UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Euller Henrique Bandeira Oliveira

Arquitetura de Microsserviços: Um Estudo de Caso

Uberlândia, Brasil

2023

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Euller Henrique Bandeira Oliveira

Arquitetura de Microsserviços: Um Estudo de Caso

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Orientador: Fabiano Azevedo Dorça

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2023

Euller Henrique Bandeira Oliveira

Arquitetura de Microsserviços: Um Estudo de Caso

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, 01 de dezembro de 2023:

Fabiano Azevedo Dorça
Orientador

Rodrigo Sanches Miani
Professor

Paulo Rodolfo da Silva Leite Coelho
Professor

Uberlândia, Brasil
2023

Dedico este trabalho ao Euller inseguro de anos atrás que duvidava de que era capaz de realizar o seu maior sonho.

Agradecimentos

À minha mãe, Elisabet: Agradeço à pessoa que desde sempre acreditou em meu potencial, agradeço pelo esforço descomunal para me ajudar a chegar até aqui, por sempre ter feito o possível e o impossível para me agradar, por sempre ter acreditado que eu conseguiria cumprir este objetivo; Ao meu pai, Edvaldo: Agradeço por ter me aconselhado a não desistir de iniciar este curso somente por odiar matemática, por ter investido na minha educação e pelo apoio cada vez maior nos últimos anos.

À minha avó, Glória: Agradeço pelos almoços, tortas e biscoitos. Agradeço por ter sido minha segunda mãe por um bom tempo. Agradeço por sempre ter acreditado que eu chegaria até aqui; Ao meu avô, Jair: Agradeço por todas as vezes que você me levou e me buscou na escola no ensino fundamental, pelas ligações na pandemia e por sempre ter me apoiado.

Ao meu tios, Flávio e Carlos: Agradeço pelo apoio e incentivo ao longo dos anos.

À minha namorada, Carolayne: Agradeço por ter me acompanhado nessa jornada nos últimos 4 anos, por todas as vezes que você ouviu minhas explicações mirabolantes atenciosamene, por sempre ter me acalmado e me confortado.

Aos meus professores do ensino médio: Agradeço pelos inúmeros incentivos e ensinamentos. Agradeço especialmente ao Ronaldo (Professor de Matemática) que devido ao fato de ter conseguido ensinar algúem que era um desastre em matemática me fez acreditar que eu poderia cursar um curso de exatas.

À Universidade Federal de Uberlândia e à Faculdade de Computação: Agradeço ao ensino gratuito e de qualidade e aos valiososos ensinamentos de seus professores. Agradeço especialmente ao Fabiano (Orientador) por ter feito eu me encantar com a linguagem Java ao lecionar a disciplina Programação Orientada a Objetos 2, por ter se interessado pela minha ideia de fazer o TCC sobre a Arquitetura de Microsserviços, por ter sugerido que eu realizasse um estudo de caso, pelas leituras minuciosas, pelas sugestões de melhorias e pelas correções.

À empresa, A.R.PHOENIX: Agradeço por terem me acolhido e me instruído, por terem notado e acreditado em meu potencial e por terem autorizado que eu realizasse este estudo de caso. Agradeço especialmente à Ana Carla por ter me dado “Um projetinho para eu chamar de meu”, ao Pablo Santos por ter sanado minhas dúvidas milhões de vezes, ao Rafael Delia por sempre dar dicas para eu me tornar um desenvolvedor melhor, ao Paulo Bastos por ser um gestor preocupado e gentil, à Karine Ruas por ter visto potencial naquele ser ansioso que apareceu para ser entrevistado e a inúmeros outros colaboradores.

"Eu estou sempre fazendo aquilo que não sou capaz,
numa tentativa de aprender como fazê-lo."
- Vincent Van Gogh

Resumo

A arquitetura de microsserviços é uma arquitetura cada vez mais popular e relevante no universo da engenharia de software. Portanto, compreendê-la minimamente é cada vez mais importante para o desenvolvedor se manter atualizado no mercado de trabalho. Em vista disso, esta monografia realizou um estudo de caso do tipo exploratório com abordagem qualitativa com o objetivo de identificar e exemplificar as vantagens e desvantagens da arquitetura de microsserviços após implementá-la e investigá-la. Seu desenvolvimento adotou o modelo iterativo no qual utilizou 4 etapas: Requisitos, Modelagem, Implementação e Implantação. Após seu desenvolvimento ser finalizado e a investigação ser realizada, as seguintes vantagens e desvantagens foram identificadas: Vantagens (Atribuição mais fácil da implementação, Implantação mais fácil, Manutenção mais fácil e Escabilidade mais fácil) e Desvantagens (Utilização limitada do Hibernate, Inadequado para geração de relatórios extensos, Implementação mais complexa e Utilização limitada de Transactions)

Palavras-chave: Engenharia de Software. Sistemas Distribuídos. Arquitetura de Software. Arquitetura Monolítica. Arquitetura de Microsserviços.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Este TCC se dedicará a estudar um ecossistema de microsserviços real desenvolvido no projeto (Migração Sistemas Internos) e na empresa em que atuo atualmente (A.R.PHOENIX).

A necessidade do desenvolvimento desse ecossistema de microsseviços surgiu devido ao fato de que o sistema interno da empresa (Sistema monolítico legado) funciona somente no Internet Explorer. Como a Microsoft encerrou o seu suporte em 15 de junho de 2022 (G1, 2022) e desabilitou sua abertura em 14 de fevereiro de 2023 (DIGITAL, 2023), é possível presumir que eventualmente o modo Internet Explorer presente no Microsoft Edge chegará ao fim e afetará o pleno funcionamento da empresa.

O ecossistema de microsserviços em questão é destinado ao controle de horas e pagamento. Por questões de segurança, regras de negócios intrínsecas à empresa serão retiradas. Sendo assim, o estudo irá se basear em uma versão mais simples do ecossistema real. A versão será mais simples, todavia terá diversas melhorias em sua arquitetura, em seu código, em sua modelagem do banco de dados etc.

O ecossistema é composto pelos seguintes microsserviços: ms-usuario, ms-tarefa, ms-timesheet, ms-horas-extras, ms-relatorio, ms-faturamento, ms-email-producer e ms-email-consumer. Quando fui alocado nesse projeto, o ecossistema estava na fase inicial de desenvolvimento, nessa fase algumas partes dos microsserviços ms-usuario, ms-tarefa e ms-timesheet já existiam.

Ao investigar esses microsserviços para aprimorar minha compreensão acerca das regras de negócios, detectei as necessidades a seguir: **Atualizar as dependências** (Externas, Framework, Linguagem), **Integrar os microsserviços** (ms-usuario, ms-tarefa e ms-timesheet), **Realizar refatorações** (Remover de sua origem os serviços que foram decompostos; Retirar métodos, classes e pastas desnecessárias; Melhorar a organização dos métodos, classes e pastas; Aperfeiçoar a lógica presente em vários métodos) e **Ampliar a documentação** (Criar READMEs, Criar diagramas, Criar Javadocs).

Logo após, recebi as seguintes responsabilidades: **Desenvolver novas funcionalidades para os microsserviços** (ms-tarefa, ms-usuario, ms-timesheet) e **Criar os microsserviços** (ms-hora-extra, ms-faturamento, ms-email-producer, ms-email-consumer, ms-relatorio).

1.1 Objetivos

Esta seção possui o objetivo de apresentar o objetivo geral e os objetivos específicos desta monografia.

1.1.1 Objetivo Geral

O objetivo geral deste estudo de caso é identificar e exemplificar as vantagens e desvantagens da arquitetura de microsserviços após implementar e investigar um ecossistema de microsserviços.

1.1.2 Objetivos Específicos

Para o objetivo geral ser concluído, é preciso que os seguintes objetivos específicos sejam cumpridos:

- Descrever os principais conceitos relacionados à arquitetura de microsserviços.
- Descrever as tecnologias geralmente utilizadas em um ecossistema de microsserviços.
- Definir os requisitos do ecossistema de microsserviços.
- Realizar a modelagem do ecossistema de microsserviços.
- Implementar o ecossistema de microsserviços (Back-End).
- Implementar o sistema web (Front-End).
- Identificar e exemplificar as vantagens da arquitetura de microsserviços.
- Identificar e exemplificar as desvantagens da arquitetura de microsserviços.

1.2 Justificativa

Compreender a arquitetura de microsserviços em seu nível teórico e prático é um requisito cada vez mais solicitado no mercado de trabalho. Em vista disso, este estudo de caso justifica-se pela necessidade de se criar uma referência para que possa ser consultada por qualquer desenvolvedor que queira entender essa arquitetura ou que esteja em busca de razões para adotá-la.

Com base na fundamentação teórica (Conceitos), desenvolvimento (Implementação) e resultados (Vantagens e desvantagens), o leitor conseguirá compreender a arquitetura de microsserviços e deliberar se sua adoção é apropriada ou não.

1.3 Metodologia

Esta seção possui a função de definir a metodologia científica adotada nesta monografia e a metodologia de desenvolvimento adotada no desenvolvimento do ecossistema de microsserviços.

1.3.1 Metodologia Científica

Esta monografia é um estudo de caso (1) do tipo exploratório (2) com abordagem qualitativa (3).

1. Estudo de caso

   De acordo com Menezes (2023), um estudo de caso é "uma estratégia de pesquisa científica que analisa um fenômeno atual em seu contexto real e as variáveis que o influenciam", seu objetivo é "produzir conhecimento a respeito de um fenômeno" e para ele ser realizado é preciso que os passos a seguir sejam seguidos: "a) Identificação de um problema de pesquisa b) Levantamento dos dados c) Análise do contexto d) Conclusões sobre o problema".

2. Pesquisa Exploratória

   A pesquisa exploratória é um tipo de pesquisa científica na qual o pesquisador deve estudar profundamente o tema escolhido antes de elaborar hipóteses com o objetivo de que ao serem feitas, possuam um embasamento teórico e científico (SIGNIFICADOS, 2023).

3. Pesquisa Qualitativa

   De acordo com Minayo (2001 apud COELHO, 2017), "A pesquisa qualitativa trabalha com o universo de significados, motivos, aspirações, crenças, valores e atitudes, o que corresponde a um espaço mais profundo das relações, dos processos e dos fenômenos que não podem ser reduzidos à operacionalização de variáveis".

1.3.2 Metodologia de Desenvolvimento

O ecossistema de microsserviços utilizou o modelo iterativo de desenvolvimento. De acordo com Larman e Basili (2003), esse modelo é uma melhoria do modelo incremental, pois ao contrário do que ocorre em tal modelo, a cada incremento realizado, isto é, a cada etapa realizada, ela pode ser iterada, ou seja, ela pode ser reiniciada para que correções ou aprimoramentos possam ser feitos.

O ecossistema em questão utilizou 4 etapas em seu desenvolvimento: Requisitos(1), Modelagem(2), Implementação(3) e Implantação(4). [Figura 1]

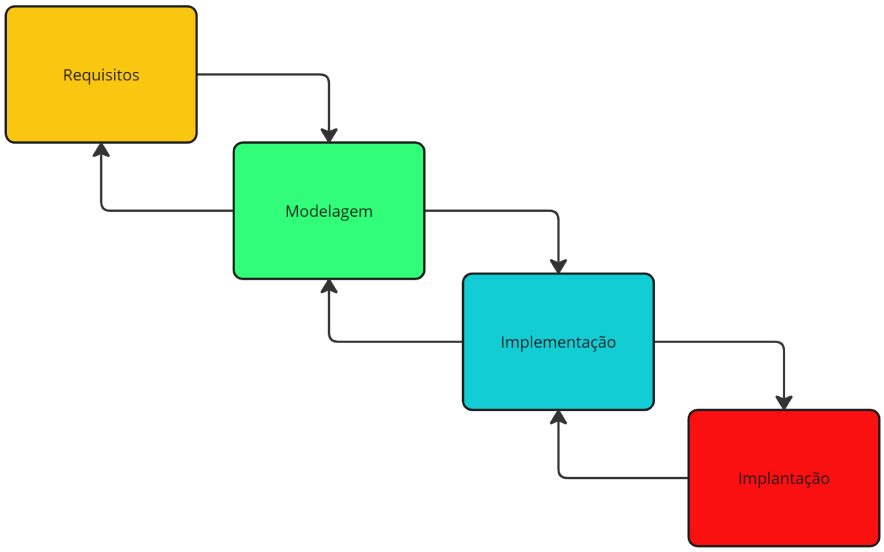


Figura 1 – Modelo Iterativo de Desenvolvimento

1. Requisitos

   Esta etapa possui o objetivo de levantar e análisar os requisitos do sistema (VALENTE, 2022a).

   Tal etapa utilizou os seguintes documentos:

   1.1. Requisitos Funcionais e Não Funcionais:

   Segundo Valente (2022a), os requisitos funcionais definem o que o sistema deve fazer, isto é, quais funcionalidades ele deve ter e os requisitos não-funcionais definem as restrições que tais funcionalides devem considerar.

   1.2. Casos de Uso

   De acordo com Valente (2022a), "casos de uso são documentos textuais de especificação de requisitos" . Tais documentos são escritos sob o ponto de vista de um ator (usuário ou outro sistema) que utilizará uma determinada funcionalidade do sistema. Um caso de uso contém o passo a passo que o ator deve seguir para a ação em questão ser realizada. Cada caso de uso contém o fluxo normal/feliz e o fluxo alternativo/triste (VALENTE, 2022a).

2. Modelagem

   A modelagem é a ação de converter os requisitos definidos anteriormente em modelos para que possam ser convertidos em códigos posteriormente. Portanto, esta etapa possui o objetivo de criar o projeto do sistema, ou seja, de definir premissas importantes que deverão ser respeitadas ao longo do desenvolvimento do software (DEVMEDIA, 2011).

   Tal etapa utilizou os seguintes modelos:

2.1. Diagramas de Caso de Uso:

Diagramas de caso de uso são representações gráficas de casos de uso. Em tais diagramas, os atores são representados graficamente como pequenos bonecos e os casos de uso como elipses. Além disso, três relacionamentos são utilizados na construção desses diagramas: 1) O ator atua no caso de uso (->), 2) Caso de uso 1 inclui caso de uso 2 («includes ») e 3) Caso de uso 2 extende caso de uso 1 («extends») (VALENTE, 2022a) (VENTURA, 2014).

2.2. Diagramas de Atividade:

Diagramas de atividade são utilizados para representar o fluxo de um processo, sistema, algoritmo ou caso de uso. Esse digarama contém 5 componentes: 1) Ações: Retângulos arredondados, 2) Nó de decisão: Diamantes, 3) Fluxo de controle: Setas conectoras e 4) Nó inicial: Círculo preto 5) Nó final: Círculo preto delineado (LUCIDCHART, 2023).

2.3. Diagrama Entidade Relacionamento (DER):

Um diagrama de entidade e relacionamento é a representação gráfica de um modelo de entidade e relacionamento (MER). Esse tipo de diagrama é utilizado principalmente para representar a estrutura de um banco de dados relacional. Em tal diagrama, existem 3 componentes: 1) Entidades (Tabelas), 2) Atributos (Colunas) e 3) Relacionamentos (1..1, 1..n, n..n) (JOEL, 2014).

2.4. Diagramas de Arquitetura:

Diagramas de arquitetura são representações visuais dos componentes pertencentes a um sistema, ou seja, da arquitetura de um sistema. Exemplos de arquitetura: Arquitetura em camadas, arquitetura cliente-servidor, arquitetura orientada a eventos e arquitetura de microsserviços (AMAZON, 2023a).

3. Implementação

A implementação é o ato de converter a teoria (modelagem baseada nos requisitos) em prática (código), ou seja, é a etapa em que se utiliza uma linguagem de programação para transformar o modelo elaborado em códigos executáveis (HUDSON, 2007).

- Conceitos utilizados:
  - Ecossistema De Microsserviços:

    Arquitetura de Microsserviços, Arquitetura Cliente-Servidor, Arquitetura Rest, Arquitetura Orientada a Eventos, API, HTTP e POO.
  - Sistema Web:

    Arquitetura Monolítica.

- Tecnologias utilizadas:
  - Banco de Dados:
    Sql, Microsoft Sql Server.
  - Ecossistema De Microsserviços:
    Java, Maven, Spring Framework, Spring Boot, JPA, Hibernate, Rest Template, Apache Kafka, Spring Email, Lombok, Sonar Lint, Jasper Reports, Jaspersoft Studio, Microsoft Azure Key Vault, IntelliJ IDEA Ultimate e Postman.
  - Sistema Web:
    Html, Css, Javascript e Visual Studio Code.

4. Implantação

   A implantação é o processo de disponibilizar o sistema para os usuários ou outros sistemas utilizarem, ou seja, é a ação de inserir o sistema desenvolvido em um servidor online (IBM, 2021).

   - Conceitos Utilizados:
     - Ecossistema De Microsserviços:
       Integração, Entrega e Implantação Contínua (CI/CD).
   - Tecnologias utilizadas:
     - Banco de Dados e Ecossistema De Microsserviço:
       Git, Docker, Kubernetes e Microsoft Azure DevOps (Boards, Repos e Pipelines).
     - Sistema Web:
       Git, Git Hub e Git Hub Pages.

2 Fundamentação Teórica

Este capítulo possui o objetivo de embasar teoricamente as seções subsequentes ao elencar e descrever os principais conceitos e as tecnologias geralmente necessárias para um ecossistema de microsserviços ser desenvolvido.

2.1 Conceitos Elementares

Esta seção possui o objetivo de apresentar e explicar os conceitos essenciais relacionados à implementação e à implantação de um ecossistema de microsserviços.

1. Implementação

    1.1. Arquitetura Monolítica

    A arquitetura monolítica é a arquitetura presente em uma aplicação na qual todas as suas funcionalidades estão presentes em um único sistema (NEWMAN, 2020a).

    De acordo com (NEWMAN, 2020a), tal arquitetura possui 2 tipos: **Sistema monolítico de um só processo** (É um sistema em que o código inteiro é implatado em um único processo. É possível ter várias instâncias desse processo com o objetivo de aumentar a robusteza ou a escala. Entretanto, isso não altera o fato de que a execução do código inteiro ocorre somente em um único processo. Tal sistema pode ser considerado um sistema distribuído simples, já que na maioria dos casos ele usa um banco de dados para ler ou armazenar dados. Devido à simplicidade desse tipo de arquitetura monolítica, possivelmente é o mais utilizado pelas empresas.) e **Sistema monolítico modular** (É um sistema em que o seu código foi dividido entre módulos, ou seja, cada módulo contém uma parte do código do sistema. Tal sistema também é implatando em um único processo. Apesar de ser possível desenvolver cada módulo de maneira indepentende, a implantação não é independente, ou seja, os módulos devem ser combinados para o código executável do sistema ser gerado. Devido ao fato de que este tipo de arquitetura monolítica é considerada uma versão mais simples da arquitetura de microsseviços por não possuir a complexidade da computação distribuída, ele pode ser uma excelente opção para muitas empresas.).

    1.2. Arquitetura de Microsserviços

    É uma arquitetura que possui serviços que são implantados de maneira independente e que são modelados com base em um domínio de negócios. Os

microsserviços expõem suas funcionalidades entre si e para sistemas web por meio de uma arquitetura cliente-servidor (API REST) ou uma arquitetura orientada a eventos (PUB/SUB). Tal característica os torna um sistema distribuído. Além disso, cada microsserviço possui seu próprio banco de dados, isto é, suas próprias tabelas (NEWMAN, 2020b) (NEWMAN, 2022).

De acordo com (NEWMAN, 2020b) e (NEWMAN, 2022), suas principais características são: **Implantações independentes** (Para a arquitetura de microsserviços ser adotada é preciso que cada sistema pertencente ao ecossistema possa ser implantado de forma independente. Em outras palavras, ao realizar uma alteração em um sistema, somente o sistema alterado deverá ser implantado novamente. Se uma funcionalidade depender de dois sistemas, dois sistemas terão que ser alterados e implantados. Para que essa alteração e implantação dupla seja evitada é preciso que exista um baixo acoplamento entre os sistemas, ou seja, é preciso que as partes da funcionalidade em questão dependam o mínimo possível uma da outra.), **Design orientado a domínios [DDD]** (Para um ecossistema ser considerado um ecossistema de microsserviço é preciso que a modelagem de cada sistema seja baseada em um domínio, isto é, é preciso que cada sistema represente um dos domínios do negócio. Por exemplo, a tabela cliente e as suas tabelas auxiliares (Tabelas fortemente acopladas a ela) são um domínio de negócio, pois os desenvolvedores e usuários irão manipulá-las ao realizarem buscas, inserções, atualizações e exclusões por meio de um sistema. Tal estratégia faz com que a probabilidade de dois sistemas terem que ser alterados quando uma parte de uma funcionalidade precisar ser corrigida seja bem menor, já que, há uma grande chance de tal parte pertencer somente à um domínio.) e **Responsáveis pelos seus próprios dados** (Para um sistema ser considerado um microsserviço é preciso que ele não compartilhe seu banco de dados, ou seja, é preciso que somente ele consiga manipular diretamente determinadas tabelas. Se o sistema x desejar buscar todos os clientes armazenados na tabela cliente (Pertencente ao sistema y), ele precisará solicitar que o sistema y faça a busca e retorne os dados para ele. Com isso, cada microsserviço terá a capacidade de decidir quais dados serão compartilhados, quais dados serão ocultos e quais operações estarão disponíveis (buscar, atualizar ou excluir). As interfaces expostas (endpoints) de cada microsserviço devem ser estáveis, ou seja, elas não devem ser alteradas constantemente para que não seja necessário fazer alterações em cada microsserviço que depende de pelo menos uma delas.).

1.3. *Arquitetura Representational State Transfer* (REST)

A transferência de estado representacional é uma arquitetura de software criada com o objetivo de definir um conjunto de restrições para que ser-

vidores web sejam mais eficientes. Restrições: Arquitetura Cliente-Servidor, Interface Uniforme, Stateless, Cache e Camadas (NOLETO, 2022).

1.4. Arquitetura Cliente-Servidor

Na arquitetura cliente-servidor, o funcionamento de um software depende de dois compontentes, isto é, o cliente e o servidor. O cliente é responsável por solicitar a um servidor que ele realize uma ação. O servidor é responsável por receber a solicitação do cliente, realizar a ação solicitada, retornar para o cliente um status (indica se a ação foi realizada ou não) e um dado (se for necessário). Ex: Front-End solicita que o Back-End busque as informações de um determinado usuário (NATIVE, 2022).

1.5. *Application Programming Interface* (API)

A interface de programação de aplicativos é um conjunto de rotinas e padrões definidos por um software com o objetivo de disponibilizar funcionalidades para outras aplicações (REDHAT, 2023b).

1.6. *Hypertext Transfer Protocol* (HTTP)

O protocolo de transferência de hipertexto determina como a comunicação entre um navegador e um servidor web deve ser realizada. Requests são realizadas por verbos HTTP para o servidor e o servidor as respondem (MOZILLA, 2023b). Os principais verbos HTTP são o GET, o POST, o PUT e o DELETE. O verbo GET é utilizado para requisitar um dado do servidor, o POST é utilizado para enviar um dado ao servidor, o PUT é utilizado para atualizar um dado armazenado no servidor e o DELETE é utilizado para apagar um dado armazenado no servidor (MOZILLA, 2023c).

1.7. Arquitetura Orientada a Eventos

Um evento é uma ação que produz uma alteração de estado de um sistema. Ex: Botão que ao ser clicado solicita que um sistema notifique outro sistema que um determinado evento ocorreu (REDHAT, 2019a).

A arquitetura orientada a eventos é uma arquitetura de software que possui a programação assíncrona como principal característica. Portanto, ao contrário do que ocorre na arquitetura cliente-servidor, o sistema que solicitou uma ação não precisa esperar que o outro sistema realize a ação e envie uma resposta. Com isso, o fluxo do sistema solicitante não é interrompido (TIBCO, 2023).

De acordo com a (REDHAT, 2019b), tal arquitetura possui três componentes: **Produtor** (É responsável por notar um evento e o converter em uma mensagem. Ele não sabe quem irá consumir a mensagem nem como o consumidor irá reagir a mensagem), **Canal** (É responsável por transmitir a mensagem para o consumidor) e **Consumidor** (É responsável por receber a mensagem

enviada pelo produtor e realizar uma determinda ação com base na mensagem recebida)

Segundo a Microsoft (2023c), quando um evento ocorre nos dois modelos de tal arquitetura (Pub/Sub, Streaming), uma mensagem é publicada em um canal e os seus subescritores recebem a mensagem. O comportamento se difere somente após a mensagem ser entregue para cada subescritor. No modelo Pub/Sub a mensagem **é excluída** do canal, ou seja, os próximos subescritores **não irão** receber a mensagem em questão quando se inscreverem. Já no modelo Streaming, a mensagem **não é excluida** do canal, ou seja, os próximos subescritores **irão receber** a mensagem em questão quando se inscreverem.

1.8. Programação Orientada a Objetos (POO)

A programação orientada a objetos é um paradigma criado com o objetivo de aproximar o mundo lógico da programação ao mundo real. Essa aproximação é feita ao definir que tudo pertencente ao mundo real é um objeto que pode ser representado no mundo lógico da programação e que assim como no mundo real, um objeto precisa ser criado por meio de uma fôrma com suas características e funcionalidades, em outras palavras, um objeto precisa ser instanciado por meio de uma classe com seus atributos e métodos (NOLETO, 2020).

De acordo com a DevMedia (2023e), tal paradigma possui quatro pilares: **Encapsulamento** (O encapsulamento é realizado ao tornar os atributos privados e ao criar métodos públicos de acesso e de modificação para esses atributos, ou seja, os getters e setters. O objetivo dessa estratégia é restringir modificações não desejadas dos atributos, já que para um atributo ser modificado é preciso que o setter seja chamado e tal setter pode ter uma lógica que definirá se a modificação será autorizada ou não.), **Herança** (O objetivo da herança é reaproveitar atributos e métodos de uma classe parecida com outra. Por exemplo, a classe Aluno é parecida com a classe Pessoa, pois um Aluno obviamente é uma Pessoa. Então, os atributos e métodos da classe Pessoa devem ser herdados pela classe Aluno.), **Abstração** (A abstração é realizada ao definir que determinados métodos de uma classe são abstratos (Isso caracteriza a classe como uma classe abstrata) ou ao definir que todos os métodos de um classe são abstratos (Isso caracteriza a classe como uma interface). Essas estratégias possuem o objetivo de isolar a implementação para que o conceito do polimorfismo possa ser aplicado.) e **Polimorfismo** (O polimorfismo se refere às várias implementações que um método pertencente a uma classe abstrata ou interface possui, ou seja, ele é implementado de várias formas diferentes nas classes filhas. Com isso, cada classe filha herdará o que for semelhante da classe pai e o que for diferente será implementado nos métodos polimórficos.).

2. Implantação

2.1. Integração, Entrega e Implantação Contínua (CI/CD)

De acordo com a JetBrains (2023), estes são os conceitos do CI/CD: **Integração Contínua [Continues Integration-CI]** (Para o desenvolvimento de um software adotar o uso da integração contínua, é preciso que cada desenvolvedor exponha frequentemente suas alterações no código ao realizar commits para um sistema de controle de versões e teste o sistema implantado para verificar se algum erro surgiu. Graças a esse princípio, o código se mantém atualizado. Com isso, se alguma falha ocorrer, o tempo para a fonte do problema ser encontrada e corrigida se torna menor.), **Entrega Contínua, [Continues Delivery-CD]** (Para o desenvolvimento de um software adotar o uso da entrega contínua, é preciso que um desenvolvedor dev-ops crie esta etapa em uma pipeline. Tal etapa realizará automaticamente o build do sistema (Transforma o código escrito em um código executável) a cada commit realizado.) e **Implantação Contínua [Continues Deployment-CD]** (Para o desenvolvimento de um software adotar o uso da implantação contínua, é preciso que um desenvolvedor dev-ops crie esta etapa na pipeline criada na etapa anterior. Tal etapa realizará automaticamente o deploy do sistema (Insere o código executável em um servidor online) a cada build gerado na etapa anterior.).

2.2 Tecnologias Utilizadas

Esta seção possui o objetivo de elencar e descrever as tecnologias geralmente utilizadas na implementação e na implantação de um ecossistema de microsserviços.

1. Implementação

1.1. Banco de Dados

1.1.1. *Structured Query Language* (SQL)

A linguagem de consulta estruturada é uma linguagem de programação utilizada para buscar e salvar valores em um banco de dados relacional. Um banco de dados relacional utiliza tabelas para armazenar informacões, cada tabela pode se conectar à outra tabela, cada coluna contém os valores associados a um determinado atributo e cada linha contém os valores associados aos atributos em uma posição da coluna (AMAZON, 2023b).

De acordo com a DevMedia (2023c), o SQL possui os seguintes tipos de comandos: **DDL** (Linguagem de definição de dados): create, alter

e drop; **DQL** (Linguagem de consulta de dados): select; **DML** (Linguagem de manipulação de dados): insert, update e delete; **DCL** (Linguagem de controle de dados): grant e revoke; **DTL** (Linguagem de controle de transações): begin, commit e rollback.

1.1.2. Microsoft SQL Server

O SQL Server é um sistema de gerenciamento de bando de dados (SGBD) desenvolvido pela Microsoft. De acordo com a IBM (2023), "Um sistema de gerenciamento de banco de dados nada mais é do que um sistema computadorizado de manutenção de dados. Os usuários do sistema recebem facilidades para executar vários tipos de operações em tal sistema, seja para manipulação dos dados no banco de dados ou para o gerenciamento da própria estrutura do banco de dados.".

1.2. Ecossistema de Microsserviços

1.2.1. Java

O Java é uma linguagem de programação orientada a objetos. O código desenvolvido por meio do Java pode ser executado em qualquer sistema operacional graças ao fato de que ao ser compilado pelo JDK (Java Development Kit - Kit de Desenvolvimento Java), ele é transformado em um bytecode e é interpretado pela JVM (Java Virtual Machine - Máquina Virtual Java). Após o desenvolvimento do código ser finalizado, ele é transformado em um .jar e executado por meio da JRE (Java Runtime Environment - Ambiente de Execução Java) que contém a JVM + as bibliotecas padrões do Java (IBM, 2023).

1.2.2. Spring Framework

O Spring é um Framework criado com o objetivo de substituir o JAVA EE para que o desenvolvimento de aplicações voltadas para a web se tornem mais fáceis. Esse framework utiliza os padrões de projeto Inversão de controle e Injeção de dependência. Por meio da inversão de controle, o container se torna o responsável por instanciar os objetos ao invés de cada classe. O container representa o núcleo do framework, ele é responsável por criar e gerenciar os componentes da aplicação, ou seja, os beans. Por meio da injeção de dependência, o controle é invertido ao indicar em determinada classe que ela quer receber uma instância de um objeto (DEVMEDIA, 2023a).

1.2.3. Spring Boot

O Spring Boot é um Framework criado com o objetivo de facilitar as configurações de um projeto que utiliza o Spring Framework. Ao invés das configurações serem realizadas por meio de arquivos xml e por meio de códigos extensos, a maioria das configurações são realizadas por meio

de anotações. Com isso, o desenvolvedor pode focar logo no início nas regras de negócio da aplicação ao invés de perder um tempo substancial realizando configurações (GEEKHUNTER, 2022b).

1.2.4. *Java Persistence API* (JPA)

A API de persistência Java é uma especificação oficial que possui classes, interfaces e anotações que ajudam o desenvolvedor a abstrair a persistência. O JPA não possui códigos executáveis, ele apenas define como deve-se implementá-lo, ou seja, ele é uma "interface" (AFONSO, 2019).

1.2.5. Hibernate

O Hibernate é um framework que implementa o JPA, ele foi criado com o objetivo de abstrair a persistência de dados ao mapear um objeto para o universo relacional, ou seja, o Hibernate é uma ferramenta de mapeamento objeto-relacional (ORM) que transforma o HQL (Hibernate Query Language) em SQL (Structured Query Language) (DEVMEDIA, 2023d).

1.2.6. Microsoft Azure Key Vault

De acordo com a Microsoft (2023a), o Azure Key Vault é um serviço desenvolvido para armazenar e acessar segredos de forma segura. Um segredo é o valor de alguma variável que pode ser acessado somente por determinados usuários.

De acordo com a Microsoft (2023g), ele pode gerenciar: **Segredos** (Ele oferece um modo descomplicado de amazenar e controlar com segurança o acesso a senhas, tokens, certificados, chaves de API e outros segredos. "O armazenamento centralizado de segredos do aplicativo no Azure Key Vault permite que você controle sua distribuição. O Key Vault reduz consideravelmente a probabilidade de os segredos serem vazados acidentalmente. Ao usar o Key Vault, os desenvolvedores de aplicativos não precisam mais armazenar informações de segurança no aplicativo. A falta de necessidade de armazenar informações de segurança em aplicativos elimina a necessidade de inserir essas informações como parte do código. Por exemplo, um aplicativo pode precisar se conectar a um banco de dados. Em vez de armazenar a cadeia de conexão no código do aplicativo, armazene-o com segurança no Key Vault." (MICROSOFT, 2023g).), **Chaves** (Ele oferece uma forma fácil de criar controlar chaves criptográficas que foram utilizadas para criptografar dados.) e **Certificados** (Ele oferece uma maneira simples de provisionar, gerenciar e implantar certificados TLS/SSL públicos e privados.).

1.2.7. Apache Kafka

De acordo com a RedHat (2022), o apache kafka "é uma plataforma distribuída de transmissão de dados que é capaz de publicar, subscrever,

armazenar e processar fluxos de registro em tempo real.". Portanto, tal plataforma utiliza a arquitetura orientada a eventos que adota o modelo pub/sub ou o modelo de streaming de eventos.

Segundo GeekHunter (2020), TIBCO (2021), Junior (2021), Zein (2020), Willians (2019) e Willians (2020), os seus principais componentes são: **Produtor** (Publica mensagens em um determinado tópico de um determinado cluster de brokers.), **Broker** (Servidor que contém o kafka e suas configurações.), **Consumidor** (Consome mensagens publicadas em um determinado tópico de um determinado cluster de brokers.), **Tópico** (É o nome da fila de dados que armazena as mensagens publicadas.), **Partição** (É uma fila sequencial de dados que armazena as mensagens publicadas. Cada tópico pode conter uma ou mais partições. Cada partição pertence a um dos brokers presentes no cluster de brokers. Partições são utilizadas para o balanceamento de carga ser realizado, isto é, quando uma mensagem é publicada em um tópico ela é publicada aleatoriamente em uma das partições disponíveis.) e **Réplica** (Cada réplica é um broker que está no cluster de brokers. Cada partição possui uma ou mais réplicas, ou seja, cada partição possui clones em outros brokers. Com isso, se o broker que armazena determinada partição (broker leader) ficar fora do ar, outra réplica assume a liderança e os dados presentes em tal partição continuam acessíveis.).

1.2.8. Jasper Reports

A maioria dos softwares suportam o armazenamento de dados, tais dados podem ser extremamente úteis para guiar as decisões de uma determinada empresa. Portanto, a possibilidade de gerar relatórios contendo informações relativas a uma área dela é crucial. O Jasper Report é um framework que permite que tal possibilidade seja oferecida de uma forma fácil e intuitiva (DEVMEDIA, 2023b).

De acordo com a DevMedia (2023b), para isso ocorrer, as seguintes etapas são necessárias: Criar o layout do relatório (.jrxml), ou seja, é preciso definir as colunas, as linhas, as cores, os espaços, as imagens, as variáveis etc. Tal template pode ser criado manualmente ou por meio de um framework (Ex: Jaspersoft Studio); Após o template ser criado, é preciso compilá-lo, ou seja, transformá-lo em um arquivo binário (.jasper) e populá-lo com os dados. No Spring Framework, tal etapa pode ser realizada por meio das funcões presentes na dependência net.sf.jasperreports; Por fim, é preciso definir no que o código binário (.jasper) deverá ser transformado (PDF, DOC, XML, DOC, etc), ou seja, o formato do arquivo que será gerado.

2. Implantação

2.1. Banco de Dados e Ecossistema de Microsserviços

2.1.1. Git

O git é um sistema de controle de versões distribuído. A função desse sistema é registrar qualquer alteração em um código para que caso seja necessário tal alteração possa ser desfeita (KRIGER, 2022).

De acordo com Kriger (2022), os comandos básicos do git são: **Git Clone** (Baixa um repositório remoto.), **Git Init** (Cria um repositório local.), **Git Add** (Adiciona arquivos na área de preparação.), **Git Commit** (Move os arquivos presentes na área de preparação para a área de commit (Repositório local).), **Git Remote Add** (Conecta o repositório local ao repositório remoto.), **Git Pull** (Baixa os arquivos presentes no repositório remoto para o repositório local.) e **Git Push** (Envia os arquivos presentes no repositório local para o repositório remoto.).

2.1.2. Docker

O Docker é uma plataforma de software que facilita a construção, a execução, o gerenciamento e a distribuição de aplicações (GEEKHUNTER, 2022a). Por meio dessa plataforma, a configuração do ambiente da aplicação pode ser feita com muita agilidade, já que todas as configurações necessárias podem ser inseridas em uma imagem, ou seja, em uma "classe" para que vários desenvolvedores possam utilizá-las ao criar um container virtualizado dessa imagem, ou seja, um "objeto". Com isso, os desenvolvedores não perderão mais tempo realizando inúmeras configurações antes de codificar (GUEDES, 2019).

2.1.3. Kubernetes

O Kubernetes é uma plataforma de software que orquestra sistemas em containers ao automatizá-los, implantá-los, dimensioná-los e gerenciá-los (REDHAT, 2023a).

De acordo com a RedHat (2023a), os seus principais componentes são: **Controle Plane** (É responsável por delegar tarefas aos nós.), **Node** (Realiza as tarefas que foram solicitadas pelo control pane.), **Pod** (É um ou mais containers que estão implantados em um nó.) e **Service** (Expõe os Pods ao mundo externo, ou seja, a uma rede.).

2.1.4. Microsoft Azure DevOps

A junção da área de desenvolvimento (Dev) e da área de operações (Ops) permite que ocorra uma união de pessoas, processos e tecnologias para que haja um fornecimento contínuo de valor para os clientes (MI-

CROSOFT, 2023f).

De acordo com a Microsoft (2023f), "O DevOps permite que funções anteriormente isoladas – desenvolvimento, operações de TI, engenharia da qualidade e segurança – atuem de forma coordenada e colaborativa para gerar produtos melhores e mais confiáveis. Ao adotar uma cultura de DevOps em conjunto com as práticas e ferramentas de DevOps, as equipes ganham a capacidade de responder melhor às necessidades dos clientes, aumentar a confiança nos aplicativos que constroem e cumprir as metas empresariais mais rapidamente.".

Os seus principais componentes são: **Azure Boards** (O Azure Boards se destina a ajudar as equipes a planejar, acompanhar e discutir o desenvolvimento de um software. Ele se caracteriza como uma plataforma personalizável e flexível para gerenciar itens, histórias, bugs, tarefas e problemas. Tal componente oferece suporte às metodologias ágeis (Scrum e Kanban) (MICROSOFT, 2023d).), **Azure Repos** (O Azure Repos é um sistema de controle de versão armazenado na nuvem (Mesmo conceito do GitHub) que oferece suporte ao Git e ao TFVC (Controle de Versão do Team Foundation) (MICROSOFT, 2022).) e **Azure Pipelines** (O Azure Pipelines oferece suporte à integração, entrega e implantação contínua (CI/CD), isto é, ofere suporte à realização automática de builds e deploys (MICROSOFT, 2023e).).

2.3 Trabalhos Correlatos

A temática deste TCC (Arquitetura de Microsserviços) já foi explorada em inúmeros trabalhos acadêmicos. Ao invés de selecionar vários para realizar breves resumos, escolhi apenas um para realizar um extenso resumo comparativo. Tal resumo utilizou a seguinte metodologia: 1) Em cada seção, apresente seu principal trecho, ou seja, sua principal ideia 2) Em cada seção, compare o trabalho correlato com este trabalho.

2.3.1 Arquitetura Monolítica vs Microsserviços: uma análise comparativa

O trabalho de conclusão de curso da Mendes (2021) (Ex aluna do curso Engenharia De Software da Universidade De Brasília - UNB) é semelhante a este TCC.

- Objetivo Geral

  O objetivo geral do TCC da Mendes (2021) é "Realizar uma análise comparativa acerca dos estilos arquiteturais de software monolítico e microsserviços com o intuito de compreender as características de cada estilo e em quais contextos cada

um é melhor aplicado. Para tal, fez-se uma pesquisa exploratória baseada em revisão bibliográfica e em relatos de casos reais nos quais empresas optaram por migrar de uma arquitetura monolítica para uma arquitetura de microsserviços ou vice-versa".

É possível notar que o objetivo geral da Mendes (2021) é mais abrangente do que o objetivo geral deste TCC, pois enquanto ela definiu que seu TCC iria abordar a arquitetura monolítica e a arquitetura de microsserviços, eu defini que este TCC iria se focar na arquitetura de microsserviços.

- Objetivos Específicos

  Os objetivos específicos são os seguintes "1) Realizar uma pesquisa exploratória acerca das características e os principais problemas encontrados nos estilos arquiteturais monolítico e microsserviços; 2) Traçar um paralelo entre a base teórica explorada e casos reais de empresas que vivenciaram as dificuldades de um ou ambos os estilos arquiteturais abordados; 3) Comparar as características dos estilos arquiteturais monolítico e microsseviços, elencando quais fatores devem ser analisados antes de adotar determinado estilo arquitetural para uso dentro das empresas." (MENDES, 2021).

  Nota-se que os objetivos específicos da Mendes (2021) possui um viés mais teórico, já que o primeiro objetivo é focado em elaborar um extenso referencial teórico, o segundo objetivo é focado em conectar a base teórica obtida com casos de uso reais (sem simular o desenvolvimento deles) e o terceiro objetivo é focado em realizar comparações entre as duas arquiteturas. Já este TCC possui um viés mais prático, pois sua primeira etapa focou-se em abordar os principais conceitos e as tecnologias que foram utilizadas na simulação do desenvolvimento do caso de uso real, em sua segunda etapa o foco foi realizar o levantamento de requisitos, a modelagem, a implementação e a implantação de um ecossistema de microsserviços, já sua última etapa focou-se em identificar e exemplificar as vantagens e desvantagens da arquitetura de microsserviços.

- Justificativa

  A sua existência "justifica-se por buscar esclarecer as diferenças entre ambos os estilos arquiteturais, monolítico e microsserviços, visando fornecer embasamento para equipes de desenvolvimento de software analisarem de forma mais consciente o contexto em que os projetos estão inseridos e como a escolha de determinado tipo arquitetural pode influenciar, positivamente ou negativamente, o alcance dos objetivos de negócio." (MENDES, 2021).

  Percebe-se que a existência do TCC da Mendes (2021) possui o objetivo de oferecer motivos para uma equipe de desenvolvimento adotar ou não adotar a arquitetura monolítica ou a arquitetura de microsserviços. Já a existência deste TCC

possui o objetivo de oferecer motivos para uma equipe de desenvolvimento adotar ou não adotar a arquitetura de microsserviços (vantagens e desvantagens), isto é, o foco deste TCC é apenas a arquietura de microsserviços.

- Metodologia

  A metodologia do TCC da Mendes (2021) adotou as seguintes etapas: "1) Levantamento teórico inicial acerca dos objetos de análise diante dos referências teóricos adotado; 2) Fichamento e agrupamento dos pontos de vista descobertos na Etapa 1, a fim de encontrar as perspectivas comuns e divergentes entre os autores estudados; 3) Síntese acerca dos aspectos levantados sobre cada estilo arquitetural; 4) Descrição e análise dos casos de estudo relevantes para o tema; 5) Análise final comparando os dois modelos arquiteturais com base nos referencias teóricos adotados e nos casos práticos estudados.".

  Nota-se que a metodologia do TCC da Mendes (2021) possui um viés mais teórico. Já a metodologia deste TCC possui um viés téorico (Metodologia Científica: Estudo de caso do tipo exploratório com abordagem qualitativa) e um viés prático (Metodologia de Desenvolvimento: Requisitos, Modelagem, Implementação e Implantação).

- Fundamentação Teórica

  Em sua fundamentação teórica, Mendes (2021) realizou uma extensa conceitualização sobre arquitetura de software, arquitetura monolítica e arquitetura de microsserviços.

  Ao contrátrio deste TCC que possui uma fundamentação teórica focada em descrever os principais conceitos e as tecnologias **geralmente necessárias** para a arquitetura de microsserviços ser adotada, o referencial teórico do TCC da Mendes (2021) focou-se em **explicar minunciosamente** a arquitetura monolítica, a arquitetura de microsserviços e suas diferenças.

- Análise dos casos de estudo

  Nesta seção, Mendes (2021) analisou os seguintes casos de uso: "1) KN Login: um monolítico legado transformado em uma série de sistemas autocontidos, com o intuito de caminhar em direção a uma arquitetura de microsserviços; 2) Otto: a reimplementação do zero de um monolítico em uma arquitetura de sistemas autocontidos que posteriormente é evoluída para microsserviços; 3) Segment: a transição de um monolítico para uma arquitetura de microsserviços, seguido do retorno para a arquitetura monolítica após as várias dificuldades encontradas; 4) RuaDois: a adoção de uma arquitetura de microsserviços por uma startup logo no início do sistema e a sua transição para uma arquitetura de monolítica.".

Como é possível perceber, ao contrário do que ocorre neste TCC, Mendes (2021) realizou análises de casos de uso reais sem simular seus desenvolvimentos, isto é, ela não realizou a etapa de desenvolvimento de software. Portanto, seu TCC possui um enfoque mais teórico.

- Resultados

  Nesta etapa do TCC, Mendes (2021) observou que a arquitetura monolítica possui elementos positivos que possuem uma tendência de ser tornarem negativos (Confiabilidade, rápido e fácil deploy e testabilidade se tornam menores a medida que o sistema cresce, ou seja, se torna mais complexo). Já a arquiteutra de microsserviços possui elementos positivos que para serem mantidos devem receber muita atenção dos desenvolvedores à medida que o ecossistema cresce (Alta tolerância a falhas, alta confiabilidade, fácil e rápido deploy).

  É perceptível que a Mendes (2021) apresentou como resultado do seu TCC uma categorização criterioza dos prós e contras da arquitetura monolítica e da arquitetura de microsserviços. Já este TCC, apresentou uma descrição detalhada dos prós e contras (Vantagens e desvantagens) da arquitetura de microsserviços identificados ao longo do desenvolvimento do ecossistema de microsserviços real utilizado como referência para o estudo de caso.

- Considerações finais

  Na última etapa do TCC, Mendes (2021) apresentou suas conclusões:

  "A arquitetura monolítica tende a ter um menor custo e por isso é mais recomendada para fase inicial do sistema como um meio de descobrir a aplicação, porém é uma arquitetura que tende a perder a sua manutenibilidade e evolucionabilidade a medida que a base de código cresce e por esta razão alguns autores a indicam como uma arquitetura de sacrifício, de forma que após explorar a problemática da aplicação essa arquitetura dê espaço para um modelo arquitetural mais sustentável.".

  "No caso dos microsserviços, este já tende a ser um modelo arquitetural mais sustentável, porém exige um custo maior de construção e manutenção além de domínio sobre o problema, o que faz com que nem todas as empresas estejam preparadas para adotar tal estilo arquitetural ou alcançar todos os benefícios que este modelo pode proporcionar".

  Enquanto a Mendes (2021) apresentou como consideração final do seu TCC sua opinião final sobre a arquitetura monolítica e a arquitetura de microsserviços, este TCC expôs em sua conclusão o que ele cumpriu e o que ele proporcionou e inspirou ao autor.

3 Desenvolvimento

Este capítulo possui o objetivo de apresentar os requisitos, a modelagem, a implementação e a implantação do sistema (Front-End + Back-End + Data-Base) (LARMAN; BASILI, 2003).

3.1 Requisitos

Esta seção possui o objetivo de especificar os requisitos funcionais, os requisitos não funcionais e os casos de uso do sistema (VALENTE, 2022a).

3.1.1 Requisitos Funcionais

As tabelas 1, 2, 3 e 4 possuem o propósito de apresentar os requisitos funcionais do sistema, ou seja, suas funcionalidades (VALENTE, 2022a).

Nome	Descrição
Buscar usuários ativos	Busca todos os usuários ativos Resposta: Lista{Matrícula e Nome}
Buscar dias não úteis	Busca todas os dias não úteis de um usuário em um mês Requisição: Matrícula e Data | Resposta: Lista{Data}
Salvar dia não útil	Cria e abre um dia não útil para um usuário Requisição: Matrícula e Data
Excluir dia não útil	Exclui um dia não útil de um usuário Requisição: Matrícula e Data

Tabela 1 – Requisitos Funcionais: Tela Dia Não Útil

Nome	Descrição
Buscar tarefas ativas	Busca as tarefas ativas Resposta: List{Código da Tarefa e Descrição da Tarefa}
Buscar usuários de uma tarefa	Busca os usuários ativos associados a uma tarefa ativa Request: Código da Tarefa|Resposta: List{Matrícula e Nome}
Buscar horas extras	Busca as horas extras de um mês | Request: Mês/Ano Resposta: Lista {Matrícula, Tarefa, Data e Hora Extra}
Salvar hora extra	Adiciona hora extra para um usuário em uma tarefa e dia Requisição: Matrícula, Tarefa, Data e Hora Extra
Excluir hora extra	Remove a hora extra de um usuário, tarefa e dia Requisição: Matrícula, Tarefa e Data

Tabela 2 – Requisitos Funcionais: Tela Hora Extra

Nome	Descrição
Buscar tarefas ativas de um usuário	Busca as tarefas ativas associadas a um usuário Resposta: Lista{Código da Tarefa e Descrição da Tarefa}
Buscar horas trabalhadas	Busca as horas trabalhadas salvas no dia, na semana e no mês Request: Matrícula e Data | Resposta: Horas trabalhadas
Buscar primeiro dia aberto	Busca o primeiro dia aberto com o faturamento aberto de um usuário Requisição: Matrícula e Data Resposta: Status (aberto ou fechado), Horas Trabalhadas e Timesheets { Sequencial, Código da Tarefa, Descrição da Tarefa e Horas Trabalhadas }
Buscar dia anterior/posterior	Busca um dia anterior/posterior a um dia de um usuário Requisição: Matrícula e Data Resposta: Status (aberto ou fechado), Horas Trabalhadas e Timesheets { Sequencial, Código da Tarefa, Descrição da Tarefa e Horas Trabalhadas }
Copiar dia anterior	Copia os timesheets do dia anterior a um dia de um usuário Requisição: Matrícula e Data
Fechar dia	Altera o status do dia (aberto ->fechado) Cria e abre o próximo dia útil Requisição: Matrícula e Data
Salvar timesheet	Salva um timesheet de um dia e de um usuário Requisição: Matrícula, Data, Tarefa e Horas Trabalhadas
Excluir timesheet	Exclui um timesheet de um dia e de um usuário Requisição: Sequencial, Matrícula, Data e Tarefa

Tabela 3 – Requisitos Funcionais Tela Timesheet

Nome	Descrição
Buscar usuários ativos	Busca os usuários ativos|Resposta: Lista{Matrícula, Nome}
Buscar faturamento(s)	Busca o faturamento do(s) usuário(s) em um mês Requisição: Mês e/ou Matrícula|Resposta: Lista{Matrícula, Nome, Horas Trabalhadas, Valor Hora, Valor Mês e Status Email}
Enviar email(s) do(s) faturamento(s)	Envia o(s) email(s) com o faturamento de determinado(s) usuário(s) | Requisição: Lista{Matrícula e Mês}
Gerar relatório do(s) faturamento(s)	Gera um relatório do(s) faturamento(s) em um mês Requisição: Mês | Resposta: Pdf

Tabela 4 – Requisitos Funcionais: Tela Faturamento do Consultor

3.1.2 Requisitos Não Funcionais

A tabela 5 possui o intuito de expor os requisitos não funcionais do sistema, ou seja, as restrições que os requisitos funcionais devem respeitar (VALENTE, 2022a).

Nome	Descrição
Arquitetura: BE-BE | BE-FE	Deve-se utilizar a arquitetura cliente-servidor
Arquitetura: Back-End	Deve-se utilizar a arquitetura de microsserviços
Arquitetura: Mensageria	Deve-se utilizar a arquitetura orientada a eventos
Data-Base	Deve-se utilizar um banco de dados relacional

Tabela 5 – Requisitos Não Funcionais

3.1.3 Casos de Uso

Esta subseção possui a função de determinar os principais casos de uso do sistema com base nos requisitos funcionais definidos na subseção anterior (VALENTE, 2022a).

- Tela: Timesheet
  - **Caso de uso: [01] Salvar timesheet**

    Fluxo normal/feliz:

    1. Usuário (Consultor): Acessa a url .../timesheet
    2. Front-End: Exibe as tarefas ativas do usuário logado
    3. Usuário (Consultor): Seleciona uma tarefa
    4. Front-End: Exibe o campo horas
    5. Usuário (Consultor): Determina as horas trabalhadas
    6. Usuário (Consultor): Clica no botão salvar
    7. Front-End: Solicita ao ms-timesheet que o timesheet seja salvo
    8. Back-End (ms-timesheet): Solicita ao ms-faturamento que ele verifique se o faturamento do usuário logado não foi realizado no mês do dia escolhido
    9. Back-End (ms-faturamento): Busca o faturamento do usuário logado no mês do dia em questão
    10. Back-End (ms-faturamento): Retorna o status 200 (OK)
    11. Back-End (ms-timesheet): Solicita ao ms-tarefa que ele verifique se o usuário logado está associado a tarefa presente no novo timesheet
    12. Back-End (ms-tarefa): Busca a associação entre o usuário e a tarefa
    13. Back-End (ms-tarefa): Retorna o status 200 (OK)
    14. Back-End (ms-timesheet): Verifica se o timesheet novo possui mais de 8 horas

14.1 Back-End (ms-timesheet): Se tiver: solicita ao ms-hora extra que ele verifique se o usuário logado possui horas extras suficientes no dia e na tarefa em questão

14.1.1 Back-End (ms-hora-extra): Busca as horas extras do usuário, do dia e da tarefa em questão e verifica se a quantidade é menor ou igual as horas trabalhadas - 8

14.1.2. Back-End (ms-hora-extra): Retorna o status 200 (OK)

14.2 Back-End (ms-timesheet): Se não tiver: O fluxo continua

15. Back-End (ms-timesheet): Salva o timesheet

16. Back-End (ms-timesheet): Altera o status do dia para aberto (Se estiver fechado)

17. Back-End (ms-timesheet): Atualiza as horas do dia associado ao timesheet

18. Front-End: Se o dia estiver fechado, o botão Dia Fechado (Cor Preta) se torna o botão Fechar Dia (Cor Verde)

19. Front-End: Exibe o timesheet salvo

Fluxo alternativo/triste:

9.1 Back-End (ms-faturamento): Se o faturamento do usuário logago no mês do dia em questão for encontrado, o status 406 (Não aceito) é retornado

9.2 Back-End (ms-timesheet): Retorna o status 406 (Não aceito)

9.3 Front-End: Exibe uma mensagem de erro

12.1 Back-End (ms-tarefa): Se o usuário logado não estiver associado a tarefa presente no timesheet em questão, o status 406 (Não aceito) é retornado

12.2 Back-End (ms-timesheet): Retorna o status 406 (Não aceito)

12.3 Front-End: Exibe uma mensagem de erro

14.1.1.1 Back-End (ms-hora-extra): Se o timesheet não possui horas extras suficientes, o status 406 (Não aceito) é retornado

14.1.1.2 Back-End (ms-timesheet): Retorna o status 406 (Não aceito)

14.1.1.3 Front-End: Exibe uma mensagem de erro

– **Caso de uso: [02] Fechar dia**

Fluxo normal/feliz:

1. Usuário (Consultor): Acessa a url .../timesheet

2. Front-End: Solicita e exibe o primeiro dia aberto com o faturamento aberto

3. Front-End: Exibe o botão Fechar Dia

4. Usuário (Consultor): Clica no botão Fechar Dia

5. Front-End: Solicita ao ms-timesheet que o dia escolhido seja fechado

6. Back-End (ms-timesheet): Solicita ao ms-faturamento que ele verifique se o faturamento do usuário logado não foi realizado no mês do dia escolhido

7. Back-End (ms-faturamento): Busca o faturamento salvo do usuário logado no mês do dia escolhido

8. Back-End (ms-faturamento): Retorna o status 200 (OK)

9. Back-End (ms-timesheet): Altera o status do dia para fechado (Se estiver aberto)

10. Back-End (ms-timesheet): Busca o próximo dia útil

11. Back-End (ms-timesheet): Cria e abre o próximo dia útil

12. Back-End (ms-timesheet): Retorna o status 200 (OK)

13. Front-End: O botão Fechar Dia (Cor Verde) se torna o botão Dia Fechado (Cor Preta)

14. Front-End: Solicita e exibe o primeiro dia aberto com o faturamento aberto

Fluxo alternativo/triste:

7.1 Back-End (ms-faturamento): Se o faturamento do usuário logago no mês do dia em questão for encontrado, o status 406 (Não aceito) é retornado

7.2 Back-End (ms-timesheet): Retorna o status 406 (Não aceito) e uma mensagem de erro

7.3 Front-End: Exibe uma mensagem de erro

- Tela: Dia Não Útil

  - **Caso de uso: [03] Excluir dia não útil**

    Fluxo normal/feliz:

    1. Usuário (Gestor): Acessa a url .../dia-nao-util

    2. Front-End: Após a busca dos dias não utéis, exibe a lista de dias não úteis com o botão Excluir ao lado de cada dia não útil salvo

    3. Usuário (Gestor): Clica no botão Excluir ao lado do dia não útil salvo escolhido

    4. Front-End: Solicita ao ms-timesheet que um dia não útil salvo associado a um determinado usuário seja excluído

    5. Back-End (ms-timesheet): Solicita ao ms-usuario que ele verifique se o usuário logado é um gestor

    6. Back-End (ms-usuario): Busca o usuário e verifica se ele é um gestor

    7. Back-End (ms-usuario): Retorna o status 200

    8. Back-End (ms-timesheet): Busca o dia não útil

    9. Back-End (ms-timesheet): Verifica se o dia não útil selecionado não possui horas adicionadas

    10. Back-End (ms-timesheet): Exclui o dia não útil

11. Back-End (ms-timesheet): Retorna o status 200 (OK)

12. Front-End: O botão não clicável Abrir (Cor cinza) se torna clicável (Cor verde) e o botão clicável Excluir (Cor vermelha) se torna não clicável (Cor cinza)

Fluxo alternativo/triste:

6.1 Back-End (ms-usuario): Se o usuário não for um gestor, o status 406 (Não aceito) e uma mensagem de erro são retornados

6.2 Back-End (ms-timesheet): Retorna o status 406 (Não aceito) e uma mensagem de erro

6.3 Front-End: A mensagem de erro é exibida

9.1 Back-End (ms-timesheet): Se o dia não útil selecionado possuir horas adicionadas, a exclusão é bloqueada

9.2 Back-End (ms-timesheet): Retorna o status 406 (Não aceito) e uma mensagem de erro

9.3 Front-End: A mensagem de erro é exibida

- Tela Hora Extra

  - **Caso de uso: [04] Excluir hora extra**

    Fluxo normal/feliz:

    1. Usuário (Gestor): Acessa a url .../hora-extra

    2. Front-End: Exibe a lista de horas extras adicionadas com o botão Excluir ao lado de cada uma

    3. Usuário (Gestor): Clica no botão Excluir localizado ao lado da hora extra escolhida

    4. Front-End: Solicita ao ms-hora-extra que a hora extra seja excluida

    5. Back-End (ms-hora-extra): Solicita ao ms-usuario que ele verifique se o usuário logado é gestor da tarefa associada ao timesheet

    6. Back-End (ms-tarefa): Busca o usuário e verifica se ele é gestor da tarefa associada a hora extra

    7. Back-End (ms-tarefa): Retorna o status 200

    8. Back-End (ms-hora-extra): Solicita ao ms-timesheet que o dia associada a hora extra seja buscado

    9. Back-End (ms-timesheet): Busca o dia solicitado

    10. Back-End (ms-timesheet): Retorna o dia solicitado

    11. Back-End (ms-hora-extra): Verifica se tal dia possui um timesheet associado a mesma tarefa da hora extra. Se tiver, verifica se esse timesheet não possui mais de 8 horas.

12. Back-End (ms-hora-extra): Exclui a hora extra

13. Back-End (ms-hora-extra): Retorna o status 200 (OK)

14. Front-End: Remove a hora extra da lista de horas extras

Fluxo alternativo/triste:

6.1 Back-End (ms-tarefa): Se o usuário não for gestor da tarefa associada a hora extra, o status 406 (Não aceito) e uma mensagem de erro são retornados

7.2 Back-End (ms-hora-extra): Retorna o status 406 (Não aceito) e uma mensagem de erro

7.3 Front-End: A mensagem de erro é exibida

11.2 Back-End (ms-hora-extra): Se o timesheet possui mais de 8 horas, significa que a hora extra já foi utilizada

11.3 Back-End (ms-hora-extra): Retorna o status 406 (Não aceito) e uma mensagem de erro

11.4 Front-End: A mensagem de erro é exibida

- Tela: Faturamento do Consultor

  - **Caso de uso: [05] Buscar os faturamentos**

    Fluxo normal/feliz:

    1. Usuário (Gestor): Acessa a url .../faturamento-consultor

    2. Front-End: Exibe o calendário (Mês/Ano)

    3. Front-End: Exibe o botão Buscar

    4. Usuário (Gestor): Seleciona um Mês/Ano

    5. Usuário (Gestor): Clica no botão Buscar

    6. Back-End (ms-faturamento): Socilita ao ms-faturamento os faturamentos de todos usuários ativos no mês/ano selecionado

    7. Back-End (ms-faturamento): Solicita ao ms-usuario que ele verifique se o usuário logado é um gestor

    8. Back-End (ms-usuario): Busca o usuário e verifica se ele é um gestor

    9. Back-End (ms-usuario): Retorna o status 200

    10. Back-End (ms-faturamento): Busca o valor-hora de todos os usuários ativos no mês/ano selecionado

    11. Back-End (ms-faturamento): Calcula o faturamento de todos os usuários ativos no mês/ano selecionado (Faturamento->Horas Trabalhadas no mês/ano selecionado x Valor Hora no mês/ano selecionado)

    12. Back-End (ms-faturamento): Retorna o faturamento de todos os usuários ativos no mês/ano selecionado

13. Front-End: Exibe o faturamentos de todos os usuários ativos no mês/ano selecionado

Fluxo alternativo/triste:

8.1 Back-End (ms-usuario): Se o usuário não for um gestor, o status 406 (Não aceito) e uma mensagem de erro são retornados

8.2 Back-End (ms-faturamento): Retorna o status 406 (Não aceito) e uma mensagem de erro

8.3 Front-End: A mensagem de erro é exibida

– **Caso de uso: [06] Enviar faturamento(s)**

Fluxo normal/feliz:

1. Usuário (Gestor): Acessa a url .../faturamento-consultor

2. Front-End: Após a busca do(s) faturamento(s), o botão Enviar Email é exibido

3. Usuário (Gestor): Clica no botão Enviar Email

4. Front-End: Solicita ao ms-faturamento que o(s) faturamento(s) seja(m) enviado(s)

5. Back-End (ms-faturamento): Busca o(s) faturamento(s) do usuários(s) no mês/ano selecionado

6. Back-End (ms-faturamento): Solicita ao ms-email-producer que o(s) email(s) seja(m) enviado(s)

7. Back-End (ms-email-producer): Publica a(s) mensagem(ns)(Email(s)) em um tópico do kafka

8. Back-End (ms-email-consumer): Se inscreve em um tópico do kafka e lê a(s) mensagem(ns) publicada(s) (Email(s)) e envia os email(s)

9. Back-End (ms-faturamento): Salva o(s) faturamento(s) que foram enviados

10. Back-End (ms-faturamento): Retorna o status 200 (OK)

11. Front-End: Altera o ícone presente na coluna Email (Vermelho -> Verde)

– **Caso de uso: [07] Gerar relatório do(s) faturamento(s)**

Fluxo normal/feliz:

1. Usuário (Gestor): Acessa a url .../faturamento-consultor

2. Front-End: Após a busca do(s) faturamento(s), o botão Gerar Relatório é exibido

3. Usuário (Gestor): Clica no botão Gerar Relatório

4. Front-End: Solicita ao ms-relatorio que o relatório (PDF) do (s) faturamento (s) seja (m) gerado(s)

5. Back-End (ms-relatorio): Solicita ao ms-faturamento o(s) faturamento(s) do(s) usuario(s) no mês/ano selecionado

6. Back-End (ms-faturamento): Calcula o(s) faturamento(s) do usuários(s) no mês/ano selecionado

7. Back-end (ms-faturamento): Retorna os faturamento(s) do usuário(s) no mês/ano selecionado

8. Back-End (ms-relatorio): Insere os dados do(s) faturamento(s) em um template do jasper e o transforma em um pdf

9. Back-End (ms-relatorio): Retorna o relatório (PDF) do(s) faturamento(s)

10. Front-End: Realiza o download do pdf

3.2 Modelagem

Esta seção possui o propósito de converter os requisitos definidos anteriormente em modelos, isto é, diagramas (DEVMEDIA, 2011).

3.2.1 Diagramas de Caso de Uso

As figuras 2, 3, 4, 5 e 6 possuem a responsabilidade de representar graficamente os principais casos de uso do sistema definidos anteriormente (VALENTE, 2022a).

- Tela: Timesheet

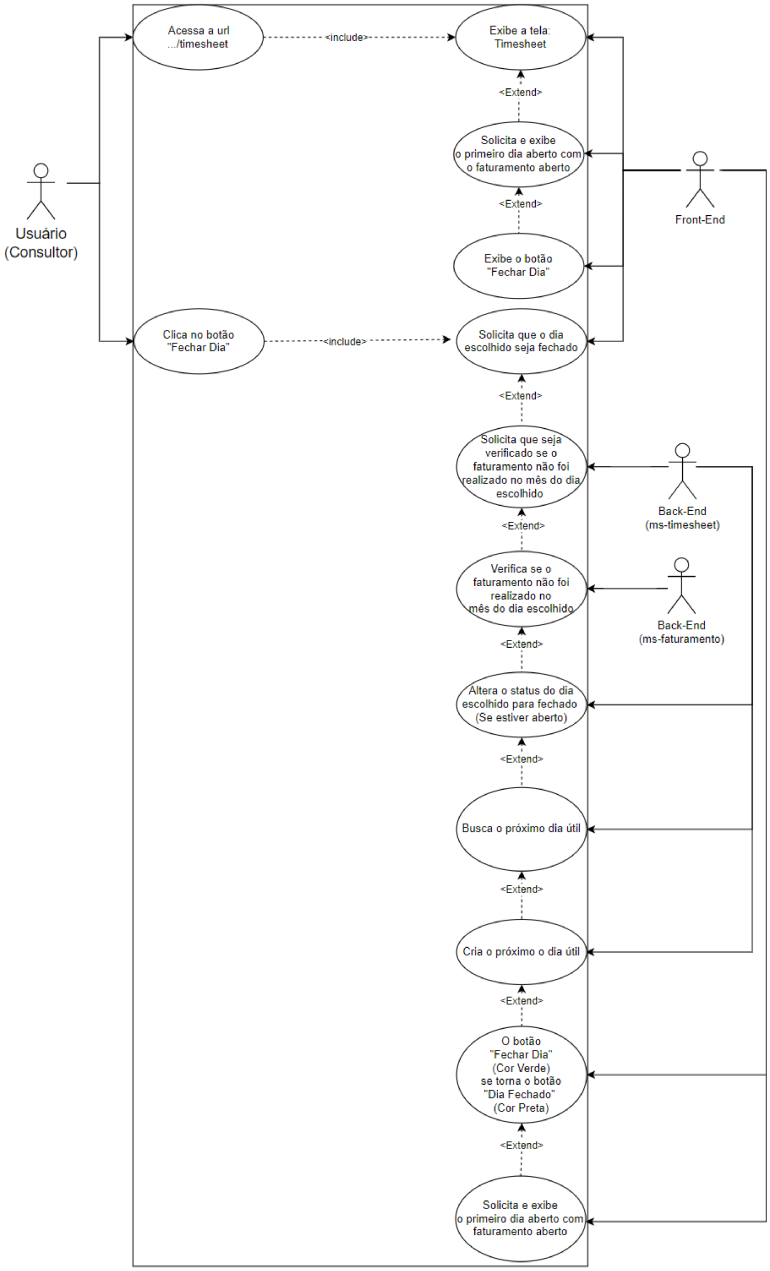


Figura 2 – Caso de Uso: [01] Fechar Dia

- Tela: Dia Não Útil

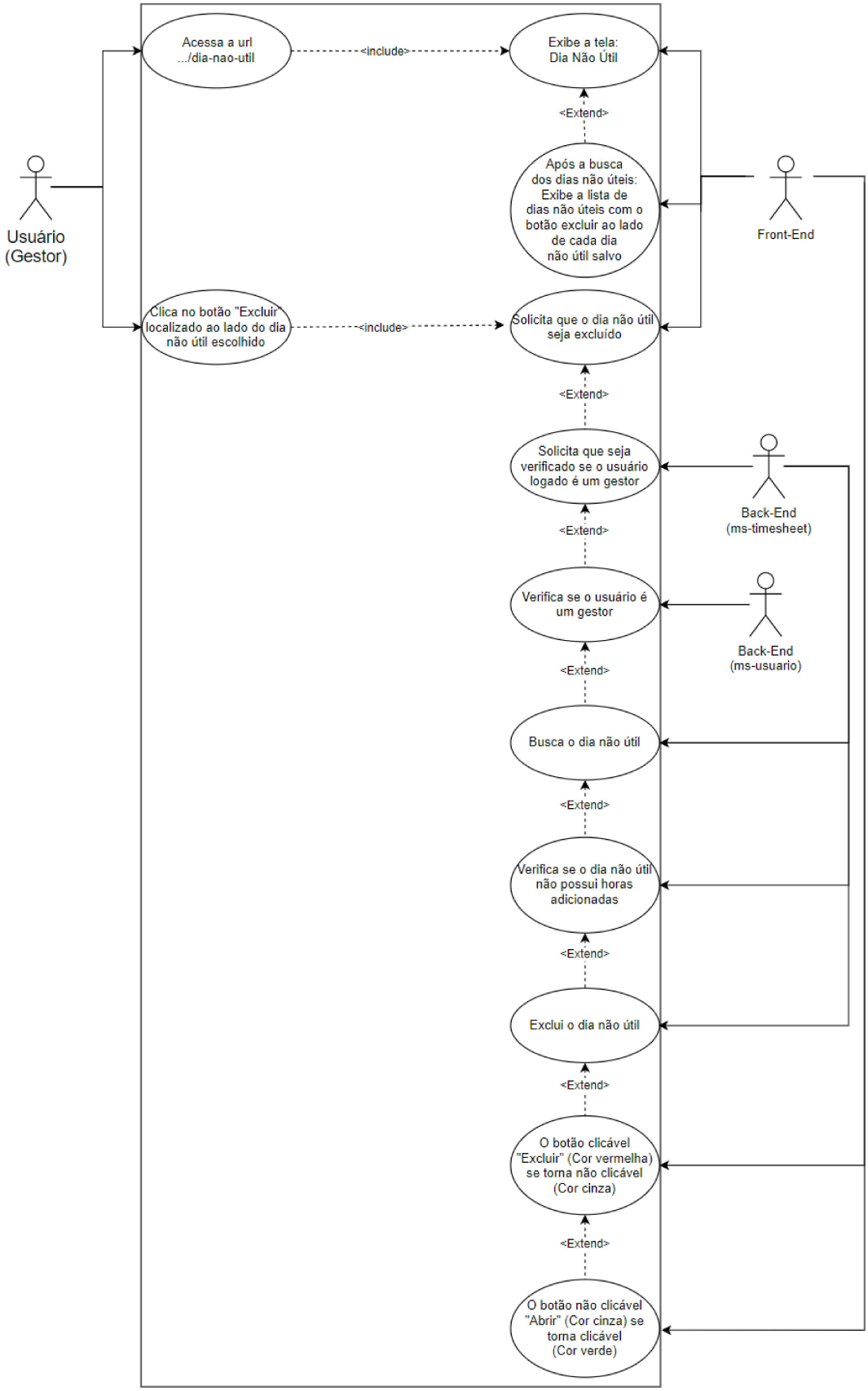


Figura 3 – Caso de Uso: [02] Excluir Dia Não Útil

- Tela: Hora Extra

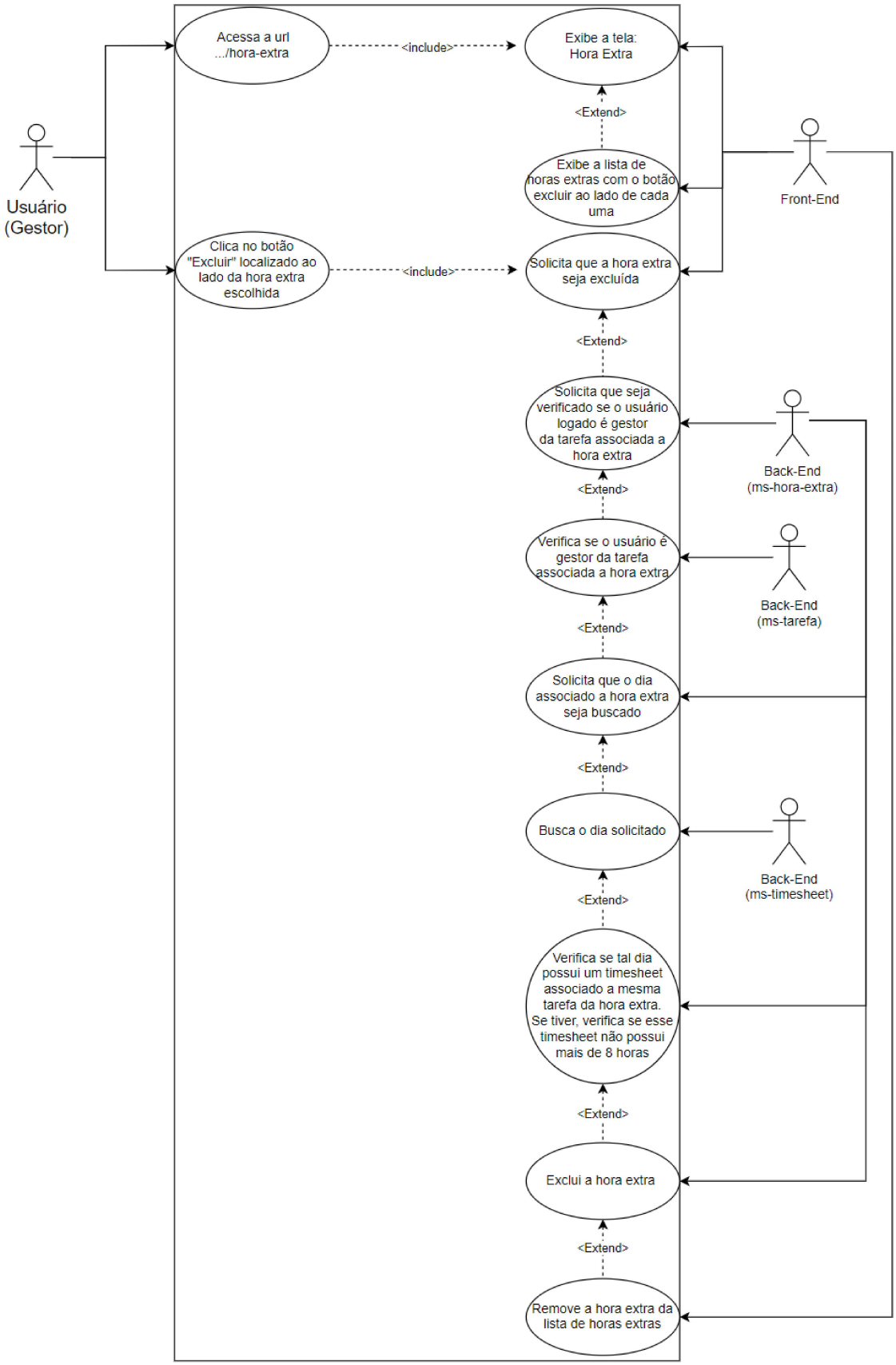


Figura 4 – Caso de Uso: [03] Excluir Hora Extra

- Tela: Faturamento do Consultor

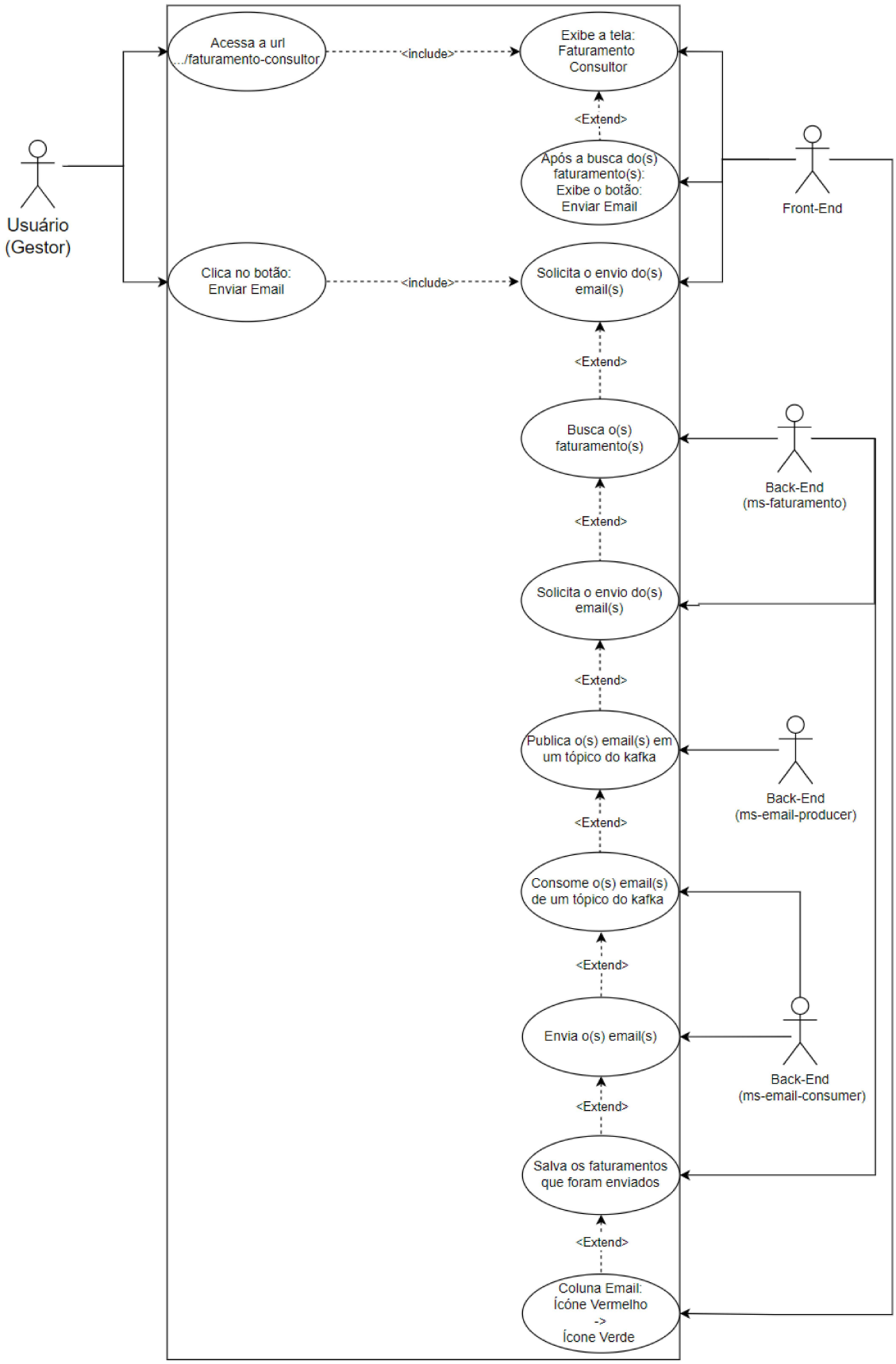


Figura 5 – Caso de Uso: [04] Enviar faturamento(s)

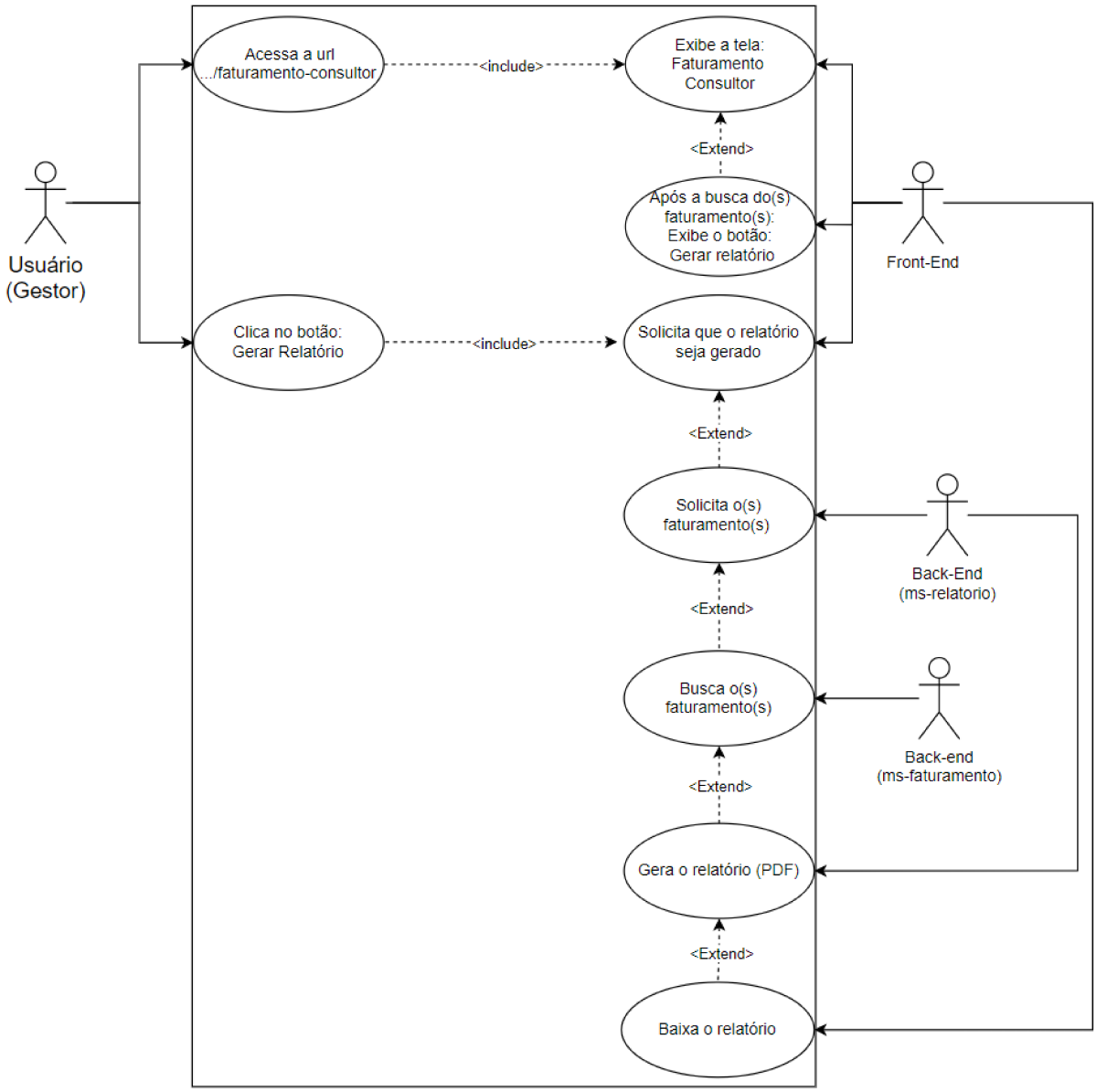
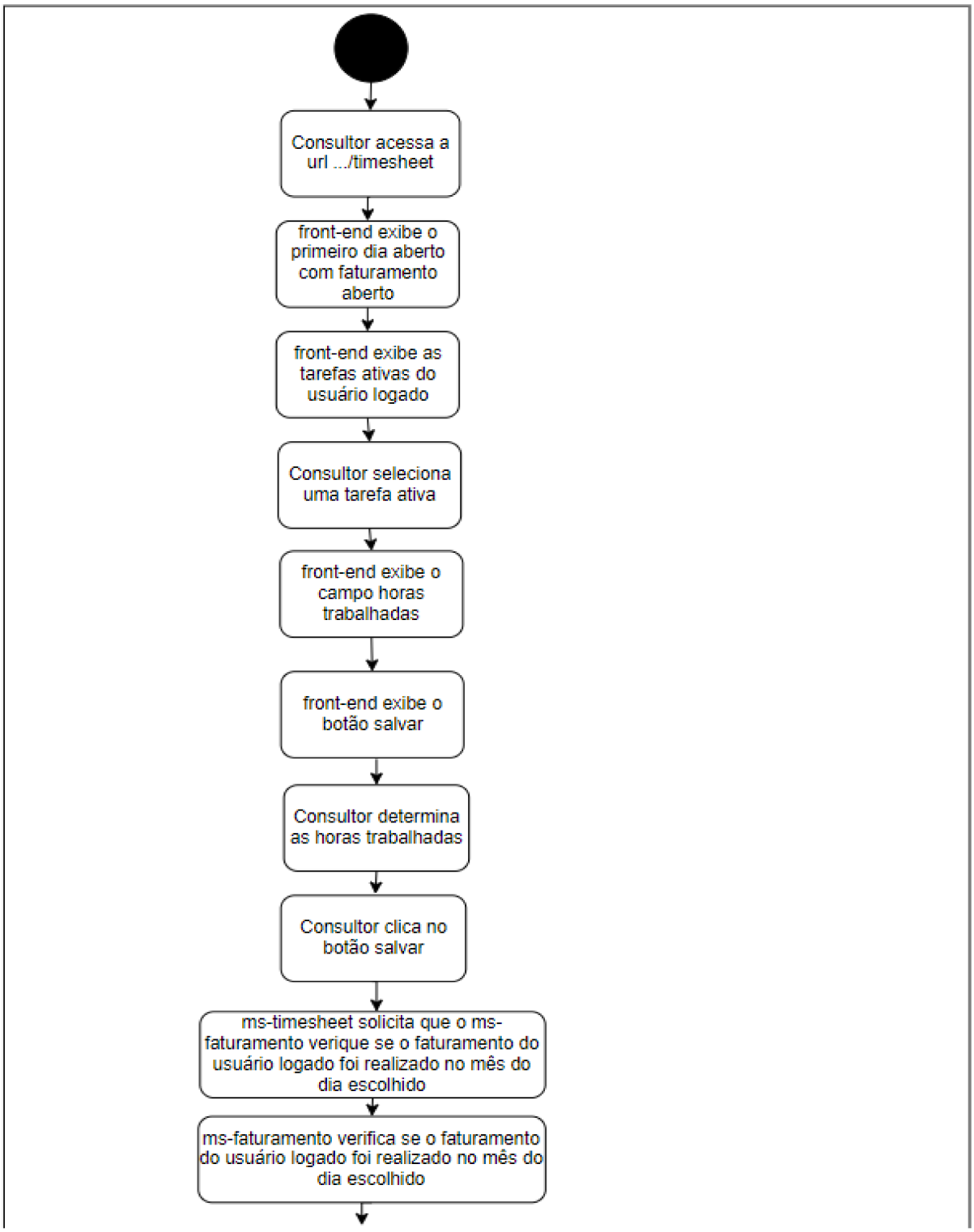


Figura 6 – Caso de Uso: [05] Gerar relatório do(s) faturamento(s)

3.2.2 Diagramas de Atividade

A figura 7 possui o objetivo de detalhar o fluxo do principal caso de uso descrito na subseção anterior por meio de uma representação visual detalhada (LUCIDCHART, 2023).

- Tela: Timesheet



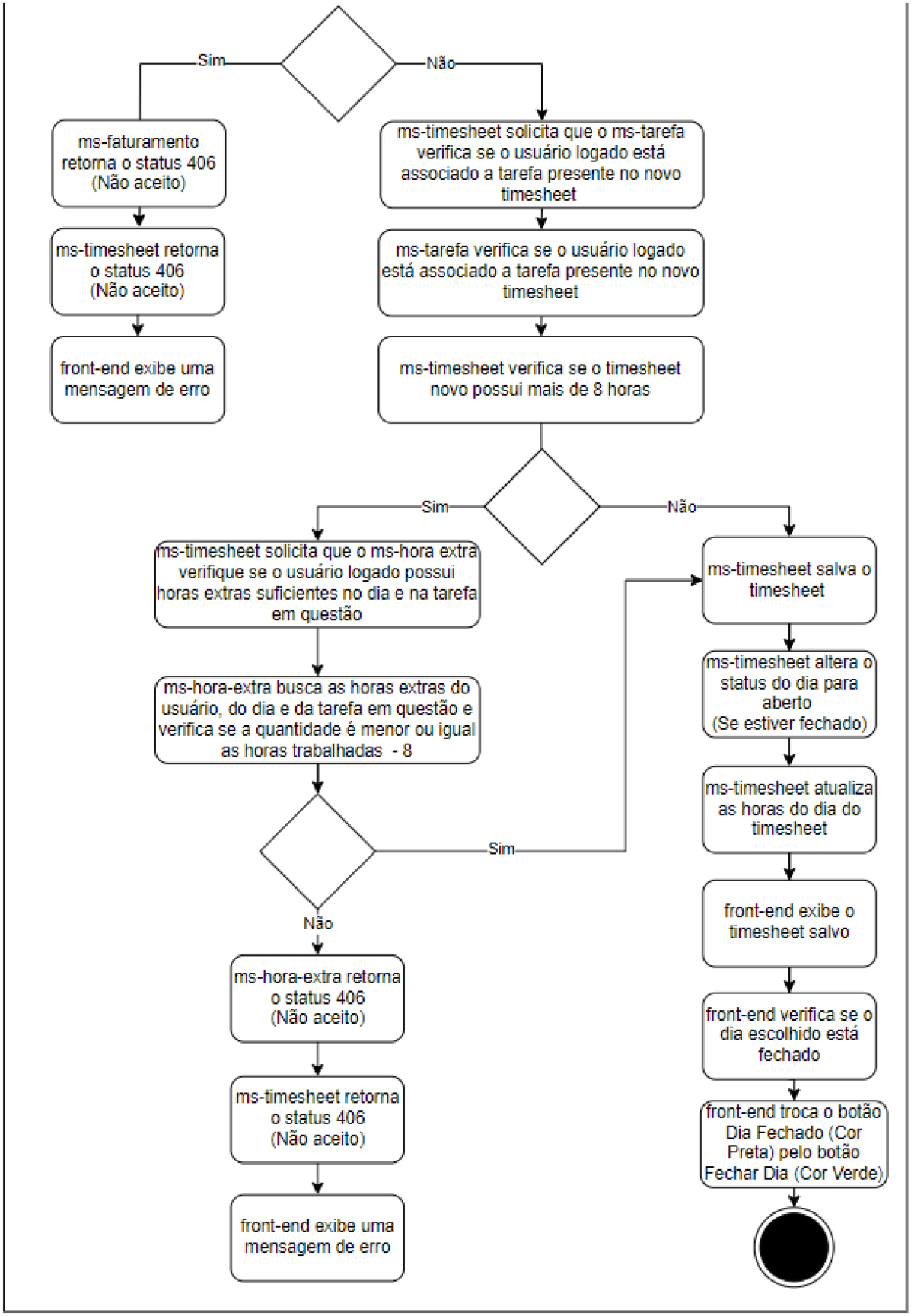


Figura 7 – Diagrama de Atividade [01]: Salvar Timesheet

3.2.3 Diagramas de Arquitetura

Esta subseção possui a finalidade de representar visualmente os componentes pertencentes ao sistema, isto é, sua arquitetura (AMAZON, 2023a).

3.2.3.1 Arquitetura em Camadas

A figura 8 possui o propósito de expor a arquitetura em três camadas do sistema (Front-End (Telas) + Back-End (Microsserviços) + Data-Base (Tabelas)) (VALENTE, 2022b).

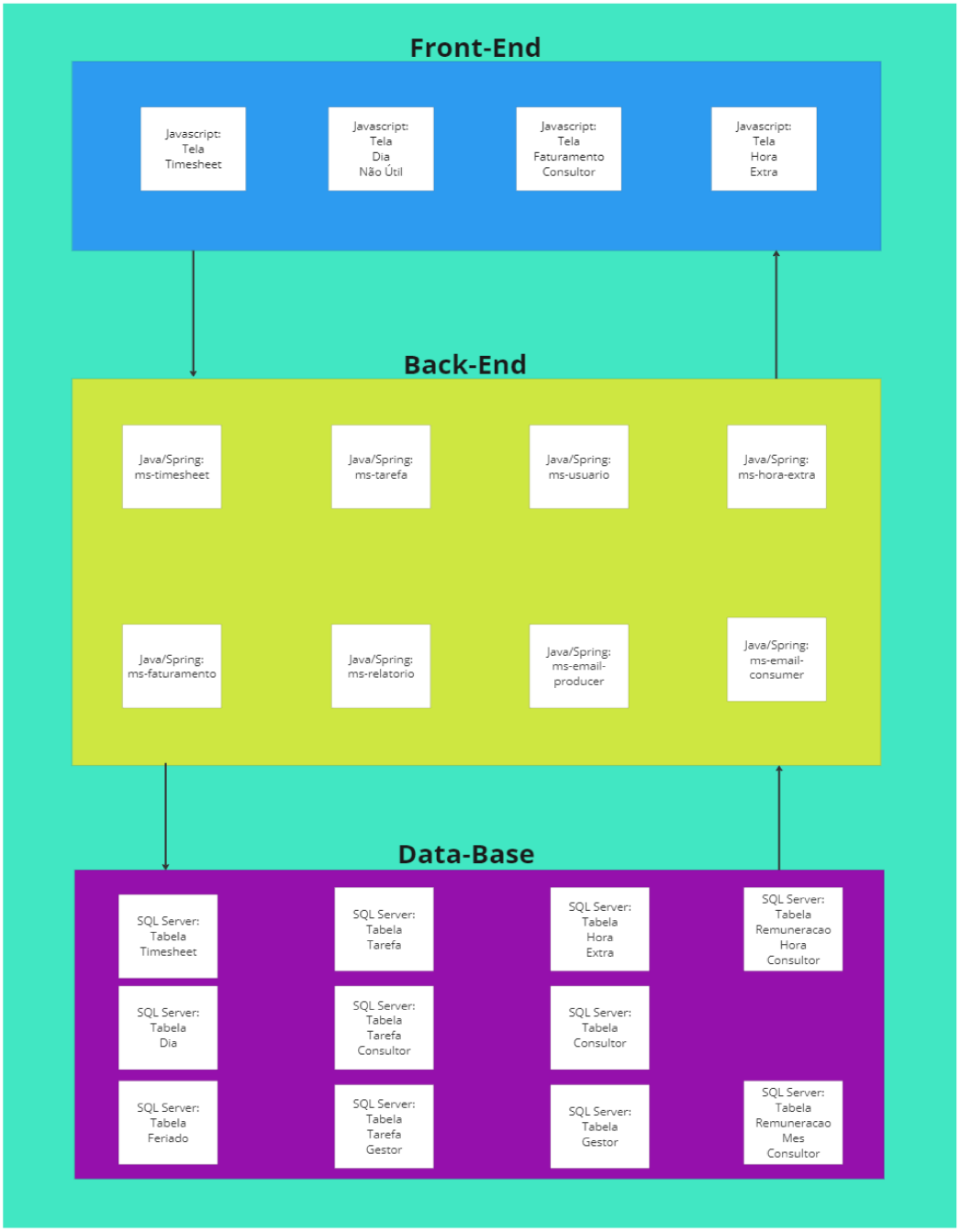


Figura 8 – Diagrama de Arquitetura [01]: Arquitetura em Camadas

3.2.3.2 Arquitetura Cliente-Servidor

A figura 9 possui a função de expor as duas primeiras camadas do sistema: Front-End e Back-End, isto é, os clientes (Telas) e seus respectivos servidores (Microsserviços) (NATIVE, 2022).

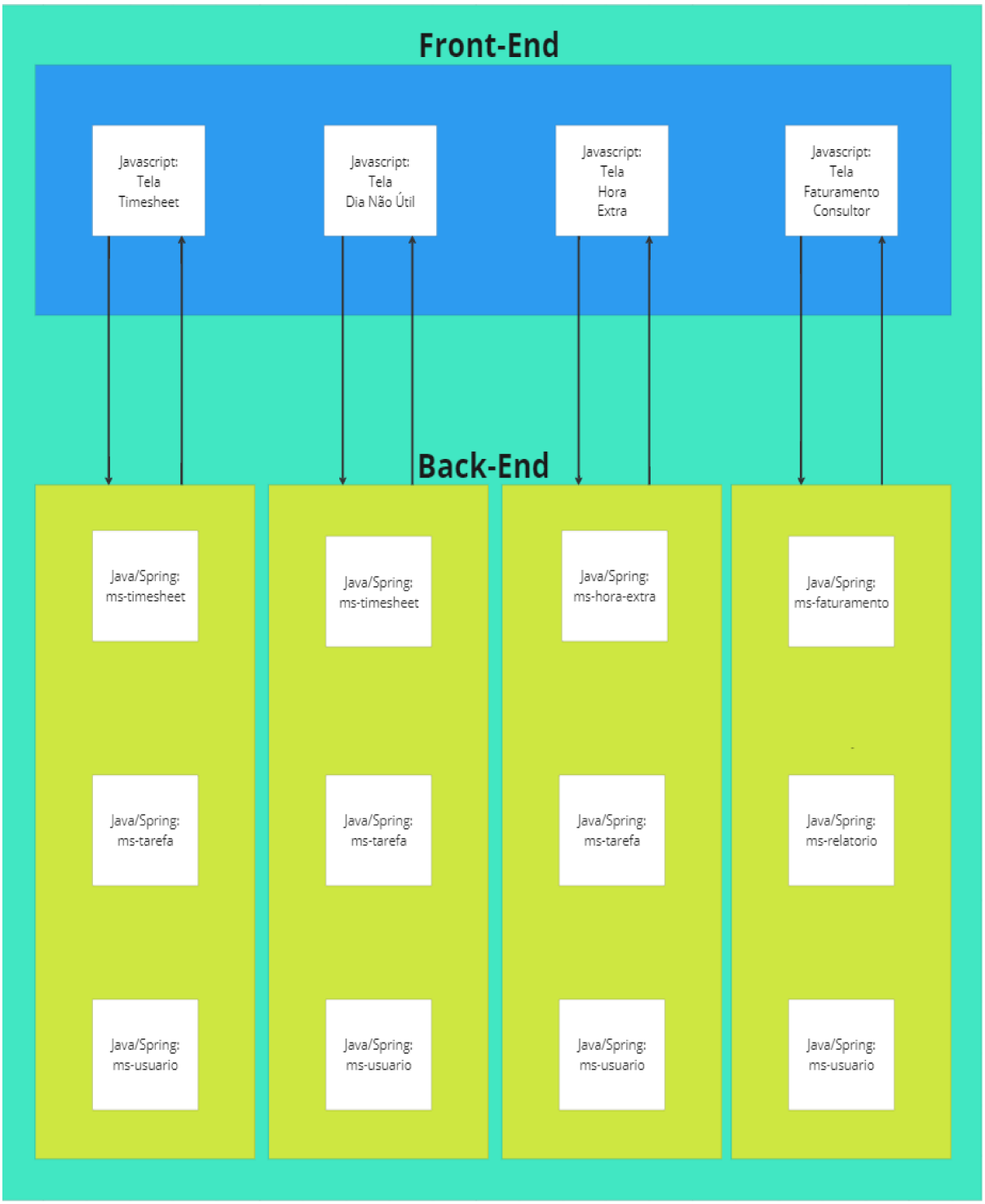


Figura 9 – Diagrama de Arquitetura [02]: Arquitetura Cliente-Servidor

3.2.3.3 Arquitetura de Microsserviços

A figura 10 possui o intuito de apresentar as conexões que ocorrem entre os microsserviços, ou seja, em como vários sistemas individuais se conectam até se tornarem um só, ou seja, um ecossistema de microsserviços (NEWMAN, 2020b) (NEWMAN, 2022).

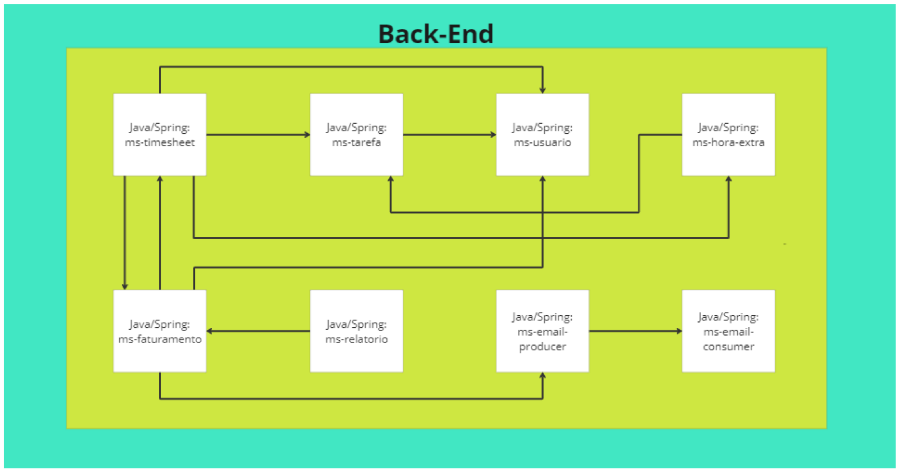


Figura 10 – Diagrama de Arquitetura [03]:Arquitetura de Microsserviços

3.2.3.4 Arquitetura de Dados

A figura 11 possui a função de expor as duas últimas camadas do sistema: Back-End e Data-Base, ou seja, os servidores (Microsserviços) e suas respectivas tabelas do banco de dados (VALENTE, 2022b).

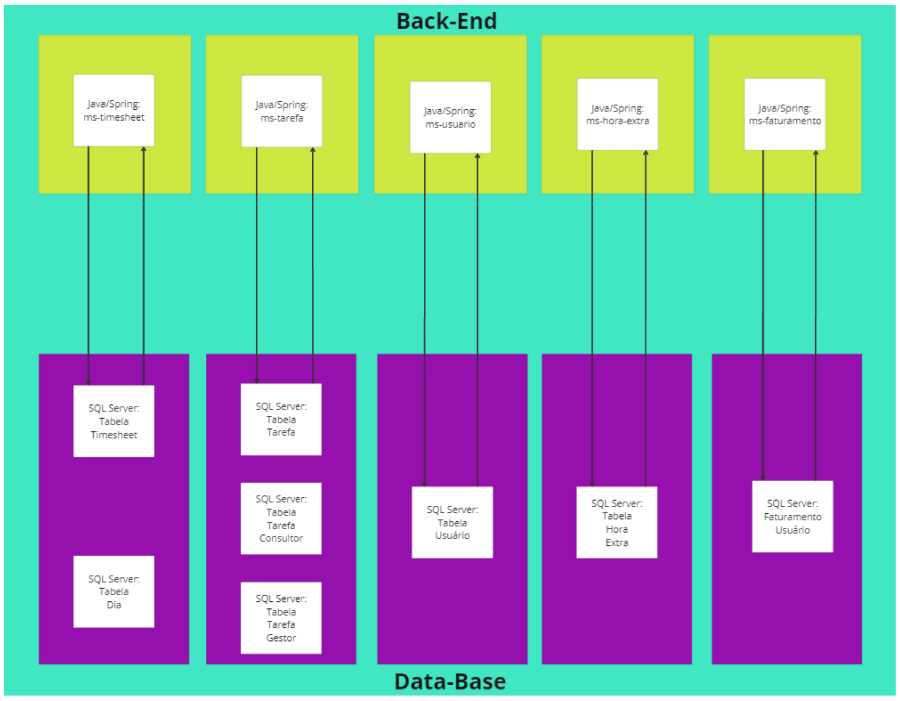


Figura 11 – Diagrama de Arquitetura [04]:Arquitetura de Dados

3.2.3.5 Arquitetura Orientada a Eventos

A figura 12 possui o objetivo de apresentar os componentes de tal arquitetura que foi aplicada ao contexto do envio de um e-mail. Componentes: o requisitante (Front-End), o intermediador (ms-faturamento), o produtor (ms-email-producer), o canal (Tópico: Email), o consumidor (ms-email-consumer) e por fim o serviço que envia o e-mail (Spring Email) (TIBCO, 2023) (REDHAT, 2019b) (MICROSOFT, 2023c).

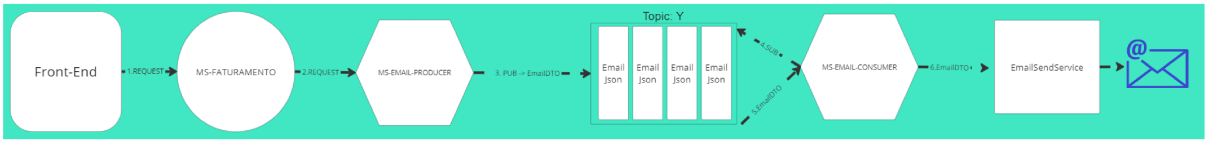
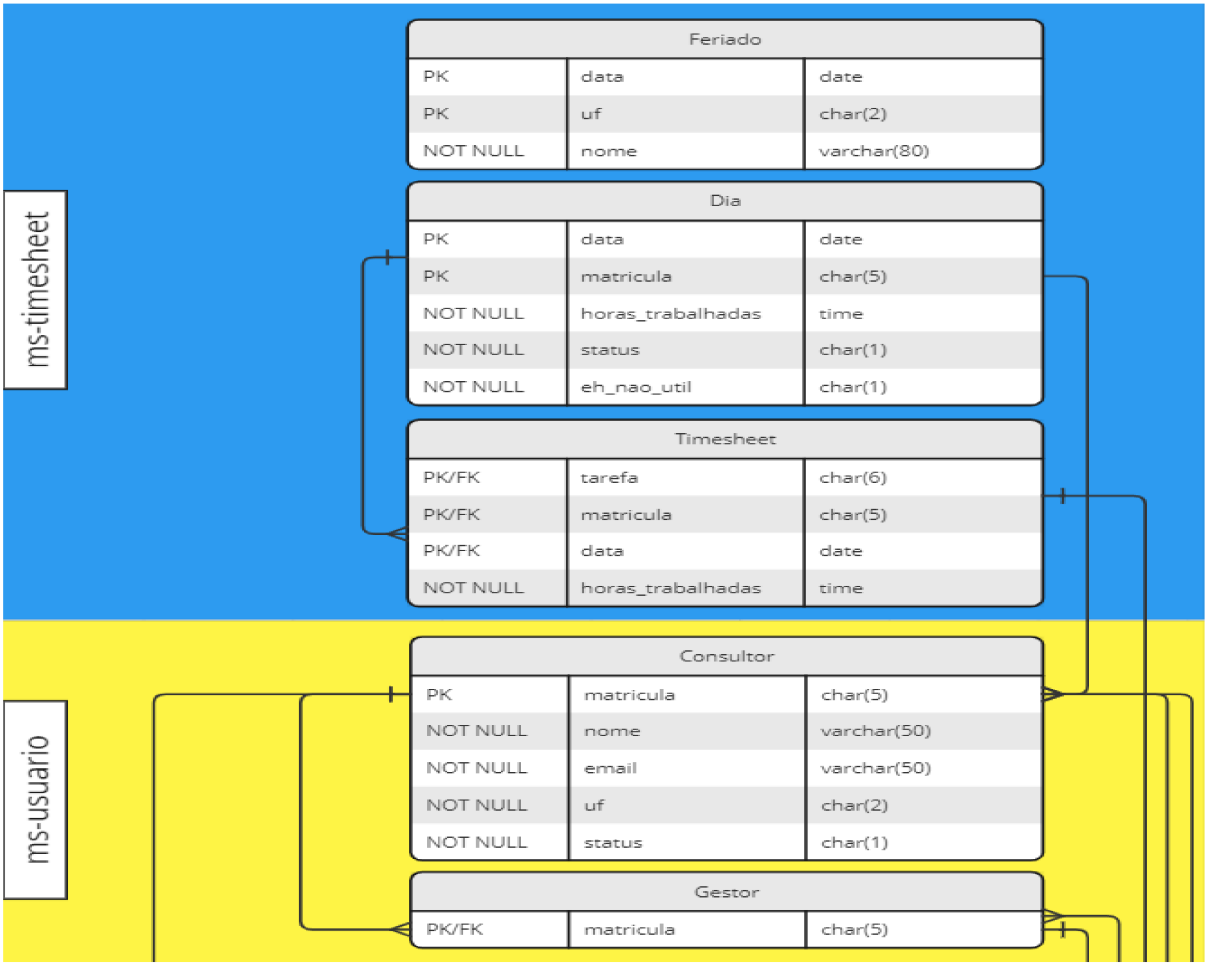


Figura 12 – Diagrama de Arquitetura [05]: Arquitetura Orientada a Eventos

3.2.4 Diagrama Entidade Relacionamento (DER)

A figura 13 possui o intuito de representar a estrutura do banco de dados relacional utilizado no sistema, ou seja, suas entidades (Tabelas), atributos (Colunas) e relacionamentos (1..1, 1..n, n..n) (JOEL, 2014).



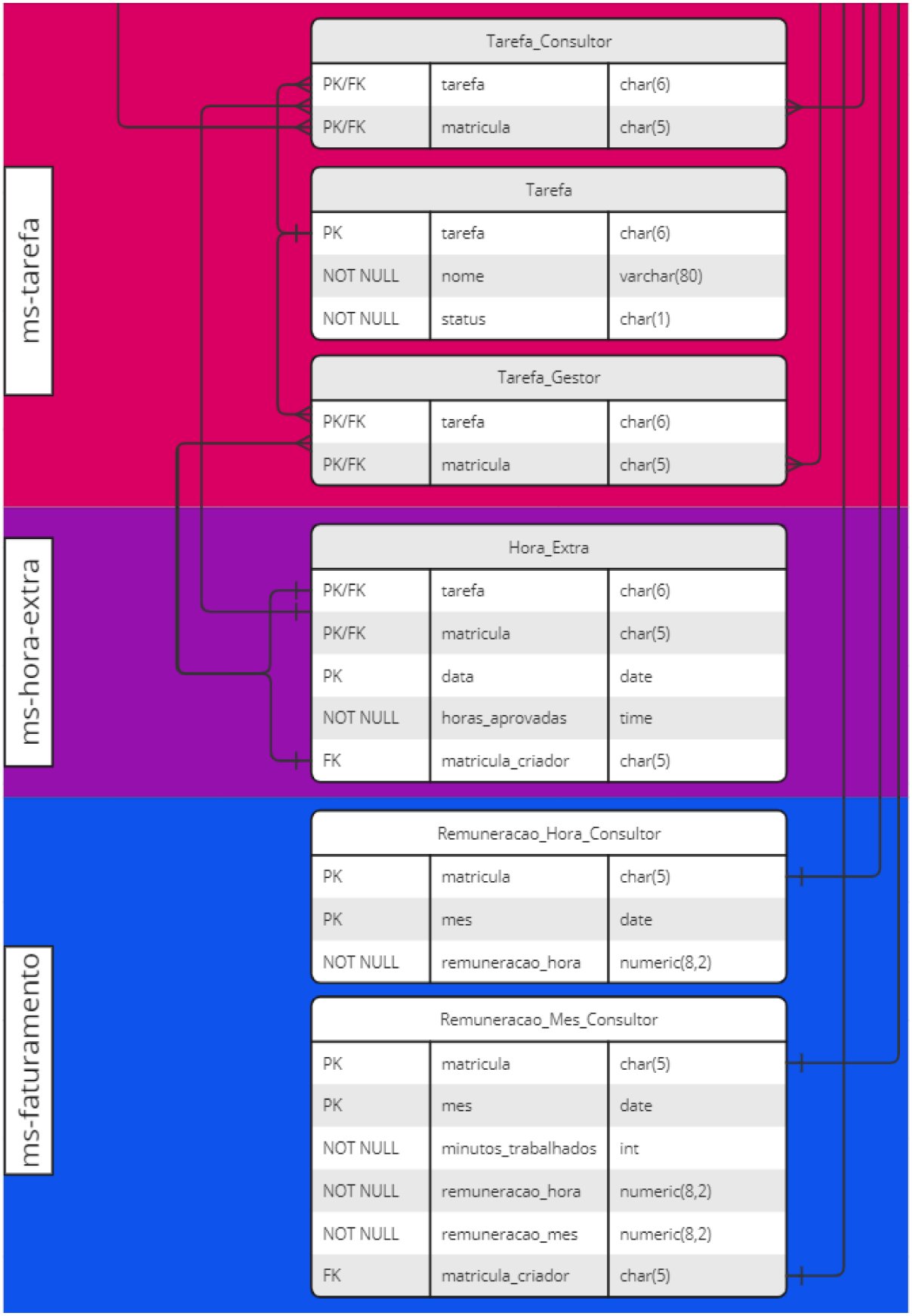


Figura 13 – Diagrama de Arquitetura [06]: Diagrama Entidade Relacionamento

3.3 Implementação

Esta seção possui o intuito de apresentar a implementação do sistema (Ato de se converter a teoria em prática por meio de linguagens de programação (HUDSON, 2007).). Tal implementação se baseou na modelagem e nos requisitos definidos anteriormente.

3.3.1 Ecossistema de Microsserviços (Back-End)

Esta subseção possui a responsabilidade de apresentar os padrões de Arquitetura (Architectural Pattern) e de Projeto (Design Pattern) utilizados em cada microsserviço.

Cada microsserviço utiliza o padrão arquitetural MVC (Model, View e Controller) em conjunto com os padrões de projeto Data Transfer Object (DTO), Service e Repository.

No padrão arquitetural MVC, o Model é responsável pela manipulação dos dados e pela implementação das regras de negócios; a View é responsável por definir os dados que devem ser exibidos; o Controller é responsável por intermediar a conexão entre o Usuário e o Model, ou seja, ele recebe solitações do Usuário, as direciona para o Model, recebe os dados do Model e retorna os dados em uma View (MOZILLA, 2023a).

Um padrão arquitetural se define como uma estrutura genérica (Alto Nível). Portanto, para adotá-lo não é necessário desenvolver o código do sistema diretamente em seus componentes. Para adotá-lo corretamente, é preciso utilizá-lo como um modelo para a implementação do sistema. Tal implementação pode utilizar padrões de projeto para decompor um ou mais componentes do padrão arquitetural em questão. Com isso, os padrões de projeto se definem como uma estrutura específica (Baixo Nível) que aumentam ainda mais a legibilidade, a manutenção e a reutilização do código (GUPTA; SAHU, 2023).

No padrão de projeto Data Transfer Object (DTO), cada objeto de transferência de dados assume a responsabilidade do componente View. Com isso, ele se torna responsável pela definição dos dados que serão transportados para outro sistema, ou seja, ele determina os dados que outro sistema poderá visualizar (FOWLER, 2002b).

No padrão de projeto Service, cada serviço assume a primeira responsabilidade do compontente Model, ou seja, ele se torna responsável pela implementação das regras de negócios (FOWLER, 2002c).

No padrão de projeto Repository, cada repositório assume a segunda responsabilidade do componente Model, ou seja, ele se torna responsável pelo acesso ao banco de dados ao se associar a uma model/entidade (Objeto que representa uma tabela) e ao se definir como uma interface que extende um mapeador de dados (Data Mapper) (Implementa os principais comandos sql: Select, Insert, Update e Delete) (FOWLER, 2002d) (FOWLER, 2002a).

Um padrão de projeto não surge artificalmente, ele é criado naturalmente ao longo do desenvolvimento de um sistema para facilitar a continuidade de seu desenvolvimento (TEDESCO, 2017). Apesar de inicialmente os microsserviços terem se baseado somente nos padrões citados anteriormente, eles não conseguiram evitar que o código se tornasse cada vez mais desorganizado a medida que novas funcionalidades eram criadas. Com isso, devido a necessidade de deixar o código mais limpo, padrões de projeto inerentes a esse ecossistema de microsserviço foram criados (Ex: DtoService, ModelService e UtilService).

As figuras 14, 15, 16, 17 e 18 apresentam a estrutura da codificação de cada microsserviço:

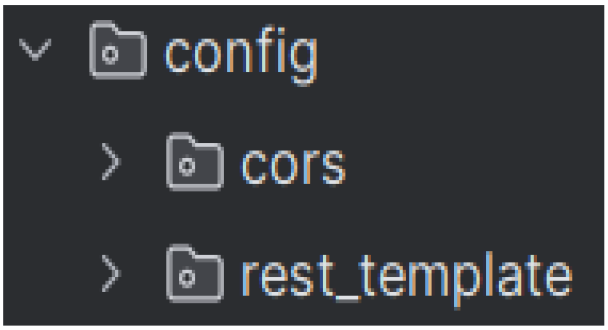


Figura 14 – Implementação [1]: Estrutura -> Config

Figura 15 – Implementação [2]: Estrutura -> Controller

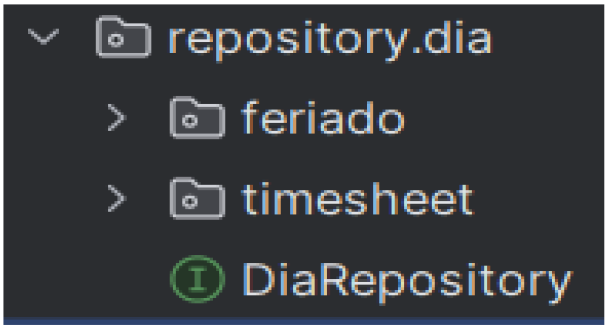


Figura 16 – Implementação [3]: Estrutura -> Repository

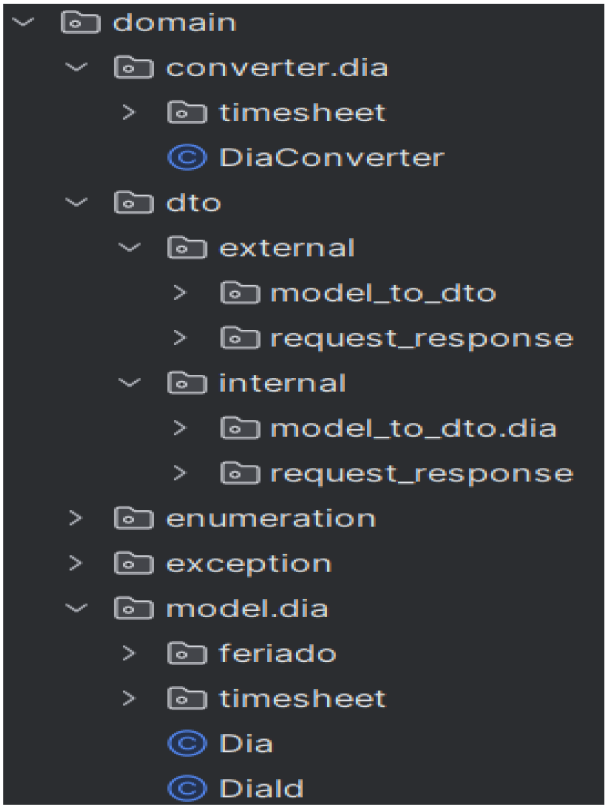


Figura 17 – Implementação [4]: Estrutura -> Domain

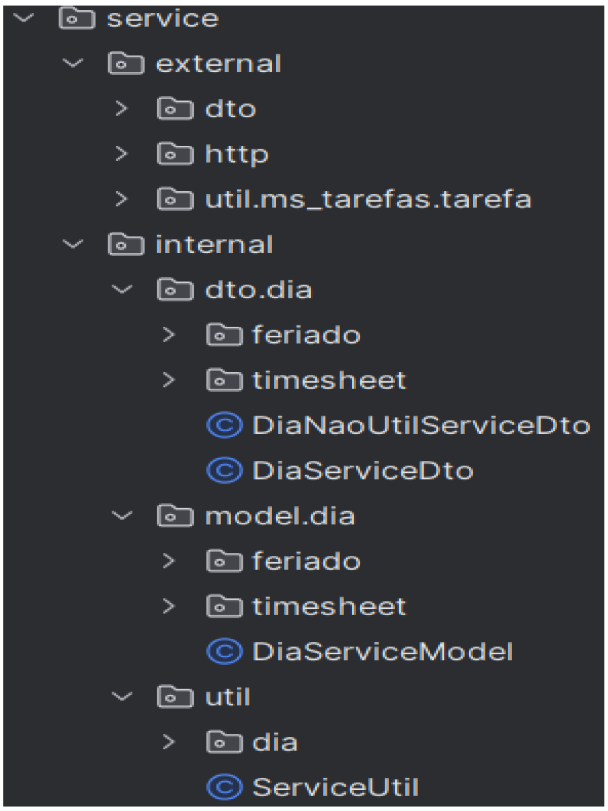


Figura 18 – Implementação [5]: Estrutura -> Service

3.3.2 Sistema Web (Front-End)

Esta subseção é responsável por apresentar as telas do sistema web e suas principais funcionalidades.

- Tela: Timesheet

  As figuras 19, 20, 21 e 22 expõem tal tela.

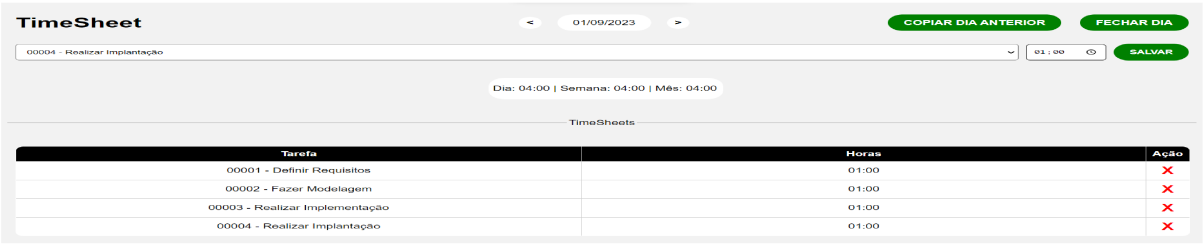


Figura 19 – Tela Timesheet [01]: Salvar Timesheet

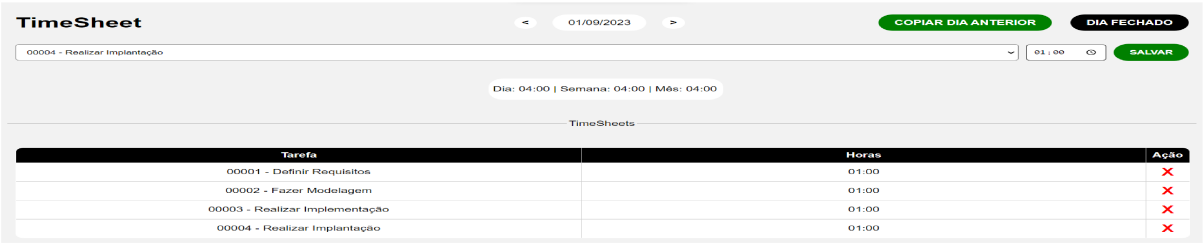


Figura 20 – Tela Timesheet [02]: Fechar o dia

Figura 21 – Tela Timesheet [03]: Copiar dia anterior

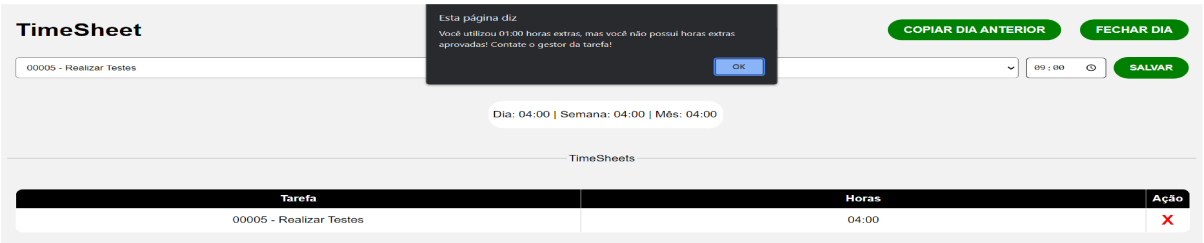


Figura 22 – Tela Timesheet [04]: Bloquear salvamento do timesheet

- Tela: Dia Não Útil

  As figuras 23 e 24 expõem tal tela.

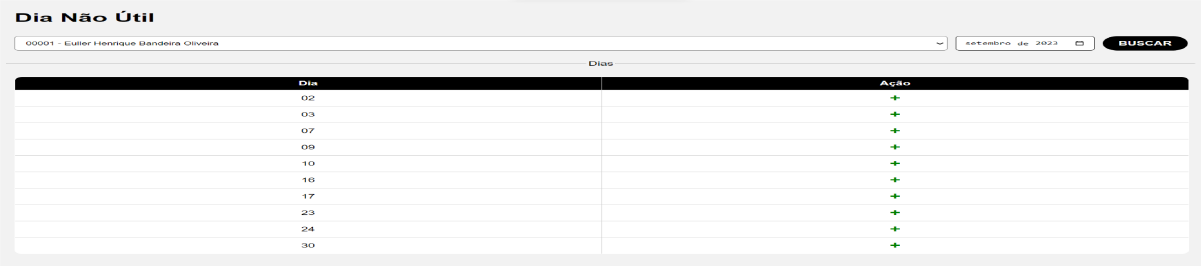


Figura 23 – Tela Dia Não Útil [01]: Salvar dia não útil

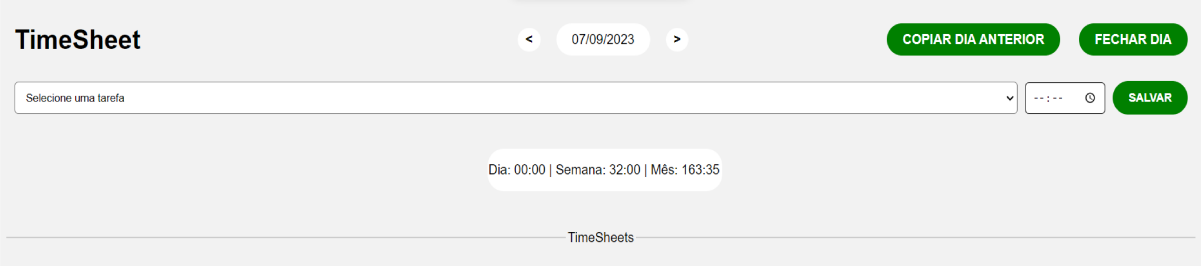


Figura 24 – Tela Dia Não Útil [02]: Salvar dia não útil -> Exibir Dia não útil

- Tela: Hora Extra

  As figuras 25 e 26 expõem tal tela.

Figura 25 – Tela Hora Extra [01]: Salvar Hora Extra

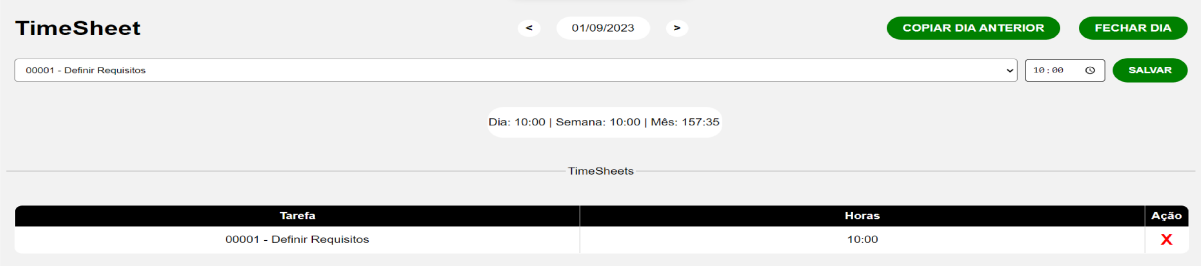


Figura 26 – Tela Hora Extra [02]: Salvar Hora Extra -> Usar Hora Extra

- Tela: Faturamento do Consultor

  As figuras 27, 28, 29 e 30 expõem tal tela.

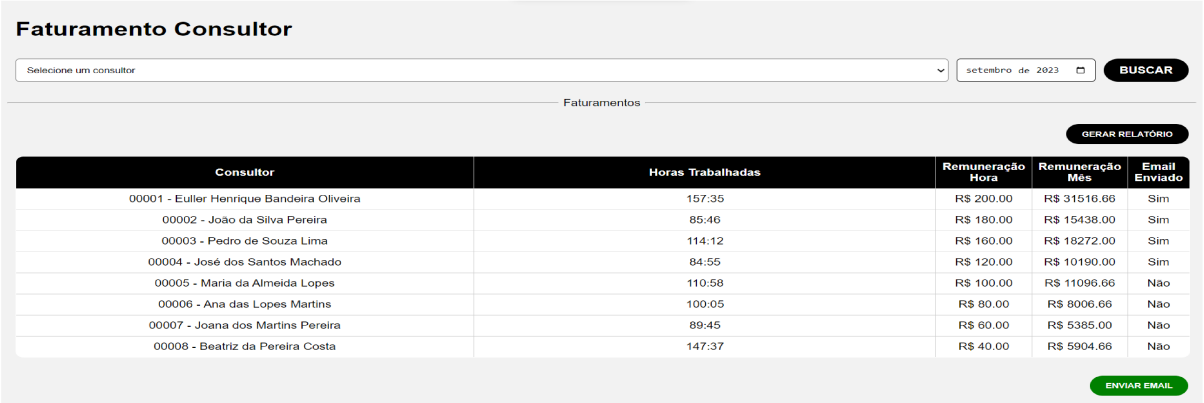


Figura 27 – Tela Faturamento do Consultor [01]: Buscar Faturamentos

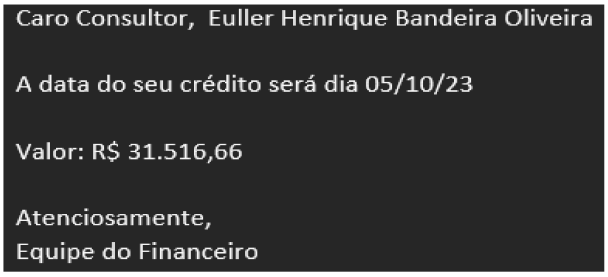


Figura 28 – Tela Faturamento do Consultor [02]: Enviar Email-> Email

Figura 29 – Tela Faturamento do Consultor [03]: Gerar relatório -> Relatório

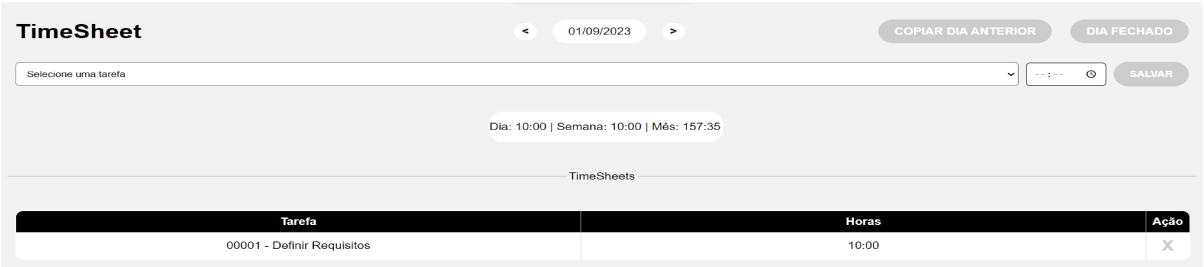


Figura 30 – Tela Faturamento do Consultor [04]: Enviar Email -> Mês Faturado

3.4 Implantação

Esta seção possuia o propósito de inserir o sistema (Back-End, Front-End e Data-Base) em um servidor online.

- Todavia, devido:

    1. Ao prazo curto para a finalização desta monografia.
    2. Ao custo de se implantar um servidor para cada microsserviço (Java/Apache Tomcat), um servidor para o banco de dados (SQL Server), um servidor de mensageria (Apache Kafka), um servidor de email (SMPT) e de se configurar o Docker, o Kubernetes, o Azure Key Vault e o Azure Pipelines (CI/CD) no Microsoft Azure.
    3. Ao fato de que ainda não possuo o conhecimento necessário para realizar todas essas configurações e implantações sozinho.

- Os seguintes itens foram implantados apenas localmente por meio do docker:

    1. Servidor para cada microsserviço (Java/Apache TomCat).
    2. Servidor de mensageria (Apache Kafka).
    3. Dashboard do Kafka (Kafka Drop). [Figura 31]
    4. Servidor de banco de dados (PostgreSQL).
    5. Servidor de email (MailHog). [Figura 32]

Portanto, esta seção pode ser considerada a seção "Trabalhos Futuros" deste TCC, pois pretendo me aprofundar no desenvolvimento DevOps após minha formatura ao fazer cursos de Docker (Conhecimento atual: Intermediário), Kubernetes (Conhecimento atual: Básico), Microsoft Azure (Conhecimento atual: Básico) e Amazon AWS (Conhecimento atual: Nenhum) para eventualmente conseguir realizar tal implantação.

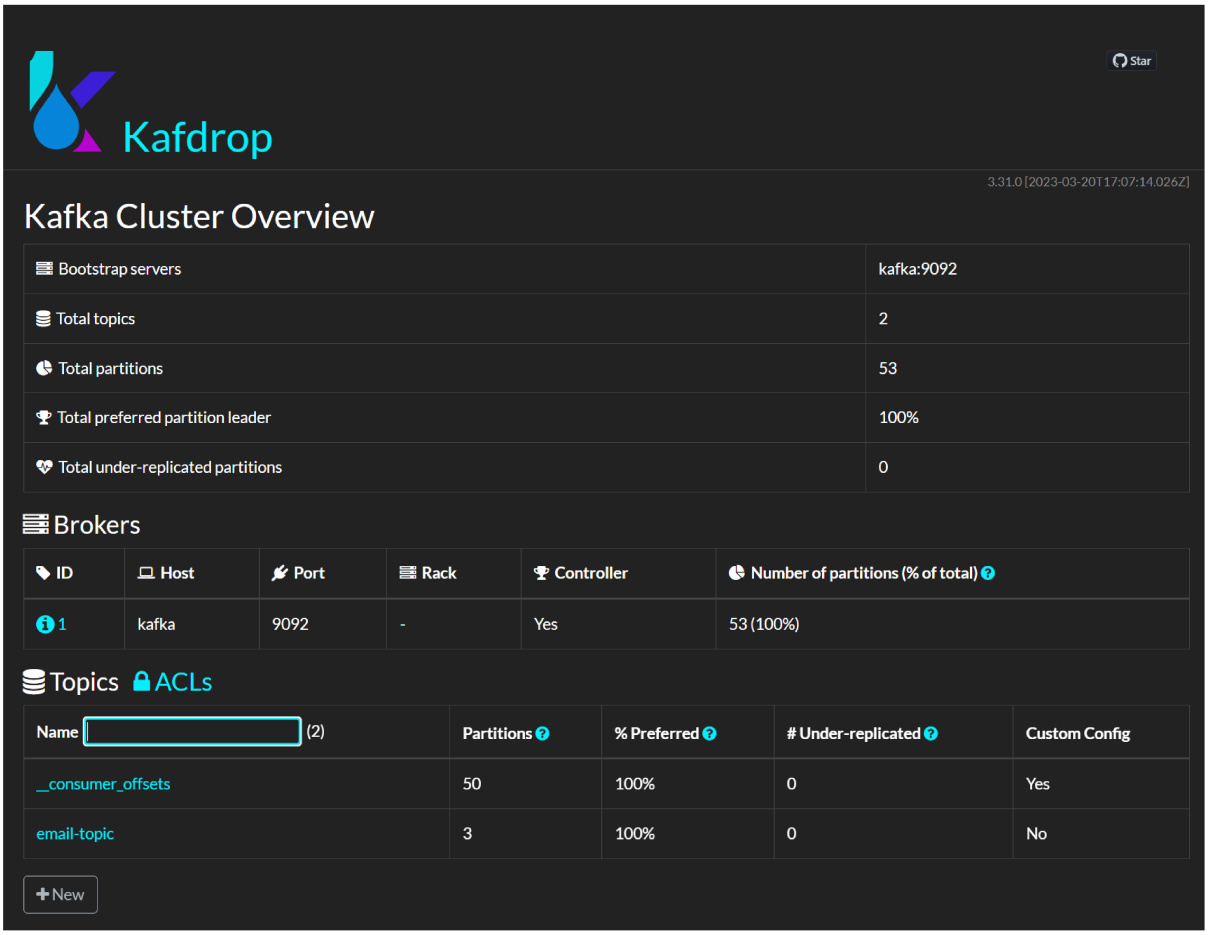


Figura 31 – Implantação [1]: Docker -> Kafkadrop

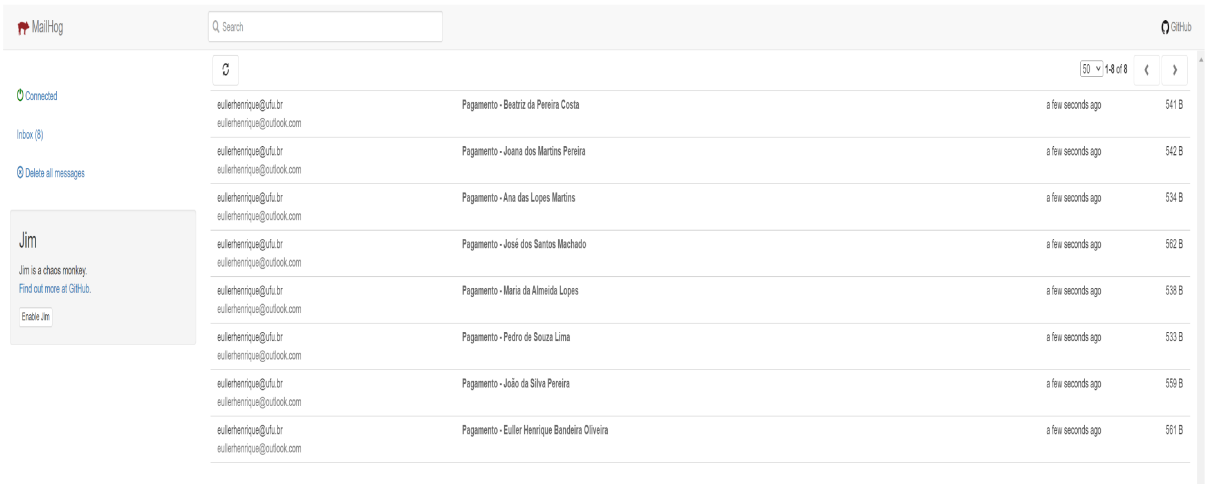


Figura 32 – Implantação [2]: Docker -> MailHog

4 Resultados

Este capítulo possui o objetivo de apresentar as vantagens e desvantagens da arquitetura de microsserviços identificadas ao longo do desenvolvimento do ecossistema de microsserviços simplificado e do ecossistema de microsserviços real.

4.1 Vantagens da Arquitetura de Microsserviços

1. Atribuição mais fácil da implementação

   A atribuição da implementação é facilitada, ou seja, realizar a atribuição da implementação à um desenvolvedor ou à um time é mais fácil, pois:

   Cada microsserviço possui o design orientado a domínios (DDD) e é responsável pelos seus próprios dados. Devido a essas características, mesmo que uma funcionalidade a ser desenvolvida em um microsserviço dependa de uma funcionalidade de outro microsserviço que ainda não foi desenvolvida, o desenvolvimento das duas funcionalidades pode ser isolado, ou seja, um desenvolvedor/time pode ser responsável pelo desenvolvimento das funcionalidades de cada microsserviço. Com isso, o desenvolvimento se torna mais rápido e fluido, já que cada desenvolvimento não irá interferir diretamente no outro (NEWMAN, 2020b) (NEWMAN, 2020c) (NEWMAN, 2022).

2. Implantação mais fácil

   A implantação é facilitada, ou seja, inserir o código executável em um servidor online é mais fácil, pois:

   - Cada microsserviço possui sua própria configuração de implantação e seu próprio servidor. Com isso, mesmo se um microsserviço possuir funcionalidades que dependem de outros microsserviços, sua implantação não será bloqueada. Desse modo, as funcionalidades que dependem somente do microsserviço em questão vão funcionar e as funcionalidades que dependem dos outros microsserviços vão funcionar quando eles forem implantados. Portanto, cada microsserviço pode ser implatando de maneira independente a qualquer momento (NEWMAN, 2020b) (NEWMAN, 2022).
   - Cada microsserviço pode ser conectar a uma pipeline que oferece suporte a integração, entrega e implantação contínua (CI/CD), ou seja, ela detecta o commit enviado ao sistema de controle de versões (Integração), transforma o código escrito em código executável (Entrega) e insere o código executável em um servidor online (Implantação). Em outras palavras, a pipeline detecta o

commit, realiza o build e o deploy do sistema de maneira automática (JETBRAINS, 2023) (MICROSOFT, 2023e).

3. Manutenção mais fácil

   A manutenção é facilitada, ou seja, realizar ajustes é mais fácil, pois:

   - Se o ecossistema de microsserviço utilizar o mesmo padrão de arquitetura e os mesmos padrões de projeto, os ajustes de cada microsserviços serão facilitados pois cada sistema possuirá a mesma estrutura/organização.
   - Cada microsserviço possui o design orientado a domínios (DDD), ou seja, cada microsserviço representa um domínio (Ex: ms-tarefa representa o dominío Tarefa (Tabela Principal)) e subdomínios (Ex: Fase da Tarefa (Tabela Auxiliar)) que são utilizados no desenvolvimento das funcionalidades. Devido a essa característica, a probabilidade de ser necessário modificar mais de um microsserviço para corrigir uma determinada funcionalidade é bem baixa, já que há uma grande chance da funcionalidade em questão pertencer somente a um domínio (NEWMAN, 2020b) (NEWMAN, 2022).

4. Escabilidade mais fácil

   A escabilidade é facilitada, ou seja, reduzir ou aumentar o desempenho é mais fácil, pois:

   - Escabilidade Vertical (Sclale-Up):

     Cada microsserviço pode ser facilmente escalado verticalmente (Redução/Aumento da quantidade de recursos (Processamento, armazenamento, memória RAM etc) de um servidor) de acordo com a demanda (Automaticamente ou não) por meio do Azure, posto que, cada microsserviço possui seu próprio servidor (MICROSOFT, 2023b) (BOUGUEZZI, 2023).
   - Escabilidade Horizontal (Scale-Out):

     Cada microsserviço pode ser facilmente escalado horizontamente (Redução/Aumento da quantidade de servidores do cluster) de acordo com a demanda (Automaticamente ou não) por meio do Kubernetes, posto que, cada microsserviço possui seu próprio cluster (MICROSOFT, 2023b) (BOUGUEZZI, 2023).

4.2 Desvantagens da Aquitetura de Microsserviços

1. Utilização limitada do Hibernate

   O Hibernate é uma ferramenta de mapeamento objeto-relacional (ORM) que permite que uma entidade (Objeto que representa uma determinada tabela) se relacione com outra(s) entidade(s) por meio do HQL (Hibernate Query Language) (DEVMEDIA, 2023d).

Para esse mapeamento de relações ser possível, é preciso que o microsserviço em questão contenha todas as entidades que serão relacionadas, isto é, é necessário que ele utilize objetos para representar todas as tabelas que serão utilizadas nas junções (joins). Devido ao fato de que cada microsserviço deve ser responsável pelos seus próprios dados (NEWMAN, 2020b) (NEWMAN, 2022), a utilização de tal recurso do hibernate se torna limitado, pois um determinado microsserviço pode se relacionar apenas às tabelas que ele acessa.

2. Inadequado para geração de relatórios extensos

   Extrair e processar numerosos dados por meio de um microsserviço não é viável, pois como cada microsserviço deve ser responsável pelos seus próprios dados (NEWMAN, 2020b) (NEWMAN, 2022), ao utilizar tal arquitetura para gerar um relatório é preciso juntar dados de várias tabelas. Com isso, é necessário que o microsserviço utilizado para gerar o relatório realize inúmeras requisições para outros microsserviços. Tal quantidade volumosa de requisições faz com que o tempo de processamento da requisição principal (Gerar Relatório) se torne extremamente longo.

   Para gerar um relatório rapidamente por meio da arquitetura de microsserviços, é preciso ignorar um dos seus pilares, ou seja, é necessário desenvolver um microsserviço "Coringa", isto é, um que possui a possibilidade de utilizar todas as tabelas disponíveis no banco de dados. Desse modo, será possível realizar consultas com junções (joins) por meio do SQL para obter todos os dados exigidos para a montagem do relatório de uma maneira rápida e simples.

3. Implementação mais complexa

   Desenvolver uma funcionalidade em um microsserviço que depende de outros microsserviços é muito mais complexo que desenvolvê-la em um único sistema, pois:

   - Se o microsserviço x crescer tanto a ponto de se considerar a possibilidade de decompô-lo, é preciso definir se tal esforço realmente vale a pena. Decompor um microsserviço não é uma atividade fácil e trivial, visto que possivelmente suas interfaces (endpoints) estão sendo usadas por outros microsserviços e suas funcionalidades estão muito acopladas. Ex: Para dividir o microsserviço ms-timesheet em dois (ms-timesheet e ms-dia) seria necessária muita cautela e muito tempo de desenvolvimento para que essa divisão não colapsasse o ecossistema de microsserviços.
   - Se o microsserviço x não tiver se conectado ao microsserviço y ainda, é preciso criar várias classes em várias pastas para que a conexão possa ser realizada.
   - Se o micrrosserviço y ainda não possuir a funcionalidade necessária, é preciso desenvolvê-la para que o desenvolvimento da funcionalidade do microsserviço x possa continuar.

- Se o microsserviço x precisa do microsserviço y somente para que ele faça uma simples busca em uma de suas tabelas, é preciso que o microsserviço x realize uma requisição e aguarde a resposta do microsserviço y. Se tais dados fossem obtidos por meio da realização de consultas com junções (joins) utilizando o SQL, o desenvolvimento seria mais rápido e simples.

4. Utilização limitada de *Transactions*

   Uma transação é um bloco de operações SQL (Iniciado por meio do comando BEGIN TRANSACTION) que devem ser executadas em sequência pelo SGDB. É preciso utilizar o comando COMMIT para definir que tais operações serão efetivadas se nenhuma falhar. Se alguma falhar, é preciso executar o comando ROLLBACK para que todas as operações sejam revertidas, ou seja, para que o banco de dados volte ao estado original (SOUSA, 2020).

   Todavia, para uma transação poder ser executada em um sistema é preciso que ele acesse todas as tabelas que serão utilizadas nas operações da transação. O problema é que segundo Newman (2020b) e Newman (2022), um dos pilares da arquitetura de microsserviços é: Cada microsserviço deve ser responsável pelos seus próprios dados. Tal característica não é compatível com a lógica das transações, pois uma determinada transação pode precisar de tabelas que estão em outros microsserviços. Com isso, o uso de transações se torna possível apenas com operações que utilizam tabelas presentes no microsserviço em questão.

5 Conclusão

Este capítulo possui o objetivo de apresentar as considerações finais deste trabalho de conclusão de curso.

- Esta monografia cumpriu:
  - O seu objetivo:

    "O objetivo geral deste estudo de caso é identificar e exemplificar as vantagens e desvantagens da arquitetura de microsserviços após implementar e investigar um ecossistema de microsserviços."
  - A sua justificativa:

    "Compreender a arquitetura de microsserviços em seu nível teórico e prático é um requisito cada vez mais solicitado no mercado de trabalho. Em vista disso, este estudo de caso justifica-se pela necessidade de se criar uma referência para que possa ser consultada por qualquer desenvolvedor que queira entender essa arquitetura ou que esteja em busca de razões para adotá-la

    Com base na fundamentação teórica (Conceitos), desenvolvimento (Implementação) e resultados (Vantagens e desvantagens), o leitor conseguirá compreender a arquitetura de microsserviços e deliberar se sua adoção é apropriada ou não."
- Esta monografia me proporcionou:
  - Conhecimento Teórico

    As pesquisas realizadas para a formulação de cada seção (Principalmente a seção da fundamentação teórica) fizeram com que eu adquirisse diversos conhecimentos interessantes e relembrasse vários conhecimentos importantes adquiridos ao longo da minha graduação.
  - Conhecimento Prático

    O desenvolvimento do ecossistema de microsserviços simplificado (Requisitos, Modelagem e Implementação) fez com que eu detectasse a necessidade de algumas melhorias no ecossistema de microsserviços real.
- Esta monografia me inspirou:
  - A adquirir mais conhecimentos da Arquitetura de Microsserviços

    Pretendo aperfeiçoar meus conhecimentos a cerca de tal arquitetura.
  - A adquirir mais conhecimentos do Desenvolvimento Dev Ops

    Desejo aprimorar meus conhecimentos sobre o Desenvolvimento Dev Ops.

Referências

AFONSO, A. **Tutorial definitivo: Tudo o que você precisa para começar bem com JPA**. 2019. Disponível em: <https://blog.algaworks.com/tutorial-jpa/>. Acesso em: 1 mai. 2023. Citado na página 23.

AMAZON. **O que é a diagramação de arquitetura?** 2023. Disponível em: <https://aws.amazon.com/pt/what-is/architecture-diagramming/>. Acesso em: 25 abr. 2023. Citado 2 vezes nas páginas 15 e 46.

_____. **O que é SQL?** 2023. Disponível em: <https://aws.amazon.com/pt/what-is/sql/>. Acesso em: 4 mai. 2023. Citado na página 21.

BOUGUEZZI, S. **Differences Between Scaling Horizontally and Vertically**. 2023. Disponível em: <https://www.baeldung.com/cs/scaling-horizontally-vertically>. Acesso em: 21 out. 2023. Citado na página 60.

COELHO, B. **Pesquisa qualitativa: entenda como utilizar essa abordagem de pesquisa**: O que é pesquisa qualitativa? 2017. Disponível em: <https://blog.mettzer.com/pesquisa-qualitativa/>. Acesso em: 21 abr. 2023. Citado na página 13.

DEVMEDIA. **Modelagem de software com UML?** 2011. Disponível em: <https://www.devmedia.com.br/modelagem-de-software-com-uml/20140>. Acesso em: 30 abr. 2023. Citado 2 vezes nas páginas 14 e 39.

_____. **Como começar com Spring?** 2023. Disponível em: <https://www.devmedia.com.br/exemplo/como-comecar-com-spring/73>. Acesso em: 1 mai. 2023. Citado na página 22.

_____. **Gerando Relatórios com JasperReports**. 2023. Disponível em: <https://www.devmedia.com.br/gerando-relatorios-com-jasperreports/24798>. Acesso em: 31 mai. 2023. Citado na página 24.

_____. **Guia Completo de SQL**: Organização da sql. 2023. Disponível em: <https://www.devmedia.com.br/guia/guia-completo-de-sql/38314>. Acesso em: 4 mai. 2023. Citado na página 21.

_____. **Guia de Referência Hibernate**. 2023. Disponível em: <https://www.devmedia.com.br/guia/hibernate/38312>. Acesso em: 1 mai. 2023. Citado 2 vezes nas páginas 23 e 60.

_____. **Os 4 pilares da Programação Orientada a Objetos**. 2023. Disponível em: <https://www.devmedia.com.br/os-4-pilares-da-programacao-orientada-a-objetos/9264>. Acesso em: 10 mai. 2023. Citado na página 20.

DIGITAL, O. **Adeus, Internet Explorer! Navegador é desativado em mais versões do Windows)**. 2023. Disponível em: <https://olhardigital.com.br/2023/02/14/internet-e-redes-sociais/adeus-internet-explorer-navegador-e-desativado-em-mais-versoes-do-windows/>. Acesso em: 09 abr. 2023. Citado na página 11.

FOWLER, M. **Data Mapper**. 2002. Disponível em: <https://martinfowler.com/eaaCatalog/dataMapper.html>. Acesso em: 30 set. 2023. Citado na página 51.

_____. **Data Transfer Object**. 2002. Disponível em: <https://martinfowler.com/eaaCatalog/dataTransferObject.html>. Acesso em: 30 set. 2023. Citado na página 51.

_____. **Repository**. 2002. Disponível em: <https://martinfowler.com/eaaCatalog/repository.html>. Acesso em: 30 set. 2023. Citado na página 51.

_____. **Service Layer**. 2002. Disponível em: <https://martinfowler.com/eaaCatalog/serviceLayer.html>. Acesso em: 30 set. 2023. Citado na página 51.

G1. **Fim de uma era: Microsoft aposenta o Internet Explorer nesta quarta-feira (15)**. 2022. Disponível em: <https://g1.globo.com/tecnologia/noticia/2022/06/15/fim-de-uma-era-microsoft-aposenta-o-internet-explorer-nesta-quarta-feira-15.ghtml>. Acesso em: 09 abr. 2023. Citado na página 11.

GEEKHUNTER. **Como fazer a integração de dados com Apache Kafka**. 2020. Disponível em: <https://blog.geekhunter.com.br/apache-kafka/>. Acesso em: 3 mai. 2023. Citado na página 24.

_____. **Como construir uma aplicação com Docker?**: O que é docker. 2022. Disponível em: <https://blog.geekhunter.com.br/docker-na-pratica-como-construir-uma-aplicacao/>. Acesso em: 1 mai. 2023. Citado na página 25.

_____. **Spring Boot: Tudo que você precisa saber!** 2022. Disponível em: <https://blog.geekhunter.com.br/tudo-o-que-voce-precisa-saber-sobre-o-spring-boot/>. Acesso em: 1 mai. 2023. Citado na página 23.

GUEDES, M. **No final das contas: o que é o Docker e como ele funciona?**: O que são esses containers? 2019. Disponível em: <https://www.treinaweb.com.br/blog/no-final-das-contas-o-que-e-o-docker-e-como-ele-funciona>. Acesso em: 1 mai. 2023. Citado na página 25.

GUPTA, R.; SAHU, A. **Software Architecture Patterns: What Are the Types and Which Is the Best One for Your Project**. 2023. Disponível em: <https://www.turing.com/blog/software-architecture-patterns-types/>. Acesso em: 29 set. 2023. Citado na página 51.

HUDSON. **Atividades básicas ao processo de desenvolvimento de Software**: Implementação. 2007. Disponível em: <https://www.devmedia.com.br/atividades-basicas-ao-processo-de-desenvolvimento-de-software/5413>. Acesso em: 23 abr. 2023. Citado 2 vezes nas páginas 15 e 51.

IBM. **Deploying software**. 2021. Disponível em: <https://www.ibm.com/docs/en/zos/2.4.0?topic=task-deploying-software>. Acesso em: 23 abr. 2023. Citado na página 16.

_____. **Como o Java funciona?** 2023. Disponível em: <https://www.ibm.com/br-pt/topics/java>. Acesso em: 17 abr. 2023. Citado na página 22.

_____. **Database management systems on z/OS**. 2023. Disponível em: <https://www.ibm.com/docs/en/zos-basic-skills?topic=zos-what-is-database-management-system>. Acesso em: 5 mai. 2023. Citado na página 22.

JETBRAINS. **Integração Contínua vs. Entrega vs. Implantação**. 2023. Disponível em: <https://www.jetbrains.com/pt-br/teamcity/ci-cd-guide/continuous-integration-vs-delivery-vs-deployment/>. Acesso em: 8 mai. 2023. Citado 2 vezes nas páginas 21 e 60.

JOEL. **MER e DER: Modelagem de Bancos de Dados**. 2014. Disponível em: <https://www.devmedia.com.br/mer-e-der-modelagem-de-bancos-de-dados/14332>. Acesso em: 26 abr. 2023. Citado 2 vezes nas páginas 15 e 49.

JUNIOR, A. L. **Apache Kafka: entenda conceitos, arquiteturas e componentes**. 2021. Disponível em: <https://imasters.com.br/software/apache-kafka-entenda-conceitos-arquiteturas-e-componentes>. Acesso em: 3 mai. 2023. Citado na página 24.

KRIGER, B. **O QUE É GIT: CONCEITOS, PRINCIPAIS COMANDOS E QUAIS AS VANTAGENS?** 2022. Disponível em: <https://kenzie.com.br/blog/o-que-e-git/>. Acesso em: 1 mai. 2023. Citado na página 25.

LARMAN, C.; BASILI, V. R. **Iterative and IncrementalDevelopment: A Brief History**. 2003. Disponível em: <https://www.cs.hmc.edu/courses/2004/fall/cs121/papers/larman.pdf>. Acesso em: 23 abr. 2023. Citado 2 vezes nas páginas 13 e 30.

LUCIDCHART. **O que é diagrama de atividades UML?** 2023. Disponível em: <https://www.lucidchart.com/pages/pt/o-que-e-diagrama-de-atividades-uml>. Acesso em: 26 abr. 2023. Citado 2 vezes nas páginas 15 e 44.

MENDES, I. S. **Arquitetura monolítica vs microsserviços: uma análise comparativa**. 2021. Disponível em: <https://bdm.unb.br/handle/10483/30715>. Acesso em: 22 mai. 2023. Citado 4 vezes nas páginas 26, 27, 28 e 29.

MENEZES, P. **Estudo de Caso**: O que é um estudo de caso? 2023. Disponível em: <https://www.significados.com.br/estudo-de-caso/>. Acesso em: 21 abr. 2023. Citado na página 13.

MICROSOFT. **O que é o Azure Repos?** 2022. Disponível em: <https://learn.microsoft.com/pt-br/azure/devops/repos/get-started/what-is-repos>. Acesso em: 9 mai. 2023. Citado na página 26.

_____. **Conceitos básicos do Azure Key Vault**. 2023. Disponível em: <https://learn.microsoft.com/pt-br/azure/key-vault/general/basic-concepts>. Acesso em: 01 jun. 2023. Citado na página 23.

_____. **Criar aplicativos para dimensionamento**. 2023. Disponível em: <https://learn.microsoft.com/pt-br/azure/well-architected/scalability/design-scale>. Acesso em: 21 out. 2023. Citado na página 60.

_____. **Estilo de arquitetura controlada por evento**. 2023. Disponível em: <https://learn.microsoft.com/pt-br/azure/architecture/guide/architecture-styles/event-driven>. Acesso em: 2 mai. 2023. Citado 2 vezes nas páginas 20 e 49.

_____. **O que é o Azure Boards?** 2023. Disponível em: <https://learn.microsoft.com/pt-br/azure/devops/boards/get-started/what-is-azure-boards?view=azure-devops>. Acesso em: 9 mai. 2023. Citado na página 26.

_____. **O que é o Azure Pipelines?** 2023. Disponível em: <https://learn.microsoft.com/pt-br/azure/devops/pipelines/get-started/what-is-azure-pipelines>. Acesso em: 9 mai. 2023. Citado 2 vezes nas páginas 26 e 60.

_____. **O que é o DevOps?** 2023. Disponível em: <https://azure.microsoft.com/pt-br/resources/cloud-computing-dictionary/what-is-devops>. Acesso em: 9 mai. 2023. Citado na página 26.

_____. **Sobre o Azure Key Vault**. 2023. Disponível em: <https://learn.microsoft.com/pt-br/azure/key-vault/general/overview>. Acesso em: 01 jun. 2023. Citado na página 23.

MOZILLA. **MVC**. 2023. Disponível em: <https://developer.mozilla.org/en-US/docs/Glossary/MVC>. Acesso em: 29 set. 2023. Citado na página 51.

_____. **Uma visão geral do HTTP**. 2023. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Overview>. Acesso em: 1 mai. 2023. Citado na página 19.

_____. **Uma visão geral do HTTP**. 2023. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>. Acesso em: 1 mai. 2023. Citado na página 19.

NATIVE, C. **Arquitetura Cliente Servidor**: O que é? 2022. Disponível em: <https://glossary.cncf.io/pt-br/client-server-architecture/>. Acesso em: 6 mai. 2023. Citado 2 vezes nas páginas 19 e 47.

NEWMAN, S. **Migrando sistemas monolíticos para microsserviços**: Padrões evolutivos para transformar seu sistema monolítico. Novatec Editora, 2020. 27-30 p. ISBN 9786586057041. Disponível em: <https://books.google.com.br/books?id=4cjYDwAAQBAJ>. Acesso em: 14 mai. 2023. Citado na página 17.

_____. **Migrando sistemas monolíticos para microsserviços**: Padrões evolutivos para transformar seu sistema monolítico. Novatec Editora, 2020. 14-26 p. ISBN 9786586057041. Disponível em: <https://books.google.com.br/books?id=4cjYDwAAQBAJ>. Acesso em: 16 mai. 2023. Citado 6 vezes nas páginas 18, 48, 59, 60, 61 e 62.

_____. **Migrando sistemas monolíticos para microsserviços**: Padrões evolutivos para transformar seu sistema monolítico. Novatec Editora, 2020. 50-61 p. ISBN 9786586057041. Disponível em: <https://books.google.com.br/books?id=4cjYDwAAQBAJ>. Acesso em: 21 out. 2023. Citado na página 59.

_____. **Criando Microsserviços**: Projetando sistemas com componentes menores e mais especializados. Novatec Editora, 2022. ISBN 9786586057881. Disponível em: <https://books.google.com.br/books?id=RZdkEAAAQBAJ>. Acesso em: 16 mai. 2023. Citado 6 vezes nas páginas 18, 48, 59, 60, 61 e 62.

NOLETO, C. **Como funciona a POO?** 2020. Disponível em: <https://blog.betrybe.com/tecnologia/poo-programacao-orientada-a-objetos/>. Acesso em: 18 abr. 2023. Citado na página 20.

_____. **Conheça a arquitetura REST e seus princípios!** 2022. Disponível em: <https://blog.betrybe.com/desenvolvimento-web/api-rest-tudo-sobre/#1>. Acesso em: 19 abr. 2023. Citado na página 19.

REDHAT. **O que é arquitetura orientada a eventos?**: O que é um evento? 2019. Disponível em: <https://www.redhat.com/pt-br/topics/integration/what-is-event-driven-architecture>. Acesso em: 2 mai. 2023. Citado na página 19.

_____. **O que é arquitetura orientada a eventos?**: Como a arquitetura orientada a eventos funciona? 2019. Disponível em: <https://www.redhat.com/pt-br/topics/integration/what-is-event-driven-architecture>. Acesso em: 2 mai. 2023. Citado 2 vezes nas páginas 19 e 49.

_____. **O que é o Apache Kafka?**: Visão geral. 2022. Disponível em: <https://www.redhat.com/pt-br/topics/integration/what-is-apache-kafka>. Acesso em: 3 mai. 2023. Citado na página 23.

_____. **Kubernetes**. 2023. Disponível em: <https://www.redhat.com/pt-br/topics/containers/what-is-kubernetes>. Acesso em: 1 mai. 2023. Citado na página 25.

_____. **O que é API?** 2023. Disponível em: <https://www.redhat.com/pt-br/topics/api/what-are-application-programming-interfaces>. Acesso em: 1 mai. 2023. Citado na página 19.

SIGNIFICADOS. **Pesquisa Exploratória (Estudo Exploratório)**: O que é uma pesquisa exploratória? 2023. Disponível em: <https://www.significados.com.br/pesquisa-exploratoria/>. Acesso em: 21 abr. 2023. Citado na página 13.

SOUSA, J. **Transações no SQL: Mantendo os dados íntegros e consistentes**. 2020. Disponível em: <https://www.alura.com.br/artigos/transacoes-no-sql-mantendo-os-dados-integros>. Acesso em: 08 out. 2023. Citado na página 62.

TEDESCO, K. **Padrões de projeto: o que são e o que resolvem**. 2017. Disponível em: <https://www.treinaweb.com.br/blog/padroes-de-projeto-o-que-sao-e-o-que-resolvem>. Acesso em: 01 out. 2023. Citado na página 52.

TIBCO. **O que é Apache Kafka?**: Conceitos do apache kafka. 2021. Disponível em: <https://www.tibco.com/pt-br/reference-center/what-is-apache-kafka>. Acesso em: 3 mai. 2023. Citado na página 24.

_____. **O que é arquitetura orientada a eventos?** 2023. Disponível em: <https://www.tibco.com/pt-br/reference-center/what-is-event-driven-architecture>. Acesso em: 2 mai. 2023. Citado 2 vezes nas páginas 19 e 49.

VALENTE, M. T. **Engenharia de Software Moderna**: Requisitos. 2022. Disponível em: <https://engsoftmoderna.info/cap3.html>. Acesso em: 23 abr. 2023. Citado 5 vezes nas páginas 14, 15, 30, 32 e 39.

_____. **Engenharia de Software Moderna**: Arquitetura. 2022. Disponível em: <https://engsoftmoderna.info/cap7.html>. Acesso em: 20 set. 2023. Citado 2 vezes nas páginas 46 e 48.

VENTURA, P. **Caso de Uso – Include, Extend e Generalização**. 2014. Disponível em: <https://www.ateomomento.com.br/caso-de-uso-include-extend-e-generalizacao/>. Acesso em: 06 out. 2023. Citado na página 15.

WILLIANS, W. **Apache Kafka: Trabalhando com Mensageria e Real Time**. 2019. Disponível em: <https://fullcycle.com.br/apache-kafka-trabalhando-com-mensageria-e-real-time/>. Acesso em: 3 mai. 2023. Citado na página 24.

_____. **O que é... Kafka?** 2020. Disponível em: <https://fullcycle.com.br/o-que-e-kafka/>. Acesso em: 3 mai. 2023. Citado na página 24.

ZEIN, N. **Desvendando o Kafka — Parte 3: Partitions e Replication**. 2020. Disponível em: <https://medium.com/creditas-tech/desvendando-o-kafka-parte-3-partitions-e-replication-23b36bb88c80>. Acesso em: 3 mai. 2023. Citado na página 24.