Prompt de Melhorias para o Sistema de Gestão de Vagas

1. Criação de Vagas (Experiência da Empresa)

Objetivo: Melhorar a experiência da empresa ao criar e gerenciar vagas, adicionando mais opções de personalização e controle.
Detalhes da Implementação:
•Novos Campos (Não Obrigatórios):
•Categoria da Vaga:
•Funcionalidade: Permitir que a empresa selecione a categoria da vaga (e.g., Tecnologia, Saúde, Educação) a partir de uma lista suspensa com sugestões. Deve haver uma opção para adicionar novas categorias, que serão persistidas no banco de dados para uso futuro.
•Implementação:
•Frontend (templates/criar_vaga.html, templates/editar_vaga.html): Adicionar um campo <select> para a categoria com opções pré-definidas e um campo de texto (<input type="text">) para adicionar novas categorias. Validar a entrada do usuário para evitar categorias duplicadas.
•Backend (app.py - rotas de criação/edição de vaga): Modificar as funções de tratamento de formulário para receber e validar o novo campo categoria. Persistir a categoria selecionada ou a nova categoria adicionada no banco de dados (tabela Vaga). Criar uma nova tabela Categoria se ainda não existir, para gerenciar as categorias disponíveis.
•Banco de Dados (recrutamento.db): Adicionar uma coluna categoria_id na tabela Vaga e criar uma nova tabela Categoria com id e nome.
•Urgência da Contratação:
•Funcionalidade: Adicionar um campo para indicar a urgência da contratação (e.g., Imediata, Em até 30 dias, Sem urgência) através de um dropdown.
•Implementação:
•Frontend (templates/criar_vaga.html, templates/editar_vaga.html): Adicionar um campo <select> com as opções de urgência.
•Backend (app.py - rotas de criação/edição de vaga): Modificar as funções para receber e persistir o campo urgencia_contratacao na tabela Vaga.
•Banco de Dados (recrutamento.db): Adicionar uma coluna urgencia_contratacao na tabela Vaga.
•Congelamento Agendado:
•Funcionalidade: Permitir o pré-agendamento do congelamento da vaga, com a seleção de uma data específica em um calendário. Ao atingir essa data, a vaga deve ser automaticamente marcada como
“congelada” ou “inativa”. * Implementação: * Frontend (templates/criar_vaga.html, templates/editar_vaga.html): Adicionar um campo de data (<input type="date">) para a seleção da data de congelamento. * Backend (app.py - rotas de criação/edição de vaga, e um novo módulo para tarefas agendadas): Modificar as funções para receber e persistir o campo data_congelamento na tabela Vaga. Implementar uma tarefa agendada (e.g., usando APScheduler ou um cron job externo) que verifique diariamente as vagas com

data_congelamento atingida e atualize seu status para “congelada” no banco de dados. * Banco de Dados (recrutamento.db): Adicionar uma coluna data_congelamento na tabela Vaga.
•Funcionalidade Completa: Todos esses novos campos devem ser funcionais no sistema, o que implica em:
•Frontend: Tratamento nos formulários de criação e edição de vagas (templates/criar_vaga.html, templates/editar_vaga.html).
•Backend: Armazenamento no banco de dados (app.py, modelos de dados). As rotas de criação e edição de vagas devem ser atualizadas para processar e validar esses novos campos.
•Lógica de Exibição e Busca: Inclusão desses campos na lógica de exibição de vagas (e.g., templates/lista_vagas.html, templates/detalhes_vaga.html) e na lógica de busca e filtragem (ver Seção 3).

1.2. Localização da Empresa

Objetivo: Simplificar o processo de cadastro de localização para as empresas e associá-lo diretamente à vaga.
Detalhes da Implementação:
•Cadastro da Empresa:
•Funcionalidade: Solicitar a localização completa da empresa (endereço, cidade, estado, CEP) durante o processo de cadastro da empresa.
•Implementação:
•Frontend (templates/cadastro_empresa.html): Adicionar campos de formulário para endereço, cidade, estado e CEP.
•Backend (app.py - rota de cadastro de empresa): Modificar a função de cadastro para receber e persistir esses dados na tabela Empresa.
•Banco de Dados (recrutamento.db): Adicionar colunas para endereco, cidade, estado, cep na tabela Empresa.
•Reutilização de Endereço na Vaga:
•Funcionalidade: Permitir que a empresa reutilize o endereço cadastrado em seu perfil ao criar uma nova vaga, evitando a redigitação. Deve haver uma opção para usar um endereço diferente, se necessário.
•Implementação:
•Frontend (templates/criar_vaga.html, templates/editar_vaga.html): Adicionar um checkbox ou botão para “Usar endereço da empresa” que preenche automaticamente os campos de localização da vaga com os dados da empresa. Caso desmarcado, os campos de localização da vaga devem ser editáveis.
•Backend (app.py - rotas de criação/edição de vaga): Ao criar ou editar uma vaga, se a opção de reutilizar o endereço da empresa for selecionada, os dados de localização da empresa devem ser copiados para os campos de localização da vaga na tabela Vaga.
•Banco de Dados (recrutamento.db): Garantir que a tabela Vaga tenha colunas para localizacao_cidade, localizacao_estado, etc., para armazenar a localização específica da vaga, que pode ser diferente da empresa.

2. Inteligência Artificial e Score

Esta seção aborda o aprimoramento da lógica de IA para cálculo de score e matching, bem como a implementação de um assistente virtual para candidatos, garantindo que os novos campos de vaga sejam considerados.

2.1. Score e Matching

Objetivo: Aprimorar a lógica de cálculo do score de compatibilidade entre candidato e vaga, tornando-o mais preciso e relevante, e integrando os novos critérios de vaga.
Detalhes da Implementação:
•Critérios de Avaliação Abrangentes:
•Funcionalidade: A IA deve considerar todos os critérios definidos pela empresa ao criar a vaga, incluindo os novos campos como categoria, urgencia_contratacao e a localização da vaga, além de experiência, formação, habilidades, etc.
•Implementação:
•Backend (avaliador/avaliador_hf.py, avaliador/avaliador_local.py, avaliador/base_avaliador.py): Modificar as funções de avaliação de score para receber e processar os novos campos da vaga. A lógica de matching deve ser atualizada para ponderar esses novos critérios conforme sua relevância.
•Banco de Dados (recrutamento.db): Acessar os dados dos novos campos da tabela Vaga para o cálculo do score.
•Proximidade Geográfica:
•Funcionalidade: Incluir a proximidade geográfica entre o candidato e a vaga como um fator relevante no cálculo do score. Isso pode envolver o cálculo de distância ou a comparação de cidades/estados.
•Implementação:
•Backend (avaliador/avaliador_hf.py, avaliador/avaliador_local.py): Integrar uma lógica de cálculo de distância (e.g., usando coordenadas geográficas ou comparação de strings de localização) entre o endereço do candidato (do perfil) e a localização da vaga. Atribuir um peso a essa distância no score final.
•Critérios Obrigatórios:
•Funcionalidade: Tratar critérios marcados como obrigatórios pela empresa. Se um candidato não atender a um critério obrigatório (e.g., uma habilidade específica, uma formação mínima, ou um dos novos campos como categoria ou urgencia_contratacao se a empresa definir como obrigatório), a vaga não deve ser exibida para ele nas listagens e buscas.
•Implementação:
•Backend (app.py - rotas de busca/listagem de vagas, avaliador/base_avaliador.py): Antes de exibir uma vaga para um candidato, realizar uma verificação dos critérios obrigatórios. Se qualquer critério obrigatório não for atendido, a vaga deve ser filtrada da lista de resultados.
•Atualização Dinâmica do Score:
•Funcionalidade: O score deve ser recalculado automaticamente sempre que o perfil do candidato (e.g., atualização de currículo, endereço) ou os detalhes da vaga (e.g., edição de descrição, atualização de novos campos) forem alterados.
•Implementação:
•Backend (app.py - rotas de atualização de perfil/vaga, avaliador/main.py): Chamar a função de cálculo de score (avaliador.main.calcular_score) sempre que houver uma atualização relevante no perfil do candidato ou nos dados da vaga. Considerar a implementação de um

sistema de filas (e.g., Celery com Redis) para processamento assíncrono de recálculos de score em larga escala, evitando impacto na performance da aplicação principal.

2.2. Assistente IA para Candidatos

Objetivo: Implementar um assistente virtual baseado em IA no painel do candidato para oferecer orientação e suporte personalizados, utilizando as informações de score aprimoradas.
Detalhes da Implementação:
•Funcionalidades do Assistente:
•Dicas de Melhoria de Score:
•Funcionalidade: Fornecer dicas personalizadas sobre como o candidato pode melhorar seu score nas vagas que marcou como favoritas. Isso pode incluir sugestões de habilidades a desenvolver, cursos a fazer, ou informações a adicionar ao currículo.
•Implementação:
•Backend (avaliador/main.py, novo módulo de IA para assistente): A IA deve analisar o gap entre o perfil do candidato e os requisitos das vagas favoritas, gerando sugestões acionáveis. Isso pode envolver o uso de modelos de linguagem (LLMs) para gerar texto explicativo.
•Frontend (templates/dashboard_candidato.html): Exibir as dicas de forma clara e acessível no painel do candidato.
•Recomendação de Vagas (“Vagas que são sua cara”):
•Funcionalidade: Sugerir vagas com alta compatibilidade com base no perfil completo do candidato (incluindo currículo, histórico de candidaturas, vagas favoritas) e comportamento na plataforma.
•Implementação:
•Backend (avaliador/main.py, novo módulo de recomendação): Utilizar o score de compatibilidade e algoritmos de recomendação (e.g., filtragem colaborativa, recomendação baseada em conteúdo) para identificar vagas relevantes. Os novos campos de vaga (categoria, urgencia_contratacao, localização) devem ser integrados a essa lógica de recomendação.
•Frontend (templates/dashboard_candidato.html): Criar uma seção dedicada na dashboard do candidato para exibir essas recomendações de forma proeminente.
•Análise de Currículo e Feedback:
•Funcionalidade: Identificar e destacar as principais falhas ou ausências de informações no currículo do candidato que possam impactar negativamente seu score ou visibilidade.
•Implementação:
•Backend (utils/resume_extractor.py, novo módulo de análise de currículo): Aprimorar o extrator de currículos para identificar lacunas (e.g., falta de experiência em áreas relevantes, ausência de certificações). Utilizar a IA para gerar feedback construtivo.
•Frontend (templates/editar_perfil_candidato.html, templates/dashboard_candidato.html): Exibir o feedback da análise de currículo de forma clara, talvez com sugestões de como preencher as lacunas.

3. Filtros e Buscas Avançadas

Esta seção detalha a implementação de filtros de busca avançada para candidatos e empresas, garantindo que os novos campos de vaga e perfil sejam utilizáveis.

3.1. Busca do Candidato

Objetivo: Oferecer ao candidato ferramentas de busca mais robustas para encontrar vagas relevantes.
Detalhes da Implementação:
•Busca por Palavra-chave:
•Funcionalidade: Permitir a busca de vagas por palavra-chave no título, descrição e outros campos relevantes da vaga.
•Implementação:
•Frontend (templates/lista_vagas.html, barra de busca): Adicionar um campo de texto para a palavra-chave.
•Backend (app.py - rota de busca de vagas): Implementar uma query no banco de dados que utilize LIKE ou um motor de busca de texto completo (e.g., SQLite FTS5, se aplicável) para pesquisar a palavra-chave nos campos relevantes da tabela Vaga.
•Filtros Adicionais:
•Funcionalidade: Adicionar filtros por Localização (cidade/estado), Categoria (utilizando o novo campo), Tipo da vaga (e.g., remoto, presencial, híbrido) e Urgência da contratação (utilizando o novo campo).
•Implementação:
•Frontend (templates/lista_vagas.html, barra lateral de filtros): Adicionar elementos de formulário (<select>, checkboxes) para cada filtro.
•Backend (app.py - rota de busca de vagas): Modificar a query de busca para incluir cláusulas WHERE baseadas nos filtros selecionados pelo usuário. As opções de categoria e urgência devem ser carregadas dinamicamente do banco de dados ou de constantes definidas.

3.2. Busca da Empresa (Manual de Candidatos)

Objetivo: Aprimorar as ferramentas de busca e gerenciamento de candidatos para a empresa, permitindo filtros refinados.
Detalhes da Implementação:
•Filtros Refinados:
•Funcionalidade: Adicionar filtros refinados na busca manual de candidatos, incluindo Curso/Formação, Experiência profissional, Localização (do candidato), Avaliação automática (score) (faixa de score), e Disponibilidade.
•Implementação:
•Frontend (nova página/modal de busca de candidatos para empresas): Criar uma interface de busca com múltiplos campos de filtro.
•Backend (app.py - nova rota de busca de candidatos para empresas): Implementar uma query complexa no banco de dados que una as tabelas Candidato, Curriculo (ou dados extraídos do currículo) e possivelmente Score para filtrar candidatos com base nos critérios fornecidos. A localização do candidato deve ser extraída do perfil ou currículo.

4. Dashboard, Favoritos e Navegação

Esta seção foca na reformulação das dashboards de candidatos e empresas para uma melhor usabilidade e acesso à informação.

4.1. Dashboard do Candidato

Objetivo: Reformular a dashboard do candidato para oferecer uma visão mais estratégica e personalizada das vagas e candidaturas.
Detalhes da Implementação:
•Filtros da Dashboard:
•Funcionalidade: Implementar filtros para organizar a exibição de vagas: Principais (top 3 vagas com melhor ranking/score), Todas (todas as vagas disponíveis), Minhas Candidaturas (vagas para as quais já se inscreveu), e Favoritas (vagas marcadas como favoritas).
•Implementação:
•Frontend (templates/dashboard_candidato.html): Adicionar abas ou botões de filtro na dashboard. A exibição das vagas deve ser dinâmica, atualizando-se com base no filtro selecionado.
•Backend (app.py - rota da dashboard do candidato): Modificar a rota para buscar as vagas de acordo com o filtro ativo. Para Principais, buscar as 3 vagas com maior score para o candidato. Para Minhas Candidaturas, buscar vagas associadas ao histórico de candidaturas do usuário. Para Favoritas, buscar vagas marcadas como favoritas.
•Funcionalidade de Favoritos:
•Funcionalidade: Permitir que o candidato marque e desmarque vagas como favoritas diretamente da listagem, utilizando um botão visível (e.g., um ícone de estrela).
•Implementação:
•Frontend (templates/lista_vagas.html, templates/dashboard_candidato.html): Adicionar um ícone clicável (e.g., estrela) ao lado de cada vaga. Usar JavaScript (via app.js) para enviar requisições assíncronas (AJAX) ao backend para adicionar/remover vagas dos favoritos.
•Backend (app.py - novas rotas para favoritar/desfavoritar vaga): Criar rotas para lidar com as requisições de favoritar/desfavoritar. Persistir essa relação em uma nova tabela CandidatoVagaFavorita no banco de dados.
•Banco de Dados (recrutamento.db): Criar a tabela CandidatoVagaFavorita com candidato_id e vaga_id.
•Visualização Completa da Vaga:
•Funcionalidade: Exibir as vagas em um formato completo e detalhado (título, empresa, salário, localização, tipo, data de criação, descrição completa) mesmo após a candidatura.
•Implementação:
•Frontend (templates/lista_vagas.html, templates/dashboard_candidato.html): Ajustar os templates para exibir todos os detalhes da vaga de forma consistente, independentemente do status da candidatura.

4.2. Dashboard da Empresa

Objetivo: Aprimorar as ferramentas de busca e gerenciamento de candidatos para a empresa.
Detalhes da Implementação:
•Busca Manual de Candidatos:
•Funcionalidade: Adicionar um botão na visualização da vaga que permita à empresa buscar candidatos manualmente no banco de dados do sistema.
•Implementação:
•Frontend (templates/detalhes_vaga.html): Adicionar um botão “Buscar Candidatos” que redirecione para a nova interface de busca de candidatos (ver Seção 3.2).
•Favoritar Candidatos:
•Funcionalidade: Permitir que a empresa favorite candidatos durante a busca manual.

•Implementação:
•Frontend (interface de busca de candidatos para empresas): Adicionar um ícone clicável para favoritar/desfavoritar candidatos. Usar JavaScript para enviar requisições assíncronas ao backend.
•Backend (app.py - novas rotas para favoritar/desfavoritar candidato): Criar rotas para lidar com essas requisições. Persistir a relação em uma nova tabela EmpresaCandidatoFavorito.
•Banco de Dados (recrutamento.db): Criar a tabela EmpresaCandidatoFavorito com empresa_id e candidato_id.
•Aba de Favoritos:
•Funcionalidade: Criar uma aba “Favoritados” na visualização da vaga, onde a empresa poderá ver todos os candidatos que foram previamente marcados como favoritos para aquela vaga específica.
•Implementação:
•Frontend (templates/detalhes_vaga.html): Adicionar uma nova aba ou seção na página de detalhes da vaga para listar os candidatos favoritos. Carregar esses dados via requisição ao backend.
•Backend (app.py - rota de detalhes da vaga): Modificar a rota para buscar os candidatos favoritos associados à vaga atual, utilizando a tabela EmpresaCandidatoFavorito.

5. Notificações e Comunicação

Esta seção detalha as melhorias no sistema de notificações e comunicação, tanto por e-mail quanto no painel interno do usuário.

5.1. Emails

Objetivo: Melhorar a qualidade e relevância dos e-mails transacionais enviados pelo sistema.
Detalhes da Implementação:
•Personalização Visual:
•Funcionalidade: Adicionar o logo da empresa no cabeçalho de todos os e-mails enviados. O logo deve ser responsivo e bem integrado ao design do e-mail.
•Implementação:
•Backend (app.py, ou um novo módulo de e-mail): Modificar as funções de envio de e-mail para incluir o logo. Isso pode envolver a utilização de templates HTML para e-mail. Verificar a biblioteca de envio de e-mails utilizada (e.g., Flask-Mail) para opções de template.
•Conteúdo Detalhado:
•Funcionalidade: Incluir nos e-mails de notificação de alteração de vaga (edição, mudança de status, reativação) os seguintes detalhes de forma clara e concisa: Nome da empresa, Título exato da vaga, Data e hora da modificação.
•Implementação:
•Backend (app.py, ou módulo de e-mail): Ao disparar e-mails de notificação, garantir que os dados da vaga e da empresa sejam passados para o template do e-mail e formatados adequadamente.
•Envio Seletivo:
•Funcionalidade: Garantir que as notificações por e-mail relacionadas a vagas sejam enviadas exclusivamente para os candidatos que já se candidataram àquela vaga específica.

•Implementação:
•Backend (app.py, ou módulo de e-mail): Antes de enviar uma notificação de e-mail sobre uma vaga, verificar no banco de dados quais candidatos se candidataram a essa vaga (tabela Candidatura) e enviar o e-mail apenas para eles.

5.2. Notificações Internas (Painel do Usuário)

Objetivo: Implementar um sistema de notificação robusto e intuitivo no painel de controle dos usuários (candidatos e empresas).
Detalhes da Implementação:
•Status de Leitura:
•Funcionalidade: Notificações não lidas devem ter o título em negrito. Ao clicar ou abrir uma notificação, ela deve ser automaticamente marcada como “lida”.
•Implementação:
•Frontend (templates/dashboard_candidato.html, templates/dashboard_empresa.html, app.js): Usar CSS para estilizar notificações não lidas. Implementar um evento de clique em JavaScript que envie uma requisição AJAX ao backend para atualizar o status da notificação.
•Backend (app.py, ou novo módulo de notificações): Criar uma rota para marcar notificações como lidas. Persistir o status de leitura em uma nova tabela Notificacao.
•Banco de Dados (recrutamento.db): Criar a tabela Notificacao com campos como id, usuario_id, tipo, conteudo, lida (booleano), data_criacao.
•Tipos de Notificação:
•Funcionalidade: Notificar sobre alterações em vagas (edições, mudanças de status, reativações). A notificação deve exibir o título da vaga e o nome da empresa.
•Implementação:
•Backend (app.py - rotas de edição/status de vaga): Ao realizar uma alteração em uma vaga, disparar a criação de uma nova notificação na tabela Notificacao para os usuários relevantes (candidatos que se candidataram, empresas).
•Modularização do Código:
•Funcionalidade: Refatorar o código, movendo a lógica de notificações do arquivo app.py para um módulo próprio (e.g., utils/notifications.py).
•Implementação:
•Estrutura de Arquivos: Criar utils/notifications.py e mover as funções relacionadas a notificações para lá. Importar e usar essas funções em app.py.
•Gerenciamento de Notificações:
•Funcionalidade: Adicionar uma funcionalidade que permita ao candidato limpar seu painel de notificações. Mensagens fixadas (como notificações de contratação) não devem ser excluídas e devem permanecer no topo.
•Implementação:
•Frontend (templates/dashboard_candidato.html): Adicionar um botão “Limpar Notificações”.
•Backend (app.py - nova rota para limpar notificações): Criar uma rota que, ao ser acionada, atualize o status de lida ou exclua notificações da tabela Notificacao, exceto aquelas marcadas como fixada.
•Banco de Dados (recrutamento.db): Adicionar uma coluna fixada (booleano) na tabela Notificacao.
•Interface do Painel de Notificações:

•Funcionalidade: Criar uma seção de notificações na dashboard do candidato com um design claro, funcional e intuitivo.
•Implementação:
•Frontend (templates/dashboard_candidato.html, static/style.css): Desenvolver a interface visual para exibir as notificações, garantindo responsividade e boa usabilidade.

6. Currículos e Transparência

Esta seção aborda a melhoria na extração de informações de currículos e a garantia de transparência no uso de dados do usuário.

6.1. Extração de Resumo de Currículo

Objetivo: Melhorar a extração de informações de currículos, garantindo que o resumo seja coletado corretamente e exibido.
Detalhes da Implementação:
•Extração de Resumo:
•Funcionalidade: Implementar uma funcionalidade para extrair e exibir um resumo automático do currículo do candidato.
•Implementação:
•Backend (utils/resume_extractor.py): Revisar e aprimorar a lógica de extração de texto do PDF e a identificação de seções chave para gerar um resumo conciso. Pode ser necessário integrar uma biblioteca de processamento de linguagem natural (NLP) para melhor qualidade do resumo.
•Banco de Dados (recrutamento.db): Adicionar uma coluna resumo_curriculo na tabela Candidato ou Curriculo para armazenar o resumo extraído.
•Frontend (templates/detalhes_candidato.html - para empresas, templates/dashboard_candidato.html - para o próprio candidato): Exibir o resumo extraído do currículo.

6.2. Transparência de Dados

Objetivo: Garantir que os usuários compreendam como seus dados são coletados e utilizados, promovendo a confiança.
Detalhes da Implementação:
•Informação Clara:
•Funcionalidade: Deixar claro para os usuários (empresas e candidatos) quais dados estão sendo coletados, como e por que estão sendo usados.
•Implementação:
•Frontend (templates/cadastro_candidato.html, templates/cadastro_empresa.html, templates/base.html - rodapé ou link para política de privacidade): Adicionar links para uma política de privacidade detalhada e/ou pop-ups informativos no momento do cadastro e em áreas relevantes do sistema.
•Mensagem Amigável no Cadastro:
•Funcionalidade: Adicionar uma mensagem amigável no momento do cadastro explicando o tratamento de dados, em conformidade com as políticas de privacidade e regulamentações (e.g., LGPD).
•Implementação:

•Frontend (templates/cadastro_candidato.html, templates/cadastro_empresa.html): Incluir um pequeno texto explicativo próximo aos formulários de cadastro, com um checkbox de consentimento.

7. Ajustes e Correções

Esta seção lista bugs e ajustes finos necessários para aprimorar a funcionalidade e a experiência do usuário.

7.1. Coleta de Informações do Currículo

Objetivo: Corrigir a extração de informações do currículo, garantindo que todos os dados relevantes sejam coletados corretamente.
Detalhes da Implementação:
•Correção da Extração:
•Funcionalidade: Corrigir a extração de informações do currículo, garantindo que o resumo e outros dados (e.g., experiência, formação, habilidades) sejam coletados corretamente.
•Implementação:
•Backend (utils/resume_extractor.py): Depurar e refatorar o código de extração para garantir a robustez e precisão na coleta de dados de diferentes formatos de currículo.

7.2. Botão “Cancelar” no Upload de Currículo

Objetivo: Garantir que o botão “Cancelar” no formulário de upload de currículo funcione conforme o esperado.
Detalhes da Implementação:
•Funcionalidade: Fazer com que o botão “Cancelar” no formulário de upload de currículo funcione corretamente, interrompendo o envio e fechando o formulário/modal.
•Implementação:
•Frontend (templates/upload_curriculo.html, app.js): Adicionar um evento de clique ao botão “Cancelar” que feche o modal de upload e/ou limpe o campo de seleção de arquivo. Se o upload já estiver em andamento, pode ser necessário implementar uma lógica para abortar a requisição.

7.3. Edição de Perfil do Candidato (Endereço)

Objetivo: Melhorar a usabilidade na edição do perfil do candidato, exibindo o endereço já cadastrado.
Detalhes da Implementação:
•Exibição do Endereço:
•Funcionalidade: Ajustar a página de edição de perfil do candidato para que o endereço já cadastrado seja exibido nos campos de formulário, permitindo que o usuário o edite facilmente.
•Implementação:
•Frontend (templates/editar_perfil_candidato.html): Ao carregar a página de edição de perfil, preencher os campos de endereço (rua, número, bairro, cidade, estado, CEP) com os dados atuais do candidato, recuperados do backend.
•Backend (app.py - rota de edição de perfil do candidato): Garantir que a rota de edição de perfil forneça os dados completos do endereço do candidato ao template.