# Desafio: Aplicação de Gestão de Contratos

# Descrição

Crie uma aplicação web que permita aos usuários fazer upload de contratos em formato PDF, ler o conteúdo para extrair uma prévia do assunto, cadastrar dados adicionais como data de assinatura e validade, e gerenciar esses contratos. A aplicação deve usar Node.js para o backend e Vue.js para o frontend.

# Requisitos

## **Funcionalidades**

- Upload de Contratos: O usuário pode fazer upload de arquivos PDF contendo contratos
- Leitura de PDF: A aplicação deve ler o conteúdo dos PDFs e fornecer uma prévia do assunto do contrato.
- Cadastro de Contratos: O usuário pode cadastrar informações adicionais sobre o contrato, como data de assinatura, data de validade, partes envolvidas, e um resumo do contrato.
- Listagem de Contratos: O usuário pode ver uma lista de todos os contratos cadastrados, com informações como título, data de assinatura, data de validade e status (ativo, expirado, etc.).
- **Detalhes do Contrato:** O usuário pode ver os detalhes completos de um contrato específico.
- Edição e Remoção de Contratos: O usuário pode editar as informações de um contrato ou removê-lo da lista.
- Notificações: A aplicação deve notificar o usuário sobre contratos que estão próximos da data de validade.

## **Tecnologias**

- Backend: Node.js com Express.
- Frontend: Vue.js com Vite.
- Banco de Dados: MySQL com Knex e Objection.js para ORM.
- Armazenamento de Arquivos: Armazenar os arquivos no sistema de arquivos do servidor, dica usar "Multer"
- Controle de Versão: Git (submeter o código em um repositório público ou privado no GitHub ou GitLab).

•

## Requisitos Técnicos

- API RESTful: Implementar uma API RESTful no backend para gerenciar os contratos.
- Upload de Arquivos: Usar bibliotecas como multer para gerenciar uploads no backend.
- Leitura de PDFs: Usar bibliotecas como pdf-parse ou pdf-lib para extrair texto dos PDFs.
- Persistência de Dados: Usar MySQL com Knex e Objection.js para armazenar e gerenciar informações sobre os contratos (nome, caminho, data de assinatura, data de validade, partes envolvidas, etc.).
- Interface de Usuário: Criar uma interface de usuário em Vue.js para interagir com a API.
- **Documentação:** Incluir um arquivo README.md com instruções sobre como configurar e executar a aplicação.

## **Extras (Opcional)**

- **Autenticação:** Adicionar um sistema de autenticação para que os usuários tenham suas próprias listas de contratos.
- **Design Responsivo:** Garantir que a interface seja responsiva e funcione bem em dispositivos móveis.
- Compartilhamento de Contratos: Permitir que os usuários compartilhem contratos com outros usuários.
- Deploy: Fazer o deploy da aplicação em uma plataforma como Heroku, Vercel ou Netlify.
- **Integração com Calendário:** Integrar com APIs de calendário (Google Calendar, por exemplo) para adicionar eventos de vencimento de contratos.

#### Avaliação

A avaliação do desafio pode considerar os seguintes critérios:

- Funcionalidade: A aplicação atende a todos os requisitos funcionais?
- Qualidade do Código: O código é limpo, bem organizado e segue boas práticas?
- Interface de Usuário: A interface é intuitiva e bem projetada?
- Documentação: As instruções para configurar e executar a aplicação são claras e completas?
- **Segurança**: A aplicação implementa boas práticas de segurança, especialmente para o upload e armazenamento de arquivos?
- **Criatividade:** Foram implementadas funcionalidades extras ou melhorias além dos requisitos básicos?

# **Entrega**

Os candidatos devem enviar um link para o repositório do código-fonte (por exemplo, no GitHub).

A entrega deve ser feita até o dia **16/7 às 12 horas** para o email ti@star.psi.br com o assunto **DESAFIO DEV 2024**. Que deve conter o link para acesso ao repositorio com o projeto.

# **Dicas para os Candidatos**

- Divida o trabalho em pequenas tarefas e faça commits frequentes.
- Documente qualquer decisão importante que tomou durante o desenvolvimento.
- Se encontrar algum problema, explique como tentou resolvê-lo.
- Priorize a funcionalidade básica e depois adicione funcionalidades extras, se tiver tempo.

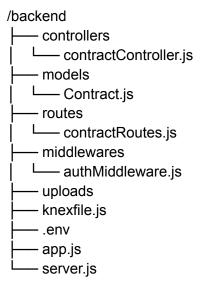
# Estrutura do Projeto

## Backend (Node.js)

#### Configuração Inicial:

Inicialize um projeto Node.js com npm init. Instale as dependências: express, knex, objection, mysql, multer, pdf-parse, jsonwebtoken, bcrypt (para autenticação, se necessário).

#### **Estrutura de Pastas:**



#### Roteamento e Controladores:

Implemente as rotas e controladores para upload de arquivos, leitura de PDFs, CRUD de contratos, e notificações.

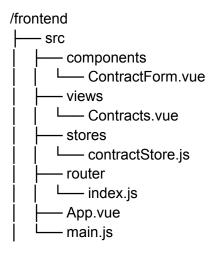
# Frontend (Vue.js)

## Configuração Inicial:

```
Crie um novo projeto Vue.js com Vite:
```

```
npm init @vitejs/app nome-do-projeto --template vue Instale as dependências: axios, vue-router, pinia.
```

#### Estrutura de Pastas:



## **Componentes e Vistas:**

Crie componentes para o formulário de contrato, lista de contratos, e detalhes do contrato. Configure as rotas para navegação entre as páginas de upload, listagem, e detalhes.

#### Gerenciamento de Estado:

Use Pinia para gerenciar o estado dos contratos e notificações.