**Estruturação do Projeto (Back-End)**

**1. Organização de Pastas**

A estrutura do projeto pode seguir a estrutura de uma aplicação Flask com SQLAlchemy para persistência de dados.

* **/app**
  + **/templates**: Contém os arquivos HTML do front-end (não será necessário para você, mas pode ser útil para a integração).
  + **/static**: Arquivos estáticos como CSS, JavaScript, imagens.
  + **/uploads**: Para os arquivos que os usuários enviam (ex: PDF de certificados).
  + **config.py**: Configurações do Flask (ex: conexão com o banco de dados).
  + **forms.py**: Formulários usados no Flask, validando e tratando os dados enviados.
  + **models.py**: Definição dos modelos de banco de dados (SQLAlchemy).
  + **routes.py**: Define as rotas e os comportamentos do sistema.
  + **run.py**: Arquivo principal que inicializa a aplicação Flask.

**2. Definição de Entidades e Banco de Dados**

* **Tabela de Candidatos**: Com os campos como nome, CPF, endereço, data de nascimento, etc.
* **Tabelas de Arquivos**: Armazenando os PDFs e outros documentos enviados pelos candidatos.
* **Tabelas de Certificados**: Armazenando a relação entre candidatos e certificados, como o certificado NR10.

**Modelo básico** em SQLAlchemy para a tabela de candidatos:

python

Copiar

from app import db

class Candidato(db.Model):

id = db.Column(db.Integer, primary\_key=True)

nome = db.Column(db.String(100))

cpf = db.Column(db.String(11), unique=True)

email = db.Column(db.String(100), unique=True)

data\_nascimento = db.Column(db.Date)

estado\_civil = db.Column(db.String(50))

# e outros campos necessários para a ficha

**3. Validações e Regras de Negócio**

* **Validação de CPF**: Como discutido, você usará o gerador de CPF para garantir que o formato seja correto. No entanto, você pode querer utilizar uma API para validar se o CPF existe ou pertence a alguém, o que pode ser feito futuramente.
* **Validação de Email**: A validação do email também pode ser implementada para garantir que ele tenha o formato correto.

**4. Roteamento e Lógica de Processamento**

* A rota /formulario será responsável por exibir o formulário ao usuário e, ao submeter, validar os dados e salvar no banco.
* A rota /confirmacao pode exibir uma tela de sucesso após o envio, como visto no exemplo.

python

Copiar

from app import app

from flask import render\_template, redirect, url\_for

from app.forms import CandidatoForm

from app.models import Candidato

@app.route('/formulario', methods=['GET', 'POST'])

def formulario():

form = CandidatoForm()

if form.validate\_on\_submit():

candidato = Candidato(nome=form.nome.data, cpf=form.cpf.data, ...)

db.session.add(candidato)

db.session.commit()

return redirect(url\_for('confirmacao'))

return render\_template('formulario.html', form=form)

@app.route('/confirmacao')

def confirmacao():

return render\_template('confirmacao.html')

**Etapas de Implementação**

* **Configuração do Flask**: Inicializar o aplicativo, configurar a conexão com o banco de dados.
* **Criação das Rotas**: Definir as rotas para exibir os formulários e processar as informações.
* **Validação de Dados**: Garantir que o CPF, o email e os outros campos sejam validados corretamente.
* **Armazenamento de Arquivos**: Implementar o envio de arquivos PDF para o servidor.
* **Testes**: Testar os fluxos de cadastro e garantir que a validação esteja funcionando corretamente.

**Requisitos para Front-End (Alan e Vagner):**

* Formulário interativo, com feedback visual em tempo real. (precisamos verificar se tá funcionando)
* Validação dos campos no lado do cliente (JavaScript). (projeto não tem nada em Java)
* Tela de confirmação de envio. (essa existe, verificar)
* Arquivos de estilo (CSS) para uma aparência profissional e limpa. (tem que ser criado ainda)