# **Relatório do Projeto: Sistema de Gerenciamento de Expedições Espaciais**

### **1. Introdução**

Este projeto implementa um Sistema de Gerenciamento de Expedições Espaciais usando o framework Flask. A aplicação permite gerenciar missões espaciais, incluindo operações de criação, leitura, atualização e exclusão (CRUD). A aplicação também possui uma interface web simples que facilita a interação com o banco de dados de missões.

### **2. Objetivo do Projeto**

O objetivo deste sistema é permitir o controle eficiente de informações sobre missões espaciais, como o nome da missão, data de lançamento, destino, tripulação, carga útil, duração, custo e status. As principais operações incluem:

* Criar novas missões.
* Listar todas as missões cadastradas.
* Atualizar informações de missões existentes.
* Excluir missões quando necessário.

### **3. Estrutura do Projeto**

#### **Arquivos e Diretórios**

* **app.py**: Arquivo principal, onde são configuradas as rotas e a aplicação Flask é inicializada.
* **forms.py**: Contém o formulário MissionForm, utilizado para validar os dados de entrada do usuário.
* **models.py**: Define o modelo Mission, representando a estrutura das tabelas do banco de dados.
* **templates**: Diretório que contém os arquivos HTML para o frontend, como index.html, create.html e update.html.
* **requirements.txt**: Lista de dependências necessárias para executar o projeto.

### **Desenvolvimento das Funcionalidades**

#### **4. Configuração Inicial**

A configuração do ambiente foi realizada no arquivo app.py. Neste arquivo:

* A aplicação Flask é configurada para usar o banco de dados SQLite (missions.db).
* Uma chave secreta é definida para garantir segurança no uso de formulários.

#### **5. Definição do Modelo (models.py)**

O modelo Mission representa uma missão espacial e é implementado usando o SQLAlchemy. Os campos são:

* id: Chave primária.
* name: Nome da missão.
* launch\_date: Data de lançamento.
* destination: Destino da missão.
* state: Estado atual da missão.
* crew: Informação sobre a tripulação.
* payload: Detalhes sobre a carga útil.
* duration: Duração estimada da missão.
* cost: Custo estimado da missão.
* status: Status atual da missão.

Esses campos foram escolhidos para permitir um acompanhamento completo das missões.

#### **6. Criação de Formulários (forms.py)**

O arquivo forms.py contém o formulário MissionForm, desenvolvido com Flask-WTF. Este formulário facilita a entrada de dados pelo usuário com os seguintes campos:

* Campos como name, launch\_date, destination, state e cost são obrigatórios, garantindo dados mínimos essenciais para cada missão.
* Os campos opcionais incluem crew, payload, duration e status, permitindo flexibilidade na descrição de cada missão.

#### **7. Desenvolvimento das Rotas (app.py)**

As principais rotas desenvolvidas foram:

* **index** (/): Exibe todas as missões em ordem decrescente de data de lançamento, facilitando a visualização das missões mais recentes.
* **create\_mission** (/mission/new): Permite criar uma nova missão. A página utiliza o template create.html, onde o usuário preenche um formulário. Após o envio, a missão é salva no banco de dados.
* **update\_mission** (/mission/<int:id>/edit): Permite editar uma missão específica. O usuário é redirecionado para update.html, onde ele pode atualizar as informações de uma missão selecionada.
* **delete\_mission** (/mission/<int:id>/delete): Permite excluir uma missão. Essa operação é feita via um botão de exclusão na página principal (index.html).

#### **8. Criação do Frontin (Templates)**

O Frontin foi desenvolvido com HTML básico e utiliza três templates principais:

* **index.html**: Exibe a lista de todas as missões, com botões para editar ou excluir cada missão.
* **create.html**: Formulário para criação de uma nova missão.
* **update.html**: Formulário para atualização de uma missão existente.