

UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO

NICOLAS VASCA GALINDO

Orientador: Vander Luis de Souza Freitas

Coorientador: Tiago Garcia de Senna Carneiro

**DESENVOLVIMENTO E ANÁLISE DE DUAS ARQUITETURAS DE
SOFTWARE PARA A OPERAÇÃO DE UM SERVIÇO EM NUVEM
DESTINADO À CONTRATAÇÃO DE PROFISSIONAIS DE LIMPEZA
DOMÉSTICA**

Ouro Preto, MG
2024

UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO

NICOLAS VASCA GALINDO

**DESENVOLVIMENTO E ANÁLISE DE DUAS ARQUITETURAS DE SOFTWARE
PARA A OPERAÇÃO DE UM SERVIÇO EM NUVEM DESTINADO À CONTRATAÇÃO
DE PROFISSIONAIS DE LIMPEZA DOMÉSTICA**

Monografia 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: Vander Luis de Souza Freitas

Coorientador: Tiago Garcia de Senna Carneiro

Ouro Preto, MG
2024

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

G158d Galindo, Nicolas Vasca.

Desenvolvimento e análise de duas arquiteturas de software para a operação de um serviço em nuvem destinado à contratação de profissionais de limpeza doméstica. [manuscrito] / Nicolas Vasca Galindo. - 2024.
90 f.

Orientador: Prof. Dr. Vander Luis de Souza Freitas.

Coorientador: Prof. Dr. Tiago Garcia de Senna Carneiro.

Monografia (Bacharelado). Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Biológicas. Graduação em Ciência da Computação .

1. Arquitetura de software. 2. Interface de programas aplicativos (Software). 3. Desempenho - Avaliação. 4. Ciência da computação. I. Freitas, Vander Luis de Souza. II. Carneiro, Tiago Garcia de Senna. III. Universidade Federal de Ouro Preto. IV. Título.

CDU 004.2

Bibliotecário(a) Responsável: Paulo Vitor Oliveira - CRB6/2551



FOLHA DE APROVAÇÃO

Nicolas Vasca Galindo

Desenvolvimento e análise de duas arquiteturas de software para a operação de um serviço em nuvem destinado à contratação de profissionais de limpeza doméstica

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 7 de Fevereiro de 2024.

Membros da banca:

Vander Luis de Souza Freitas (Orientador) - Doutor - Universidade Federal de Ouro Preto
Tiago Garcia de Senna Carneiro (Coorientador) - Doutor - Universidade Federal de Ouro Preto
Guilherme Tavares de Assis (Examinador) - Doutor - Universidade Federal de Ouro Preto
Daniel Ludovico Guidoni (Examinador) - Doutor - Universidade Federal de Ouro Preto

Vander Luis de Souza Freitas, Orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 16/02/2024.



Documento assinado eletronicamente por **Vander Luis de Souza Freitas, PROFESSOR DE MAGISTERIO SUPERIOR**, em 16/02/2024, às 15:57, 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 **0665671** e o código CRC **BA841FA9**.

Resumo

Nos últimos anos, observou-se um crescimento significativo no uso e adesão das pessoas a serviços através de aplicativos, devido à popularização dos smartphones. Os aplicativos, comumente conhecidos como apps, tornaram-se uma solução tecnológica amplamente comercializada. Esta monografia propõe o desenvolvimento e a análise de desempenho de duas arquiteturas de software para a operação de um serviço em nuvem destinado à contratação de profissionais de limpeza doméstica. Apresentamos a prototipação dos aplicativos aos quais a API atenderá, utilizando a linguagem de programação JavaScript para o desenvolvimento, juntamente com o banco de dados NoSQL MongoDB para o armazenamento de dados. Apresenta-se a API em duas versões, uma com arquitetura monolítica e outra baseada em microserviços, visando atender a um conjunto de requisitos funcionais e não-funcionais. Ambas são hospedadas utilizando os serviços disponibilizados pela Amazon Web Services (AWS). As arquiteturas são confrontadas a partir de uma análise de desempenho, por meio da ferramenta Apache JMeter, levando-se em conta a taxa de requisições e o tempo de resposta. Na análise dos resultados de cada conjunto de dados, destaca-se que, ao examinar 10 e 100 threads, há pouca diferença significativa nos tempos de resposta e nas porcentagens de erro. No entanto, ao aumentar para 1000 threads, torna-se evidente a superioridade da arquitetura de microserviços devido ao seu trabalho em paralelo. Além disso, ao ampliar para 2000 e 4000 threads, observa-se que o processamento total escolhido para as arquiteturas tornou-se limitante.

Palavras-chave: API, Interface de Programação de Aplicação. Arquitetura de Software. Análise de Desempenho.

Abstract

In recent years, there has been significant growth in the use and adherence of people to services through applications, due to the popularization of smartphones. Applications, commonly known as apps, have become a widely commercialized technological solution. This monograph proposes the development and performance analysis of two software architectures for the operation of a cloud service intended for hiring domestic cleaning professionals. We present the prototyping of the applications that the API will serve, using the JavaScript programming language for development, together with the NoSQL MongoDB database for data storage. The API is presented in two versions, one with a monolithic architecture and the other based on microservices, aiming to meet a set of functional and non-functional requirements. Both are hosted using services provided by Amazon Web Services (AWS). The architectures are compared based on a performance analysis, using the Apache JMeter tool, taking into account the request rate and response time. In analyzing the results of each data set, it is highlighted that when examining 10 and 100 threads, there is little significant difference in response times and error percentages. However, when increasing to 1000 threads, the superiority of the microservices architecture becomes evident due to its parallel work. Furthermore, when expanding to 2000 and 4000 threads, it is observed that the total processing chosen for the architectures became limiting.

Keywords: API, Application Programming Interface. Software Architectures. Performance Analysis.

Lista de ilustrações

Figura 2.1 – Taxa de Desocupação no Brasil e Grandes Regiões no 1º trimestre 2022 Fonte: IBGE	4
Figura 2.2 – Identidade visual do aplicativo Get Ninjas para clientes e para profissionais. Fonte: Google Play Store	8
Figura 2.3 – Identidade visual do aplicativo Helpling. Fonte: Google Play Store	9
Figura 2.4 – Identidade visual do aplicativo Diaríssima. Fonte: Google Imagens	9
Figura 2.5 – Identidade visual do aplicativo Parafuzo. Fonte: Google Imagens	10
Figura 2.6 – Exemplo de Uma Arquitetura Monolítica. Fonte: (MICROSOFT,)	13
Figura 2.7 – Exemplo de Uma Arquitetura de Microserviços. Fonte: (MICROSOFT,)	14
Figura 3.1 – Protótipo de tela de Login do Usuário. Fonte: Elaborado pelo autor	21
Figura 3.2 – Protótipo de tela de Registro do Usuário. Fonte: Elaborado pelo autor	22
Figura 3.3 – Protótipo de tela de Registro Endereço do Usuário. Fonte: Elaborado pelo autor	22
Figura 3.4 – Protótipo de tela Home. Fonte: Elaborado pelo autor	23
Figura 3.5 – Protótipo de tela de Agendamento - Primeiras Informações. Fonte: Elaborado pelo autor	24
Figura 3.6 – Protótipo de tela de Agendamento - Finalização. Fonte: Elaborado pelo autor	25
Figura 3.7 – Protótipo de tela de Login dos Prestadores de Serviço. Fonte: Elaborado pelo autor	26
Figura 3.8 – Tela de Registro dos Prestadores de Serviço - Etapa 1. Fonte: Elaborado pelo autor	27
Figura 3.9 – Protótipo de tela de Registro dos Prestadores de Serviço - Etapa 2. Fonte: Elaborado pelo autor	28
Figura 3.10–Protótipo de tela de Perfil dos Prestadores de Serviço. Fonte: Elaborado pelo autor	29
Figura 3.11–Protótipo de tela Home - Solicitações dos Prestadores de Serviço. Fonte: Elaborado pelo autor	30
Figura 3.12–Protótipo de tela Home - Minha Agenda dos Prestadores de Serviço. Fonte: Elaborado pelo autor	31
Figura 3.13–Relações das entidades do Banco de Dados. Fonte: Elaborado pelo autor	32
Figura 3.14–Modelo de Usuários. Fonte: Elaborado pelo autor	33
Figura 3.15–Modelo de Endereço dos Usuários. Fonte: Elaborado pelo autor	34
Figura 3.16–Modelo de Prestadores de Serviço. Fonte: Elaborado pelo autor	36
Figura 3.17–Modelo de Regras. Fonte: Elaborado pelo autor	37
Figura 3.18–Modelo de Permissões. Fonte: Elaborado pelo autor	38
Figura 3.19–Modelo de Regras do Usuário. Fonte: Elaborado pelo autor	38
Figura 3.20–Modelo de Regras dos Prestadores de Serviço. Fonte: Elaborado pelo autor	39

Figura 3.21–Modelo de Agendamentos. Fonte: Elaborado pelo autor	40
Figura 3.22–Arquitetura Monolítica. Fonte: Elaborado pelo autor	52
Figura 3.23–Arquitetura de Microserviços. Fonte: Elaborado pelo autor	54
Figura 3.24–Exemplo de saída do Apache JMeter de uma rotina com 1 Threads de Usuários. Fonte: Elaborado pelo autor	56
Figura A.1–Rotina com 10 Threads de Usuários 1/3. Fonte: Elaborado pelo autor	66
Figura A.2–Rotina com 10 Threads de Usuários 2/3. Fonte: Elaborado pelo autor	66
Figura A.3–Rotina com 10 Threads de Usuários 3/3. Fonte: Elaborado pelo autor	67
Figura A.4–Rotina com 100 Threads de Usuários 1/3. Fonte: Elaborado pelo autor	67
Figura A.5–Rotina com 100 Threads de Usuários 2/3. Fonte: Elaborado pelo autor	67
Figura A.6–Rotina com 100 Threads de Usuários 3/3. Fonte: Elaborado pelo autor	68
Figura A.7–Rotina com 1000 Threads de Usuários 1/3. Fonte: Elaborado pelo autor	68
Figura A.8–Rotina com 1000 Threads de Usuários 2/3. Fonte: Elaborado pelo autor	68
Figura A.9–Rotina com 1000 Threads de Usuários 3/3. Fonte: Elaborado pelo autor	69
Figura A.10–Rotina com 2000 Threads de Usuários 1/3. Fonte: Elaborado pelo autor	69
Figura A.11–Rotina com 2000 Threads de Usuários 2/3. Fonte: Elaborado pelo autor	69
Figura A.12–Rotina com 2000 Threads de Usuários 3/3. Fonte: Elaborado pelo autor	70
Figura A.13–Rotina com 4000 Threads de Usuários 1/3. Fonte: Elaborado pelo autor	70
Figura A.14–Rotina com 4000 Threads de Usuários 2/3. Fonte: Elaborado pelo autor	70
Figura A.15–Rotina com 4000 Threads de Usuários 3/3. Fonte: Elaborado pelo autor	71
Figura A.16–Rotina com 10 Threads de Usuários 1/3. Fonte: Elaborado pelo autor	71
Figura A.17–Rotina com 10 Threads de Usuários 2/3. Fonte: Elaborado pelo autor	71
Figura A.18–Rotina com 10 Threads de Usuários 3/3. Fonte: Elaborado pelo autor	72
Figura A.19–Rotina com 100 Threads de Usuários 1/3. Fonte: Elaborado pelo autor	72
Figura A.20–Rotina com 100 Threads de Usuários 2/3. Fonte: Elaborado pelo autor	72
Figura A.21–Rotina com 100 Threads de Usuários 3/3. Fonte: Elaborado pelo autor	73
Figura A.22–Rotina com 1000 Threads de Usuários 1/3. Fonte: Elaborado pelo autor	73
Figura A.23–Rotina com 1000 Threads de Usuários 2/3. Fonte: Elaborado pelo autor	73
Figura A.24–Rotina com 1000 Threads de Usuários 3/3. Fonte: Elaborado pelo autor	74
Figura A.25–Rotina com 2000 Threads de Usuários 1/3. Fonte: Elaborado pelo autor	74
Figura A.26–Rotina com 2000 Threads de Usuários 2/3. Fonte: Elaborado pelo autor	74
Figura A.27–Rotina com 2000 Threads de Usuários 3/3. Fonte: Elaborado pelo autor	75
Figura A.28–Rotina com 4000 Threads de Usuários 1/3. Fonte: Elaborado pelo autor	75
Figura A.29–Rotina com 4000 Threads de Usuários 2/3. Fonte: Elaborado pelo autor	75
Figura A.30–Rotina com 4000 Threads de Usuários 3/3. Fonte: Elaborado pelo autor	76

Lista de tabelas

Tabela 3.1 – Tabela de serviços e funcionalidades. Fonte: Elaborado pelo autor	18
Tabela 4.1 – Tempo de execução dos experimentos realizados na arquitetura monolítica com 10 threads.	57
Tabela 4.2 – Tempo de execução dos experimentos realizados na arquitetura monolítica com 100 threads.	57
Tabela 4.3 – Tempo de execução dos experimentos realizados na arquitetura monolítica com 1000 threads.	57
Tabela 4.4 – Tempo de execução dos experimentos realizados na arquitetura monolítica com 2000 threads.	58
Tabela 4.5 – Tempo de execução dos experimentos realizados na arquitetura monolítica com 4000 threads.	58
Tabela 4.6 – Tempo de execução dos experimentos realizados na arquitetura de microserviços com 10 threads.	58
Tabela 4.7 – Tempo de execução dos experimentos realizados na arquitetura de microserviços com 100 threads.	58
Tabela 4.8 – Tempo de execução dos experimentos realizados na arquitetura de microserviços com 1000 threads.	59
Tabela 4.9 – Tempo de execução dos experimentos realizados na arquitetura de microserviços com 2000 threads.	59
Tabela 4.10–Tempo de execução dos experimentos realizados na arquitetura de microserviços com 4000 threads.	59

Lista de abreviaturas e siglas

API	Interface de Programação de Aplicação
IBGE	Instituto Brasileiro de Geografia e Estatística
PNAD	Pesquisa Nacional por Amostra de Domicílios
AWS	Amazon Web Services

Sumário

1	Introdução	1
1.1	Justificativa	2
1.2	Objetivos	2
1.3	Organização da Monografia	3
1.3.1	Estrutura da Monografia	3
2	Revisão Bibliográfica	4
2.1	Embasamento Histórico	4
2.1.1	Desemprego no Brasil	4
2.1.2	Setor de serviços	5
2.1.3	Economia compartilhada	6
2.2	Trabalhos Relacionados	7
2.2.1	Get Ninjas	7
2.2.2	Helpling	8
2.2.3	Diaríssima	9
2.2.4	Parafuzo	10
2.3	Fundamentação Teórica	11
2.3.1	Linguagens de programação	11
2.3.1.1	JavaScript	11
2.3.1.2	NodeJs	11
2.3.2	Frameworks Web	12
2.3.2.1	ExpressJs	12
2.3.3	Banco de Dados	12
2.3.3.1	NoSQL	12
2.3.3.2	MongoDB	12
2.3.4	Arquitetura de software	12
2.3.4.1	Arquitetura Monolítica	13
2.3.4.2	Arquitetura de Microserviços	13
2.3.5	Infraestrutura	14
2.3.5.1	Amazon Web Services (AWS)	14
2.3.5.1.1	Amazon Elastic Compute Cloud (Amazon EC2)	15
2.3.6	Análise de Desempenho	15
2.3.6.1	Apache JMeter	15
2.3.7	Figma	15
3	Metodologia	16
3.1	Requisitos	16
3.1.1	Requisitos Funcionais	16

3.1.2	Requisitos Não Funcionais	19
3.1.2.1	Usabilidade	19
3.1.2.2	Confiabilidade	19
3.1.2.3	Manutenibilidade	19
3.2	Prototipação	20
3.2.1	Aplicativo dos Usuários	20
3.2.1.1	Login	20
3.2.1.2	Registro	21
3.2.1.3	Home	23
3.2.1.4	Agendamentos	23
3.2.2	Aplicativo dos Prestadores de Serviço	25
3.2.2.1	Login	25
3.2.2.2	Registro	26
3.2.2.3	Perfil	28
3.2.2.4	Home	29
3.3	Banco de Dados	31
3.3.1	Users	32
3.3.2	UserAddresses	33
3.3.3	Cleanears	35
3.3.4	Roles	37
3.3.5	Permissions	37
3.3.6	UserRoles	38
3.3.7	CleanerRoles	38
3.3.8	Schedules	39
3.4	Desenvolvimento	41
3.4.1	AuthenticationService	42
3.4.1.1	userLogin	42
3.4.1.2	userSignup	42
3.4.1.3	cleanerLogin	43
3.4.1.4	cleanerSignup	43
3.4.2	AuthorizationService	43
3.4.2.1	checkUserPermission	43
3.4.2.2	checkCleanerPermission	44
3.4.3	UsersService	44
3.4.3.1	login	44
3.4.3.2	create	45
3.4.3.3	find	45
3.4.3.4	findById	45
3.4.4	UserAddressesService	45

3.4.4.1	create	45
3.4.4.2	find	46
3.4.4.3	findById	46
3.4.4.4	destroy	46
3.4.5	CleanersService	46
3.4.5.1	login	47
3.4.5.2	create	47
3.4.5.3	find	47
3.4.5.4	findById	47
3.4.5.5	update	48
3.4.6	RolesService	48
3.4.6.1	find	48
3.4.6.2	findById	48
3.4.7	PermissionsService	49
3.4.7.1	find	49
3.4.7.2	findById	49
3.4.8	UserRolesService	49
3.4.8.1	create	49
3.4.8.2	find	50
3.4.9	CleanerRolesService	50
3.4.9.1	create	50
3.4.9.2	find	50
3.4.10	SchedulesService	51
3.4.10.1	create	51
3.4.10.2	find	51
3.4.10.3	findById	51
3.4.10.4	update	52
3.5	Arquiteturas	52
3.5.1	Arquitetura Monolítica	52
3.5.2	Arquitetura de Microserviços	53
3.6	Infraestrutura	54
3.7	Análise de Desempenho	55
4	Resultados	57
4.1	Arquitetura Monolítica	57
4.2	Arquitetura de Microserviços	58
4.3	Arquitetura Monolítica x Arquitetura de Microserviços	59
5	Considerações Finais	61
5.1	Trabalhos Futuros	61

Referências	62
------------------------------	-----------

Apêndices	65
------------------	-----------

APÊNDICE A Capturas de tela dos experimentos realizados com o Apache JMeter	66
--	-----------

A.1 Análise de Desempenho Arquitetura Monolítica	66
A.1.1 10 Threads	66
A.1.2 100 Threads	67
A.1.3 1000 Threads	68
A.1.4 2000 Threads	69
A.1.5 4000 Threads	70
A.2 Análise de Desempenho Arquitetura Microserviços	71
A.2.1 10 Threads	71
A.2.2 100 Threads	72
A.2.3 1000 Threads	73
A.2.4 2000 Threads	74
A.2.5 4000 Threads	75

1 Introdução

No ano de 2019, a Fundação Getúlio Vargas informou, por meio de uma pesquisa, que o número de dispositivos móveis estavam ultrapassando consideravelmente o número de brasileiros (EASP, 2019). O Instituto Brasileiro de Geografia e Estatística (IBGE) apontou que eram, em média, 230 milhões de aparelhos celulares para 210 milhões de pessoas (IBGE, 2019).

Diante do acesso a esse instrumento tecnológico tão comum - o celular - e a evolução e expansão da rede, também conhecida como 3G e 4G que possibilitou a disseminação da tecnologia móvel (BERGHE, 2020), os aplicativos, comumente conhecidos como apps, passaram a ser uma solução tecnológica que tem tornado os produtos comercializados em meio digital acessíveis a todas as pessoas e em todos os lugares.

Os aplicativos tem facilitado a vida das pessoas que utilizam frequentemente essa tecnologia, trazendo conforto e agilidade em processos básicos do dia a dia como visualizar uma fatura, verificar o saldo bancário e até mesmo providenciar um lanche. Essa potencialidade tecnológica ganhou destaque durante a Pandemia da COVID-19. Diante de todas as medidas e restrições sanitárias, segundo os estudos feitos pela as pessoas passaram a utilizar ainda mais os aplicativos como uma das formas de estarem em isolamento, para evitarem o contato e contágio pelo vírus, como indicado pelo estudo feito pela Mobills¹, startup de gestão de finanças pessoais, os gastos com os principais aplicativos de entregas focados no *delivery* de comida cresceram 149%.

Outro aspecto relevante foi que, diante de uma economia visivelmente abalada pelo novo vírus, a população precisou demonstrar resiliência e se reinventar no novo cenário para lidar com suas questões financeiras. O alto índice de desemprego transformou o mercado e muitas pessoas migraram para o mercado informal. Na busca por novas fontes de renda, a utilização dos aplicativos aumentou em proporção a necessidade das pessoas o que resultou na migração de muitos trabalhadores para este serviço. Com isso, surgiram mais motoristas de aplicativos, entregadores e diaristas, somando 24, 5 milhões de pessoas nessa modalidade de trabalho (RUFINO, 2021).

Conforme dados do IBGE (2021), estima-se que mais de 7 milhões de mulheres atuem como empregadas domésticas. Embora haja homens nessa ocupação, a predominância é do gênero feminino, com destaque para mulheres negras pertencentes às classes sociais mais baixas (DIESSE, 2020).

O município de Ouro Preto-MG, assim como as demais localidades do Brasil, conta com uma significativa demanda de pessoas que utilizam e prestam serviços domésticos. Um exemplo típico na cidade são as repúblicas universitárias que comumente solicitam esse serviço,

¹ <<https://www.idinheiro.com.br/arquivos/noticias/gastos-com-delivery-crescem-durante-pandemia>>

como reportado pelo Jornal Voz Ativa ². Geralmente, a disseminação das demandas de trabalho acontece por meio de indicações e referências pessoais.

A partir de um contexto específico de prestação de serviços, com o objetivo de melhorar a busca por diaristas qualificadas e aumentar a oferta de serviços para empregadas domésticas, este estudo propõe o desenvolvimento do *backend* de um aplicativo de serviços. Serão exploradas duas arquiteturas distintas, monolítica e de microsserviços, a fim de identificar qual delas melhor atende às demandas do aplicativo.

O desenvolvimento *backend* é uma área fundamental no processo de criação de aplicações, responsável pela implementação e gerenciamento de todo o lado do servidor. Ele lida com a lógica de negócios, processamento de dados, integrações com sistemas externos e gerenciamento de banco de dados (FREMAN; BATES, 2004).

A análise de desempenho é uma área fundamental da engenharia de software, cujo objetivo é verificar como o sistema se comporta em termos de velocidade, escalabilidade, estabilidade e capacidade de resposta em diferentes cargas de trabalho (NGUYEN; KHA; HOANG, 2017). Para tal, foram realizados testes em duas arquiteturas, monolítica e de microsserviços, com 10, 100, 1000, 2000 e 4000 threads, considerando usuários simultâneos conectados, levando em conta o tempo médio total de 90% das iterações realizadas.

1.1 Justificativa

A sociedade moderna está cada vez mais acelerada e as pessoas dispendo de menos tempo para resolverem determinadas tarefas exigidas no dia a dia. Posto isso, se torna relevante o uso de uma ferramenta (como a que é proposta neste trabalho) que atenda com agilidade e praticidade as demandas do cotidiano, como a busca por um serviço de limpeza doméstica.

A motivação para a pesquisa se dá pela possibilidade de se criar uma API que seja capaz de contribuir no fornecimento de informações confiáveis acerca da procura e oferta de serviços domésticos em Ouro Preto, disponibilizando dados e referências para uma contratação de serviços com de forma segura e confiável, para ambos os lados. Ainda que a oferta e procura desse serviço seja grande, a contratação de um profissional qualificado e com boas referências exige tempo, visto que existe a preocupação com a segurança de todos os envolvidos (contratante e contratado) e confiabilidade na execução do serviço.

1.2 Objetivos

O objetivo é realizar desenvolver um serviço em nuvem destinado à contratação de profissionais de limpeza doméstica, com o intuito de melhorar a busca por diaristas qualificadas e

² <<https://jornalvozativa.com/destaque/na-coluna-trabalhadores-direitos-o-emprego-domestico-em-republicas-estudantis>>

a oferta de serviços para empregadas domésticas. Foram analisadas e comparadas duas arquiteturas distintas, monolítica e de microsserviços, para determinar qual delas melhor atende às necessidades do serviço.

Para atingir o objetivo geral, existem algumas atividades metodológicas:

- Desenvolver o backend de um aplicativo de contratação de serviços de limpeza;
- Implementar e Investigar a diferença de uma arquitetura monolítica e arquitetura em micro serviços;
- Realizar a análise de desempenho para avaliar qual das arquitetura terá o menor tempo de resposta

1.3 Organização da Monografia

Na introdução são apresentados o contexto do trabalho, os objetivos e a justificativa. Para o referencial teórico, apresenta-se fontes pertinentes ao assunto, ferramentas, serviços e produtos existentes diretamente relacionados ao tema do trabalho. Na metodologia, descreve-se os método de pesquisa, as técnicas e instrumentos que foram utilizados para a coleta de dados, o projeto da aplicação e as decisões tomadas. Nos resultados e discussões, apresenta-se o sistema pronto e sua aplicabilidade. Por fim, na seção de conclusão e trabalhos futuros, são descritas as conclusões obtidas e apontamentos para outras tarefas que poderão ser desenvolvidas.

1.3.1 Estrutura da Monografia

Capítulo 1: Introdução.

Capítulo 2: Revisão Bibliográfica/ Embasamento Teórico (com o referencial teórico e trabalhos relacionados).

Capítulo 3: Metodologia (material e métodos).

Capítulo 4: Resultados e Discussões.

Capítulo 5: Considerações Finais (com conclusão e trabalhos futuros).

2 Revisão Bibliográfica

Neste capítulo são apresentadas todas as referências utilizadas para a realização desta monografia, divididas em

1. Embasamento Histórico
2. Trabalhos Relacionados
3. Fundamentação Teórica

2.1 Embasamento Histórico

Neste capítulo, será demonstrado o embasamento histórico e social ao qual esta monografia se dedicará, agilizando o processo e facilitando a contratação do serviço informal de faxina.

2.1.1 Desemprego no Brasil

O Instituto Brasileiro de Geografia e Estatística (2022), diz que o desemprego se refere às pessoas com mais de 14 anos que não estão trabalhando, mas estão disponíveis e tentam encontrar trabalho. Assim, para alguém ser considerado desempregado, não basta não possuir um emprego. A PNAD (2015) Contínua é a pesquisa que mostra quantos desempregados há no Brasil. Nela, o que aparece como taxa de desocupação é a porcentagem de pessoas na força de trabalho que estão desempregadas. A Figura 2.1 demonstra os dados de desocupação do mercado de trabalho no Brasil, de acordo com os últimos resultados da PNAD Contínua.

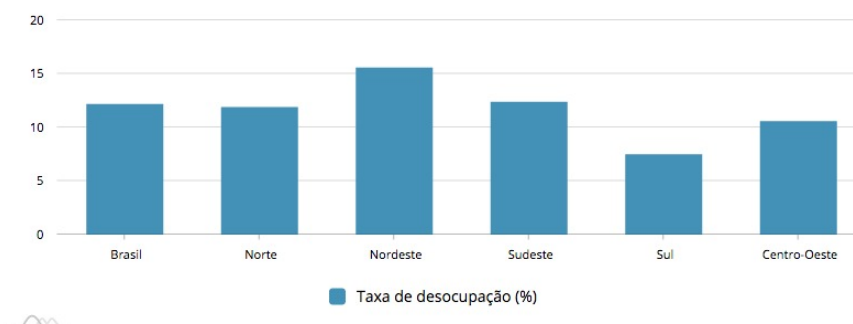


Figura 2.1 – Taxa de Desocupação no Brasil e Grandes Regiões no 1º trimestre 2022 Fonte: IBGE

Segundo os dados, a taxa de desocupação no Brasil no primeiro trimestre de 2022 era de 11,1%, o que corresponde a 11,9 milhões de pessoas desempregadas (INSTITUTO

[BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA, 2022](#)). A região com a maior taxa de desemprego foi a Nordeste (14,9%), seguida por Norte (11,7%), Sudeste (11,1%), Centro-oeste (8,5%) e por último a região Sul (6,5%), apresentando o menor índice de desemprego.

Desde 2015, o desemprego vem aumentando entre a população brasileira e cresceu ainda mais após o início da pandemia, devido às medidas sanitárias que foram estabelecidas ([NEVES, 2021](#)). Foi exigido o isolamento social e isso impossibilitou a abertura de negócios e empresas. Com a perda de clientes, algumas empresas não resistiram, resultando em mais desemprego. Diante disso, no final de 2020, o país atingiu o recorde de 13,4 milhões de pessoas esperando por uma oportunidade de trabalho ([PEREIRA; FONTÃO; LOPES, 2021](#)). Com o declínio dos empregos nas indústrias, ocorreu um aumento em serviços autônomos e informais. Embora a informalidade seja um fenômeno antigo do mercado de trabalho, há um novo fenômeno potencializado pela pandemia, chamando trabalho ‘uberizado’, ou seja, o trabalho exercido por meio de plataformas digitais ([LEITE, 2020](#)).

2.1.2 Setor de serviços

Na era do conhecimento, há uma tendência de os empregos serem mais frequentes na área de serviços, já que para a produção de bens o homem foi substituído pelas máquinas ([SANTANA; NETO, 2021](#)). Em 1950 o setor de serviços no Brasil ocupava 26% da força de trabalho empregada, em 1973 esse número atingiu 39% e atualmente esse número chega a 70% da população brasileira. Contudo, a pandemia COVID-19 desequilibrou muitos setores da economia brasileira, e um deles foi o setor de serviços. Segundo o IBGE, o volume de serviços em 2020 diminuiu 7,8%. No entanto, nos últimos meses de 2021, o setor apresentou um crescimento significativo, ganhando mais destaque no mercado e representando uma importante recuperação na saúde dos negócios ([PEREIRA; FONTÃO; LOPES, 2021](#)).

As plataformas criam ambientes mais flexíveis e atrativos, sem a rigidez dos empregos tradicionais, o que contribui na atração e adesão de mais pessoas. Assim, as formas flexíveis em tempo parcial de trabalho, terceirização e informalidade são exercidas cada vez mais pelos trabalhadores ([SANTANA; NETO, 2021](#)).

O Instituto Brasileiro de Geografia e Estatística (2020), aponta que o trabalho informal compreende unidades econômicas que produzem bens e serviços com o intuito principal de gerar ocupação e rendimento para as pessoas envolvidas, operando, tipicamente, com alguma ou nenhuma divisão entre trabalho, com baixo nível de organização e sendo em pequena escala. Isso demonstra que nem todo trabalho informal foi consequência ou fuga do índice de desocupação ([SANTANA; NETO, 2021](#)).

2.1.3 Economia compartilhada

[Botsman e Rogers \(2011\)](#) explicam que a economia compartilhada traduz a necessidade da sociedade por um mercado mais colaborativo, no qual se prioriza o acesso a bens e serviços ao invés da posse destes. Assim, definiram a economia compartilhada como um modelo econômico baseado no compartilhamento de bens e serviços subutilizados, que podem ser de espaços a habilidades, de coisas ou propriedades a benefícios monetários ou não monetários. Para os autores, a economia compartilhada contempla 3 possíveis tipos de sistemas:

- a) Mercados de redistribuição: Baseia-se no princípio do “reduza, reuse, recicle, conserte e redistribua”. Ou seja, um bem antigo passa a ter um novo dono.
- b) Estilos de vida colaborativos: baseia-se no compartilhamento de recursos, tais como dinheiro, habilidades e tempo. Pessoas dividem interesses similares e ajudam uns aos outros. Este tipo é bastante facilitado pela tecnologia digital.
- c) Sistemas de produtos e serviços: ocorre quando o consumidor paga pelo benefício do produto, e não pelo produto em si.

As economias compartilhadas são aquelas nas quais o dinheiro, ou a possibilidade de lucrar, não é o fator mais relevante no incentivo ou motivação à participação. Elas podem ser divididas em dois tipos: economias compartilhadas de produção e economias compartilhadas de consumo. Em grande parte, estas economias compartilhadas de produção são permitidas pelas redes tecnológicas presentes na internet ([JOHN, 2013](#)).

Já [Belk \(2014\)](#) aborda a economia compartilhada como uma aquisição e distribuição de um recurso em troca de uma taxa ou outra compensação combinada, num conceito parecido com Botsman, porém mais simplificado. Ele trata os consumidores como colaboradores deste tipo de economia, afirmando que esta economia compartilhada apresenta à sociedade sentimentos de vínculo, colaboração e proximidade, dando um sentido diferente ao consumo.

[Hamari, Sjöklint e Ukkonen \(2015\)](#) definem como características da economia compartilhada: a colaboração online, o comércio social (integração das interações sociais nas redes com ações de venda online), a noção de compartilhamento online e a ideologia de consumo. Ele explica ainda que vários exemplos de economia compartilhada também têm em comum estas características, ou seja, há uma nítida colaboração e um compartilhamento online, a promoção de um comércio social, e também alguma forma de ideologia subjacente, tal qual um propósito coletivo ou bem comum.

A economia compartilhada também é compreendida como um conjunto de iniciativas de consumo conectado, que enfatiza a reutilização de produtos e incentiva as conexões peer-to-peer, aproveitando para eliminar intermediários e possibilitando as interações face

a face, além de proporcionar uma nova configuração dos modelos de negócio da economia tradicional (SILVEIRA; PETRINI; SANTOS, 2016).

A criação de um aplicativo de serviço de limpeza está em sintonia com a tendência da economia compartilhada. Portanto, ao desenvolver um aplicativo de serviço de limpeza que facilite a conexão entre prestadores de serviços e clientes, estará sendo incentivada a economia compartilhada, trazendo benefícios como a reutilização de recursos e a aproximação entre pessoas que compartilham interesses similares, como defendido por Belk (2014). O aplicativo atenderá às necessidades da sociedade moderna, oferecendo soluções eficientes de limpeza, enquanto promove a colaboração e o compartilhamento de recursos.

2.2 Trabalhos Relacionados

Mais do que apenas movidos por bons salários, os profissionais no mercado estão à procura de boas condições de trabalho, comodidade, flexibilidade, e, acima de tudo, de uma atividade que lhes permita se sentirem recompensados pelo investimento de tempo e esforço e que gere prazer e satisfação. Diante disso, o empreendedorismo no Brasil tem se fortalecido cada vez mais e as pessoas vêm encontrando novas formas de guiar o próprio negócio com baixo investimento inicial e rápidos retornos (FREITASE, 2018).

Atualmente observamos que existe uma interação virtual entre os indivíduos e os serviços. As empresas entenderam que a tecnologia é uma aliada para fornecer seus serviços e expandir seus negócios. Junto ao crescente uso da tecnologia, o número de pessoas que moram sozinhas também aumentou, e, conseqüentemente, uma crescente procura por serviços domésticos (SOUZA, 2018).

No mercado de trabalho já existem alguns modelos de prestação de serviços domésticos. Souza (2018) listou os pontos negativos e positivos de alguns desses aplicativos conforme descritos a seguir.

2.2.1 Get Ninjas

O aplicativo Get Ninjas¹ oferece serviços em diversas áreas: aulas, design, moda e beleza, saúde e serviços domésticos. O público do aplicativo é diversificado, pois oferece vários tipos de prestação de serviços. Os pontos positivos são: as categorias bem organizadas, o cadastro fácil e o mapa de navegação é intuitivo. Os pontos negativos são: não possui avaliação dos usuários, o layout e design não são explorados, há pouca microinteração, existem muitas perguntas desnecessárias e é preciso esperar o prestador de serviço agendar. A Figura 2.2 mostra a identidade visual das versões do aplicativo, tanto para clientes como para os profissionais.

¹ <<https://play.google.com/store/apps/details?id=br.com.getninjas.client>>



Figura 2.2 – Identidade visual do aplicativo Get Ninjas para clientes e para profissionais. Fonte: Google Play Store

Segue a lista de tecnologias utilizadas pelo Get Ninjas:

- a) **Linguagens de programação:** O GetNinjas utiliza várias linguagens de programação, como JavaScript, Python e Ruby, para desenvolver diferentes partes de sua plataforma.
- b) **Frameworks web:** Eles usam alguns *frameworks* web populares, como Node.js, React.js e Ruby on Rails, para desenvolver a parte do servidor e a interface do usuário do site.
- c) **Banco de dados:** O GetNinjas utiliza bancos de dados relacionais, como PostgreSQL e MySQL, para armazenamento e gerenciamento dos dados de usuários, serviços e outras informações relacionadas.
- d) **Infraestrutura e Hospedagem:** A infraestrutura do GetNinjas é hospedada na AWS² (Amazon Web Services), uma plataforma de serviços em nuvem.

2.2.2 Helpling

O aplicativo Helping³ é um aplicativo internacional presente na Alemanha e em mais 14 países na Europa. Tem objetivo facilitar a contratação de serviços de limpeza para suas casas. Também oferece limpeza de estofados, janelas, tapetes, pós-obra, mudança, desmontagem de móveis e pintor. O público alvo são pessoas que não disponibilizam de tempo para esses serviços. Os pontos positivos do aplicativo são: existem várias formas de pagamento, possui sistema de avaliações depoimento dos usuários, o mapa de navegação é intuitivo e possui facilidade de agendamento. Já o ponto negativo é que ele só funciona na Europa. A Figura 2.3 a seguir mostra a identidade visual do aplicativo.

² <<https://aws.amazon.com/>>

³ <<https://play.google.com/store/apps/details?id=com.helpling.app.h2.provider>>



Figura 2.3 – Identidade visual do aplicativo Helpling. Fonte: Google Play Store

Segue a lista de tecnologias utilizadas pela Helping:

- a) **Linguagens de programação:** O Helpling utiliza várias linguagens de programação, como JavaScript, Python e Ruby, para desenvolver diferentes partes de sua plataforma.
- b) **Frameworks web:** Eles utilizam alguns frameworks web populares, como Node.js, React.js e Ruby on Rails, para desenvolver a parte do servidor e a interface do usuário do site.
- c) **Banco de dados:** O Helpling utiliza bancos de dados relacionais, como PostgreSQL e MySQL, para armazenamento e gerenciamento dos dados de usuários, serviços e outras informações relacionadas.
- d) **Infraestrutura e Hospedagem:** A infraestrutura do Helpling é hospedada na AWS (Amazon Web Services), uma plataforma de serviços em nuvem.

2.2.3 Diaríssima

O aplicativo Diaríssima é um aplicativo de serviços exclusivo para diaristas e faxineiras e está presente em 92 cidades brasileiras. O público alvo é composto por adultos que não possuem tempo para se dedicar a limpeza de seus lares. As vantagens são que esse aplicativo possui opção de busca rápida e tem um sistema de avaliação e depoimento dos usuários. Os aspectos negativos são: possui agendamento confuso e desorganizado, o cadastro não é intuitivo, possui problemas na interação e a experiência é cansativa. A Figura 2.4 mostra a identidade visual do aplicativo.



Figura 2.4 – Identidade visual do aplicativo Diaríssima. Fonte: Google Imagens

Segue a lista de tecnologias utilizadas pela Diaríssima:

- a) **Linguagens de programação:** A Diaríssima utiliza linguagens de programação como JavaScript e Python para o desenvolvimento de diferentes componentes e funcionalidades da plataforma.
- b) **Frameworks web:** Eles fazem uso de frameworks web populares, como React.js e Django, para construir a interface do usuário e gerenciar a lógica do lado do servidor.
- c) **Banco de dados:** A Diaríssima utiliza bancos de dados relacionais, como PostgreSQL ou MySQL, para armazenar e gerenciar dados importantes, como informações de usuários, agendamentos e avaliações.
- d) **Infraestrutura e Hospedagem:** A infraestrutura da Helpling é hospedada na AWS (Amazon Web Services), uma plataforma de serviços em nuvem.

2.2.4 Parafuzo

A Parafuzo⁴ é uma empresa brasileira que visa atender às necessidades domésticas. Ela oferece serviços de contratação de profissionais qualificados para diversas demandas específicas como, por exemplo, a limpeza de casa ou escritório, com opções de limpeza padrão, intensiva, pós-reforma ou pré-mudança, ou para o cuidado das roupas e a montagem dos móveis, que garante a presença do especialista adequado. Com operações no mercado nacional, a empresa se destaca por auxiliar na busca pelo profissional ideal para cada tarefa, proporcionando um serviço satisfatório e de qualidade. As vantagens são que o aplicativo oferece uma variedade de serviços de limpeza, incluindo limpeza residencial, limpeza pós-obra, limpeza de escritórios, entre outros, atendendo a diferentes necessidades dos usuários, flexibilidade de agendar os serviços de acordo com sua conveniência, escolhendo datas e horários que se adequem às suas necessidades. Os aspectos negativos são que dependendo da região a disponibilidade de profissionais de limpeza é limitada e a limitação em relação à personalização dos serviços de limpeza, já que os usuários precisam escolher entre as opções disponíveis no aplicativo.



Figura 2.5 – Identidade visual do aplicativo Parafuzo. Fonte: Google Imagens

Segue a lista de tecnologias utilizadas pela Parafuzo:

⁴ <<https://play.google.com/store/apps/details?id=com.parafuzo>>

- a) **Linguagens de programação:** A Parafuzo utiliza algumas linguagens de programação JavaScript (Node.js), HTML, CSS, para desenvolver diferentes partes de sua plataforma.
- b) **Frameworks web:** Eles utilizam frameworks web populares, como React.js, Next.js e Express.js , para criar a interface do usuário e gerenciar a lógica do lado do servidor.
- c) **Banco de dados:** A Parafuzo utiliza bancos de dados nao-relacionais, como MongoDB, para armazenar e gerenciar dados importantes, como informações de usuários, agendamentos e pagamentos.
- d) **Infraestrutura e Hospedagem:** A infraestrutura da Parafuzo é hospedada na AWS (Amazon Web Services), uma plataforma de serviços em nuvem e Firebase.
- e) **Ferramentas de Desenvolvimento:** Visual Studio Code, Git.

2.3 Fundamentação Teórica

As APIs com arquitetura monolítica e de microserviços foram desenvolvidas na linguagem de programação JavaScript (Node.js), conectadas ao banco de dados NoSQL MongoDB, utilizando o framework Express.js. Para a prototipação, foi utilizada a ferramenta Figma. Por fim, para a análise de desempenho, foi empregada a ferramenta Apache JMeter.

2.3.1 Linguagens de programação

2.3.1.1 JavaScript

JavaScript é uma linguagem de programação de alto nível, dinâmica e versátil, utilizada em desenvolvimento de aplicações web. De acordo com [Doe \(2021\)](#), ela foi criada originalmente visando adicionar recursos de interatividade às páginas da web. Com o tempo, a linguagem passou por evoluções e se tornou uma das principais opções para o desenvolvimento de aplicativos front-end e back-end.

De acordo com [Smith \(2021\)](#), o JavaScript permite a criação de funcionalidades avançadas, como manipulação de elementos HTML, interações do usuário, validação de formulários, animações e outras. Sua sintaxe simples e flexível torna a linguagem acessível para programadores iniciantes e avançados.

2.3.1.2 NodeJs

[Oliveira \(2021\)](#) aborda que o Node.js é uma plataforma de desenvolvimento JavaScript. Ele atua como um ambiente de execução com sua própria máquina virtual, projetada para interpretar e executar scripts de forma autônoma. Quando um comando escrito em JavaScript é executado, o Node.js é composto por dois componentes principais e seus

módulos. O core é construído em C/C++ e combinado com o mecanismo do Google V8 que é um compilador open source JIT (Just In Time) escrito em C++. (SILVA, 2022a).

2.3.2 Frameworks Web

2.3.2.1 ExpressJs

O Express.js é um framework para Node.js que oferece recursos essenciais para a construção de servidores web HTTP (SILVA, 2022b). De acordo com Silva (2023), sua principal proposta é simplificar o desenvolvimento de aplicativos, servindo como um facilitador para as tarefas comumente encontradas nesse contexto, como o desenvolvimento de programas web. Com uma ampla popularidade, se torna uma opção poderosa para os desenvolvedores construírem servidores eficientes e escaláveis usando o Node.js (OLIVEIRA, 2021).

2.3.3 Banco de Dados

2.3.3.1 NoSQL

Por muitos anos, os bancos de dados relacionais foram amplamente adotados como a forma de armazenamento (PARAMOUND; MARTIN, 2012). No entanto, outro modelo começou a ganhar destaque: o NoSQL. Ele se refere a uma base de dados não relacional e enquanto em modelos de dados relacional a entidade é armazenada com referência de outra, no NoSQL a entidade inteira é armazenada.

2.3.3.2 MongoDB

O MongoDB⁵ é um sistema de gerenciamento de banco de dados NoSQL (não relacional) amplamente utilizado em aplicações modernas, graças à sua flexibilidade, escalabilidade e desempenho ágil (MARTINS, 2022). Sua principal característica é o modelo flexível de armazenamento de dados, utilizando documentos no formato JSON (JavaScript Object Notation), o que permite que os dados sejam armazenados em documentos semelhantes a objetos. Devido a essas características o MongoDB é popular entre desenvolvedores e empresas que lidam com grandes volumes de dados (MARTINS, 2022).

2.3.4 Arquitetura de software

Bass, Clements e Kazman (1998) afirmam que a arquitetura de software de um sistema são as estruturas que compõem o sistema, incluindo os componentes de software, suas características externamente observáveis e as interações entre eles. GARLAN (2008) complementa, dizendo que a arquitetura é caracterizada pelos elementos que compõem um sistema, suas interações e pelos princípios e diretrizes que orientam o seu design e evolução

⁵ <<https://www.mongodb.com/>>

ao longo do tempo. Portanto a arquitetura de software determina como as funcionalidades estão distribuídas entre os componentes que formam o sistema.

2.3.4.1 Arquitetura Monolítica

Uma arquitetura monolítica é comumente encontrada devido à sua simplicidade no desenvolvimento e teste de aplicações. As IDEs (*Integrated Development Environments*) geralmente são focadas na construção de aplicações individuais, o que torna favorável o uso da Arquitetura Monolítica (CHRIS; FLOYD, 2016).

Em uma API monolítica, todas as funções estão em um único pacote. Sua principal característica é acoplamento de módulos, sendo possível acessar código de módulos distintos sem a necessidade de realizar integrações, conforme ilustra a Figura 2.6.

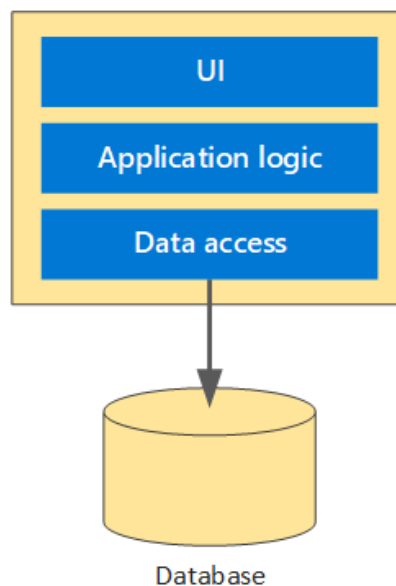


Figura 2.6 – Exemplo de Uma Arquitetura Monolítica. Fonte: (MICROSOFT,)

Devido a essas características, ela é considerada de rápido desenvolvimento, reforçando a sua utilização mesmo em sistemas atuais. Um exemplo é a empresa Microsoft ⁶ na arquitetura de rede do Windows, publicada em seu site oficial ⁷.

2.3.4.2 Arquitetura de Microserviços

A Arquitetura de Microserviços está ganhando destaque devido à sua abordagem de dividir as funcionalidades em pequenos e autônomos serviços. Cada serviço é independente e implementa uma única funcionalidade, resultando em uma maior modularização. Conforme citado por Chris e Floyd (2016) ao invés de desenvolver uma única aplicação monolítica,

⁶ <<https://www.microsoft.com>>

⁷ <<https://learn.microsoft.com/pt-br/windows-hardware/drivers/network/windows-network-architecture-and-the-osi-model>>

a ideia é separar a aplicação em um conjunto de pequenos serviços interconectados. A Figura 2.7 ilustra a divisão externa e interna da arquitetura de microserviços.

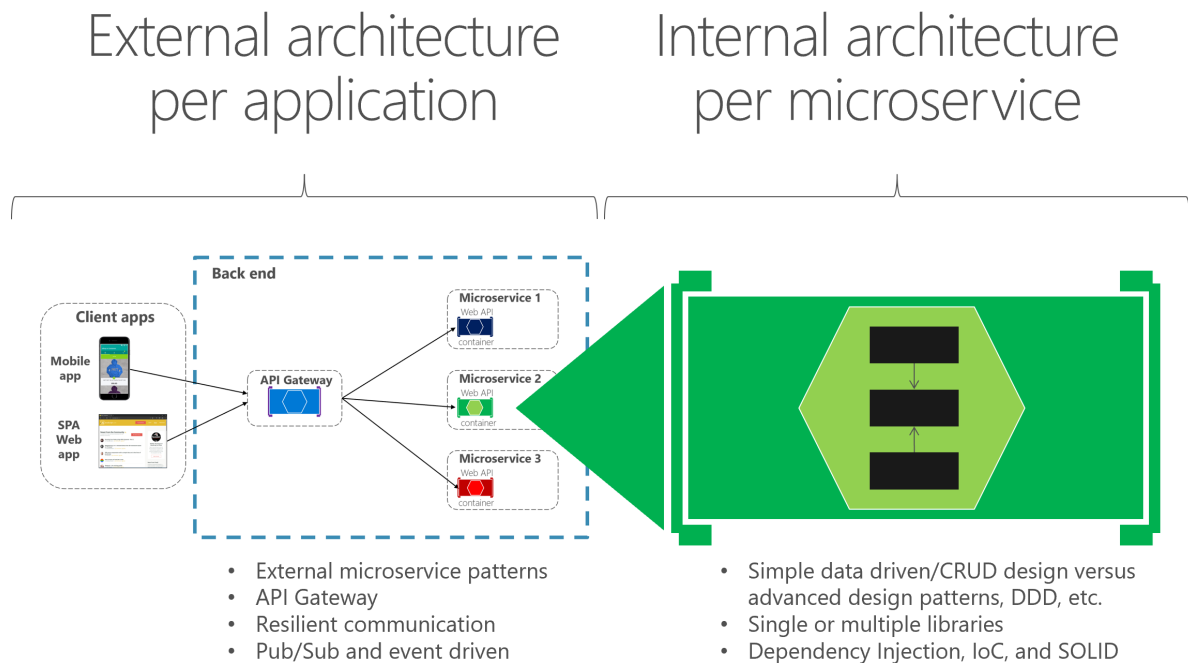


Figura 2.7 – Exemplo de Uma Arquitetura de Microserviços. Fonte: (MICROSOFT,)

Com base em Newman (2021), ao implementar os microserviços, é importante considerar dois termos amplamente utilizados no contexto de sistemas orientados a objetos. O primeiro termo é o ‘fraco acoplamento’, que se refere à ideia de que uma mudança em um determinado serviço não deve causar a necessidade de alterações em outros serviços. O segundo termo é a ‘alta coesão’, que enfatiza que funcionalidades relacionadas devem estar agrupadas juntas, enquanto funcionalidades não relacionadas devem ser mantidas em lugares separados. Uma grande empresa que utiliza a arquitetura de microserviços é a Netflix ⁸(MESHENBERG, 2016)

2.3.5 Infraestrutura

2.3.5.1 Amazon Web Services (AWS)

A Amazon Web Services (AWS) é uma plataforma de serviços em nuvem feita pela Amazon. Ela oferece serviços e soluções que ajudam empresas e desenvolvedores a construírem e implantarem seus aplicativos e infraestruturas de forma escalável e flexível.

⁸ <<https://www.netflix.com>>

2.3.5.1.1 Amazon Elastic Compute Cloud (Amazon EC2)

Conforme a documentação oficial da Amazon, o Amazon Elastic Compute Cloud (Amazon EC2) ⁹, fornece uma ampla variedade de opções de instâncias para atender às diferentes necessidades de carga de trabalho. Existem tipos de instâncias de computação geral, otimizadas para computação, otimizadas para memória, otimizadas para armazenamento e com computação acelerada, disponíveis para oferecer o equilíbrio ideal para suas cargas de trabalho.

2.3.6 Análise de Desempenho

Segundo [Nguyen, Kha e Hoang \(2017\)](#) a análise de desempenho é uma atividade realizada no campo da engenharia de software para avaliar e medir o desempenho de um sistema, aplicativo ou componente sob condições específicas. O objetivo é verificar como o sistema se comporta em termos de velocidade, escalabilidade, estabilidade e capacidade de resposta em diferentes cargas de trabalho. Durante o teste de desempenho, são simuladas condições reais de uso, como múltiplos usuários acessando simultaneamente, volume de dados elevado ou picos de tráfego. Essa análise sistemática permite identificar gargalos, pontos fracos e possíveis problemas de desempenho.

2.3.6.1 Apache JMeter

O JMeter ¹⁰, do grupo Apache, é uma aplicação desktop desenvolvida em Java, projetada para realizar testes funcionais e medir o desempenho de aplicações. Embora tenha sido inicialmente criado para testar aplicações web, seu uso se expandiu para outras funções de teste. O JMeter oferece uma ampla gama de recursos e funcionalidades, permitindo a simulação de diferentes cargas de trabalho, análise de desempenho, monitoramento de servidores, entre outros.

2.3.7 Figma

O Figma ¹¹ é uma ferramenta de design baseada em nuvem que é utilizada por designers de interface de usuário (UI) e de experiência do usuário (UX). O Figma é utilizado para criar wireframes, protótipos interativos e designs de interfaces de aplicativos e site, permitindo o acesso aos projetos em qualquer dispositivo, eliminando a necessidade de instalação de software adicional. O Figma se tornou uma escolha popular entre designers devido à sua usabilidade, eficiência e funcionalidades abrangentes.

⁹ <<https://aws.amazon.com/pt/ec2/>>

¹⁰ <<http://jmeter.apache.org/index.html>>

¹¹ <<https://www.figma.com/>>

3

Metodologia

Para o desenvolvimento de qualquer sistema, é fundamental compreender os requisitos aos quais o sistema deve atender e como ele deve se comportar. Por essa razão, neste trabalho, inicialmente foi descrito os requisitos funcionais e não funcionais utilizados como base para o desenvolvimento.

3.1 Requisitos

Nesta seção, são descritos os requisitos funcionais e não funcionais obrigatórios no sistema.

3.1.1 Requisitos Funcionais

O sistema apresenta os seguintes requisitos funcionais.

- Serviço de Autenticação
 - Registrar como prestador de serviço
 - Registrar como usuário
 - Logar como prestador de serviço
 - Logar como usuário
- Serviço de Usuário
 - Buscar um usuário
 - Buscar usuários
- Serviço de Prestador de Serviço
 - Buscar um prestador de serviço
 - Buscar prestadores de serviço
 - Atualizar informações de um prestador de serviço
- Serviço de Endereço
 - Criar um endereço como usuário
 - Buscar um endereço
 - Buscar endereços
 - Deletar um endereço cadastrado como usuário
- Serviço de Agendamento

- Criar um agendamento como usuário
- Buscar um agendamento
- Buscar agendamentos
- Atualizar informações de um agendamento

A tabela [3.1](#) descreve o que cada serviço permite:

Serviço	Funcionalidade	Descrição
Serviço de Autenticação	Registrar como prestador de serviço	Permite que um prestador de serviço se registre no sistema
	Registrar como usuário	Permite que um usuário se registre no sistema
	Logar como prestador de serviço	Permite que um prestador faça <i>login</i> no sistema
	Logar como prestador de serviço	Permite que um usuário faça <i>login</i> no sistema
Serviço de Usuário	Buscar um usuário	Permite buscar um específico usuário registrado no sistema
	Buscar usuários	Permite buscar mais de um usuário registrado no sistema
Serviço de Prestador	Buscar um prestador de serviço	Permite buscar um específico prestador registrado no sistema
	Buscar prestadores de serviço	Permite buscar mais de um prestador registrado no sistema
	Atualizar informações de um prestador de serviço	Permite um prestador atualizar suas informações
Serviço de Endereço	Criar um endereço como usuário	Permite um usuário criar um endereço
	Buscar um endereço	Permite buscar um específico endereço registrado no sistema
	Buscar endereços	Permite buscar mais de um endereço registrado no sistema
	Deletar um endereço cadastrado como usuário	Permite um usuário deletar um endereço cadastrado por ele
Serviço de Agendamento	Criar um agendamento como usuário	Permite um usuário criar um agendamento
	Buscar um agendamento	Permite buscar um específico agendamento registrado no sistema
	Buscar agendamentos	Permite buscar mais de um agendamento registrado no sistema
	Atualizar informações de um agendamento	Permite que o usuário ou prestador atualize as informações de um agendamento correspondente a ele

Tabela 3.1 – Tabela de serviços e funcionalidades. Fonte: Elaborado pelo autor

3.1.2 Requisitos Não Funcionais

3.1.2.1 Usabilidade

A usabilidade é um aspecto essencial para garantir a satisfação do usuário ao interagir com o sistema.

- **Facilitar para usuários leigos** - O sistema deve fornecer aos usuários facilidade na instalação, de forma que esta seja completamente automatizada.

3.1.2.2 Confiabilidade

A confiabilidade é a capacidade do sistema de desempenhar suas funções de forma correta e consistente, mesmo diante de situações adversas ou inesperadas.

- **Disponibilidade** - O sistema, junto de todas as suas funcionalidades e dependências, deverá ficar disponível 24 horas por dia, 7 dias por semana, porém, fatores externos como estado do servidor, tráfego de rede e outros eventuais problemas podem afetar essa disponibilidade.
- **Tempo de reparo** - Após uma possível falha ou *bug* ser comunicado aos desenvolvedores do sistema, estes serão corrigidos da forma mais rápida possível, de forma que o tempo de correção é dependente da gravidade do problema.
- **Controle de redundância** - O sistema deve possuir controle de redundância, de forma que os dados armazenados no banco de dados não sejam duplicados e nem inutilizados, garantindo assim mais velocidade ao sistema.

3.1.2.3 Manutenibilidade

A manutenibilidade diz respeito à facilidade com que o sistema pode ser mantido e evoluído ao longo do tempo.

- **Padrão de projeto** - O sistema será desenvolvido seguindo os padrões de desenvolvimento guiado por comportamento, ou BDD.
- **Dependências** - Todas as dependências adicionadas ao sistema quando em desenvolvimento devem garantir suporte aos desenvolvedores e ter bom conceito diante do mercado de softwares.
- **Compatibilidade mobile** - O sistema mobile deve ser compatível com as versões mais atuais do sistema Android. Deve garantir suporte a versões mais antigas destes sistemas operacionais, permitindo assim maior número de usuários do produto.

Após a descrição de todos os requisitos e para determinar a abordagem mais adequada, será desenvolvidas duas APIs: uma com arquitetura monolítica e outra com uma arquitetura orientada a microserviços. Com a realização dos Análise de Desempenho, será escolhida a

solução a ser utilizada em produção. Para alcançar esse objetivo, o processo de construção foi dividido em 6 etapas, que são as seguintes:

- a) Prototipação
- b) Banco de Dados
- c) Desenvolvimento
- d) Arquiteturas
- e) Infraestrutura e Hospedagem
- f) Análise de Desempenho

3.2 Prototipação

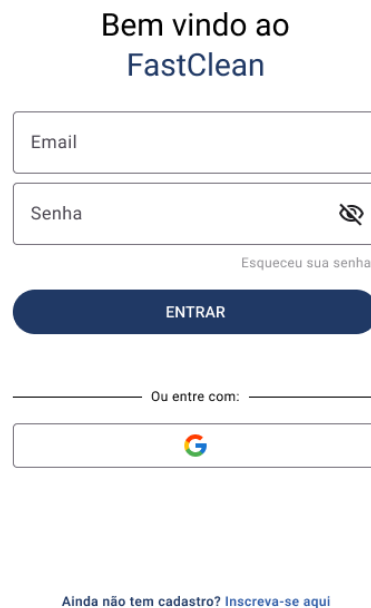
Com o objetivo de atender às necessidades tanto dos usuários quanto dos prestadores de serviços, foi planejada a criação de dois aplicativos distintos. Utilizando o software de design FIGMA, foram desenvolvidos os protótipos para ambos os aplicativos. Além disso, as duas APIs foram projetadas e implementadas de forma a suportar adequadamente os fluxos de telas correspondentes. Essas ações visam proporcionar uma experiência otimizada e satisfatória aos usuários, bem como facilitar o trabalho dos prestadores de serviços envolvidos.

3.2.1 Aplicativo dos Usuários

Para o Aplicativo dos Usuários contaremos com as principais funções: Login, Registro, Home e Agendamentos.


3.2.1.1 Login

Para atender ao requisito de logar como usuário, utilizaremos a tela mostrada na Figura 3.1.



Bem vindo ao
FastClean


Email

Senha 

[Esqueceu sua senha?](#)

ENTRAR

— Ou entre com: —



[Ainda não tem cadastro? Inscreva-se aqui](#)

Figura 3.1 – Protótipo de tela de Login do Usuário. Fonte: Elaborado pelo autor

A API realizará uma verificação dos valores inseridos, comparando-os com os dados armazenados no Banco de Dados, a fim de assegurar sua exatidão. Após essa verificação, a API retornará o respectivo Modelo de Usuário.

3.2.1.2 Registro

Para atender aos requisitos de registro de usuário e criação de endereço, a tela de registro será dividida em duas partes. Após a conclusão da primeira parte do registro, demonstrado na tela da Figura 3.2, o usuário será direcionado para a tela de adicionar um endereço, conforme demonstrado na Figura 3.3.



← Criar uma conta

Nome

Email

Senha

Confirme sua senha

CRIE SUA CONTA

Ou entre com:



[Você possui uma conta? Faça o login agora](#)

Figura 3.2 – Protótipo de tela de Registro do Usuário. Fonte: Elaborado pelo autor



Ola Gabriel lima!

Para continuar precisamos cadastrar seu endereço

Cep

Rua/Avenida

Numero/Apto *

Cidade Estado

Bairro

Complemento (Opcional)

Cadastrar

Figura 3.3 – Protótipo de tela de Registro Endereço do Usuário. Fonte: Elaborado pelo autor

A API será responsável por adicionar ao banco de dados de Usuários os valores correspondentes, além de realizar a criptografia da senha para que seja armazenada de forma

segura no Banco de Dados. Adicionará também ao banco de UserAddresses com os valores referentes ao endereço indicado.

3.2.1.3 Home

Para atender o requisito de buscar prestadores de serviços terá a tela *Home* Figura 3.4. Mostrando todos os Prestadores de serviços inscritos no sistema que estão disponíveis para agendamentos.

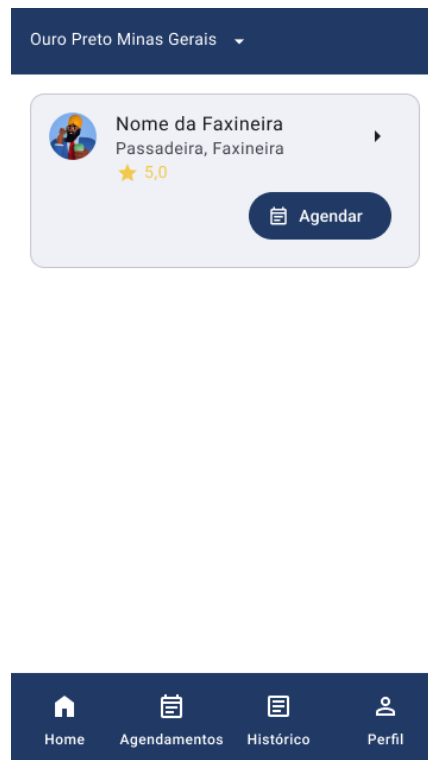


Figura 3.4 – Protótipo de tela Home. Fonte: Elaborado pelo autor

A Api retornará um vetor com todos os Prestadores registrados ou um vetor vazio caso nenhum ainda estiver registrado.

3.2.1.4 Agendamentos

Para atender o requisito de criar um agendamento como usuário na tela Home, conforme ilustrado na Figura 3.4, ao clicar no botão "Agendar", será enviado para a tela correspondente na Figura 3.5:

← Agendamento

Como é a sua residencia?

☒ Casa ☐ Apartamento

Sua casa possui quantos quartos?

Select de 1 a 10 quartos

Sua casa possui quantos banheiros?

Select de 1 a 10 banheiros

Confirmar informações

Figura 3.5 – Protótipo de tela de Agendamento - Primeiras Informações. Fonte: Elaborado pelo autor

Após preencher os dados necessários e clicar no botão localizado no final da tela da Figura 3.5, ocorrerá outro redirecionamento para a última etapa do processo de agendamento. Nessa tela, será preciso preencher as informações restantes, conforme demonstrado na Figura 3.6, a fim de fazer o pedido de agendamento.

← Agendamento

Resumo do serviço

Data e hora

Dia 14/04/2023 as 08:00 -12:00

Quantidade de comodors

4 banheiros 5 quartos

Endereço

Rua João Pedro da Silva, 2D
Morro do Cruzeiro, Ouro Preto - MG
Fica nas republica federais do campus
Apto 1001, casa D

Valor da Faxina: R\$ 100,00
Taxa de serviço: R\$ 10,00
Total do serviço: R\$ 110,00

Forma de pagamento

Mastercard **** 0000

Confirmar agendamento

Figura 3.6 – Protótipo de tela de Agendamento - Finalização. Fonte: Elaborado pelo autor

Após confirmar os dados ao clicar no botão localizado no final da tela da Figura 3.6, neste momento a API não possuirá nenhuma informação de formas e processamento de pagamentos. Portanto, a API deve realizar a verificação de agendamentos para a mesma data e período, garantindo que não haja nenhum agendamento conflitante. Após essa verificação, a API será responsável por calcular o valor a ser cobrado do usuário, o qual corresponde ao valor da faxina acrescido de 10%.

3.2.2 Aplicativo dos Prestadores de Serviço

Para o Aplicativo dos Prestadores de Serviço contaremos com as principais funções: Login, Registro, Perfil e Home.

3.2.2.1 Login

Para atender ao requisito de logar como prestador de serviço, utilizaremos a tela mostrada na Figura 3.1.

Bem vindo ao
FastClean

Email

Senha 

[Esqueceu sua senha?](#)

ENTRAR

— Ou entre com: —



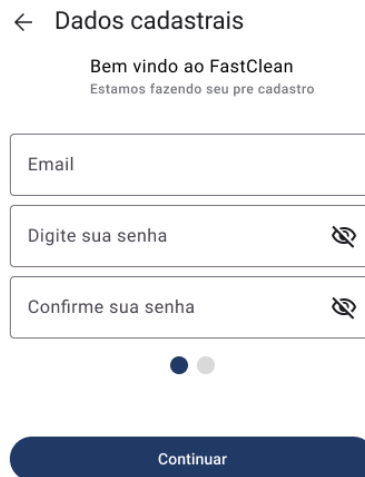
[Ainda não tem cadastro? Inscreva-se aqui](#)

Figura 3.7 – Protótipo de tela de Login dos Prestadores de Serviço. Fonte: Elaborado pelo autor

A API realizará uma verificação dos valores inseridos, comparando-os com os dados armazenados no Banco de Dados, a fim de assegurar sua exatidão. Após essa verificação, a API retornará o respectivo Modelo de Prestador.

3.2.2.2 Registro

Para atender aos requisitos registrar como prestador de serviço e atualizar informações de um prestador de serviço. A tela de registro foi dividida em duas partes: a primeira destinada à captura de e-mail e senha, conforme exemplificado na Figura 3.8.



← Dados cadastrais

Bem vindo ao FastClean
Estamos fazendo seu pre cadastro

Email

Digite sua senha

Confirme sua senha

Continuar

The image shows a registration form for 'FastClean'. It has a back arrow and the title 'Dados cadastrais'. Below is a welcome message: 'Bem vindo ao FastClean' and 'Estamos fazendo seu pre cadastro'. The form contains three input fields: 'Email', 'Digite sua senha' (with a toggle icon), and 'Confirme sua senha' (with a toggle icon). Below the fields are two progress dots, the first of which is filled. At the bottom is a dark blue 'Continuar' button.

Figura 3.8 – Tela de Registro dos Prestadores de Serviço - Etapa 1. Fonte: Elaborado pelo autor

Ao clicar no botão "Continuar", da tela na Figura 3.8, a API será responsável por adicionar ao banco de dados de Prestadores os valores correspondentes, além de realizar a criptografia da senha para que seja armazenada de forma segura no Banco de Dados. Caso o retorno da API seja um status de sucesso, o prestador será direcionado para a segunda etapa do registro, na qual preencherá as demais informações solicitadas na tela ilustrada na Figura 3.9.

← Dados cadastrais

Bem vindo ao FastClean
Estamos fazendo seu pre cadastro

Nome completo

Digite o seu cpf

Telefone

☐ Li e concordo com os [Termos e Condições de Uso.](#)

Cadastrar

Figura 3.9 – Protótipo de tela de Registro dos Prestadores de Serviço - Etapa 2. Fonte: Elaborado pelo autor

Após o preenchimento dos dados e ao clicar no botão "Cadastrar", a API deverá realizar a verificação da validade do CPF e garantir que não exista outro prestador cadastrado com o mesmo CPF.

3.2.2.3 Perfil

Após concluir a etapa de registro, o Prestador será redirecionado para a tela de perfil conforme representada na Figura 3.10, onde poderá fornecer as informações finais necessárias para o pleno funcionamento do sistema, esta etapa ainda se refere ao requisito de atualizar informações de um prestador de serviço.



← Informações adicionais

Adicione uma foto

Cidade UF

Minhas Habilidades

☒ Faxina Comercial ☒ Faxina Residencial

☒ Lavar ☒ Passar

Dias da semana disponíveis

S T Q Q S S D

Horario disponíveis

Adicionar horario disponível

Valor a ser cobrado na faxina

R\$120,00

Cadastrar

Figura 3.10 – Protótipo de tela de Perfil dos Prestadores de Serviço. Fonte: Elaborado pelo autor

Neste primeiro estágio, o envio da foto de perfil não será utilizado, uma vez que não será armazenado durante o teste. Após o preenchimento e ao clicar no botão "Cadastrar", a API deverá concluir o processo de salvamento dos últimos dados capturados no modelo de Prestadores de Serviços.

3.2.2.4 Home

Para atender aos requisitos de buscar um endereço, buscar um usuário, buscar agendamentos e atualizar informações de um agendamento teremos a Home do prestado.

Logo após o prestador realizar o *login* ou o registro, o Prestador será direcionado para a "Home - Solicitações", conforme demonstrado na tela representada na Figura 3.11.

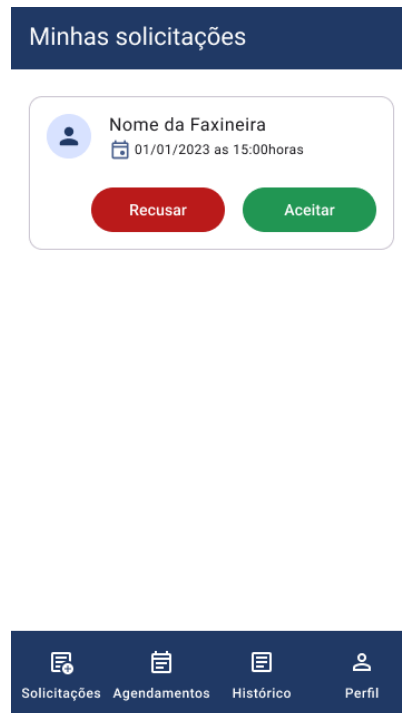


Figura 3.11 – Protótipo de tela Home - Solicitações dos Prestadores de Serviço. Fonte: Elaborado pelo autor

A API, retornará um vetor vazio ou um vetor contendo as solicitações de agendamento com status "PENDENTE", e o "Prestador" terá a opção de aceitar ou recusar o agendamento. O prestador terá acesso aos agendamentos que estão confirmados, como demonstrado na tela da Figura 3.12

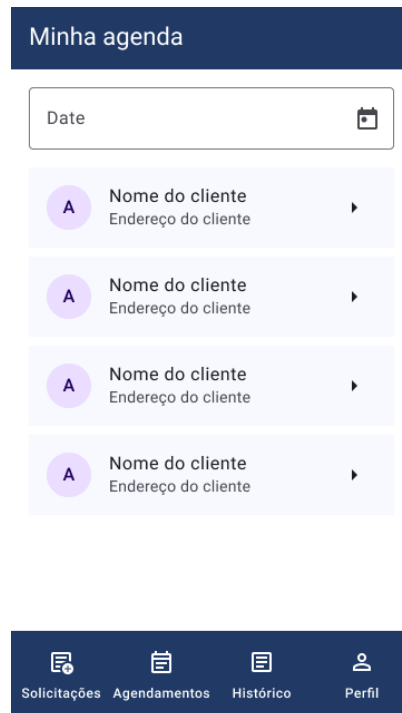


Figura 3.12 – Protótipo de tela Home - Minha Agenda dos Prestadores de Serviço. Fonte: Elaborado pelo autor

A API, retornará um vetor vazio ou um vetor os agendamentos já aceitos, ordenado pela data do agendamento em ordem crescente.

3.3 Banco de Dados

Para o desenvolvimento do Banco de Dados, foi utilizado o banco de dados NoSQL MongoDB. Por ser capaz de lidar com grandes volumes de dados, suportar cargas de trabalho intensas e apresentar eficiência em consultas complexas, o MongoDB proporcionará um desempenho otimizado no acesso aos dados.

Então foi necessário modelar a base de dados completa da API, onde as duas Arquiteturas utilizaram a mesma modelagem para fazer suas requisições, logo terá os seguintes modelos:

- Users: Usuários
- UserAddresses: Endereço dos Usuários
- Cleaners: Prestadores de Serviço
- Roles: Regras
- Permissions: Permissões
- UserRoles: Regras de Usuários
- CleanerRoles: Regras de Prestadores de Serviço

- Schedules: Agendamentos

Tanto o identificador único (ID) quanto as datas de criação e atualização dos modelos são realizadas automaticamente, não sendo necessário adicioná-las nos modelos. A Figura 3.13 demonstra como o banco de dados foi concebido e como as entidades se comunicam entre si.

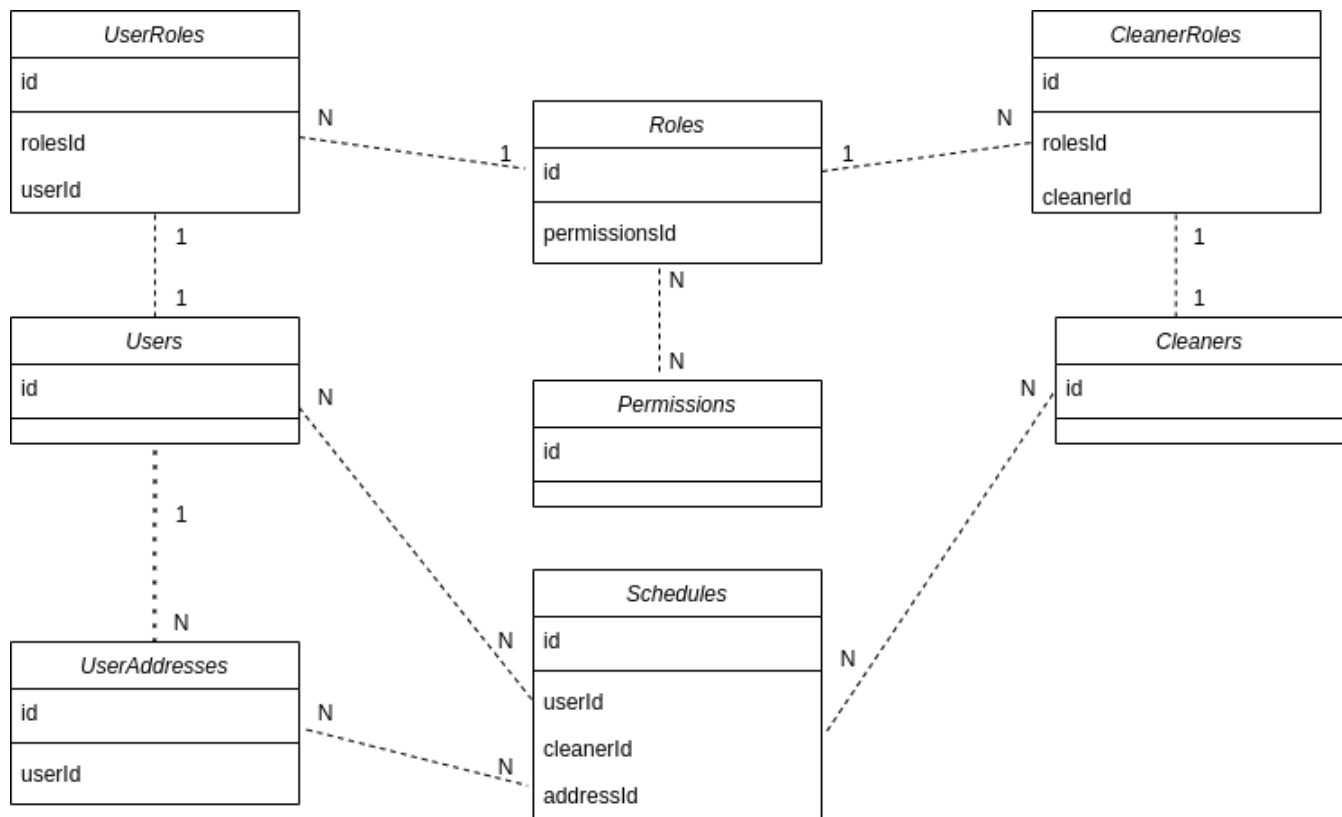


Figura 3.13 – Relações das entidades do Banco de Dados. Fonte: Elaborado pelo autor

3.3.1 Users

Como demonstra a Figura 3.14 o modelo de Users possui:

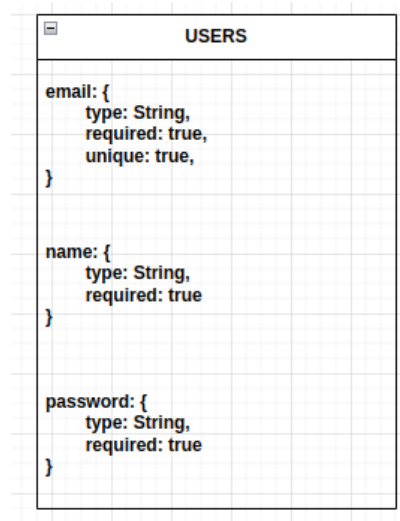


Figura 3.14 – Modelo de Usuários. Fonte: Elaborado pelo autor

- email: Email do Usuário do tipo String, obrigatório e único para cada Usuário
- name: Nome do Usuário do tipo String, obrigatório
- password: Senha do Usuário do tipo String, obrigatório. A senha será uma string criptografada.

3.3.2 UserAddresses

Como demonstra a Figura 3.15 o modelo de UserAddresses possui:

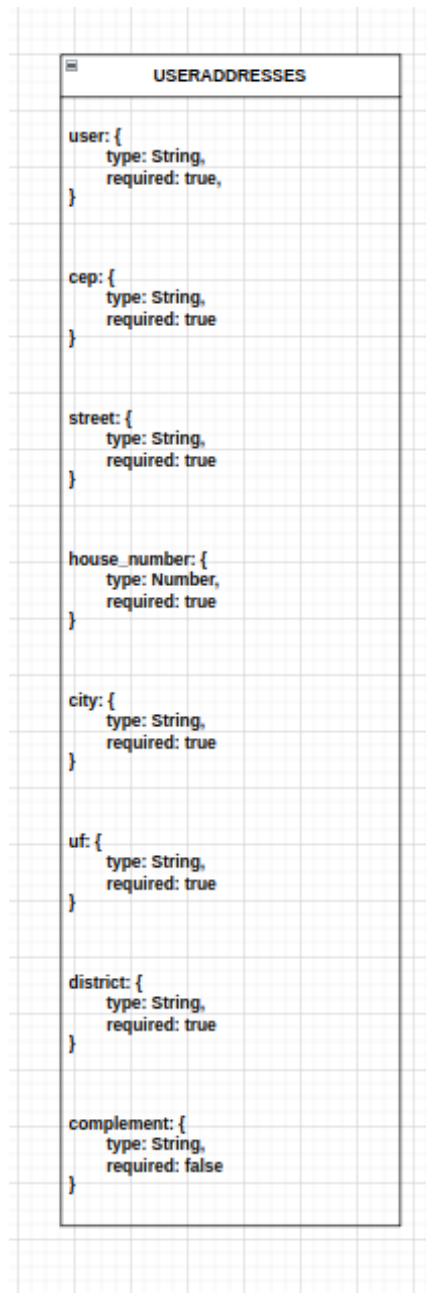


Figura 3.15 – Modelo de Endereço dos Usuários. Fonte: Elaborado pelo autor

- user: Id do Usuário do tipo String, obrigatório
- cep: Cep do Endereço do tipo String, obrigatório
- street: Rua do Endereço do tipo String, obrigatório
- house_number: Numero da Casa do tipo Number, obrigatório
- city: Cidade do Endereço do tipo String, obrigatório
- uf: UF do Estado do Endereço do tipo String, obrigatório
- district: Bairro do Endereço do tipo String, obrigatório
- complement: Complemento do Endereço do tipo String.

3.3.3 Cleanears

Como demonstra a Figura [3.16](#) o modelo de Cleanears possui:

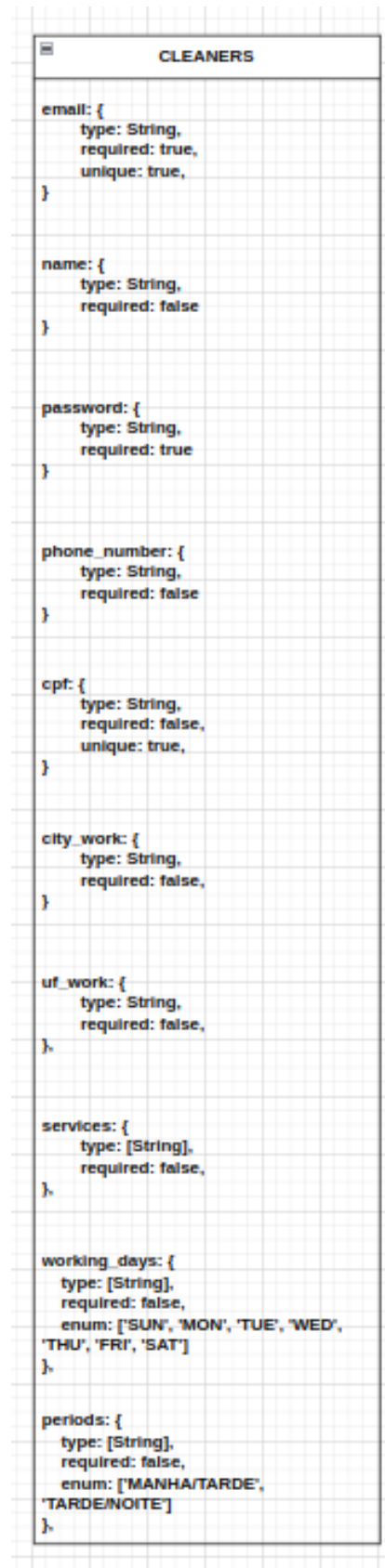


Figura 3.16 – Modelo de Prestadores de Serviço. Fonte: Elaborado pelo autor

- email: Email do Prestador de Serviço do tipo String, obrigatório e único para cada

Prestador

- name: Nome do Prestador de Serviço do tipo String
- password: Senha do Prestador de Serviço do tipo String, obrigatório. A senha será uma String criptografada.
- phone_number: Numero do Celular do Prestador de Serviço do tipo String
- cpf: CPF do Prestador de Serviço do tipo String e único para cada Prestador
- city_work: Cidade em que Prestador de Serviço atende do tipo String
- uf_work: UF do Estado em que Prestador de Serviço atende do tipo String
- services: Array de Strings, contendo o tipo de serviço que o Prestador de Serviço realiza. Ex: ['FAXINA', 'PASSAR ROUPA']
- working_days: Array de Strings, contendo os dias da semana que o Prestador de Serviço trabalha. Sendo os valores da String limitados em : ['SUN', 'MON', 'TUE', 'WED', 'THU', 'FRI', 'SAT']
- periods: Array de Strings, contendo os periodos do dia que o Prestador de Serviço trabalha. Sendo os valores da String limitados em : ['MANHA/TARDE', 'TARDE/NOITE']

3.3.4 Roles

Como demonstra a Figura 3.17 o modelo de Roles possui:

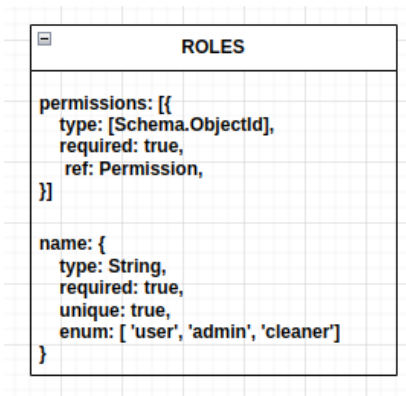


Figura 3.17 – Modelo de Regras. Fonte: Elaborado pelo autor

- permissions: Array de ObjectsId, contendo os ids do modelo de Permissions
- name: Nome da Regra do tipo String, obrigatório e único. Sendo os valores da String limitados em: ['user', 'cleaner', 'admin']

3.3.5 Permissions

Como demonstra a Figura 3.18 o modelo de Permissions possui:

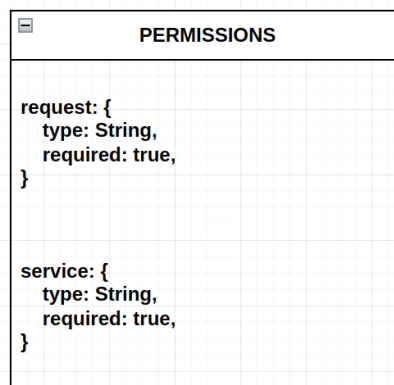


Figura 3.18 – Modelo de Permissões. Fonte: Elaborado pelo autor

- request: Nome da Requisição do tipo String e obrigatório.
- service: Nome do Serviço da Requisição do tipo String e obrigatório

3.3.6 UserRoles

Como demonstra a Figura 3.19 o modelo de UserRoles possui:

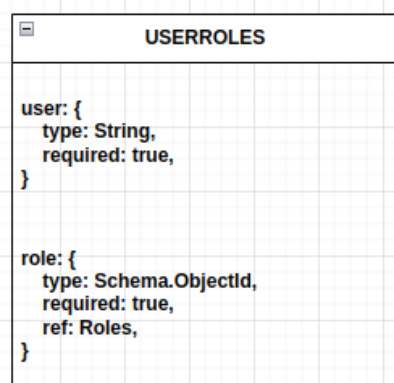


Figura 3.19 – Modelo de Regras do Usuário. Fonte: Elaborado pelo autor

- role: Id do modelo de Roles do tipo ObjectId e obrigatório.
- user: Id do Usuário do tipo String e obrigatório

3.3.7 CleanerRoles

Como demonstra a Figura 3.20 o modelo de CleanerRoles possui:

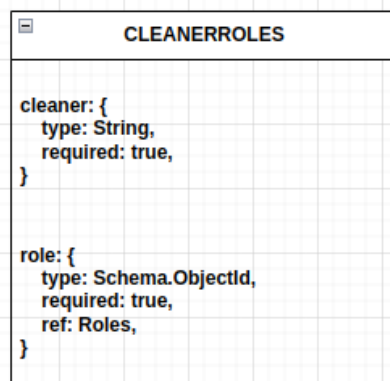


Figura 3.20 – Modelo de Regras dos Prestadores de Serviço. Fonte: Elaborado pelo autor

- role: Id do modelo de Roles do tipo ObjectId e obrigatório.
- cleaner: Id do Prestador do tipo String e obrigatório

3.3.8 Schedules

Como demonstra a Figura 3.21 o modelo de Schedules possui:

SCHEDULES
<pre>cleaner: { type: String, required: true, } user: { type: String, required: true, } address: { type: String, required: true, } status: { type: [String], enum: ['PENDING', 'ACCEPTED', 'REJECTED', 'CONCLUDED', 'CANCELLED'], default: 'PENDING', } services: { type: [String], required: false, }, date: { type: Date, required: true, }, periods: { type: [String], required: false, enum: ['MANHA/TARDE', 'TARDE/NOITE'] }, value: { type: Number, required: true, }, tax_total: { type: Number, default: 0.1, }, value_total: { type: Number, required: false, }, tax_tarrif: { type: Number, default: 0.03, }, value_tarrif: { type: Number, default: 0, }, value_retain: { type: Number, default: 0, },</pre>

Figura 3.21 – Modelo de Agendamentos. Fonte: Elaborado pelo autor

- cleaner: Id do Prestador do tipo String e obrigatório

- user: Id do Usuário do tipo String e obrigatório.
- address: Id do Endereço do tipo String e obrigatório.
- status: Status do Agendamento do tipo String, obrigatório. Sendo os valores da String limitados em: ['PENDING', 'ACCEPTED', 'REJECTED', 'CONCLUDED', 'CANCELLED']
- services: Array de Strings, contendo o tipo de serviço que será realizado no agendamento. Ex: ['FAXINA', 'PASSAR ROUPA']
- address: Dia do Agendamento do tipo Date e obrigatório.
- period: Período do dia que será agendado do tipo String. Sendo os valores da String limitados em : ['MANHA/TARDE', 'TARDE/NOITE']
- value: Valor cobrado pelo Prestador de Serviço do tipo Number e obrigatório.
- tax_total: Taxa total que será cobrada do Usuário do tipo Number, com valor inicial de 10%.
- value_total: Valor que será cobrado do Usuário do tipo Number, sendo: $value_total = value + (value * tax_total)$.
- tax_tarrif: Valor da taxa cobrado normalmente por sistemas de pagamento do tipo Number, com valor inicial de 3%
- value_tarrif: Valor que será repassado para o sistema de pagamento do tipo Number, com valor inicial de igual a 0. Sendo o $value_tarrif = value_total * tax_tarrif$.
- value_retain: Valor retido pelo agendamento do tipo Number, com valor inicial de 0. Sendo $value_retain = value_total - value - value_tarrif$.

3.4 Desenvolvimento

Para o desenvolvimento foi utilizado NodeJs, com o framework Express, por oferecer eficiência, escalabilidade, facilidade de desenvolvimento, uma vasta comunidade de suporte e uma rica biblioteca de pacotes, tornando o desenvolvimento uma tarefa mais ágil e eficaz.

No projeto, o backend foi estruturado em serviços distintos, cada um responsável por uma função específica. Dessa forma, foi possível visualizar e compreender claramente o caminho da rota, a requisição e a resposta associadas a cada função. Além disso, é fundamental que as duas APIs sejam capazes de processar as mesmas funções, garantindo assim a consistência e padronização do sistema, a partir dos seguintes serviços:

- AuthenticationService
- AuthorizationService
- UsersService

- UserAddressesService
- CleanersService
- RolesService
- PermissionsService
- UserRolesService
- CleanerRolesService
- SchedulesService

Cada serviço utilizou os verbos padrões HTTP que são:

- GET: recupera um ou mais recursos.
- POST: Cria novos recursos.
- PUT: Atualiza um conteúdo.
- DELETE: Elimina um recurso

3.4.1 AuthenticationService

O serviço de AuthenticationService ficou responsável por toda a autenticação, ou seja, lidou com os registros e logins

3.4.1.1 userLogin

- METHOD: POST
- URL: /auth/users/login
- BODY: email: String, password: String,
- RESPONSE: token: String, user: Modelo de Usuário

Foi realizada a chamada da função "login" do serviço UserService, responsável por comparar os dados fornecidos com os dados armazenados. Após essa verificação, o objeto "user" retornado, permitindo a geração do JWT Token com base no id desse usuário. Dessa forma, sendo possível garantir a segurança e autenticação adequadas no sistema.

3.4.1.2 userSignup

- METHOD: POST
- URL: /auth/users/signup
- BODY: email: String, password: String, name: String
- RESPONSE: token: String, user: Modelo de Usuário

Foi realizada a chamada da função "create" do serviço UserService, responsável por criar um novo usuário. Em seguida, o objeto "user" retornado, permitindo a geração do JWT Token com base no id desse usuário. Dessa forma, foi possível garantir a segurança e autenticação adequadas no sistema.

3.4.1.3 cleanerLogin

- METHOD: POST
- URL: /auth/users/login
- BODY: email: String, password: String,
- RESPONSE: token: String, user: Modelo de Usuário

Foi realizada a chamada da função "login" do serviço UserService, responsável por comparar os dados fornecidos com os dados armazenados. Após essa verificação, o objeto "cleaner" retornado, permitindo a geração do JWT Token com base no id desse prestador. Dessa forma, foi possível garantir a segurança e autenticação adequadas no sistema.

3.4.1.4 cleanerSignup

- METHOD: POST
- URL: /auth/users/signup
- BODY: email: String, password: String, name: String
- RESPONSE: token: String, cleaner: Modelo de Prestadores de Serviço

Foi realizada a chamada da função "create" do serviço ClenearService, responsável por criar um novo prestador. Em seguida, o objeto "cleaner" retornado, permitindo a geração do JWT Token com base no id desse prestador. Dessa forma, foi possível garantir a segurança e autenticação adequadas no sistema.

3.4.2 AuthorizationService

O AuthorizationService será responsável pela autorização do sistema.

3.4.2.1 checkUserPermission

- METHOD: POST
- URL: /authorization/users, sendo apenas uma chamada interna do sistema
- QUERY: user: String (id do usuário), service: String, request: String
- RESPONSE: permission: String. Sendo permission: 'allowed' ou permission: 'denied'

Esta função, por meio do id do usuario ("user"), buscou se o mesmo possui uma regra ativa dentro do UserRole. Em seguida, utilizando parametros service e request, realizando uma busca dentro de Permission para encontrar o ID da permissão solicitada. Após essa busca, verificando se o usuário possui a permissão desejada, onde dentro de "role" de UserRoles já populado buscará no array de permissions se encontra o id da "permission". Se for o caso, retorna que o usuário tem permissão para a requisição. Caso contrário, o acesso será negado.

3.4.2.2 checkCleanerPermission

- METHOD: POST
- URL: /authorization/cleaners, sendo apenas uma chamada interna do sistema
- QUERY: cleaner: String (id do prestador), service: String, request: String
- RESPONSE: permission: String. Sendo permission: 'allowed' ou permission: 'denied'

Esta função, por meio do id do prestador ("cleaner"), buscará se o mesmo possui uma regra ativa dentro do CleanerRole. Em seguida, utilizando parametros service e request, será realizada uma busca dentro de Permission para encontrar o ID da permissão solicitada. Após essa busca, será verificado se o usuário possui a permissão desejada, onde dentro de "role" de CleanerRoles já populado buscará no array de permissions se encontra o id da "permission". Se for o caso, será retornado que o usuário tem permissão para a requisição. Caso contrário, o acesso será negado.

3.4.3 UsersService

O UserService será responsável por todas as funcionalidades relacionadas ao gerenciamento de usuários.

3.4.3.1 login

- METHOD: POST
- URL: /users/login, sendo apenas uma chamada interna do sistema
- BODY: email: String, password: String
- RESPONSE: user: Modelo de Usuários.

Responsável por comparar os dados fornecidos com os dados armazenados. Para o caso dos dados realmente estiverem corretos retornar o modelo de "users" encontrado. Em caso dos dados não corresponderem retornará erro.

3.4.3.2 create

- METHOD: POST
- URL: /users, sendo apenas uma chamada interna do sistema
- BODY: email: String, password: String, name: String
- RESPONSE: user: Modelo de Usuários.

Responsável por criar um novo usuário através dos dados fornecidos no corpo da requisição, também chamará a função "create" de UserRolesService e criar uma nova regra de usuário, no caso de sucesso retornar o modelo do usuário gerado.

3.4.3.3 find

- METHOD: GET
- URL: /users
- RESPONSE: [user: Modelo de Usuários].

Responsável por fazer uma busca de usuários no sistema, retornando um vetor contendo todos os usuários encontrados ou um vetor vazio caso nada for encontrado.

3.4.3.4 findById

- METHOD: GET
- URL: /users/:id
- PARAMS: id: Id do usuário
- RESPONSE: user: Modelo de Usuários.

Responsável por fazer uma busca pelo id do usuário no sistema, retornando o usuário correspondente ou um erro de nada encontrado.

3.4.4 UserAddressesService

O serviço de UserAddressesService será responsável por todas as funcionalidades relacionadas ao gerenciamento de endereços de usuários.

3.4.4.1 create

- METHOD: POST
- URL: /users/:id/addresses
- PARAMS: id: Id do usuário

- BODY: cep: String, street: String, house_number: Number, city: String, uf: String, district: String, complement: String,
- RESPONSE: address: Modelo de Endereços.

Responsável por criar um novo endereço através dos dados fornecidos no corpo da requisição em caso de sucesso retornar o modelo do usuário gerado.

3.4.4.2 find

- METHOD: GET
- URL: /users/:id/addresses
- PARAMS: id: Id do usuário
- RESPONSE: [address: Modelo de Endereços].

Responsável por fazer uma busca de endereços no sistema, retornando um vetor contendo todos os endereços do usuário ou um vetor vazio caso nada for encontrado.

3.4.4.3 findById

- METHOD: GET
- URL: /users/:id/addresses/:addressId
- PARAMS: id: id do Usuário , addressId: id do Endereço
- RESPONSE: address: Modelo de Endereços

Responsável por fazer uma busca pelo "addressId" id do endereço no sistema, retornando o endereço correspondente ou um erro de nada encontrado.

3.4.4.4 destroy

- METHOD: GET
- URL: /users/:id/addresses/:addressId
- RESPONSE: address: Modelo de Endereços

Responsável por fazer uma busca pelo "addressId" id do endereço no sistema, deletando do banco de dados e retornando o endereço correspondente ou um erro em caso de falha na busca ou de deletar.

3.4.5 CleanersService

O CleanersService será responsável por todas as funcionalidades relacionadas ao gerenciamento de prestadores de serviço.

3.4.5.1 login

- METHOD: POST
- URL: /cleaners/login, sendo apenas uma chamada interna do sistema
- BODY: email: String, password: String
- RESPONSE: cleaner: Modelo do Prestador.

Responsável por comparar os dados fornecidos com os dados armazenados. Para o caso dos dados realmente estiverem corretos retornar o modelo de "cleaners" encontrado. Em caso dos dados não corresponderem retornará erro.

3.4.5.2 create

- METHOD: POST
- URL: /cleaners, sendo apenas uma chamada interna do sistema
- BODY: email: String, password: String,
- RESPONSE: cleaner: Modelo do Prestador.

Responsável por criar um novo prestador através dos dados fornecidos no corpo da requisição, também chamar a função "create" de `CLeanerRolesService` e criar uma nova regra de prestador, em caso de sucesso retornar o modelo do prestador gerado.

3.4.5.3 find

- METHOD: GET
- URL: /cleaners
- RESPONSE: [cleaner: Modelo do Prestador].

Responsável por fazer uma busca de prestadores de serviço no sistema, retornando um vetor contendo todos os prestadores encontrados ou um vetor vazio caso nada for encontrado.

3.4.5.4 findById

- METHOD: GET
- URL: /cleaners/:id
- PARAMS: id: Id do prestador
- RESPONSE: cleaner: Modelo do Prestador.

Responsável por fazer uma busca pelo id do prestador no sistema, retornando o prestador correspondente ou um erro de nada encontrado.

3.4.5.5 update

- METHOD: PUT
- URL: /cleaneers/:id
- PARAMS: id: Id do prestador
- BODY: name: String, phone_number: String, cpf: String, city_work: String, uf_work: String, services: [String], working_days: [String] enum: ["SUN", "MON", "TUE", "WED", "THU", "FRI", "SAT"], periods: String enum: ["MANHA/TARDE", "TARDE/NOITE"]
- RESPONSE: cleaner: Modelo do Prestador.

A função em questão tem a responsabilidade de localizar o prestador correspondente por meio do seu ID e atualizar os valores deste prestador com base nos dados enviados no corpo da requisição. E retornar o modelo de cleaner com os valores atualizados.

3.4.6 RolesService

O RolesService será responsável por todas as funcionalidades relacionadas ao gerenciamento de regras. Sendo assim, todas as regras serão adicionadas manualmente no banco de dados.

3.4.6.1 find

- METHOD: GET
- URL: /roles, sendo apenas uma chamada interna do sistema
- QUERY: name: String
- RESPONSE: [role: Modelo de Regras].

Responsável por fazer uma busca de regras no sistema, retornando um vetor contendo todas as regras com o nome correspondente ou um vetor vazio caso nada for encontrado.

3.4.6.2 findById

- METHOD: GET
- URL: /roles/:id, sendo apenas uma chamada interna do sistema
- PARAMS: id: Id da regras
- RESPONSE: role: Modelo de Regras.

Responsável por fazer uma busca pelo id de regras no sistema, retornando a regra correspondente ou um erro de nada encontrado.

3.4.7 PermissionsService

O PermissionsService será responsável por todas as funcionalidades relacionadas ao gerenciamento de permissões. Sendo assim, todas as permissões serão adicionadas manualmente no banco de dados.

3.4.7.1 find

- METHOD: GET
- URL: /permissions/:id , sendo apenas uma chamada interna do sistema
- QUERY: service: String, request: String
- PARAMS: id: Id da Permission
- RESPONSE: [permission: Modelo de Permissions].

Responsável por fazer uma busca de permissões no sistema, retornando um vetor contendo todas as regras com o "service" ou "request" correspondente ou um vetor vazio caso nada for encontrado.

3.4.7.2 findById

- METHOD: GET
- URL: /permissions/:id, sendo apenas uma chamada interna do sistema
- PARAMS: id: Id do prestador
- RESPONSE: permission: Modelo de Permissions.

Responsável por fazer uma busca pelo id de regras no sistema, retornando a permissão correspondente ou um erro de nada encontrado.

3.4.8 UserRolesService

O UserRolesService será responsável por todas as funcionalidades relacionadas ao gerenciamento de regras de usuário.

3.4.8.1 create

- METHOD: POST
- URL: /user-roles, sendo apenas uma chamada interna do sistema
- BODY: user: String (id do usuário), role: String (id da regra),
- RESPONSE: userRole: Modelo de Regras de Usuário

Responsável por criar uma nova regra de usuário através dos dados fornecidos no corpo da requisição em caso de sucesso retornar o modelo de regras de usuário gerado.

3.4.8.2 find

- METHOD: GET
- URL: /user-roles, sendo apenas uma chamada interna do sistema
- QUERY: user: String (id do usuário)
- RESPONSE: [userRole: Modelo de Regras de Usuário].

Responsável por fazer uma busca de regras de usuário no sistema, retornando um vetor contendo todas as regras de usuários correspondente a "user" ou um vetor vazio caso nada for encontrado.

3.4.9 CleanerRolesService

O CleanerRolesService será responsável por todas as funcionalidades relacionadas ao gerenciamento de regras de prestadores de serviços.

3.4.9.1 create

- METHOD: POST
- URL: /user-roles, sendo apenas uma chamada interna do sistema
- BODY: user: String (id do usuário), role: String (id da regra),
- RESPONSE: cleanerRole: Modelo de Regras de Prestadores

Responsável por criar uma nova regras de prestadores de serviços. através dos dados fornecidos no corpo da requisição em caso de sucesso retornar o modelo de regras de prestador gerado.

3.4.9.2 find

- METHOD: GET
- URL: /user-roles, sendo apenas uma chamada interna do sistema
- QUERY: cleaner: String (id do usuário)
- RESPONSE: [cleanerRole: Modelo de Regras de Prestadores].

Responsável por fazer uma busca de regras de prestadores de serviços no sistema, retornando um vetor contendo todas as regras de prestadores de serviços correspondente a "cleaner" ou um vetor vazio caso nada for encontrado.

3.4.10 SchedulesService

O serviço de UserAddressesService será responsável por todas as funcionalidades relacionadas ao gerenciamento de endereços de usuários.

3.4.10.1 create

- METHOD: POST
- URL: /schedules
- BODY: user: String, cleaner: String, services: [String], address: String, date: Date, period: String, value: Number,
- RESPONSE: schedule: Modelo de Agendamento

Responsável por criar um novo agendamento através dos dados fornecidos no corpo da requisição em caso de sucesso retornar o modelo de agendamento gerado.

3.4.10.2 find

- METHOD: GET
- URL: /schedules
- QUERY: user: String, cleaner: String, status: String
- RESPONSE: [schedule: Modelo de Agendamento].

Responsável por fazer uma busca de agendamentos no sistema, retornando um vetor contendo todos os agendamentos que corresponde aos dados enviados pela "query" ou um vetor vazio caso nada for encontrado. Em caso de sucesso, sendo a busca for feita por um usuário, deverá popular o "cleaner" invocando a função "findById" de CleanersService. Caso contrário for uma busca feita por prestador, deverá popular o "user" invocando a função "findById" de UsersService

3.4.10.3 findById

- METHOD: GET
- URL: /schedules/:id
- PARAMS: id: id do agendamento
- RESPONSE: schedule: Modelo de Agendamento

Responsável por fazer uma busca pelo id do agendamento no sistema, retornando o agendamento correspondente ou um erro de nada encontrado. Em caso de sucesso, sendo a busca

for feita por um usuário, deverá popular o "cleaner" invocando a função "findById" de CleanersService. Caso contrário for uma busca feita por prestador, deverá popular o "user" invocando a função "findById" de UsersService

3.4.10.4 update

- METHOD: PUT
- URL: /cleaneers/:id
- PARAMS: id: Id do prestador
- BODY:
 - status: String. Sendo os valores da String limitados em: ['ACCEPTED', 'REJECTED', 'CANCELLED']
- RESPONSE: schedule: Modelo de Agendamento.

A função em questão tem a responsabilidade de localizar o agendamento correspondente por meio do seu "ID" e atualizar os valores deste agendamento com base nos dados enviados no corpo da requisição e retornar o modelo de agendamento com os valores atualizados.

3.5 Arquiteturas

Após a definição dos serviços presentes nas APIs, foi realizado o projeto das duas arquiteturas propostas: a Monolítica e a baseada em Microserviços.

3.5.1 Arquitetura Monolítica

A API Monolítica prevê que todos os serviços fiquem em um mesmo local, ou seja todos os processos funcionam em um único servidor, conforme mostra a Figura 3.22

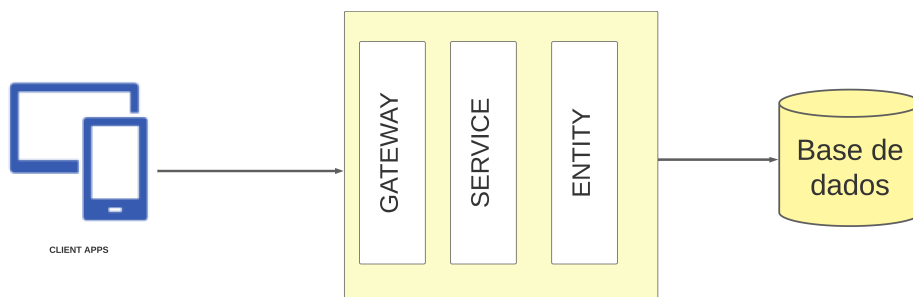


Figura 3.22 – Arquitetura Monolítica. Fonte: Elaborado pelo autor

3.5.2 Arquitetura de Microserviços

A API Microserviços é composta por um gateway responsável por encaminhar as requisições para o serviço correto, permitindo que cada processo funcione de forma paralela. Para atingir esse objetivo, os serviços foram divididos da seguinte maneira:

- AuthenticationService
- AuthorizationService
 - AuthorizationService
 - PermissionsService
 - RolesService
 - UserRolesService
 - CleanerRolesService
- UsersService
 - UsersService
 - UserAddressesService
- CleanersService
- SchedulesService

Desta maneira AuthorizationService e UsersService acloparam mais de um serviço como uma das suas responsabilidades, sendo assim temos a arquitetura demonstrada na Figura [3.23](#).

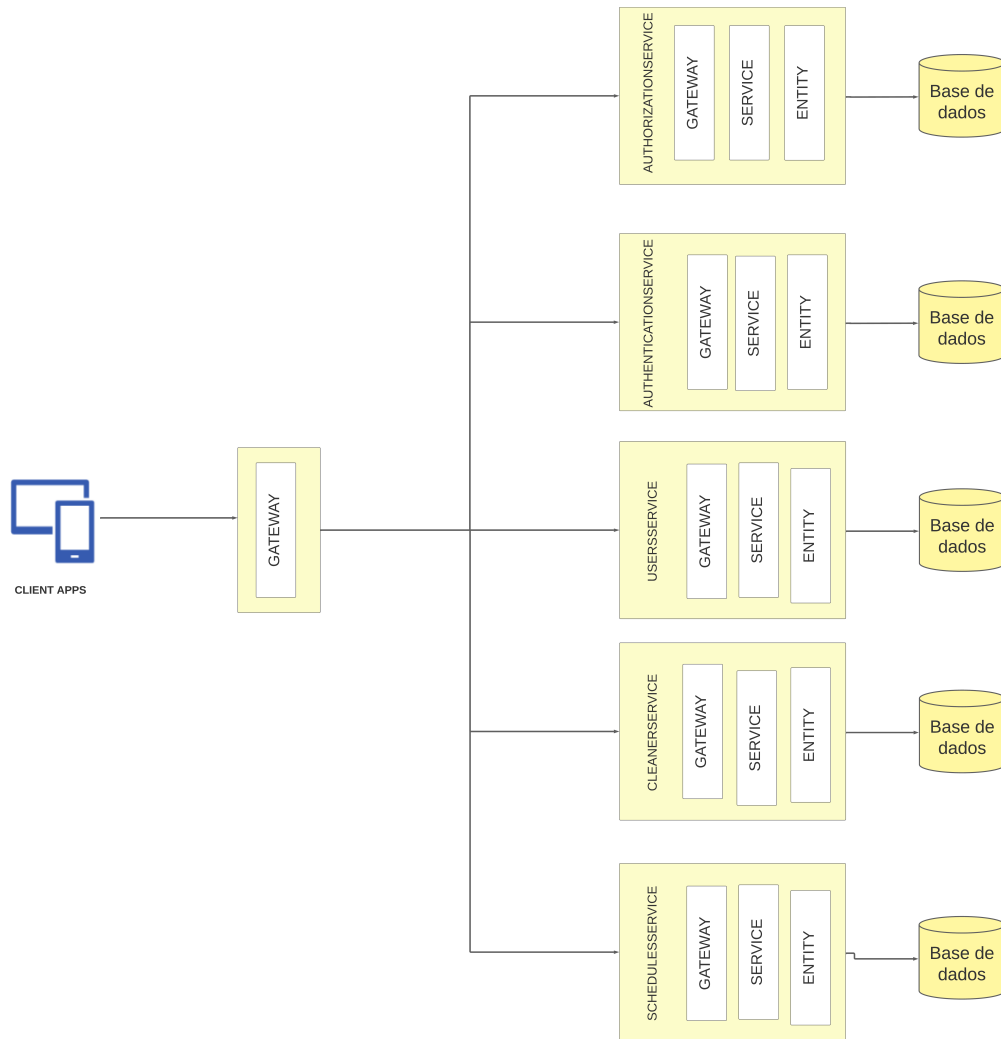


Figura 3.23 – Arquitetura de Microserviços. Fonte: Elaborado pelo autor

Como indicado na Figura 3.23, cada serviço possui bancos de dados separados, ou seja, nenhum dos outros tem conhecimento ou acesso a outra base de dados, exceto a sua.

3.6 Infraestrutura

Para a disponibilização das APIs, utiliza-se os serviços da Amazon Web Services (AWS).

Na arquitetura monolítica, utilizaremos a EC2, que consiste em um único servidor conectado e atende as necessidades do sistema, o qual não requer uma estrutura distribuída complexa.

Na arquitetura de microserviços, em que cada serviço é separado e tratado de forma independente, cada um é hospedado em uma instância EC2 distinta.

Para esta monografia, as duas arquiteturas possuem disponíveis a quantidade de 16 GB de memória RAM, cada.

Na arquitetura monolítica, utiliza-se uma única máquina com 16 GB sendo ela uma t3.xlarge com 4 núcleos e performance de rede em até 5 Gigabits por segundo (Gbps). Já na arquitetura

de microsserviços, a distribuição é feita de acordo com a demanda, atribuindo maior capacidade de processamento aos serviços mais chamados. A distribuição proposta é a seguinte:

- Gateway: t2.small 2 GB 1 núcleo e performance de rede baixa moderada
- AuthenticationService: t2.small 2 GB 1 núcleo e performance de rede baixa moderada
- AuthorizationService: t2.small 2 GB 1 núcleo e performance de rede baixa moderada
- UsersService: t2.medium 4 GB 2 núcleos e performance de rede baixa moderada
- CleanersService: t2.medium 4 GB 2 núcleos e performance de rede baixa moderada
- SchedulesService: t2.small 2 GB 1 núcleo e performance de rede baixa moderada

3.7 Análise de Desempenho

Para a Análise de Desempenho, utiliza-se o Apache JMeter com uma carga de trabalho consistindo em uma taxa constante de requisições. Dessa forma, replica-se um fluxo que o usuário pode percorrer no sistema, incorporando as rotinas que são executadas em cada thread de usuário, as quais incluem:

- User Register: POST /auth/users/signup
- User Login: POST /auth/users/login
- User Addresses: POST /users/:id/addresses
- Cleaner Register: POST /auth/cleaners/signup
- Cleaner Login: POST /auth/users/login
- Update Cleaner: PUT /cleaners
- Find Users: GET /users
- Find Cleaners: GET /cleaners
- FindById Users: GET /users/:id
- FindbyId Cleaners: /cleaners/:id
- Find UserAddresses: GET /users/:id/addresses
- FindById UserAddresses: GET /users/:id/addresses/:addressId
- Create Schedule: POST /schedules
- Update Schedules: PUT /schedule/:id, status: 'CONFIRMED'
- Find Schedules: GET /schedules
- FindById Schedules: GET /schedules/:id
- Update Schedules: PUT /schedule/:id, status: 'CANCELED'

Com essas configurações, é possível testar as chamadas que serão feitas pelos aplicativos à API. São executadas as rotinas indicadas em três iterações, utilizando as seguintes quantidades de threads: 10, 100, 1000, 2000, 3000 e 4000. A cada interação, o banco de dados é completamente resetado para garantir que não influencie na coleta dos resultados.

Os testes foram conduzidos em um Notebook Dell Vostro 3520, com processador Intel Core I7-1255u, 16GB de memória RAM, SSD de 512GB, placa de vídeo Mx550 com 2GB GDDR6 e sistema operacional Ubuntu 22.04. Essas configurações estabelecem uma limitação de até 4000 threads de usuários simultâneos para os testes.

Ao utilizarmos o JMeter, foi possível verificar o número de chamadas, o tempo de resposta médio da aplicação, os tempos de 90%, 95% e 99% dos usuários, bem como a porcentagem de erros gerados. Para uma melhor ilustração, a Figura 3.24 apresenta um exemplo de análise de desempenho realizado no sistema.

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
User Register	1	73	73	73	73	73	73	73	0.00%	13.7/sec	9.19	4.25
User Login	1	41	41	41	41	41	41	41	0.00%	24.4/sec	16.24	7.41
User Addresses	1	27	27	27	27	27	27	27	0.00%	37.0/sec	21.67	25.97
Cleaner Regis...	1	46	46	46	46	46	46	46	0.00%	21.7/sec	15.69	6.81
Cleaner Login	1	44	44	44	44	44	44	44	0.00%	22.7/sec	16.29	6.97
Update Cleaner	1	31	31	31	31	31	31	31	0.00%	32.3/sec	23.72	27.41
Find Users	1	27	27	27	27	27	27	27	0.00%	37.0/sec	34.90	16.46
Find Cleaners	1	25	25	25	25	25	25	25	0.00%	40.0/sec	83.67	17.62
FindById Users	1	31	31	31	31	31	31	31	0.00%	32.3/sec	14.43	15.12
FindById Clea...	1	26	26	26	26	26	26	26	0.00%	38.5/sec	28.28	17.88
Find UserAddr...	1	26	26	26	26	26	26	26	0.00%	38.5/sec	56.57	18.40
FindById User...	1	24	24	24	24	24	24	24	0.00%	41.7/sec	24.17	20.96
Create Sched...	1	29	29	29	29	29	29	29	0.00%	34.5/sec	24.55	23.88
Update Sche...	1	27	27	27	27	27	27	27	0.00%	37.0/sec	26.22	18.95
Find Schedules	1	33	33	33	33	33	33	33	0.00%	30.3/sec	76.85	13.58
FindById Sche...	1	27	27	27	27	27	27	27	0.00%	37.0/sec	31.32	17.51
Update Sche...	1	31	31	31	31	31	31	31	0.00%	32.3/sec	22.87	16.76
TOTAL	17	33	29	44	46	73	24	73	0.00%	29.1/sec	27.18	14.09

Figura 3.24 – Exemplo de saída do Apache JMeter de uma rotina com 1 Threads de Usuários.

Fonte: Elaborado pelo autor

Como métrica de resultados, foi considerado apenas o tempo de resposta do percentil de 90% dos usuários, a fim de comparar os tempos de resposta. Isso significa que estaremos analisando o tempo de resposta em que 90% dos usuários executaram suas ações dentro desse período. Essa escolha é baseada na ideia de que o percentil de 90% dos usuários representa uma medida mais realista e abrangente do desempenho do sistema. Ao considerar estas respostas, estamos levando em conta a maioria dos usuários, excluindo *outliers*¹ ou eventos extremos que podem distorcer a análise.

Para a análise, teremos alguns casos específicos a serem considerados. Por sempre seguirmos o caminho ideal de cada rotina, a porcentagem de erro deve tender a 0%. Portanto, será considerada uma média de até 10% de erro para que a interação seja aceitável. Outro fator a ser observado é que, se uma rotina atingir 100% de erro, isso significa que ela alcançou o limite máximo de processamento, e o servidor está fora de operação. Em outras palavras, atingiu o limite máximo possível para as requisições, invalidando a iteração.

¹ outlier é um dado que se distancia radicalmente dos demais que compõem a amostra analisada

4

Resultados

Este Capítulo apresenta os resultados dos experimentos para as duas arquiteturas investigadas, definidas nas Seções 4.1 e 4.2, respectivamente. Capturas de tela das saídas do Apache JMeter podem ser consultadas no Apêndice A.1 e A.2.

4.1 Arquitetura Monolítica

A Tabela 4.1 demonstra os dados coletados das 3 iterações rodadas para 10 threads.

ITERAÇÃO	MÉDIA DE ERROS	TEMPO MÉDIO
01	0%	130 ms
02	0%	117 ms
03	0%	116 ms

Tabela 4.1 – Tempo de execução dos experimentos realizados na arquitetura monolítica com 10 threads.

A Tabela 4.2 demonstra os dados coletados das 3 iterações rodadas para 100 threads.

ITERAÇÃO	MÉDIA DE ERROS	TEMPO MÉDIO
01	0%	2008 ms
02	0%	2097 ms
03	0%	2175 ms

Tabela 4.2 – Tempo de execução dos experimentos realizados na arquitetura monolítica com 100 threads.

A Tabela 4.3 demonstra os dados coletados das 3 iterações rodadas para 1000 threads.

ITERAÇÃO	MÉDIA DE ERROS	TEMPO MÉDIO
01	4,98%	130762 ms
02	9,69%	130379 ms
03	10,82%	INVÁLIDO

Tabela 4.3 – Tempo de execução dos experimentos realizados na arquitetura monolítica com 1000 threads.

A Tabela 4.4 demonstra os dados coletados das 3 iterações rodadas para 2000 threads.

ITERAÇÃO	MÉDIA DE ERROS	TEMPO MÉDIO
01	LIMITE ATINGIDO	INVÁLIDO
02	LIMITE ATINGIDO	INVÁLIDO
03	LIMITE ATINGIDO	INVÁLIDO

Tabela 4.4 – Tempo de execução dos experimentos realizados na arquitetura monolítica com 2000 threads.

A Tabela 4.5 demonstra os dados coletados das 3 iterações rodadas para 4000 threads.

ITERAÇÃO	MÉDIA DE ERROS	TEMPO MÉDIO
01	LIMITE ATINGIDO	INVÁLIDO
02	LIMITE ATINGIDO	INVÁLIDO
03	LIMITE ATINGIDO	INVÁLIDO

Tabela 4.5 – Tempo de execução dos experimentos realizados na arquitetura monolítica com 4000 threads.

Com base nas coletas realizadas, é possível identificar que a arquitetura monolítica atendeu aos requisitos para até 1000 threads simultâneas. No entanto, aumentar para 2000 e 4000 threads observa-se que ela se torna ineficiente. Os tempos gerados para o caso de 100 e 1000 threads mostram tempo de execução bem altos.

4.2 Arquitetura de Microserviços

A Tabela 4.6 demonstra os dados coletados das 3 iterações rodadas para 10 threads.

ITERAÇÃO	MÉDIA DE ERROS	TEMPO MÉDIO
01	0%	266 ms
02	0%	126 ms
03	0%	123 ms

Tabela 4.6 – Tempo de execução dos experimentos realizados na arquitetura de microserviços com 10 threads.

A Tabela 4.7 demonstra os dados coletados das 3 iterações rodadas para 100 threads.

ITERAÇÃO	MÉDIA DE ERROS	TEMPO MÉDIO
01	0%	2347 ms
02	0%	2427 ms
03	0%	2145 ms

Tabela 4.7 – Tempo de execução dos experimentos realizados na arquitetura de microserviços com 100 threads.

A Tabela 4.8 demonstra os dados coletados das 3 iterações rodadas para 1000 threads.

ITERAÇÃO	MÉDIA DE ERROS	TEMPO MÉDIO
01	0%	64606 ms
02	0%	66049 ms
03	0%	64797 ms

Tabela 4.8 – Tempo de execução dos experimentos realizados na arquitetura de microserviços com 1000 threads.

A Tabela 4.9 demonstra os dados coletados das 3 iterações rodadas para 2000 threads.

ITERAÇÃO	MÉDIA DE ERROS	TEMPO MÉDIO
01	LIMITE ATINGIDO	INVÁLIDO
02	LIMITE ATINGIDO	INVÁLIDO
03	LIMITE ATINGIDO	INVÁLIDO

Tabela 4.9 – Tempo de execução dos experimentos realizados na arquitetura de microserviços com 2000 threads.

A Tabela 4.10 demonstra os dados coletados das 3 iterações rodadas para 4000 threads.

ITERAÇÃO	MÉDIA DE ERROS	TEMPO MÉDIO
01	LIMITE ATINGIDO	INVÁLIDO
02	LIMITE ATINGIDO	INVÁLIDO
03	LIMITE ATINGIDO	INVÁLIDO

Tabela 4.10 – Tempo de execução dos experimentos realizados na arquitetura de microserviços com 4000 threads.

Como evidenciado nas tabelas, é possível observar que a arquitetura de microserviços atendeu a todas as iterações com 10, 100 e 1000 threads de usuários. No entanto, foi identificado que, nas situações com 2000 e 4000 threads, as máquinas não foram capazes de atender à demanda.

4.3 Arquitetura Monolítica x Arquitetura de Microserviços

Comparando as duas arquiteturas iteração a iteração, observamos que, para 10 iterações, demonstraram tempos médios muito similares e atenderam a todas as rotinas com 0% de erro. O mesmo ocorre ao compararmos as duas arquiteturas para 100 threads. Até o momento, a arquitetura monolítica aparenta ser um pouco melhor em termos de desempenho.

No entanto, é importante notar que, devido ao número limitado de iterações realizadas nestes dois casos, não podemos concluir definitivamente qual arquitetura é superior, sendo necessários mais experimentos e análises para uma comparação mais precisa entre as duas abordagens.

No entanto, ao chegar nas iterações de 1000 threads, é possível ver que a arquitetura de microserviços demonstra sua superioridade em relação à monolítica, tanto em porcentagem de erro quanto no tempo. O tempo é reduzido pela metade e a porcentagem de erro atinge 0%.

A superioridade demonstrada nessas iterações ocorre principalmente porque a arquitetura de microserviços divide um bloco único da arquitetura monolítica em blocos menores, realizando assim o paralelismo das tarefas. Isso evita a congestão em um único fluxo de chamadas, permitindo a execução simultânea de diversas tarefas de forma mais eficiente. O resultado é uma redução significativa no tempo de processamento, já que as operações podem ser distribuídas e executadas de forma concorrente, ao passo que a porcentagem de erro atinge 0%, evidenciando a eficácia desse modelo na gestão de múltiplas threads. Essa abordagem modular e distribuída proporciona um desempenho superior em comparação com a abordagem monolítica, especialmente em situações de carga mais elevada.

Quanto às iterações de 2000 e 4000 threads, ambas as arquiteturas não conseguiram atender, atingindo seus limites. Isso resultou na máquina sendo incapaz de processar as chamadas e saindo de operação, evidenciando que os 16GB de RAM, da maneira como foram desenvolvidos, representam o limite de processamento. Nesse contexto, torna-se claro que as demandas mais intensivas dessas iterações excederam a capacidade de recursos disponíveis, indicando a necessidade de considerar ajustes ou melhorias na infraestrutura para suportar cargas de trabalho mais elevadas.

5

Considerações Finais

Esta monografia apresentou uma proposta de análise de desempenho de duas arquiteturas de software para operação de um serviço em nuvem destinado à contratação de profissionais de limpeza doméstica.

Com uma revisão bibliográfica, foram apresentados a justificativa sobre a necessidade gerada dos tempos atuais, abordando quatro aplicativos que oferecem os mesmos serviços. Ademais, foi realizada uma apresentação abrangente de todas as tecnologias utilizadas na metodologia, incluindo linguagens de programação, frameworks web, banco de dados, arquitetura de software, infraestrutura e análise de desempenho.

Na metodologia, foram descritos os requisitos funcionais e não funcionais que as APIs atenderam. Foi realizada a prototipação das telas, a descrição e construção dos modelos de banco de dados, a especificação de todos os serviços que a API deverá conter e suas respectivas descrições, o desenho das duas arquiteturas, a descrição das duas infraestruturas e, por fim, a explanação detalhada da análise de desempenho, incluindo métricas e casos particulares.

Os resultados da análise de desempenho apontam que as duas arquiteturas estudadas possuem tempos de resposta similares com até 100 threads. Ao serem submetidas a testes com 1000 threads, a arquitetura de microserviços apresenta tempos de resposta menores, indicando sua superioridade em relação à arquitetura monolítica em relação à escalabilidade da solução.

5.1 Trabalhos Futuros

Como trabalhos futuros, pretende-se realizar a implementação dos aplicativos, utilizando a API desenvolvida a partir da arquitetura de microserviços. Além disso, planeja-se realizar testes em máquinas e threads de usuários em maior escala, tanto para a arquitetura monolítica quanto para a de microserviços. Outro objetivo é realizar o cálculo do custo associado a ambas as arquiteturas para uma comparação mais abrangente.

Referências

BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software Architecture in Practice*. [S.l.]: Addison-Wesley, 1998.

BELK, R. You are what you can access: Sharing and collaborative consumption online. *Journal of business research*, v. 67, n. 8, p. 1595-1600, 2014. Disponível em: <https://www.researchgate.net/publication/262490610_You_are_what_you_can_access_Sharing_and_collaborative_consumption_online>.

BERGHE, R. *Do 1g ao 5g: conheça a evolução da internet no celular*. [S.l.], 2020. Disponível em: <<https://www.zoom.com.br/celular/deumzoom/do-1g-ao-5g-evolucao-internet-no-celular>>.

BOTSMAN, R.; ROGERS, R. *What's mine is yours: how collaborative consumption is changing the way we live*. [S.l.], 2011.

CHRIS, R.; FLOYD, S. *Microservices From Design to Deployment*. [S.l.], 2016. Disponível em: <https://ungrid.unal.edu.co/img/Microservices_Designing_Deploying.pdf>.

DIESE, D. I. d. E. e. E. S. *Trabalho doméstico no Brasil*. [S.l.], 2020. Disponível em: <<https://www.dieese.org.br/outraspublicacoes/2021/trabalhoDomestico.html>>.

DOE, J. *JavaScript: A linguagem de programação para a web*. [S.l.]: Editora ABC, 2021.

EASP, F. *Pesquisa anual do uso de ti nas empresas*. [S.l.], 2019. Disponível em: <https://easp.fgv.br/sites/easp.fgv.br/files/noticias2019fgvcia_2019.pdf>.

FREITASE, M. L. A. L. Desenvolvimento de um aplicativo para facilitar a contratação de serviços domésticos. *Sumé - PB: [s.n]*, 2018. Disponível em: <<http://dspace.sti.ufcg.edu.br:8080/jspui/bitstream/riufcg/5030/3/MARIA%20LUIZA%20ALVES%20LIMA%20FREITAS%20-%20TCC%20Engenharia%20de%20Prod%20c3%a7%20c3%a3o%202018..pdf>>.

FREMAN, E.; BATES, B. *Head First Design Patterns*. O'Reilly Media. [S.l.], 2004.

GARLAN, D. *Software architecture*. [S.l.], 2008.

HAMARI, J.; SJÖKLINT, M.; UKKONEN, A. The sharing economy: Why people participate in collaborative consumption. *Journal of the association for information science and technology*, v. 67, n. 9, p. 2047-2059, 2015. Disponível em: <https://www.researchgate.net/publication/255698095_The_Sharing_Economy_Why_People_Participate_in_Collaborative_Consumption>.

IBGE. *Projeções e estimativas da população do Brasil e das unidades da Federação*. [S.l.], 2019. Disponível em: <<https://www.ibge.gov.br/apps/populacao/projecao/index.html/>>.

INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA. [S.l.], 2020. Disponível em: <<https://www.ibge.gov.br>>.

INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA. *Desemprego*. [S.l.], 2022. Disponível em: <<https://www.ibge.gov.br/explica/desemprego.php>>.

- JOHN, N. A. The social logics of sharing. *The Communication Review*, v. 16, n. 3, p. 113-131, 2013. Disponível em: <<https://sci-hub.se/https://doi.org/10.1080/10714421.2013.807119>>.
- LEITE, K. C. A (in)esperada pandemia e suas implicações para o mundo do trabalho. *Psicologia Sociedade*, 2020. Disponível em: <<https://www.scielo.br/j/psoc/a/5kJx53cdZNmRDWfrmNW76cD/?format=html&lang=pt#>>.
- MARTINS, P. *MongoDB: O banco de dados NoSQL para aplicações modernas*. [S.l.]: Editora XYZ, 2022.
- MESHENBERG, R. *Netflix - Scaling Microservices*. (Conferência em vídeo). [S.l.], 2016. Disponível em: <<https://www.youtube.com/watch?v=57UK46qfBLY>>.
- MICROSOFT, L. *Learn Microsoft*. Site de educação da empresa Microsoft. Disponível em: <<https://learn.microsoft.com>>.
- NEVES, J. A. e. a. Desemprego, pobreza e fome no brasil em tempos de pandemia por covid-19. *Revista de Nutrição*, v. 34, p. e200170, 2021. Disponível em: <https://www.researchgate.net/profile/Jose-Neves-30/publication/352065007_Unemployment_poverty_and_hunger_in_Brazil_in_Covid-19_pandemic_times/links/60b80b56a6fdccb96f4d857a/Unemployment-poverty-and-hunger-in-Brazil-in-Covid-19-pandemic-times.pdf>.
- NEWMAN, S. *Building Microservices*. [S.l.]: "O'Reilly Media, Inc.", 2021.
- NGUYEN, S. C.; KHA, H. H.; HOANG, M. N. Glossary of software engineering terminology. *IEEE*, 2017. Disponível em: <<https://ieeexplore.ieee.org/document/7849804>>.
- OLIVEIRA, P. *Node.js: Desenvolvimento eficiente e flexível*. [S.l.]: Editora ABC, 2021.
- PARAMOUND, A.; MARTIN, B. *Novas Perspectivas em Bancos de Dados: A Ascensão do Modelo NoSQL*. [S.l.]: Editora XYZ, 2012.
- PEREIRA, D. G.; FONTÃO, H.; LOPES, E. M. Estudo de viabilidade de um aplicativo de serviços autônomos. *Revista de Pesquisa Aplicada e Tecnologia – REPATEC*, v. 03, n. 5, p. 22-36, São Paulo, 2021. Disponível em: <<http://www.repatec.com.br/index.php/periodico/article/view/29/26>>.
- PNAD, I. B. d. G. e. E. *Pesquisa Nacional por Amostra de Domicílios*. [S.l.], 2015.
- RUFINO, J. D. d. A. Severino: uma aplicação mobile para contratação de serviços. *Universidade Federal de Uberlândia*, 2021. Disponível em: <<http://repositorio.ufu.br/bitstream/123456789/32445/1/SeverinoUmaAplica%C3%A7%C3%A3o.pdf>>.
- SANTANA, J. M. B.; NETO, I. F. F. Shared economy: Uber como catalisador do índice de desemprego e oportunidade de mercado no maranhão. *Brazilian Journal of Business*, v. 3, n. 3, p. 2627-2643, 2021. Disponível em: <<https://www.brazilianjournals.com/index.php/BJB/article/viewFile/34388/26914>>.
- SILVA, J. *Node.js: A Plataforma de Execução de JavaScript*. [S.l.]: Editora XYZ, 2022.
- SILVA, M. *Node.js: Desenvolvimento de Aplicações Web Eficientes*. [S.l.]: Editora XYZ, 2022.
- SILVA, M. *Express.js: Desenvolvimento de Aplicativos Web Simples e Eficientes*. [S.l.]: Editora XYZ, 2023.

SILVEIRA, L. M.; PETRINI, M.; SANTOS, A. C. M. Z. Economia compartilhada e consumo colaborativo: o que estamos pesquisando?. *REGE-Revista de Gestão*, v. 23, n. 4, p. 298-305, 2016. Disponível em: <<https://www.revistas.usp.br/rege/article/view/129033/125686>>.

SMITH, J. *JavaScript: A Linguagem de Programação para a Web Moderna*. [S.l.]: Editora XYZ, 2021.

SOUZA, T. L. F. d. Ux ui design - interface gráfica para busca de prestadores de serviços domésticos. *Trabalho de Conclusão de Curso (Graduação em Design)* - Universidade Federal de Uberlândia, 2018. Disponível em: <<http://clyde.dr.ufu.br/bitstream/123456789/22277/1/UxUiDesignInterface>>.

Apêndices

APÊNDICE A – Capturas de tela dos experimentos realizados com o Apache JMeter

A.1 Análise de Desempenho Arquitetura Monolítica

A.1.1 10 Threads

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
User Register	10	91	85	122	122	124	59	124	0.00%	9.9/sec	6.62	3.07
User Login	10	63	62	75	75	94	35	94	0.00%	10.2/sec	6.82	3.11
User Addresses	10	55	52	86	86	88	23	88	0.00%	10.0/sec	5.84	7.00
Cleaner Regis...	10	90	80	132	132	135	71	135	0.00%	9.5/sec	6.84	2.97
Cleaner Login	10	62	63	82	82	84	36	84	0.00%	10.2/sec	7.28	3.12
Update Cleaner	10	116	115	136	136	175	79	175	0.00%	9.3/sec	6.82	7.87
Find Users	10	90	93	101	101	116	71	116	0.00%	9.3/sec	15.95	4.13
Find Cleaners	10	102	103	110	110	130	77	130	0.00%	8.8/sec	31.20	3.89
FindByld Users	10	98	93	106	106	169	71	169	0.00%	9.1/sec	4.07	4.27
FindByld Clea...	10	84	59	120	120	161	39	161	0.00%	9.6/sec	7.04	4.45
Find UserAddr...	10	83	95	105	105	114	40	114	0.00%	9.7/sec	29.33	4.62
FindByld User...	10	91	101	124	124	125	39	125	0.00%	10.2/sec	5.93	5.14
Create Sched...	10	94	77	132	132	190	42	190	0.00%	11.1/sec	7.89	7.68
Update Sche...	10	83	73	133	133	183	35	183	0.00%	12.3/sec	8.70	6.29
Find Schedules	10	84	59	131	131	193	36	193	0.00%	13.7/sec	55.35	6.12
FindByld Sche...	10	70	42	135	135	143	24	143	0.00%	15.5/sec	13.11	7.33
Update Sche...	10	60	37	118	118	184	20	184	0.00%	18.6/sec	13.15	9.64
TOTAL	170	83	81	130	135	184	20	193	0.00%	82.8/sec	103.05	40.07

Figura A.1 – Rotina com 10 Threads de Usuários 1/3. Fonte: Elaborado pelo autor

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
User Register	10	91	92	119	119	138	53	138	0.00%	9.7/sec	6.49	3.00
User Login	10	66	50	97	97	140	39	140	0.00%	10.0/sec	6.67	3.04
User Addresses	10	74	68	93	93	125	46	125	0.00%	9.6/sec	5.59	6.70
Cleaner Regis...	10	82	77	106	106	111	56	111	0.00%	9.0/sec	6.51	2.83
Cleaner Login	10	75	64	98	98	131	45	131	0.00%	9.3/sec	6.64	2.84
Update Cleaner	10	93	90	145	145	149	35	149	0.00%	8.9/sec	6.58	7.60
Find Users	10	85	80	98	98	117	67	117	0.00%	8.8/sec	14.63	3.90
Find Cleaners	10	92	91	123	123	130	59	130	0.00%	9.1/sec	31.18	3.99
FindByld Users	10	94	90	131	131	131	61	131	0.00%	9.1/sec	4.08	4.28
FindByld Clea...	10	85	85	106	106	138	51	138	0.00%	9.5/sec	7.00	4.43
Find UserAddr...	10	76	65	108	108	114	36	114	0.00%	9.8/sec	29.65	4.67
FindByld User...	10	78	72	113	113	162	26	162	0.00%	9.9/sec	5.74	4.97
Create Sched...	10	82	70	104	104	161	51	161	0.00%	10.5/sec	7.49	7.29
Update Sche...	10	81	71	107	107	135	50	135	0.00%	11.8/sec	8.35	6.03
Find Schedules	10	77	69	95	95	128	52	128	0.00%	12.1/sec	46.85	5.41
FindByld Sche...	10	72	70	93	93	97	35	97	0.00%	13.6/sec	11.47	6.41
Update Sche...	10	63	61	87	87	90	29	90	0.00%	14.9/sec	10.55	7.73
TOTAL	170	80	78	117	131	149	26	162	0.00%	80.7/sec	99.06	39.08

Figura A.2 – Rotina com 10 Threads de Usuários 2/3. Fonte: Elaborado pelo autor

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
User Register	10	89	95	102	102	109	60	109	0.00%	10.0/sec	6.71	3.11
User Login	10	74	65	105	105	109	36	109	0.00%	10.1/sec	6.73	3.07
User Addresses	10	73	68	118	118	122	25	122	0.00%	9.8/sec	5.71	6.85
Cleaner Regis...	10	92	80	119	119	129	72	129	0.00%	9.5/sec	6.82	2.96
Cleaner Login	10	78	73	104	104	123	54	123	0.00%	9.9/sec	7.13	3.05
Update Cleaner	10	106	108	135	135	157	68	157	0.00%	9.2/sec	6.76	7.82
Find Users	10	85	80	107	107	112	58	112	0.00%	9.2/sec	15.80	4.10
Find Cleaners	10	102	100	121	121	127	61	127	0.00%	9.6/sec	33.83	4.24
FindByld Users	10	79	71	116	116	126	50	126	0.00%	10.0/sec	4.48	4.70
FindByld Clea...	10	79	79	113	113	118	34	118	0.00%	10.5/sec	7.73	4.89
Find UserAddr...	10	74	80	113	113	115	22	115	0.00%	11.6/sec	35.38	5.57
FindByld User...	10	72	71	108	108	111	30	111	0.00%	12.8/sec	7.43	6.44
Create Sched...	10	79	57	127	127	150	35	150	0.00%	13.4/sec	9.52	9.26
Update Sche...	10	67	55	98	98	100	38	100	0.00%	14.9/sec	10.52	7.60
Find Schedules	10	70	63	96	96	123	34	123	0.00%	16.4/sec	68.38	7.36
FindByld Sche...	10	52	46	80	80	97	29	97	0.00%	19.4/sec	16.42	9.18
Update Sche...	10	47	37	76	76	83	26	83	0.00%	21.0/sec	14.89	10.91
TOTAL	170	77	78	116	122	135	22	157	0.00%	86.4/sec	108.04	41.82

Figura A.3 – Rotina com 10 Threads de Usuários 3/3. Fonte: Elaborado pelo autor

A.1.2 100 Threads

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
User Register	100	2035	2141	2366	2406	2442	222	2452	0.00%	35.7/sec	23.94	11.08
User Login	100	521	506	612	1913	1965	178	2165	0.00%	31.7/sec	21.11	9.63
User Addresses	100	1365	1365	1885	1933	2105	446	2106	0.00%	32.3/sec	18.91	22.66
Cleaner Regis...	100	1213	1154	1642	1716	1764	628	1775	0.00%	31.2/sec	22.54	9.79
Cleaner Login	100	484	395	984	1370	1392	189	1404	0.00%	38.0/sec	27.24	11.65
Update Cleaner	100	1067	939	1613	1883	2237	779	2384	0.00%	32.4/sec	23.84	27.54
Find Users	100	604	605	674	689	694	504	700	0.00%	71.7/sec	138.19	31.85
Find Cleaners	100	648	649	696	702	714	592	715	0.00%	67.6/sec	324.53	29.78
FindByld Users	100	526	514	640	655	668	459	672	0.00%	74.0/sec	33.11	34.70
FindByld Clea...	100	499	491	543	548	553	445	555	0.00%	81.2/sec	59.69	37.73
Find UserAddr...	100	627	633	710	721	723	466	733	0.00%	70.2/sec	643.16	33.58
FindByld User...	100	687	700	758	764	769	579	772	0.00%	65.0/sec	37.72	32.70
Create Sched...	100	691	666	768	832	884	648	901	0.00%	61.9/sec	44.05	42.85
Update Sche...	100	712	704	800	817	894	559	925	0.00%	60.9/sec	43.14	31.18
Find Schedules	100	1958	2219	2454	2462	2497	800	2507	0.00%	29.1/sec	497.72	13.02
FindByld Sche...	100	1284	988	2314	2451	2485	634	2486	0.00%	30.3/sec	25.64	14.33
Update Sche...	100	558	557	640	646	654	375	720	0.00%	80.7/sec	57.22	41.93
TOTAL	1700	911	663	2008	2284	2452	178	2507	0.00%	105.7/sec	260.21	51.16

Figura A.4 – Rotina com 100 Threads de Usuários 1/3. Fonte: Elaborado pelo autor

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
User Register	100	2003	2019	2285	2311	2332	566	2543	0.00%	37.1/sec	24.87	11.51
User Login	100	427	441	512	524	1118	224	1770	0.00%	37.8/sec	25.15	11.47
User Addresses	100	1674	1746	2367	2452	2614	693	2637	0.00%	28.6/sec	16.76	20.09
Cleaner Regis...	100	1671	1569	2270	2391	2405	815	2405	0.00%	27.6/sec	19.95	8.66
Cleaner Login	100	432	375	438	1366	1368	262	1371	0.00%	33.9/sec	24.31	10.40
Update Cleaner	100	1149	1133	1283	1334	1350	986	1934	0.00%	35.8/sec	26.36	30.45
Find Users	100	729	734	786	792	811	652	812	0.00%	60.0/sec	115.64	26.65
Find Cleaners	100	763	751	805	809	817	673	821	0.00%	58.8/sec	282.01	25.88
FindByld Users	100	596	594	646	664	709	510	756	0.00%	66.9/sec	29.92	31.35
FindByld Clea...	100	495	492	511	514	538	468	575	0.00%	82.9/sec	60.92	38.51
Find UserAddr...	100	594	582	744	771	791	457	793	0.00%	70.8/sec	648.62	33.87
FindByld User...	100	723	733	777	781	795	595	799	0.00%	64.7/sec	37.55	32.55
Create Sched...	100	777	763	826	830	838	711	933	0.00%	60.0/sec	42.68	41.51
Update Sche...	100	654	644	699	715	730	627	731	0.00%	69.7/sec	49.37	35.68
Find Schedules	100	2181	2359	2717	2734	2745	696	2745	0.00%	28.9/sec	494.93	12.95
FindByld Sche...	100	1108	909	2112	2510	2569	503	2693	0.00%	28.5/sec	24.10	13.47
Update Sche...	100	550	568	595	596	601	187	601	0.00%	99.7/sec	70.69	51.80
TOTAL	1700	972	736	2097	2359	2693	187	2745	0.00%	99.4/sec	244.74	48.11

Figura A.5 – Rotina com 100 Threads de Usuários 2/3. Fonte: Elaborado pelo autor

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
User Register	100	1765	1854	2088	2107	2138	372	2147	0.00%	38.5/sec	25.81	11.95
User Login	100	508	468	547	1416	1580	197	1798	0.00%	36.9/sec	24.57	11.20
User Addresses	100	1311	1406	1539	1542	1684	829	1684	0.00%	39.0/sec	22.80	27.33
Cleaner Regis...	100	1331	1366	1490	1767	1774	863	1775	0.00%	31.5/sec	22.76	9.89
Cleaner Login	100	440	320	1110	1123	1129	208	1155	0.00%	38.3/sec	27.43	11.74
Update Cleaner	100	1071	975	1871	1904	1971	895	1972	0.00%	36.8/sec	27.03	31.24
Find Users	100	612	606	647	649	652	564	655	0.00%	69.2/sec	133.41	30.75
Find Cleaners	100	577	580	608	610	616	543	622	0.00%	71.1/sec	341.14	31.30
FindByld Users	100	513	505	556	561	570	479	588	0.00%	76.1/sec	34.04	35.67
FindByld Clea...	100	534	522	590	593	599	476	600	0.00%	75.7/sec	55.67	35.19
Find UserAddr...	100	682	695	754	772	778	504	781	0.00%	61.5/sec	563.66	29.43
FindByld User...	100	680	681	758	774	786	580	786	0.00%	58.6/sec	34.00	29.48
Create Sched...	100	707	699	764	787	827	666	829	0.00%	60.9/sec	43.33	42.14
Update Sche...	100	739	687	852	870	946	653	947	0.00%	57.0/sec	40.34	29.16
Find Schedules	100	18264	22808	34904	34912	34928	836	34940	0.00%	2.8/sec	47.63	1.25
FindByld Sche...	100	17295	13000	34796	34893	34928	544	34931	2.00%	2.8/sec	2.46	1.30
Update Sche...	100	1422	719	2201	2210	2213	531	13158	0.00%	6.4/sec	4.56	3.34
TOTAL	1700	2850	699	2175	23123	34894	197	34940	0.12%	34.0/sec	83.71	16.43

Figura A.6 – Rotina com 100 Threads de Usuários 3/3. Fonte: Elaborado pelo autor

A.1.3 1000 Threads

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/s...	Sent KB/sec
User Register	999	26339	21981	22551	22685	129711	19925	129760	4.00%	7.6/sec	5.76	2.28
User Login	999	4866	4227	5658	5809	32093	955	37440	4.00%	6.7/sec	4.41	2.04
User Addresses	999	16735	16124	27449	29610	31713	34	33347	4.00%	6.8/sec	5.12	4.68
Update Schedule CANCELLED	998	56137	51388	87664	99806	130785	50	151487	7.11%	58.8/min	0.73	0.50
Update Schedule ACCEPTED	998	99433	92251	141222	147077	185840	26	195259	8.52%	1.1/sec	0.97	0.52
Update Cleaner	999	139666	140678	173439	177830	187865	64	188107	3.70%	2.4/sec	1.94	1.95
TCC SCRIPT:User Register	1	1615247	1615247	1615247	1615247	1615247	1615247	1615247	100.00%	2.2/hour	0.00	0.00
TCC SCRIPT:Update Schedule A...	1	750894	750894	750894	750894	750894	750894	750894	100.00%	4.8/hour	0.00	0.00
Find Users	999	75791	70336	108491	129950	131086	74	133048	7.41%	2.5/sec	4.93	1.02
Find UserAddresses	999	69885	67737	76294	89768	130258	2846	137440	1.90%	1.6/sec	14.42	0.75
Find Schedules	998	125031	129542	152823	156753	176295	104	195226	2.10%	59.7/min	11.02	0.44
Find Cleaners	999	58909	59738	69364	75487	89848	21	130589	4.00%	2.6/sec	11.86	1.12
FindByld Users	999	65896	61362	85179	129595	129988	198	131912	8.61%	2.3/sec	1.35	1.01
FindByld UserAddresses	999	71405	68032	88085	103047	139579	102	141019	5.71%	1.4/sec	0.93	0.71
FindByld Schedules	998	100443	107636	141978	151342	161208	54	192054	7.92%	58.5/min	1.01	0.44
FindByld Cleaners	999	61970	61223	75044	86926	98105	29	131893	4.00%	1.9/sec	1.73	0.88
Create Schedule	999	91132	90809	107490	123349	154248	20	160293	6.31%	1.2/sec	1.13	0.81
Cleaner Register	999	70988	84933	99366	109496	110897	5336	111529	0.80%	4.7/sec	3.49	1.48
Cleaner Login	999	43617	34018	83236	84821	129549	5155	131014	4.30%	3.0/sec	2.38	0.90
TOTAL	16981	69433	65354	130762	143848	172342	20	1615247	4.98%	10.5/sec	22.87	4.94

Figura A.7 – Rotina com 1000 Threads de Usuários 1/3. Fonte: Elaborado pelo autor

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
User Register	1000	29946	20841	21127	130400	130448	18628	130478	8.40%	7.6/sec	6.43	2.17
User Login	1000	8470	3801	5554	21648	130982	1055	131032	8.40%	4.1/sec	2.90	1.21
User Addresses	1000	13988	14678	21103	21836	22293	48	131058	8.40%	2.7/sec	2.57	1.83
Cleaner Regis...	998	64164	86960	91084	91855	105503	8149	107715	2.20%	2.3/sec	1.73	0.70
Cleaner Login	996	37307	36631	51889	122116	130675	4160	131192	7.03%	2.1/sec	1.72	0.62
Update Cleaner	993	168695	186813	212068	214812	214919	34	271462	9.97%	1.8/sec	1.70	1.37
Find Users	992	94590	93737	130807	132358	169028	1414	196009	12.50%	1.8/sec	3.72	0.70
Find Cleaners	991	68812	70751	92842	129762	130762	26	172161	13.62%	2.0/sec	9.66	0.82
FindByld Users	990	63508	60313	76364	84703	129829	1589	132604	10.10%	1.8/sec	1.06	0.82
FindByld Clea...	990	61782	61412	76397	94995	132605	17	138937	10.71%	1.6/sec	1.86	0.72
Find UserAddr...	990	74210	71579	95491	130494	138911	37	143906	7.37%	1.4/sec	12.67	0.65
FindByld User...	989	79144	75590	94783	130822	140356	2047	144036	12.13%	1.3/sec	0.99	0.61
Create Sched...	988	95216	100544	117624	129862	161009	28	190137	10.93%	1.1/sec	1.25	0.72
Update Sche...	987	101343	102887	131853	140378	182421	27	190029	14.89%	59.2/min	1.14	0.47
Find Schedules	987	115073	121315	131836	139421	163107	1373	185146	4.86%	56.4/min	10.33	0.41
FindByld Sche...	987	89391	97168	128848	131125	143136	41	163179	11.75%	55.7/min	1.13	0.41
Update Sche...	987	54056	51909	91211	105784	119983	129	130836	11.55%	55.9/min	0.74	0.47
TOTAL	16865	71644	70737	130379	163654	197128	17	271462	9.69%	13.1/sec	29.50	5.98

Figura A.8 – Rotina com 1000 Threads de Usuários 2/3. Fonte: Elaborado pelo autor

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
User Register	1000	30706	21013	21621	129851	129893	19419	129903	8.90%	7.6/sec	6.54	2.16
User Login	1000	14738	3913	5447	131053	131062	1317	131072	8.90%	4.1/sec	3.47	1.15
User Addresses	999	20516	14855	22509	131061	131068	65	131071	8.91%	2.7/sec	2.31	1.76
Cleaner Regis...	999	79253	102867	106895	107768	131069	6528	132130	3.40%	2.0/sec	1.57	0.61
Cleaner Login	999	29891	25445	57022	57201	79666	7139	130801	3.50%	1.9/sec	1.36	0.59
Update Cleaner	999	111095	110363	146563	169979	196457	32	196813	5.01%	1.8/sec	1.60	1.45
Find Users	997	128123	132285	159060	163765	212654	1472	212693	11.33%	1.9/sec	3.95	0.76
Find Cleaners	996	64804	57338	129412	130714	141636	35	161442	17.07%	2.0/sec	9.13	0.76
FindByld Users	996	65322	51326	115046	129247	145608	1469	178540	11.24%	2.2/sec	1.42	0.98
FindByld Clea...	996	94174	113746	130031	130269	144638	26	148168	16.77%	1.8/sec	2.33	0.75
Find UserAddr...	996	81009	78124	119480	129289	136449	35	145567	8.84%	1.5/sec	12.72	0.65
FindByld User...	995	74672	66919	104701	130212	155434	36	165040	13.57%	1.4/sec	1.10	0.63
Create Sched...	994	102848	110454	130310	139884	174129	28	185989	14.08%	1.2/sec	1.45	0.76
Update Sche...	992	99506	105428	131071	144991	186236	40	211199	17.74%	1.1/sec	1.25	0.50
Find Schedules	992	124562	130185	148182	160740	209626	1336	211172	5.24%	1.0/sec	11.28	0.44
FindByld Sche...	992	91147	97911	140730	144696	146927	35	170289	15.12%	60.0/min	1.23	0.44
Update Sche...	992	50090	52734	77437	88413	116589	36	151712	14.42%	1.0/sec	0.84	0.51
TOTAL	16934	74211	75416	131071	145536	165676	26	212693	10.82%	12.7/sec	28.91	5.77

Figura A.9 – Rotina com 1000 Threads de Usuários 3/3. Fonte: Elaborado pelo autor

A.1.4 2000 Threads

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received ...	Sent KB/sec
User Regi...	2000	82081	129310	131007	131108	131204	17548	176338	55.15%	11.2/sec	20.34	1.56
User Login	2000	68833	59100	131069	131072	131075	886	145220	55.20%	6.9/sec	11.51	1.04
User Addr...	2000	71460	20765	131088	131104	163501	42	209974	55.15%	4.2/sec	8.11	1.46
Cleaner R...	1998	92142	103378	131061	141365	160660	6255	251391	28.48%	3.7/sec	4.79	0.83
Cleaner L...	1961	62919	36740	131062	131068	131075	4795	251063	35.34%	2.7/sec	3.52	0.56
Update Cl...	947	114427	131064	153658	231932	312832	22	314312	82.68%	1.3/sec	3.34	0.29
Find Users	849	80167	74879	224707	226124	233334	32	255246	69.14%	1.0/sec	3.43	0.25
Find Clea...	721	95333	103299	127520	131082	215326	38	263948	79.33%	54.0/min	3.95	0.22
FindByld U...	569	84502	96425	98614	211681	211748	38	211769	97.72%	33.8/min	1.41	0.04
FindByld C...	540	34119	1991	113483	113650	113694	34	131070	100.00%	32.2/min	2.50	0.12
Find User...	525	78160	97837	113555	149162	149195	22	149387	100.00%	27.3/min	2.03	0.09
FindByld U...	520	82671	100546	113782	115257	190006	48	190013	100.00%	24.9/min	1.32	0.05
Create Sc...	519	45812	40559	100617	102245	149191	49	149202	100.00%	24.3/min	1.85	0.15
Update S...	519	61731	59490	102107	113022	130907	34	193806	100.00%	22.4/min	1.55	0.08
Find Sche...	519	80543	94923	189993	189999	190010	48	190015	100.00%	21.1/min	1.26	0.05
FindByld S...	519	49289	34444	111620	113017	113027	30	128718	100.00%	20.8/min	1.49	0.07
Update S...	519	52211	18752	111622	111625	113023	11	173175	100.00%	20.8/min	1.42	0.07
TOTAL	17225	75368	94868	131070	131104	224727	11	314312	65.31%	10.0/sec	25.08	2.12

Figura A.10 – Rotina com 2000 Threads de Usuários 1/3. Fonte: Elaborado pelo autor

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received ...	Sent KB/sec
User Regi...	2000	82978	129369	130179	131111	131192	17601	131212	57.35%	15.0/sec	27.93	1.99
User Login	2000	74642	130985	131031	131069	131073	945	131083	57.35%	8.1/sec	14.55	1.11
User Addr...	2000	76298	131040	131070	131076	131091	39	131095	57.35%	5.3/sec	10.15	1.69
Cleaner R...	2000	87571	77160	131053	131067	137802	6129	193801	41.40%	4.0/sec	6.18	0.73
Cleaner L...	1997	66186	35388	131063	131068	131074	4313	225136	42.71%	2.8/sec	4.15	0.52
Update Cl...	1621	139237	131067	206501	227716	239798	31	241039	52.19%	2.2/sec	4.16	0.96
Find Users	1489	118653	130991	225040	232081	240460	35	286550	65.61%	1.7/sec	5.18	0.37
Find Clea...	1465	92643	100793	131111	139405	171476	27	240317	62.66%	1.5/sec	6.99	0.42
FindByld U...	1178	134869	131068	211150	227923	230585	43	231900	79.63%	1.0/sec	3.39	0.23
FindByld C...	821	136531	154479	193861	193912	193973	23	211203	99.88%	42.3/min	2.60	0.08
Find User...	664	86368	17116	211066	211107	211143	16	211174	100.00%	34.2/min	1.60	0.02
FindByld U...	468	5243	1034	7137	7787	130549	13	142973	100.00%	24.0/min	1.25	0.02
Create Sc...	375	2565	46	7300	7312	7337	10	139356	100.00%	19.2/min	1.05	0.03
Update S...	350	3775	1353	7312	7323	15663	13	211032	100.00%	17.9/min	0.99	0.02
Find Sche...	342	3108	68	7296	7306	9063	13	106143	100.00%	17.5/min	0.98	0.02
FindByld S...	306	1747	23	5686	7306	10443	12	99148	100.00%	15.7/min	0.88	0.02
Update S...	238	3473	24	4580	6829	130381	13	139236	100.00%	12.2/min	0.74	0.02
TOTAL	19314	85364	106138	154573	193860	230580	10	286550	64.02%	14.2/sec	34.64	2.86

Figura A.11 – Rotina com 2000 Threads de Usuários 2/3. Fonte: Elaborado pelo autor

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received ...	Sent KB/sec
User Regi...	2000	83176	130085	130880	130992	131076	18821	131088	57.05%	15.2/sec	28.06	2.02
User Login	2000	74959	131022	131066	131071	131076	1183	146177	57.05%	7.7/sec	14.01	1.04
User Addr...	2000	77812	131035	131106	131112	131120	72	227439	57.05%	5.3/sec	10.01	1.67
Cleaner R...	2000	91840	130971	131028	131040	131079	6030	131111	53.70%	4.0/sec	7.19	0.58
Cleaner L...	2000	79786	130949	131024	131031	131077	4197	131084	55.00%	3.2/sec	5.66	0.47
Update Cl...	1996	141315	131030	208077	212506	219109	29	259929	55.01%	2.7/sec	5.20	1.10
Find Users	1955	125320	131009	176672	190826	208663	25	238043	57.54%	2.5/sec	6.26	0.50
Find Clea...	1927	70131	78773	130987	131001	167212	34	218219	59.78%	2.2/sec	9.31	0.55
FindByd U...	1874	127644	131039	185375	185403	234959	28	252862	58.80%	1.8/sec	4.44	0.51
FindByd C...	1307	136699	151598	192482	192511	225592	41	288126	76.43%	1.1/sec	3.80	0.27
Find User...	285	83520	129857	167175	167254	202795	32	278902	97.19%	14.3/min	0.88	0.03
FindByd U...	138	19480	5772	130550	131019	131073	23	167203	100.00%	9.4/min	0.71	0.03
Create Sc...	131	31847	5653	130546	130959	131054	28	185372	96.95%	7.7/min	0.58	0.05
Update S...	113	19805	4352	8648	131038	192506	25	271378	100.00%	5.7/min	0.46	0.03
Find Sche...	108	10583	4234	8212	17020	167563	20	167586	100.00%	5.5/min	0.45	0.02
FindByd S...	105	9192	4175	7386	32071	131065	26	131065	100.00%	8.7/min	0.73	0.04
Update S...	103	5193	3093	7120	8649	37529	24	131067	100.00%	8.6/min	0.73	0.04
TOTAL	20042	96336	130850	163722	180422	212507	20	288126	60.09%	14.3/sec	34.87	3.23

Figura A.12 – Rotina com 2000 Threads de Usuários 3/3. Fonte: Elaborado pelo autor

A.1.5 4000 Threads

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received ...	Sent KB/sec
User Regi...	4000	105534	129648	131105	131193	131284	15935	131318	77.35%	30.3/sec	68.77	2.13
User Login	4000	100654	130966	131047	131076	131092	1148	131140	77.35%	16.2/sec	36.21	1.18
User Addr...	4000	98514	130964	131050	131065	131105	43	131122	77.35%	10.7/sec	25.35	1.97
Cleaner R...	3620	114159	130959	131083	131094	227220	5705	282417	72.43%	6.8/sec	14.82	0.59
Cleaner L...	3579	107783	131038	131270	131305	131439	4818	150735	76.14%	5.5/sec	11.91	0.46
Update Cl...	3128	125936	130678	193289	222415	240506	41	298349	86.35%	3.5/sec	9.81	0.74
Find Users	2816	126764	130937	144225	144249	166050	104	307898	99.40%	3.0/sec	8.97	0.14
Find Clea...	2610	129397	130956	166082	167531	167692	1392	310196	99.96%	2.9/sec	9.88	0.21
FindByd U...	485	113968	129638	165497	167429	167525	58	256876	100.00%	32.2/min	2.03	0.06
FindByd C...	143	41817	17506	131096	137129	144258	36	144264	100.00%	9.9/min	0.67	0.03
Find User...	125	45040	17474	130864	143324	165589	1373	167457	100.00%	8.3/min	0.65	0.03
FindByd U...	113	48335	17499	130989	132327	143324	33	143325	100.00%	7.9/min	0.62	0.03
Create Sc...	105	67634	31271	131088	132375	165579	593	167395	100.00%	7.0/min	0.56	0.05
Update S...	88	69673	5926	132346	165491	167410	1237	167472	100.00%	5.9/min	0.47	0.03
Find Sche...	73	61414	5718	165505	165564	166021	54	167468	100.00%	4.9/min	0.39	0.02
FindByd S...	48	14538	3446	7105	129446	165462	37	165462	100.00%	3.2/min	0.26	0.01
Update S...	44	6904	3253	6162	12722	130287	25	130287	100.00%	7.9/min	0.66	0.04
TOTAL	28977	110346	130945	131194	144198	214047	25	310196	82.69%	26.4/sec	68.08	2.86

Figura A.13 – Rotina com 4000 Threads de Usuários 1/3. Fonte: Elaborado pelo autor

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received ...	Sent KB/sec
User Regi...	4000	105663	129683	131070	131193	131247	19277	131273	77.42%	30.3/sec	68.78	2.12
User Login	4000	99623	130968	131031	131051	131095	1240	131121	77.42%	16.4/sec	36.24	1.25
User Addr...	3999	99377	130989	131072	131105	131142	29	131186	77.42%	10.7/sec	25.19	1.92
Cleaner R...	3470	115020	130946	130992	131064	242122	5232	308501	70.37%	6.3/sec	13.36	0.58
Cleaner L...	3357	108447	131038	135076	135096	135165	5939	167605	75.01%	5.1/sec	10.54	0.48
Update Cl...	2359	111348	130956	131051	131069	176870	35	207386	99.96%	3.0/sec	9.95	0.48
Find Users	2278	124400	131204	141045	186313	188457	63	204046	99.96%	2.6/sec	8.86	0.18
Find Clea...	533	105110	140938	141130	141225	160774	41	212713	99.06%	37.2/min	2.43	0.10
FindByd U...	191	47097	18560	142357	142409	160798	43	311002	100.00%	14.8/min	1.09	0.05
FindByd C...	167	50575	15745	142377	160743	180168	57	233823	100.00%	13.0/min	0.93	0.04
Find User...	147	50769	15743	142387	142429	160764	47	160766	100.00%	11.4/min	0.80	0.03
FindByd U...	117	53471	16780	142379	160770	160813	94	160851	100.00%	9.1/min	0.65	0.03
Create Sc...	98	63292	19890	142403	160808	160851	23	301220	100.00%	7.6/min	0.55	0.04
Update S...	73	31182	6260	141003	142365	142439	1151	160857	100.00%	5.7/min	0.43	0.02
Find Sche...	61	7771	4444	7983	23618	35809	1390	142314	100.00%	4.9/min	0.42	0.02
FindByd S...	60	4808	4158	9157	9158	9172	36	10923	100.00%	11.5/min	0.89	0.04
Update S...	60	4096	4225	6859	7995	8002	24	10120	100.00%	11.4/min	0.97	0.06
TOTAL	24970	105104	130961	134919	135081	187863	23	311002	81.65%	25.6/sec	65.35	2.84

Figura A.14 – Rotina com 4000 Threads de Usuários 2/3. Fonte: Elaborado pelo autor

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received ...	Sent KB/sec
User Regi...	4000	105830	130313	131071	131164	131270	19702	131291	77.40%	30.3/sec	68.90	2.13
User Login	4000	100539	130991	131065	131082	131108	1015	131137	77.40%	16.4/sec	36.79	1.20
User Addr...	4000	100560	130946	131070	131081	131103	36	131150	77.40%	10.7/sec	25.10	1.90
Cleaner R...	3986	114447	130921	131046	131078	166990	6424	220010	74.41%	6.8/sec	15.09	0.55
Cleaner L...	3983	109516	130873	131029	131042	131073	7183	216061	76.90%	5.0/sec	11.28	0.38
Update Cl...	3852	130912	130968	199694	201230	202684	40	306012	78.01%	4.9/sec	12.66	1.29
Find Users	3475	133654	131091	193574	241098	270661	34	302329	89.53%	3.9/sec	11.88	0.35
Find Clea...	3002	131193	130624	150404	217729	217871	42	217997	99.93%	3.4/sec	9.81	0.11
FindByld U...	533	98650	67153	217913	217938	217987	22	218007	100.00%	36.3/min	2.18	0.08
FindByld C...	203	79499	67047	151679	162737	162761	23	266603	100.00%	13.9/min	0.91	0.04
Find User...	115	38966	7763	111052	117486	162610	30	162756	100.00%	7.9/min	0.60	0.03
FindByld U...	83	32081	4950	162732	162737	162749	20	162755	100.00%	5.7/min	0.47	0.02
Create Sc...	81	16191	3731	67092	67166	67178	29	150539	100.00%	5.6/min	0.46	0.04
Update S...	67	4428	2810	4444	11435	19345	27	67090	100.00%	4.6/min	0.39	0.02
Find Sche...	66	3398	3044	5255	6057	6062	32	9470	100.00%	8.5/min	0.71	0.03
FindByld S...	66	3594	2935	5762	9518	13336	43	13338	100.00%	8.5/min	0.73	0.04
Update S...	66	3158	3127	5423	6050	6052	32	6053	100.00%	8.5/min	0.72	0.04
TOTAL	31578	112762	130912	131151	153421	217926	20	306012	81.43%	29.9/sec	75.36	3.38

Figura A.15 – Rotina com 4000 Threads de Usuários 3/3. Fonte: Elaborado pelo autor

A.2 Análise de Desempenho Arquitetura Microserviços

A.2.1 10 Threads

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received ...	Sent KB/sec
User Regi...	10	154	115	266	266	374	81	374	0.00%	9.9/sec	6.62	3.07
User Login	10	71	65	97	97	109	43	109	0.00%	14.5/sec	10.84	4.41
User Addr...	10	86	73	132	132	135	36	135	0.00%	15.8/sec	9.27	11.09
Cleaner R...	10	285	261	389	389	395	151	395	0.00%	14.7/sec	10.60	4.60
Cleaner L...	10	132	139	155	155	169	64	169	0.00%	21.4/sec	17.03	6.55
Update Cl...	10	188	179	198	198	271	161	271	0.00%	17.5/sec	12.88	14.88
Find Users	10	157	148	199	199	224	108	224	0.00%	22.2/sec	42.84	9.87
Find Clea...	10	174	163	224	224	228	143	228	0.00%	17.6/sec	83.07	7.74
FindByld U...	10	145	141	193	193	214	86	214	0.00%	18.2/sec	8.15	8.54
FindByld C...	10	162	149	226	226	238	85	238	0.00%	20.7/sec	15.19	9.60
Find User...	10	108	123	136	136	150	55	150	0.00%	31.9/sec	104.02	15.29
FindByld U...	10	80	84	114	114	128	32	128	0.00%	38.0/sec	22.09	19.12
Create Sc...	10	177	175	207	207	213	149	213	0.00%	33.6/sec	23.89	23.23
Update S...	10	174	170	203	203	209	130	209	0.00%	31.5/sec	22.33	16.14
Find Sche...	10	493	554	577	577	595	215	595	0.00%	13.7/sec	81.08	6.16
FindByld S...	10	198	172	337	337	343	134	343	0.00%	16.4/sec	18.54	7.62
Update S...	10	75	70	102	102	104	45	104	0.00%	26.9/sec	19.06	13.97
TOTAL	170	168	149	266	389	568	32	595	0.00%	49.8/sec	73.51	24.09

Figura A.16 – Rotina com 10 Threads de Usuários 1/3. Fonte: Elaborado pelo autor

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received ...	Sent KB/sec
User Regi...	10	98	85	126	126	152	67	152	0.00%	9.9/sec	6.64	3.07
User Login	10	60	62	74	74	84	41	84	0.00%	10.9/sec	8.10	3.30
User Addr...	10	62	59	84	84	90	39	90	0.00%	10.9/sec	6.40	7.65
Cleaner R...	10	88	72	112	112	144	69	144	0.00%	10.2/sec	7.36	3.20
Cleaner L...	10	55	47	71	71	78	42	78	0.00%	10.9/sec	8.68	3.34
Update Cl...	10	90	86	118	118	156	40	156	0.00%	10.7/sec	7.85	9.07
Find Users	10	72	70	101	101	105	39	105	0.00%	10.8/sec	18.22	4.81
Find Clea...	10	84	76	109	109	195	48	195	0.00%	9.4/sec	32.22	4.14
FindByld U...	10	80	76	97	97	157	45	157	0.00%	9.4/sec	4.20	4.40
FindByld C...	10	80	81	105	105	130	34	130	0.00%	9.3/sec	6.84	4.32
Find User...	10	80	72	108	108	136	39	136	0.00%	9.9/sec	29.24	4.73
FindByld U...	10	76	75	109	109	119	35	119	0.00%	10.4/sec	6.07	5.25
Create Sc...	10	84	68	116	116	135	45	135	0.00%	10.2/sec	7.23	7.03
Update S...	10	86	77	123	123	128	61	128	0.00%	10.5/sec	7.42	5.36
Find Sche...	10	118	120	149	149	152	83	152	0.00%	10.0/sec	39.55	4.50
FindByld S...	10	99	94	140	140	185	38	185	0.00%	10.9/sec	12.38	5.09
Update S...	10	79	64	120	120	136	35	136	0.00%	12.3/sec	8.74	6.41
TOTAL	170	82	76	126	139	157	34	195	0.00%	75.9/sec	95.07	36.72

Figura A.17 – Rotina com 10 Threads de Usuários 2/3. Fonte: Elaborado pelo autor

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received ...	Sent KB/sec
User Regi...	10	93	79	115	115	116	65	116	0.00%	9.9/sec	6.64	3.07
User Login	10	53	47	71	71	79	38	79	0.00%	10.6/sec	7.89	3.21
User Addr...	10	67	66	76	76	85	54	85	0.00%	10.4/sec	6.08	7.28
Cleaner R...	10	101	106	114	114	150	65	150	0.00%	9.8/sec	7.06	3.07
Cleaner L...	10	61	51	78	78	105	40	105	0.00%	10.4/sec	8.33	3.20
Update Cl...	10	106	80	134	134	258	66	258	0.00%	8.6/sec	6.32	7.31
Find Users	10	95	57	130	130	237	46	237	0.00%	8.8/sec	14.44	3.89
Find Clea...	10	101	100	113	113	167	55	167	0.00%	8.3/sec	29.14	3.67
FindById U...	10	98	98	116	116	194	30	194	0.00%	8.3/sec	3.70	3.88
FindById C...	10	83	79	108	108	146	39	146	0.00%	8.2/sec	6.03	3.81
Find User...	10	92	81	122	122	133	70	133	0.00%	8.1/sec	23.82	3.85
FindById U...	10	91	82	102	102	231	37	231	0.00%	8.6/sec	4.98	4.31
Create Sc...	10	85	78	108	108	142	58	142	0.00%	8.6/sec	6.11	5.94
Update S...	10	71	62	86	86	100	52	100	0.00%	9.2/sec	6.53	4.72
Find Sche...	10	121	108	129	129	255	75	255	0.00%	8.8/sec	34.34	3.96
FindById S...	10	86	87	100	100	170	47	170	0.00%	9.3/sec	10.59	4.35
Update S...	10	91	59	186	186	250	32	250	0.00%	9.9/sec	6.99	5.12
TOTAL	170	88	80	123	150	250	30	258	0.00%	72.9/sec	91.20	35.26

Figura A.18 – Rotina com 10 Threads de Usuários 3/3. Fonte: Elaborado pelo autor

A.2.2 100 Threads

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received ...	Sent KB/sec
User Regi...	100	2507	2502	2634	2664	2711	2294	2722	0.00%	29.9/sec	20.08	9.30
User Login	100	449	446	792	812	814	42	817	0.00%	54.5/sec	40.64	16.54
User Addr...	100	981	919	1362	1453	1571	501	1601	0.00%	40.0/sec	23.45	28.06
Cleaner R...	100	1683	1885	2182	2193	2238	722	2254	0.00%	24.9/sec	17.97	7.81
Cleaner L...	100	526	508	802	817	1626	144	1627	0.00%	26.1/sec	20.81	8.01
Update Cl...	100	1627	1401	2356	2405	2811	1112	2850	0.00%	20.9/sec	15.37	17.76
Find Users	100	637	545	1025	1030	1066	345	1075	0.00%	28.9/sec	55.76	12.85
Find Clea...	100	706	770	991	1027	1044	258	1045	0.00%	25.1/sec	120.39	11.05
FindById U...	100	649	572	1045	1081	1103	264	1155	0.00%	24.8/sec	11.11	11.65
FindById C...	100	637	725	860	877	879	177	907	0.00%	29.0/sec	21.35	13.50
Find User...	100	770	854	1007	1039	1050	269	1086	0.00%	27.7/sec	254.55	13.25
FindById U...	100	514	552	764	788	899	225	944	0.00%	35.4/sec	20.55	17.79
Create Sc...	100	1085	1216	1386	1416	1507	311	1508	0.00%	27.1/sec	19.29	18.76
Update S...	100	1451	1540	2027	2058	2069	393	2139	0.00%	20.2/sec	14.33	10.36
Find Sche...	100	2152	2521	2955	3015	3096	362	3104	0.00%	13.9/sec	160.44	6.25
FindById S...	100	1117	1130	1704	1840	1864	490	1864	0.00%	13.7/sec	15.47	6.36
Update S...	100	802	373	2347	2385	2398	29	2398	0.00%	16.0/sec	11.34	8.31
TOTAL	1700	1076	853	2347	2531	2880	29	3104	0.00%	84.7/sec	183.05	40.99

Figura A.19 – Rotina com 100 Threads de Usuários 1/3. Fonte: Elaborado pelo autor

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received ...	Sent KB/sec
User Regi...	100	2690	2724	2812	2853	2895	2362	2904	0.00%	28.7/sec	19.28	8.93
User Login	100	543	576	955	965	995	39	995	0.00%	47.8/sec	35.70	14.53
User Addr...	100	847	896	1024	1075	1089	566	1105	0.00%	39.2/sec	22.94	27.45
Cleaner R...	100	2028	2328	2474	2493	2530	173	2543	0.00%	24.7/sec	17.85	7.75
Cleaner L...	100	534	522	816	876	888	45	1943	0.00%	21.1/sec	16.79	6.46
Update Cl...	100	1582	1367	2759	2852	2894	197	2906	0.00%	18.9/sec	13.89	16.05
Find Users	100	708	665	963	969	1152	274	1158	0.00%	16.5/sec	31.80	7.33
Find Clea...	100	635	636	1009	1014	1023	253	2248	0.00%	15.3/sec	73.31	6.76
FindById U...	100	678	760	950	968	995	238	995	0.00%	21.2/sec	9.49	9.95
FindById C...	100	748	890	1012	1014	1019	249	1022	0.00%	19.5/sec	14.35	9.07
Find User...	100	727	765	887	908	940	259	948	0.00%	19.7/sec	180.74	9.41
FindById U...	100	603	551	908	925	954	263	958	0.00%	21.5/sec	12.51	10.83
Create Sc...	100	1307	1407	2073	2090	2092	273	2093	0.00%	17.7/sec	12.62	12.28
Update S...	100	1220	1361	1618	1803	1911	312	1931	0.00%	15.5/sec	11.00	7.95
Find Sche...	100	2181	2303	2849	2901	2961	905	2965	0.00%	11.1/sec	127.08	4.99
FindById S...	100	1151	994	1967	2405	2627	422	2628	0.00%	11.8/sec	13.42	5.52
Update S...	100	556	190	1921	1944	2206	93	2209	0.00%	13.1/sec	9.28	6.80
TOTAL	1700	1102	889	2427	2734	2854	39	2965	0.00%	83.9/sec	180.47	40.56

Figura A.20 – Rotina com 100 Threads de Usuários 2/3. Fonte: Elaborado pelo autor

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received ...	Sent KB/sec
User Regi...	100	2530	2566	2680	2696	2700	2089	2706	0.00%	30.1/sec	20.21	9.35
User Login	100	530	500	934	945	1050	76	1057	0.00%	44.1/sec	32.93	13.40
User Addr...	100	1073	1066	1534	1639	1654	421	1710	0.00%	36.7/sec	21.50	25.73
Cleaner R...	100	1631	1931	2055	2076	2109	299	2135	0.00%	25.4/sec	18.36	7.97
Cleaner L...	100	481	436	832	858	871	77	905	0.00%	25.3/sec	20.12	7.74
Update Cl...	100	1722	1612	2398	2491	2510	1181	2592	0.00%	19.5/sec	14.37	16.60
Find Users	100	573	512	949	971	976	230	982	0.00%	28.1/sec	54.09	12.47
Find Clea...	100	723	798	1188	1230	1238	296	1238	0.00%	27.0/sec	129.62	11.89
FindById U...	100	677	684	1041	1052	1086	276	1086	0.00%	23.0/sec	10.30	10.80
FindById C...	100	617	561	859	920	1123	306	1139	0.00%	22.0/sec	16.19	10.23
Find User...	100	667	654	936	963	1015	375	1036	0.00%	26.0/sec	238.88	12.43
FindById U...	100	559	514	915	931	971	295	981	0.00%	23.3/sec	13.57	11.74
Create Sc...	100	1003	922	1364	1432	1494	313	1497	0.00%	21.1/sec	15.03	14.62
Update S...	100	1238	1253	1720	1733	1792	536	1796	0.00%	16.7/sec	11.81	8.53
Find Sche...	100	1996	2093	2753	2801	2841	668	2855	0.00%	14.2/sec	163.26	6.36
FindById S...	100	1089	1143	1421	1448	1540	504	1555	0.00%	14.4/sec	16.37	6.73
Update S...	100	552	221	1608	1700	1709	37	1845	0.00%	16.2/sec	11.48	8.41
TOTAL	1700	1039	858	2145	2512	2698	37	2855	0.00%	87.6/sec	189.16	42.36

Figura A.21 – Rotina com 100 Threads de Usuários 3/3. Fonte: Elaborado pelo autor

A.2.3 1000 Threads

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received ...	Sent KB/sec
User Regi...	1000	28970	29432	30081	30249	30314	18258	30339	0.00%	32.1/sec	21.52	9.96
User Login	1000	5875	6027	10164	10742	11227	36	11376	0.00%	41.2/sec	30.75	12.52
User Addr...	1000	10447	10138	14419	14617	15069	2609	15255	0.00%	37.8/sec	22.17	26.53
Cleaner R...	1000	16314	20887	21294	21325	21501	1368	21663	0.00%	25.1/sec	18.11	7.87
Cleaner L...	1000	3333	2747	6658	7210	13660	107	15841	0.00%	23.9/sec	19.01	7.31
Update Cl...	1000	21559	11990	66522	66798	67220	1776	79061	0.00%	8.3/sec	6.09	7.03
Find Users	1000	60829	77006	85705	91856	92574	3760	92635	0.00%	5.6/sec	10.79	2.49
Find Clea...	1000	49147	59752	64654	64847	71050	5743	71252	0.00%	4.4/sec	21.26	1.95
FindById U...	1000	56036	60434	66987	68645	70699	5293	78565	0.00%	3.7/sec	1.67	1.75
FindById C...	1000	54272	57295	65324	67761	72804	5246	72822	0.00%	3.1/sec	2.30	1.45
Find User...	1000	60159	60258	66483	66567	68234	36209	68520	0.00%	2.7/sec	24.63	1.28
FindById U...	1000	57859	57615	61751	61864	63511	52462	64728	0.00%	2.6/sec	1.49	1.29
Create Sc...	1000	54172	54424	58041	61081	68234	45349	68455	0.00%	2.7/sec	1.89	1.84
Update S...	1000	50266	47577	61856	63667	63825	44410	63993	0.00%	2.8/sec	1.98	1.43
Find Sche...	1000	60670	53134	86396	89985	98183	46110	102518	0.00%	2.5/sec	28.76	1.12
FindById S...	1000	32943	32316	55373	61622	62983	3481	63370	0.00%	2.9/sec	3.25	1.34
Update S...	1000	17099	10540	48095	56517	56795	32	56824	0.00%	3.4/sec	2.45	1.79
TOTAL	17000	37644	46643	64606	72791	86436	32	102518	0.00%	24.2/sec	52.38	11.73

Figura A.22 – Rotina com 1000 Threads de Usuários 1/3. Fonte: Elaborado pelo autor

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
User Register	1000	30789	30580	31582	31688	31757	23107	31790	0.00%	30.6/sec	20.53	9.50
User Login	1000	6061	6167	10519	11076	11534	40	11734	0.00%	47.0/sec	35.05	14.27
User Addresses	1000	12804	13055	16746	16868	17889	3148	18049	0.00%	34.7/sec	20.36	24.36
Cleaner Register	1000	21588	24349	24645	24859	25003	52	25026	0.00%	22.6/sec	16.34	7.10
Cleaner Login	1000	4959	4883	9129	9630	15482	42	18680	0.00%	19.7/sec	15.68	6.03
Update Cleaner	1000	12223	13238	13654	15913	16587	37	18864	0.00%	14.4/sec	10.56	12.20
Find Users	1000	54264	65795	66833	74445	80139	5882	81537	0.00%	7.1/sec	13.66	3.15
Find Cleaners	1000	68498	63782	71621	75568	76527	19291	80148	0.00%	5.3/sec	25.28	2.32
FindById Users	1000	63890	67380	81952	81979	82006	5469	82145	0.00%	4.1/sec	1.82	1.91
FindById Cleaners	1000	55101	58352	75933	76032	76079	4949	76112	0.00%	3.4/sec	2.47	1.56
Find UserAddresses	1000	58468	54816	56809	58383	60036	4336	61225	0.00%	2.9/sec	26.54	1.38
FindById UserAddr...	1000	50921	52069	55825	58389	58642	6039	58697	0.00%	2.5/sec	1.48	1.28
Create Schedule	1000	53580	52395	58952	62108	65206	40553	80131	0.00%	2.3/sec	1.60	1.56
Update Schedule ...	1000	56500	57602	62771	65766	69254	46699	71312	0.00%	2.3/sec	1.59	1.15
Find Schedules	1000	76280	53403	129876	138617	143289	46248	143419	0.00%	2.0/sec	22.59	0.88
FindById Schedules	1000	42330	40396	64692	71113	78567	12806	92155	0.00%	2.2/sec	2.45	1.01
Update Schedule ...	1000	11859	4223	26445	54727	56083	33	58579	0.00%	2.5/sec	1.76	1.29
TOTAL	17000	39866	48974	66548	76022	120498	33	143419	0.00%	23.0/sec	49.66	11.12

Figura A.23 – Rotina com 1000 Threads de Usuários 2/3. Fonte: Elaborado pelo autor

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
User Register	1000	30829	31052	31701	31873	32039	23308	32131	0.00%	30.7/sec	20.60	9.54
User Login	1000	6984	7261	11648	12133	12625	41	13048	0.00%	45.7/sec	34.11	13.88
User Addresses	1000	11312	11464	15786	16692	16966	2520	16974	0.00%	33.9/sec	19.86	23.76
Cleaner Regis...	1000	17773	22800	22981	22991	23034	308	23120	0.00%	22.9/sec	16.54	7.18
Cleaner Login	1000	5043	5025	9152	14891	16395	50	16541	0.00%	20.2/sec	16.12	6.20
Update Cleaner	1000	31753	35673	59141	59318	59452	47	59539	0.00%	9.3/sec	6.84	7.90
Find Users	1000	47834	59967	67835	67890	68053	3387	70679	0.00%	5.7/sec	11.05	2.55
Find Cleaners	1000	49748	58251	62921	65905	66383	17911	66420	0.00%	4.6/sec	21.85	2.00
FindById Clea...	1000	57264	64771	79934	80028	81441	2153	81494	0.00%	3.5/sec	1.58	1.66
FindById Clea...	1000	54499	63628	71322	74756	76761	6167	78310	0.00%	3.0/sec	2.21	1.39
Find UserAddr...	1000	56511	54252	67039	68943	79937	44090	80129	0.00%	2.6/sec	24.06	1.25
FindById User...	1000	57760	57700	61180	66887	70525	49288	70625	0.00%	2.5/sec	1.48	1.28
Create Sched...	1000	50146	47678	59163	59848	62610	39384	65816	0.00%	2.6/sec	1.86	1.81
Update Sched...	1000	49897	43323	67133	85049	85080	39262	85125	0.00%	2.7/sec	1.94	1.40
Find Schedules	1000	52509	45251	74690	83768	85303	40854	85436	0.00%	2.6/sec	30.12	1.17
FindById Sche...	1000	35391	31915	52923	53691	54616	4929	54811	0.00%	3.1/sec	3.51	1.44
Update Sched...	1000	28476	24473	58723	59029	60264	34	60304	0.00%	3.7/sec	2.61	1.91
TOTAL	17000	37866	42400	64797	69307	81431	34	85436	0.00%	24.3/sec	52.42	11.74

Figura A.24 – Rotina com 1000 Threads de Usuários 3/3. Fonte: Elaborado pelo autor

A.2.4 2000 Threads

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
User Register	2000	58468	58395	60394	60581	61350	39235	61534	0.00%	32.1/sec	21.57	9.98
User Login	2000	12067	11923	20307	21687	22542	37	22820	0.00%	43.8/sec	32.68	13.30
User Addresses	2000	26631	25944	33630	33738	36861	4223	40371	0.00%	30.9/sec	18.08	21.64
Cleaner Regis...	2000	151910	179300	206291	208028	208161	63	214808	0.00%	7.9/sec	5.70	2.48
Cleaner Login	2000	3693	2668	7914	8415	23179	38	38592	0.00%	8.0/sec	6.34	2.44
Update Cleaner	2000	123973	128296	154000	178775	180514	35	183227	0.00%	5.2/sec	3.79	4.38
Find Users	2000	115581	123464	128455	134759	136765	12269	139758	0.00%	3.9/sec	7.50	1.73
Find Cleaners	2000	113343	116217	127606	128379	129283	38328	130660	0.00%	3.3/sec	15.80	1.45
FindById Users	2000	115252	110601	112053	177477	178180	105678	178263	0.00%	3.0/sec	1.35	1.41
FindById Clea...	2000	107346	106301	111093	122359	125817	103099	125927	0.00%	3.4/sec	2.48	1.57
Find UserAddr...	2000	110709	109719	111922	124907	127015	103679	127091	0.00%	3.4/sec	31.71	1.65
FindById User...	2000	106490	106184	109086	111598	113348	102207	113855	0.00%	3.6/sec	2.07	1.79
Create Sched...	2000	107635	107395	110457	111071	112407	99987	112764	0.00%	3.6/sec	2.58	2.51
Update Sched...	2000	99715	99465	103036	106552	107807	94101	108968	0.00%	3.7/sec	2.63	1.90
Find Schedules	2000	305826	308128	501641	505687	512671	97492	512751	63.55%	2.2/sec	11.59	0.65
FindById Sche...	1298	169556	124636	359728	409171	414614	1728	415809	61.17%	1.6/sec	3.10	0.34
Update Sched...	575	47791	3438	106825	192750	193487	112	261807	12.35%	50.5/min	0.78	0.38
TOTAL	31873	105570	107179	178033	208090	491820	35	512751	6.70%	17.8/sec	33.63	8.22

Figura A.25 – Rotina com 2000 Threads de Usuários 1/3. Fonte: Elaborado pelo autor

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
User Register	2000	58534	58644	60547	60837	60875	39566	61472	0.00%	32.2/sec	21.61	10.00
User Login	2000	14276	15371	23346	25172	25963	40	26293	0.00%	41.6/sec	31.01	12.62
User Addresses	2000	52540	64909	68109	72672	73396	4994	73656	0.00%	17.6/sec	10.31	12.34
Cleaner Regis...	2000	594700	902905	1035194	1035380	1036207	2671	1046773	75.10%	1.8/sec	2.73	0.30
Cleaner Login	1044	248837	64785	807998	890593	954161	58	954325	52.30%	56.9/min	1.26	0.18
Update Cleaner	728	169804	7880	514186	598189	892953	2725	1001760	35.71%	39.7/min	0.86	0.38
Find Users	497	81771	72587	72920	104274	408559	47510	953763	6.04%	27.2/min	0.89	0.19
Find Cleaners	471	789734	963759	1011196	1020302	1020488	24320	1020579	99.79%	27.5/min	0.64	0.11
FindById Users	259	309964	281441	588553	783562	837901	86	856146	100.00%	15.5/min	0.59	0.00
FindById Clea...	60	354721	246396	598222	598226	598247	236956	954257	100.00%	3.8/min	0.16	0.00
Find UserAddr...	2	246226	246184	246268	246268	246268	246184	246268	100.00%	29.2/hour	0.02	0.00
TOTAL	11061	211388	59771	1009866	1020609	1035357	40	1046773	28.29%	9.5/sec	10.03	3.33

Figura A.26 – Rotina com 2000 Threads de Usuários 2/3. Fonte: Elaborado pelo autor

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
User Register	2000	58489	59763	61248	61438	61552	366	61690	0.00%	31.8/sec	21.30	9.86
User Login	2000	12375	12191	20639	22084	37517	39	45265	0.00%	23.5/sec	17.56	7.15
User Addresses	2000	25640	28417	32983	33262	33995	4500	35290	0.00%	31.0/sec	18.16	21.73
Cleaner Regis...	2000	140666	145342	185013	185036	191322	65	191555	0.00%	8.7/sec	6.31	2.74
Cleaner Login	2000	3580	2333	7796	8372	22664	39	37340	0.00%	8.9/sec	7.08	2.73
Update Cleaner	2000	126435	137684	143844	157408	168119	62	168247	0.00%	5.4/sec	4.00	4.63
Find Users	2000	114286	121034	129394	136536	138378	21614	140159	0.00%	4.1/sec	7.82	1.80
Find Cleaners	2000	113940	120874	123825	124534	126145	39659	127561	0.00%	3.4/sec	16.49	1.51
FindById Users	2000	115722	111574	115681	153747	168078	106665	168277	0.00%	3.1/sec	1.37	1.43
FindById Clea...	2000	110179	108002	113047	133193	135639	102155	139158	0.00%	3.3/sec	2.43	1.53
Find UserAddr...	2000	109714	108804	112124	118094	127582	101923	127782	0.00%	3.4/sec	31.69	1.65
FindById User...	2000	104629	103526	108906	112325	118180	100158	119960	0.00%	3.5/sec	2.05	1.78
Create Sched...	2000	104754	104213	107604	113065	115492	99289	115570	0.00%	3.6/sec	2.58	2.51
Update Sche...	2000	95574	94173	101706	106807	108051	91177	108241	0.00%	3.7/sec	2.65	1.91
Find Schedules	2000	321649	327093	545092	548886	552595	92324	554839	62.70%	2.1/sec	11.31	0.61
FindById Sche...	1288	165372	104160	404963	456740	460885	82	461745	54.66%	1.6/sec	2.78	0.37
Update Sche...	655	78487	34111	186095	214067	400601	3052	400688	25.95%	54.3/min	0.82	0.41
TOTAL	31943	105803	105822	145647	214316	537517	39	554839	6.66%	17.6/sec	33.24	8.14

Figura A.27 – Rotina com 2000 Threads de Usuários 3/3. Fonte: Elaborado pelo autor

A.2.5 4000 Threads

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
User Register	4000	118797	114731	135425	136191	138110	28480	140882	0.90%	28.3/sec	18.93	8.79
User Login	4000	18108	20676	26132	27933	29487	47	60246	0.90%	28.9/sec	21.48	8.77
User Addresses	4000	311328	344660	372771	380829	385389	307	392345	6.98%	9.2/sec	5.29	6.43
Cleaner Regis...	3975	188989	188373	235281	249355	302371	2966	304816	17.89%	7.0/sec	7.28	1.81
Cleaner Login	3815	520610	471202	1053945	1172565	1245005	77	1245089	69.10%	2.1/sec	2.87	0.39
Update Cleaner	3168	152321	130295	160662	298559	1114979	116	1226341	88.42%	1.7/sec	3.88	0.20
Find Users	2800	166611	131051	299370	431696	891800	225	1198441	93.36%	1.5/sec	3.74	0.05
Find Cleaners	2387	160333	131058	131090	163384	1129515	517	1195292	94.26%	1.4/sec	3.88	0.05
FindById Users	1304	139773	131061	131071	131075	486368	7170	1179391	99.77%	50.4/min	2.27	0.00
FindById Clea...	1153	132351	131060	131072	131083	131114	86019	896381	99.74%	45.0/min	2.05	0.00
Find UserAddr...	1032	131353	131058	131074	131081	131103	128886	484885	100.00%	44.8/min	2.05	0.00
FindById User...	916	130934	131043	131069	131073	131095	86015	149504	100.00%	1.0/sec	2.79	0.00
Create Sched...	837	130899	131038	131068	131072	131096	128911	131101	100.00%	1.0/sec	2.85	0.00
Update Sche...	718	130933	131036	131069	131072	131111	128978	131153	100.00%	1.1/sec	3.07	0.00
Find Schedules	218	131068	131042	131066	131073	131095	128906	149508	100.00%	23.7/min	1.08	0.00
FindById Sche...	3	130991	130969	131050	131050	131050	130995	131050	100.00%	27.5/hour	0.02	0.00
Update Sche...	1	131149	131149	131149	131149	131149	131149	131149	100.00%	27.4/hour	0.02	0.00
TOTAL	34327	194747	131042	362797	688120	1123353	47	1245089	51.09%	16.6/sec	26.71	3.64

Figura A.28 – Rotina com 4000 Threads de Usuários 1/3. Fonte: Elaborado pelo autor

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
User Register	4000	118527	116997	133435	135076	137284	26097	138757	0.15%	28.5/sec	19.10	8.79
User Login	4000	20815	22590	29878	30368	32394	46	1906499	0.18%	2.0/sec	1.49	0.60
User Addresses	3999	380808	444881	461206	462605	466272	317	468970	11.10%	7.7/sec	4.42	5.41
Cleaner Regis...	3999	405736	188926	1432225	1447635	1704866	4322	1756704	80.75%	2.1/sec	4.13	0.17
Cleaner Login	3391	180880	131061	307174	354449	1521100	104	1773554	88.91%	1.8/sec	4.18	0.07
Update Cleaner	3318	134907	131021	312020	369355	412493	104	987459	95.54%	1.8/sec	4.34	0.13
Find Users	3308	1215424	131017	1048762	1050878	1052956	5	1535274	99.64%	2.2/sec	5.56	0.04
Find Cleaners	2904	55194	1131	131066	131073	309616	6	1052920	100.00%	2.1/sec	5.73	0.00
FindById Users	2902	15148	83	1284	131073	202964	6	1052967	100.00%	2.5/sec	6.74	0.01
FindById Clea...	2890	5632	84	1149	3105	131071	6	987058	100.00%	2.9/sec	8.00	0.00
Find UserAddr...	2886	221	71	285	1091	3231	6	15419	100.00%	30.6/sec	83.63	0.00
FindById User...	2886	233	71	296	1108	3230	6	7304	100.00%	30.9/sec	84.35	0.00
Create Sched...	2886	153	61	241	345	1236	5	15419	100.00%	31.0/sec	84.54	0.00
Update Sche...	2886	166	56	204	365	2232	5	7306	100.00%	31.0/sec	84.70	0.00
Find Schedules	2886	115	54	177	232	1191	6	7285	100.00%	31.1/sec	85.03	0.00
FindById Sche...	2886	92	47	158	199	1063	5	6375	100.00%	31.3/sec	85.38	0.00
Update Sche...	2886	73	41	146	182	1024	5	3215	100.00%	31.3/sec	85.59	0.00
TOTAL	54913	103799	1063	351036	447627	1426008	5	1906499	76.60%	27.1/sec	58.74	2.95

Figura A.29 – Rotina com 4000 Threads de Usuários 2/3. Fonte: Elaborado pelo autor

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
User Register	4000	120824	127737	142437	144150	321887	34493	370240	5.70%	10.8/sec	7.09	3.33
User Login	4000	423663	266835	1786189	1813743	1815690	2783	1815895	59.17%	2.1/sec	3.60	0.28
User Addresses	3422	448620	131062	1506527	1541346	1543774	148	1543988	67.50%	1.9/sec	2.98	0.62
Cleaner Regis...	2864	295700	130457	834895	919366	1412085	5448	1413815	49.37%	1.6/sec	1.94	0.34
Cleaner Login	2780	320523	131069	1279246	1280975	1410582	7671	1410749	86.15%	1.5/sec	3.18	0.11
Update Cleaner	2450	391565	131067	1279490	1281094	1410685	6296	1506446	100.00%	1.3/sec	3.55	0.01
Find Users	1773	246407	131067	1017132	1019303	1070567	1385	1525969	99.77%	59.6/min	2.66	0.00
Find Cleaners	1494	130548	131065	131071	131080	131114	66905	1019187	100.00%	58.2/min	2.60	0.00
FindById Users	1381	124851	131063	131070	131072	131094	1334	1295137	100.00%	57.4/min	2.55	0.00
FindById Clea...	1149	110940	131059	131070	131074	131097	1327	1005169	100.00%	55.2/min	2.41	0.00
Find UserAddr...	784	114986	131055	131070	131074	131098	1498	1035750	100.00%	42.1/min	1.86	0.00
FindById User...	614	126348	131061	131070	131071	131094	1415	131116	100.00%	37.3/min	1.69	0.00
Create Sched...	590	127059	131058	131070	131071	131080	5422	131105	100.00%	41.3/min	1.86	0.00
Update Sche...	534	90913	99673	131066	131069	131087	3390	131101	100.00%	44.1/min	1.82	0.00
Find Schedules	172	77594	62847	131067	131069	131071	3483	131075	100.00%	17.3/min	0.73	0.00
FindById Sche...	76	61439	30208	131064	131066	131071	27946	131072	100.00%	9.8/min	0.40	0.00
Update Sche...	24	92924	131027	131066	131067	131073	27995	131073	100.00%	4.3/min	0.19	0.00
TOTAL	28107	272305	131064	780831	1281204	1786402	148	1815895	70.27%	14.7/sec	27.96	1.97

Figura A.30 – Rotina com 4000 Threads de Usuários 3/3. Fonte: Elaborado pelo autor