

Impacto da Linguagem de Programação no Tempo de Resposta de uma API

Larissa Galvão Barcelos^{1*}; Heinrich Da Solidade Santos²

¹ Engenheira Eletricista. Condomínio RK Conjunto Antares Quadra M Casa 6; 73252-200 Brasília, DF, Brasil

² Mestre pelo Programa de Pós-graduação em Ensino de Ciências e Matemática. Brasil

*autor correspondente: larissabarcelos2001@gmail.com

Impacto da Linguagem de Programação no Tempo de Resposta de uma API

Resumo

Este trabalho teve como objetivo analisar o desempenho das linguagens de programação Python, Java e TypeScript no desenvolvimento de APIs, com foco no tempo de resposta de endpoints REST. A escolha das linguagens baseou-se na sua alta demanda no mercado de trabalho. Para isso, realizou-se um experimento com testes controlados utilizando dados simulados. Foram criados endpoints em cada linguagem para simular o cadastro de clientes, com aumento gradual de complexidade: desde a simples gravação de dados no banco, passando pela validação de CPF com serviço externo, até o envio de notificações assíncronas e o processamento paralelo de múltiplos cadastros. O desempenho foi medido por ferramentas como Insomnia, permitindo a comparação precisa dos tempos de resposta entre as linguagens. Os resultados indicaram diferenças significativas conforme a complexidade da operação, fornecendo subsídios para decisões técnicas na escolha de linguagem conforme os requisitos de desempenho de APIs.

Palavras-chave: desempenho; APIs REST; linguagens de programação; tempo de resposta.

Introdução

O avanço tecnológico observado nos últimos anos vem evidenciando cada vez mais a importância do desenvolvimento de software para a sociedade, sendo possível afirmar que a geração atual vive na era do software devido ao papel essencial que esse desempenha para a manutenção do modo de vida vigente, assim como afirmado por Gallotti (2016). Nesse contexto, as empresas se veem na necessidade de incorporar recursos tecnológicos em seus processos para se adequarem às necessidades do mercado, uma vez que a adoção de novas tecnologias deixou de ser um diferencial e passou a ser um requisito fundamental para sua competitividade.

Em virtude disso, desenvolvedores enfrentam, a cada nova demanda, o desafio de selecionar os recursos tecnológicos mais adequados para a construção de softwares que atendam às especificações e expectativas das empresas. Nesse sentido, a escolha apropriada desses recursos pode fornecer ao código gerado diversos benefícios, como maior facilidade de extensão e manutenibilidade, segurança contra ataques cibernéticos, alta disponibilidade e melhor desempenho do serviço. Ademais, dentre alguns recursos que podem afetar esses aspectos, destacam-se infraestruturas de computação em nuvem, bancos de dados, frameworks, metodologias ágeis, ferramentas de automação e, inevitavelmente, a linguagem de programação.

Nesse sentido, é destacada a importância que a linguagem de programação pode exercer na qualidade e eficiência do software produzido. Da mesma forma, Melo (2012) afirma a relevância para o desenvolvedor, como profissional, de entender os princípios e fundamentos de linguagens de programação. Esses conhecimentos possibilitam a

identificação de características específicas de cada linguagem, o que contribui para o desenvolvimento de programas mais adequados ao contexto em que serão implementados.

Entretanto, atualmente, o desenvolvimento de software pode ser desdobrado em diversos subtópicos de análise. Dentre estes, destaca-se o desenvolvimento de Application Programming Interfaces [APIs] para provimento de endpoints Representational State Transfer [REST] que, de acordo com Saudate (2021), consiste em um método de comunicação entre dois sistemas sem que a implementação interna de cada um deles interfira nessa intercomunicação. Notadamente, essa importância é dada pois esse processo é amplamente utilizado por diversos sistemas complexos, viabilizando tanto o fornecimento de informações quanto a atualização de dados em diferentes plataformas.

Nesse contexto, assim como analisado por Pasunoori (2025), o setor de gerenciamento de APIs tem apresentado um crescimento exponencial nos últimos anos, impulsionando crescente adoção de arquiteturas baseadas em microsserviços e evidenciando o papel central que esse segmento desempenha na comunidade de software atual. Esse cenário reforça a relevância da escolha da linguagem de programação no desenvolvimento de APIs, pois essa decisão pode impactar diretamente a experiência do usuário que usufruirá do código gerado.

Dessa forma, o propósito geral de programar sistemas de suporte é fazer com que seja possível reproduzir software mais rápido e com a mesma mão de obra, sem degradação, e possivelmente com alguma melhoria na qualidade do software, pensamento esse, refletido por Kennedy et al. (2004). Seguindo essa afirmação, o objetivo deste trabalho foi analisar o desempenho das linguagens de programação Python, Java e TypeScript no desenvolvimento de APIs, com foco no tempo de resposta de endpoints REST.

Material e Métodos

Este estudo adotou o método de pesquisa experimental, que se caracteriza pela manipulação de variáveis controladas em um ambiente controlado, com o objetivo de observar seus efeitos sobre um resultado específico assim como afirmado por Gil (2008). No contexto desta proposta, a variável manipulada corresponde à linguagem de programação utilizada (Python, Java e TypeScript), enquanto a variável observada é o tempo de resposta de endpoints REST. Essa abordagem metodológica foi escolhida porque permite estabelecer comparações diretas e objetivas entre as linguagens, garantindo que as diferenças observadas sejam atribuídas de forma mais precisa ao fator em análise, reduzindo a influência de elementos externos.

A partir desse delineamento, a pesquisa experimental foi estruturada em testes controlados, conduzidos em condições equivalentes, onde apenas a linguagem de programação foi alterada a cada cenário. Para assegurar neutralidade e evitar vieses, foram utilizados dados simulados, cuja veracidade não afeta os resultados, além de garantir conformidade com princípios éticos por não envolver informações sensíveis. Essa configuração possibilitou a observação sistemática do comportamento das três linguagens diante de diferentes níveis de complexidade, desde operações simples de banco de dados até o processamento paralelo.

Além disso, para garantir a consistência das comparações, todas as implementações foram executadas em um mesmo ambiente de testes, configurado de forma padronizada quanto a infraestrutura, banco de dados e ferramentas de monitoramento. Essa padronização foi essencial para que a variação nos tempos de resposta estivesse exclusivamente relacionada à linguagem de programação utilizada, reforçando a confiabilidade dos dados obtidos.

Dessa forma, os endpoints foram desenvolvidos considerando o cenário de uma empresa que deseja adicionar um cliente em sua base de dados. Inicialmente, as operações foram mantidas em seu nível mais simples e, de maneira progressiva, novas camadas de complexidade foram introduzidas. Os passos planejados foram os seguintes:

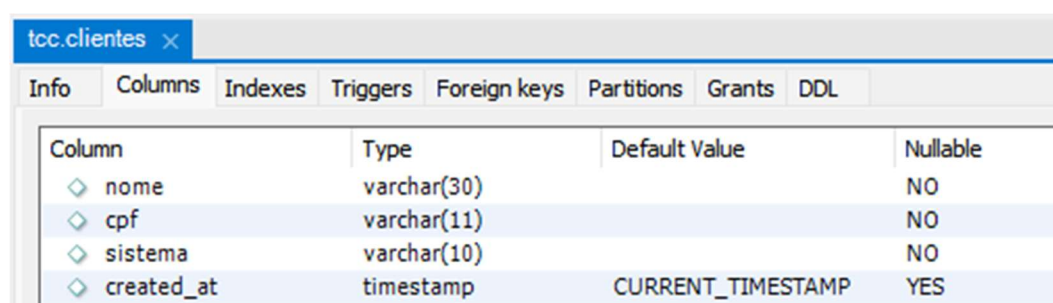
- Versão um – Criação de um endpoint básico nas três linguagens que receba o CPF de um cliente e o armazene no banco de dados. Complexidade vinculada: interação com o banco de dados, que para esse cenário foi utilizado o servidor MySQL.
- Versão dois – Aprimoração do endpoint criado para verificar se o CPF informado é válido, acionando um serviço externo. Complexidade adicionada: comunicação síncrona com outros serviços, que para esse cenário foi utilizado os endpoints públicos da invertexto.
- Versão três – Aprimoração do endpoint criado para que, após criado um novo cliente, ele seja notificado via um sistema de eventos. Complexidade adicionada: comunicação assíncrona com outros serviços, nesse caso foi escolhido o servidor do RabbitMQ para a realização dos testes.
- Versão quatro – Aprimoração do endpoint criado para que ao invés de receber apenas um CPF, seja possível informar uma lista com diversos CPFs. Complexidade adicionada: processamento em paralelo desses registros.

Para a realização das medições de desempenho e coleta de dados foi considerado o tempo de resposta necessário para fornecer a confirmação de sucesso dos clientes criados. Esse tempo foi medido com o auxílio de ferramentas como Insomnia, que auxilia na execução de requisições REST e medição do tempo de resposta.

Resultados e Discussão

Persistir um cliente sem validações adicionais

Nesta primeira etapa, o intuito foi desenvolver a versão 1 de um endpoint em que a única complexidade envolvida fosse a persistência de dados em um banco relacional. Para todas as linguagens analisadas, utilizou-se o mesmo banco de dados MySQL, previamente configurado em um servidor. Nesse banco, a modelagem foi realizada de tal forma que sua estrutura suportasse registrar o nome do cliente, seu CPF, qual linguagem realizou a inserção do registro e o momento em que esse foi registrado, como a estrutura mostrada na Figura 1.



toc_clientes			
Info	Columns	Indexes	Triggers
Foreign keys	Partitions	Grants	DDL
Column	Type	Default Value	Nullable
nome	varchar(30)		NO
cpf	varchar(11)		NO
sistema	varchar(10)		NO
created_at	timestamp	CURRENT_TIMESTAMP	YES

Figura 1. Colunas pertencentes ao banco de dados utilizado para a realização dos testes.
Fonte: Resultados originais da pesquisa

A implementação em Java, com uso do framework Spring Boot, demandou o maior tempo de desenvolvimento. A escolha por seguir boas práticas e padrões de projeto resultou em uma estrutura mais robusta e organizada, ainda que mais trabalhosa de construir. Ao testar o endpoint via Insomnia, foram realizadas dez requisições como registrado na Figura 2 para ser possível calcular a média do tempo de resposta o que resultou em 54,21 milissegundos [ms] por requisição.

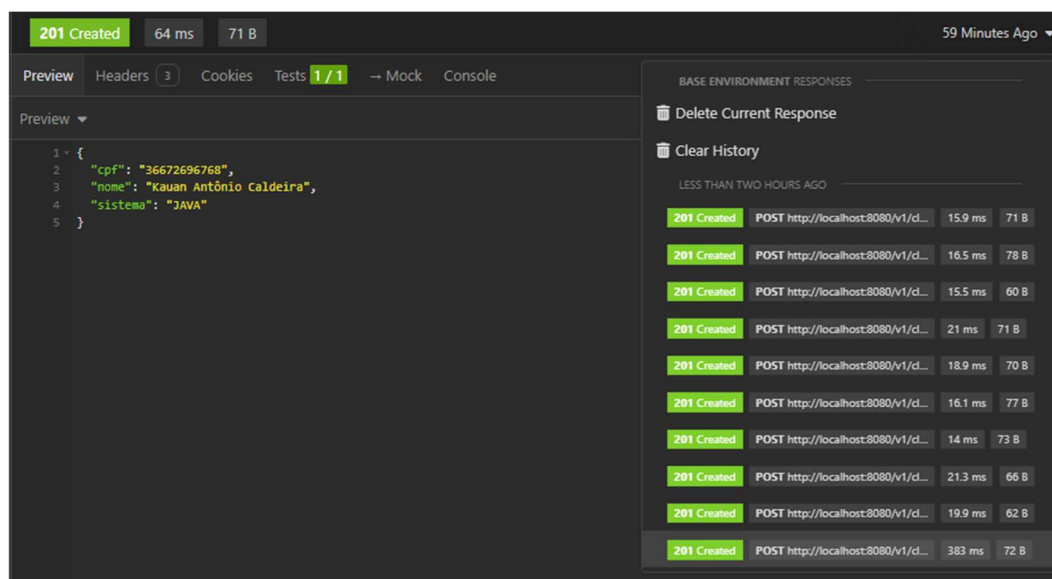


Figura 2. Tempos de resposta de dez requisições do endpoint em Java – Versão um

Fonte: Resultados originais da pesquisa

Nota: À direita estão listados os resultados, o terceiro valor é o tempo de resposta em ms

A versão em TypeScript foi construída com Node.js e Express. Embora também tenha seguido padrões estruturais adequados, sua implementação foi menos custosa em termos de tempo do que a versão em Java. O código final manteve boa legibilidade e possibilidade de extensão, com tempo médio de resposta registrado em 16,96 ms calculado como a média dos valores registrados na Figura 3.

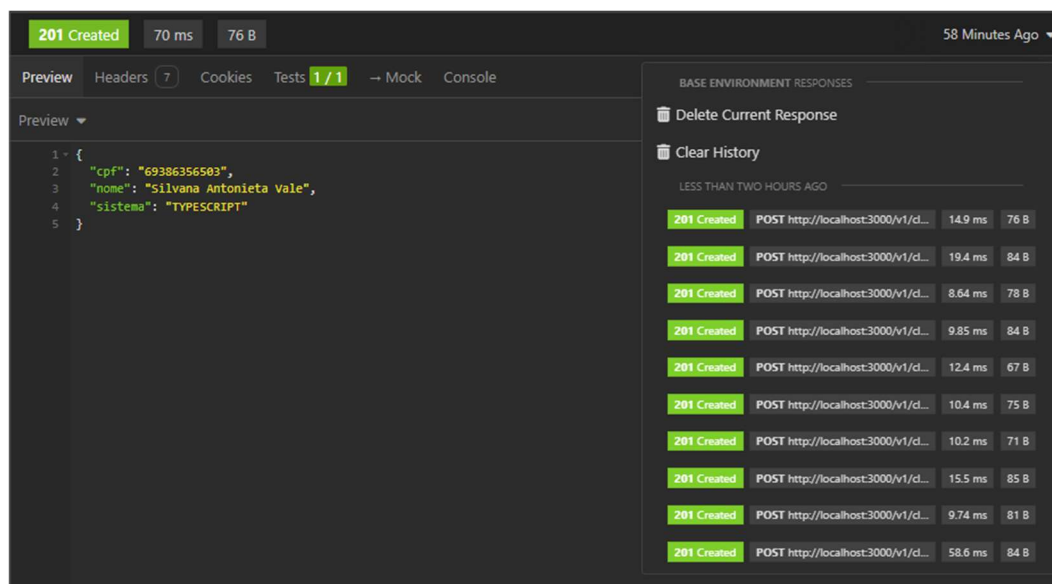


Figura 3. Tempos de resposta de dez requisições do endpoint em TypeScript – Versão um

Fonte: Resultados originais da pesquisa

Nota: À direita estão listados os resultados, o terceiro valor é o tempo de resposta em ms

Já a implementação em Python, utilizando o framework FastAPI, foi a mais ágil de desenvolver, mesmo considerando práticas de organização e padronização. No entanto, essa rapidez refletiu em uma estrutura menos robusta e, potencialmente, mais limitada em termos de manutenibilidade futura. Por outro lado, apresentou o melhor desempenho entre as três abordagens, com um tempo de resposta de 13,46 ms calculado como a média dos valores registrados na Figura 4.

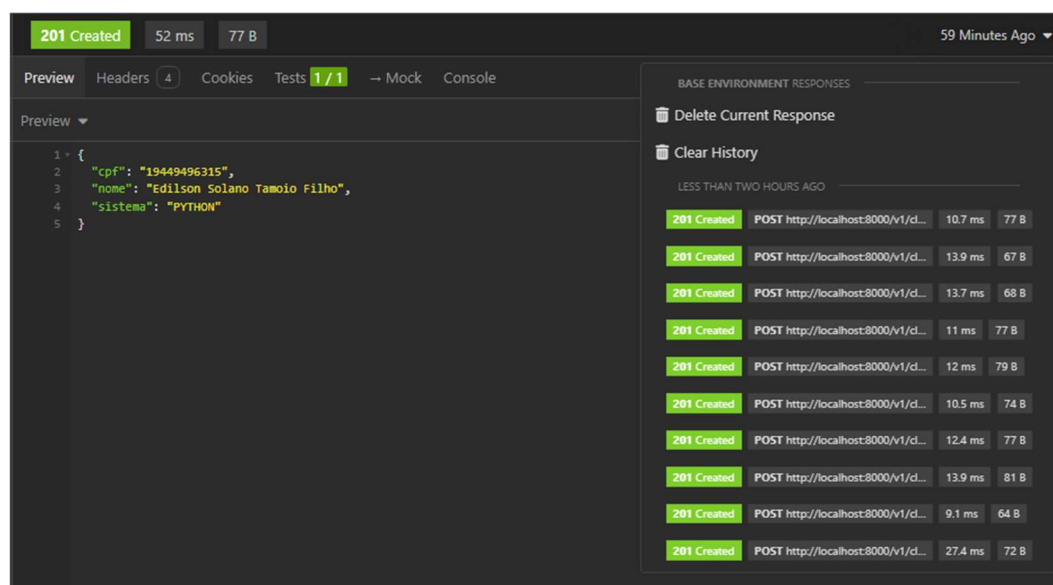


Figura 4. Tempos de resposta de dez requisições do endpoint em Python – Versão um
 Fonte: Resultados originais da pesquisa
 Nota: À direita estão listados os resultados, o terceiro valor é o tempo de resposta em ms

Dessa forma, para um endpoint em sua versão um, que a única complexidade está relacionada à integração com o banco de dados, a linguagem Python demonstrou ser a mais eficiente em termos de desempenho, seguida por TypeScript e, por fim, Java que possuiu um desempenho consideravelmente pior em comparação com as demais, como evidenciado pela Tabela 1.

Tabela 1. Comparativo de tempos de resposta – Versão um

				(continua)
EXECUÇÃO	JAVA	PYTHON	TYPESCRIPT	
1	383	27,4	58,6	
2	19,9	9,1	9,74	
3	21,3	13,9	15,5	
4	14	12,4	10,2	
5	16,1	10,5	10,4	
6	18,9	12	12,4	
7	21	11	9,85	
8	15,5	13,7	8,64	

Tabela 1. Comparativo de tempos de resposta – Versão um

EXECUÇÃO	(conclusão)		
	JAVA	PYTHON	TYPESCRIPT
9	16,5	13,9	19,4
10	15,9	10,7	14,9
TEMPO MÉDIO (milissegundos)	54,21	13,46	16,96
TEMPO TOTAL (segundos)	0,542	0,135	0,170

Fonte: Resultados originais da pesquisa

Em contrapartida, observa-se que, em contextos de maior complexidade, a facilidade de extensão e manutenção da aplicação tende a seguir a ordem inversa — sendo Java a linguagem que oferece uma estrutura mais sólida e preparada para evolução, enquanto Python, embora mais ágil na entrega inicial, pode demandar retrabalho estrutural à medida que o sistema cresce.

Persistir um cliente com validação do CPF informado

Nesta etapa, com base na estrutura previamente desenvolvida, cada implementação foi aprimorada para sua versão dois que inclui uma chamada síncrona a um serviço público externo de validação de CPF. Essa abordagem introduz uma camada adicional de confiabilidade ao sistema, garantindo que apenas CPFs válidos sejam persistidos no banco de dados. Ao validar o CPF antes da gravação, evita-se a propagação de dados incorretos e reduz-se a chance de falhas ou inconsistências em processos posteriores que dependem dessas informações, aumentando a robustez e a integridade geral do serviço de cadastro de clientes.

Para isso, foi utilizada a API da invertexto, que permite verificar a validade de números de CPF e CNPJ por meio de uma requisição REST simples. A lógica central permaneceu semelhante, mas a adição dessa dependência externa trouxe uma nova camada de complexidade: a latência da rede e a necessidade de tratar eventuais falhas de comunicação.

A versão em Java manteve boa organização estrutural, e mesmo com a inclusão da chamada externa, seu tempo médio de resposta, calculado com base nos dados evidenciados pela Figura 5, foi de 217,70 ms, demonstrando estabilidade frente ao aumento da complexidade.

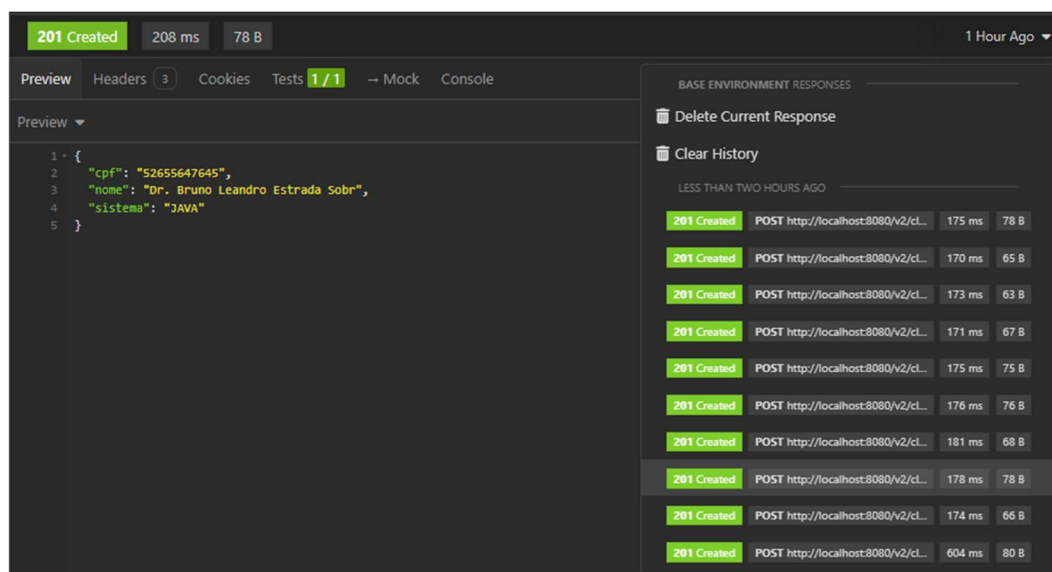


Figura 5. Tempos de resposta de dez requisições do endpoint em Java – Versão dois

Fonte: Resultados originais da pesquisa

Nota: À direita estão listados os resultados, o terceiro valor é o tempo de resposta em ms

No caso do TypeScript, a inclusão da validação foi integrada de forma relativamente simples, graças à leveza do stack utilizado. O tempo médio de resposta, calculado com base na Figura 6, foi de 225,40 ms, praticamente equivalente ao do Java, com diferenças que podem estar mais associadas a leve variações do serviço externo do que à implementação em si.

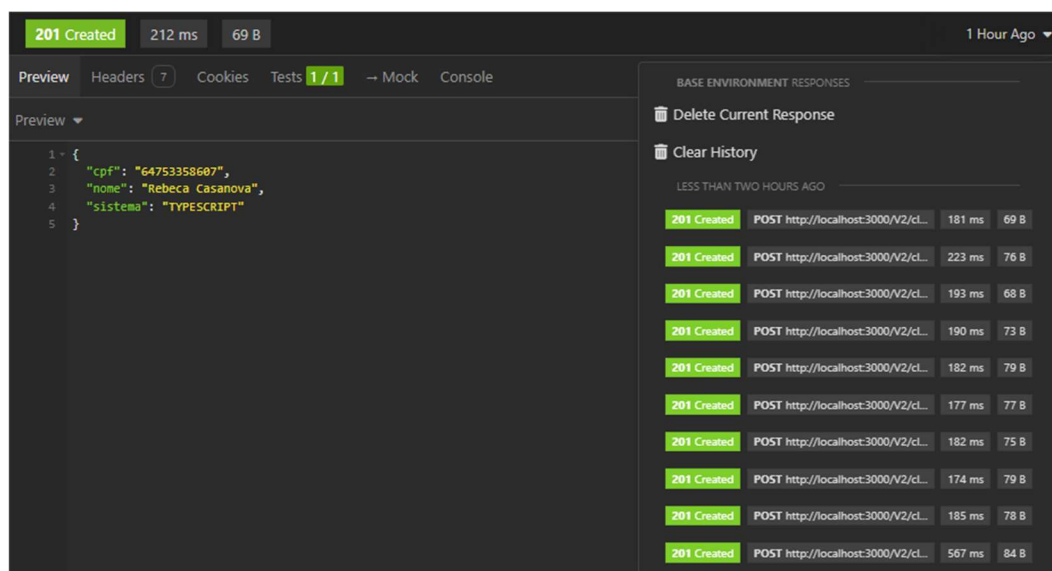


Figura 6. Tempos de resposta de dez requisições do endpoint em TypeScript – Versão dois

Fonte: Resultados originais da pesquisa

Nota: À direita estão listados os resultados, o terceiro valor é o tempo de resposta em ms

Já na versão em Python, embora a integração com a API externa tenha sido implementada sem grandes obstáculos, o tempo de resposta evidenciado na Figura 7 foi significativamente maior, chegando a 930,80 ms. Esse resultado pode indicar um impacto maior na forma como a requisição síncrona foi tratada pelo framework.

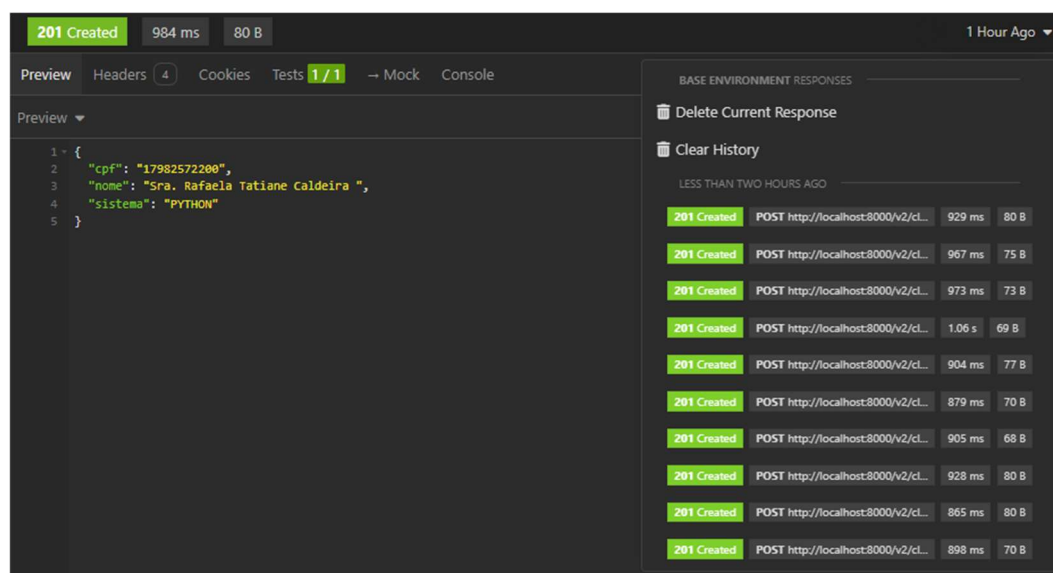


Figura 7. Tempos de resposta de dez requisições do endpoint em Python – Versão dois

Fonte: Resultados originais da pesquisa

Nota: À direita estão listados os resultados, o terceiro valor é o tempo de resposta em ms

Dessa forma, ao introduzir uma dependência externa síncrona, a eficiência de resposta se manteve equilibrada entre Java e TypeScript, enquanto Python apresentou maior sensibilidade ao aumento da complexidade, como consolidado pela Tabela 2.

Tabela 2. Comparativo de tempos de resposta – Versão 2

EXECUÇÃO	JAVA	PYTHON	TYPESCRIPT
1	604	898	567
2	174	865	185
3	178	928	174
4	181	905	182
5	176	879	177
6	175	904	182
7	171	1060	190
8	173	973	193
9	170	967	223
10	175	929	181
TEMPO MÉDIO (milissegundos)	217,70	930,80	225,40
TEMPO TOTAL (segundos)	2,177	9,308	2,254

Fonte: Resultados originais da pesquisa

Persistir um cliente com notificação de criação

Nesta etapa, com base na estrutura previamente desenvolvida, cada implementação foi aprimorada para sua versão três que inclui uma publicação assíncrona destinada a notificar sistemas interessados sobre a criação de um novo cliente.

Para isso, foi utilizada uma infraestrutura de mensageria disponibilizada via RabbitMQ, executada em um contêiner Docker previamente configurado. A lógica central de persistência do cliente foi mantida, mas com a introdução dessa nova dependência, tornou-se necessário lidar com a entrega confiável das mensagens, garantir que falhas de rede ou indisponibilidade temporária do broker não comprometam a consistência do fluxo, e considerar estratégias de re-tentativa e tolerância a falhas.

Essa abordagem introduz uma camada de desacoplamento entre o serviço de cadastro e os sistemas consumidores, permitindo maior escalabilidade e flexibilidade na integração de novos processos que dependem do evento de criação de cliente.

A versão em Java manteve boa organização estrutural, e mesmo com a inclusão da comunicação assíncrona, seu tempo médio de resposta, calculado com base nos dados evidenciados pela Figura 8, aumentou pouco com relação a versão anterior e foi de 232,30 ms, demonstrando estabilidade frente ao aumento da complexidade.

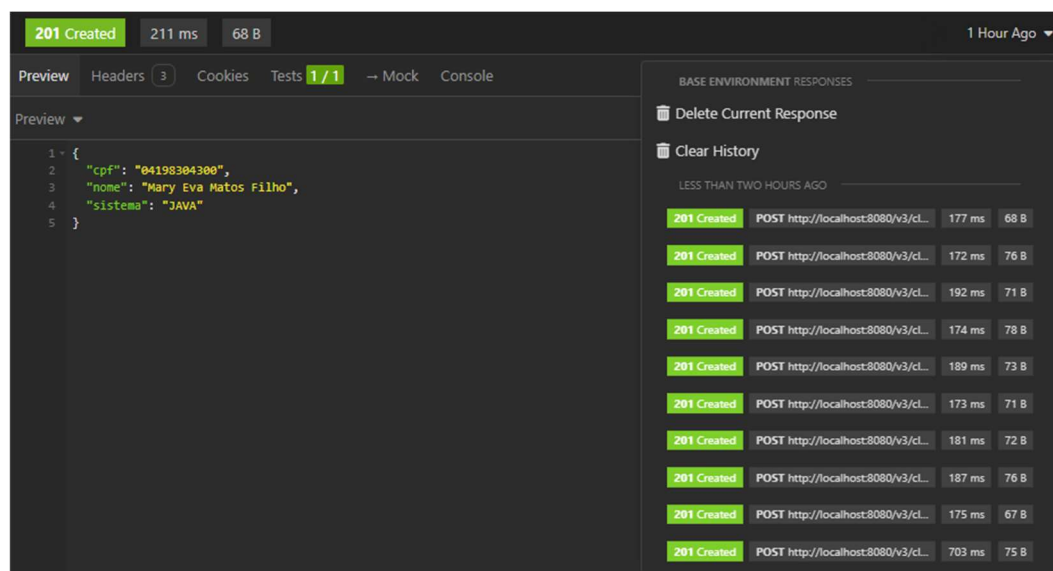


Figura 8. Tempos de resposta de dez requisições do endpoint em Java – Versão três

Fonte: Resultados originais da pesquisa

Nota: À direita estão listados os resultados, o terceiro valor é o tempo de resposta em ms

No caso do TypeScript, a inclusão da comunicação assíncrona também não resultou em grandes mudanças se comparado com sua versão anterior. O tempo médio de resposta, calculado com base na Figura 9, foi de 214,20 ms.

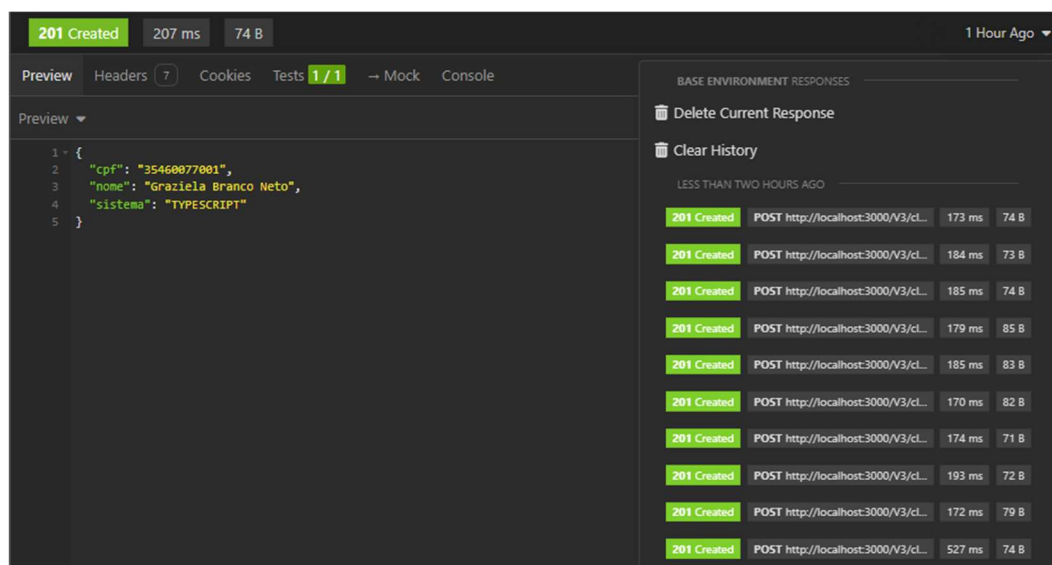


Figura 9. Tempos de resposta de dez requisições do endpoint em TypeScript – Versão três
Fonte: Resultados originais da pesquisa

Nota: À direita estão listados os resultados, o terceiro valor é o tempo de resposta em ms

Já na versão em Python, fica notável que a cada complexidade adicionada o tempo de resposta permanece crescente, nesse caso atingindo a média de 1 segundo [seg] para a criação de cada novo cliente como evidenciado na Figura 10.

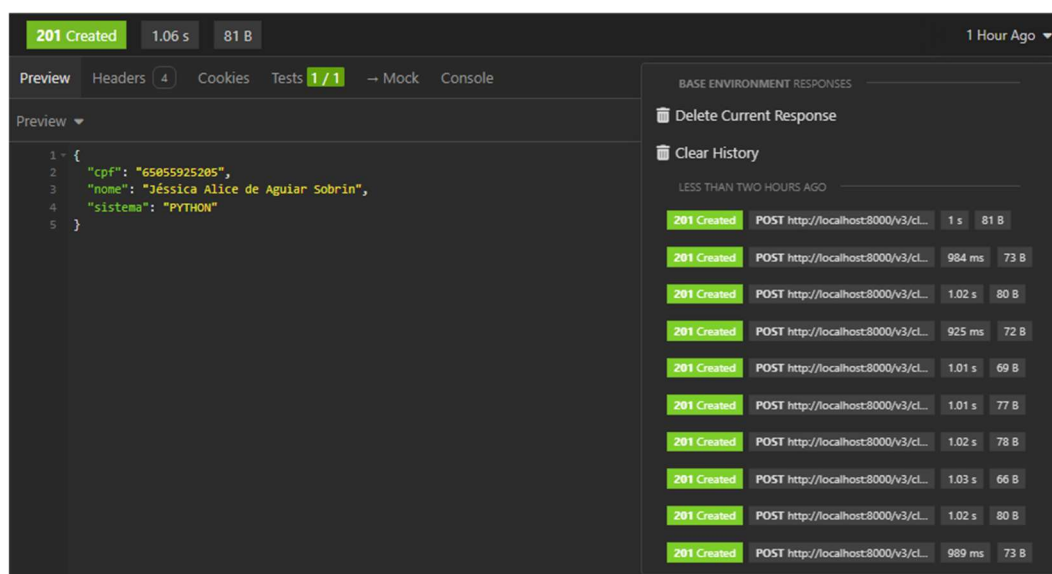


Figura 10. Tempos de resposta de dez requisições do endpoint em Python – Versão três
Fonte: Resultados originais da pesquisa

Nota: À direita estão listados os resultados, o terceiro valor é o tempo de resposta em ms

Dessa forma, ao introduzir uma integração com o RabbitMQ para a realização de comunicação assíncrona, a eficiência de resposta se manteve equilibrada entre Java e TypeScript, enquanto Python continuou a apresentar maior sensibilidade ao aumento da complexidade, como consolidado pela Tabela 3.

Tabela 3. Comparativo de tempos de resposta – Versão 3

EXECUÇÃO	JAVA	PYTHON	TYPESCRIPT
1	703	989	527
2	175	1020	172
3	187	1030	193
4	181	1020	174
5	173	1010	170
6	189	1010	185
7	174	925	179
8	192	1020	185
9	172	984	184
10	177	1000	173
TEMPO MÉDIO (milissegundos)	232,30	1000,80	214,20
TEMPO TOTAL (segundos)	2,323	10,008	2,142

Fonte: Resultados originais da pesquisa

Processamento paralelo da inserção de clientes

Nesta etapa, para que seja possível o processamento de diversos clientes simultaneamente garantindo maior agilidade no processo, foi adicionado na lógica previamente definida uma paralelização das requisições, criando assim a versão quatro dos endpoints. Dessa forma, ao invés do endpoint receber os dados de apenas um cliente, ele foi adaptado para receber uma lista com esses dados. Com isso, o processamento de cada cliente dessa lista seria feito paralelamente. Nos casos de teste efetuados os endpoints foram chamados com uma lista de cinco clientes a cada requisição.

Em quesitos de implementação, para todas as três linguagens de programação o ajuste foi relativamente simples por ter sido feito na estrutura do projeto já funcional. Já em relação aos tempos de resposta, a versão em Java conseguiu processar os cinco novos clientes em pouco mais de tempo do que sua versão anterior que processaria apenas um registro, tendo uma média de 299,20 ms como evidenciado pela Figura 11. O que mostra os benefícios positivos no tempo de processamento por registro que a paralelização do processamento pode trazer.

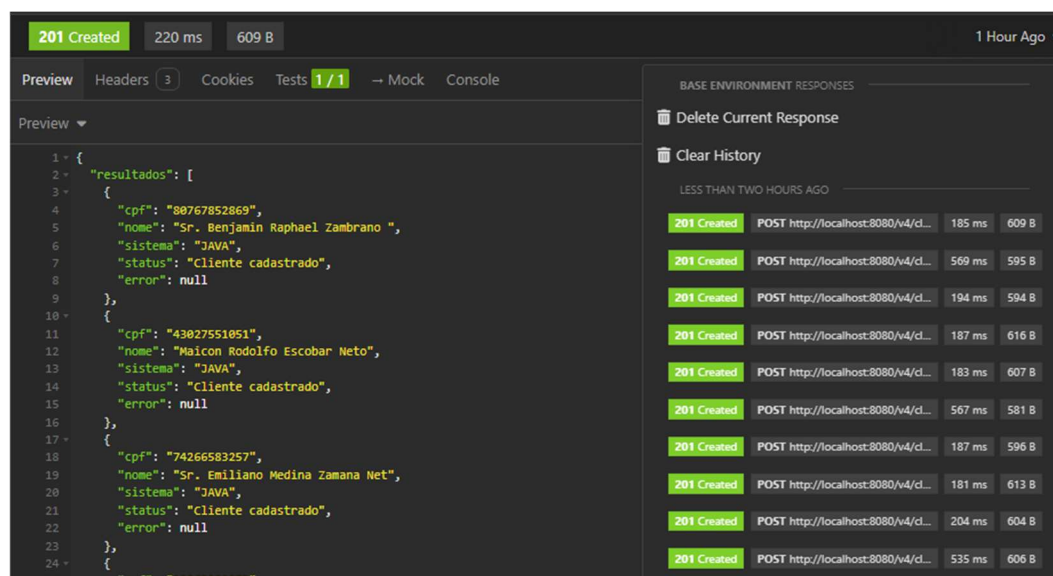


Figura 11. Tempos de resposta de dez requisições do endpoint em Java – Versão quatro
Fonte: Resultados originais da pesquisa

Nota: À direita estão listados os resultados, o terceiro valor é o tempo de resposta em ms

No caso do TypeScript, o processamento paralelo teve um pouco mais de impacto se comparado com sua versão anterior, mas ainda assim processou os cinco registros em pouco tempo a mais do que processaria apenas um. No final das contas, esses foram processados em uma média de 304,40 ms como evidenciado pela Figura 12.

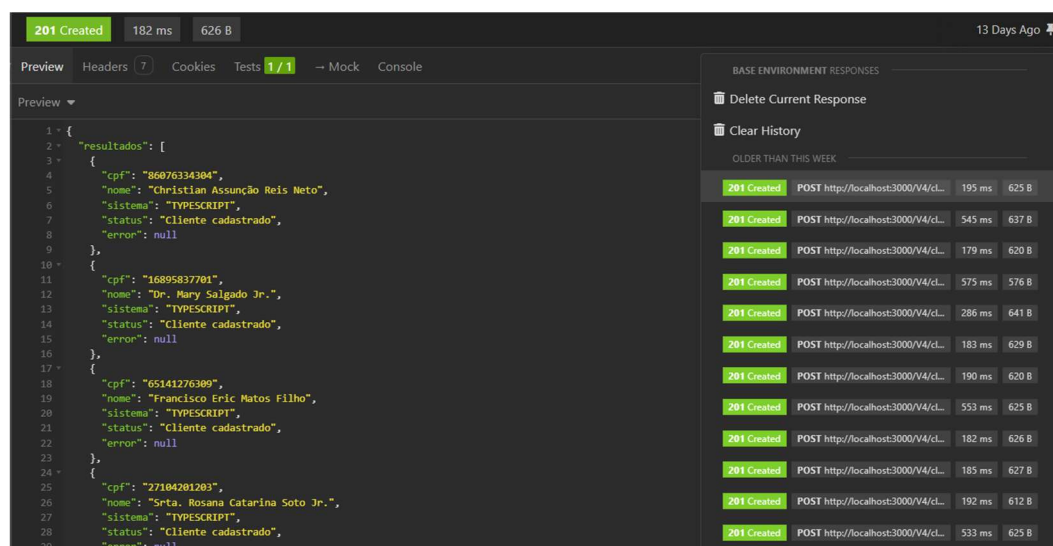


Figura 12. Tempos de resposta de dez requisições do endpoint em TypeScript – Versão quatro
Fonte: Resultados originais da pesquisa

Nota: À direita estão listados os resultados, o terceiro valor é o tempo de resposta em ms

Novamente, é notável o impacto do aumento de complexidade para o endpoint em Python que diferentemente dos demais teve um acréscimo de 1 seg se comparado com sua

versão anterior e totalizando 2,73 seg para completar o processamento de cada uma das requisições como evidenciado pela Figura 13.

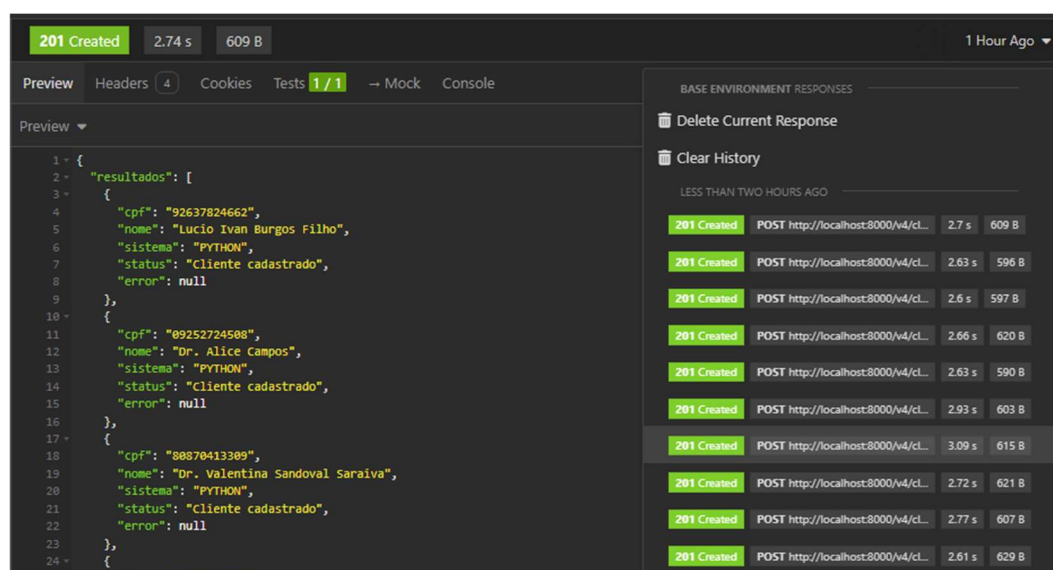


Figura 13. Tempos de resposta de dez requisições do endpoint em Python – Versão quatro

Fonte: Resultados originais da pesquisa

Nota: À direita estão listados os resultados, o terceiro valor é o tempo de resposta em ms

Dessa forma, introduzir o processamento paralelo efetivamente economiza tempo se comparado com o processamento individual, mas ainda assim tem seu peso no tempo de processamento. Esse superavit foi quase imperceptível para os endpoints em typescript e java, porém foi notável para o endpoint em Python, como consolidado pela Tabela 4.

Tabela 4. Comparativo de tempos de resposta – Versão 4

(continua)

EXECUÇÃO	JAVA	PYTHON	TYPESCRIPT
1	535	2610	533
2	204	2770	192
3	181	2720	185
4	187	3090	182
5	567	2930	553
6	183	2630	190
7	187	2660	183
8	194	2600	286
9	569	2630	545
10	185	2700	195
TEMPO MÉDIO (milissegundos)	299,20	2734,00	304,40
TEMPO TOTAL (segundos)	2,992	27,340	3,044

Fonte: Resultados originais da pesquisa

Considerações Finais

Dessa forma, o estudo permitiu identificar diferenças relevantes no desempenho entre as três linguagens analisadas no contexto do desenvolvimento de APIs REST. Ainda que os resultados não sejam suficientes para estabelecer a superioridade absoluta de uma linguagem sobre outra, ficou evidente que cada uma apresenta pontos fortes e limitações que podem variar conforme o cenário de aplicação.

Nos testes realizados, a linguagem Python apresentou maior tempo de resposta em comparação às demais, o que pode impactar negativamente a performance de endpoints em ambientes que exigem alta eficiência. Já Java e TypeScript demonstraram resultados bastante próximos entre si, sendo que o TypeScript apresentou leve vantagem tanto em desempenho quanto em facilidade de implementação.

Assim, para o caso específico aqui estudado — uma API de cadastramento de clientes com operações de validação, comunicação externa e processamento paralelo — conclui-se que a adoção de TypeScript tende a oferecer melhores benefícios ao desenvolvedor, conciliando bom desempenho com maior produtividade no desenvolvimento.

Além disso, com o intuito de favorecer a transparência e a reprodutibilidade dos resultados obtidos, os códigos desenvolvidos no estudo foram disponibilizados publicamente via GitHub, permitindo que outros pesquisadores e desenvolvedores possam reproduzir os experimentos realizados e adaptá-los a diferentes contextos. Os repositórios podem ser acessados em:

- Java: <https://github.com/LarissaGB01/sistema-cadastro-clientes-java>
- TypeScript: <https://github.com/LarissaGB01/sistema-cadastro-clientes-typescript>
- Python: <https://github.com/LarissaGB01/sistema-cadastro-clientes-python>

Por fim, destaca-se que os resultados se referem ao cenário experimental adotado, podendo variar em contextos distintos, como arquiteturas distribuídas de microserviços ou ambientes de alta concorrência.

Referências

GALLOTTI, Giocondo Marino Antonio (org.). Arquitetura de software. São Paulo: Pearson, 2016. E-book. Disponível em: <https://plataforma.bvirtual.com.br>. Acesso em: 08 mar. 2025.

GIL, Antonio Carlos. Métodos e técnicas de pesquisa social. 6. ed. São Paulo: Atlas, 2008.

KENNEDY, K.; KOELBEL, C.; SCHREIBER, R. Defining and Measuring the Productivity of Programming Languages. Int. J. High Perform. Comput. Appl., Thousand Oaks, CA, USA, v.18, p.441-448, November 2004.

MELO, Ana Cristina Vieira de. Princípios de linguagem de programação. São Paulo: Editora Blucher, 2003. E-book. p.16. ISBN 9788521214922. Disponível em: <https://integrada.minhabiblioteca.com.br/reader/books/9788521214922/>. Acesso em: 08 mar. 2025.

PASUNOORI, VijayKumar. EMERGING TRENDS IN API GATEWAYS FOR CLOUD MICROSERVICES: A TECHNICAL DEEP DIVE. Technology (IJRCAIT), v. 8, n. 1, 2025.

SAUDATE, Alexandre. APIs REST: seus serviços prontos para o mundo real. São Paulo, SP: Casa do Código, 2021. E-book. Disponível em: <https://plataforma.bvirtual.com.br>. Acesso em: 09 mar. 2025.