Desenvolvimento de uma API em Python para consulta de CNPJ, voltada para empresas de médio porte

Nattan Mendes Tinonin*
nattan078@gmail.com
Instituto Federal Catarinense - Câmpus Blumenau
Blumenau, Santa Catarina

ABSTRACT

O presente artigo tem o objetivo de apresentar a forma de construção, impasses, resultados e conceitos envolvidos na construção de uma API em python para uso comercial de média escala.

CCS CONCEPTS

• **Applied computing** → **Event-driven architectures**.

KEYWORDS

Computação, API, Consulta em tempo real, SQL

1 INTRODUÇÃO

No atual cenário empresarial, a disponibilidade de informações precisas e atualizadas sobre empresas é essencial para tomadas de decisões estratégicas. Nesse contexto, a utilização de *APIs* tem se tornado uma prática comum para acessar e compartilhar dados de forma eficiente e segura.

Este artigo apresenta o desenvolvimento de uma *API* de consulta de CNPJ (Cadastro Nacional da Pessoa Jurídica) e dados empresariais, utilizando a linguagem de programação *Python* em conjunto com o *framework Flask*. O objetivo é fornecer uma solução viável e acessível para empresas de média escala, permitindo o acesso rápido e simplificado às informações cadastrais das empresas.

Ao longo do artigo, serão abordados diversos aspectos relevantes relacionados às *APIs*, como a definição e conceitos fundamentais, a aplicação desenvolvida, as principais regras de negócio do projeto, as tecnologias, um pequeno comparativo entre duas ferramentas, testes de estresse e os benefícios que uma empresa pode obter ao utilizar essa solução.

Além disso uma sessão inteira é destinada a apresentar a forma que o banco de dados foi populado e as tecnologias que foram usadas para acelerar esse processo.

Por fim, o artigo apresentará as conclusões alcançadas durante o desenvolvimento da *API*, resumindo os principais pontos abordados e destacando a relevância desse tipo de solução para empresas de média escala. As referências utilizadas no trabalho também serão fornecidas para possibilitar a consulta de fontes confiáveis e aprofundamento nos temas discutidos.

Com essa abordagem estruturada, espera-se fornecer aos leitores uma visão abrangente sobre o desenvolvimento de *APIs*, o uso do *Python* e *Flask* para construir uma *API* de consulta de CNPJ e os benefícios e vantagens que essa solução pode trazer para as empresas.

2 API

Em suma, *API* [2] (*Application Programming Interface*) é um conjunto de regras e protocolos que permite a interação e comunicação entre diferentes elementos de software. Define métodos e formatos sobre os dados utilizados na solicitação e fornecimento de informações entre diferentes sistemas, aplicativos ou serviços.

Muito utilizadas em uma variedade de contextos, desde integrações entre sistemas até disponibilização de serviços para desenvolvedores. Possuem papel fundamental na construção de ecossistemas virtuais interconectados, facilitando a criação de aplicações robustas, flexíveis e interoperáveis [1].

2.1 Conceitos fundamentais

Uma *API* atua como uma interface [7], permitindo que os desenvolvedores acessem os recursos e serviços de um sistema de forma padronizada. Ela define os métodos e formatos de dados que podem ser utilizados na troca de informações.

A integração é uma das principais propriedades das *APIs*. Elas permitem a conexão e colaboração entre sistemas, aplicativos e serviços distintos. Ao fornecer uma forma padronizada de comunicação, as *APIs* facilitam a interoperabilidade e a troca de dados entre essas entidades.

Uma *API* também oferece uma camada de abstração, ocultando os detalhes internos de implementação dos sistemas subjacentes. Isso permite que os desenvolvedores utilizem os recursos fornecidos pela *API* sem precisar conhecer todos os detalhes de como eles são implementados, simplificando o processo de desenvolvimento.

Padronização é um aspecto fundamental [7] das *APIs*. Elas seguem convenções e especificações que garantem a consistência e interoperabilidade entre diferentes sistemas. Essa padronização facilita a comunicação e a colaboração entre aplicativos e serviços desenvolvidos por equipes diferentes, permitindo a reutilização de componentes existentes.

Segurança também é uma consideração importante nas *APIs*. Mecanismos de autenticação e autorização podem ser implementados para garantir que apenas usuários autorizados acessem os recursos da *API*. Além disso, a criptografia pode ser aplicada para proteger a confidencialidade dos dados transmitidos através da *API*.

Por fim, as *APIs* são projetadas para serem escaláveis, permitindo que um grande número de clientes acesse os serviços fornecidos por elas simultaneamente, sem prejudicar o desempenho ou a disponibilidade do sistema.

Esses conceitos fundamentais das *APIs* fornecem a base para entender sua importância e aplicabilidade em diferentes contextos de desenvolvimento de software.

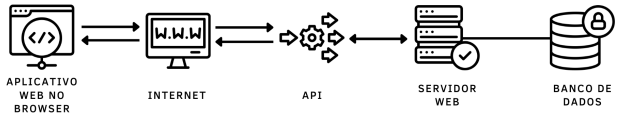

Figura 1: Fluxograma geral de uma *API*

2.2 Tecnologias usadas

As tecnologias usadas no projeto visam aplicar da maneira mais simples possível a ideia da aplicação. Por esse motivo elas são amplamente utilizadas e conhecidas no mercado. Essa seção apresentará primeiramente a linguagem, o SGBD, e logo após as bibliotecas.

2.2.1 Python. A escolha da linguagem *Python* [8] para o desenvolvimento da *API* se deve à sua popularidade e flexibilidade. *Python* é amplamente adotado na comunidade de desenvolvimento de software, oferecendo uma vasta gama de bibliotecas e *frameworks* que facilitam a criação de soluções robustas e escaláveis.

2.2.2 MySQL. O *MySQL* [5] é um sistema de gerenciamento de banco de dados relacional (*RDBMS* ou SGBD) de código aberto, amplamente utilizado em todo o mundo. Ele fornece uma plataforma confiável e escalável para armazenar, gerenciar e recuperar dados de forma eficiente. Oferece suporte a uma ampla gama de tecnologias, incluindo replicação para alta disponibilidade e distribuição geográfica, clusterização para escalabilidade e balanceamento de carga, particionamento para lidar com grandes volumes de dados, e recursos avançados de segurança para proteger a integridade dos dados. Com sua linguagem de consulta padrão *SQL*, o *MySQL* é compatível com muitas aplicações e oferece uma variedade de recursos, como gatilhos, procedimentos armazenados e visões, permitindo uma manipulação poderosa dos dados. Com sua combinação de desempenho, confiabilidade e flexibilidade, o *MySQL* continua sendo uma escolha popular para desenvolvedores e empresas que buscam soluções de banco de dados eficientes e de alto desempenho.

2.2.3 Flask. O *framework* [6] oferece uma variedade de recursos e tecnologias para o desenvolvimento de aplicativos *web* em *Python*. Com seu sistema de roteamento flexível, é possível criar *URLs* amigáveis e mapear diferentes tipos de solicitações *HTTP*. Além disso, suporta a renderização de *templates*, permitindo a criação de páginas *web* dinâmicas. O *framework* também integra-se facilmente a diversos sistemas de gerenciamento de bancos de dados, possibilitando a persistência eficiente de dados. Além disso, o *Flask* suporta extensões que fornecem funcionalidades adicionais, como autenticação, armazenamento em *cache* e integração com *APIs* externas. Ferramentas que futuramente podem ser muito úteis para a continuidade do projeto dentro das empresas.

2.2.4 MySQL Connector. O *MySQL Connector* [5] é uma biblioteca essencial para desenvolvedores que desejam trabalhar com bancos de dados *MySQL* em seus aplicativos. Essa tecnologia fornece uma camada de conexão entre o *Python* e o *MySQL*, permitindo que os desenvolvedores executem consultas e manipulem dados de forma eficiente. O *MySQL Connector* oferece uma ampla gama de recursos, incluindo suporte para transações, consultas parametrizadas, recuperação e gravação de dados em diferentes formatos, além de suporte a recursos avançados do *MySQL*, como *stored procedures* e *triggers*. Com uma sintaxe simples e intuitiva, é fácil realizar operações de *CRUD* (criação, leitura, atualização e exclusão) no banco de dados *MySQL* usando o *MySQL Connector*. Além disso, essa biblioteca é altamente otimizada, garantindo um desempenho rápido e eficiente em aplicações que manipulam grandes volumes de dados. A documentação abrangente e a comunidade ativa em torno do *MySQL Connector* tornam mais fácil para os desenvolvedores encontrar suporte e exemplos de uso. Em resumo, o *MySQL Connector* é uma ferramenta essencial para a integração de aplicativos *Python* com bancos de dados *MySQL*, fornecendo recursos poderosos e facilitando a manipulação eficiente de dados.

Essas foram as tecnologias utilizadas no desenvolvimento da *API*, existem outras ferramentas disponíveis com ideias tão eficientes quanto. Podendo ser aplicadas de acordo com o tamanho da aplicação. No contexto do atual artigo, essas tecnologias são mais do que o suficiente para empresas de pequeno e médio porte.

2.3 ORM

O leitor pode estar pensando nesse exato momento: Mas por que não foi utilizado *SQLAlchemy* (ou qualquer outro *ORM - Object-Relational Mapping*) para a realização do projeto? Foram três os motivos da não escolha dessa abordagem dentro do projeto aqui apresentado.

2.3.1 Dependências. Se tratando de uma aplicação desenvolvida visando ser utilizada em meios empresariais, é importante que os códigos fonte tenham o mínimo de dependência possível com outros fontes que não sejam de controle da empresa. Isso é um aspecto importante para a segurança e confiabilidade das aplicações. Uma *ORM* precisa de várias dependências para que seu funcionamento ocorra, desde os conectores até bibliotecas de otimização de valores.

2.3.2 Erros nos dados. Os dados disponibilizados pelo Governo Federal não foram adequados de forma correta para o modelo relacional de banco de dados, tendo que ser feitas várias alterações nos fontes de importação conforme será discutido no capítulo 5. Dessa forma, a aplicação não pode ser feita utilizando *frameworks* com *ORM*, podendo essa ser uma futura implementação em outra versão da aplicação.

2.3.3 Otimização. O principal motivo para o não uso de nenhuma *ORM* no projeto foi a otimização do tempo de acesso aos dados, antes de uma implementação total da *API* utilizando qualquer *ORM*, foram feitos testes comparativos de tempo de consultas na máquina local do servidor.

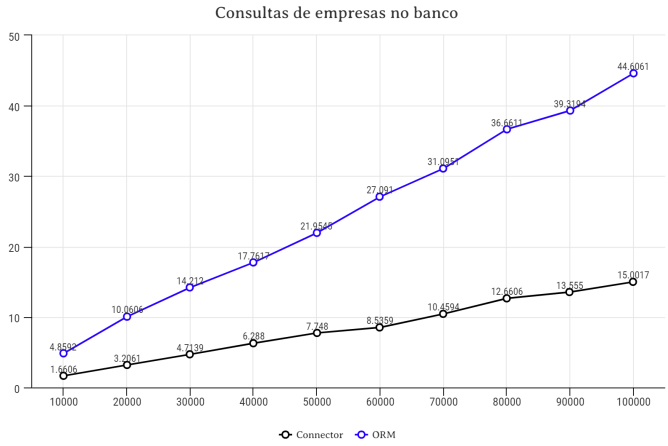

Figura 2: Comparativo temporal entre os tempos de consulta

Na esquerda do gráfico está o tempo (segundos) e abaixo a quantidade de consultas realizadas. A proporção de velocidade entre ambos os métodos de consulta é cerca de 32,35% conforme abaixo. As variáveis ΔC, ΔO, Ψ e N referem-se a variação de tempo do *Connector*, variação de tempo da *ORM*, proporção entre ambos e a quantidade de seleções respectivamente.

$$\Delta C = \frac{1}{N} \times \sum_{Connector} \tag{1}$$

$$\Delta O = \frac{1}{N} \times \sum_{ORM} \tag{2}$$

Portanto, a fórmula da proporção seria:

$$\Psi = \frac{\Delta C}{\Delta O} \tag{3}$$

Dessa forma, aplicando os valores da tabela acima na fórmula Ψ, obtemos o valor de 32,35%.

Com os dados apresentados, é evidente a melhor performance do *Connector* em relação ao *SQLAlchemy*, entretanto, possíveis falhas de segurança poderão ocorrer, ficando a cargo do programador do sistema proteções contra injeções *SQL*, por exemplo.

3 REGRAS DE NEGÓCIO

A regra de negócio foi desenvolvida de forma a tornar o processo o mais simples e intuitivo possível, exigindo um conhecimento mínimo por parte do usuário. Tendo como funcionamento base os seguintes pontos:

(1) O usuário insere a *URL* da intranet seguida de uma barra invertida ("\") e, em seguida, digita o CNPJ (Cadastro Nacional de Pessoa Jurídica) da empresa desejada, sem a utilização de acentos.
(2) Com base nas informações fornecidas, o sistema realizará uma busca no banco de dados para encontrar os dados correspondentes à empresa solicitada.
(3) Os dados encontrados serão retornados em formato *JSON*, contendo todas as informações disponíveis no banco de dados. Caso algum dado não seja encontrado, ele será retornado como "*null*" no *JSON*.

O fluxo do sistema segue os seguintes pontos:

(1) A *API* inicia o processo coletando a empresa a ser consultada no banco de dados e popula o objeto correspondente com os dados já coletados.
(2) Em seguida, a *API* realiza buscas adicionais para obter informações sobre a qualificação, natureza, sócios e estabelecimentos relacionados à empresa consultada.

4 APLICAÇÃO DESENVOLVIDA

O projeto é composto de três principais partes: *Models*, *Helpers* e a *Main*. Nos *Models* estão localizados todas as entidades do projeto, indo desde entidades fiscais até a classe de conexão com o banco de dados. Nos *Helpers* está localizada a *Main-Helper*, que, como o nome indica, é uma classe ajudante da *Main*, nela estão localizados todos os métodos que realizam a consulta dentro do banco, populando os dados dentro de objetos e retornando um objeto Empresa para a *Main*, que logo em seguida pede para o *helper* configurar a response e logo em seguida retornar para o usuário. Todo o projeto foi feito utilizando da Programação Orientada a Objetos oferecida pelo *Python*, encapsulando métodos e reutilizando códigos repetidos.

Na figura 2 está o código da classe principal da aplicação, sendo esse o ponto chave que liga as outras classes.

```
from flask import Flask
from Helpers.MainHelper import MainHelper
from Model.Connection import Connection

app = Flask(__name__)
connection = Connection('host',
                        'database_name',
                        'user',
                        'password')
connection.connect()
helper = MainHelper(connection)

@app.route('/<path:text>', methods=['GET'])
def read_url_data(text):

    response = helper.verificaCnpj(text)
    if response is not None:
        return response

    empresa = helper.carregaEmpresa(text)
    response = helper.retorna(empresa)
    return response

if __name__ == '__main__':
    app.run(host='0.0.0.0', debug=False)
```

Figura 2: Classe principal da aplicação.

Os diagramas de classes, relacionais e *links* para *download* da aplicação estão localizados no *GitHub* *.

5 BANCO DE DADOS

O banco de dados desempenha um papel fundamental no projeto, e os dados utilizados foram coletados no portal de dados abertos do governo federal brasileiro. Esses dados estão disponíveis em vários arquivos formatados em *CSV* (*comma-separated values*). É importante ressaltar que a atualização desses dados ocorre em períodos não especificados ao longo do ano, sendo responsabilidade dos desenvolvedores manter o banco de dados atualizado. No total, o volume de dados é de aproximadamente 20GB bruto e cerca de 80GB dentro do SGBD, abrangendo desde informações sobre a qualificação dos sócios das empresas até os estabelecimentos correspondentes. A leitura dos arquivos para a memória foi feita utilizando a biblioteca *Pandas* [3], uma poderosa ferramenta de análise de dados em *Python*, oferecendo estruturas de dados flexíveis e eficientes para manipulação e processamento de dados tabulares. Para realizar a população do banco, foi utilizado um *script* devido a vários erros encontrados nos dados baixados do portal federal, exigindo correções e adequações às regras estabelecidas no modelo desenvolvido. O código-fonte do *script* foi dividido em três principais partes: *Connection*, *Commands* e *Importer*. A classe *Connection* tem como objetivo estabelecer a conexão com o banco de dados e manter a sessão ativa. A classe *Commands* abriga todos os arquivos de importação e correção para cada tipo de tabela, sendo a parte principal do *script*. Já a classe *Importer* é responsável por instanciar uma conexão com a classe *Connection* e chamar os métodos da classe *Commands* para realizar a população do banco de dados. Um dos principais empecilhos na realização do projeto foi o tempo para o preenchimento do banco de dados, sendo necessárias algumas técnicas para acelerar esse processo, reduzindo assim o tempo de inoperabilidade da *API* e do servidor onde ela está localizada

5.1 Otimização da importação

De acordo com o manual do *MySQL* [4] as proporções temporais de cada processo são as seguintes:

- `Conectar`: (3)
- `Enviar query ao servidor`: (2)
- `Analisar query`: (2)
- `Inserir linhas`: (1 * tamanho da linha)
- `Inserir índices`: (1 * número de índices)
- `Fechar conexão`: (1)

De modo a melhorar o processo de inserção dos dados no banco de dados esse processo foi otimizado da seguinte maneira: O tempo de conexão não precisaria ser contado em todas as inserções se o acesso ao banco continuasse aberto enquanto a *API* estivesse ativa, sendo assim é desnecessário a reconexão em cada *query*, o mesmo vale para a desconexão com o banco. O envio, análise, inserção de linhas e índices sempre serão necessários para uma inserção correta dentro do banco, portanto, sempre serão contabilizados na equação. Além da melhora nos tempos fixos serão utilizados *inserts* de várias linhas ao mesmo tempo, aumentando drasticamente a produtividade.

A equação abaixo demonstrará a diferença de tempo entre o pior tempo possível de inserção e a inserção otimizada. O tamanho da linha, o número de índices, a quantidade de linhas, uma linha por vez, várias linhas por vez e ganho de tempo serão representados por X, Y, N, ψ, φ e Δ respectivamente.

$$\psi = (3 + 2 + 2 + 1 \times X + 1) \times N \tag{4}$$

$$\phi = 3 + 2 + 2 + 1 \times X + 1 \tag{5}$$

$$\Delta = \frac{\psi - \phi}{\psi} \tag{6}$$

Assumindo que o valor de (3 + 2 + 2 + 1 ×X + 1) pode ser fixo para cada tabela a ser populada, o assumiremos daqui para frente como Ω para facilitar o calculo da proporção.

*Link para o GitHub: https://github.com/NaTTaNMendes/API_CNPJ

$$\Delta = \frac{\Omega \times N - \Omega}{\Omega \times N} \tag{7}$$

Dessa forma o ganho de tempo equivale a:

$$\Delta = \frac{N - 1}{N} \tag{8}$$

Considerando a fórmula Δ, se o N for grande, o termo "-1" se torna negligível em comparação à N, com Δ se aproximando de 1. Dessa forma a notação Big O seria de complexidade O(1), ou seja, de complexidade constante independente do tamanho dos dados. Portanto, a lentidão dependeria da quantidade de memória e velocidade do computador do usuário para processar e armazenar em memória volátil os dados antes de fazer a inserção no banco, que seria quase instantânea.

6 TESTES DE ESTRESSE

Neste capítulo, abordaremos a avaliação de desempenho da *API* por meio de um teste de estresse. O teste de estresse de *API* tem como objetivo verificar a capacidade da *API* de lidar com uma carga intensa de solicitações simultâneas, garantindo sua estabilidade e capacidade de resposta.

Os testes foram conduzidos utilizando um *script* em *Python* para realizar as requisições à *API*. Para estabelecer a conexão, foi utilizada a biblioteca *requests*, amplamente reconhecida e utilizada na comunidade *Python* para comunicação via *HTTP*. Essa biblioteca oferece uma interface simplificada e robusta, permitindo a realização de requisições de forma eficiente e confiável. Dessa forma, garantimos uma abordagem confiável e padronizada para avaliar o desempenho da sua *API* durante o teste de estresse.

No estudo de caso em questão, cada sessão do teste durou um minuto, e os resultados obtidos foram os seguintes:

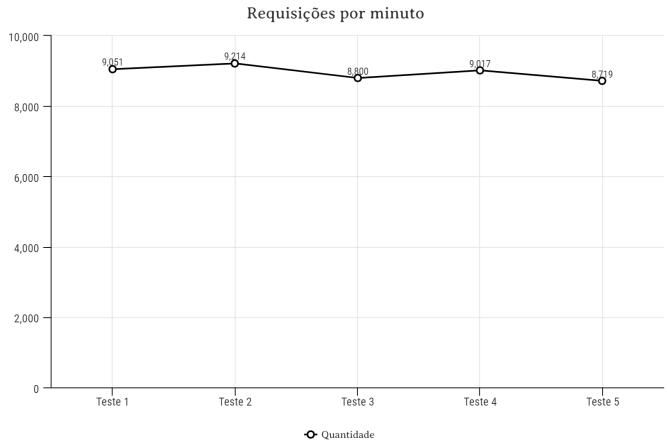

Figura 3: Teste de estresse

Esses valores representam a quantidade de solicitações processadas pela *API* em cada sessão de um minuto.

O modelo da máquina que realizou os testes é 82MG0009BR e possui as seguintes configurações gerais:

- `Intel Core i5-11300H @ 3.10GHz 3.11GHz`
- `8GB RAM DDR4`
- `Windows 11 22H2 Home 64 bits`
- `GTX 1650 4GB Dedicada`
- `512GB SSD M2`

Para calcular a média de requisições por segundo de resposta da *API* durante o teste de estresse, somamos a quantidade de respostas de todas as sessões e dividimos pelo tempo total em segundos que levaram os testes. No nosso caso como cada teste durava um minuto, basta multiplicar a quantidade de minutos por 60.

$$Media = \frac{\sum_{Testes}}{Minutos \times 60} \tag{9}$$

Fazendo o cálculo com os valores obtidos nos teste obtemos o tempo médio de 149,74 respostas por minuto. Dessa forma, concluímos que nessas configurações a *API* consegue realizar uma resposta a cada 0,1497 segundos.

7 BENEFÍCIOS PARA AS EMPRESAS

Este capítulo destaca os benefícios da adoção de uma *API* de consulta de CNPJ (Cadastro Nacional da Pessoa Jurídica) para empresas brasileiras de médio e pequeno porte. Exploraremos como essa *API* pode fornecer acesso a informações confiáveis, agilizar processos, auxiliar na análise de mercado, combater fraudes e promover a integração e escalabilidade dos negócios.

7.0.1 Informações confiáveis. A *API* de consulta de CNPJ permite que empresas obtenham informações atualizadas e confiáveis sobre outras empresas. Esses dados incluem informações legais, endereços, quadro societário e outros detalhes relevantes. Com acesso a essas informações precisas, as empresas podem tomar decisões informadas sobre fornecedores, parceiros comerciais e clientes, reduzindo riscos e garantindo relacionamentos comerciais mais seguros.

7.0.2 Agilidade e automação de processos. A adoção da *API* de consulta de CNPJ oferece às empresas a capacidade de automatizar processos de verificação de informações cadastrais. Em vez de realizar pesquisas manuais demoradas em diversas fontes de dados, a *API* permite acesso rápido e eficiente às informações necessárias. Essa automatização economiza tempo e recursos, permitindo que as equipes se concentrem em atividades mais estratégicas e de maior valor agregado. Um exemplo seria a economia em *captchas* gastos no acesso ao portal federal sobre a consulta de empresas.

7.0.3 Análise de mercado. Para empresas de médio e pequeno porte, compreender o mercado e a concorrência é fundamental para o sucesso dos negócios. A *API* de consulta de

CNPJ pode fornecer *insights* valiosos sobre o cenário empresarial, permitindo que as empresas identifiquem tendências, segmentos de mercado em crescimento e possíveis parceiros comerciais. Com essas informações em mãos, as empresas podem ajustar suas estratégias de negócios, identificar oportunidades de crescimento e tomar decisões mais assertivas.

7.0.4 Combate a fraude. A *API* de consulta de CNPJ desempenha um papel importante na prevenção de fraudes e lavagem de dinheiro. Ao verificar a autenticidade dos dados cadastrais de outras empresas, é possível identificar possíveis irregularidades e reduzir o risco de envolvimento em transações suspeitas. Essa camada adicional de segurança ajuda as empresas a manterem sua integridade e reputação no mercado.

7.0.5 Facilidade de integração e escalabilidade. As *APIs* de consulta de CNPJ são projetadas para facilitar a integração com outros sistemas e aplicativos. Isso permite que empresas de médio e pequeno porte incorporem essa funcionalidade em seus sistemas existentes, sem a necessidade de grandes investimentos em infraestrutura. Além disso, à medida que os negócios crescem, a *API* pode acompanhar essa expansão, garantindo escalabilidade e eficiência contínuas.

Com esses benefícios as empresas tornam-se melhor equipadas para tomadas de decisão, facilitando o impulsionamento e crescimento em um cenário empresarial brasileiro competitivo.

8 CONCLUSÃO

Após o desenvolvimento da *API* em *Python* para consulta de CNPJ, voltada para empresas de médio porte, é possível concluir que a solução apresentada é viável e eficiente para fornecer acesso rápido e simplificado às informações cadastrais das empresas. A utilização de *APIs*, como a desenvolvida, torna-se cada vez mais comum no cenário empresarial, permitindo a integração e o compartilhamento de dados de forma segura e eficiente.

No entanto, é importante ressaltar que nem todo o potencial do banco de dados foi explorado na implementação atual da *API*, devido a limitações como erros nos dados fornecidos pelo Governo Federal e a necessidade de otimização do tempo de acesso aos dados. Essas limitações apontam para possíveis melhorias em futuras implementações da *API*.

Além disso, as tecnologias utilizadas no desenvolvimento da *API*, como a linguagem de programação *Python*, o sistema de gerenciamento de banco de dados *MySQL* e o *framework Flask*, demonstraram ser escolhas adequadas para a criação da solução. Essas tecnologias são amplamente utilizadas, oferecem uma ampla gama de recursos e são bem documentadas, facilitando o desenvolvimento e a manutenção da *API*.

Apesar de não ter sido utilizada uma *ORM* (*Object-Relational Mapping*) no projeto, essa pode ser uma opção para futuras implementações da *API*, visando a simplificação do acesso e manipulação dos dados do banco de dados. Indo de acordo com cada empresa se vale a pena a perda de desempenho.

No contexto empresarial, a disponibilidade de informações precisas e atualizadas sobre empresas é fundamental para tomadas de decisões estratégicas. A *API* desenvolvida proporciona uma forma acessível e eficiente de consultar os dados cadastrais das empresas, contribuindo para a agilidade e confiabilidade dessas decisões.

Em resumo, a *API* desenvolvida apresenta-se como uma solução viável para empresas de médio porte que necessitam consultar informações cadastrais de CNPJ. Embora ainda haja espaço para melhorias e aprimoramentos futuros, a aplicação das tecnologias escolhidas e a abordagem adotada demonstram ser eficientes e satisfatórias para atender às necessidades de acesso rápido e simplificado aos dados empresariais.

REFERENCES

[1] Bryan Cooksey. [n.d.]. An introduction to APIs. https://zapier.com/resources/guides/apis/introduction.

[2] Petr Gazarov. [n.d.]. What is an API? In English, please. https://www.freecodecamp.org/news/what-is-an-api-in-english-please-b880a3214a82/.

[3] Inc NumFOCUS. [n.d.]. pandas. https://pandas.pydata.org/.

[4] ORACLE. [n.d.]. Optimizing INSERT Statements. https://dev.mysql.com/doc/refman/8.0/en/insert-optimization.html.

[5] ORACLE. [n.d.]. Oracle Cloud Infrastructure Documentation. https://docs.oracle.com/en-us/iaas/mysql-database/doc/getting-started.html.

[6] Pallets. [n.d.]. User's Guide. https://flask.palletsprojects.com/en/2.3.x/.

[7] Various. [n.d.]. API. https://en.wikipedia.org/wiki/API.

[8] Various. [n.d.]. Python (programming language). https://en.wikipedia.org/wiki/Python_(programming_language).