⚠️ INSTRUÇÕES ⚠️

☝🏻 Apenas um dos integrantes deve fazer a entrega do grupo.

⏳ A entrega deve ser feita pela atividade correspondente no classroom, entrega de outras formas **não** serão aceitas. Passado o prazo no classroom o sistema **não** admite mais entregas.

🤖 Fazer a atividade com a ajuda de IA é **diferente** da IA fazer sua atividade. Use-a com **sabedoria**, como para entender o que você fez de errado e como fazer da forma correta. Seu futuro profissional agradece.

A entrega deve ser feita por meio de um LINK do repositório no github, com indentificação do grupo (ex: `Entrega atividade ... Grupo 10`).

O objetivo deste exercício é evoluir a aplicação vista em aula: https://github.com/giovbon/MVC-flask. Com base nesse esqueleto você irá construir um “Gerenciador de Tarefas” (To-Do List) onde os usuários, já existentes no seu banco de dados, podem ter tarefas associadas a eles.

A aplicação utilizará a arquitetura MVC para separar as responsabilidades, o Flask como framework, o SQLAlchemy para interagir com o banco de dados SQLite, e o Jinja2 (templates do Flask) para criar a interface do usuário.

Objetivo Principal

- Aplicar o padrão MVC de forma prática em um novo módulo da aplicação.
- Criar e gerenciar relacionamentos entre tabelas no banco de dados (um Usuário tem várias Tarefas).
- Realizar operações CRUD (Create, Read, Update, Delete) completas em um novo recurso (“Tarefas”), interagindo com o banco de dados.
- Construir uma interface web simples com HTML para interagir com a aplicação.

Requisitos Funcionais

Você continuará trabalhando no mesmo projeto, adicionando a funcionalidade de “Tarefas”.

1. Model (`models/task.py`)

- Crie um novo arquivo `models/task.py`.
- Dentro dele, defina um novo modelo `Task` que representará a tabela de tarefas no banco de dados.
- A tabela `tasks` deve ter os seguintes campos:
  - `id` (Integer, Chave Primária)
  - `title` (String, não pode ser nulo)
  - `description` (String, pode ser nulo)
  - `status` (String, não pode ser nulo, com valor padrão ‘Pendente’)
  - `user_id` (Integer, Chave Estrangeira que se relaciona com o `id` da tabela `users`).

2. Controller (`controllers/task_controller.py`)

- Crie um novo arquivo `controllers/task_controller.py`.

- Dentro dele, crie uma classe `TaskController` com métodos estáticos para gerenciar a lógica das tarefas:
  - Listar Tarefas (`list_tasks`):
    - Busca todas as tarefas do banco de dados. (Dica: para exibir o nome do usuário, você precisará usar o relacionamento definido no model).
    - Renderiza um template HTML (`tasks.html`), passando a lista de tarefas para ele.
  - Criar Tarefa (`create_task`):
    - Usando o método POST, é capturado os dados do formulário (título, descrição, usuário selecionado), cria um novo objeto `Task`, salva no banco de dados e redireciona o usuário para a página de listagem de tarefas.
  - Atualizar Status da Tarefa (`update_task_status`):
    - Recebe o `task_id` pela URL.
    - Busca a tarefa no banco de dados. Se a encontrar, altera seu status de 'Pendente' para 'Concluído' (ou vice-versa).
    - Redireciona de volta para a lista de tarefas.
  - Excluir Tarefa (`delete_task`):
    - Recebe o `task_id` pela URL.
    - Encontra a tarefa no banco de dados e a remove.
    - Redireciona de volta para a lista de tarefas.

3. Views (`view/templates/`)

- Crie dois novos arquivos HTML na sua pasta de templates:
  - `tasks.html`:
    - Deve exibir o título "Minhas Tarefas".
    - Deve ter um link para a página de criação de nova tarefa.
    - Deve listar todas as tarefas em uma tabela ou lista. Para cada tarefa, exibir: título, descrição, status e o nome do usuário a quem ela pertence.
    - Para cada tarefa, deve haver um botão/link para "Concluir" (que chama a rota de atualização de status) e um botão/link para "Excluir".
  - `create_task.html`:
    - Deve conter um formulário (`<form method="POST">`) para criar uma nova tarefa.
    - O formulário deve ter campos para `title` (texto), `description` (área de texto) e um campo de seleção (`<select>`) para escolher a qual usuário a tarefa será atribuída. Este `<select>` deve ser populado dinamicamente com os usuários vindos do controller.

4. Roteamento (`app.py`)

- No seu arquivo `app.py`, importe o novo `TaskController`.
- Adicione as novas rotas, conectando as URLs aos métodos do `TaskController`:
  - `GET /tasks` -> `TaskController.list_tasks`
  - `GET, POST /tasks/new` -> `TaskController.create_task`
  - `POST /tasks/update/<int:task_id>` -> `TaskController.update_task_status`
  - `POST /tasks/delete/<int:task_id>` -> `TaskController.delete_task`

Requisitos Técnicos

1. Banco de Dados: Use o mesmo arquivo de banco de dados SQLite (`users.db`). O SQLAlchemy irá criar a nova tabela `tasks` nele automaticamente quando a aplicação for executada.

2. Estrutura MVC: Siga rigorosamente a separação de responsabilidades. Nenhuma lógica de banco de dados deve estar na View. Nenhuma tag HTML deve estar no Controller.
3. Relacionamento: A chave do exercício é implementar corretamente o relacionamento `One-to-Many` (Um Usuário para Muitas Tarefas) usando a `ForeignKey` do SQLAlchemy.