**Autor: Newton Augusto Reinoldes Miotto**

**Profissão: Desenvolvedor Backend Python/Django**

**DOCUMENTO DE DETALHE DA CODIFICAÇÃO**

Este documento tem o objetivo de explicar de forma sucinta o desenvolvimento do exercício proposto. Além de demonstrar possíveis problemas e como mitigá-los.

**DESENVOLVIMENTO:**

Por conta da rotina corrida da semana, foi desenvolvido de forma simples, sem adicionar muita complexidade.

No entanto, criei todos os módulos de forma organizada. Desenvolvi mais endpoints do que o proposto e desenvolvi testes que não foram mencionados no exercício.

Não fiz telas, portanto todo o código deve ser testado por meios de APIs.

**RESILIÊNCIA:**

Para controlar erros inesperados todo o código deve ser iniciado por um bom caso de teste. Apesar de não cobrir todas as possibilidades é de extrema importância cobrir boa parte dos problemas que são esperados.

Além disso, todas as respostas devem ser tratadas de forma consistente pelos serializers, atentando aos campos obrigatórios e opcionais e as possíveis tentativas de cadastro incorretas ou maliciosas.

Sempre que mexemos nos models do Django deve ser gerado um Migration. E esse Migration deve ser aplicado no banco. Em alguns projetos isso é feito de forma automática pelo próprio pipeline.

É obrigação do teste prever possíveis entradas de valores inválidos e sempre antes de subir uma release ou correção de bug, deve-se rodar os testes. Para exemplificar, adicionei alguns casos de testes simples, mas funcionais.

Comando:

Python manage.py test

**PERFORMANCE:**

No banco de dados, deve-se evitar registrar informações desnecessárias. Além de tomar cuidado com as queries. Na maioria das vezes, um software é lento por causa de código ruim. Muitos cruzamentos de tabelas e tratamentos que poderiam ser simplificados. Também pelo uso de ferramentas inapropriadas. Como utilizar um banco SQL quando se poderia utilizar um NoSQL em alguns casos.

Além disso, atualmente com o crescente uso dos microsserviços deve-se tomar cuidado com a quantidade de conexões simultâneas e provisionamento do banco.

O ideal é que se mantenha um ambiente em nuvem que tenha a possibilidade de escalar em momentos de pico, e talvez utilizar proxy para reduzir as conexões.

**SEGURANÇA:**

É primordial que a API do sistema seja segura. Para garantir a segurança, deve-se seguir as boas práticas. Como transacionar as informações criptografadas, utilizando protocolos como SSL/TLS e HTTPS.

Manter um mecanismo de autenticação e autorização, controlando os acessos e as permissões, através de tokens de segurança.

Também é importante monitorar o tráfego e comportamentos maliciosos, como: ataques, vazamento de dados, disponibilidade e desempenho da API.

Os endpoints devem ter certo tipo de isolamento para que não seja possível acessar recursos que não podem ser liberados para aquele agente. Isso pode ser feito pelo Backend através de controle do tráfego de saída de informações, testes de segurança e de integração.

E sempre ter disponível profissionais experientes de segurança, cloud, devops para mitigar possíveis problemas.

**SIMULTANEIDADE:**

Atualmente podemos contar com os Banco de dados poderosos, Microsserviços, Auto Scaling vertical e horizontal graças a computação em Cloud. É possível configurar na AWS por exemplo a multiplicação das máquinas em um caso de pico de requisições. Além de contar com Lambdas que fornecem computação descentralizada, podendo trabalhar simultaneamente com milhões de requisições.

E no universo da programação é possível trabalhar com treads e processos para diminuir a concorrência entre os serviços.

Obs.: Nos arquivos do repositório, incluí um backup do banco de dados, as requisições no postman, prints das requisições e um arquivo contendo as bibliotecas necessárias para rodar. (requirements.txt)