ANÁLISE DE IMPLEMENTAÇÃO E DESEMPENHO DOS FRAMEWORKS WEB PYTHON: FLASK E DJANGO

Faculdade do Centro Leste, Rod. ES – 010, km 6, S/N, Manguinhos, Serra - ES

Resumo

Visando analisar as diferenças, tanto computacionais quanto operacionais dos frameworks Django e Flask, o presente artigo utiliza uma metodologia de revisão bibliográfica das linguagens e das tecnologias aplicadas no desenvolvimento do artigo, assim como um projeto prático para avaliar e parametrizar desempenhos. Concluiu-se que *ambas as estruturas estão amadurecidas e prontas para produção. Cada um tem seus recursos exclusivos e padrões específicos. Uma das principais conquistas foi que o estudo comparativo ajudou entender ambas as tecnologias.*

Palavras-chaves: *Python, Django, Flask, Programação*

1. INTRODUÇÃO

Desde o início das linguagens de programação modernas no início dos anos 50 do século 20, várias linguagens de programação foram inventadas. Algumas foram descontinuados ao longo do tempo e outras sobreviveram com a apreciação de cientistas da computação, programadores e engenheiros. Daquelas que sobreviveram, linguagens como C, C++, Java, Python, JavaScript e Ruby são algumas das linguagens de programação essenciais para construir atualmente aplicativos para web, desktops, dispositivos móveis e outros.

Fundado em dezembro de 1989 por Guido Van, Python é uma das mais populares linguagens de programação em geral. É uma linguagem interpretada, portanto, é compilada para código de bytes e é executada instantaneamente.

Uma das razões para a popularidade do Python é sua legibilidade. Segundo Guido (2014), a nova linguagem deveria preencher o vazio entre o C e o Shell Script. Esse foi o objetivo principal na criação do Python.

Neste artigo, no entanto, vamos buscar comparar e identificar os vários aspectos dos frameworks web mais populares da linguagem de programação Python, ou seja, Flask e Django. De acordo com a documentação, o Flask é um microframework, por ser leve e ter os recursos de flexibilidade. Por outro lado, Django é conhecido por ter vários recursos já incluídos em seu framework, que afirma ter a maioria das extensões e bibliotecas necessárias para inicializar um aplicativo genérico, fornecendo ao desenvolvedor mais tempo para implementar a lógica de negócios. Então, quais os requisitos que definem quando usar cada um? Essas definições de implementação e desempenho são gerais ou existe uma combinação de fatores para que isso aconteça? São perguntas que formaram a base de pesquisa do presente artigo.

2. TECNOLOGIAS ADOTADAS

2.1 Tecnologias Web

O desenvolvimento de aplicações Web é um processo de desenvolvimento para que software possam ser executados diretamente no navegador.

Os aplicativos web e a tecnologia cliente-servidor avançaram bastante com o passar dos anos. Atualmente, as aplicações web são desenvolvidas de diferentes formas, como estáticas, dinâmicas, sistema de gerenciamento de conteúdo, e-commerce, jogos e redes sociais. As tecnologias comumente compartilhadas nos tipos de aplicativos mencionados acima são às tecnologias de back-end e front-end.

O desenvolvimento back-end lida com o lado lógico do aplicativo. Diz respeito principalmente às linguagens de programação, arquitetura central e regras de negócio. Essas lógicas são escritas em linguagens de programação que podem ser executadas em servidores. Também influencia como os dados são armazenados, acessados e servidos a partir dos servidores.

Por outro lado, o desenvolvimento front-end está mais preocupado com a estética, usabilidade e exibição do conteúdo. Um de seus desafios mais difíceis é conseguir mostrar o material nos diversos tipos de dispositivos e navegador. O design do site, a usabilidade e a facilidade de uso são os fatores essenciais que são abordados durante esse desenvolvimento.

2.2 Frameworks Web

Um framework web é uma biblioteca de código que torna o desenvolvimento web mais rápido e fácil, fornecendo padrões comuns para a construção de aplicativos web confiáveis, escaláveis e sustentáveis. Isso significa que um framework tira todas as tarefas complexas e repetitivas e permite que você se concentre na construção do que deseja desenvolver.

Frameworks fornecem funcionalidades em seu código ou por meio de extensões para executar aplicações web. Essas operações comuns incluem:

- Roteamento de URL
- Tratamento e validação do requisições
- HTML, XML, JSON e outros formatos de saída com um mecanismo de modelagem
- Configurações de conexão de banco de dados e manipulação de dados por meio de uma ORM
- Segurança Web contra falsificação de solicitação entre sites (CSRF), injeção de SQL, scripts entre sites (XSS) e outros ataques mal-intencionados
- Armazenamento e recuperação de sessão

Agora, nem todos os frameworks têm todas as funcionalidades listadas acima. Alguns frameworks adotam a abordagem de "tudo incluso", onde tudo o que é possível vem junto com o framework, enquanto outros têm um pacote básico mínimo que pode ser estendido posteriormente. É aqui que a abordagem de ‘Flask’ e ‘Django’ diferem.

2.2.1 Flask

Flask é uma estrutura de aplicativo da web WSGI leve. Ele foi projetado para tornar os primeiros passos rápidos e fáceis, com a capacidade de escalar para aplicativos complexos. Ele começou como um invólucro simples em torno de Werkzeug e Jinja e se tornou um dos frameworks web Python mais populares. Agora, o que isso significa basicamente é que o Flask implementa o mínimo e deixa os recursos para add-ons ou para o desenvolvedor.

Exemplo de aplicações que usam Flask:

- Airbnb
- Netflix
- Uber

2.2.2 Django

Django é um framework Web Python de alto nível que incentiva o desenvolvimento rápido e um design limpo e pragmático. É uma estrutura de aplicativo da Web com uma filosofia de "tudo incluso". O princípio por trás de ter tudo incluso é que as funcionalidades comuns para a construção de aplicativos web deve vir com a estrutura, em vez de necessitar de bibliotecas separadas. Portanto, as funcionalidades comuns listadas acima estão todas disponíveis no Django para serem implementadas rapidamente.

Exemplo de aplicações que usam Django:

- Instagram
- Spotify
- Dropbox

3. COMPARAÇÃO E OBSERVAÇÃO

Embora o Django seja um framework completo com várias funcionalidades já feitas, o Flask deixa tudo sob seu controle. Mas esta não é a única diferença;

Tabela 1. Mostrando as principais diferenças entre Flask e Django

Django	Flask
Estrutura full stack	Estrutura leve com recursos minimalistas
O sistema ORM integrado permite que os desenvolvedores usem qualquer banco de dados e executem tarefas comuns de banco de dados sem ter que escrever longas consultas.	Com o Flask, os desenvolvedores precisam trabalhar com diferentes bancos de dados usando sistemas ORM para Python e SQLAlchemy como o kit de ferramentas SQL. As consultas SQL devem ser escritas para tarefas comuns.
Django não é adequado para projetos onde os requisitos mudam dinamicamente.	Com o Flask, um aplicativo simples pode ser alterado posteriormente para adicionar mais funcionalidade e torná-lo complexo. Ele fornece flexibilidade para expandir o aplicativo rapidamente.
Django é adequado para projetos mais significativos que precisam de muitas funcionalidades. Para projetos mais simples, os recursos podem ser uma overdose.	Flask é uma estrutura simples e sem opinião; ele não decide como seu aplicativo deve se parecer - os desenvolvedores fazem.
O framework Django garante que os desenvolvedores usem as melhores práticas, pois tudo tem um modelo.	O Flask é mais aberto e os desenvolvedores podem ou não seguir as melhores práticas.
Para a mesma funcionalidade, o Django precisa de mais de 2 vezes mais linhas de código do que o Flask.	O aplicativo Flask requer muito menos linhas de código para uma tarefa simples.

4. METODOLOGIA DE PESQUISA

A metodologia utilizada envolveu primeiramente uma revisão bibliográfica das linguagens e das tecnologias aplicadas no desenvolvimento do artigo.
Observa-se que para o sucesso do trabalho é uma etapa importante, pois segundo Silva e Menezes (2005, p. 30), "a revisão de literatura é fundamental, porque fornecerá elementos para evitar a duplicação de pesquisas sobre o mesmo enfoque do tema, e favorecerá a definição de contornos mais precisos do problema a ser estudado".
O estudo das tecnologias e das linguagens foi de natureza aplicada e com objetivo exploratório. A pesquisa aplicada "objetiva gerar conhecimentos para aplicação prática dirigidos à solução de problemas específicos", e a pesquisa exploratória "visa proporcionar maior familiaridade com o problema com vistas a torná-lo explícito ou a construir hipóteses, envolve levantamento bibliográfico"(Silva e Menezes (2005, p. 20)).

4.1 Projeto e Análise

O projeto foi desenvolvido em etapas a fim de organizar o desenvolvimento da aplicação, sendo elas:

- Definição;
- Estudo de viabilidade;
- Pesquisa;
- Implementação/Construção;
- Análises e Conclusões

Para a comparação e análise das tecnologias propostas, foram levantados alguns requisitos como:

- Inicialmente, criar um projeto “Hello World” para implementar uma arquitetura básica em ambos os frameworks
- Desenvolver um CRUD, simulando um estoque simples
- Adotar uma ferramenta de consumo de API para padronizar um benchmark de desempenho computacional para ambas as tecnologias.
- Analisar desempenho operacional e computacional para ambos os frameworks

5. DESENVOLVIMENTO

5.1 Arquitetura Básica

5.1.2 SQLite

SQLite é uma biblioteca multiplataforma escrita na linguagem C que ao ser implementada no projeto, permite acesso a um banco de dados SQL. Seu código é de domínio público, sendo livre para uso em qualquer propósito, seja comercial ou privado.

O SQLite não exige instalação, apenas sua integracação como uma biblioteca no programa em desenvolvimento, portanto, foi utilizada para esse projeto pela sua facilidade de gerenciamento e instalação.

5.1.2 Flask

Uma boa prática é isolar seu ambiente Python antes de instalar qualquer pacote. Isso pode ser feito usando ambientes virtuais. A ferramenta que utilizei para fazer isso foi o 'virtualenv'. Usando uma distribuição baseada em debian como o Ubuntu, realiza-se o seguinte comando:

```
sudo apt-get install virtualenv
```

Com o virtualenv instalado, deve ser criado um ambiente virtual e ativá-lo.

```
virtualenv -p python3 venv/
source venv/bin/activate
```

A primeira coisa a fazer para se usar o Flask é instalá-lo. Isso pode ser feito facilmente usando pip.

```
pip install flask
```

Em seguida, foi criado um arquivo python chamado flaskhello.py e inserido o seguinte código:

```
from flask import Flask

app = Flask(__name__)
@app.route("/")
def hello():
 return "Hello, World!"
if __name__ == "__main__":
 app.run()
```

5.1.3 Django

O Django também pode ser instalado por meio do pip.

```
pip install django
```

Ao se instalar o Django, ele também configura o comando Django-admin, que é usado para iniciar um projeto Django:

```
django-admin startproject hellodjango
```

Aqui é criado um "projeto" Django, e ele criará o diretório hellodjango. No diretório hellodjango, ele criou um arquivo chamado manage.py e um subdiretório que também é chamado hellodjango. Dentro do subdiretório, existem três scripts Python. Nessa etapa é necessário atenção apenas com urls.py.

A próxima etapa é usar o Django para criar um aplicativo, que é uma estrutura organizacional abaixo de um projeto (um projeto pode conter muitos aplicativos). Usarei o arquivo manage.py que o comando anterior criou, para criar o aplicativo. No diretório externo hellodjango, executamos o seguinte comando:

```
python manage.py startapp helloworld
```

Isso cria o aplicativo helloworld e o torna parte de projeto hellodjango. Agora é preciso configurar o roteamento de URL (como fizemos com @app.route no Flask). Como os projetos Django têm mais estrutura padrão do que os aplicativos Flask, teremos algumas etapas extras. O comando anterior criou um diretório helloworld dentro do diretório hellodjango externo. Agora é necessário abrir o arquivo helloworld/views.py criado automaticamente e adicione o seguinte código:

```
from django.http import HttpResponse
def index (request):
  return HttpResponse("Hello, World!")
```

Agora é preciso criar um arquivo urls.py para o aplicativo. Isso é feito criando o arquivo helloworld/urls.py e adicionando o seguinte código:

```
from django.conf.urls import url
from . import views

urlpatterns = [
url(r'^$', views.index, name='index'),
]
```

Essa é a configuração de URL para o aplicativo (helloworld). Também é necessário de uma configuração de URL para o projeto (hellodjango). Agora é necessário editar o arquivo hellodjango / hellodjango / urls.py, adicionando o seguinte código:

```
from django.conf.urls import include, url

urlpatterns = [
url(r'^hello/',include('helloworld.urls')),
]
```

Isso é semelhante ao arquivo anterior. No entanto, em vez de rotear URLs de um padrão específico para uma visualização específica, estamos roteando-os para um aplicativo específico. Nesse caso, qualquer URL que tenha /hello depois disso será enviado para nosso aplicativo helloworld e procurará em helloworld.urls para descobrir qual visualização chamar. Agora voltando para o diretório hellodjango (aquele que contém o arquivo manage.py), é necessário executar o seguinte comando:

```
python manage.py runserver
```

Isso executa o servidor de desenvolvimento Django, que permite visitar o aplicativo no localhost, como foi feito com o Flask.

5.2 Análise de Implementação

Após criar a arquitetura básica, foi realizada a implementação do CRUD em ambas às tecnologias.

Como definido, foi desenvolvido um CRUD simples de um estoque, com os seguintes métodos:

```
GET - /produto       | Seleciona Produtos
GET - /produto/<id>  | Seleciona Individual
POST - /produto      | Cadastra Novo Produto
PUT - /produto/<id>  | Atualiza Produto
DELETE- /produto/<id>| Deleta Produto
```

Sendo a tabela criada e trabalhada a seguinte:

```
Estoque:
{
        id – pk
        produto – string
        preco - string
}
```

- Nesta análise, foram utilizados os métodos GET e POST
- A máquina utilizada contém:
    - Processador I5 7th Gen
    - 8GB de RAM
    - 1TB de HD
    - SO Windows 10 64bits

5.2.1 Flask

Para realizar a implementação em Flask, foi necessário de início pesquisar por uma ORM que fizesse a comunicação com o banco de dados, nesse caso a escolhida foi a SQLALCHEMY.
Foram realizadas as configurações da SQLAlchemy com o banco de dados SQLite.
Em seguida, foi implementado manualmente as operações de criação do banco de dados e o CRUD, usando a forma descritas na documentação do Flask + SQLAlchemy.

A arquitetura ficou a seguinte:

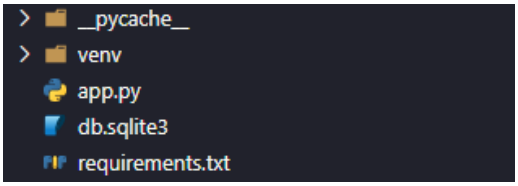

Figura 1. Arquitetura do CRUD Flask

- **_pycache_:** Arquivos cache do flask
- **venv:** Ambiente Virtual
- **app.py:** arquivo onde ficam as configurações e implementações do CRUD
- **db.sqlite3:** Banco de dados
- **requirements.txt**: local onde as dependências de instalação de todas as bibliotecas do projeto ficam salvas.

5.2.2 Django

No Django já existe uma ORM integrada junto ao framework, o que delimita sua escolha em entender a documentação para aplicar no projeto.
O framework utiliza um conceito de modularidade, onde em um projeto pode ser criado vários apps.
Seguindo essa ideia, foi criado o projeto “Estoque” e o app “Produto”.
Além disso, é necessário instalar a biblioteca “djangorestframework”, que é quem possibilita a criação do crud no Django de forma nativa.
Ao instalar essa biblioteca, além da parte de modelagem de dados foi necessário criar mais duas camadas, que foram: serializers.py e view.py.
A realização do CRUD depende a implementação dessas camadas.
Por fim, foram configuradas as rotas tanto no app quanto no projeto.
A arquitetura ficou a seguinte:

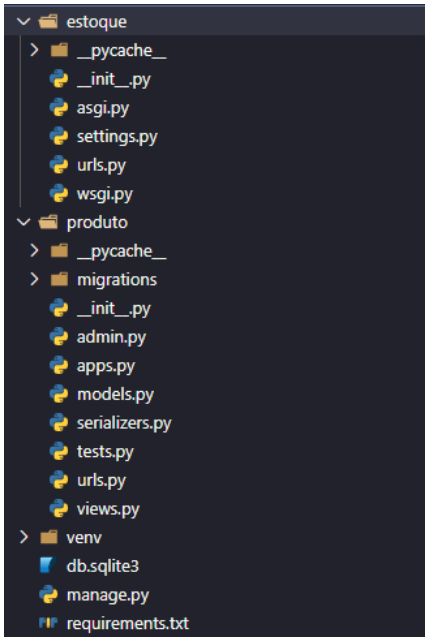

Figura 2. Arquitetura do CRUD Django.

- **estoque:** Pasta do projeto com as configurações “settings.py” e rotas “urls.py”
- **produto:** Pasta do app, com o modelo de dados “models.py”, roteamento do app “urls.py”, serialização do modelo de dados “serializers.py” e as views que definem o CRUD “views.py”
- **venv:** Ambiente Virtual
- **db.sqlite3:** Banco de dados
- **requirements.txt**: local onde as dependências de instalação de todas as bibliotecas do projeto ficam salvas.

Com essa configuração, o Django já dispõe de métodos de autenticação, tokens, painel de administração e criação de usuários com diferentes permissões. Isso devido a todas as ferramentas que já vem integradas na ferramenta.

5.2 Análise de Desempenho

A análise de desempenho se deu através dos resultados de requisições para cada endpoint definido e seus métodos. Para isso foi utilizado o software Apache Jmeter, que é uma ferramenta para realizar testes de carga e de estresse em recursos estáticos ou dinâmicos oferecidos por sistemas computacionais

5.2.1 Análise Método GET

```
GET - /produto | Seleciona todos os Produtos
```

Para este método em ambos os frameworks a análise se baseou nos seguintes parâmetros:

- 50 itens no banco de dados
- Simulação de requisição simultânea de 10,20,50,100,200,400,600,800 e 1000 usuários

A partir disso, gerou-se uma tabela e gráficos que mostram de forma visual a performance desse resultado.

Tabela 2. Mostrando as diferenças de performance em Flask no método GET

Usuários	Média	Mediana	90%	95%	99%	Min	Max	Tx Erro %
10	4	4	5	5	9	3	9	0.0
20	4	4	5	5	7	3	7	0.0
50	6	5	9	10	75	3	75	0.0
100	5	4	7	9	28	3	30	0.0
200	186	220	303	315	324	20	326	0.0
400	702	848	1020	1031	1038	31	1040	0.0
600	1841	1853	2763	3024	3137	8	3169	0.83%
800	1899	2030	2983	3159	3240	15	3249	3.38%
1000	1934	2021	2904	2979	3179	30	3184	24.7%

Tabela 3. Mostrando as diferenças de performance em Django no método GET

Usuários	Média	Mediana	90%	95%	99%	Min	Max	Tx Erro %
10	11	10	14	14	28	7	28	0.0
20	11	11	13	15	23	7	23	0.0
50	34	20	82	90	126	7	126	0.0
100	260	188	540	711	1032	26	1034	0.0
200	888	785	1749	1800	2139	45	2160	0.0
400	1535	1884	2616	2738	3310	24	3708	24.5%
600	1696	2032	2206	2674	3189	38	3258	51.66%
800	1825	2030	2053	2190	2789	47	2941	73.12%
1000	1894	2032	2117	2746	3400	69	3600	69.6%

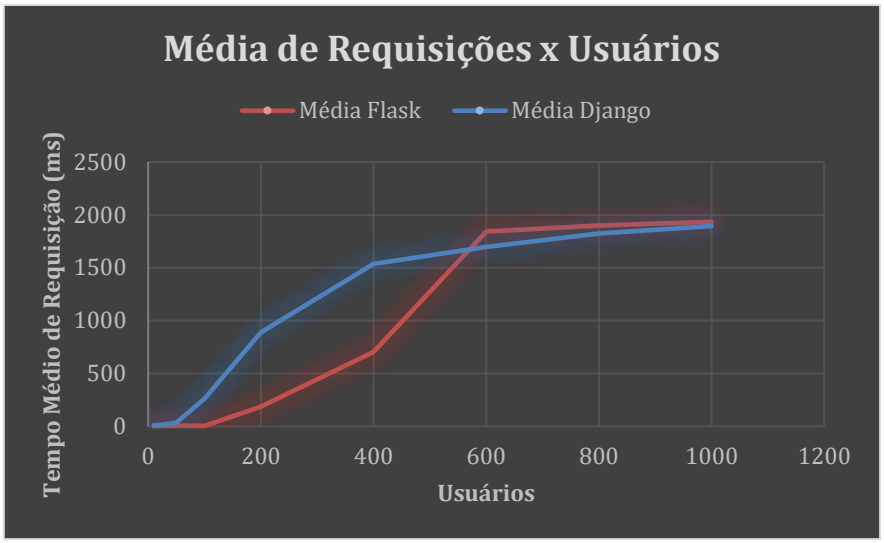

Figura 3. Média de tempo de requisição por quantidade de usuários no método GET

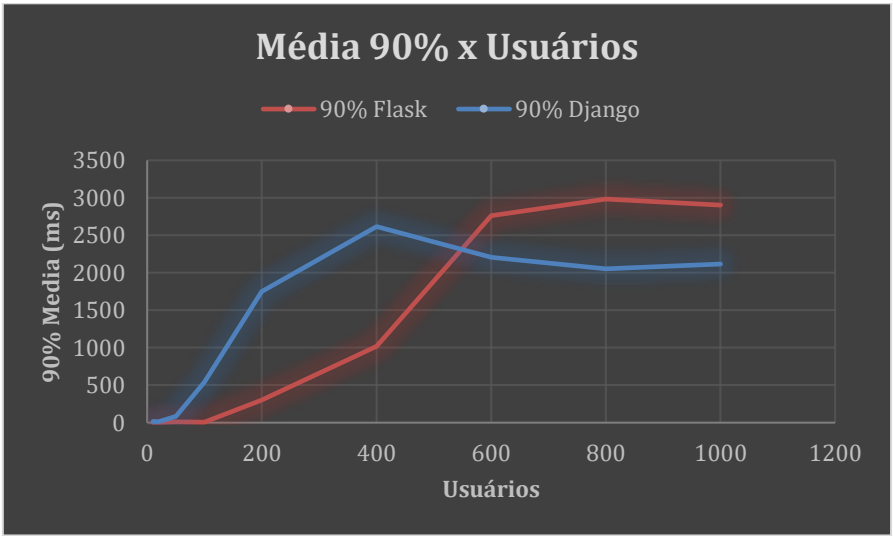

Figura 4. Média de tempo em 90% das requisições em ordem crescente por quantidade de usuários no método GET

Figura 5. Porcentagem de erro das requisições por quantidade de usuários

5.2.1 Análise Método POST

```
POST - /produto      | Cadastra Novo Produto
```

Da mesma forma que o método GET, no POST foi padronizado os seguintes testes:

- Simulação de cadastro de produto simultâneo por 10,20,50,100,200,400,600,800 e 1000 usuários
- Cada usuário cadastra apenas 1 produto simples.

Os resultados se encontram nas tabelas e gráficos abaixo:

Tabela 4. Mostrando as diferenças de performance em Flask no método POST

Usuários	Média	Mediana	90%	95%	99%	Min	Max	Tx Erro %
10	5	5	7	7	8	4	8	0
20	5	5	7	7	15	4	15	0
50	7	5	15	17	29	3	29	0
100	34	22	92	105	112	4	115	0
200	388	414	552	564	575	16	583	0
400	1233	1276	1891	1943	1947	62	1951	0
600	1913	2067	2611	2970	3089	234	3117	0,5%
800	2196	2070	3487	3781	3810	27	3834	20,38%
1000	2232	2041	3279	3364	3424	27	3456	55%

Tabela 5. Mostrando as diferenças de performance em Django no método POST

Usuários	Média	Mediana	90%	95%	99%	Min	Max	Tx Erro %
10	35	20	78	78	85	12	85	0%
20	21	21	25	29	35	15	35	0%
50	652	284	1532	1719	1809	14	1809	0%
100	2870	2661	5951	6132	6307	50	6656	0%
200	8759	7601	15071	15281	15540	58	15739	45,5%
400	25200	31252	33824	34221	34751	77	35044	80,75%
600	31198	42256	46773	47145	47679	266	48435	88,17%
800	8675	3718	24729	25281	26146	455	26502	95,25%
1000	8019	2035	27500	28450	29238	243	29865	93%

Figura 6. Média de tempo de requisição por quantidade de usuários no método POST

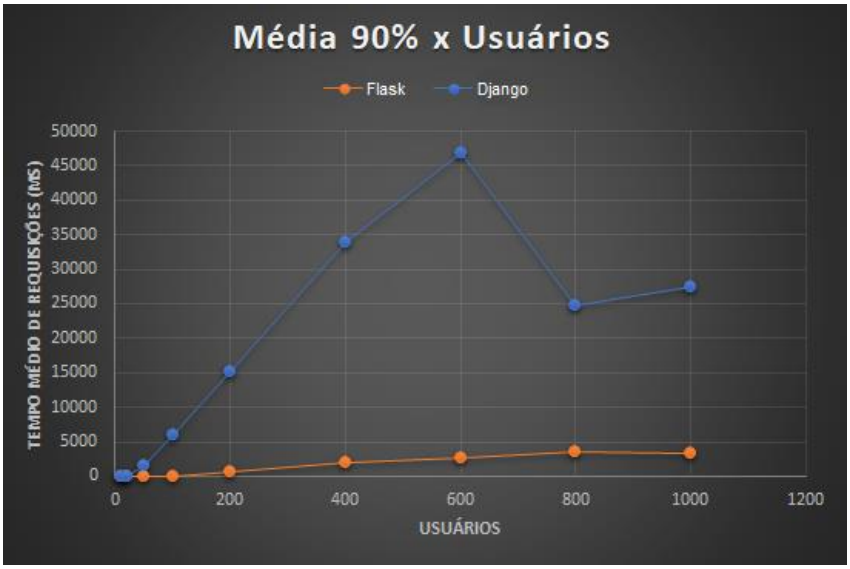

Figura 7. Média de tempo em 90% das requisições em ordem crescente por quantidade de usuários no método POST

Figura 8. Porcentagem de erro das requisições por quantidade de usuários no método POST

6. CONCLUSÃO

O objetivo do projeto era comparar os dois frameworks web Python.
A comparação foi realizada durante o desenvolvimento de uma aplicação web simples com cada um dos frameworks. Cada estrutura tem suas vantagens e desvantagens, mas ambos tentam cumprir o mesmo objetivo com uma abordagem diferente em cada framework.
Cada aplicação web é feita para resolver problemas com um motivo comercial específico. Esses motivos de negócios podem vir de uma pequena startup para uma grande empresa.
Ambas as estruturas estão amadurecidas e prontas para produção. Cada um tem seus recursos exclusivos e padrões específicos.
Uma das principais conquistas deste projeto foi que o estudo comparativo ajudou entender ambas as tecnologias.

6.1 Implementação

Os dois frameworks parecem ser opções viáveis se forem utilizadas na resolução de problemas de negócio.
O Flask tem um padrão inicial minimalista, dando a opção e ao mesmo tempo obrigando a uma escolha de ferramentas complementares ao desenvolvimento, como foi o caso da ORM no projeto. A curva de aprendizado básica é simples, mas ao mesmo tempo não existem muitos recursos iniciais além de um servidor rodando na máquina.
O Django por outro lado tem uma curva de aprendizado básico, sendo justo, um pouco maior. Mas isso ocorre devido as configurações e arquitetura de projeto utilizada, porém com a configuração básicas da aplicação já conta com diversas ferramentas essenciais como autenticação, tokens, segurança do software e painel para administração.
Porém, com base no estudo, é evidente que o Django pode ser mais adequado para projetos com o custo da curva de aprendizado, visto que a arquitetura base do framework é pré-definida, o que diminui a curva de aprendizagem de um programador que entra em um projeto em desenvolvimento já conhecendo o básico do Django. O Flask é o mais adequado para prototipagem e projetos de pequena escala, mas não se limita a eles, pois pode ser aprendido e configurado rapidamente, principalmente por profissionais mais experientes, mas quando trata-se de gerenciar e manter, requer mais trabalho do que o anterior, pois toda definição de ferramentas complementares e arquitetura de projeto é independente, o que pode causar diferentes combinações que geram um aumento no custo de curva de aprendizagem, mas em contrapartida, podem gerar um ganho de performance quando comparado ao Django, que muitas vezes tem configurações e recursos que não são usados em projetos pequenos, por exemplo, mas ficam acoplados ao funcionamento básico do framework.
Podemos concluir que Django é completo, então requer menos decisões a serem tomadas. No entanto, se houver discordância de alguma escolha que o Django faz ou se um projeto tiver requisitos exclusivos que limitam o número de recursos dos quais você pode tirar vantagem no Django, o Flask se torna interessante mesmo em grandes aplicações.
Existem também limitações para o estudo, uma vez que nem todos os aspectos e possibilidades das estruturas foram estudadas. Este tipo de estudo requer comparação altamente detalhada de todos os recursos.

6.2 Desempenho

o Flask é um microframework, o que significa que ele oferece um desempenho muito melhor em termos de velocidade e isso é comprovado nas tabelas 2 e 4 quando comparadas com as tabelas 3 e 5.

No método GET, com 10 usuários simultâneos o tempo mínimo do Flask foi 3ms e o máximo 9ms. Já o Django teve o desempenho mínimo de 7ms e máximo de 28ms.

Com 1000 usuários simultâneos o Flask teve um tempo mínimo de 30ms e máximo de 3184ms e o Django um mínimo de 69ms e máximo de 3600.

A taxa de erro é algo importante de constatar, pois no método GET o Flask começa a ter erros nas requisições com 600 usuários, sendo a taxa de 0,83%. Já o Django começa a ter erros com 400 usuários, com uma taxa de 24,5%.

Isso influência no tempo médio de requisição, que pode ser observado na figura 4, onde a partir de 400 usuários o tempo começa a diminuir, e isso ocorre pois como a taxa de requisições com erro aumenta, o processamento de dados por requisição diminui visto que o servidor retorna um aviso de erro ao invés de uma carga de dados, que levaria mais tempo.

Em geral, em sua estrutura básica o Flask é mais performático. Mas é necessário ser justo e considerar que o Django é muito mais robusto e já vem com uma base de código acoplada ao básico muito superior ao do Flask, pois conta com padrões de segurança, tokens e autenticação que ficaram visíveis no projeto.

Quando falamos sobre desempenho, também precisamos considerar o poder que cada estrutura fornece quando você deseja construir aplicativos grandes. Django é melhor nesse sentido, pois permite que você crie aplicativos massivos de nível empresarial atuando como uma estrutura full-stack, que pode ser facilmente integrada com JavaScript para construir ótimos aplicativos. Por outro lado, o Flask, em sua estrutura básica, não é adequado para grandes aplicações. Então, escolher uma das duas tecnologias depende principalmente do tipo de aplicação que será desenvolvida.

REFERÊNCIAS

[1] GRINBERG, Miguel. **Flask Web Development:** Developing Web Applications with Python. 2 ed. São Paulo: Novatec, 2019.

[2] GHIMIRE, Devndra. Comparative study on Python web frameworks: Flask and Django. 2020. Bachelor's Thesis - Metropolia University of Applied Sciences, Finlândia, 2020.

[3] SILVA, E. L.; MENEZES, E. M. Metodologia da Pesquisa e Elaboração de Dissertação. 4.ed. Florianópolis: UFSC, 2005. Disponível em: https://projetos.inf.ufsc.br/arquivos/Metodologia_de_pesquisa_e_elaboracao_de_teses_e_dissertacoes_4ed.pdf. Acesso em: 15 jun, 2021.

[4] GUIDO, Van Rossum, **A história do Python**. Holanda, 2014.

[5] VYRAS, Ishan. Django vs Flask in 2021: Battle of Backend Web Development. Citrusbug, India, 01 de abr. de 2021. Disponível em: <https://citrusbug.com/article/django-vs-flask>. Acesso em: 27 de jun. de 2021.

[6] RICARDO, Augusto. DESENVOLVIMENTO DE APLICAÇÕES MULTIPLATAFORMA COM KIVY. 2016. UNIARA, Araraquara – SP, 2016.

[7] HARGAN, John. Django vs Flask: Picking the Right Python Web Framework. TIVIX, 12 de abr. de 2021. Disponível em: <https://www.tivix.com/blog/django-vs-flask-picking-the-right-python-framework>. Acesso em: 27 de jun. de 2021.