Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

HANIEL ROANES SILVA

ARQUITETURA DISTRIBUÍDA BASEADA EM MICROSSERVIÇOS

Assis/SP
2022

Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

HANIEL ROANES SILVA

ARQUITETURA DISTRIBUÍDA BASEADA EM MICROSSERVIÇOS

Projeto de pesquisa apresentado ao curso de Análise e desenvolvimento de sistemas do Instituto Municipal de Ensino Superior de Assis – IMESA e a Fundação Educacional do Município de Assis – FEMA, como requisito parcial à obtenção do Certificado de Conclusão.

Orientando(a): Haniel Roanes Silva
Orientador(a): Me. Guilherme de Cleva Farto

Assis/SP
2022

FICHA CATALOGRÁFICA

S586a Silva, Haniel Roanes.

Arquitetura distribuída baseada em microsserviços / Haniel Roanes Silva – Assis, SP: FEMA, 2022.

75 f.

Trabalho de Conclusão de Curso (Graduação) – Fundação Educacional do Município de Assis – FEMA, curso de Análise e Desenvolvimento de Sistemas, Assis, 2022.

Orientador: Prof. M.e Guilherme de Cleva Farto.

1. Microsserviços. 2. Arquitetura. 3. API.

CDD 005.1

Biblioteca da FEMA

ARQUITETURA DISTRIBUÍDA BASEADA EM MICROSSERVIÇOS

HANIEL ROANES SILVA

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis, como requisito do Curso de Graduação, avaliado pela seguinte comissão examinadora:

Orientador: _____
Me. Guilherme de Cleva Farto

Examinador: _____
Dr. Almir Rogério Camolesi

Assis/SP
2022

AGRADECIMENTOS

Agradeço primeiramente a Deus por ter me dado saúde, sabedoria e orientação nessa etapa da minha vida.

Gratulo também aos familiares e amigos quem sempre me apoiaram, me dando assim ânimo para seguir em frente e conquistar meus objetivos.

E por ultimo, agradeço as instruções e criticas dos professores e orientador que me guiaram e auxiliaram nesse processo de formação.

“Sonhos determinam o que você quer. Ações determinam o que você conquista.”

Aldo Novak (1962-)

RESUMO

Dentre as varias abordagens existentes para criação de sistemas, o desenvolvimento baseado em microsserviços é consideravelmente um dos mais relevantes métodos de implementação utilizados pelas empresas atualmente. Então baseando-se nesse estilo arquitetural, o objetivo deste trabalho é discorrer sobre esse modelo de arquitetura e apresentar algumas das diversas ferramentas e conceitos utilizados na construção de microsserviços.

Palavras-chave: Microsserviços; Arquitetura; API.

ABSTRACT

Among the several existing approaches for creating systems, the development based on microservices is considerably one of the most relevant implementation methods used by companies today. So, based on this architectural style, the objective of this work is to discuss this architectural model and present some of the various tools and concepts used in the construction of microservices.

Keywords: Microservices; Architecture; API.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SOA	ARQUITETURA ORIENTADA A SERVIÇOS
EAP	ESTRUTURA ANALÍTICA DO PROJETO
API	INTERFACE DE PROGRAMAÇÃO DE APLICAÇÕES
DDD	*DESENVOLVIMENTO ORIENTADO A DOMÍNIO*
ER	ENTIDADE-RELACIONAMENTO
CI	INTEGRAÇÃO CONTÍNUA
CD	ENTREGA CONTÍNUA
DB	BANCO DE DADOS
IDE	AMBIENTE DE DESENVOLVIMENTO INTEGRADO
PaaS	Plataforma como um Serviço

Sumário

1. INTRODUÇÃO

A arquitetura de software vem seguindo em constante evolução ao decorrer dos anos, e com essas evoluções, surgem novas abordagens e estilos arquiteturais para o desenvolvimento de sistemas.

Dentre esses estilos, o modelo de arquitetura baseado em microsserviços é muito utilizado para implementação de sistemas pelas suas características que envolvem serviços fracamente acoplados e independentes, que geram aplicações escaláveis, flexíveis e robustas.

Segundo Sam Newman (Monólitos para Microsserviços. 2019), esse tipo de arquitetura é baseada em uma modelagem independente dos domínios de negócio, onde eles são encapsulados e expostos em diferentes microsserviços, que são responsáveis por disponibilizarem os recursos deste negócio. Os microsserviços ainda, podem se comunicarem um com os outros por meio de redes, o que os tornam colaborativos e também uma forma de sistema distribuído.

Os microsserviços são, segundo Sam Newman (Construindo Microsserviços, 2° edição. 2021), um tipo de arquitetura orientada a serviços, com limites traçados em que a implementação independente é fundamental para eles serem indiferentes as tecnologias usadas uns pelos outros.

1.1 OBJETIVOS

O presente projeto tem como objetivo demonstrar através de um estudo de caso, o processo de desenvolvimento do *back-end* de um sistema que implemente a arquitetura de microsserviços, e ainda explorar tecnologias e conceitos utilizadas no decorrer desse processo.

1.2 JUSTIFICATIVA

Estudar a arquitetura de microsserviços pode ser complicado e levar a vários resultados diferentes que também, em alguns casos, podem ser errôneos. Ainda, com a variedade de ferramentas e tecnologias disponíveis no mercado sobre o assunto, escolher quais usar pode ser trabalhoso e cansativo.

Dito isso, esse trabalho busca amenizar essa dificuldade de busca apresentado ferramentas, tecnologias e conceitos utilizados na construção de sistemas, e ainda, oferecer uma base de como desenvolver e disponibilizar os serviços criados.

1.3 MOTIVAÇÃO

O presente projeto de pesquisa tem como motivação oferecer a interessados no estudo da arquitetura baseada em microsserviços, uma fonte para o estudo de tecnologias e conceitos utilizados nesse tema, assim também como um exemplo de como abordar do inicio ao fim, o desenvolvimento de sistemas baseados nesse estilo arquitetural.

Dito isso, com a criação dos microsserviços e de toda a infraestrutura utilizada, espera-se facilitar e ajudar estudantes ou curiosos dessa área, há implementar sistemas utilizando esse tipo de arquitetura.

1.4 PERSPECTIVAS DE CONTRIBUIÇÃO

A contribuição esperada com o desenvolvimento desse projeto, é fornecer um modelo guia que facilite o entendimento da arquitetura baseada em microsserviços, e como implementá-la, para isso são mostradas as tecnologias e conceitos que podem ser considerados uma alternativa na construção de sistemas utilizando esse tipo de abordagem.

1.5 METODOLOGIA DE PESQUISA

O objetivo final desse projeto é alcançado através de pesquisas teóricas realizadas em artigos científicos, livros, monografias, dissertações, teses, guias práticos e técnicos, livros e fontes digitais confiáveis, capacitando-se para elaboração e desenvolvimento dos microsserviços presentes na proposta de arquitetura do projeto.

No tocante ao estudo de caso definido, as regras de negócio e requisitos do sistema, foi feita uma análise referente a aplicativos para oferta e procura de serviços, aonde forma observadas soluções já existentes hoje no mercado, focando principalmente nas insatisfações dos usuários dessas plataformas, afim de definir um *software* que atenda as necessidades desses clientes de forma efetiva.

Para o desenrolar do projeto, foram realizados estudos sobre conceitos e ferramentas especificas de cada etapa na construção da aplicação, que se encontra dividida nas seguintes fases de desenvolvimento:

- Análise;
- Modelagem;
- Desenvolvimento;
- Testes;
- *Deploy*;

Cada etapa será detalhada ao decorrer deste trabalho assim como o uso das ferramentas e conceitos utilizados.

1.6 ESTRUTURA DO TRABALHO

A estrutura do corrente trabalho se encontra dívida em 6 capítulos, sendo apresentado no 1° capítulo, a introdução do trabalho acadêmico,seguido pelo 2° capitulo onde serão explorados os conceitos da arquitetura de microsserviços, e para que haja um objetivo no desenvolvimento dos microsserviços, será no 3° capítulo, apresentado um estudo de caso que fornecera os requisitos e regras de negocio utilizados pelo sistema. Em seguida, o 4°

capítulo conterá as documentações resultantes da análise feita no estudo de caso desse projeto. Logo após terá o 5° capítulo, onde será descrito um pouco sobre o as ferramentas, metodologias e conceitos utilizados para o desenvolvimento da plataforma. No 6° e último capitulo será apresentada a conclusão do projeto assim como suas propostas de melhorias e trabalhos futuros.

2. ARQUITETURA DE MICROSSERVIÇOS

A arquitetura de microsserviços é uma abordagem na qual um único sistema é composto por diversos serviços menores que são implementados de forma independente e fracamente acoplada.

Como são independentes, cada um desses serviços tem a liberdade de possuir características próprias como, a linguagem utilizada, o banco de dados que gerencia e a forma de acesso aos seus serviços.

Os microsserviços tem sua modelagem em torno de domínios de negocio, ou seja, eles são separados por contextos limitados. Isso facilita as alterações, já que os serviços são menores, e ao mesmo tempo agiliza os lançamentos de implementações ou correções, por ser necessários fazer o *deploy* de serviços menores.

Quando falamos de arquitetura de microsserviços, surgem algumas comparações comumente utilizadas que são, compará-los com um monólito ou SOA. Em relação aos monólitos, que são serviços estruturados de forma única, fortemente acoplados e limitados a uma única definição de linguagem utilizada para sua construção, o que os diferencia é justamente essa estruturação, já que uma arquitetura orientada a microsserviços, é composta por vários serviços pequenos e fracamente acoplados. Ainda, relacionando-a com a SOA, não obtemos uma comparação oposta de definições, mas sim uma forma de evolução desse modelo arquitetural, sendo que a diferenciação dos dois modelos de arquitetura se da em relação ao escopo, ou seja, a SOA tem um escopo corporativo, onde os serviços web de uma corporação tem um padrão de integração e conversação e buscam a reutilização de componentes desenvolvidos, já os microsserviços tem um escopo de aplicação, onde ela é especifica de uma parte da organização e tem implementação própria e independente de cada serviço.

2.1 VANTAGENS

A utilização de uma arquitetura baseada em microsserviços fornece inúmeros benefícios em sua correta implementação. Se combinarmos os conceitos de design orientado a do-

mínios (DDD) com o poder dos sistemas distribuídos, os microsserviços nós fornece ganhos significativos em relação a outros tipos de arquiteturas distribuídas.

2.1.1 Heterogeneidade tecnológica

Como é uma arquitetura composta por vários serviços independentes e colaborativos, isso nos permite utilizar diferentes tecnologias que atendam melhor cada serviço em si. Além disso, essa flexibilidade nos permite ainda utilizar diferentes bases de dados para cada microsserviço, o que é muito útil em certos contextos.

2.1.2 Robustez

Ao se obter as regras de negocio separada em diferentes serviços, isso resulta em um maior controle em relação a falhas, pois se um componente falhar, não necessariamente toda a aplicação falha, como acontece em um monólito. A abordagem de microsserviços nos permite isolar esse sistema com falha, e fazer com que o resto do sistema continua funcionando.

2.1.3 Escalabilidade

Com serviços menores, é possível escalá-los de forma mais simples, do que em sistemas gigantes e estruturados de forma única. Isso permite que quando necessário, seja possível dimensionar apenas os serviços necessários, deixando os outros utilizando hardwares menos poderosos.

2.1.4 Facilidade de implantação

Utilizar microsserviços, facilita muito nas implantações de sistemas, pois essa abordagem elimina a necessidade de ter que implantar toda a aplicação sempre que houver alguma alteração no código, ao invés disso, com serviços isolados, conseguimos implantá-los independentemente uns dos outros, o que traz velocidade e praticidade para esse tipo de processo.

2.1.5 Alinhamento organizacional

Gerenciar uma equipe que trabalha em um grande sistema monolítico é uns dos principais problemas nas organizações que trabalham nessa área, mas quando é implementado uma arquitetura de microsserviços, isso permite que essa equipe seja divida de acordo com com as necessidades dos microsserviços existentes, isso traz produtividade e maior controle sobre cada time.

2.1.6 Componibilidade

Com as inúmeras plataformas de acesso existentes no mercado como celulares, computadores, televisões inteligentes e entre outras, ter um sistema onde seja possível a reutilização de componentes é algo fundamental para o consumo de todos esses dispositivos.

Ao utilizarmos microsserviços, é possível construí-los de diferentes maneiras, com diferentes formas de acesso, o que nos gera uma facilidade quando em relação a reutilização e acessibilidade dos componentes.

2.2 DESVANTAGENS

Mesmo que traga muitas vantagens, ao utilizarmos uma arquitetura de microsserviços, adicionamos muitas complexidades em coisas que seriam mais simples, como por exemplo, se implementado em um monólito. Esse modelo arquitetural traz consigo alguns problemas que encontramos ao se trabalhar com sistemas distribuídos, e portante, outros tipos que seguem esse modelo também estão sujeitos aos mesmos problemas.

2.2.1 Sobrecarga de tecnologias

Como os microsserviços oferecem a opção de se utilizar diferentes linguagens de programação, banco de dados e outros tipos de tecnologias, o que os torna resilientes, se mal utilizado, também acarreta em uma grande quantidade de tecnologias sendo usadas, e isso acaba por trazer complexidades muitas vezes desnecessárias aos serviços.

2.2.2 Custo

Fica evidente que ao se adotar a arquitetura de microsserviços, serão necessários a utilização de mais recursos do que se utilizarmos por exemplo, um monólito. Isso gera custos financeiros adicionais quando se trata de hardware, pois com mais serviços, são necessários mais processos, mais redes, mais computadores e entre outros recursos.

Além do aumento dos custos financeiros, provavelmente no inicio da implementações, por se tratar de um tipo complexo de arquitetura, o tempo a se implementar as funcionalidades e componentes sera maior, pois existe todo um processo de aprendizado, principalmente para desenvolvedores iniciantes.

2.2.3 Monitoramento

O monitoramento é algo fundamental em qualquer tipo de sistema, e ao utilizar arquitetura de microsserviços, isso se torna algo bastante complexo, já que são por vezes, centenas de processos ocorrendo em serviços diferentes que se comunicam e efetuam a trocam dados. Manter uma observabilidade efetiva sobre eles pode ser um desafio no inicio, e encontrar boas soluções podem demandar tempo de pesquisa.

2.2.4 Testes

Sendo parte importante para garantir a qualidade dos softwares, os testes são fundamentais para garantir o correto funcionamento das aplicações.

Existem diversas formas de teste, os comumente utilizados como os testes de ponta a ponta, os testes manuais ou testes de integração se tornam mais difíceis de serem criados, executados e mantidos em uma arquitetura de microsserviços, pelo fato de que na maioria dos casos, há escopos muito grandes para serem testados, além de problemas de infraestrutura, como serviços não instanciados e que fazem parte do escopo de testes, problemas no limite de tempo de requisições entre os microsserviços e entre outros problemas.

Todos esses pontos devem ser levados em consideração ao se construir testes para o sistema, e isso os torna mais complexos.

3. ESTUDO DE CASO

No momento em que vivemos, segundo Campos (2020), grande parte dos serviços prestados no Brasil são autônomos, informais ou terceirizados, e mesmo com esse número de trabalhadores, encontrar um prestador de serviços qualificado e com comprovação dessa qualificação é algo complicado a se fazer.

Olhando para essa afirmação, obtém-se duas perspectivas para se discorrer, a primeira seria a dificuldade para um fornecedor em divulgar seus serviços, e a segunda seria, na visão do contratante, como confirmar com segurança a eficiência do prestador de serviços.

Uma pesquisa realizada por Malek, Oyakawa e Silva (2020) solicitou a opinião de internautas a respeito do tema e obtiveram que ¾ dos entrevistados acharam válida a ideia de criar um portal de serviços, o que também reforça a pesquisa feita por Saldanha (2020), onde ele faz referência ao marketing digital e apresenta a melhora em relação a exposição no mercado por meio da divulgação de serviços na web. Estas pesquisas demonstram que há a necessidade da criação de uma plataforma de compartilhamento e pesquisa de prestadores de serviços.

A forma mais usada para divulgação ou procura de serviços atualmente, são as redes sociais, o que pode ser uma boa ideia a princípio já que assim garante-se um grande alcance de personas, mas no meio de tantas informações que circulam nesses meios de comunicação, essas postagens ou anúncios, acabam passando despercebidos ou esquecidos pelos usuários. Do mesmo modo, caso por meio dessas postagens sejam encontrados alguns clientes, ainda é difícil comprovar a eficiência dos serviços já prestados ou provar a satisfação por parte de contratantes passados.

Como soluções alternativas, para resolução desses problemas, podemos citar alguns aplicativos ou sites que focam na divulgação de serviços como o GetNinjas, Craft e Triider, porém eles ou não contemplam algumas áreas de atuação dos prestadores, ou para ser vinculados com esses sistemas, deve ser feito um processo de contratação pela plataforma, o que limita o uso de todo interessado em divulgar seus serviços.

3.1 IMPLEMENTAÇÃO

A partir das informações apresentadas, foi criado o *back-end de* um sistema denominado Job. Para isso foram utilizados dos conceitos e tecnologias referentes a arquitetura de microsserviços, passando por cada uma das fases de desenvolvimento necessárias e fundamentais para a criação de um sistema robusto e eficiente.

4. ANÁLISE E DOCUMENTAÇÃO DO PROJETO

Neste capítulo são apresentadas as especificações dos requisitos funcionais e a modelagem de diagramas oriundos do estudo de caso.

4.1 REQUISITOS

De acordo com a análise realizada e baseando-se em funcionalidades e comentários de usuários já existentes em outros aplicativos semelhantes, foram levantados os seguintes requisitos:

- Manter Clientes;
- Manter Serviço;
- Manter Categoria;
- Manter Pagamento;
- Manter agenda;
- Manter Agendamentos
- Manter Prestador;
- Controle da agenda pelos prestadores;
- Compartilhamento dessa agenda com os clientes;
- Controle do agendamento pelo prestador e pelo cliente;
- Chats de comunicação entre cliente e prestador;
- Notificações enviadas de novas mensagens e agendamentos solicitados;
- Alertas ao prestador e cliente sobre agendamentos no dia;
- Histórico de serviços já contratados para os clientes e prestadores;
- Pagamentos devem ser feitos na plataforma (Crédito ou PIX);

Porém como foco para esse trabalhou, somente foram implementados os seguintes recursos:

- Manter Clientes;
- Manter Serviço;
- Manter Categoria;
- Manter agenda;
- Manter Agendamentos
- Manter Prestador;
- Controle da agenda pelos prestadores;
- Compartilhamento dessa agenda com os clientes;
- Controle do agendamento pelo prestador e pelo cliente;
- Histórico de serviços já contratados para os clientes e prestadores;

4.2 MAPA MENTAL

A partir dos requisitos definidos, foi criado o mapa mental a seguir com intuito de obter uma organização das ideias acerca da plataforma e registrar os possíveis recursos e funcionalidades que o software deverá oferecer.

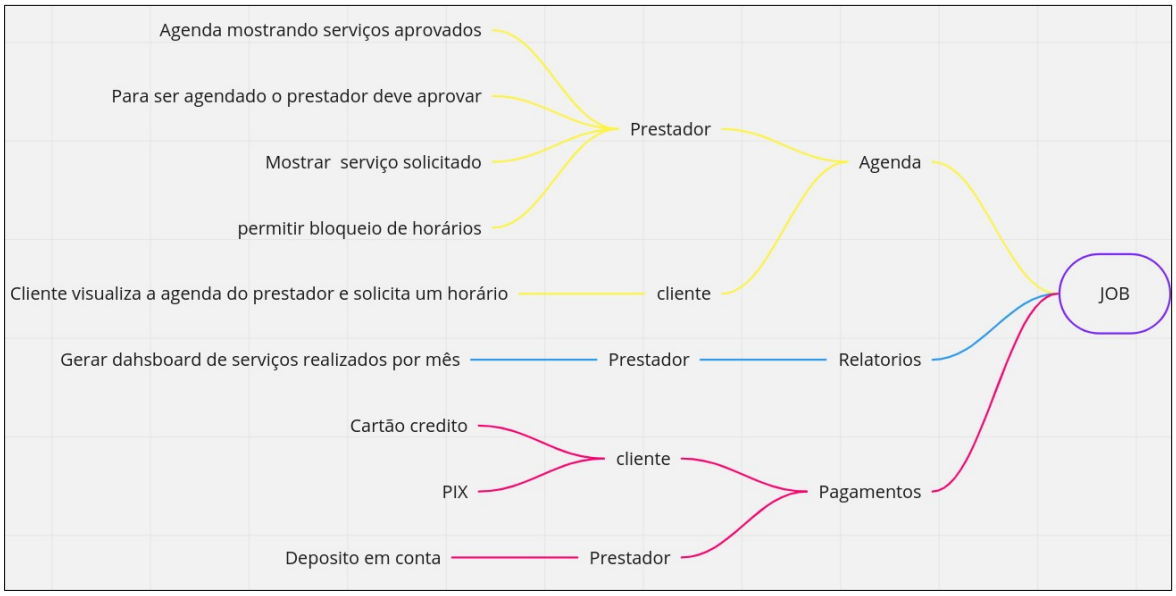

Figura 1 *- Mapa mental (lado esquerdo)*

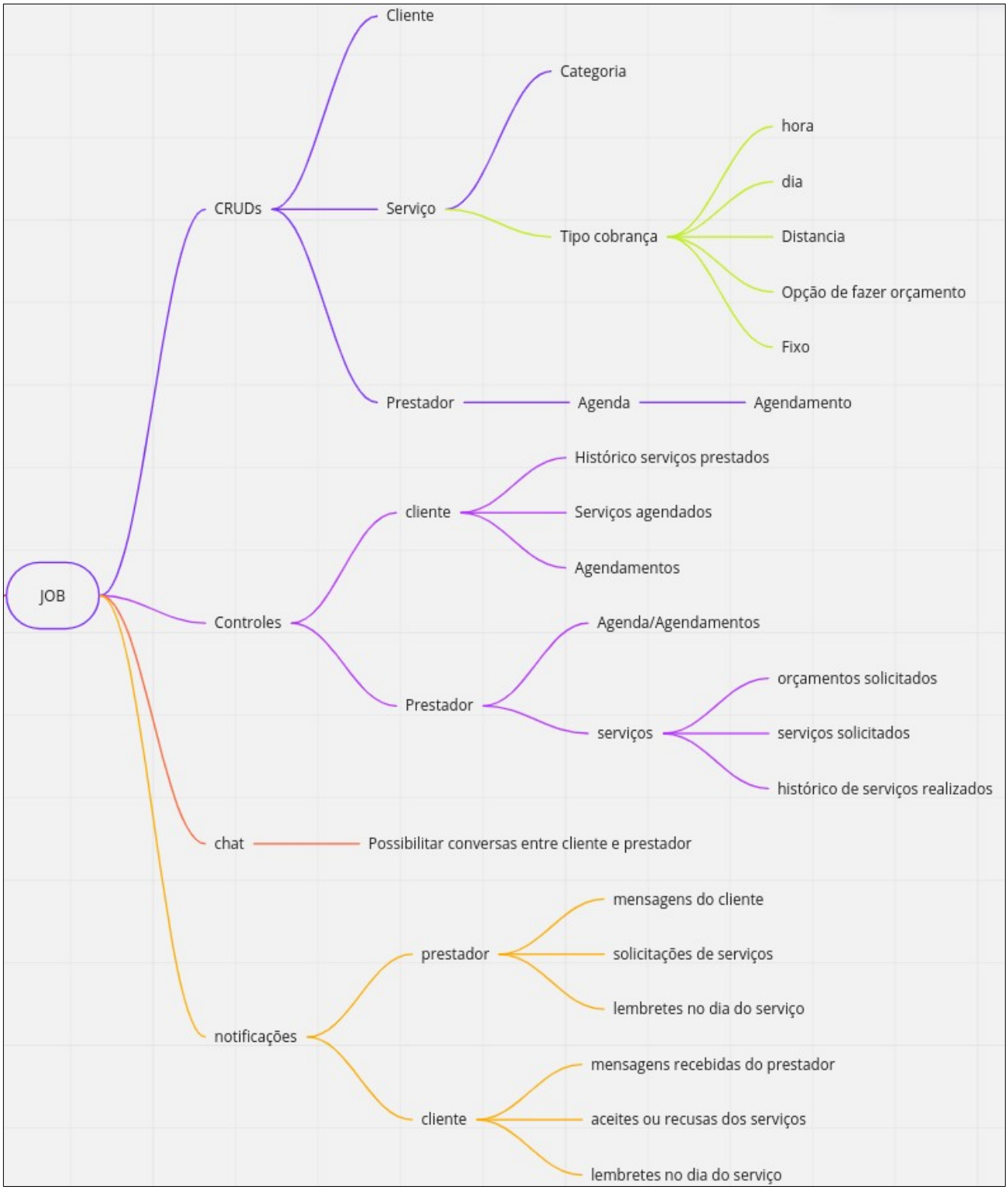

Figura 2 *- Mapa mental (lado direito)*

4.3 CASOS DE USO

Com a definição dos requisitos e recursos necessários para o funcionamento eficiente da plataforma, afim de definir os atores e suas relações foram criados os casos de uso da aplicação.

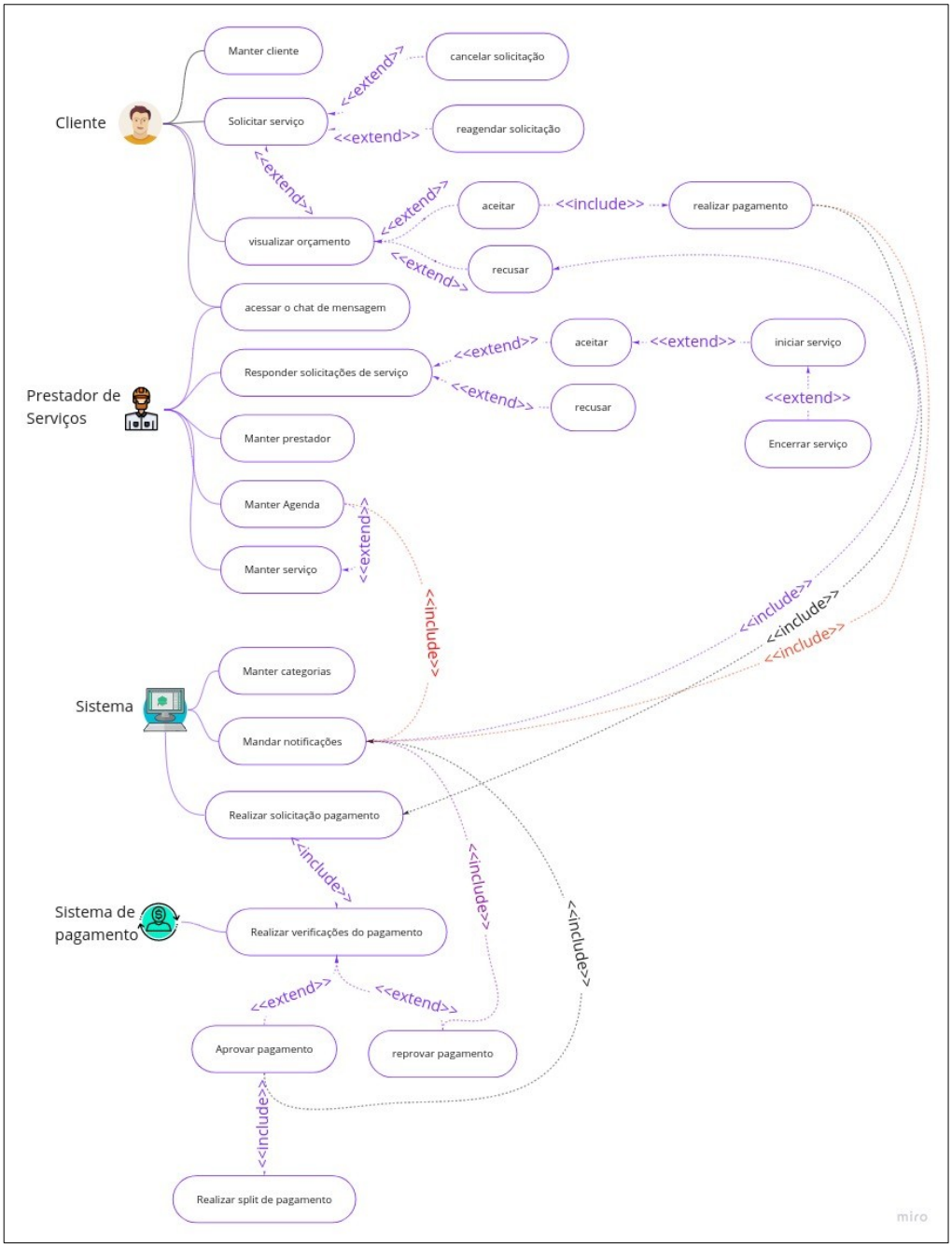

Figura 3 - Casos de uso

4.3.1 Narrativas dos casos de uso

Para melhorar o detalhamento dos fluxos da aplicação, foram elaborados as narrativas de alguns casos de uso, onde o objetivo é descrever com mais enfoque sobre os fluxos definidos, seus atores e as regras de negocio envolvidas.

- Solicitação de um serviço:

Objetivo	Atores	Pré-condições	Evento incial	Fluxo Principal	Fluxo alternativo	Fluxo de Exceção	casos de testes
O objetivo é oferecer para o cliente uma forma de solicitar o serviço que necessita.	• Cliente • Prestador de serviços • Sistema de pagamento	• Estar logado na aplicação;	O Cliente acessa a aba de busca de serviços.	1. Abre a tela com as categorias de serviços listadas; 2. O cliente seleciona a categoria desejada[A1]; 3. É apresentado uma listagem dos serviços pertencentes a categoria selecionada; 4. O cliente seleciona o serviço desejado; 5. São apresentados os Prestadores que fornecem o serviço escolhido[E1]; 6. O cliente escolhe o Prestador desejado; 7. É exibida uma tela com algumas informações do Prestador e do serviço; 8. O cliente clica em "Agendar serviço"[A2]; 9. É exibida uma agenda com os horários livres do Prestador; 10. O Cliente seleciona o horário desejado ; 11. O Cliente clica em "confirmar"; 12. É exibida uma tela com as informações do serviço, local do serviço, data selecionada, total, desconto e forma de pagamento; 13. O Cliente clica em "Enviar solicitação"[A3]; 14. A solicitação com os valores e dados do Cliente é enviada para o Sistema de pagamento que devera fazer a validação (O pagamento real só é realizado após a conclusão do serviço) [E3]; 15. A solicitação do serviço é enviada ao Prestador; 16. Após confirmada a solicitação pelo Prestador do serviço, o Cliente deverá receber um e-mail e uma notificação no celular avisando da confirmação[A4][A5][E4]; 17. Fim do caso de uso;	[A1] - Buscar serviço 1. O cliente clica no campo de pesquisa; 2. escreve o nome do serviço desejado; 3. retorna ao passo 5 do fluxo principal. [A2] - Serviço com tipo de cobrança por orçamento 1. O Cliente clica em "solicitar orçamento"; 2. Abrira uma tela com as informações solicitadas pelo Prestador para a definição de preço do serviço; 3. O Cliente preenche as informações; 4. O Cliente clica em "Solicitar orçamento"[E2]; 5. retorna ao passo 17 do fluxo principal; [A3] - Cancelar solicitação 1. O cliente clica no botão "cancelar"; 2. É solicitado a confirmação do cancelamento [E2]; 3. volta ao passo 7 do fluxo principal. [A4] - Solicitação reagendada 1. Prestador nega a solicitação do Cliente e envia uma sugestão de horário; 2. Essa sugestão é passada ao cliente por uma notificação do aplicativo; 3. O Cliente confirma a sugestão; 4. O serviço é agendado; 5. Volta ao passo 17 do fluxo principal; [A5] - Solicitação negada 1. O Prestador nega a solicitação do Cliente; 2. volta ao passo 17 do fluxo principal;	[E1] - Caso não haja nenhum prestador que ofereça o serviço, dever ser mostrado a seguinte mensagem "Infelizmente não há prestadores oferecendo esse serviço no momento, tente mudar o nome de busca." [E2] - Caso não seja confirmado o cancelamento, o Cliente permanece na tela de finalização da solicitação. [E3] - Se o sistema invalidar o pagamento, o Cliente deve ser informado do problema, e deve ser sugerido a ele para tentar com outra forma de pagamento, e a solicitação não devera ser enviada ao prestador; [E4] - Caso seja negada ou reagendada a solicitação é enviado um e-mail e uma notificação(celular) ao Ator (Cliente ou Prestador) como forma de aviso;	[T1] - Quando orçamento for enviado pelo Prestador, o Cliente deve ser notificado;

Tabela 1 *- Solicitação de serviços*

- Responder solicitação de serviço:

Objetivo	Atores	Pré-condições	Evento incial	Fluxo Principal	Fluxo alternativo	Fluxo de Exceção	casos de testes
O objetivo é oferecer para o Prestador uma forma de responder as solicitações de agendamentos.	• Prestador de serviços; • Cliente;	• Estar logado na aplicação; • Estar cadastrado como Prestador; • Ter Solicitações pendentes	No menu principal o Ator clica na aba agenda;	1. Abre-se a tela com a agenda do prestador, contendo os serviços agendados, os horários livres e os bloqueados, e acima destacadas, devem estar as solicitações de agendamento ainda não respondidas[T1]; 2. O Prestador clica em uma solicitação ainda não respondida; 3. É exibido os detalhes da solicitação; 4. O Prestador clica em "aceitar" [A1][A2]; 5. É enviada uma notificação ao cliente; 6. O Prestador é enviado a tela de controle de agenda que já deve estar atualizada; 7. fim do caso de uso;	[A1] - Recusar solicitação; 1. O Prestador clica em "recusar"; 2. É exibida a tela de confirmação; 3. O Prestador clica em "aceitar"[E1]; 4. retorna ao passo 7 do fluxo principal; [A2] - Recusar solicitação e sugerir uma outra data; 1. O Prestador clica em "sugerir outra data"; 2. É aberta a agenda do Prestador; 3. O Prestador seleciona a data desejada; 4. O Prestador clica em "enviar"[E1][T2]; 5. O Cliente é notificado; 6. O Cliente visualiza e aceita o horário sugerido[E2]; 7. O Prestador é notificado; 8. O agendamento é marcado; 9. retorna ao passo 7 do fluxo principal;	[E1] - Caso o Prestador clique em "cancelar" retorna para o passo 3 do fluxo principal; [E2] - Caso o Cliente negar a solicitação, o Prestador deve ser notificado e o agendamento não deve ser possível;	[T1] - Caso não haja nenhuma solicitação pendente, devera ser exibida a mensagem " sem solicitações pendentes no momento!" [T2] - Deve-se testar se o Cliente já não possui agendamentos nesse horário, se possuir, não deve ser possível enviar a sugestão de horário;

Tabela 2 *- Responder solicitação de serviço*

- Responder solicitação de orçamento:

Objetivo	Atores	Pré-condições	Evento incial	Fluxo Principal	Fluxo alternativo	Fluxo de Exceção	casos de testes
O objetivo é oferecer para o Prestador uma forma passar o preço quando o cliente solicita um orçamento.	• Prestador de serviços; • Cliente;	• Estar logado na aplicação; • Estar cadastrado como Prestador • Ter algum orçamento solicitado por um cliente;	No menu principal o Ator clica na aba de serviços;	1. Abre-se a tela na parte de orçamentos, com o orçamento solicitado destacado; 2. O Prestador clica no orçamento; 3. É aberto uma tela com as informações respondidas pelo Cliente; 4. O Prestador analisa o orçamento e passa seu valor; 5. O Cliente deve ser notificado da resposta do orçamento; 6. O Cliente aceita o valor passado pelo Prestador[A1] [A2]; 7. O Cliente faz o agendamento; 8. O Prestador é notificado sobre o agendamento; 9. O Prestador Confirma o agendamento[A3][A4]; 10. fim do caso de uso;	[A1] - O Cliente recusa o valor passado e o Prestador da um desconto; 1. O Cliente recusa o valor passado; 2. O Prestador é notificado; 3. O Prestador acessa o orçamento novamente; 4. O Prestador clica em "editar valor" 5. O Prestador edita o valor passado, e da um desconto; 6. retorna ao passo 5 do fluxo principal; [A2] - O Cliente recusa o valor passado; 1. O Cliente recusa o valor passado; 2. O Prestador é notificado; 3. retorna ao passo 10 do fluxo principal. [A3] - Solicitação reagendada 1. Prestador nega a solicitação do Cliente e envia uma sugestão de horário; 2. Essa sugestão é passada ao cliente por uma notificação do aplicativo; 3. O Cliente confirma a sugestão; 4. O serviço é agendado; 5. Volta ao passo 10 do fluxo principal; [A4] - Solicitação negada 1. O Prestador nega a solicitação do Cliente; 2. volta ao passo 10 do fluxo principal;	não há fluxo de exceção	Não é necessário nenhum teste;

Tabela 3 *- Responder solicitação de orçamento*

- Visualizar um orçamento:

Objetivo	Atores	Pré-condições	Evento incial	Fluxo Principal	Fluxo alternativo	Fluxo de Exceção	casos de testes
O objetivo é oferecer para o cliente uma forma de visualizar o orçamento passado ao cliente pelo prestador.	• Cliente • Prestador de serviços • Sistema de pagamento	• Estar logado na aplicação; • O cliente deve ter feito a solicitação de um orçamento;	O cliente acessa a aba serviços;	1. Abre a tela de controle/histórico de serviços, onde se encontra destacado o orçamento solicitado; 2. O cliente acessa o orçamento e verifica o valor, e a descrição dos serviços necessários; 3. O cliente aprova o orçamento e clica no botão "agendar serviço" [A4]; 4. É exibida uma agenda com os horários livres do Prestador; 5. O Cliente seleciona o horário desejado ; 6. O Cliente clica em "confirmar"; 7. É exibida uma tela com as informações do serviço, local do serviço, data selecionada, total, desconto e forma de pagamento; 8. O Cliente clica em "Enviar solicitação"[A1]; 9. A solicitação com os valores e dados do Cliente é enviada para o Sistema de pagamento que devera fazer a validação (O pagamento real só é realizado após a conclusão do serviço) [E1]; 10. A solicitação do serviço é enviada ao Prestador; 11. Após confirmada a solicitação pelo Prestador do serviço, o Cliente deverá receber um e-mail e uma notificação no celular avisando da confirmação[A2][A3][E2]; 12. Fim do caso de uso;	[A1] - Cancelar solicitação 1. O cliente clica no botão "cancelar"; 2. É solicitado a confirmação do cancelamento [E3]; 3. volta ao passo 12 do fluxo principal. [A2] - Solicitação reagendada 1. Prestador nega a solicitação do Cliente e envia uma sugestão de horário; 2. Essa sugestão é passada ao cliente por uma notificação do aplicativo; 3. O Cliente confirma a sugestão; 4. O serviço é agendado; 5. Volta ao passo 12 do fluxo principal; [A3] - Solicitação negada 1. O Prestador nega a solicitação do Cliente; 2. volta ao passo 12 do fluxo principal; [A4] - Recusar orçamento 1. O Cliente clica em "Recusar orçamento"; 2. É enviado uma notificação para o Prestador com a informação; 3. Volta ao passo 12 do fluxo principal;	[E1] - Se o sistema invalidar o pagamento, o cliente deve ser informado do problema, e deve ser sugerido a ele para tentar com outra forma de pagamento, e a solicitação não devera ser enviada ao prestador; [E2] - Caso seja negada ou reagendada a solicitação é enviado um e-mail e uma notificação(celular) ao Ator (Cliente ou Prestador) como forma de aviso; [E3] - Caso não seja confirmado o cancelamento, o Cliente permanece na tela de finalização da solicitação;	Não é necessário nenhum teste;

Tabela 4 *- Visualização dos orçamentos*

- Iniciar serviço:

Objetivo	Atores	Pré-condições	Evento incial	Fluxo Principal	Fluxo alternativo	Fluxo de Exceção	casos de testes
O objetivo é oferecer para aos Prestador uma forma de se iniciar um serviço.	• Prestador; • Cliente;	• Estar logado na aplicação; • O Prestador precisa estar com um serviço agendado;	O Prestador no menu principal acessa a aba de agenda.	1. É aberta a tela com a agenda do Prestador e acima destacado deve estar o serviço agendado para aquele momento; 2. O Prestador clica no serviço em destaque; 3. É exibida uma tela com as informações do serviço e do Cliente; 4. O Prestador clica em "iniciar atendimento"[TT]; 5. É exibida uma tela de confirmação; 6. O Prestador clica em "confirmar"[A1]; 7. É solicitado para o Prestador ler o código QRcode que o Cliente deve gerar[A2]; 8. O Prestador faz a leitura do código; 9. O serviço é iniciado; 10. Fim do caso de uso;	[A1] - Cancelar; 1. O Prestador clica em "cancelar"; 2. retorna ao passo 3 do fluxo principal; [A2] - Gerar código; 1. O Cliente gera um código normal; 2. O Cliente passa esse código para o Prestador; 3. O Prestador seleciona a opção de inserir o código manualmente; 4. O Prestador insere o código passado pelo Cliente; 5. retorna ao passo 9 do fluxo principal;	[E1] - Agendamentos com mais de 5 horas de atraso, não podem ser iniciados, e são cancelados automaticamente (não haverá cobrança),	Não há testes necessarios;

Tabela 5 *- Iniciar serviço*

- Encerrar serviço:

Objetivo	Atores	Pré-condições	Evento incial	Fluxo Principal	Fluxo alternativo	Fluxo de Exceção	casos de testes
O objetivo é oferecer para aos Prestador uma forma de se encerrar um serviço.	• Prestador; • Cliente; • Sistema de Pagamento	• Estar logado na aplicação; • O Prestador precisa estar com um serviço agendado;	O Prestador no menu principal acessa a aba de agenda."	1. É aberta a tela com a agenda do Prestador e acima destacado deve estar o serviço agendado para aquele momento; 2. O Prestador clica no serviço em destaque; 3. É exibida uma tela com as informações do serviço e do Cliente; 4. O Prestador clica em "encerrar atendimento"; 5. É exibida uma tela de confirmação; 6. O Prestador clica em "confirmar"[A1]; 7. É solicitado para o Prestador ler o código QRcode que o Cliente deve gerar[A2]; 8. O Prestador faz a leitura do código; 9. O serviço é encerrado; 10. É enviado uma solicitação ao Sistema de pagamento fazer a liberação do dinheiro ao Prestador; 11. O Prestador é notificado sobre a efetuação do pagamento; 12. Fim do caso de uso;	[A1] - Cancelar; 1. O Prestador clica em "cancelar" [E1]; 2. retorna ao passo 3 do fluxo principal; [A2] - Gerar código; 1. O Cliente gera um código normal; 2. O Cliente passa esse código para o Prestador; 3. O Prestador seleciona a opção de inserir o código manualmente; 4. O Prestador insere o código passado pelo Cliente; 5. retorna ao passo 9 do fluxo principal;	[E1] - Se o serviço não for encerrado nem cancelado, o pagamento é cancelado após 24 hrs do termino do agendamento;	Não há testes necessarios;

Tabela 6 *- Encerrar serviço*

4.4 DIAGRAMAS DE CLASSES

Como a plataforma é desenvolvida com base em uma arquitetura orientada a microsserviços, são criados dois diagramas de classes, um com uma representação do desenho de forma monolítica para fins de exemplificação, e outro em sua versão já com os serviços divididos.

4.4.1 Diagrama de classes da arquitetura monolítica

Ao analisarmos a figura 4, veremos a definição das classes de forma monolítica, ou seja, as classes estão desenhadas dentro de um único serviço, em que os códigos utilizados em sua programação ficam contidos em um só lugar.

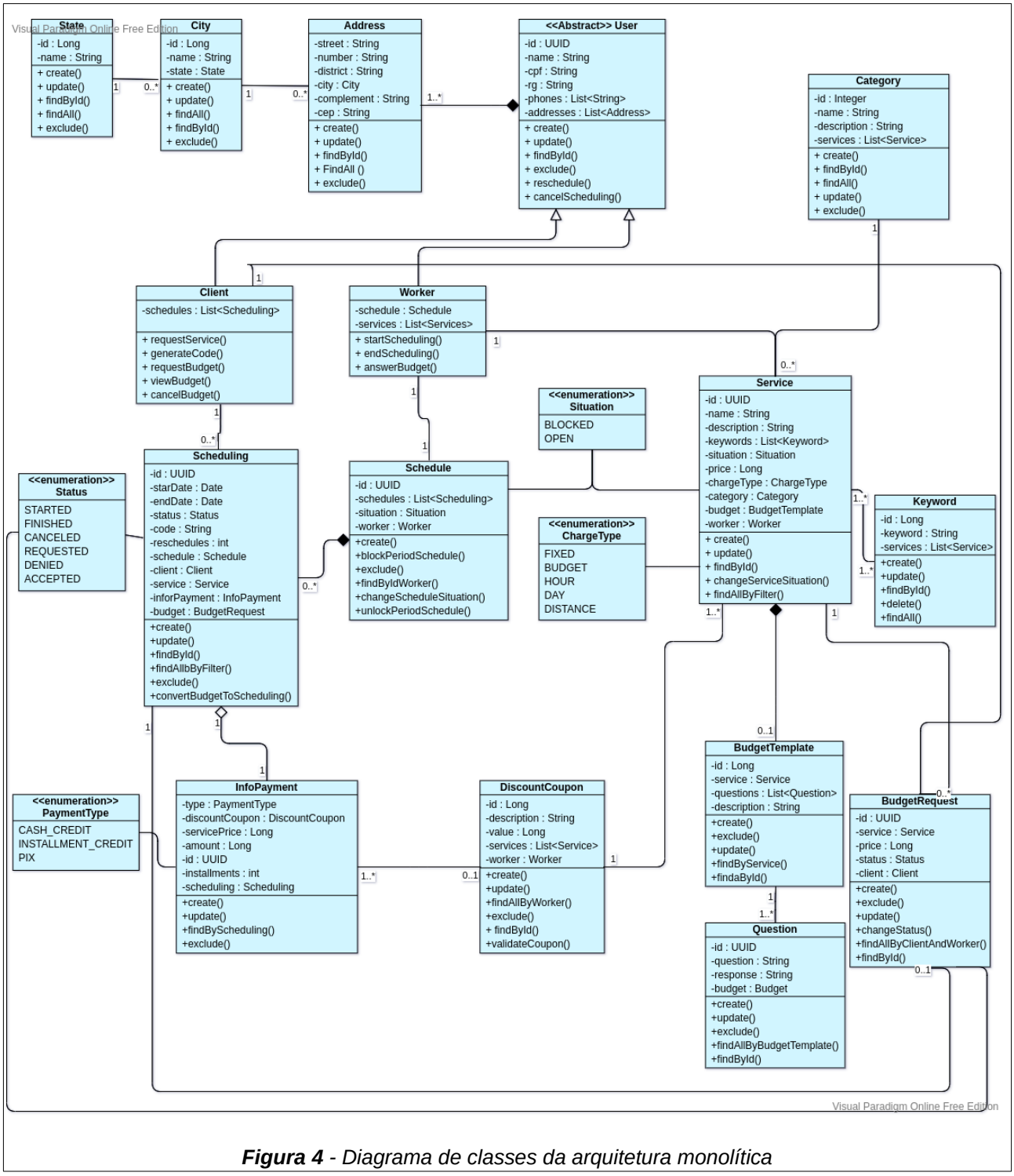

Figura 4 *- Diagrama de classes da arquitetura monolítica*

4.4.2 Diagrama de classes da arquitetura de microsserviços

Na figura 5 é possível ver o diagrama de classes completo, que esta dividido por contextos, onde cada um deles representa o que depois se tornou um microsserviço fracamente acoplado.

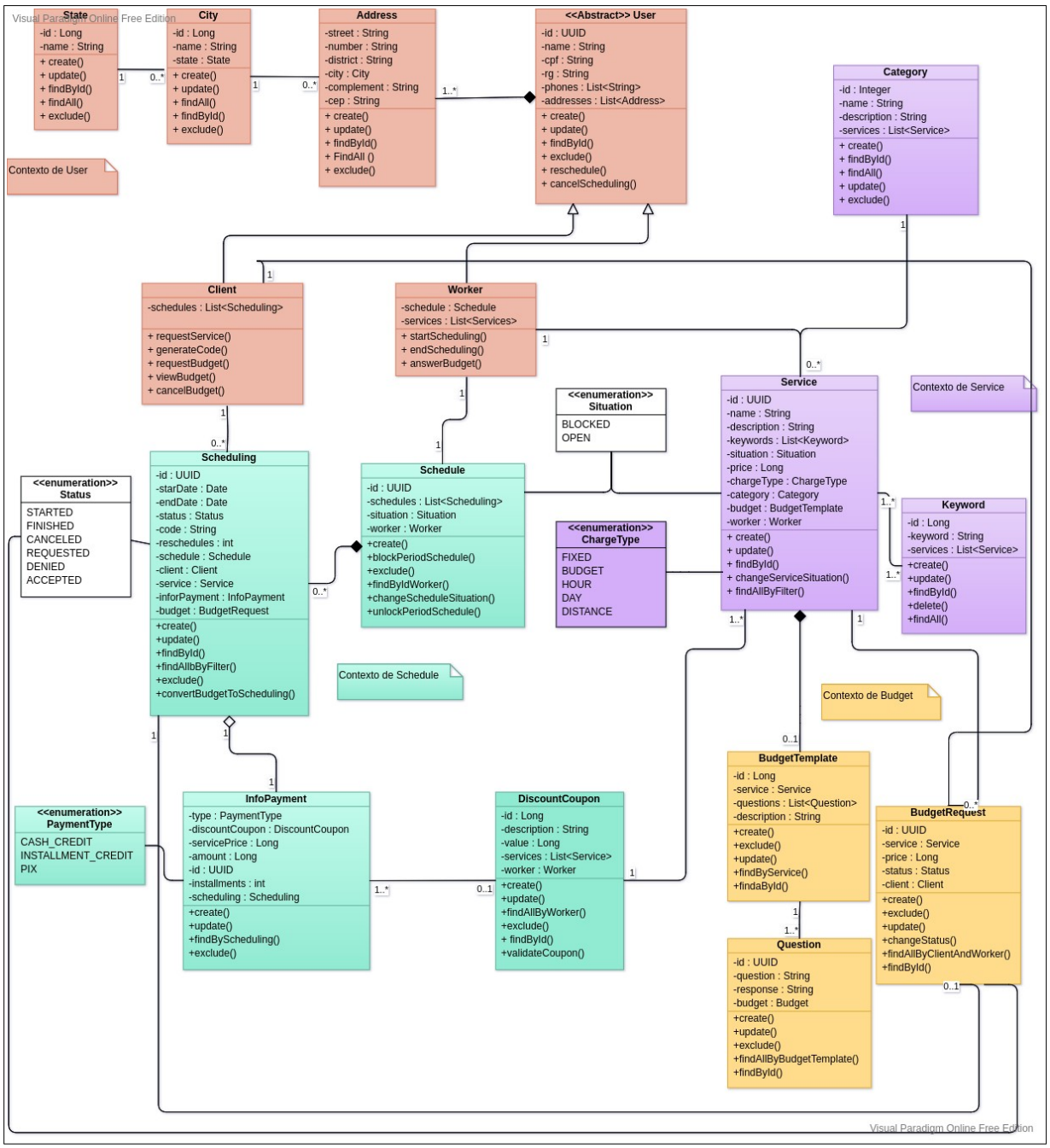

Figura 5 *- Diagrama de classes da arquitetura de microsserviços*

4.4.3 Diagrama de classes do microsserviço de *User*

O diagrama de classes exemplificado na figura 6 compõe às classes do microsserviço de *User* e seus respectivos atributos, métodos e relacionamentos.

Esse microsserviço é responsável pelas operações de cadastro, consulta, edição e deleção dos usuários, além de provisionas informações para os outros serviços que compõe a aplicação.

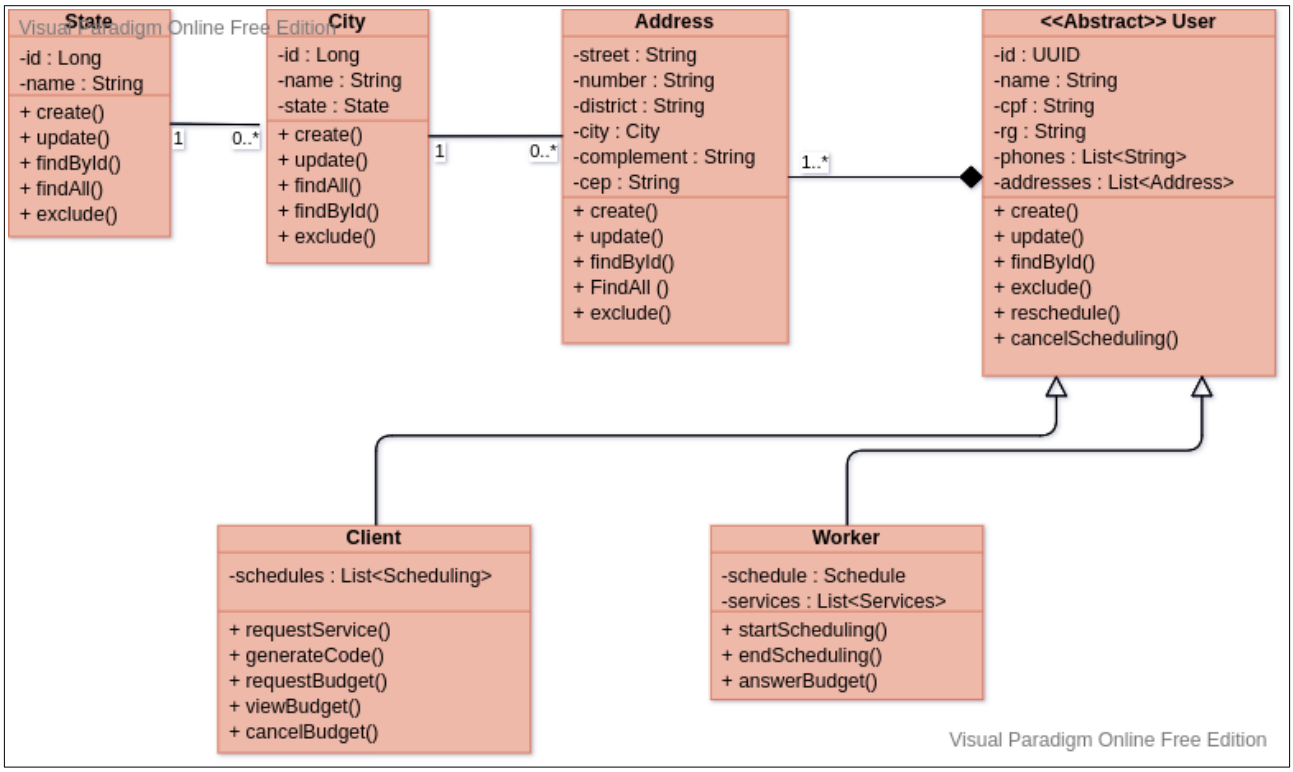

Figura 6 *- Diagrama de classes microsserviço de User*

4.4.4 Diagrama de classes do microsserviço de Service

O diagrama de classes exemplificado na figura 7 compõe às classes do microsserviço de *Service* e seus respectivos atributos, métodos e relacionamentos.

Esse microsserviço é responsável pelas operações de cadastro, consulta, edição e deleção dos serviços oferecidos pelos prestadores, além de provisionas informações para os outros serviços que compõe a aplicação.

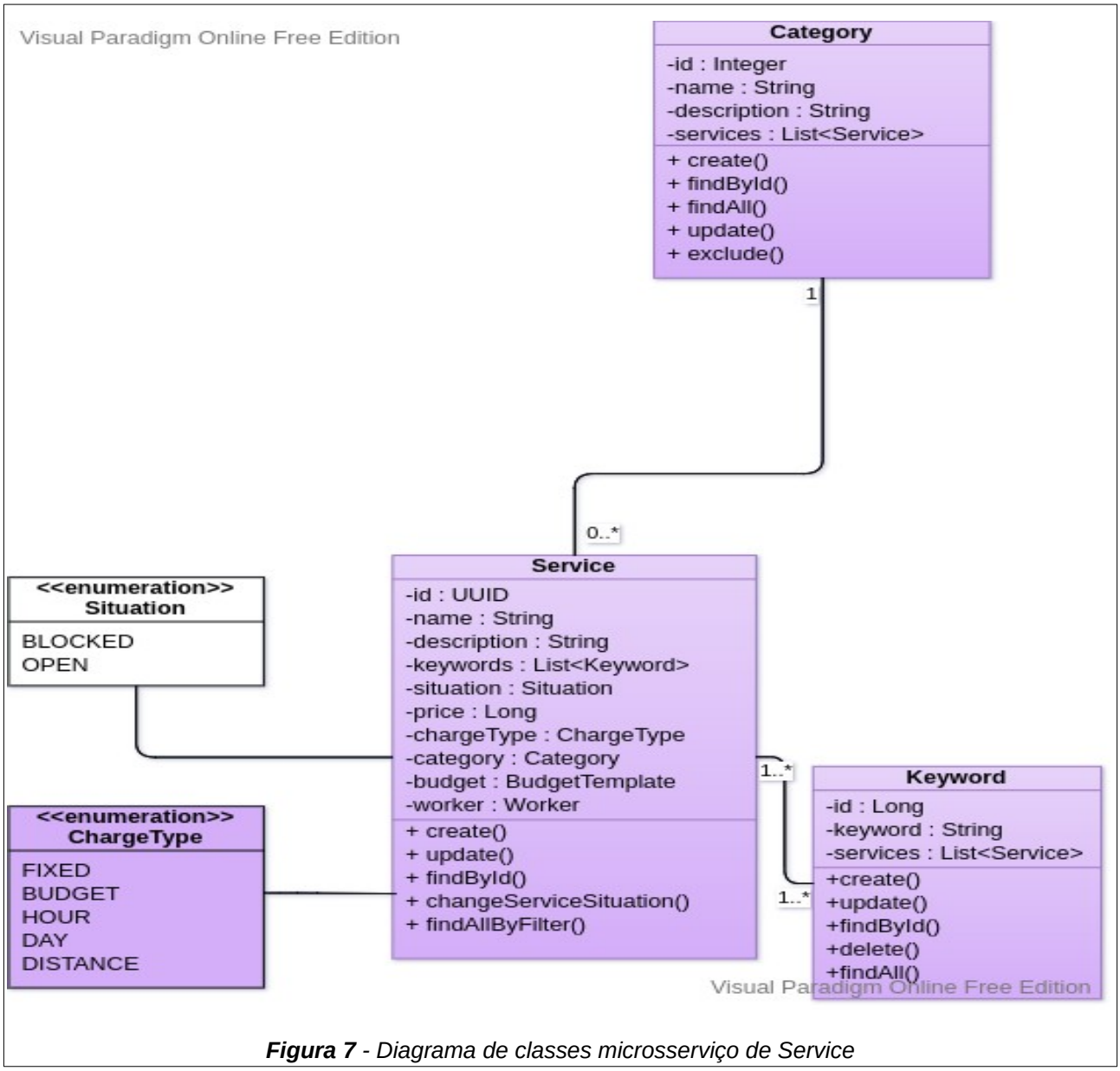

Figura 7 *- Diagrama de classes microsserviço de Service*

4.4.5 Diagrama de classes do microsserviço de Schedule

O diagrama de classes exemplificado na figura 8 compõe às classes do microsserviço de *Schedule* e seus respectivos atributos, métodos e relacionamentos.

Esse microsserviço é responsável pelas operações de solicitação de agendamentos, efetivação de orçamentos e gerenciamentos das informações gerais de uma contratação.

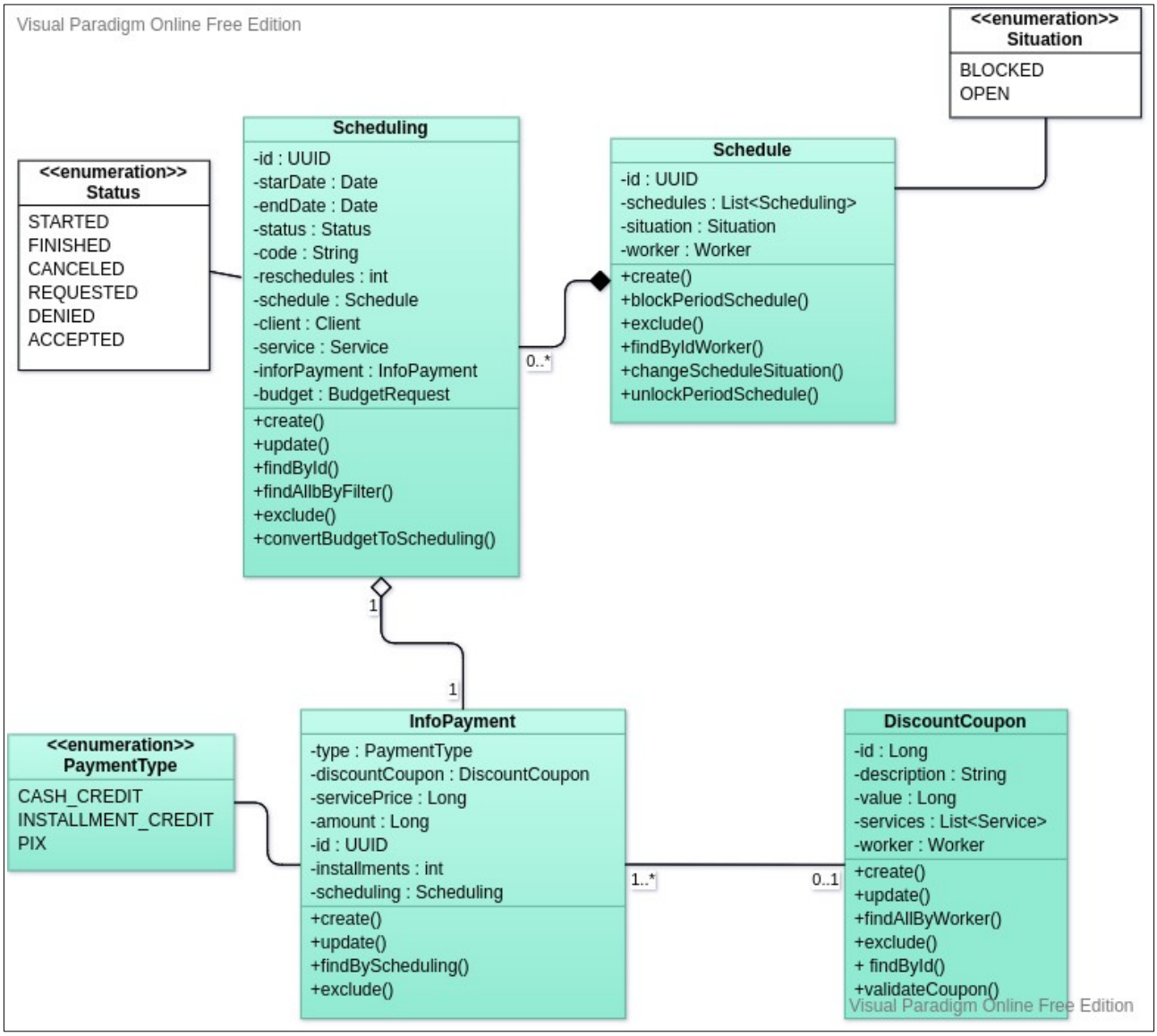

Figura 8 *- Diagrama de classes microsserviço de Schedule*

4.4.6 Diagrama de classes do microsserviço de Budget

O diagrama de classes exemplificado na figura 9 compõe às classes do microsserviço de *Budget* e seus respectivos atributos, métodos e relacionamentos.

Esse microsserviço é responsável pelas operações de criação, solicitação e aprovação de orçamentos, solicitação de orçamentos, além de provisionas informações para os outros serviços que compõe a aplicação.

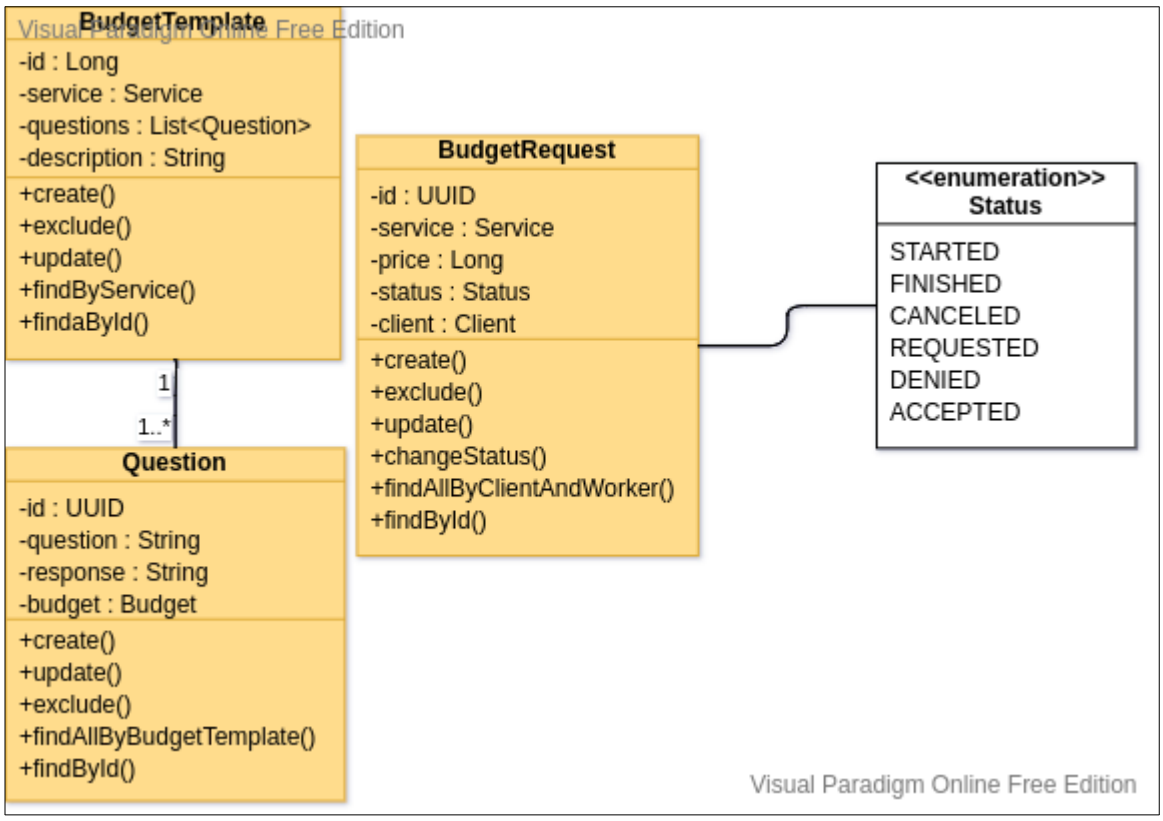

Figura 9 *- Diagrama de classes microsserviço de Budget*

4.5 ARQUITETURA

A arquitetura *back-end* do projeto é baseada em microsserviços, um estilo arquitetural que se baseia em em desmembrar os serviços de uma aplicação em contextos limitados, os tornando assim independentes e fracamente acoplados.

A figura 10 retrata o resultado desse desmembramento dos serviços da aplicação por contextos, além de outros componentes que compõe a arquitetura.

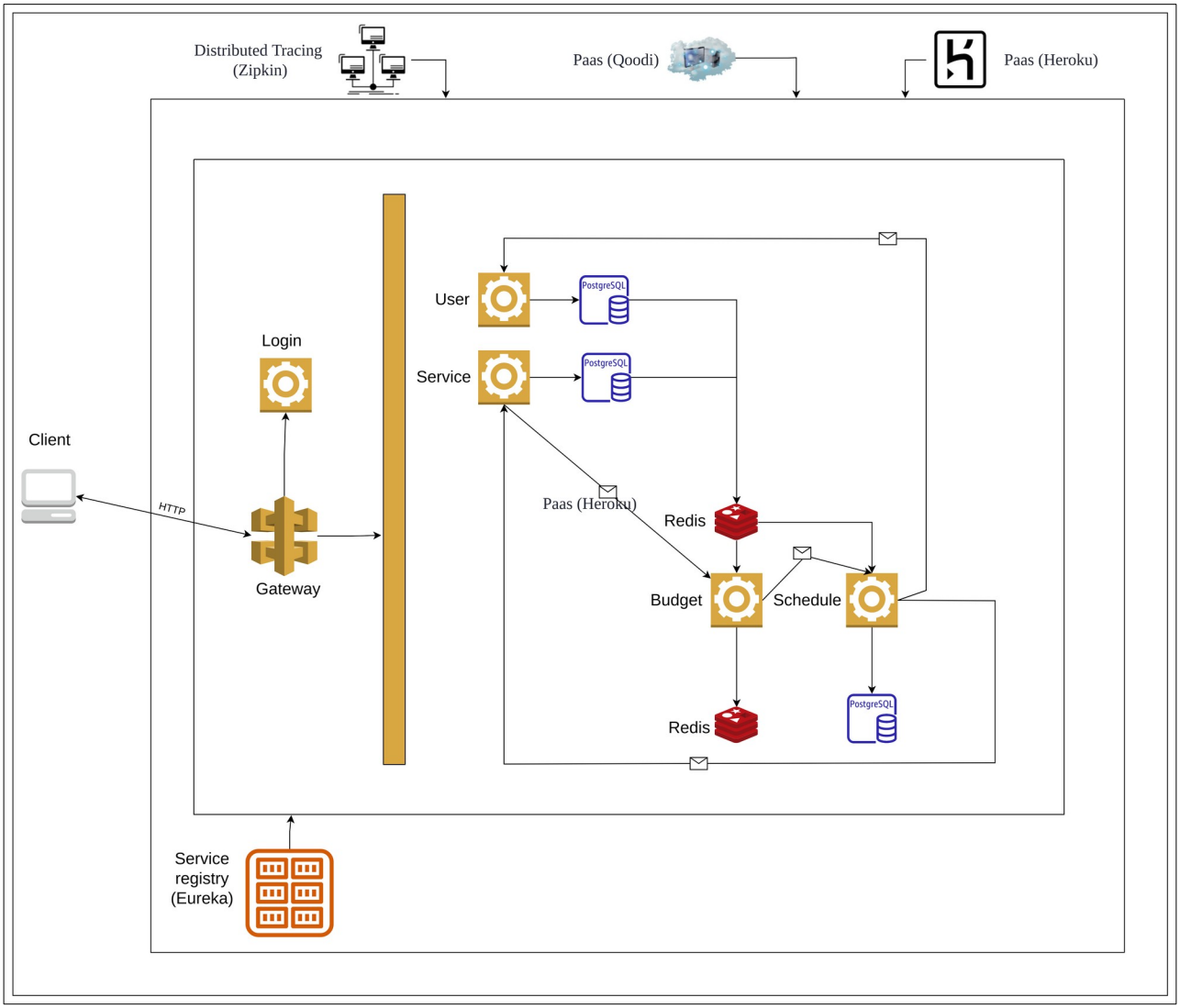

Figura 10 *- Desenho arquitetural*

Além dos serviços já demonstrados anteriormente, a arquitetura ainda apresenta outros componentes, esses elementos servem para melhorar a performance, escalabilidade, observabilidade e segurança da aplicação.

4.5.1 Gateway

O API Gateway é um gerenciador de tráfego que faz interface com os serviços de *back-end* real ou de dados e aplica políticas, autenticação e controle de acesso geral para chamadas de APIs de forma a proteger e simplificar a utilização dos serviços.

Fazer seu uso é a maneira de controlar o acesso aos seus sistemas e serviços e foi projetado para otimizar a comunicação entre clientes externos e seus serviços de *back-end*, oferecendo aos seus clientes uma experiência mais simples já que não necessita ficar se preocupando com chamadas em diferentes *hosts* e métodos de autenticação.

4.5.2 Service registry

O registro de serviços é um componente essencial quando falamos de se obter uma aplicação auto escalável, é nele que os serviços se auto registram ao serem instanciados e assim podem ser acionado dinamicamente por um *Load Balancer* que fica junto com esse componente e é responsável por balancear as chamadas ao serviços que estão registrados nesse componente.

4.5.3 Distributed tracing

O rastreamento distribuído é crucial para solucionar problemas e entender o fluxo de chamadas dos microsserviços, ou seja, é muito útil quando precisamos rastrear a solicitação que passa por vários microsserviços. O rastreamento distribuído também pode ser usado para medir o desempenho dos microsserviços.

Com o uso de ferramentas que façam esse serviço, se torna mais fácil identificar qual microsserviço está com falha ou com problemas de desempenho sempre que houver várias chamadas de serviço em uma única solicitação.

4.6 ENTIDADE RELACIONAMENTO

Foi criado uma modelagem de entidade relacionamento para cada microsserviço, afim de seguir as normas e manter um bom padrão de projeto para os mesmos, fazendo com que cada serviço tivesse seu próprio banco de dados independente e de acesso restrito.

Para um microsserviço fazer acesso a algum dado que esteja fora de seu banco de dados, ele deve de alguma maneira requisitar para o serviço responsável pelo acesso do banco desejado.

- ER do microsserviço de User:

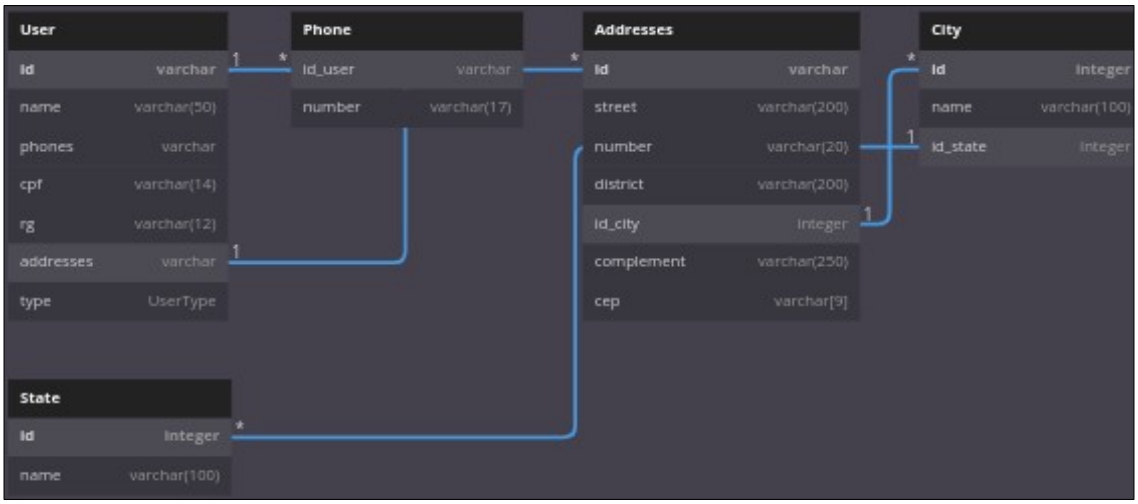

Figura 11 *- ER do microsserviço de User*

- ER do microsserviço de Service:

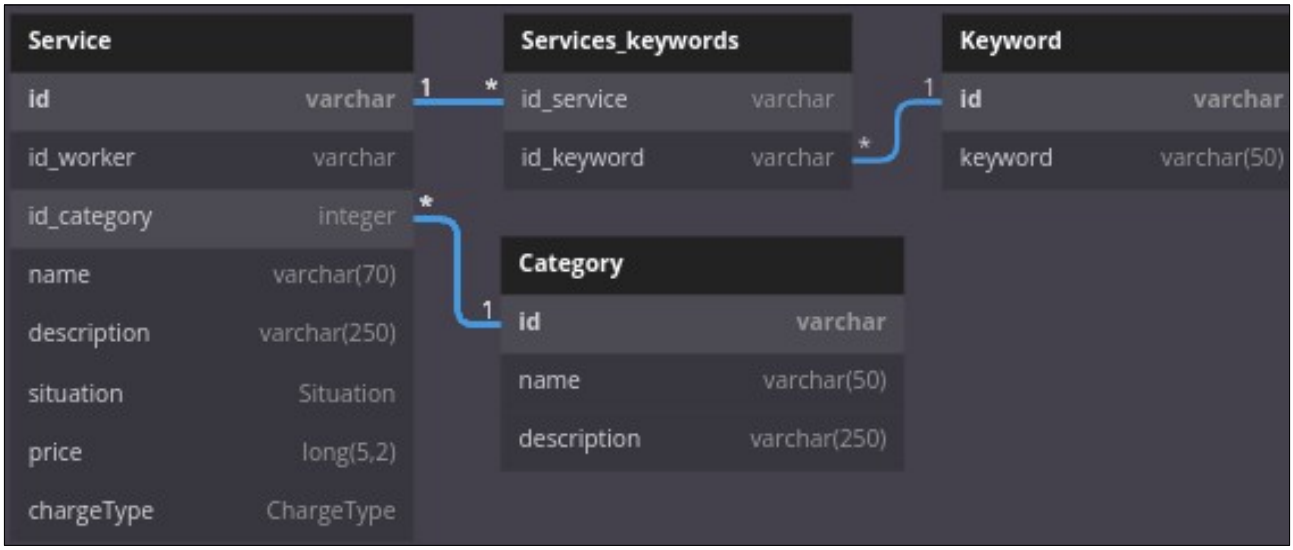

Figura 12 *- ER do microsserviço de Service*

- ER do microsserviço de Budget:

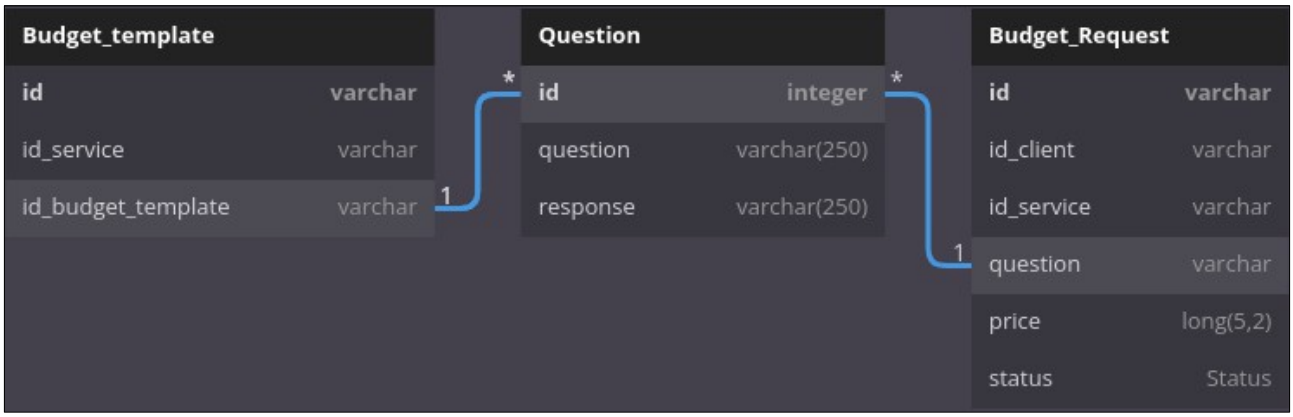

Figura 13 *- ER do microsserviço de Budget*

- ER do microsserviço de Schedule:

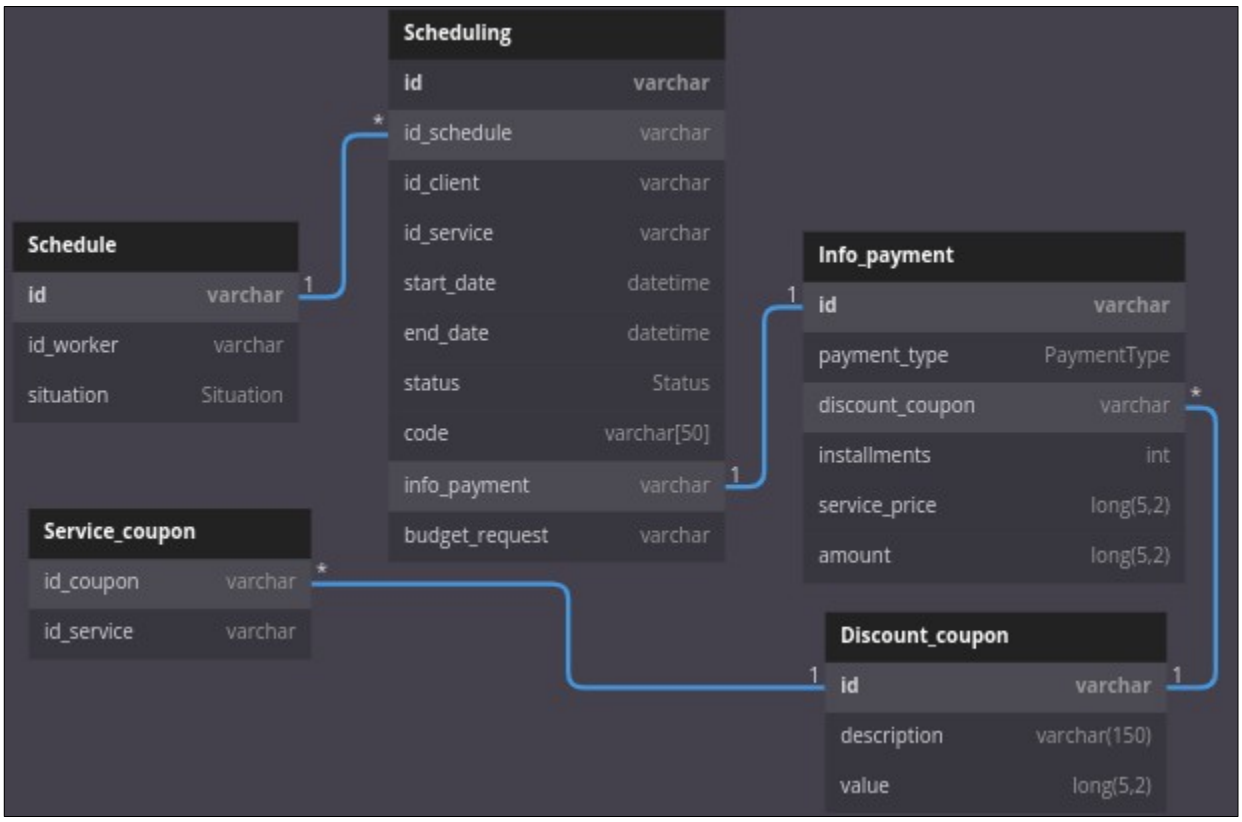

Figura 14 *- ER do microsserviço de Schedule*

4.7 DIAGRAMA DE SEQUÊNCIA

O diagrama de sequencia apresentado na figura 15 mostra o fluxo representativo de acesso a algum serviço presente na aplicação, ele mostra como o sistema irá funcionar ao ser acessado por algum usuário.

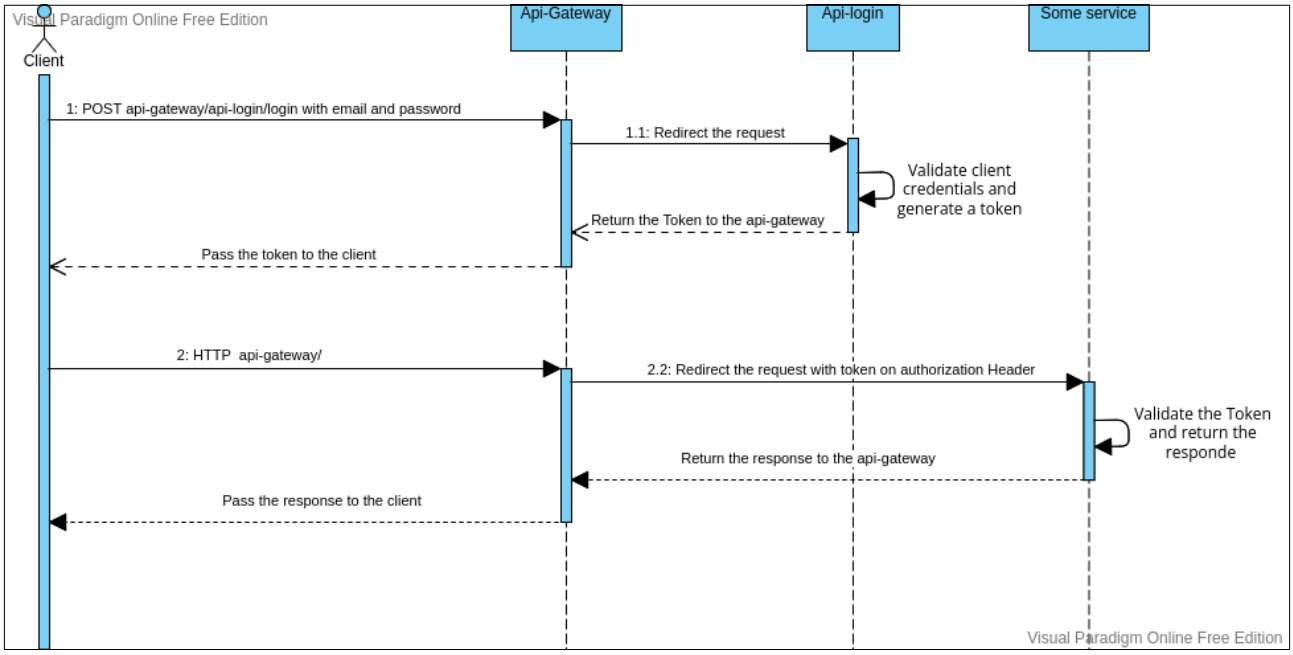

Figura 15 *- Diagrama de sequência de acesso a aplicação*

4.8 ESTRUTURA ANALÍTICA DO PROJETO

Com as definições dos diagramas apresentados, foi elaborado uma EAP para fazer um mapeamento das atividades e fornecer uma melhor organização e orientação do projeto.

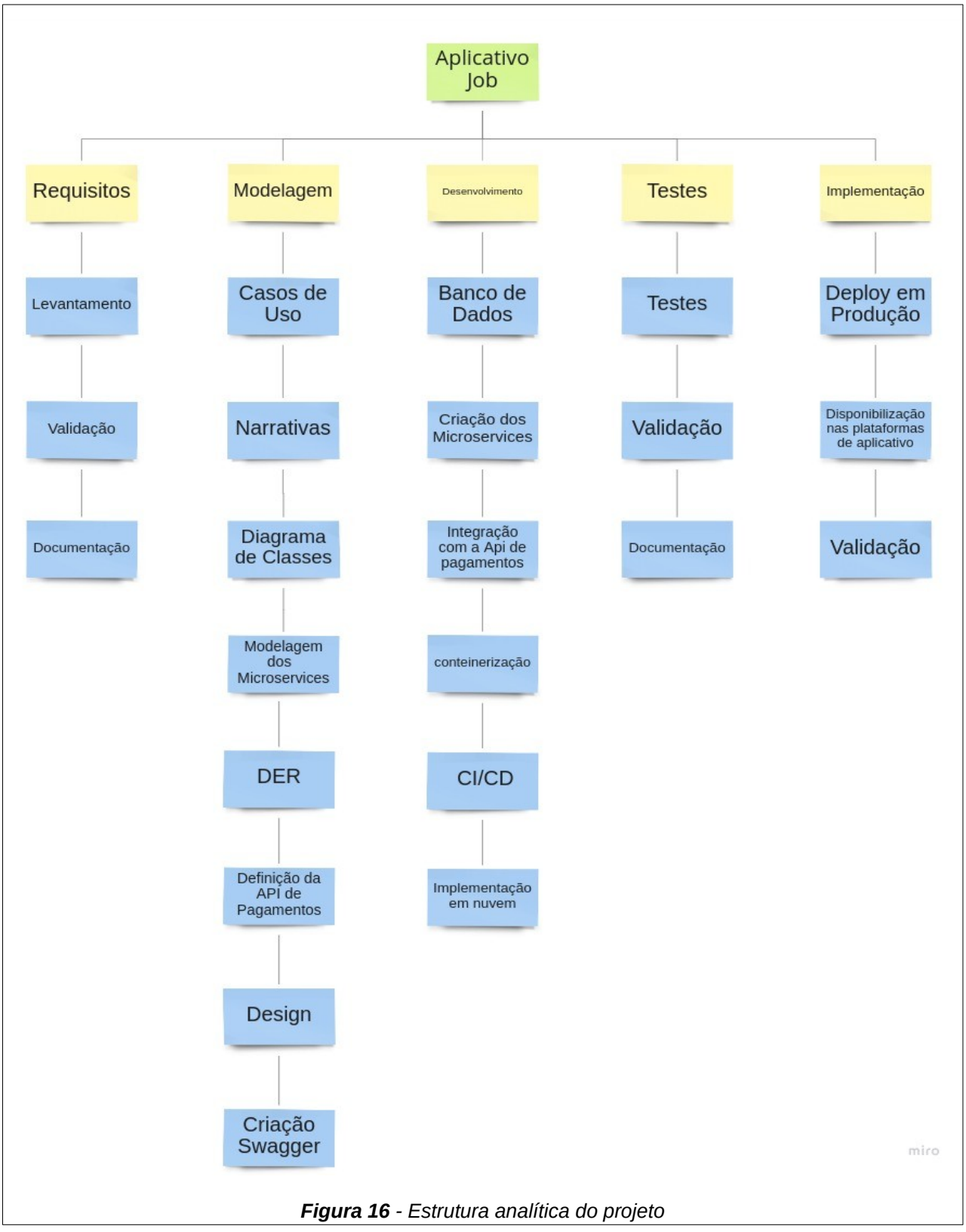

Figura 16 *- Estrutura analítica do projeto*

5. CONCEITOS E TECNOLOGIAS UTILIZADAS NO TRABALHO

Neste capitulo são decorridas sobre as etapas, conceitos e tecnologias utilizadas durante a construção da plataforma.

5.1 ANÁLISE

Na primeira etapa foi realizada uma análise em aplicativos semelhantes como os já referidos na introdução, e também nas avaliações feitas por seus usuários, a fim de se obterem regras de negocio consistentes e que atendam as suas necessidades. Nessa fase, além dos sites de pesquisa referentes as plataformas semelhantes, foram utilizadas também as seguintes ferramentas:

- Miro;
- Canvanizer

5.1.1 Miro

Miro é uma plataforma de lousa interativa digital que possibilita a escrita de forma livre, além de possibilitar a inserção de tabelas, diagramas, imagens, etc.

Sua utilização na etapa de análise foi para a junção de ideias referentes ao aplicativo Job, além da criação da EAP apresentada na figura 16.

5.1.2 Canvanizer

A plataforma do Canvanizer possibilitou a criação de um Canvas da aplicação, ou seja, foi uma ferramenta utilizada para fazer um gerenciamento estratégico do aplicativo e permi-

tiu esboçar o modelo de negocio em uma mapa visual, possibilitando assim obter uma melhor perspectiva das regras e dos requisitos do mesmo.

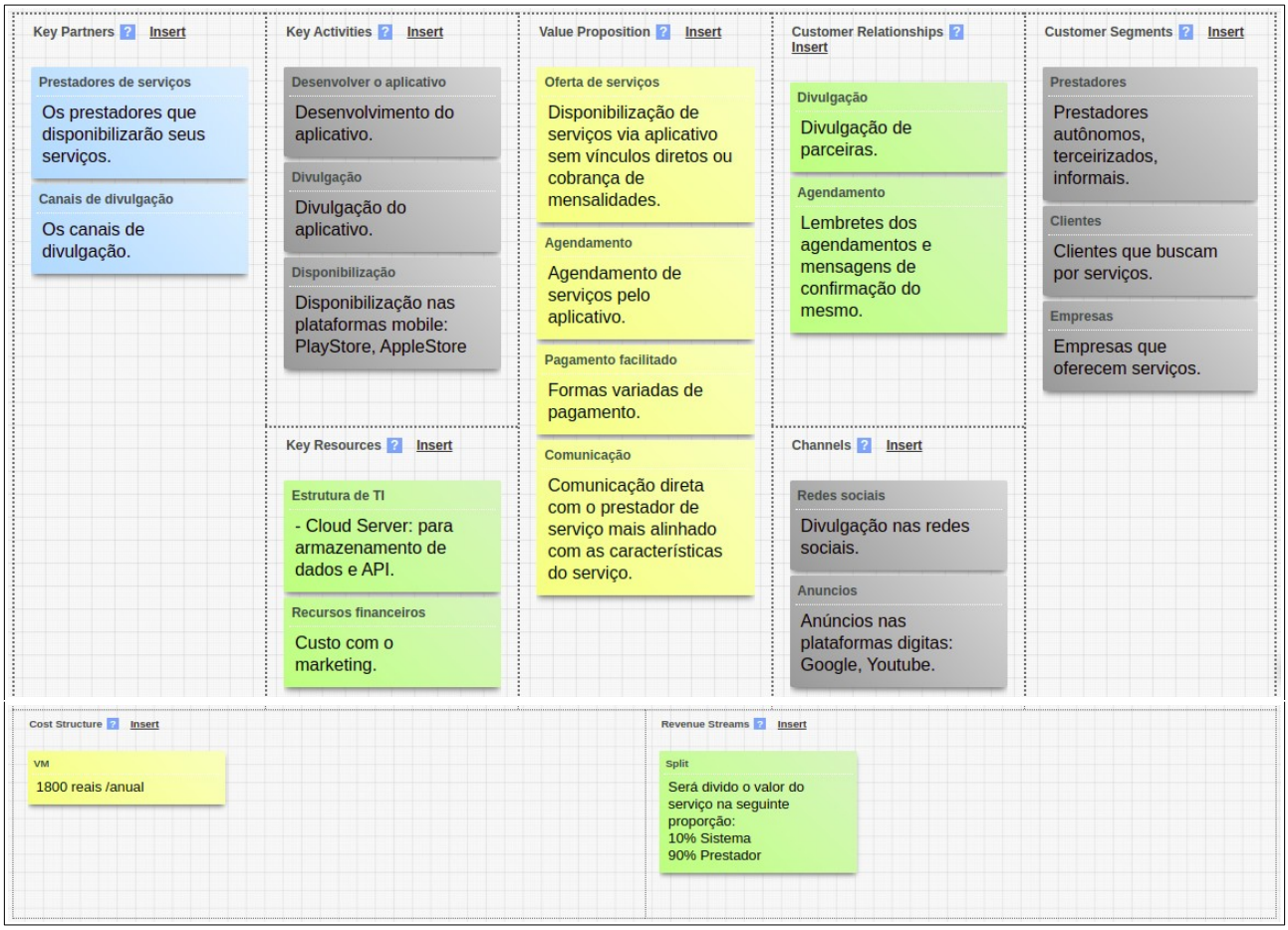

Figura 17 - *Canvas*

5.2 MODELAGEM

Na fase de modelagem foram feitas as modelagens de acordo com o modelo de negocio e requisitos levantados. Além disso, foram definidos os casos de uso e suas narrativas, além da criação do diagrama de classes e o diagrama de entidade relacionamento, e ainda, como a arquitetura da aplicação foi baseada em microsserviços, é nessa etapa que eles, de acordo com os princípios do DDD foram definidos, assim como o desenho da arquitetura *back-end* da plataforma. As ferramentas utilizadas nesse passo são:

- Visual Paradigm;

- dbdiagram.io
- diagrams.net
- Miro

5.2.1 DDD

O DDD é um conjunto de princípios para projeto de software, organizados e sistematizados por Eric Evans no livro Domain-Driven Design: Tackling Complexity in Software (2003), os princípios defendidos por DDD têm, no seu conjunto, um objetivo central que é permitir o desenvolvimento de sistemas cujo design é centrado em conceitos próximos e alinhados com um domínio de negócio.

Seguindo o conceito de DDD foi preciso obter os *Bounded Contexts* que são os delimitadores de contextos da aplicação.

Como cada contexto deve possuir suas responsabilidades claramente definidas, fui utilizada a estrategia de criar uma matriz onde foram inseridas as entidades da aplicação, e também as funções que a aplicação deveria executar.

Após inseridos esses dados, foi mapeado de acordo com como as entidades iriam se relacionar com as funções nos seguintes ações:

- C = create (criação);
- R = read (leitura);
- U = update (atualização);
- D = delete (deleção)

Na figura 18 podemos ver o resultado dessa matriz, e destacados de cores diferentes temos os agrupamentos das funções e quais ações seriam executadas pelas entidades definidas, gerando assim alguns dos contextos principais.

CRUD Matrix Capabilities & Entities Products Microservices Systems		Entities													
		Client	Andress	Worker	Schedule	scheduling	BudgetRequest	InforPayment	DiscountCoupon	Service	KeyWord	BudgetTemplate	Question	Category	payment system
Capabilities	Keep Client	CRUD	CRUD												
	Keep Worker		CRUD	CRUD	CD										
	keep schedule (Worker)	R		R	RU	RU				R				R	
	Request a scheduling (Client)	R	R	R	R	CRUD		CRUD		R	R			R	R
	Keep scheduling	R	R	R	R	RU	R	CRUD	CRUD	R	R		R	R	R
	Request a budget (Client)	R		R	R	CRUD	CRUD			R	R	R	RU	R	
	answer scheduling request (Worker)	R		R		RU		R		R				R	
	answer budget request (Worker)	R				RU	RU			R			R	R	
	keep Service			R						CRUD	CRUD	CRUD	CRUD	R	
	Keep Cayegory													CRUD	
	Sending e-mails	R		R		R	R	R		R			R		
	Chats for communication	R		R		R									

Figura 18 *- Matriz de contextos da aplicação*

Os contextos resultantes da matriz, e também adicionando um contexto Budget que é responsável pela parte de orçamentos, um contexto de Payment que futuramente, quando implementado será responsável pelos pagamentos realizados dentro da plataforma, e também um contexto de autenticação que ficará responsável por autenticar os usuários da aplicação estão representados na figura 19 além disso eles estão coloridos de acordo com o tipo de domínio. Os tipos possíveis presentes são:

- Main domain (domínio principal) = significa que esse domínio representa alguma função muito importante para aplicação, sem ele o sistema não estaria completo;
- Generic domain (domínio genérico) = São domínios que ajudam no funcionamento do domínio principal, porem de forma independentes e podendo não ser relacionados com somente um único domínio principal;
- Auxiliary domain (domínio auxiliar) = São domínios que auxiliam toda a aplicação em seu funcionamento, tanto domínios genéricos quanto os domínios principais;

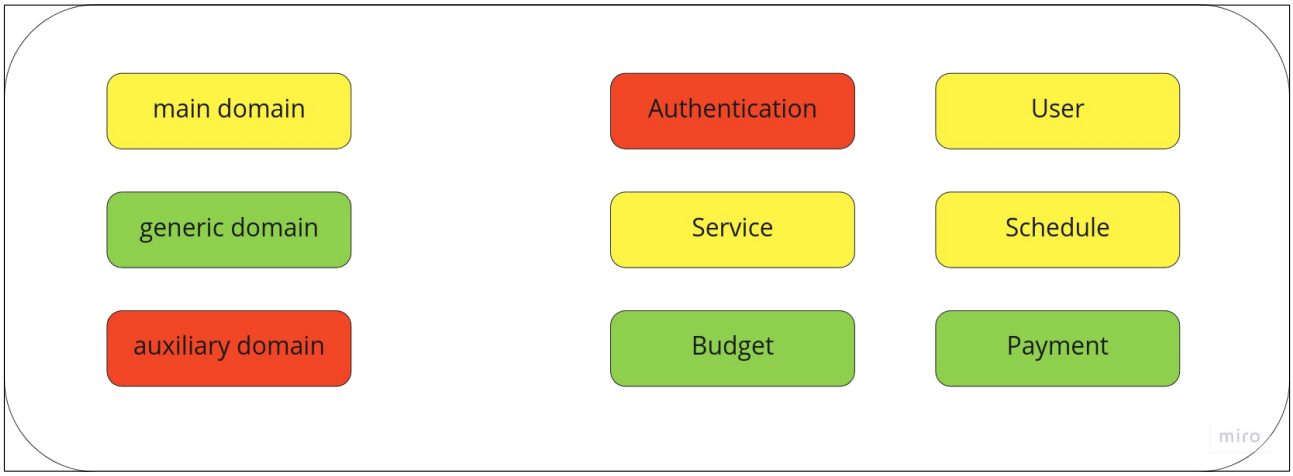

Figura 19 *- Contextos da aplicação*

Depois de já obtidos os contextos da aplicação e seus tipos de domínios, criou-se o mapeamento dos contextos, que é a relação entre os tipos domínios.

Apresentado na figura 20 está a relação entre os domínios principais e genéricos que são *upstream e downstream*, respectivamente. Isso porque o domínio principal é prioridade em relação ao genérico. Se alguma coisa mudar, terá que ser do lado do domínio genérico. É uma relação de cliente fornecedor

No caso da relação entre domínios genéricos, prevalece o que não temos autonomia para alterar. No caso de Schedule e Payment, não temos controle sobre a API de Paymente, pois será integrado a alguma plataforma já desenvolvida, logo a relação é Paymante e Schedule, respectivamente *upstream* e *downstream*. Neste caso, chamamos de relação conformista. O que podemos fazer é criar uma camada anticorrupção para não termos que refazer todo o processo de pagamento, caso haja alguma alteração na API que sera integrada.

Em relação a autenticação temos um núcleo compartilhado onde todos poderão acessar esse processo. Na relação do processo de autenticação também é possível criar uma camada anticorrupção. Com isso, caso haja alteração na forma de autenticação, não haverá problemas com o resto do sistema, pois estaremos acessando a camada anticorrupção.

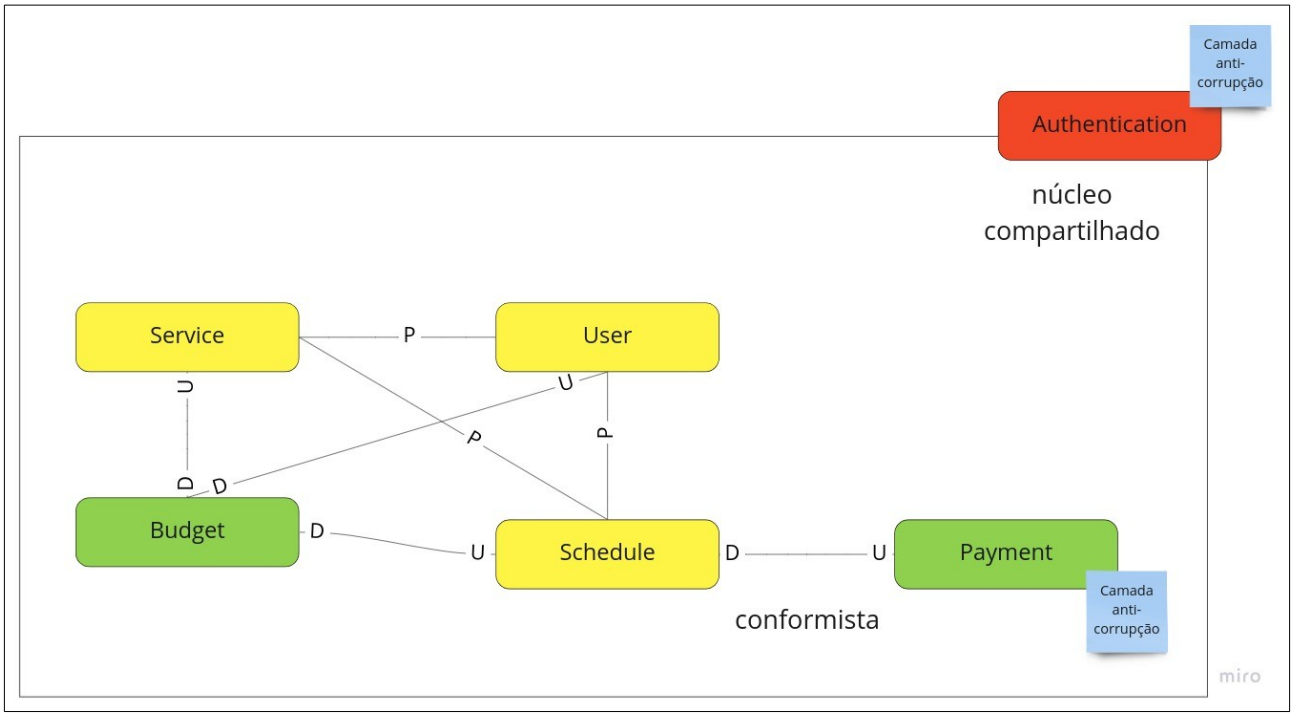

Figura 20 *- Mapeamento dos contextos*

Com a definição dos domínios e de seus relacionamentos, foi então definido quais seriam os serviços do aplicativo Job.

5.2.2 Visual Paradigm

O Visual Paradigm é um site que oferece diversas ferramentas pagas e gratuitas em diversas áreas, sendo que algumas delas são oferecidas para a construção de diagramas, que foi o intuito ao qual ela foi utilizada, para a definição dos diagramas de classes apresentados da figura 4 à figura 9, e do diagrama de sequencia mostrado na figura 15

5.2.3 DBDiagram.io

É uma ferramenta online e gratuita projetada para desenha ER onde o do desenho é todo feito através de linhas de códigos próprio, ou seja, é uma forma dos desenvolvedores criarem um ER de forma amigável devido ao modo como é feito.

O código utilizado para a construção dos ERs referentes ao projeto esta demonstrado nos figuras seguintes.

- Código usado no ER da figura 11:

```
Table User {
  id varchar [pk]
  name varchar(50) [not null]
  phones varchar
  cpf varchar(14) [not null]
  rg varchar(12) [not null]
  addresses varchar [ref: < Addresses.id, not null]
  type UserType [not null]
}
Table Phone {
  id_user varchar [ref: > User.id, not null]
  number varchar(17) [not null, unique]
}
Table Addresses {
  id varchar [pk]
  street varchar(200) [not null]
  number varchar(20) [not null]
  district varchar(200) [not null]
  id_city integer [ref: < City.id, not null]
  complement varchar(250)
  cep varchar[9] [not null]
}
Table City {
  id integer [pk]
  name varchar(100) [not null, unique]
  id_state Integer [ref: < State.id, not null]
}
Table State {
  id integer [pk]
  name varchar(100) [not null, unique]
}
```

Figura 21 *- Tabelas do microsserviço de User*

- Código usado no ER da figura 12:

```
Table Category{
  id varchar [pk]
  name varchar(50) [not null, unique]
  description varchar(250) [not null]
}

Table Service {
  id varchar [pk]
  id_worker varchar [ref: > User.id, not null]
  id_category integer [ref: > Category.id, not null]
  name varchar(70) [not null]
  description varchar(250)
  situation Situation
  price long(5,2) [not null, note: "if the charge type is BUDGET,
                                    the price inital must be 0"]
  chargeType ChargeType [not null]

}

Table Keyword {
  id varchar [pk]
  keyword varchar(50) [not null, unique]
}

Table Services_keywords {
  id_service varchar [ref: > Service.id]
  id_keyword varchar [ref: > Keyword.id]
}
```

Figura 22 *- Tabelas do microsserviço de Service*

- Código usado no ER da figura 15:

```
Table Budget_template {
  id varchar [pk]
  id_service varchar [ref: - Service.id, not null]
  id_budget_template varchar [ref: < Question.id, not null]
}

Table Question {
  id integer [pk]
  question varchar(250) [not null]
  response varchar(250)
}

Table Budget_Request {
  id varchar [pk]
  id_client varchar [ref: > User.id, not null]
  id_service varchar [ref: > Service.id, not null]
  question varchar [ref: < Question.id]
  price long(5,2)
  status Status
}
```

Figura 23 *- Tabelas do microsserviço de Budget*

- Código usado no ER da figura 13:

```
Table Schedule {
  id varchar [pk]
  id_worker varchar [ref: - User.id, not null]
  situation Situation
}

Table Scheduling {
  id varchar [pk]
  id_schedule varchar [ref: > Schedule.id, not null]
  id_client varchar [ref: > User.id, not null]
  id_service varchar [ref: > Service.id, not null]
  start_date datetime [not null]
  end_date datetime [not null]
  status Status
  code varchar[50] [note: "generated code for authentication and
                          star/end of scheduling"]
  info_payment varchar [ref: - Info_payment.id, not null]
  budget_request varchar  [ref: - Budget_Request.id, note: "Used for when
                                  the service is BUDGET service type"]
}
```

Figura 24 *- Tabelas de schedule e scheduling do microsserviço de Schedule*

```
Table Info_payment {
  id varchar [pk]
  payment_type PaymentType [not null]
  discount_coupon varchar [ref: > Discount_coupon.id]
  installments int [default: "0"]
  service_price long(5,2)
  amount long(5,2)
}

Table Discount_coupon {
  id varchar [pk]
  description varchar(150) [not null]
  value long(5,2)
}

Table Service_coupon {
  id_coupon varchar [ref: > Discount_coupon.id, not null]
  id_service varchar [ref: > Service.id, not null]
}
```

Figura 25 *- Tabelas de relacionadas as informações de pagamento*

5.2.4 Diagrams.net

O *software* Diagrams.net é usado para criação de desenhos gráficos como fluxogramas, *wireframes* e também pode ser utilizado para criação do desenhos arquiteturais, como o mostrado na figura 9, que é o desenho representativo da arquitetura utilizada na construção dessa aplicação.

5.2.5 Miro

Na etapa de modelagem, o miro foi utilizado para a criação do mapa mental mostrado nas figuras 1 e 2, dos casos de uso representado pela figura 3, além da suas narrativas que estão presentes da tabela 1 até a tabela 6.

5.3 DESENVOLVIMENTO

Na etapa de desenvolvimento, foi implementado todo o *back-end* contente das regras de negocio da aplicação, além da criação e configuração dos demais componentes que compõe a arquitetura presente na figura 10.

5.3.1 IntelliJ IDEA

Segundo JetBrains (2020), Intellij IDEA20 é uma plataforma de código aberto com objetivo de oferecer ferramentas para desenvolvimento. Essa IDE foi desenvolvida na linguagem de programação Java e oferece um comportamento de plataforma cruzada com o intuito de desenvolver ferramentas para quaisquer tipos de linguagens. Todo seu código encontra-se no GitHub, sendo assim, possibilita a criação de *plug-ins* para auxiliar o desenvolvimento da comunidade interessada.

Seu uso se deu pelos motivos dessa IDE possuir uma interface amigável ao usuário, além oferecer ferramentas integradas como o sistema de controle de versões e um *debug* muito eficiente e pratico.

5.3.2 JAVA

Java é uma linguagem de programação que foi criada pela empresa Microsystems no ano de 1995, conforme JAVA (2020). Esta linguagem é orientada a objetos e seu maior objetivo é de construir apenas um código que seja utilizado por diversos tipos de dispositivos, pois possui uma Java Virtual Machine (JVM), que traduz o código para todos os dispositivos, porém para que isso seja realmente possível, o requisito mínimo é a instalação do próprio Java. Em algumas aplicações é necessária a utilização do Java em seu computador ou até mesmo em seu dispositivo móvel.

Essa linguagem foi escolhida por ser uma linguagem orientada a objetos, além de possuir uma coleção muito grande de bibliotecas de código aberto, ser gratuita e possuir *frameworks* muito bons de desenvolvimento voltados para sua utilização como o Quarkus e o E*cossistema* Spring.

5.3.3 Ecossistema Spring

o Spring é um ecossistema de desenvolvimento que facilita a criação de aplicações Java utilizando diversos módulos independentes.

Os módulos utilizados desse ecossistema para a criação dos serviços foram os seguintes:

- Spring Boot: facilita a criação de aplicativos independentes baseados em Spring. Ele traz toda uma configuração automática de dependências, além de incorporar servidores como o Tomcat.
- O Spring Data JPA: facilita a implementação de repositórios baseados em JPA. Este módulo lida com suporte aprimorado para camadas de acesso a dados basea-

das em JPA. Facilita a criação de aplicativos baseados em Spring que usam tecnologias de acesso a dados.

- Spring Data Redis: oferece fácil configuração e acesso ao Redis a partir de aplicativos Spring.
- Spring Cloud Netflix: fornece integrações Netflix OSS para aplicativos Spring Boot por meio de configuração automática, Ele ajuda na criação de servidores e consumidores de registradores de serviços.
- Spring Cloud Gateway: Este projeto fornece uma biblioteca para construção de um API Gateway.
- Spring Cloud Security: Oferece recursos para configuração de serviços em nuvem como um interceptador de chamadas que ajuda nas configurações de autenticação e autorização.
- Spring AMQP: auxilia o desenvolvimento de soluções de mensagens baseadas em AMQP. Ele fornece um modelo como uma abstração de alto nível para enviar e receber mensagens.

5.3.4 Modelmapper

O ModelMapper é um mapeador de objetos, ele determinando automaticamente como um modelo de objeto é mapeado para outro, isso significa que essa ferramenta consegue transformar um tipo de objeto em outro da mesma forma como um humano faria.

Essa ferramenta foi utilizada por facilitar e acelerar o desenvolvimento, poupando tempo quando se trata da criação de objetos de transferência dentro dos serviços, além de eliminar muito código repetitivo que é necessário na criação desses objetos.

Um exemplo de utilização está presente na figura 26, onde mapeamos um objeto do tipo UserModel que foi recebido a partir de uma consulta ao banco, para um outro do Tipo UserResponse que seria retornado pelo método.

```java
public UserResponse getUserById(UUID id) {
    UserModel userModel = getUserModelById(id);
    UserResponse userResponse = modelMapper.strictMapTo(userModel, UserResponse.class);

    return userResponse;
}
```

Figura 26 *- Exemplo de utilização do ModelMapper*

5.3.5 Lombok

O Lombok é uma biblioteca java que se conecta automaticamente a IDE e cria ferramentas, que geram códigos de forma automática através de anotações feitas nas classes, métodos e atributos presentes na linguagem Java.

Essa biblioteca foi utilizada pelas ferramentas fornecidas que eliminam códigos repetitivos e tos tornam mais legível e de fácil escrita.

Na figura 27 temos um exemplo de utilização feita na aplicação, onde a classe UserModel foi anotada com algumas anotações fornecidas pelo Lombok, e com isso a classe passa ter acesso aos métodos das anotações.

```java
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Getter @Setter
public class UserModel {

    @Id
    @GeneratedValue(generator = "uuid2")
    @GenericGenerator(name = "uuid2", strategy = "uuid2")
    private UUID id;

    @Column(nullable = false)
    private String name;

    4 usages
    @Column(nullable = false)
    private String fullName;
```

Figura 27 *- Exemplo de utilização do Lombok*

5.3.6 Flyway

A ferramenta Flyway nos permite fazer migrações de banco de dados facilmente, ou seja, a partir de *scripts* criados, sempre ao inicializar a aplicação, são executadas todas as instruções descritas nesse arquivos.

A escolha dessa ferramenta foi feita pela compatibilidade com o DB PostgreSQL, por ser facilmente configurável e de fácil execução.

5.3.7 Hibernate

O Hibernate é o framework para consulta e persistência de dados que auxilia no mapeamento de uma representação de dados em um modelo de objetos para um modelo de dados relacional baseado em um esquema ER.

O uso desse *framework* se deu pela agilidade que ele traz no processo de desenvolvimento, além de ter bastante documentação para busca no caso de duvidas ou problemas.

Na figura 28 esta um exemplo de utilização com o mapeamento da Entidade de usuário, onde através de anotações é possível fornecer as informações necessárias para o Hibernate fazer toda representação para um modelo relacional.

```java
@Entity
@Table(name = "user_db")
public class UserModel {

    @Id
    @GeneratedValue(generator = "uuid2")
    @GenericGenerator(name = "uuid2", strategy = "uuid2")
    private UUID id;

    @Column(nullable = false)
    private String name;

    4 usages
    @Column(nullable = false)
    private String fullName;

    @Column(nullable = false, unique = true)
    private String email;


    @ElementCollection
    @CollectionTable(name = "user_cell_phones", joinColumns=@JoinColumn(name="user_id"))
    @Column(name="cell_phone")
    private Set<String> cellPhones = new HashSet<>();
```

Figura 28 *- Exemplo de utilização do Hibernate*

5.3.8 Banco de dados Postgressql

O PostgreSQL é um sistema de banco de dados objeto-relacional de código aberto com mais de 30 anos de desenvolvimento ativo que lhe rendeu uma forte reputação de confiabilidade, robustez de recursos e desempenho.

E por esses motivos, além de possuir muita documentação para consultas, ele foi escolhido como o banco de dados para armazenamento de dos presente nos microsserviços.

5.3.9 Banco de dados Redis

O Redis é um armazenamento de estrutura de dados de chave-valor de código aberto e em memória. Ele suporta diferentes tipos de estruturas de dados abstratas como, string, listas, mapas, conjuntos entre outros.

Seu uso se deu por conta da sua velocidade e facilidade de uso, foi utilizado para armazenar os caches das consultas realizadas na aplicação, afim de tornar as consultas mais dinâmicas e eficientes.

5.3.10 RabbitMQ

O RabbitMQ é um servidor de mensageiria de condigo aberto, implementado para suportar mensagens do protocolo AMQP (Advanced Message Queuing Protocol). Ele possibilita lidar com o tráfego de mensagens de forma rápida e confiável, além de ser compatível com diversas linguagens de programação, possuir interface de administração nativa e ser multiplataforma.

Para a comunicação assíncrona entre os microsserviços utilizamos desse servidores para trafegar as mensagens geradas através de publicações em uma fila para então serem consumidas por outros serviços.

5.3.11 WebClient

O WebClient é uma interface que possibilita a realização de solicitações web baseadas no protocolo HTTP. Com ele conseguimos montar requisições utilizando os métodos HTTP e ainda mapear as repostas para transformá-las em classes Java com muita facilidade.

Essa interface foi utilizada para realização de chamadas síncronas entre os microsserviços da aplicação para possibilitar que os serviços se comuniquem de forma rápida e instantânea.

5.3.12 Zipkin e sleuth

O Zipkin é um sistema de rastreamento distribuído. Com ele conseguimos coletar dados e logs necessários para solucionar problemas em arquiteturas de serviço.

A partir de um id de rastreamento é possível se obter dados da execução dos serviços, como a porcentagem de tempo gasto em um serviço, se as operações falharam ou não e ainda é possível rastrear a chamada por mais de um serviço e ver os logs gerados por eles.

Para se obter informações mais detalhadas ainda dos serviços, foi utilizado também o Sleuth, Ele implementa uma solução para rastreamento, que gera algumas informações necessárias como o nome da aplicação, um “spanID” que é um identificar que muda de um serviço para outro e também gera um “traceID” que é um identificados propagado pelas aplicações em casos de chamadas externas. Com o Sleuth também é possível inserir r informações personalizadas nesses logs.

5.3.13 Json web token

O JWT (Json Web Token), é um padrão para autenticação e troca de informações, ele consiste em um conjunto de solicitações se faz essencial por ser uma forma extremamente segura de compartilhamento de informações e autenticação de usuários. É um formato baseado em texto e amplamente aceito por diversas linguagens, característica que carrega por utilizar JSON como base.

Essa tecnologia foi adotada como forma de autenticar os usuários que tentam utilizar os serviços da aplicação, onde primeiro se deve obter um *token* gerado pela API de Login e depois passar esse *token* para os outros serviços da aplicação para conseguir acessá-los.

Para criptografar e validar os *tokens* enviados e recebidos está sendo adotado o modelo de chaves assimétricas, que consiste em um par de chaves, uma privada que criptografa e deve ser guardada em local seguro, e outra publica que descriptografa e valida se quem criou o *token* foi realmente o emissor esperado, essa chave é entregue a quem deseja validar a autenticidade do emissor dos *tokens recebidos*.

5.3.14 Junit e Mockito

O JUnit é um *framework* de código aberto que auxilia na criação, execução e automação de testes unitários em Java. Para auxiliar ainda mais na criação dos testes foi utilizado também o *framework* Mockito pois ele ajuda na instanciação de classes, além de oferecer formas de controlar o comportamento dessas instancias, o que ajuda a isolar os testes e garantir maior confiabilidade e qualidade dos mesmos.

5.3.15 Git e Gitlab

O Git é um sistema de controle de versionamento distribuído que guarda o histórico de alteração de arquivos, o que é muito útil para desenvolvimento de softwares já que é possível passar pelas versões e verificar diferenças ou comparações.

Para fazer o gerenciamento dos repositórios da aplicação, foi utilizado o GitLab, uma plataforma baseada em Git que oferece diversas ferramentas de versionamento de código, CI/CD e compartilhamento com colaboradores.

5.4 TESTES

Existem dois momentos em que a aplicação é direcionada aos testes, durante o desenvolvimento, quando é desenvolvido alguma nova funcionalidade, ou quando há alteração em códigos já escritos, e também no momento do *deploy*, onde são rodados todos os testes unitários criados durante o desenvolvimento.

5.4.1 Insomnia

O Insomnia é uma ferramenta de Api Client que facilita na realização de requisições *Rest.* Com ela é possível criar coleções onde as requisições ficam salvas, e ainda criar variáveis de ambiente para centralizar configurações e parâmetros repetitivos, além de ser compartilhável. Ela foi escolhida por sua interface amigável e facilitações que oferece para testar as requisições dos serviços.

5.4.2 Testes unitários

Os testes unitários são feitos a nível de código, e buscam aferir sua corretude na menor fração possível, ou seja, na orientação a objetos, essa menos parte poderia ser um método de uma classe. Além de testar minuciosamente o código, é importante ressaltar que determinado método pode resultar em diferentes saídas, então os testes unitários devem cobrir esses cenários afim de garantir um comportamento esperado para a função.

5.5 DEPLOY

Na etapa de *deploy,* ou seja, no momento em que liberamos nosso código para algum ambiente que não seja local, foram utilizadas ferramentas de CI/CD para integração e entrega contínuas e também duas plataformas Paas para alocação dos serviços utilizados pela aplicação.

5.5.1 Integração contínua

A Integração continua é uma pratica de desenvolvimento que visa tornar a integração de códigos mais eficiente, através de processos de construção automática de código e a execução de testes automatizados.

5.5.2 Entrega contínua

A entrega continua é uma evolução da integração continua, pois além das praticas já utilizadas, esse conceito ainda visa que o código esteja apto para entrar nos ambientes de produção através de métodos para simplificar, agilizar e automatizar o lançamento desse código nos repositórios desejados.

5.5.3 GitLab CI/CD

GitLab CI/CD é uma ferramenta para desenvolvimento de software utilizando as metodologias contínuas CI e CD.

Na figura 29 esta um exemplo de *script* usado para realizar o processo de CI na API de User, onde no passo referenciado como “build” é construído o serviço e feito o download de suas dependências e depois na etapa referente a “test” é realizado os testes unitários para verificar se não houve quebra nos métodos.

```yaml
build:
  stage: build
  script:
    - chmod +x ./mvnw
    - ./mvnw clean package
  artifacts:
    paths:
      - target/api-user-0.0.1-SNAPSHOT.jar

test:
  stage: test
  script:
    - chmod +x ./mvnw
    - ./mvnw test
```

Figura 29 *- Script de execução de integração contínua*

E na figura 30 temos a continuação desse exemplo que seria parte de CD da API de User, onde a passo referenciado como “deploy” é responsável por enviar o código para a plataforma Heroku.

```yaml
deploy:
  stage: deploy
  image: ruby:latest
  script:
    - apt-get update -qy
    - apt-get install -y ruby-dev
    - gem install dpl
    - dpl --provider=heroku --app=job-api-user --api-key=$HEROKU_API_KEY
  only:
    - master
```

Figura 30 *- Script de execução de entrega contínua*

5.5.4 Heroku

A Heroku é uma plataforma de nuvem como serviço que suporta várias linguagens de programação. É nela onde foram hospedados alguns dos serviços criados, além dos seus respectivos bancos.

Como ela possui uma utilização gratuita limitada a 5 aplicativos hospedados, os seguintes serviços estão disponibilizados nessa plataforma:

- API de Login e seu DB (PostgresSQL)
- API de User e seus DBs (PostgresSQL e Redis)
- API de Service e seu DB (PostgresSQL)
- API Schedule e seu DB (PostgresSQL)
- API Budget e seu DB (PostgresSQL)

5.5.5 Qoddi

A Qoddi é uma plataforma de nuvem como serviço que suporta várias linguagens de programação. É nela onde foram hospedados alguns dos serviços criados, além dos seus respectivos bancos. Como ela possui uma utilização gratuita limitada a 3 aplicativos hospedados, os seguintes serviços estão disponibilizados nessa plataforma:

- API Gateway
- Service Registry (Eureka)
- *Distributed tracing (*Zipkin Server)

6. CONCLUSÃO

Ao termino deste trabalho, obteve-se toda a documentação das regras de negocio e também houve o desenvolvimento do *back*-end do sistema, baseando-se no estudo de caso apresentado.

Para possibilitar o desenvolvimento desse trabalho, foram estudados novos conceitos e tecnologias, dentre as utilizadas, destaca-se o Ecossistema do Spring, que com seus módulos disponibilizadas fornece uma vasta quantidade de bibliotecas que facilitam o desenvolvimento das aplicações. Além dele, trabalhar com as plataformas Paas resultou na aquisição de conhecimento dos fluxos de deploy que antes eram desconhecidos pelo autor.

Sobre as adversidades encontradas no processo de desenvolvimento, vale ressaltar a dificuldade do entendimento sobre os conceitos de DDD que foram usados para as definições dos microsserviços, pois há pouco conteúdo que realmente esclareça e oriente sobre como realizar a abordagem desse tema. Além disso, houveram problemas na disponibilização dos serviços em ambiente cloud, pois muitas das configurações utilizadas localmente, não surtiam o mesmo efeito nas plataformas PaaS usadas, o que acarretou na necessidade de modificar os arquivos de configurações em vários pontos dos aplicativos.

Com isso, vale destacar o grau de complexidade das tecnologias e conceitos utilizados para o que se obteve até o presente momento e enfatizar a qualidade do que se foi implementado, e, tendo esses pontos em mente, conclui-se que o aplicativo tem grandes chances de comercialização após uma futura e possível finalização, e também que possui uma arquitetura robusta que atende bem aos requisitos do sistema.

6.1 TRABALHOS FUTUROS

Com a análise e documentação prontas, o aplicativo tem grandes chances de ser comercializável, então para isso, como proposta de trabalhos futuros, traria a implementação de uma interface que consumisse os serviços já implementados, além da implementação dos requisitos que não foram implementados durante o desenvolvimento desse trabalho, e

ainda, futuramente, seria possível realizar a integração do Job com alguma plataforma de pagamentos, tornando assim a aplicação realmente utilizável para fins comerciais e por possíveis clientes.

REFERÊNCIAS

BAFNA, Jitendra. **Spring Cloud and Spring Boot, Part 2: Implementing Zipkin Server For Distributed Tracing.** Disponível em: <https://dzone.com/articles/spring-cloud-amp-spring-bootimplementing-zipkin-se>. Acesso em: 7 ago. 2022.

BERNARDO, Fernanda. **Git: o que é, para que serve e principais comandos Git**. Disponível em: <https://blog.betrybe.com/git/>, Acesso em: 7 ago. 2022.

CAMPOS, Ana Cristina. **IBGE: informalidade atinge 41,6% dos trabalhadores no país em 2019**. Disponível em <https://agenciabrasil.ebc.com.br/economia/noticia/2020-11/ibge-informalidade-atinge-416-dos-trabalhadores-no-pais-em-2019#:~:text=A informalidade no mercado de,39%2C3 milhões de pessoas>. Acesso em: 01 mai. 2021.

CEDRO TECNOLOGIES. **RabbitMQ: o que é e como utilizar**. Disponível em: <https://blog.cedrotech.com/rabbitmq-o-que-e-e-como-utilizar>. Acesso em: 7 ago. 2022.

CI/CD. **CI/CD: integração e entrega contínuas.** Disponível em: <https://ww.redhat.com/pt-br/topics/devops/what-is-ci-cd>. Acesso em: 7 ago. 2022.

CRAFT. **Site do Craft**. Disponível em: <https://www.crafty.work/>. Acesso em: 17 mar. 2022.

DDD. **Domain-Driven Design (DDD): Um Resumo.** Disponível em: <https://engsoftmoderna.info/artigos/ddd.html>. Acesso em: 7 ago. 2022.

DIGITÉ. O que é teste de unidade?. Disponível em: <https://www.digite.com/pt-br/agile/testes-unitarios/>. Acesso em: 7 ago. 2022.

EUREKA SERVICE DISCOVERY. Lista gerenciada pela Alura. Disponível em: <https://cursos.alura.com.br/forum/topico-eureka-service-discovery-124159>.Acesso em: 7 ago. 2022.

FERREIRA, Luiz. **Autenticando APIs utilizando JWT com chaves Simétricas e Assimétricas**. Disponível em: <https://medium.com/@luizfrra/autenticando-apis-utilizando-jwt-com-chaves-sim%C3%A9tricas-e-assim%C3%A9tricas-eeb23d85fce7#:~:text=No%20contexto%20de%20JWT%2C%20%C3%A9,valida%C3%A7%C3%A3o%2C%20permitindo%20que%20os%20mesmos>. Acesso em: 7 ago. 2022.

FLYWAY. **Site do Flyway**. Disponível em: <https://flywaydb.org/>. Acesso em: 7 ago. 2022.

GET NINJAS. **Site do Getninjas**. Disponível em: <https://www.getninjas.com.br/>. Acesso em: 17 mar. 2022.

GITLAB. **Site do GitLab**. Disponível em: <https://about.gitlab.com/>. Acesso em: 7 ago. 2022.

GITLAB CI/CD, **GitLab CI/CD**. Disponível em: <https://docs.gitlab.com/ee/ci/>. Acesso em: 7 ago. 2022.

HEROKU. **Site do Heroku**. Disponível em: <https://www.heroku.com/>. Acesso em: 17 mar. 2022.

HIBERNATE. **Site do Hibernate**. Disponível em: <https://hibernate.org/>. Acesso em: 17 mar. 2022.

INTELLIJ IDEIA. **What is IntelliJ IDEA?**. Disponível em: <https://www.jetbrains.com/idea/features/>, Acesso em: 17 mar.2022.

IZALMO. **Desenvolvendo com Hibernate**. Disponível em: <https://www.devmedia.com.-br/desenvolvendo-com-hibernate/14756>. Acesso em: 17 mar. 2022.

JUNIT . **The 5th major version of the programmer-friendly testing framework for Java and the JVM.** Disponível em: <https://junit.org/junit5/>. Acesso em: 17 mar. 2022.

LOMBOK. **Project Lombok**. Disponível em: <https://projectlombok.org/>. Acesso em: 7 ago. 2022.

MALEK, Renata dos Santos; SILVA, Washington Sidney; OYAKAMA, Ricardo Satoshi. **Offer and Demand Portal for Residential Services**. 2020. 20p. Trabalho de Conclusão de Curso - Faculdade de Tecnologia da Zona Leste, Avenida Águia de Haia, 2983. Cidade A.E. Carvalho, Brasil, 2020.

MODELMAPPER. **Modelmapper Simple, Intelligent, Object Mapping**. Disponível em: <http://modelmapper.org/>. Acesso em: 7 ago. 2022.

MOCKITO. **Tasty mocking framework for unit tests in Java**. Disponível em: <https://site.mockito.org/>. Acesso em: 7 ago. 2022.

NEWMAN, Sam. **Monolith to microservices**, 1nd edition. O'Reilly Media, Inc, 2019.

NEWMAN, Sam. **Building microservices**, 2nd edition. O'Reilly Media, Inc, 2021.

NOLETO, Cairo. **JWT(JSON Web Tokens): o que é e como usar na prática?**. Disponível em: <https://blog.betrybe.com/tecnologia/jwt-json-web-tokens/>. Acesso em: 7 ago. 2022.

PERRY, Tim. **Sending HTTP requests with Spring WebClient**. Disponível em: <https://reflectoring.io/spring-webclient/>. Acesso em: 7 ago. 2022.

POLIANA, Barbara. **Domain Driven Design: o que é DDD, 3 princípios e como usar?.** Disponível em: <https://blog.betrybe.com/tecnologia/domain-driven-design/>. Acesso em: 7 ago. 2022.

POSTGRESSQL. **PostgreSQL: The World's Most Advanced Open Source Relational Database.** Disponível em: <https://www.postgresql.org/>. Acesso em: 7 ago. 2022.

QOODI. **A modern hosting app platform**. Disponível em: <https://qoddi.com/>. Acesso em: 7 ago. 2022.

REDIS. **A vibrant, open source database**. Disponível em: <https://redis.io/>. Acesso em: 7 ago. 2022.

SALDANHA, Alexandre. **Análise do impacto das campanhas de marketing digital no crescimento profissional dos profissionais autônomos**. 2020. 22p. Trabalho de Conclusão de Curso - Universidade do Sul de Santa Catarina, Santa Catarina, 2020.

SPRING. **Projects**. Disponível em: <https://spring.io/projects>. Acesso em: 7 ago. 2022.

DEVMEDIA. **E aí? Como você testa seus códigos?**. Disponível em: <https://www.devmedia.com.br/e-ai-como-voce-testa-seus-codigos/39478>. Acesso em: 7 ago. 2022.

TIBCO. **O que é um API Gateway?**. Disponível em: <https://www.tibco.com/pt-br/reference-center/what-is-an-api-gateway>. Acesso em: 7 ago. 2022.

TRIIDER. **Uma nova forma decontratar profissionais**. Disponível em: <https://www.triider.com.br/>. Acesso em: 17 mar.2022.

WILLIANS, Wesley. **O que é DDD – Domain Driven Design**. Disponível em: <https://fullcycle.com.br/domain-driven-design/>. Acesso em: 7 ago. 2022.

ZIPKIN. **Site do ZIPKIN**, 2022. Disponível em: <https://zipkin.io/>. Acesso em: 7 ago. 2022.