**BACK-END**

O ambiente backend é responsável por armazenar os dados de todas as empresas, usuários, vagas, currículos e por disponibilizar esses dados para a interface. A aplicação do backend é uma API utilizando o framework Django codificado na linguagem de programação Python. A aplicação também implementa o framework REST, encapsulado como uma biblioteca otimizada para ser utilizada juntamente com o framework do Django. A aplicação faz uso de um banco de dados relacional MySQL. para facilitar o processo e reduzir os recursos necessários durante o desenvolvimento, utilizamos uma variação independente do MySQL, o sqlite, o qual permite persistir os dados em um banco de dados relacional não precisando hospedar ou se conectar a um servidor.

A API disponibiliza 19 endpoints com 15 deles possuindo operações CRUD completas, para facilitar o desenvolvimento o backend possui uma documentação completa de todas as APIs com uma interface gráfica utilizando o Swagger, o qual é um conjuntos de ferramentas de API na qual a especificação OpenAPI é baseada.

Além da interface gráfica, também implementamos uma coleção completa de endpoints hospedada no Postman, uma plataforma de API para desenvolvedores. Permitindo realizar chamadas para os endpoints do backend e avaliar as respostas, os parâmetros e outros pontos necessários para o desenvolvimento de uma API. A aplicação possui um método de autenticação do tipo Bearer Token, o qual é um tipo de token de acesso em uma API utilizado para autenticação e autorização. Os tokens Bearer são gerados com base em protocolos e especificações como OAuth e JWT (JSON Web Token).

Toda requisição para o ambiente de backend e seus endpoints devem possuir um token, caso não seja informada, o acesso não será autorizado. Para gerar um token, o usuário deve possuir uma conta de administrador no ambiente de backend e solicitar a geração de um token de acesso a partir de um endpoint único, o qual deve se autenticar utilizando usuário e senha.

Toda a camada de endpoint, autenticação e resposta é tratada pelo próprio framework do Django com a ajuda do framework do rest. A arquitetura do ambiente de backend foi modularizada e seccionada cuidadosamente para criar uma arquitetura que forneça uma melhor manutenibilidade e escalonamento.

Texto

Descrição gerada automaticamente com confiança média

A pasta api possui os arquivos responsáveis pela informação geral da API. A pasta api\_admins possui os arquivos responsáveis pelo armazenamento e gerenciamento de usuários administrativos, os quais poderão gerar seus tokens para realizar acessos remotos via endpoints. A pasta company é responsável pela lógica, armazenamento e gerenciamento de todos os dados (conta e perfil) das empresas cadastradas no site.

Assim como a company, a pasta users é responsável pela lógica, armazenamento e gerenciamento de todos os dados (conta e perfil) dos usuários cadastrados no site. A pasta vacancy, assim como o nome define, é responsável pelo tratamento de todas as vagas criadas. Por fim, a pasta resumes é responsável pela lógica, armazenamento e gerenciamento de todos os currículos gerados pelos usuários, os quais serão utilizados para se candidatar nas vagas, as quais por sua vez, serão publicadas pelas empresas.

A estrutura de urls e endereços da API são definidas em arquivos específicos dentro do framework do Django, nomeados de urls.py. Nesses arquivos, são definidos os endereços os quais serão acessados pelos usuário através dos endpoints, essas definições irão vincular a parte lógica da API (também conhecidas como views) e a área externa da aplicação (os endpoints).

As views, por sua vez, possuem uma abordagem orientada a objetos, também conhecida como class-based views, sendo ela uma das abordagens disponibilizada pelo framework do Django. A criação de uma API consiste em duas classes, um schema, o qual será responsável por produzir as informações necessárias para gerarmos a documentação do Swagger, e uma classe onde será responsável pelo endpoint.

A classe responsável pelo endpoint deverá conter os métodos das requisições que deverão ser suportadas pelo endpoint, ou seja, se o endpoint permite uma requisição no método GET, uma função get deve existir dentro da classe pelo fato de que ela será executada ao receber uma requisição desse método.

Essa função também deverá retornar uma resposta que possa ser serializada e enviada via HTTP, permitindo uma comunicação entre APIs sem a renderização de uma página. Essa resposta deverá receber um dicionário que irá conter os dados que deverão ser retornados na resposta, assim como um código de status.

Outro modelo de resposta que também é muito utilizado é o retorno de objetos do banco de dados, porém, diferente de um texto comum, um objeto que seria uma classe na linguagem Python não pode ser serializado automaticamente pelo framework do rest. Para permitir a serialização de um objeto, é necessário definir um serializador que realizará essa conversão de um objeto para algo que possa ser emitido via protocolos HTTP.

Texto

Descrição gerada automaticamente

O exemplo mostrado na imagem **<insert image number here>** é o serializador de objetos de vagas, ele é responsável por converter uma entrada do banco de dados em um objeto que possa ser serializado e transmitido para o requisitante via protocolo HTTP. Em resumo, o serializador possui uma meta classe que irá definir duas principais variáveis, a variável model será responsável por definir de qual tabela os dados deverão ser buscados, já a variável fields informará quais informações (colunas) devem ser selecionadas e serializadas.

Além dessas duas variáveis, o exemplo exibe também algumas variáveis computadas, essas variáveis são únicas e geradas durante a serialização, geralmente utilizadas para obter informações de outras tabelas que estão vinculadas por chaves estrangeiras.

No framework Django, o banco de dados é tratado completamente pelo próprio framework, sendo necessária a criação de um modelo. O modelo é uma classe a qual possui uma estrutura específica que pode ser interpretada pelo framework e manipulada como tabelas em um banco de dados relacional.

Texto

Descrição gerada automaticamente

O exemplo mostrado na imagem **<insert image number here>**, é o modelo da tabela das vagas criadas por uma empresa, o modelo herda de uma classe específica do framework que permitirá que o Django interprete suas variáveis, métodos e metaclasses e gere comandos SQL a partir dessas interpretações. As variáveis declaradas na classe são as variáveis que irão definir as colunas das tabelas, os seus valores são os tipos / atributos nos quais os valores dessa coluna serão tratados. Assim como a herança da classe, os valores das variáveis são classes específicas do framework, os quais permitirão a interpretação e manipulação conforme necessário.

Devido a responsabilidade de manipulação do banco de dados ser completamente do framework, o Django disponibiliza nativamente uma interface de administração para que os dados do banco possam ser facilmente acessados e modificados sem a necessidade uma conexão manual com o banco. O registro de um modelo para exibição na página de administração do framework é realizado por um arquivo específico de sua estrutura, nele são criadas classes que irão fornecer informações necessárias para a renderização das páginas.