Um estudo comparativo entre tecnologias de *back-end*: Node.js, Django REST Framework e ASP.NET Core

Klayver Ximenes Carmo

Orientador: Prof. Dr. Fischer Jônatas Ferreira

Curso de graduação em Engenharia de Computação
Universidade Federal do Ceará

28 de novembro de 2023

UNIVERSIDADE FEDERAL DO CEARÁ

Sumário

1 Introdução

2 Fundamentação Teórica

3 Metodologia

4 Resultados e discussões
- Comparação teórica
- Comparação prática

5 Contribuições

6 Conclusões e trabalhos futuros

UNIVERSIDADE FEDERAL DO CEARÁ

Contexto

- O desenvolvimento Web teve início na década de 90 com a criação dos primeiros sites;
- Com o passar do tempo o desenvolvimento Web ficou mais dinâmico e interativo;
- Surgimento dos termos *front-end* e *back-end*;
- Com esta camada é possível gerenciar servidores, bancos de dados, processar e armazenar dados.

Contexto

Figura: Frameworks *back-end*

Fonte: O autor.

Contexto

Em 2022, o StackOverflow conduziu uma pesquisa com mais de 70 mil desenvolvedores, elencando os *frameworks* mais utilizados.

Figura: Frameworks *back-end*

Fonte: O autor.

Stack Overflow Annual Developer Survey

Objetivo

A definição dos objetivos foram feitos com base no modelo objetivo-questão-métrica (GQM).
Foram **comparados** *frameworks back-end* de maneira teórica e prática;
com a finalidade de i) identificar funcionalidades nativas em fase de desenvolvimento e ii) analisar sua performance e desempenho;
no que diz respeito a auxiliar profissionais e pesquisadores na escolha da abordagem mais adequada para suas estratégias;
sob a perspectiva de desenvolvedores e pesquisadores na área de desenvolvimento *back-end* no contexto de construção e estudo de aplicações Web.

Objetivos Específicos

- Realizar estudos teóricos sobre os *frameworks*;
- Pontuar características de desenvolvimento entre os *frameworks*;
- Desenvolver um *software* alvo utilizando os *frameworks* em estudo;
- Aplicar e analisar métricas com base no *software* alvo desenvolvido.

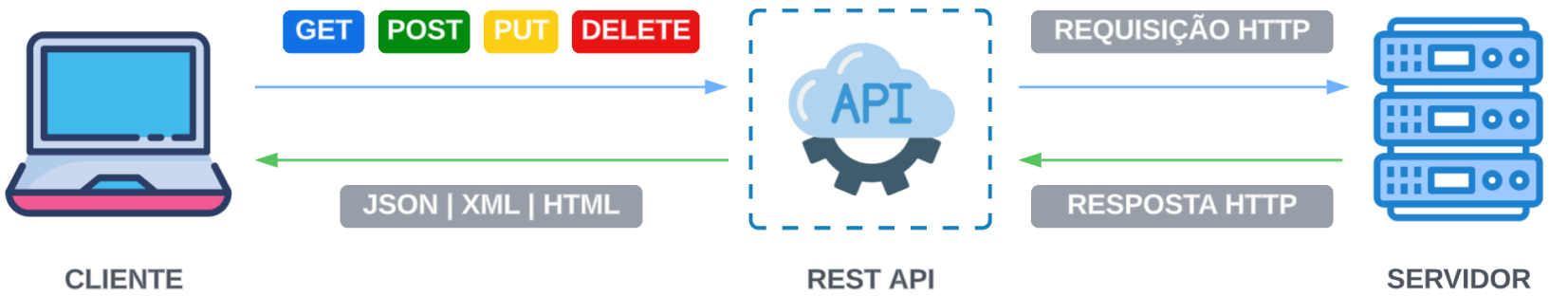
UNIVERSIDADE FEDERAL DO CEARÁ

Motivação

Com base em pesquisas bibliográficas, até o momento, não foram encontrados estudos comparativos focados à avaliação das compatibilidades/funcionalidades teóricas e à análise de cenários práticos de testes nos três *frameworks* em estudo.

Fundamentação Teórica

Figura: Modelo arquitetural de uma API REST



Fonte: O autor.

Fundamentação Teórica

Node.js

- O Node.js é um *framework* baseado em JavaScript;
- Foi construído sobre a *engine* V8 para o Chrome;
- Modelo de programação: Assíncrono e orientado a eventos.

https://nodejs.org/

Fundamentação Teórica

Django REST Framework

- É um *framework* Web baseado na linguagem Python;
- DRF funciona sob a estrutura do Django;
- Versão do Django voltada para o desenvolvimento de APIs;

https://www.django-rest-framework.org/

Fundamentação Teórica

ASP.NET Core

- Linguagem principal: C#;
- Framework multiplataforma desenvolvido pela Microsoft;
- Modelo de programação: Orientado a Objetos;

https://learn.microsoft.com/en-us/aspnet/core

Testes de carga

Cenários de teste

- Teste de pico
  - Um servidor é submetido à uma grande quantidade de requisições durante um curto intervalo de tempo;
- Teste de carga crescente
  - Um servidor é submetido à uma carga crescente de requisições e monitorada para analisar o comportamento sob esta situação.
- Teste de resistência
  - Um servidor é submetido a uma carga constante durante um longo período de tempo, buscando identificar o comportamento geral do sistema.

Questões de pesquisa

QP_1: *Qual é o framework com mais funcionalidades nativas seguindo as características escolhidas para estudo?*

Cenário 1 - Teste de pico

- ***QP_2:*** *Qual é o framework mais otimizado com relação ao consumo de recursos no cenário de teste de pico?*
- ***QP_3:*** *Qual é o framework mais otimizado com relação ao tempo de resposta no cenário de teste de pico?*

Questões de pesquisa

Cenário 2 - Teste de carga crescente

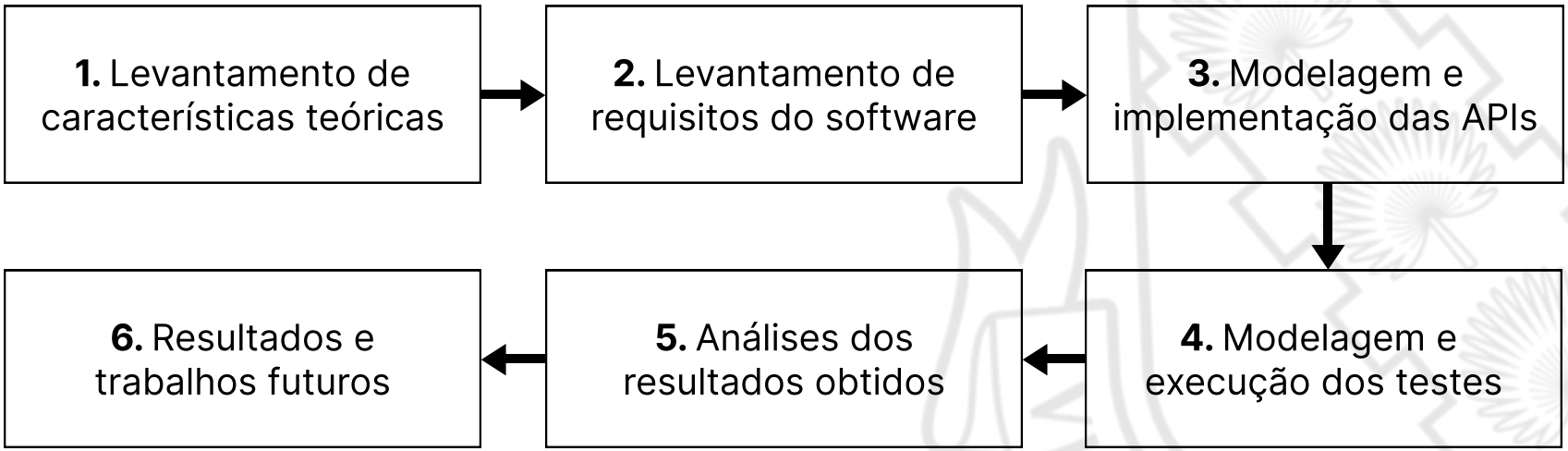
- ***QP*$_4$*:*** *Qual é o framework mais otimizado com relação ao consumo de recursos no cenário de teste de carga crescente?*
- ***QP*$_5$*:*** *Qual é o framework mais otimizado com relação ao tempo de resposta no cenário de teste de carga crescente?*

Cenário 3 - Teste de resistência

- ***QP*$_6$*:*** *Qual é o framework mais otimizado com relação ao consumo de recursos no cenário de teste de resistência?*
- ***QP*$_7$*:*** *Qual é o framework mais otimizado com relação ao tempo de resposta no cenário de teste de resistência?*

Metodologia

Figura: Etapas metodológicas



Fonte: O autor.

Metodologia

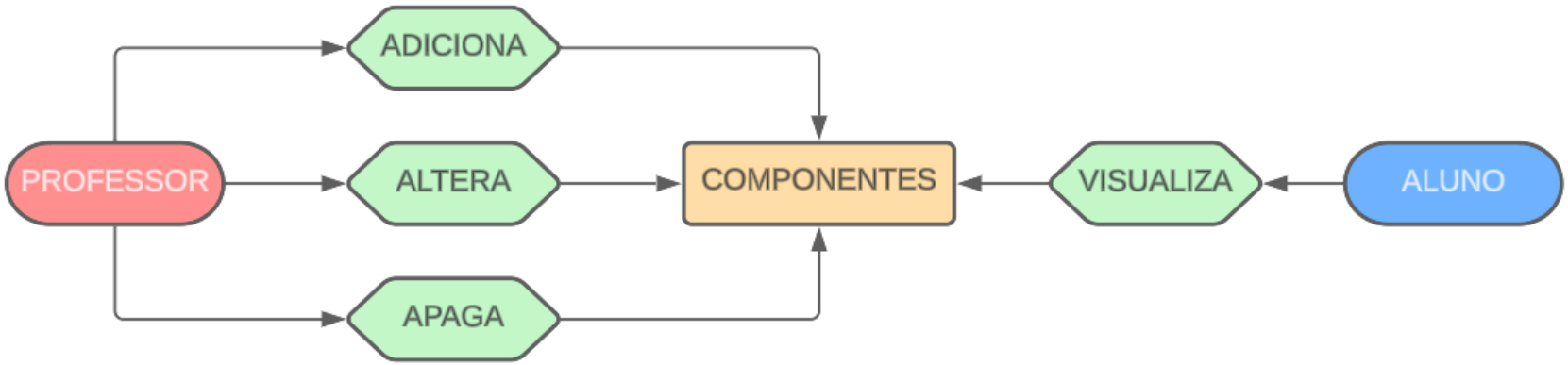
A partir de uma revisão bibliográfica foram definidas as seguintes características teóricas para comparação:

- Suporte a diferentes sistemas operacionas;
  - Windows;
  - Linux;
  - MacOS;
- Suporte a bancos de dados;
  - MySQL;
  - PostgreSQL;
  - SQLite;
  - MongoDB;
  - SQL Server;
- Suporte a ORM nativo;
- Documentação nativa da API em tempo de desenvolvimento.

Metodologia

Levantamento de requisitos do *software* alvo

Figura: Diagrama do *software* alvo



Fonte: O autor.

Metodologia - modelagem do *software* alvo

Tabela: Métodos do *software* alvo

Método	Ação	Exemplo de uso
GET	Ler	Buscar informações dos componentes
POST	Criar	Inserir um componente na base de dados
PUT	Atualizar	Alterar informações de um componente
DELETE	Apagar	Remover um componente da base de dados

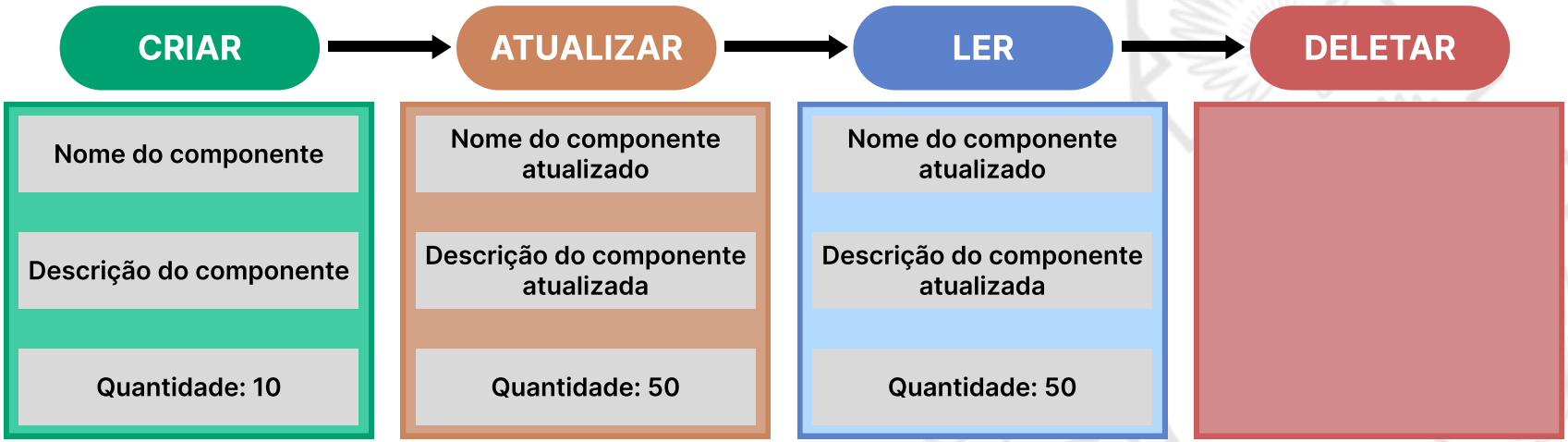
Metodologia

Métricas comparativas práticas

- Taxa de erros de requisição
  - Essencial para analisar a proporção de solicitações que resultam em erros em relação ao número total de solicitações feitas durante os testes;
- Tempo de requisição
  - Importante para medir o tempo que uma API leva para responder ou completar uma solicitação de um cliente;
- Consumo de memória RAM
  - Verificar o quanto de memória é consumido durante a execução das solicitações.
- Consumo de CPU
  - Similar ao consumo de memória, verifica o gerenciamento de recursos de uma aplicação com relação a CPU;

Modelagem dos testes

Figura: Fluxo de requisições dos testes



Fonte: O autor.

Execução dos testes

- Modelagem dos testes feitos no JMeter;
  - Tempo de resposta;
  - Taxa de erros de requisição;
- Execução no Linux;
  - Consumo de CPU;
  - Consumo de RAM;
- Para a otimização do processo de testes foram utilizadas variáveis ambiente no JMeter e um *script* para a execução dos testes via linha de comando;

UNIVERSIDADE FEDERAL DO CEARÁ

https://jmeter.apache.org/

Resultados comparação teórica

CARACTERÍSTICAS		Node.js	DRF	.NET Core
Sistemas operacionais	**Windows**	Sim	Sim	Sim
	Linux	Sim	Sim	Sim*
	MacOS	Sim	Sim	Sim*

Tabela: Suporte das tecnologias aos principais SOs.

**É importante ressaltar que apenas a estrutura do ASP.NET Core é multiplataforma, se diferenciando da estrutura .NET Framework que tem suporte apenas para o sistema operacional Windows.*

Resultados comparação teórica

CARACTERÍSTICAS		Node.js	DRF	ASP.NET Core
Banco de dados	**MySQL**	Sim	Sim	Sim
	PostgreSQL	Sim	Sim	Sim
	SQLite	Sim	Sim	Sim
	MongoDB	Sim	Não*	Sim
	SQL Server	Sim	Sim	Sim

Tabela: Suporte das tecnologias aos principais bancos de dados.

**Bancos de dados não relacionais não são suportados nativamente pelo Django.*

Resultados comparação teórica

CARACTERÍSTICAS	Node.js	DRF	ASP.NET Core
ORM nativo	Não	Sim	Não

Tabela: Suporte das tecnologias ao ORM nativo

Resultados comparação teórica

CARACTERÍSTICAS	Node.js	DRF	ASP.NET Core
Documentação nativa	Não	Não*	Sim

Tabela: Suporte das tecnologias à documentação nativa.

**A geração de documentação pelo DRF não é feita de forma automática, porém existe suporte integrado para esquemas OpenAPI.*

Resultados e discussões

QP1: Qual é o *framework* com mais funcionalidades nativas seguindo as características escolhidas para estudo?

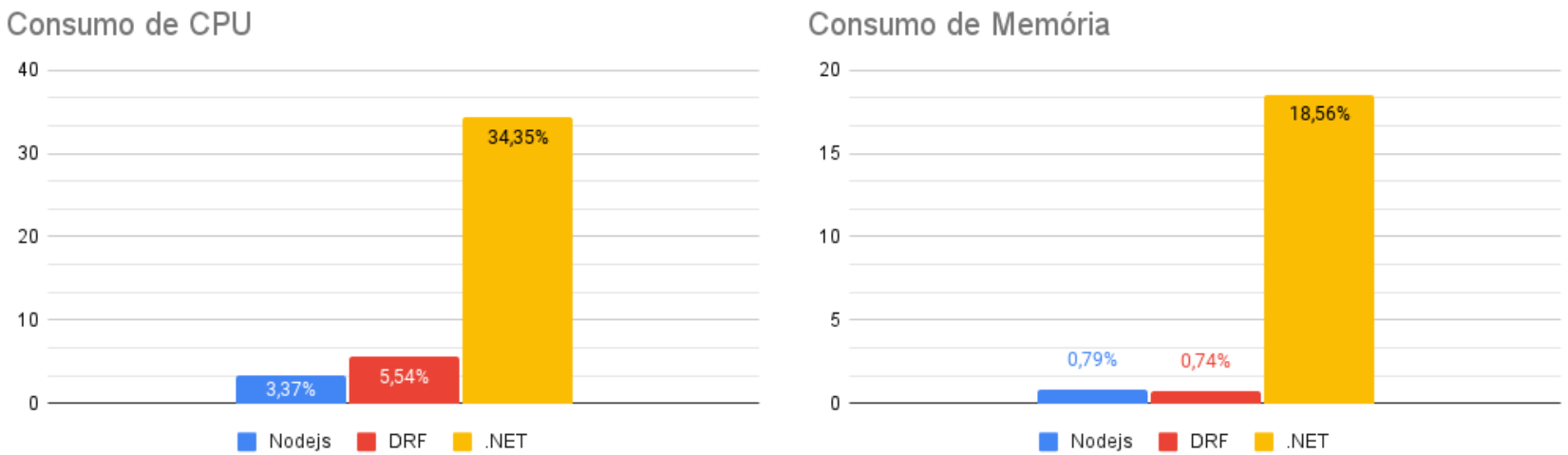
De acordo com os estudos bibliográficos realizados, o DRF e o .Net apresentaram mais funcionalidades nativas. No entanto, os três *framework* possuem compatibilidade para diversas funcionalidades, principalmente utilizando bibliotecas de terceiros. Portanto, os três são flexíveis para adaptação dependendo do contexto que serão utilizados.

Resultados - outras observações

- Todas os *framework* ofereciam suporte para a construção das APIs;
- Dificuldades no desenvolvimento em .Net devido à liguagem e seu paradigma;
- Flexibilidade no desenvolvimento com Node.js e DRF;

Resultados - teste de pico

Figura: Consumo médio de recursos - teste de pico



Fonte: O autor.

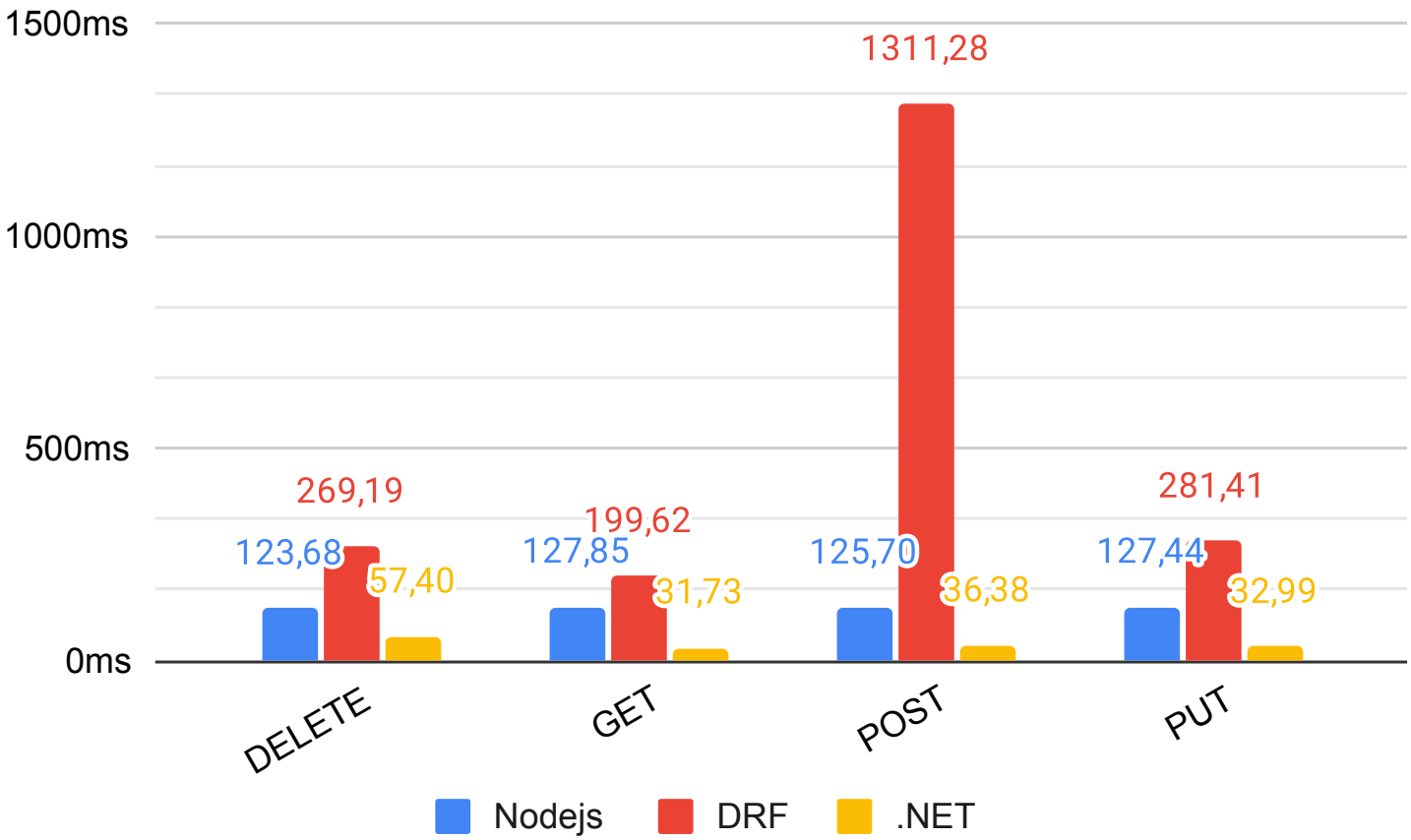
Resultados - teste de pico

QP2: Qual é o *framework* mais otimizado com relação ao consumo de recursos no cenário de teste de pico?

O Node.js apresentou menor consumo de CPU enquanto o DRF mostrou melhor resultado no consumo de RAM.

Resultados - teste de pico

Figura: Tempo de resposta médio dos métodos HTTP - teste de pico



Fonte: O autor.

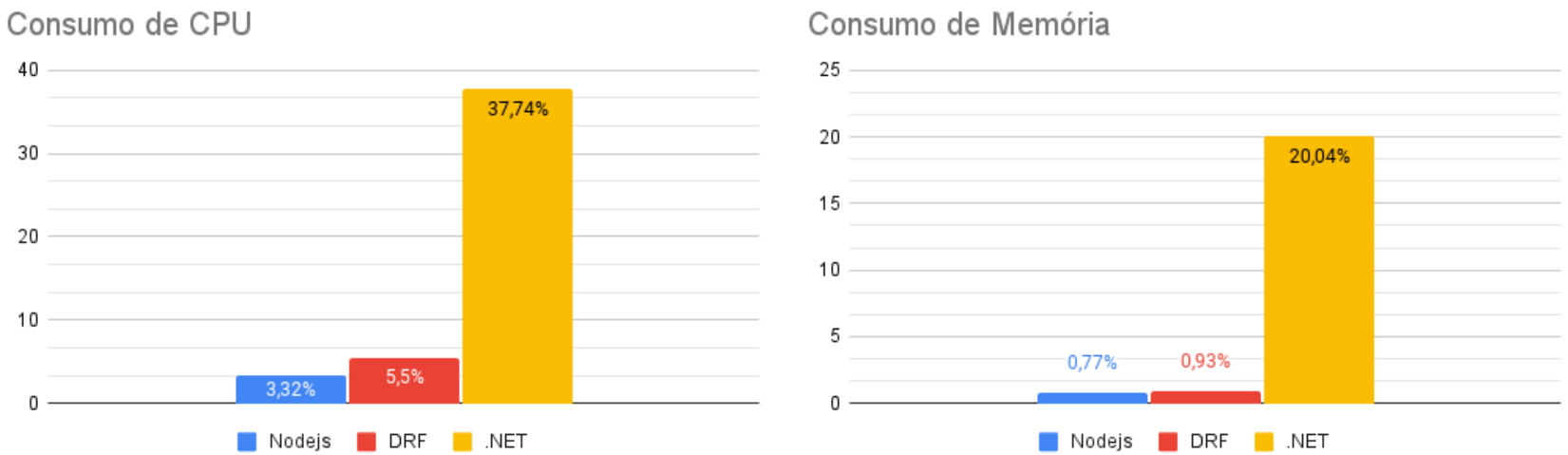
Resultados - teste de pico

QP3: Qual é o *framework* mais otimizado com relação ao tempo de resposta no cenário de teste de pico?

O .Net apresentou os melhores resultados em tempo de resposta em todos os métodos.

Resultados - teste de carga crescente

Figura: Consumo médio recursos - teste de carga crescente



Fonte: O autor.

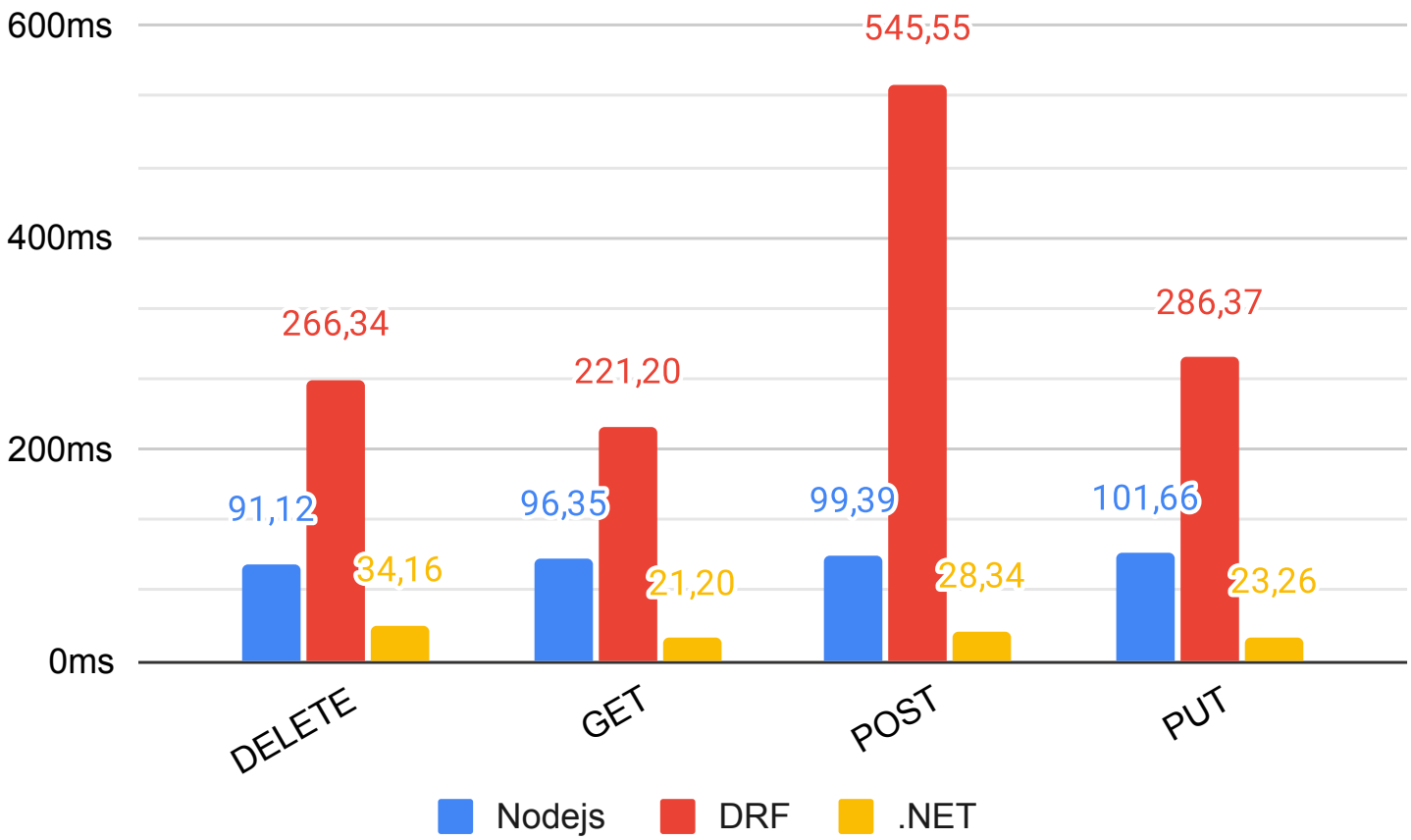
Resultados - teste de carga crescente

QP4: Qual é o *framework* mais otimizado com relação ao consumo de recursos no cenário de teste de carga crescente?

O Node.js apresentou os melhores resultados, tanto para consumo de CPU quando RAM.

Resultados - teste de carga crescente

Figura: Tempo de resposta médio dos métodos HTTP - teste de carga crescente



Fonte: O autor.

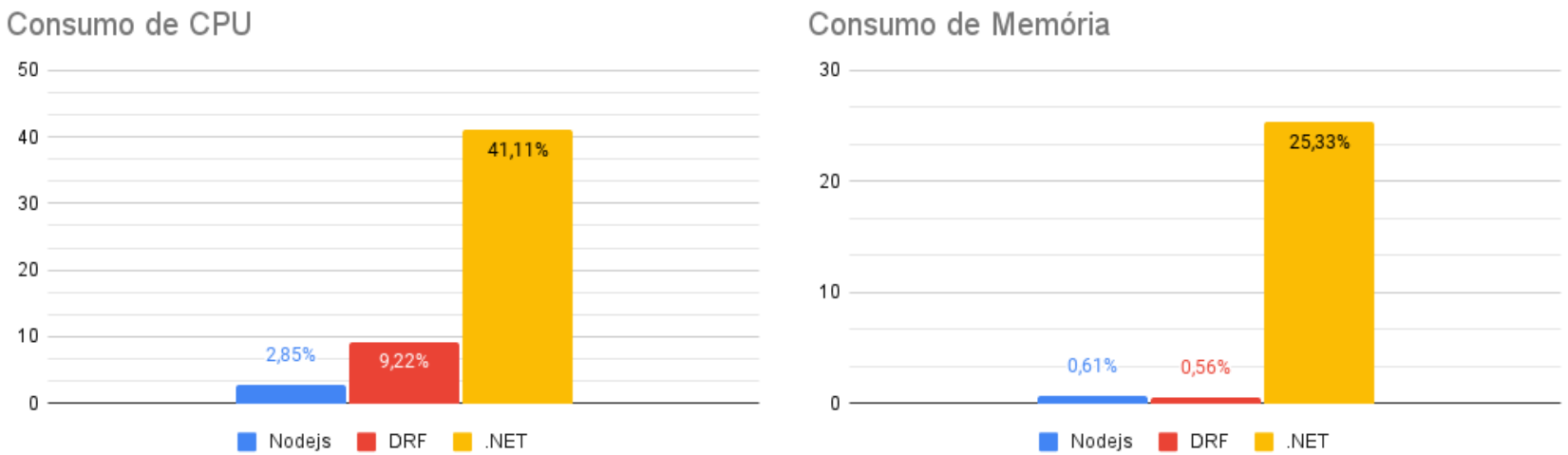
Resultados - teste de carga crescente

QP5: Qual é o *framework* mais otimizado com relação ao tempo de resposta no cenário de teste de carga crescente?

O .Net apresentou os melhores resultados em tempo de resposta em todos os métodos no cenário de carga crescente.

Resultados - teste de resistência

Figura: Consumo médio de recursos - teste de resistência



Fonte: O autor.

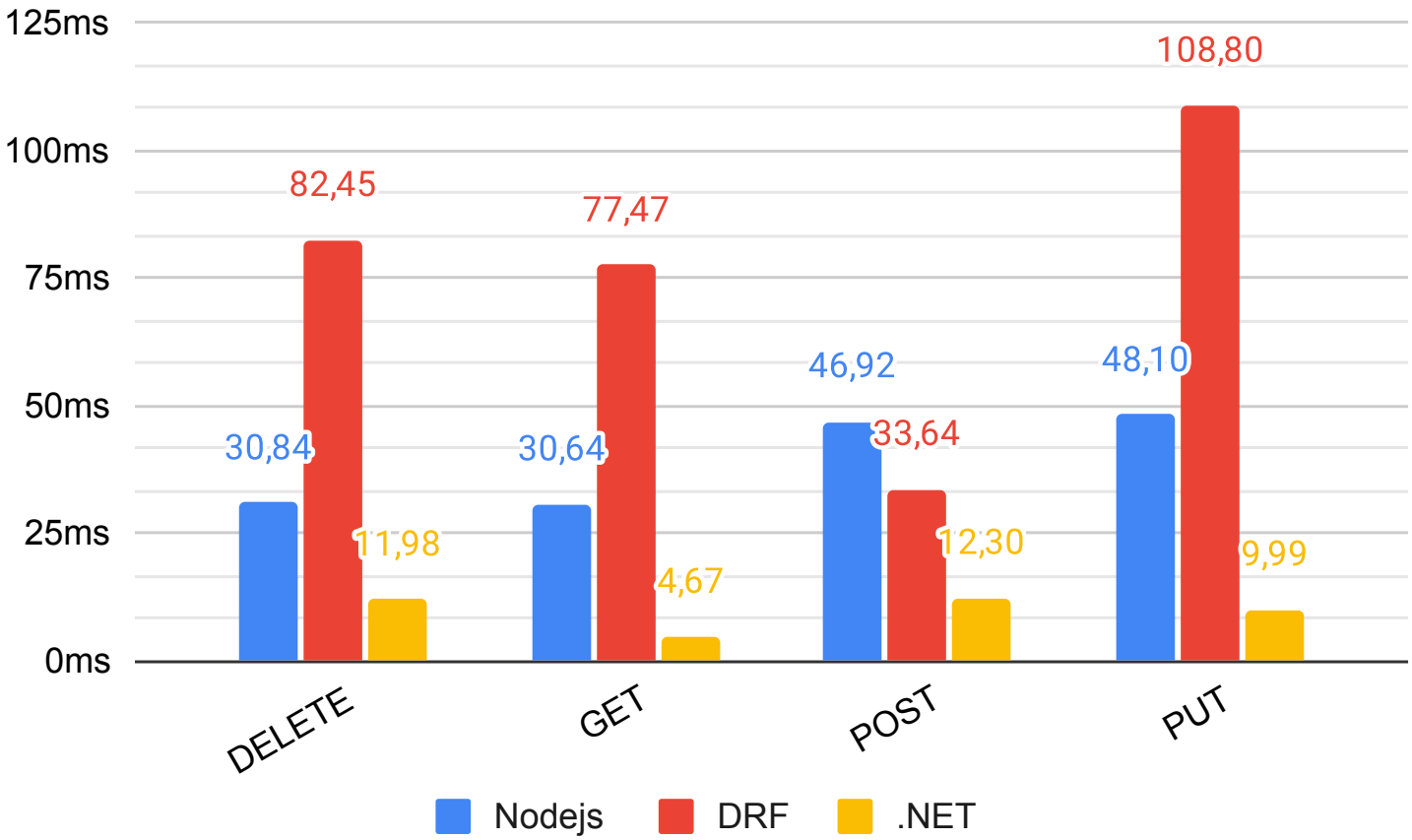
Resultados - teste de resistência

QP6: Qual é o *framework* mais otimizado com relação ao consumo de recursos no cenário de teste de resistência?

Assim como na QP2, o Node.js apresentou menor consumo de CPU enquanto o DRF mostrou melhor resultado no consumo de RAM.

Resultados - teste de resistência

Figura: Tempo de resposta médio dos métodos HTTP - teste de resistência



Fonte: O autor.

Resultados - teste de carga crescente

QP7: Qual é o *framework* mais otimizado com relação ao tempo de resposta no cenário de teste de resistência?

O .Net apresentou os melhores resultados em tempo de resposta em todos os métodos no cenário de resistência.

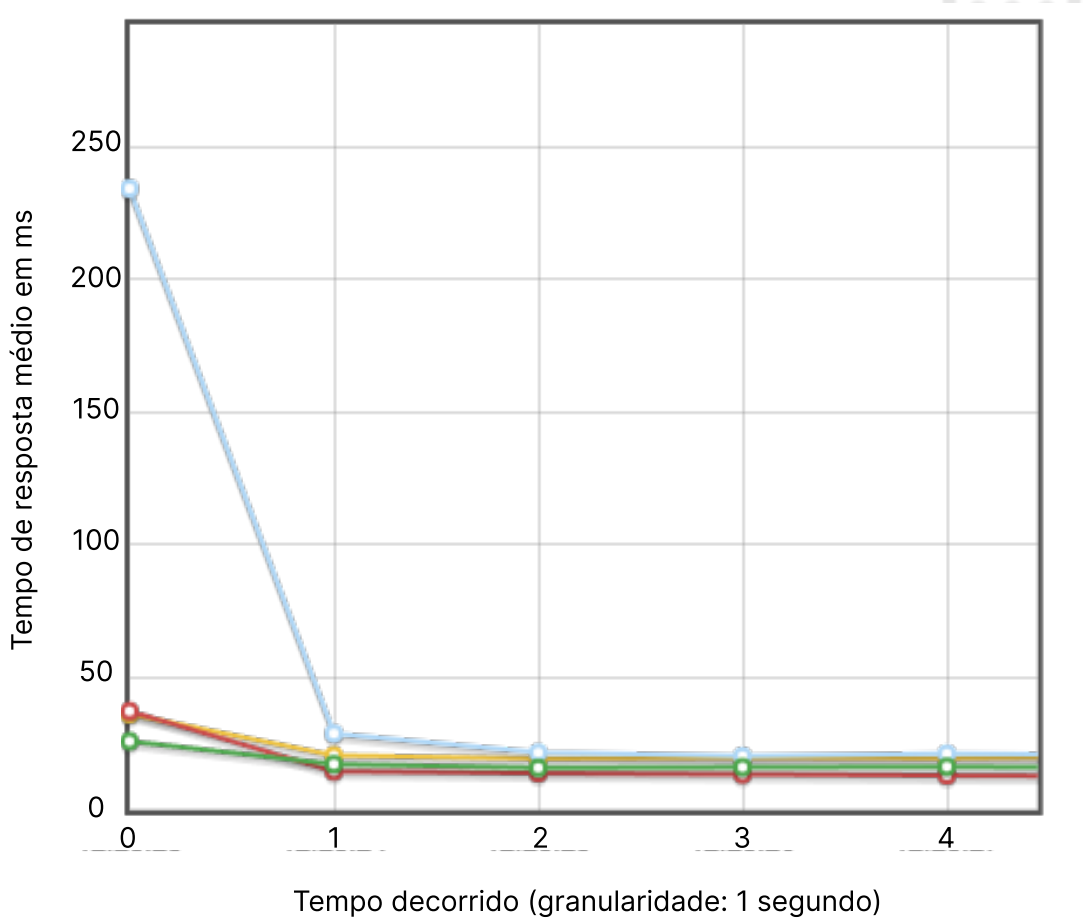
Resultados - outras observações

Taxa de erros de requisições

- Foi observado, durante os testes práticos, que nenhum dos *framework* demonstrou falhas nas requisições. Em todos os cenários avaliados, a taxa de erros de requisição foi igual a 0. Com estes resultados conclui-se que, para os cenários modelados, os *framework* possuem bom rendimento.

Resultados - outras observações

Figura: Inicialização do .Net no teste de carga crescente



Fonte: O autor.

Contribuições

O presente estudo foi submetido em formato de artigo no SBSI 2024

- XX Simpósio Brasileiro de Sistemas de Informação;

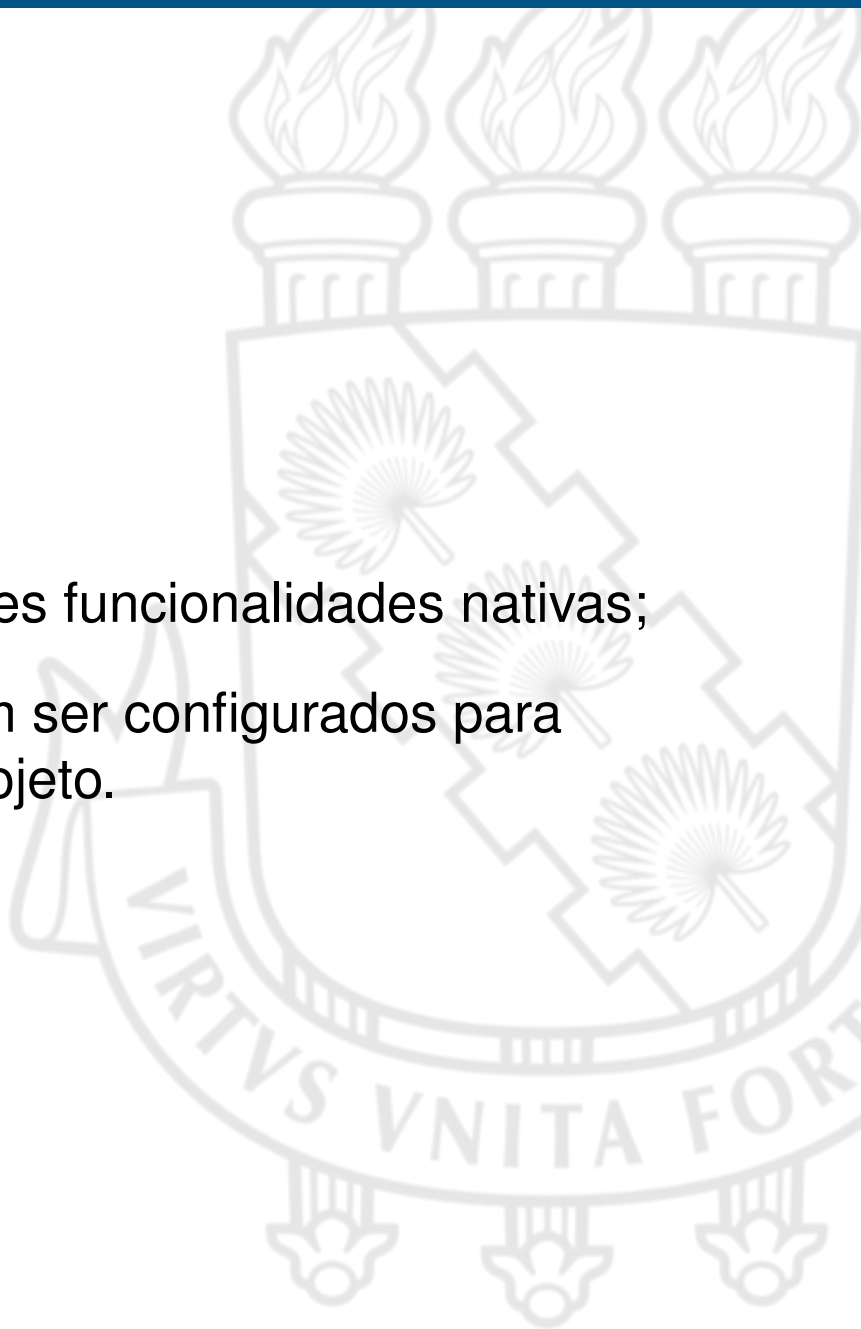
Figura: SBSI 2024

Fonte: Simpósio Brasileiro de Sistemas de Informação

Conclusões e trabalhos futuros

Conclusões comparação teórica

- Cada *framework* apresenta diferentes funcionalidades nativas;
- Apesar das diferenças, todos podem ser configurados para adequar às necessidades de um projeto.



Conclusões e trabalhos futuros

Conclusões comparação prática

- Nenhum *framework* apresentou erros de requisição;
- Node.js e Django REST Framework com melhores resultados em consumo de recursos;
- .Net Core apresentou melhores resultados com relação ao tempo de resposta.

Conclusões e trabalhos futuros

Trabalhos futuros

- Utilização de outros bancos de dados;
- Reaplicação dos testes em outros cenários;
- Reaplicação dos testes simulando outras funcionalidades.

Referências I

[1] StackOverflow. **Stack Overflow Annual Developer Survey**. Acessado em 14-abril-2023. 2022. URL: https://survey.stackoverflow.co/2022/#technology-most-popular-technologies.

[2] V.R. Basili e H.D. Rombach. "The TAME project: towards improvement-oriented software environments". Em: **IEEE Transactions on Software Engineering** 14.6 (1988), pp. 758–773. DOI: 10.1109/32.6156.

[3] Node.js. **Documentação do Node.js**. Acessado em 20-abril-2023. 2023. URL: https://nodejs.org.

[4] Django. **Documentação do Django REST Framework**. Acessado em 20-abril-2023. 2023. URL: https://www.django-rest-framework.org/.

[5] Microsoft. **Documentação do .Net**. Acessado em 20-abril-2023. 2023. URL: https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-8.0.

Referências II

[6] Zhen Ming Jiang e Ahmed E. Hassan. “A Survey on Load Testing of Large-Scale Software Systems”. Em: **IEEE Transactions on Software Engineering** 41.11 (2015), pp. 1091–1118. DOI: 10.1109/TSE.2015.2445340.

[7] Apache JMeter. **Documentação do Apache JMeter**. Acessado em 13-setembro-2023. 2023. URL: https://jmeter.apache.org/.

Obrigado(a) pela Atenção!

Contato:

klayver@alu.ufc.br

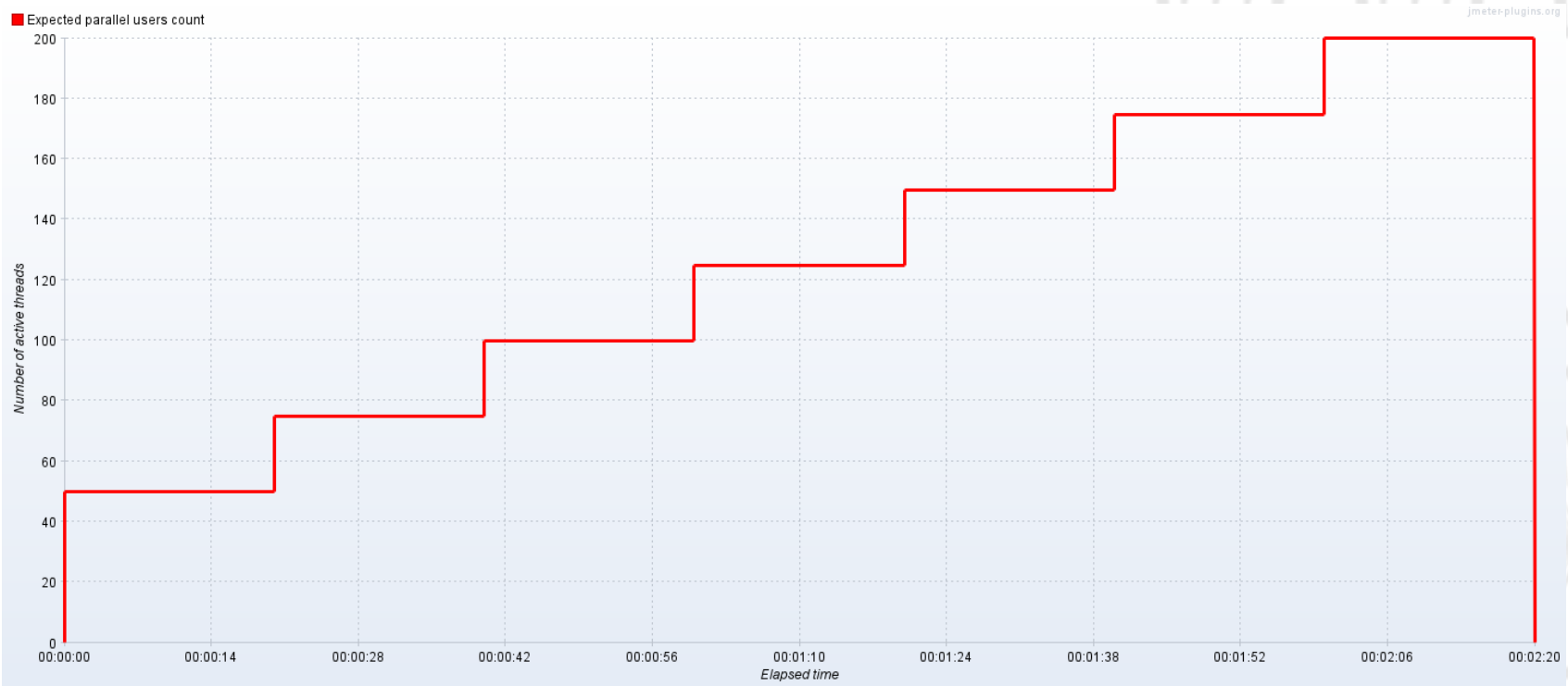
UNIVERSIDADE FEDERAL DO CEARÁ

UNIVERSIDADE FEDERAL DO CEARÁ

Figura: Modelagem do teste de pico

Fonte: O autor.

Figura: Modelagem do teste de carga crescente

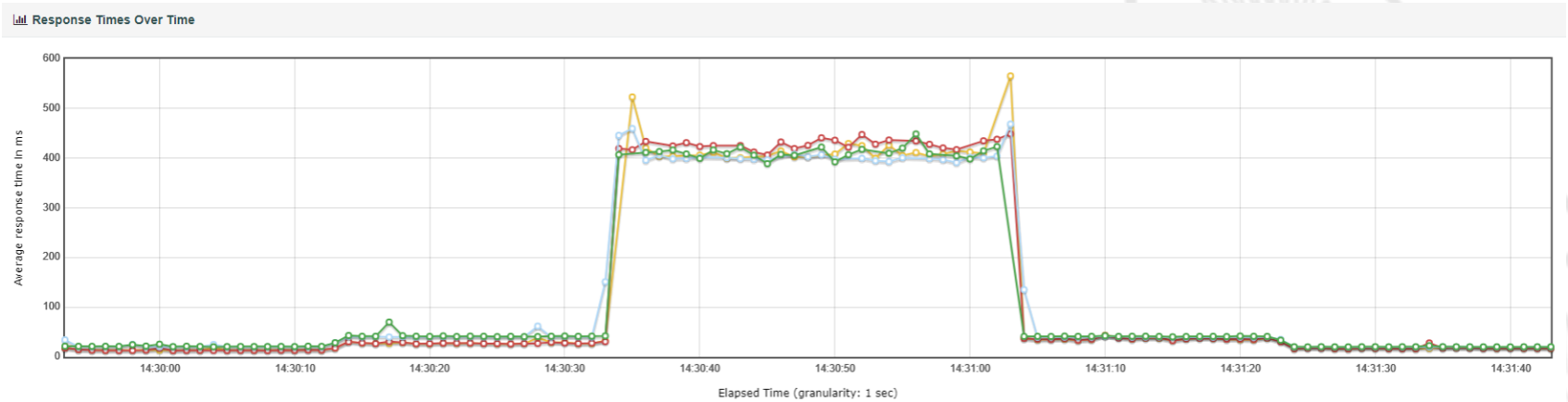


Fonte: O autor.

Figura: Modelagem do teste de resistência

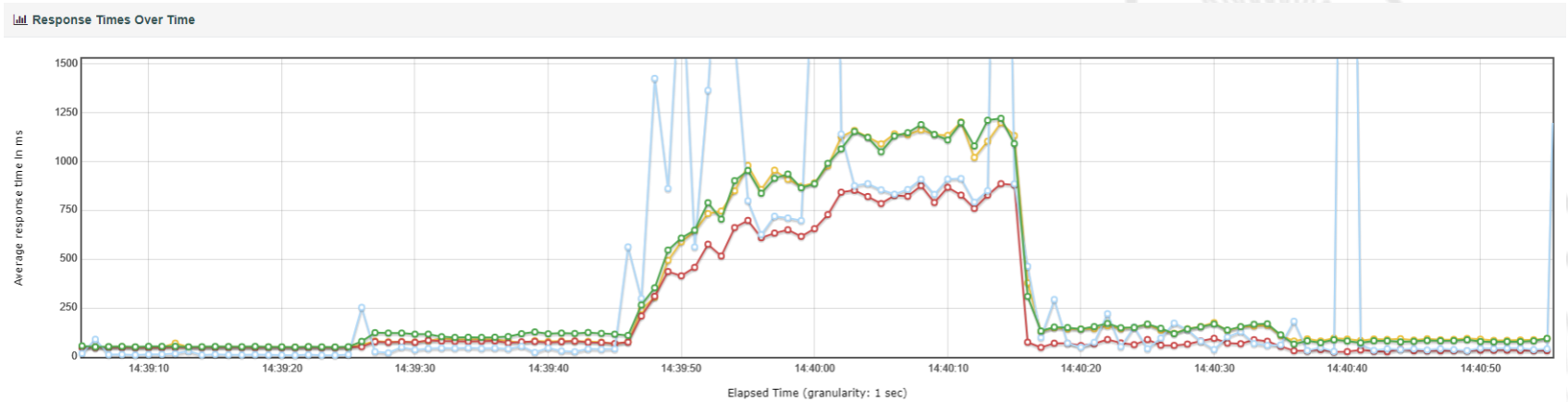
Fonte: O autor.

Figura: Tempo de resposta Node.js



Fonte: JMeter

Figura: Tempo de resposta DRF



Fonte: JMeter

Figura: Tempo de resposta .Net

Fonte: JMeter