Investigando o Uso do Dispositivo de Baixo Custo Raspberry Pi como Servidor de Aplicações Web utilizando o servidor Microsoft Kestrel

Lucas D. P. dos Santos, André C. da Silva

Grupo de Pesquisa Mobilidade e Novas Tecnologias de Interação
Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas
Instituto Federal de Educação, Ciência e Tecnologia de São Paulo (IFSP)
Campus Hortolândia – SP – Brasil

lucas.dieroti@aluno.ifsp.edu.br, andre.constantino@ifsp.edu.br

Abstract. *Web applications require different hardware resources from the servers that are installed according to the functions they perform, and a key point is choosing the right equipment so as not to increase the implementation cost. In this context, this work aims to investigate the feasibility of using low-cost general-purpose computer, such as Raspberry's, to implement a web application, performing tests related to their performance and understanding the capacity of these cards. The data demonstrate that they are a viable alternative for web applications that do not require large processing and memory resources and, because they are low cost, they are a cheap, easy-to-access, easy-to-maintain and highly portable solution.*

Resumo. *Aplicações web exigem diferentes recursos de hardware dos servidores que são instaladas conforme as funções que desempenham, sendo um ponto chave a escolha adequada do equipamento de modo a não elevar o custo de implantação. Nesse contexto, este trabalho se propõe a investigar a viabilidade do uso de computadores de baixo custo, como as Raspberry's, para a implantação de uma aplicação web, realizando testes referentes a sua performance e entendendo a capacidade dessas placas. Os dados demonstram que elas são uma alternativa viável para aplicações web que não necessitam de grandes recursos de processamento e memória e, por possuírem baixo custo, são uma solução barata, de fácil acesso, de fácil manutenção e grande portabilidade.*

1. Introdução

Com o avanço das pesquisas e trabalhos envolvendo Internet das Coisas (IoT), testemunhamos um progresso exponencial no desenvolvimento das placas de prototipagem. Esta evolução se reflete não apenas em termos de poder de processamento e eficiência energética, mas também na incorporação de novos recursos e tecnologias. De fato, as placas de prototipagem, que no passado eram limitadas, agora apresentam uma gama impressionante de funcionalidades e capacidades, em termos de poder de processamento, as placas de prototipagem avançaram significativamente, processadores cada vez mais rápidos, como os baseados em arquiteturas ARM, proporcionam uma base sólida para a implementação de soluções IoT mais complexas e eficientes.

Outro aspecto fundamental do progresso das placas de prototipagem é a incorporação de módulos de comunicação avançados, como Wi-Fi, Bluetooth e NFC. Tais tecnologias de conectividade têm sido essenciais para viabilizar a comunicação eficiente entre dispositivos IoT e a infraestrutura de rede, possibilitando aplicações como monitoramento remoto, controle automatizado e interação entre dispositivos heterogêneos. A capacidade disponibilizada por esses tipos de hardware nos permite explorar possibilidades e necessidades que antes por questões de custo ou acesso à tecnologia, eram inviáveis impossibilitando diversas ideias inovadoras e benéficas para a sociedade.

A Internet no mundo atual está presente em todos os aspectos da nossa vida, seja no nosso trabalho, nos estudos, nas interações sociais e entretenimento. Aplicações web surgem diariamente para diversas finalidades e objetivos, aplicações comerciais, organizacionais e para fins sociais, e toda essa gama de aplicações dependem de uma infraestrutura física bem definida e arquitetada para seus diferentes contextos, onde na maior parte dos projetos este custo de infraestrutura pode chegar a representar 40% do custo final do projeto (Armbrust *et al.*, 2010; Varia & Mathew, 2014). Muitas aplicações não necessitam de uma grande infraestrutura de servidores e bancos de dados, aplicações pequenas podem desperdiçar todo esse poder de infraestrutura. Porém para determinados contextos e aplicações, as placas de prototipagem, com suas capacidades e desempenhos cada vez melhores, se tornam possibilidades viáveis e baratas para implementação de soluções web que possam reduzir o custo da hospedagem de aplicações web.

Assim este projeto tem a finalidade de investigar a capacidade de uma placa de prototipagem (*Raspberry pi Model B+*), trabalhando como um servidor web *Microsoft Kestrel* de uma aplicação *Web API*. Vamos analisar o comportamento das principais métricas da placa durante seu funcionamento como servidor web, pretendemos analisar a taxa de utilização da Memória *RAM*, do processador, o tempo médio das requisições de rede entre outras métricas, e concluir a capacidade e possibilidade do uso dessas placas como servidores de aplicações que não demandem hardwares mais caros.

2. Referencial Teórico

Nesta Seção abordaremos as principais referências teóricas que nortearam este projeto, e ajudam a entender como foi pensado e montado este projeto.

2.1. Raspberry Pi

A Raspberry Pi é uma série de microcomputadores de placa única desenvolvido pela Raspberry Pi Foundation, uma organização educacional sem fins lucrativos do Reino Unido. A História da Raspberry começa em 2006 quando Eben Upton, Rob Mullins, Jack Lang e Alan Mycroft, da Universidade de Cambridge, começaram a discutir a crescente falta de habilidade de seus alunos quando o assunto era programação (RASPBERRY PI FOUNDATION. Raspberry Pi. Disponível em: https://www.raspberrypi.org. Acesso em: 14 mar. 2025.).

Em resposta a essa preocupação a equipe começou a trabalhar em um projeto que pudesse ajudar e facilitar o aprendizado dos alunos em programação. Após anos de desenvolvimento, em 2012, a Raspberry Pi Model B foi lançada. Este pequeno computador do tamanho de um cartão de crédito, possui um processador ARM, porta HDMI, USB e rodava uma variedade de sistemas operacionais, incluindo Linux.

Ao longo dos anos a Raspberry Pi Foundation foi lançando várias versões da Raspberry Pi, e em cada nova versão uma nova tecnologia era adicionada e melhorias em termos de performance, processamento, memória e conectividade era apresentada aos usuários da plataforma. Na Tabela 1 é apresentada a família Raspberry Pi com alguns dos modelos lançados e suas especificações técnicas, que foi compilada a partir dos dados obtidos em Gamer Antigo (2024).

Uma das maiores vantagens da Raspberry Pi são: sua portabilidade e seu baixo consumo de energia (Tabela 1). Medindo apenas 8,5 cm por 5,6 cm nas versões B e somente 6,5 cm por 3 cm nas versões Zero). Além disso a Comunidade em torno da Raspberry Pi foi se desenvolvendo e com ela uma série de acessórios e projetos foram desenvolvidos e disponibilizados ao público em geral, o que amplia ainda mais as possibilidades de uso bem como a quantidade de entusiastas da plataforma. Desde servidores domésticos e centros de mídia até sistemas de automação residencial e dispositivos de IoT, a Raspberry Pi encontrou uma ampla gama de aplicações sejam residenciais, educacionais ou na indústria.

						
	Raspberry Pi 4 Modelo B	Raspberry Pi 3 Modelo b+	Raspberry Pi 3 modelo B	Raspberry Pi 2 Modelo B	Raspberry Pi Modelo B+	Raspberry Pi Modelo A
Porta Ethernet	GigaBit Ethernet	GigaBit Ethernet	GigaBit Ethernet	Fast Ethernet	Fast Ethernet	Fast Ethernet
GPU	Videocore VI	Videocore VI	Videocore VI	Videocore VI	Videocore VI	Videocore VI
Saida de Video	2 HDMI	1 HDMI	1 HDMI	1 HDMI	1 HDMI	1 HDMI
WiFi	2.4 Ghz e 5 Ghz IEEE 802.11b/g/n/ac Wireless LAN	2.4 Ghz e 5 Ghz IEEE 802.11b/g/n/ac Wireless LAN	2.4 Ghz e 5 Ghz IEEE 802.11b/g/n/ac Wireless LAN	Não Possui	Não Possui	Não Possui
Bluetooth	Bluetooth 5.0 BLE	Bluetooth 4.2 BLE	Bluetooth 4.2 BLE	Não Possui	Não Possui	Não Possui
Armazenamento	MicroSD	MicroSD	MicroSD	MicroSD	MicroSD	MicroSD
Processador	Broadcom BCM2711 Quad Core Cortex A72 (ARM v8) 64bit SoC @ 1.5Ghz com 1MB L2 cache, 32KB L1	Broadcom BCM2837B0 Cortex A53 (ARM v8) 64bit SoC @ 1.4Ghz	Broadcom BCM2837 64bit Quad Core 1.2 CPU	Broadcom BCM2826 32bit Quad Core Cortex A72 (ARM v8) 64bit SoC @ 900Ghz	Broadcom BCM2835 32bit Quad Core Cortex A72 (ARM v8) 64bit SoC @ 900Ghz 700Ghz ARM 11	Broadcom BCM2835 32bit Quad Core Cortex A72 (ARM v8) 64bit SoC @ 900Ghz 700Ghz ARM 11
Memoria RAM	1GB, 2GB, 4GB LPDDR4	1GB LPDDR2 SDRAM	1GB LPDDR2 SDRAM	1GB LPDDR2 SDRAM	512 MB RAM	256 MB RAM
GPIO	40 Pinos	40 Pinos	40 Pinos	40 Pinos	40 Pinos	26 Pinos
USB	2 USB 2.0 & 2 USB 3.0	4 USB 2.0	4 USB 2.0	4 USB 2.0	2 USB 2.0	1 USB 2.0
Voltagem/Amperagem	5V e 3A	5V e 2,5A	5V e 2A	5V e 1,8A	5V e 1,8A	5V e 700mA

Tabela 1. Seis modelos da Família Raspberry Pi e suas características.

2.2. Aplicações Web, Servidores Web e Sistema Operacional

Nesta Seção abordaremos a essência das aplicações web e os servidores web, dois componentes essenciais no ecossistema da internet. A compreensão profunda deste tema é crucial para qualquer desenvolvedor, administrador de sistema ou pesquisador interessado no mundo da web e suas aplicações.

Uma aplicação web é um software que pode ser acessado e executado através de um navegador. Ou seja, diferente dos programas tradicionais que precisam ser instalados para serem utilizados, a aplicação web roda diretamente em navegadores, como Firefox, Google Chrome e Safari, ou entre as próprias aplicações via protocolo HTTP/HTTPS.

Um sistema operacional é um software fundamental que gerencia os recursos de hardware e software de um computador, atuando como intermediário entre o usuário e o hardware. Ele controla dispositivos de entrada e saída, gerencia memória e processadores, e permite a execução de programas e aplicativos.O Raspberry Pi OS (anteriormente

chamado de Raspbian) é uma distribuição do sistema operacional Linux baseada no Debian, desenvolvida especificamente para o Raspberry Pi.

Algumas das principais características que este trabalho aborda sobre as aplicações web são:

- Acessibilidade: As aplicações web são desenvolvidas em sua essência para serem utilizadas por pessoas distantes umas das outras e distantes do servidor que disponibiliza essa aplicação, logo a capacidade da aplicação de se manter acessível aos acessos é vital para a existência da aplicação;
- Processamento: Buscamos analisar qual a capacidade de processamento necessária para que uma aplicação web possa funcionar de forma eficaz e atender as necessidades de seus usuários. Tal afirmação necessita de uma ampla gama de testes, bem como as ferramentas utilizadas, tanto no desenvolvimento quanto na hospedagem da aplicação são fatores que influenciam o resultado;
- Capacidade: Buscamos analisar as necessidades em relação a capacidade de armazenamento de uma aplicação web, nosso trabalho busca analisar os números de uma aplicação web específica e verificar quais as suas necessidades referentes a armazenamento, seja em banco de dados ou servidores de arquivos para um funcionamento eficaz e performático;
- Tecnologias: Analisaremos também diferentes tecnologias e suas capacidades em relação ao trabalho conjunto com as aplicações web, analisaremos servidores muito utilizados no mercado em conjunto com placas de prototipagem, analisando a performance da plataforma Web nesses diferentes contextos.

Tanenbaum (2016) define um servidor como um tipo específico de sistema operacional projetado para fornecer serviços a múltiplos usuários simultaneamente. Esses sistemas são otimizados para operações que exigem alta confiabilidade, segurança e desempenho.

Já Silberschatz (2014), em seu livro *Fundamentos de Sistemas Operacionais*, aborda servidores no contexto de sistemas distribuídos, onde um servidor é uma entidade que fornece serviços a outros programas ou dispositivos, conhecidos como clientes. Essa arquitetura cliente-servidor é fundamental para a organização e funcionamento de sistemas distribuídos, permitindo a distribuição de tarefas e recursos de forma eficiente.

3. Trabalhos Correlatos

Na tese de conclusão de curso de Istifanos e Tekahun (2020), de título "*Performance Evaluation of a Raspberry Pi as a Web Server",* os autores buscaram analisar o comportamento de uma Raspberry Pi 3 Modelo B+ como um servidor web em diferentes contextos e realizando uma comparação com uma máquina mais robusta, um *laptop*. Os autores utilizaram ambas as máquinas, o Raspberry e *laptop*, como servidor de 2 aplicações, uma estática e outra dinâmica, e analisaram o comportamento de 2 servidores HTTP, o NGINX e o Apache.

Os autores chegaram à conclusão de que o *laptop*, por ter uma configuração técnica muito maior que a do Raspberry, se sai melhor nas análises com tempos de latência e capacidade de conexão simultânea maior, bem como uma menor utilização de

processamento, porém os autores mostram que mesmo com a diferença entre os equipamentos a Raspberry entrega todas as necessidades das aplicações, performance, processamento e memória, ou seja, mesmo que o *laptop* apresenta uma capacidade técnica melhor essa vantagem acaba sendo desnecessária para aplicações simples, já que a Raspberry com suas especificações técnicas inferiores é capaz de atender aos cenários propostos.

No trabalho de título "*Sistema Operacional em Raspberry Pi para Serviços Web e Monitoramento",* de autoria de Silva (UTFPR, 2020), *é* proposto a configuração de uma Raspberry pi 3 Model B como um 'Mini servidor de bolso', capaz de hospedar serviços web (como banco de dados e hospedagem), e monitorar ambientes de *datacenter*. Esse trabalho aborda a capacidade de integração com serviços de nuvem (como AWS e AZURE) e a viabilidade de reduzir os custos em pequenas e médias empresas. A metodologia empregada pelo autor inclui a instalação de sistemas operacionais portáteis com diferentes distribuições de Linux, e a análise do desempenho em tarefas de monitoramento térmico e performance de hospedagem

Silva (2020) utilizou as ferramentas Zabbix e Grafana para a respectiva coleta e visualização gráfica dos dados gerados pela execução da aplicação. O trabalho chega à conclusão de que a operação ainda pode ser melhorada com o uso de novas tecnologias de hardware já que foi utilizada uma Raspberry Pi defasada, porém com ótimas perspectivas para pequenas aplicações.

4. Materiais e Métodos

4.1. Raspberry Pi

Para este projeto, como hardware para a instalação dos servidores HTTP e dos recursos necessários para este trabalho, utilizaremos a Raspberry Pi 3 Model B+ (Figura 1). A Raspberry Pi 3 Model B+ possui 1GB de memória RAM, e um processador ARM de 1.4Ghz dedicado apenas aos serviços necessários para seu próprio funcionamento e para a hospedagem da nossa aplicação web. As demais especificações desse hardware estão na Tabela 1.

Figura 1. Raspberry Pi 3 Modelo B+. Fonte: Raspberrypi.com (2024)

4.2. Armazenamento – Cartão de memória

Para garantir uma capacidade de armazenamento adicional e lidar com a quantidade de dados que serão processados, utilizaremos um cartão de memória de 64GB. Essa expansão permitirá uma ampla margem de armazenamento, essencial para garantir que

todos os dados relevantes sejam capturados e processados de forma eficiente pela placa Raspberry Pi. Note que este cartão de memória é de tecnologia simples, possuindo uma velocidade de escrita reduzida. O Ideal e recomendado seriam cartões de memória classe 10, que possuem uma maior velocidade de escrita e leitura, reduzindo erros de *IO*, corrupção de dados em escritas concorrentes e tem uma maior durabilidade.

4.3. Sistemas Operacionais e Servidores

Neste projeto foi utilizado como sistema operacional o sistema nativo da Raspberry Pi, o Raspbian. O Raspbian é uma distribuição Linux baseada no Debian otimizada para dispositivos Raspberry Pi. O Raspbian é a escolha ideal por sua compatibilidade com os processadores ARM e sua vasta comunidade de suporte, além de ser leve e adequado para soluções IoT e servidores de pequena escala (GILL, SINGH e KAUR, 2015).

O Kestrel é o servidor web embutido padrão no ASP.NET Core. Ele foi projetado para ser leve, rápido e escalável, utilizando programação assíncrona e a infraestrutura de I/O do sistema operacional (baseado originalmente na biblioteca libuv, embora atualmente use o transporte padrão via sockets) (MICROSOFT, 2025). Possui as seguintes características:

- **Processamento de Requisições: q**uando a aplicação ASP.NET Core inicia, o Kestrel é configurado (normalmente via *Program.cs* ou pelas configurações padrão do *CreateDefaultBuilder*) para escutar por requisições HTTP. Ao receber uma requisição, ele a processa utilizando um *loop* de eventos e operações assíncronas, o que permite o processamento de um grande número de conexões simultâneas com baixa sobrecarga;
- ***Pipeline* de Middleware: a**pós o Kestrel fazer a leitura e o *parsing* inicial do HTTP, a requisição é passada para o *pipeline* de *middleware* da aplicação, onde diversos componentes (como autenticação, roteamento, manipulação de erros etc.) são executados até que uma resposta seja gerada e enviada de volta ao cliente;
- **Suporte a Protocolos:** Kestrel suporta HTTP/1.1 e HTTP/2 e, em versões mais recentes, até HTTP/3, garantindo alta performance em diferentes cenários. Ele também oferece suporte a conexões seguras (HTTPS) através da configuração de certificados SSL/TLS.

4.4. Softwares

Foram utilizados diversos softwares de apoio para a realização desse projeto, desde softwares necessários para codificações referentes a plataforma web que hospedaremos no servidor, até softwares para compressão, teste, e monitoramento do servidor e da aplicação.

Como ambiente de programação escolheu-se o Visual Studio, um software da empresa Microsoft, utilizado para codificação, teste, compilação e compartilhamento de código em diversas linguagens; é amplamente utilizado seja no mundo acadêmico ou corporativo. Sua facilidade de utilização, a ampla gama de linguagens disponíveis bem como sua leveza e robustez fazem uma das plataformas de programação mais utilizadas no mundo.

Como sistema gerenciador de banco de dados optou-se pelo SQLite para armazenamento dos dados. Diferentemente de outros sistemas de gerenciamento de

bancos de dados, o SQLite não requer um servidor separado para operar; em vez disso, ele lê e grava diretamente em arquivos de banco de dados no disco. Essa característica o torna uma opção leve e eficiente para aplicações que necessitam de um banco de dados integrado, como soluções web e móveis.

Foi utilizado para coleta das métricas da Raspberry pi o software Prometheus, O Prometheus é uma ferramenta de monitoramento e alerta de código aberto, originalmente desenvolvida pela SoundCloud. Ele é amplamente utilizado para coletar, armazenar e analisar métricas como processamento, memoria, rede entre outras informações de sistemas e aplicações em tempo real.

O Grafana é uma plataforma de visualização de dados de código aberto que permite criar *dashboards* interativos e monitorar métricas de sistemas, aplicações e infraestruturas em tempo real. Com uma interface intuitiva e flexível, o Grafana oferece integração com uma ampla gama de fontes de dados, incluindo bancos de dados relacionais e não relacionais, sistemas de monitoramento e serviços em nuvem. Isso possibilita às equipes de TI e DevOps unificar informações dispersas em painéis personalizados, facilitando a análise e a tomada de decisões baseadas em dados. Sua natureza *open source* e a ativa comunidade de desenvolvedores garantem constante evolução e a disponibilidade de diversos *plugins* e extensões para ampliar suas funcionalidades.

5. Desenvolvimento

A primeira etapa de desenvolvimento foi a criação de uma aplicação web para *upload* e *download* de arquivos de mídia (como imagens e vídeos). Em seguida, a configuração da plataforma de testes, com a instalação da aplicação criada e as ferramentas para coleta de dados. A terceira etapa foi o desenvolvimento de uma aplicação simples para consumo da API. Em seguida, a realização da coleta e da análise dos dados.

5.1. Desenvolvimento da aplicação web API – *Web Test Performance*

Para este trabalho foi desenvolvida uma aplicação Web baseada em API utilizando as tecnologias do .NET *framework Core* 8, que fornece as principais bibliotecas nativas da Microsoft para o desenvolvimento de aplicações web. Como banco de dados utilizamos o SQLite por sua simplicidade, poder, e por rodar em uma diversidade de ambientes sem a necessidade de instalação de servidores dedicados para o banco de dados. E para a interface gráfica utilizamos o *framework* Swagger, que mapeia todos os *controllers* de uma aplicação Web com API e fornece uma interface gráfica simples para os testes que faremos. A IDE utilizada no desenvolvimento desta aplicação foi o Visual Studio.

O Primeiro passo no desenvolvimento foi a instalação das bibliotecas necessárias para a aplicação. A biblioteca *Entity Framework* será o ORM (*Object-Relational Model*) utilizado na aplicação para facilitar as operações e mapeamentos do banco de dados. A biblioteca *SwashBuckle* nos fornecerá a capacidade de empregar o Swagger como interface gráfica.

Após a instalação das bibliotecas necessárias, começamos o desenvolvimento da aplicação definindo as classes necessárias. Iniciou-se programando classe *MediaFile* (Figura 2). Nossa API fará operações de salvar e retornar arquivos de mídia, seja ele foto, vídeo ou música. Logo nossa classe é definida com os atributos necessários, além de

atributos para armazenar o tempo de início e fim do *upload* do arquivo (linhas 10 e 11, respectivamente).

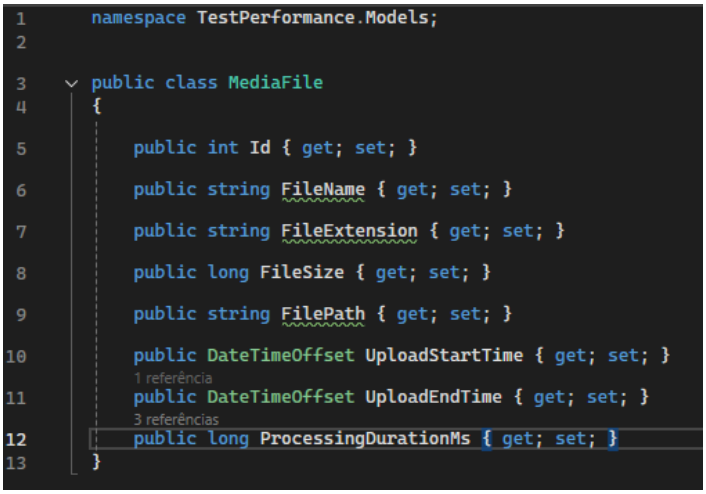

Figura 2. Código da *classe MediaFile* e seus métodos.

A Figura 3 exibe o código da classe criada a seguir, a classe *AppDbContext*, que herda de *DbContext* (linha 6). Essa classe realiza o mapeamento do ORM da classe *MediaFile*.

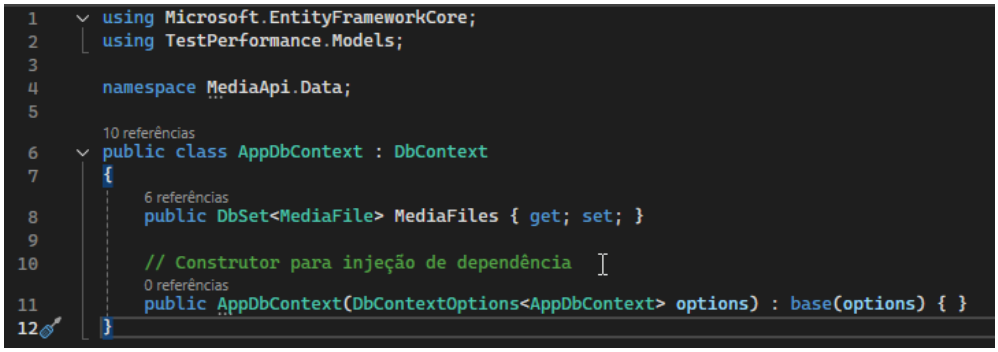

Figura 3. Código da classe *AppDbConext* que realiza o mapeamento ORM da classe *MediaFile*.

Agora implementamos o *Web Controller* que definirá e controlará as operações disponíveis pela nossa API (Figura 4). As linhas 21 e 22 definem que o método *UploadFile* irá tratar a requisição POST */upload*. As linhas 58 e 59 definem que o método *ListFiles* irá tratar a requisição GET */list*. As linhas 65 e 66 definem que o método *ViewFile* irá tratar a requisição GET */view/{id}*, cujo valor da variável *id* é passado como parâmetro para o método *ViewFile.* As linhas 77 e 78 definem que o método *DeleteFile* irá tratar a requisição DELETE */delete/{id}*, cujo valor da variável *id* é passado como parâmetro para o método *DeleteFile.* As linhas 92 e 93 definem que o método *GetStatistics* irá tratar a requisição GET */stats* . As linhas 113 e 114 definem que o método *ResetDatabase* irá tratar a requisição POST */reset-database* .

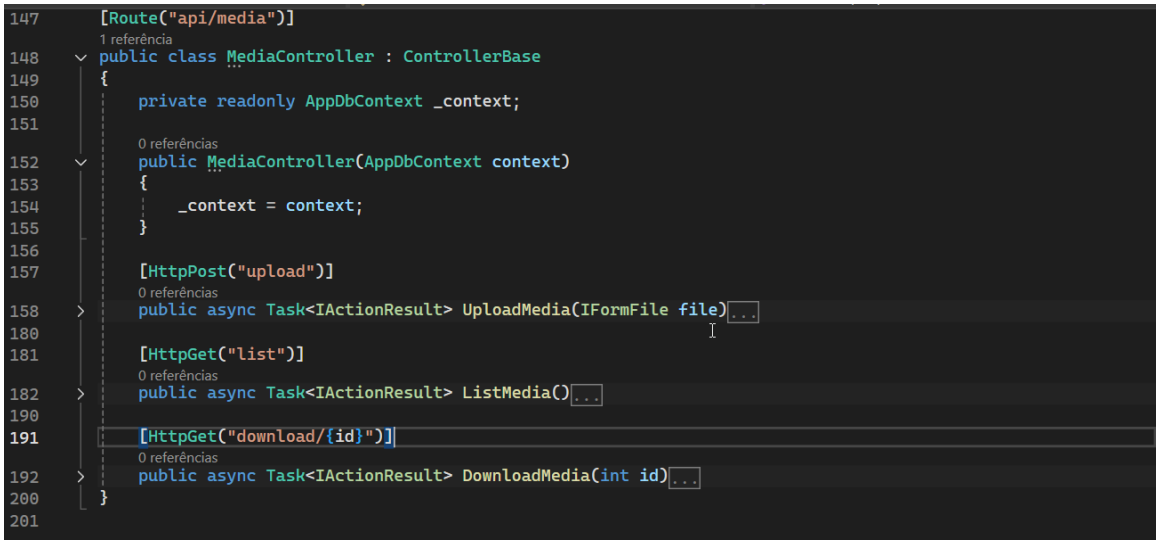

Figura 4. Classe *Controller* e seus métodos que tratam as requisições HTTP.

O Método *Upload* (Figura 5), que é executado ao receber uma requisição do tipo POST em */upload*, é responsável por capturar, salvar e extrair os metadados das mídias que forem enviadas para a aplicação tais como mídias são convertidas para bytes e salvas no banco de dados da aplicação.

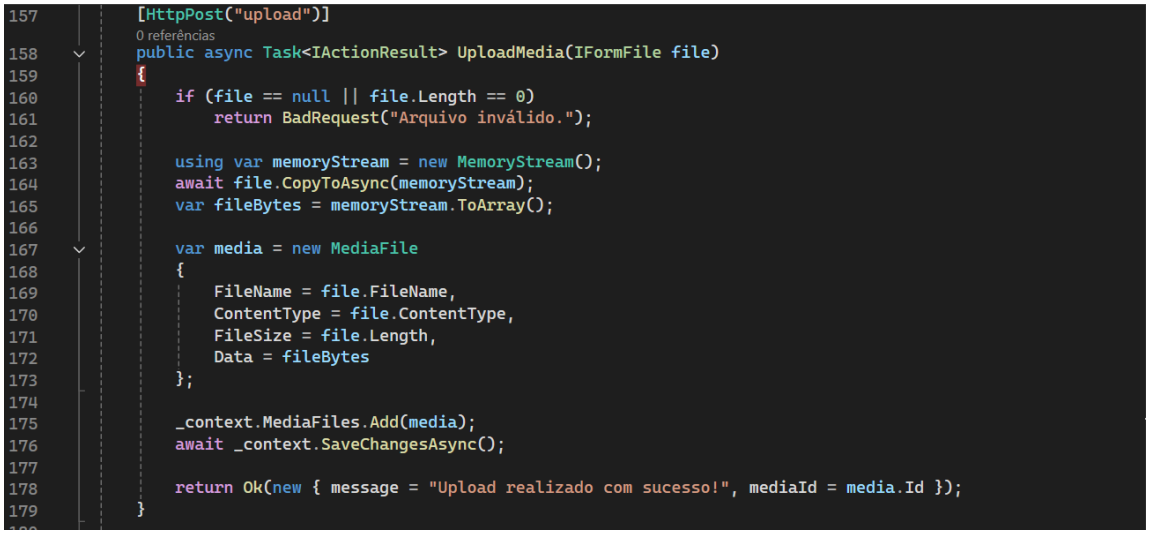

Figura 5. Código do método *Upload* da classe *Controller*.

O método *Upload* é o principal método deste trabalho, porém, para a realização de diferentes cenários de testes, foram criados outros métodos HTTP, o *ListFiles* (Figura 6), *DownloadMedia* (Figura 7), *DeleteFile* e *GetStatistics*. O método *GetStatistics* retorna as estatísticas atuais da aplicação, como quantidade de mídias salvas, o tempo médio de *upload* dessas mídias e os tipos de mídias salvas.

A aplicação final terá uma interface gráfica fornecida pelo *framework* Swagger (Figura 8), que pode ser utilizada para acionar as funcionalidades (Figura 9).

```
181     [HttpGet("list")]
        0 referências
182     public async Task<IActionResult> ListMedia()
183     {
184         var mediaFiles = await _context.MediaFiles
185             .Select(m => new { m.Id, m.FileName, m.ContentType, m.FileSize, m.UploadedAt })
186             .ToListAsync();
187
188         return Ok(mediaFiles);
189     }
```

Figura 6. Código do método *List* da classe *Controller*.

```
191     [HttpGet("download/{id}")]
        0 referências
192     public async Task<IActionResult> DownloadMedia(int id)
193     {
194         var media = await _context.MediaFiles.FindAsync(id);
195         if (media == null)
196             return NotFound("Arquivo não encontrado.");
197
198         return File(media.Data, media.ContentType, media.FileName);
199     }
```

Figura 7. Código do método *Download* da classe *Controller*.

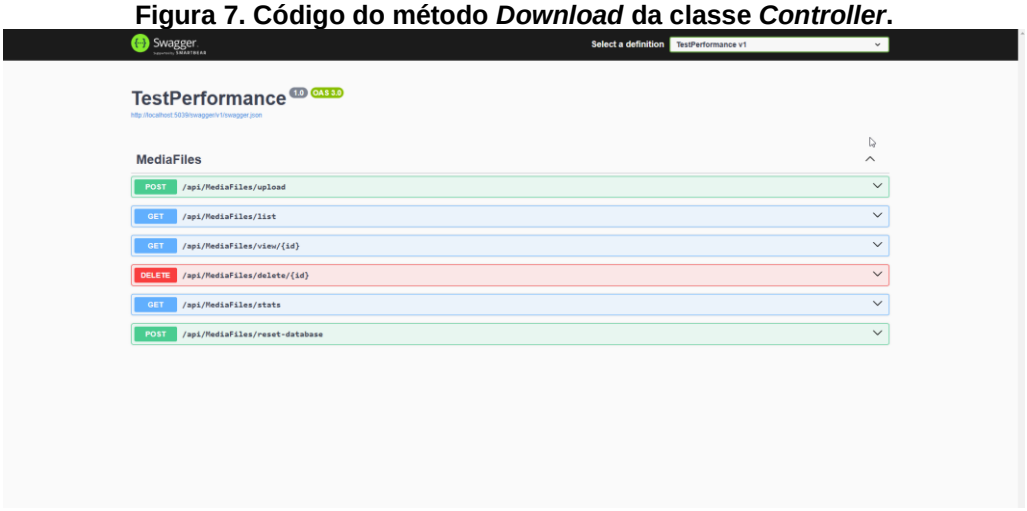

Figura 8. Interface de usuário gerada pelo Swagger UI para acesso a API criada.

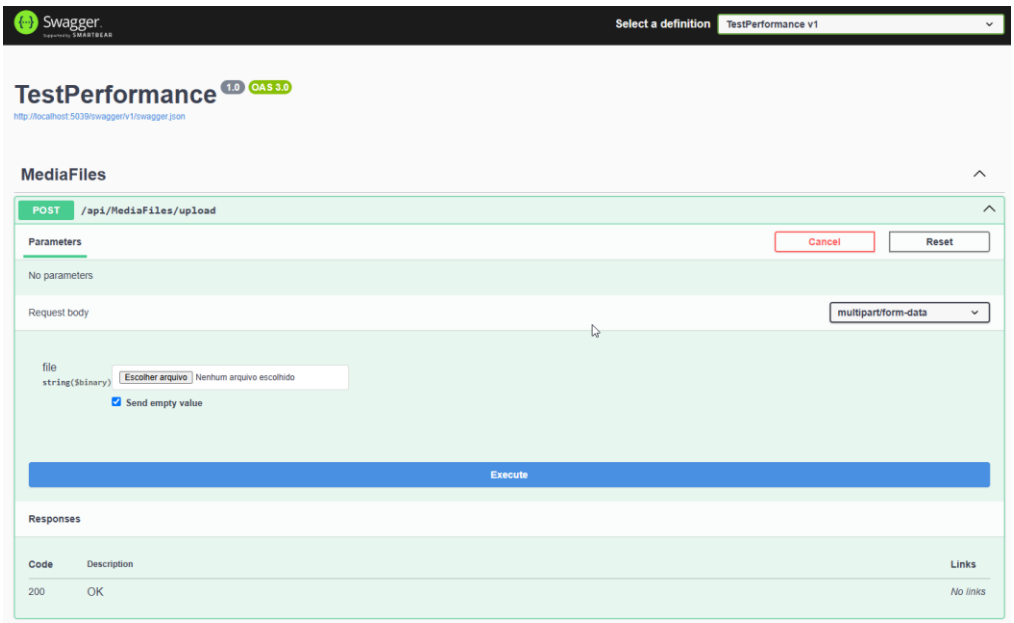

Figura 9. Exemplificação do uso da interface gerada pelo Swagger UI para *upload* de arquivos.

5.2. Configuração do ambiente de coleta de dados

Para iniciar as configurações da Raspberry Pi, utilizamos um cartão de memória de 64GB, e o software Raspberry Pi Imager (Disponível em www.Raspberrypi.com). O cartão de memória foi utilizado para a instalação do sistema operacional Raspibian OS ARM64.

Para a configuração do ambiente da Raspberry Pi via linha de comandos Linux realizamos a instalação dos seguintes softwares necessários para a nossa aplicação e servidor web:

- SQLite – Banco de dados da Aplicação Web;
- SDK .NET - Kit de bibliotecas de desenvolvimento .NET;
- Entity Framework - ORM (Mapeador de Objeto-Relacional) para acesso e manipulação do banco de dados;

Para a configuração do servidor web realizamos as seguintes instalações e configurações na Raspberry pi:

- Instalação e configuração do servidor .Net Kestrel (Porta 5000);
- Instalação e configuração do Prometheus (Porta 9100);
- Instalação e configuração do Grafana (Porta 9300).

Para as configurações de rede foram realizadas as seguintes etapas:

- Iniciamos na Raspberry Pi definindo o IP estático no arquivo do DHCPCD.conf;
- Depois acessamos o painel de controle do roteador de internet do ambiente de produção do servidor web, e realizamos o roteamento de porta, onde definimos a porta 5000, como a porta que o servidor será atendido, e sinalizando o IP da Raspberry pi como destino das requisições TCP/IP. Deixando assim a aplicação acessível a rede.

Após os ambientes necessários estarem prontos realizamos a transferência dos arquivos da aplicação web para a pasta do servidor e a execução do binário da aplicação:

- Transferência dos arquivos binários da aplicação via comando:
  scp : scp -r
  "C:\Users\lucas\source\repos\TestPerformance\TestPerformance\bin\Release\net8.0\linux-arm64\publish"
  ifsp@192.168.0.166:/home/ifsp/Desktop/TestPerformance”
- Execução do binário da aplicação na pasta do servidor via comando:
  dotnet [TestPerformance.dll]
  Como estamos usando o *Entity Framework*, sempre que o comando *dotnet [TestPerformance.dll]* for executado, a própria aplicação disponibilizará o seu banco de dados.

5.3. Desenvolvimento de uma aplicação para consumo da API

Foi desenvolvido um aplicativo de console .NET simples (Figura 10), utilizando a linguagem de programação C#, para realizar diversas requisições HTTP para a aplicação, requisições essas assíncronas para simular um volume de acessos a nossa aplicação

(métodos *DownloadMediaFile* e *UploadFileToApi* nas linhas 35 e 61, respectivamente). Neste cenário enviar 1.000 requisições com arquivos de fotos de tamanho 12,0 KB em média, com requisições de intervalo de 1 segundo entre elas (linha 29).

```
7
                0 referências
8   class MediaApiConsoleApp
9   {
10      private static readonly HttpClient httpClient = new HttpClient();
11      private const string apiBaseUrl = "http://179.159.108.59:5000/api/media";
12      private const string mediaSourceUrl = "https://picsum.photos/200/300";
13
                0 referências
14      static async Task Main(string[] args)
15      {
16          Console.WriteLine("Iniciando envio automatizado de arquivos...");
17
18          for (int i = 1; i <= 1000; i++)
19          {
20              var fileName = $"image_{i}.jpg";
21              var filePath = await DownloadMediaFile(fileName);
22
23              if (filePath != null)
24              {
25                  await UploadFileToApi(filePath);
26                  File.Delete(filePath);
27              }
28
29              await Task.Delay(100); // Pequeno intervalo entre envios
30          }
31
32          Console.WriteLine("Envio concluído.");
33      }
34
                1 referência
35      private static async Task<string?> DownloadMediaFile(string fileName)...
60
                1 referência
61      private static async Task UploadFileToApi(string filePath)...
89
```

Figura 10. Programa desenvolvido para realizar as requisições automaticamente a aplicação no Raspberry Pi.

Esta aplicação utiliza a API do PicSum (Disponível em *https://picsum.photos*) (linha 12) para o *download* das imagens que serão posteriormente enviados para nossa aplicação hospedada na Raspberry Pi.

5.4. Cenários de testes

Os testes foram realizados em um dia de verão com temperatura média no momento dos testes de 26 graus, os testes foram realizados no período noturno. Vale destacar que a placa utilizada não possui nenhum tipo de refrigeração tornando este dado de temperatura ambiente pertinente para o trabalho. Para a execução dos testes, utilizamos algumas ferramentas para auxiliar a automatizar e gerenciar os testes. Dividimos a apresentação em três cenários.

Cenário 1: uso de um computador desktop para envio das requisições, este computador está na mesma rede *wi-fi* que a Raspberry Pi.

Cenário 2: nesse cenário utilizamos um *smartphone*, em conjunto com um software de automação de cliques e a nossa interface gráfica do Swagger (Figura 11) para simular envios de arquivos para a nossa aplicação, as mídias enviadas a partir do celular são maiores para fins de teste com tamanhos entre 5MB e 10MB, e vídeos entre 10MB e 60MB.

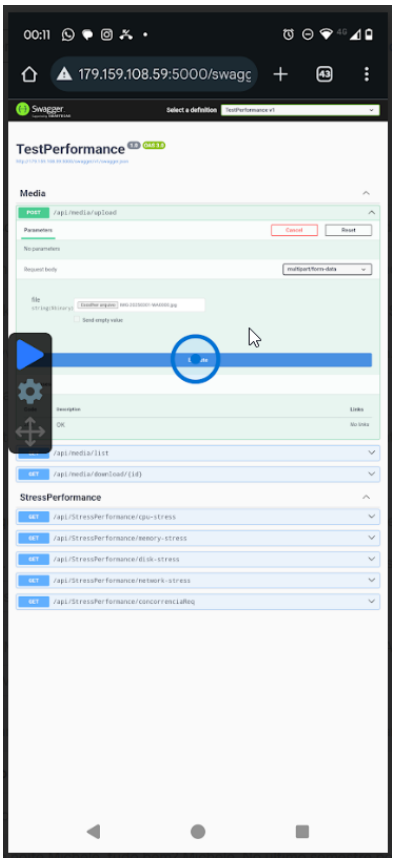

Figura 11. Interface Gráfica do Swagger executando em um *smartphone* Android.

Cenário 3: utilizaremos um terceiro computador, executando a mesma aplicação do cenário de teste 1, e utilizando o software desenvolvido para a automação, para realizar requisições de *upload* de arquivos. Diferente do Cenário 1, este cenário a aplicação estará fora da rede da Raspberry Pi, em uma rede de internet móvel 4G/5G.

5.5. Coleta de dados

Os testes foram feitos em um intervalo de 3 horas (19:15 à 21:15), utilizando diferentes combinações dos cenários citados acima onde todas as máquinas estavam em uma rede externa a rede do servidor.

Em estado ocioso nossa Raspberry Pi tinha seus recursos com a utilização, a saber, a temperatura (Figura 12), a memória RAM (Figura 13) e uso de CPU (Figura 14).

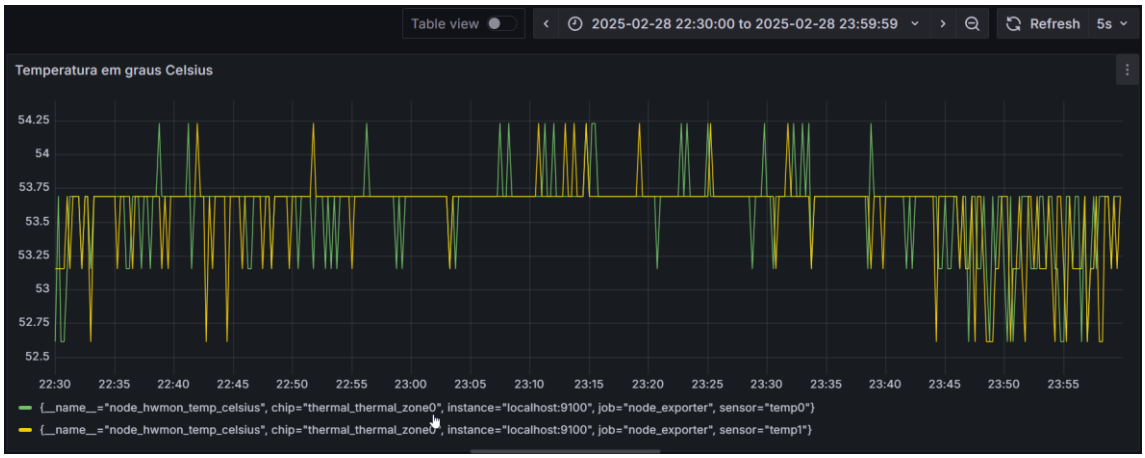

Figura 12. Temperatura da Raspberry Pi em ociosidade.

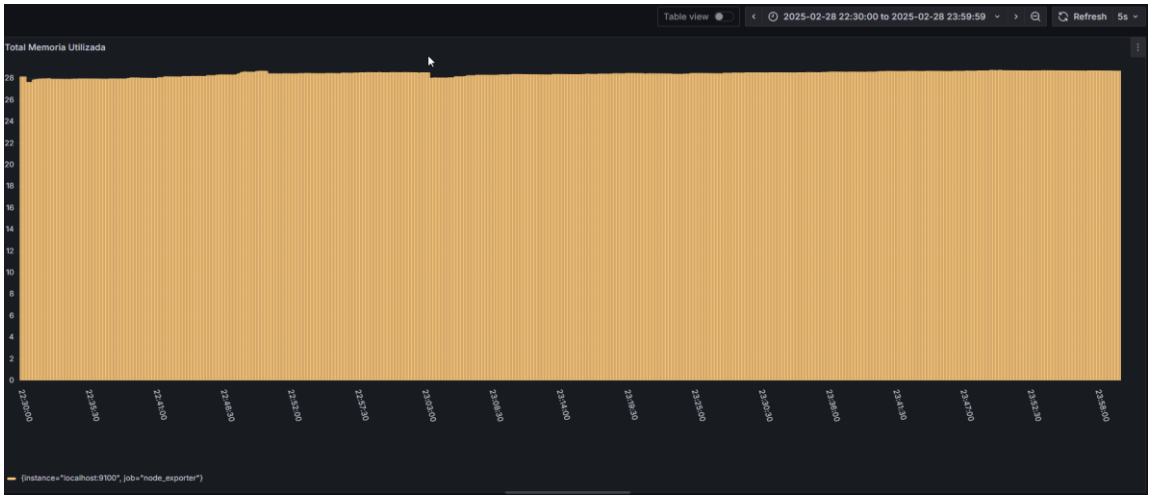

Figura 13. Memória RAM da Raspberry Pi em ociosidade.

Figura 14. CPU da Raspberry pi em ociosidade.

Os gráficos das Figuras 15 a 18 exibem os dados coletados. No eixo X estão os horários dos dados coletados entre os horários de 18:15 até o horário de 21:15, divididos da seguinte forma:

- 18:15 até 19:00 – Cenário 1.
- 19:00 até 20:00 – Cenário 1 e Cenário 2.
- 20:00 até 21:15 – Cenário 1, Cenário 2 e Cenário 3.

As linhas verticais azuis ou amarelas delimitam os espaços de tempo.

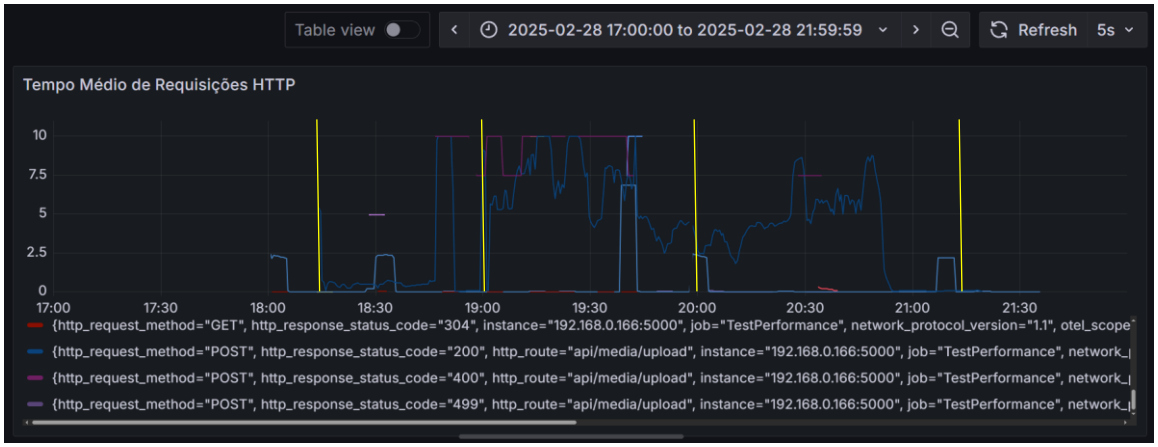

Figura 15. Grafana UI Análise de Tempo de Requisição.

Com o gráfico da Figura 15 podemos analisar que, durante a execução do cenário 1, o tempo médio de resposta da API ficou em menos de 2 segundos para as operações de *upload* de arquivos. Podemos notar que próximo às 19:00 houve falhas nas requisições HTTPs, o que fez com que o tempo de resposta para as outras operações, principalmente as requisições POSTs experienciasse um aumento significativo no tempo da requisição de 10 segundos.

No segundo período, onde iniciamos a execução em paralelo do segundo cenário, podemos notar que as requisições tiverem um aumento em sua taxa de resposta, passando a serem atendidas em uma média de 7,5 segundos. Também podemos notar que a taxa de falha das requisições também aumentou, essas falhas ocorreram, pois conforme mencionado no Cenário 2, havia alguns arquivos de mídia de tamanho um pouco maior, pois eram vídeos, e em nossos testes estamos utilizando um cartão de memória comum com tempo de escrita de 10 megabyte por segundo, mais lento que um HD convencional. Isso acarretou erros de '*Timeout*', logo com o servidor ocupado em requisições demoradas as outras requisições também têm seu tempo aumentado.

No período 3, no qual todos os cenários foram executados em paralelo, vale salientar que nesse período o Cenário 2, no qual utilizados o celular, os arquivos enviados passaram a ser fotos, mais leves. O gráfico mostra que a taxa de falhas das requisições teve uma redução drástica, e quanto ocorreu não impactou tanto outras requisições. Podemos notar que as requisições de POSTs de mídias foram atendidas em média em 5 segundos, com um leve aumento notável durante e após algumas falhas de requisição.

Na Figura 16 temos podemos analisar a taxa de utilização da memória RAM da nossa Raspberry Pi durante os cenários de teste. No primeiro período podemos ver que a taxa de utilização da memória RAM ficou em média em 40%. Já no segundo período de análise, a taxa de memória se manteve estável, porém conforme analisado no gráfico anterior, as falhas acabaram gerando um pequeno pico de utilização de memória durante esse período, o que não foi experienciado no último período com os 3 cenários em

execução paralela, onde houve um pequeno aumento na utilização da memória RAM passando a fica na média de 47% e em poucas ocasiões passando dos 50%.

Figura 16. Grafana UI Análise de Memória RAM.

No gráfico de uso de CPU da Figura 17 podemos fazer as algumas constatações a respeito da análise da CPU. No primeiro cenário podemos notar que o uso da CPU teve uma média de utilização abaixo dos 20%, porém com um pico de uso de CPU em utilização de mais de 40%. Esse aumento pode ser explicado pela falha que aconteceu em uma requisição HTTP que podemos observar no gráfico da Figura 15. No segundo cenário a taxa de utilização também ficou em menos de 20% com um grande pico no final do período, gerado pelas falhas nas requisições de arquivos de vídeo do cenário 2, como a quantidade de falhas desse cenário foi bem alta nos contextos dos nossos testes, esse pico de utilização também foi bem alto, levando a utilização da CPU para mais de 80%. Já no terceiro período, sem as ocorrências de falhas, o tempo das requisições foi estável ficando abaixo dos 20%.

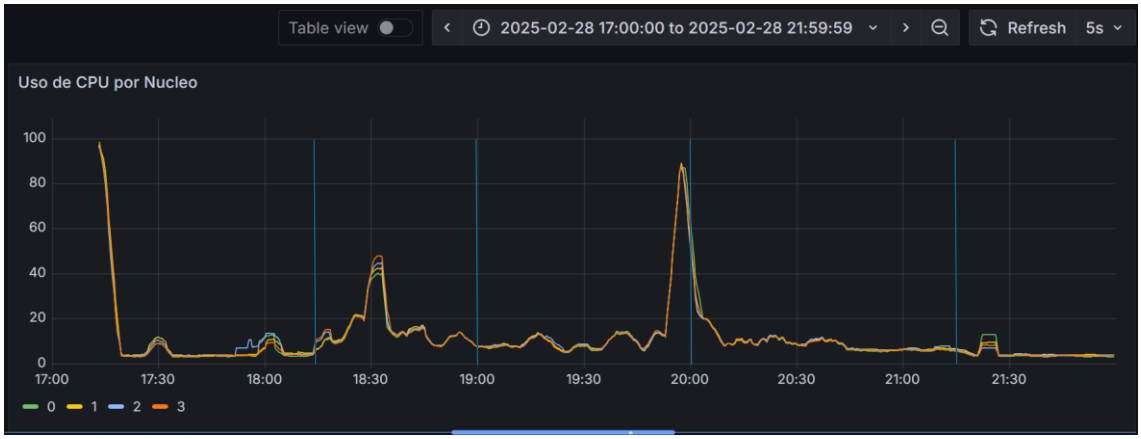

Figura 17. Grafana UI Análise de CPU.

Na Figura 18 podemos analisar como foi o comportamento da temperatura da Raspberry Pi durante os testes. Podemos analisar que a média de temperatura da placa foi a mesma durante todo o teste ficando na casa dos 55 graus Celsius, porém podemos analisar que as falhas mostradas nos gráficos das figuras anteriores também afetam a temperatura da Raspberry Pi, onde podemos notar nos um e dois que as falhas das requisições acarretaram um pico de temperatura que passou dos 60 graus celsius.

Figura 18. Grafana UI Análise de Temperatura.

6. Conclusões

Este projeto teve como objetivo investigar a capacidade de uma placa de prototipagem (Raspberry pi Model B+), trabalhando como um servidor web de uma aplicação Web API, considerando o tempo de requisição, uso de memória RAM, uso de CPU e a temperatura. Foi construído uma aplicação web para receber requisições para *upload* e *download* de arquivos de mídia e, em seguida, configurado um ambiente para a realização dos testes. Os testes conduzidos nos mostraram que a Raspberry pi 3 Model B se comportou de uma maneira aceitável como um servidor web Kestrel para uma aplicação web. Podemos notar que o fato de utilizarmos para o teste um cartão de memória simples nos mostrou que a escolha dos recursos impacta a operação, pois os erros que podemos visualizar nos picos dos gráficos de recursos foram ocasionados pelo fato de a escrita neste cartão de memória ser lenta, gerando erros de *timeout.* É muito importante mencionar que as ferramentas utilizadas para a coleta e análise dos dados, Grafana e Prometheus, estavam instalados e executando na Raspberry Pi em conjunto com a API, para que pudéssemos acompanhar tanto dados da placa quanto das requisições, porém a execução dessas ferramentas também consumia recursos da Raspberry Pi, algo que em um ambiente de produção não seria necessário.

Desconsiderando os momentos em que os erros de requisições ocorriam, os recursos utilizados estavam dentro de limites aceitáveis para um servidor web e, mesmo nos momentos de falhas em requisições, os picos de recurso não chegaram aos seus limites.Ou seja, em nossos testes utilizamos uma placa que já está obsoleta e pode ser substituída por opções mais atuais e com muito mais recursos de hardware, o que otimizaria o uso da placa como um servidor web. O uso de uma memória física com escrita mais rápida diminuiria os problemas causados pela demora na escrita do cartão de memória, e a temperatura poderia ser otimizada com o uso de *coolers* para o processador.

Assim podemos analisar que a Raspberry pi, pode sim ser uma boa opção para aplicações com um volume de acessos controlado, e de menor complexidade reduzindo custos, e viabilizando possibilidades para a sociedade no desenvolvimento de ferramentas que não dispõem de grandes recursos para o seu desenvolvimento.

Referências

ANATEL, Agência nacional de Telecomunicações; Panorama Setorial de Telecomunicações. Disponível em: https://www.gov.br/anatel/pt-br/search?SearchableText=quantidade%20celulares%20no%20brasil. Acesso em 01 jun. 2023.

GAMER ANTIGO. Sobre a Raspberry Pi: Detalhes e Dicas. In: Gamer Antigo. [S. l.], 2024. Disponível em: https://gamerantigo.blogspot.com/2020/04/sobre-raspberry-pi-detalhes-e-dic as.html. Acesso em: 5 abr. 2024.

GILL, Sukhdeep; SINGH, Gurvinder; KAUR, Prabhjot.Raspberry Pi based smart home for deployment in the smart grid.

IBGE – INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA: Informações atualizadas sobre tecnologias da informação e comunicação. Rio de Janeiro: IBGE, 2021 Disponível em: https://educa.ibge.gov.br/jovens/materias-especiais/21581-informacoes-atualizadas-sobre-tecnologias-da-informacao-e-comunicacao.html. Acesso em 29 mar. 2023.

ISTIFANOS, Meron; TEKAHUN, Israel. Performance evaluation of Raspberry pi 3B as a web server. 2020. Trabalho de conclusão de curso (Graduação em Engenharia de Software) - Blekinge Institute of Technology, [S. l.], 2020. Disponível em: https://www.diva-portal.org/smash/get/diva2:1439759/FULLTEXT01.pdf. Acesso em: 5 abr. 2024.

LEE, Valentino; SCHNEIDER, Heather; SCHELL, Robbie. Aplicações móveis: Arquitetura, projeto e desenvolvimento. São Paulo: Pearson Education do Brasil, 2005. 2 p.

KING HOST. O que é uma aplicação web? King Host, 2023. Disponível em: https://king.host/blog/tecnologia/aplicacao-web/. Acesso em: 3 mar. 2025.

KINGSTON TECHNOLOGY. Escolha do armazenamento para Raspberry Pi. Disponível em: https://www.kingston.com/br/blog/personal-storage/choosing-storage-for-raspberry-pi. Acesso em: 14 mar. 2025.

OTANI, Mário; MACHADO, Waltair Vieira. A proposta de desenvolvimento de gestão da manutenção industrial na busca da excelência ou classe mundial. Revista Gestão Industrial. Vol.4, n.2, 2008.

RASPBERRY PI. Raspberry Pi: Foundation. In: Raspberry Pi Foundation: About Us. [S. l.], 2024. Disponível em: https://www.raspberrypi.org/about/. Acesso em: 5 abr. 2024.

SILBERSCHATZ, Abraham; GALVIN, Peter B.; GAGNE, Greg. Fundamentos de sistemas operacionais. 9. ed. Rio de Janeiro: LTC, 2015.

SILVA, Fransérgio Aparecido de Souza; PRADO, Ely Fernando do. Análise teórica sobre o desenvolvimento de aplicativos nativos, híbridos e webapps. Revista EduFatec: educação, tecnologia e gestão, Franca, v.2 n.1, p. 1-18 – jan. /jun. 2019. Disponível em: https://revistaedufatec.fatecfranca.edu.br/wp-content/uploads/2019/09/AN%C3%81LISE-TE%C3%93RICA-SOBRE-O-DESENVOLVIMENTO-D

E-APLICATIVOS-NATIVOS-H%C3%8DBRIDOS-E-WEBAPPS.pdf. Acesso em: 24 mai. 2023.

SISTEMA operativo. In: WIKIPÉDIA: a enciclopédia livre. São Francisco: Wikimedia Foundation, 7 mar. 2025. Disponível em: https://pt.wikipedia.org/wiki/Sistema_operativo?. Acesso em: 7 mar. 2025.

TANENBAUM, Andrew S.; BOS, Herbert. Sistemas operacionais modernos. 4. ed. São Paulo: Pearson Education do Brasil, 2016.

Documento Digitalizado Público

Anexo I - artigo - TCC

Assunto: Anexo I - artigo - TCC
Assinado por: Andre Constantino
Tipo do Documento: Relatório
Situação: Finalizado
Nível de Acesso: Público
Tipo do Conferência: Documento Digital

Documento assinado eletronicamente por:

- **Andre Constantino da Silva**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 15/03/2025 00:27:23.

Este documento foi armazenado no SUAP em 15/03/2025. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifsp.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 1967852
Código de Autenticação: 54caaa41d5