INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
DO SUDESTE DE MINAS GERAIS

CAMPUS RIO POMBA

ESPECIALIZAÇÃO EM DESENVOLVIMENTO WEB / MOBILE

Denis Pereira Raymundo

GENERICMICROSERVICEDPR: uma alternativa para a padronização de microsserviços a partir de um interpretador de códigos-fontes armazenados em repositórios diversos.

RIO POMBA

2021

Denis Pereira Raymundo

GENERICMICROSERVICEDPR: uma alternativa para a padronização de microsserviços a partir de um interpretador de códigos-fontes armazenados em repositórios diversos.

Trabalho de Conclusão de Curso apresentado ao Departamento Acadêmico de Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais, Campus Rio Pomba, como parte dos requisitos para obtenção do título de Especialista em Desenvolvimento Web / Mobile.

Orientador: Prof. MSc. Gustavo Henrique da Rocha Reis

Rio Pomba

2021

Ficha Catalográfica elaborada pela Biblioteca Jofre Moreira – Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais / Campus Rio Pomba
Bibliotecária: Tatiana dos Reis Gonçalves Ferreira - CRB 6/2711

R153g
Raymundo, Denis Pereira.

GenericMicroServiceDPR: uma alternativa para a padronização de microsserviços a partir de um interpretador de códigos-fontes armazenados em repositórios diversos. / Denis Pereira Raymundo. – Rio Pomba, 2021.
117f.: il.
Orientador: MSc Gustavo Henrique da Rocha Reis.

Trabalho de Conclusão de Curso em Pós-graduação *lato sensu* - Especialização em Desenvolvimento Web e Mobile. - Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais - Campus Rio Pomba.

1. Sistemas distribuídos. 2. Framework. 3. Códigos-fontes. 4. Repositórios. I. REIS, Gustavo Henrique da Rocha (Orient.). II. Título.
CDD: 004.67

Denis Pereira Raymundo

GENERICMICROSERVICEDPR: uma alternativa para a padronização de microsserviços a partir de um interpretador de códigos-fontes armazenados em repositórios diversos.

Trabalho de Conclusão de Curso apresentado ao Departamento Acadêmico de Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais, Campus Rio Pomba, como parte dos requisitos para obtenção do título de Especialista em Desenvolvimento Web / Mobile.

Aprovado em: 04/06/2021.

BANCA EXAMINADORA

Prof. Doutor Lucas Grassano Lattari
IF SUDESTE MG

Prof. Doutor José Rui Castro de Sousa
IF SUDESTE MG

Prof. MSc. Gustavo Henrique da Rocha Reis
IF SUDESTE MG

Dedico este trabalho à minha esposa, Analee Marie, e aos meus filhos, Júlia e Miguel, presentes preciosos concedidos por Deus!

Aos meus pais, Maria Joana e Sebastião, pelo carinho e amor sem medida!

AGRADECIMENTOS

A Deus que sempre tem me dado condições e oportunidades necessárias e suficientes para conquistar meus objetivos dentro dos seus planos.

Aos meus pais, por estarem sempre orando por mim e dedicando seu amor.

À minha esposa e meus filhos, pela paciência ao ceder parte do meu tempo que deveria ser dedicado a eles.

Ao Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais pela criação e manutenção desta especialização.

Ao coordenador do curso de pós-graduação em Desenvolvimento Web/Mobile pelo empenho na produção de um curso de alta qualidade.

Aos meus professores que me fizeram avançar na ciência e tecnologias usadas no desenvolvimento Web e Mobile.

Aos meus colegas que durante o curso sempre deram apoio e auxílio na solução de dúvidas.

À Tek-System Informática, empresa onde trabalho há vários anos e tem sido uma das maiores fontes de conhecimento e experiências enriquecedoras que já experimentei. Onde posso converter em sistemas computacionais as ideias que costumam borbulhar em minha mente enquanto não são colocadas em prática.

RESUMO

Desenvolvimento de um *framework* (chassi) de microsserviços que atende requisições web e realiza interpretação de codificações em *Object Pascal*, com possibilidade de invocar outras linguagens. Os códigos-fontes podem estar armazenados localmente ou em repositórios remotos. Objetiva atender uma demanda por microsserviços simples, padronizados, estáveis, confiáveis, escaláveis, tolerantes a falhas, com alto desempenho, monitorados e documentados. Configurando-se como uma alternativa aos atuais servidores web*,* porém utilizando *Object Pascal*, algo praticamente inexistente. Traz facilidade na orquestração de ecossistemas de microsserviços por centralizá-los em apenas um executável que pode acessar diversos bancos de dados, microsserviços e APIs. O *GenericMicroServiceDPR* é uma aplicação independente e portátil, não necessitando de instalação ou servidores web para sustentá-lo. Destacam-se como vantagens: Rápida execução; Facilidade no compartilhamento de bibliotecas de funções ou *frameworks*; Possibilidade de invocação a outras linguagens; Migrações extremamente simples de servidores; Possibilidade de inovação, atualização e correções instantâneas, sem necessidade de Integração Contínua (CI), Desenvolvimento Contínuo *(CD)* ou mesmo upload de unidades de codificação*,* pois o repositório estaria on-line; Testes integrados diretamente em ambiente de produção; Versionamento facilitado de códigos-fontes; Segurança dos códigos-fontes por não precisar estar exposto nem mesmo no servidor web; Curva de aprendizado para familiarizados com Pascal/Delphi. Resumo do funcionamento: Ao receber uma requisição com o nome de uma unidade de codificação e parâmetros necessários, localiza tal unidade de codificação em um dos repositórios pré-definidos e a interpreta resultando em *HTML*, *CSS*, *JavaScript*, *JSON*, *XML*, *CSV* etc. Atende requisições para arquivos estáticos quando não houver necessidade de interpretação. Unidades de codificação podem realizar referências a outras unidades no mesmo repositório ou em outros. Garante a escalabilidade e divisão de tarefas aceitando diversas instâncias com configurações e portas diferentes, inclusive podendo ser executado em contêineres e ter balanceamento de carga. Pode ser utilizado facilmente tanto em ambientes com poucos recursos quanto em ambientes corporativos, facilitando a migração ou divisão de funções em outras instâncias.

Palavras-chave: Microsserviços. Interpretador. Repositório de Código-Fonte. Sistemas Distribuídos. *Framework.* Servidor Web. Pascal. Delphi. Horse. JVCL.

ABSTRACT

GENERICMICROSERVICEDPR: an alternative for standardizing microservices as from an interpreter of source-codes stored in different repositories.

Developing a microservice chassis that responds to web requests and performs the interpretation of source codes in Object Pascal, with the possibility of invoking other programming languages. The source codes can be stored locally or in remote public repositories. The main objective is to meet a demand for simple, standardized, stable, reliable, scalable, fault-tolerant, high-performance, monitored and documented microservices. An alternative to current web servers, but using Object Pascal, which is practically nonexistent. It makes it easier to orchestrate microservice ecosystems by centralizing them in just one executable, which can access various databases, microservices and API. GenericMicroServiceDPR is a stand-alone and portable application, without need for installation and web servers to support it. Advantages: Fast execution; Ease of sharing functions libraries and frameworks; Possibility of calls other programming languages; Extremely simple server migrations; Possibility of innovation, updating and instant fixes, without need for Continuous Integration (CI), Continuous Deployment (CD) or even upload coding units, due to the repository has already being online; Tests can be integrated directly into the production environment; Easy versioning control; Security of source codes that do not even need to be exposed on the webserver; Learning curve for those familiar with Pascal/Delphi. Summary of operation: Upon receiving a request with the name of a coding unit and the required parameters, the application will find that coding unit in one of the pre-defined repositories and interpret it resulting in any type of information. The application will handle requests for static files when there is no need for interpretation. Coding units can make calls (references) to other units in the same repository or in others. Ensures scalability and division of tasks by accepting instances with different configurations and ports, including being able to run in containers and load balancing. This microservice chassis can be easily used both in low-resource environments and in corporate environments, facilitating the migration or division of functions in other instances.

Keywords: Microservices. Interpreter. Source-Code Repository. Distributed Systems. Web Server. Framework. Pascal. Delphi. Horse. JVCL.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS, SIGLAS E SÍMBOLOS

API	Application Programming Interface
ASP	Active Server Pages
BAT	Batch file
CGI	Common Gateway Interface
CORS	Cross Origin Resource Sharing
CRUD	Create, Read, Update and Delete
CSS	Cascading Style Sheets
CSV	Comma-separated values
ERP	Enterprise Resource Planning
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
GUID	Globally Unique Identifier
JOSE	JSON Object Signing and Encryption
JS	JavaScript
JSON	JavaScript Object Notation
JSP	Java Server Pages
JWT	JSON Web Token
MIDAS	Multi-Tier Distributed Application Services
ODBC	Open Database Connectivity
PHP	Hypertext Preprocessor
PWA	Progressive Web App
REST	Representational State Transfer
RTTI	Run-time Type Information
SLA	Service Level Agreement
SGBD	Sistema Gerenciador de Banco de Dados
SQL	Standard Query Language
SSL	Secure Sockets Layer
URL	Uniform Resource Locator
WEB	World Wide Web
XML	Extensible Markup Language
YAML	Ain’t Markup Language

SUMÁRIO

1 INTRODUÇÃO

No mundo empresarial e acadêmico, muito se tem estudado e utilizado da arquitetura de microsserviços em substituição aos sistemas monolíticos. Este novo modelo de arquitetura traz inúmeros benefícios, mas também diversas barreiras que ainda precisam ser vencidas. Ao longo deste trabalho será apresentada uma alternativa para a padronização de ecossistemas de microsserviços. Para que se possa compreender o desenvolvimento do trabalho é necessário elucidar dois conceitos principais do desenvolvimento de aplicações web ou *mobile*: *Backend* e *Frontend*.

Backend se refere àquilo que é executado no lado servidor, normalmente atendendo diversas requisições simultâneas vindas de milhares de usuários, tais como acesso/escrita em bancos de dados, processamentos complexos, autenticação, entrega de arquivos estáticos, controles de logs etc. Na maioria das vezes desenvolvido como uma API e atualmente usando o padrão REST. É a parte da infraestrutura que a equipe de desenvolvimento e de operações (*DevOps*) possuem mais controle. Poderia ser comparado aos motores e engrenagens de um veículo.

Frontend, por sua vez, se refere àquilo que o usuário final da aplicação visualiza, percebe e interage diretamente. É a interface do sistema com suas cores e estilos visuais agradáveis, que se comunica com o *backend* e realiza certos processamentos mais leves, se comparado ao *backend.* Justamente porque os recursos do lado cliente, são mais diversificados e imprevisíveis, podendo não ser tão poderosos quanto os recursos do servidor. Exemplos de processamentos no *frontend* são download de arquivos estáticos, suas junções e renderização (desenho em tela). Isto é normalmente executado pelos navegadores (*browsers*) de computadores, *tablets* ou *smartphones*. Poderia ser comparado à cor do veículo, seu design ou seus instrumentos de controle como velocímetro, volante, pedais de acelerador e freio.

A aplicação *backend* pode ser um grande sistema com milhões de linhas de código escrito em uma única linguagem (monolítico) ou pequenos serviços com funções únicas escritos em diversas linguagens (microsserviços) que se comunicam entre si de forma harmoniosa. Para o usuário final isto é sempre transparente, não tendo ele sequer a noção de como pode estar desenvolvido o servidor *backend*. Quanto mais funcionalidades um servidor monolítico possuir, maior será sua complexidade. Semelhantemente, também, devido à liberdade de utilização de

diversas linguagens e conceitos, a arquitetura de microsserviços, pode ter alta complexidade na coordenação ou orquestração destes variados microsserviços que podem facilmente chegar aos milhares.

Neste trabalho, objetiva-se criar uma plataforma que venha permitir a flexibilidade trazida pelos microsserviços, mas também uma padronização na forma de execução, sem perder a liberdade trazida pela adição de novas linguagens. Junto a isto, procura trazer a velocidade de uma linguagem compilada e a simplicidade da codificação de scripts interpretados. Aproveitando a oportunidade para apresentar uma alternativa de servidor web em Pascal com uma abordagem diferente das demais conhecidas.

Introduz-se o conceito de múltiplos repositórios de códigos-fontes visando dar liberdade de escolha ao desenvolvedor. Repositórios locais trazem o benefício da maior velocidade, enquanto repositórios remotos trazem o benefício da facilidade dos testes, correções e publicação (colocação em produção).

Com o *GenericMicroServiceDPR*, como será nomeada esta tecnologia, é possível, por exemplo, ter um microsserviço respondendo a cada uma das funções: autenticação; API REST com operações de CRUD para uma tabela ou várias, em um banco de dados ou vários; processamentos de sumarização de dados ou de inteligência artificial; servidor de arquivos estáticos; servidor de downloads; provedor de uploads; disparador de e-mails; realização de *backups* e outros processos agendados; *web scraping;* integrações; descoberta de serviços ou até mesmo um serviço que faça tudo isto. O desenvolvedor pode decidir como e quando fará o escalonamento ou divisão das funções, migrando de um monolítico para um ecossistema com diversos microsserviços em execução sem muito esforço.

No referencial teórico será realizada uma revisão sobre arquiteturas de sistemas e sobre interpretadores. Nos trabalhos relacionados serão discutidas as tecnologias *Pascal Pages for Web* (PPW), *Pascal Server Pages* (PSP) e *Servicer*. Na metodologia serão apresentadas as propostas de desenvolvimento do software*,* suas configurações, repositórios de códigos-fontes e testes. Nos resultados estarão a execução do sistema, suas rotas (*endpoints)* e configurações. Serão comentados os testes com múltiplos repositórios, com os diversos métodos HTTP, desempenho sob *stress,* geração de páginas complexas, invocação de outras linguagens e por fim empacotamento em imagem Docker e balanceamento de carga.

2 REFERENCIAL TEÓRICO

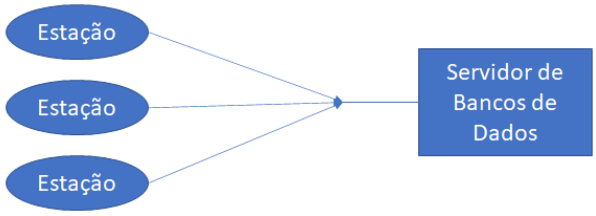
Como o objetivo principal deste trabalho se baseia em uma junção de um chassi de microsserviço com um interpretador Pascal, faz-se necessária uma revisão teórica sobre estes principais conceitos.

2.1 MICROSSERVIÇOS

Sobre a arquitetura de sistemas distribuídos, convém fazer algumas explicações:

Arquitetura Cliente-Servidor (*client/server architecture*) distribui o processamento entre um banco de dados que está em um computador servidor e um aplicativo sendo executado em outros computadores chamados de estações de trabalho, conforme pode ser visto na Figura 1. Normalmente, neste caso, as estações fazem requisições diretas ao banco de dados e realizam operações de tratamento nestes dados para transformá-los em informações (ex.: relatórios ou gráficos). Esta arquitetura se mostra ineficiente para escalar, quando a quantidade de estações cresce, devido ao gargalo de rede causado e processamento em um único servidor de bancos de dados. Requer também que cada estação tenha um relativo poder de processamento impactando diretamente nos custos com hardware.

Figura 1 - Arquitetura Cliente-Servidor – Estações acessam banco de dados.

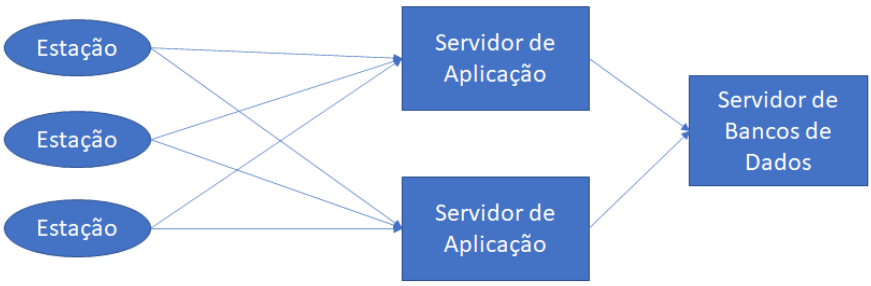


Fonte: Elaborado pelo próprio autor.

Arquitetura Multicamadas (*multitier architecture*) adiciona uma camada intermediária entre o servidor de banco de dados e as estações de trabalho. Esta camada é conhecida como servidor de aplicação ou servidor web. Este servidor de aplicação é quem faz o acesso ao Sistema Gerenciador de Banco de Dados (SGBD) e devolve informações já processadas para que as estações de trabalho possam exibi-las aos usuários. As estações não possuem mais acesso direto ao banco de dados.

Este sistema é muito utilizado hoje em dia por sites. No caso de haver um aumento da quantidade de estações pode-se aumentar o número de servidores de aplicação. Com isto, introduz-se o conceito de balanceamento de carga (*load-balance*). Esta arquitetura, apesar de melhor escalável que a anterior, também possui suas limitações. Com o aumento das necessidades inerentes aos negócios da organização, estes servidores de aplicação tendem a acumular uma grande quantidade de funções e regras dificultando a sua distribuição. Uma ilustração desta arquitetura pode ser vista na Figura 2.

Figura 2 - Arquitetura Multicamadas – Estações acessam servidor de aplicação que acessa banco de dados.



Fonte: Elaborado pelo próprio autor.

Lima (2019) fez uma revisão histórica das arquiteturas de software, apresentando alguns padrões comumente utilizados e pontuou sobre um dos grandes problemas destes sistemas, que têm sido chamados de monolíticos, que poderia ser entendido como um “faz-tudo”.

> *Conforme novas funcionalidades são adicionadas, as tecnologias vão se atualizando e novas demandas vão surgindo, o monolítico tende a ficar cada vez mais difícil de ser otimizado: adicionar funcionalidades se torna uma atividade demorada e custosa pelos bugs que aparecem em partes do sistema que não deveriam ser atingidas, fazer o deploy se torna um desafio pelo tamanho do sistema, memory leak em uma funcionalidade compromete o resultado de outra completamente diferente, e o sistema quebra (sistemas locais) ou sai do ar (sistemas web)* (LIMA, 2019).

Lima (2019) revisa ainda outras arquiteturas que buscam distribuir o processamento, como a arquitetura baseada em eventos, que faz uso de filas, canais

e processadores de eventos e a arquitetura de *microkernel,* que faz uso de um pequeno módulo central e outros módulos plugáveis *(plug-ins)* que vão adicionando funcionalidades extras. Tudo isto serviu para inspirar a arquitetura de microsserviços.

Diversos autores têm apresentado definições para a arquitetura de microsserviços as quais se complementam.

Fowler (2017) definiu microsserviço como uma pequena aplicação que executa uma única tarefa e com eficiência. Um pequeno componente que pode ser facilmente substituído e que é desenvolvido e implantado de forma independente. Ressalta, porém, que nenhum microsserviço sobrevive sozinho, pois é parte de um sistema maior, um ecossistema de microsserviços, onde eles interagem entre si para realizarem tarefas que normalmente seriam tratadas por uma grande aplicação autônoma.

> *Microsserviços são uma abordagem arquitetônica e organizacional do desenvolvimento de software na qual o software consiste em pequenos serviços independentes que se comunicam usando APIs bem definidas. Esses serviços pertencem a pequenas equipes autossuficientes. As arquiteturas de microsserviços facilitam a escalabilidade e agilizam o desenvolvimento de aplicativos, habilitando a inovação e acelerando o tempo de introdução de novos recursos no mercado (*Amazon, 2020).

Richardson (2020) apresenta microsserviços como sendo um estilo arquitetural que estrutura uma aplicação como uma coleção de serviços que são:

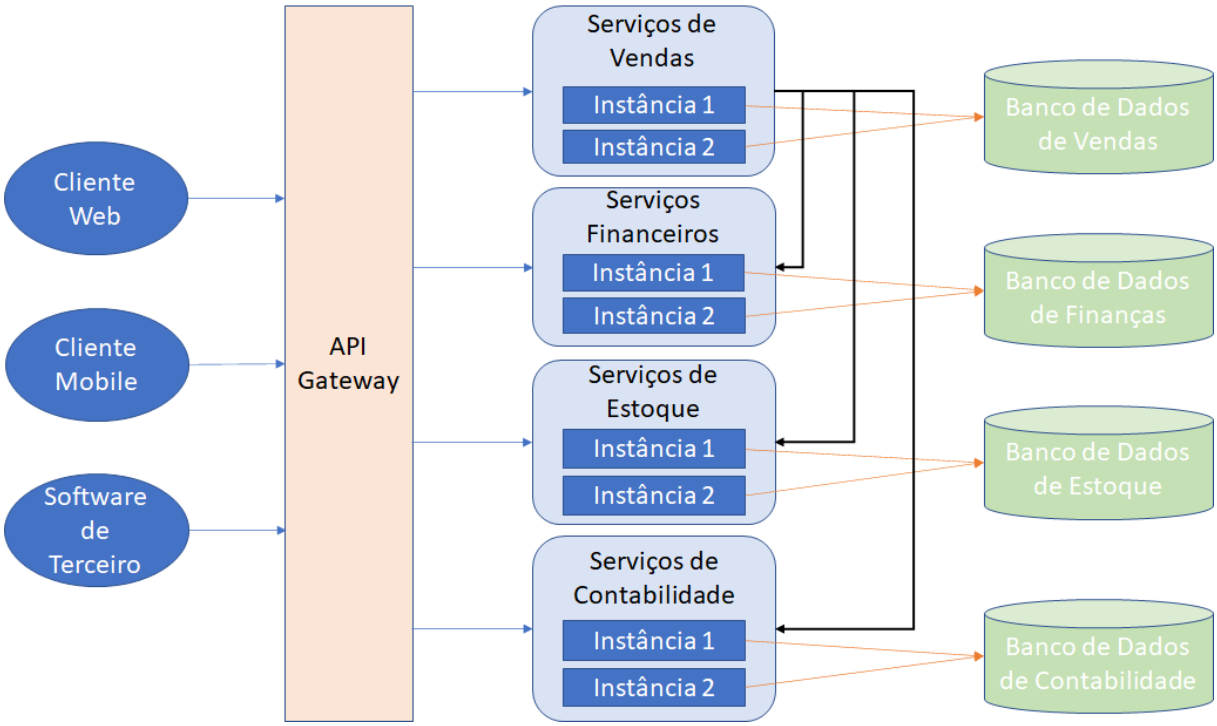
- Altamente sustentáveis e testáveis;
- Fracamente acoplados;
- Implantáveis independentemente;
- Organizado em torno de recursos de negócios;
- Propriedade de uma pequena equipe.

A arquitetura de microsserviço permite a entrega rápida, frequente e confiável de aplicativos grandes e complexos. Além disto, também permite que uma organização desenvolva sua pilha de tecnologia.

Conforme Shoup (2014), não existe arquitetura perfeita para todos os produtos em todas as escalas.

Na Figura 3 é possível perceber que um cliente web, *mobile* ou mesmo softwares de terceiros autorizados podem fazer solicitações a qualquer um dos serviços disponibilizados. Cada serviço disponibilizado pode ter uma ou mais instâncias e apenas uma delas será acionada por cada requisição. Cada instância pode se comunicar diretamente com o seu banco de dados e também com outros microsserviços.

Figura 3 - Arquitetura Microsserviços – Cada serviço com uma função e um banco de dados.



Fonte: Elaborado pelo próprio autor.

Por exemplo: o site da empresa, quando um cliente realiza uma compra, faria uma requisição para uma das instâncias do microsserviço “serviços de vendas”. Este faria o registro da venda no seu banco de dados e acionaria uma instância de “serviços financeiros” para que as duplicatas fossem registradas para o cliente. Paralelamente seria acionada uma instância do microsserviço “serviços de estoque” para realizar a baixa do estoque e uma instância do microsserviço “serviços de contabilidade” para efetuar os respectivos registros contábeis no banco de dados apropriado. Uma aplicação móvel poderia fazer uma requisição para saber o status do pedido. Um software de terceiros, por exemplo, uma transportadora, poderia fazer uma integração para saber se existem entregas disponíveis ou registrar os status das entregas.

2.2 CHASSIS DE MICROSSERVIÇOS

Uma das premissas da arquitetura de microsserviços é permitir que equipes autônomas realizem a escolha das ferramentas e linguagens que vão utilizar para construir, de forma independente, seus microsserviços. Essa liberdade, contudo, normalmente converge para um número reduzido das linguagens atuais. Mesmo assim, é comum nesse ambiente, ter-se diversas equipes procurando resolver os mesmos problemas, porém de maneiras diferentes.

Uma solução possível para evitar essa duplicação de trabalho e diminuir a redundância do sistema, segundo OPUS Software (2021), seria a adoção de um *framework* que padronizasse elementos comuns e tratasse questões gerais a todas as tecnologias, mantendo assim, o alinhamento por meio de ferramentas que disponibilizem a infraestrutura ideal para a execução dos microsserviços.

A criação de *frameworks* visa solucionar problemas recorrentes da computação usando uma abordagem genérica. Os *frameworks* propõem-se a evitar que desenvolvedores reescrevam software sempre que precisarem resolver um mesmo problema.

Concernente à arquitetura de microsserviços, esses mecanismos que lidam com questões transversais, costumam ser baseados no *Microservice Chassis Pattern*[1]. Richardson (2019) denomina *Microservice Chassis* ao conjunto de tecnologias especializadas em resolver os desafios inerentes à arquitetura de microsserviços e evitar retrabalho por parte da equipe de desenvolvimento. Conforme OPUS Software (2021), o propósito de um *Microservice Chassis,* que nada mais é do que um chassi compartilhado por vários microsserviços, é facilitar a criação de novos microsserviços minimizando os riscos envolvidos. Isso ocorre justamente porque oferecem um conjunto de padrões e funcionalidades já testadas, aprovadas e até mesmo em produção, incluindo combinações de linguagens e bibliotecas principais.

O compartilhamento de soluções prontas, bibliotecas e experiências pode representar um grande ganho para as equipes. Além de, no caso de se encontrar uma vulnerabilidade em uma biblioteca, a atualização ser unificada e disponível para todos.

Um chassi, ou *framework* de microsserviços, fornece um conjunto de padrões que devem ser respeitados por todos os colaboradores envolvidos no projeto,

[1] https://microservices.io/patterns/microservice-chassis.html

independente da equipe a qual pertença. Isso facilita a entrada de novos membros na equipe, o entendimento do código, a troca de experiências e a criação de um vocabulário comum, refletindo na melhor gestão do tempo de execução dos projetos.

Algumas funcionalidades que normalmente estão presentes nesses chassis:

- Configuração externalizada (*externalized configuration*) – por exemplo, da base de dados e de credenciais de acesso;
- Verificação de saúde (*health check*) – uma URL que um serviço monitor possa requisitar para determinar se a instância está operacional;
- Tratamento de exceções (*handle exception*);
- Registro histórico de acessos (*access log*);
- Métricas (*metrics*) – medições que forneçam uma visão sobre o que aplicativo está fazendo e como está o seu desempenho;
- Autenticação (*authentication*);
- Descoberta de Serviços (*discover*);
- Relatório de erros (*error reporting*);
- Balanceamento de carga (*load balancing*);
- Limitação de taxas (*rate limiting*).

OPUS Software (2021) acrescenta ainda que essa estrutura padronizada permite abstrair a lógica comum em outra camada. Questões relacionadas à infraestrutura, tais como conectividade, configuração e monitoramento podem ser transferidas para esse *framework*.

> *O trabalho envolvido em criar, manter e atualizar um microservice chassis pattern é capaz de trazer um enorme retorno para a organização, porque gera um setup correto, testado e reutilizável. Isso evita que você e sua equipe precisem gastar tempo na adequação dos serviços ao ambiente de execução, repetindo essa ação cada vez que um novo microsserviço for criado* (OPUS Software, 2021).

No presente trabalho foi construída uma aplicação que se enquadra na definição de *framework* de microsserviços, pois atende aos requisitos, funcionalidades e benefícios expostos sobre o *Pattern Microservice Chassis*.

2.3 INTERPRETADORES

Em informática, interpretador é um programa que traduz código de programação de alto nível em código de máquina, sem criar um arquivo executável do programa traduzido, conforme dicionário de português proporcionado pela *Oxford Languages* (Google, 2020).

Seu funcionamento pode variar de acordo com a implementação. Em alguns casos, o interpretador lê o código-fonte linha a linha e o converte em código-objeto (*bytecode*) à medida que o executa. Em outros casos, converte o código-fonte por inteiro e depois o executa (Wikipédia, 2020).

> *Interpretadores são, em geral, menores que compiladores e facilitam a implementação de construções complexas em linguagens de programação. Entretanto, o tempo de execução de um programa interpretado é geralmente maior que o tempo de execução deste mesmo programa compilado, pois o interpretador deve analisar cada declaração no programa a cada vez que é executado e depois executar a ação desejada, enquanto que o código compilado apenas executa a ação dentro de um contexto fixo, anteriormente determinado pela compilação. Este tempo no processo de análise é conhecido como "overhead interpretativa" (Farias, 2010).*

> *Os interpretadores usam suas próprias bibliotecas internas para processar o código do software: uma vez que uma linha do código-fonte foi convertida nas instruções legíveis por máquina correspondentes, ela é enviada diretamente para o processador. O processo de conversão não está completo até que todo o código tenha sido interpretado. Ele só vai parar mais cedo se ocorrer um erro durante o processamento. Isso torna a depuração muito mais fácil, pois a linha de código que contém o bug é imediatamente identificada quando o erro ocorre (SOS Digital, 2020).*

Normalmente a maioria dos servidores web atuais como PHP, ASP, NodeJS e Python realizam interpretação das suas unidades de codificação (códigos-fontes) para responder às requisições.

Na implementação proposta por este trabalho foi desenvolvida uma plataforma de microsserviços compilada que realiza a interpretação das codificações linha a linha, sem gerar *bytecodes*, semelhante às linguagens citadas acima, contudo sem perda de performance. Este microsserviço responderá como um servidor web interpretando a linguagem *Object Pascal*. Esta abordagem foi escolhida por dois motivos principais:

1) Não existe atualmente um interpretador *Object Pascal* funcionando como um servidor web sendo utilizado em larga escala. Uma iniciativa recente é o Fano Framework[2] - *Web application framework for modern Pascal programming language*.
2) Pela facilidade de se testar codificações interpretadas sem a necessidade de recompilação de toda a codificação, facilitando o tempo para desenvolvimento, mesmo que estejamos seguindo alguns padrões definidos.

[2] https://fanoframework.github.io/

3 TRABALHOS RELACIONADOS

Nesta seção serão apresentados como trabalhos relacionados duas ferramentas que procuraram se enquadrar como servidores web utilizando a linguagem Pascal e, por último, uma plataforma para desenvolvimento rápido de microsserviços em diferentes linguagens.

3.1 PASCAL PAGES FOR WEB (PPW) E PASCAL SERVER PAGES (PSP)

Werther Filho (2006) apresentou uma tecnologia baseada na linguagem *Object Pascal* para geração de páginas web dinâmicas, conjugada com um ambiente de execução semelhante ao *Common Gateway Interface (CGI)*. Com o uso desta tecnologia, denominada *Pascal Pages for Web (*PPW), o desenvolvedor poderia criar páginas dinâmicas utilizando *scriptlets*[3] para permitir a inclusão de código Pascal entre as *tags* HTML. O mestre justificou com dois motivos a utilização do Pascal: Primeiro, porque era uma linguagem muito bem estruturada, fácil de assimilar (inglês coloquial), fortemente “tipada” (*typesafe*) e que possuia divisões bem definidas das declarações. Segundo, porque era uma das linguagens mais adotadas nos primeiros semestres dos cursos de Ciência da Computação e similares. Acrescentou ainda que como não existia uma plataforma ou tecnologia suficientemente difundida em meios acadêmicos ou no mercado para a programação web baseada em Pascal, a familiaridade e os conhecimentos adquiridos nos primeiros semestres eram perdidos na migração natural dos alunos para o ambiente web. Presumiu também que o Pascal ou qualquer extensão que venha dela (como o *Object Pascal* e/ou Delphi) provavelmente possuíssem uma curva de aprendizado menor do que as outras linguagens, o que poderia tornar mais fácil a formação de programadores em tecnologias baseadas em Pascal (e similares).

A ideia principal do seu artigo foi mostrar que o PPW fornece a mesma praticidade de desenvolvimento que as aplicações web que utilizam páginas dinâmicas, como o ASP e o JSP, com a mesma rapidez dos executáveis de aplicações CGI. Werther Filho (2006) utilizou o próprio compilador do Delphi para compilar e executar os trechos interpolados e observou que a velocidade ficou superior ao JSP,

[3] Ver glossário.

por exemplo. Isto ocorreu mesmo que não tenha desenvolvido um servidor HTTP, pois se utilizou do Apache para atender às requisições.

Werther Filho (2006) citou algumas outras tecnologias dentro desta mesma linha, mas com abordagens diferentes. Dentre elas a *Pascal Server Pages* (PSP)[4], um projeto da Nemesis Pascal que também permitia a inclusão de códigos Pascal em páginas HTML através do uso de *scriptlets*. A PSP pareceu utilizar-se de compilação prévia em *bytecodes* usando o *Pascal Scripts* com posterior execução, porém sua implementação era bem simplificada e ainda dependia de um servidor web HTTP.

Na Codificação 1 é possível visualizar um exemplo que seria suportada tanto pela tecnologia PPW quanto pela PSP.

Codificação 1 - Codificação suportada pelas tecnologias PPW e PSP.

```
<html>
  <head>
    <title>This is a Pascal Server Page</title>
  </head>
  <body>
    <%
       begin
         Write('Hello World');
       end.
    %>
    <p> I am going to use Pascal Script to write a few numbers... </p>
    <%
       var I: Integer;
       begin
         for I := 1 to 10 do Writeln(I);
       end.
    %>
  </body>
</html>
```

Fonte: Elaborado pelo próprio autor.

[4] Para mais informações sobre a tecnologia PSP consulte Pascal (2012).

O presente trabalho, objetiva-se mostrar que é possível ter servidores web rápidos, porém diferentemente do trabalho de Werther Filho (2006), não foi utilizado o compilador do Delphi para realizar a interpretação, mas o componente JVInterpreter que realiza a interpretação de *Object Pascal* sem precisar de outro aplicativo de terceiro, tornando o GenericMicroServiceDPR independente. Foram, inclusive, adicionadas inúmeras funções próprias compiladas à interpretação Pascal para facilitar a codificação de aplicações web e trazer maior velocidade na execução.

Não foi adotada a mesma ideia de Werther Filho (2006) de se trabalhar com *scriptlets*, que são a substituição de trechos de códigos no meio de páginas HTML. Foi adotada uma estratégia inversa: ter a página que vai gerar o HTML toda em Pascal, mas permitindo importar outras páginas HTML puras e realizar substituições em seu conteúdo por trechos processados previamente em Pascal, conforme pode ser visualizado na Codificação 2 e Codificação 3 .

Codificação 2 - Codificação HTML_PURO - Trechos substituídos posteriormente.

```
<html>
  <head>
    <title> Microsserviço Genérico </title>
  </head>
  <body>
    <p><%Frase%></p>
    <p>Estou usando Generic Microservice DPR para escrever alguns números</p>
    <%Numeros%>
  </body>
</html>
```

Fonte: Elaborado pelo próprio autor.

Codificação 3 - Codificação Pascal - importa HTML, substitui textos e gera HTML final.

```
unit UsarHTMLBasico;

function Main: String;
var
  PaginaHTML, Numeros: String;
  I: Integer;
begin
```

```
  Numeros := '';
  for i := 1 to 10 do
    Numeros := Numeros + IntToStr(I) + '<br>';

  PaginaHTML := CodificacaoUnit('HTML_PURO.HTML');
  PaginaHTML := Troca(PaginaHTML, '<%Frase%>',  'Hello Word');
  PaginaHTML := Troca(PaginaHTML, '<%Numeros%>', Numeros);

  Result := PaginaHTML;
end;

end.
```

Fonte: Elaborado pelo próprio autor.

Com esta abordagem evitou-se algumas das restrições que foram impostas na tecnologia PPW, a saber:

- Todo comando (ou sequências de comandos) *Object Pascal* deve(m) estar sempre entre *scriptlets*. Não são permitidos *scriptlets* intercalados.
- Cada página pode conter no máximo uma declaração uses, uma declaração *type*, uma declaração *const* e uma declaração var. Cada declaração *uses, type, const* ou *var* deve estar obrigatoriamente associada a um único par de *scriptlets*.
- Não é permitida a criação de *procedures* ou *functions* dentro das páginas PPW. Entretanto, *units (códigos-fontes)* Pascal podem ser utilizadas através da cláusula uses da página PPW.

Com o GenericMicroServiceDPR pode-se realizar diferentes declarações, tais como *uses, const, type, var* tanto públicas quanto privadas às funções. Pode-se e até recomenda-se a criação de procedimentos (*procedures* e *functions*) para melhorar a organização e reutilização de códigos.

3.2 SERVICER - PLATAFORMA PARA DESENVOLVIMENTO RÁPIDO DE MICROSSERVIÇOS MULTILINGUAGENS

Os desafios advindos da adoção da arquitetura de microsserviços contam com várias soluções em forma de *framework*, porém na grande maioria dos casos estas estão restritas a apenas uma linguagem, conforme observou Chaves (2021) após comparar diversas ferramentas. Baseado neste cenário, o *Servicer* de Chaves (2021) se destaca como uma proposta de plataforma modular e extensível através de *plugins*, que busca facilitar o desenvolvimento de microsserviços automatizando processos de integração entre módulos escritos em diferentes linguagens de programação. Tal proposta baseou-se em uma arquitetura geral implementando um conjunto específico de tecnologias e conceitos, utilizando-se de uma interface comum para a definição de microsserviços e de uma ferramenta que automatiza processos relacionados à criação de uma aplicação distribuída entre diferentes linguagens. Requer-se apenas que a linguagem escolhida suporte a escrita do módulo necessário à integração.

O *Servicer* possui um componente central denominado *Application Manager,* responsável por gerenciar a aplicação, mantendo o registro de seus diferentes módulos. Esse registro contém uma especificação de um conjunto mínimo de definições necessárias para expressar um microsserviço em alto nível, conforme pode-se observar na Codificação 4. Nesta codificação é possível visualizar a definição do microsserviço e das entidades que exportará, bem como suas rotas (*paths*) incluindo *health check*, dependências de outros microsserviços, valores de *circuit breaker* e registros de *middlewares*.

Codificação 4 - Exemplo de definição YAML de um microsserviço no Servicer.

```yaml
service:
  name: livros-info-service
  entities:
    - name: Livro
      definition: com.lib.livrosinf.api.models.Livro
    - name: LivroForm
      definition: com.lib.livrosinf.api.models.LivroForm
  routes:
    - httpMethod: GET
      path: /api/catalogo
```

```
      handlerId: com.lib.livrosinf.api.LivrosInfService.catalogo()
      requestEntity: none
      responseEntity: [Livro]
    - httpMethod: GET
      path: /api/catalogo/:livroId
      handlerId: com.lib.livrosinf.api.LivrosInfService.livroDet(id: livroId)
      requestEntity: none
      responseEntity: Livro
      circuitBreaker: default
  healthCheck: /health
  gatewayRouting:
    rule: PathPrefix("/livros")
    middlewares: livros-strip-prefix
  dependencies:
    - livros-rating-service
forReuse:
  circuitBreaker:
    default:
      enabled: true
      maxFailures: 10
      callTimeout: 10
      resetTimeout: 15
  gateway:
    middlewares:
      livros-strip-prefix:
        stripprefix:
          prefixes: /livros
```

Fonte: Elaborado pelo próprio autor.

Após ler a definição de um microsserviço, o *Servicer* cria-o em tempo de execução, quando possível, sem gerar artefatos que necessitem ser gerenciados pelo desenvolvedor.

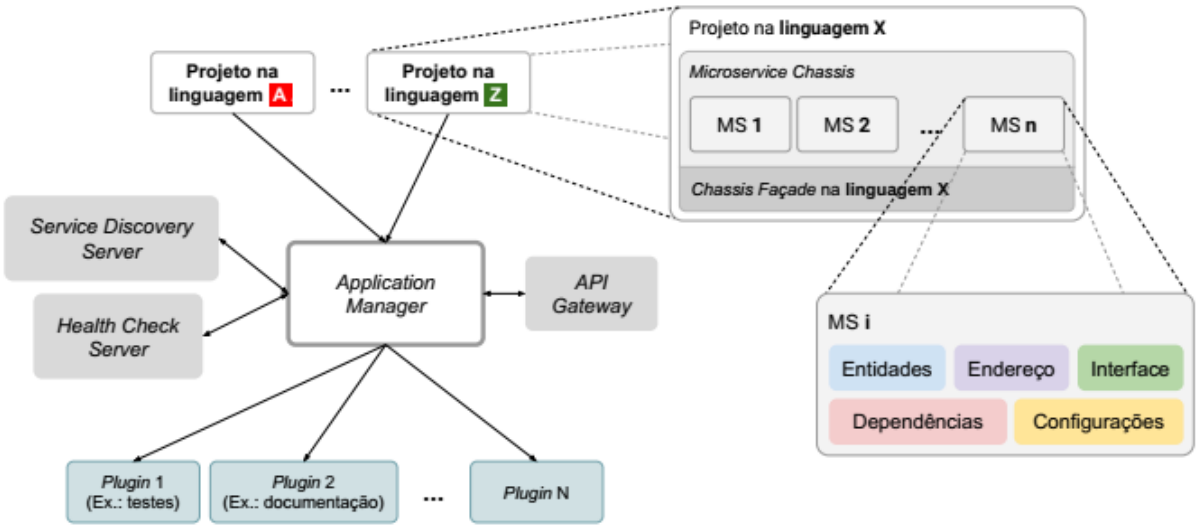
Essa plataforma serviu-se do Traefik[5], que é uma solução de código aberto desenvolvida com objetivo de facilitar o redirecionamento de requisições para os componentes responsáveis por respondê-las. No Traefik foi implementada a

[5] https://traefik.io/

estratégia de *API Gateway* permitindo-se configurar regras de redirecionamento e *middlewares* sem necessidade de programação em linguagem específica. Foi utilizado o Consul[6] para prover a descoberta de serviços e o monitoramento de disponibilidades.

Uma visão da arquitetura geral do *Servicer* é encontrada na Figura 4, na qual é possível observar que cada projeto em uma linguagem qualquer precisa implementar um *Microservice Chassis* que suporte vários microsserviços. Cada microsserviço precisa ter definidos seus endereços, suas entidades, interfaces, dependências e configurações. É necessário também um *Chassis Façade* que realize a tradução da definição em alto nível do microsserviço para a API do *Microservice Chassis* permitindo que, independentemente da linguagem utilizada, o microsserviço seja definido seguindo o mesmo conjunto de configurações. Sempre que houver modificações nos arquivos de definições dos modelos, essas diferenças são persistidas automaticamente, pelo *Chassis Façade,* no catálogo de entidades no *Application Manager*, sem precisar reiniciar o processo do Traefik para que as mudanças nas configurações ocorram.

Figura 4 - Arquitetura geral do Servicer.



Fonte: Chaves (2021)

O presente trabalho também pretende desenvolver um tipo de *Microservice Chassis* que facilite o desenvolvimento rápido de microsserviços, com possibilidade

[6] https://www.consul.io/

de se invocar diferentes linguagens, contudo, diferentemente do trabalho de Chaves (2021) não será dependente de outras tecnologias, procurando ser o mais aberto possível, bastando que seja desenvolvida uma unidade de codificação em Pascal que invoque a outra linguagem[7]. Não será obrigatório que os microsserviços precisem expressar entidades que os vinculem a bancos de dados, pois prezará pela liberdade de ter microsserviços apenas realizando processamentos com os dados recebidos sem necessidade de persisti-los, apenas retornando o resultado do processamento.

Chaves (2021) expressou as definições dos modelos em YAML precisando realizar conversão para JSON *Schema*[8] e posteriormente para tipos em *Typescript*[9]. Constatou, inclusive, que a normalização dessas definições para os diferentes *frameworks* acarretou perda de expressividade, consequência de algumas particularidades de cada tecnologia. Essa tradução não será necessária na proposta deste trabalho, pois com os recursos implementados para interpretação será possível retornar coleções de dados diretamente em JSON, XML ou CSV.

Um recurso que foi citado como possibilidade para trabalho futuro em Chaves (2021) foi permitir a integração com serviços externos ao próprio *Servicer*, algo que também será resolvido neste trabalho, pois as funções interpretadas permitem que sejam feitas requisições HTTP e HTTPS a serviços externos em qualquer parte da internet.

[7] Ver seção 5.6 TESTE DE INVOCAÇÃO DE OUTRA LINGUAGEM
[8] https://json-schema.org/
[9] https://www.typescriptlang.org/

4 METODOLOGIA

Como o presente trabalho propõe-se a desenvolver um sistema, a metodologia se dará pelas propostas de desenvolvimento, de configurações, de uso de repositórios de códigos-fontes e de testes a serem efetuados no aplicativo.

4.1 PROPOSTA DE DESENVOLVIMENTO

Para o desenvolvimento do sistema GenericMicroServiceDPR foi utilizada a linguagem **Delphi**[10] na ferramenta *Embarcadero® Delphi 10.1 Berlin Version 24.0.24468.8770* na qual é possível desenvolver praticamente qualquer tipo de aplicação, inclusive servidores web, que poderiam ser *DataSnap REST, DataSnap Server, DataSnap WebBroker, IntraWeb, Web Server, SOAP Server, ISAPI dynamic link library, CGI stand-alone executable* etc.

Foi construída uma aplicação sem interface gráfica *(Console Application)* devido aos seguintes motivos: ser o padrão mais adotado atualmente para servidores web*;* ser a opção que gera um menor executável; permitir total controle sobre as requisições e suas respectivas respostas; facilitar a execução em servidores sem interface gráfica, por exemplo, em contêineres. Esta aplicação foi construída usando a arquitetura 64 bits por ser o padrão mais atual no mercado e conseguir tirar melhor proveito dos recursos das máquinas.

Para atender as requisições web foi utilizado o componente **Horse**[11]**,** *free* e *open-source***,** versão 2.0.13, da HashLoad. Este componente é um *web framework* para Delphi inspirado no ExpressJS, projetado para facilitar o desenvolvimento rápido de uma forma minimalista e com alto desempenho. Seu funcionamento permite a adição de *middlewares* de acordo com a necessidade do projeto.

Existem diversos *middlewares* desenvolvidos para o Horse e que podem ser consultados na página oficial. Sendo também possível desenvolver facilmente os próprios *middlewares*. Para este projeto foram utilizados *middlewares* para adicionar as seguintes funcionalidades ao Horse: tratamento de JSON, tratamento de *Cross Origin Resource Sharing* (CORS), compactação de tráfego, registro de *logs,*

[10] https://www.embarcadero.com/br/products/delphi
[11] https://github.com/HashLoad/horse

download/upload de arquivos, validação de cache, servidor de arquivos estáticos, tratamento de exceções, conforme pode ser visualizado no Quadro 1.

Quadro 1 - Middlewares utilizados com o Horse.

Repositório do ***Middleware***	Descrição Original do ***Middleware***
https://github.com/HashLoad/jhonson	*Middleware for parse JSON in HORSE.*
https://github.com/HashLoad/handle-exception	*Middleware for handle exception in HORSE.*
https://github.com/HashLoad/horse-cors	*Middleware for inject CORS headers in HORSE.*
https://github.com/HashLoad/horse-octet-stream	*Middleware for work with application/octet-stream in HORSE.*
https://github.com/paolo-rossi/delphi-jose-jwt	*Delphi implementation of JWT (JSON Web Token) and the JOSE (JSON Object Signing and Encryption) specification suite. This library supports the JWS (JWE support is planned) compact serializations with several JOSE algorithms.*
https://github.com/HashLoad/horse-jwt	*Basic JWT middleware for Horse.*
https://github.com/HashLoad/horse-compression	*Middleware for compression in HORSE.*
https://github.com/bittencourtthulio/Horse-ETag	*Middleware* para Servidor Horse para Controle de *Etags.*
https://github.com/CarlosHe/horse-staticfiles	*Middleware for StaticFiles in HORSE.*

Fonte: Elaborado pelo próprio autor.

Foram desenvolvidos outros dois *middlewares* para a execução deste trabalho: um para assinar todas as respostas às requisições e outro para interceptar todas as requisições e respostas, criando a possibilidade de realizar diversos tipos de tratamentos. Alguns ajustes também foram necessários nos *middlewares* de terceiros utilizados.

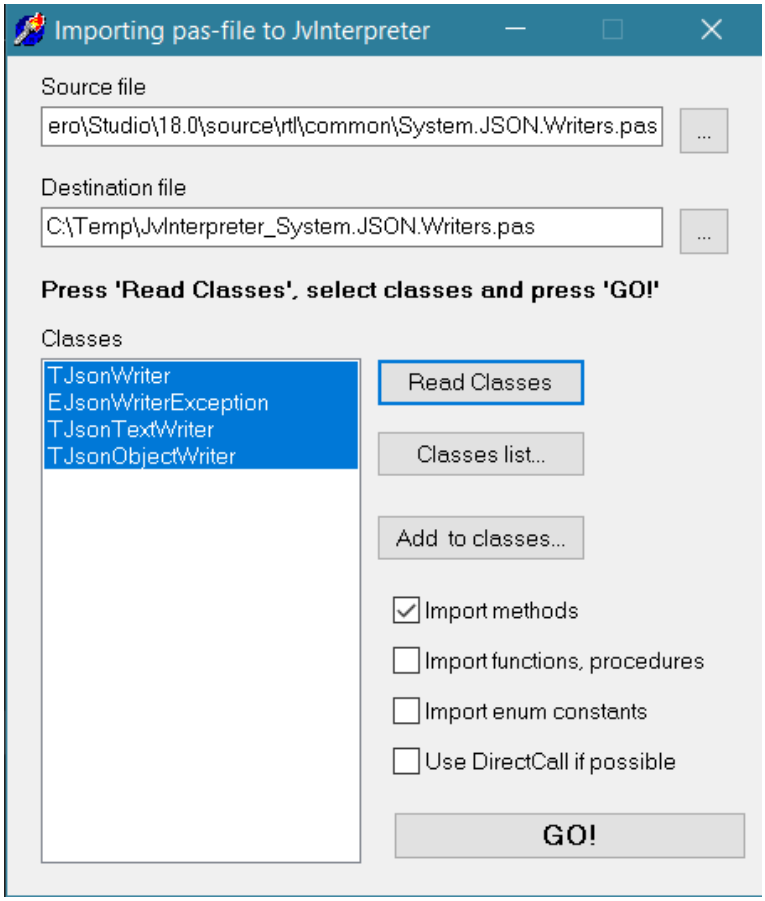
Para realizar a interpretação das unidades de codificação em *Object Pascal* foi utilizado o componente **JVInterpreter**[12]**,** *free* e *open-source,* da suíte de componentes *JEDI Visual Component Library (JVCL)*[13]. Este componente vem com uma lista bem grande de classes e *units* interpretadas do Delphi (*System, SysUtils, Classes, Windows, Graphics, Controls, Buttons, StdCtrls, ComCtrls, ExtCtrls, Forms, Dialogs,*

[12] Para visualizar alguns exemplos de uso do JVInterpreter consulte http://tetros.chat.ru/JvInterpreter.html.
[13] https://github.com/project-jedi/jvcl

Menus, Grids, Db, DbCtrls, DbGrids, Math) e ainda permite que novas *units* e classes possam ser adicionadas ao interpretador.

Foi necessário adicionar interpretação para algumas outras *units* e classes nativas do Delphi / Horse, tais como *DateUtils, TClientDataSet,* TXMLDocument, TWebRequest, TWebResponse, JOSE.Core.Builder, JOSE.Core.JWT, JOSE.Core.JWS, JOSE.Core.JWK, JOSE.Consumer, Horse.HTTP, Horse.Exceptions e também para outras que foram criadas para facilitar a codificação, tais como EnviarEmail, FuncoesFiredac, FuncoesHash, FuncoesJSON, FuncoesNumero, FuncoesODBC, FuncoesRede, FuncoesRTTI, FuncoesSistemaOperacionais, FuncoesSQL, FuncoesStrings, Helper_Grid. A própria suíte JVCL[14] vem com um aplicativo chamado **Pas2Rai2.exe** que permite converter classes nativas Delphi para os seus respectivos correspondentes com funcionalidade de interpretação. Um exemplo do aplicativo em execução pode ser visto na Imagem 1.

Imagem 1 - Pas2Rai2 - Importando pas-file para o JVInterpreter.

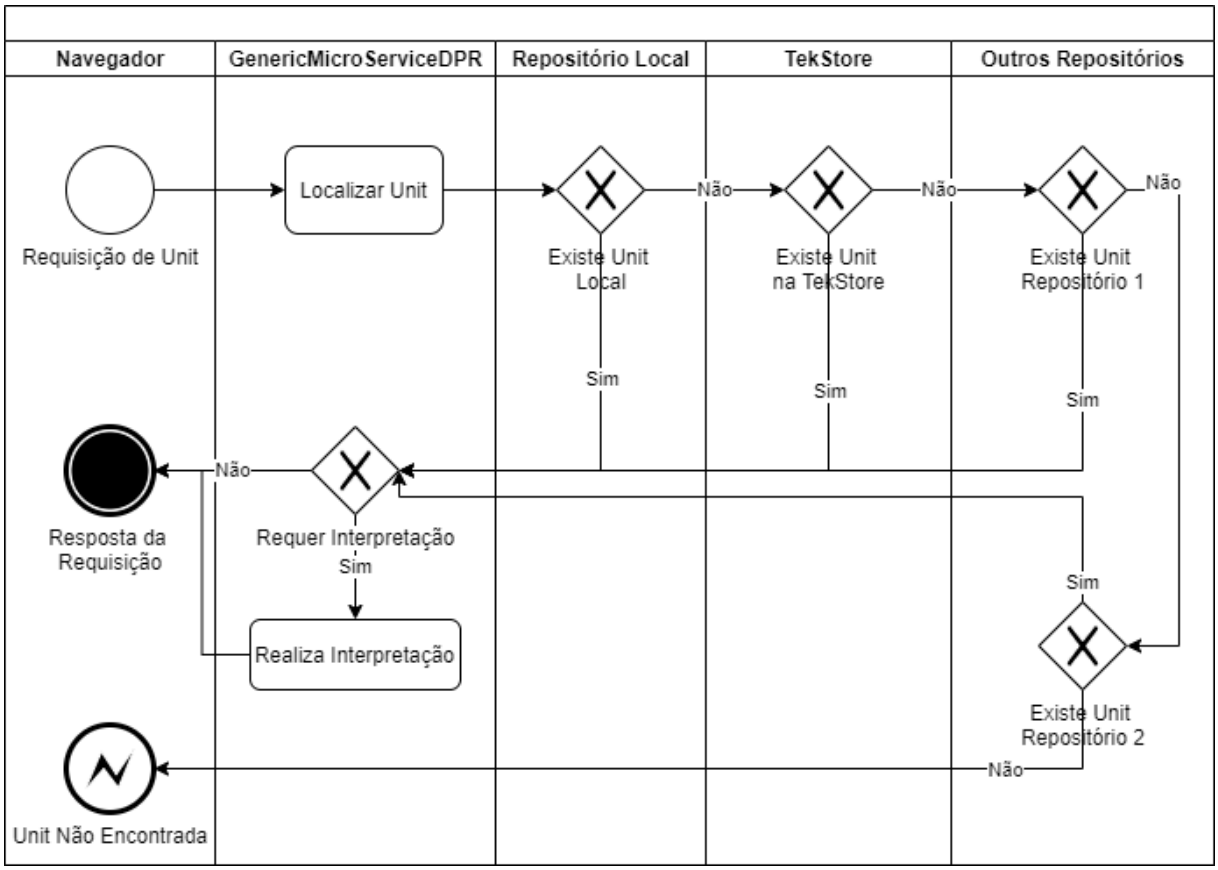


Fonte: Elaborado pelo próprio autor.

[14] https://github.com/project-jedi/jvcl

O GenericMicroServiceDPR recebe, via linha de comando, um arquivo de configuração em formato JSON com as definições de execução daquela instância específica. Cada instância responde como um microsserviço independente dividindo as tarefas. Assim que for iniciado, começa a atender requisições para unidades de codificação (código-fonte em Pascal). Localiza tal codificação em um dos repositórios definidos, realiza sua interpretação e devolve o resultado do processamento como uma resposta para a aplicação cliente. Um esquema do funcionamento pode ser visto no Esquema 1.

Esquema 1 - Esquema de funcionamento do GenericMicroServiceDPR.



Fonte: Elaborado pelo próprio autor.

4.2 PROPOSTA DE CONFIGURAÇÕES

Como o arquivo de configuração deve estar em formato JSON, cumpre a obrigação de descrever cada uma das opções de configuração contidas no arquivo. Na primeira execução, o próprio GenericMicroServiceDPR criará um modelo de arquivo, bastando apenas editá-lo posteriormente em um editor de texto puro.

A configuração de **descricao** serve apenas como um indicativo simples e rápido da função do microsserviço que está atendendo naquela porta.

O grupo de configuração **horseOptions** contém configurações do Horse e de seus *middlewares*.

A configuração **horseOptions.porta** serve exatamente para definir em qual porta o microsserviço atenderá. Vale lembrar que esta porta sempre deverá estar liberada no firewall da máquina onde o microsserviço está em execução para que este aceite requisições externas à máquina. No caso de estar sendo executada em um contêiner, a porta exposta pela imagem padrão criada é a 9000, mas a porta externa ligada a ela pode ser definida no momento de execução do contêiner.

A configuração **horseOptions.permitirDownloads** determina se a instância realizará downloads de arquivos que estejam na pasta **/downloads** acima do local onde a aplicação foi executada ou em um volume mapeado, no caso de estar sendo executado dentro de um contêiner. A execução do download se dará através de requisição do tipo GET ao *endpoint* /download/:arquivo.

Inversamente, a configuração **horseOptions.permitirUploads** determina se a instância aceitará que arquivos sejam enviados através de uploads para a pasta **/uploads** acima do local onde a aplicação foi executada. Os arquivos que forem subidos, receberão uma identificação única junto ao seu nome, para evitar conflitos. A execução se dará através de requisição do tipo POST ao *endpoint* /upload/:arquivo.

Se o microsserviço não aceitar downloads e nem uploads, o *middleware* correspondente não será carregado.

A configuração **horseOptions.controlarCrossOriginResourceSharing** determina se serão adicionadas automaticamente diretivas aos *headeres* das respostas HTTP para evitar bloqueios no compartilhamento de recursos com origens diferentes, erro conhecido como CORS. Se esta configuração estiver *false* o plugin correspondente não será ativado visando ganhar desempenho nas requisições.

A configuração **horseOptions.compactarRespostaMaiorQue1024Bytes** informa para o microsserviço que este deve ativar o plugin de compactação para os dados trafegados, sempre que estes forem superiores a 1024 bytes. Tamanhos menores do que isto não chegam a compensar a compactação, visto que reduzirá o tráfego, mas perderá tempo e processamento na compactação e descompactação dos dados.

A configuração **horseOptions.calcularEtagParaResultadosJSON** informa para o microsserviço que ele deve calcular um *hash,* para os conteúdos JSON, que serão devolvidos nas respostas das requisições através do cabeçalho (*header*) ETag[15]. Essa informação pode ser utilizada pela aplicação cliente ao enviar uma segunda requisição para o mesmo *endpoint* preenchendo o cabeçalho (*header*) *If-None-Match*. Desta forma o microsserviço pode, nas próximas requisições, verificar se o conteúdo JSON produz o mesmo *hash*. Caso gere o mesmo *hash* devolverá apenas o HTTP *Status Code 304-Not Modified* (sem modificação), mas não o conteúdo JSON em si. O Status Code 304 indica para a aplicação cliente que o conteúdo que esta possui, da requisição anterior, está atualizado e que poderá utilizá-lo. Evitando, assim, tráfego desnecessário de grandes coleções de dados.

A configuração **horseOptions.parseJson** ativa o *middleware* correspondente para realizar análise e conversão (*parse)* de JSON tanto para as requisições quanto para as respostas.

A configuração **horseOptions.handleExceptionAsJSON** ativa o *middleware* que converte os possíveis erros em uma resposta no formato JSON. Neste caso, requer-se que horseOptions.parseJson esteja habilitado.

A configuração **horseOptions. servirArquivosPublicos** ativa o *middleware* que permite aos arquivos que estejam na pasta **/public,** acima da pasta onde o GenericMicroServiceDPR estiver executando, sejam fornecidos para o *frontend*.

A configuração **horseOptions.unitInterceptacao** aponta para uma possível *unit* que interceptará todas as requisições e respostas. No caso de ser configurada, esta *unit* deverá possuir dois métodos: **_Antes** e **_Depois**. Ambos receberão respectivamente os objetos de requisição (THorseRequest) e de resposta (THorseResponse). Neste ponto é possível realizar qualquer tipo de interceptação,

[15] Para mais informações sobre ETag consulte https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Headers/ETag.

visando depurar algum problema, registrar algum *log* específico, validar ou modificar os cabeçalhos etc. O desenvolvedor pode adicionar seu próprio *middleware* aqui.

O grupo de configurações **horseOptions.certificado** contém informações para a ativação do protocolo seguro HTTPS, simplesmente apontando o nome do arquivo que contém o certificado (opção **arquivoCertificado**), o nome do arquivo que contém a chave do certificado (opção **arquivoChave**) e caso seja necessária a raiz do arquivo certificado (opção **raizArquivoCertificado**). Estes arquivos devem estar na pasta **/certificados** acima da pasta onde o aplicativo está em execução ou em um volume mapeado para isto, caso esteja sendo executado em um contêiner. Foge ao escopo deste trabalho explicar como os arquivos de certificado devem ser gerados.

O grupo de configurações **baseDeDadosPadrao** contém informações de conexão a um banco de dados padrão para o microsserviço. Normalmente microsserviços se conectam a apenas um banco de dados, mas havendo necessidade de se conectar a mais de um, a informação de conexão aos outros bancos de dados secundários precisará ser informada nas funções específicas para isso.

Semelhantemente, o grupo de configurações **servidorEmailPadrao** contém informações de uma conta de e-mail padrão que será utilizada pelo microsserviço para o envio de e-mails em suas codificações, quando solicitado. É possível também enviar e-mails a partir de outras contas, mas nessa situação deve-se explicitar outra configuração na invocação da classe de envio de e-mails.

A configuração **repositorioUnitsLocal** determina se o microsserviço procurará pelas unidades de codificação (códigos-fontes) armazenados localmente, ou seja, junto ao microsserviço. Este, aliás, é o comportamento padrão de todos os servidores web atuais. No GenericMicroServiceDPR, caso este parâmetro esteja como *true*, ele procurará primeiramente na pasta **/units/<numero_porta>** acima de onde ele foi executado. Posteriormente procurará na pasta **/units** ou no volume destinado às unidades de codificação, no caso de estar rodando em contêiner. Por exemplo, se a aplicação for executada a partir da pasta C:\microsservico e atende na porta 9000, esta procurará primeiramente na pasta C:\microsservico\units\9000 e depois na pasta C:\microsservico\units. Isso permite que cada tipo de instância de microsserviço tenha as suas próprias *units* associadas à sua porta e compartilhe de *units* gerais como

bibliotecas ou *frameworks* da empresa. Um uso clássico seria, por exemplo, não ter o *Bootstrap*[16] em cada pasta de porta, mas apenas na pasta compartilhada entre eles.

O grupo de configurações **repositorioUnitsTekStore** determina se o microsserviço deve ou não buscar no repositório privado TekStore[17] sempre que não encontrar uma *unit* local.

Há que se destacar a importância da configuração **repositorioUnitsTekStore.expressaoRegularParceirosAceitos**. É nesta configuração que se consegue limitar de quais parceiros da TekStore a instância do GenericMicroServiceDPR estará habilitada a aceitar codificações. Aqui reside não apenas a possibilidade de se distribuir funções entre as diversas instâncias, mas também uma responsabilidade de segurança. Cada microsserviço deve saber exatamente de quem pode aceitar codificações remotas para serem interpretadas.

A configuração **repositoriosUnitsPublicos** é um vetor (*array*) com uma lista de endereços web de repositórios abertos. Neste trabalho foi utilizado como exemplo o GitHub, mas poderia se referenciar ao S3 da Amazon[18] ou a outros.

O grupo de configuração **paginasEspeciais**[19] permite configurar redirecionamentos para páginas de documentação, saúde, testes unitários, autenticação e publicidade relativas a cada microsserviço, visando padronizar todo o ecossistema.

A configuração **log_Sucesso** determina se o sistema fará um registro (*log*) no *console* quando uma *unit* for executada sem problemas. Em tempo de desenvolvimento é interessante este registro, mas em produção, dependendo da velocidade que se queira imputar ao microsserviço, talvez seja interessante não realizar estes registros.

A configuração **log_FormatoJson** determina se o sistema fará os registros (*logs)* em formato JSON, ideal quando se tem algum outro sistema capturando essas informações para realizar análises. Caso seja *false* fará o registro em texto simples com destaques coloridos para as informações mais importantes.

Convém observar que as configurações de bancos de dados e e-mail padrão não possuem configurações para suas senhas. Por segurança, as senhas padrões

---

[16] https://getbootstrap.com/

[17] Para mais informações sobre TekStore veja seção 4.3 PROPOSTA DE REPOSITÓRIOS DE CÓDIGOS-FONTES.

[18] https://aws.amazon.com/pt/s3/

[19] Para mais informações sobre paginasEspeciais veja seção 5.1 EXECUÇÃO DO SISTEMA E SUAS ROTAS (ENDPOINTS).

são capturadas das variáveis de ambiente BDD_PASSWORD e MAIL_PASSWORD respectivamente. Primeiramente procura-se em uma variável que finalize com o número da porta escutada pela instância, por exemplo BDD_PASSWORD_9000. Caso o resultado esteja vazio, procura-se pela variável genérica BDD_PASSWORD. Desta forma permite-se configurar senhas diferentes para cada banco de dados ou conta de e-mail de cada microsserviço sendo executado na mesma máquina. Esse recurso de numeração das variáveis de ambiente de acordo com as portas não é necessário quando se usa contêineres, pois cada instância de contêiner tem suas próprias variáveis de ambiente.

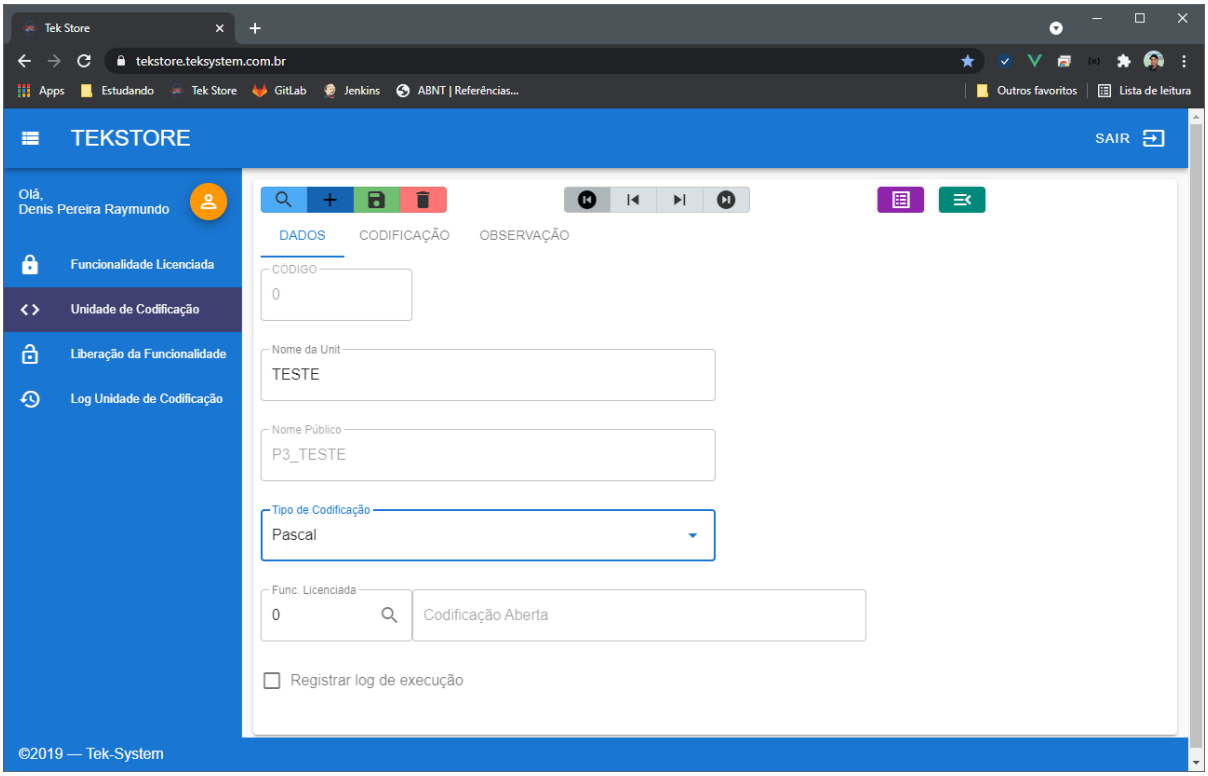
4.3 PROPOSTA DE REPOSITÓRIOS DE CÓDIGOS-FONTES

Como exemplo de repositório público protegido de códigos-fontes foi utilizada a **TekStore[20],** um repositório de códigos-fontes da *software house* Tek-System[21], que fornece uma estrutura para que os seus clientes possam codificar partes customizáveis nos seus sistemas *Enterprise Resource Planning* (ERP). Quando um código-fonte é solicitado, é necessário informar uma autenticação específica para que trafegue criptografado e em um canal seguro (HTTPS). O GenericMicroServiceDPR precisa decriptar o arquivo antes de o interpretar. Dessa forma existe uma segurança quanto aos códigos-fontes que não precisam estar armazenados junto ao servidor web. Ou seja, pode-se executar o GenericMicroServiceDPR em um computador e solicitar a interpretação de uma *unit* que está na TekStore.

A TekStore trabalha com um sistema de parceria, no qual cada desenvolvedor pode solicitar uma conta, se aprovado recebe um código de parceria que identifica as *units* criadas por ele. Por exemplo, uma *unit* com o nome “TESTE” criada por um desenvolvedor cujo código de parceria seja 3, recebe o nome público P3_TESTE, conforme pode ser visualizado na Imagem 2 e na Imagem 3.

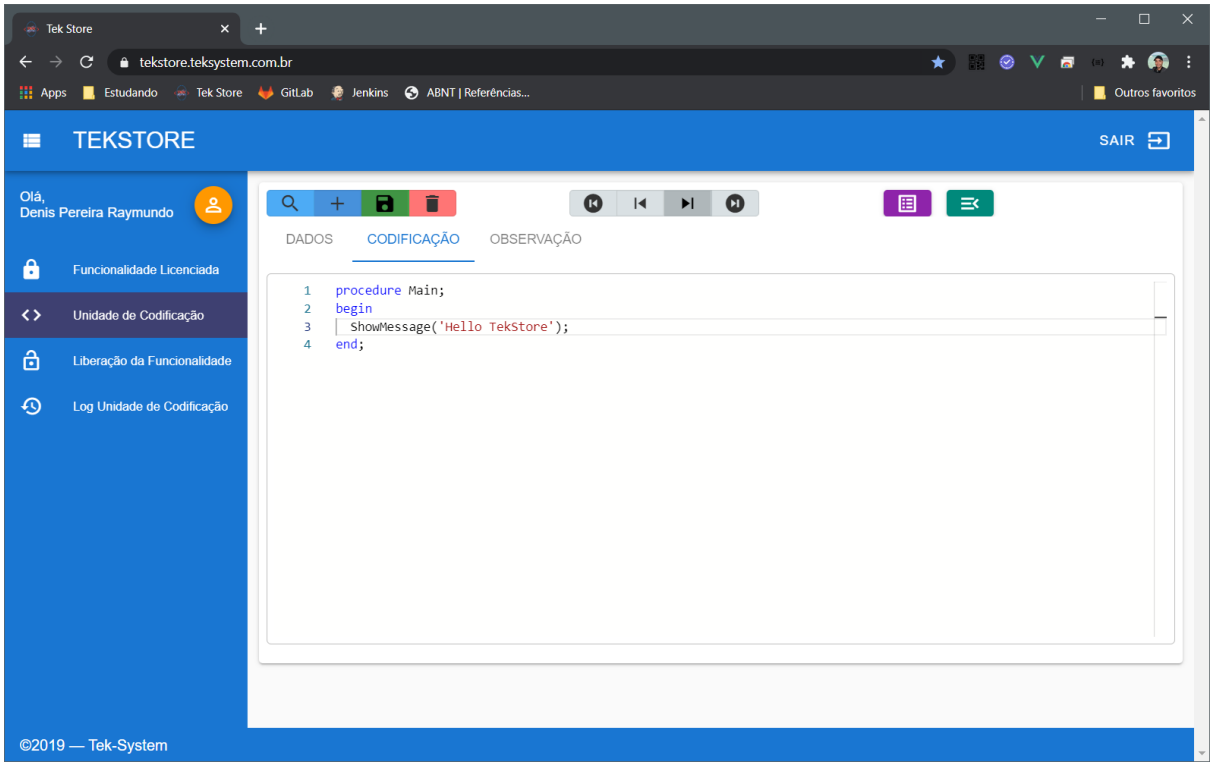
[20] https://tekstore.teksystem.com.br/
[21] https://www.teksystem.com.br/

Imagem 2 - Cad. Unidade de Codificação na TekStore - Parte 1.



Fonte: Elaborado pelo próprio autor.

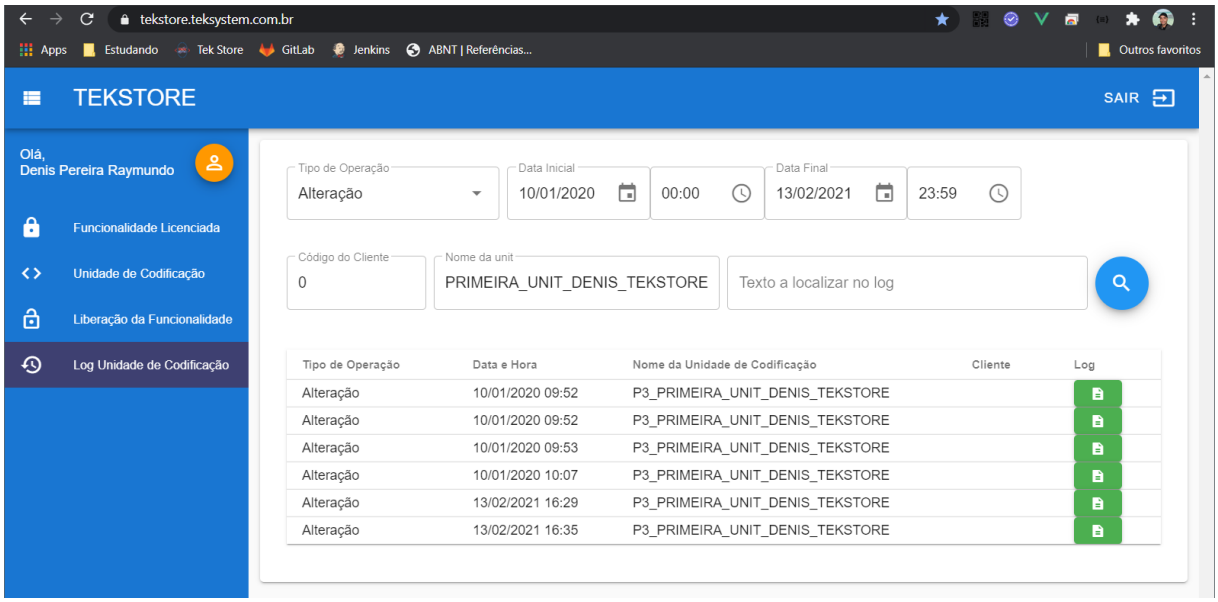
Imagem 3 - Cad. Unidade de Codificação na TekStore - Parte 2.



Fonte: Elaborado pelo próprio autor.

Este repositório permite o armazenamento de codificações Pascal, HTML, CSS, JavaScript, SQL, JSON, INI, XML, BAT, PHP, Python, Java. Na verdade, qualquer texto puro (*text-plain*), mas o destaque da sintaxe está disponível apenas para os formatos citados acima, até o momento da escrita deste trabalho. A TekStore possui um controle de versionamento de fontes embutido, sendo possível ter acesso a versões anteriores através do registro (*log)*, conforme pode ser visto na Imagem 4.

Imagem 4 - Registro de logs na TekStore.



Fonte: Elaborado pelo próprio autor.

Na TekStore também é possível trabalhar com licenças por funcionalidades. Resumindo, pode-se atribuir uma licença a uma ou mais unidades de codificação e apenas clientes com a referida licença em dia terão acesso à codificação. Não está no escopo deste trabalho dar mais detalhes sobre este funcionamento.

Como exemplo de repositório público aberto de códigos-fontes foi utilizado o **GitHub**[22] por ser um dos maiores e mais utilizados na comunidade de desenvolvedores atualmente, dispensando demais comentários. Contudo, poderia ter sido utilizado o BitBucket[23] ou qualquer outro.

Foi utilizada também uma pasta web simples como exemplo de um repositório particular e aberto.

[22] https://github.com/
[23] https://bitbucket.org/

4.4 PROPOSTA DE TESTES

Encerrado o desenvolvimento do GenericMicroServiceDPR, foi realizada uma bateria de testes com diversos tipos de requisições utilizando-se os navegadores **Google Chrome**[24] Versão 89.0.4389.72 (Versão oficial) 64 bits, **Firefox**[25] 86.0 (64-bits) e **Microsoft Edge**[26] Versão 89.0.774.45 (Compilação oficial) (64 bits).

Para realizar a automatização de requisições foi utilizado o **curl**[27] 7.55.1 (Windows) libcurl/7.55.1 WinSSL.

Para testar os diversos métodos HTTP e analisar as requisições foi utilizada a aplicação **Postman**[28] versão 8.6.1, por ser uma das principais ferramentas utilizadas durante o desenvolvimento de APIs REST, capaz de auxiliar as etapas de testes, construção e refatoração dessas APIs (RANGA; SONI, 2019).

Para realizar os testes de performance foi utilizada a aplicação **RESTful Stress**[29] versão 1.6.4.

Encerrados os testes do GenericMicroServiceDPR como uma aplicação independente (*stand-alone*), foi gerada uma imagem para o **Docker**[30] versão 20.10.2 OS/Arch: windows/amd64 baseada na imagem mcr.microsoft.com/windows/servercore:ltsc2019[31].

> *Docker é um conjunto de produtos de plataforma como serviço que usam virtualização de nível de sistema operacional para entregar software em pacotes chamados contêineres. Os contêineres são isolados uns dos outros e agrupam seus próprios softwares, bibliotecas e arquivos de configuração* (Wikipédia, 2021).

[24] https://www.google.com/intl/pt-BR/chrome/
[25] https://www.mozilla.org/pt-BR/firefox/new/
[26] https://www.microsoft.com/pt-br/edge
[27] https://curl.se/windows/
[28] https://www.postman.com/downloads
[29] https://github.com/maurobussini/restful-stress
[30] https://www.docker.com/products/docker-desktop
[31] https://hub.docker.com/_/microsoft-windows-servercore

Gerada a imagem para o Docker foram repetidos todos os testes, porém agora com o GenericMicroServiceDPR encapsulado em um contêiner em execução. Após a repetição destes testes foi publicada a imagem gerada no repositório **DockerHub**[32].

Para certificar-se da possibilidade de atender a demandas empresariais, foi usada uma imagem com o **NGinX**[33] em um contêiner para realizar o balanceamento de carga entre dois contêineres idênticos que contém a imagem Docker com o GenericMicroServiceDPR. Poderia também ter sido usado um cluster com **Docker Swarm** (ferramenta instalada junto com Docker Desktop). Foram refeitos novamente todos os testes, mas desta vez observando-se a alternância entre os contêineres.

Todas as ferramentas citadas neste trabalho, com exceção do Delphi, estão em suas versões mais atualizadas na data da escrita deste trabalho.

[32] https://hub.docker.com
[33] https://hub.docker.com/_/nginx

5 RESULTADOS E DISCUSSÃO

5.1 EXECUÇÃO DO SISTEMA E SUAS ROTAS (*ENDPOINTS*)

Conforme foi proposto, assim que o sistema é iniciado é feita a verificação se foi informado (por meio do parâmetro **-F:**) o nome de um arquivo de configurações a ser utilizado. Se nenhum arquivo for passado, adota-se o arquivo padrão **GenericMicroServiceDPR-config.json**. Se este arquivo não existir, o GenericMicroServiceDPR o criará com valores iniciais considerados padrões (*defaults*) em uma pasta *config* na estrutura de diretórios acima de onde ele está executando ou, no caso de estar encapsulado em um contêiner, em um volume mapeado para conter todos os arquivos de configuração.

Após carregar as configurações do arquivo informado via linha de comando ou do arquivo padrão, o sistema imprime, no *console*, informações de identificação do microsserviço, bem como a configuração[34] adotada.

Em seguida, são ajustadas as configurações regionais; carrega-se o arquivo de certificado SSL, caso tenha sido configurado; registra-se todos os *middlewares* habilitados e as rotas que serão aceitas; por fim começa-se a atender às requisições na porta e protocolo para os quais foi configurado. Este processo de inicialização demora menos de um segundo.

Na Imagem 5 pode-se visualizar o sistema em execução, as configurações adotadas, as variáveis de ambiente utilizadas e a data e hora do momento em que o microsserviço foi iniciado. A impressão do momento de inicialização permite que facilmente sejam aferidas suas métricas quanto à disponibilidade do microsserviço.

Para um encerramento adequado da aplicação pode-se digitar no próprio *console* da aplicação uma das palavras a seguir e pressionar a tecla *enter*: FIM, STOP, CLOSE, QUIT, EXIT. Desta forma pode-se visualizar o momento do encerramento e o tempo total em execução. Pode-se, alternativamente, acionar CTRL+C no *console* ou simplesmente fechar a janela que o mantém aberto para que seja encerrado imediatamente.

Deve-se tomar cuidado ao se selecionar textos na interface *console*, pois isto pausará o atendimento às requisições enquanto o texto estiver selecionado.

---

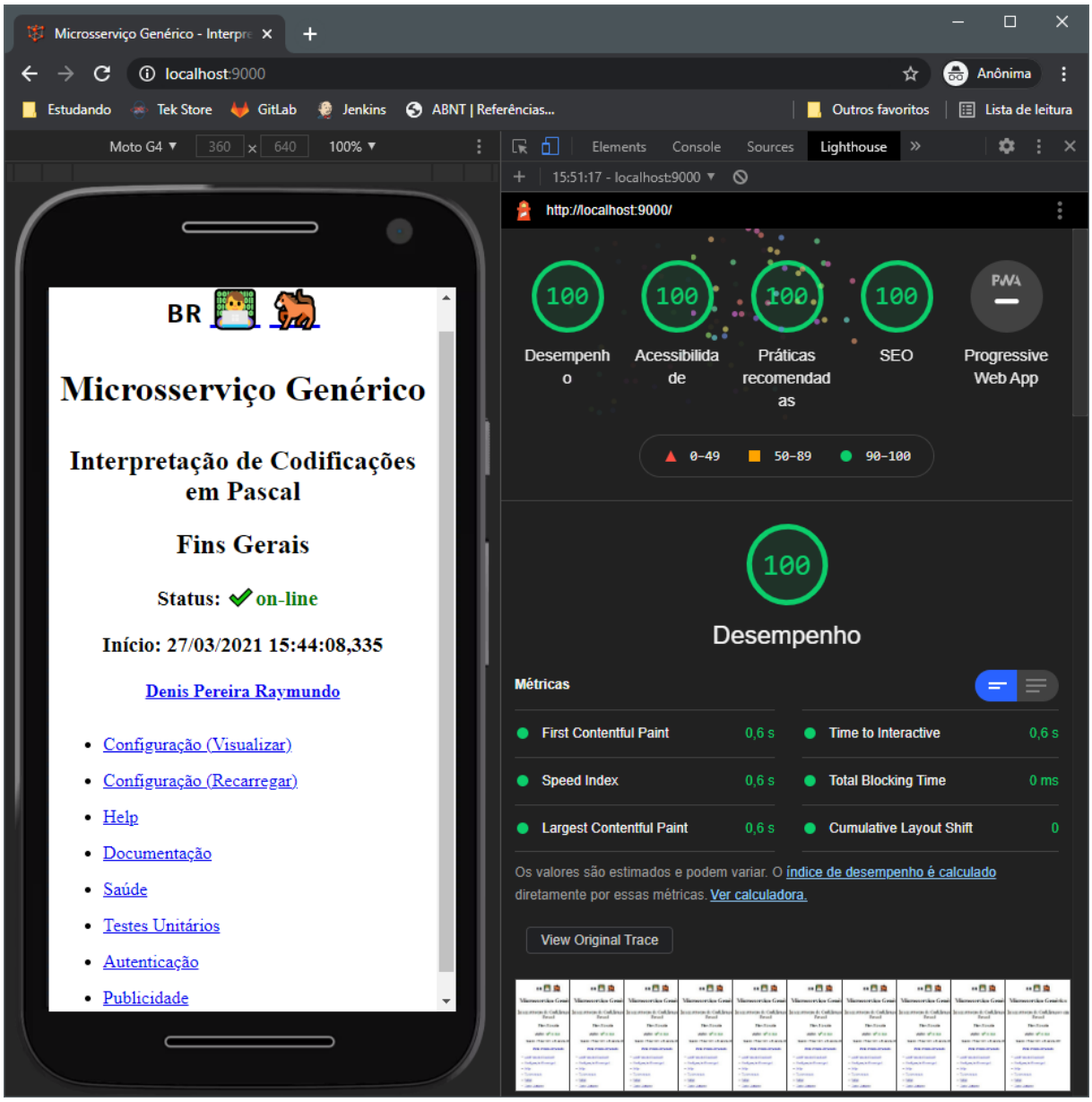
[34] Para mais informações sobre configurações veja seção 4.2 PROPOSTA DE CONFIGURAÇÕES

Imagem 5 - GenericMicroServiceDPR em execução no servidor.

Fonte: Elaborado pelo próprio autor.

Na Imagem 6 é possível visualizar no navegador a resposta ao *endpoint* principal (/) também conhecido como *index*, na porta configurada (9000) e o relatório de testes do *Lighthouse*[35].

Imagem 6 - GenericMicroServiceDPR em execução no navegador.



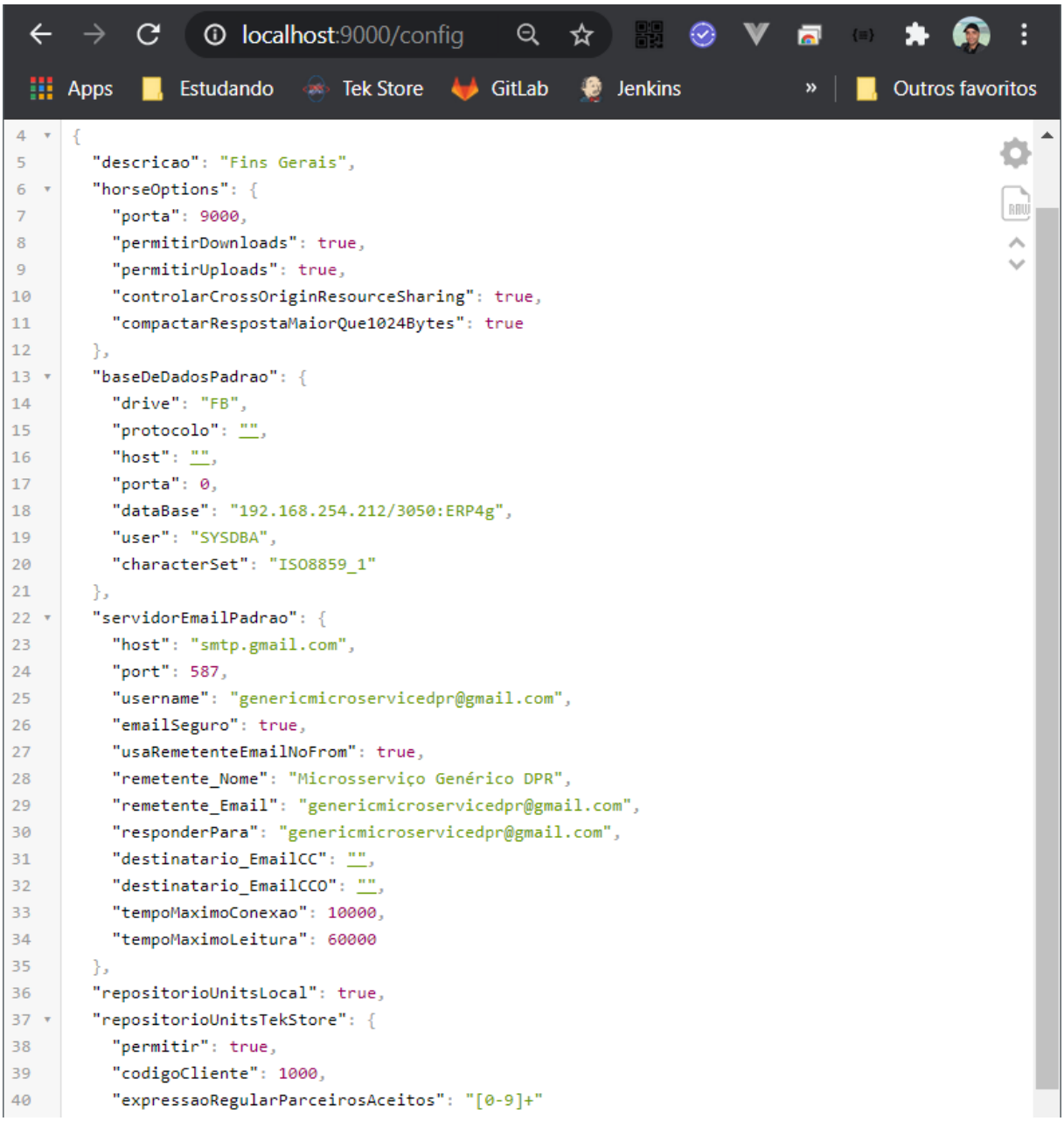
Fonte: Elaborado pelo próprio autor.

Todos as rotas (*endpoints)* diferenciam maiúsculas de minúsculas (são *case-sensitive*) e estão sempre em caixa baixa. Seguem explicações sobre as opções apresentadas como links:

[35] Veja glossário ao final do trabalho.

O link **configuração (Visualizar)** abre, em uma nova aba, o *endpoint* **/config** que apresenta as configurações que estão regendo a referida instância do microsserviço na porta solicitada. Caso não se queira deixar este ou qualquer outro *endpoint* público, pode-se utilizar um interceptador[36] para limitar o acesso a apenas usuários autenticados. Na Imagem 7 é possível visualizar um exemplo de configuração em execução.

Imagem 7 - Exemplo de configurações do GenericMicroServiceDPR.



Fonte: Elaborado pelo próprio autor.

[36] Para visualizar exemplos de interceptação consulte APÊNDICE I - Exemplo de Codificação de Interceptação

O link **configuração (recarregar)**, recarrega remotamente a configuração, que deve ter sido modificada previamente, através da chamada ao *endpoint* **/config/reload,** sem precisar reiniciar o serviço. A exceção está no nó **horseOptions**. Alterações neste nó vão requerer reinício do serviço. A possibilidade de mudar grande parte da configuração sem precisar reiniciar o serviço colabora para que os microsserviços consigam cumprir os seus Contratos de Nível de Serviço (*Service Level Agreement* - SLA) evitando indisponibilidade.

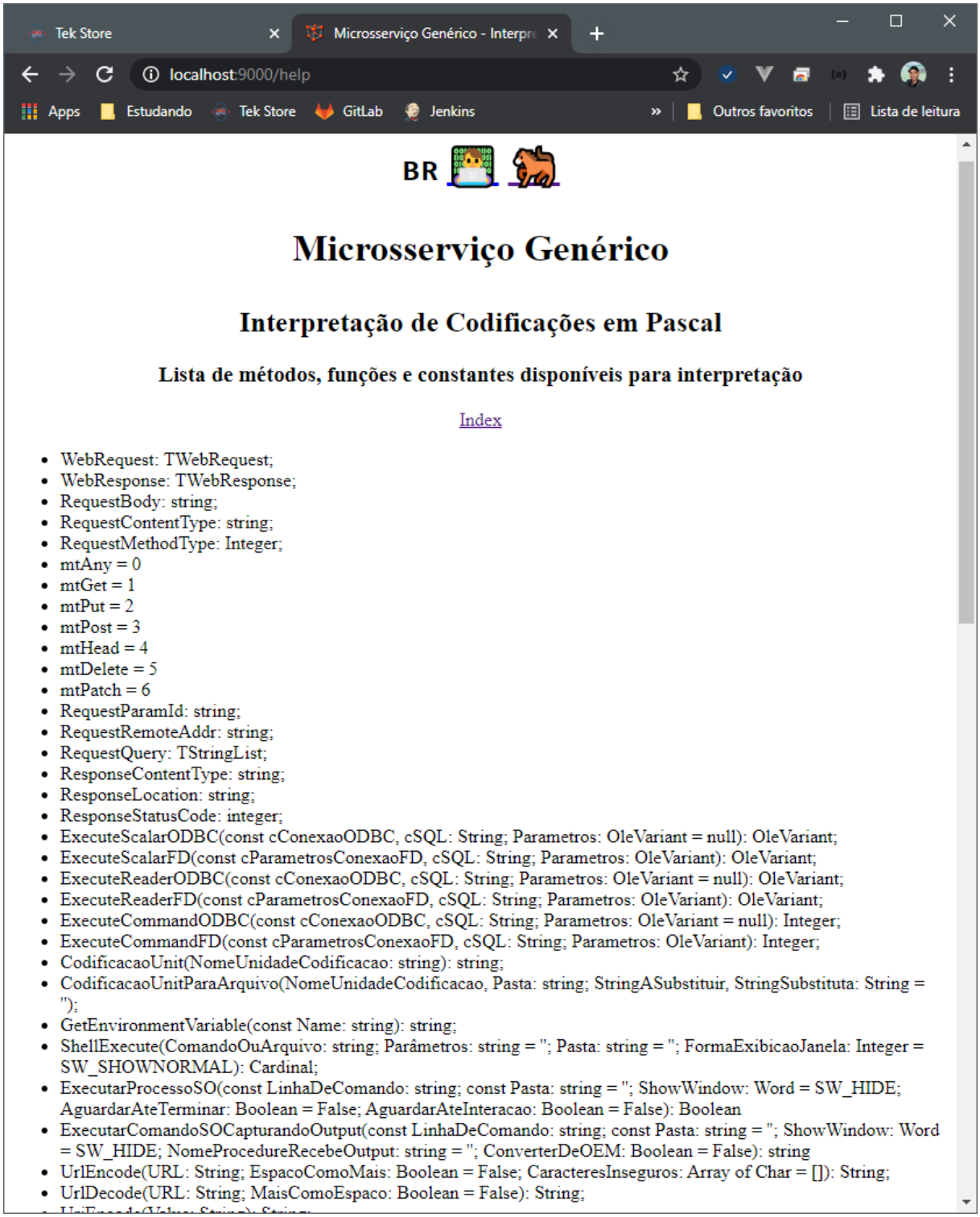
O link **Help**, com *endpoint* homônimo, apresenta uma lista de algumas das funções próprias que foram criadas e disponibilizadas na interpretação como uma espécie de incremento à linguagem Pascal original, veja Imagem 8. Dentre as funções específicas criadas para serem aceitas na interpretação seguem os grupos, para não tornar este trabalho extenso demais:

- Funções para manipulação de datas, números, *strings e SQL*.
- Funções para envio de e-mail.
- Funções do *Firedac*[37] para acesso genérico a bancos de dados.
- Funções para acesso a bancos de dados através de fontes de dados ODBC.
- Funções de manipulação de *TClientDataSet*[38].
- Funções de *Hash*.
- Funções para manipulação de conteúdo JSON e XML.
- Funções relacionadas a rede e acesso à web.
- Funções de RTTI.
- Funções relacionadas ao sistema operacional.
- Funções para geração e validação de *tokens* JWT.
- Funções para *encode* e *decode* em base64.
- Funções para manipulação de objetos de requisição e respostas HTTP.

[37] FireDAC é uma biblioteca de acesso universal a dados para o desenvolvimento de aplicativos para vários dispositivos, conectados a bancos de dados corporativos.
https://www.embarcadero.com/br/products/rad-studio/firedac

[38] TClientDataSet é um componente do Delphi que funciona como o personagem central da tecnologia de acesso a dados chamada MIDAS.

Imagem 8 - Help - Lista de funções aceitas pelo GenericMicroServiceDPR.



Fonte: Elaborado pelo próprio autor.

Os demais links **Documentação, Saúde**, **Testes Unitários**, **Autenticação**, **Publicidade** apontam, respectivamente, para os *endpoints* /doc, /heath, /test, /auth e /publicity. Os quais redirecionam para páginas especiais que foram configuradas

previamente. Desta forma cada microsserviço pode ter a sua página específica, mas todos devem implementar conforme foi acordado ou solicitado pela empresa.

O *endpoint* de documentação **/doc** redireciona a navegação para uma página especial na qual deve estar a documentação da API do microsserviço.

O *endpoint* de saúde **/health** redireciona a navegação para uma página especial que deve executar testes específicos e determinar se aquela instância está funcionando corretamente. Na maioria dos casos apenas uma checagem de resposta de requisição (*HttpStatusCode* 200) seria suficiente, mas em determinados casos, pode ser necessário, por exemplo, saber se a instância está se comunicando com o banco de dados ou com outros microsserviços dos quais dependa. Isto é extremamente útil, quando em ambientes corporativos e com grandes quantidades de instâncias de microsserviços em execução, para determinar se aquela instância ainda está operacional ou se precisa ser eliminada e o fluxo direcionado para outra instância disponibilizada em seu lugar.

Da mesma forma o *endpoint* **/test** redireciona para uma página onde podem ser realizados os testes unitários do microsserviço em questão. Esta página pode ser um menu com diversos testes individuais ou apenas o resultado da execução de diversos testes. Os desenvolvedores devem padronizar que todo microsserviço tenha a sua página de testes unitários conforme determina a gerência do setor ou empresa.

O *endpoint* **/auth** é um redirecionamento para uma página, neste ou em outro microsserviço, que fará a autenticação relacionada aos usuários e suas permissões.

Por fim, o *endpoint* de publicidade **/publicity** é um redirecionamento para uma página que faz a divulgação daquele microsserviço, podendo conter informações de propaganda, custos de licença, contato com desenvolvedores responsáveis, links para suporte etc.

As rotas de funcionamento não estão sendo exibidas na tela inicial porque requerem parâmetros ou outras informações para seu funcionamento. São os casos de /static, /public, /download, /upload e /interpretar.

Para ter acesso a *units* exatamente como são, sem realizar interpretação, pode-se utilizar a rota **/static** que fará buscas nos repositórios configurados. Isto é particularmente útil para arquivos dos tipos: HTML, CSS, JS, TXT, SQL, INI, JSON, XML, CSV, BAT ou CMD, os quais, pela própria característica, não faz sentido proteger a codificação, pois é necessária no próprio navegador ou sistema operacional. Em se tratando dos arquivos PAS, PY, JAVA, PHP esta rota proibirá a

exibição dos seus conteúdos, visto que suas codificações não devem ser acessíveis por questões de segurança.

A rota **/public** fornece os arquivos que estiverem na pasta /public e em suas subpastas ou no respectivo volume mapeado do contêiner. Neste caso, diferentemente da rota /static, não se fará busca em outros repositórios. É indicado para ser utilizado como um servidor de arquivos *frontend* para sites.

As rotas **/download** e **/upload** permitem realizar, respectivamente, download e upload de arquivos.

A rota **/interpretar**, que é o principal objetivo deste trabalho, aceitará algumas variações de chamadas, como pode ser visto na tabela abaixo.

No Quadro 2, pode-se visualizar um resumo das rotas e *endpoints* aceitos pelo GenericMicroServiceDPR:

Quadro 2 - Rotas e endpoints do GenericMicroServiceDPR com suas explicações.

Método HTTP: *endpoint / rota*	**Breve explicação**
GET: /	Apresenta informações básicas sobre o microsserviço e menu com rotas básicas e padronizadas.
GET: /config	Apresenta o arquivo de configuração que foi utilizado para reger aquela instância do GenericMicroServiceDPR.
GET: /config/reload	Recarrega o arquivo de configuração. Utilizado quando se faz modificação nas configurações e queira-se que o microsserviço passe a adotá-las, sem ser necessário reiniciá-lo. As modificações no atributo **horseOptions** só terão efeito após reinício do serviço.
GET: /help	Exibe uma pequena lista de funções e constantes específicas adicionadas à interpretação padrão do *Object Pascal*, bem como seus parâmetros e seus retornos.

GET: /doc	Redireciona para uma página especial configurada no atributo **paginasEspeciais.documentacao** que deve conter a documentação da API para aquele microsserviço específico.
GET: /health	Redireciona para uma página especial configurada no atributo **paginasEspeciais.saude** que deve validar a saúde ou estabilidade daquela instância específica do microsserviço.
GET: /test	Redireciona para uma página especial configurada no atributo **paginasEspeciais.testesUnitarios** que deve conter um menu com links para os diversos testes unitários que aquele microsserviço precisa atender ou a execução de todos os testes unitários com os seus respectivos resultados.
GET: /auth	Redireciona para uma página especial configurada no atributo **paginasEspeciais.autenticacao** que deve fornecer recursos para se autenticar um usuário.
GET: /publicity	Redireciona para uma página especial configurada no atributo **paginasEspeciais.publicidade** que pode fornecer uma espécie de propaganda sobre aquele microsserviço, inclusive informações de contato com a equipe ou custos para sua utilização.
GET: /favicon.ico	Retorna para o navegador um arquivo **favicon.ico** que deve estar na mesma pasta onde o GenericMicroServiceDPR

	está sendo executado. Navegadores utilizam este ícone nas abas em execução e nos atalhos quando o site é incluído na lista de favoritos.
GET: /download/:arquivo	Caso o microsserviço esteja configurado para aceitar downloads (atributo **permitirDownloads** da configuração), fornece o arquivo, caso exista na pasta /downloads, a quem o requisitou.
POST: /upload/:arquivo	Caso o microsserviço esteja configurado para aceitar uploads (atributo **permitirUploads** da configuração) faz o upload do arquivo, colocando-o na pasta /uploads e acrescentando um GUID ao seu nome para evitar conflitos e retorna o nome do arquivo criado no servidor.
GET: /static/:Unit.Extensao	Fornece arquivos estáticos HTML, CSS, JavaScript, TXT, CSV, JSON, XML, INI, SQL, BAT, CMD para que sejam interpretados pelo navegador ou sistema operacional. Esses arquivos podem estar no repositório local ou remoto. Não é permitido informar subpastas nesta rota.
GET: /public{/subpasta}{/subpasta}/arquivo.*	Fornece arquivos estáticos HTML, CSS, JavaScript, imagens, ícones, fontes etc. Esses arquivos devem estar obrigatoriamente na pasta /public acima da pasta de execução do GenericMicroServiceDPR. Não se procurará em repositórios remotos.
GET: /interpretar/:UnitEMetodo GET: /interpretar/:UnitEMetodo/:id PUT: /interpretar/:UnitEMetodo	Recebe a requisição para uma *unit* (unidade de codificação que contém código-fonte). Pode também ser

POST: /interpretar/:UnitEMetodo POST: /interpretar/:UnitEMetodo/:id HEAD: /interpretar/:UnitEMetodo DELETE: /interpretar/:UnitEMetodo DELETE: /interpretar/:UnitEMetodo/:id PATCH: /interpretar/:UnitEMetodo PATCH: /interpretar/:UnitEMetodo/:id	informado um método diferente de Main, separando-o por um ponto. Segue-se a sequência de passos executados: 1) Localiza a *unit* no repositório local, caso esteja configurado para tal (atributo **repositorioUnitsLocal** da configuração); no repositório TekStore, caso esteja configurado para tal (atributo **repositorioUnitsTekStore**) ou em outros repositórios públicos registrados no atributo **repositoriosUnitsPublicos**. 2) Repassa as informações da requisição como *MethodType*, *ContentType*, *RemoteAddr*, *ParamId*, *Query Params*, *Body* e outros para o interpretador. 3) Realiza a interpretação, podendo inclusive fazer referências a outras *units*. 4) Captura o resultado da interpretação e o devolve como *response* para a requisição. 5) Ajusta o *ContentType*, *StatusCode* e *Location* do response de acordo com o que foi disponibilizado pelo interpretador.

Fonte: Elaborado pelo próprio autor.

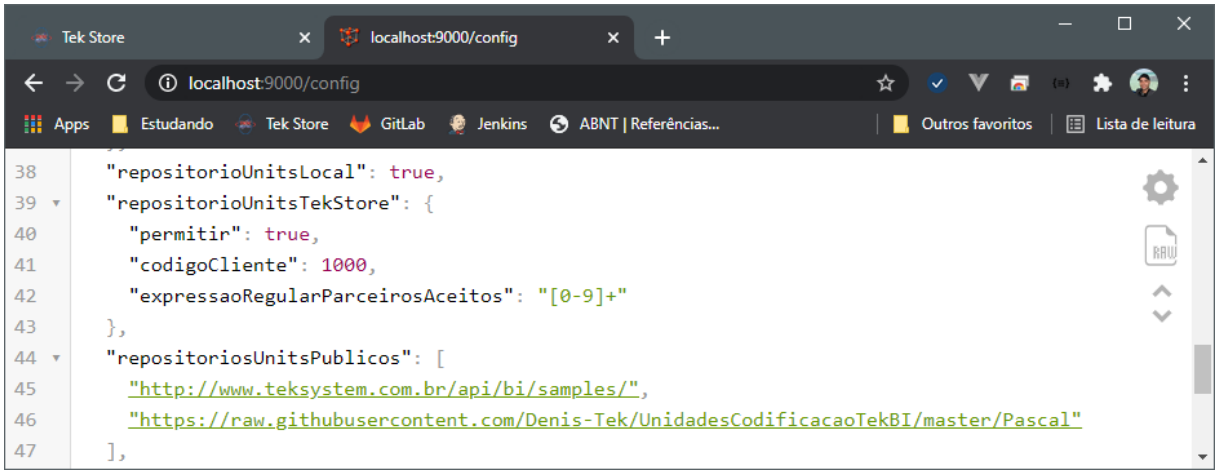
A estrutura do *endpoint* /interpretar, permite, por exemplo, criar uma API que suporte ações CRUD servindo-se dos métodos HTTP[39].

[39] Para ver mais detalhes sobre construção de um CRUD veja a seção 5.3 TESTES COM DIVERSOS MÉTODOS HTTP

5.2 TESTES COM MÚLTIPLOS REPOSITÓRIOS

Para analisar os resultados quanto à possibilidade de uso de diferentes repositórios, foi empregada uma configuração que acesse repositório local, TekStore, pasta web e GitHub, conforme Imagem 9.

Imagem 9 - Configuração com repositório local, TekStore, pasta web e GitHub.

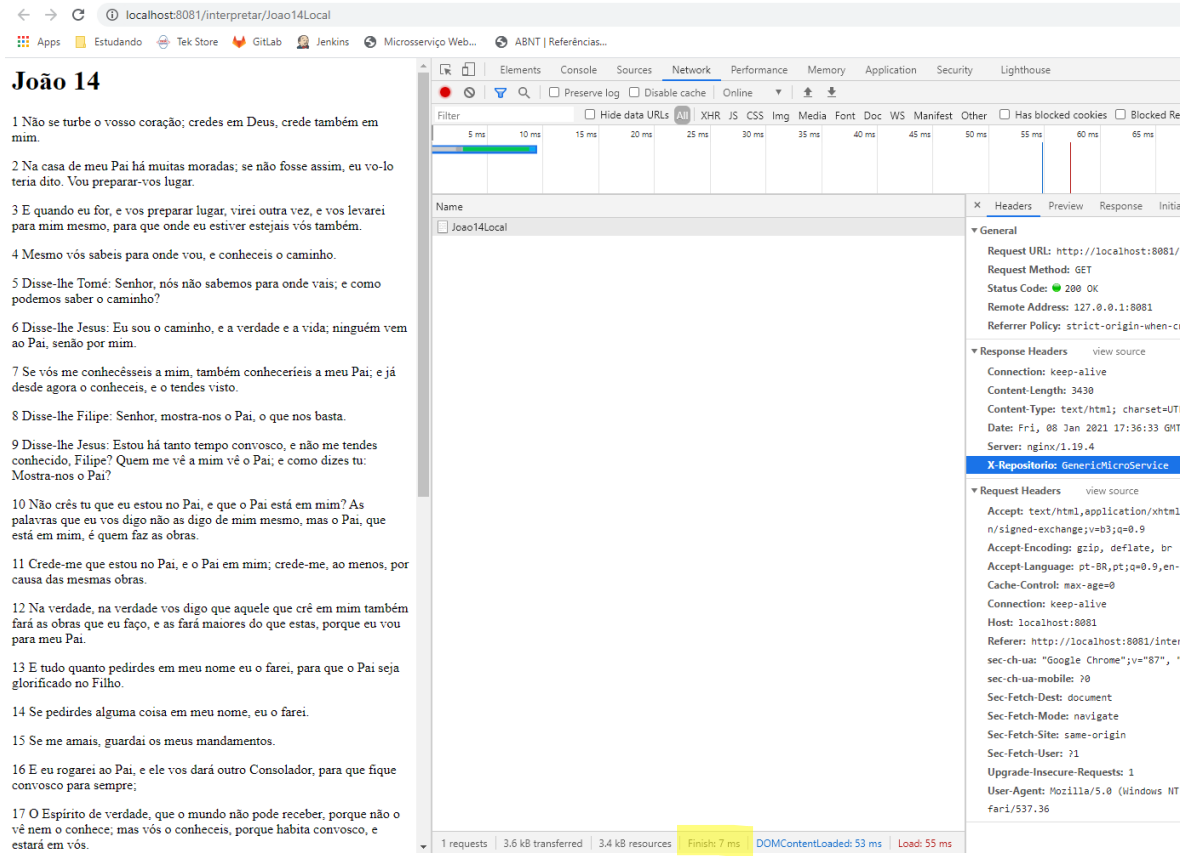


Fonte: Elaborado pelo próprio autor.

Foram obtidos os resultados que se seguem na Imagem 10, Imagem 11, Imagem 12 e Imagem 13. Observe que foi acrescentada uma identificação customizada no cabeçalho (*header*) de retorno **X-Repositorio** para comprovar onde o GenericMicroServiceDPR localizou a *unit* correspondente. Atentando-se para a métrica *Finish, já que* o tempo na métrica *Load* é maior por envolver o tempo de renderização no navegador que não deve ser considerado nesta comparação. Essas métricas encontram-se na barra central inferior das imagens.

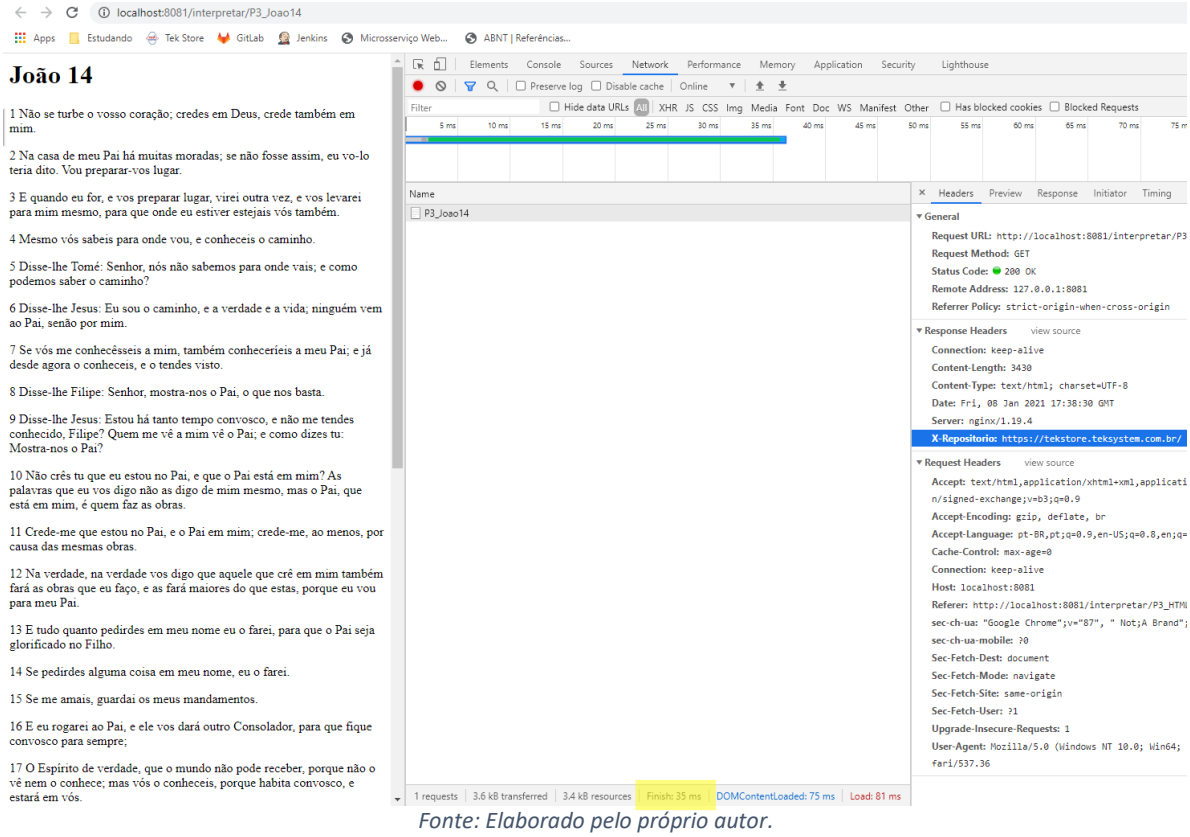
Foi utilizada rede *wireless* e internet via fibra ótica com velocidade nominal de 200 megabits por segundo.

Imagem 10 - Interpretação de unit local - 7ms.



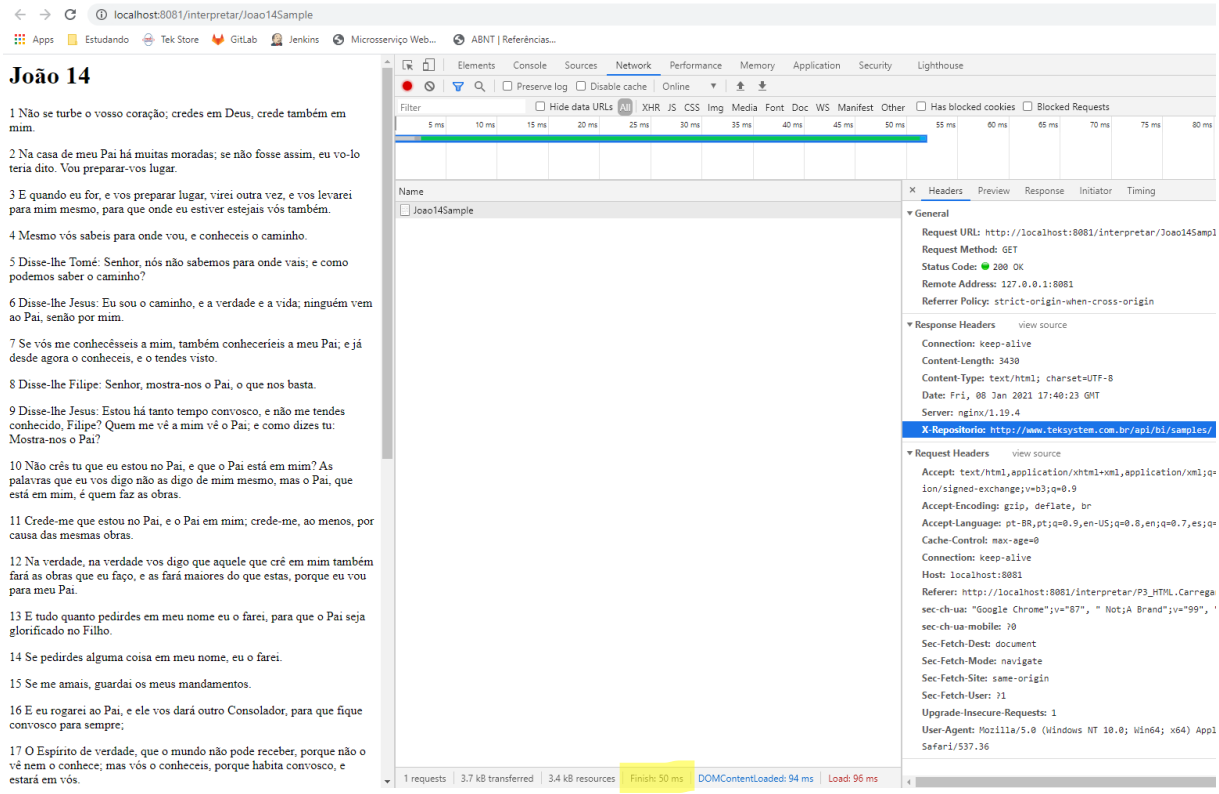
Fonte: Elaborado pelo próprio autor.

Imagem 11 - Interpretação de unit armazenada na TekStore - 35 ms.



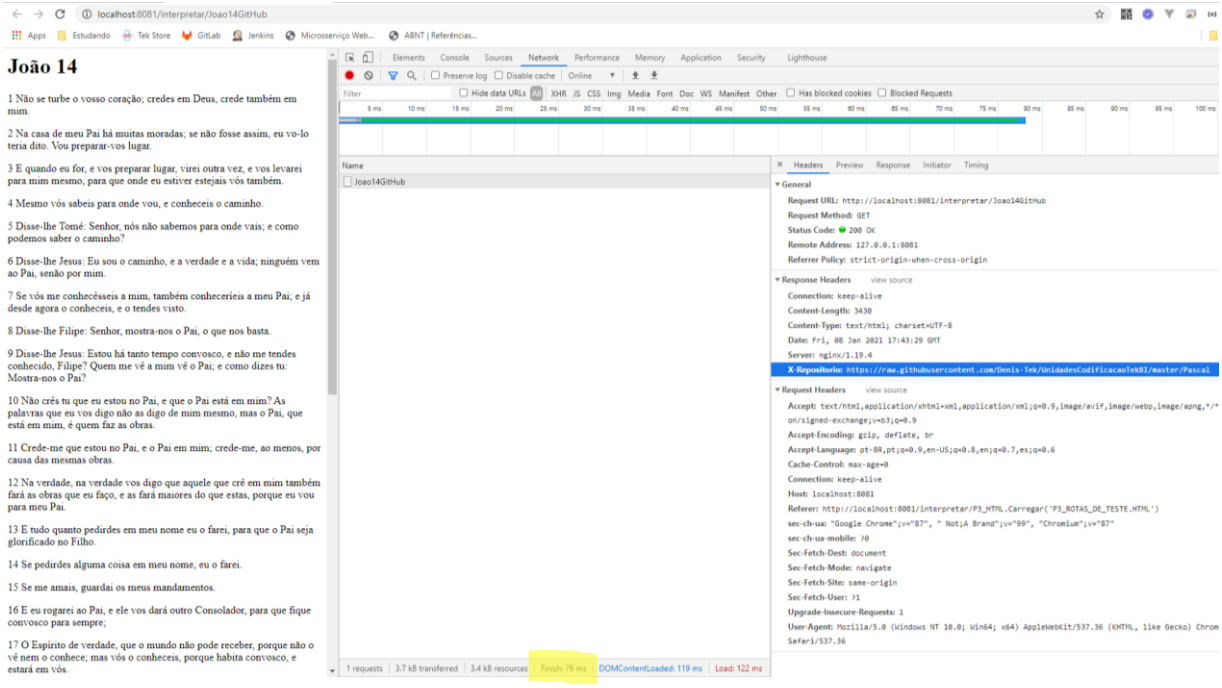
Fonte: Elaborado pelo próprio autor.

Imagem 12 - Interpretação de unit armazenada em pasta web - 50 ms.



Fonte: Elaborado pelo próprio autor.

Imagem 13 - Interpretação de unit armazenada no GitHub - 75 ms.

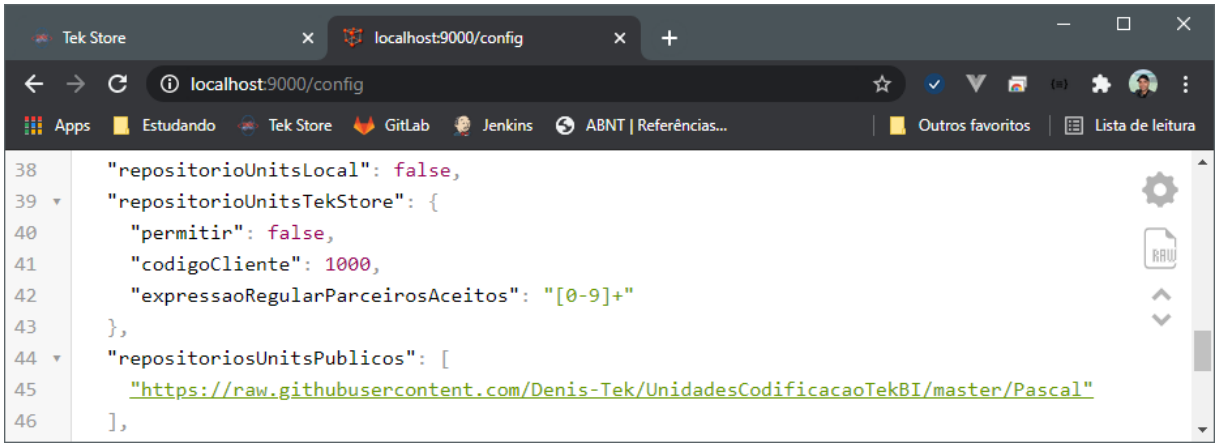


Fonte: Elaborado pelo próprio autor.

A diferença de tempo de resposta entre uma *unit* local e uma remota logicamente é bem perceptível (de 7 para 35 milissegundos) porque dependeu da rede *wireless* e internet para localizar o arquivo, mas a diferença entre os repositórios on-line (35, 50 e 75 milissegundos) não se dá porque os repositórios são mais lentos e sim porque a configuração obrigou a realização de tentativas de localização em cada repositório seguindo uma ordem específica. A fim de agilizar, pode-se desativar a possibilidade de se localizar em algum dos repositórios.

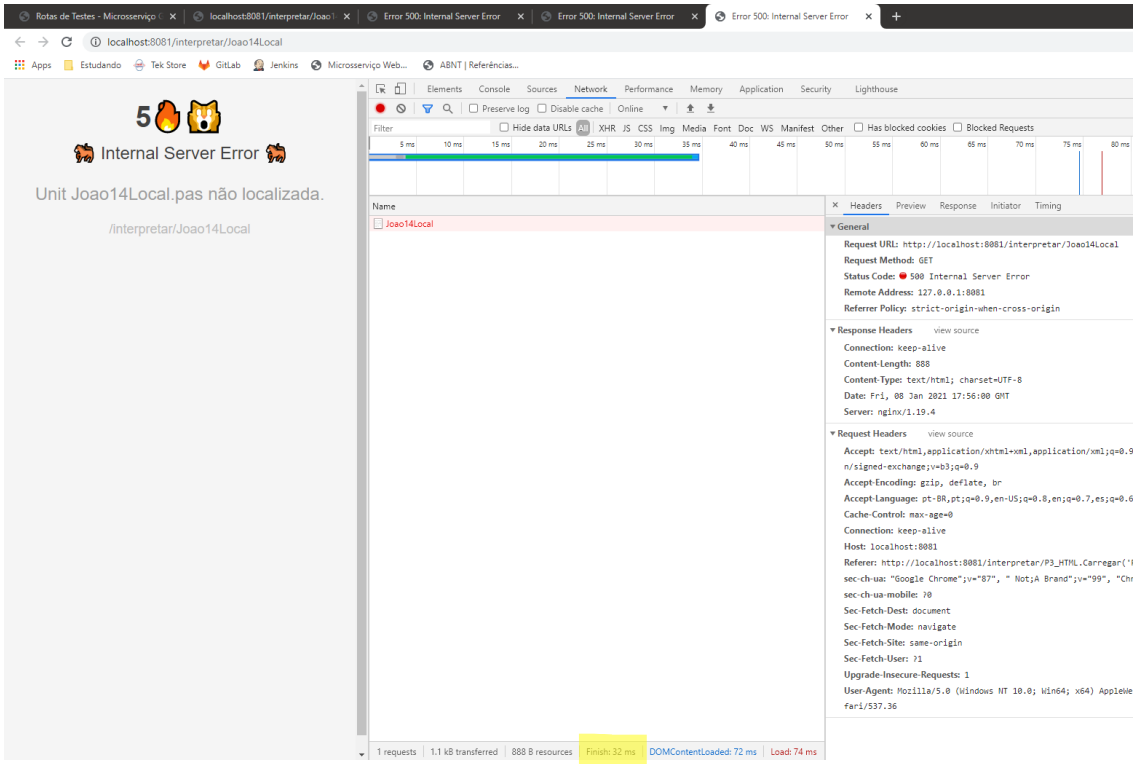
Na Imagem 14 é possível visualizar uma configuração utilizando apenas o GitHub como repositório. Devido a isto, é possível visualizar na Imagem 15, Imagem 16 e Imagem 17 o não reconhecimento das *units* armazenadas nos outros respectivos repositórios e também a execução mais rápida de *unit* armazenada no GitHub conforme Imagem 18.

Imagem 14 - Configuração trabalhando apenas com repositório GitHub.



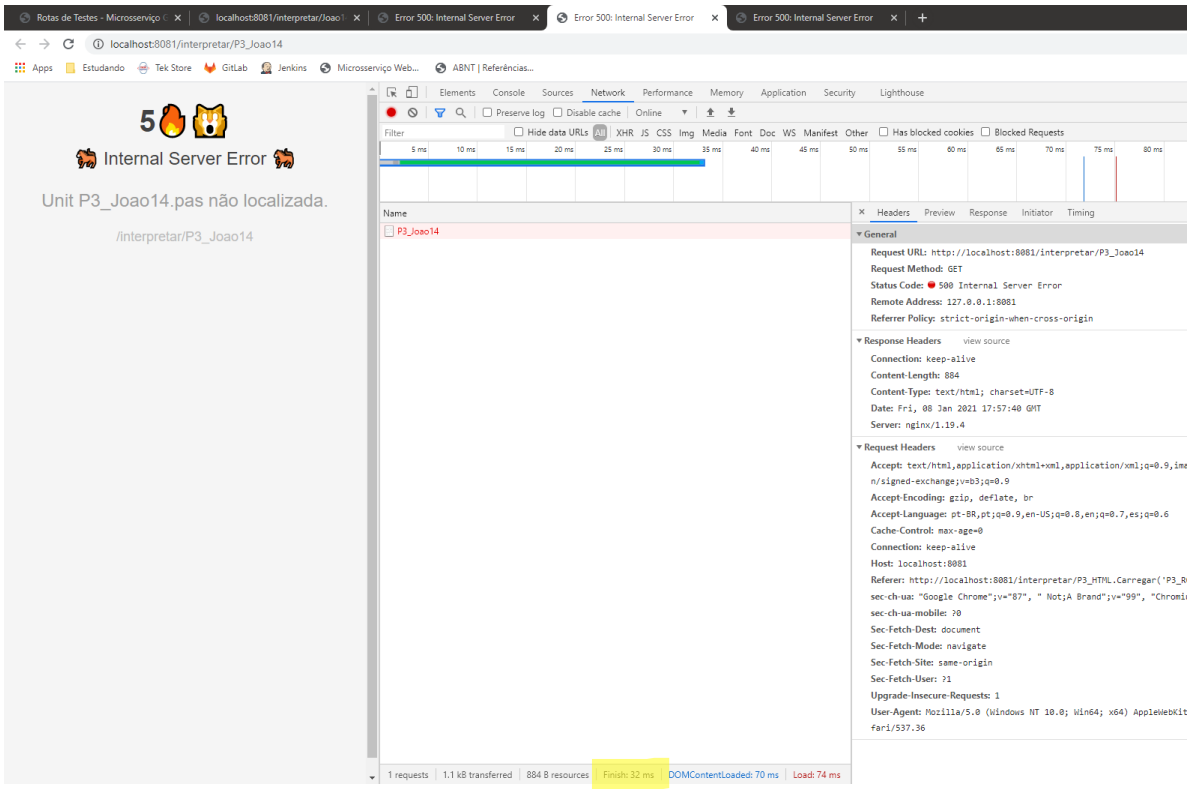
Fonte: Elaborado pelo próprio autor.

Imagem 15 - Não reconhecimento de unit local - 32 ms.



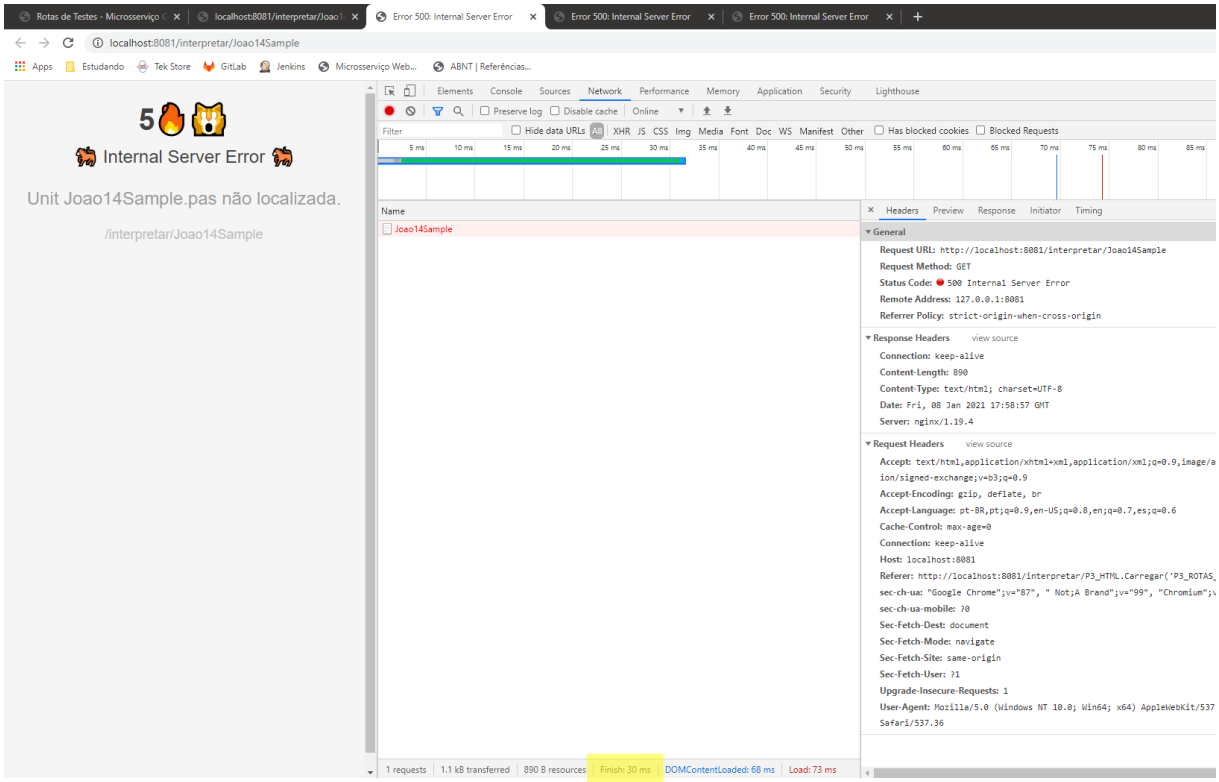
Fonte: Elaborado pelo próprio autor.

Imagem 16 - Não reconhecimento de unit armazenada na TekStore - 32 ms.



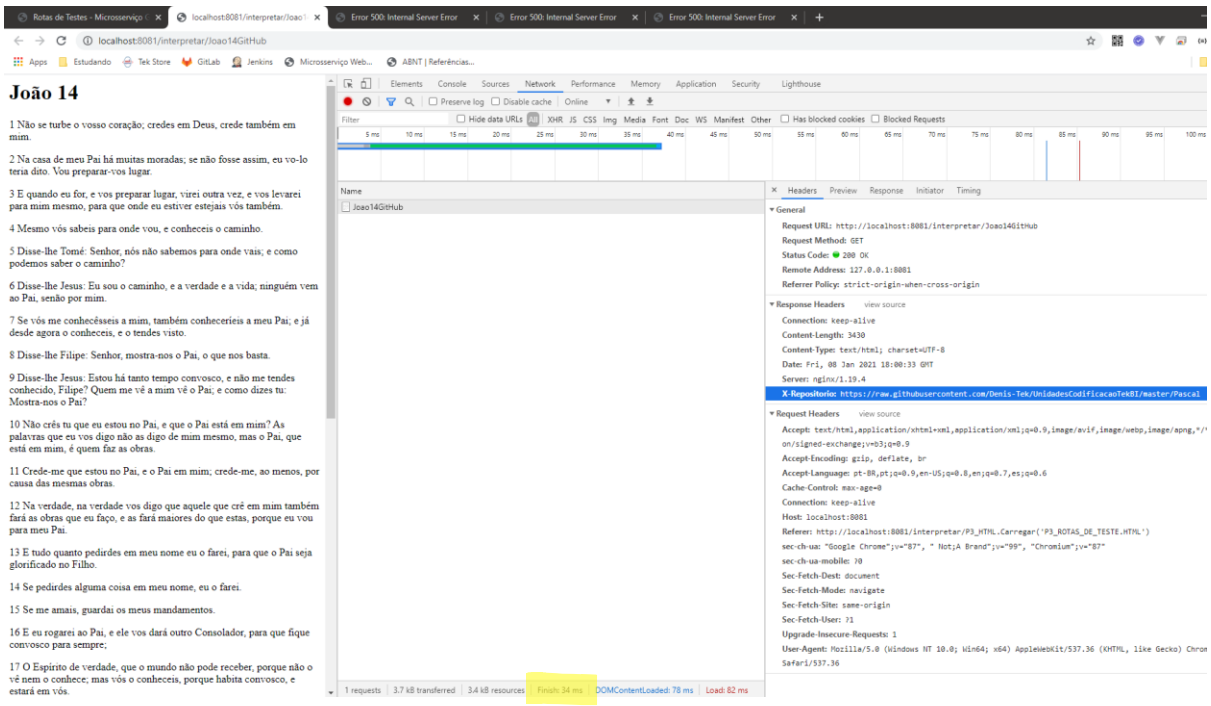
Fonte: Elaborado pelo próprio autor.

Imagem 17 - Não reconhecimento de unit armazenada em pasta web - 30 ms.



Fonte: Elaborado pelo próprio autor.

Imagem 18 - Interpretação de unit armazenada no GitHub – 34 ms.



Fonte: Elaborado pelo próprio autor.

Um resumo dos resultados obtidos pode ser visualizado no Quadro 3.

Quadro 3 - Resumo dos resultados quanto aos diversos repositórios.

Configuração utilizando repositórios local, TekStore, Web e GitHub	
Chamada a *unit* armazenada localmente	07 ms – Localizada
Chamada a *unit* armazenada na TekStore	35 ms – Localizada
Chamada a *unit* armazenada em pasta web	50 ms – Localizada
Chamada a *unit* armazenada no GitHub	75 ms – Localizada
Configuração utilizando somente o GitHub	
Chamada a *unit* armazenada localmente	32 ms – Não localizada
Chamada a *unit* armazenada na TekStore	32 ms – Não localizada
Chamada a *unit* armazenada em pasta web	30 ms – Não localizada
Chamada a *unit* armazenada no GitHub	34 ms – Localizada

Fonte: Elaborado pelo próprio autor.

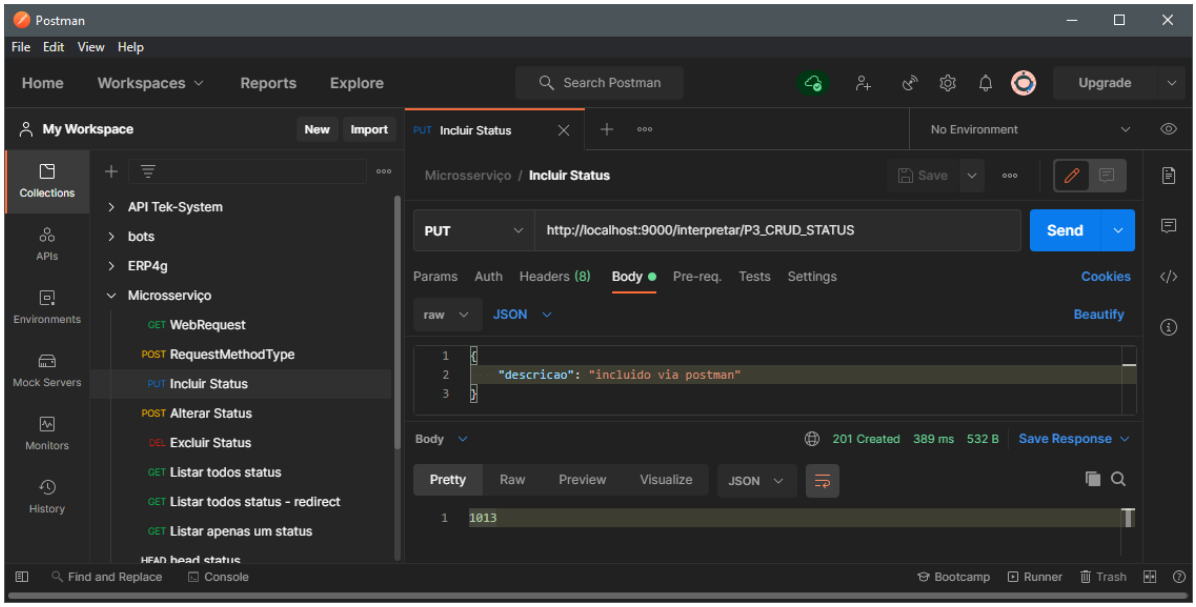
Observação: A não localização consome um tempo aproximado ao da localização, pois foi necessário realizar um acesso web ao GitHub para verificar se existe tal unidade de codificação armazenada lá, mesmo que não tenha baixado o arquivo e realizado a sua interpretação.

5.3 TESTES COM DIVERSOS MÉTODOS HTTP

Conforme proposto, foram realizados testes de funcionamento através da criação de uma API REST (*backend)* de um possível cadastro (CRUD) de status.

Na Imagem 19 é possível visualizar a API respondendo ao método **PUT** para incluir registro no banco de dados. Foram informados os dados JSON através do *Body*. O retorno foi 201 *Created,* trazendo o código do registro incluído, mas poderia trazer todos os dados do registro incluído, se assim fosse desejado.

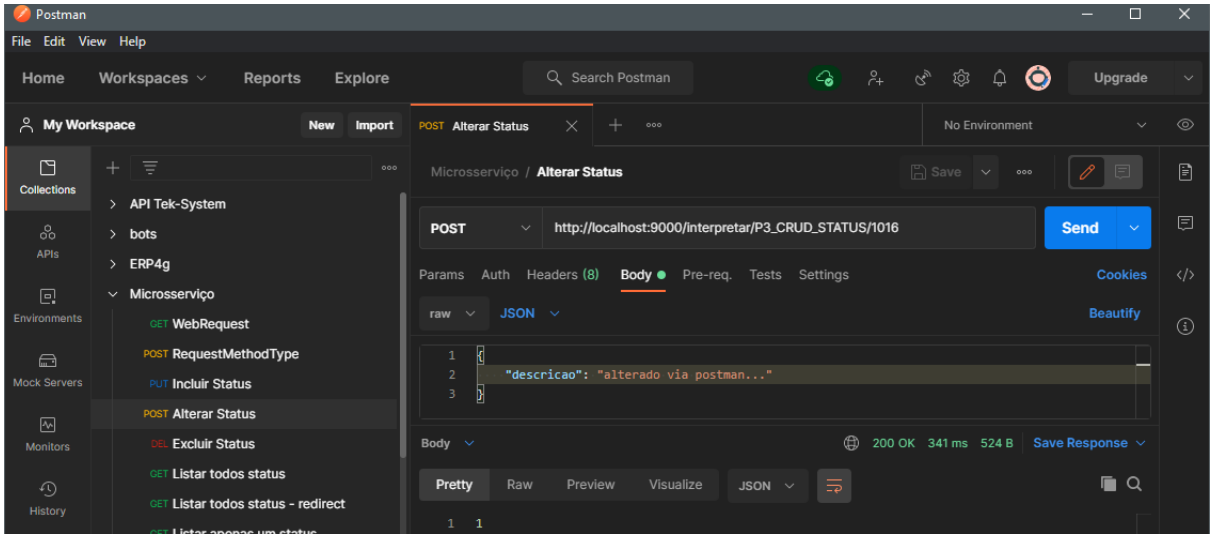
Imagem 19 - API respondendo requisição para método PUT.



Fonte: Elaborado pelo próprio autor.

Na Imagem 20 é possível visualizar a API respondendo ao método **POST** para alterar registro no banco de dados. Foram informados os dados JSON através do *Body*. O retorno foi 200 Ok, trazendo a quantidade de registros alterados, mas poderia trazer todos os dados do registro alterado, se assim fosse desejado.

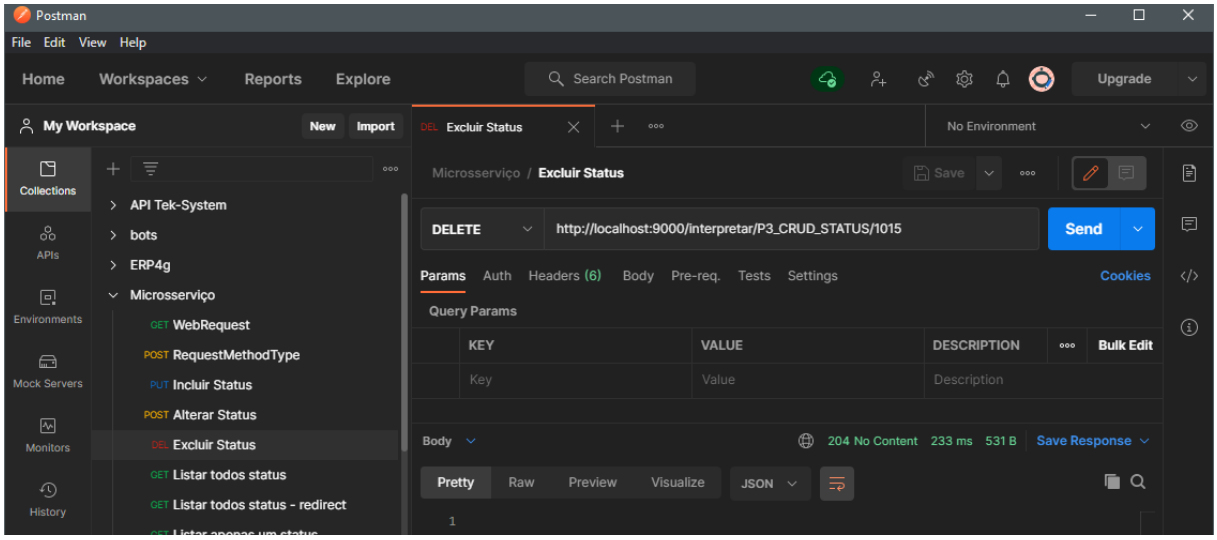
Imagem 20 - API respondendo requisição para método POST.



Fonte: Elaborado pelo próprio autor.

Na Imagem 21 é possível visualizar a API respondendo ao método **DELETE** para excluir registro no banco de dados. Foi informado o código do registro a ser excluído através da própria URL. Seria possível também informar por *query params*. O retorno foi 204 *No Content*.

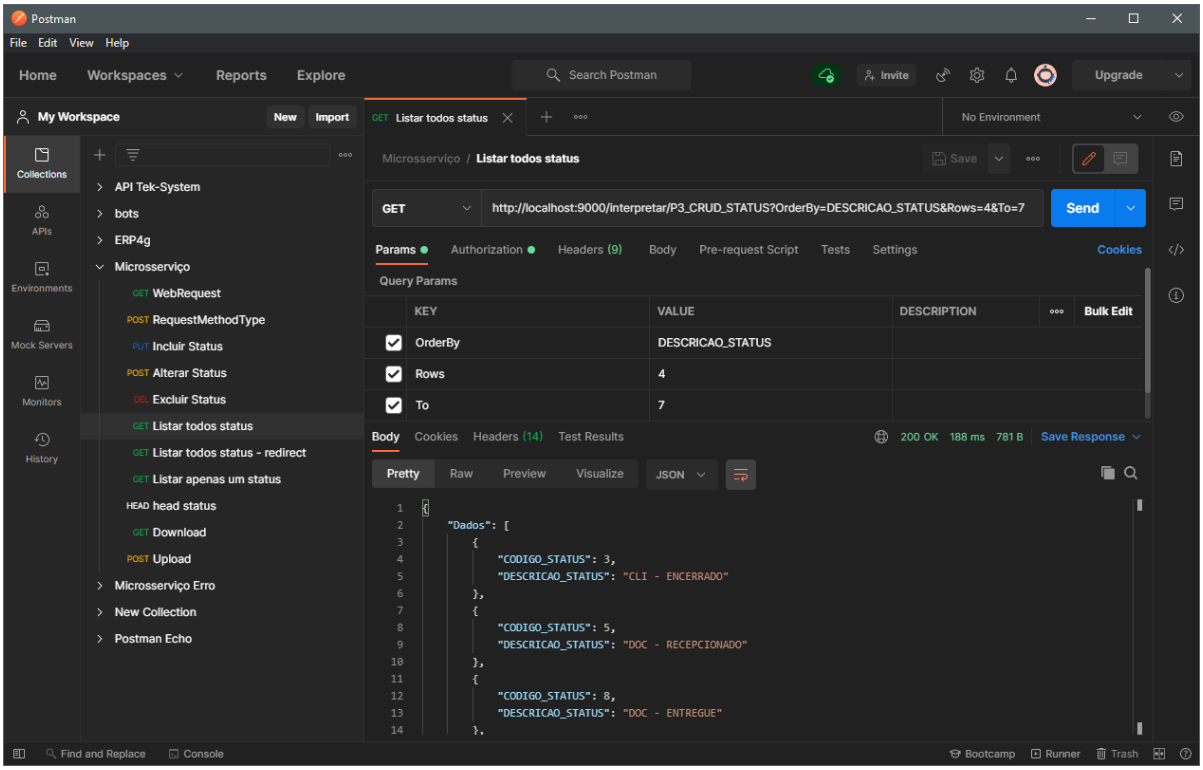
Imagem 21 - API respondendo requisição para método DELETE.



Fonte: Elaborado pelo próprio autor.

Na Imagem 22 visualiza-se a resposta da API ao método **GET** listando registros do banco de dados. Foi informado, através de *query params,* para a API realizar a ordenação dos registros e paginação. A resposta está no formato **JSON** conforme foi solicitado através do *header Content-Type: application/json*.

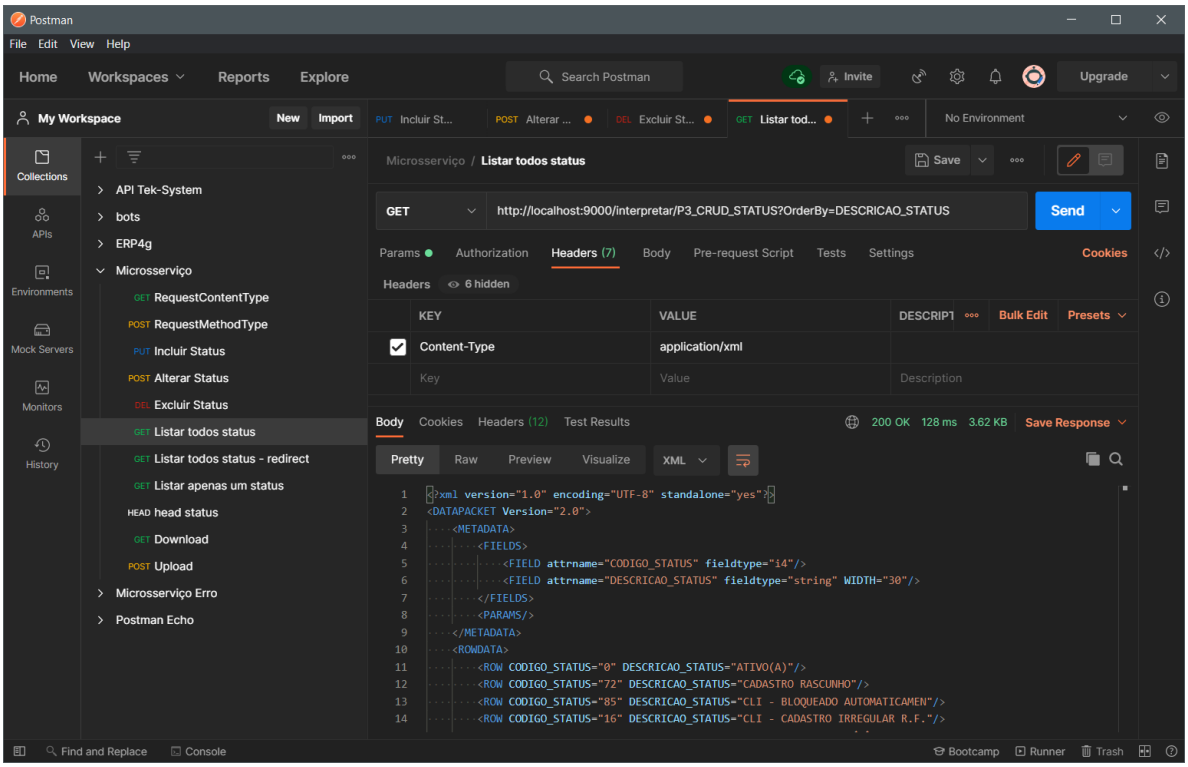
Imagem 22 - API respondendo requisição para método GET - Content-type JSON.



Fonte: Elaborado pelo próprio autor.

Na Imagem 23 visualiza-se a resposta da API ao método **GET,** porém desta vez com *header Content-Type: application/xml*. Os dados são os mesmos, mas a informação está no formato de **XML**.

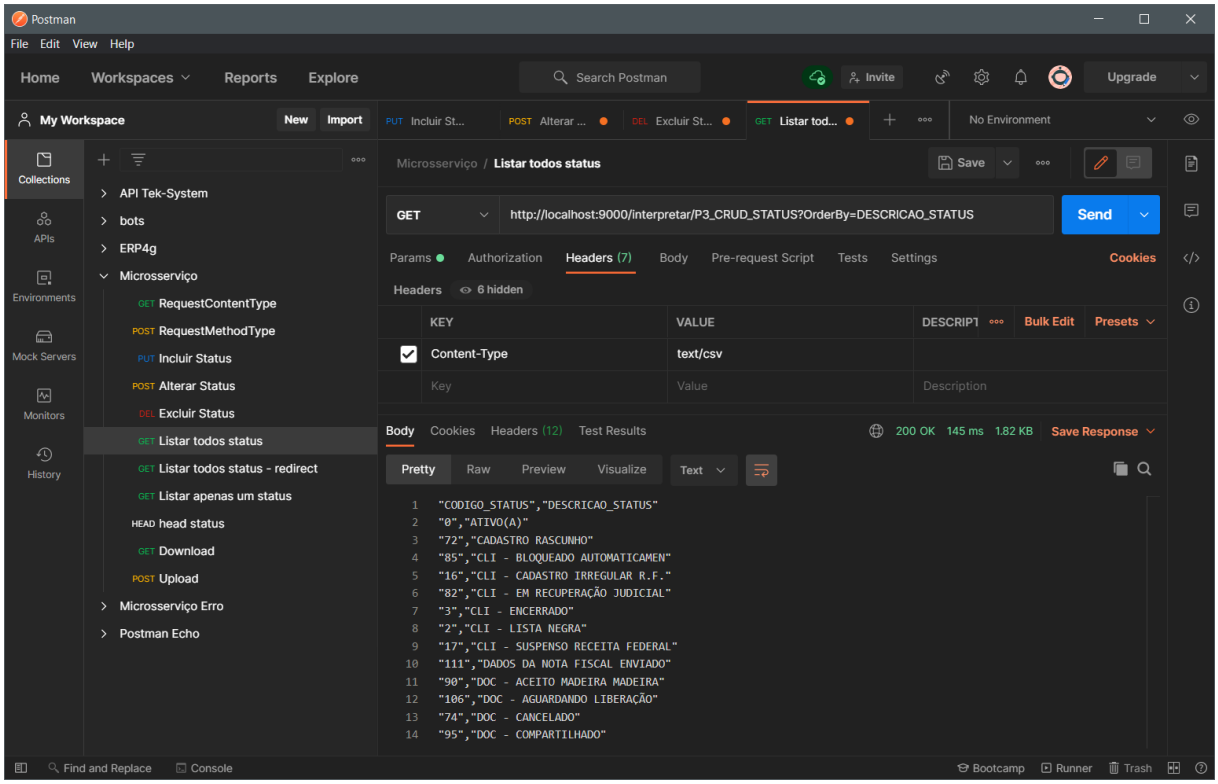
Imagem 23 - API respondendo requisição para método GET - Content-Type XML.



Fonte: Elaborado pelo próprio autor.

Na Imagem 24 visualiza-se a resposta da API ao método **GET,** porém desta vez com *header Content-Type: text/csv*. Os dados são os mesmos, mas a informação está no formato de **CSV**. Desta forma pode-se observar a flexibilidade da API construída.

Imagem 24 - API respondendo requisição para método GET - Content-Type CSV.



Fonte: Elaborado pelo próprio autor.

Esta API REST foi construída com apenas dois arquivos:

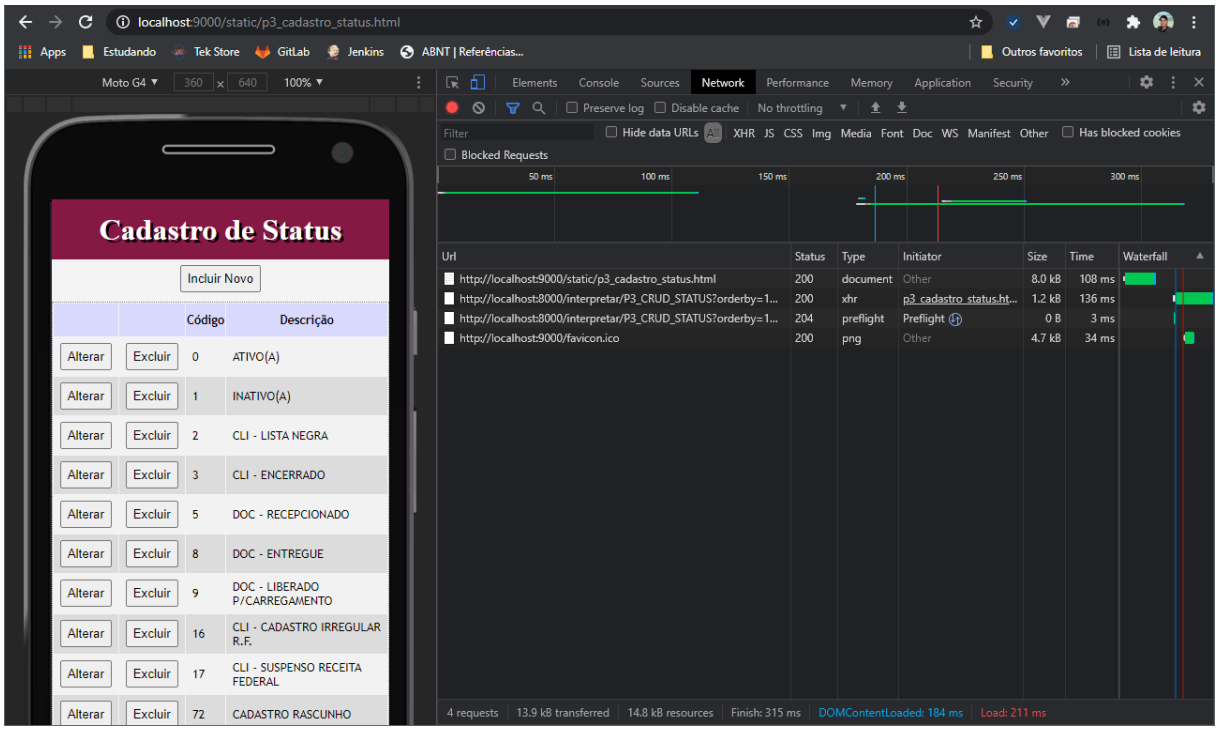
- P3_CRUD.pas[40] – codificação base para suportar qualquer CRUD.
- P3_CRUD_STATUS.pas[41] - configurações e referência ao P3_CRUD.

Havendo necessidade de construção de outros CRUD seria necessário apenas duplicar esta última *unit* ajustando as constantes, bem como a lista de campos e validações.

[40] A codificação pode ser lida no APÊNDICE A - Codificação P3_Crud.pas
[41] A codificação pode ser lida no APÊNDICE B - Codificação P3_Crud_Status.pas

A Imagem 25 mostra uma página dinâmica em execução que permite realizar as operações de inclusão, alteração, exclusão, listagem e ordenação. Esta página contém HTML, CSS e JavaScript que vieram como resposta à chamada ao *endpoint* /static/P3_Cadastro_Status.html[42] de uma instância do microsserviço em execução na porta 9000. Para consumo da API REST, o JavaScript contido nesta página está realizando requisições ao *endpoint* /interpretar/P3_Crud_Status de uma instância do microsserviço em execução na porta 8000. Em outras palavras, um microsserviço fornecendo arquivos estáticos (*frontend)* e outro atendendo como uma API REST (*backend)*.

Imagem 25 - Frontend - Cadastro de Status em execução.



Fonte: Elaborado pelo próprio autor.

[42] A codificação pode ser lida no APÊNDICE C - Codificação P3_Cadastro_Status.html (Frontend do cadastro)

5.4 TESTES DE DESEMPENHO SOB ESTRESSE

Os testes de desempenho foram executados em um Notebook Dell Vostro 5481 com processador Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz 1.99 GHz, 16 GB de RAM, com Windows 10 Pro versão 20H2, compilação 19042.804. Foram feitas dez mil requisições para a Codificação 5, usando a configuração massiva, com três threads cada e *delay* de um milissegundo.

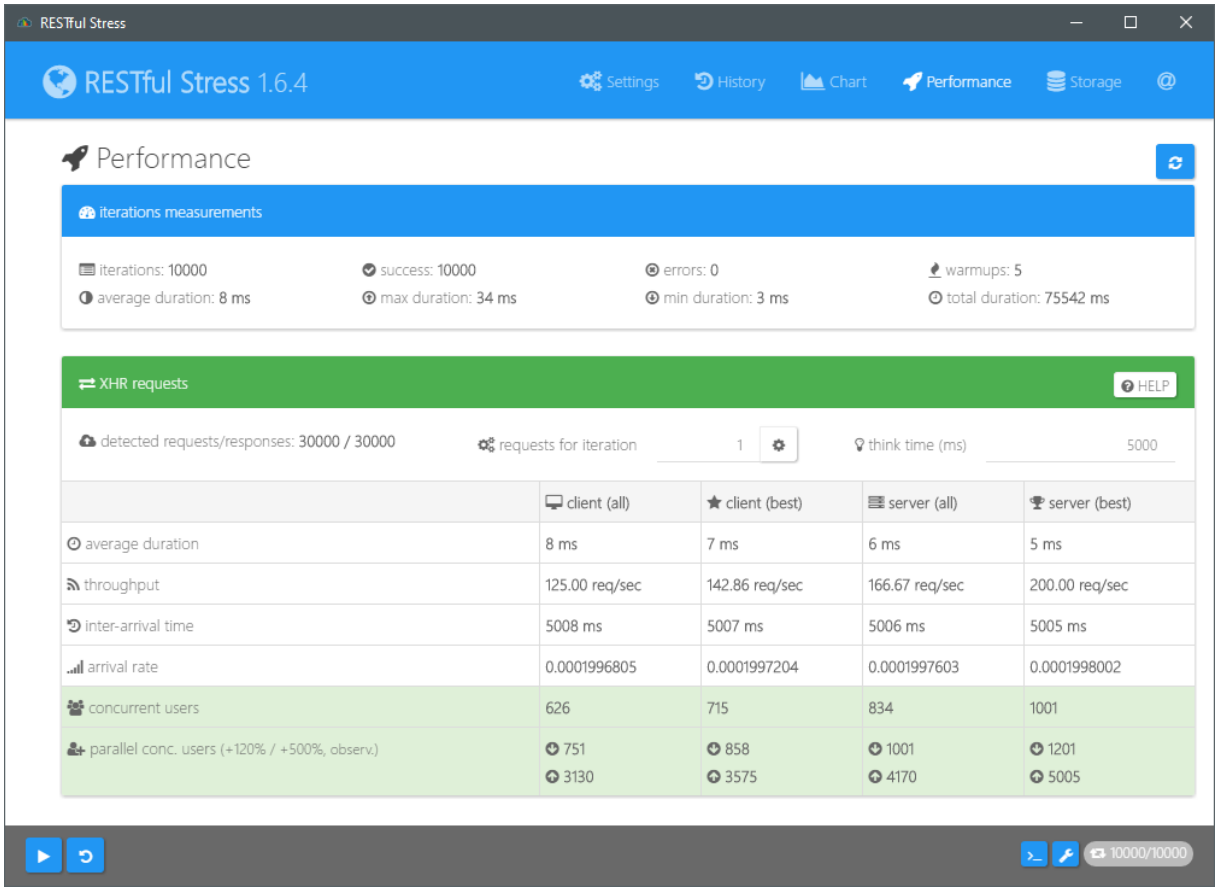
Codificação 5 - HelloWorld.pas - retornando texto plano.

```
unit HelloWorld;

function Main: string;
begin
  ResponseContentType := 'text/plain';
  Result := 'Hello World';
end;

end.
```

Fonte: Elaborado pelo próprio autor.

Uma instância do GenericMicroServiceDPR ao processar esse *HelloWorld* local conseguiu interpretar com uma média de 8 milissegundos de duração, atendendo entre 125 e 200 requisições por segundo, concluindo o processo em 75.542 milissegundos conforme Imagem 26. Lembrando que cada requisição atendeu a 3 chamadas em thread.

Imagem 26 - Teste de estresse - Interpretação no GenericMicroServiceDPR.



Fonte: Elaborado pelo próprio autor.

Valores bem semelhantes foram obtidos com requisições a rotas compiladas usando Java 8 / SpringBoot.

Usando o NodeJS versão 14.15.5, sem carregar um arquivo do disco, conforme pode ser visto na Codificação 6, foi possível atender com uma média de 6 milissegundos de duração, a 166 requisições por segundo, concluindo o processo em 64.911 milissegundos conforme Imagem 27.

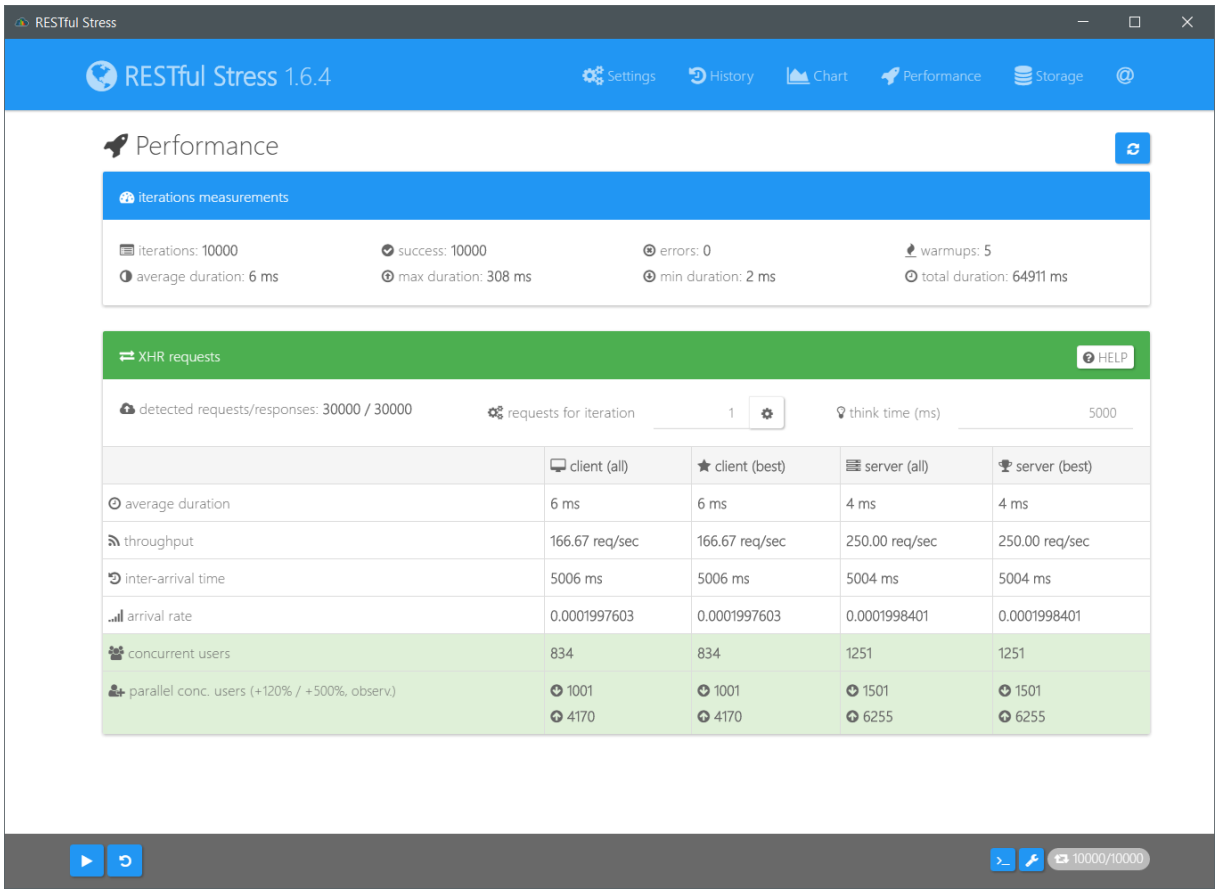
Codificação 6 - HelloWorld.js - retornando texto plano.

```
const http     = require('http');
const hostname = '127.0.0.1';
const port     = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
```

```
  res.end('Hello World');
});

server.listen(port, hostname, () => {
  console.log(`Servidor rodando em http://${hostname}:${port}/`);
});
```

Fonte: Elaborado pelo próprio autor.

Imagem 27 - Teste de estresse - Requisição simples no NodeJS.



Fonte: Elaborado pelo próprio autor.

Valores semelhantes foram obtidos com uma rota fixa (compilada) com a codificação embutida no GenericMicroServiceDPR.

No entanto, o NodeJS não precisava carregar um arquivo do disco a cada iteração, pois a rota (*endpoint*) mapeada já estava em memória. Para tentar aproximar as métricas, foi modificado o arquivo *HelloWorld* para que o NodeJS carregasse também o mesmo arquivo do disco, porém não interpretando seu conteúdo.

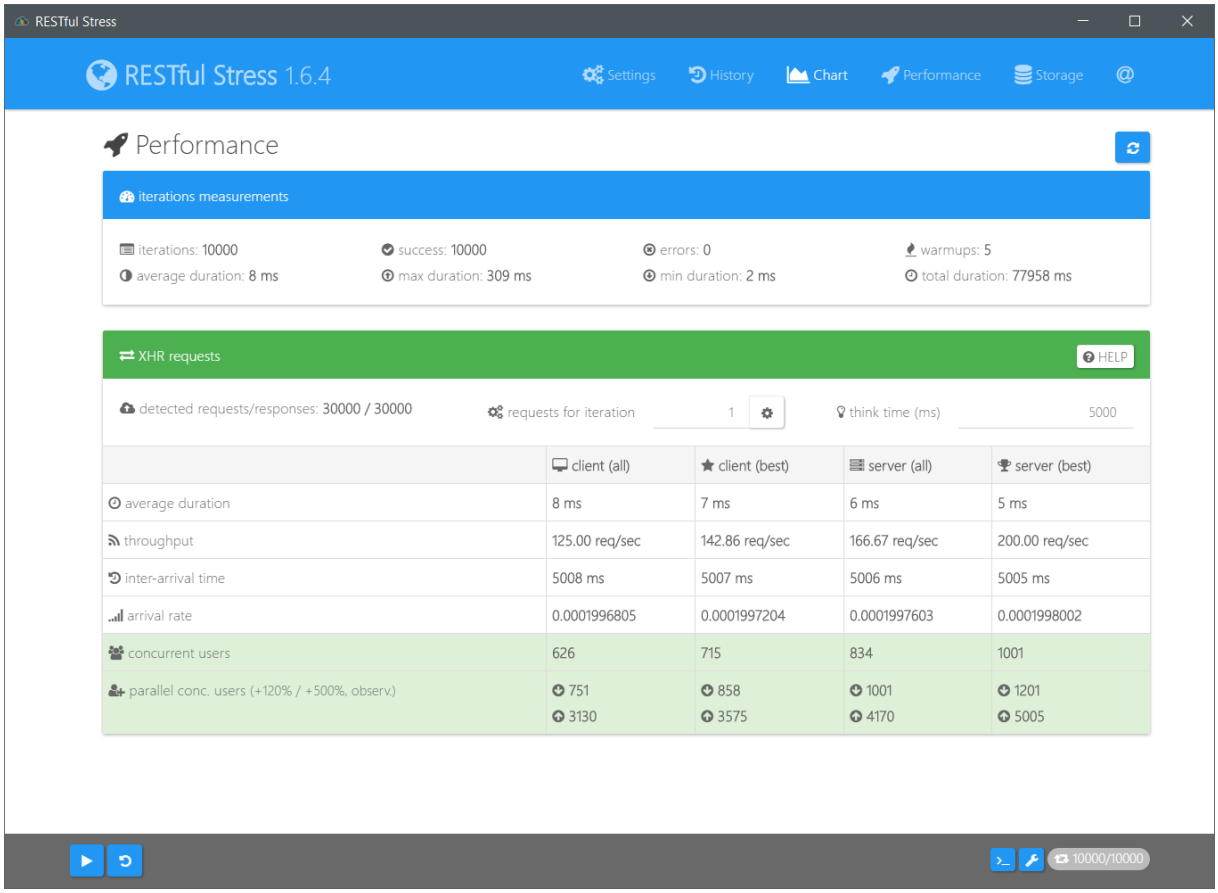
Nesta nova simulação, conforme Codificação 7, o NodeJS conseguiu responder em média com 8 milissegundos, atendendo entre 125 e 200 requisições por segundo, com tempo total de 77.958 milissegundos (Imagem 28), um pouco abaixo do GenericMicroServiceDPR.

Codificação 7 - HelloWorld.js - ajustado para carregar arquivo do disco.

```
const http = require('http');
const fs   = require('fs');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  fs.readFile('./HelloWorld.pas', 'utf-8', function (err, data) {
    if(err) throw err;
  });
  res.end('Hello World');
});

server.listen(port, hostname, () => {
  console.log(`Servidor rodando em http://${hostname}:${port}/`);
});
```

Fonte: Elaborado pelo próprio autor.

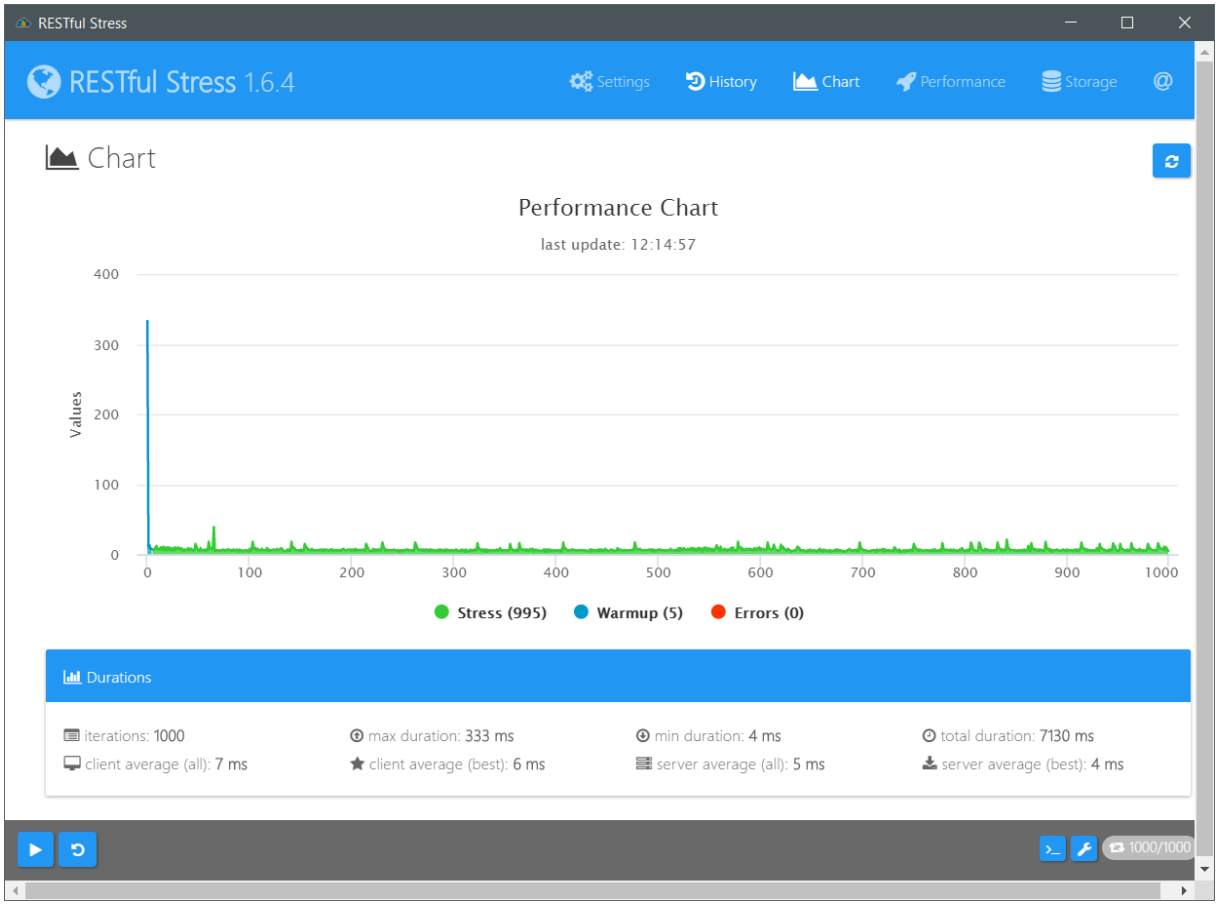
Imagem 28 - Teste de estresse - Requisição carregando arquivo no NodeJS.



Fonte: Elaborado pelo próprio autor.

O NodeJS pareceu usar alguma espécie de cache, pois ao se observar o gráfico das requisições (Imagem 29) verifica-se que as primeiras são sempre mais lentas, levando a duração máxima para 333 milissegundos.

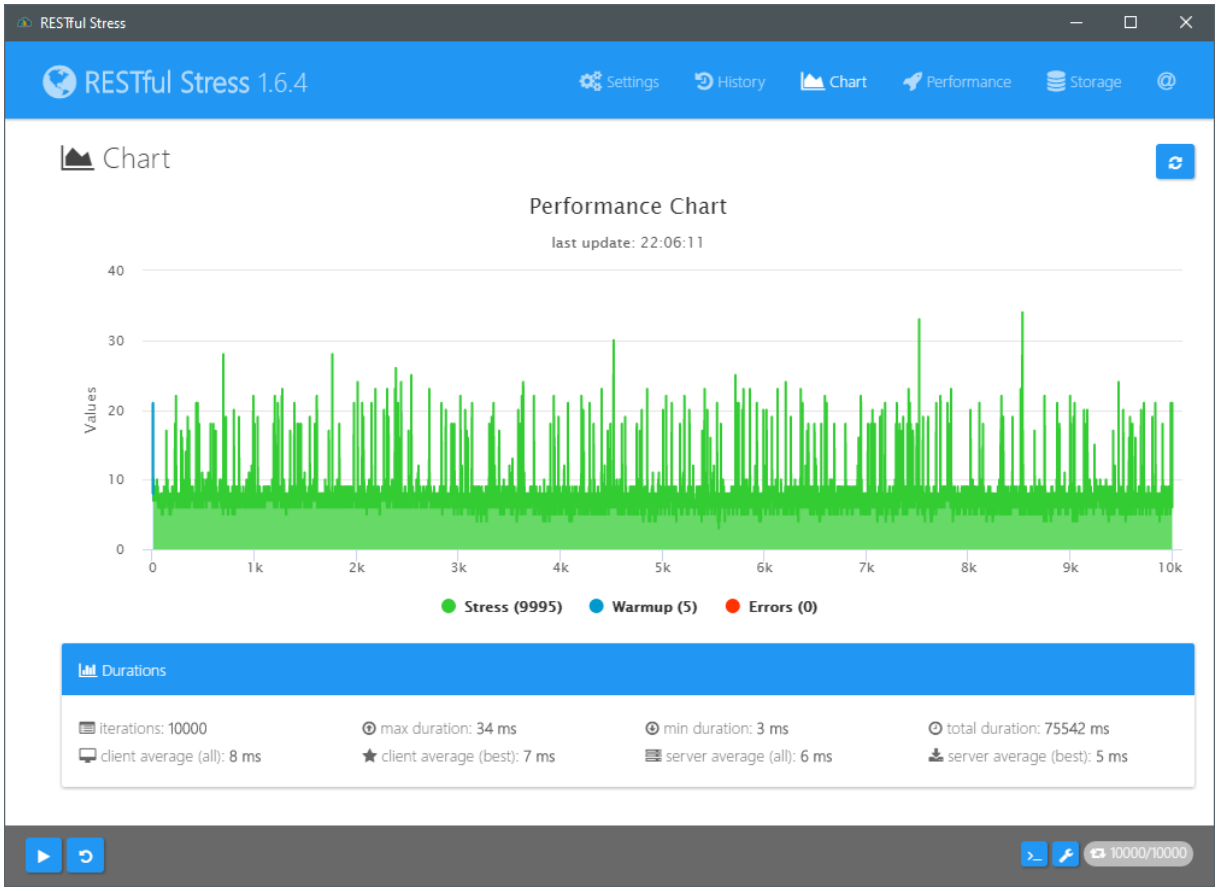
Imagem 29 - Performance Chart do NodeJS.



Fonte: Elaborado pelo próprio autor.

Isso não acontece com o GenericMicroServiceDPR, cuja média foi de 8 milissegundos, mas a máxima não passou de 34 milissegundos (Imagem 30) e a mínima foi de 3 milissegundos. Vale destacar também que o NodeJS é monolítico ao atender as requisições, enquanto o GenericMicroServiceDPR é *multithread* devido ao uso do Horse.

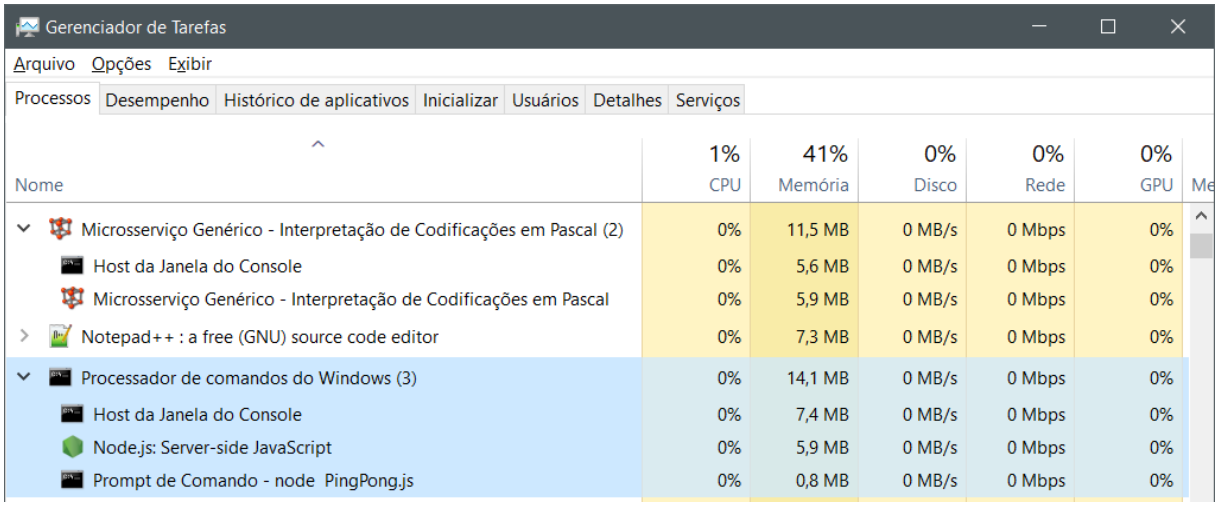
Imagem 30 - Performance Chart do GenericMicroServiceDPR.



Fonte: Elaborado pelo próprio autor.

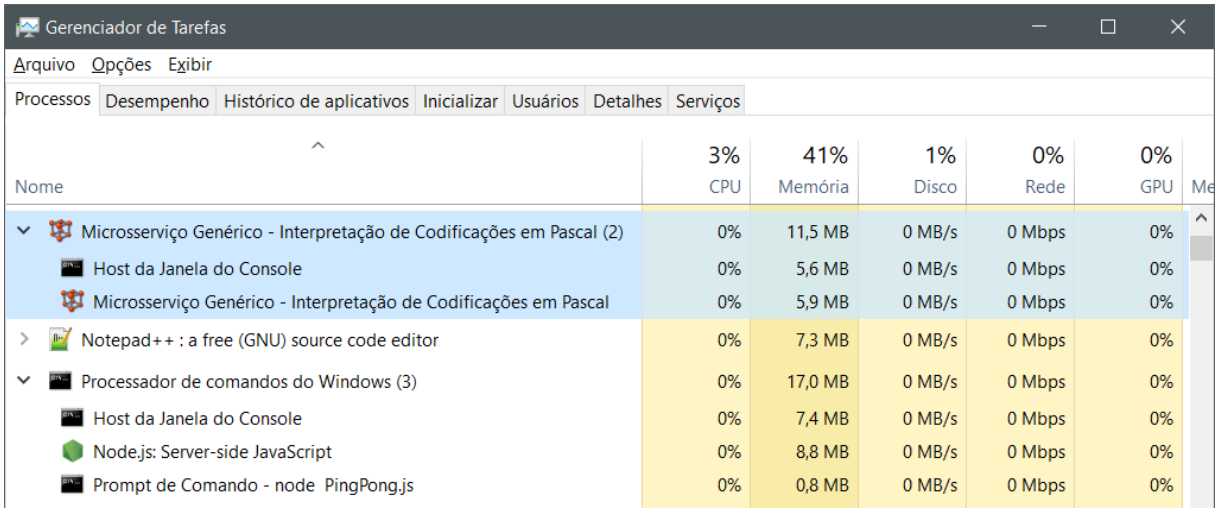
Esta percepção se torna mais clara quando se observa o consumo de memória de ambos, antes e após a execução deste último teste de estresse. O GenericMicroServiceDPR manteve o consumo de memória (5,9 Mb), indicando que destruiu todos os objetos que criou. Ao passo que o NodeJS aumentou o seu consumo de memória, de 5,9 Mb para 8,8 Mb, indicando que manteve em memória algo que tenha criado. O consumo de memória antes e após a execução desses processos, pode ser observado na Imagem 31 e Imagem 32.

Imagem 31 - Consumo de memória antes do teste de stress.



Fonte: Elaborado pelo próprio autor.

Imagem 32 - Consumo de memória após o teste de stress.



Fonte: Elaborado pelo próprio autor.

O incremento de memória no NodeJS ao final dos testes também foi observado quanto maiores fossem o número das interações.

Tanto o NodeJS quanto o GenericMicrosserviceDPR mantiveram os níveis de memória e de processamento em CPU baixos durante a execução dos testes, diferentemente do Java que demandou um consumo inicial de 46 Mb e encerrou com 52,2 Mb consumindo mais CPU e, consequentemente, mais energia.

5.5 TESTE DE GERAÇÃO DE PÁGINA HTML COMPLEXA

Na Imagem 33 é apresentada uma página web fruto da sequência de passos descritos abaixo, após receber a requisição:

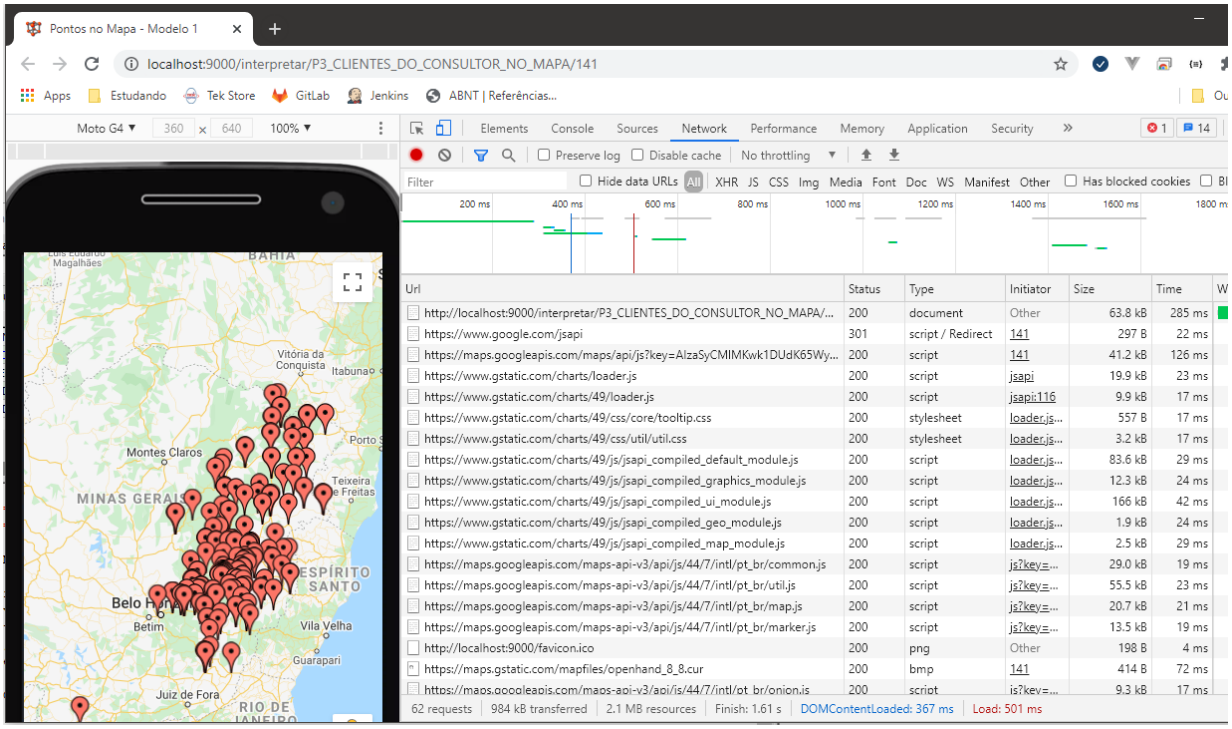
- Baixar *unit* P3_CLIENTES_DO_CONSULTOR_NO_MAPA[43] do repositório TekStore;
- Iniciar a interpretação;
- Baixar *unit* TEK_MAPS_GOOGLE do repositório TekStore, pois foi feita referência a ela. Esse é um bom exemplo de *framework* que poderia ser compartilhado por diversos microsserviços;
- Continuar a interpretação;
- Efetuar a busca de informações de clientes de um consultor de vendas no banco de dados Firebird[44];
- Encerrar o processamento da lógica para montar o HTML;
- Retornar o HTML para o navegador (63.8kb em 285ms);
- Navegador realizar a renderização, baixando todas as dependências da Google (CSS, XML, Fontes, JS, Imagens).

O tempo total de carga da página, tendo esvaziado o cache anteriormente: 501 milissegundos. Desses, 285 milissegundos para os dois acessos à TekStore, interpretação das unidades de codificação e acesso ao banco de dados Firebird. O restante, 216 milissegundos, para carregamento dos mapas.

Note que foi passado o código do consultor como um parâmetro na URL, desta forma caso se queira ver os clientes de outro consultor, basta informar o código deste outro consultor.

[43] A codificação pode ser lida no APÊNDICE D - Codificação P3_Clientes_do_Consultor_no_Mapa.pas
[44] http://firebirdsql.org/

Imagem 33 - Clientes do consultor plotados no mapa.



Fonte: Elaborado pelo próprio autor.

5.6 TESTE DE INVOCAÇÃO DE OUTRA LINGUAGEM

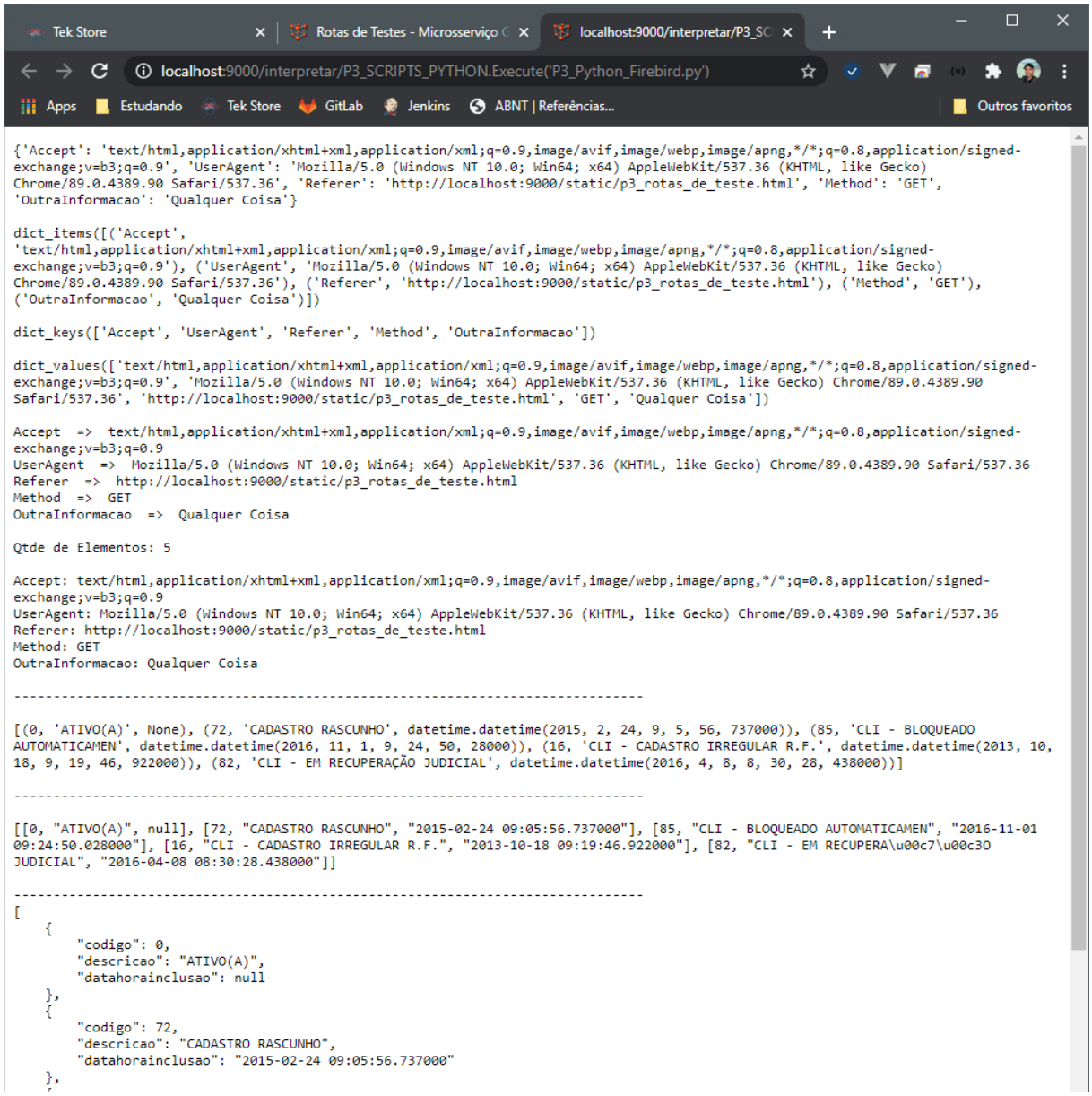
Através da chamada /interpretar/P3_SCRIPTS_PYTHON.Execute('P3_PYTHON_FIREBIRD.PY') que envolve duas unidades de código fonte, a primeira P3_SCRIPTS_PYTHON[45] em Pascal e a segunda P3_PYTHON_FIREBIRD[46] em Python, foi possível testar a invocação à linguagem Python. Neste exemplo, recebe os parâmetros da conexão e qualquer outra informação desejada, executa a impressão das informações recebidas e realiza um acesso ao banco de dados Firebird.

A forma de transmissão de informações entre as linguagens foi a mera substituição do texto %DADOS_REQUISICAO_WEB% no arquivo Python[47]. Outras abordagens, no entanto, poderiam terem sido utilizadas, como por exemplo, informar parâmetros via linha de comando ou gravar dados em um arquivo temporário. Esta forma foi apenas ilustrativa.

Na Imagem 34 é possível ver o retorno da requisição tendo utilizado o Python para gerar as informações.

[45] A codificação pode ser lida no APÊNDICE E - Codificação P3_Scripts_Python.pas
[46] A codificação pode ser lida no APÊNDICE F - Codificação P3_Python_Firebird.py
[47] https://www.python.org/

Imagem 34 - Execução de script Python a partir do GenericMicroServiceDPR.



Fonte: Elaborado pelo próprio autor.

Diversos outros testes foram executados, tais como integração com outros sistemas (microsserviços), consumo de API JSON, acesso ao MySQL[48] e ao SQLServer[49], envio de e-mail, upload / download de arquivos, *Cross Origin Resource Sharing* (CORS), *Discover* (*Redirect*), geração e consumo de *tokens* JWT, interceptação de requisições, servidor de arquivos públicos e geração de *Progressive Web App* (PWA). Testes esses, que podem ser verificados através do link

[48] https://www.mysql.com/
[49] https://www.microsoft.com/pt-br/sql-server/sql-server-downloads

http://localhost:9000/static/p3_rotas_de_teste.html com o GenericMicroServiceDPR em execução em uma máquina local ouvindo a porta 9000 e com acesso ao repositório TekStore (Imagem 35). Alguns destes códigos-fontes podem ser encontrados nos apêndices deste trabalho.

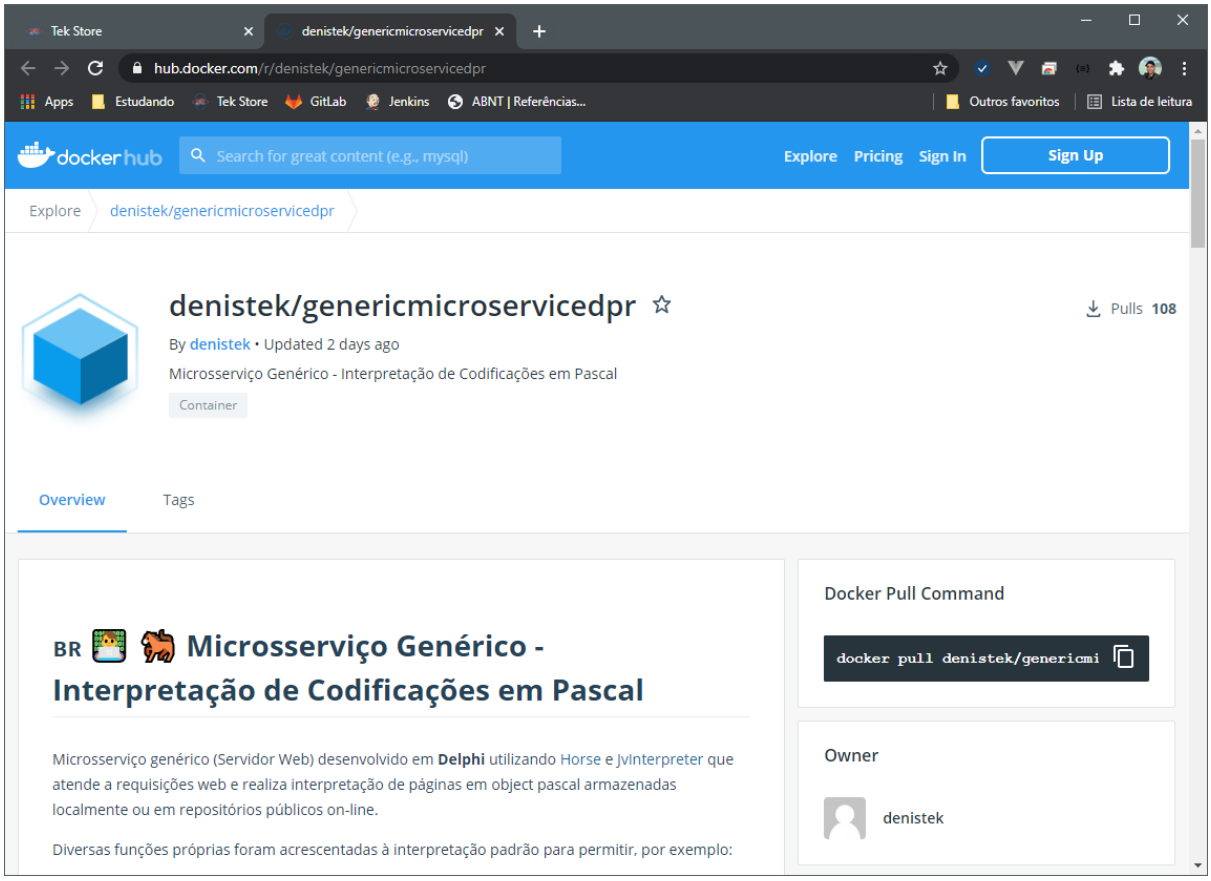
Imagem 35 - Testes unitários realizados no GenericMicroServiceDPR.

Fonte: Elaborado pelo próprio autor.

5.7 EMPACOTAMENTO EM IMAGEM DOCKER

Encerrado o desenvolvimento e os testes do GenericMicroServiceDPR, foi gerada uma imagem Docker para verificar a possibilidade de atender a demandas empresariais. Esta imagem Docker foi publicada e pode ser acessada em https://hub.docker.com/r/denistek/GenericMicroServiceDPR, conforme Imagem 36. Foge ao escopo deste trabalho detalhar a criação desta imagem Docker.

Imagem 36 - Imagem com o GenericMicroServiceDPR publicada no DockerHub.



Fonte: Elaborado pelo próprio autor.

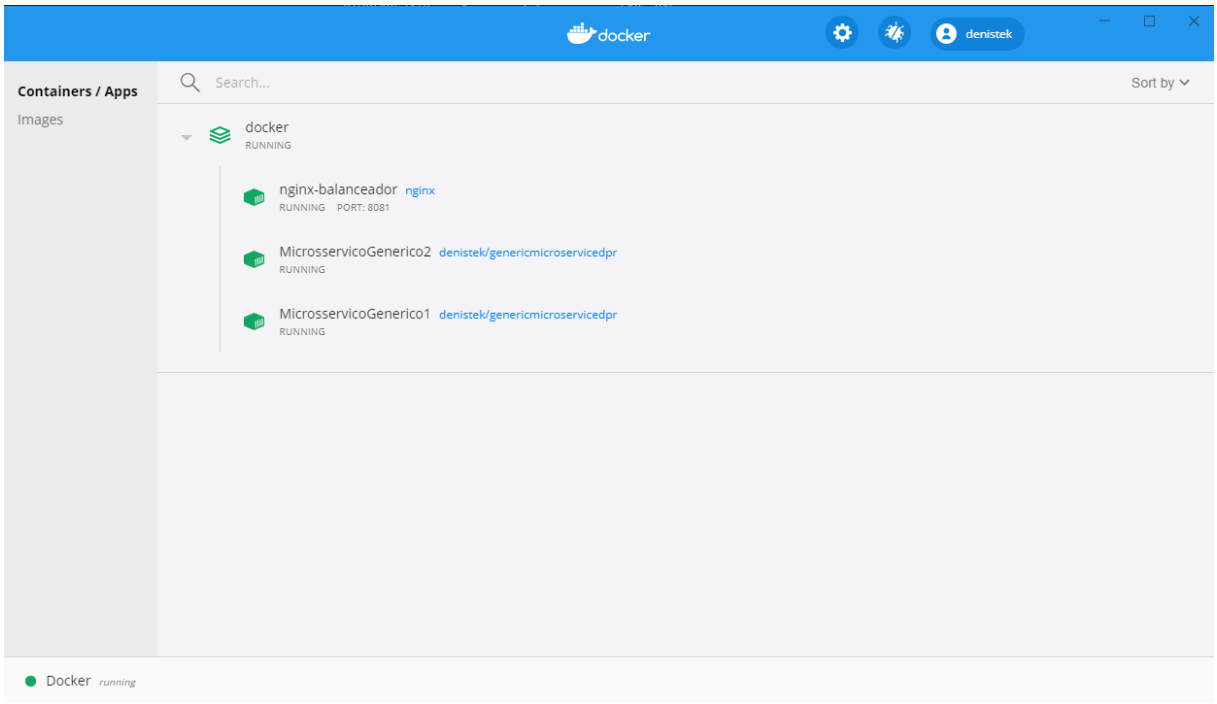
Caso queira-se realizar testes com este ambiente, deve-se instalar o Docker Desktop[50] em uma máquina com Windows 10 e executar o comando “docker pull denistek/genericmicroservicedpr” para realizar o download da imagem. Logo em seguida deve-se seguir os passos descritos na documentação da imagem para a execução do ambiente.

[50] https://www.docker.com/products/docker-desktop

5.8 TESTES COM BALANCEAMENTO DE CARGA EM CONTÊINER

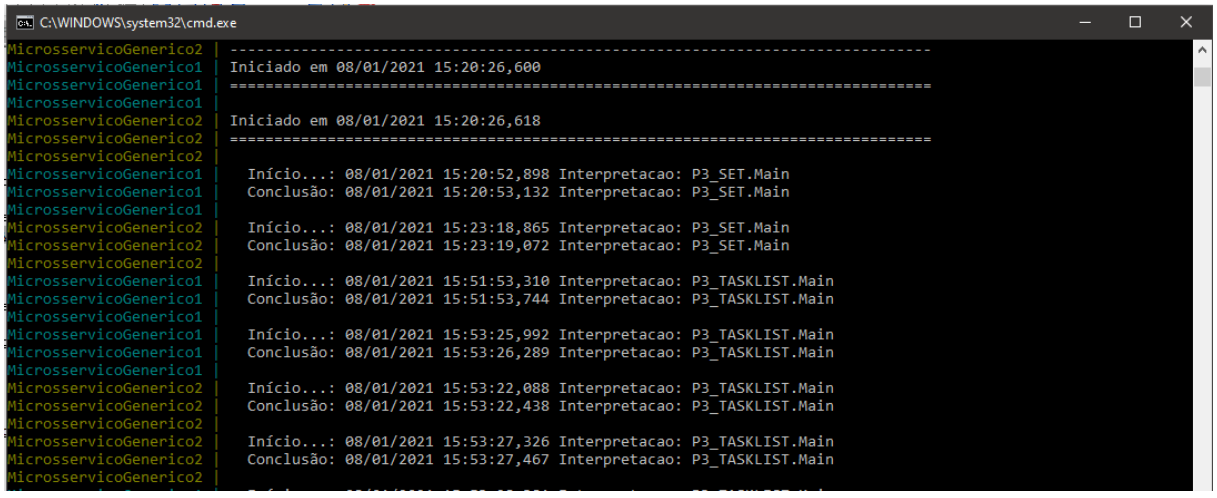
Na Imagem 37 é possível visualizar um ambiente com NGinX como balanceador de duas instâncias idênticas do GenericMicroServiceDPR em execução dentro de contêineres Docker. Na Imagem 38 é apresentado o registro das duas instâncias alternando a cada chamada.

Imagem 37 - Ambiente com balanceamento de carga usando NGinX.



Fonte: Elaborado pelo próprio autor.

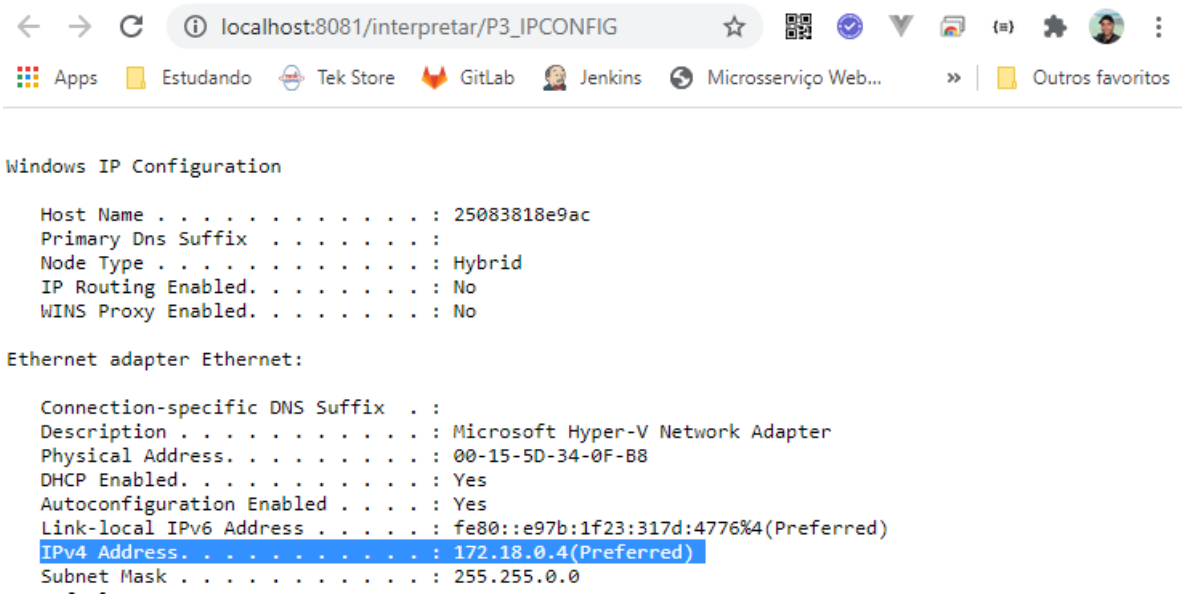
Imagem 38 - Log de instâncias alternando no balanceamento de carga.



Fonte: Elaborado pelo próprio autor.

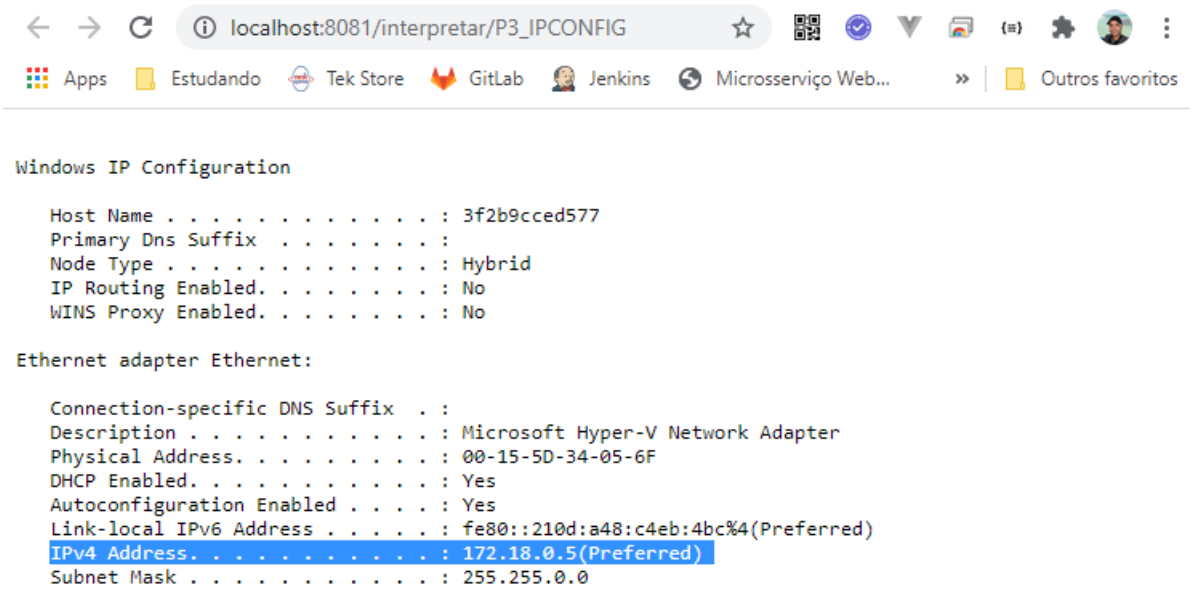
Ao solicitar uma *unit* que executa um comando IPConfig dentro da instância, pôde-se observar a alternância entre os endereços IP devolvidos por chamadas sequenciais (Imagem 39 e Imagem 40). A porta da execução foi a 8081 do NGinX e não a 9000 padrão da imagem.

Imagem 39 - IPCONFIG instância 1.



Fonte: Elaborado pelo próprio autor.

Imagem 40 - IPCONFIG instância 2.



Fonte: Elaborado pelo próprio autor.

6 CONCLUSÃO

Conforme foi apresentado, o GenericMicroServiceDPR cumpre bem o papel a que se propõe, de ser uma alternativa para a padronização de ecossistemas de microsserviços, sendo uma plataforma que permite a flexibilidade trazida pelos microsserviços ao permitir dividir sistemas monolíticos e utilizar outras linguagens de programação além da linguagem Pascal nativa. Se mostrou, também, como uma alternativa de servidor web em Pascal com velocidades compatíveis com os melhores servidores web atuais, chegando a superar o NodeJS e o Java.

Foi introduzido o conceito de múltiplos repositórios de códigos-fontes, locais ou remotos, bem como as facilidades decorrentes desta abordagem ainda não mencionada na literatura nem utilizada por servidores web atuais.

Por ser uma aplicação *stand-alone* pode ser transportada e executada até mesmo em um pen-drive permitindo que rapidamente um novo servidor seja colocado em execução em ambiente com poucos recursos. Por ser possível realizar a sua conteinerização também se adequa facilmente em ambientes que demandem alto desempenho, balanceamento de carga e tolerância a falhas.

Como trabalhos futuros, pode-se indicar as seguintes implementações: possibilidade de se configurar o tamanho máximo para upload de arquivos, bem como a frequência máxima permitida; possibilidade de uso de cache; pré-carregamento de algumas *units* mais utilizadas para a memória visando ganhar ainda mais desempenho do que o carregamento de arquivos do disco; controle de sessão; funções de *callback*; possibilidade de acessar outros repositórios privados que requeiram autenticação; possibilidade de gerar arquivos PDF com informações vindas de diversos bancos de dados; possibilidade de o GenericMicroServiceDPR ser instalado como um serviço do Windows.

Antes de se colocar em produção, é de suma importância configurar a segurança do tráfego de informações através do uso de certificado SSL confiável, provendo, desta forma, o acesso pelo protocolo HTTPS.

Espera-se que este trabalho venha contribuir positivamente para a Ciência da Computação no âmbito do Desenvolvimento Web e Mobile.

7 REFERÊNCIAS BIBLIOGRÁFICAS

AMAZON Web Services. **Microsserviços**: Crie microsserviços altamente disponíveis como base para aplicativos de qualquer porte e escala. [S.l.], 2020. Disponível em: <https://aws.amazon.com/pt/microservices/>. Acesso em: 7 dez. 2020.

CHAVES, Samir Braga. **Servicer:** Uma Plataforma para o Desenvolvimento Rápido de Microsserviços Multilinguagem**.** Orientador: Prof. Dr. José Antônio Macêdo. 2021. 58 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Universidade Federal do Ceará, Fortaleza, 2021.

FARIAS, Gilberto. **Introdução à Computação**. [S.l.], 2020. Disponível em: <http://producao.virtual.ufpb.br/books/camyle/introducao-a-computacao-livro/livro/livro.chunked/index.html> Acesso em: 13 dez. 2020.

FOWLER, Susan J. **Microsserviços prontos para a produção**: construindo sistemas padronizados em uma organização de engenharia de software. 1. ed. São Paulo: Novatec Editora Ltda., 2017. 224 p. v. 1. ISBN 978-85-7522-621-6.

Google. **Interpretador**. 2020. Disponível em: <https://www.google.com/search?q=interpretador> Acesso em: 13 dez. 2020.

LIMA, Marina Medeiros. **Análise do processo de migração de sistemas de arquitetura monolítica para microsserviço**. Orientador: José Carlos Cavalcanti. 2019. 55 p. Trabalho de conclusão de curso (Graduação em Engenharia da Computação) - Universidade Federal de Pernambuco, Recife, 2019.

OPUS SOFTWARE. **Microsserviços framework**: saiba como essa prática pode agilizar o processo de desenvolvimento de software. [*S. l.*]: OPUS Software, 14 abr. 2021. Disponível em: <https://www.opus-software.com.br/microsservicos-um-framework-reutilizavel/>. Acesso em: 21 abr. 2021.

PASCAL Server Pages – Pascal Script. **Code Project® for those who code**. Toronto, 17 dez. 2012. Disponível em

<https://www.codeproject.com/Articles/509625/PascalplusServerplusPagesplus-e-plusPascalpl>. Acesso em: 16 fev. 2021.

RANGA, V.; SONI, A. Api features individualizing of web services: Rest and soap. **International Journal of Innovative Technology and Exploring Engineering**, v. 8, 08 2019.

RICHARDSON, Chris. **Microservices patterns: with examples in Java.** [S. l.]: Manning Publications, 2019.

RICHARDSON, Chris. **Microservice Architecture.** [S.l.]: 2020. Disponível em: <https://microservices.io/>. Acesso em: 7 dez. 2020.

SHOUP, Randy. (2014). **Evolutionary Architecture**: Good Enough is Good Enough. Disponível em: <https://randyshoup.silvrback.com/evolutionary-architecture>. Acesso em: 7 dez. 2020.

SOS Digital**. Saiba quais são as diferenças entre compilador e interpretador**. [S.l.], 2020. Disponível em: <https://www.sosdigital.com.br/saiba-quais-sao-as-diferencas-entre-compilador-e-interpretador/>. Acesso em: 13 dez. 2020.

WERTHER FILHO, João. PPW: **a Web Technology of dynamic pages based on Pascal**. 2006. 121 f. Dissertação (Mestrado em Sistemas e Computação) - Universidade Salvador, Salvador, 2006. The 32nd Latin American Conference on Informatics, 2006, Santiago, Chile. 2006.

WIKIPÉDIA. **Docker**. 2021. Disponível em: <https://pt.wikipedia.org/wiki/Docker_(software)>. Acesso em: 21 abr. 2021.

WIKIPÉDIA. **Interpretador**. 2020. Disponível em: <https://pt.wikipedia.org/wiki/Interpretador>. Acesso em: 13 dez. 2020.

8 GLOSSÁRIO

API	Conjunto de rotinas, protocolos e ferramentas para construir aplicações.
Backend	Parte de um sistema computacional ou aplicação que não é diretamente acessado pelo usuário, tipicamente responsável por armazenar e manipular dados.
Bytecode	Resultado de um processo semelhante ao dos compiladores de código-fonte que não é imediatamente executável, será interpretado em uma máquina virtual que fará a execução.
DELETE	Verbo HTTP utilizado para requisições de exclusão de dados.
Deploy	Disponibilizar um sistema para uso, seja num ambiente de desenvolvimento, testes ou em produção.
DevOps	Contração de *development* e *operations.* Cultura na engenharia de software que aproxima os desenvolvedores de software (dev) e os operadores do software / administradores do sistema (ops).
Endpoint	URL onde o serviço pode ser acessado por uma aplicação cliente.
Frontend	Frontend de um website é tudo com o qual o usuário interage. Sinônimo de interface de usuário.
GET	Verbo HTTP utilizado para requisições de consulta de dados.
Hash	Algoritmo ou função que converte um valor em outro, normalmente usando um mapeamento de dados.
JWT	Método RCT 7519 padrão da indústria para realizar autenticação entre duas partes, em requisições web, por meio de um *token* assinado. Esse *token* é um código em Base64 que armazena objetos JSON com dados que permitem essa autenticação da requisição.
Lighthouse	Ferramenta automatizada de código aberto da Google que pode ser executada em qualquer página web para medir a sua qualidade. Audita o desempenho, acessibilidade e otimização de mecanismo de pesquisa.

Mobile	Toda tecnologia que permite a uma pessoa se movimentar sem abdicar dela.
Middleware	Componente de software que atua com um intermediário entre a requisição e a resposta para facilitar, controlar, registrar a comunicação.
NodeJS	Ambiente de execução JavaScript no lado servidor.
Pascal	Linguagem de programação imperativa e procedural, desenvolvida por Niklaus Wirth como uma pequena e eficiente linguagem com a intenção de encorajar boas práticas de programação usando programação e dados estruturados. Seu nome honra o matemático, filósofo e físico francês Blaise Pascal.
POST	Verbo HTTP utilizado para alterações de dados.
PUT	Verbo HTTP utilizado para inclusão de dados.
Python	Linguagem de programação de alto nível, interpretada de script, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte.
REST	Transferência Representacional de Estado, é um estilo de arquitetura de software que define um conjunto de restrições a serem usadas para a criação de *web services*.
Scriptlet	Pedaço de código em outra linguagem embutido em código HTML. O *scriptlet* é todo o conteúdo que está dentro das *tags* <% %>. Entre elas o usuário pode adicionar qualquer *scriptlet* válido, isto é, qualquer código nesta outra linguagem.
Unit(s)	Unidade de codificação, arquivo de texto-puro que contém códigos-fontes em alguma linguagem de programação.
Web Scraping	Coleta ou extração de dados web, também conhecido como raspagem web, é uma forma de mineração que permite a extração de dados de sites da web convertendo-os em informação estruturada para posterior análise.

APÊNDICE A - Codificação P3_Crud.pas

```
// Codificação Pascal - serve como base para montagem de API (backend) CRUD.

unit P3_CRUD;

const Conexao = ''; // Usa configuração padrão de conexão ao banco de dados.

function Main: String;
var id: Integer;
begin
  id := StrToIntDef(RequestParamId, -1); // Evitar SQL injection.
  case RequestMethodType of
    mtPut:    Result := Incluir;
    mtPost:   Result := Alterar(id); // Alterar todo o registro.
    mtDelete: Result := Excluir(id);
    mtGet:    Result := Listar(id);
    mtHead:   Result := '{}';     // Apenas retornar os headeres igual ao GET
    mtPatch:  Result := Alterar(id); // Alterar parte, mesma instrução do POST
  end;
end;

function CriarTabela: Integer;
begin
  Result := ExecuteCommandFD(Conexao, SQLCreateTable);
end;

function CriarGenerator: Integer;
begin
  Result := ExecuteCommandFD(Conexao, SQLCreateGenerator);
end;

function ConfigGenerator(NovoValor: Integer): Integer;
begin
  Result := ExecuteCommandFD(Conexao, Format(SQLSetGenerator, [NovoValor]));
end;
```

```
function Excluir(id: Integer): Integer; // retorna qtde de registros excluídos
begin
  Result := ExecuteCommandFD(Conexao, SQLDelete, [id]);
  ResponseStatusCode := 204; // No Content;
end;

function Listar(id: Integer): string;
var CDS: TClientDataSet; SQL, Ordenacao, InicioReg, FimReg: String;
begin
  CDS := TClientDataSet.Create;
  try
    SQL := SQLSelectAll;

    if (id <> -1) then // Registro único
      SQL := SQL + Format(' where %s = %d', [CampoChave, id]);

    // Ordenação
    if (RequestQuery.IndexOfName('orderby') >= 0) then
      begin
        Ordenacao := RequestQuery.Values['orderby'];
        SQL := SQL + ' order by ' + Ordenacao;
      end;

    // Paginação
    if (RequestQuery.IndexOfName('rows') >= 0) then
      begin
        InicioReg := RequestQuery.Values['rows'];
        SQL := SQL + ' rows ' + InicioReg;

        if (RequestQuery.IndexOfName('to') >= 0) then
          begin
            FimReg := RequestQuery.Values['to'];
            SQL := SQL + ' to ' + FimReg;
          end;
      end;

    WriteLn('SQL: ' + SQL); // Escreve a SQL no Console
    CDS.Data := ExecuteReaderFD(Conexao, SQL); // Carrega registros do banco.
```

```
    if (id <> -1) and (CDS.IsEmpty) then
      raise EHorseException.Create(404, Format('Registro %d não existe.',
[id]));

    if RequestContentType = 'text/csv' then Result := CDS.ToCSV('"', ',') else
    if RequestContentType = 'application/xml' then Result := CDS.XMLData  else
    {if RequestContentType = 'application/json' then}
      begin
        Result := CDS.ToJsonSimples;
        if (id <> -1) then
          Result := GetObjectJson(Result, 'Dados[0]'); // Remove elemento do
vetor
      end;
    ResponseContentType := RequestContentType; // Devolve no tipo solicitado
  finally
    CDS.Free;
  end;
end;

end.
```

APÊNDICE B - Codificação P3_Crud_Status.pas

```
// Codificação em Pascal para fornecer um CRUD para uma tabela STATUS.
// Faz referência à codificação P3_CRUD.

unit P3_CRUD_STATUS;

uses P3_CRUD;

const
  CampoChave         = 'CODIGO_STATUS';
  SQLCreateTable     = 'CREATE TABLE STATUS (CODIGO_STATUS integer default 0
not null, DESCRICAO_STATUS varchar(30) not null)';
  SQLCreateGenerator = 'CREATE SEQUENCE GEN_STATUS';
  SQLSetGenerator    = 'ALTER SEQUENCE GEN_STATUS RESTART WITH %d INCREMENT BY
1';
  SQLInsert          = 'insert into STATUS (CODIGO_STATUS, DESCRICAO_STATUS)
values (GEN_ID(GEN_STATUS, 1), :descricao) returning CODIGO_STATUS';
  SQLUpdate          = 'update STATUS set DESCRICAO_STATUS = :descricao where
CODIGO_STATUS = :id';
  SQLDelete          = 'delete from STATUS where CODIGO_STATUS = :id';
  SQLSelectAll       = 'select CODIGO_STATUS, DESCRICAO_STATUS from STATUS';

var descricao: String;

function Main: String;
begin
  Result := P3_CRUD.Main;
end;

function Incluir: String;
begin
  descricao := Copy(GetValueJson(RequestBody, 'descricao'), 1, 30);
  Result := ExecuteScalarFD(Conexao, SQLInsert, [descricao]); // retorna id
criado
  //Result := Listar(StrToInt(Result)); // poderia retornar JSON c/dados
incluídos
```

```
  ResponseStatusCode := 201; // Created;
end;

function Alterar(id: Integer): Integer; // retorna a qtde de registros
modificados
begin
  descricao := Copy(GetValueJson(RequestBody, 'descricao'), 1, 30);
  Result    := ExecuteCommandFD(Conexao, SQLUpdate, [descricao, id]);
end;

end.
```

APÊNDICE C - Codificação P3_Cadastro_Status.html (*Frontend* do cadastro)

```
<!-- Foram colocadas as codificações em HTML, JavaScript e CSS em um mesmo
arquivo por motivos didáticos, mas deveriam estar separados em três arquivos
distintos fazendo-se os respectivos links de referência conforme recomendam as
melhores práticas de desenvolvimento web. --!>

<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="utf-8">
    <meta name="Description"  content="Cadastro de Status">
    <meta name="viewport"     content="width=device-width, initial-scale=1.0">
    <meta http-equiv="Content-Type"     content="text/html; charset=utf-8"/>
    <meta http-equiv="content-language" content="pt-br"/>
    <meta http-equiv="X-UA-Compatible"  content="IE=edge">
    <title>Cadastro de Status</title>
  </head>
  <body>
    <h1>Cadastro de Status</h1>
    <p id="ferramenta">
    <input type="button" class="btn" value="Incluir Novo" onClick="Incluir()">
    </p>
    <p id="info_erro"></p>
    <table id="tabela"></table>
  </body>
</html>

<script>
  let link_crud  = window.location.protocol + '//' +
                   window.location.hostname +
                   ':8000/interpretar/P3_CRUD_STATUS';
  let $          = document.querySelector.bind(document);
  let info_erro  = $("#info_erro");
  let tabela     = $("#tabela");
  let ordemAtual = 1;
  let asc_desc   = 'asc';
```

```
let Registros  = [];
let selectEtag = '';

CarregarRegistros();

function CarregarRegistros(ordem) {
  if (ordem) {
    asc_desc = (ordemAtual == ordem ?
                (asc_desc == 'asc' ? 'desc' : 'asc') : 'asc');
    ordemAtual = ordem;
  }
  info_erro.innerHTML = '';
  let xhr = new XMLHttpRequest();
  xhr.open('GET', link_crud + '?orderby=' + ordemAtual + ' ' + asc_desc);
  xhr.setRequestHeader('If-None-Match', selectEtag);
  xhr.onreadystatechange = function() {
    if (xhr.readyState === XMLHttpRequest.DONE) {
      if (xhr.status == 304) {
        ExibirRegistros();
      } else if (xhr.status == 200) {
        selectEtag = xhr.getResponseHeader('ETag');
        let resposta = xhr.responseText;
        //console.log(resposta);
        Registros = JSON.parse(resposta).Dados;
        ExibirRegistros();
      } else {
        console.log(xhr.responseText);
        info_erro.innerHTML = 'Erro ao carregar os registros.';
      }
    }
  };
  xhr.onerror = TrataErroRequisicaoAPI;
  xhr.send();
};

function ExibirRegistros() {
  let shtml = '<thead><tr class="titulo_colunas">' +
              '<th></th><th></th>' +
```

```
                '<th><a onClick="CarregarRegistros(1);">Código</a></th>' +
                '<th><a onClick="CarregarRegistros(2);">Descrição</a></th>' +
                '</tr></thead>';
    Registros.forEach((Registro) => {
      shtml +=
        `<tr class="linha">
          <td class="coluna"><input type="button"
onClick="Alterar(${Registro.CODIGO_STATUS})" value="Alterar"
class="btn"/></td>
          <td class="coluna"><input type="button"
onClick="Excluir(${Registro.CODIGO_STATUS})" value="Excluir"
class="btn"/></td>
          <td class="coluna">${Registro.CODIGO_STATUS}</td>
          <td class="coluna">${Registro.DESCRICAO_STATUS}</td>
        </tr>`
    })
    tabela.innerHTML = shtml;
  }

  function Incluir() {
    let NovaDescricao = window.prompt('Informe a nova descrição:');
    if (NovaDescricao) {
      info_erro.innerHTML = '';
      let dados = {"descricao": NovaDescricao};
      let dadosJSON = JSON.stringify(dados);

      let xhr = new XMLHttpRequest();
      xhr.open('PUT', link_crud, true);
      xhr.setRequestHeader('Content-type', 'application/json; charset=utf-8');
      xhr.onreadystatechange = function() {
        if (xhr.readyState === XMLHttpRequest.DONE) {
          if (xhr.status == 201) {
           //alert('Registro ' + xhr.responseText + ' incluído com sucesso.');
            CarregarRegistros();
          } else {
            console.log(xhr.responseText);
            info_erro.innerHTML = "Erro ao tentar incluir o registro.";
          }
```

```
        }
      }
      xhr.onerror = TrataErroRequisicaoAPI;
      xhr.send(dadosJSON);
    }
  }

  function Alterar(Codigo) {
    let DescricaoAtual = Registros.find(element => element.CODIGO_STATUS ==
Codigo).DESCRICAO_STATUS;
    let NovaDescricao = window.prompt(
      'Informe a nova descrição para o código ' + Codigo, DescricaoAtual);
    if (NovaDescricao) {
      info_erro.innerHTML = '';
      let dados = {"descricao": NovaDescricao};
      let dadosJSON = JSON.stringify(dados);

      let xhr = new XMLHttpRequest();
      xhr.open('POST', link_crud + '/' + Codigo, true);
      xhr.setRequestHeader('Content-type', 'application/json; charset=utf-8');
      xhr.onreadystatechange = function() {
        if (xhr.readyState === XMLHttpRequest.DONE) {
          if (xhr.status == 200) {
            //alert('Registro ' + Codigo + ' alterado com sucesso.');
            CarregarRegistros();
          } else {
            console.log(xhr.responseText);
            info_erro.innerHTML = "Erro ao tentar atualizar o registro.";
          }
        }
      }
      xhr.onerror = TrataErroRequisicaoAPI;
      xhr.send(dadosJSON);
    }
  }

  function Excluir(Codigo) {
```

```
    if (window.confirm('Você realmente quer excluir o registro ' + Codigo +
'?')) {
      info_erro.innerHTML = '';
      let xhr = new XMLHttpRequest();
      xhr.open('DELETE', link_crud + '/' + Codigo, true);
      xhr.onload = function() {
        if (xhr.readyState === XMLHttpRequest.DONE) {
          if (xhr.status == 204) {
            //alert('Registro ' + Codigo + ' excluído com sucesso.');
            CarregarRegistros();
          } else {
            console.log(xhr.responseText);
            info_erro.innerHTML = "Erro ao tentar excluir o registro.";
          }
        }
      }
      xhr.onerror = TrataErroRequisicaoAPI;
      xhr.send(null);
    }
  }

  function TrataErroRequisicaoAPI(){
    info_erro.innerHTML = "Erro ao fazer requisição para a API. Certifique-se
de que ela esteja em execução.";
  }
</script>

<style>
  html, body {margin: 0; text-align: center; }
  body {background-color: rgb(245 244 244);}
  h1 {
    background-color: rgb(133 25 68);
    color: white;
    line-height: 2em;
    text-shadow: 3px 3px #000;
    vertical-align: middle;
    position: fixed;
    margin: 0;
```

```
  top: 0;
  right: 0;
  left: 0;
  z-index: 1030;
}
#ferramenta {
  background-color: rgb(245 244 244);
  position: fixed;
  top: 3.0em;
  right: 0;
  left: 0;
  z-index: 1000;
  line-height: 2.5em;
}
#info_erro {
  color: red;
  background-color: #f443361a;
}
#tabela {
  border-style: dotted;
  display: table;
  font-family: Trebuchet MS;
  font-size: 0.8em;
  margin: 0 auto;
  margin-top: 8.5em;
  margin-bottom: 2em;
  border-spacing: 1px;
  border-width: 1px;
}
.titulo_colunas {
  display: table-row;
  height: 35px;
  width: 100%;
  background-color: rgb(217, 217, 253);
  font-weight: bold;
}
.linha {
  display: table-row;
```

```
    width: 100%;
    height: 35px;
    background: rgb(220, 220, 220);
  }
  .linha:nth-child(2n+1) { background: rgba(240, 240, 240, 0.48); }
  .linha:hover { background-color: rgb(207, 247, 209); }
  .coluna {
    display: table-cell;
    vertical-align: middle;
    padding: 1ch;
    text-align: left;
  }
  .btn { height: 1.8rem; }
</style>
```

APÊNDICE D - Codificação P3_Clientes_do_Consultor_no_Mapa.pas

```
Unit P3_CLIENTES_DO_CONSULTOR_NO_MAPA;


uses Tek_Maps_Google; // Uso de biblioteca proprietária da Tek-System


function Main: String;
var
  Consultor: Integer;
  Pessoas: string;
  CDS: TClientDataSet;
begin
  Consultor := StrToIntDef(RequestParamId, -1);
  Pessoas   := ' ["Lat", "Long", "Informação"],' + #13;

  CDS := TClientDataSet.Create;
  try
    CDS.Data := ListaDeClientes(Consultor);
    CDS.First;
    while (not CDS.Eof) do
    begin
      Pessoas := Pessoas + ' [' +
        Troca(CDS.FieldByName('Latitude').AsString,  ',', '.') + ', ' +
        Troca(CDS.FieldByName('Longitude').AsString, ',', '.') + ', ' +
        QuotedStr(
          'Cliente: '    + CDS.FieldByName('Código').AsString + '-' +
                           CDS.FieldByName('Razão Social').AsString +
          '\nCidade: '   + CDS.FieldByName('Cidade').AsString + '-' +
                           CDS.FieldByName('Estado').AsString +
          '\nEndereço: ' + CDS.FieldByName('Endereço').AsString +
          '\nBairro: '   + CDS.FieldByName('Bairro').AsString +
          '\nCep: '      + CDS.FieldByName('Cep').AsString +
          '\nTelefone: ' + CDS.FieldByName('Telefone').AsString +
          '\nE-mail: '   + CDS.FieldByName('E-mail').AsString) + ']';

      if (CDS.Recno <> CDS.RecordCount) and
         (CDS.Recno <> cMaxPlotMapsGoogle) then
        Pessoas := Pessoas + ',' + #13;

      if (CDS.Recno = cMaxPlotMapsGoogle) then
        Break;

      CDS.Next;
    end;
  finally
```

```
    CDS.Free;
  end;

Tek_Maps_Google.SetarCredencialGoogle(GetEnvironmentVariable('CHAVE_API_GOOGLE
_MAPS'));
  Result := Tek_Maps_Google.MontarPaginaMapsGoogle(Pessoas);
  ResponseContentType := 'text/html; charset=UTF-8';
end;

function ListaDeClientes(Consultor: Integer): OleVariant;
var SQL: string;
begin
  SQL :=
  'select' + #13 +
  '  PESSOA.CODIGO_PESSOA                              "Código",'        + #13 +
  '  PESSOA.RAZAOSOCIAL_PESSOA                         "Razão Social",'  + #13 +
  '  PE.ENDERECO_PESSOA_END || '','' || PE.NUMERO_PESSOA_END "Endereço",'#13 +
  '  PE.BAIRRO_PESSOA_END                              "Bairro",'        + #13 +
  '  C.DESCRICAO_CIDADE                                "Cidade",'        + #13 +
  '  UF.SIGLA_UF                                       "Estado",'        + #13 +
  '  PE.CEP_PESSOA_END                                 "CEP",'           + #13 +
  '  PT.TELEFONE_PESSOA_TEL                            "Telefone",'      + #13 +
  '  PM.EMAIL_PESSOA_EMAIL                             "E-mail",'        + #13 +
  '  PE.LATITUDE_PESSOA_END                            "Latitude",'      + #13 +
  '  PE.LONGITUDE_PESSOA_END                           "Longitude",'     + #13 +
  '  PE.STATUSGEOREF_PESSOA_END                        "Status Geo-Ref"' + #13 +
  'from PESSOA_CLIENTECONSULTOR PC' + #13 +
  'inner join PESSOA on (PESSOA.CODIGO_PESSOA=PC.CLIENTE_PESSOA_CLICON)'#13 +
  'inner join PESSOA_ENDERECO PE on (PE.AUTOINC_PESSOA_END =
PESSOA.ENDERECO_PESSOA)' + #13 +
  'inner join CIDADE C on (C.CODIGO_CIDADE=PE.CIDADE_PESSOA_END)'+ #13 +
  'inner join UF       on (UF.CODIGO_UF= C.UF_CIDADE)'           + #13 +
  'left join PESSOA_TELEFONE PT on (PT.AUTOINC_PESSOA_TEL =
PE.TELEFONEPADRAO_PESSOA_END)' + #13 +
  'left join PESSOA_EMAIL    PM on (PM.AUTOINC_PESSOA_EMAIL =
PESSOA.EMAILPADRAO_PESSOA)'    + #13 +
  'where PC.CONSULTOR_PESSOA_CLICON = ' + IntToStr(Consultor) + #13 +
  '  and PESSOA.CLIENTE_PESSOA      = ''S''' + #13 +
  '  and PE.STATUSGEOREF_PESSOA_END = ''OK''';

  Result := ExecuteReaderFD('', SQL);
end;

end.
```

APÊNDICE E - Codificação P3_Scripts_Python.pas

```
// Invoca o Python para executar quaisquer outras unidades de codificação.
// Transmite os dados da requisição por substituição de texto.

Unit P3_SCRIPTS_PYTHON;

var DirTarget: String;

function Execute(ArquivoScript: String): String;
begin
  DirTarget := GetEnvironmentVariable('TMP');
  Result    := ExecutarScriptPython(ArquivoScript);
  ResponseContentType := 'text/plain';
end;

function ExecutarScriptPython(NomeUnit: String): string;
var NomeArquivoMontado, Comando: string;
begin
  NomeArquivoMontado := CodificacaoUnitParaArquivo(NomeUnit, DirTarget,
    '%DADOS_REQUISICAO_WEB%',
    Format('{"Accept": "%s", "UserAgent": "%s", "Referer": "%s", "Method":
"%s", "OutraInformacao": "%s"}',
    [WebRequest.Accept, WebRequest.UserAgent, WebRequest.Referer,
WebRequest.Method, 'Qualquer Coisa']));

  Comando := Format('python "%s"', [NomeArquivoMontado]);

  Result := ExecutarComandoSOCapturandoOutput(Comando, '', 0);
end;

end.
```

APÊNDICE F - Codificação P3_Python_Firebird.py

```
# Recebe dados da requisição, acessa Firebird e devolve informações.
import os
import fdb
import json
import datetime

# Imprimindo dados da requisição
dadosRequisicaoWeb = %DADOS_REQUISICAO_WEB% # Enviados da chamada em Pascal
print(dadosRequisicaoWeb,          '\n')
print(dadosRequisicaoWeb.items(),  '\n')
print(dadosRequisicaoWeb.keys(),   '\n')
print(dadosRequisicaoWeb.values(), '\n')
for k, v in dadosRequisicaoWeb.items():
  print(k, " => ", v)
print('\nQtde de Elementos:', len(dadosRequisicaoWeb), '\n')
print('Accept:',          dadosRequisicaoWeb['Accept'])
print('UserAgent:',       dadosRequisicaoWeb['UserAgent'])
print('Referer:',         dadosRequisicaoWeb['Referer'])
print('Method:',          dadosRequisicaoWeb['Method'])
print('OutraInformacao:', dadosRequisicaoWeb['OutraInformacao'])
print('');

senha = os.environ["BDD_PASSWORD"]
meuip = os.environ["MEUIP"]

conexao = fdb.connect(
    host     = meuip,
    port     = 3050,
    database = 'Erp4g',
    user     = 'SYSDBA',
    password = senha,
    charset  = 'ISO8859_1')

SQL = \
```

```
  " select first 5 S.CODIGO_STATUS, S.DESCRICAO_STATUS,
S.DATAHORAINCLUSAO_STATUS" \
  " from STATUS S order by S.DESCRICAO_STATUS"

cursor = conexao.cursor()

cursor.execute(SQL)

registros = cursor.fetchall()

# https://code-maven.com/serialize-datetime-object-as-json-in-python
def myconverter(o):
  if isinstance(o, datetime.datetime):
    return o.__str__()
  if isinstance(o, datetime.date):
    #return "{}-{}-{}".format(o.day, o.month, o.year)
    return o.isoformat()

# Exportação dos dados - Nativa Python
print('-' * 80, '\n')
print(registros, '\n')

# Exportação dos dados - Usando um convertedor dos campos de data/hora
print('-' * 80, '\n')
print(json.dumps(registros, default=myconverter), '\n')

# Exportação dos dados - Capturando informações separadamente e formatando
print('-' * 80)
dados = []
for (codigo, descricao, datahorainclusao) in registros:
  registro = {
    "codigo": codigo,
    "descricao": descricao,
    "datahorainclusao": datahorainclusao
  }
  dados.append(registro)
#print(dados)
#exit(dados)
```

```
print(json.dumps(dados, default=myconverter, indent=4, separators=(",", ": "),
sort_keys=False), '\n')

cursor.close() # Fechamento da conexão
```

APÊNDICE G - Exemplo de Codificação para Envio de E-mail

```
function Main: String;
var EE: TEnviarEmail;
begin
  EE := TEnviarEmail.Create;
  try
    EE.PrepararParaEnviarOutro(False, False, False);
{ // informações padrões, definidas na configuração não precisam ser passadas
    EE.ServidorEmail_Host            := 'smtp.gmail.com';
    EE.ServidorEmail_Port            := 587;
    EE.ServidorEmail_Username        := 'genericmicroservicedpr@gmail.com';
    EE.ServidorEmail_Password        := GetEnvironmentVariable('MAIL_PASSW');
    EE.ServidorEmail_EmailSeguro     := False;
    EE.UsaRemetenteEmailNoFrom       := True;
    EE.Remetente_Nome                := 'Microsserviço Genérico DPR';
    EE.Remetente_Email               := 'genericmicroservicedpr@gmail.com';
    EE.ResponderPara                 := 'genericmicroservicedpr@gmail.com';
    EE.TempoMaximoConexao            := 10 * 1000;
    EE.TempoMaximoLeitura            := 60 * 1000;
}
    EE.Destinatario_Email            := 'denisuba@gmail.com';
    EE.Destinatario_EmailCC          := '';
    EE.Destinatario_EmailCCO         := '';
    EE.Assunto                       := 'teste e-mail microsserviço';
    EE.MensagemHTML                  := '<h1>mensagem em HTML</h1>';
    EE.Mensagem                      := 'mensagem de texto';
    EE.SolicitarConfirmacaoDeLeitura := False;
    EE.ArquivosAnexados.Add('.\downloads\arquivo.txt');
    EE.Enviar;

    Result := Format('{"message": "Envio c/sucesso", "momento": "%s"}',[Now]);
  finally
    EE.Free;
  end;
end;
```

APÊNDICE H - Exemplo de Codificação para Autenticação JWT

```
const
  MinhaAssinaturaSecreta = 'AssinaturaSecreta-NinguemPodeSaber';
  MinutosExpiracaoToken  = 15;
  TimeZone               = -3;

function Main: string;
var
  TokenSimples, TokenComplexo, Info: String;
  Valido: Boolean;
  Vetor: TStringDynArray;
begin
  TokenSimples  := CriarTokenJWTSimples;
  TokenComplexo := CriarTokenJWTComplexo;
  Valido        := TokenValido(TokenSimples);

  Vetor         := SplitString(TokenSimples, '.');
  Info          := JsonFormatado(DecodeFromBase64(Vetor[0])) + #13 +
                   JsonFormatado(DecodeFromBase64(Vetor[1]));

  Result := 'TokenSimples: '  + #13 + TokenSimples  + #13#13 +
            'TokenComplexo: ' + #13 + TokenComplexo + #13#13 +
            iif(Valido, 'Token Válido', 'Token Inválido') + #13#13 +
            'Informação Decodificada: ' + #13 + Info;

  ResponseContentType := 'text/plain';
end;

function UsuarioSenhaValido: Boolean;
var Usuario, Senha: String;
begin
  if Trim(RequestBody) = '' then
    raise EHorseException.Create('Usuário ou senha não foram passados no corpo
da requisição');

  // Receptionar Usuário e Senha vindos no Body
  Usuario := GetValueJson(RequestBody, 'user');
  Senha   := GetValueJson(RequestBody, 'password');

  // Verificar se são autênticos, realizando acesso a banco de dados
  Result := True;
end;

function CriarTokenJWTSimples: string;
var LJWT: TJWT;
begin
  if not UsuarioSenhaValido then
```

```
    EHorseException.Create('Usuário ou senha inválidos');

  LJWT := TJWT.Create();
  try
    LJWT.Claims.JWTId      := 5; // Id do usuário
    LJWT.Claims.Audience   := 'App Destino';
    LJWT.Claims.IssuedAt   := Now;
    LJWT.Claims.Issuer     := 'GenericMicroServiceDPR';
    LJWT.Claims.Subject    := 'Nome do Usuário';
    LJWT.Claims.Expiration := IncMinute(Now, MinutosExpiracaoToken);
    LJWT.Claims.JSON.AddPair('Outra Informacao', 'Bla Bla Bla');
    Result := TJOSE.SHA256CompactToken(MinhaAssinaturaSecreta, LJWT);
  finally
    LJWT.Free;
  end;
end;

function CriarTokenJWTComplexo: string;
var
  LJWT: TJWT;
  LKey: TJWK;
  LSigner: TJWS;
const // TJOSEAlgorithmId:
  HS256 = 2;
  HS384 = 3;
  HS512 = 4;
begin
  if not UsuarioSenhaValido then
    EHorseException.Create('Usuário ou senha inválidos');

  LJWT := TJWT.Create();
  try
    LJWT.Claims.JWTId      := 5; // Id do Usuário
    LJWT.Claims.Audience   := 'Aplicação Destino';
    LJWT.Claims.IssuedAt   := Now;
    LJWT.Claims.Issuer     := 'GenericMicroServiceDPR';
    LJWT.Claims.Subject    := 'Nome do Candango';
    LJWT.Claims.Expiration := IncMinute(Now, MinutosExpiracaoToken);
    LJWT.Claims.JSON.AddPair('Outra Informacao', 'Bla Bla Bla');

    LKey := TJWK.Create(MinhaAssinaturaSecreta); // Cria sua chave
    try
      LSigner := TJWS.Create(LJWT); // Criar um assinador
      try
       // Com esta opção pode-se ter chaves menores que o tamanho do algoritmo
        LSigner.SkipKeyValidation := True;
        LSigner.Sign(LKey, HS256); // Assina o token!
        Result := ' Header: '        + LSigner.Header    + #13
                  ' Payload: '       + LSigner.Payload   + #13
```

```
                  ' Signature: '     + LSigner.Signature + #13
                  ' Compact Token: ' + LSigner.CompactToken;
      finally
        LSigner.Free;
      end;
    finally
      LKey.Free;
    end;
  finally
    LJWT.Free;
  end;
  ResponseContentType := 'text/plain';
end;

function TokenValido(CompactToken: String): Boolean;
var
  LKey: TJWK;
  LToken: TJWT;
  Expiracao: Int64;
begin
  Result := False;
  if (Trim(CompactToken) = '') then Exit;

  CompactToken := Troca(CompactToken, 'Bearer ', '');

  LKey := TJWK.Create(MinhaAssinaturaSecreta);
  try
    // Descompacta e verifica o token!
    LToken := TJOSE.Verify(LKey, CompactToken);
  finally
    LKey.Free;
  end;

  if Assigned(LToken) then
    try
      Result := LToken.Verified;

      if Result then
        begin
          Expiracao :=
GetValueJson(DecodeFromBase64(CorteAte(CorteApos(CompactToken, '.'), '.')),
'exp');
          Result := SecondsBetween(Now, StrToDate('01/01/1970')) <= Expiracao
+ (TimeZone * 60 * 60);
        end;
    finally
      LToken.Free;
    end
end; end.
```

APÊNDICE I - Exemplo de Codificação de Interceptação

```
Unit P3_INTERCEPT;

uses P3_GENERIC_MICROSERVICE_AUTENTICACAO;

procedure _Antes(Req: THorseRequest; Res: THorseResponse);
var S, Rota, InfoToken: String; Expiracao: Int64;
begin
  // Atribuição de cabeçalhos extras na resposta
  Res.RawWebResponse.SetCustomHeader('X-GMSDPR-debug', True);

  // Capturando a rota que está sendo processada
  Rota := AnsiLowerCase(Req.RawWebRequest.PathInfo);

  // Registro de log de requisição
  S := 'Antes........: ' + FormatDateTime('dd/mm/yyyy hh:nn:ss,zzz', Now) +
       '|' + ClientIP +
       '|' + Req.RawWebRequest.Method + '|    ' +
       '|' + Rota;
  WriteLn(S);

  // Caso se queira registrar um log com mais informações sobre cada
requisição, útil para depuração
  //WriteLn('Propriedades da requisição: ' + #10 +
PropriedadesDeObjeto(WebRequest));

  // Capturando informações de cabeçalho padrão.
  if (WebRequest.Authorization <> '') then
    begin
      InfoToken :=
DecodeFromBase64(CorteAte(CorteApos(WebRequest.Authorization, '.'), '.'));
      Expiracao := GetValueJson(InfoToken, 'exp');

      WriteLn('Authorization: ' + WebRequest.Authorization);
      WriteLn(InfoToken);
      WriteLn('Expiração do Token: ' +
DateTimeToStr(IncSecond(StrToDate('01/01/1970'), Expiracao + (TimeZone * 60 *
60))));
    end;

  // Capturando informações de cabeçalho extra.
  if (Req.Headers['X-API-KEY'] <> '') then
    WriteLn('X-API-KEY....: ' + Req.Headers['X-API-KEY']);

  // Usando um validador p/aceitar requisições à API e evitar tentativas de
invasões.
  {if (Rota = '/') and
```

```
      (Req.Headers['X-API-KEY'] <> 'chave_usuario_api') then
    raise EHorseException.Create(403, 'Chave de API não é adequada!');}

  // Exemplo de tratamento p/rotas que só podem ser acessadas se autenticado.
  if StartsStr('/config', Rota) and
//  if MatchText(Rota, ['/config', '/config/reload']) and
//  if IsMatch(Rota, '^/interpretar/.+$') and
//  if (RotaPrivada1(Rota)) and
//  if (RotaPrivada2(Rota)) and
//  if (not RotaPublica(Rota)) and
     (not
(P3_GENERIC_MICROSERVICE_AUTENTICACAO.TokenValido(WebRequest.Authorization)))
then
    raise EHorseException.Create(401, 'Você não está autorizado a acessar este
recurso ou seu token expirou!');
end;

procedure _Depois(Req: THorseRequest; Res: THorseResponse);
var S: String;
begin
  // Registro de log de conclusão
  S := 'Depois.......: ' + FormatDateTime('dd/mm/yyyy hh:nn:ss,zzz', Now) +
       '|' + WebRequest.RemoteAddr +
       '|' + WebRequest.Method +
       '|' + IntToStr(WebResponse.StatusCode) +
       '|' + WebRequest.PathInfo + #10;
  WriteLn(S);
end;

function RotaPrivada1(Rota: string): Boolean;
var V: array[0..1] of string;
begin
  V[0] := '/config';
  V[1] := '/config/reload';

  Result := MatchText(Rota, VarArrayOf(V));
end;

function RotaPrivada2(Rota: string): Boolean;
var ListaRotasPrivadas: TStringList;
begin
  ListaRotasPrivadas := TStringList.Create;
  try
    ListaRotasPrivadas.Clear;
    ListaRotasPrivadas.Add('/config');
    ListaRotasPrivadas.Add('/config/reload');

    Result := ListaRotasPrivadas.IndexOf(Rota) >= 0;
  finally
```

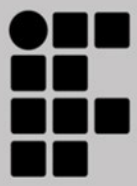
```
    ListaRotasPrivadas.Free;
  end;
end;

function RotaPublica(Rota: string): Boolean;
var ListaRotasPublicas: TStringList;
begin
  ListaRotasPublicas := TStringList.Create;
  try
    ListaRotasPublicas.Clear;
    ListaRotasPublicas.Add('/');
    ListaRotasPublicas.Add('/help');
    ListaRotasPublicas.Add('/doc');

    Result := ListaRotasPublicas.IndexOf(Rota) >= 0;
  finally
    ListaRotasPublicas.Free;
  end;
end;

end.
```

APÊNDICE J - Exemplo de Codificação executando comandos do Sistema Operacional

```
function Main: String;
begin
  Result :=
      ExecutarComandoSOCapturandoOutput('ipconfig /all', '', 0, '', True);

  ResponseContentType := 'text/plain';
end;
```

APÊNDICE K - Exemplo de Codificação consumindo JSON de fonte web.

```
function CarregarUltimosDados: string;
const LinkSerieIPCA =
'https://api.bcb.gov.br/dados/serie/bcdata.sgs.433/dados?formato=json';
var
  sJSON: string;
  JSONValue: TJSONValue;
  JSONArray: TJsonArray;
  UltimoPreco: Currency;
  UltimaData: TDate;
begin
  sJSON := HttpRestRequest(LinkSerieIPCA);
  JSONValue := TryParseJSONValue(sJSON);
  try
    JSONArray := TJSONArray(JSONValue);
    sJSON := JSONArray.Items(JSONArray.Count - 1).ToString;

    UltimoPreco := StrToCurr(Troca(GetValueJson(sJSON, 'valor'), '.', ','));
    UltimaData  := StrToDate(GetValueJson(sJSON, 'data'));

    Result := sJSON;
  finally
    JSONValue.Free;
  end;
end;
```



INSTITUTO
FEDERAL
SUDESTE DE
MINAS GERAIS