UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

CENTRO DE CIÊNCIAS EXATAS, NATURAIS E DE SAÚDE | CCENS

FELIPE PITANGA BARCELOS

UMA PROPOSTA DE SISTEMA DE GESTÃO DE ORDENS DE SERVIÇO PARA DESENVOLVIMENTO E MANUTENÇÃO DE SOFTWARE

Alegre, ES

2021

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

CENTRO DE CIÊNCIAS EXATAS, NATURAIS E DE SAÚDE | CCENS

UMA PROPOSTA DE SISTEMA DE GESTÃO DE ORDENS DE SERVIÇO PARA DESENVOLVIMENTO E MANUTENÇÃO DE SOFTWARE

Trabalho de conclusão de curso apresentado à Universidade Federal do Espírito Santo, como parte das exigências para a obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Prof.º. Giuliano Prado de Morais Giglio

Alegre

2021

BANCA EXAMINADORA

Prof. Giuliano Prado de Morais Giglio

Afiliações

Prof. Bruno Vilela Oliveira

Afiliações

Prof. Marcelo Otone Aguiar

Afiliações

Resumo

Este trabalho visa realizar um levantamento de vários tipos de ordens de serviço, solicitações de mudança e requisições de serviço existentes no mercado, analisar suas características fundamentais para assim elaborar uma solução computacional, que permita profissionais do setor de desenvolvimento de *software* utilizarem ordens de serviço como ferramenta auxiliar de controle das atividades. Será realizada a implementação de um Sistema *Web* para ampla distribuição e fácil manutenção, que possibilite os usuários gerarem, emitirem relatórios e manterem um controle das ordens de serviço emitidas na organização, como alternativa mais completa e aderente aos sistemas similares existentes.

Palavras Chave: Sistema, *Web*, Ordem de Serviço, Solicitação de Mudança.

Abstract

This work aims to survey various types of work orders, change requests and service requests in the market, analyze their fundamental characteristics in order to develop a computational solution that allows professionals of the software development sector to use work orders as auxiliary tools for controlling activities. Implementation of a Web System for widespread deployment and easy maintenance will enable users to generate, issue reports and keep track of work orders issued within the organization, as a more complete and adherent alternative to existing similar systems.

Keywords: *System, Web, Work Order, Change Request.*

LISTA DE FIGURAS

Figura 1 - Visão Arquitetural do Sistema. ……...……..... 31
Figura 2 - Funcionalidade de Manter Usuários. ……..……... 34
Figura 3 - Funcionalidade de Manter Ordens de Serviço.……..…….....34
Figura 4 - Funcionalidade de Manter Requisitos……...….35
Figura 5 - Diagrama de Classes (Modelo). ...……...….. 36
Figura 6 - Diagrama de Classes (Controle). ……...…...36
Figura 7 - Diagrama de Classes (DAO). ……..……...…...37
Figura 8 - Sequência de Eventos de Fluxo Principal do Sistema. ..……..... 38
Figura 9 - Sequência de Criação de Usuário no Sistema……...39
Figura 10 - Modelo Relacional do Banco de Dados. ……..…...……...…….41
Figura 11 - Interface de Conexão com Banco de Dados…...……...…46
Figura 12 - Método de Conexão com Banco de Dados Oracle...…….....46
Figura 13 - Implementação Singleton na Conexão Oracle……...….47
Figura 14 - Estrutura do arquivo de dados da conexão……...……...…...47
Figura 15 - Interface de entidade DAO……...…..……...…….50
Figura 16 - Entidade de Nova Ordem de Serviço…...……...….51
Figura 17 - Criação de Repositório em Memória……...…..52
Figura 18 - Serviço de registro de uma Ordem de Serviço...……...……...……..52
Figura 19 - Método de interpretação de uma requisição REST...……...…..53
Figura 20 - Método de interpretação de uma requisição do tipo PUT.……...….54
Figura 21 - Método principal instanciando contextos da api…….……...….55
Figura 22 - Tela de Login…….……...….56
Figura 23 - Método para envio de requisições get ao servidor…….…….57
Figura 24 - Método de solicitar autenticação de Login……...…...57
Figura 25 - Método para autenticação de login……...…...58
Figura 26 - Menu do Sistema……..59
Figura 27 - Rotas……....……...….59
Figura 28 - Método de cálculo de Rotas……..……...….60

Figura 29 - Menu de Usuários……..……...….60
Figura 30 - Criação de uma tabela dinâmica de Interface……...……...….61
Figura 31 - Editando Permissões de Usuário……..……...….62
Figura 32 - Cadastro de Usuário……...……...….62
Figura 33 - Menu de Requisitos……...……...….63
Figura 34 - Cadastro de Requisito……...……...….63
Figura 35 - Edição de Requisito……...……...….63
Figura 36 - Configuração (Status).....……...….64
Figura 37 - Configuração (Prioridades).....……...….65
Figura 38 - Menu de Ordens de Serviço. Fonte: Do autor (2021).....……...….65
Figura 39 - Menu de Ordens de Serviço……..……...….66
Figura 40 - Cadastro de uma Ordem de Serviço……...……...….66
Figura 41 - Edição de uma Ordem de Serviço……...…..67
Figura 42 - Menu de Equipes……..68
Figura 43 - Visualização da Equipe……..……..68
Figura 44 - Teste de comunicação com Banco de Dados……..75
Figura 45 - Teste de Cadastro de Requisito com Data……...76
Figura 46 - Teste de comunicação Servidor-*Browser*……..76
Figura 47 - Teste de Cadastro Usuário……..77
Figura 48 - Fluxo de Cadastro de Requisito…...……..93
Figura 49 - Fluxo de Cadastro de Ordem de Serviço...……..94
Figura 50 - Fluxo de Cadastro de Equipes.……..94

LISTA DE GRÁFICOS

Gráfico 1 - O número de ECRs como uma função do tempo para um produto em uso. ……..... 21
Gráfico 2 - Fração de ECRs gerados ao longo de um cronograma para projetos de *NPD* na empresa de telecomunicações. ……..23

LISTA DE TABELAS

Tabela 1 - Planilha de Propriedades de cada modelo de ordem de serviço. …….28
Tabela 2 - Tabela dos Usuários. …….43
Tabela 3 - Tabela dos Requisitos. ……..43
Tabela 4 - Tabela de Ordens de Serviço……..……..45
Tabela 5 - Tabela de Equipes…….45
Tabela 6 - Tabela de Prioridades……...45
Tabela 7 - Tabela de Status……45
Tabela 8 - Estrutura do Projeto Java…….49
Tabela 9 - Tratativas Estabelecidas para Requisições REST por tipo……...…..54
Tabela 10 - Resultados dos Testes de Integração com Banco de Dados…..…...…..70
Tabela 11 - Resultados dos Testes de Integração Cliente-Servidor……..……..71
Tabela 12 - Resultados dos Testes de Integração entre Módulos……....……..72
Tabela 13 - Resultados dos Testes Estruturais……...….75
Tabela 14 - Planilha de Características realizadas x Características observadas no levantamento de Requisitos…….79

SUMÁRIO

1. Introdução **11**
1.1 Enterprise Resource Planning - ERP **12**
1.2 Ordens de Serviço em ERP's **12**
2. Objetivos **14**
2.1 Objetivo Geral **14**
2.2 Objetivos Específicos **14**
3. Revisão de Literatura **15**
3.1 Processo ECR **21**
4. Metodologia **24**
4.1 Levantamento de Requisitos **25**
4.2 Requisitos Funcionais 29
4.3 Arquitetura Web **30**
4.4 Modelagem UML **32**
4.4.1 Diagrama de Caso de Uso **33**
4.4.2 Diagrama de Classes **35**
4.4.3 Diagrama de Sequência **37**
4.5 Java 39
4.6 JavaScript **40**
4.7 Oracle **40**
5. Desenvolvimento **42**
5.1. Geração da base de dados **42**
5.1.1 Conexão com Banco de Dados 46
5.2. Aplicação Java Web **48**
5.2.1 Objetos de Acesso a Dados 49
5.2.2 Comunicação REST 50
5.3. Criação da Aplicação JavaScript **56**
5.3.1 Comunicação com o Servidor 56
5.3.2 Navegação 59
6. Testes **69**
6.1 Testes de Integração **69**
6.2 Testes de Componente 72
6.3 Testes Estruturais **73**
6.3.1 Código de Testes Caixa-Branca Realizados. 75
7. Resultados alcançados e conclusões **77**

7.1 Recomendações a Trabalhos Futuros 80

8. Referências Bibliográficas 83

9. Apêndice I - Narrativas de Caso de Uso 89

9.1 Caso de uso 001 – Cadastrar Usuário 89

9.2 Caso de uso 002 – Cadastrar Ordem de Serviço 90

9.3 Caso de uso 003 – Cadastrar Requisito 91

10. Apêndice II - Diagramas de Sequência 93

1. Introdução

Com o crescimento das atividades nos setores de serviços e comércio nos últimos anos, tem aumentado a demanda por sistemas de automação que possibilitem a organização do espaço de trabalho e das tarefas realizadas, a forma que seja possível para grandes organizações com um número considerável de empregados seja capaz de controlar e administrar todas as tarefas efetuadas de forma ordenada e padronizada.

Segundo Alvarenga e Silveira (2021), em fevereiro de 2021 o setor de serviços cresceu 3,7%, sendo uma necessidade indispensável inclusive em situações de calamidade pública.

Empresas como companhias de serviço, que implantam vários funcionários em uma ampla área geográfica para atender a uma infraestrutura ou base de clientes dispersos enfrentam a tarefa particularmente trabalhosa de comunicar atribuições de trabalho e dados relacionados a funcionários que estão dispersos no campo.

> Normalmente, o pessoal já está no campo quando novas tarefas de serviço ou ordens de serviço são geradas. Assim, as empresas de serviços se deparam com a tarefa muito complexa de receber ordens de serviço, identificando pessoal de campo mais adequado para o trabalho conforme determinado pelo treinamento, experiência e proximidade da área de trabalho e comunicando ao pessoal de campo que uma ordem de trabalho específica foi atribuída a eles. Em resposta, o pessoal de campo deve comunicar ao operador de despacho uma aceitação ou rejeição da ordem de serviço (KHALESSI; ARDALAN, 2003).

1.1 *Enterprise Resource Planning* - ERP

Um *Enterprise Resource Planning*, ERP, é um *software* capaz de integrar todos os dados e informações de uma empresa em um único sistema, descomplicando a integração entre departamentos distintos e facilitando o fluxo de informações. Além de melhorar a comunicação interna, promover a efetividade e a produtividade, um sistema de gestão empresarial pode transformar e aprimorar a rotina de setores estratégicos da companhia (PORTAL ERP, 2021).

Toda empresa precisa fazer um planejamento estratégico, estabelecendo metas e objetivos e definindo quais ferramentas serão utilizadas para atingir os resultados esperados. Nesse sentido, o sistema de gestão ERP surgiu para auxiliar as empresas a monitorar e a controlar todas as suas operações diárias, tendo um crescimento gradativo e sustentável (PORTAL ERP, 2021).

1.2 Ordens de Serviço em *ERP's*

Quando uma empresa contrata um Sistema de ERP para uso próprio, o faz para atender uma necessidade de organização e gestão, dentre estas múltiplas necessidades que um ERP atende, está a organização das demandas internas de uma empresa, das tarefas a serem realizadas, por quem serão realizadas e quando serão realizadas (PORTAL ERP, 2021).

Neste sentido, deve-se haver em um Sistema de Gestão ERP um elemento que confira e atenda esta necessidade com a organização destas demandas internas, que permita ao gestor ou supervisor de atividades da empresa organizar estes fatores (Tarefa, Executante, Período, Valor). Inclusive não deve sobrecarregar os funcionários e promover produtividade interna por meio da gestão das demandas.

É conhecidamente prejudicial o hábito de realizar solicitações de serviço e atendimento, principalmente em setores de informática, sem haver uma solicitação formal do serviço, dentro de organizações. Este tipo de prática comum leva a uma queda no desempenho no setor e na qualidade dos atendimentos uma vez que a maioria é realizada de forma semi-informal e não registrada.

> Um sistema de chamada ajuda toda a equipe de gestão de TI a aprimorar processos e tornar mais eficiente o serviço que presta aos seus clientes. As ferramentas disponíveis em algumas soluções também permitem um pré-atendimento, que é uma espécie de triagem de problemas onde são apresentadas informações relevantes para que o próprio cliente consiga solucionar situações específicas (QUALITOR , 2020).

"Esses recursos permitem uma otimização do tempo de trabalho, fazendo com que o time de TI possa dedicar a sua atividade à resolução de casos mais elaborados" (QUALITOR , 2020).

No mercado de desenvolvimento de *software* brasileiro, ainda é escassa a utilização de sistemas de Gestão de Ordens de Serviço para organização das demandas evolutivas de um sistema. Com a popularização de metodologias ágeis entre as equipes de desenvolvimento, é comum não haver mais documentação em excesso e, além disso, muitas vezes, nenhuma documentação sobre a demanda, o que pode ser prejudicial a longo prazo para a equipe.

Dado a natureza das ordens de serviço como artefato comunicativo em uma grande variedade de áreas de negócio, e a demanda crescente do mercado por sistemas de gestão de ordens de serviço eficientes e eficazes, com critérios de priorização que se adaptem às necessidades dos clientes e praticidade de uso, este projeto visa realizar o desenvolvimento de uma solução computacional atendendo aos requisitos mais comuns de um sistema desta natureza.

O intuito deste trabalho é trazer uma solução prática, para gestão das ordens de serviço e de demandas dentro de equipes de desenvolvimento e fábricas de *software*.

2. Objetivos

2.1 Objetivo Geral

Desenvolver um sistema de informação para gestão de ordens de serviços, com o objetivo de automatizar o processo de documentação das atividades realizadas dentro de uma fábrica de *software*, tanto em demandas evolutivas quanto corretivas. Este sistema será desenvolvido a partir de um levantamento, a ser realizado, de sistemas já existentes deste tipo, buscando reunir as melhores características observadas nos sistemas identificados, e que ao final seja um sistema funcional para gestão de ordens de serviço, específico para o mercado de desenvolvimento de *software*, a ser utilizado por equipes de desenvolvimento e seus gestores, e que se seja uma proposta mais completa que os sistemas observados.

2.2 Objetivos Específicos

- Permitir o controle do ciclo de vida de uma Ordem de Serviço: as ordens de serviço referente a uma demanda, deverá estar definida qual etapa aquela demanda se encontra, entre em espera até concluída.

- Permitir guardar um valor referente a custos de uma Ordem de Serviço: ao ser cadastrada uma ordem de serviço, o sistema deverá guardar os custos monetários informados para esta.

- Permitir guardar o esforço necessário para realização de uma Ordem de Serviço: guardar um valor referente a quantidade relativa de esforço necessário para realização de uma demanda, quando esta for informada ao sistema.

- Permitir a priorização da realização de uma Ordem de Serviço: guardar um valor referente a prioridade de realização de uma demanda, quando esta for informada ao sistema.

- Comparar a evolução de uma Ordem de Serviço, ao longo do seu ciclo de vida: uma ordem de serviço deve poder evoluir dentro de um ciclo, em que, se uma demanda for referente ou derivada de outra demanda (ex. um processo de cadastro de preços dentro de um software depende da importação destes preços por *excel* em formato *.csv*) então o sistema de gestão de ordens de serviço deverá permitir esta rastreabilidade e dependência entre as demandas.

3. Revisão de Literatura

Segundo Jacobs (2008), uma ordem de serviço complexa é uma tarefa a ser executada por um ou mais membros de uma força de trabalho, o que requer que duas ou mais ordens de serviço relacionadas sejam concluídas normalmente em uma ordem prescrita ou preferida. São fornecidas nestas, estruturas de dados, processos e sistemas para gerenciar ordens de serviço complexas para facilitar a coordenação eficiente da força de trabalho entre os técnicos que trabalham nas ordens de serviço relacionadas.

> O gerenciamento do agendamento e distribuição de ordens de serviço para uma força de trabalho composta por vários técnicos de serviço móvel foi facilitado nos últimos anos, por meio de programas de computador para rastreamento, programação e atribuição de ordens de serviço desde o recebimento da solicitação de um pedido até tempo de conclusão pelo técnico de serviço. Coletivamente, tais programas de computador são designados como sistemas de gerenciamento de força de trabalho (*WMS*) (JACOBS , 2008).

Essa coordenação engloba uma variedade de aspectos, incluindo, mas não limitados a, assegurar que algumas ordens de serviço sejam concluídas antes que outras ordens de serviço sejam iniciadas em um local diferente (por exemplo, onde um técnico deve executar o trabalho em uma subestação antes que outro técnico possa fazer alguns reparos elétricos nas instalações do cliente); garantir que uma coleta de ordens de serviço para a mesma instalação seja feita durante a mesma visita ao cliente (por exemplo, uma visita às instalações do cliente para fornecer um serviço de telecomunicações pode envolver a ativação de determinados canais de entretenimento, a configuração do acesso à *Internet* e recursos como chamada em espera); ou, mais tipicamente, garantir que o tempo de início e conclusão de sub-ordens separadas para a mesma premissa seja realizado em uma sequência produtiva (por exemplo, garantir que um técnico conclua a instalação de um cabo antes que outro técnico chegue para instalar um decodificador).

A coordenação de ordens de serviço complexas também engloba a alocação de recursos, o agendamento, a atribuição e / ou a otimização da distribuição de ordens de serviço para membros de uma força de trabalho (JACOBS, 2008).

Em geral, a solução adotada nas empresas para gerenciamento do atendimento comercial é a execução dessa atividade por técnicos que realizam manualmente a análise das informações disponíveis ao distribuir as ordens entre as equipes no início dos dias de trabalho. Após distribuídas as ordens, cabe a cada equipe definir também manualmente a sequência em que realizará seus

atendimentos. Tanto a distribuição das ordens quanto às sequências de atendimento admitem um grande número de combinações, o que as tornam tarefas difíceis de serem otimizadas sem o auxílio de uma ferramenta computacional (BARBOSA et al.,2015).

Antes da introdução de sistemas de agendamento informatizados, o agendamento de ordens de serviço em uma instalação de fabricação era realizado manualmente (LILLY; LAYNE, 1998).

Segundo Lilly e Layne (1998), o processo de agendamento manual envolvia a determinação do trabalho a ser executado e o tempo, materiais e recursos (máquinas, trabalhadores, ferramentas etc.) necessários para executá-lo, além de outras informações que influenciavam o cronograma, como por exemplo: o trabalho era mais importante que outro trabalho. Este processo é caro, muito tedioso e muitas vezes impreciso. A frequência com que o processo de agendamento deve ser realizado contribui para a dificuldade de obter agendamento manual oportuno e preciso.

No final da década de 1950 e durante a década de 1960, foram feitos esforços para implementar o Planejamento de Requisitos de Materiais ou o *MRP*. O *MRP* tinha um escopo limitado para lidar apenas com os requisitos de material (ou seja, quais peças ou materiais eram necessários e quando). Esses sistemas computadorizados iniciais não eram capazes de programar operações com base na disponibilidade de materiais ou capacidade de recursos.

Durante as décadas de 1970 e 1980, muitos sistemas comercialmente disponíveis foram criados, o que proporcionou um *MRP* mais abrangente, e acabou sendo renomeado como *Manufacturing Resource Planning*. O novo *MRP* lidou com problemas adicionais, como previsão, gerenciamento de estoque, controle de chão de fábrica, engenharia e gerenciamento de mudanças. Alguns desses sistemas

lidavam com a capacidade da planta para realizar o trabalho. Uma dessas abordagens foi chamada de planejamento de capacidade bruta e, como o nome sugere, forneceu uma estimativa grosseira da capacidade necessária para satisfazer os requisitos de trabalho. O planejamento da capacidade de corte bruto determinou a capacidade necessária, mas não foi capaz de programar ordens de serviço com base nos limites de capacidade.

Foi em meados da década de 1980 que se desenvolveu um sistema informatizado de programação orientada a eventos capaz de programar ordens de serviço e as operações necessárias para executar as ordens de serviço, levando em conta a capacidade finita dos diversos recursos necessários para o processo de fabricação, por exemplo, expediente finito. A *ProfitKey*, uma empresa sediada em *New Hampshire*, desenvolveu um dos primeiros sistemas de informação de recursos finitos disponíveis no mercado. Usando o Agendador *ProfitKey*, um fabricante pode determinar a data de início e término das ordens de serviço e suas operações com base na capacidade e disponibilidade de recursos. Um sistema semelhante foi desenvolvido pela *Lilly Software Associates, Inc*. e comercializado como parte do sistema *VISUAL Manufacturing* ™ a partir do início de 1993 (LILLY; LAYNE, 1998).

Outros sistemas de agendamento incluem o *Microsoft Project* ™ e o *Norton's Timeline da Symantec*. Esses programas são capazes de armazenar e recuperar informações sobre um projeto definido pelo usuário usando um método consistente com a teoria do *Critical Path Method (CPM)*. Essas ferramentas são projetadas para lidar com projetos que têm eventos definíveis que são interdependentes, semelhantes a uma única ordem de serviço em um processo de fabricação. Esses sistemas não são, no entanto, capazes de gerenciar centenas de ordens de serviço de cada vez, como é típico na maioria das fábricas (LILLY; LAYNE, 1998).

Ordem de Serviço também é amplamente utilizado no mercado de fábrica de software, além de serviços públicos em geral, a exemplo de Guimarães (2011).É

utilizado principalmente como um mecanismo de controle de prestação de serviço e especificação.

Segundo Antoniol et al. (2008), métricas de projeto e código-fonte, arquitetura e recursos de *design* são usados como variáveis independentes nos modelos que explicam ou prevêem defeitos. A previsão pode ser realizada em diferentes níveis de granularidade, isto é, podem estar relacionados para unidades como componentes, classes, arquivos, ou métodos; o número de defeitos na unidade ou a probabilidade de que a unidade contém pelo menos um defeito é quase sempre uma variável altamente dependente.

Projetos de código aberto, bem como de código fechado, gerenciar problemas usando sistemas de rastreamento de *bugs* (*BTS*) deve ser usado principalmente para gerenciar questões relacionadas à manutenção corretiva, ou seja, *bugs*. No entanto, eles geralmente contêm entradas outras atividades de *software*, como manutenção preventiva; pedido novas características; questões legais e de licenciamento; discussões sobre mudanças arquitetônicas e reestruturação.

O pressuposto subjacente é que uma taxa de previsão de defeitos pode direcionar de forma lucrativa as atividades de verificação e validação para o subconjunto de artefatos com uma maior propensão a erros.

Uma cadeia padrão de “reação” aos bugs é a seguinte: cliente, analista ou desenvolvedor comete um erro ao descrever o problema ou desenvolvimento de artefatos de software. Este erro pode ou não resultar em falha no sistema, um estado inesperado. Esse estado inesperado pode ou não resultar em um *bug*, um evento visível e não desejado do ponto de vista de um usuário. Quando um *bug* é descoberto, ele se manifesta, é documentado e os detalhes são postados no *BTS*.

O estudo de Antoniol (2008) , mostra que a informação linguística contida nas entradas do *BTS* é suficiente para distinguir automaticamente a manutenção corretiva de outras atividades. Isso é relevante nesse aspecto pois ela abre a possibilidade de construir sistemas de roteamento automáticos, ou seja, sistemas que classificam automaticamente os tickets enviados e encaminhá-los para a equipe de manutenção (*bugs*) ou para o líder da equipe (solicitações de aprimoramento e outros problemas). Também relata a experiência em classificar manualmente 1.800 problemas do *BTS* de três grandes sistemas de código aberto: *Mozilla, Eclipse* e *JBoss*. Mostrando que, usando um mecanismo de votação simples, é possível classificar esse grande número de problemas, distinguindo entre "*bugs*" (relacionados à manutenção corretiva) e "não-*bug*" (relacionados a outras atividades de manutenção, documentação e assim por diante).

Segundo Canfora et al. (2006), em empresas que projetam e produzem produtos complexos, especialmente produtos personalizados, mudanças e modificações frequentemente ocorrem no projeto do produto à medida que ele evolui. Muitas dessas mudanças são formalmente iniciadas pelo cliente como novos requisitos ou pela empresa como especificações modificadas ou mudanças de fabricação. Essas solicitações, chamadas de solicitações de alteração de engenharia (*ECRs*), podem ocorrer antes, durante ou depois da produção. Normalmente, quanto mais tarde eles ocorrem, mais significativo é o tempo e o esforço necessários para implementá-los (PIKOSZ, 1998; MALMQVIST, 1998; REIDELBACH, 1991). As alterações também podem ocorrer durante o uso de um produto devido a erros de *design*, alternativas para peças de reposição ou modificações para melhoria no final da vida útil do produto.

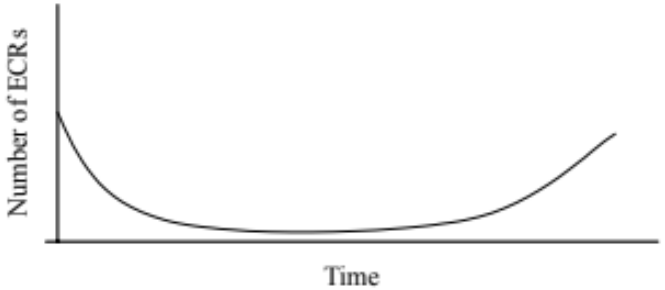
A implementação de *ECRs* afeta várias funções em uma empresa, como engenharia, vendas, fabricação, compras, controle de estoque, contabilidade, entre outras. Com o tempo de comercialização sendo um fator competitivo importante, é desejável processar de forma rápida e precisa os *ECRs*. Além disso, pode ser um

processo dispendioso devido a possível sucata, atrasos, novas compras, substituição de componentes no estoque, entre outros. (DIPRIMA, 1982). Quão bem uma empresa responde a uma mudança conduzida pelo cliente ou pela empresa geralmente determina se um novo produto é entregue no prazo e com lucro (CANFORA et al., 2006).

3.1 Processo *ECR*

Todas as empresas devem fazer alterações de design em seus produtos para se adaptar às mudanças nos requisitos do cliente, mudanças nas tecnologias ou mudanças no ambiente, como alguns exemplos. Além disso, um *ECR* pode ser emitido para problemas relacionados à qualidade de uma parte, revisão de uma parte, correção de um erro em um documento, revisão e / ou renovação de um documento, desenvolvimentos recentes em tecnologia ou resultados de testes de protótipos (PIKOSZ, 1998; MALMQVIST, 1998). Os *ECRs* surgem como parte do ciclo de vida do produto. Uma vez que um produto foi vendido, há muitas mudanças sugeridas no início do uso do produto (produto não está funcionando corretamente ou não atende às expectativas), algumas após o período de uso inicial (um período relativamente longo de uso estável) e mudanças no final do ciclo de vida (adaptação devido à obsolescência do produto). O diagrama do Gráfico 1 mostra a tendência no número de *ECRs* durante o tempo de vida de um produto (curva em forma de banheira semelhante a uma curva de confiabilidade).

Gráfico 1 - O número de *ECRs* como uma função do tempo para um produto em uso.



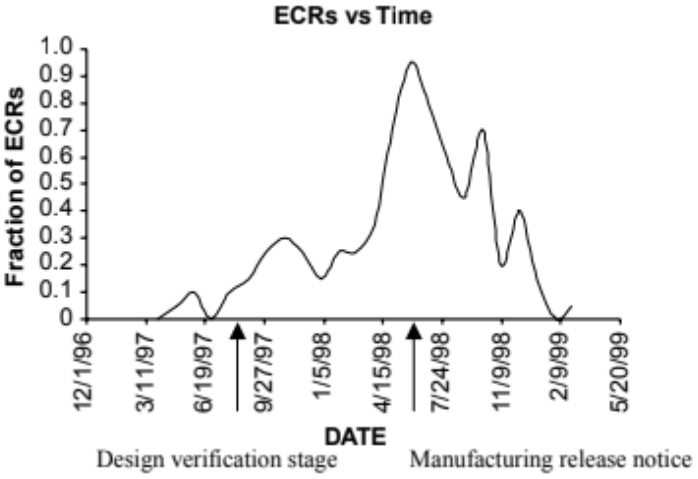
Fonte: (CANFORA et al., 2006)

Quando um produto está em uso existem duas estratégias principais para executar um *ECR*: (1) executar a *ECR* imediatamente devido a preocupações com segurança ou funcionalidade crítica; ou (2) acumular *ECRs* e executá-los como um grupo (CROW, 2002).

A última tática é usada por vários motivos: eficiência na execução de *ECRs* juntos, tornando os *ECRs* parte de uma nova versão ou lançamento de modelo ou a necessidade de criar um kit de várias alterações devido ao impacto de entregar e ter o cliente (ou outros) efetuar as alterações no campo. Essas táticas são usadas por muitas empresas e podem ser vistas em indústrias como: automotiva, de *software*, eletrônica e aeroespacial. Por exemplo, para reelaborar um molde de injeção na indústria automotiva, a execução imediata de *ECR* exigiria que, para cada *ECR*, o molde fosse desmontado e retrabalhado. Se, no entanto, os *ECRs* forem agrupados, muitas alterações poderão ser feitas em uma configuração de reorganização, economizando significativamente nos custos de desmontagem e configuração (TERWIESCH, 1999; LOCH, 1999).

Há também *ECRs* que ocorrem durante a criação de um novo produto. O Gráfico 2 mostra uma linha do tempo e o número associado de *ECRs* gerados durante o desenvolvimento de um novo produto composto de uma montagem de placa de circuito. Este gráfico vem de um estudo de processos de Desenvolvimento de Novos Produtos (*NPD*) realizado em uma empresa de alta tecnologia que projeta e fabrica sistemas de telecomunicações (BHUIYAN et al., 2004 a, b).

Gráfico 2 - Fração de *ECRs* gerados ao longo de um cronograma para projetos de *NPD* na empresa de telecomunicações



Fonte: (CANFORA et al., 2006)

Como pode ser visto, o número de *ECRs* aumenta substancialmente logo após o estágio de verificação do projeto, e mais ainda, logo após o início da fabricação. A maioria das alterações no processo de *NPD* ocorre de maneira *ad hoc* durante o *design*, e muitas sugestões de alteração ocorrem fora de sequência dentro do processo de *design* (BHUIYAN et al., 2004 a, b).

O processamento desses *ECRs* se enquadra nas duas principais táticas: executar um *ECR* imediatamente ou agrupar muitos *ECRs* e executá-los durante um ciclo natural do processo de *design* ao criar protótipos ou novas versões de *design*. Na indústria eletrônica, é bastante comum projetar um produto com uma versão "*respin*" ou *design* para fazer um *layout* final e seleção de peças para acomodar apropriadamente os *ECRs* que ocorreram durante o processo de projeto (BHUIYAN et al., 2004 a, b).

Quando ocorrem mudanças, uma maneira formal de gerenciá-las é necessária para a eficiência. O gerenciamento do processo de *ECR* é um processo de negócios especial que se preocupa com o gerenciamento de mudanças e é frequentemente usado durante o processo de desenvolvimento de produtos. Depois que uma alteração é autorizada, os detalhes das novas especificações são liberados

para todas as partes interessadas. O cronograma de produção mestre e o arquivo de lista de materiais são então verificados para explicitar como a mudança impacta os requisitos de materiais e componentes, por exemplo, para lidar com peças de reposição, peças agora em falta, ou peças que não são mais necessárias. Assim, o estoque é atualizado, todas as novas compras são ordenadas, os cronogramas de produção são alterados, os registros contábeis são atualizados e os recursos são preenchidos, conforme necessário. Tudo isso requer informações precisas e atualizadas, uma grande quantidade de mudanças no documento e uma comunicação oportuna com o pessoal-chave. Os procedimentos reais para processar *ECRs* podem variar dependendo de como o *ECR* surge e quais funções em uma organização estão envolvidas (CANFORA et al., 2006).

A pesquisa de Cânfora (CANFORA et al., 2006), enfoca os *ECRs* que ocorrem durante o processo de *NPD*, embora os resultados sejam considerados aplicáveis a outros processos envolvendo *ECRs*.

4. Metodologia

Segundo Baribieri (2018)*, software* é um produto qualitativo. Dado a natureza de ser um produto feito em apenas uma unidade, a maior parte das características de um *software* são qualitativas e, devido a isso, este estudo baseia-se em aspectos qualitativos para desenvolvimento de *software*, de caráter descritivo sobre os elementos fundamentais para um sistema de gestão de ordens de serviço.

Para o levantamento de dados e informação, será utilizado tanto o levantamento de dados bibliográficos quanto estudos laboratoriais de desenvolvimento, e que, ao fim do procedimento, seja possível comparar o sistema desenvolvido aos demais sistemas do mesmo tipo existentes no mercado.

Foi realizado um levantamento de ordens de serviço presentes em diversos tipos de negócio, a critério de serem ordens de serviço utilizadas no mercado tendo assim utilidade prática. Destas, foi feita uma análise para quais características presentes em cada ordem se serviço eram as mais relevantes a serem utilizadas em um sistema para gestão de desenvolvimento de *software*. De todas as ordens de serviço analisadas, uma amostra foi separada para ser utilizada como base para os requisitos, dado que as características contidas neste grupo de ordens de serviço eram necessidades comuns a diversos ramos de negócio distintos do mercado.

Os procedimentos utilizados envolvem uma análise preliminar de *softwares* similares já existentes, suas características e propriedades. A partir desta análise foi possível realizar o levantamento de requisitos preliminares e efetuar a modelagem da ordem de serviço a ser utilizada pelo sistema.

4.1 Levantamento de Requisitos

As ordens de serviço utilizadas para levantamento de requisitos, seguidas de seus respectivos motivos para escolha foram:

- **Ordem de Serviço Governo Federal:** Ordem de serviço em descrição formal, utilizada por autoridades de estado, tem uma descrição clara e objetiva da tarefa a ser realizada (GOVERNO FEDERAL , 2021).

  Este modelo de ordem de serviço foi selecionado a partir de uma busca no portal do Governo Federal, a ser utilizada como modelo de ordens de serviço dentro do ministério da economia, mostrando ser comum a sistemas *Web* de Gestão uma necessidade de ter modelos pré-fabricados para documentos oficiais a fim de estabelecer um padrão nos documentos utilizados.

- **Ordem de Serviço AgoraOS:** Tem o diferencial de ser uma ordem de serviço personalizável de acordo com a regra de negócio do cliente (AGORAOS, 2019).

  Este exemplo de ordem de serviço é utilizada para registrar serviços de manutenção em componentes eletrônicos, mostrando relevância de ser utilizada como fonte de requisitos por conter informações sobre produtos e serviços utilizados assim como informação de valor e solicitante da ordem de serviço, sendo assim utilizada também como utilidade tanto comercial quanto para fins de registro e gestão das demandas.

- **Ordem de Serviço Sofit:** Foi escolhida por conter em detalhes campos para todos os indicadores do executante e controle de status do andamento da demanda.

  Este modelo de ordem de serviço não só contém as informações do executante como características do mesmo, apontando um sistema com a necessidade de cadastro de usuários executantes para registro de ordens de serviço, além disso a informação de status da ordem de serviço mostra a necessidade de não apenas controlar as demandas mas organizar como estas são efetuadas de forma organizada em relação ao tempo (SOFIT, 2019).

- **Ordem de Serviço Soften Sistemas:** Além dos controles de Status da demanda, permite a adição de produtos que deverão ser entregues juntamente a realização do serviço (SOFTEN SISTEMAS, 2019).

  A regra de negócio da Soften Sistemas utiliza os produtos alvo da ordem de serviço (ex. Celular) como objeto principal da demanda, mostra a necessidade da ordem de serviço conter registros detalhados do motivo da execução e do que foi realizado para que um sistema seja capaz de manter um histórico de serviços realizados e soluções apresentadas, desta forma

inclusive facilitando demandas futuras que podem ser iguais ou similares a demandas já atendidas.

- **Ordem de Serviço Engeman:** Permite a utilização de Sub-Tópicos e Sub-OS para as demandas cadastradas, este conceito permite que múltiplas Sub-OS sejam lançadas como *thread* de atualização de status detalhado sobre a solução da demanda, ela agrega a prática de montar um banco de conhecimento sobre solução de demandas já resolvidas para facilitar a execução de demandas futuras mas além disso, permite que múltiplos usuários atuem como parte da solução de uma ordem de serviço (ENGEMAN, 2019).

- **Ordem de Serviço Gestão Click:** Esta ordem de serviço foi escolhida pelo diferencial de ser uma solução 100% *Web*, onde os registros ficam condensados em um servidor com a interface para manipulação de OS distribuídas para os usuários. Apontando assim uma das necessidades principais de um sistema de gestão: ser um sistema distribuído e de fácil acesso aos usuários autorizados a manipulação de ordens de serviço (GESTÃO CLICK, 2019).

- **Modelo Ordem de Serviço ERP Sirius:** Por estar integrado com um ERP, este módulo de OS permite o cálculo dos custos e receitas referentes a OS e utiliza estes para gerar financeiros em um integração com o módulo financeiro do ERP.

  Este exemplo mostra a necessidade importante de ser possível para um módulo de gestão de ordem de serviço integrar com outros módulos e conter informações que sejam relevantes para outras funcionalidades (ex. Gestão Financeira), não necessariamente a gestão de financeiros está no escopo do componente de gestão de ordens de serviço mas deve permitir a extensão da informação para uso em outros componentes caso necessário (SIRIUS , 2019).

A Tabela 1 apresenta uma comparação entre as informações disponíveis em cada ordem de serviço, dentre todas as ordens de serviço levantadas, quais contém um tipo de informação e quais não contém sinalizada pelo símbolo X, nas colunas constam as Ordens de Serviço analisadas e nas linhas os tipos de informação fornecidas.

	Engeman	**Soften**	**Sofit**	**Gestão Click**	**Governo Federal**	**AgoraOS**	**Sírius**
Código	X	X	X		X	X	X
Status	X		X	X			X
Nome do Solicitante	X	X		X		X	X
Data/Hora Emissão/Entrega	X	X	X	X		X	X
Descrição	X	X	X	X	X	X	X
Quantitativo					X	X	X
Priorização	X						
Valor		X	X			X	X
Setor/Equipe	X						

Tabela 1 - Planilha de Propriedades de cada modelo de ordem de serviço. Fonte: Do autor (2021).

Também foram levantados requisitos funcionais com base no fluxo de uma ordem de serviço dentro de um cenário corporativo, com a emissão de relatórios de relevância para cada atividade, em resumo, as ordens de serviços utilizadas na amostra de estudo seguem um fluxo padrão:

1. É realizada uma solicitação de um serviço, que dá origem a demanda.
2. A demanda é analisada e as demais informações relevantes a demanda são coletadas e registradas.
3. A demanda é efetuada por um executante.
4. O esforço necessário para realização é registrado e a demanda é concluída.

Após a especificação dos requisitos, foi realizada a modelagem *UML* e a elaboração dos diagramas de Caso de Uso, de Classes e de Sequência. Para serem utilizados como documentação auxiliar na implementação do *software*.

4.2 Requisitos Funcionais

Os Requisitos Funcionais identificados a partir do levantamento realizado foram:

RF01 - Manter Usuários - O Sistema deve manter um controle de usuários cadastrados e informações relevantes referentes a estes, como forma de controlar e identificar os utilizadores do sistema.

RF02 - *Login* - O Sistema precisa conter um módulo de autenticação de usuários utilizadores para garantir que apenas estes poderão acessar o sistema.

RF03 - Manter Equipes - O Sistema deverá permitir agrupar os usuários e demandas em grupos, referidos como equipes, onde será possível identificar os seus integrantes.

RF04 - Manter Requisitos - O Sistema deverá ser capaz de manter as necessidades que estão sendo atendidas e associar estas aos respectivos serviços realizados referentes a cada necessidade cadastrada.

RF05 - Manter Ordens de Serviço - O Sistema deverá guardar as demandas que estão sendo realizadas, inclusive identificando a necessidade que originou aquela demanda, com todas as informações relevantes a demanda como o que está sendo realizado, o solicitante da demanda, o encarregado, quando esta demanda será realizada e qual o esforço para realização desta.

RF06 - Manter Status e Prioridades - O Sistema também deverá guardar o status de andamento das demandas cadastradas, e as respectivas prioridades de execução de cada demanda.

4.3 Arquitetura *Web*

O sistema contará com arquitetura *web* para melhor disponibilidade e facilidade de manutenção.

De acordo com Oliveira Júnior e Fortes (2007) esse tipo de sistema não exige que a sua execução seja realizada mediante sua prévia instalação local, assim como é feito com aplicações *desktop*. Muito pelo contrário, a execução da grande maioria de aplicações *Web* acontece somente com o uso de um navegador *Web*.

"Atualmente a literatura apresenta um conjunto de vários padrões arquiteturais para sistemas distribuídos e interativos. As aplicações *Web* são consideradas sistemas interativos sendo que o padrão mais conhecido é o *MVC* (*Model-View-Controller*)" (BUSCHMANN et al., 1996; PRESSMAN, 2006).

A Figura 1 mostra como cada um dos componentes do padrão *MVC* estão dispostos e utilizados dentro do modelo arquitetural *Web* no sistema que será desenvolvido.

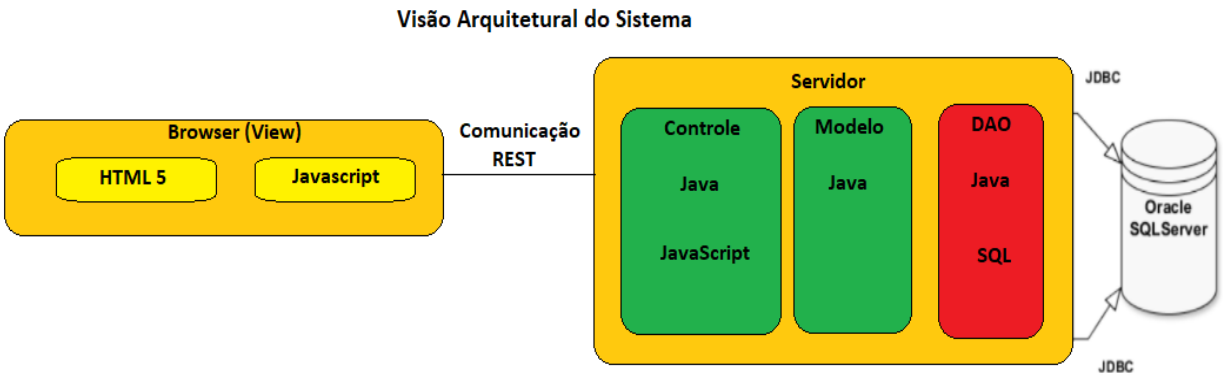


Figura 1 - Visão Arquitetural do Sistema. Fonte: Adaptado de Sankhya Code (2021)

O *MVC* é formado por três componentes principais, sendo eles:

• **Modelo (*Model*):** engloba o conteúdo e a lógica de processamento específicos da aplicação, incluindo todos os objetos de conteúdo (persistentes) e dados externos de informação. Esta parte do sistema estará disposta como na figura 1, localizada no *server-side* do sistema e desenvolvida majoritariamente em Java.

• **Visão (*View*):** contém todas as funções específicas da interface e permite a apresentação do conteúdo e da lógica de processamento, incluindo todos os objetos de conteúdos, dados de informação e a funcionalidade de processamento requerida pelo usuário final. A *view*, como mostrado na Figura 1, está localizada no *client-side* do sistema, realizando comunicação direta com o servidor, terá elementos visuais manipulados por Javascript.

• **Controlador (*Controller*):** gera o acesso ao Modelo e à Visão e coordena o fluxo de dados entre eles. No sistema a entidade de controle atuará no servidor, processando as requisições recebidas da *view* e acionando os demais mecanismos do sistema quando necessário, inclusive montando respostas de retorno ao *front-end* da aplicação. Como mostrado na Figura 1, ela será desenvolvida majoritariamente em Java e Javascript.

O *MVC* tem como objetivo desacoplar a interface da navegação e do comportamento da aplicação permitindo uma manutenção mais fácil e uma maior reutilização.

Além destes, o Sistema contará com um componente específico para Acesso a Dados, utilizando para isso o padrão *DAO* de desenvolvimento, segundo a *Sun Microsystems inc.* (2007):

> Em engenharia de *software*, um objeto de acesso a dados (*DAO*) é um padrão que fornece uma interface abstrata para algum tipo de banco de dados ou outro mecanismo de persistência. Ao mapear as chamadas do aplicativo para a camada de persistência, o DAO fornece algumas operações de dados específicas sem expor detalhes do banco de dados. Esse isolamento apóia o princípio de responsabilidade única. Ele separa quais dados de acesso o aplicativo precisa, em termos de objetos específicos de domínio e tipos de dados (a interface pública do *DAO*), de como essas necessidades podem ser satisfeitas com um Banco de dados específico, esquema de banco de dados, etc.

O Padrão DAO será utilizado como modelo principal para comunicação com o Banco de Dados.

4.4 Modelagem *UML*

> A *UML* (Linguagem Unificada de Modelagem) é uma linguagem gráfica de modelagem para visualização, especificação, construção e documentação para desenvolver sistemas computacionais orientados a objeto, esta modelagem é apresentada através de diagramas. É utilizada para uma melhor compreensão do sistema que será desenvolvido, especificar comportamentos internos que são as variáveis que interagem com o sistema (usuário e ambiente) e internos (sistema e suas especificações) e documentar as decisões tomadas (VERGILIO, 2011).

A modelagem *UML* será utilizada como principal forma de documentar o sistema.

4.4.1 Diagrama de Caso de Uso

"O diagrama de casos de uso especifica um conjunto de funcionalidades, através do elemento sintático 'casos de uso', e os elementos externos que interagem com o sistema, através do elemento sintático 'ator'" (SILVA, 2007). Além de casos de uso e atores, este diagrama contém relacionamentos de dependência, generalização e associação e são basicamente usados para fazer a modelagem de visão estática do caso de uso do sistema.

Essa visão proporciona suporte principalmente para o comportamento de um sistema, ou seja, os serviços externamente visíveis que o sistema fornece no contexto de seu ambiente. Neste caso os diagramas de caso de uso são usados para fazer a modelagem do contexto de um sistema e fazer a modelagem dos requisitos de um sistema.

Os Diagramas de Caso de Uso também servirão para apresentar uma visão das permissões e capacidades de cada ator do sistema, quais funcionalidades são acessíveis a cada tipo de usuário.

O Diagrama de Caso de Uso, montado de acordo com os requisitos levantados, é mostrado nas Figuras 2, 3 e 4; com todas as funcionalidades previstas no Sistema.

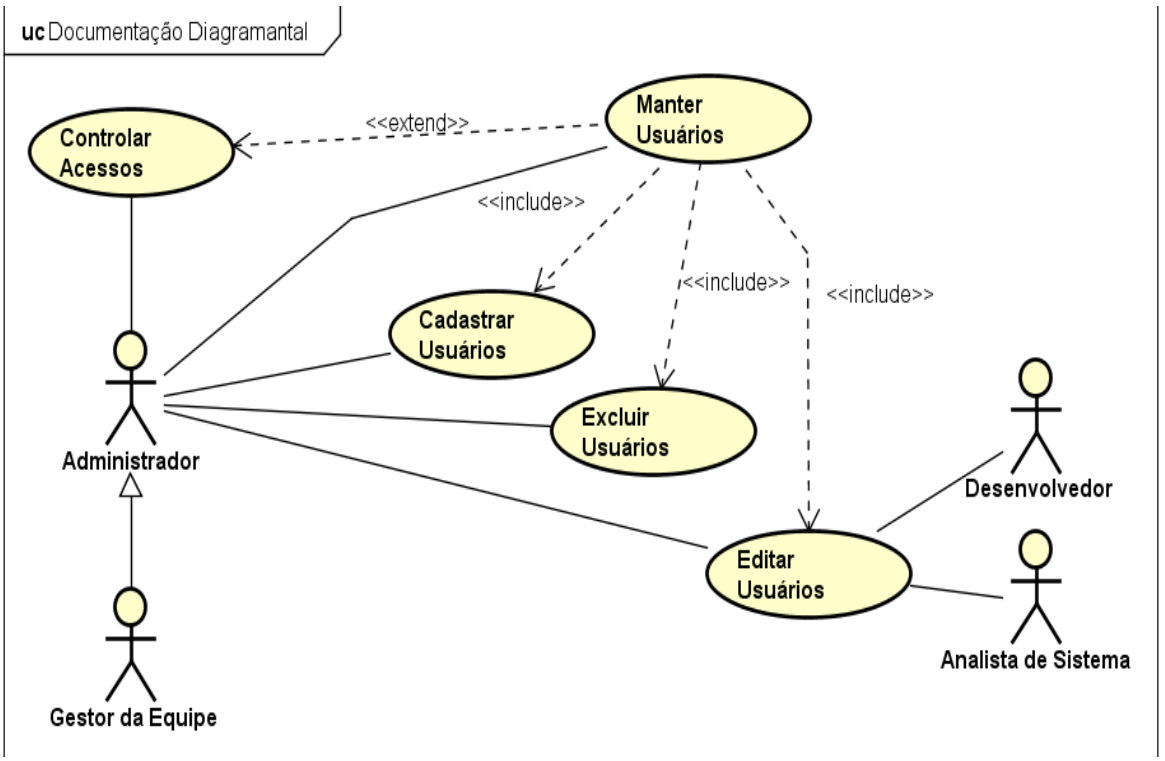


Figura 2 - Funcionalidade de Manter Usuários Fonte: Do autor (2021)]

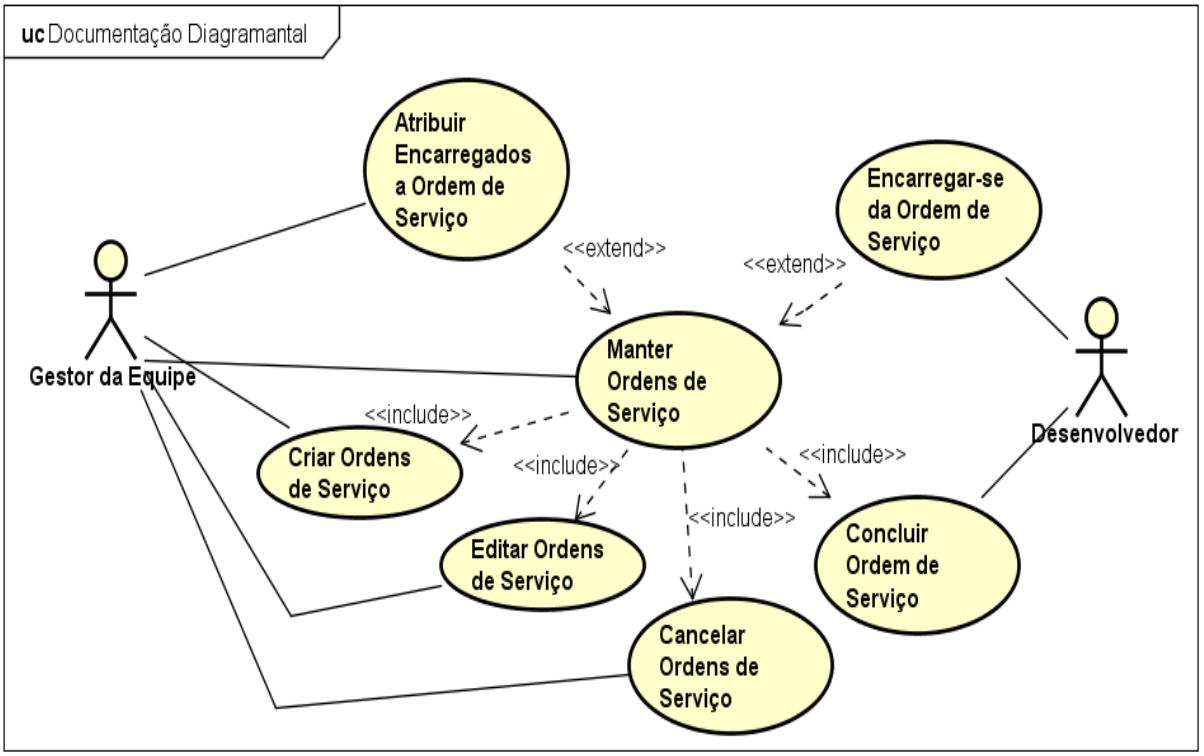


Figura 3 - Funcionalidade de Manter Ordens de Serviço Fonte: Do autor (2021)

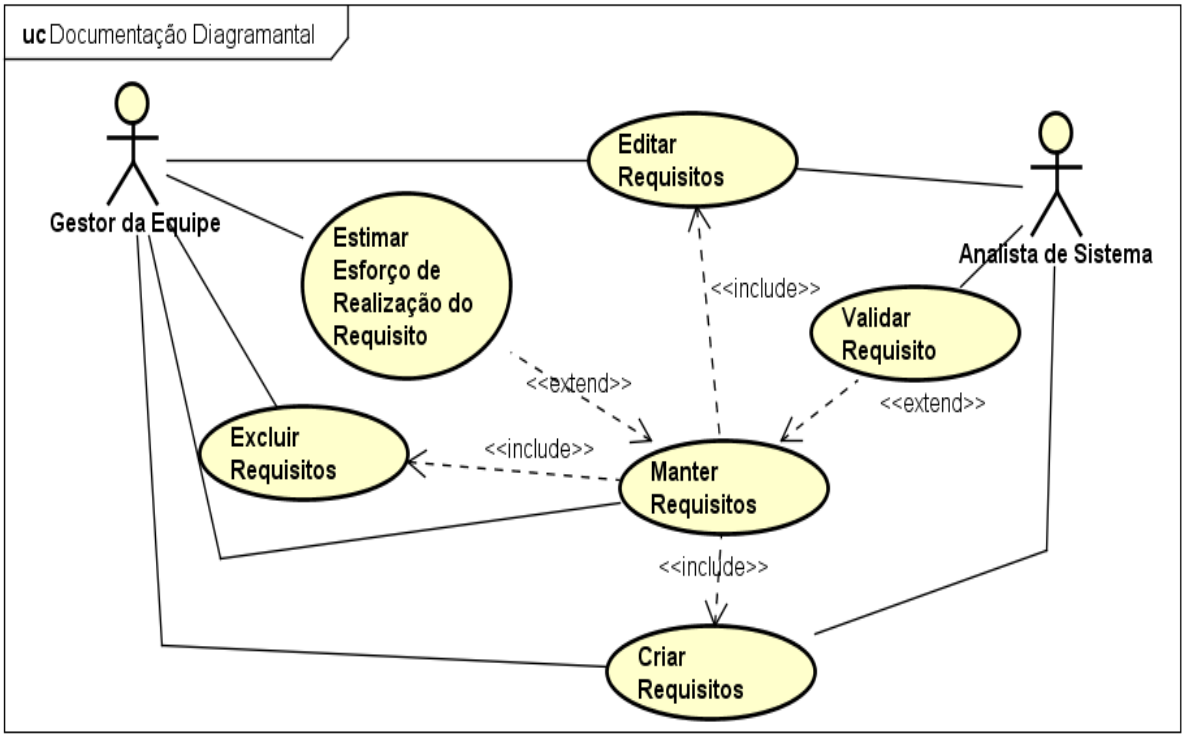


Figura 4 - Funcionalidade de Manter Requisitos Fonte: Do autor (2021)

4.4.2 Diagrama de Classes

Um diagrama de classes é um modelo fundamental de uma especificação orientada a objetos. Produz a descrição mais próxima da estrutura do código de um programa, ou seja, mostra o conjunto de classes com seus atributos e métodos e os relacionamentos entre classes. Classes e relacionamentos constituem os elementos sintáticos básicos do diagrama de classes (SILVA, 2007).

O Diagrama de classes será utilizado como um facilitador, caso seja necessário, no processo de desenvolvimento, como forma de identificar quais entidades estarão presentes no *software* e como as mesmas estarão relacionadas. As Figuras 5, 6 e 7 mostram os Diagramas de Classe do Sistema.

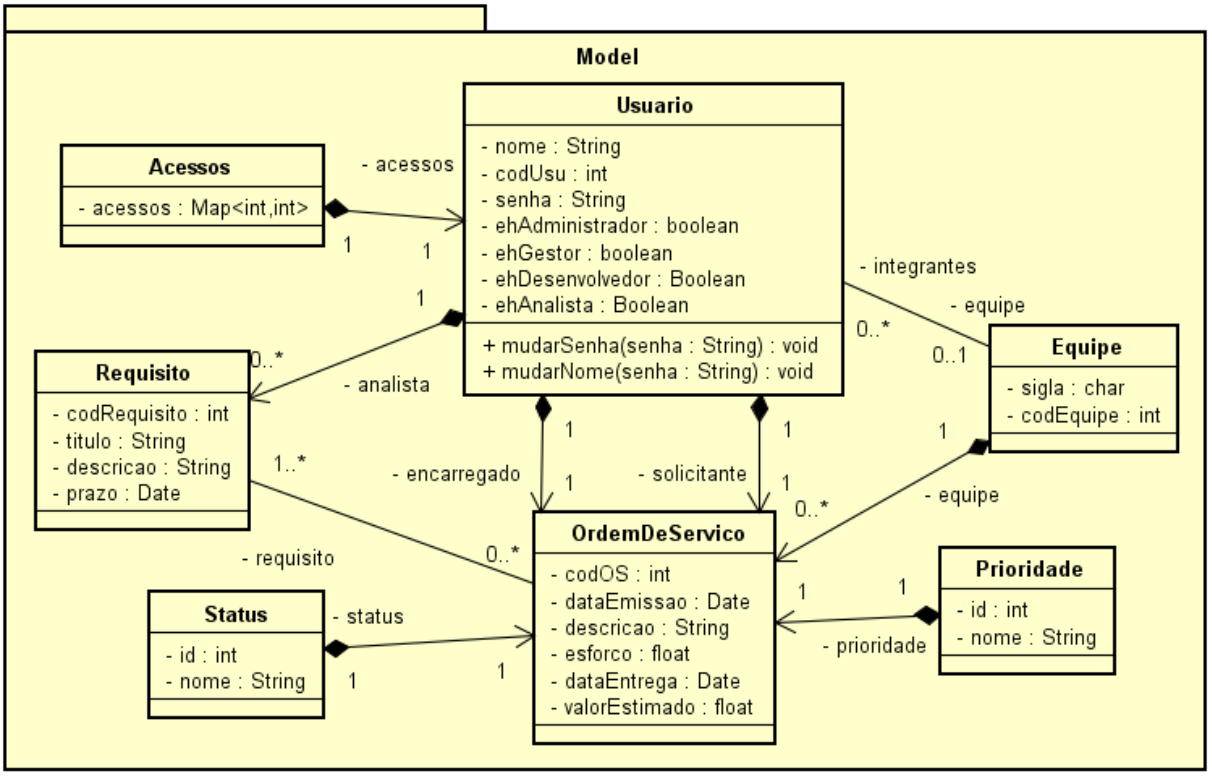


Figura 5 - Diagrama de Classes (Modelo). Fonte: Do autor (2021)

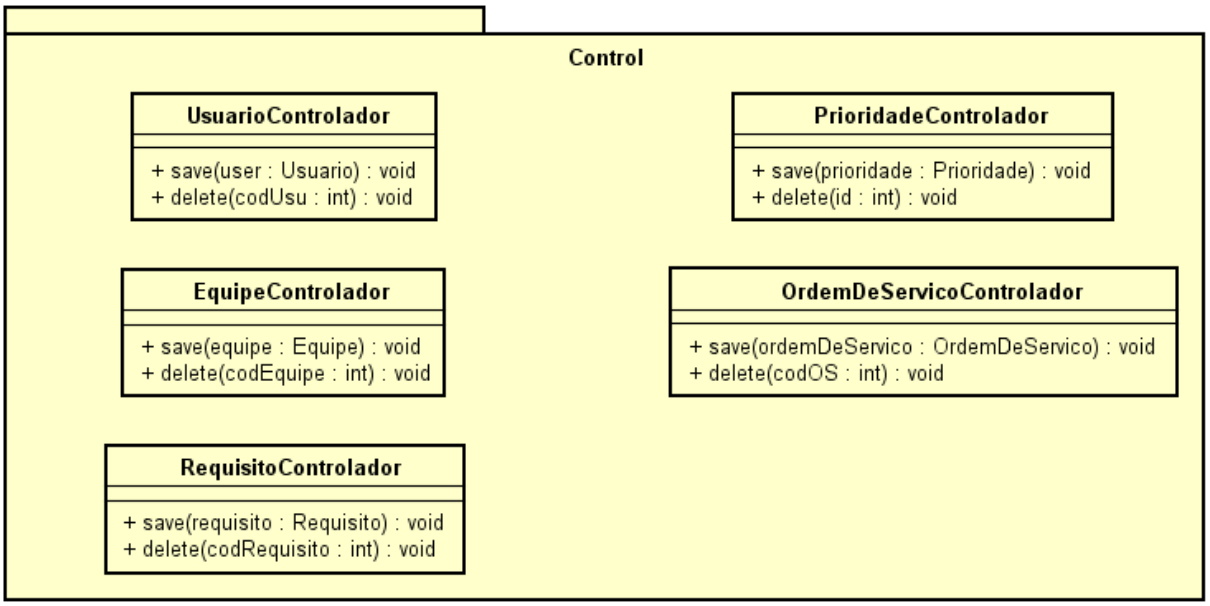


Figura 6 - Diagrama de Classes (Controle). Fonte: Do autor (2021)

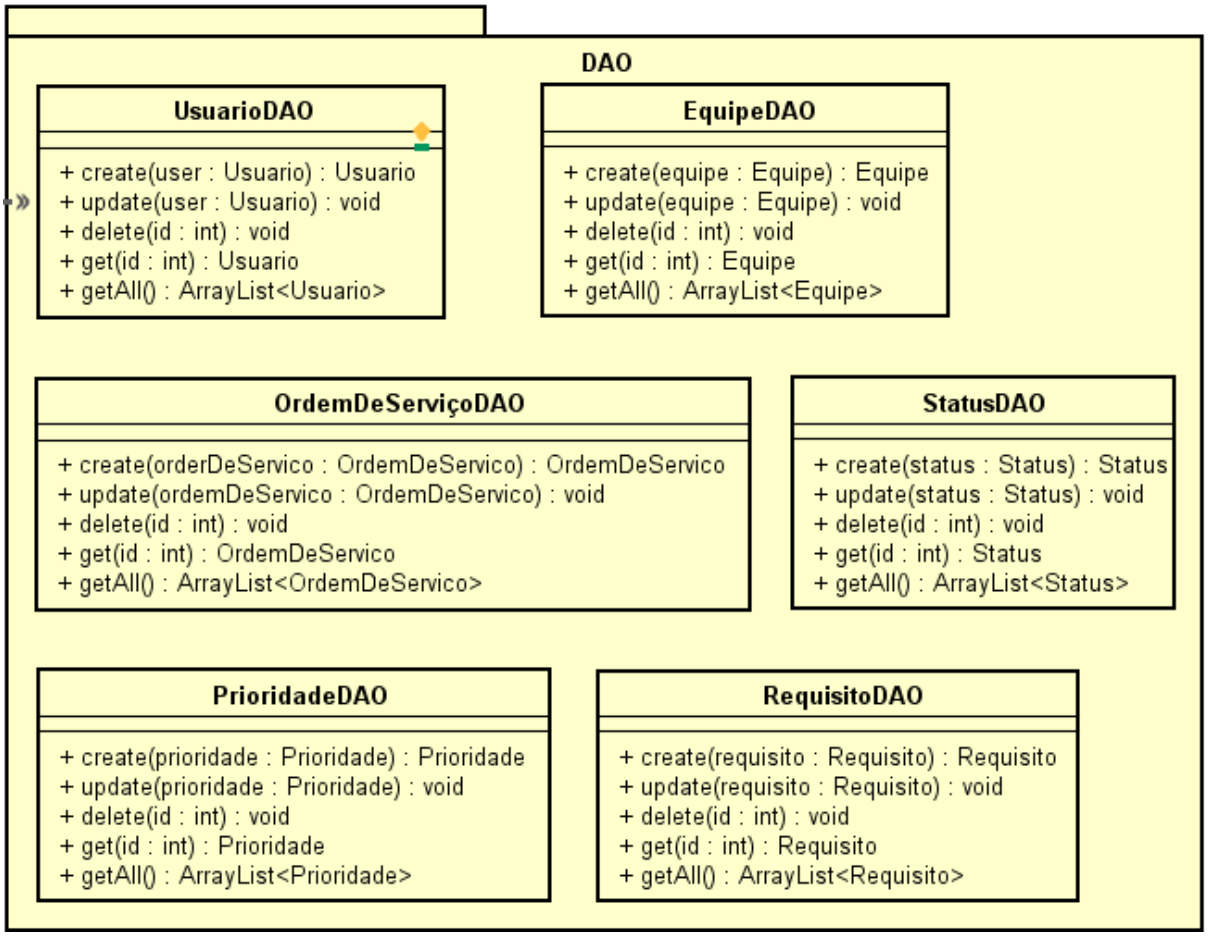


Figura 7 - Diagrama de Classes (DAO). Fonte: Do autor (2021)

4.4.3 Diagrama de Sequência

O diagrama de sequência mostra a troca de mensagens entre diversos objetos, em uma situação específica e delimitada no tempo. Coloca ênfase especial na ordem e nos momentos nos quais mensagens para os objetos são enviadas. Em diagramas de sequência, objetos são representados através de linhas verticais tracejadas (denominadas como linha de existência), com o nome do objeto no topo (CLAIR, 2010).

O Diagrama de sequência mostrado na Figura 8 será utilizado para documentar o fluxo de dados e informação dentro do *software*.

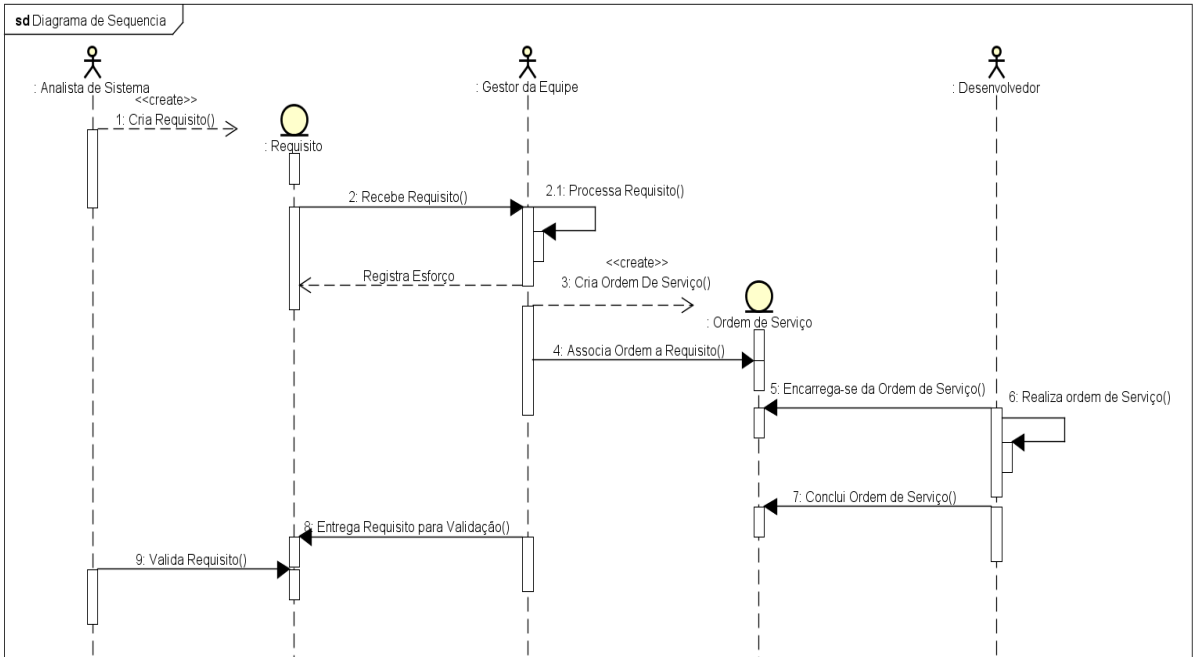


Figura 8 - Sequência de Eventos de Fluxo Principal do Sistema. Fonte: Do autor (2021)

O Fluxo principal mostra a troca de informações no processo de cadastro de Ordens de Serviço no sistema (Figura 3), e apresenta inclusive a dependência deste processo com o de cadastro de Requisitos, uma vez que, necessariamente uma ordem de serviço estará associada a um Requisito do Sistema (Figura 4), um Requisito será um conjunto de uma ou várias Ordens de Serviço registradas.

Os diagramas de sequência das figuras 8, 9, 48, 49 e 50, apresentados no Apêndice II, são amostras dos fluxos relevantes ao funcionamento do sistema, os fluxos referentes aos demais casos de uso apresentam variações similares dos diagramas de sequência apresentados.

O Diagrama de sequência apresentado na Figura 9 mostra o Fluxo de informação no processo de criação de um Usuário no Sistema, referindo-se ao caso de uso de Manter Usuários (Figura 2).

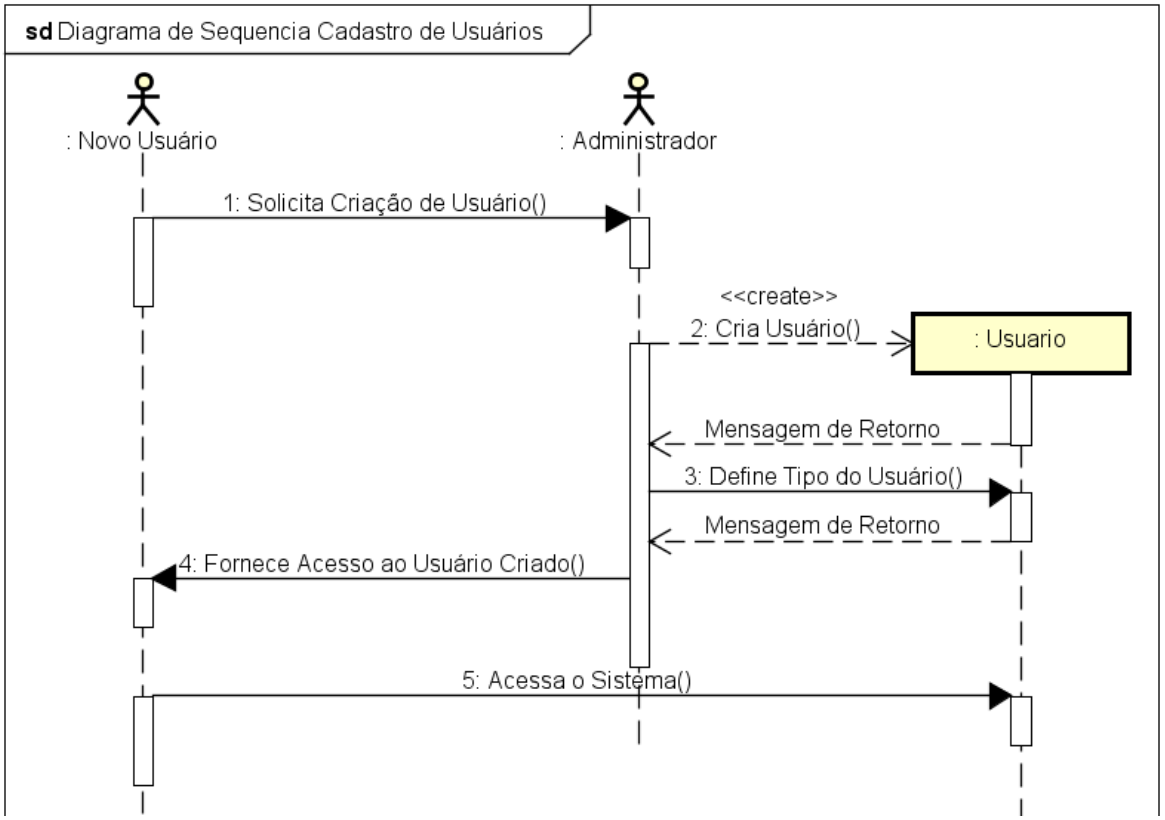


Figura 9 - Sequência de Criação de Usuário no Sistema. Fonte: Do autor (2021)

4.5 Java

Java é uma linguagem de programação orientada a objetos desenvolvida na década de 90 por uma equipe de programadores chefiada por James Gosling, na empresa *Sun Microsystems*. Diferente das linguagens de programação modernas, que são compiladas para código nativo, a linguagem Java é compilada para um *bytecode* que é interpretado por uma máquina virtual (Java *Virtual Machine*, mais conhecida pela sua abreviação *JVM*).

A tecnologia Java será utilizada principalmente para criação da lógica *backend* da aplicação, em que serão atendidas as regras de negócio do módulo de gestão de OS.

Além disso, a aplicação Java *Web* será utilizada como principal agente na comunicação *REST*, viabilizando esta no *Server-Side* do sistema, sobre esta, o *World Wide Web Consortium* define:

> "Transferência Representacional de Estado, é um estilo de arquitetura de *software* que define um conjunto de restrições a serem usadas para a criação de *web services* (serviços *Web*). Os *web services* que estão em conformidade com o estilo arquitetural *REST*, denominados *web services RESTful*, fornecem interoperabilidade entre sistemas de computadores na *Internet*."(*World Wide Web Consortium*, 2004)

4.6 JavaScript

JavaScript é uma linguagem leve, interpretada e baseada em objetos com funções de primeira classe, mais conhecida como a linguagem de *script* para páginas *Web*, mas usada também em vários outros ambientes sem *browser*, tais como *node.js, Apache CouchDB* e *Adobe Acrobat* (*MDN Web Docs* , 2020).

A tecnologia Javascript será utilizada principalmente para elaboração dos componentes de *front-end* e criação de controladores para comunicação entre interface e lógica de *back-end* do *software*.

4.7 Oracle

> "O Oracle é um *SGBD* (sistema gerenciador de banco de dados) que surgiu no fim dos anos 70, [...]
> Além da base de dados, a Oracle desenvolve uma suíte de desenvolvimento chamada de *Oracle Developer Suite*, utilizada na construção de programas de computador que interagem com a sua base de dados. A Oracle também criou a linguagem de programação PL/SQL, utilizada no processamento de transações." (Oracle, 2014).

O Banco de Dados Oracle 18c *Express Edition* será utilizado no servidor do sistema para persistência de dados e disponibilidade destes para a aplicação quando necessário.

A Figura 10 apresenta o modelo relacional do banco de dados do sistema, com todas as Tabelas, Campos, Tipos de Dados e relacionamentos entre as principais entidades do Sistema.

As Ordens de Serviço serão guardadas de forma a conter sempre um status, e podendo ou não conter uma prioridade, sendo dessa forma, a priorização opcional a Ordem de Serviço. As Ordens de Serviço também deverão conter um requisito associado a elas, assim como um usuário solicitante e possivelmente um usuário encarregado, porém o usuário encarregado não será obrigatoriamente atribuído de imediato, podendo este não estar associado no momento da criação de uma Ordem de Serviço.

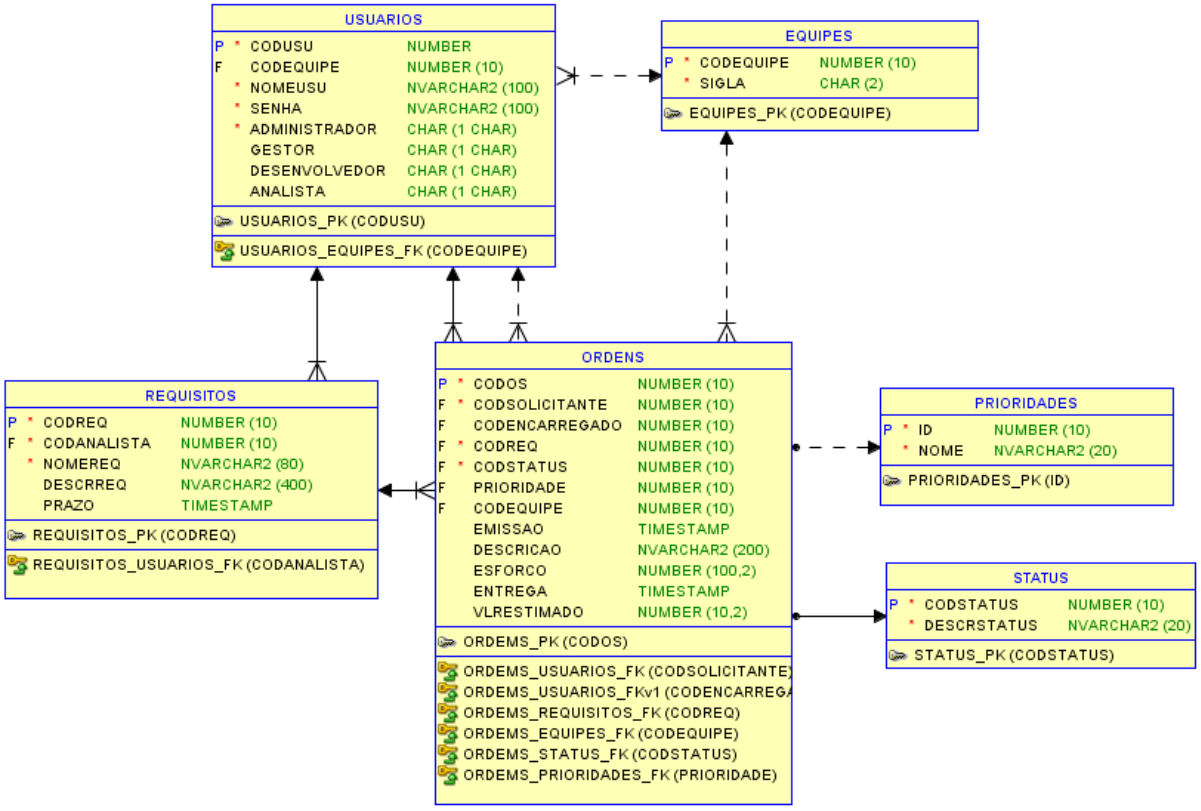


Figura 10 - Modelo Relacional do Banco de Dados Fonte: Do autor (2021)

5. Desenvolvimento

Após a realização da modelagem do sistema, foi realizado o desenvolvimento do mesmo, inicialmente com a criação de uma instância do Banco de Dados Oracle, seguido do desenvolvimento de uma aplicação Java com comunicação *REST* para atender as Requisições destinadas ao *Server-Side* da aplicação e por último implementado as interfaces em JavaScript para serem utilizadas no *Client-Side* via *Browser*.

5.1. Geração da base de dados

O Banco de Dados está sendo acessado pela aplicação do Sistema no mesmo servidor, é manipulado por Objetos de Acesso a Dados (*DAO*) presentes na aplicação, consiste nas seguintes Tabelas:

Coluna	**Tipo**	**Descrição**
CODUSU	NUMBER(10,0)	Código do Usuário
CODEQUIPE	NUMBER(10,0)	Código da Equipe do Usuário
NOMEUSU	NVARCHAR2(100 CHAR)	Nome do Usuário
SENHA	NVARCHAR2(100 CHAR)	Senha do Usuário com criptografia SHA-256
ADMINISTRADOR	CHAR(1 CHAR)	Marcação se Usuário é

		Administrador
GESTOR	CHAR(1 CHAR)	Marcação se Usuário é Gestor
DESENVOLVEDOR	CHAR(1 CHAR)	Marcação se Usuário é Desenvolvedor
ANALISTA	CHAR(1 CHAR)	Marcação se Usuário é Analista

Tabela 2 - Tabela dos Usuários. Fonte: Do autor (2021)

Coluna	**Tipo**	**Descrição**
CODREQ	NUMBER(10,0)	Código do Requisito
CODANALISTA	NUMBER(10,0)	Código do Analista Encarregado
NOMEREQ	NVARCHAR2(80 CHAR)	Título do Requisito
DESCRREQ	NVARCHAR2(400 CHAR)	Descrição Textual do Requisito
PRAZO	TIMESTAMP(6)	Prazo para entrega do Requisito

Tabela 3 - Tabela dos Requisitos. Fonte: Do autor (2021)

Coluna	Tipo	Descrição
CODOS	NUMBER(10,0)	Código da Ordem de Serviço
CODSOLICITANTE	NUMBER(10,0)	Código do Solicitante da Ordem de Serviço
CODENCARREGADO	NUMBER(10,0)	Código do Executor da Ordem de Serviço
CODREQ	NUMBER(10,0)	Código do Requisito a que a Ordem de Serviço se Refere
CODSTATUS	NUMBER(10,0)	Código do Status do Andamento da Ordem de Serviço
PRIORIDADE	NUMBER(10,0)	Código da Prioridade de Realização da Ordem de Serviço
CODEQUIPE	NUMBER(10,0)	Código da Equipe Responsável pela Ordem de Serviço
EMISSAO	TIMESTAMP(6)	Data da Emissão da Ordem de Serviço
DESCRICAO	NVARCHAR2(200 CHAR)	Descrição do Serviço a ser Realizado
ESFORCO	NUMBER(30,2)	Esforço Estimado para Realização do Serviço

ENTREGA	TIMESTAMP(6)	Data da Conclusão da Ordem de Serviço
VLRESTIMADO	NUMBER(10,2)	Valor Estimado a ser cobrado pelo serviço

Tabela 4 - Tabela de Ordens de Serviço. Fonte: Do autor (2021)

Coluna	**Tipo**	**Descrição**
CODEQUIPE	NUMBER(10,0)	Código da Equipe
SIGLA	CHAR(10 BYTE)	Sigla da Equipe

Tabela 5 - Tabela de Equipes. Fonte: Do autor (2021)

Coluna	**Tipo**	**Descrição**
ID	NUMBER(10,0)	Código da Prioridade
NOME	NVARCHAR2(20 CHAR)	Nome da Prioridade

Tabela 6 - Tabela de Prioridades. Fonte: Do autor (2021)

Coluna	**Tipo**	**Descrição**
CODSTATUS	NUMBER(10,0)	Código do Status
DESCRSTATUS	NVARCHAR2(20 CHAR)	Descrição do Status

Tabela 7 - Tabela de Status. Fonte: Do autor (2021)

5.1.1 Conexão com Banco de Dados

A conexão da aplicação com o banco de dados é realizada por uma entidade de conexão presente no projeto java, ele contém um pacote de entidades onde todas as entidades de conexão com banco de dados implementam interface da figura 11.

```
public interface IConnector {
    public Connection connect();
    public void disconect();
}
```

Figura 11 - Interface de Conexão com Banco de Dados Fonte: Do autor (2021)

A Figura 12 mostra um exemplo de implementação desta interface pela entidade de conexão ao Banco de Dados Oracle.

```
@Override
public Connection connect() {
    String url;
    String user;
    String pass;
    if (null == this.getConn()) {
        try {
            Properties prop = ManipuladorDePropriedades.getProp();
            url = prop.getProperty("dburl");
            user = prop.getProperty("user");
            pass = prop.getProperty("pass");
            this.conn = DriverManager.getConnection(url, user, pass);
        } catch (SQLException e) {
            System.out.println("SQL Error! Alert: ");
            e.printStackTrace();
        }
    }
    return this.conn;
}
```

Figura 12 - Método de Conexão com Banco de Dados Oracle Fonte: Do autor (2021)

Uma parte importante do componente de conexão ao banco de dados é que ele utiliza o padrão de projeto *Singleton* para garantir que apenas uma instância de conexão esteja aberta a todo o momento, segundo Gamma *et al*.:

> Em engenharia de *software*, o padrão *singleton* é um padrão de projeto de *software* que restringe a instância de uma classe a uma instância "única". Isso é útil quando exatamente um objeto é necessário para coordenar ações em todo o sistema. O termo vem do conceito matemático de um *singleton*. (GAMMA, Erich *et al*., 1994)

```
private OracleConnector() {
}

public static OracleConnector getInstance() {
    return NewSingletonHolder.INSTANCE;
}

private static class NewSingletonHolder {

    private static final OracleConnector INSTANCE = new OracleConnector();
}
```

Figura 13 - Implementação Singleton na Conexão Oracle Fonte: Do autor (2021)

A figura 13 mostra a implementação do padrão *Singleton* no componente de conexão oracle.

Também é importante verificar que o componente de conexão ao Banco de Dados retira os dados da conexão de um arquivo *.properties,* responsável por manter esses valores e permitindo sua edição, de forma a garantir maior configurabilidade ao sistema, a estrutura do arquivo é mostrada na Figura 14.

```
dburl=jdbc:oracle:thin:@localhost:1521:xe
user=Felps
pass=tecsis
```

Figura 14 - Estrutura do arquivo de dados da conexão Fonte: Do autor (2021)

5.2. Aplicação Java *Web*

A aplicação Java foi desenvolvida para atender não somente a comunicação e manipulação do banco de dados, mas para atuar como uma Aplicação *Web* de comunicação com o *Client-Side* da aplicação (*Browser*), de forma a ter entidades para controle (*Controler*), manter as entidades de Modelo (*Model*) que representam as entidades persistidas em Banco de Dados e mantém as entidades de Regra de Negócio (*Business*). Os Pacotes Estruturais da Aplicação Java são mostrados na Tabela 8.

Pacote	**Função**
api	Contém todas as entidades Básicas para comunicação *REST* com o *Client-Side* da aplicação, com entidades para montar, receber e enviar Requisições *HTTP* dos tipos *GET, POST, OPTION, PUT, DELETE*, além de enviar objetos e entidades em formato *JSON* na requisição quando necessário.
control	Contém, entidades para a manipulação de objetos dentro da aplicação *Web* quando necessário.
dao	Contém as entidades de Persistência em Banco de Dados, segue a estrutura de conter uma interface de modelo de *DAO*, que são implementadas por todas as demais entidades deste pacote, garantindo que todas terão minimamente os métodos de: ● *delete*: deleta um registro do Banco de Dados. ● *get*: seleciona um registro do Banco de Dados pelo id ● *getAll*: seleciona todos os registros de uma determinada entidade.

	• *save*: Insere um novo registro de uma entidade. • *update*: modifica um registro já existente de uma entidade.
erros	Contém as diversas entidades para tratativa de possíveis erros ocorridos na aplicação (*Handlers*).
main	Contém a Entidade principal da Aplicação (*Main*), que inicia todos os objetos necessários para execução da Aplicação *Web*.
model	Contém os modelos de Entidade padrão do sistema, como Usuários, Requisitos e Ordens de Serviço.
repository	Contém um repositório de arquivos *JSON* das unidades e entidades, além de manter repositório em memória de alguns objetos, com objetivo de realizar *cache* das requisições mais recentes e tornar o tempo de resposta da Requisição *Web* menor.
service	Contém entidades de Regra de negócio para comunicação com a *API* da Aplicação *Web* e padrões para reconhecimento das requisições recebidas.
SQL *Utils*	Contém as entidades para conexão com o banco de dados, com entidades para conexão com banco de dados Oracle, SQLite e mySQL.
utils	Contém entidades utilitárias para manipulação de Arquivos em geral, arquivos *JSON* e Arquivos de Propriedades (.*properties*).

Tabela 8 - Estrutura do Projeto Java. Fonte: Do autor (2021)

5.2.1 Objetos de Acesso a Dados

Os Objetos de Acesso a Dados do sistema são utilizados principalmente para acionar as operações SQL básicas de *SELECT, INSERT, UPDATE, DELETE* para os

dados guardados de cada entidade existente no sistema, cada entidade tem a sua própria entidade de *DAO* e todas implementam a interface mostrada na figura 15.

```
public interface IDAO<T> {

    public T get(int id) throws SQLException;

    public T get(String str) throws SQLException;

    public void save(T obj) throws SQLException;

    public void update(T obj) throws SQLException;

    public void delete(int id) throws SQLException;

    public ArrayList<T> getAll() throws SQLException;
}
```

Figura 15 - Interface de entidade *DAO* Fonte: Do autor (2021)

Somente as operações básicas são acionadas para que a única responsabilidade do Banco de Dados seja guardar e fornecer os dados, toda a manipulação e lógica sobre os dados guardados ocorre dentro da aplicação do sistema, com a exceção de criação de chaves primárias para identificação dos registros. Todas as entidades de *DAO* ao implementar os métodos da interface retorna um objeto de seu tipo em métodos de consulta e recebe um objeto de seu tipo em métodos de persistência. A responsabilidade de tratar as possíveis exceções e defeitos de comunicação com o banco de dados ficou para os objetos que utilizarem estes métodos, uma vez que em suas estruturas internas estes métodos acionam o banco de dados múltiplas vezes.

5.2.2 Comunicação *REST*

A comunicação *Rest* é realizada e tratada pelas entidades dos pacotes *repository*, *service* e *api*; o pacote *repository* contém um subpacote *json* onde ficam as entidades modelo para requisições de criação de novas entidades no sistema.

A figura 16 mostra a entidade de modelo de uma nova ordem de serviço no sistema.

```
@Value
@Builder
public class NovaOrdem {
    Usuario solicitante;
    Usuario encarregado;
    Requisito requisito;
    Status status;
    Prioridade prioridade;
    Equipe equipe;
    Timestamp emissao;
    String descricao;
    BigDecimal esforco;
    Timestamp entrega;
    BigDecimal vlrEstimado;
}
```

Figura 16 - Entidade de Nova Ordem de Serviço Fonte: Do autor (2021)

Uma característica importante desta entidade é que ela não contém o atributo de identificação (ID) da ordem de serviço, pois este só passará a existir quando a entidade for persistida no Banco de Dados, que é responsável por gerar as chaves de identidade de cada tipo de entidade do sistema, esta abordagem foi escolhida para evitar situações em que a chave de identificação no Banco de Dados seja diferente da identificação da entidade na aplicação.

O pacote *repository* mantém uma lista em memória para ser acionada de forma primária pela aplicação, de forma a diminuir o tempo de resposta das requisições de consulta a entidades, o *script* de criação da lista de repositório de ordens de serviço é mostrado na figura 17.

```
public static final Map<BigDecimal, Ordem> ORDENS_STORE = new ConcurrentHashMap<BigDecimal, Ordem>();
```

Figura 17 - Criação de Repositório em Memória. Fonte: Do autor (2021)

O pacote *service* contém entidades de controle, responsáveis por acionar os métodos de guardar e atualizar os registros, tanto em banco de dados quanto em memória. A figura 18 mostra o serviço de registro de uma nova ordem de serviço.

```
@Override
public int create(NovaOrdem novaOrdem) {
    Ordem ordem = new Ordem(
             novaOrdem.getSolicitante()
            ,novaOrdem.getEncarregado()
            ,novaOrdem.getRequisito()
            ,novaOrdem.getStatus()
            ,novaOrdem.getPrioridade()
            ,novaOrdem.getEquipe()
            ,novaOrdem.getEmissao()
            ,novaOrdem.getDescricao()
            ,novaOrdem.getEsforco()
            ,novaOrdem.getEntrega()
            ,novaOrdem.getVlrEstimado()
            );
    try {
        OrdemDao.getInstance().save(ordem);
    } catch (SQLException e) {
        System.out.println("Error! Alert: ");
        e.printStackTrace();
    }

    ORDENS_STORE.put(new BigDecimal(ordem.getCodOs()), ordem);

    return ordem.getCodOs();
}
```

Figura 18 - Serviço de registro de uma Ordem de Serviço. Fonte: Do autor (2021)

O sistema contém no pacote *service* um entidade de controle para cada uma das entidades existentes no sistema. Além do método de registrar uma nova entidade, contém um método similar para atualizar a entidade em banco de dados e memória.

O pacote *api* contém as entidades responsáveis por realizar o processamento das requisições recebidas e montar as entidades de resposta de acordo com as regras de comunicação estabelecidas para cada método de comunicação *REST* a figura 19 mostra o método padrão para interpretação das requisições recebidas.

```
@Override
protected void execute(HttpExchange exchange) throws IOException {
    byte[] response;
    if ("POST".equals(exchange.getRequestMethod())) {
        @SuppressWarnings("rawtypes")
        ResponseEntity e = doPost(exchange.getRequestBody());
        exchange.getResponseHeaders().putAll(e.getHeaders());
        exchange.getResponseHeaders()
        .add(Constants.ACESS_CONTROL_ALLOW_ORIGIN, Constants.ASTERISC);
        exchange.sendResponseHeaders(e.getStatusCode().getCode(), 0);
        response = super.writeResponse(e.getBody());
    } else if("PUT".equals(exchange.getRequestMethod())) {
        @SuppressWarnings("rawtypes")
        ResponseEntity e = doPut(exchange.getRequestBody());
        exchange.getResponseHeaders()
        .add(Constants.ACESS_CONTROL_ALLOW_ORIGIN, Constants.ASTERISC);
        exchange.sendResponseHeaders(e.getStatusCode().getCode(), 0);
        response = super.writeResponse(e.getBody());
    }else if("OPTIONS".equals(exchange.getRequestMethod())) {
```

Figura 19 - Método de interpretação de uma requisição *REST*. Fonte: Do autor (2021)

De acordo com o método presente no cabeçalho da requisição, o método de interpretação acionará outro método, responsável por efetuar a ação pré estabelecida pelo tipo de requisição e montar o corpo da resposta de retorno, a figura 20 mostra o método de tratativa para a requisição do tipo *PUT* de uma ordem de serviço.

```
private ResponseEntity<OrdemUpdateResponse> doPut(InputStream is) {
    OrdemUpdateRequest updateRequest;
    updateRequest = super.readRequest(is, OrdemUpdateRequest.class);

    Ordem ordem = new Ordem(
              updateRequest.getCodOs()
            , updateRequest.getSolicitante().getUsuario()
            , updateRequest.getEncarregado().getUsuario()
            , updateRequest.getRequisito().getRequisito()
            , updateRequest.getStatus().getStatus()
            , updateRequest.getPrioridade().getPrioridade()
            , updateRequest.getEquipe().getEquipe()
            , updateRequest.getEmissao()
            , updateRequest.getDescricao()
            , updateRequest.getEsforco()
            , updateRequest.getEntrega()
            , updateRequest.getVlrEstimado()
            );

    ordemService.update(ordem);

    return new ResponseEntity<>(null,
             getHeaders(Constants.ACESS_CONTROL_ALLOW_HEADERS
                     , Constants.HEADERS), StatusCode.OK);
}
```

Figura 20 - Método de interpretação de uma requisição do tipo *PUT*. Fonte: Do autor (2021)

Todos os métodos de interpretação da requisição fazem uma verificação se a entidade que veio no corpo da requisição é compatível com a entidade esperada, e retornam uma entidade de Resposta de acordo com a regra estabelecida, a tabela 9 contém uma lista das ações tomadas para cada tipo de requisição *REST*.

Tipo de Requisição	**Tratativa Estabelecida**
GET	Verifica o ID que deve vir no cabeçalho da requisição e retorna a entidade montada a partir do registro do ID recebido.
POST	Recebe uma entidade para ser cadastrada, solicita o cadastro e retorna a entidade cadastrada, contendo o ID gerado pelo banco de dados.

PUT	Recebe uma entidade a ser atualizada no sistema, solicita a atualização desta, retorna uma entidade resposta sem corpo com status *OK*.
DELETE	Recebe o ID de uma entidade a ser deletada do sistema, solicita a eliminação do registro e retorna uma entidade de resposta sem corpo com status *OK*.
OPTIONS	Retorna uma série de cabeçalhos na entidade de Resposta, exigidos pelos protocolos de segurança dos navegadores para comunicação *REST*.

Tabela 9 - Tratativas Estabelecidas para Requisições REST por tipo. Fonte: Do autor (2021)

A entidade principal da aplicação do sistema é responsável por instanciar os contextos do servidor, associando as entidades do pacote de *api* a seus respectivos contextos, desta forma, a comunicação com o *handler* de cada entidade é por meio de seu contexto específico, a figura 21 apresenta todos os contextos instanciados pelo método principal.

```
public static void main(String[] args) throws IOException {

    server = HttpServer.create(
            new InetSocketAddress(
                    Constants.SERVER_PORT), 0);

    UsuarioHandler usuarioHandler = new UsuarioHandler(getObjectMapper(),
            getErrorHandler());
    server.createContext("/api/users", usuarioHandler::handle);

    StatusHandler statusHandler = new StatusHandler(getObjectMapper(),
            getErrorHandler());
    server.createContext("/api/status", statusHandler::handle);

    PrioridadeHandler prioridadeHandler = new PrioridadeHandler(getObjectMapper(),
            getErrorHandler());
    server.createContext("/api/prioridade", prioridadeHandler::handle);

    RequisitoHandler requisitoHandler = new RequisitoHandler(getObjectMapper(),
            getErrorHandler());
    server.createContext("/api/requisito", requisitoHandler::handle);

    EquipeHandler equipeHandler = new EquipeHandler(getObjectMapper(),
            getErrorHandler());
    server.createContext("/api/equips", equipeHandler::handle);

    OrdemHandler ordemHandler = new OrdemHandler(getObjectMapper(),
            getErrorHandler());
    server.createContext("/api/ordem", ordemHandler::handle);
```

Figura 21 - Método principal instanciando contextos da *api*. Fonte: Do autor (2021)

Ao ser iniciado o servidor Java, os controladores em Javascript são responsáveis por realizar a comunicação com a *api*.

5.3. Criação da Aplicação JavaScript

Foi desenvolvido uma aplicação *Web Browser* em JavaScript utilizando o *framework* Node.js, a interface foi desenvolvida com o conceito de Aplicação de Página Única (*SPA*) na qual apenas uma página *Web* (*HTML*) é criada e seu código interno é modificado em tempo real de acordo com as iterações do Usuário.

Inicialmente, o usuário deverá Fornecer suas Credenciais na Tela de *Login* (Figura 22) para autenticação do Login antes de haver um usuário criado, o sistema contém um usuário administrador padrão a ser utilizado conforme demonstrado no caso de uso 001 do Apêndice I.

Figura 22 - Tela de *Login*. Fonte: Do autor (2021)

5.3.1 Comunicação com o Servidor

A Comunicação com o Servidor é realizada por um componente em javascript, responsável por montar os cabeçalhos e enviar para o servidor, contendo

em sua url, o contexto específico a ser acessado, a figura 23 mostra o método utilizado para montar uma requisição do tipo *GET*.

```
class HttpClientLogin {

    constructor() {}

    getSinc = (theUrl) => {
        var xmlHttp = new XMLHttpRequest();
        // false para requsição síncrona
        xmlHttp.open("GET", theUrl, false);
        xmlHttp.send(null);
        return xmlHttp.responseText;
    }
```

Figura 23 - Método para envio de requisições get ao servidor. Fonte: Do autor (2021)

Quando é necessário enviar uma requisição com corpo para o servidor, o mesmo componente de comunicação envia a entidade necessária na requisição em formato *JSON*, conforme demonstrado na figura 24.

```
autenticate = (theUrl, jsonString, callback) => {
    var json = JSON.stringify(jsonString);

    var xmlHttp = new XMLHttpRequest();
    //verdadeiro para requisições assícronas.
    xmlHttp.open("POST", theUrl, true);
    xmlHttp.setRequestHeader("Content-Type"
    , "application/json; charset=utf-8");
    xmlHttp.onreadystatechange = () => {
        if (xmlHttp.readyState === 4 && xmlHttp.status === 200) {
            let obj = JSON.parse(xmlHttp.response);
            if (obj.response) {
                window.location.replace(obj.url);
            }
        }
        if (xmlHttp.readyState === 4 && xmlHttp.status === 401) {
            alert("Login/Senha Inválidos!");
        }
    };
    xmlHttp.send(json);
}
```

Figura 24 - Método de solicitar autenticação de *Login*. Fonte: Do autor (2021)

A figura 24 mostra o método específico criado para solicitar autenticação de login de um usuário dentro do sistema, para garantir a segurança e integridade desta comunicação, a autenticação é realizada pelo servidor, e o objeto de acesso ao sistema só é fornecido ao *Browser* em caso de autenticação positiva, conforme mostrado na figura 25.

```
private ResponseEntity<LoginResponse> doPost(InputStream is) {
    LoginRequest registerRequest = super.readRequest(is, LoginRequest.class);
    Boolean resp = Boolean.FALSE;
    String url = null;
    int id = 0;
    try {
        ArrayList<Usuario> usuarios = UsuarioDao.getInstance().getAll();
        for(Usuario user : usuarios) {
            if(user.getNome().toUpperCase()
                    .equals(registerRequest.getLogin().toUpperCase())
                    && user.getPass().equals(registerRequest.getPassword())
                    ) {
                resp = Boolean.TRUE;
                url = Constants.URL_IP+":"+
                Integer.toString(Constants.SERVER_PORT) +"/login/"
                        + Integer.toString(user.getId());
                id = user.getId();
            }
        }
    } catch (SQLException e) {
        System.out.println("Alert!! Error: ");
        e.printStackTrace();
    }
    LoginResponse response = LoginResponse.builder()
            .response(resp).url(url).id(id).build();
    if(resp) {
    return new ResponseEntity<>(response,
        getHeaders(Constants.CONTENT_TYPE
                , Constants.APPLICATION_JSON)
        , StatusCode.OK);
    }else {
        return new ResponseEntity<>(response,
                getHeaders(Constants.CONTENT_TYPE
                        , Constants.APPLICATION_JSON)
                , StatusCode.UNAUTHORIZED);
    }
}
```

Figura 25 - Método para autenticação de login. Fonte: Do autor (2021)

5.3.2 Navegação

Após efetuado o *login* e autenticado com sucesso, o usuário terá acesso a interface principal, onde toda a navegação é efetuada em uma única tela, modificada de acordo com suas iterações no menu lateral, mostrado na figura 26.

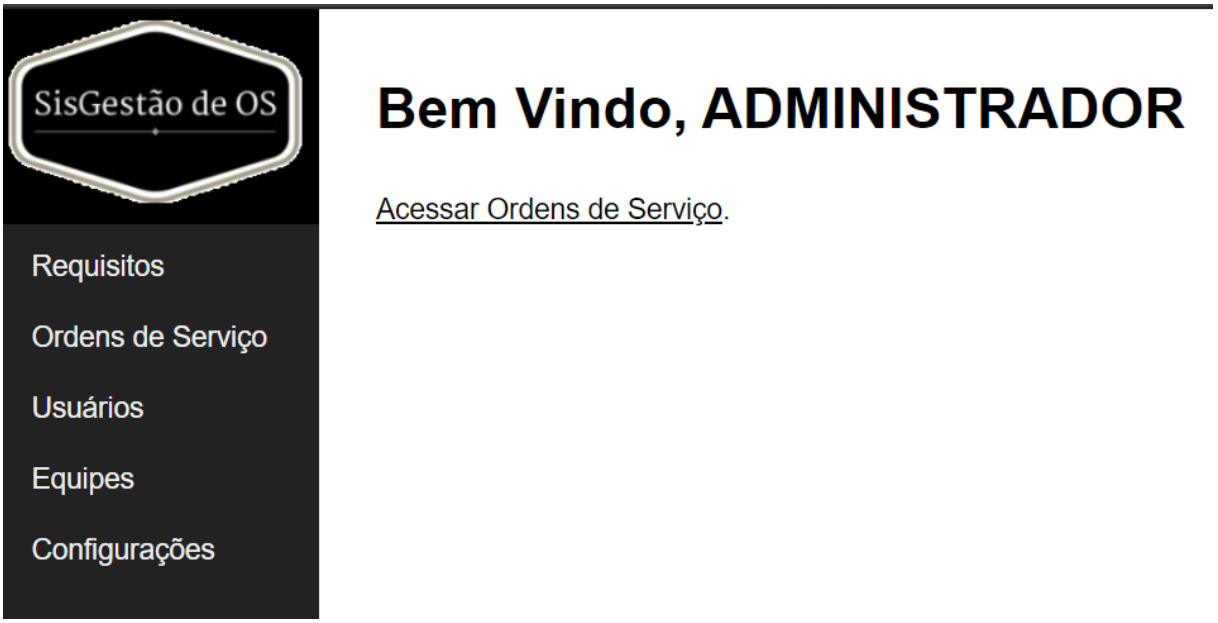


Figura 26 - Menu do Sistema. Fonte: Do autor (2021)

Para controlar a Navegação do Usuário pelo sistema, foi criada uma função roteadora de navegação, que identifica a rota acessada pelo usuário no menu lateral e substitui a visualização principal de acordo com o contexto acessado, conforme mostrado nas figura 27 e 28.

```
const router = async() => {
    const routes = [
        { path: "/", view: BemVindo },
        { path: "/login/:id", view: BemVindo },
        { path: "/users", view: Users },
        { path: "/ordens", view: Ordens },
        { path: "/equips", view: Equipes },
        { path: "/requisitos", view: Requisitos },
        { path: "/equips/cad", view: Equipes },
        { path: "/users/cad", view: UserCad },
        { path: "/requisitos/cad", view: Requisitos },
```

Figura 27 - Rotas. Fonte: Do autor (2021)

```
// Testa cada Rota por matchs potenciais
const potentialMatches = routes.map(route => {
    return {
        route: route,
        result: location.pathname.match(pathToRegex(route.path))
    };
});

let match = potentialMatches.find(potentialMatch => potentialMatch.result !== null);

const view = new match.route.view(getParams(match));

document.querySelector("#app").innerHTML = await view.getHtml();
```

Figura 28 - Método de cálculo de Rotas. Fonte: Do autor (2021)

Todas as entidades de tela possíveis são variações polimórficas de uma entidade de tela genérica a ser utilizada pelo algoritmo de roteamento, onde a tela implementada varia de acordo com a rota escolhida pelo usuário.

Um exemplo de navegação é o acesso aos demais Usuários cadastrados no Sistema, no caso de acesso como um Usuário administrador, a figura 29 mostra o menu de usuários.

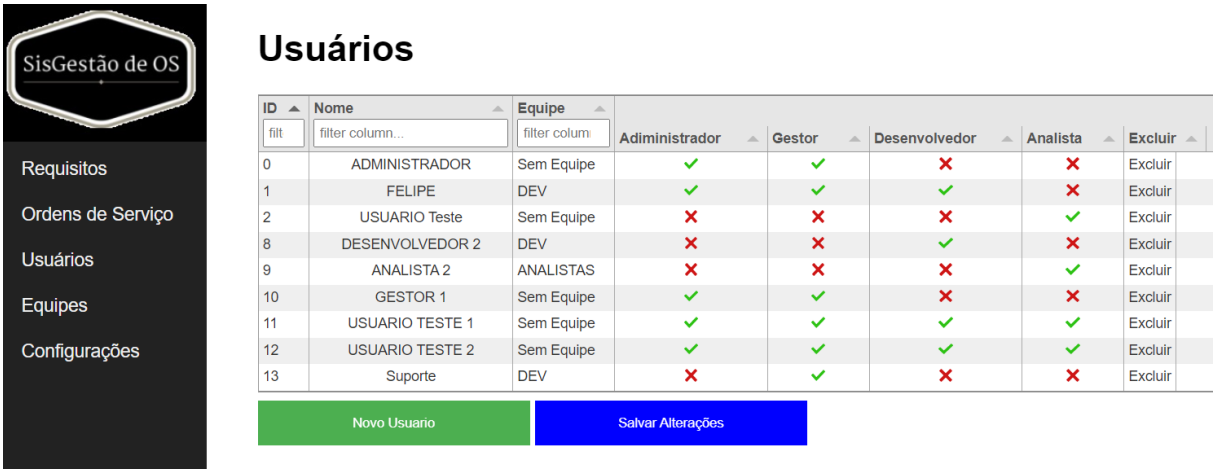


Figura 29 - Menu de Usuários. Fonte: Do autor (2021)

O Menu de Usuários apresenta uma visão geral dos usuários cadastrados, assim como permite a manipulação destes, por meio de modificações na tabela.

A tabela presente na interface de Usuários e nas demais interfaces principais do sistema, foi montada utilizando uma biblioteca javascript própria para a construção de tabelas instantâneas, essa abordagem foi utilizada para permitir a manipulação de dados de forma dinâmica pelo usuário, a figura 30 mostra um exemplo da construção de uma tabela de Usuários de uma Equipe por meio da biblioteca *Tabulator*.

```
var table = new Tabulator("#equipeUsersTable", {
    data: tabledata,
    layout: "fitData",
    columns: [{
            title: "ID",
            field: "id",
            sorter: "number",
            width: 50,
            headerFilter: "number",
            editor: false,
            cellClick: function(e, cell) {
                window.location.replace(urlRaiz + "/users/" + cell.getValue());
            },
        },
        { title: "Nome",
          field: "nome",
          sorter: "string",
          headerFilter: "input",
          hozAlign: "center",
          width: 200,
          editor: false
        }
    ],
    initialSort: [{ column: "id", dir: "asc" }]

});

return table;
```

Figura 30 - Criação de uma tabela dinâmica de Interface. Fonte: Do autor (2021)

Ao clicar no ID de um usuário, é possível ver as informações do mesmo (Figura 31), e editá-las, também é possível editar informações diretamente na tabela de Usuários, além de excluí-los clicando em excluir pela tabela.

Figura 31 - Editando Permissões de Usuário. Fonte: Do autor (2021)

Ao clicar no botão “Salvar Alterações” os dados preenchidos serão salvos, e o botão "Excluir" aciona a exclusão do usuário do ID selecionado.

Ao clicar no botão de Novo Usuário (Figura 29), um formulário é fornecido para o cadastro de um Novo Usuário no sistema (Figura 32).

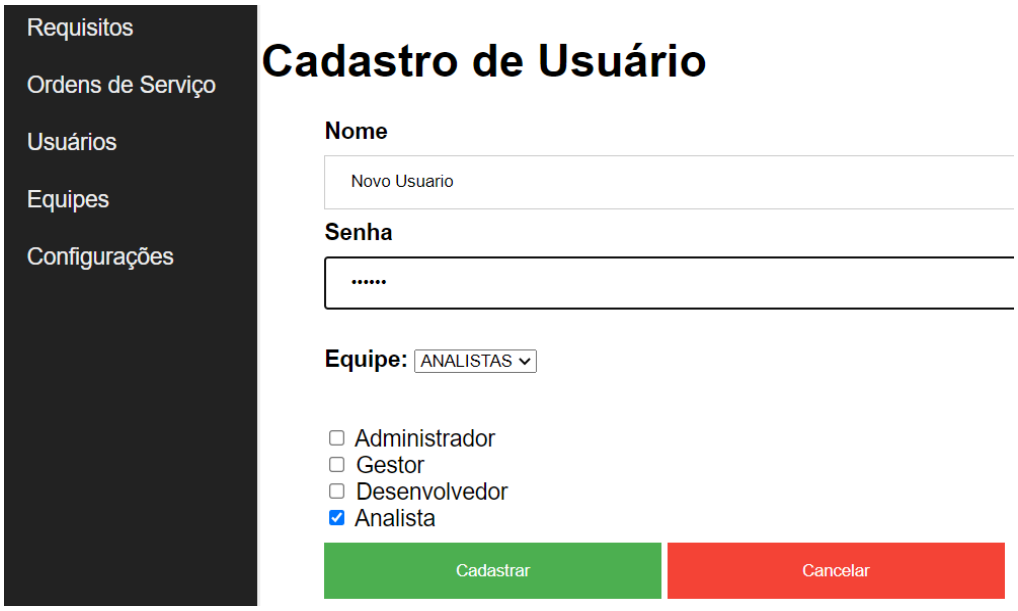


Figura 32 - Cadastro de Usuário. Fonte: Do autor (2021)

Nas interfaces de Menu, Edição e Cadastro de Usuários (Figuras 29, 31, e 32) é possível modificar as informações de nome, equipe a qual o usuário pertence, e definir os papéis do usuário no sistema (Administrador, Gestor, Desenvolvedor e Analista), papéis que estão definidos nos casos de uso das figuras 2, 3 e 4. A mudança de senha é possível somente nas telas de edição e cadastro de usuário.

Assim como o Cadastro de Usuários, os demais cadastros seguem o mesmo padrão, as Figuras 33, 34 e 35 mostram respectivamente o menu de tabela, o formulário de cadastro e a tela de edição de Requisitos.

Figura 33 - Menu de Requisitos. Fonte: Do autor (2021)

Figura 34 - Cadastro de Requisito. Fonte: Do autor (2021)

Figura 35 - Edição de Requisito. Fonte: Do autor (2021)

As telas de cadastro, edição e menu de tabela de requisitos permitem modificar as informações de Título do Requisito, o Analista encarregado daquele requisito e a descrição das informações referentes ao requisito, assim como o prazo para entrega daquele requisito ao cliente, se houver.

As Figuras 36 e 37 mostram as Configurações Adicionais possíveis de serem adicionadas pelo Sistema.

Figura 36 - Configuração (Status) Fonte: Do autor (2021)

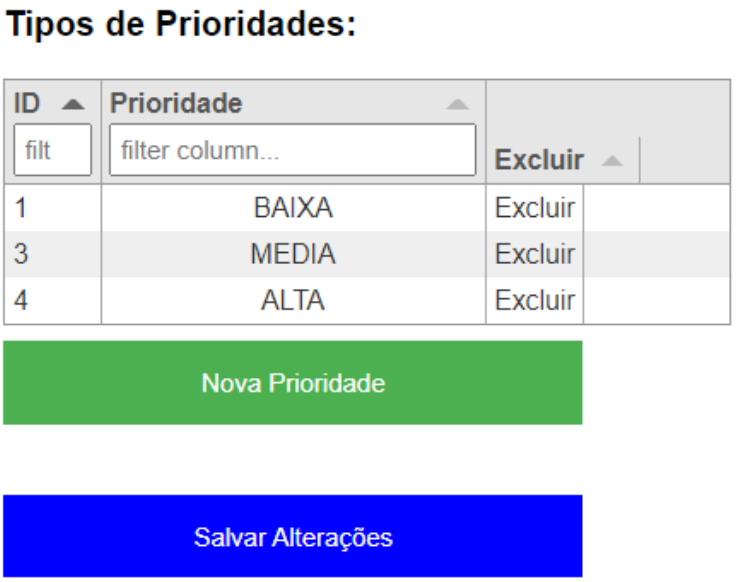


Figura 37 - Configuração (Prioridades) Fonte: Do autor (2021)

A tabela de configuração de status (figura 36) e a tabela de configuração de prioridades (figura 37) fazem parte da mesma tela, nesta, é possível cadastrar prioridades e status novos pelos botões “Novo Status” e “Nova Prioridade” respectivamente; editar as configurações existentes e excluir pela opção de excluir de cada tabela. O botão “Salvar Alterações” guarda as mudanças realizadas.

As figuras 38, 39 e 40 mostram respectivamente, as telas de menu de tabela e cadastro de uma nova ordem de serviço no sistema.

Figura 38 - Menu de Ordens de Serviço. Fonte: Do autor (2021)

Figura 39 - Menu de Ordens de Serviço. Fonte: Do autor (2021)

Figura 40 - Cadastro de uma Ordem de Serviço. Fonte: Do autor (2021)

As Figuras 38 e 39 mostram campos diferentes da mesma tabela de registros de Ordem de Serviço, nesta tela é possível realizar todas as operações de manipulação dos campos de informação de uma ordem de serviço, o botão “Nova Ordem” abre a tela da figura 40, para permitir o cadastro de uma nova ordem de serviço no sistema.

A entidade de Ordem de Serviço, também visível na figura 41 em seu modo de edição, contém as informações de qual usuário solicitou a demanda, preferivelmente será um usuário Gestor/Administrador, porém é possível que outros usuários sejam marcados como solicitantes da Ordem de Serviço, pode ser atribuído também um usuário encarregado da realização da Ordem de Serviço a qualquer momento. É necessário informar um status a Ordem de Serviço, sendo opcional informar uma prioridade, a ordem de serviço pode ser atribuída a uma equipe e deve conter uma descrição do serviço a ser realizado, assim como uma data na qual ela foi emitida, podendo esta ser qualquer data. Também pode ser registrado a qualquer momento o esforço necessário para realização do serviço e qual o valor monetário do serviço, se houver.

Figura 41 - Edição de uma Ordem de Serviço. Fonte: Do autor (2021)

Além de Editar uma Ordem de Serviço, nesta tela é possível anexar um arquivo a ordem de serviço de diferentes tipos (imagens, documentos, vídeos) e efetuar download clicando no nome do arquivo.

As equipes podem ser editadas pelo Menu de equipes, mostrado na figura 42.

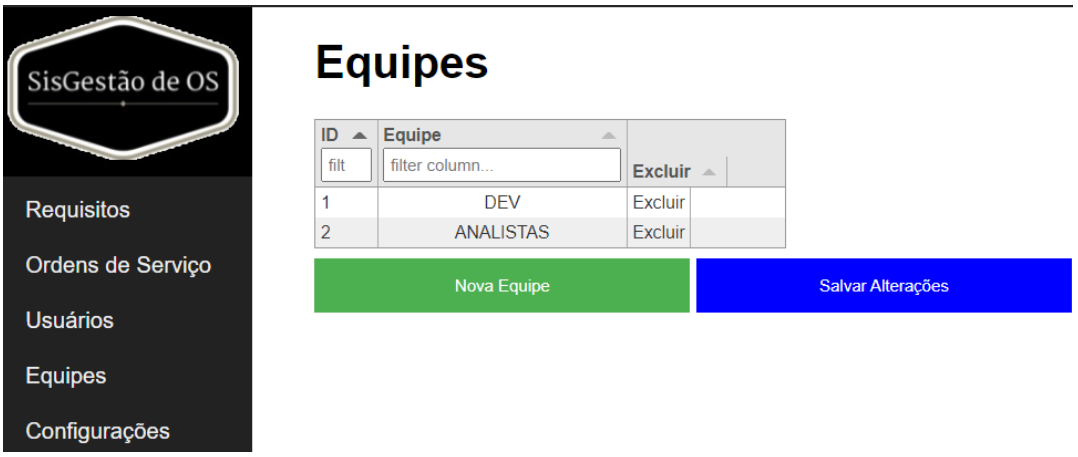


Figura 42 - Menu de Equipes. Fonte: Do autor (2021)

Assim como nas telas anteriores é possível cadastrar equipes novas pelo botão “Nova Equipe”, alterar as equipes existentes e excluir as equipes desejadas pela tabela, ao clicar no id de uma equipe será apresentado o nome da equipe e seus integrantes, conforme mostrado na figura 43.

Figura 43 - Visualização da Equipe. Fonte: Do autor (2021)

Ao clicar no ID de um desenvolvedor, abrirá o painel de edição deste. Em todos os menus de tabela apresentados, é possível filtrar registros em colunas que contenham filtro e ordenar as colunas pelo ordenador da tabela.

6. Testes

Para garantir a qualidade do sistema desenvolvido e seu funcionamento adequado, foram realizados 3 tipos de testes sobre este de forma a cobrir os pontos críticos do projeto, aqueles que eram mais susceptíveis a apresentar problemas:

- Por ser um sistema que envolve comunicação *REST* e acesso a Banco de Dados foi indispensável a realização de Testes de Integração.

- Por ser um sistema distribuído em Módulos referentes a entidades (Requisitos, Ordens de Serviço, Usuários) foram realizados Testes de Componentes para garantir que as entradas, saídas e comportamento esperado de cada módulo estivesse funcionando corretamente.

- Foram realizados Testes Estruturais (Caixa-Branca) durante todo o processo de desenvolvimento, para correção de eventuais problemas de funcionamento encontrados, de forma imediata e interativa.

6.1 Testes de Integração

Os testes de Integração foram separados sobre 3 objetos de Teste: a comunicação entre Banco de Dados e Aplicação, a comunicação Cliente-Servidor pelos métodos de *REST* e a integração inter-modular do sistema, de acordo com a ISO 24765 de 2010, “O teste de integração , abreviado *I&T* é a fase de teste de

software em que os módulos de *software* individuais são combinados e testados como um grupo. [...] é conduzido para avaliar a conformidade de um sistema ou componente com os requisitos funcionais especificados.” (ISO 24765 , 2010)

> “Alguns tipos diferentes de teste de integração são *big bang*, misto (sanduíche), de cima para baixo e de baixo para cima. Outros padrões de integração são: integração de colaboração, integração de *backbone*, integração de camadas, integração *cliente-servidor*, integração de serviços distribuídos e integração de alta frequência” (BINDER, Robert. 1999).

As abordagens utilizadas para teste foram as de Teste de Integração Misto, (de baixo para cima e de cima para baixo), testes de comunicação e compatibilidade Cliente-Servidor.

Defeito	**Teste Realizado**	**Motivo (Erro)**	**Solução**
Exceção lançada a partir de Restrição de Chave Primária da Tabela do Banco de Dados.	Inserção de Registros de Usuário em Tabelas do Banco de Dados utilizando os métodos das Classes de DAO	As chaves primárias não eram incrementais, ao tentar inserir um segundo registro na tabela dispara um conflito de chave primária.	As chaves primárias de todas as tabelas foram reconfiguradas para serem identidade e Auto-Geradas.
Exceção lançada pelo Banco de Dados: escala de precisão inválida.	Inserção de Registros de Ordem de Serviço em Tabelas do Banco de Dados utilizando os métodos das Classes	Havia uma escala de precisão de 10 casas decimais para valores destinados a ser guardados como	Foi retirada a escala de precisão dos campos inteiros do banco de dados.

	de DAO.	números inteiros.	

Tabela 10 - Resultados dos Testes de Integração com Banco de Dados. Fonte: Do autor (2021)

Defeito	**Teste Realizado**	**Motivo (Erro)**	**Solução**
Uma exceção de comunicação do protocolo de cross-origin resource sharing (CORS) era disparada pelo Browser.	Envio de uma requisição HTTP de método GET para a instância da aplicação no servidor.	O protocolo de segurança de comunicação CORS não permite comunicação entre endereços de mesma origem sem a adição de cabeçalhos específicos à requisição.	Um cabeçalho do Tipo ACESS_CONTROL_ALLOW _ORIGIN foi adicionado a todos os métodos HTTP utilizados pelas classes de comunicação REST.
Uma Exceção de comunicação do protocolo HTTP “METHOD_NOT_A LLOWED” era lançada pelo Browser nas requisições de método POST e PUT.	Foi enviado um arquivo JSON de cadastro para o servidor do sistema. Utilizando os Métodos POST e PUT.	Antes de estabelecer comunicação com o servidor de aplicação, o Browser envia uma requisição do tipo OPTIONS para validação da comunicação que precisa ser	Foi criado uma tratativa (Handler) para resolver a comunicação em métodos tipo OPTIONS e criar uma requisição OPTIONS de resposta na eventual solicitação de tipo OPTIONS.

		respondida.	

Tabela 11 - Resultados dos Testes de Integração Cliente-Servidor. Fonte: Do autor (2021)

Defeito	**Teste Realizado**	**Motivo (Erro)**	**Solução**
Falha Geral do Sistema na Tela de Equipes	Exclusão de uma Equipe com usuários cadastrados para ela.	Banco de Dados disparava exceção de restrição de exclusão por Chave estrangeira existente e sistema não se recuperava da Falha.	Criado verificador de registros existentes antes da exclusão de registros inter-dependentes de associação. Mensagem de Restrição de Exclusão é encaminhada ao usuário no front-end.

Tabela 12 - Resultados dos Testes de Integração entre Módulos. Fonte: Do autor (2021)

6.2 Testes de Componente

Os testes de componente foram executados sobre os processos existentes das 3 entidades chave do sistema: Ordens de Serviço, Requisitos e Usuários; Para isto foram fornecidas entradas de cadastro para cada módulo e foi realizado a manipulação de registros destes por todas as interfaces possíveis de manipulação de dados: tabela de registros, telas de edição, telas de cadastro, também foi realizada a exclusão de registros de todas as entidades como teste.

> No teste de componentes, os componentes do *software* são testados separadamente de acordo com a especificação e estrutura das funcionalidades. Estes componentes são as integrações de interface e

> unidades do *software*, com diversas classes no seu desenvolvimento (CARDOSO, 2006).

Com a realização dos testes de componentes, foi constatado a necessidade de criação de 3 tipos de componente para visualização e controle de cada uma das entidades: uma classe para o formulário de cadastro ser exibido, uma para a apresentação em forma de tabela e outra para a edição individual do registro; porém, esta necessidade não se fez presente para 3 entidades não principais: Equipes, Status e Prioridades, pois estas continham uma estrutura simples de registro e foram organizadas de forma a ter somente uma classe para controle/cadastro unificada e outra para apresentação.

6.3 Testes Estruturais

Ao longo do projeto foram realizados diversos testes estruturais para verificar o fluxo dos dados, integridade e segurança destes dentro da estrutura do sistema. Estas verificações, validações e correções foram efetuadas de forma interativa e dinâmica ao decorrer do projeto, para permitir melhor fluidez do desenvolvimento e facilitar na escolha de padrões de estrutura de dados a serem utilizados no projeto.

> O teste de caixa branca (também conhecido como teste de caixa transparente, teste de caixa de vidro, teste de caixa transparente e teste estrutural) verifica as estruturas internas ou funcionamento de um programa, em oposição à funcionalidade exposta ao usuário final (KASSEM, 2009).

“No teste de caixa branca, uma perspectiva interna do sistema (o código-fonte), bem como habilidades de programação, são usadas para projetar casos de teste. O testador escolhe entradas para exercitar caminhos através do código e determinar as saídas apropriadas” (LIMAYE, 2009).

Defeito	**Teste Realizado**	**Motivo (Erro)**	**Solução**
Requisito não era guardado pelo sistema, suas informações de prazo não eram visualizadas quando consultado um requisito pelo sistema.	Cadastro de Requisitos	Tipos de dados de Data e Tempo eram incompatíveis entre banco de dados, aplicação e front-end.	Foi padronizado a estrutura de dados *Timestamp* para tratar data e período dentro do sistema. Foram criadas funções de conversão de tipos de dados para data formatada de *JSON* e de dado recebido por entrada de usuário.
Usuário após fazer login não era identificado	*Login* como usuário Administrador	O id do usuário logado não era armazenado.	Foi acrescentado a rotina de *login*, logo após o ponto de retorno da autenticação, a estrutura e permissões do usuário logado, adicionando estes a variável de *localStorage* do *Browser*.
Não era possível identificar objetos de sistema (Usuários, Equipes) dentro de outros objetos de sistema na comunicação cliente-servidor.	Cadastro de Requisitos	Havia objetos onde a estrutura de montagem de *JSON* era diferente da montagem do objeto no servidor do sistema.	Foi criado entidades padronizadas no mesmo formato *JSON*/classe para todas as entidades e novas instâncias dos objetos lidos passaram a ser criadas e adicionadas a estrutura de retorno.
O sistema falhava	Filtragem no Menu	Os ponteiros de	Foi adicionados pontos de

após muitas atualizações das Tabelas de apresentação dos dados.	de Usuários	acesso ao banco de dados nos objetos de *DAO* eram abertos e executados porém não eram fechados pela aplicação.	encerramento das conexões abertas com o Banco de Dados após a execução das tarefas de cada *Script*

Tabela 13 - Resultados dos Testes Estruturais Fonte: Do autor (2021)

6.3.1 Código de Testes Caixa-Branca Realizados.

Dos testes iterativos, realizados sob-demanda para problemas críticos encontrados durante o desenvolvimento do sistema, segue os principais testes realizados, que ajudaram a elucidar possíveis defeitos existentes no sistema.

A figura 44 mostra um teste realizado para testar o método de recuperar um registro do banco de dados.

```
try {
    System.out.println(Integer.toString(UsuarioDao.getInstance().get(11).getId()));
} catch (SQLException e) {
    e.printStackTrace();
}
```

Figura 44 - Teste de comunicação com Banco de Dados. Fonte: Do autor (2021)

Com ele foi possível verificar o defeito resultante de casas decimais para unidades inteiras no Banco de Dados, apresentado na Tabela 9.

A figura 45 mostra o teste realizado de cadastro de Requisito com ele foi possível identificar o defeito de datas não-padronizadas apresentado na tabela 12.

```
Date time;
time = new Date(Timestamp.valueOf(LocalDateTime.now()).getTime());
System.out.println(time);
Connection conn = OracleConnector.getInstance().connect();
Requisito req = new Requisito(
new Usuario(9, "ANALISTA 2", "tecsis", Boolean.FALSE, Boolean.FALSE,Boolean.FALSE,Boolean.TRUE,
        new Equipe(
                2, "ANALISTAS "
                )
        )
, "X", "X", new Timestamp(Timestamp.valueOf(LocalDateTime.now()).getTime())
);
```

Figura 45 - Teste de Cadastro de Requisito com Data. Fonte: Do autor (2021)

A Figura 46 mostra um dos primeiros testes realizados, para validar a comunicação do servidor com o *Browser*.

```
try {
    int serverPort = 8080;
    HttpServer server = HttpServer.create(new InetSocketAddress(serverPort), 0);

    server.createContext("/api/hello", (exchange -> {
        if ("GET".equals(exchange.getRequestMethod())) {
            Map<String, List<String>> params = ApiUtils.splitQuery(exchange.getRequestURI().getRawQuery());
            String semNome = "Anônimo";
            String name = params.getOrDefault("name", List.of(semNome)).stream().findFirst().orElse(semNome);
            String respText = String.format("Hello %s!", name);
            exchange.sendResponseHeaders(200, respText.getBytes().length);
            OutputStream output = exchange.getResponseBody();
            output.write(respText.getBytes());
            output.flush();
        } else {
            exchange.sendResponseHeaders(405, -1);// 405 Method Not Allowed
        }
        exchange.close();
    }));
    server.setExecutor(null);
    server.start();

    System.out.println("Teste concluido com sucesso");

}catch(Exception e) {
    System.out.println("Error! Alert: ");
    e.printStackTrace();
}
```

Figura 46 - Teste de comunicação servidor-*Browser*. Fonte: Do autor (2021)

O teste apresentado na figura 47 mostra a validação do cadastro de um usuário no Banco de Dados.

```
try {
    Usuario sup = new Usuario("FELPS", "tecsis", false, false, true, false, null);
    UsuarioDao.getInstance().save(sup);
} catch (SQLException e) {
    System.out.println("Error! Alert: ");
    e.printStackTrace();
}
```

Figura 47 - Teste de Cadastro Usuário. Fonte: Do autor (2021)

Após a realização dos testes, e as correções realizadas, foi verificado uma melhoria substancial na qualidade do sistema, nas características de fluidez da navegação, redução das ocorrências de travamento por erros críticos e aumento do número de mensagens de iteração com o usuário, melhorando a facilidade de uso do sistema, a interface durante os testes também sofreu modificações leves para se tornar mais amigável e intuitiva, também foram implementadas mensagens de log no servidor para permitir a visualização das iterações realizadas pelo *client-side* do sistema.

7. Resultados alcançados e conclusões

Ao final do trabalho, foi concluída a implementação de um Sistema de Informação funcional para gestão de ordens de serviço na atividade de desenvolvimento de *software*, com todas as características essenciais a um sistema deste tipo. Também sendo possível disponibilizá-lo para uso gratuito na plataforma *Web*, espera-se que o mesmo possa vir a ser utilizado por equipes independentes de desenvolvimento de *software* como ferramenta de auxílio e evolução das práticas de desenvolvimento e gestão de equipe, sendo este projeto uma proposta para tal.

O sistema desenvolvido permite a gestão de equipes de desenvolvimento e análise de sistemas, assim como atribuição e controle de integrantes dessas equipes, também permite o controle de Requisitos a serem desenvolvidos, porém

não controla os projetos existentes, pelo fato de Gestão de Projetos a nível comercial e organização destes estar fora do escopo do sistema desenvolvido, sendo uma possível melhoria a ser acrescentada em modificações futuras.

Também é possível determinar demandas (Ordens de Serviço) a serem atribuídas a um usuário e associadas a um grupo de usuários (Equipe) sendo assim possível controlar e, por meio das marcações de Status e Prioridade, identificar em qual ponto do fluxo interno de processo a Ordem de Serviço se encontra.

Como o fluxo de processo interno da equipe é uma característica específica do utilizador do sistema, é possível customizar os status e prioridades possíveis por meio do menu de configurações, permitindo a possibilidade do utilizador do sistema criar seus próprios pontos de fluxo que uma demanda pode atingir.

O sistema também utiliza a estrutura de tabelas como forma de facilitar a escalabilidade futura, uma vez que esta permite filtrar os dados por campos específicos e organizá-los na melhor conveniência ao utilizador, os registros da tabela permitem acesso ao registro específico por meio do clique e todos os identificadores de registro contém *links* entre si, permitindo uma navegação dinâmica e fácil entre os registros do sistema. Também foi possível atender a necessidade de guardar a quantidade de esforço referente às demandas, esta atribuição de esforço é propositalmente relativa, para corresponder ao fato de que cada utilizador do sistema terá uma forma própria de quantificar esforço.

Apesar de conter, associado a cada ordem de serviço, um campo referente ao valor monetário do serviço realizado, a utilização deste é opcional e o sistema não realiza a gestão financeira dos serviços prestados, por não ser este o escopo definido do projeto proposto, contudo a modularidade do sistema permite a expansão para este tipo de funcionalidade no futuro.

A Tabela 14 resume as características e funcionalidades do sistema desenvolvido, em relação aos sistemas utilizados para o levantamento de requisitos e suas capacidades.

	Sistema Desenvolvido	Engeman	Soften	Sofit	Gestão Click	Governo Federal	AgoraOS	Sírius
Código	X	X	X	X		X	X	X
Status	X	X		X	X			X
Nome do Solicitante	X	X	X		X		X	X
Nome do Encarregado	X							
Data/Hora Emissão/Entrega	X	X	X	X	X		X	X
Descrição	X	X	X	X	X	X	X	X
Quantitativo (Esforço)	X					X	X	X
Priorização	X	X						
Valor	X		X	X			X	X
Setor/Equipe	X	X						
Rastreamento de Requisito	X							

Tabela 14 - Planilha de Características realizadas x Características observadas no levantamento de Requisitos. Fonte: Do autor (2021).

Vale ressaltar que o sistema desenvolvido não foi disponibilizado para uso em ambiente real (de produção), com isto, apesar de ter mecânicas voltadas para escalabilidade como recurso de cache em memória e filtros para busca de registros nas interfaces de tabela, não houve homologação do sistema quanto a estes

recursos com usuários reais, além de testes de escalabilidade que precisam ser realizados para este tipo de homologação.

Com o desenvolvimento do projeto em questão foi possível para o aluno aperfeiçoar-se no uso das tecnologias utilizadas, como parte da formação como desenvolvedor, e que a ferramenta disponibilizada possa ser usada como referência de capacidade profissional. Também foi possível compreender as características fundamentais a um Sistema de Gestão de Ordens de Serviço e a utilidade deste para as atividades de Gestão em especial de Desenvolvimento de *Software*. Assim como foi possível sedimentar o conhecimento sobre o funcionamento de Sistemas de Informação em Geral, suas estruturas fundamentais e fluxo de informações dentro de um Sistema.

7.1 Recomendações a Trabalhos Futuros

Como proposta para trabalhos futuros, é recomendado realizar a homologação e os testes de escalabilidade sobre todos os componentes e funcionalidades do sistema, para viabilizar a operação do sistema em um ambiente de produção real, tornando possível com isso, medir e analisar resultados deste processo de implantação do sistema.

Como dito anteriormente, o sistema desenvolvido contém a característica de ser um sistema modular e está aberto para expansão de módulos adicionais e integração com outros sistemas por meio de sua *API* de comunicação *REST*. Este pode ser aprimorado com funcionalidades para gestão de projetos, montagem de cronograma de equipes, utilizando os valores de prazos e datas já disponíveis no sistema. Também é possível aprimorar a funcionalidade de guardar valores financeiros, criando um módulo para controle financeiro das atividades do utilizador.

Apesar de terem sido realizados testes sobre a integração dos componentes internos do sistema desenvolvido, um aprimoramento possível de ser realizado é testar a integração deste sistema com outros sistemas utilizados em um ambiente de implantação, por meio da *API* de comunicação *REST* que o sistema contém. Outro aprimoramento quanto a integração que pode ser realizado é viabilizar a migração dos dados do sistema para outros sistemas, por meio da exportação destes dados em arquivos de formato *.csv ou .xml*; inclusive criando um componente para envio e recebimento de arquivos por protocolo *FTP* como uma adição aos recursos já existentes no sistema.

Também é possível a criação de um componente para geração de relatórios das ordens de serviço em formato *.pdf*, permitindo a utilização destas como documento formal impresso. Além disso, pode ser criado um conjunto de funcionalidades para o controle de *workflow* das informações geradas pelo sistema, de forma que as ações realizadas (concluir ordem de serviço, cadastrar requisito, entre outras), possam disparar ações de fluxo (notificação de usuários, envio de email, cadastro automático), referentes as ações realizadas, que o fluxo de trabalho do sistema seja apresentado em forma gráfica aos usuários de perfil gestor, e que seja possível modificar o fluxo de trabalho como uma configuração possível de ser realizada pelo sistema.

O Sistema pode ser ainda aprimorado na qualidade do seu componente de conexão a Bancos de Dados, pois atualmente ele aceita conexões de banco de dados Oracle, mySQL e SQLite, podendo ser expandido para outros tipos de bancos de dados, com a implementação de classes de conexão que implementem a interface de conexão a base de dados do sistema. Além disso, a comunicação *REST* pode ser aprimorada, por não implementar todos os métodos de comunicação *HTTP* existentes, podendo os demais métodos serem implementados no futuro para tornar a comunicação *web* do sistema uma interface *RESTful* que utilize todos os métodos de comunicação *HTTP* possíveis.

Por fim, uma funcionalidade importante de melhoria geral do sistema, seria a ampliação de seu componente de conexão com banco de dados para utilizar múltiplas bases de dados simultaneamente, permitindo o cruzamento de informações de diversos bancos de dados conectados ao sistema.

8. Referências Bibliográficas

ANTONIOL, Giuliano *et al*. ***Is it a bug or an enhancement?: a text-based approach to classify change requests***. Em: CASCON. 2008. p. 304-318.

ALVARENGA, Darlan; SILVEIRA, Daniel; Setor de serviços cresce 3,7% em fevereiro e supera pela 1ª vez nível pré-pandemia. **O Globo**, São Paulo e Rio de Janeiro, 5 de abril de 2021. Economia.
Disponível em: <https://g1.globo.com/economia/noticia/2021/04/15/setor-de-servicos-cresce-37percent-em-fevereiro-aponta-ibge.ghtml>. Acesso em: 19/05/2021.

BARBOSA, Denilson F.; SILLA JR, Carlos N.; KASHIWABARA, André Y. Aplicação da otimização por colônia de formigas ao problema de múltiplos caixeiros viajantes no atendimento de ordens de serviço nas empresas de distribuição de energia elétrica. **Anais do XI Simpósio Brasileiro de Sistemas de Informação**, p. 23-30, 2015.

BARIBIERI, Sérgio Luis. **Teste de Software na indústria: Um estudo qualitativo.** 2018. 77 f. Dissertação (Mestrado) - Curso de Sistemas de Informação, Universidade de São Paulo, São Paulo, 2018.
Disponível em: <https://teses.usp.br/teses/disponiveis/100/100131/tde-12062018-115510/publico/Dissertacao_Sergio_Barbieri_NUSP9414338.pdf>. Acesso em: 11 nov. 2019.

Bhuiyan, N., Gerwin, D. and Thomson, V. (2004a), "*Simulation of NPD process and performance*", *Management Science*, Vol. 50 No. 12, pág. 1690-1703.

Bhuiyan, N., Thomson, V. and Gerwin, D. (2004b), "*A systematic framework for implementing concurrent engineering*", *Research-Technology Management*, Vol. 49 No. 1, 2006.

Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P.; Stal, M. *Pattern-oriented software architecture: a system of patterns*. *New York*, NY, USA: *John Wiley & Sons, Inc*., 1996.

CANFORA, Gerardo; CERULO, Luigi. *Supporting change request assignment in open source development*. In: ***Proceedings of the 2006 ACM symposium on Applied computing***. ACM, 2006. p. 1767-1772.

CARDOSO, Josiane Ap., **Um método de Testes de Integração para Sistemas Baseados em Componentes**. Dissertação de Mestrado, Unicamp, Campinas, São Paulo, Fevereiro, 2006.

CLAIR, S. V. **A história de UML e seus diagramas**. 18 de março de 2016.

Crow, K. (2002), ***Configuration Management and Engineering Change Control**, DRM Associates*, Palos Verdes, CA, 2613 Via Olivera.

DiPrima, M.R. (1982), "*Engineering change control and implementation considerations*", ***Production and Inventory Management**,* Vol. 23 No. 1, pp. 81-7.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. ***Design Patterns: Elements of Reusable Object-Oriented Software*** 1994.

Governo Federal. **Modelo de Ordem de Serviço.** Disponível em: <https://www.governodigital.gov.br/sisp/ncti-nucleo-de-contratacoes-de-tecnologia-da-informacao/nuvem_tab9.jpg/view>. Acesso em: 26 out. 2019.

GUIMARÃES, FERNANDO AUGUSTO MELLO. EDITAL DE CONCORRÊNCIA Nº 02/2011. Disponível em: https://fattocs.com/files/pt/editais/TCE-PR/02-2011.PDF Acesso em: 26/06/2019.

ISO/IEC/IEEE *International Standard* - ***Systems and software engineering.*** ISO/IEC/IEEE 24765:2010(E). 15 Dec. 2010.

JACOBS, Simon. ***Methods and systems for scheduling complex work orders for a workforce of mobile service technicians***. U.S. Patent n. 7,346,531, 18 mar. 2008.

KASSEM, A. Saleh, 2009. ***Software Engineering***. *J. Ross Publishing*. pág. 224–241.

KHALESSI, Amir; ARDALAN, Sasan H. ***Mobile crew management system for distributing work order assignments to mobile field crew units***. *U.S. Patent* n. 6,633,900, 14 out. 2003.

LIMAYE, Milind G. (2009). ***Software Testing***. *Tata McGraw-Hill Education*. pág. 108–111. ISBN 9780070139909.

LILLY, Richard T.; LAYNE, David V. ***Method and apparatus for scheduling work orders in a manufacturing process***. *U.S. Patent* n. 5,787,000, 28 jul. 1998.

MDN Web Docs;
Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript> Acesso em: 8 de out. de 2020.

OLIVEIRA JUNIOR, Edson Alves de; FORTES, Renata Pontin de Mattos. **Arquitetura de Software na Web Atual: Processamento no Servidor.** 2007. 34 f. Universidade de São Paulo, São Carlos, 2007.

Oracle 11g XML DB vs DB 2 9.5 pureXML v3.0, Oracle.
Disponível em:
<www.oracle.com/technetwork/database/features/xmldb/db.pdf>
Acesso em: 29/03/2021.

Ordem de Serviço Governo Federal
Disponível em: <https://www.gov.br/previdencia/pt-br/acesso-a-informacao/conselho-de-recursos-da-previdencia-social/ordem-de-servico-os>. Acesso em: 01 abril. 2021

Ordem de Serviço AgoraOS
Disponível em: <https://www.agoraos.com.br/blog/ordem-de-servico-os-o-que-e-e-como-fazer/modelo_ordem_servico_agoraos/>. Acesso em: 26 out. 2019.

Ordem de Serviço Sofit
Disponível em: <https://www.sofit4.com.br/blog/ordem-de-servico/>. Acesso em: 26 out. 2019

Ordem de Serviço Soften Sistemas
Disponível em: <https://blog.softensistemas.com.br/como-fazer-uma-ordem-de-servico/>. Acesso em: 26 out. 2019

Ordem de Serviço Engeman
Disponível em: <https://engeman.com.br/pt-br/caracteristicas/>. Acesso em: 26 out. 2019

Ordem de Serviço Gestão Click

Disponível em: <https://gestaoclick.com.br/ordem-de-servico>. Acesso em: 26 out. 2019.

Ordem de Serviço ERP Sirius

Disponível em: <http://www.sirius.inf.br/o-sistema-e-recursos/ordem-de-servico/>. Acesso em: 26 out. 2019.

Pikosz, P. and Malmqvist, J. (1998), "*A comparative study of engineering change management in three Swedish companies*", ***Proceedings of DETC98 1998 ASME Design Engineering Technical Conference***, Atlanta, GA, September, pág. 1-11.

PORTAL ERP. **Entenda o que é ERP.** (2021) Dìsponível em: <https://portalerp.com/entenda-erp> Acesso em:

Pressman, R. S. *Software engineering: A practitioner's approach*. McGraw-Hill Higher Education, 2006.

QUALITOR (2020) Como um sistema de chamados auxilia na produtividade do time de TI? Disponível em: <blog.qualitor.com.br/como-um-sistema-de-chamados-auxilia-na-produtividade-do-time-de-ti/>

Reidelbach, P.E. (1991), "*Engineering change management for long-lead-time production*", ***Production and Inventory Management***, Vol. 32 No. 2, pág. 84-8.

Sankhya Code, (2021), Guia para Desenvolvedores Web Sankhya. Disponível em: <http://code.sankhya.com.br/guia-para-desenvolvedores-web-sankhya/> Acesso em: 03/04/2021.

SILVA, R. P. e. UML 2 em Modelagem Orientada a Objetos. Florianópolis: Visual Books, 2007.

Sun Microsystems Inc. ***Core J2EE Patterns - Data Access Objects.*** 2007. Disponível em: http://www.oracle.com/technetwork/java/dataaccessobject-138824.html
Acesso em: 28/04/2021.

Terwiesch, C. and Loch, C.H. (1999), "*Managing the process of engineering change orders: the case of the climate control system in automobile development*", *Journal of Product Innovation Management*, Vol. 16 No. 2, pág. 160-72.

VERGILIO, Silva (2011) Introdução a UML.

World Wide Web Consortium. ***Web Services Architecture*** 11 de fevereiro de 2004. Cap. 3.1.3 *Relationship to the World Wide Web and REST Architectures*. Disponível em: <https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest>
Consultado em 29 de abril de 2021.

9. Apêndice I - Narrativas de Caso de Uso

9.1 Caso de uso 001 – Cadastrar Usuário

Descrição: Este caso de uso descreve como é feito o cadastro do usuário do sistema.

Atores: Administrador.

Pré-Condições: Nenhuma.

Pós-Condições: Usuário cadastrado no sistema.

Fluxo normal

1. O Administrador realiza login no Sistema com o usuário pré-cadastrado padrão para administrador ADMINISTRADOR - tecsis.
2. O Administrador Clica em Usuários > “Novo Usuário”.
3. O Administrador insere o nome do novo usuário.
4. O Administrador cria senha para o novo usuário.
5. O Administrador define Tipo de Usuário (Administrador, Gestor, Desenvolvedor, Analista) podendo um usuário ser mais de um tipo simultaneamente.
6. O Administrador clica em “Salvar”.[FA01]
7. O Sistema Guarda Dados do Usuário.
8. Retorna ao Menu de Usuários.

Fluxos alternativos

FA01 - Administrador Cancela a Operação.

1. O Administrador clica em “Cancelar”.
2. O Sistema descarta os dados do usuário.
3. Retorna ao Fluxo Normal no Passo 9.

9.2 Caso de uso 002 – Cadastrar Ordem de Serviço

Descrição: Este caso de uso descreve como é cadastrada uma Ordem de Serviço no Sistema.

Atores: Gestor da Equipe.

Pré-Condições: Há Cadastrado pelo menos um Gestor, há um Requisito no sistema, o Gestor da Equipe deve estar logado no Sistema.

Pós-Condições: Uma Ordem de Serviço é cadastrada no sistema.

Fluxo normal

1. O Gestor clica no menu Ordens de Serviço > “Nova Ordem”.
2. O Gestor Indica sobre qual Requisito será a OS.
3. O Gestor informa qual usuário é o Solicitante da Ordem de Serviço, podendo ser ele próprio, um Analista ou um outro usuário (Terceiro). [FA01]
4. O Gestor inclui a data de emissão da Ordem de Serviço.
5. O Gestor informa o esforço estimado para realização do serviço.[FA02]
6. O Gestor define uma prioridade para a ordem de serviço.
7. O Gestor define a equipe que será atribuída a ordem de Serviço.
8. O Gestor marca o Status como “Em Aberto”.[FA03]
9. O Gestor clica em Salvar. [FA05]
10. O Sistema Guarda os Dados da OS criada.
11. Retorna ao Menu de Ordens de Serviço.

Fluxos alternativos

FA01 - O Gestor atribui uma data de entrega.

1. O Gestor informa uma data de entrega prevista para a Ordem de Serviço.

2. Retorna ao fluxo normal no passo 4.

FA02 - O Gestor define um Valor Estimado para a OS.

1. O Gestor informa um Valor Monetário em R$ para a realização da OS.
2. Retorna ao fluxo normal no passo 6.

FA03 - O Gestor encarrega um Desenvolvedor para a OS.

1. O Gestor marca um desenvolvedor da equipe como encarregado da OS.
2. O gestor marca o Status como “Em Andamento”.
3. Retorna ao fluxo normal no passo 9.

FA04 - O Gestor clica em cancelar.

1. O Gestor clica no botão “Cancelar”.
2. O Sistema não guarda os dados da Nova OS.
3. Retorna ao fluxo normal no passo 11.

9.3 Caso de uso 003 – Cadastrar Requisito

Descrição: Este caso de uso descreve como é cadastrada um Requisito no Sistema.

Atores: Analista de Sistema.

Pré-Condições: Há Cadastrado pelo menos um Analista de Sistemas, o Analista deve estar logado no Sistema.

Pós-Condições: Um Requisito é cadastrado no sistema.

Fluxo normal

1. O Analista clica no menu Requisitos > “Novo Requisito”.
2. O Analista informa o Título do Requisito.

3. O Analista informa uma Descrição do Requisito e os Detalhes da demanda.
4. O Analista informa o prazo máximo para entrega do requisito.[FA1][FA2]
5. Retorna ao menu de Requisitos.

Fluxos alternativos

FA01 - O Requisito não tem prazo máximo de entrega.

1. O Analista não informa nenhum prazo para o Requisito.
2. Retorna ao fluxo normal no passo 4.

FA02 - O Analista clica em cancelar.

1. O Analista clica no botão “Cancelar”.
2. O Sistema não guarda os dados do novo Requisito.
3. Retorna ao fluxo normal no passo 5.

10. Apêndice II - Diagramas de Sequência

Diagramas de sequência auxiliares usados para o desenvolvimento do Sistema:

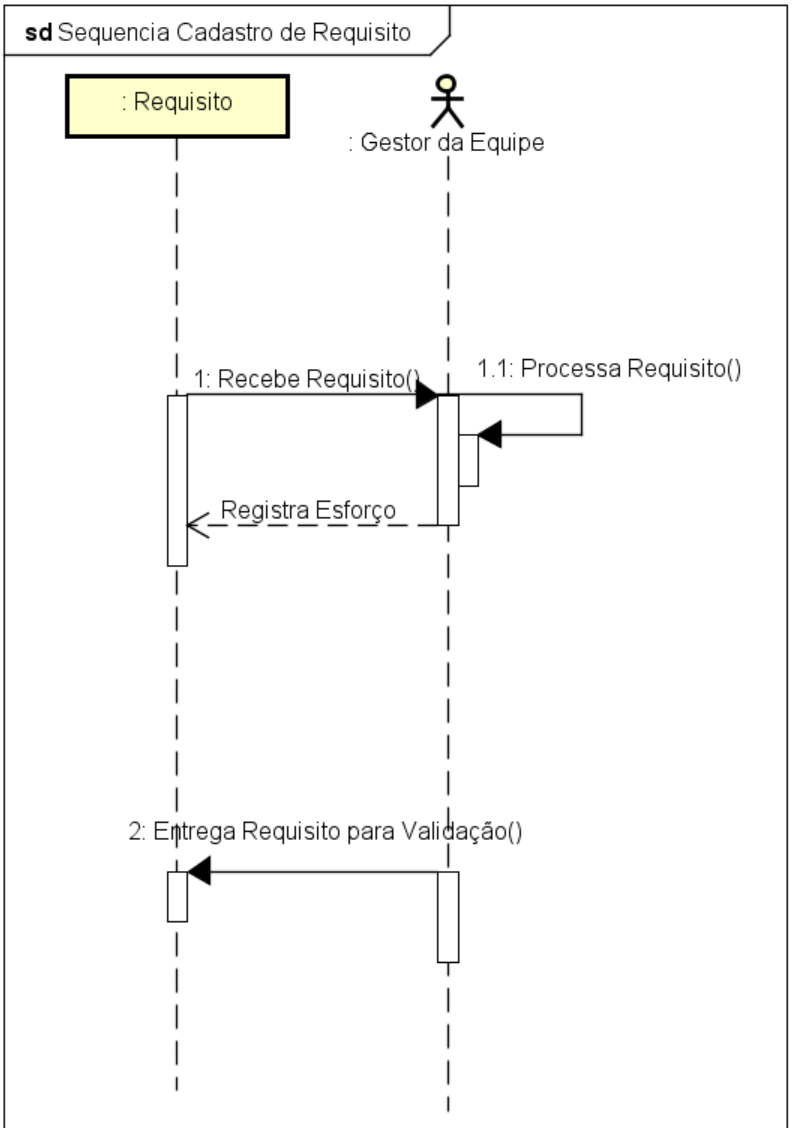


Figura 48 - Fluxo de Cadastro de Requisito. Fonte: Do autor (2021)

A Figura 48 apresenta a sequência de eventos do caso de uso de Manter Requisitos (Figura 4).

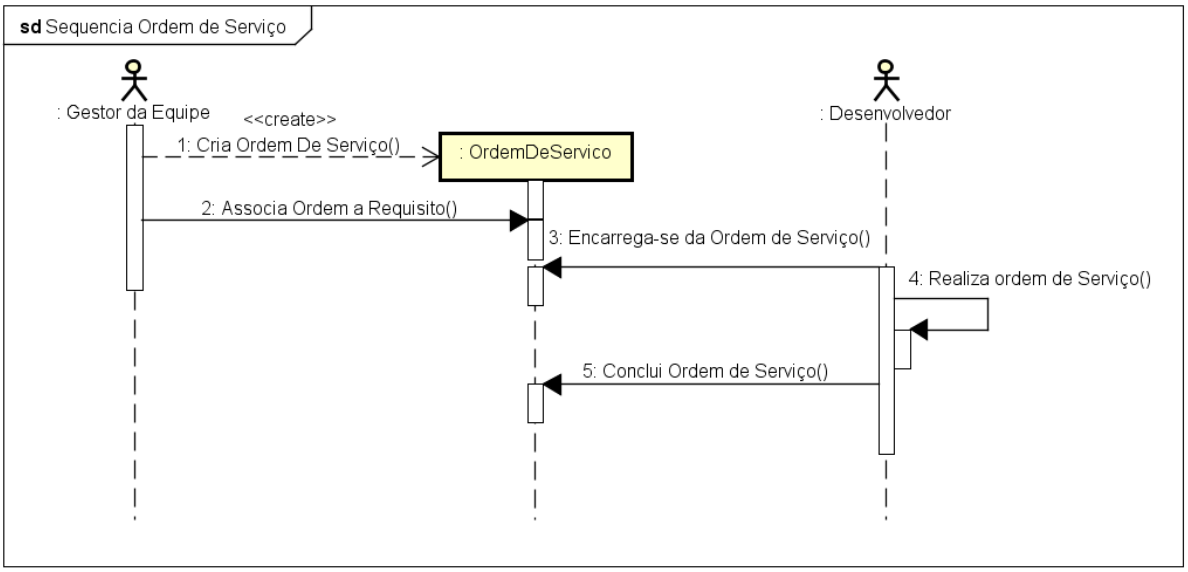


Figura 49 - Fluxo de Cadastro de Ordem de Serviço. Fonte: Do autor (2021)

A Figura 49 apresenta a sequência de eventos do caso de uso de Manter Ordens de Serviço (Figura 3).

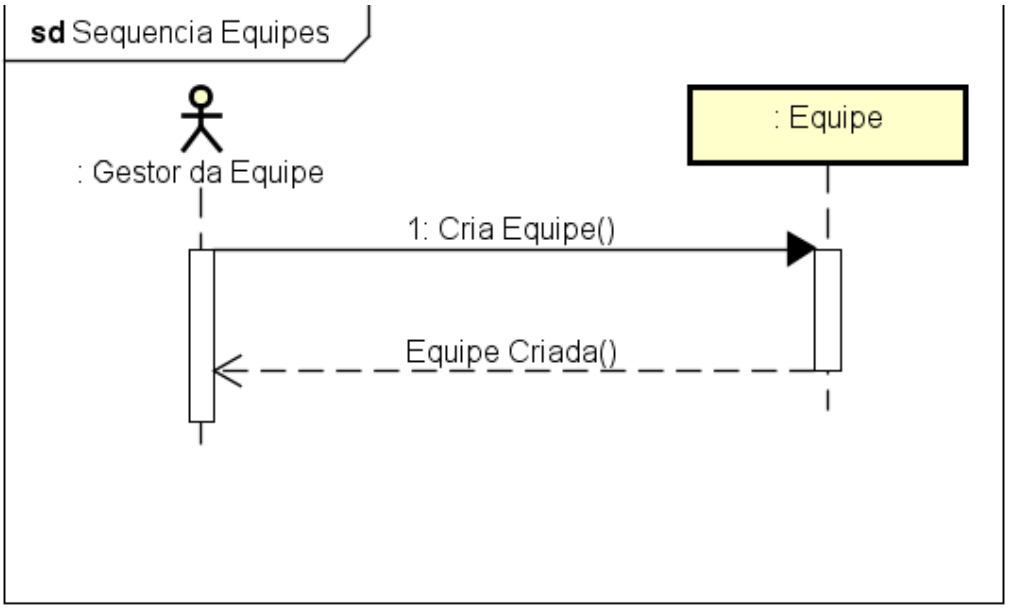


Figura 50 - Fluxo de Cadastro de Equipes. Fonte: Do autor (2021)

A Figura 50 apresenta a sequência de eventos Cadastro de uma Equipe no Sistema, sendo um procedimento similar para cadastro de Status e Prioridades.