programação utilizada para a construção do sistema, devido à sua versatilidade, simplicidade e amplo suporte para bibliotecas de interfaces gráficas.

**SQLite**: Banco de dados leve e eficiente utilizado para armazenar as tarefas e informações do usuário, proporcionando uma solução simples para a persistência dos dados.

**Bibliotecas**: tkinter

## Recursos e Restrições

Recursos Disponíveis:
**Flávio**: Responsável pelo Frontend e design da interface, garantindo uma experiência de usuário fluida e agradável.
**Eduarda**: Responsável pelo Backend e documentação técnica, assegurando que a lógica do sistema esteja bem implementada e documentada.
**Jenifer**: Responsável pelos testes e pela documentação, realizando testes de funcionalidade e garantindo a qualidade do sistema.

## Aplicação de Design de Software no Projeto

Utilizamos o SRP no organizador de tarefas, separando as responsabilidades entre a interface gráfica e o gerenciamento de tarefas.

A interface gráfica deve ser responsável apenas pela apresentação e interação com o usuário, enquanto o gerenciamento das tarefas, como adição, remoção e armazenamento, deve ser feito por outra classe.

Ficando assim:
Classe de Tarefa: Responsável por armazenar e manipular os dados das tarefas. Classe de Interface Gráfica: Responsável apenas pela interação com o usuário.

## Modelos de Design

Escolhemos MVC (Model-View-Controller) como padrão de design de software, pois ele separa as preocupações e responsabilidades do software em três componentes principais:

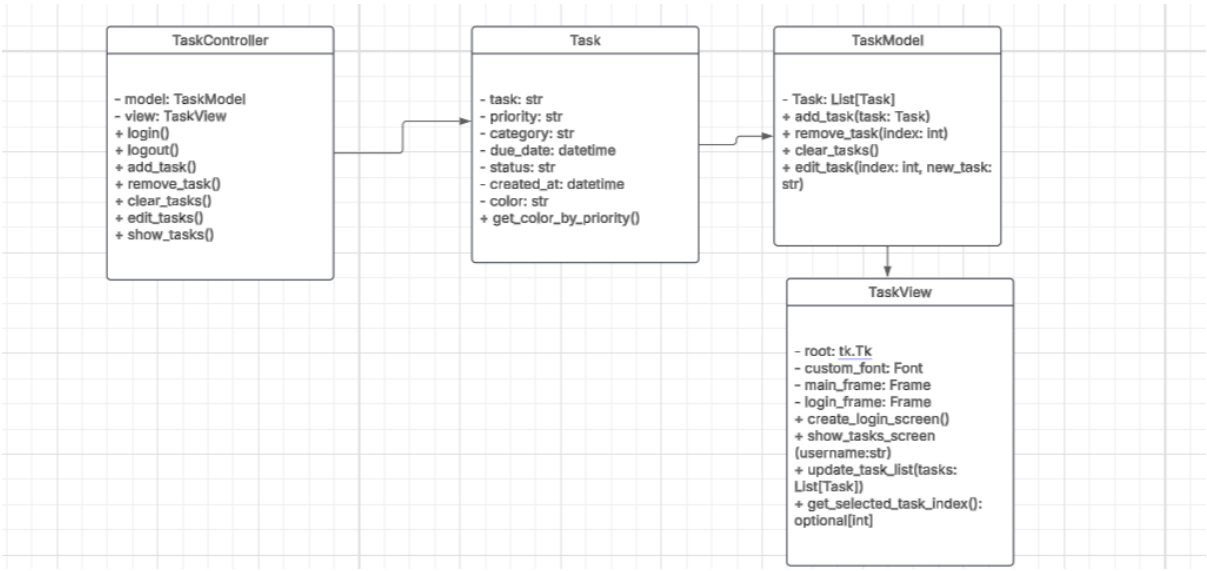
**Model (Modelo):** que é responsável pela lógica de negócios e pela manipulação dos dados. Ele armazena as informações e a lógica de como essas informações são alteradas.

**View (Visão):** cuida da apresentação visual para o usuário. Ele exibe os dados do Model e reflete qualquer alteração feita neles.
**Controller (Controlador):** atua como intermediário entre a View e o Model. Ele recebe entradas do usuário, processa essas entradas (interagindo com o Model) e atualiza a View.

Isso permite uma maior flexibilidade e modularidade do código, facilitando a manutenção e a evolução do software ao longo do tempo. Como o projeto já menciona o SRP, o MVC se encaixa perfeitamente, pois reforça essa separação de responsabilidades.

## UML Caso de uso:

## Diagrama de Classe:



## Padrões de Criação:

Integramos o padrão Factory Method de forma que o Controller use a factory para criar instâncias de Task e gerenciar as tarefas.

Criamos uma factory para criar tarefas com diferentes características, como diferentes prioridades. Além de fazer algumas melhorias na organização e na maneira como as tarefas são manipuladas.

```
from abc import ABC, abstractmethod
from model import Task

class TaskFactory(ABC):
    @abstractmethod
    def create_task(self, task_name, priority, category, due_date):
        pass

class LowPriorityTaskFactory(TaskFactory):
    def create_task(self, task_name, priority, category, due_date):
        return Task(task_name, "Baixa", category, due_date)

class MediumPriorityTaskFactory(TaskFactory):
    def create_task(self, task_name, priority, category, due_date):
        return Task(task_name, "Média", category, due_date)

class HighPriorityTaskFactory(TaskFactory):
    def create_task(self, task_name, priority, category, due_date):
        return Task(task_name, "Alta", category, due_date)
```

## Padrões de estruturais:

O **padrão Facade** foi utilizado para gerenciar diferentes aspectos das tarefas, como o agendamento, a execução, e o controle de status.

Utilizamos o padrão Facade, para criar uma interface simples para o usuário interagir com o gerenciador de tarefas, sem precisar entender toda a complexidade por trás disso e simplificando a interação entre as classes.

## Padrões comportamentais:

### Command (Comando)

Utilizamos o padrão Command para encapsular todas as informações necessárias para realizar uma ação ou comando em uma tarefa. Ex:

- Adicionar uma nova tarefa
- Marcar uma tarefa como concluída
- Editar detalhes de uma tarefa
- Deletar uma tarefa

Cada ação se torna um objeto de comando, o que permite um controle maior sobre as ações executadas no sistema, incluindo a possibilidade de desfazer ou refazer ações.

## Arquitetura de sistemas:

**Análise e Pesquisa:**

O TaskFlow é um gerenciador de tarefas (no estilo Kanban), organizado em uma estrutura MVC (Model-View-Controller), que promove uma clara separação de responsabilidades. Para definir a arquitetura que mais se adequa ao nosso sistema, foram analisados os seguintes requisitos:

**Escalabilidade**: O sistema é local e modular, mas não é totalmente monolítico, pois já está dividido em componentes bem definidos (Model, View, Controller). Isso facilita a escalabilidade futura, pois cada componente pode ser aprimorado ou substituído independentemente.

**Manutenção**: A separação em Model, View e Controller facilita a manutenção, pois cada componente tem uma responsabilidade clara. O Model lida com a lógica relacionada ao gerenciamento das tarefas, a View lida com a interface do usuário, e o Controller gerencia a interação entre os dois. Isso permite que alterações sejam feitas de forma isolada, sem impactar outras partes do sistema.

**Complexidade**: A complexidade atual do sistema ainda é baixa, pois ele lida apenas com a criação, edição, exclusão e organização de tarefas. No entanto, a estrutura MVC permite a adicionar novas funcionalidades sem impactar muito o código existente.

**Flexibilidade**: A arquitetura em camadas (MVC) proporciona flexibilidade para futuras modificações.

## Pesquisa de Exemplos Reais:

### Todoist (versão inicial)

**Descrição**: O Todoist é um gerenciador de tarefas popular, mas em sua versão inicial, era um sistema simples, focado em listas de tarefas e prioridades.

**Arquitetura**: Na sua fase inicial, o Todoist provavelmente utilizava uma arquitetura monolítica ou em camadas (MVC), pois o foco era a simplicidade e a entrega rápida de funcionalidades básicas.

## Escolha da Arquitetura

Com base na análise das necessidades do sistema e na estrutura atual, a arquitetura mais adequada para o TaskFlow é a arquitetura em camadas (MVC). Informações sobre o porquê da escolha:

**Separação de Responsabilidades**: A estrutura MVC já está implementada no projeto, com a separação clara entre Model, View e Controller. Isso facilita a manutenção e a evolução do sistema.

**Modularidade**: A arquitetura em camadas permite que o sistema seja dividido em componentes independentes, o que facilita a adição de novas funcionalidades e a substituição de partes do sistema sem afetar o todo.

**Simplicidade**: é mais simples de desenvolver e implantar, especialmente para um sistema com funcionalidades básicas

## Justificativa da Escolha

A escolha da arquitetura em camadas (**MVC**) é baseada nos seguintes aspectos:

**Escalabilidade**: No momento, o sistema não precisa escalar horizontalmente, pois é destinado a um uso local e individual. Caso haja necessidade de suportar múltiplos usuários no futuro, a migração pode ser considerada.

**Simplicidade de Deploy**: Uma arquitetura em camadas é mais fácil de implantar, pois consiste em componentes bem definidos que podem ser executados em um único ambiente.

**Complexidade de Desenvolvimento**: Como o sistema possui funcionalidades simples, a arquitetura em camadas reduz a complexidade de desenvolvimento, permitindo que a equipe se concentre na implementação das funcionalidades principais.

## Implementação do Protótipo

O protótipo do **Task Flow** foi desenvolvido utilizando as seguintes tecnologias:

**Python**: A linguagem foi escolhida por sua simplicidade e amplo suporte a bibliotecas de interface gráfica, como o **Tkinter**, que foi utilizado para criar a interface do usuário.

**MVC (Model-View-Controller)**: O padrão foi adotado para separar as responsabilidades do sistema, com o **Model** responsável pela lógica de negócios e armazenamento de dados, a **View** pela interface do usuário e o **Controller** pela intermediação entre os dois.

## Funcionalidades Implementadas:

As funcionalidades implementadas no protótipo incluem:
**Cadastro de Tarefas:** O usuário pode criar, editar e excluir tarefas, atribuindo categorias e prazos.

**Gestão de Prioridades:** As tarefas podem ser classificadas por prioridade, facilitando a organização das atividades mais urgentes.

## Referências:

## LINK DO REPOSITÓRIO:

*https://github.com/FreitasLopes/TASKFLOW*

## LINK DAS RESPOSTAS DO FORMULÁRIO:

TASK FLOW (respostas).xlsx

## LINK LUCIDCHART – UML CASO DE USO:

https://lucid.app/lucidchart/8ffec047-f180-4d4d-a4fd-c8fb2ed5598b/edit?invitationId=inv_dd099546-a877-47f1-927b-0083e6ddf314

## LINK LUCIDCHART – UML DIAGRAMA DE CLASSE:

https://lucid.app/lucidchart/040bd115-9e0c-4f0e-93b3-c884fdd90e56/edit?viewport_loc=-1401%2C-742%2C2462%2C942%2C0_0&invitationId=inv_46b5d148-d41a-4f3e-a619-94a17cdb62b1