UNIVERSIDADE FEDERAL DE OURO PRETO

INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS

DEPARTAMENTO DE COMPUTAÇÃO

Alan Vasconcellos Erse

DESENVOLVIMENTO E ANALISE DO BACKEND DO PROJETO CUIDAIDOSO

Ouro Preto, MG
2021

Alan Vasconcellos Erse

DESENVOLVIMENTO E ANALISE DO BACKEND DO PROJETO CUIDAIDOSO

Monografia II apresentada ao Curso de Ciência da Computação da Universidade Federal de Ouro Preto como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Saul Emanuel Delabrida Silva
Coorientador: Rafael Ferreira Vitor

Ouro Preto, MG
2021

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E BIOLOGICAS
DEPARTAMENTO DE COMPUTACAO

FOLHA DE APROVAÇÃO

Alan Vasconcellos Erse

Desenvolvimento e analise do backend do projeto CuidaIdoso

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal
de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Ciência da Computação

Aprovada em 17 de Agosto de 2021.

Membros da banca

Saul Emanuel Delabrida Silva (Orientador) - Doutor - Universidade Federal de Ouro Preto
Rafael Ferreira Vitor (Coorientador) - Mestre - Deck Layer
Rodrigo Geraldo Ribeiro (Examinador) - Doutor - Universidade Federal de Ouro Preto
Vinicius Antonio de Oliveira Martins (Examinador) - Mestre - Universidade Federal de Ouro Preto

Saul Emanuel Delabrida Silva, Orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 17/08/2021.

Documento assinado eletronicamente por **Saul Emanuel Delabrida Silva**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 17/08/2021, às 14:09, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0206585** e o código CRC **DFF3D076**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35400-000
Telefone: 3135591692 - www.ufop.br

Agradecimentos

O desenvolvimento deste trabalho só foi possível graças a participação de pessoas fundamentais em minha vida. Gostaria de agradecer:

Aos meus familiares, Cláudio do Amaral Erse, Iná-Silvia Ferreira Vasconcellos, Silbene Ferreira Vasconcellos, Silverio de Carvalho Vasconcellos e Benedita Ferreira Vasconcellos por sempre me fornecer todo apoio, incentivo, amor e companheirismo ao longo de toda a vida.

Ao meu orientador Saul Emanuel Delabrida Silva, por confiar em minha capacidade, pela paciência e apoio para a realização deste trabalho.

Ao meu coorientador Rafael Ferreira, por toda paciência, apoio na escrita e revisão deste trabalho.

Aos meus amigos Jonata Souza, Luis Roberto, Gabriel Carvalho e Gustavo Presoti por nunca me deixarem desistir, por toda energia positiva, apoio e por estarem sempre ao meu lado para o que precisar.

Ao meu amigo Ricardo Manuel, por todo suporte durante o desenvolvimento e integração das aplicações do projeto.

A toda equipe do projeto CuidaIdoso, por sempre estarem se empenhando ao máximo para fazer o projeto crescer e prosperar.

A todos os meus amigos e familiares, que me acompanharam nesta jornada sempre torcendo pelo meu sucesso.

À Universidade Federal de Ouro Preto (UFOP), por todo suporte fornecido.

Resumo

O início da pandemia global derivada da doença COVID-19 marcou o ano de 2020 e 2021 no Brasil. Estudos apontam que cerca de 80% das pessoas hospitalizadas por COVID-19 possuem mais de 65 anos, exigindo que eles tenham mais cuidado e sejam bem informados. Porém, com o crescente número de *fake news* (informações falsas) circulando, se torna difícil discernir entre o que é verdade ou não. Desta forma, nasceu o projeto CuidaIdoso que tem como principal objetivo fornecer informações confiáveis sobre saúde. Para alcançar o maior número possível de pessoas, o projeto busca estar presente em diversas plataformas e redes sociais. De modo a evitar o retrabalho em implementar diferentes soluções tecnológicas para diferentes plataformas, foi implementado um *backend* que é capaz de fornecer os mesmos recursos e informações para diferentes tipos de interface. Este *backend* foi desenvolvido na linguagem JavaScript e conta com uma API REST que disponibiliza suas funcionalidades através de *endpoints*. Usuários em diferentes plataformas podem realizar chamadas nestes *endpoints* através do protocolo HTTP para obterem as informações ou recursos desejados. Com a implementação desta solução, é possível desenvolver as funcionalidades do sistema do CuidaIdoso em paralelo às suas diferentes interfaces. No atual momento, este *backend* já possui suas principais funcionalidades desenvolvidas e em funcionamento. Foram desenvolvidos testes automatizados para garantir a corretude do sistema. No mercado, encontramos diversas tecnologias e ferramentas para se construir um *backend*. Dessa forma, com o intuito de comparar o desempenho do *backend* desenvolvido, foi implementado um segundo *backend* na linguagem Golang. Para efeito comparativo apenas as funcionalidades CRUD de uma entidade específica do sistema foram implementadas para possibilitar a comparação com o primeiro *backend*. Com os resultados percebe-se que existem indícios de que o uso de diferentes tecnologias impactam diretamente na performance da aplicação. Nos testes realizados, foram comparados as seguintes métricas: custo de memória, porcentagem do uso da CPU e volume de dados trafegados na rede. Apesar de ambos os *backends* realizarem as mesmas atividades, e estarem sujeitos ao mesmo cenário de teste, suas métricas se diferenciaram.

Palavras-chave: *backend*, API REST, CuidaIdoso, idosos.

Abstract

The beginning of the global pandemic derived from the COVID-19 disease marked the year 2020 and 2021 in Brazil. Studies show that about 80% of people hospitalized for COVID-19 are over 65 years old, demanding that they be more careful and well-informed. However, with the increasing number of fake news circulating, it becomes difficult to discern between true and not. Thus, the CuidaIdoso project was born, whose main objective is to provide reliable information on health. To reach as many people as possible, the project seeks to be present on various platforms and social networks. In order to avoid the rework in implementing different technological solutions for different platforms, a backend that is able to provide the same resources and information for different types of interfaces was implemented. This backend was developed in JavaScript language and has a REST API that makes its functionalities available through endpoints. Users on different platforms can make calls on these endpoints through the HTTP protocol to obtain the desired information or resources. With the implementation of this solution, it is possible to develop the functionalities of the CuidaIdoso system in parallel with its different interfaces. At the moment, this backend already has its main functionalities developed and in operation. Automated tests were developed to guarantee the correctness of the system. In the market, we find several technologies and tools to build a backend. Thus, in order to compare the performance of the developed backend, a second backend was implemented in the Golang language. For comparison purposes, only the CRUD functionalities of a specific system entity were implemented to enable the comparison as the first backend. With the results, it is clear that there is evidence that the use of different technologies directly impact the performance of the application. In the tests carried out, the following metrics were compared: memory cost, percentage of CPU usage and volume of data transferred on the network. Although both backends perform the same activities, and are subject to the same test scenario, their metrics differed.

Keywords: backend, API REST, CuidaIdoso, elderly.

Lista de Ilustrações

Lista de Abreviaturas e Siglas

API	Interface de programação de aplicações / *Application Programming Interface*
BSON	JSON Binário / *Binary JSON*
CD	Entrega contínua / *Continuous delivery*
CI	Integração contínua / *Continuous integration*
DECOM	Departamento de Computação
HTTP	Protocolo de Transferência de Hipertexto / *Hypertext Transfer Protocol*
JS	JavaScript
JSON	Notação de objeto JavaScript / *JavaScript Object Notation*
REST	Transferência de Estado Representacional / *Representational State Transfer*
RoR	*Ruby on Rails*
RPC	Chamada de procedimento remoto / *Remote Procedure Call*
SOAP	Protocolo de acesso a objetos simples / *Simple Object Access Protocol*
SQL	Linguagem de consulta estruturada / *Structured Query Language*
UFOP	Universidade Federal de Ouro Preto
NTI	Núcleo de Tecnologia da Informação

Sumário

1 Introdução

O ano de 2020 foi marcado pelo início de uma pandemia mundial, desencadeada pelo vírus, SARS-CoV-2, causador da doença COVID-19. Tal pandemia possuiu o registro de mais de 106 milhões de casos confirmados e mais de 2 milhões de mortes pelo mundo [1]. Pessoas que apresentam comorbidades como obesidade, diabetes e problemas respiratórios, possuem uma chance maior de terem complicações mais severas ao contrair a doença (MUELLER; MCNAMARA; SINCLAIR, 2020). Além disso, os autores ainda citem que dentre as pessoas diagnosticadas com o vírus que precisaram ser hospitalizadas, cerca de 80% possuem idade acima de 65 anos. Por este motivo, idosos são considerados grupos de risco da doença e devem ter cuidados mais rigorosos para não serem infectados.

Atualmente, a disseminação de informações a respeito da doença é comumente realizada através da internet, em diferentes canais como as redes sociais e os portais de notícia. Estes são meios de comunicação pouco familiares para pessoas idosas, devido ao seu pouco contato com as tecnologias existentes. Além disso, nem sempre o conteúdo que é passado digitalmente é confiável, uma vez que existem informações falsas ou adulteradas, *fakenews*, sobre a COVID-19 em grande volume nas redes (NETO et al., 2020).

Neste cenário nasceu o projeto CuidaIdoso que tem o objetivo de disseminar informações confiáveis e verificadas por profissionais de saúde, sobre os cuidados que os idosos devem ter para se prevenir de doenças e obter uma melhor qualidade de vida por meio da tecnologia. De modo a obter um maior alcance, o projeto busca captar o maior número possível de idosos e seus cuidadores através de diferentes plataformas e tecnologias. Para alcançar este objetivo, foi criado um site em conjunto a perfis em redes sociais onde são disponibilizadas tais informações. Alem disto, no atual momento, está sendo desenvolvido um aplicativo para dispositivos móveis para alcançar um maior número de pessoas, além de fornecer diversas funcionalidades. Contamos com a colaboração de alunos e professores de diferentes cursos da UFOP (e.g. Farmácia, Direito) para garantir a confiabilidade e legalidade da divulgação das informações.

Desta forma, a motivação para o desenvolvimento deste projeto é a busca por uma solução capaz de integrar e fornecer os serviços e informações referentes ao CuidaIdoso de forma padronizada e independente. Esta solução irá disponibilizar os dados registrados para quaisquer novas plataformas que forem futuramente desenvolvidas, além de possibilitar o desenvolvimento em paralelo das interfaces do usuário e funcionalidades do sistema.

Pelo fato de existirem diversos tipos de tecnologias e *frameworks* que possibilitam a criação de *backends*. Cada tecnologia possui suas próprias particularidades, benefícios e malefícios. Este trabalho também se dispõe a investigar a seguinte questão de pesquisa: *Backends*

[1] <https://bit.ly/3sbYhKx> (acessado em 09/02/2020 as 16:22)

desenvolvidos em diferentes linguagens possuem diferentes desempenhos?

Para isso, este trabalho desenvolveu de um *backend* sustentável e escalável que forneça as funcionalidades necessárias para o funcionamento do sistema do CuidaIdoso, de forma simples e que, ainda, possa ser utilizada por diferentes interfaces. Além disso, este trabalho também desenvolveu de um segundo *backend*, seguindo as mesmas diretrizes do primeiro, para que sejam realizadas comparações entre o uso de duas diferentes tecnologias para a resolução de um mesmo problema.

1.1 Justificativa

Com o surgimento da COVID-19, ficou clara a importância de se manter bem informado. A todo momento, novas medidas de prevenção, cuidados básicos e informações sobre uma vacina são liberados nas redes, porém, nem sempre estas informações são confiáveis. Além disto, nem sempre existe a preocupação se a informação está chegando para a população. O público idoso, em geral, não possui contato com as redes sociais, e desta forma, não possuem acesso às informações, além de não conseguirem distinguir se uma informação é real ou falsa. Por se encontrarem em um grupo de risco, é de fundamental importância estarem sempre atualizados com informações corretas. Dessa forma, o principal objetivo do CuidaIdoso é fornecer de forma confiável a idosos e seus cuidadores, informações a respeito da pandemia, bem como cuidados com a saúde, em geral.

Devido ao interesse do projeto CuidaIdoso em atingir o maior número possível de pessoas, divulgando tais informações, é necessário adotar uma estratégia de múltiplos canais. Dessa forma, independente da plataforma, dispositivo ou interface utilizada pelo idoso, ou pela pessoa responsável por cuidar deste idoso, a informação estará acessível. Redes sociais, como Facebook, Instagram, por exemplo, disponibilizam diversas opções de acesso para o usuário, como smartphones, Smart TVs, navegadores web entre outros.

Para evitar retrabalho e viabilizar a presença do CuidaIdoso em diversos tipos de dispositivos, é necessário dividir sua aplicação entre *frontend* correspondente a interface, e em *backend*, responsável por toda regra de negócio. Com isso é possível garantir a simetria de informação, centralizar o desenvolvimento e evitar para cada novo dispositivo a aplicação necessite ser alterada. Além disso, ao se desenvolver um *backend* torna-se possível realizar o desenvolvimento de funcionalidades do sistema de forma independente e em paralelo à implementação das interfaces. Outro ponto positivo desta decisão, é o isolamento de problemas gerados pelo desenvolvimento. Caso ocorra algum problema no *backend*, nenhuma interface que esteja em desenvolvimento será afetada e vice-versa.

Devido a grande quantidade de tecnologias que podem ser utilizadas para se desenvolver um *backend*, este trabalho optou por desenvolver diferentes abordagens e realizar experimentos comparativos quanto a performance de cada um deles quanto ao custo de memória, tráfego de

rede, consumo de CPU. Dependendo dos recursos e infraestrutura disponíveis para implantação do sistema, é necessário observar qual tecnologia será adequada e que irá fornecer um melhor desempenho ao ser utilizada.

1.2 Objetivos

O objetivo principal deste trabalho é o desenvolvimento de um backend sustentável e escalável que forneça as funcionalidades necessárias para o funcionamento do sistema do CuidaIdoso, de forma simples e que, ainda, possa ser utilizada por diferentes interfaces.

Como objetivo específico, este trabalho busca:

- Desenvolver uma API REST que possibilite a comunicação entre diferentes plataformas.
- Fornecer as funcionalidades necessárias para as aplicações do CuidaIdoso.
- Criar testes automatizados para verificar a corretude das funcionalidades da aplicação.
- Comparar a aplicação de diferentes tecnologias para o desenvolvimento da API REST para o CuidaIdoso

1.3 Metodologia

Para dar suporte ao desenvolvimento do CuidaIdoso, este trabalho propôs a seguinte metodologia: primeiramente foi necessário dividir a aplicação em *backend* e *frontend*. Com isso, este trabalho pode centralizar a regra de negócio do CuidaIdoso em um único sistema, facilitando a gestão e evitando retrabalho. Para validar a melhor tecnologia para o desenvolvimento do *backend*, foram desenvolvidas duas APIs REST em duas tecnologias distintas. As funcionalidades básicas (CRUD) como criar, obter, atualizar e deletar, de uma entidade específica do sistema, foram implementadas em ambas as APIs para possibilitar a comparação de desempenho. As duas abordagens foram comparadas por meio de testes desenvolvidos na ferramenta JMeter, de modo a encontrar a de melhor performance.

Este trabalho alcançou os seguintes resultados:

- Criação de uma API REST, desenvolvida em JavaScript, que fornece todas as funcionalidades necessárias para as aplicações do CuidaIdoso.
- Criação de uma API REST, desenvolvida em Golang, que fornece algumas funcionalidades utilizadas nas aplicações do CuidaIdoso para fins de comparação entre duas tecnologias diferentes.
- Indícios de que diferentes linguagens e tecnologias impactam diretamente na performance de uma API.

1.4 Organização do Trabalho

Este trabalho foi organizado de acordo com a estrutura apresentada abaixo. Inicialmente é apresentado o projeto do CuidaIdoso e seus objetivos. Posteriormente é feita uma revisão bibliográfica apresentando softwares já existentes que se assemelham a proposta apresentada além de pesquisas realizadas nas mesmas áreas presentes na literatura. Alem disto, a definição e explicação de conceitos fundamentais para o entendimento das tecnologias e desenvolvimento da ferramenta. Em seguida, é detalhado os procedimentos, ferramentas e decisões tomadas para o desenvolvimento do trabalho. Por fim é realizada uma análise a respeito dos resultados obtidos, seguido da conclusão e trabalhos futuros. Atualmente o *deploy* do *backend* e do banco de dados são realizados em plataformas online com planos gratuitos. Por não possuirem custos, estes planos limitam a usabilidade e escalabilidade do sistema. A fim de evitar tais problemas, os testes comparativos entre as APIs

1.4.1 Estrutura da Monografia

Capítulo 1: Introdução.

Capítulo 2: O projeto CuidaIdoso

Capítulo 3: Revisão Bibliográfica.

Capítulo 4: Desenvolvimento.

Capítulo 5: Resultados.

Capítulo 6: Considerações Finais.

2 O projeto CuidaIdoso

O CuidaIdoso[1] é um projeto de extensão desenvolvido por um conjunto de laboratórios de uma universidade federal pública. Sua origem ocorreu em meio à pandemia da COVID-19 com o objetivo de auxiliar uma instituição focada na estadia de idosos na cidade. Esta instituição, que abriga vários idosos, classificados como grupo de risco, entrou em contato com um dos coordenadores do projeto apresentando questionamentos referentes aos cuidados e medidas necessárias para se evitar a contaminação e propagação desta nova doença.

Seus principais objetivos são:

- Orientar, de forma confiável, cuidadores e idosos à respeito da pandemia e cuidados com a saúde em geral.
- Promover interação voluntária entre quem precisa de ajuda e quem pode ajudar.
- Entreter os usuários das plataformas existentes.

A fim de alcançar estes objetivos, inicialmente foram criados perfis em redes sociais (Facebook e Instagram) juntamente a um website que possuíam o objetivo de fornecer informativos sobre cuidados com idosos e informações relevantes sobre a COVID-19 como cuidados necessários para evitar contaminação, sintomas, dicas de higiene pessoal, além de divulgar o projeto para os usuários destas redes.

Em seguida, foi proposto o desenvolvimento de um aplicativo para *smartphones* que fosse capaz de proporcionar um ambiente colaborativo entre pessoas. Tal escolha se baseou pela popularidade e facilidade de obtenção deste tipo de aparelho, tornando a aplicação mais acessível para a população. Sua ideia central é promover o a interação entre voluntários e idosos através da realização de atividades gravadas em forma de vídeos. Estes vídeos são fornecidos por voluntários cadastrados no aplicativo e possuem a finalidade de entreter os usuários da plataforma.

O projeto atualmente continua em crescimento, fornecendo informações através de posts em suas redes sociais, buscando aumentar seu alcance e seu número de usuários, alem do desenvolvimento de novas funcionalidades e melhoria no aplicativo.

[1] <http://cuidaidoso.net.br/> (acessado em 17/09/2020 às 23:01)

3 Revisão Bibliográfica

Na literatura existem diversos tipos de frameworks e linguagens de programação pensadas para o desenvolvimento de aplicações. De forma geral, estes recursos buscam auxiliar no desenvolvimento e integração de diferentes aplicações desenvolvidas em diversas plataformas, mantendo sua eficiência e escalabilidade.

A Seção 3.1 apresenta trabalhos relacionados que nortearam o desenvolvimento deste trabalho. Em seguida, a Seção 3.2 apresenta a fundamentação teórica para embasar o entendimento deste estudo.

3.1 Trabalhos Relacionados

Ao pesquisarmos sobre ferramentas, aplicativos e sites que possuem conteúdos relacionados a idosos, encontramos uma grande quantidade de informações e serviços fornecidos nesta temática. Dicas e cuidados para idosos, serviços como acompanhante, transporte, lazer, são alguns dos conteúdos encontrados. Além disto, existem na literatura estudos para melhorar e aumentar o relacionamento de idosos com as tecnologias existentes.

O Longevo [1] é um app auxiliar para profissionais de educação física e fisioterapeutas para realizar avaliações das condições físicas de seus pacientes idosos. Este app, conta com testes de força muscular, flexibilidade, equilíbrio, dentre vários outros. Após a realização das atividades, é gerado um relatório dos resultados e é atribuída uma avaliação ao idoso baseado com dados normativos da população.

Seniors é um aplicativo desenvolvido por EngenhoSit [2] que tem por objetivo auxiliar no acompanhamento/gerenciamento das atividades diárias relacionadas a sua saúde. Através dele, é possível armazenar informações referentes ao uso de medicamentos, histórico médico e afins. As funcionalidades contam com notificações para garantir a execução das atividades no horário adequado. Sua avaliação na Play Store conta com a nota máxima, porém possui poucas avaliações.

A Lotus TI Soluções de Sistemas criou um aplicativo nomeado Cuidar Idoso [3]. Este por sua vez, tem como principal função localizar profissionais e estabelecimentos relacionados à saúde baseado na localização inserida na busca. Dentre os profissionais encontram-se médicos, cuidadores, enfermeiros, fisioterapeutas entre outros. As avaliações feitas pelos usuários da aplicação relatam problemas de usabilidade e mau funcionamento de algumas funcionalidades.

[1] <https://apps.apple.com/br/app/longevo/id1462002352> (acessado em 08/04/2021 as 16:42)
[2] <https://play.google.com/store/apps/details?id=br.com.engenho.seniorsapp> (acessado em 08/04/2021 as 16:45)
[3] <https://play.google.com/store/apps/details?id=xdk.cuidaridoso.aplicativo.projeto> (acessado em 08/04/2021 as 16:50)

O Helpmy [4] é um aplicativo para encontrar cuidadores para idosos e familiares. A ferramenta já conta com uma extensa base de cuidadores cadastrados. O usuário ao buscar por um cuidador, deve informar as condições e características do paciente em questão para que seja feita uma recomendação adequada do profissional indicado. Além disto, o aplicativo conta com um sistema de avaliação para que auxilie os usuários no momento da escolha de um cuidador. Eles ainda contam com um blog, onde são postados conteúdos relacionados a idosos, cuidados com a saúde e curiosidades.

Cia Do Idoso [5] é um site onde são fornecidos serviços para a população idosa. Serviços relacionados a transporte, lazer, podologia e fisioterapia são alguns exemplos. Para solicitar um de seus serviços, basta entrar em contato através dos números de telefone disponibilizados no site, ou ainda via *WhatsApp* ou email. A página conta com publicações de notícias e conteúdos relacionados a idosos e serviços direcionados para este público. Para fazer parte do time de profissionais do site, basta preencher um formulário de inscrição presente no site e aguardar o retorno da equipe.

A página Portal do Idoso [6] possui uma grande variedade de conteúdos direcionados para o público idoso. Neste site, é possível encontrar textos sobre saúde, exercícios físicos, nutrição, informações sobre o estatuto do idoso, etc. Os conteúdos da página são organizados pelo geriatra Prof. Dr. José Eduardo Martinelli e profissionais do Instituto Martinelli de Geriatria e Gerontologia, o que garante a qualidade e confiabilidade das informações ali presentes.

Os autores de (LIU et al., 2020) apresentam um estudo a respeito do aumento da demanda por aplicativos de saúde para o público idoso. Estes ainda apresentam uma pesquisa, realizada no ano de 2014, por aplicativos relacionados à saúde e foram encontrados mais 5400 aplicações diferentes. Apesar de sua grande quantidade, estas aplicações apresentam funcionalidades simples e não demonstram uma grande preocupação no quesito usabilidade. Neste trabalho, é citado um método de pesquisa criado por Noriaki Kano, onde são classificadas e priorizadas as demandas dos usuários em aplicações. Este método é aplicado em um grupo de idosos presentes na China a fim de avaliar suas principais demandas em um aplicativo voltado para a saúde.

Em (HOUGH; KOBYLANSKI, 2009) é discutido o como a interação com a tecnologia pode trazer benefícios à qualidade de vida da população idosa. A construção de *smart houses* (casas inteligentes), o uso de plataformas digitais de comunicação e vendas e dispositivos para monitoramento a respeito da saúde do indivíduo, são exemplos do uso da tecnologia em benefício ao idoso. Além disto, ao se possuir acesso às redes, idosos podem buscar informações sobre saúde, bem-estar, realizar a compra de itens, obter notícias e informações atualizadas de todo o mundo. Neste artigo é dito que apesar de seus benefícios, a maior parte da população idosa não se relaciona com as tecnologias. É citado que em uma pesquisa realizada com a população

[4] <https://helpmy.com.br/> (acessado em 08/04/2021 as 16:54)
[5] <https://www.ciadoidoso.com.br/index.php> (acessado em 08/04/2021 as 16:54)
[6] <https://idosos.com.br/> (acessado em 08/04/2021 as 17:26)

dos Estados Unidos, apenas cerca de 4% dos usuários da internet no país são idosos e ainda que 56% dos idosos americanos não utilizam computadores. A proposta apresentada para solucionar este problema é a realização de campanhas de *marketing* em conjunto ao modelo de aceitação tecnológica (TAM). Este modelo é fundamentado por dois fatores imprescindíveis: mostrar ao idoso a utilidade da tecnologia e que seu uso é uma tarefa simples.

Na literatura, encontramos alguns trabalhos onde são feitos experimentos e comparações de diferentes tipos de linguagem e tecnologias. Alguns de seus principais pontos de avaliação são referentes ao tempo de execução, custo de memória e linhas de código necessárias para se resolver um mesmo tipo de problema.

O autor de (PRECHELT, 2000) apresenta em seu artigo diversas comparações entre 7 diferentes linguagens de programação. Em todas as linguagens, foi implementado o mesmo algoritmo para serem realizadas as comparações. Este código possui a finalidade de converter uma série de números em *strings*. No passado, para se enviar mensagens de texto pelo celular, era necessário escrever as mensagens a partir das teclas numéricas. Cada tecla possuia entre 2 a 3 letras. A fugura 3.1 apresenta a configuração de números e letras utilizada nos códigos. Nesta imagem podemos ver que o digito 1 pode representar 3 diferentes caracteres: "j", "n"ou "q". Caso o digito 1 apareça uma única vez, ele será convertido na letra "j". Caso apareça a sequência "11", ela será convertida para a letra "n", e por fim, com a sequência "111"obtemos a letra "q". Este comportamento se repete para todos os outros digitos numéricos com suas respectivas letras.

e	jnq	rwx	dsy	ft	am	civ	bku	lop	ghz
0	111	222	333	44	55	666	777	888	999

Figura 3.1 – Mapeamento de números e letras utilizado (PRECHELT, 2000)

Os principais pontos utilizados pelo autor para realizar a comparação dos códigos foram: tempo de execução (com e sem carregamento dos dados), consumo de memória, quantidade de linhas de código totais, tempo gasto para se desenvolver cada código e ainda uma relação de linhas de código por hora.

No artigo (JACKSON; CLYNCH, 2018) é apresentada uma análise sobre o impacto do uso de diferentes tecnologias em aplicações *serverless*. Estas aplicações possuem a característica de delegar toda a responsabilidade relacionada a infraestrutura, armazenamento e comunicação com a rede, para uma outra plataforma. AWS Lambda, fornacida pela Amazon e a Microsoft Azure Functions, são exemplos de plataformas *serverless*, onde um programador pode inserir o código de sua API, por exemplo, e a prórpia plataforma trata toda a parte de infraestrutura, alocação de recursos e *deploy*. As tecnologias escolhidas no artigo para serem feitas as comparações foram: NodeJS, Go, Python, Java e .NET Core 2. Cada tecnologia foi submetida a 2 tipos de teste: *cold-start* e *warm-start*.

O *cold-start* é um cenário onde a plataforma *serverless* irá criar um novo *container*, em seguida inserir o código criado pelo desenvolvedor, e por fim executar este código. Este processo demanda uma certa quantidade de tempo para ser executado que varia de acordo com a tecnologia utilizada. Uma vez criado o ambiente, caso este não seja utilizado, as plataformas "congelam"seu estado atual por um determinado período de tempo. O teste nomeado de *warm-start* ocorre em um cenário onde um *container* "congelado"é utilizado novamente. A plataforma possui um tempo de resposta mais rápido pois não é necessário refazer todos os passos do teste anterior para a criação do *container*. Assim como no primeiro, o tempo para reativar uma aplicação congelada também varia para cada tecnologia utilizada.

Cada plataforma *serverless* possui diferentes planos de assinaturas. Estes possuem diferentes características como número máximo de requisições, memória disponível, intervalo de tempo para que um *container* seja "congelado", entre outros. Todos os processos referentes a aplicação (e.g deploy, requisições, execuções de métodos) possuem um custo definido nestes pacotes. Além disto, o tempo de execução de uma aplicação também é contabilizado nos custos. Desta forma, é feita uma comparação no artigo à respeito da relação custo/tempo que cada aplicação demanda para ser inicializada do zero (cold-start). Aplicações que demandam de menos tempo para serem inicializadas possuem um melhor custo benefício pois ficam disponiveis para serem utilizadas mais cedo.

Com isso, percebe-se que existem diversas abordagens que buscam desenvolver sistemas voltados a idosos. Entretanto, estas não se destacam como uma solução largamente utilizada, como é possível constatar pelo número de downloads nas principais lojas de apps. Apesar disso, é possível analisar seus pontos fortes e direcionar o desenvolvimento deste trabalho. Artigos revisados também mostram metologias de priorização de funcionalidades e formas de melhor atender as necessidades dos idosos. Por fim, quanto ao aspecto do desempenho da tecnologia, foram revisados artigos que buscaram comparar a performance de diferentes linguagens frente a um mesmo problema.

3.2 Fundamentação Teórica

Nesta seção é feita uma descrição dos fundamentos teóricos necessários para o melhor entendimento deste trabalho. Na Seção 3.2.1 são apresentadas linguagens de programação utilizadas na atualidade para o desenvolvimento web, bem como seus principais frameworks e recursos. Em seguida, a Seção 3.2.2 apresenta as principais arquiteturas, técnicas e protocolos utilizados para o desenvolvimento de aplicações web. Definições e conceitos sobre bancos de dados relacionais e não relacionais serão apresentados na Seção 3.2.3. Por fim, são apresentadas as subdivisões de um projeto com suas respectivas responsabilidades, na Seção 3.2.7.

3.2.1 Linguagens e frameworks

Nesta seção teremos as seções 3.2.1.1 que aborda a linguagem Python e um de seus principais frameworks chamada Django, 3.2.1.2 onde são apresentados definições e conceitos sobre a linguagem Ruby e seu framework Ruby on Rails, em seguida a seção 3.2.1.3 expõe sobre a linguagem JavaScript, seu ambiente de execução de aplicações nomeado Node.js e um de seus frameworks chamado Express.js seguido da seção 3.2.1.4 é apresentada a linguagem Golang e seu pacote Gorilla Mux que realiza o tratamento de requisições HTTP. Por fim, ainda nesta seção, teremos a seção 3.2.1.5 onde é feita uma comparação entre as ferramentas seguida da justificativa de escolha do framework.

3.2.1.1 Python e Django

O Python é uma linguagem de programação interpretada criada no ano de 1990 [7]. Esta linguagem é bastante utilizada por desenvolvedores devido a sua sintaxe de simples entendimento, além de apresentar uma ampla documentação sobre seu funcionamento e funcionalidades. Além disto, a linguagem é open source (código aberto), o que facilita a modificação e contribuição de funcionalidades em códigos pela extensa comunidade de usuários e desenvolvedores.

Um dos frameworks mais populares para desenvolvimento web baseado em Python se chama Django [8]. Este framework possui uma grande quantidade de funcionalidades já implementadas, além de apresentar uma sintaxe simples e sucinta, otimizando o processo de desenvolvimento. O framework também da suporte à escalabilidade de aplicações, assim como no Python, entre diversos outros benefícios. Intagram, Mozilla e Pinterest são alguns exemplos de sites que utilizam o Django em suas aplicações (YUNG, 2018).

Em contrapartida, a ferramenta apresenta alguns problemas como a utilização de expressões regulares para especificação dos endereços das páginas, aplicações monolíticas (todos os serviços e funcionalidades encontram-se implementadas em um único local), etc. (YUNG, 2018).

3.2.1.2 Ruby e Ruby on Rails

A linguagem Ruby [9] foi desenvolvida por Yukihiro Matsumoto no ano de 1995 que buscava integrar aspectos positivos de outras linguagens como Perl, Ada, entre outros em uma nova linguagem. Esta é uma linguagem múlti plataforma, orientada a objetos, que conta com uma grande comunidade que desenvolve novas funcionalidades que podem ser importadas para dentro do seu projeto para serem utilizadas. Apesar disto, uma de sua desvantagem é o grande consumo de memória de suas aplicações (YUNG, 2018).

O Ruby on Rails [10], RoR, é um framework baseado em Ruby utilizado para o desenvolvi-

[7] <https://docs.python.org/2.0/ref/node92.html> (acessado em 15/06/2020 as 14:39)
[8] <https://www.djangoproject.com/> (acessado em 15/06/2020 as 14:07)
[9] <https://www.ruby-lang.org/pt/about/>(acessado em 15/06/2020 as 15:04)
[10] <https://rubyonrails.org/>(acessado em 15/06/2020 as 15:22)

mento de aplicações web. Além do fato do Ruby ser orientado a objetos, são aplicadas normas e convenções para a padronização de projetos, auxiliando na legibilidade de códigos desenvolvidos no RoR. Tanto a linguagem como o framework possuem uma extensa comunidade. Grandes empresas como GitHub, Twitch e Airbnb, desenvolvem suas soluções no Ruby on Rails.

Apesar de sua simplicidade em escrever códigos, a linguagem Ruby possui uma velocidade em tempo de execução que tende a desejar além de seu grande consumo de memória (YUNG, 2018).

3.2.1.3 JavaScript, Node.js e Express.js

O JavaScript, comumente abreviado como JS, é uma linguagem de programação com sintaxe baseada em linguagens como Java e C++. A semelhança de sintaxe foi selecionada com o objetivo de reduzir a curva de aprendizado do desenvolvedor ao migrar de uma dessas linguagens popularmente conhecidas. Códigos em JS são executados nos browsers dos usuários, realizando manipulações em paginas web (NETWORK, 2019). Alem disto, o JavaScript possui uma comunidade ativa e que sempre disponibiliza novos pacotes e recursos que agregam no desenvolvimento das aplicações que podem ser facilmente integrados a projetos em desenvolvimento. Apesar de seus benefícios, arquivos JavaScript só eram possíveis de serem executados em browsers e, desta forma, limitando que a linguagem fosse utilizada em outros ambientes. A fim de solucionar este problema, surgiu o Node.js.

Node.js é um ambiente para a criação e execução de aplicações escaláveis em um servidor com suas respectivas funcionalidades através de JavaScript [11]. Seu funcionamento se baseia na engine V8 desenvolvida pelo Google, que é uma ferramenta para acelerar a execução de arquivos JavaScript no Google Chrome.

Desta forma, o JavaScript tornou-se uma linguagem utilizável para a criação de aplicações completas, desde páginas web onde ocorre a interação com o usuário, até serviços e funcionalidades fornecidas pela aplicação. A fim de otimizar o desenvolvimento e evitar a repetição de códigos/funcionalidades, frameworks foram desenvolvidos contendo uma grande quantidade de ferramentas e métodos padronizados. Em meio a estes frameworks, um dos mais populares é chamado de Express.js.

O Express.js [12] é um framework baseado em JS que busca facilitar no desenvolvimento de aplicações web. Este framework, permite ao desenvolvedor utilizar requisições HTTP de forma simples, além de permitir a criação e utilização de middlewares. Estes middlewares permitem a manipulação das informações enviadas ou recebidas antes de serem enviadas para alguma função do sistema (YUNG, 2018).

Um dos principais problemas ao se utilizar Node.js em conjunto ao Express é a importação de funcionalidades desenvolvidas pela comunidade no projeto. Ao realizarmos uma importação

[11] <https://nodejs.org/en/about/>(acessado em 15/06/2020 as 17:01)
[12] <https://expressjs.com/pt-br/> (acessado em 15/06/2020 as 15:22)

de um pacote hipotético, é feito de forma automática a importação de outros pacotes que nem sempre são necessários para o funcionamento do pacote hipotético, gerando assim um consumo desnecessário de memória.

3.2.1.4 Golang e Gorilla Mux

Golang [13] é uma linguagem de programação desenvolvida por um time do Google de iniciativa *open source*. Um de seus principais benefícios é o fato de ser uma linguagem compilada, onde o código desenvolvido é tranformado para o formato de linguagem de máquina diretamente. Por ser *open source*, a linguagem do Golang (ou simplesmente Go), possui diversos plugins e IDE's que auxiliam durante o desenvolvimento, disponibilizada pela comunidade. Um exemplo de seu uso é o Docker, que foi desenvolvido nesta linguagem.

Outro benefício deste linguagem são as *Goroutines*. Estas rotinas permitem que aplicações consigam executar diversas tarefas simultaneamente. Possuem a mesma funcionalidade das tradicionais *threads*, mas as *Goroutines* são gerenciadas em tempo de execução pelo prórpio programa Go. Outro benefício, é que ao se escalonar uma nova *Goroutines*, o sistema não realiza interrupções, que é como ocorre na instanciação de *threads*. Este comportamento de interrupção no sistema, muitas vezes faz com que a tarefa que a *thread* executa, demore mais que o esperado para ser finalizada.

O Gorilla Mux [14] é um pacote, desenvolvido na linguagem Go, que fornece funcionalidadaes para realizar o tratamento de requisições HTTP. Ao se incorporar tais funcionalidades em um programa, o programador consegue definir qual funcionalidade desenvolvida no código será executada quando uma rota específica for solicitada. Seu funcionamento se assemelha ao Express.js.

3.2.1.5 Comparação e escolha do framework

No ambiente de desenvolvimento web existem inúmeras tecnologias, linguagens e ferramentas para a solução de problemas. As linguagens mais utilizadas na atualidade foram apresentadas nos tópicos acima em conjunto a seus principais frameworks de desenvolvimento. Cada framework é diretamente influenciado pela linguagem a qual foi desenvolvido, seja de forma positiva ou negativa.

Utilizando o Google Trends, ferramenta utilizada para avaliar o índice de pesquisas sobre termos e palavras em um período de tempo específico, foi realizada uma comparação entre as quatro linguagens apresentadas como mostra a figura 3.2

Através do gráfico, pode-se ver que no cenário atual, o JavaScript se destaca dentre a comunidade e possui uma demanda por busca de informações maior quando comparado as

[13] <https://golang.org/doc/> (acessado em 04/07/2021 as 17:25)
[14] <https://github.com/gorilla/mux> (acessado em 07/08/2021 as 16:25)

outras linguagens. Além disso, o Node.js apresenta uma grande quantidade de recursos gratuitos disponíveis para serem utilizados em projetos, possuindo uma comunidade extremamente ativa. O JavaScript permite a criação de uma aplicação web de forma completa utilizando a mesma linguagem, onde o desenvolvimento das funcionalidades da aplicação é feito de forma paralela à interface do usuário. Desta forma, o framework selecionado para o desenvolvimento deste trabalho foi o Express.js em conjunto ao Node.js para o primeiro *backend*. Já para o segundo, foi selecionada a linguagem Go, em conjunto ao Gorrila Mux pelo fato de possuirem o recurso das *Goroutines*, que permitem que suas aplicações consigam realizar tarefas de forma simultânea sem realizar interrupções.

Figura 3.2 – Comparação entre JavaScript, Ruby, Python e Go [15]

[15] <https://trends.google.com.br/> (acessado em 07/08/2021 as 16:11)

3.2.2 Arquiteturas e protocolos

Nesta seção teremos as seções 3.2.2.1 que contém uma breve explicação sobre os protocolos HTTP e HTTPS, 3.2.2.2 apresentando a definição e características de arquivos JSON, 3.2.2.3 onde são abordadas as definições de WebServices e APIs, e por fim 3.2.2.4 que contem os conceitos dos tipos de arquiteturas SOAP e REST.

3.2.2.1 HTTP e HTTPS

O Hypertext Transfer Protocol, conhecido como HTTP, é um protocolo utilizado na web para a realização de troca de dados e informações (FIELDING et al., 1999). Este protocolo é baseado na ideia de cliente-servidor, onde um usuário realiza uma requisição (*request*) sobre algum dado e este é fornecido pela aplicação através de uma resposta *(response)*. Cada resposta é acompanhada de um código para indicar o sucesso ou fracasso da solicitação feita pelo usuário. Através do HTTP é possível realizar o tráfego de arquivos de texto, imagens, vídeos, páginas web. Pelo fato de que os dados trafegados através deste protocolo serem emitidos de forma textual, surgem problemas relacionados à segurança da informação. Desta forma, usuários maliciosos com o auxilio de ferramentas de coleta de tráfego de dados, são capazes de roubar informações sigilosas como credenciais, senhas e derivados.

A fim de solucionar estes problemas, o protocolo HTTPS possui o mesmo funcionamento básico do HTTP tendo como principal diferença o ato de criptografar os dados antes de serem enviados. (NERCESSIAN, 2018) Além disto, este protocolo impede que intrusos modifiquem as comunicações entre cliente-servidor e capturem informações, garantindo integridade e segurança dos dados e seu meio de comunicação. (BASQUES, 2018)

3.2.2.2 JSON

O JSON [16], JavaScript Object Notation, criado no ano de 2000, e um formato de arquivo utilizado para se definir estruturas de dados em arquivos de texto. Sua forma de representar os dados se baseia em elementos compostos por chave-valor. Esta forma de representação se popularizou por possuir um menor volume de texto para a representação das informações, tornando-se mais legível para humanos e de fácil interpretação por máquinas. Este tipo de arquivo é bastante utilizado por aplicações web que necessitam enviar e receber informações ao longo de sua execução. Em arquivos JavaScript, o JSON é a representação de objetos criados e suas respectivas estruturas. A figura 3.3 exemplifica um arquivo com esta estrutura.

3.2.2.3 WebServices e APIs

O termo *WebService*, é denominado a softwares que possuem a finalidade de prover serviços através da web entre diferentes dispositivos (HAAS, 2004). Estes serviços são disponibi-

[16] <https://www.json.org/json-en.html> (acessado em 15/06/2020 as 12:25)

```
{
    "map": {
        "dimX":9,
        "dimY":9,
        "nodes": [
            { "posX": 1, "posY": 1, "estado":"alive" },
            { "posX": 1, "posY": 2, "estado":"alive" },
            { "posX": 2, "posY": 1, "estado":"alive" },
            { "posX": 3, "posY": 2, "estado":"alive" },
            { "posX": 3, "posY": 4, "estado":"alive" },
            { "posX": 5, "posY": 4, "estado":"alive" },
            { "posX": 5, "posY": 6, "estado":"alive" },
            { "posX": 6, "posY": 7, "estado":"alive" },
            { "posX": 7, "posY": 6, "estado":"alive" },
            { "posX": 7, "posY": 7, "estado":"alive" },
        ]
    }
}
```

Figura 3.3 – Exemplo de arquivo em formato JSON

lizados na web através de APIs (*Application Programming Interface*), interfaces que fornecem definições e protocolos para utilização e integração de seus serviços.

Tais APIs evitam que a implementação/código de suas funcionalidades sejam expostas a terceiros. Para a utilização da aplicação, usuários utilizam o browser para realizar requisições a APIs através de *endpoints*, endereços que identificam unicamente cada funcionalidade do sistema. A comunicação entre usuário e sistema ocorre através de requisições e respostas (GAZAROV, 2019). Após uma requisição feita, a API, baseada na regra de negócio implementada, realiza a manipulação das informações e as retorna a resposta para o usuário. A figura 3.4 exemplifica a estrutura de uma aplicação web.

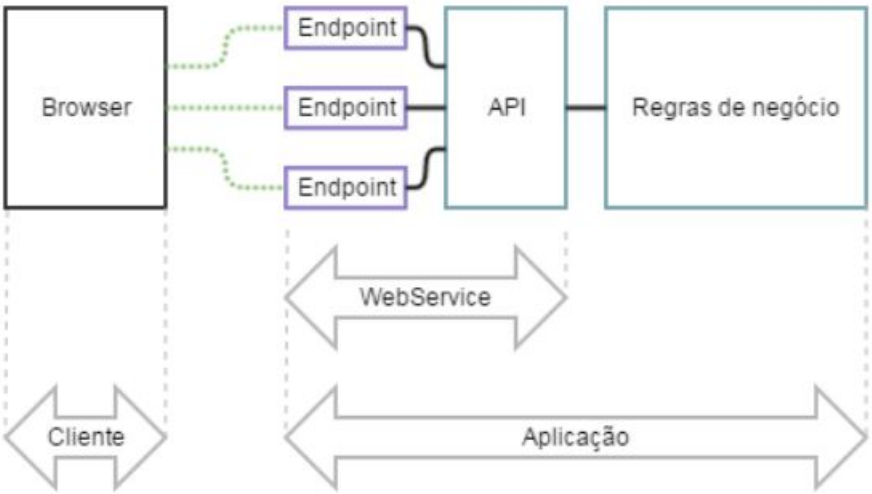

Figura 3.4 – Estrutura de uso de uma aplicação web [17]

[17] <https://medium.com/@gabrielpolo/o-que-%C3%A9-um-webservice-c5104d847a85> (acessado em

A título de exemplo, o Google possui uma página onde é possível encontrar documentações com exemplos de uso para utilização de seus serviços através de suas APIs como mostra a figura 3.5. Seus principais serviços, como Maps, Drive, Youtube, entre outros, podem ser utilizados através de APIs.

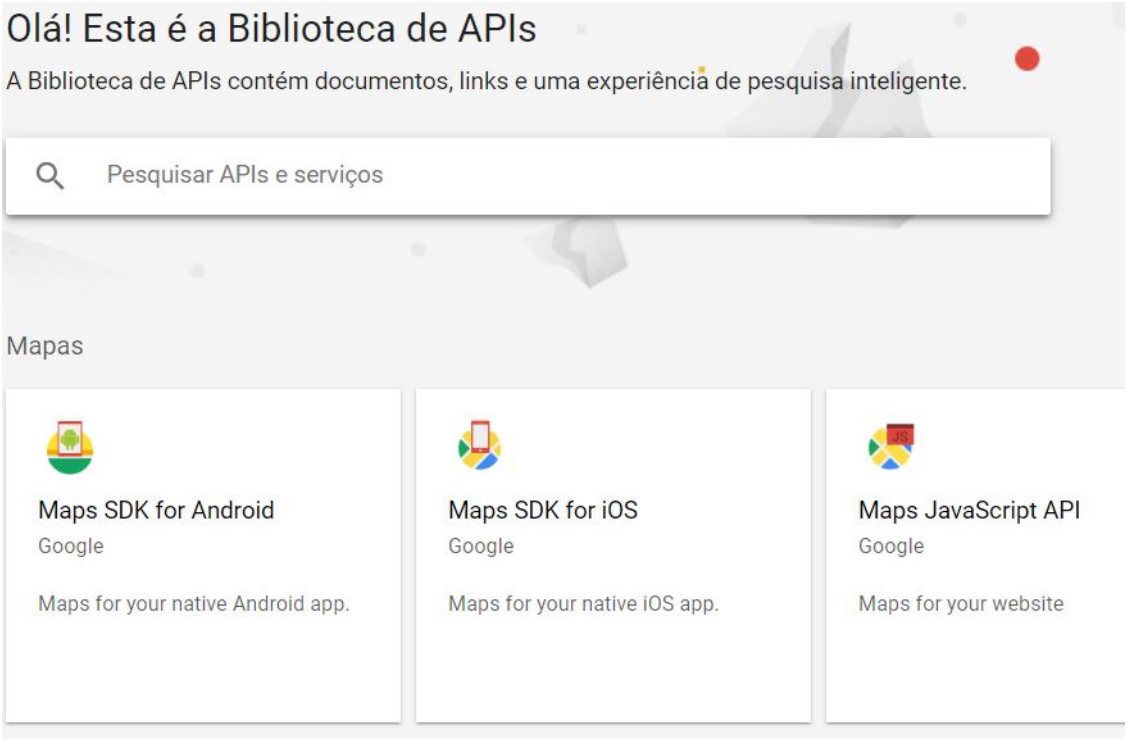

Figura 3.5 – Biblioteca de APIs do Google [18]

Cada API possui sua própria arquitetura que dita sua forma de lidar com os dados e realizar comunicações. As duas arquiteturas mais utilizadas são nomeadas de SOAP e REST.

3.2.2.4 SOAP e REST

SOAP e REST são arquiteturas que definem as particularidades e comportamentos de uma API. Ambas disponibilizam *endpoints* para o usuário realizar requisições na aplicação.

O SOAP (*Simple Object Access Protocol*) realiza a transmissão de informações a partir de arquivos XML(*Extensible Markup Language*), arquivos onde são definidas estruturas a partir da utilização de marcadores (MULLIGAN; GRAČANIN, 2009). A figura 3.6 exemplifica um arquivo XML onde cada marcador identifica unicamente cada informação.

As definições dos serviços que serão disponibilizados por APIs baseadas em SOAP são descritas através de arquivos WSDL. Estes arquivos, escritos em XML, possuem informações específicas sobre cada serviço, como o tipo de dado esperado, quais informações serão retornadas, tipos de parâmetros, entre outros.

12/04/2021 as 18:10)

[18] <https://console.cloud.google.com/apis/library> (acessado em 12/04/2021 as 20:10)

```
<SampleXML>
  <Colors>
    <Color1>White</Color1>
    <Color2>Blue</Color2>
    <Color3>Black</Color3>
    <Color4 Special="Light">Green</Color4>
    <Color5>Red</Color5>
  </Colors>
  <Fruits>
    <Fruits1>Apple</Fruits1>
    <Fruits2>Pineapple</Fruits2>
    <Fruits3>Grapes</Fruits3>
    <Fruits4>Melon</Fruits4>
  </Fruits>
</SampleXML>
```

Figura 3.6 – Exemplo de arquivo XML

O protocolo utilizado para a realização de requisições é chamado de RPC (*Remote Procedure Calls*). Este protocolo permite que sejam feitas requisições entre diferentes máquinas através de endereços especificados (MICROSYSTEMS, 1988). Porém, pelo fato de seus arquivos serem baseados em XML, a quantidade de rótulos e metainformações aumentam seu tamanho em memória além de dificultar no entendimento/processamento das respostas retornadas de uma requisição.

REST, acrônomo de *Representational State Transfer*, é uma arquitetura que se baseia no conjunto de regras do protocolo HTTP para a estruturação de interfaces de APIs. O protocolo HTTP, em uma de suas atualizações, sofreu mudanças a respeito de seus métodos para torná-los mais semânticos, facilitando o entendimento do propósito de cada requisição (PIRES, 2017).Dados seus benefícios, a arquitetura REST se baseou nestes princípios para a definição de seus métodos, dentre eles:

- GET: Utilizado para recuperação de dados.
- POST: Utilizado para envio de dados.
- PUT: Utilizado para atualização de dados.
- DELETE: Utilizado para remoção de dados.

Sistemas arquitetados em REST possuem a característica de desconsiderar os detalhes da implementação de suas funcionalidades e buscam realizar a interação entre diversos componentes (FIELDING, 2000). Diferentemente ao SOAP, a transferência de informações ocorre através de arquivos JSON. O REST foi a arquitetura escolhida para o desenvolvimento deste trabalho pelo fato de seus métodos serem mais semânticos e estruturados, além da manipulação e transferência da informação ser feita através da notação JSON.

3.2.3 Bancos de Dados

Em sistemas computacionais, é muito comum a demanda por armazenamento de dados. Estes dados por sua vez, podem possuir diferentes objetivos como representar um usuário, métricas ou informações sobre o sistema, registros de documentos ou atividades etc, ou ainda serem utilizados em funcionalidades do sistema. Na literatura existem diversos tipos de bancos de dados, com diferentes propósitos, mas que se baseiam em dois tipos tipos de modelo: Seção 3.2.3.1 e Seção 3.2.3.2.

3.2.3.1 Relacionais

Bancos de dados relacionais são aqueles baseados em entidades e relacionamentos. Este tipo de banco de dados é utilizado a muitos anos e sua forma estruturada de representação dos dados é utilizada até os dias atuais. Cada entidade é representada por uma tabela, onde cada linha (túpla) representa um registro de tal entidade. As colunas representam os atributos da entidade. Cada registro possui um atributo chamado de chave primária, que possui a função de identificar unicamente cada registro (DATE, 2004). A figura 3.7 apresenta uma representação de tabelas em bancos relacionais.

Fornecedor

CodFor	Nome	Tipofor	Cidade
F01	Antônio	Física	Belo Horizonte
F02	Expressos Ligeirinho	Jurídica	João Monlevade
F03	Maria	Física	Belo Horizonte
F04	Jorge	Física	Ipatinga
F05	Samara	Física	Rio de Janeiro

Produtos

Codpro	Descrição	Tipo	Valor
P01	Mouse	Suprimentos	20
P02	Cartucho Impressora	Suprimentos	60
P03	Teclado	Suprimentos	15
P04	Papel A4	Papelaria	8
P05	Caneta	Papelaria	3

Pedidos

Codfor	Codpro	qtde
F01	P01	3
F01	P02	1
F01	P03	2
F02	P01	4
F02	P05	1
F03	P02	2
F04	P04	2
F04	P05	5

Figura 3.7 – Exemplo de tabelas de bancos relacionais [19]

No mercado, grandes empresas como Oracle e Microsoft, desenvolveram seus próprios sistemas de bancos de dados relacionais que ainda são presentes em grande parte de sistemas computacionais. Nestes bancos relacionais, a execução de operações, como leitura, escrita, entre outras, é realizada através da linguagem SQL (*Structured Query Language*). Com ela, escreve-se uma sequencia de comandos que especificam quais operações serão executadas em determinadas estruturas.Com o passar do tempo, devido ao aumento da complexidade e volume dos dados,

[19] <https://sites.google.com/site/fkbancodedados1/_/rsrc/1470402642324/algebrarelacional/6.png> (acessado em 12/04/2021 as 22:10)

este tipo de modelo apresentou dificuldades em escalabilidade devido a sua rigidez estrutural de entidades e relacionamentos. Surge então como alternativa os bancos baseados em modelos não relacionais.

3.2.3.2 Não relacionais

Os modelos não relacionais, diferentemente dos relacionais, armazenam seus dados em pares no formato chave-valor, tornando mais flexível a estrutura dos dados. Bancos de dados não relacionais tornaram-se populares nos últimos tempos devido a sua facilidade de uso, velocidade e escalabilidade (LI; MANOHARAN, 2013). Além disto, grande parte destes bancos são iniciativas *open-source*, tornando mais transparente as implementações dos bancos.

Ao se basear no conceito de chave-valor, é possível criar diferentes tipos de estruturas para representação dos dados. Cada tipo de estrutura possui suas vantagens e desvantagens, podendo ser escolhida e aplicada no cenário adequado. Uma das estruturas mais populares e utilizadas é o banco baseado em documentos. Estes bancos representam estruturas a partir de arquivos JSON, XML ou BSON (arquivos JSON criptografados em binário) onde cada informação sobre a estrutura é descrita através de um par chave-valor. A figura 3.8 apresenta diferentes bancos de dados não relacionais, bem como seus melhores cenários de uso e sua forma de modelar os dados.

O Banco de dados MongoDB foi escolhido para o desenvolvimento. Este banco possui uma ferramenta de própria de busca, que otimiza a recuperação da informação dentro da base de dados. Além disto, pelo fato de sua modelagem de dados ser baseada em documentos, este banco lida de forma eficiente com mudanças estruturais dos dados armazenados.

3.2.4 Docker

Docker é uma plataforma que permite aos seus usuários realizar a separação entre a aplicação desenvolvida e sua infraestrutura. Para isto, todo o código da aplicação é empacotado em objetos nomeados *containers*. Ao se utilizar o Docker, é possivel transferir códigos presentes nestas estruturas para diferentes máquinas com facilidade. Um de seus maiores benefícios é sua portabilidade. Um *container* Docker pode ser aplicado em diferentes infraestruturas, desde uma máquina simples até máquinas virtuais em super-computadores. A figura 3.9 exemplifica um cenário onde uma infraestrutura qualquer em conjunto com seu sistema operacional, executa o Docker que fica resposável por administrar seus *containers*, onde cada *container* possui sua própria aplicação.

Para a criação de um *container*, é necessário criar uma cópia da aplicação original(nomeada de imagem), que possui o exato código fonte desenvolvido. Dentro da pasta do código fonte

[20] Adaptado de <https://dzone.com/articles/top-4-nosql-databases> (acessado em 12/04/2021 as 22:15)

[21] Adaptado de <https://fullcycle.com.br/docker-e-docker-composer-na-pratica-criando-ambiente-laravel/> (acessado em 22/06/2021 as 20:45)

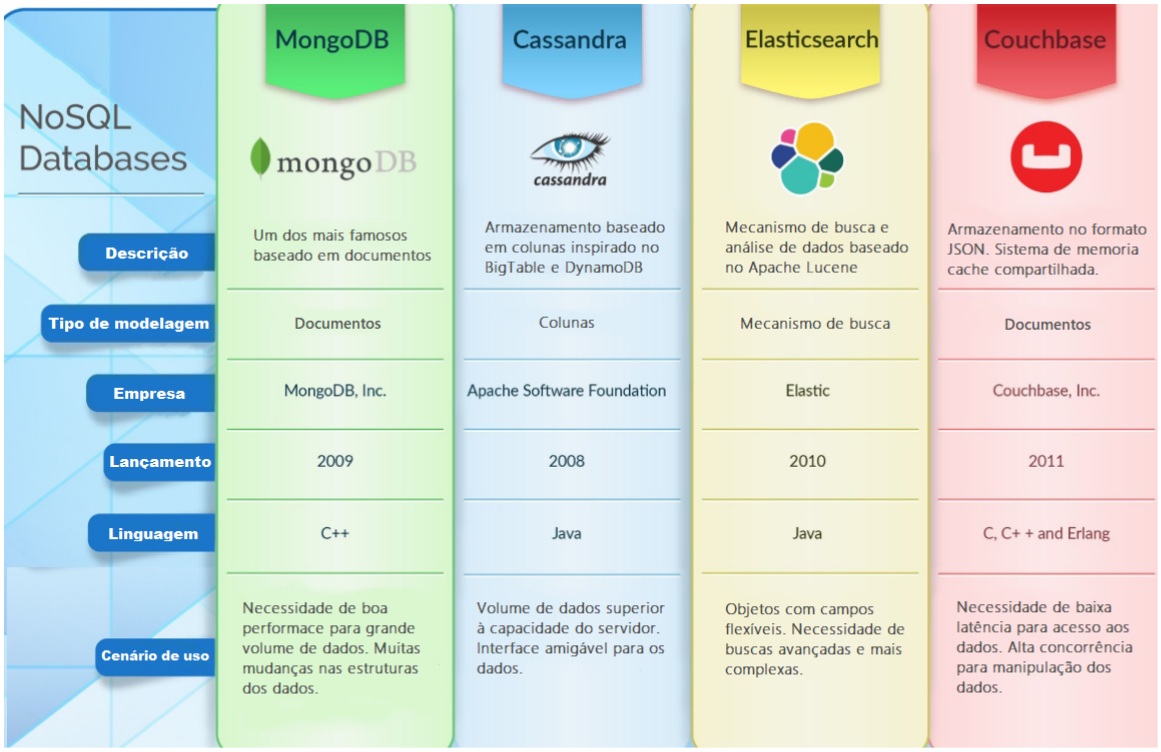

Figura 3.8 – Bancos não relacionais [20]

do projeto, deve-se criar um arquivo chamado Dockerfile, que possui as especificações para a criação da imagem, como comandos de inicialização da aplicação, configuração de variaveis de ambiente para execução e dependencias para serem baixadas. Em seguida, esta imagem é aplicada a um *container* que passa a ser uma versão executável da aplicação. O Docker possui uma plataforma, nomeada DockerHub, onde usuários podem disponibilizar suas imagens na rede. Imagens públicas podem ser baixadas e customizadas para utilização em sua aplicação.

3.2.5 Portainer

O Portainer [22] é uma ferramenta *open source* que realiza o monitoramento de *containers*. Com ela, é possível obter métricas à respeito da aplicação, como comportamentos, custo de memória, cpu e tráfego de rede, além de fornecer uma interface gráfica para o usuário. A ferramenta ainda apresenta o recurso de exibir os *logs* dos containers sendo compatível com as tecnologias Docker e Kubernetes.

[22] <https://www.portainer.io/> (acessado em 04/07/2021 as 16:25)

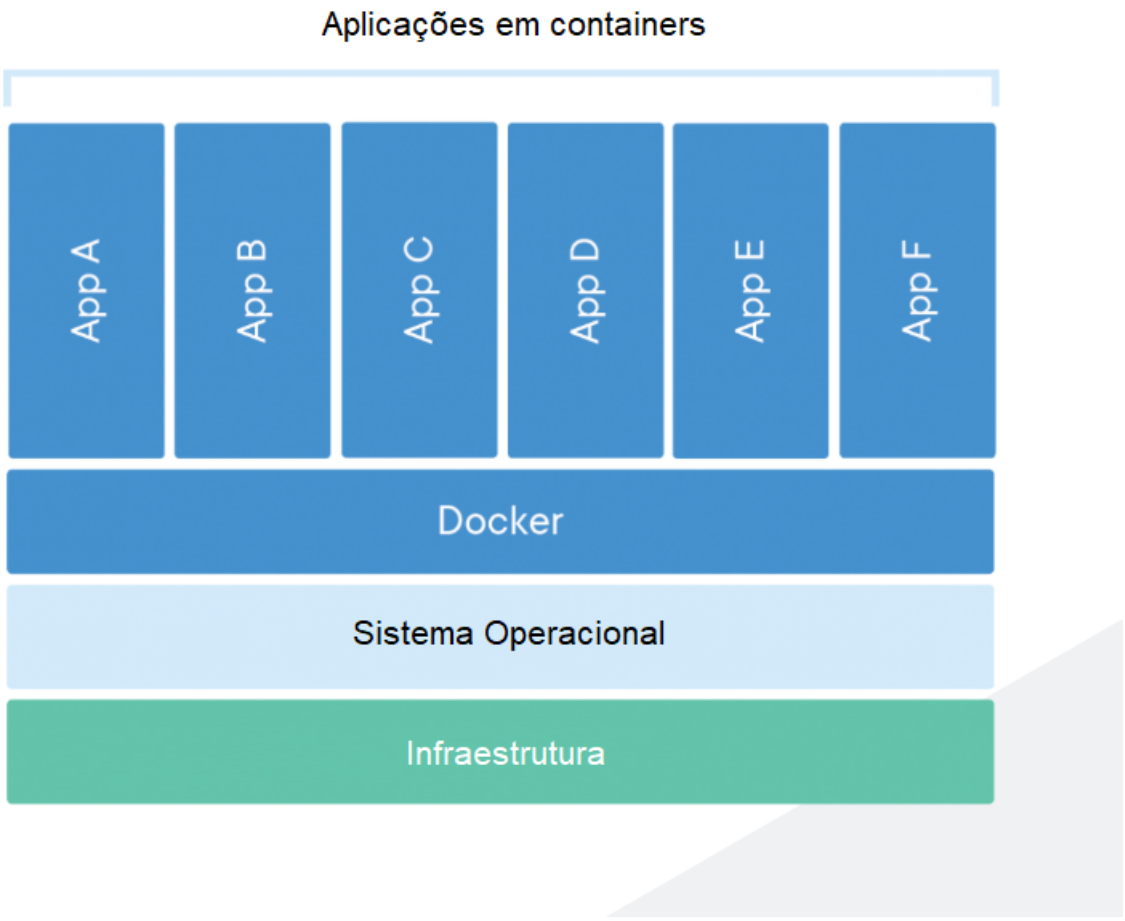

Figura 3.9 – Estrutura Docker [21]

3.2.6 Apache JMeter

O Apache JMeter, é um software desenvolvido pela empresa Apache, que realiza a simulação de comportamentos de usuário para auxiliar em testes de desempenho em softwares. Inicialmente, foi concebida para realizar testes em aplicações web. A ferrameta possibilita a simulação de diversos usuários através de *threads*, que realizam requisições HTTP conforme as configurações fornecidas.

3.2.7 Divisões de uma aplicação

Durante o desenvolvimento de aplicações Web, uma prática muito comum é a separação de *frontend* e *backend*. O textitfrontend é o responsável pela interface do usuário, lidando com tópicos como acessibilidade, experiência de uso do usuário, responsividade a diferentes tipos de aparelhos e tamanhos, entre outros.O *backend* lida com a lógica de negócio do sistema, definição e funcionamento da API e suas funcionalidades, formas de comunicação entre esta API e outros dispositivos, etc. Ao realizar a separação, torna-se possível o desenvolvimento em paralelo,

otimizando o avanço e amenizando a dependência entre ambas as partes.

O foco deste trabalho é o desenvolvimento do *backend* da aplicação do CuidaIdoso através da construção de uma API REST, que disponibiliza suas funcionalidades através de rotas baseadas no protocolo HTTP e ainda construir um segundo *backend*, em uma diferente tecnologia, para serem realizados testes comparativos. Para isto foi utilizado a linguagem JavaScript em conjunto ao framework Express.js e Node.js para o primeiro *backend*. O segundo foi desenvolvido em Golang e Gorilla Mux para serem realizadas as comparações entre ambas as APIs. Para o armazenamento dos dados, foi selecionado o MongoDB. Foi utilizado o Docker para criar diferentes *containers* para cada API, para que suas execuções não tenham impactos entre si. O JMeter foi aplicado para o desenvolvimento e execução do cenário de testes para ambas as aplicações e, por fim, foram criadas instâncias do Portainer para realizar o monitoramento e extração das informações de cada *container*.

4 Desenvolvimento

Neste capítulo é feita uma descrição, em detalhes, da metodologia para o desenvolvimento dos *backend* do CuidaIdoso. A Seção 4.1 irá apresentar a arquitetura do sistema, em seguida na 4.2 onde serão abordadas as entidades presentes no banco de dados, seguida da Seção 4.3 serão apresentadas as funcionalidades já desenvolvidas. Na Seção 4.4 serão abordadas as plataformas e estratégias utilizadas para a hospedagem e disponibilização do *backend* na web.

4.1 Arquitetura

A arquitetura do sistema do CuidaIdoso é baseada nas arquiteturas utilizadas em APIs REST, onde através de *endpoints*, são realizadas as operações e manipulações de dados entre a aplicação e o banco de dados. Cada *endpoint*, representa uma única funcionalidade do sistema e desta forma, dependem de dados específicos para a execução correta de suas operações. A figura 4.1 representa a estrutura do sistema do CuidaIdoso onde a API desenvolvida se comunica diretamente com o banco de dados, e fornece suas funcionalidades através de *endpoints* para as aplicações. Seu fluxo básico de funcionamento ocorre inicialmente com uma requisição realizada pelos usuários através de uma das plataformas do CuidaIdoso em um dos *endpoints* fornecidos pela API. Em seguida, a API busca as informações necessárias no banco de dados, realiza a manipulação destes dados baseados nas funções desenvolvidas, e retorna para o usuário o resultado final de sua requisição.

O Banco de dados MongoDB realiza a modelagem de seus dados baseados em documentos, e desta forma, cada entidade/estrutura possui sua respectiva coleção, onde cada documento representa um objeto diferente. Pelo fato do projeto estar em produção ativa, não serão especificados detalhes sobre as estruturas dos dados.

4.2 Entidades do sistema

Atualmente, as entidades presentes no banco de dados são: Usuários, Instituições, Ajudas, Atividades e Dicas. As estruturas de Usuário e Instituições representam usuários/instituições que realizaram seu cadastro na aplicação.

4.2.1 Usuários

Esta entidade armazena informações dos usuários da aplicação. São armazenadas informações como nome, email, foto de perfil. Estas informações são utilizadas para identificação do usuário por toda a aplicação.

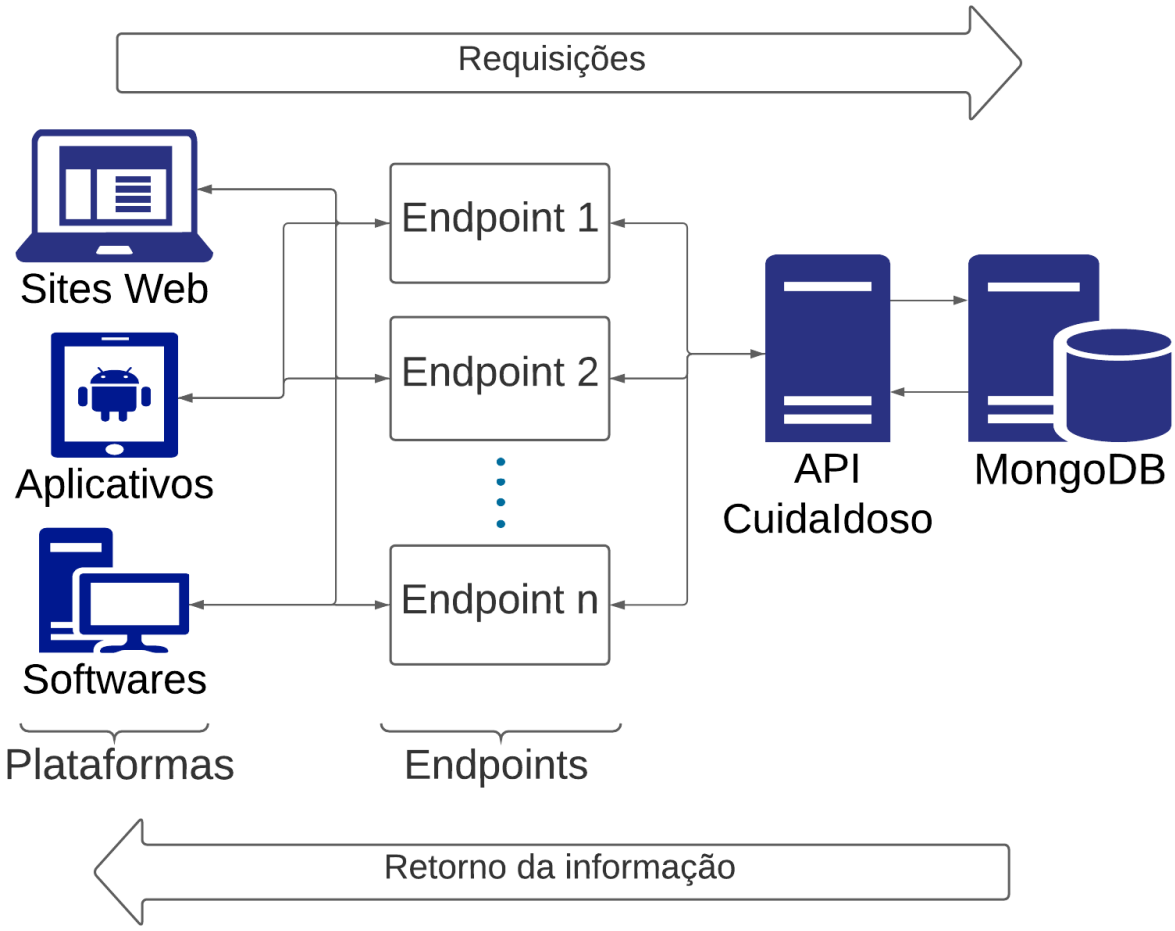

Figura 4.1 – Arquitetura do sistema CuidaIdoso

4.2.2 Atividades

Os usuários cadastrados na aplicação podem criar atividades na aplicação. Estas atividades consistem em vídeos de conteúdos diversos. Alguns exemplos de conteúdos são: contos de histórias, atividades físicas, aulas práticas sobre algum tema. Estes conteúdos possuem o intuito de entreter os usuários da plataforma em formato de vídeos. Após o usuário realizar o cadastro de uma atividade, esta é avaliada pela equipe do projeto a fim de garantir que o conteúdo não possua nada inapropriado para a plataforma. Uma vez sendo aprovada, o vídeo da atividade é colocado em um canal do Youtube específico do projeto onde são armazenadas todas as atividades e por fim são disponibilizados os links dentro da aplicação.

4.2.3 Instituições

Outra entidade presente na aplicação são as instituições. Estas "instituições"representam, por exemplo, casas de idosos, asilos e demais instituições que fornecem apoio a idosos. O cadastro de instituições somente pode ocorrer através de uma conta de usuário já cadastrada e devem ser fornecidas informações sobre a instituição como CNPJ, endereço e um documento oficial. As

instituições são compostas por 1 administrador e por seus participantes.

4.2.4 Ajudas

Dentro da aplicação do CuidaIdoso é permitido que instituições cadastradas realizem a criação de uma "Ajuda". Estas Ajudas, representam campanhas de arrecadação diversas (e.g. roupas, alimentos, dinheiro) que são exibidas para os usuários. Desta forma, instituições podem divulgar suas campanhas dentro da aplicação do CuidaIdoso.

4.2.5 Dicas

Assim como nos perfis das redes sociais, a equipe de conteúdo do CuidaIdoso realiza a criação de posts com conteúdos relevantes para os usuários. Estes conteúdos são nomeados como dicas no sistema, e seguem o mesmo padrão de publicações nas redes sociais. Sua estrutura possui uma imagem ilustrativa, seguida de uma descrição textual e ainda podendo possuir um link para acesso a algum conteúdo externo.

4.3 Funcionalidades

Para o *backend* feito em JavaScript, foram criadas as funcionalidades básicas (criar, ler, atualizar e deletar) para todas as entidades do sistema com suas respectivas particularidades. Pelo fato de existirem imagens relacionadas com algumas entidades, foram criadas funcionalidades para a realização da comunicação entre o *backend* e a API do Cloudinary.

Em seguida, a fim de tornar a aplicação mais segura, foram inseridos JWT's (*JSON Web Tokens*) para garantir a segurança durante a transmissão de informação. Estes tokens são utilizados para garantir que os usuários que realizam requisições no *backend* estão ou não autorizados a receber a resposta do método chamado. A geração deste token ocorre através de uma chave definida no servidor que é criptografado em conjunto a informações vindas da requisição. Ao se receber uma requisição, a aplicação extrai o token enviado pelo usuário e realiza a verificação. Além disto, foi implementado um sistema de login e senha para que os usuários possam acessar a aplicação e suas funcionalidades.

Durante o cadastro de instituições, é solicitado um documento oficial além de informações sobre seu responsável. Para realizar este armazenamento, criou-se um método para enviar este documento para o Google Drive. Para isto criou-se uma chave de autenticação OAuth2 (protocolo utilizado pelo Google para acesso a seus serviços), em conjunto ao método de comunicação com sua API.

Visando facilitar a administração do sistema, foram criados métodos específicos para administradores do CuidaIdoso. Durante a criação de ajudas/instituições, existe uma fase onde é realizada uma verificação dos dados/conteúdos informados. Dentro da própria aplicação, foram

criadas funcionalidades para aprovar e reprovar a criação destas entidades no sistema, podendo ainda informar para o usuário o motivo pelo qual não foi aceito em caso de reprovação.

Como o segundo *backend*, desenvolvido em Go, possui o intuito de apenas comparar o desempenho entre as linguagens, foram desenvolvidas apenas as funcionalidades básicas (criar, ler, atualizar e deletar) para a entidade de Ajudas.

4.4 Infraestrutura

Nesta seção serão abordadas as estratégias e plataformas utilizadas para a realização da implantação (*deploy*) do sistema do CuidaIdoso disponível na rede.

4.4.1 Deploy

Atualmente, o *backend* do CuidaIdoso se encontra hospedado em uma plataforma chamada Heroku [1]. Esta plataforma é responsável por realizar o *deploy* da aplicação, removendo a responsabilidade dos desenvolvedores quanto a serviços externos como infraestrutura e hospedagem.

4.4.2 Repositório de imagens

Para o armazenamento das imagens existentes nos conteudos, perfis e dicas do CuidaIdoso, é utilizada a plataforma do Cloudinary [2]. Esta plataforma realiza o armazenamento das imagens em nuvem. Além do armazenamento, a plataforma fornece através da sua API funcionalidades para a manipulação de imagens como transformações, otimizações, responsividade entre outros.

4.4.3 Repositório de documentos

Os documentos submetidos nos cadastros de instituições são armazenados no Google Drive. Esta plataforma possui uma API muito bem documentada, além de possuir diversas funcionalidades para gestão de arquivos.

4.4.4 Banco de dados

O banco de dados MongoDB do projeto se encontra em uma plataforma online nomeada MongoDB Atlas [3]. Esta plataforma, criada pela propria empresa MongoDB, armazena informações na nuvem através da instanciação de *clusters* em locais previamente configurados pelo desenvolvedor. Esta plataforma fornece segurança para dados sensíveis nas aplicações, escalabilidade, além de possuir uma interface simples para administração do banco.

[1] https://www.heroku.com/
[2] https://cloudinary.com/
[3] https://www.mongodb.com/cloud/atlas

4.4.5 Git e CI/CD

O código fonte do *backend* do CuidaIdoso se encontra em 3 diferentes *branches* (ramificações de códigos para controle de versão) na plataforma do Gitlab: desenvolvimento, homologação e produção. No *branch* de desenvolvimento é onde são implementadas e testadas as novas funcionalidades do sistema pela equipe de desenvolvimento. Já em homologação, encontra-se o ultimo código testado e aprovado pela equipe de desenvolvimento. Nesta *branch*, são realizados os testes pelos *stakeholders* da aplicação. Por fim, em produção é armazenado a ultima versão dó código já validada. Esta versão é mesma utilizada pelos usuarios finais.

A fim de agilizar o processo de desenvolvimento da aplicação, foram aplicadas técnicas de CI/CD (*continuous integration/continuous delivery*, em português, integração contínua/entrega contínua). Tais técnicas permitem que os códigos, após testados, sejam automaticamente incorporados nas outras *branches*. Dessa forma, ao final de teste, o código é migrado automaticamente para a proxima *branch*.

4.5 Limitações

Apesar de serem práticas e auxiliarem muito no *deploy* da aplicação, a utilização de plataformas (Atlas, Heroku e Cloudinary) em seus planos gratuitos restringem muito a usabilidade e escalabilidade do sistema.

Aplicações com plano gratuito no Heroku ao não serem utilizadas em um intervalo de 30 minutos entram em um estado de *sleep*. Ao entrar neste estado, o *backend* fica inativado até a primeira próxima requisição ser feita. Porem, quando esta requisição é realizada, existe um *delay* para a aplicação voltar ao seu funcionamento normal, demandando um tempo maior para enviar suas respostas.

O sistema de planos no Cloudinary se baseia em créditos. Cada imagem armazenada em seu sistema bem como manipulações e transporte, consomem uma parte destes créditos. Ao se utilizar um plano gratuito, o número de créditos é limitado e, dessa forma, impede a escalabilidade da aplicação.

O Atlas MongoDB permite que qualquer usuário consiga um plano gratuito para seu banco de dados. Porém, o administrador do banco não possui auxílio/assistência da equipe do Atlas em caso de falhas ou problemas no banco. Este suporte só é fornecido em planos pagos. Além disto, as regiões disponíveis para a alocação das máquinas virtuais e espaço de armazenamento também são limitadas, podendo influenciar diretamente no tempo de comunicação entre o banco de dados é o *backend* além de impactar na escalabilidade do sistema.

5 Resultados

Atualmente o *backend* do projeto CuidaIdoso possui as funcionalidades citadas em 4.3 já implementadas seguindo os padrões de uma API moderna. Além disto, durante o desenvolvimento, foram aplicadas técnicas e conceitos de arquiteturas REST para garantir uma melhor semântica e estruturação de seus métodos, tonando a inserção de novas funcionalidades uma atividade menos trabalhosa.

Pelo fato do projeto não estar vinculado a nenhum recurso financeiro, foi definido que, em um primeiro momento, seriam utilizados recursos gratuitos para disponibilizar o *backend* no ar. Tal escolha impacta diretamente na escalabilidade do sistema, uma vez que estes planos fornecem recursos limitados.

Na seção 5.1 são apresentados os resultados dos téstes unitários, seguida da seção 5.2 que apresenta os resultados dos testes de desempenho.

5.1 Testes unitários

A fim de garantir a corretude de todos os métodos criados, foram desenvolvidos testes automatizados baseados em propriedades utilizando o pacote Jest. Com este pacote, é possivel criar códigos de teste onde são descritos os cenários de aplicação de cada rota do sistema, especificando quais tipos de dados são esperados como entrada e quais são esperados como resultado. O teste baseado em propriedades possui o intuito de executar testes com dados aleatórios a fim de encontrar conta-exemplos durante a execução de algum método. As funcionalidade de cada entidade do sistema possuem um teste que segue esta abordagem. Tais dados aleatórios são gerados por um código implementado no sistema.

A figura 5.1 apresenta os resultados obtidos ao se executar os testes automatizados desenvolvidos. A primeira coluna da tabela indica o nome de cada arquivo presente em todo repositório. Na segunda, exibe a porcentagem de comandos que de fato foram executados durante os testes. A terceira coluna indica a porcentagem de *branches* que não foram executadas. Comandos condicionais (e.g if/else), geram diferentes caminhos de execução (*branch*) em um código. Dessa forma, essa coluna indica a porcentagem de caminhos que não foram executados. Na quarta coluna, é apresentada a porcentagem de funções que foram chamadas durante a execução dos testes. Na coluna seguinte, é exibida a porcentagem de linhas que não foram cobertas e por fim a ultima coluna apresenta extamente quais linhas não foram cobertas durante a execução do teste para cada arquivo.

Atualmente, os testes possuem uma cobertura de 84.56% de todas as linhas de código produzidas. Entidades como *Session*, Usuário e Instituição lidam com dados sensíveis do usuário,

além de realizarem a manipulação de imagens e arquivos, e devido a isto, seus testes ainda não foram finalizados, pois está sendo analisada a melhor forma para a realização de tais testes.

5.2 Testes de desempenho

Pelo fato de atualmente todo o *backend* depender do uso de planos gratuitos em plataformas online para *deploy*, o que impoe limitações para a aplicação, os testes de desempenho foram realizados utilizando 2 máquinas físicas. Na primeira máquina, foram instanciados dois *containers*. O primeiro contendo a imagen da API desenvolvida em JavaScript, uma imagem do MongoDB e uma imagem do Portainer. Já o segundo, possui a imagem da API desenvolvida em Golang, uma imagem do MongoDB e uma imagem do Portainer. Apesar de usarem as mesmas imagens, cada *container* possui uma instancia própria do Mongo e do Portainer, para que a execução dos métodos de suas APIs não gerem impacto entre si. Esta primeira máquina conta com um processador Intel Core i7-7700HQ de 2.80GHz, 16,0 GB de memória RAM, utilizando o sistema operacional Windows 10.

Na segunda máquina, foi instalado o JMeter e configurado o plano de testes. Neste plano são definidas quais as funcionalidades da API serão testadas, a quantidade de *threads* (usuários) simultâneos e o intervalo de tempo em que estes usuários irão realizar as chamadas na aplicação. As chamadas dos métodos ocorre de forma aleatória. Esta segunda máquina conta com um processador Intel Core i3 M370 de 2.40GHz, 4,0GB de memória RAM, utilizando o sistema operacional Windows 10.

Foram realizados 3 diferentes planos de teste, onde foi variada a quantidade de usuários simultâneos usando as aplicações. O número de usuários definidos foram 500, 1000 e 1200 respectivamente. Todos os testes possuem a duração de 2100 segundos.

A Figura 5.2 (a), 5.2 (b) e 5.2 (c), apresenta os custos de memória obtidos durante os teste do *container* que possui a API desenvolvida em Node. Na Figura 5.3 (a), 5.3 (b) e 5.3 (c), são exibidos os os custos de memória obtidos durante os testes realizados com a API desenvolvida em Golang. O eixo X nas figuras 5.2 e 5.3, representa o instante de tempo no decorrer do teste, enquanto o eixo Y representa a quantidade de memória utilizada a cada momento.

A Figura 5.4 (a), 5.4 (b) e 5.4 (c), apresenta os consumos de CPU obtido durante o teste do *container* que possui a API desenvolvida em Node. Na Figura 5.5 (a), 5.5 (b) e 5.5 (c), são exibidos os os consumos de CPU obtidos durante os testes realizados com a API desenvolvida em Golang. O eixo X nas figuras 5.4 e 5.5, representa o instante de tempo no decorrer do teste, enquanto o eixo Y representa a porcentagem da CPU utilizada a cada momento.

A Figura 5.6 (a), 5.6 (b) e 5.6 (c), apresentam os tráfegos de rede obtidos durante o teste do *container* que possui a API desenvolvida com Node. Na Figura 5.7 (a), 5.7 (b) e 5.7 (c), são exibidos os os tráfegos de rede obtidos durante os testes realizados com a API desenvolvida

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	84.25	46.4	85.38	84.56	
src	97.7	100	0	97.7	
app.ts	100	100	100	100	
routes.ts	97.22	100	0	97.22	20,54
src/config	50	100	0	50	
emailBody.ts	50	100	0	50	5
src/controllers	59.34	46.15	57.14	60.1	
AjudaController.ts	94.87	100	80	94.87	77,102
AtividadeController.ts	94.34	100	85.71	94.34	39,76,177
DicaController.ts	86.96	56.25	100	90.91	22,37,64,96
InstituicaoController.ts	64.49	52.38	68.75	64.96	...32,276,364-448
SessionController.ts	30.56	0	0	30.56	13,42-120
UsuarioController.ts	9.52	0	0	9.76	11-224
src/middlewares	100	100	100	100	
auth.ts	100	100	100	100	
multer.ts	100	100	100	100	
src/models	100	100	100	100	
Ajuda.ts	100	100	100	100	
Atividade.ts	100	100	100	100	
Dica.ts	100	100	100	100	
Instituicao.ts	100	100	100	100	
Usuario.ts	100	100	100	100	
src/tests/classes/Ajuda	99.1	50	100	99.08	
AjudaAjudaIdMissing.ts	100	100	100	100	
AjudaAjudaIdNotFound.ts	100	100	100	100	
AjudaAuthorizationInvalid.ts	100	100	100	100	
AjudaInstituicaoIdMissing.ts	100	100	100	100	
AjudaInstituicaoIdNotFound.ts	100	100	100	100	
AjudaTemplate.ts	94.44	50	100	94.44	14
AjudaUserEmailNotFound.ts	100	100	100	100	
AjudaValid.ts	100	100	100	100	
src/tests/classes/Atividade	98.44	50	96.15	98.41	
AtividadeAuthorizationInvalid.ts	100	100	100	100	
AtividadeIdMissing.ts	100	100	100	100	
AtividadeIdNotFound.ts	100	100	100	100	
AtividadeInfosMissing.ts	100	100	100	100	
AtividadeTemplate.ts	95	50	100	95	18
AtividadeUserNotFound.ts	100	100	100	100	
AtividadeValid.ts	100	100	100	100	
...idadeValidWithoutDataEvento.ts	92.31	100	50	92.31	19
src/tests/classes/Dica	98.28	50	100	98.28	
DicaAuthorizationInvalid.ts	100	100	100	100	
DicaFileMissing.ts	100	100	100	100	
DicaIdAndFileMissing.ts	100	100	100	100	
DicaIdMissing.ts	100	100	100	100	
DicaTemplate.ts	92.31	50	100	92.31	12
DicaValid.ts	100	100	100	100	
src/tests/classes/Instituicao	99.61	50	100	99.6	
InstituicaoAlreadyExists.ts	100	100	100	100	
...ituicaoAuthorizationInvalid.ts	100	100	100	100	
...DocumentoVerificadorMissing.ts	100	100	100	100	
InstituicaoIdAndPhotoMissing.ts	100	100	100	100	
InstituicaoIdMissing.ts	100	100	100	100	
InstituicaoIdNotFound.ts	100	100	100	100	
InstituicaoInfosError.ts	100	100	100	100	
InstituicaoInfosMissing.ts	100	100	100	100	
InstituicaoTemplate.ts	97.06	50	100	97.06	23
InstituicaoUserNotFound.ts	100	100	100	100	
InstituicaoValid.ts	100	100	100	100	

```
Test Suites: 4 passed, 4 total
Tests:       64 passed, 64 total
Snapshots:   0 total
Time:        31.561 s
Ran all test suites.
Done in 32.84s.
```

Figura 5.1 – Resultado dos testes unitários

em Golang. O eixo X nas figuras 5.2 e 5.3, representa o instante de tempo no decorrer do teste, enquanto o eixo Y representa o volume de dados total transmitido pela rede a cada momento.

Ao analisarmos as imagens referentes a quantidade de memória utilizada, a API desenvolvida em Golang apresenta resultados melhores, pois demandou de uma menor quantidade. Além disto, é possivel reparar que uma parte desta memória foi alocada na memória cache, o que otimiza o retorno da informação ao ser solicitada.

No quesito uso de CPU, a API desenvolvida em Golang se sobresaiu ao ser comparada com a API desenvolvida com Node. Durante a maior parte do tempo, seus valores oscilam abaixo dos 100% de uso, chegando a picos de cerca de 130% de uso da CPU. Enquanto isso, a API desenvolvida com Node apresentou valores um pouco mais altos e mais constantes próximo ao 100% de uso, chegando a picos de quase 150% de uso da CPU.

Por fim, ao se comparar o volume de dados transmitidos na rede entre as aplicações, a API feita com Node se destaca. Nesta aplicação, ao serem executados os cenários de testes executados, o volume de dados transmitidos e recebidos chegaram em valores próximos a 2GB. Ja na aplicação desenvolvida em Golang, estes valores sobem para aproximadamente, 10GB tanto para envio quanto para recebimento de dados.

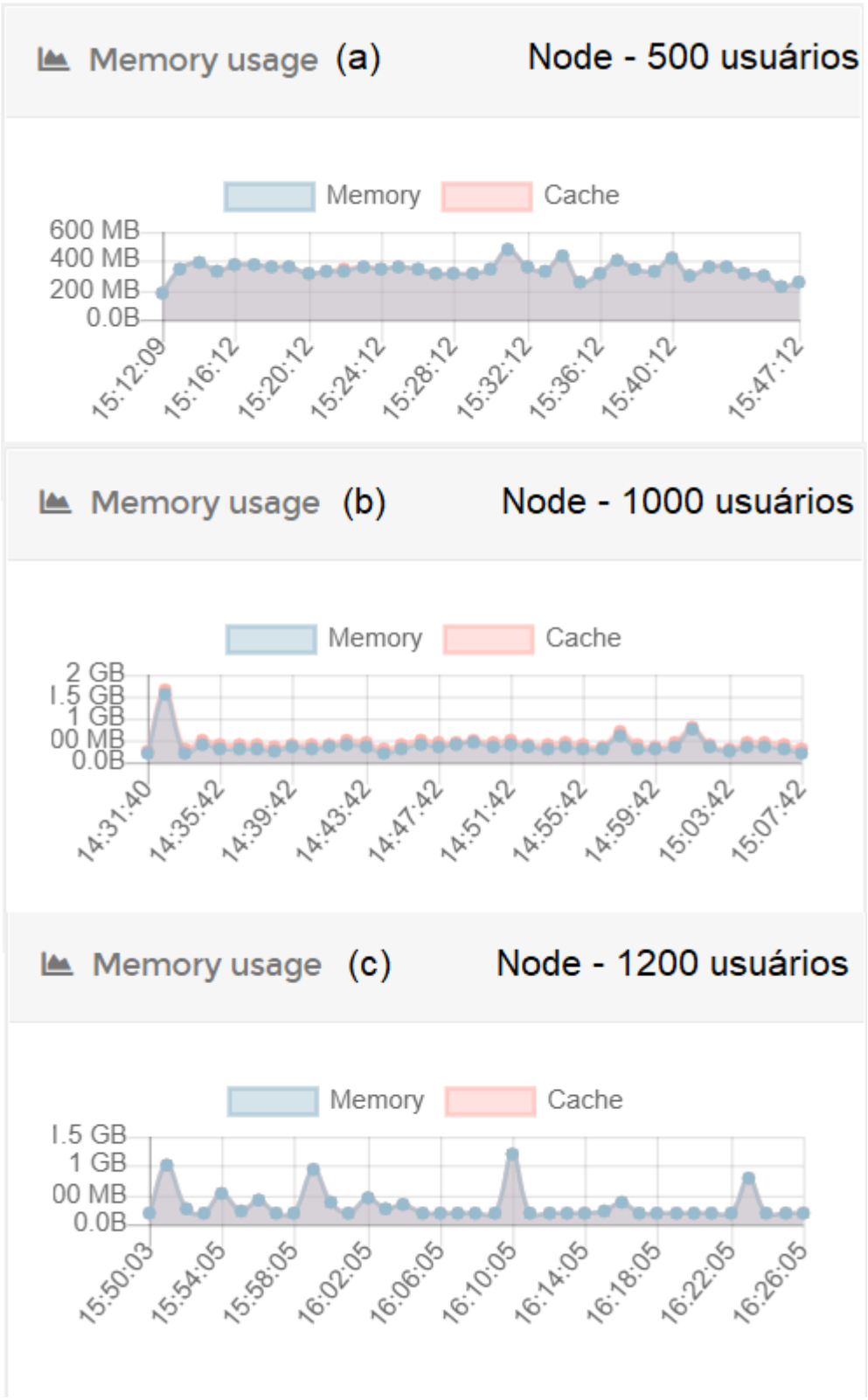

Figura 5.2 – Custo de memória testes em Node

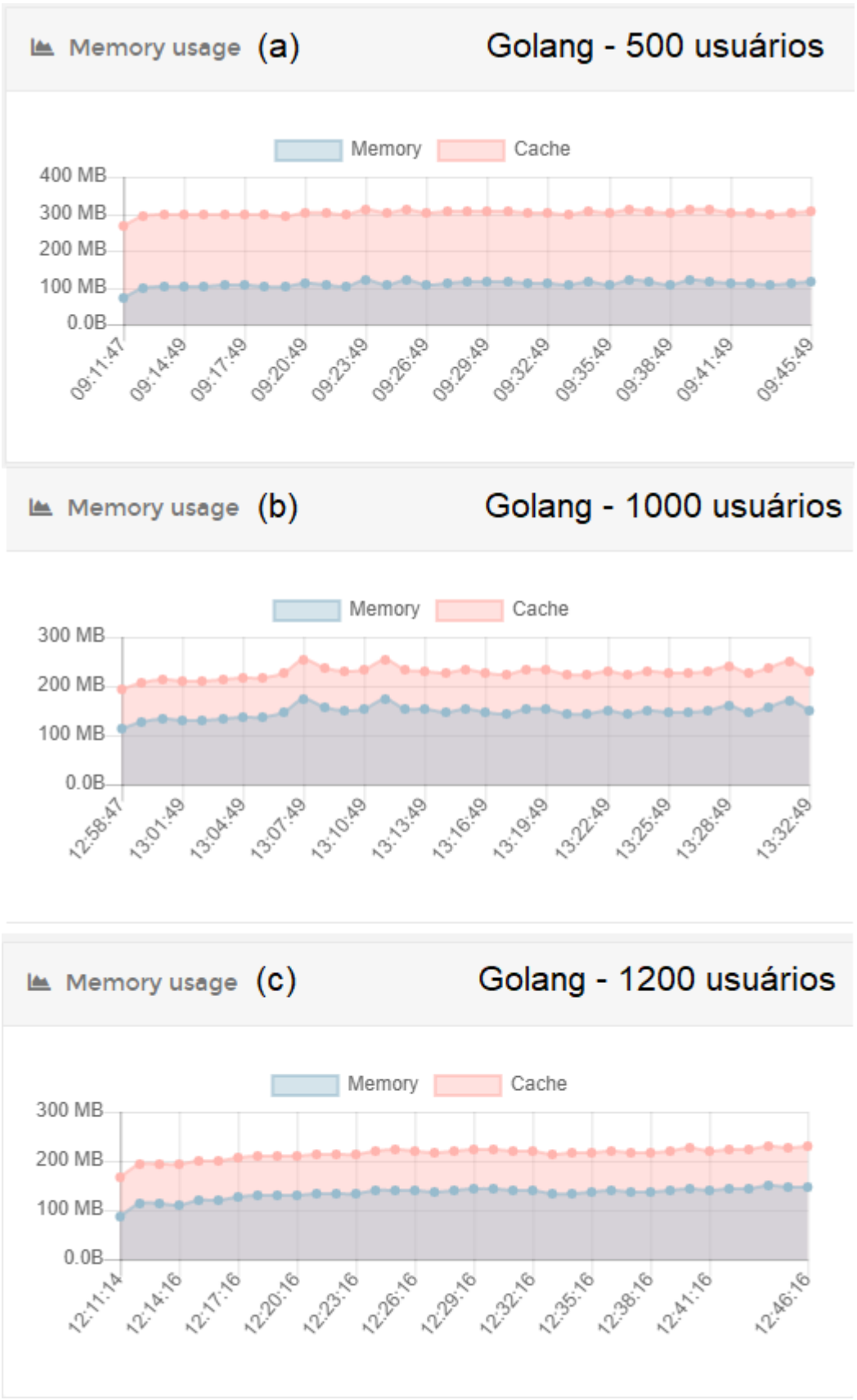

Figura 5.3 – Custo de memória testes em Golang

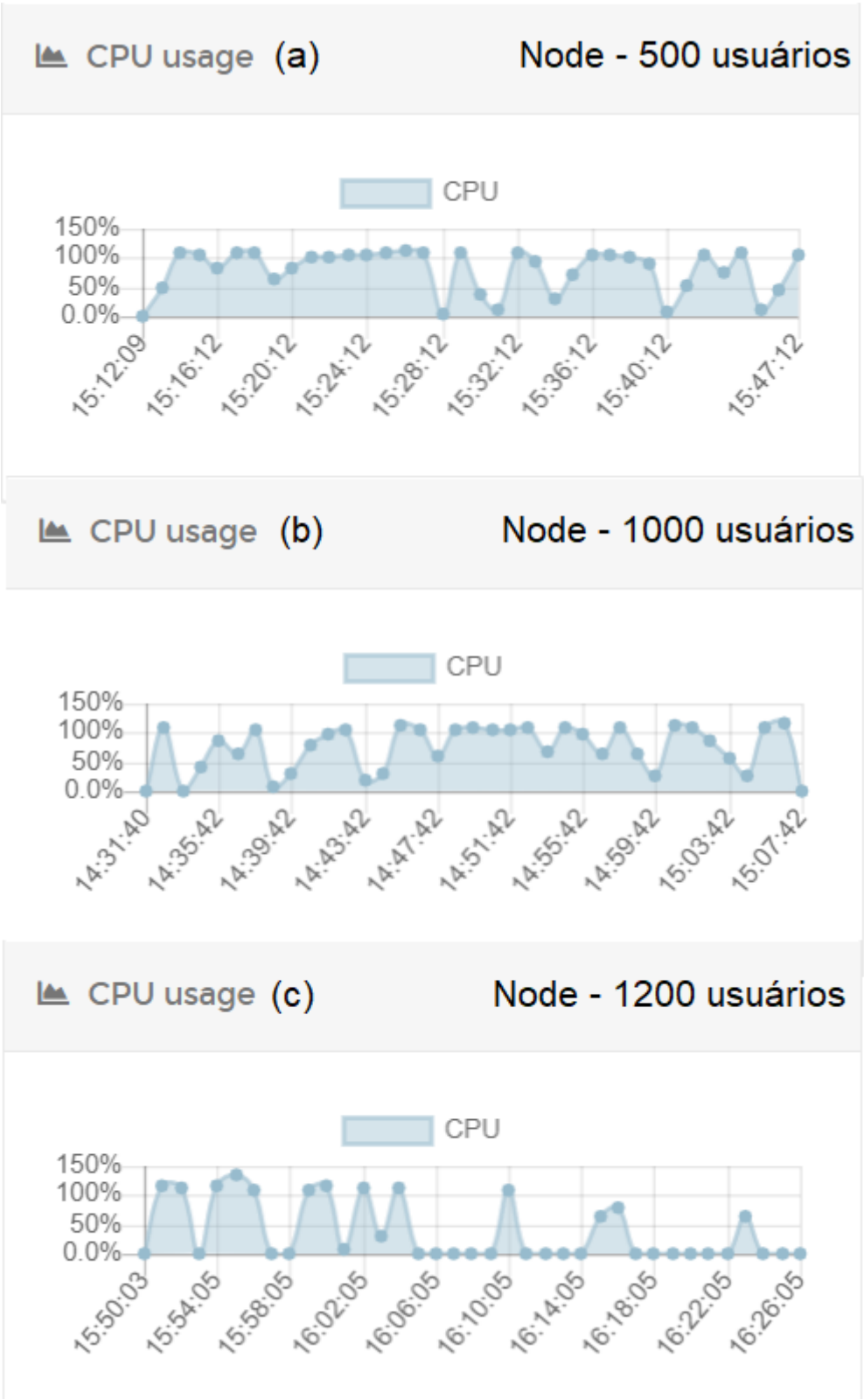

Figura 5.4 – Consumo de CPU testes em Node

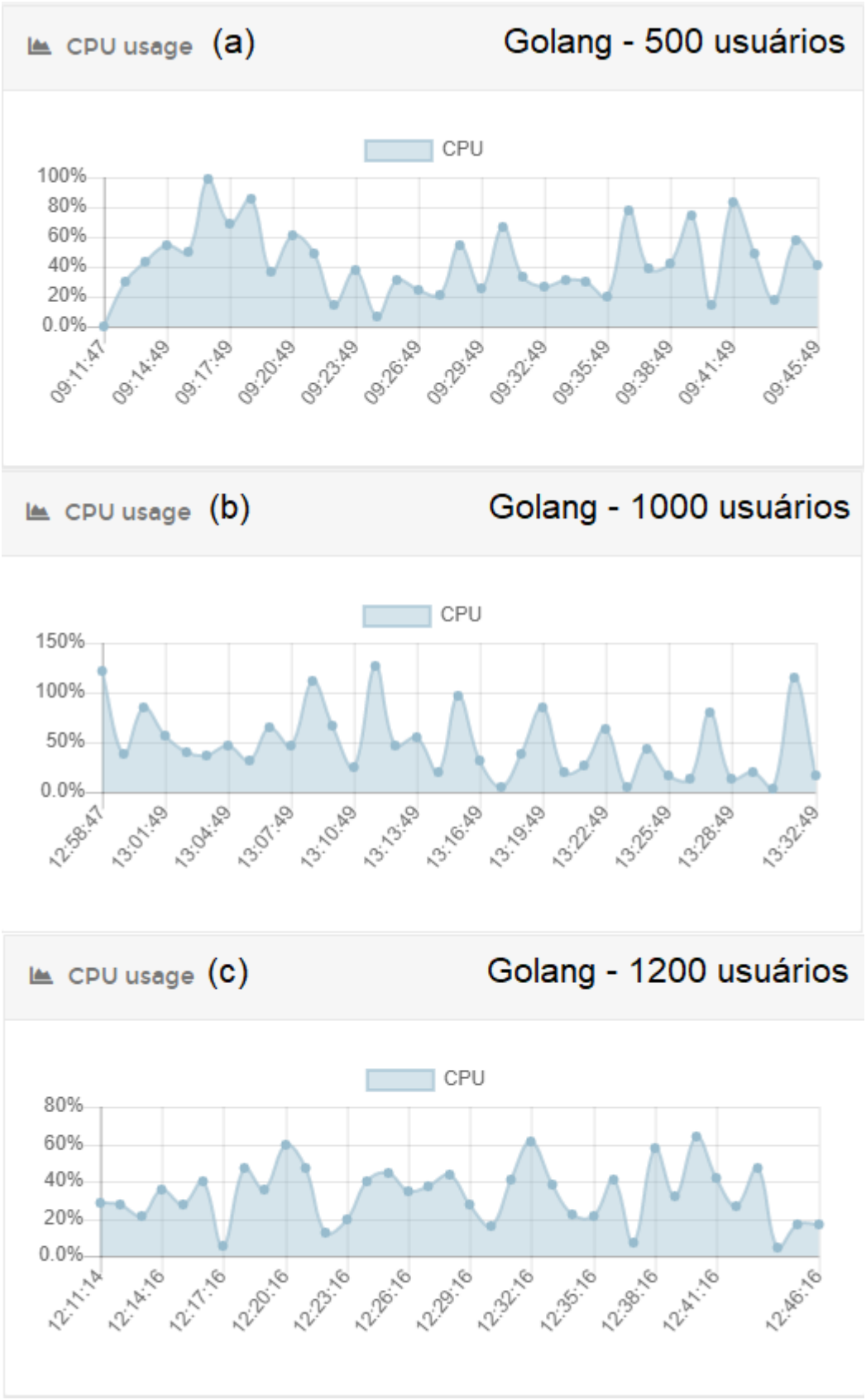

Figura 5.5 – Consumo de CPU testes em Golang

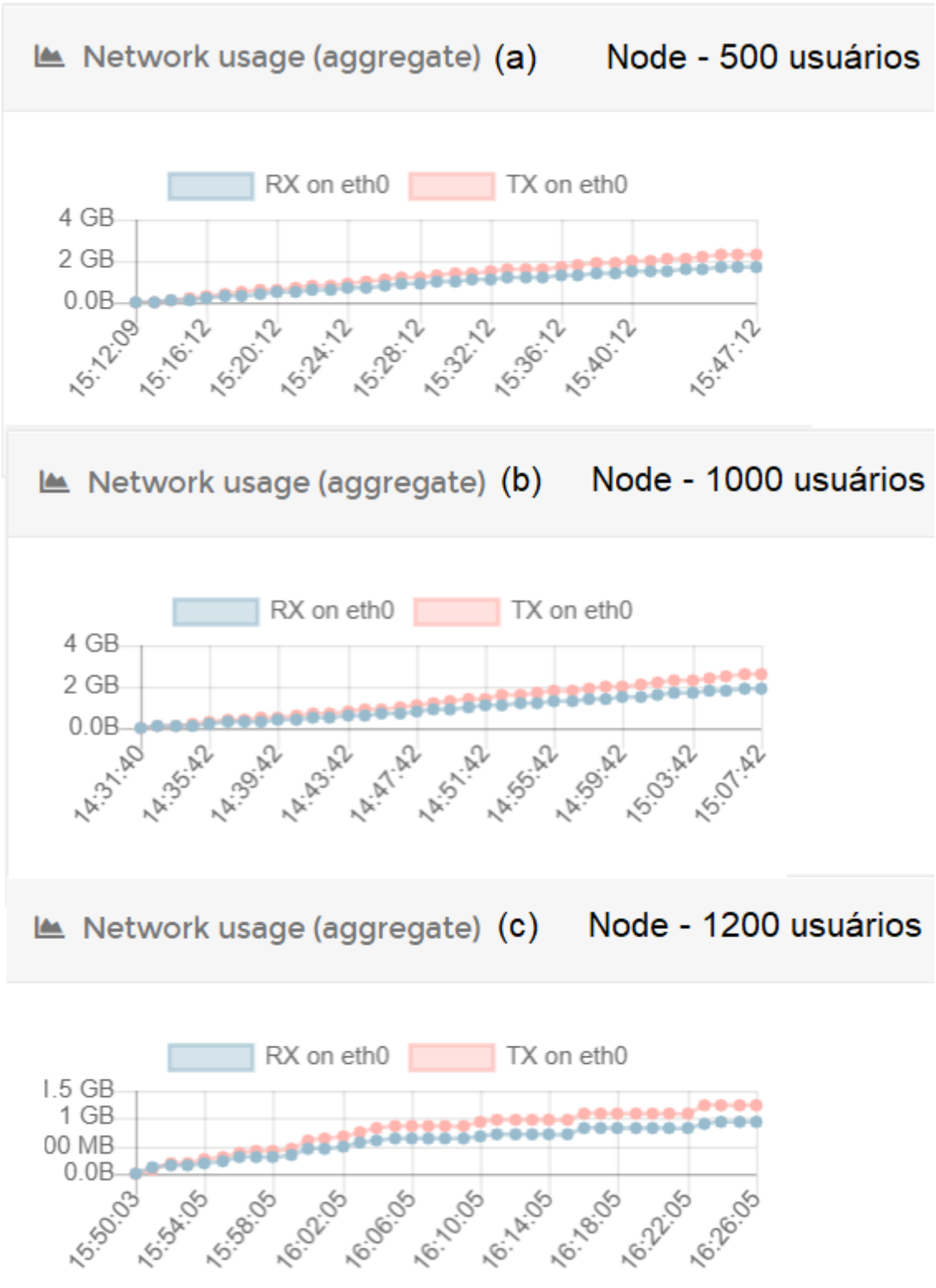

Figura 5.6 – Tráfego de rede testes em Node

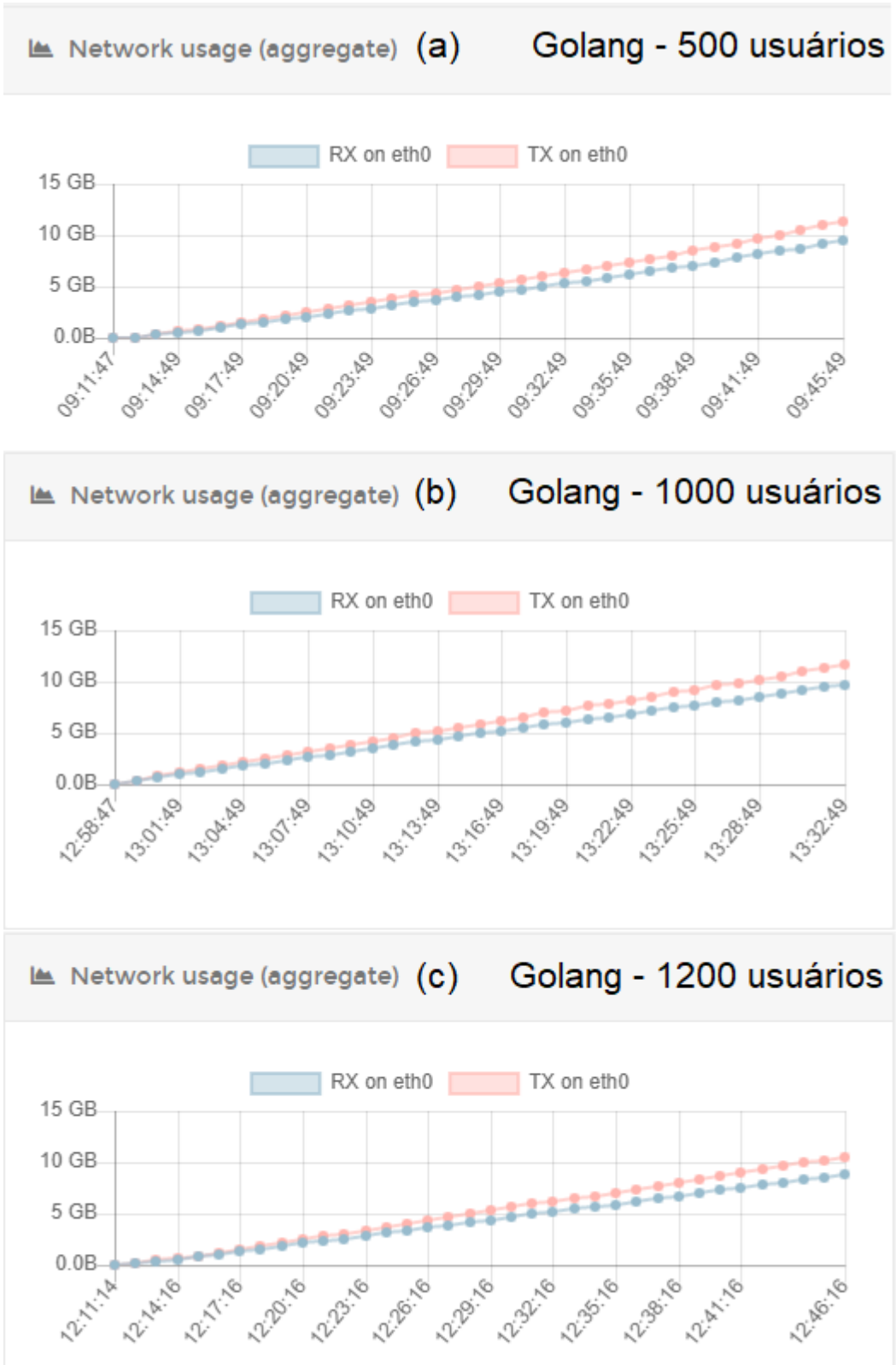

Figura 5.7 – Tráfego de rede testes em Golang

6 Considerações Finais

Neste capítulo são apresentadas as conclusões deste trabalho. Na seção 6.1 são apresentadas as conclusões, seguida da seção 6.2 que apresenta os trabalhos futuros.

6.1 Conclusão

A fim de fornecer as funcionalidades para o sistema do CuidaIdoso, foi implementada uma API REST que é capaz de se comunicar com diversos tipos de plataformas e interfaces, aumentando assim o número de plataformas que o projeto pode disponibilizar suas funcionalidades. Ao utilizarmos esta API, suas funcionalidades já implementadas podem ser utilizadas por todas as plataformas. Outro benefício gerado por tal escolha, é a paralelização dos desenvolvimentos, onde é possível desenvolver aplicativos, sites e afins, de forma isolada às funcionalidades do *backend*. Além disto, o resultado dos testes de desempenho apresentam indícios de que diferentes linguagens e *frameworks* apresentam desempenhos diferentes mesmo que sejam desenvolvidas aplicações para a resolução de um mesmo problema. A API desenvolvida em Golang, apresentou melhores resultados nos aspectos de consumo de memória e uso de CPU, porém, resulta em um maior fluxo de rede quando comparada a API desenvolvida com Node. Outro ponto a se levar em consideração é a popularidade das linguagens JavaScript e Golang, onde a primeira possui uma grande comunidade ativa e diversos recursos gratuitos para serem utilizados, enquanto a linguagem Golang encontra-se ainda em ascensão.

6.2 Trabalhos Futuros

O próximo passo para o *backend* do CuidaIdoso, será realizar a implantação do sistema na infraestrutura do Nucleo de Tecnologia da Informação da Universidade Federal de Ouro Preto (NTI). Nesta infraestrutura, é utilizada a tecnologia do Kubernetes, que funciona como um organizador de *containers*. Esta ferramenta é comumente utilizada para a criação de *clusters*. Desta forma, todos os recursos disponíveis das máquinas presentes no *cluster* são gerenciados e distribuidos entre as aplicações através do Kubernetes. Além do *backend*, será instanciada também uma versão do MongoDB. Para a realização destes *deploys*, é necessário desenvolver arquivos que ao serem executados, realizam a criação automatizada dos *containers*, especificando informações como: imagem a ser utilizada, especificações do *container* (e.g. portas a serem utilizadas), espaço de armazenamento necessário, etc. Estes arquivos ja se encontram em desenvolvimento.

Outro ponto importante, é a finalização dos testes automatizados para as entidades *Session*, Usuário e Instituição, para garantir que todas as entidades do sistema possuam uma automatização

de seus testes. Com isto, ao serem adicionadas novas funcionalidades na API, é possivel verificar se nenhuma outra entidade foi impactada com o novo recurso.

Atualmente, a API principal do CuidaIdoso está sendo desenvolvida em JavaScript em conjunto ao Node. Porém, baseado nos testes de desempenho apresentados em 5.2, é necessário realizar uma avaliação dos recursos fornecidos pela infraestrutura do NTI, para identificar qual a melhor tecnologia para ser utilizada, a fim de garantir que a API apresente o melhor desempenho possível.

Além disto, o projeto CuidaIdoso continua em crescimento e sempre buscando evoluir o seu sistema agregando cada vez mais recursos e funcionalidades. Desta forma, é necessário dar continuidade na manutenção e desenvolvimento das novas demandas que surgirem no projeto.

Referências

BASQUES, K. *Por que usar o HTTPS?* 2018. Disponível em: <https://developers.google.com/web/fundamentals/security/encrypt-in-transit/why-https?hl=pt-br>.

DATE, C. J. *Introdução a sistemas de bancos de dados*. [S.l.]: Elsevier Brasil, 2004.

FIELDING, R.; GETTYS, J.; MOGUL, J.; FRYSTYK, H.; MASINTER, L.; LEACH, P.; BERNERS-LEE, T. Hypertext transfer protocol–http/1.1. RFC 2616, june, 1999.

FIELDING, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. 2000. Disponível em: <https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm>.

GAZAROV, P. *What is an API? In English, please.* 2019. Disponível em: <https://www.freecodecamp.org/news/what-is-an-api-in-english-please-b880a3214a82/>.

HAAS, A. B. H. *Web Services Glossary*. 2004. Disponível em: <https://www.w3.org/TR/ws-gloss/#defs>.

HOUGH, M.; KOBYLANSKI, A. Increasing elder consumer interactions with information technology. *Journal of Consumer Marketing*, Emerald Group Publishing Limited, 2009.

JACKSON, D.; CLYNCH, G. An investigation of the impact of language runtime on the performance and cost of serverless functions. In: IEEE. *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*. [S.l.], 2018. p. 154–160.

LI, Y.; MANOHARAN, S. A performance comparison of sql and nosql databases. In: IEEE. *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*. [S.l.], 2013. p. 15–19.

LIU, D.; HSIEH, H.-H.; TSAY, W.-D.; KE, M.-H. Analysis on functional demand of mobile-health app for elders. In: IEEE. *2020 IEEE 2nd Eurasia Conference on Biomedical Engineering, Healthcare and Sustainability (ECBIOS)*. [S.l.], 2020. p. 122–125.

MICROSYSTEMS, I. S. *RPC: Remote Procedure Call Protocol Specification Version 2*. 1988. Disponível em: <https://tools.ietf.org/html/rfc1050>.

MUELLER, A. L.; MCNAMARA, M. S.; SINCLAIR, D. A. Why does covid-19 disproportionately affect older people? *Aging (Albany NY)*, Impact Journals, LLC, v. 12, n. 10, p. 9959, 2020.

MULLIGAN, G.; GRAČANIN, D. A comparison of soap and rest implementations of a service based interaction independence middleware framework. In: IEEE. *Proceedings of the 2009 Winter Simulation Conference (WSC)*. [S.l.], 2009. p. 1423–1432.

NERCESSIAN, R. *Qual é a diferença entre HTTP e HTTPS?* 2018. Disponível em: <https://www.alura.com.br/artigos/qual-e-diferenca-entre-http-e-https>.

NETO, M.; GOMES, T. de O.; PORTO, F. R.; RAFAEL, R. d. M. R.; FONSECA, M. H. S.; NASCIMENTO, J. Fake news no cenário da pandemia de covid-19. *Cogitare Enfermagem*, v. 25, 2020.

NETWORK, M. D. *Sobre JavaScript*. 2019. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/About_JavaScript>.

PIRES, J. *O que é API? REST e RESTful? Conheça as definições e diferenças!* 2017. Disponível em: <https://becode.com.br/o-que-e-api-rest-e-restful/>.

PRECHELT, L. An empirical comparison of seven programming languages. *Computer*, IEEE, v. 33, n. 10, p. 23–29, 2000.

YUNG, Z. *Python vs. Ruby vs. Node.js – Which Platform Is a Fit for Your Project?* 2018. Disponível em: <https://railsware.com/blog/python-vs-ruby-vs-node-js-which-platform-is-a-fit-for-your-project/>.