

**FACULDADE DE TECNOLOGIA SENAI JARAGUÁ DO SUL
GRADUAÇÃO TECNOLÓGICA EM SISTEMAS PARA INTERNET**



AGIL IT
SISTEMA GERENCIADOR DE ORDENS DE MANUTENÇÃO DE
EQUIPAMENTO PARA A INDÚSTRIA DE ALIMENTOS

JULIO CESAR THOMAZELLI JUNIOR
LUCAS MATHEUS DA SILVA GONÇALVES
MÁRCIO HENRIQUE MEIER

JARAGUÁ DO SUL, SC
2020

**JULIO CESAR THOMAZELLI JUNIOR
LUCAS MATHEUS DA SILVA GONÇALVES
MÁRCIO HENRIQUE MEIER**

AGIL IT

**SISTEMA GERENCIADOR DE ORDENS DE MANUTENÇÃO DE
EQUIPAMENTO PARA A INDÚSTRIA DE ALIMENTOS**

Trabalho de Conclusão de Curso apresentado à
Faculdade de Tecnologia SENAI Jaraguá do Sul
como requisito parcial para obtenção do título de
Tecnólogo em Sistemas para Internet

Professora Orientadora:
Msc. Tathiana Duarte do Amarante

**JARAGUÁ DO SUL, SC
2020**

RESUMO

Atualmente, o processo de ordens de manutenção da indústria Duas Rodas é realizado manualmente, fazendo com que a empresa consuma muito papel, prejudicando o meio ambiente e como consequência, perde-se o controle com a quantidade de informações acumuladas. Sendo assim, o projeto tem como objetivo apresentar e transformar o antigo processo em um fluxo totalmente digital com a aplicação web para gerenciar as ordens e a aplicação mobile para executá-las.

Palavras-chaves: Sistema, WEB, Mobile, Ordem de Manutenção, Digital.

ABSTRACT

Currently, the maintenance order process for the Duas Rodas industry is carried out manually, causing the company to consume a lot of paper, damaging the environment and, as a consequence, there is a lack of control with the amount of accumulated information. Therefore, the project aims to present and transform the old process into a fully digital flow with a web application to manage orders and a mobile application to execute them.

Key-words: System, WEB, Mobile, Maintenance Order, Digital.

LISTA DE FIGURAS

Figura 2.1 –Planejamento de Projeto	15
Figura 2.2 –Exemplo de modelo conceitual especificado em UML	16
Figura 3.1 –Cenário Atual	19
Figura 3.2 –Cenário Agil.It	21
Figura 5.1 –Fluxo do AGIL.IT	28
Figura 5.2 –Caso de Uso do AGIL.IT	29
Figura 5.3 –Entidade de Relacionamento	30
Figura 6.1 –Cronograma de Aprendizagem: Terceiro Semestre	31
Figura 6.2 –Cronograma de Aprendizagem: Quarto Semestre	32
Figura 6.3 –Cronograma de Aprendizagem: Quinto Semestre	32
Figura 6.4 –Cronograma de Aprendizagem: Sexto Semestre	32
Figura 6.5 –Diagrama de Causa e Efeito	35
Figura 6.6 –EAP AGIL.IT	37
Figura 6.7 –Fluxo e Processos	38
Figura 6.8 –Fluxo e Processos AGIL.IT	39
Figura 6.9 –Cadastro de Usuários	41
Figura 6.10 Monitor de Ordem de Manutenção: Cards	42
Figura 6.11 Checklist de Segurança	43
Figura 6.12 Monitor	44
Figura 6.13 Notificações	45
Figura 6.14 Ordem de Manutenção	46
Figura 8.1 –Cadastros do Sistema	64
Figura 8.2 –Cadastro de Equipamento	65
Figura 8.3 –Consulta de Equipamento	66
Figura 8.4 –Dashboard: Visualização por tabela	67
Figura 8.5 –Dashboard: Visualização por cards	67
Figura 8.6 –Análise de pendências de assinatura	68
Figura 8.7 –Ordem de Manutenção	69
Figura 8.8 –Assinatura da Ordem de Manutenção	69
Figura 8.9 –Monitor de Ordens de Manutenção	70
Figura 8.10 Filtros do Monitor	71
Figura 8.11 Ordem de Manutenção Corretiva/Preventiva	72
Figura 8.12 Ordem de Manutenção Lista e Rota	73
Figura 8.13 Ordem de Manutenção Lista e Rota	74
Figura 8.14 Ordem de Manutenção: Apontamento	75
Figura 8.15 Ordem de Manutenção: Apontamento	76

Figura 9.1 –Critério Geral de Apresentação	79
Figura 9.2 –Critérios Gerais	80

LISTA DE ABREVIATURAS E SIGLAS

MVP	Minimum Viable Product
API	Application Programming Interface
ORM	Object-Relational Mapping
OM	Ordem de Manutenção
ERP	Enterprise Resource Planning
SAP	Systeme, Anwendungen und Produkte in der Datenverarbeitung
WEB	Internet
UC	Unidade Curricular
JWT	Json Web Token
UI	User Interface
UX	User Experience
TI	Tecnologia da Informação
SCM	Source-Control Management
CRUD	Create, Read, Update, Delete
CORS	Cross-origin resource sharing
JSON	JavaScript Object Notation

SUMÁRIO

1	Introdução	10
1.1	Problema	10
1.2	Objetivos Gerais	11
1.3	Objetivos Específicos	11
1.4	Justificativa	11
1.5	Método de Trabalho	12
1.6	Organização do Trabalho	12
1.7	Glossário	12
2	Fundamentação Teórica	13
2.1	Sistemas Computacionais	13
2.2	Computação Verde	13
2.3	Planejamento e Gerenciamento de Projetos de Software	14
2.4	UML - <i>Unified Modeling Language</i>	15
2.5	SCRUM	16
2.6	GIT	17
2.7	ORM - <i>Object-Relational Mapping</i>	17
3	Descrição Geral do Sistema	18
3.1	Descrição do Problema	18
3.1.1	Cenário Atual	18
3.1.2	Cenário Com Agil.It	19
3.2	Principais Envolvidos e suas Características	21
3.2.1	Usuários do Sistema	21
3.2.2	Desenvolvedores do Sistema	22
3.3	Regras de Negócio	22
4	Pesquisa de Anterioridade	24
4.1	Produttivo	24
4.2	SoftByte	24
4.3	ProdWin	24
4.4	Diferenciais Agil.It	24
5	Requisitos do Sistema	25
5.1	Requisitos Funcionais	25
5.2	Requisitos Não Funcionais	26
5.3	Fluxograma do Sistema Desenvolvido	27
5.4	Diagrama de Caso de Uso do Sistema	28
5.5	Modelo Entidade-Relacionamento do Banco de Dados	29

6 Planejamento do Sistema	31
6.1 Cronograma do Projeto	31
6.2 Análise de riscos	32
6.2.1 Diagrama de Causa e Efeito	33
6.3 PMBOK	36
6.3.1 Estrutura Analítica do Projeto	36
6.3.2 Fluxo de Processos	38
6.4 Protótipo	40
6.5 Aplicação WEB	41
6.5.1 Cadastro de Usuário	41
6.5.2 Monitor de Ordem de Manutenção: Cards	42
6.5.3 Checklist de Segurança	43
6.6 Aplicação Mobile	43
6.6.1 Monitor	44
6.6.2 Central de Notificações	45
6.6.3 Ordem de Manutenção	46
7 Implementação	47
7.1 Tecnologias	47
7.1.1 Aplicação WEB	47
7.1.2 Aplicação Mobile	47
7.1.3 Aplicação do Servidor	48
7.2 Códigos Desenvolvidos	48
7.2.1 WEB	48
7.2.1.1 Componentes	48
7.2.1.2 Cookies	49
7.2.1.3 Helpers	50
7.2.2 Mobile	54
7.2.2.1 Ações da Ordem de Manutenção	54
7.2.2.2 Dados Armazenados no Local Storage	56
7.2.3 Servidor	57
7.2.3.1 Mapeamento do Banco de Dados com <i>TypeOrm</i>	57
7.2.3.2 Validação de Objetos com o Class Validator	60
7.2.3.3 API	61
7.2.3.4 Estratégia de Exclusão de Dados	63
8 Agil.It	64
8.1 WEB	64
8.1.1 Cadastros do sistema	64
8.1.2 Dashboard	66
8.1.3 Análise de Pendências de Assinatura	68
8.1.4 Ordem de Manutenção	69

8.2 Mobile	70
8.2.1 Monitor de Ordens de Manutenção	70
8.2.2 Ordem de Manutenção Corretiva/Preventiva	72
8.2.3 Ordem de Manutenção Lista e Rota	73
8.2.4 Operações da Ordem de Manutenção	74
8.2.5 Apontamento	75
8.2.6 Assinatura	76
9 Testes de Usabilidade	77
9.0.1 Elaboração dos Testes Aplicados	77
9.0.2 Discussão dos Resultados	78
10 Considerações Finais	82
REFERÊNCIAS BIBLIOGRÁFICAS	83

1 INTRODUÇÃO

A faculdade SENAI (Serviço Nacional de Aprendizagem Industrial)¹ visa sistematizar os conhecimentos adquiridos pelos estudantes durante o decorrer do curso, como também, oferecer vivência prática-profissional mediante a aplicação dos conhecimentos em situações reais. Além disso, busca propiciar ao estudante o contato com o universo acadêmico da iniciação científica, através da implantação do *Projeto Integrador*, que tem exatamente essas características em locais que está sendo implantado. Portanto, o Senai está em parceria com a empresa Duas Rodas desenvolvendo este projeto.

A empresa Duas Rodas possui sua sede em Jaraguá do Sul, Santa Catarina e tem como principal ramo a indústria de alimentos². Para a execução de suas tarefas diárias, seus funcionários contam com o auxílio direto de máquinas de pequeno, médio e grande porte, e sabe-se que esses equipamentos necessitam de manutenção preventiva, preditiva e corretiva de tempos em tempos. Para tanto, se faz necessário ajustes adequados e rápidos para que não haja baixa produtividade e consequentemente uma redução em seus proveitos.

Esses equipamentos precisam passar por uma série de cuidados e estar em bom estado para o uso dos seus colaboradores. Portanto, é de extrema importância para a empresa Duas Rodas ter um controle e um bom fluxo de manutenções desses equipamentos, de modo a não haver uma inatividade desnecessária, e quando necessário a manutenção, esta seja rápida e eficiente.

Entendendo o cenário acima, este trabalho se propõe a melhorar o processo de manutenção de equipamentos da empresa Duas Rodas, através de um sistema para gerenciamento de ordens de manutenção que irá linkar a parte administrativa com o operacional, fazendo com que o administrador possa ter uma comunicação rápida com o técnico e visualizar em tempo real o andamento da manutenção. Sendo assim, este processo ficará mais eficiente e totalmente digital, fazendo com que a empresa reduza a quantidade de papel impresso, gerando assim uma economia e tornando-a mais sustentável.

1.1 Problema

O processo de manutenção de equipamentos de um parque fabril necessita de um acompanhamento constante das tarefas que serão executadas. No cenário da empresa Duas Rodas, o acompanhamento das ordens de manutenção ocorre de forma impressa, o que torna difícil a gestão da mesma, pois necessita que as informações sejam transportadas fisicamente entre os colaboradores, gestores e técnicos. No final da manutenção é preciso que seja digitado ao ERP SAP todo o processo executado pelos técnicos nos equipamentos.

¹ <<http://sc.senai.br/pt-br/faculdade-senai-jaragua-do-sul>>

² <<https://www.duasrodas.com/>>

1.2 Objetivos Gerais

O presente estudo visa aperfeiçoar os processos administrativos no que diz respeito às ordens de manutenção feitas pela empresa Duas Rodas, através de um sistema digital que irá facilitar a execução da manutenção de equipamentos e a gestão do processo, desde a abertura de uma ordem até o seu encerramento.

1.3 Objetivos Específicos

- Identificar referências bibliográficas voltadas a softwares e gestão de manutenção.
- Elaborar a análise dos requisitos e a prototipação do sistema.
- Desenvolver o sistema.
- Analisar e validar o desenvolvimento do sistema.
- Testes necessários.
- Apresentar o sistema desenvolvido conforme o problema proposto.

1.4 Justificativa

O desenvolvimento do projeto irá tornar o processo de manutenção de equipamentos da empresa Duas Rodas mais eficiente, através de um sistema de gerenciamento de manutenção que irá interligar toda a parte administrativa ao setor operacional, deixando o processo mais fluído, irá também reduzir os gastos com papéis e o tempo demandado para a gestão do processo.

O ciclo de vida da informação, é a mudança no valor da informação com o decorrer do tempo. Quando os dados são criados, muitas vezes possuem seu valor mais alto e são usados com frequência. Porém, conforme o tempo passa, os dados não digitais se perdem com mais facilidade e tem menos valor para a organização. As empresas modernas precisam que seus dados estejam protegidos, íntegros e disponíveis em tempo integral. Com isso, os sistemas digitais podem fornecer a otimização apropriada de armazenamento, uma política eficaz de gerenciamento de dados necessária para dar suporte e potencializar os benefícios da empresa, explica (SOMASUNDARAM, 2009).

As empresas não devem apenas beneficiar os proprietários, mas toda a sociedade, especialmente as classes mais penalizadas. Não basta somente a responsabilidade social, pois a comunidade deve ser pensada junto a interface com a natureza, da qual é um subsistema. Com isso, se introduziu a responsabilidade socioambiental, programas que tem por objetivo diminuir a pressão que a atividade produtiva e industrialista faz sobre o meio ambiente. As inovações tecnológicas ajudam neste propósito sem mudar o rumo do crescimento e desenvolvimento que implica a dominação da natureza, de acordo com (BOFF, 2017).

Exposto isso, o sistema proposto objetiva a diminuição do consumo excessivo de papel no processo de manutenção de equipamentos, deixando a empresa Duas Rodas mais sustentável e preparada para as tendências futuras envolvendo o meio ambiente.

1.5 Método de Trabalho

Para o desenvolvimento do projeto foi utilizado a metodologia SCRUM no formato MVP (Produto Mínimo Viável). Essa metodologia consiste em quebrar o sistema em várias partes pequenas e fazer entregas a cada ciclo, que normalmente possuem de 1 a 2 semanas. E o formato MVP prega o desenvolvimento de algo com o menor investimento possível, a fim da validação da ideia ou conceito utilizado.

1.6 Organização do Trabalho

O desenvolvimento do trabalho será composto pela fundamentação teórica, descrição geral do sistema, pesquisa de anterioridade, requisitos do sistema, planejamento do sistema, a implementação do sistema e por fim, testes de usabilidade do sistema.

1.7 Glossário

SCRUM: Metodologia ágil de desenvolvimento de projetos.

MVP: Produto com o mínimo valor possível, visado para validação da ideia do projeto.

API: Interface para comunicação entre diferentes aplicações.

ORM: Tecnologia que auxilia o gerenciamento do banco de dados através da modelagens de classes.

Express: Tecnologia que abstrai requisições web.

Sequelize: Biblioteca de ORM para bancos relacionais, incluindo SQL Server.

Feedback: Retorno a um acontecimento.

Software: Programa de computador.

EAP: Estrutura Analítica do Projeto.

PMBOK: Conhecimento em Gerenciamento de Projetos.

ERP: Software de gerenciamento de empresas.

SAP: ERP alemão conhecido mundialmente.

Front-end: Parte visual do sistema.

Back-end: Parte lógica do sistema.

CRUD: Definição de uma rotina capaz de cadastrar, ler, atualizar e deletar registros de uma tabela.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Sistemas Computacionais

Os sistemas computacionais tomaram conta da sociedade atual, o uso de tecnologias computacionais vem tendo muito espaço na comunidade, seja fazendo uso de um computador pessoal, smartphone ou um tablet. Sendo assim, tais dispositivos foram criados essencialmente para satisfazer as necessidades das empresas de forma a garantir e resolver seus problemas organizacionais, ou seja, a criação de uma ferramenta que possa armazenar e cruzar dados e informações sem que sejam necessárias pilhas de papéis. Estes sistemas garantem às empresas a possibilidade de atingir mercados em locais mais distantes. (MATTIOLI, 2020).

Destaca-se, que o sistema da informação é um tipo especializado de sistema, que pode ser definido de várias formas. Com isso, pode-se entender que sistemas são séries de elementos ou componentes inter-relacionados que coletam entrada de dados, manipulando-os, processando-os, disseminando a saída dos dados e informações, fornecendo assim um mecanismo de *feedback*. (STAIR, 2008).

Atualmente a palavra **sistema** é mal utilizada, usa-se de forma indiscriminada e sem qualquer tipo de fundamento, ou ainda, é usada para expressar determinadas situações dentro de um software, principalmente nos meios empresariais conforme explica (ROSSINI, 2006).

Diante disso, é importante conceituar dados, sendo estes como fatos básicos, por exemplo, o nome e a quantidade de horas trabalhadas em uma semana de um funcionário, quantidade de peça em estoque ou pedidos. Importante mencionar que as informações são compostas por um conjunto de fatos organizados de modo a terem valor adicional, além do valor dos fatos propriamente ditos. Portanto, quando dados são organizados ou alcançados de maneira significativa, se transformam em informações. (STAIR, 2008).

Conforme informações acima dispostas, apesar de o termo sistema estar muito difundido e utilizado muitas vezes de forma leviana, seu significado é bastante preciso. Um sistema é um conjunto de elementos que trabalham de forma integrada a atingir uma ou mais finalidades. Para que o sistema funcione corretamente, é necessário transformar dados em informações de forma que seus objetivos sejam alcançados, desde a finalidade única de cada elemento até a totalidade das funcionalidades integradas do mesmo.

No próximo capítulo será abordado a computação verde.

2.2 Computação Verde

Computação ou TI Verde é um conjunto de iniciativas de investimento na implantação, uso e gerenciamento que tem a finalidade de minimizar o impacto negativo ambiental na área de tecnologia da informação. (QUEIRÓS, 2020).

O investimento de capital em computação verde é composto por três dimensões: estrutural, humano e relacional. O capital estrutural refere-se a infraestrutura da TI que engloba *hardware*, *software*, redes e tecnologia da informação. O capital humano é a capacidade e experiência dos profissionais de TI a respeito de conservação de energia em tecnologia e desenvolvimento pessoal com capacidades em TI verde, obtida através de treinamento e estudos. Por último, o capital relacional envolve o gerenciamento da TI verde e a relação das organizações com seus parceiros e usuários, implementando conceitos de proteção ambiental em produtos e serviços. (QUEIRÓS, 2020).

Com isso, a adoção das práticas de TI verde propicia uma operação mais sustentável às organizações, gerando economia com energia, papel, água, transporte, espaço físico, manutenção e descarte, proporcionando assim valor tanto para a organização quanto para a sociedade (MOURA, 2017).

No próximo capítulo será abordado o planejamento e gerenciamento de projetos de software.

2.3 Planejamento e Gerenciamento de Projetos de Software

Um projeto é um empreendimento temporário que objetiva criar um produto, resultado ou serviço único, que no caso de projeto de software é o sistema em funcionamento (COUTO, 2018).

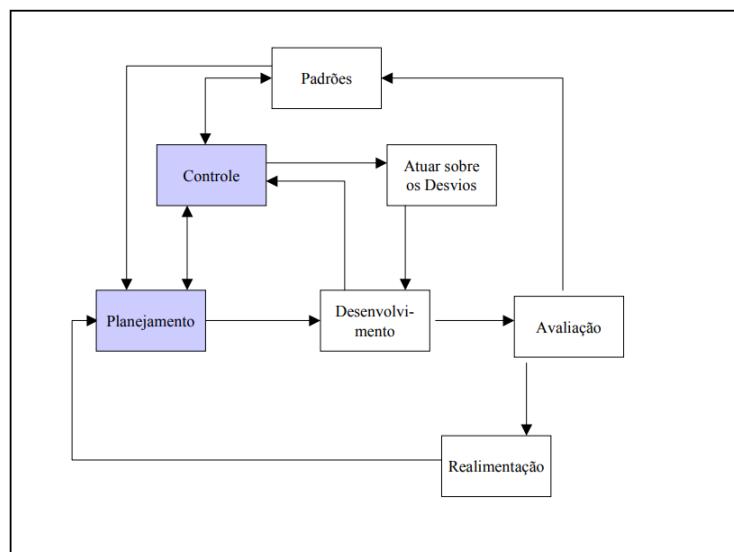
Os projetos de software apresentam particularidades principalmente por desenvolver produtos incompreensíveis e pela dificuldade do seu gerenciamento, além da falta de comunicação entre os gerentes/desenvolvedores e os clientes/usuários (PRADO, 1999). As etapas de desenvolvimento seguem um ciclo de vida com fases próprias, tais como a especificação de requisitos, análise, projeto, implementação, testes e implantação (MOURA, 2018).

A competitividade e os avanços tecnológicos provocaram um aumento na exigência de qualidade e na complexidade de decisões administrativas. Para obter êxito em um projeto é necessário planejar e gerenciar com eficiência a execução de diversas atividades independentes, pois o grau de risco e incerteza quanto ao sucesso do projeto são elevados (HAUFE, 2001).

Para gerir um projeto, é possível utilizar modelos mais prescritivos como o PMBoK, mais adaptativos como o Scrum ou híbridos. Os modelos prescritivos possuem uma estrutura formal de elementos do processo como atividades e tarefas, um fluxo de trabalho que descreve como cada um destes elementos deve ocorrer e como eles são relacionados. A busca pela estrutura e a ordem é uma característica importante deste tipo de modelo, que normalmente são compostos por *frameworks* como PMBoK, PRINCE2 e IPMA(COUTO, 2018). Enquanto os modelos prescritivos presam principalmente pela metodologia e pelo fluxo dos processos, modelos adaptativos como os oriundos do manifesto ágil, são focados na interação entre os usuários no processo de construção do produto através de uma abordagem iterativa e adaptativa(COUTO, 2018). Os métodos ágeis possuem valores fundamentais, que foram definidos no manifesto ágil, de 2001:

- Indivíduos e interação entre eles mais que processos e ferramentas;
- Software em funcionamento mais que documentação abrangente;
- Software em funcionamento mais que documentação abrangente;
- Responder a mudanças mais que seguir um plano.

Figura 2.1: Planejamento de Projeto



Fonte: (HAUFE, 2001)

A figura 2.1 mostra o ciclo de gerenciamento de projetos de software. O controle atua tanto no planejamento quanto no desenvolvimento, proporcionando desvios que originam ações corretivas. Entretanto, a avaliação ao final do projeto, reabastece o planejamento com novos projetos e ideias, prosseguindo assim em ciclo indeterminado. Já os padrões são definidos a partir dos controles e das avaliações para controlar e planejar o decorrer do projeto.

No próximo capítulo será abordado a UML.

2.4 UML - *Unified Modeling Language*

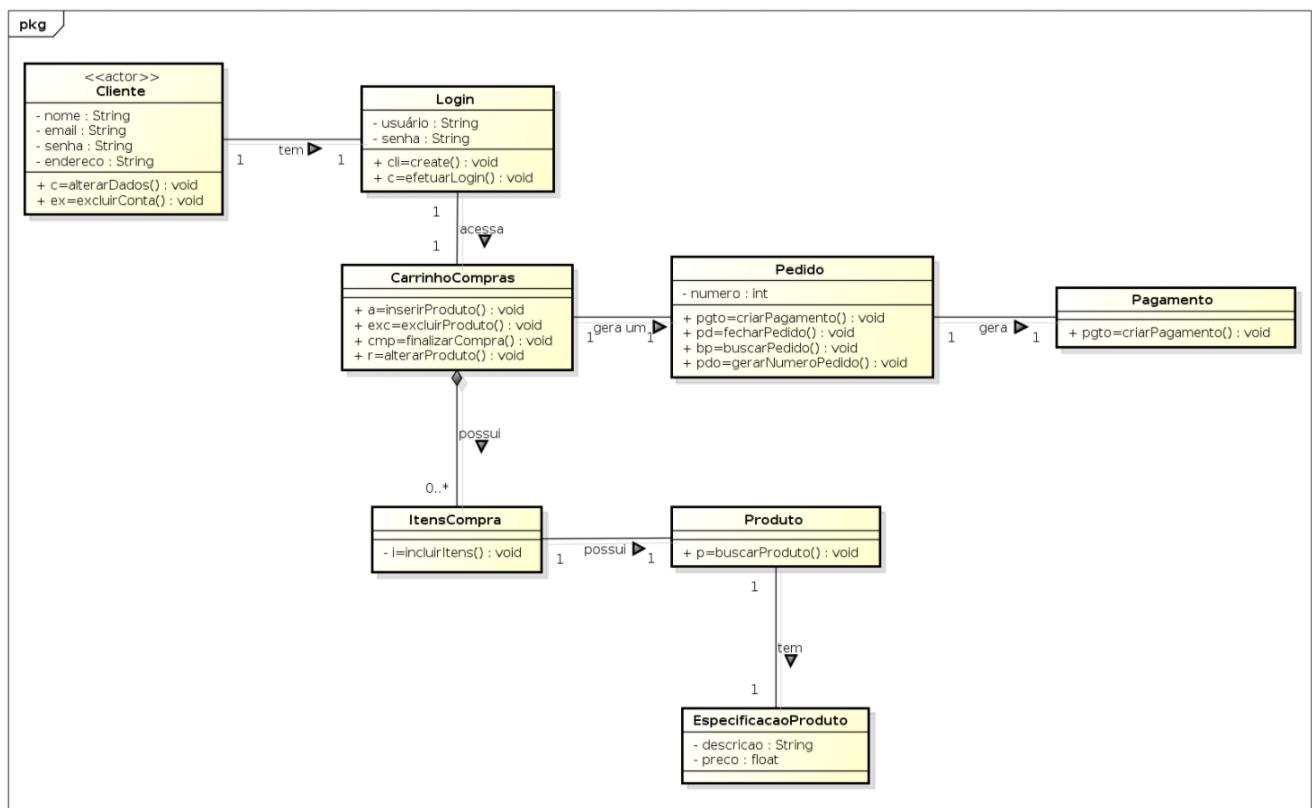
A UML foi desenvolvida com o propósito de fornecer a arquitetos e engenheiros, ferramentas para análise, design e implementação de software, bem como oferecer suporte à modelagem de negócio (GASPARINI, 2018). A UML é dividido em dois grupos: estrutural que permite representar a parte estática do sistema e comportamental que permite representar objetos dinâmicos no sistema (SILVA, 2018).

Os modelos estruturais utilizam classes de objetos e seus relacionamento, os relacionamentos importantes que podem ser documentados nesse estágio são os de generalização e composição. Enquanto os modelos dinâmicos mostram as interações entre os objetos do sistema, as interações que podem ser documentadas incluem a sequência de solicitação de serviço

feitas pelos objetos e as mudanças de estado que são disparadas por essas interações de objetos (SOMMERVILLE, 2011).

A figura 2.2 representa um modelo de diagrama de classes que aborda uma experiência em um *e-commerce*. Um cliente pode fazer o login na plataforma, inserir, excluir e alterar produtos e finalizar a compra. Cada item do carrinho possui o produto e a quantidade, e o produto possui preço e descrição. Após finalizar o carrinho a plataforma gera um pedido que gera um pagamento.

Figura 2.2: Exemplo de modelo conceitual especificado em UML



Fonte: (PRIMO, 2014)

No próximo capítulo será abordado a metodologia ágil de desenvolvimento SCRUM.

2.5 SCRUM

O Scrum, criado em 1993 por Ken Schwaber e Jeff Sutherland, tem a origem de seu nome no jogo de rúgbi e se refere à maneira como um time trabalha junto para avançar com a bola no campo. Tudo se alinha: posicionamento cuidadoso, unidade de propósito, clareza de objetivo, tudo se unindo (ROCHA, 2015).

O Scrum é um *framework* utilizado em projetos ágeis para o gerenciamento e desenvolvimento de produtos, tendo como foco a entrega de valor de um negócio no menor tempo possível (CRUZ, 2013). Baseia-se em aproveitar a maneira como as equipes trabalham de fato, forne-

cendo ferramentas para se auto organizarem e otimizarem a velocidade e qualidade do trabalho (SUTHERLAND, 2016).

Por conta disso, o Scrum vem sendo adotado por organizações de diversos tamanhos e tipos, desde multinacionais à *startups*, de famosas a desconhecidas. É utilizado em projetos de características igualmente diversificadas, projetos críticos de centenas de milhares de dólares e em projetos internos e simples (SABBAGH, 2014).

No próximo capítulo será abordado a tecnologia GIT.

2.6 GIT

O Git é um sistema de controle de versão (SCM) distribuído e um sistema de gerenciamento de código fonte. Foi projetado e desenvolvido por Linus Torvalds, o criador do Linux (SOUZA, 2017). O Git permite que uma equipe de desenvolvedores possam trabalhar de forma colaborativa em um projeto sem a preocupação com perda de informação na edição ou criação de arquivos (KONNORATE, 2019). Apesar de ser distribuído e não depender de um servidor central, normalmente existe um servidor principal chamado de *origin*, a partir dele os desenvolvedores do projeto cloram e trabalham nesse repositório sem necessitar mais da comunicação com o repositório principal até ser necessário sincronizar as mudanças novamente (CUNHA, 2018). Enquanto os desenvolvedores trabalham em suas atividades, mudanças paralelas podem ocorrer, resultando em conflitos ao integrar o código. Resolver estes conflitos não é uma tarefa trivial, podendo se tornar custosa ao desenvolvedor. Estes tipos de problemas não possuem uma solução exata, pois existe mais de uma maneira de solucioná-los (CUNHA, 2018).

No próximo capítulo será abordado a técnica ORM.

2.7 ORM - *Object-Relational Mapping*

ORM é um método de programação para troca de dados entre tipos de sistemas incompatíveis de banco de dados relacionais e linguagem orientada a objetos. A tecnologia ORM constitui um “banco de dados orientado a objetos virtual” que consegue lidar internamente com a linguagem de programação (NAZÁRIO, 2019). O seu principal objetivo é reduzir a impedância da programação orientada a objetos na utilização de bancos de dados relacionais, nela, as tabelas do banco de dados são mapeadas em classes e os registros de cada tabela são mapeados em instâncias das classes correspondentes (MICHALSKY, 2012), de forma que o objeto não precisa saber nada a respeito do banco de dados e nem o banco de dados ter algum conhecimento em relação ao objeto ou a estrutura de dados da aplicação, toda a conversão é realizada pela ORM (FAYYAZ; MUNIR, 2014). O uso de *frameworks* ORM reduzem significativamente o esforço de não apenas comunicar com o banco de dados mas também de lidar com operações básicas do mesmo, como inserir, atualizar, ler e deletar registros, já que as alterações de objetos são propagadas automaticamente aos registros correspondentes na base de dados (NAZÁRIO, 2019).

3 DESCRIÇÃO GERAL DO SISTEMA

O Software de gestão empresarial utilizado pela empresa Duas Rodas atualmente é o SAP, nele é controlado todo fluxo operacional e de funcionamento da empresa. O Agil.It trabalhará em conjunto com o SAP no módulo de manutenção de equipamentos, atuando como meio digitalizado e fazendo ponte com o SAP onde irá integrar informações prévias e competentes às ordens de manutenção.

Dividido em duas aplicações independentes, o sistema é capaz de trabalhar de forma autônoma, podendo receber e enviar dados via API ou realizar cadastros manualmente na aplicação WEB. Enquanto no aplicativo será possível acompanhar as ordens, fazer apontamentos e realizar breves consultas.

3.1 Descrição do Problema

A gestão do processo de manutenção de máquinas é feita de forma manual, com isso, há bastante retrabalho para repassar todos os dados ao sistema, outra preocupação é o gasto com folhas de papel. Em adendo, não há uma boa análise dos problemas comuns, equipamentos mais problemáticos, eficiência dos técnicos, entre outras.

Como o processo é feito manualmente, há bastante retrabalho por parte dos administradores e líderes do setor em questão. O tempo consumido pelos colaboradores para a correção dos problemas não permite tempo hábil para fazer a análise dos defeitos ocorridos, sendo assim existe uma dificuldade em realizar métricas desde equipamentos problemáticos até ineficiência dos técnicos.

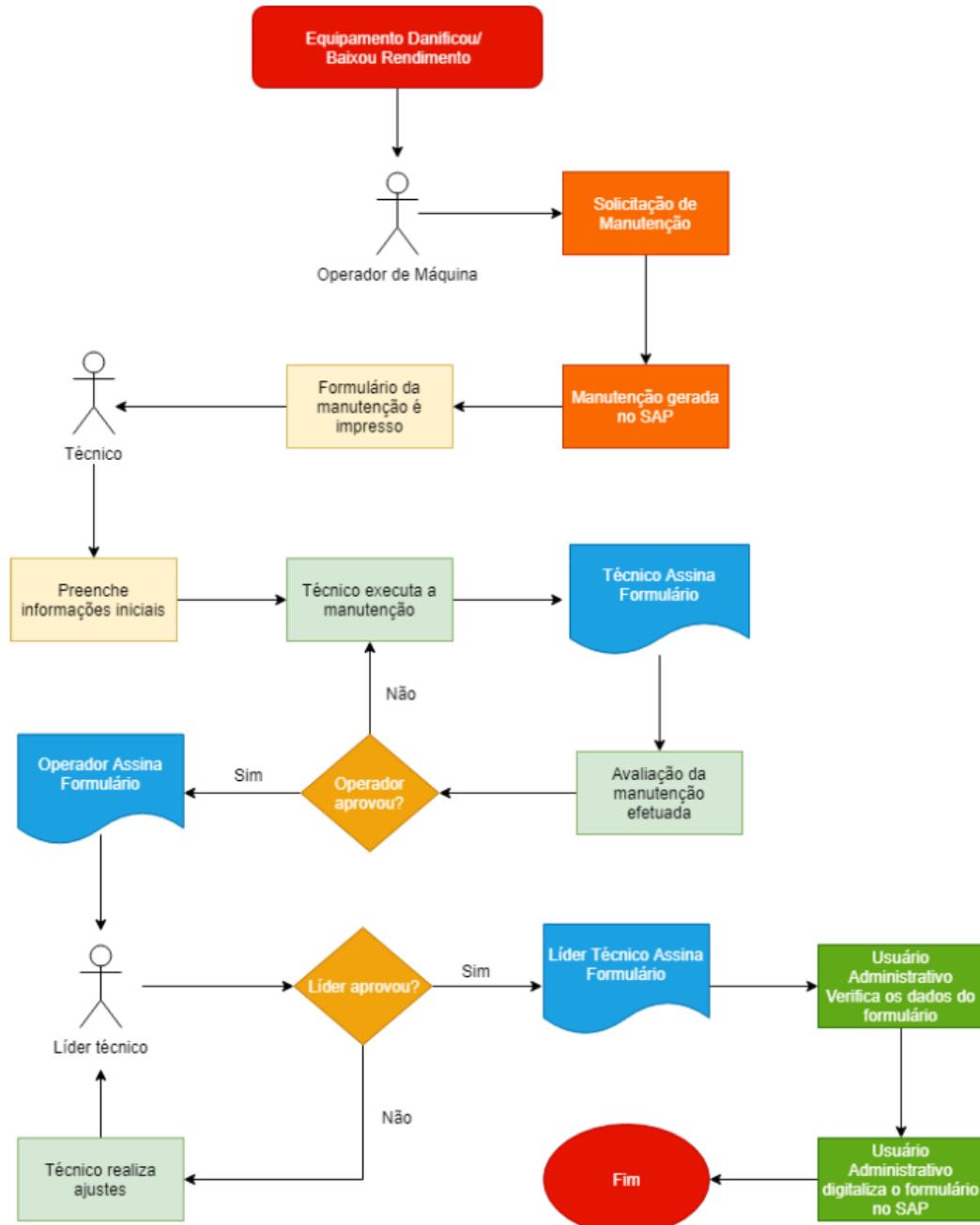
Outra preocupação que objetiva a automatização das ordens de manutenção é o consumo excessivo de papel, pois a tendência das empresas é serem cada vez mais sustentáveis e portanto, ao digitalizar o processo de ordem de manutenção, a empresa Duas Rodas poderá ter maiores índices de sustentabilidade e alcançar um maior destaque entre as demais empresas.

3.1.1 Cenário Atual

No processo de manutenção de equipamentos, a empresa Duas Rodas emite uma ordem de manutenção pelo SAP. Após a emissão, é impresso um formulário e entregue ao técnico elencado para a realização da manutenção. O técnico então preenche os campos básicos do formulário e depois preenche os campos que detalham as operações realizadas, bem como os componentes utilizados. Ao finalizar a manutenção, são colhidas 3 assinaturas, a do operador da máquina, a do técnico da manutenção e a do líder de manutenção, sendo que cada uma das partes pode negar a assinatura e solicitar alterações na manutenção realizada. Com todas as assinaturas rubricadas, o formulário é enviado a um funcionário administrativo que digitaliza as informações preenchidas no SAP. Além de gerar um retrabalho imenso, há um problema com a caligrafia dos técnicos, o que dificulta a digitalização dos dados. Com o volume atual de

manutenções, é inviável manter o fluxo dessa forma. Com isso, entra o projeto Agil It. Atuando como meio digitalizador, faz a ponte entre a manutenção na fábrica e o sistema SAP, fazendo com que não haja mais retrabalho e eliminando totalmente os papéis utilizados nos formulários.

Figura 3.1: Cenário Atual



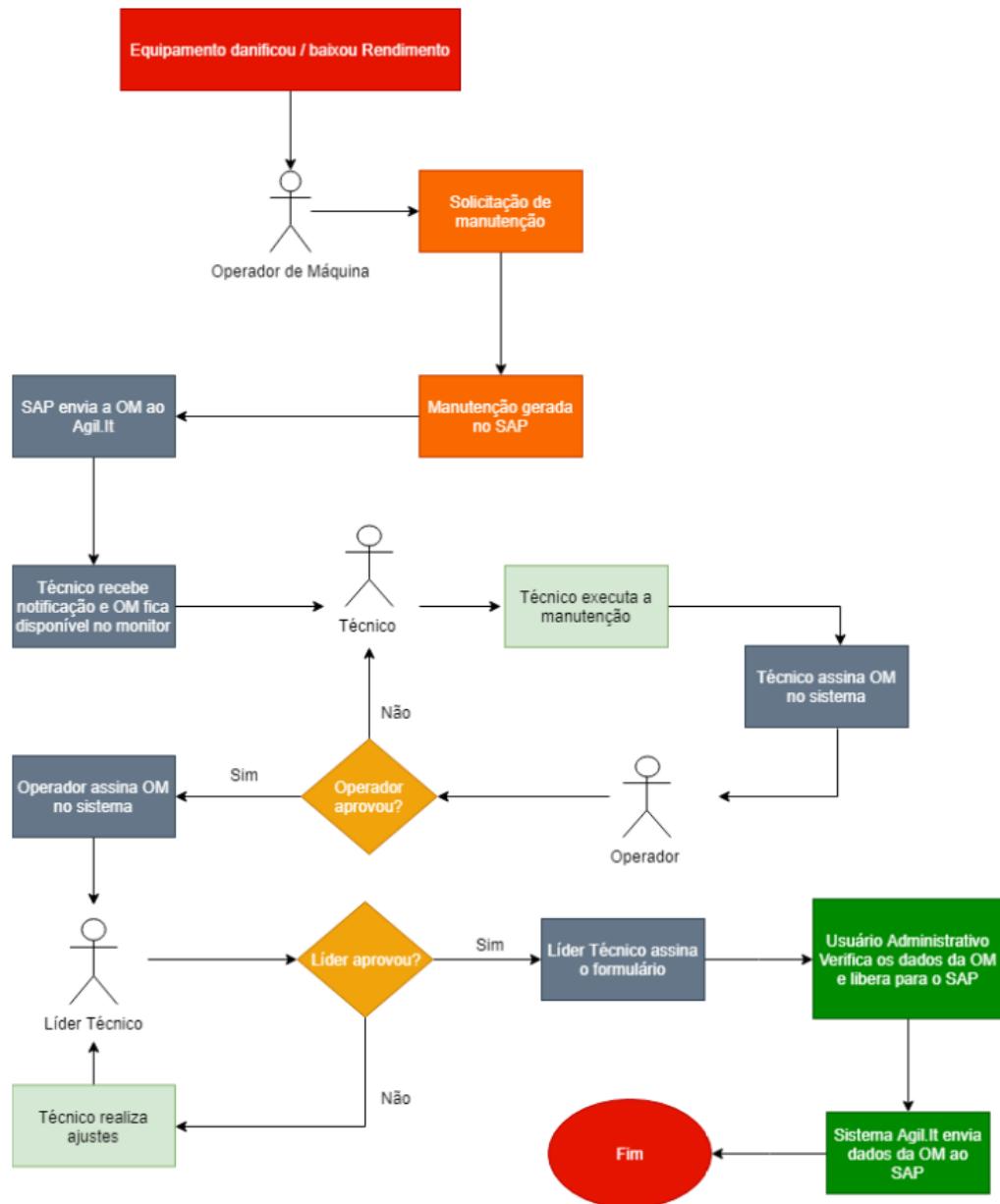
Fonte: Os Autores(2020)

3.1.2 Cenário Com Agil.It

No processo de manutenção de equipamentos com a Agil.It, a ordem de manutenção será gerada no SAP e os dados serão integrados ao sistema Agil.It. Ao ser integrado, o técnico de manutenção receberá uma notificação informando que tem uma nova OM e ele poderá

visualizá-la em seu monitor no aplicativo mobile ou na aplicação WEB. Ele poderá então iniciar a manutenção e fazer os apontamentos das operações e componentes. Após finalizar a manutenção, o técnico assinará a ordem de manutenção. Após isso, o operador do equipamento será notificado e a OM ficará disponível para sua avaliação, podendo este aceitar ou recusar de acordo com as realizações do trabalho. Caso recuse, o operador deve informar o motivo da rejeição ao sistema que então, irá reabrir a OM e solicitar ao técnico para verificar. Caso o operador aceite a execução do trabalho de manutenção, o mesmo deverá assinar a OM e com isto, o sistema notificará o líder técnico, sendo o trabalho deste, averiguar a execução da tarefa, o mesmo também poderá rejeitar e solicitar alterações. Após concluir, ele deve assinar a OM. Com as 3 assinaturas colhidas, o sistema irá notificar o usuário administrador e ele poderá liberar a OM para integração com o SAP.

Figura 3.2: Cenário Agil.It



Fonte: os autores (2020)

3.2 Principais Envolvidos e suas Características

3.2.1 Usuários do Sistema

O Sistema contemplará 4 tipos de usuários, o administrador, o técnico de manutenção, o chefe de manutenção e o operário. Cada usuário terá acessos e funções diferentes no sistema. Todos terão acesso tanto a parte web da aplicação quanto a parte mobile (aplicativo).

- O usuário administrador será responsável pelos cadastros gerais, poderá consultar e finalizar ordens de manutenção.

- O técnico de manutenção terá a visão de ordens de manutenção pendentes para ele, podendo iniciar ou pausar alguma a qualquer momento. Após finalizada, poderá realizar a assinatura da ordem.
- O chefe de manutenção receberá requisições de ordens de manutenção feitas por operadores e irá cadastrar ao sistema. Poderá distribuir as ordens aos técnicos, realizar consultas gerais e assinar as ordens.
- O operador, que poderá requisitar uma ordem de manutenção ao administrador, acompanhar as ordens solicitadas por ele e assinar a ordem após a conclusão.

3.2.2 Desenvolvedores do Sistema

Por trás de todo o desenvolvimento do sistema Agil.it, existe uma equipe de três pessoas formada por Júlio Thomazelli que atua como desenvolvedor na Voce Pede Softwares para Bares e Restaurantes há aproximadamente um ano e meio, com experiência nas tecnologias Angular, Ionic, Object Pascal e Java. A equipe também conta com o desenvolvedor e líder do setor de tickets da startup Clinicorp Solutions, trabalha como fullstack utilizando Node.js no back-end , React.js no front-end e React-Native para dispositivos mobiles há dois anos. Fechando a equipe, a Agil.it possui como gerente de projeto Márcio Henrique Meier, desenvolvedor na empresa Adapcon Processamento de Dados com grande experiência com as linguagens de programação Caché, Vue.js e Node.js, vivencia a área há exatamente três anos. Os três desenvolvedores são formados como Técnico em Desenvolvimento de Software e acadêmicos do sexto semestre do Tecnólogo em Sistemas para Internet na faculdade Senai de Jaraguá do Sul.

Tabela 3.1: Tabela de desenvolvedores Agil.it

Desenvolvedor	Atividade
Julio	Aplicativo Mobile
Lucas	Aplicativo Web
Márcio	Back End / Integração

Fonte: os autores (2020)

3.3 Regras de Negócio

No contexto da Engenharia de Software as Regras de Negócios são tratadas como alguns requisitos de software, pois sem elas, o software não existiria. Regras de negócio são premissas e restrições aplicadas a uma operação comercial de uma empresa, que precisam ser atendidas para que o negócio funcione da maneira esperada (CRERIE, 2008).

Segundo (PÁDUA, 2001) Regras de negócios são mais que declarações sobre campos de dados ou implementação do sistema, elas definem tarefas dos atores da organização, os serviços que a organização dispõem e os recursos necessários para apoiar os processos do negócio.

Já (KILOV; SIMMONDS, 1998) são mais enfáticos, para eles uma regra de negócio deve ser objetiva e definida em termos de notações matemáticas de proposições, mostrando que precisão é essencial quando formula-se as regras de negócio. Uma proposição é um fato ou estado observável de negócio envolvendo uma ou mais entidades pelo qual é possível afirmar ou negar a veracidade dessas entidades.

Desta forma as regras de negócio da empresa de Alimentos Duas Rodas seriam:

- Os administradores, líderes de manutenção e líderes de setor poderão criar ordens de manutenção;
- Apenas o administrador poderá finalizar ordens de manutenção;
- Os cadastros só serão possíveis na versão web;
- Cada ordem de manutenção deve conter uma assinatura do técnico de manutenção, uma do operador ou líder de setor e uma do administrador;
- Cada ordem de manutenção só pode ser finalizada após ter as 3 assinaturas;
- Somente administradores terão acesso à tela de pendência de assinaturas;
- Cada técnico de manutenção só pode ter uma ordem iniciada por vez;
- Técnicos de manutenção podem pausar ordens de manutenção;
- Técnicos de manutenção podem ter várias ordens pausadas ao mesmo tempo;
- Pode haver mais de um técnico de manutenção por ordem de manutenção;
- Uma ordem deve ter um técnico de manutenção principal;

4 PESQUISA DE ANTERIORIDADE

Durante a análise de requisitos, foram procurados outros sistemas que fazem esse processo de gerenciamento de ordem de manutenção para nos ajudar e entender quais são os problemas mais comuns desse ramo e quais as principais funcionalidades que o mercado oferece.

4.1 Produttivo

Produttivo é um sistema que tem vários módulos e diferentes aplicações. Um deles é o de Ordem de Serviço de Manutenção. Utilizando o Produttivo, um supervisor planeja e acompanha as atividades de manutenção utilizando o sistema web. Os técnicos de manutenção, através de um smartphone ou tablet, alimentam dados ao programa de acordo com os defeitos identificados, bem como a solução abordada. Na finalização da ordem de serviço de manutenção, o responsável realiza a assinatura digital. (PRODUTTIVO, 2019)

4.2 SoftByte

Visual Machine, da empresa SoftByte tem um fluxo bem mais simples quando comparado à Produttivo. Porém, possui algumas ferramentas interessantes, como o agendamento de manutenções preventivas e preditivas e possui um sistema de controle de investimentos em manutenção em cada um dos equipamentos. (SOFTBYTE, 2019)

4.3 ProdWin

ProdWin Pw-1 em seu módulo de manutenção promove controle do tempo gasto e dos materiais utilizados nas manutenções. Também dispõe de agendamento das manutenções, podendo relacioná-las a um técnico e ser acompanhadas em tempo real. O seu sistema de manutenção preditiva adiciona automaticamente a máquina na tela de alerta de manutenções, quando aproxima-se de seu limite operacional, conforme as informações previamente cadastradas. (PRODWIN, 2019)

4.4 Diferenciais Agil.It

Analizando os softwares encontrados, será implementado para o Agil.It o diferencial de ter integração entre sistemas, uma central de notificação e ser modelado a partir das necessidades da empresa Duas Rodas.

5 REQUISITOS DO SISTEMA

Os requisitos são capacidades que devem ser atendidas ou possuídas por um sistema para resolver um problema ou atingir um objetivo. O conjunto de todos os requisitos que formam a base para o desenvolvimento subsequente de um software (VAZQUEZ; SIMÕES, 2016). Neles são definidos como serão os comportamentos do sistema e seus fluxos. Eles foram abordados em dois segmentos: os requisitos funcionais que são fluxos do sistema e os requisitos não funcionais que são necessários para a utilização do sistema.

5.1 Requisitos Funcionais

Conforme (SOMMERVILLE, 2011), os requisitos funcionais de um sistema descrevem o que ele deve fazer, são dependentes do tipo de software a ser desenvolvido, de quem são seus possíveis usuários e da abordagem geral adotada pela organização ao escrever os requisitos. (REZENDE, 2005) define requisitos funcionais como necessidades descritas pelo cliente, onde a equipe do projeto analisa e especifica as funções, desempenho, interfaces e restrições, conforme as fases das metodologias aplicadas. Sua finalidade é distinguir as dependências do sistema para que o mesmo funcione de acordo com os requisitos informados pelo cliente. (SOMMERVILLE, 2011) acrescenta que quando os requisitos funcionais são expressos como requisitos de usuário, eles são normalmente descritos de forma abstrata, para serem compreendidos pelos usuários de sistema, entretanto, requisitos de sistema funcionais mais específicos descrevem em detalhes as funções do sistema como suas entradas e saídas.

A tabela 5.1 demonstra os requisitos funcionais do sistema Agil.It, nela foi feito a separação por tipo de registro, que é disposto de cadastro, consulta e funcionalidade e o requisito em si.

Tabela 5.1: Requisitos Funcionais do Sistema Agil.It

Tipo do Requisito	Requisito
Cadastro	Usuários
Cadastro	Centros de Trabalho
Cadastro	Setores
Cadastro	Locais de Instalação
Cadastro	Parametrizações de Segurança
Cadastro	Tipos de Máquina
Cadastro	Unidades de Medida
Cadastro	Peças
Cadastro	Equipamentos
Cadastro	Equipamentos Superior
Cadastro	Layouts de Ordem de Manutenção
Cadastro	Causas do Defeito
Cadastro	Sintomas do Defeito
Cadastro	Observações Padrão
Cadastro	Operações Padrão
Cadastro	Ordens de Manutenção
Consulta	Máquinas
Consulta	Ordens de Manutenção
Funcionalidade	Login
Funcionalidade	Logoff
Funcionalidade	Monitor de Ordens de Manutenção em Aberto
Funcionalidade	Solicitação de Abertura de Ordem de Manutenção
Funcionalidade	Central de Notificação

Fonte: os autores (2020)

5.2 Requisitos Não Funcionais

Requisitos não funcionais não estão diretamente ligados com os serviços específicos oferecidos pelo sistema a seus usuários. Eles podem estar relacionados às propriedades emergentes do sistema, como confiabilidade, tempo de resposta e ocupação de área (SOMMERVILLE, 2011). O propósito dos requisitos não funcionais é descrever as qualidades requeridas para um sistema, como sua usabilidade e seu desempenho (TORONTO, 2005). Em sistemas alguns destes requisitos podem determinar tecnologias ou algoritmos específicos a serem utilizados, garantindo compatibilidade com sistemas existentes (MARTINS, 2007).

A tabela 5.2 demonstra os requisitos não funcionais do sistema Agil.It que foram definidos em conjunto com a empresa Duas Rodas.

Tabela 5.2: Requisitos Não Funcionais do Sistema Agil.It

Requisito
Sistema Desenvolvido para Web
Versão Mobile para os Técnicos
Utilizar o Banco de Dados MS Sql Server
Usuários Devem ter Acesso à Computadores e/ou Dispositivos Móveis

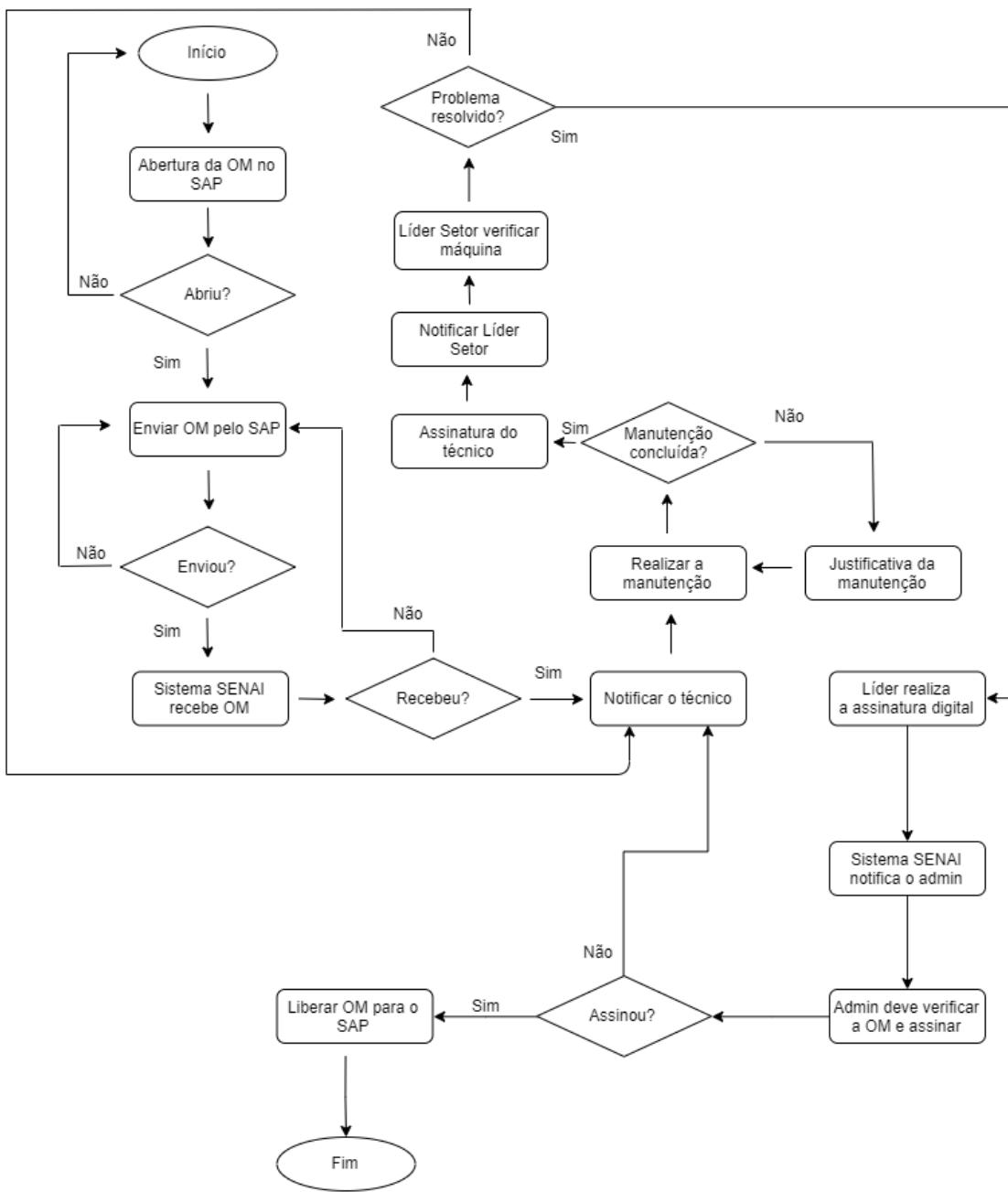
Fonte: os autores (2020)

5.3 Fluxograma do Sistema Desenvolvido

Segundo (GASQUES, 2002) fluxograma é uma representação gráfica das tarefas de um determinado processo, sendo de forma sequencial a execução delas. (HURT, 2014) relata que um fluxograma fornece um panorama de alto nível sobre um sistema de informação.

As formas geométricas têm a finalidade de direcionar o fluxo da informação ao usuário enquanto as linhas e setas descrevem a sequência das atividades, mostrando o caminho da informação de modo estruturado e as transformações à medida que os dados vão se movimentando desde a entrada até a saída (SOUZA, 2017).

Figura 5.1: Fluxo do AGIL.IT



Fonte: os autores (2020)

Na Figura 5.1 é possível verificar todo o fluxo do sistema onde começa e termina com a integração do SAP. O sistema irá receber a OM do SAP e notificar o técnico responsável pela execução do serviço. Após o técnico realizar a manutenção, os usuários (técnico, e líder do setor) realizam as assinaturas e o usuário administrador analisa e libera a OM para a integração.

5.4 Diagrama de Caso de Uso do Sistema

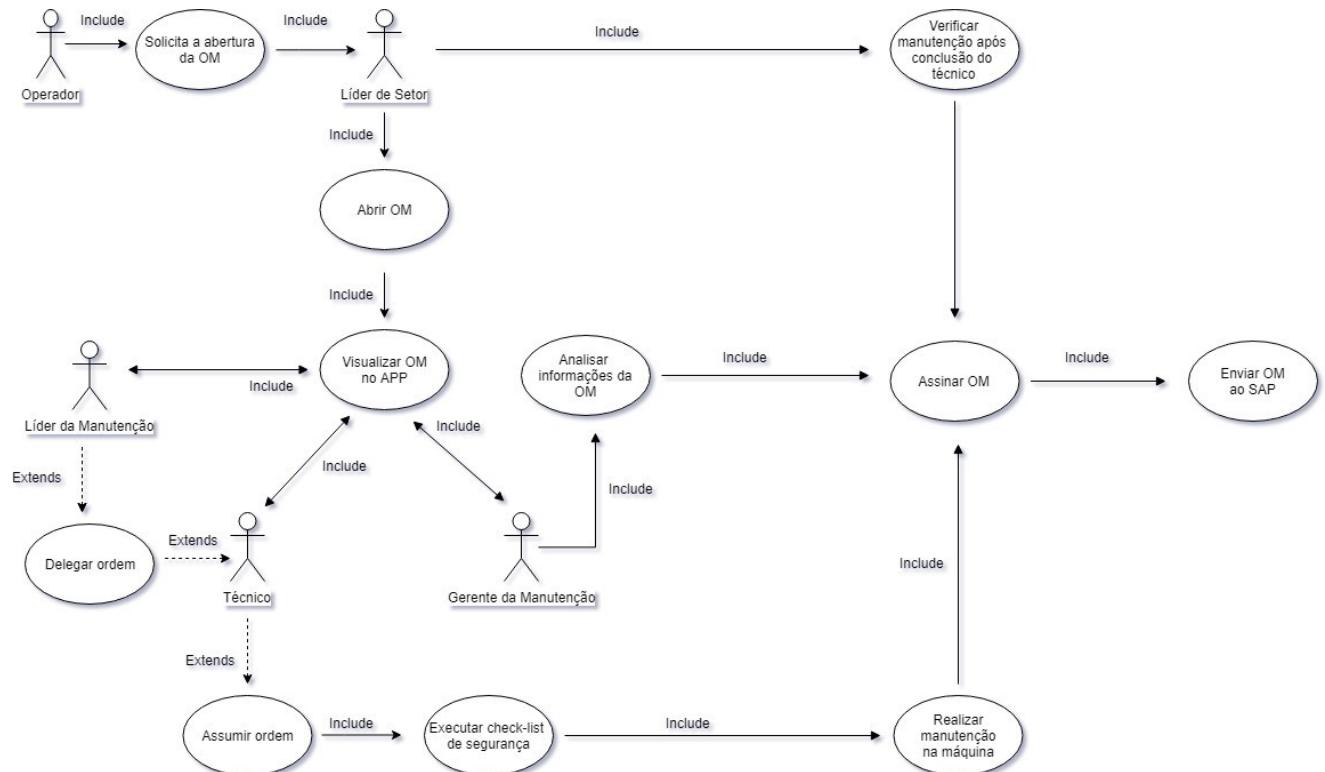
Diagramas dos casos de uso são técnicas utilizadas para captar os requisitos funcionais de um sistema, descrevem as interações entre usuários de um sistema e o próprio sistema,

fornecendo uma narrativa sobre como ele é utilizado (FOWLER, 2005).

Para (CARNIELLO, 2003) seu objetivo principal consiste em definir o comportamento de um sistema, sem revelar sua estrutura interna.

Em sua forma mais simples, um caso de uso identifica os atores envolvidos em uma interação e dá nome ao tipo de interação, ela pode ser complementada por informações adicionais que descrevem a interação com o sistema. Essas informações adicionais podem ser descritas textualmente ou por modelos gráficos (SOMMERVILLE, 2011).

Figura 5.2: Caso de Uso do AGIL.IT



Fonte: os autores (2020)

Na Figura 5.2 é possível verificar quais ações cada ator desempenhará no sistema. Os operadores de máquinas e os líderes de manutenção verificam a manutenção feita e assinam a OM. O técnico realizará a manutenção e verificará as peças necessárias para realizar a assistência ao equipamento e se as mesmas possuem estoque disponível. E o administrador realizará cadastros e encerrará as ordens de manutenção.

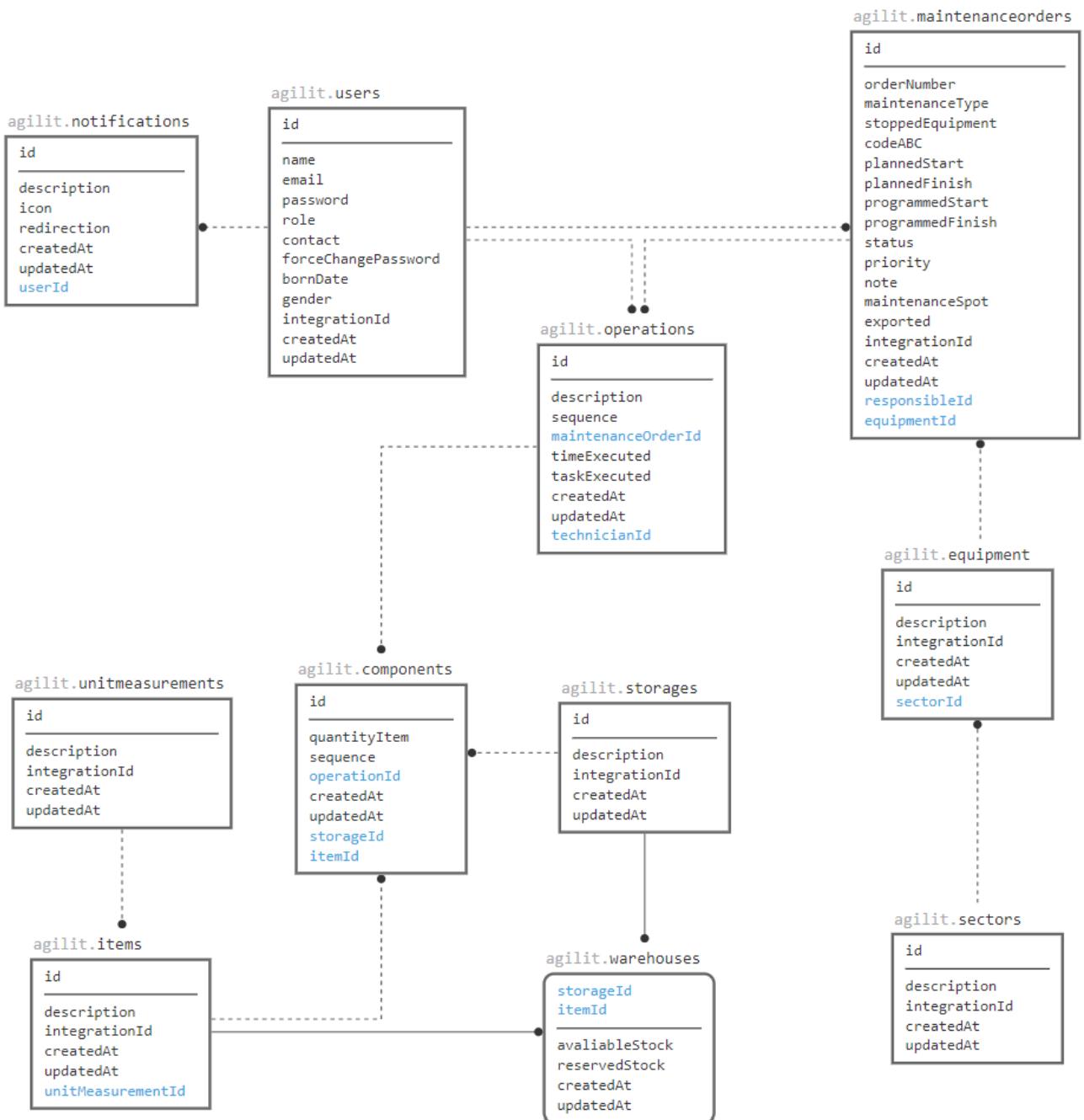
5.5 Modelo Entidade-Relacionamento do Banco de Dados

Utilizar o modelo entidade-relacionamento, tem como objetivo, obter uma descrição de forma abstrata dos dados que serão armazenados no banco de dados (ROCHA; TERRA, 2010).

Como o próprio nome sugere, o modelo entidade-relacionamento é composto por entidades e relacionamentos. Entidades possuem um nome e um ou mais atributos que podem ser

compostos ou simples, enquanto os relacionamentos são utilizados para montar a ligação entre as entidades (NOGUERA, 2018). A entidade pode ser forte, se não depender de outra para existir ou fraca, se sua existência no modelo estiver condicionada à presença de outra entidade (DANTAS, 2016).

Figura 5.3: Entidade de Relacionamento



Fonte: os autores (2020)

A Figura 5.3 mostra toda a entidade de relacionamento do banco de dados, onde cada tabela representa uma estrutura de dados no banco de dados e as ligações entre elas demonstram relacionamentos que essas tabelas possuem entre si.

6 PLANEJAMENTO DO SISTEMA

O planejamento do sistema é crucial para o sucesso do projeto. Nele deve-se estipular cronogramas de desenvolvimento e fazer a análise de risco para calcular a probabilidade e impacto de problemas que possam ocorrer durante o desenvolvimento do projeto. Um bom planejamento também deve ter uma análise de projeto que envolva recursos de diversas naturezas bem como humanos, capitais, técnicos e tecnológicos. Outro ponto importante é a prototipação do sistema para ter uma avaliação inicial com os *stakeholders* e ter uma base a ser seguida durante o desenvolvimento.

6.1 Cronograma do Projeto

Todo bom projeto deve ter um cronograma e um planejamento de ação, baseado nisso, foi elaborado quatro cronogramas, um para cada semestre, no qual aborda os conteúdos apresentados em cada semestre.

As figuras a seguir, são cronogramas que demonstram a trajetória do desenvolvimento do projeto. Divididos por semestre, os cronogramas listam as atividades aprendidas nas UCs, com todas as partes teóricas e práticas onde o objetivo principal é implementar todo conhecimento adquirido em prol de um projeto único: A unificação de todas as UCs do curso de Sistemas para Internet.

Figura 6.1: Cronograma de Aprendizagem: Terceiro Semestre

	02/2019	03/2019	04/2019	05/2019	06/2019	07/2019
Análise de Requisitos de Sistema						
Desenvolvimento de Canvas						
Apresentação do Projeto Duas Rodas						
Desenvolvimento Mobile						
Desenvolvimento de Casos de Uso						
Desenvolvimento de Modelo de Classes						
Desenvolvimento de APIs						
Desenvolvimento WEB						
Mapeamento de Regras de Negócio						
Mapeamento de Riscos do Projeto						
Desenvolvimento de Protótipos						

Figura 6.2: Cronograma de Aprendizagem: Quarto Semestre

	07/2019	08/2019	09/2019	10/2019	11/2019	12/2019
Análise de Requisitos de Sistema						
Desenvolvimento de Canvas						
Apresentação do Projeto Duas Rodas						
Desenvolvimento Mobile						
Desenvolvimento de Casos de Uso						
Desenvolvimento de Modelo de Classes						
Desenvolvimento de APIs						
Desenvolvimento WEB						
Mapeamento de Regras de Negócio						
Mapeamento de Riscos do Projeto						
Desenvolvimento de Protótipos						

Figura 6.3: Cronograma de Aprendizagem: Quinto Semestre

	02/2020	03/2020	04/2020	05/2020	06/2020	07/2020
Características e Aplicações do Software						
Análise de Processos						
Diagrama dos Fluxos de Dados						
Validação de Requisitos						
Estruturação do Sistemas						
Arquitetura de Objetos Distribuídos						
Padronização de Desenvolvimento						
Objetos e Classes de Objetos						
Desenvolvimento Baseado em Componente						
Apresentação das Informações						
5S						
Teste de Usuário						
Teste para Detecção de Defeitos						
Medições e Métricas das Funcionalidades do Software						
Técnicas de Recuperação do Software em Quedas						

Figura 6.4: Cronograma de Aprendizagem: Sexto Semestre

	07/2020	08/2020	09/2020	10/2020	11/2020
Desenvolvimento de Regras de Negócio					
Desenvolvimento de Melhorias de Usabilidade					
Desenvolvimento de Melhorias na Segurança do Sistema					
Desenvolvimento de Integração					
Testes de Interface					

6.2 Análise de riscos

A análise de riscos tem como objetivo identificar os possíveis problemas durante e após o desenvolvimento do projeto de modo a elaborar um plano de ação para solucionar rapidamente o problema de fato (SCHMITZ; ALENCAR, 2012).

Segundo (FILHO, 2003), um grande volume de dados publicados aponta para os riscos que correm os projetos de software executados sem a utilização de processos adequados. Um

levantamento publicado, a partir de uma base de dados de 4000 projetos, constatou a ocorrência frequente dos seguintes problemas:

- 70% dos projetos de grandes aplicativos sofre de instabilidade dos requisitos. Os requisitos crescem tipicamente cerca de 1% ao mês, atingindo níveis de mais de 25% de inchaço ao final do projeto.
- Pelo menos 50% dos projetos são executados com níveis de produtividade abaixo do normal.
- Pelo menos 25% do software de prateleira e 50% dos produtos feitos por encomenda apresentam níveis de defeitos superiores ao razoável.
- Produtos feitos sob pressão de prazos podem quadruplicar o número de defeitos.
- Pelo menos 50% dos grandes projetos de software estouraram seu orçamento e seu prazo.

Sendo assim, foi identificado os fatores de risco, no qual o projeto em questão possa estar exposto, conforme tabela 6.1 abaixo. Nela, apresenta-se uma análise do impacto e probabilidade de fatores prejudiciais ao projeto.

Tabela 6.1: Tabela de Riscos Agil.it

Riscos	Probabilidade	Impacto
Mudança de escopo	90%	2
Entrega no prazo	70%	3
Integração com SAP	70%	2
Implantação na empresa	60%	2
Conexão com o banco de dados	60%	3
Aceitação da usabilidade do sistema	50%	2
Usuários inexperientes	40%	2
Mudanças na tecnologia	20%	3
Segurança dos dados	15%	2
Conexão com a rede	10%	2
Falta de profissionais	5%	3

Fonte: os autores (2020)

Na tabela 6.1 estão mapeados os principais riscos identificados para o projeto Agil.it. Nela, a probabilidade indica a chance do risco ocorrer, as cores acompanham a porcentagem da mesma, sendo 70% ou mais a cor vermelha, entre 40% e 69% a cor amarela e de 0% a 39% a cor verde e o impacto é uma escala de um a três (1-3) do quanto o risco pode afetar a conclusão e entrega do projeto. A seguir será abordado o diagrama de causa e efeito.

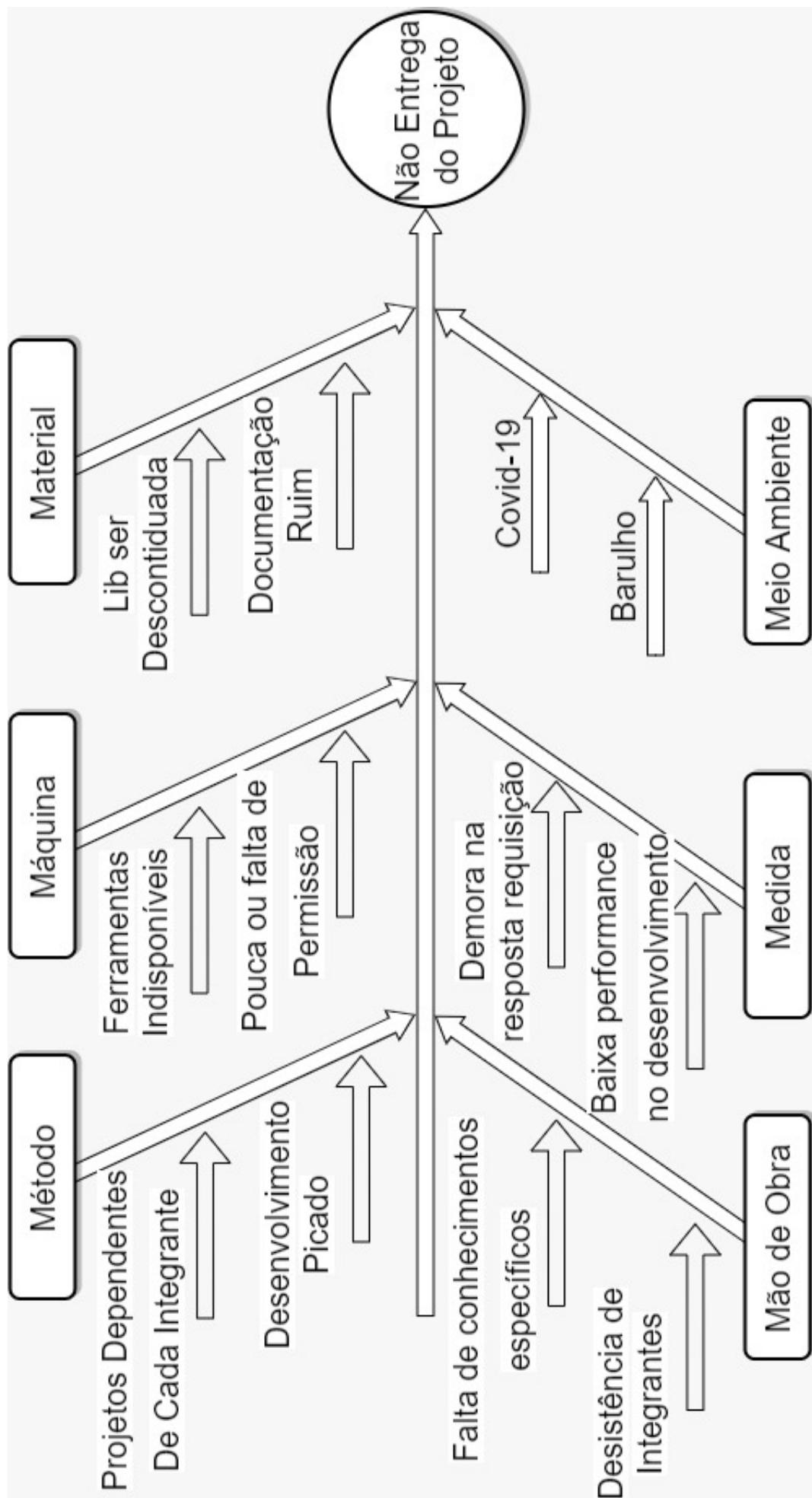
6.2.1 Diagrama de Causa e Efeito

O diagrama de Ishikawa, diagrama espinha de peixe ou diagrama de causa e efeito foi desenvolvido por Kaoru Ishikawa em 1943 e é definida como uma representação gráfica usada

como análise das causas de um determinado problema (SANTOS; BALANCIERI, 2017). O diagrama foi desenvolvido para consolidar os estudos realizados por uma fábrica e identificar as causas que deram início a ocorrência de um problema, por possibilitar a geração de melhorias e conhecimento do processo, os gestores utilizam amplamente (FILHO, 2017).

Com a forma de uma espinha de peixe, o modelo original sugeria quatro grandes grupos de causas que deveriam ser analisadas. Esses quatro grupos (também conhecidos como quatro M's) eram: materiais, mão de obra, métodos e máquinas. Versões mais recentes desse diagrama sugerem a análise orientada por seis grandes grupos de causas: materiais, mão de obra, métodos, máquinas, medidas e meio ambiente (SELNER, 1999).

Figura 6.5: Diagrama de Causa e Efeito



Fonte: os autores (2020)

6.3 PMBOK

PMBOK é uma metodologia de gerenciamento de projeto internacionalmente reconhecida, essas práticas podem auxiliar na resolução dos recursos humanos, capitais, tecnológicos e técnicos. Além disso, utilizando PMBOK é possível gerir melhor o andamento do projeto e de forma mais coordenada. Segundo (VARGAS, 2018) o PMBOK tem conhecimentos já comprovados que são amplamente utilizados, assim como o conhecimento de práticas mais inovadoras e avançadas. Dentre as técnicas de planejamento, pode-se utilizar para estudos e desenvolvimento a EAP - Estrutura Analítica de Projetos, descrita a seguir.

6.3.1 Estrutura Analítica do Projeto

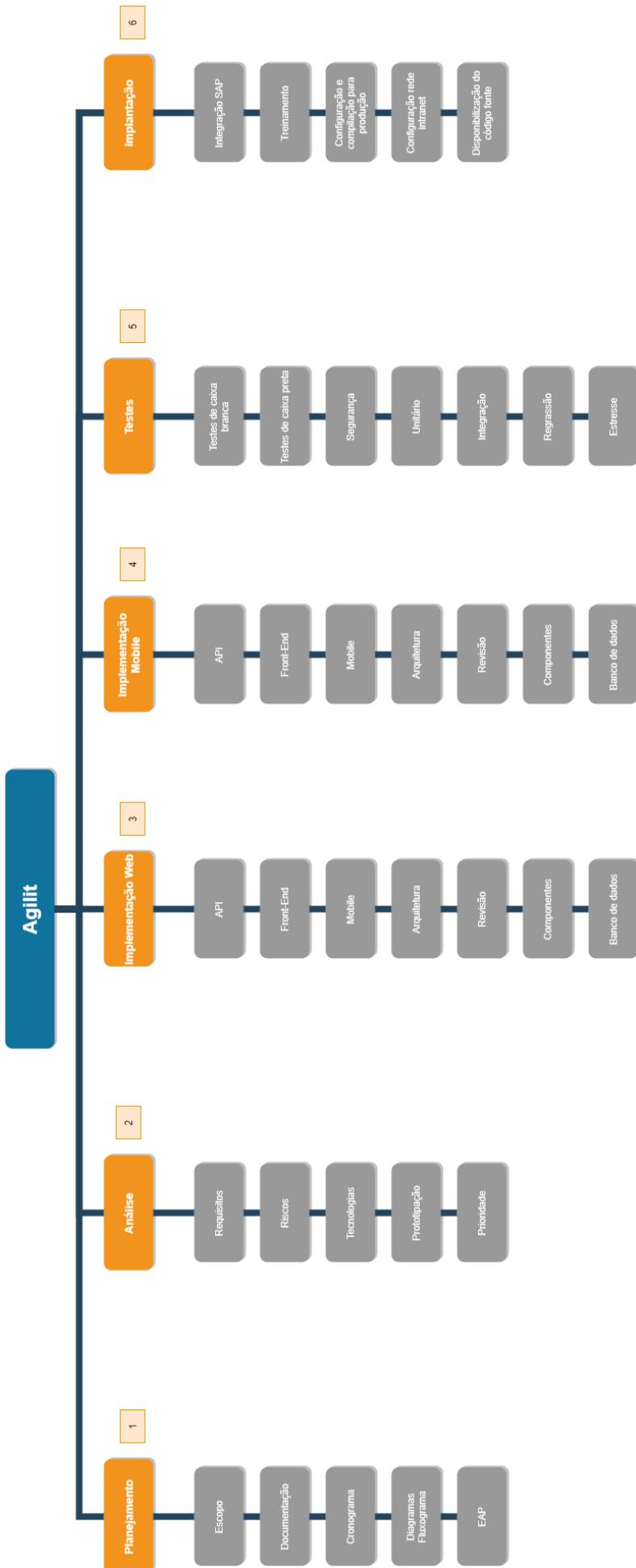
A Estrutura Analítica do Projeto (EAP) é a divisão estruturada de trabalho do projeto dividido em faixas gerenciadas cuja sua totalidade significa em um entregável ao projeto final.

Segundo (INSTITUTE, 2018) o detalhamento da EAP deve chegar até o nível do pacote de trabalho, nível mais baixo na EAP, que é o ponto no qual o custo e o cronograma do trabalho podem ser estimados de forma confiável. Porém o nível de detalhamento desse pacote varia de acordo com a complexidade de cada projeto. (KERZNER, 2017) defende que a EAP deve ser composta por até três níveis, pois se for detalhado demais o custo com o gerenciamento serão também excessivos.

Para o projeto Agil-it foi decidido utilizar 6 faixas de entregáveis:

- Planejamento;
- Análise;
- Implementação Web;
- Implementação Mobile;
- Testes;
- Implantação.

Figura 6.6: EAP AGIL.IT



Fonte: os autores (2020)

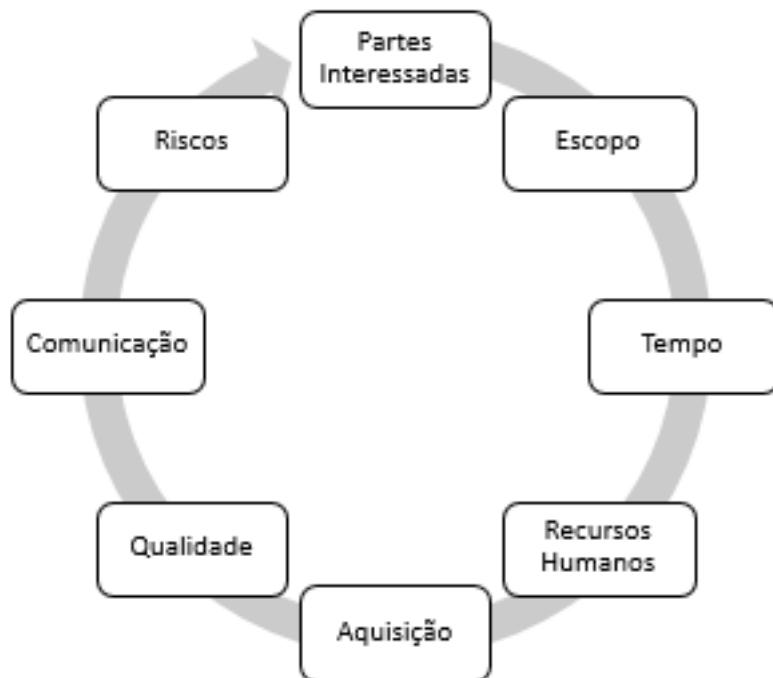
A figura 6.6 mostra a estrutura e o desenvolvimento que cada faixa requer no projeto. Ela não segue uma ordem cronológica, portanto não é necessário finalizar uma faixa para começar outra, muitas vezes elas são desenvolvidas em conjunto para uma melhor utilização do tempo de projeto.

6.3.2 Fluxo de Processos

O fluxo de processo é subdividido em cinco fases principais sobre o fluxo do projeto, iniciação sendo a primeira seguido de planejamento, execução, monitoramento, controle e finalização. Para (VARGAS, 2018) um projeto é desenvolvido a partir de uma ideia, em seguida parte para um plano e após isso é executado e concluído.

Para a definição dos processos, foi utilizado os seguintes recursos:

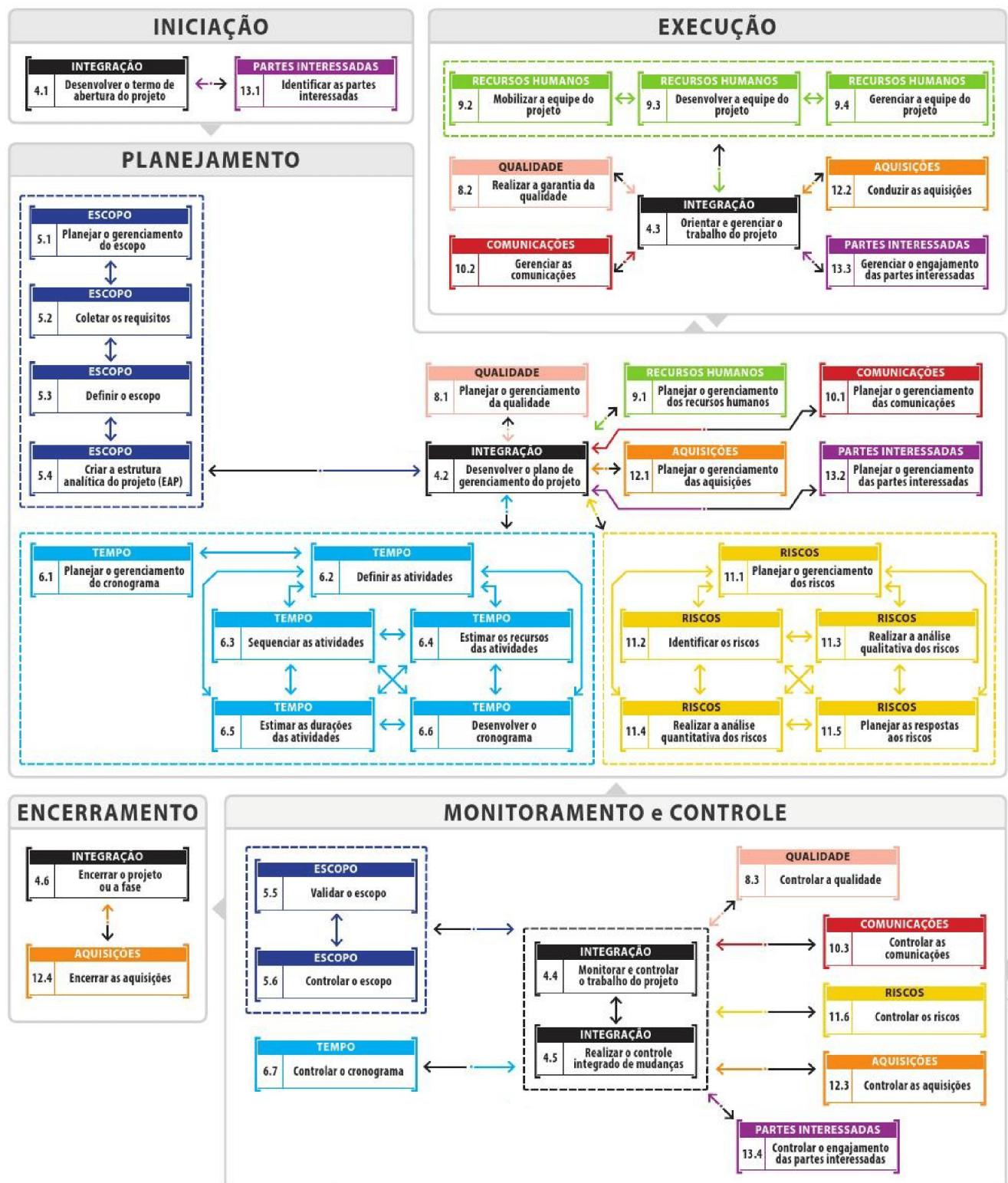
Figura 6.7: Fluxo e Processos



Fonte: (VARGAS, 2018)

A figura abaixo contém todos os fluxos do projeto, no qual aborda as fases de iniciação, planejamento, execução, monitoramento e controle, e por fim, o encerramento do projeto. Possui também fluxos de integração entre as fases, no qual são destacados em preto.

Figura 6.8: Fluxo e Processos AGIL.IT



Fonte: os autores (2020)

Iniciação - Esta é a fase inicial do projeto, nela é determinada a necessidade do projeto, com seus objetivos e justificativas, então é desenvolvido os documentos iniciais onde as

melhores estratégias são identificadas e selecionadas.

Planejamento - Fase onde deve ser detalhado minuciosamente tudo aquilo que será realizado no projeto, desde cronogramas, interdependências entre atividades, alocação dos recursos envolvidos, análise de custos, etc. Esta parte é muito importante pois a execução do projeto será baseado nas atividades para que sejam executadas sem dificuldades e imprevistos.

Execução - Nesta fase é onde tudo que foi planejado anteriormente se torna realidade, tendo que ser executado e realizado conforme planejado, qualquer erro cometido nas fases anteriores fica evidente durante essa fase.

Monitoramento e Controle - Esta fase acontece paralelamente às demais fases do projeto, acompanhando e controlando aquilo que está sendo realizado no projeto como um todo, podendo propor ações corretivas e preventivas no menor espaço de tempo possível após a detecção de erros.

Encerramento - Esta fase consiste em formalizar o fechamento do projeto, balanço e registro de erros e acertos, a fim de se preparar melhor para os próximos empreendimentos. Recomenda-se catalogar esses ensinamentos em um documento para que fique arquivado na empresa.

6.4 Protótipo

A prototipação é uma etapa de suma importância no desenvolvimento de projeto de software. Além de melhorar a produtividade da equipe, ela facilita o entendimento dos requisitos do sistema e permite a apresentação de conceitos e funcionalidades da aplicação de modo simplificado. Nesse trabalho foi utilizado a prototipação visual cujo ênfase se aplica a estética e usabilidade. Nesse tipo de protótipo é possível identificar o layout e a identidade visual da aplicação. (DEXTRA, 2019)

Protótipos podem ser gerados de acordo com as seguintes categorias (COYETTE, 2004): protótipos em baixa fidelidade que focam na interação, em componentes de interface e na estrutura geral do sistema; protótipos em alta fidelidade que produzem uma imagem real do sistema; protótipos executáveis que produzem o código em uma linguagem de programação, focando em navegação, mas sem ainda levar em consideração as regras de negócio. Cada categoria serve para um propósito específico: protótipos em baixa fidelidade são úteis para demonstrar aos usuários quais atividades o sistema atende e as possibilidades de navegação no sistema, assim como para proporcionar uma visão geral do sistema. Protótipos em alta fidelidade são úteis para demonstrar padrões e guias de estilo. Protótipos executáveis são úteis para demonstrar navegação e testar o uso da interface (ROSEMBERG, 2008). Seguindo as definições, o projeto desenvolvido utiliza os protótipos de baixa fidelidade

6.5 Aplicação WEB

A aplicação web tem como foco o gerenciamento de toda a aplicação envolvendo consultas e cadastros gerais do sistema. Apesar desse foco acentuado à gestão, é possível desempenhar todos os papéis dentro da aplicação web.

6.5.1 Cadastro de Usuário

Figura 6.9: Cadastro de Usuários

Cadastro de Usuário

Código do usuário

Nome do usuário

Perfil usuário

Gerar Senha

Trocar Senha no Primeiro Login

Senha

Email

Contato

Data Nascimento

Gênero

Masculino Feminino

Cancelar

Salvar

Fonte: os autores (2020)

A tela 6.9 é utilizada para cadastrar usuários no sistema. Caso queira atualizar um registro basta colocar o código dele no primeiro campo ou pesquisar no ícone de lupa. Para cadastrar um novo basta deixar o primeiro campo vazio.

6.5.2 Monitor de Ordem de Manutenção: Cards

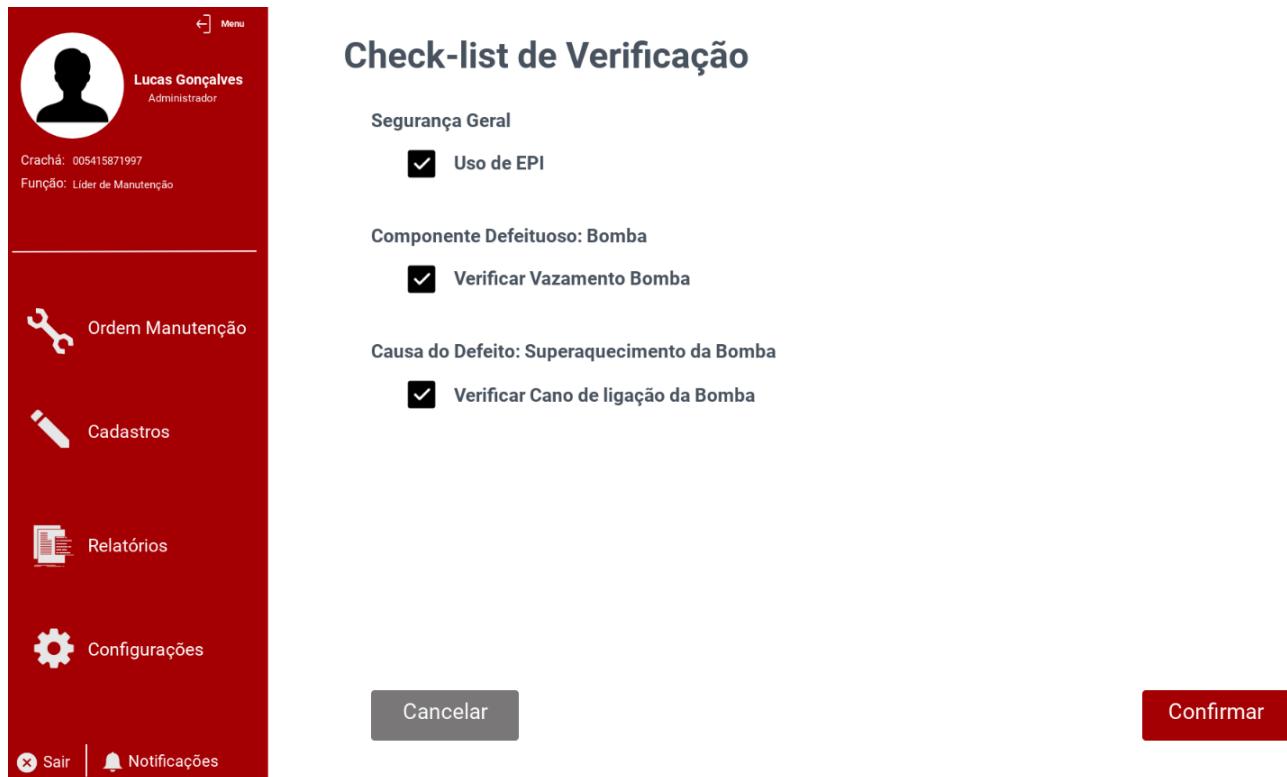
Figura 6.10: Monitor de Ordem de Manutenção: Cards

Fonte: os autores (2020)

A tela 6.10 permite a consulta rápida e dinâmica das ordens de manutenção. Nela você pode aplicar os filtros de acordo com as necessidades e será listada em forma de cartões, eles darão acesso à uma tela de detalhamento de ordem de manutenção.

6.5.3 Checklist de Segurança

Figura 6.11: Checklist de Segurança



Fonte: os autores (2020)

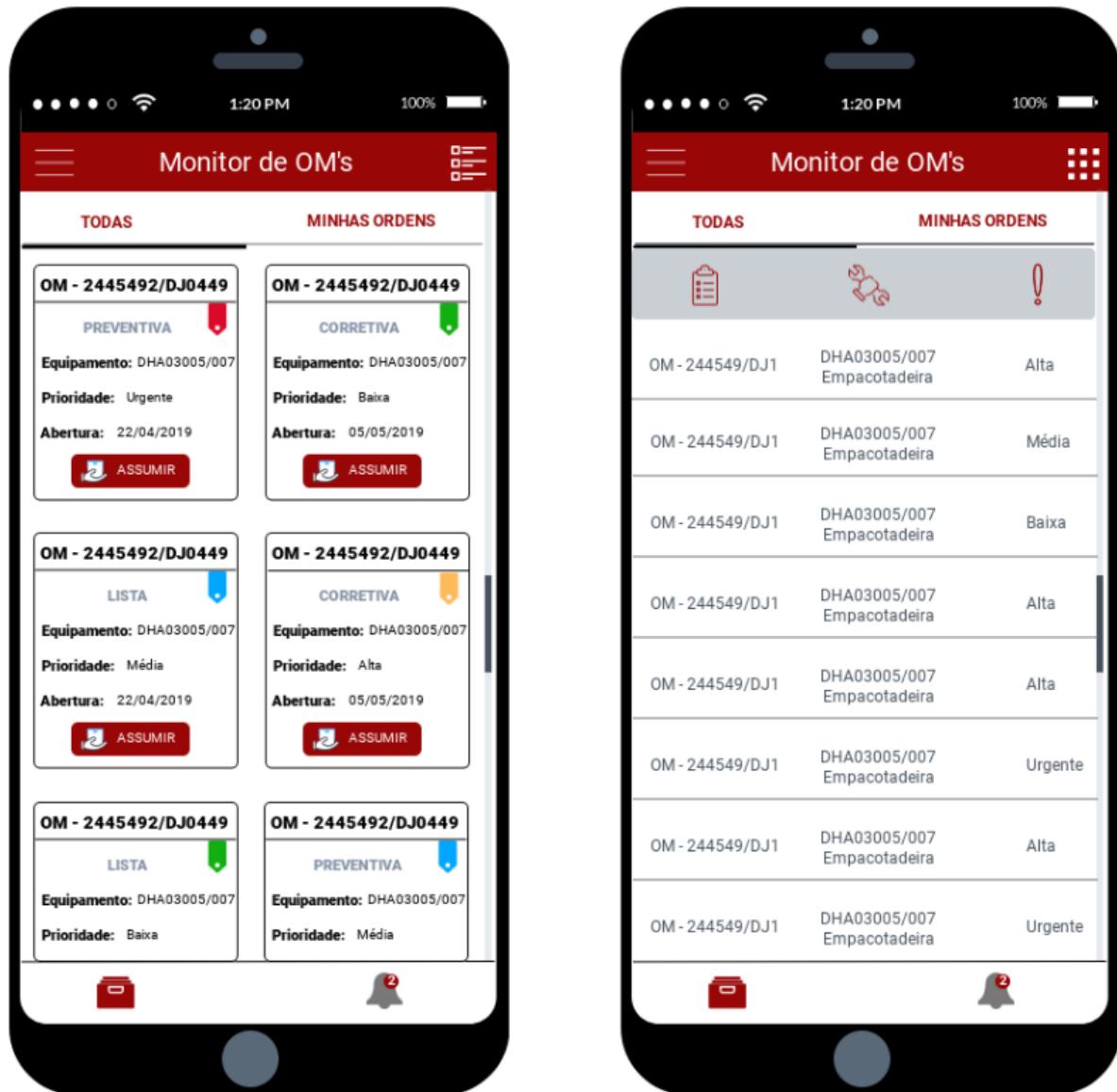
Na tela 6.11 o técnico irá marcar a lista de segurança antes de iniciar a ordem de manutenção. Essa listá será gerada dinamicamente de acordo com a parametrização de segurança cadastrado no sistema.

6.6 Aplicação Mobile

A aplicação Mobile é um ponto estratégico do produto, pois sua mobilidade permite com que os técnicos possam atuar na manutenção e realizar anotações e apontamentos no sistema através de um smartphone ou tablet.

6.6.1 Monitor

Figura 6.12: Monitor



Fonte: os autores (2020)

Na figura 6.12 é possível verificar o monitor do técnico de manutenção. Nesse monitor, o técnico consegue rapidamente visualizar as OMs pendentes e seus respectivos status através das bandeiras indicadas nos cartões. As vermelhas indicam que a OM tem uma prioridade urgente, as amarelas têm prioridade alta, as azuis têm prioridade média e as verdes possuem uma prioridade baixa.

6.6.2 Central de Notificações

Figura 6.13: Notificações

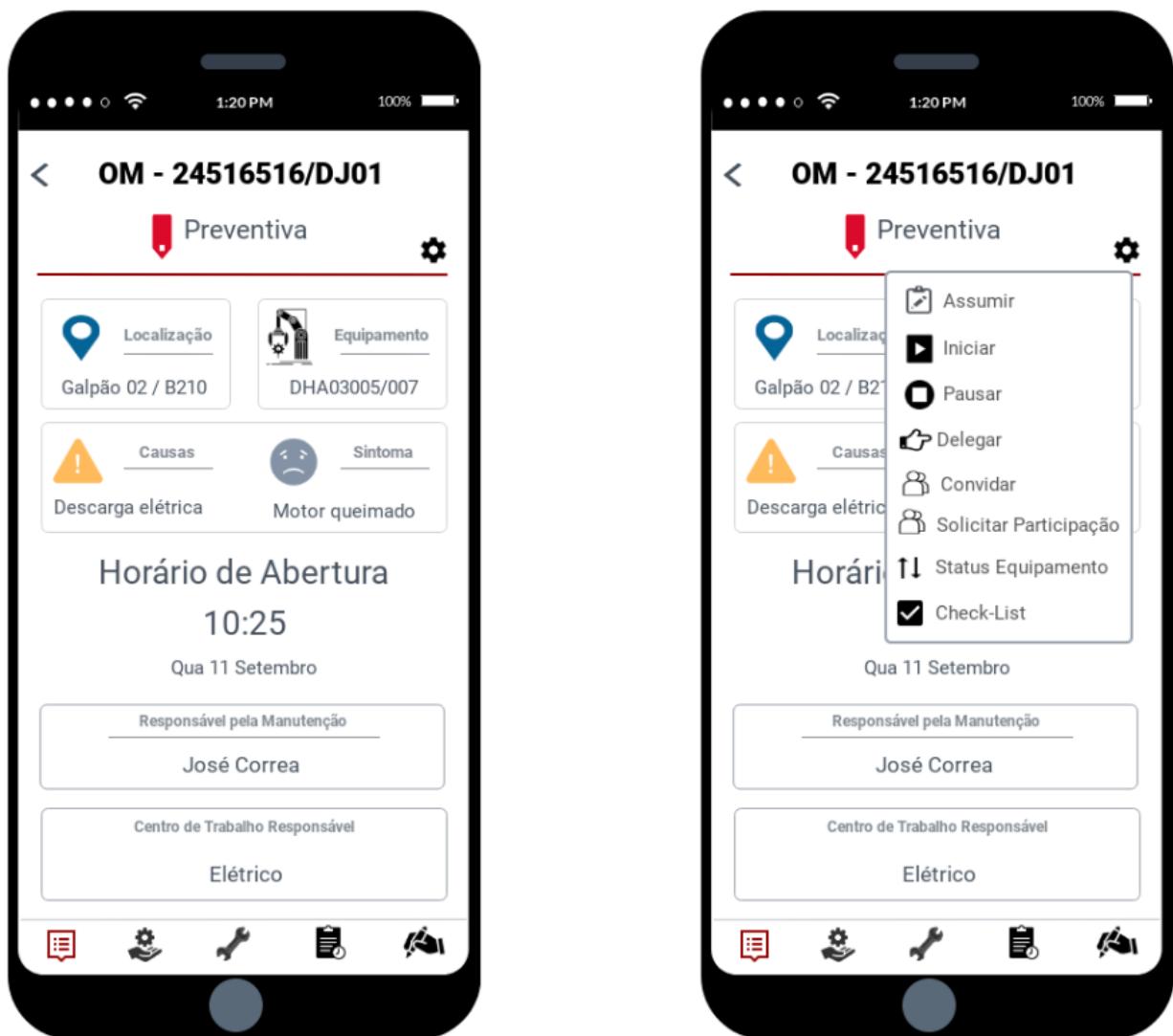


Fonte: os autores (2020)

Na tela 6.13 é possível verificar notificações do usuário autenticado no sistema.

6.6.3 Ordem de Manutenção

Figura 6.14: Ordem de Manutenção



Fonte: os autores (2020)

Na tela 6.14 é possível acompanhar o andamento de uma ordem de manutenção preventiva e corretiva, ver informações referente à ordem e executar ações nela, como alterar status, adicionar operações e realizar assinaturas.

7 IMPLEMENTAÇÃO

Nessa etapa, o sistema codificado a partir da descrição computacional das fases de projeto, como visto nos capítulos anteriores, conta com três aplicações: WEB, Mobile e Servidor. As aplicações foram desenvolvidas ao longo de três semestres no qual disponibilizou-se períodos de implementação e apresentação aos envolvidos, sendo estes a Faculdade de Tecnologia Senai e a Indústria Duas Rodas com o objetivo de alinhar o desenvolvimento e as expectativas do software em relação ao processo de manutenção executado atualmente.

7.1 Tecnologias

Para o desenvolvimento do sistema, foram divididas em três aplicações: uma visando a parte WEB, outra visando dispositivos móveis com os sistemas Android e IOS e para efetuar a comunicação entre os sistemas e o banco dados, foi elaborado uma aplicação de servidor cujo valida as requisições e efetua a comunicação entre as aplicações visíveis e o banco de dados.

7.1.1 Aplicação WEB

Para o desenvolvimento foi utilizada as tecnologias pré requisitadas pela empresa Duas Rodas, sendo elas HTML5¹, SASS² e como complemento, utilizamos a biblioteca ReactJS³.

HTML5 - É uma linguagem de marcação utilizada para desenvolvimento Web, esta nova versão traz consigo importantes mudanças quanto ao papel do HTML no mundo da Web, através de novas funcionalidades como semântica e acessibilidade.

SASS - É uma folha de estilo interpretada e compilada em Cascading style sheets, ao ser interpretado é criado blocos de códigos de regras CSS. Resumidamente é utilizada para fazer o Design do sistema, com cores personalizadas, etc.

ReactJs - O React é uma biblioteca JavaScript de código aberto com foco em criar interfaces de usuário em páginas web.

7.1.2 Aplicação Mobile

Utilizando as tecnologias Ionic 4⁴ e TypeScript⁵ esta aplicação é desenvolvida paralelamente com as demais aplicações, tem como objetivo a mobilidade, facilidade de acesso e interação podendo ser utilizado em qualquer lugar dentro da empresa.

TypeScript - TypeScript é um superconjunto de JavaScript desenvolvido pela Microsoft que adiciona tipagem e alguns outros recursos a linguagem. É utilizada somente pelos desen-

¹ <<https://html.spec.whatwg.org/>>

² <<https://sass-lang.com/>>

³ <<https://reactjs.org/>>

⁴ <<https://ionicframework.com/>>

⁵ <<https://www.typescriptlang.org/>>

volvedores, pois auxilia na construção do código fonte. Seu código é transpilado para JavaScript, no final das contas tudo escrito em TypeScript vira JavaScript.

Ionic 4 - O Ionic é um *framework* de código aberto para desenvolvimento de aplicativos móveis multiplataforma.

7.1.3 Aplicação do Servidor

A aplicação do servidor é o canal centralizador das demais plataformas. Ela tem acesso único e exclusivo ao banco de dados. É responsável por todas as validações e executa regras de negócio. O servidor foi desenvolvido em Typescript e utiliza a TypeOrm⁶ para integração com o banco de dados, para a API utiliza o Express⁷.

TypeOrm - TypeOrm é uma ORM para Typescript, utiliza injeção de dependências para modelar as classes e transformar em tabelas.

Express - Express é uma biblioteca que abstrai a camada HTTP e gerencia requisições recebidas pelo protocolo.

7.2 Códigos Desenvolvidos

Nesta etapa será apresentado alguns trechos dos códigos implementados para ilustrar o funcionamento e o entendimento de partes específicas do código fonte, onde os mesmos contém regras complexas e de suma importância ao leitor.

7.2.1 WEB

7.2.1.1 Componentes

A plataforma web da Agil.It tem como padrão componentizar os elementos exibidos no *front-end*, para isso, utiliza-se a biblioteca *React-md*, com componentes acessíveis por propriedades (parâmetros) e totalmente personalizáveis. Os estilos podem ser configurados tanto em tempo de compilação quanto em tempo de execução pelas variáveis SASS configuráveis e pelo uso de CSS. A biblioteca deve ser instalada no projeto e importada no código fonte conforme é mostrado no código 7.1.

Listing 7.1 Importando o componente de Ícone do react-md

```
1 import React, { PureComponent } from 'react';
2 import { FontIcon } from 'react-md';

4 export class C_Icon extends React.Component {
5   constructor(props) {
6     super(props);
7   }
}
```

⁶ <<https://typeorm.io/>>

⁷ <<https://expressjs.com/>>

```

9   render() {
10     return (
11       <FontIcon
12         className={this.props.className}
13         primary={this.props.primary}
14         forceFontSize={!!this.props.iconSize}
15         secondary={this.props.secondary}
16         forceSize={this.props.iconSize}
17         label={this.props.label}
18         style={this.props.style}
19         onClick={this.props.action}
20         disabled={this.props.disabled}
21         name={this.props.name}
22       >
23         {this.props.icon}
24       </FontIcon>
25     );
26   }
27 }

```

7.2.1.2 Cookies

Ao efetuar o login na plataforma, as informações do usuário, exceto sua senha, são armazenadas nos *cookies* através da biblioteca *universal-cookie* e são manipuladas para ocultar telas de determinados usuários de acordo com o seu perfil ou exibir alguma tela adicional de controle e análise, caso o perfil logado seja de um administrador. Outro exemplo seria impedir/permitir que tal usuário possa executar determinada ação ocultando elementos na tela de acordo com seu perfil. Com isso, o controle de informações e ações que o sistema oferece se torna muito mais fácil.

Após cada requisição feita ao servidor, o mesmo retorna um novo *token* renovado e então atualiza-se o *token* e usuário do *cookie*. Conforme demonstrado no código 7.2

Listing 7.2 Gravar dados no cookie

```

1 setToken(token) {
2   const user = JWTHelper.decomposeJwt(token)
3
4   this.cookies.set('token', token, { path: '/' })
5   this.cookies.set('user', user, { path: '/' })
6 }

```

Para recuperar os valores dos *cookies* basta instanciar o *universal-cookies* e utilizar o método *get*. Para remover o dado basta utilizar o método *remove*, de acordo com o código 7.3.

Listing 7.3 Recuperar dados do cookie

```

1 class App extends Component {
2   constructor() {
3     super()

```

```

5     this.cookies = new Cookies();
6
7     this.state = {
8         token: this.cookies.get('token'),
9         user: this.cookies.get('user'),
10    };
11 }

13 onLogout() {
14     this.cookies.remove('token');
15     this.setState({ token: undefined })
16 }
17 }

```

7.2.1.3 Helpers

Com o objetivo de agilizar o desenvolvimento do projeto, foram implementadas diversas classes de ajuda como *DateHelper*, *MaintenanceOrderHelpers* e *SearchModel*.

DateHelper é responsável por manipular e retornar as datas no formato desejado, possui funções que podem ser chamadas em qualquer classe para auxiliar o time de desenvolvimento no ganho de tempo. Como manipular datas é trabalhoso, segue um exemplo de como deixar um pouco mais simples, como pode-se ver no código 7.4.

Listing 7.4 Função que formata a data em dia/mês/ano - horas/minutos

```

1 static formatDate(dateTime) {
2     var date = this.getDate(dateTime);
3
4     var day = StringHelper.JustifyLeft(date.getDate(), 2, 0);
5     var month = StringHelper.JustifyLeft(date.getMonth() + 1, 2, 0);
6     var year = date.getFullYear();
7     var hour = date.getHours();
8     var minutes = date.getMinutes();
9
10    return `${day}/${month}/${year} às ${hour}h${minutes}`;
11 }

```

A *MaintenanceOrderHelper* é responsável por auxiliar na manipulação dos dados referentes as ordens de manutenção, possui funções que fazem a tradução de algumas propriedades dos objetos que são salvos em inglês no servidor.

Código 7.5 exemplifica a utilização da função de tradução de propriedades da ordem na classe de Ordem de Manutenção.

Listing 7.5 Utilizando a tradução de propriedades da ordem de manutenção

```

1 <span>{HelperOM.translate("status", order.orderStatus)}</span>
2 <span>{HelperOM.translate("color", order.orderStatus)}</span>
3 <span>{HelperOM.translate("layout", order.orderLayout)}</span>
4 <span>{HelperOM.translate("priority", order.priority)}</span>

```

Código 7.6 demonstra a função que traduz as propriedades da Ordem.

Listing 7.6 Função de tradução

```
1 static translate(prop, value) {
2     let props;
4
5     if (prop == "priority") props = this.getPriority();
6     else if (prop == "status") props = this.getStatus();
7     else if (prop == "color") props = this.getColorPriority();
8     else if (prop == "layout") props = this.getLayoutType();
9     else return value;
10
11 }
12
13 static getLayoutType() {
14     return {
15         default: 'Corretiva | Preventiva',
16         route: 'ROTA',
17         list: 'LISTA',
18     }
19 }
20
21 static getColorPriority() {
22     return {
23         low: "#03a140",
24         medium: "#3177e8",
25         high: "#ffd300",
26         urgent: "red",
27     }
28 }
29
30 static getPriority() {
31     return {
32         low: "Baixa",
33         medium: "Média",
34         high: "Alta",
35         urgent: "Urgente",
36     }
37 }
38
39 static getStatus() {
40     return {
41         created: "Aberta",
42         assumed: "Assumida",
43         started: "Iniciada",
44         paused: "Pausada",
45         stopped: "Parada",
46         canceled: "Cancelada",
47         "signature-pending": "Assinatura Pendente",
48         signatured: "Assinada",
49         finished: "Finalizada",
50         "no_status": "Sem Status",
51     };
52 }
```

Ainda no *MaintenanceOrderHelper* existem funções que ordenam as OM por prioridade e data de abertura na tela de monitoramento. As ordens serão ordenadas das urgentes às com baixa prioridade e como um segundo fator será a data de abertura, sendo as mais antigas mostradas primeiro.

Código 7.7 mostra a lógica utilizada para a ordenação.

Listing 7.7 Funções responsáveis pela ordenação

```

1 static sortOrders(list) {
2   var ordenatedPriority = this.ordenatedPriority();
3
4   return list.sort((a, b) => {
5     if (ordenatedPriority[a.priority] > ordenatedPriority[b.priority])
6       return -1;
7     else if (ordenatedPriority[a.priority] < ordenatedPriority[b.priority])
8       return 1;
9
10    var dateTimeA = DateHelper.getDate(a.openedDate).getTime();
11   var dateTimeB = DateHelper.getDate(b.openedDate).getTime();
12
13   if (dateTimeA > dateTimeB) return 1;
14   else if (dateTimeA == dateTimeB) return 0;
15   else return -1;
16 });
17
18
19 static ordenatedPriority() {
20   return {
21     urgent: 3,
22     high: 2,
23     medium: 1,
24     low: 0,
25   };
26 }

```

O *SearchModel* contém funções pré-configuradas das colunas que serão exibidas nas telas de CRUD afim de uma melhor visualização e organização dos dados exibidos. A configuração é composta por um *array* de objetos que possui três propriedades: *name*, *property* e *defaultValue*. A propriedade *name* será o cabeçalho da coluna da tabela, *property* é a propriedade da listagem que será renderizado na coluna e *defaultValue* é o valor que será apresentado caso não haja dados para a *property* na listagem.

Código 7.8 mostra a definição da configuração do *SearchModel*.

Listing 7.8 Definição da configuração para consulta de dados

```

1 const machineTypeColumns = () => [
2   {
3     name: "ID",
4     property: "id",
5     defaultValue: "Sem ID",
6   },

```

```

7   {
8     name: "Descrição",
9     property:"description",
10    defaultValue: "Sem Descrição",
11  },
12 ];
```



```

14 const equipmentColumns = () => [
15  {
16    name: "Código",
17    property:"code",
18    defaultValue: "Sem Código",
19  },
20  {
21    name: "Descrição",
22    property:"description",
23    defaultValue: "Sem Descrição",
24  },
25  {
26    name: "Tipo Máquina",
27    property:"machineType.description",
28    defaultValue: "Sem Tipo de Máquina",
29  },
30 ];
```

O código 7.9 mostra a utilização das funções de busca na classe de cadastro de equipamentos.

Listing 7.9 Tela para cadastro de equipamento

```

1 class CreateMachine extends Component {

3   constructor(props) {
4     super(props);

6     this.state = {
7       visible: true,
8       equipmentId: '',
9       machineType: '',
10      fields: {},
11      equipmentList: [],
12      machineTypeList: [],
13      equipmentColumns: equipmentColumns(),
14      machineTypeColumns: machineTypeColumns(),
15    };
16  };
17 }
```

Ao obter as colunas em um *state*, os dados são enviados ao componente *C_AutoComplete* que possui um ícone de lupa no qual o usuário pode clicar para visualizar todos os equipamentos cadastrados, conforme visto no código 7.10.

Listing 7.10 Utilizando o componente C_AutoComplete

```

1   <C_AutoComplete
2     id="id"
3     name="id"
4     value={this.state.equipmentId}
5     label={"Equipamento"}
6     rightIcon={"search"}
7     list={this.state.equipmentList}
8     searchColumns={this.state.equipmentColumns}
9   />

```

Código 7.11 mostra a função dentro do componente *C_AutoComplete* que exibe a tabela de consulta de acordo com as colunas e lista de dados enviados por parâmetro.

Listing 7.11 Componente C_AutoComplete: Função executada ao clicar na lupa

```

1   onClickIcon() {
2     if (!Array.isArray(this.props.searchColumns)) return;
3
4     confirmAlert({
5       customUI: ({ onClose }) => (
6         <C_SearchTable
7           columns={this.props.searchColumns}
8           content={this.props.list}
9           onClick={this.tableSelected}
10          extraFunction={onClose}
11        />
12      )
13    });
14  }

```

7.2.2 Mobile

7.2.2.1 Ações da Ordem de Manutenção

A estrutura da classe *AgilitActionUtils* é feita com o intuito de reutilizar códigos, sendo que cada tipo de OM deve ser assinada da mesma forma que qualquer outra, mudando apenas seu atributo identificador, a seguir seguem algumas características importantes desta classe.

A primeira característica desta classe é realizar as ações de uma determinada OM de forma centralizada, recebendo como parâmetro para qual OM deverá ser feita a ação e a ação propriamente dita, conforme mostra código 7.12.

Listing 7.12 Alterar situação da ordem de manutenção

```

1 public async changeStatus(orderID, orderStatus : AgilitOrderStatus){
2   return this.restOrder.orderActions(orderID, orderStatus);
3 }
4
5 public orderActions(orderID : number, agilitOrderStatus :
6   AgilitOrderStatus){
7   const orderStatus : any = {
8     ...
9   }
10  ...
11 }

```

```

7     orderStatus: agilitOrderStatus
8 }

10    if (agilitOrderStatus == AgilitOrderStatus.ASSUMED){
11      let userInfo : any = AgilitStorageUtils.getDataJSON(
12        AgilitStorageTypes.USERDATA);

13      orderStatus.userId = userInfo.id;
14    }

16    this.http.url = this.http.getBaseUrl() + this.restAction + '/'+ orderID
17      + '/status';
18  return ProviderHelper.put(this.http, orderStatus);
19 }

```

Outra característica muito importante desta classe é realizar as assinaturas de uma determinada OM, sendo passada como parâmetro a OM respectivamente e a senha de assinatura. Ao realizar uma assinatura a classe irá executar algumas validações da OM a fim de verificar se está de acordo para enviar ao servidor, com isso a promessa deve ser resolvida ou rejeitada. O código 7.13 demonstra a implementação.

Listing 7.13 Assinar a ordem de manutenção

```

1 public async signOrder(order, assignaturePassword): Promise<any>{
2   return new Promise(async (resolve, reject) => {
3     this.resolvePromise = resolve;
4     this.rejectPromise = reject;
5
6     try {
7       if (!this.validateOrder(order, assignaturePassword)){
8         return;
9     }
10
11    const user = AgilitStorageUtils.getDataJSON(
12      AgilitStorageTypes.USERDATA);
13
14    if (AgilitUtils.isNullOrUndefined(user) ||
15      AgilitUtils.isNullOrUndefined(user.id) || user.id == ''){
16      this.rejectPromise('Dados de usuário inválido!');
17      return;
18    }
19
20    await this.restOrder.orderAssignment(order.id, user.id).then(
21      (response: any) => {
22        if (AgilitUtils.isNullOrUndefined(response)){
23          return;
24        }
25        this.resolvePromise(response);
26      }.catch(
27        error => {
28          this.rejectPromise(error);
29        }
30      )
31    }
32  }
33 }

```

```

30      );
31  } catch (error) {
32      this.rejectPromise(error);
33  }
34 });
35 }
```

Para que uma OM seja assinada com sucesso a classe deve conter as validações dos estados atuais da OM a ser assinada, por exemplo, se o usuário realizar a assinatura de uma OM com status cancelada esta implementação tem de validar a mesma. Código 7.14 mostra a implementação de validação da ordem antes da assinatura.

Listing 7.14 Validar estados da OM

```

1 private validateOrder(order, password) : boolean{
2   if (password != AgilitStorageUtils.getData(AgilitStorageTypes.PASSWORD))
3     {
4       this.rejectPromise('Senha incorreta!');
5       return;
6     }
7
7   if (order.orderStatus == AgilitOrderStatus.SIGNATURED){
8     this.rejectPromise('OM já está assinada!');
9     return;
10  }
11
12  if (order.orderStatus == AgilitOrderStatus.FINISHED){
13    this.rejectPromise('OM já está assinada!');
14    return;
15  }
16
17  if (order.orderStatus == AgilitOrderStatus.CANCELED){
18    this.rejectPromise('OM está cancelada!');
19    return;
20  }
21
22  return true;
23 }
```

7.2.2.2 Dados Armazenados no Local Storage

Algumas informações, por exemplo, usuário e token de acesso devem ficar salvas em memória para posteriormente serem utilizadas novamente, com isso desenvolvemos a classe chamada *AgilitStorageUtils* contendo regras, limpezas específicas dos dados e separação dos tipos de dados que estão sendo armazenados e buscados.

O código 7.15 mostra os tipos de dados armazenados.

Listing 7.15 Tipos de dados armazenados

```

1 export enum AgilitStorageTypes{
2   USERDATA = 'user',
```

```
3   USERNAME = 'username',
4   PASSWORD = 'password',
5   TOKEN    = 'token'
6 }
```

O código 7.16 mostra as ações de limpeza, busca e gravação de dados.

Listing 7.16 Tipos de dados armazenados

```
2 // Remove de dados conforme o tipo especificado.
3 public static clearSpecificData(agilitStorageType : AgilitStorageTypes) {
4   window.localStorage.removeItem(agilitStorageType);
5 }

7 // Buscando dados de um tipo específico armazenado na memória do
     navegador.
8 public static getData(agilitStorageType : AgilitStorageTypes) {
9   return window.localStorage.getItem(agilitStorageType);
10 }

12 // Buscando dados de um tipo específico armazenado e convertendo para
     objeto.
13 public static getDataJSON(agilitStorageType : AgilitStorageTypes) {
14   return JSON.parse(window.localStorage.getItem(agilitStorageType));
15 }

17 // Este método faz uma verificação do tipo do dado que está sendo inserido
     na memória do navegador.
18 public static setData(agilitStorageType : AgilitStorageTypes, data :
     Object|Array<any>|string) {
19   if (AgilitUtils.isNullOrUndefined(data)){
20     return;
21   }

23   if (typeof data === 'object'){
24     window.localStorage.setItem(agilitStorageType, JSON.stringify(data));
25     return;
26   }

28   if (typeof data === 'string'){
29     window.localStorage.setItem(agilitStorageType, data);
30   }
31 }
```

7.2.3 Servidor

7.2.3.1 Mapeamento do Banco de Dados com *TypeOrm*

A *TypeOrm* trabalha com injeção de dependências, então em cada propriedade tem um *decorator* que vai definir o mapeamento dela. Para gerar uma *primary key* com auto increment, basta usar a injeção *PrimaryGeneratedColumn*, para uma coluna simples, apenas *column*, para

data de criação do registro, *CreateDateColumn* e para data de atualização do registro *UpdateDateColumn*. Com a *TypeOrm* é possível ter classes abstratas para abranger propriedades em comum no banco de dados, para tanto foi criado a classe *baseClass* que irá conter todos os campos em comum de todas as tabelas do banco de dados, conforme mostrado no código 7.17.

Listing 7.17 BaseClass: Mapeamento de propriedades de tabelas do banco de dados

```

1 export abstract class BaseClass {

3   @PrimaryGeneratedColumn()
4   public id: number | undefined = undefined;

6   @Column({ default: '' })
7   public integrationID: string = '';

9   @CreateDateColumn()
10  public createdAt: Date | undefined = undefined;

12  @Column()
13  public createdBy: number | undefined = undefined;

15  @UpdateDateColumn()
16  public updatedAt: Date | undefined = undefined;

18  @Column()
19  public updatedBy: number | undefined = undefined;

21  @Column({
22    type: Boolean,
23    default: false
24  })
25  public deleted: boolean = false;

27  constructor() {
28  }

30 }

```

Para facilitar as tabelas genéricas de CRUD, onde têm apenas os dados base e o campo de descrição, foi criado a classe abstrata *crudClass*, conforme mostrado no código 7.18.

Listing 7.18 CrudClass: Mapeamento de propriedades para CRUDs genéricos

```

1 export abstract class CrudClass extends BaseClass {

3   @Column({ nullable: false })
4   @IsNotEmpty({
5     message: 'Descrição: Campo obrigatório.'
6   })
7   public description: string = '';

9   constructor() {
10     super();
11   }

```

```
12 }
```

E por fim, o código 7.19 mostra a aplicação de uma classe que irá gerar de fato uma tabela no banco de dados.

Listing 7.19 Mapeamento de propriedades da tabela sector

```
1 @Entity('sector')
2 export class Sector extends CrudClass {
3
4   constructor() {
5     super();
6   }
7
8 }
```

No primeiro parâmetro da injeção *Entity* define-se o nome da tabela, e como a classe *Sector* estende a classe *CrudClass* que estende, subsequentemente a classe *BaseClass* a tabela terá todas as colunas definidas nas injeções de dependências mostradas anteriormente.

Para fazer relações entre as tabelas existem alguns *decorators* específicos: *ManyToOne* para relações muitos para um, *ManyToMany* para relações muitos para muitos, *OneToOne* para relações um para um e *OneToMany* para relações um para muitos.

O código 7.20 mostra a definição da relação *ManyToOne*.

Listing 7.20 Mapeamento de propriedades da tabela área de installation_area

```
1 @Entity("installation_area")
2 export class InstallationArea extends CrudClass {
3
4   @ManyToOne(type => Sector, sector => sector.id, { nullable: false })
5   @JoinColumn()
6   public sector: Sector = undefined;
7
8   constructor() {
9     super();
10  }
11
12 }
```

A *TypeOrm* tenta fazer a relação automaticamente pelo nome dos campos, porém, se for uma relação mais complexa é possível usar o *decorator JoinColumn* para definir qual coluna da tabela A se relaciona com qual coluna da tabela B e explicitar em qual das tabelas deve ficar a *foreign key*, conforme mostrado no código 7.21.

Listing 7.21 Mapeamento de propriedades da tabela maintenance_worker

```
1 @Entity('maintenance_worker')
2 export class MaintenanceWorker extends BaseClass {
3
4   @ManyToOne(type => User, user => user.id)
5 }
```

```

5   @JoinColumn()
6   public user : User;

8   @ManyToOne(type => MaintenanceOrder, maintenanceOrder =>
      maintenanceOrder.id, { cascade: false })
9   @JoinColumn()
10  public maintenanceOrder : MaintenanceOrder | undefined = undefined;

12  @Column()
13  public isMain: boolean = false;

15  @Column()
16  public isActive: boolean = false;

18  @OneToMany(type => WorkerRequest, workerRequest =>
      workerRequest.maintenanceWorker, { cascade: false })
19  public workerRequest: Array<WorkerRequest>;

21  @OneToMany(type => WorkedTime, workedTime =>
      workedTime.maintenanceWorker, { cascade: false })
22  public workedTime: Array<WorkedTime>;
23 }

```

7.2.3.2 Validação de Objetos com o Class Validator

Para validações de campos da tabela utiliza-se outra biblioteca com injeção de dependências que funciona muito bem quando mesclada com a *TypeOrm*, o *Class Validator*. Com ele pode-se utilizar, por exemplo o *decorator IsNotEmpty* na propriedade *description*. Esse *decorator* irá validar se a propriedade está vazia ou não, caso esteja, irá retornar o erro *Descrição: Campo obrigatório*. conforme definição no próprio *decorator* visto no código 7.22.

Listing 7.22 Definição de validação de propriedades com o Class Validator

```

1 @IsNotEmpty({
2   message: 'Descrição: Campo Obrigatório.',
3 })
4 public description: string = '';

```

E para executar a validação conforme as injeções feitas, é preciso importar a função *validate* do *class validator* e executar passando uma instancia do objeto.

Listing 7.23 Validação de Objetos com o Class Validator

```

1 async validate(entity: Entity) : Promise<any> {
2   const errors = await validate(entity);

4   if (errors.length === 0) {
5     return undefined;
6   }

8   let errorList = [];

```

```

10   errors.forEach(error => {
11     let constraints = error.constraints;
12
13     for (const key in constraints) {
14       if (constraints.hasOwnProperty(key)) {
15         errorList.push(constraints[key]);
16       }
17     }
18   });
19
20   return errorList;
21 }

```

O retorno vem em forma de *array* e traz diversas informações a respeito dos erros ocorridos, no caso da API, é mostrado apenas a mensagem de erro em si, então a listagem é filtrada para mostrar apenas os erros.

7.2.3.3 API

Na API é utilizado o *Express* para gerenciar as requisições feitas pela aplicação web, mobile e integração de sistemas externos. Foi adicionado os *plugins* *Cors* para lidar com o CORS, *BodyParser* para converter JSON e o *Helmet* para melhorar a segurança da aplicação.

No script de inicialização do servidor é estabelecido conexão com o banco de dados, instanciado a aplicação *Express* e instalado seus *plugins*, conforme visto no código 7.24

Listing 7.24 Script de inicialização do servidor

```

1 createConnection().then(async connection => {
2
3   // create express app
4   const app = express();
5   app.use(bodyParser.json());
6   app.use(cors({exposedHeaders: 'token'}));
7   app.use(helmet());
8
9   // Get all system routes
10  let routes = new Routes();
11
12  routes.getCollections().forEach((collection: Collection) => {
13    collection.getRoutes().forEach((route: Route) => {
14      // Register each route
15      route.registerRoute(app)
16    });
17  })
18
19
20  app.listen(4000);
21
22 }).catch(error => console.log(error));

```

O roteamento é composto por várias coleções de rotas, no qual cada coleção tem um conjunto de rotas que é composto pela url que a rota responderá, os métodos que aceitará e a função que será invocada. Por fim, o método *registerRoute* irá registrar a rota da coleção ao *express*. Esses métodos que as rotas respondem ficam no *package controller* que fazem a ponta entre receber a requisição, processar os dados recebidos e responder o que foi solicitado, seja cadastrar, atualizar, excluir ou consultar um ou mais registros. Da mesma forma que foi feita com os *models*, foi desenvolvido *CrudController*, uma classe que utiliza *Generics* que para cada instancia recebe o valor da entidade do CRUD e irá se adequar conforme o tipo fornecido. Essa classe também recebe no construtor o acesso ao repositório que será gravado, no caso, a tabela do banco de dados, conforme visto no código 7.25.

Listing 7.25 CrudController

```

1 export class CrudController<Entity> {

3   private repositoryEntity : Repository<Entity>;

5   constructor(repositoryEntity: Repository<Entity>) {
6     this.repositoryEntity = repositoryEntity;
7   }
}
```

Todas as rotas são protegidas, com exceção da rota de login. Essa proteção é feita através de um *middleware* colocado ao registrar a rota no *express*, esse *middleware* vai capturar os cabeçalhos *token* para autenticação de sessão e *authorization* para usuários de integração, caso nenhum dos cabeçalhos seja fornecido a API irá retornar o status “401 - Unauthorized”. Caso seja passado o cabeçalho *authorization* o *middleware* irá verificar se chave informada é válida e se está na base de dados de usuários de integração. Se passado *token*, o sistema irá decompor e validar o *token* utilizando a biblioteca *jsonwebtoken*, caso a validação do JWT informado falhe, seja por não estar com a assinatura válida da aplicação ou por ter expirado o tempo da utilização, retorna um erro para a API informando que o *token* não é válido. Caso a validação passe, o *token* é regerado com expiração renovada para 5 horas e é adicionado ao cabeçalho da resposta. Lembrando que, o *token* pode ser obtido pela primeira vez através da rota de Login.

O código 7.26 demonstra a validação de usuário e *token* feita pelo *middleware* da API.

Listing 7.26 Função para validação de autenticação da API

```

1 export const checkJwt = (request: Request, response: Response, next: NextFunction) => {

3   const authorization = <string>request.headers["authorization"];

5   if (authorization) {      // The Authorization was passed in so now we
     validate it
6     this.checkIntegration(authorization)
7     .then(valid => {
8       if (valid === true) {
9         return next();
10      } else {

```

```

11         return response.status(401).send();
12     }
13   }).catch(err => {
14     return response.status(500).send({
15       "success": false,
16       "error": err.message
17     });
18   })
19 } else {
20   //Get the jwt token from the request's header
21   const token = <string>request.headers["token"];
22   let jwtPayload;
23
24   //Try to validate the token and get data
25   try {
26     jwtPayload = <any>jwt.verify(token, JWT.jwtSecret);
27     response.locals.jwtPayload = jwtPayload;
28   } catch (error) {
29     //If token is not valid, respond with unauthorized
30     return response.status(401).send();
31   }
32
33   //set to response's header the new token
34   response.append('token', new UserController().generateJwtToken(<User>
35     jwtPayload));
36
37   next();
38 }
39 };

```

7.2.3.4 Estratégia de Exclusão de Dados

A estratégia para exclusão de dados adotado no projeto é não deletar os registros, mas alterar a coluna *deleted* para *true*. Desta maneira é possível rapidamente desfazer uma exclusão errada e também não ocorrerá problemas com histórico caso as entidades sejam deletadas.

Código 7.27 mostra lógica executada durante a “exclusão” de um registro.

Listing 7.27 Método para exclusão de registro

```

1 async removeEntity(entity: Entity) {
2   entity["deleted"] = true;
3
4   return this.getRepositoryEntity().save(entity)
5 }

```

8 AGIL.IT

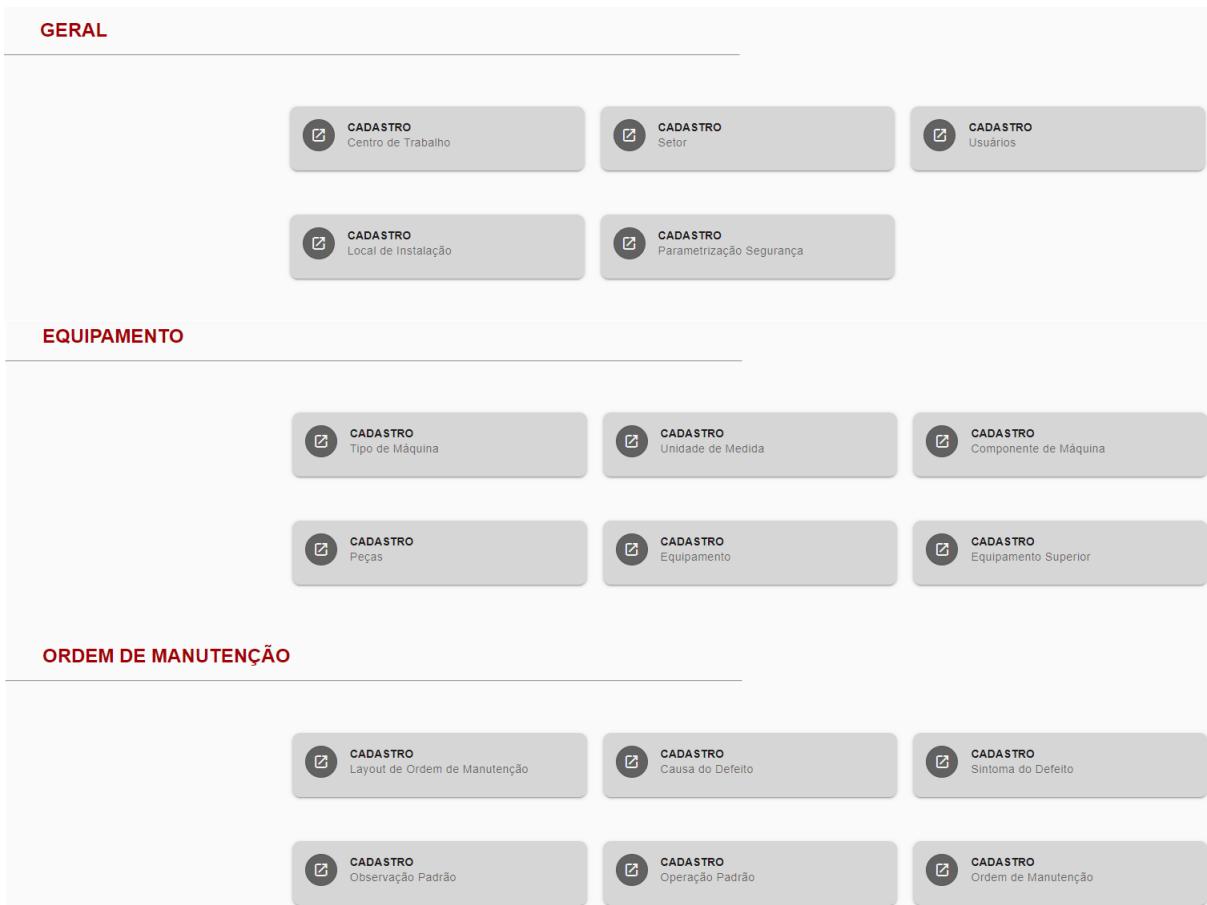
No capítulo anterior foi detalhado as principais características da estrutura e organização do código fonte do sistema. A seguir será apresentado as principais telas e funcionalidades disponíveis aos usuários.

8.1 WEB

A aplicação WEB tem seu foco na parte administrativa e gerencial do sistema, através dela o administrador poderá realizar os cadastros, assinaturas e acompanhar as situações de todas as ordens que estão em andamento.

8.1.1 Cadastros do sistema

Figura 8.1: Cadastros do Sistema

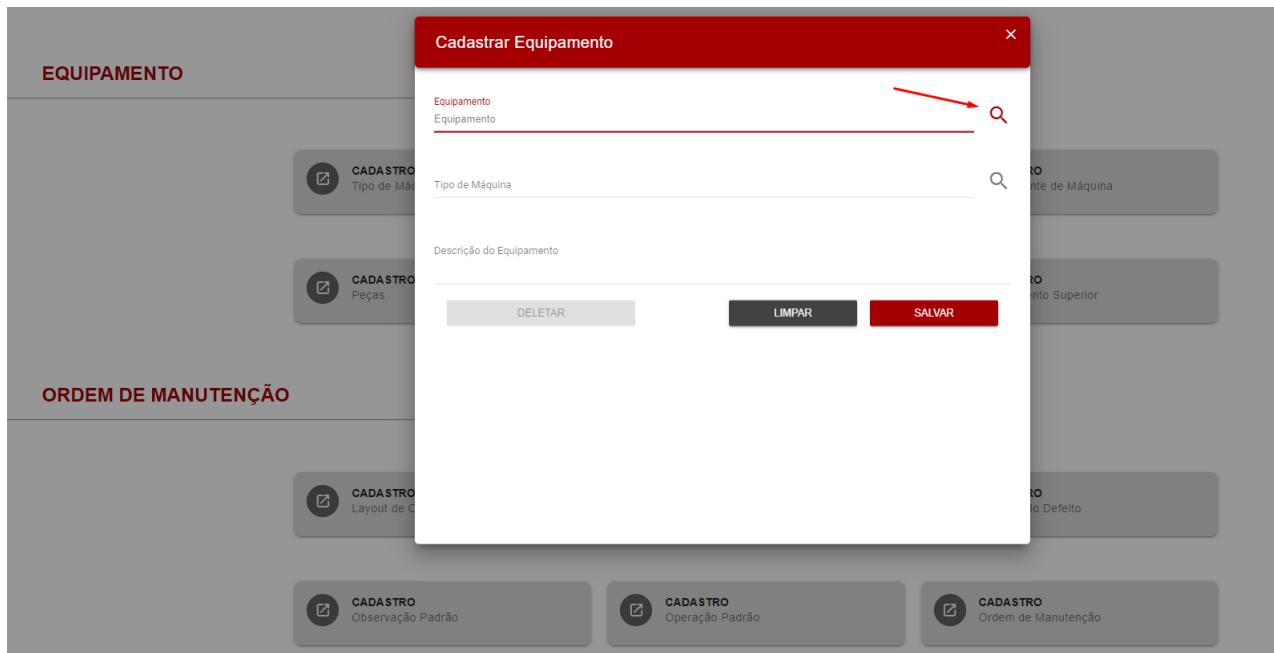


Fonte: os autores (2020)

A figura 8.1 mostra a tela de lista de cadastros disponíveis no sistema. Nela, foi separado cadastros em 3 grupos: geral, equipamento e ordem de manutenção. Nos cadastros gerais será

possível realizar cadastros básicos do sistema, como usuário, setor, parametrização de segurança, entre outros. Na parte de cadastros de equipamento são os cadastros relacionados aos equipamentos da empresa, como o próprio equipamento em si, o equipamento superior, o tipo de máquina, dentre outros. E por fim, os cadastros de ordens de manutenção que tem o cadastro da ordem de manutenção, sintoma e causa do defeito, layout da ordem, etc.

Figura 8.2: Cadastro de Equipamento



Fonte: os autores (2020)

A figura 8.2 mostra o cadastro de Equipamento, as demais telas de cadastro do sistema seguem o mesmo padrão, nas cores vermelho e branco e com campos de auto preenchimento conforme as informações são digitadas, os usuários conseguem efetuar a busca de dados de uma forma fácil e ágil. Os cadastros possuem também uma tela de consulta que fica visível ao clicar no ícone de lupa disponível no lado direito dos campos de auto preenchimento. Entretanto, a tela Cadastro de Ordem de Manutenção é a única que contém abas para facilitar no preenchimento das informações por ser um cadastro mais extenso e trabalhoso.

Na figura 8.3 mostra um exemplo de como fica a visualização da tela de consulta no cadastro de Equipamentos.

Figura 8.3: Consulta de Equipamento

Código	Descrição	Tipo Máquina
DRJ4684/000	SECADOR SPRAY DRYER MOD. DR (PILOTO)	SECADOR
DRJ3812/000	VENTILADOR MODELO VA 92/300-E	VENTILADOR
DRJ4884/000	CONDICIONADOR DE AR SPLIT - 18.000 BTU'S	CONDICIONADOR
DRJ2846/000	CORTINA DE AR	CORTINA
DRJ3774/000	TRANSFORMADOR WEG 500KVA	TRANSFORMADOR
DRJ3442/005	AQUECEDOR PARA ÓLEO - SPRAY-8	AQUECEDOR
DRJ5369/000	PENEIRA VIBRATÓRIA PRD 750 -SPR8 - SGSA	PENEIRA
DRJ921/000	COZINHADOR - LINHA SIF	COZINHADOR
DRJ3780/000	ARMAZEM VERTICAL SHUTTLE SH-40 N°2	ARMAZEM
DRJ5598/000	COMPRESSOR DE AR ATLAS COPCO GA90 VSD	COMPRESSOR
Filtrar tabela		Linhas 10 ▾ 1-10 of 283 < >

Fonte: os autores (2020)

8.1.2 Dashboard

Essa tela tem como objetivo facilitar o monitoramento das ordens de manutenção, com a disponibilidade de filtros por data, status e prioridade os usuários conseguem encontrar o que desejam de uma forma muito mais rápida. Além disso, a Dashboard conta com duas opções de visualização das ordens listadas em tela, a primeira opção seria em forma de tabela com um filtro adicional por texto, conforme figura 8.4 e a segunda em forma de cards que contém cores para identificar a prioridade das ordens, conforme figura 8.5.

Figura 8.4: Dashboard: Visualização por tabela

Monitor de Ordens de Manutenção

De 01/05/2020 Até 01/08/2020 Status ABERTA Prioridade TODAS LISTAR

Abertura	Descrição	Equipamento	Prioridade	Status
19/07/2020	Limpar o componente	CONDICIONADOR DE AR SPLIT - 18.000 BTU'S	Urgente	Aberta
03/05/2020	Verificar a integridade do componente	SECADOR SPRAY DRYER MOD. DR (PILOTO)	Alta	Aberta
08/05/2020	Soldar o componente	CONDICIONADOR DE AR SPLIT - 18.000 BTU'S	Alta	Aberta
11/05/2020	Apertar os parafusos	VENTILADOR MODELO VA 92/300-E	Alta	Aberta
18/06/2020	Verificar a integridade do componente	CONDICIONADOR DE AR SPLIT - 18.000 BTU'S	Alta	Aberta
18/07/2020	Apertar os parafusos	VENTILADOR MODELO VA 92/300-E	Alta	Aberta
22/07/2020	Apertar os parafusos	VENTILADOR MODELO VA 92/300-E	Alta	Aberta
22/06/2020	Trocar o componente	VENTILADOR MODELO VA 92/300-E	Média	Aberta
21/07/2020	Limpar o componente	SECADOR SPRAY DRYER MOD. DR (PILOTO)	Baixa	Aberta

Filtrar tabela Linhas 10 1-9 of 9

Fonte: os autores (2020)

Figura 8.5: Dashboard: Visualização por cards

Monitor de Ordens de Manutenção

De 01/05/2020 Até 01/08/2020 Status ABERTA Prioridade TODAS LISTAR

AGL-000183	AGL-000186	AGL-000088
Descrição: Limpar o componente Equipamento: CONDICIONADOR DE AR SPLIT - 18.000 BTU'S Prioridade: Urgente Abertura: 19/07/2020 Status: Aberta	Descrição: Verificar a integridade do componente Equipamento: SECADOR SPRAY DRYER MOD. DR (PILOTO) Prioridade: Alta Abertura: 03/05/2020 Status: Aberta	Descrição: Soldar o componente Equipamento: CONDICIONADOR DE AR SPLIT - 18.000 BTU'S Prioridade: Alta Abertura: 08/05/2020 Status: Aberta
AGL-000011	AGL-000040	AGL-000068
Descrição: Apertar os parafusos Equipamento: VENTILADOR MODELO VA 92/300-E Prioridade: Alta Abertura: 11/05/2020 Status: Aberta	Descrição: Verificar a integridade do componente Equipamento: CONDICIONADOR DE AR SPLIT - 18.000 BTU'S Prioridade: Alta Abertura: 18/06/2020 Status: Aberta	Descrição: Apertar os parafusos Equipamento: VENTILADOR MODELO VA 92/300-E Prioridade: Alta Abertura: 18/07/2020 Status: Aberta
AGL-000059	AGL-000027	AGL-000118
Descrição: Apertar os parafusos Equipamento: VENTILADOR MODELO VA 92/300-E Prioridade: Alta Abertura: 22/07/2020 Status: Aberta	Descrição: Trocar o componente Equipamento: VENTILADOR MODELO VA 92/300-E Prioridade: Média Abertura: 22/06/2020 Status: Aberta	Descrição: Limpar o componente Equipamento: SECADOR SPRAY DRYER MOD. DR (PILOTO) Prioridade: Baixa Abertura: 21/07/2020 Status: Aberta

Fonte: os autores (2020)

8.1.3 Análise de Pendências de Assinatura

Figura 8.6: Análise de pendências de assinatura

Ordem	Técnico	Solicitante	Administrador	Exportado
AGL-000006	●	●	●	●
AGL-000002	●	●	●	●
AGL-000013	●	●	●	●
AGL-000028	●	●	●	●
AGL-000033	●	●	●	●
AGL-000038	●	●	●	●
AGL-000050	●	●	●	●
AGL-000057	●	●	●	●
AGL-000065	●	●	●	●
AGL-000070	●	●	●	●

Fonte: os autores (2020)

A figura 8.6 mostra a tela de análise de pendências de assinatura que busca de forma automática todas as ordens de manutenção que estão com assinaturas pendentes seja pelo técnico, solicitante ou administrador. Os responsáveis pela ordem que ainda não assinaram são identificados pelo círculo na cor vermelha e os que já assinaram na cor verde. A ordem de manutenção irá permanecer na lista de pendências até ser exportada ao sistema SAP. Apenas o usuário com o perfil administrador tem acesso a essa tela.

8.1.4 Ordem de Manutenção

Figura 8.7: Ordem de Manutenção

The screenshot shows a maintenance order (AGL-000059) with the following details:

- Detalhes da Ordem:**
 - Descrição do Problema: Apertar os parafusos
 - Solicitante: Natália
 - Centro de Trabalho: Obras
 - Tipo: ZPM1
- Equipamentos:**
 - DRJ3812/000
 - Equipamento Superior: VENTILADOR MODELO VA 92/300-E (Status: OK)
 - Local de Instalação: DR-1000-1000-ALM-ALMOXARIFADO
 - Setor: ALMOXARIFADO
 - Tipo de Máquina: VENTILADOR
 - Causa do Defeito: Componente Frouxo
 - Sintoma do Defeito: Sobreaquecimento
 - Requer Parada?: NÃO

Fonte: os autores (2020)

Na figura 8.7 é possível visualizar todas as informações relacionadas a ordem e todas as ações executadas nela, como por exemplo, quem solicitou a abertura de determinada ordem, a data e hora de abertura, todos os equipamentos e operações realizadas em cada um deles. Além disso, no canto superior direito da tela existe um menu com uma série de ações, onde se encontra a opção para efetuar a assinatura da ordem de manutenção, a parte mais importante no fluxo do processo. Para solicitar a assinatura é necessário informar a senha do usuário logado e concordar com todas as informações existentes na ordem, com isso, os dados são enviados e analisados pelo servidor que retorna uma mensagem de sucesso ou erro de acordo com a análise feita, conforme mostra a figura 8.8.

Figura 8.8: Assinatura da Ordem de Manutenção

The screenshot shows a signature dialog box for maintenance order AGL-000059:

- ASSINATURA:**
 - Senha: [REDACTED]
 - Concordo com todas as informações preenchidas na Ordem.
- Botões:** CANCELAR, ASSINAR, ASSINATURA

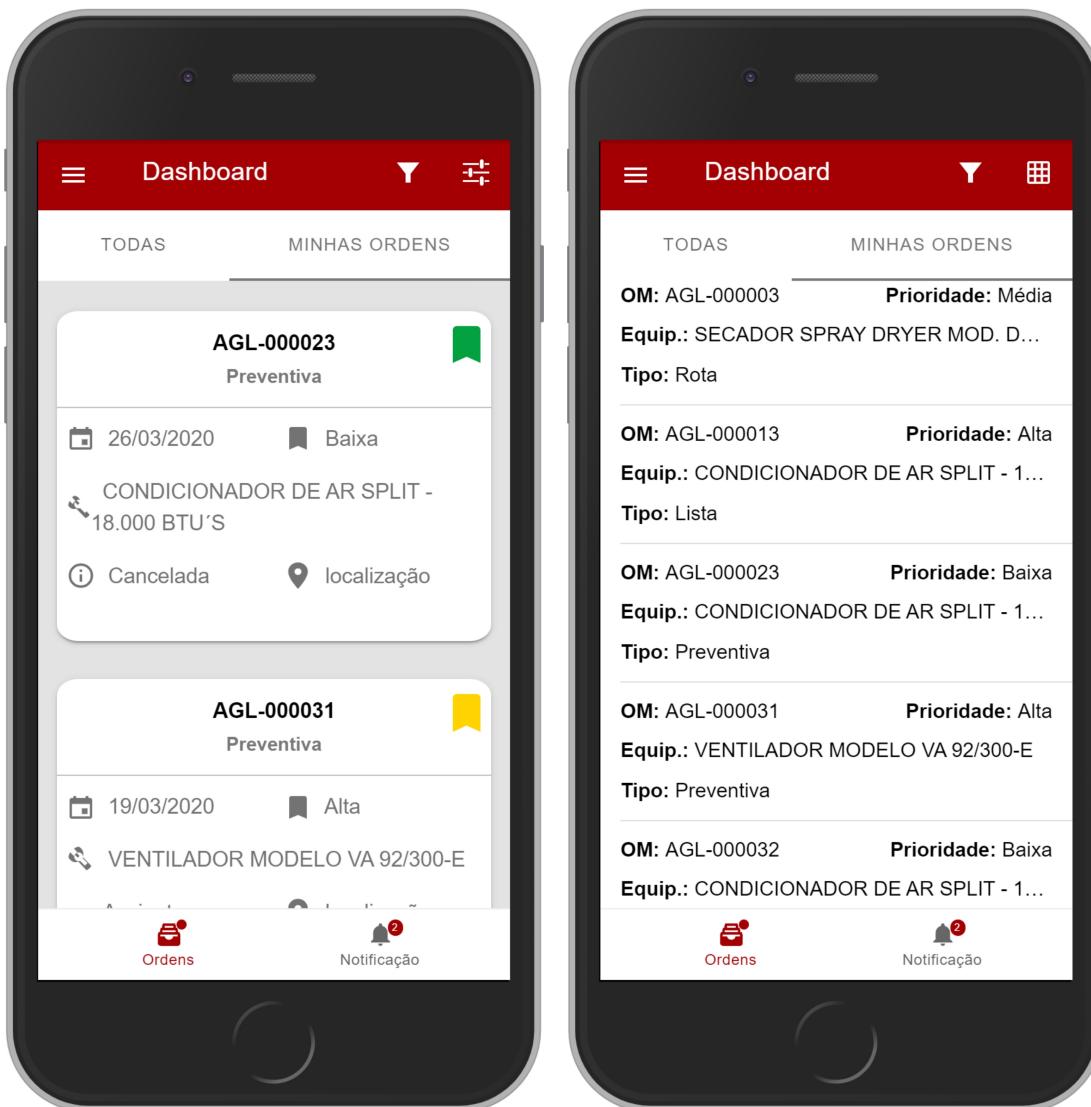
Fonte: os autores (2020)

8.2 Mobile

O aplicativo mobile será de suma importância para a execução das ordens de manutenção por parte dos técnicos, nele poderá ser executado todas as ações no que diz respeito às ordens de manutenção. A facilidade de uso possibilita a execução de ordens de manutenção com maior agilidade e eficiência.

8.2.1 Monitor de Ordens de Manutenção

Figura 8.9: Monitor de Ordens de Manutenção

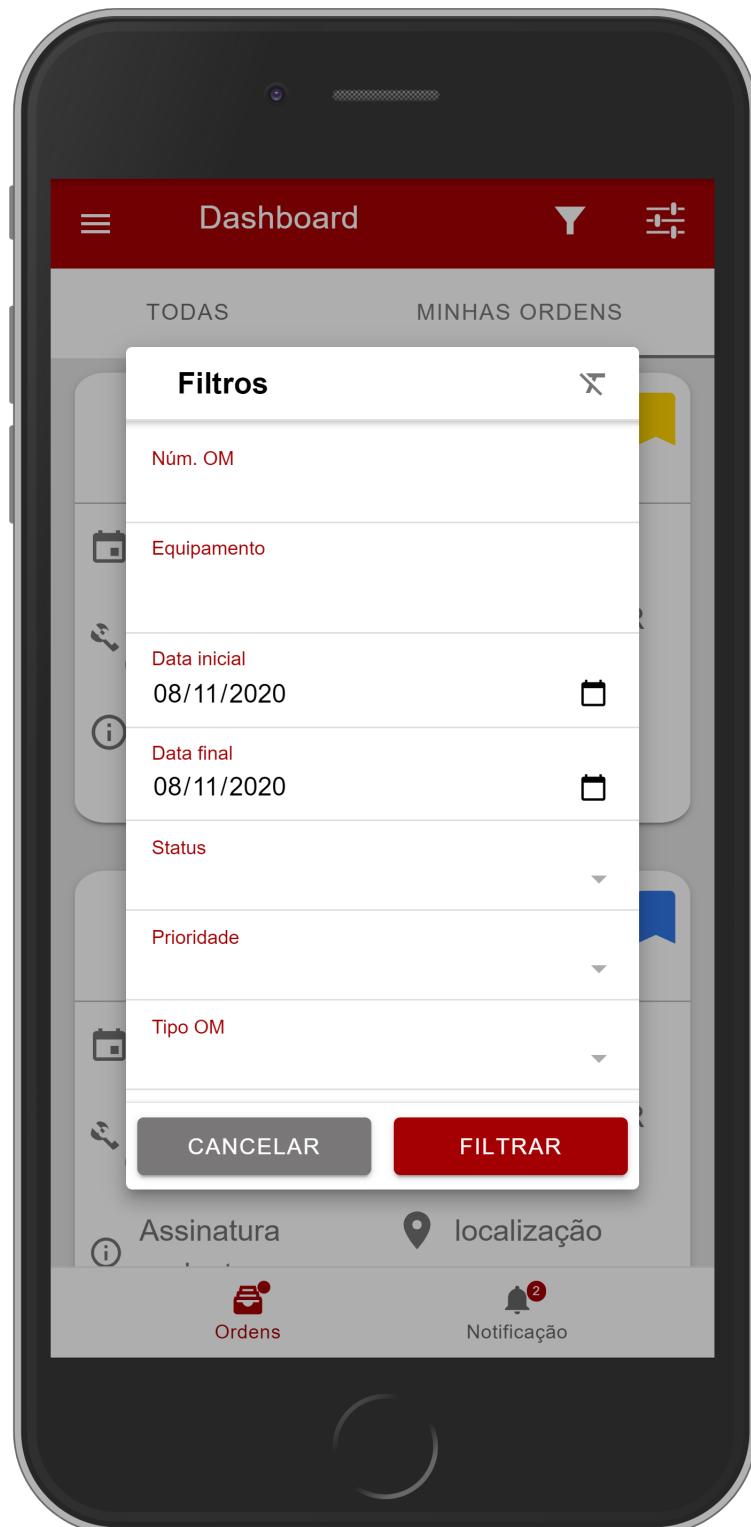


Fonte: os autores (2020)

A figura 8.9 mostra o monitor que os técnicos irão utilizar, ele tem duas visualizações: Cartões e Lista, e separa as ordens em duas abas: Todas e Minhas Ordens para que o técnico possa rapidamente verificar em quais ordens ele está envolvido.

O monitor também possui filtros para facilitar a busca de Ordens de manutenção, conforme mostra a figura 8.10

Figura 8.10: Filtros do Monitor



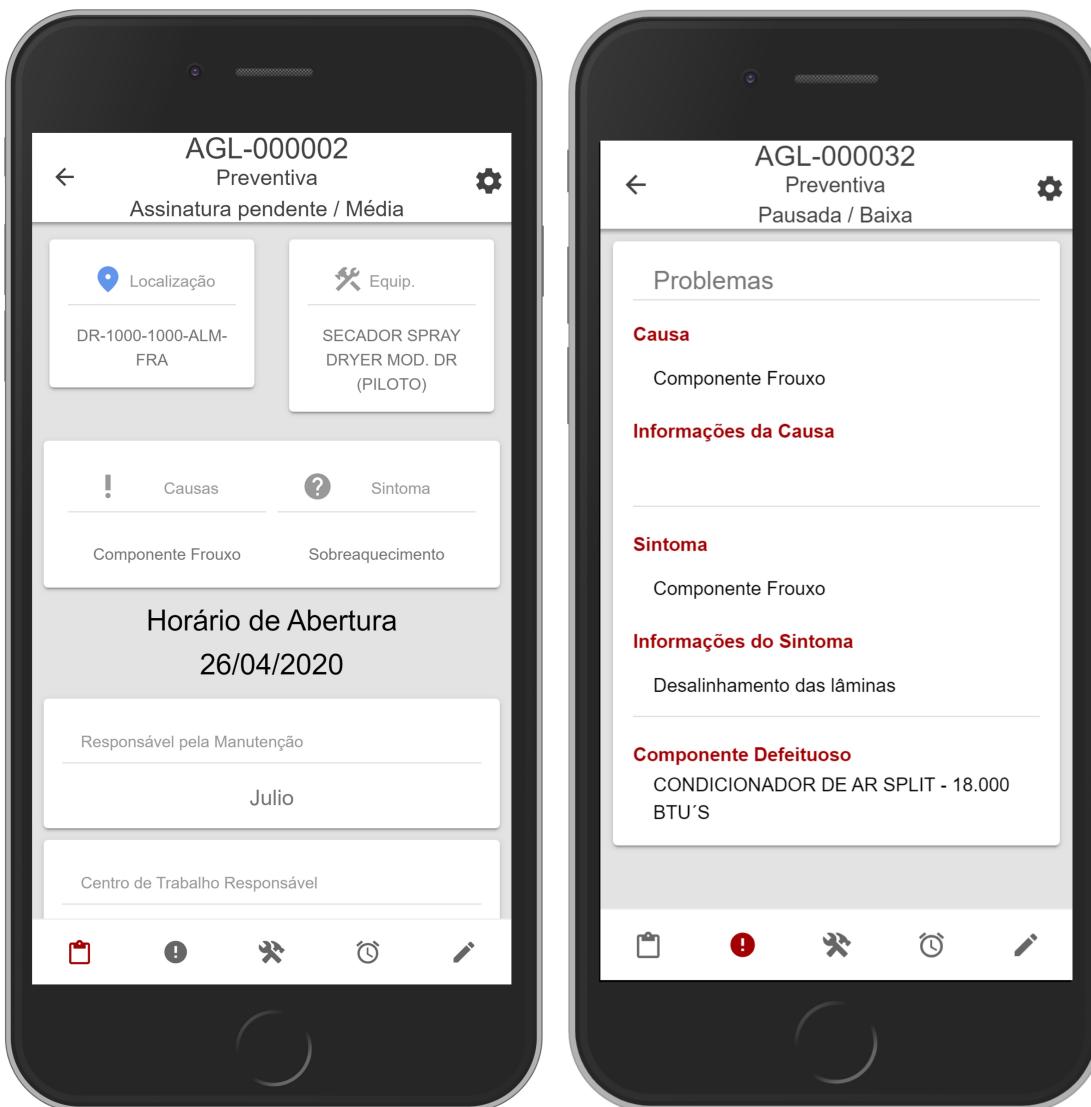
Fonte: os autores (2020)

Por padrão, o monitor não apresenta ordens que estão nas situações finais: finalizada ou

cancelada, só serão exibidas se forem informados explicitamente nos filtros.

8.2.2 Ordem de Manutenção Corretiva/Preventiva

Figura 8.11: Ordem de Manutenção Corretiva/Preventiva

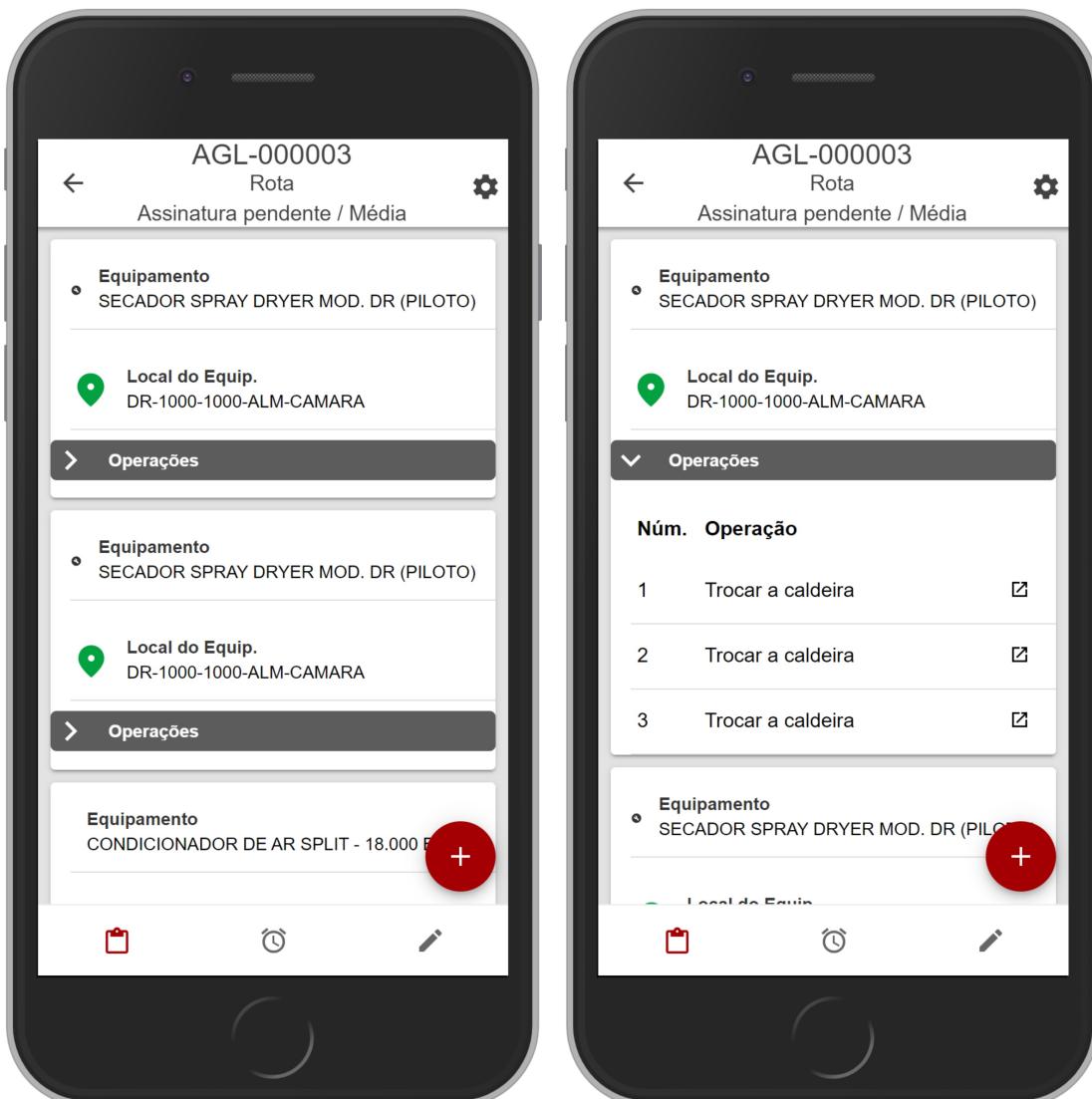


Fonte: os autores (2020)

A figura 8.11 mostra o detalhamento das duas primeiras abas da ordem do *layout* corretiva/preventiva, a primeira aba mostra dados pertinentes ao técnico como o equipamento e a localização do mesmo, a causa e o sintoma do problema. Na segunda aba é um detalhamento mais aprofundado do problema identificado, mostrando a causa, sintoma e o componente defeituoso. A terceira aba é de operações, a quarta de apontamentos e a quinta de assinaturas, todas serão abordadas a seguir.

8.2.3 Ordem de Manutenção Lista e Rota

Figura 8.12: Ordem de Manutenção Lista e Rota

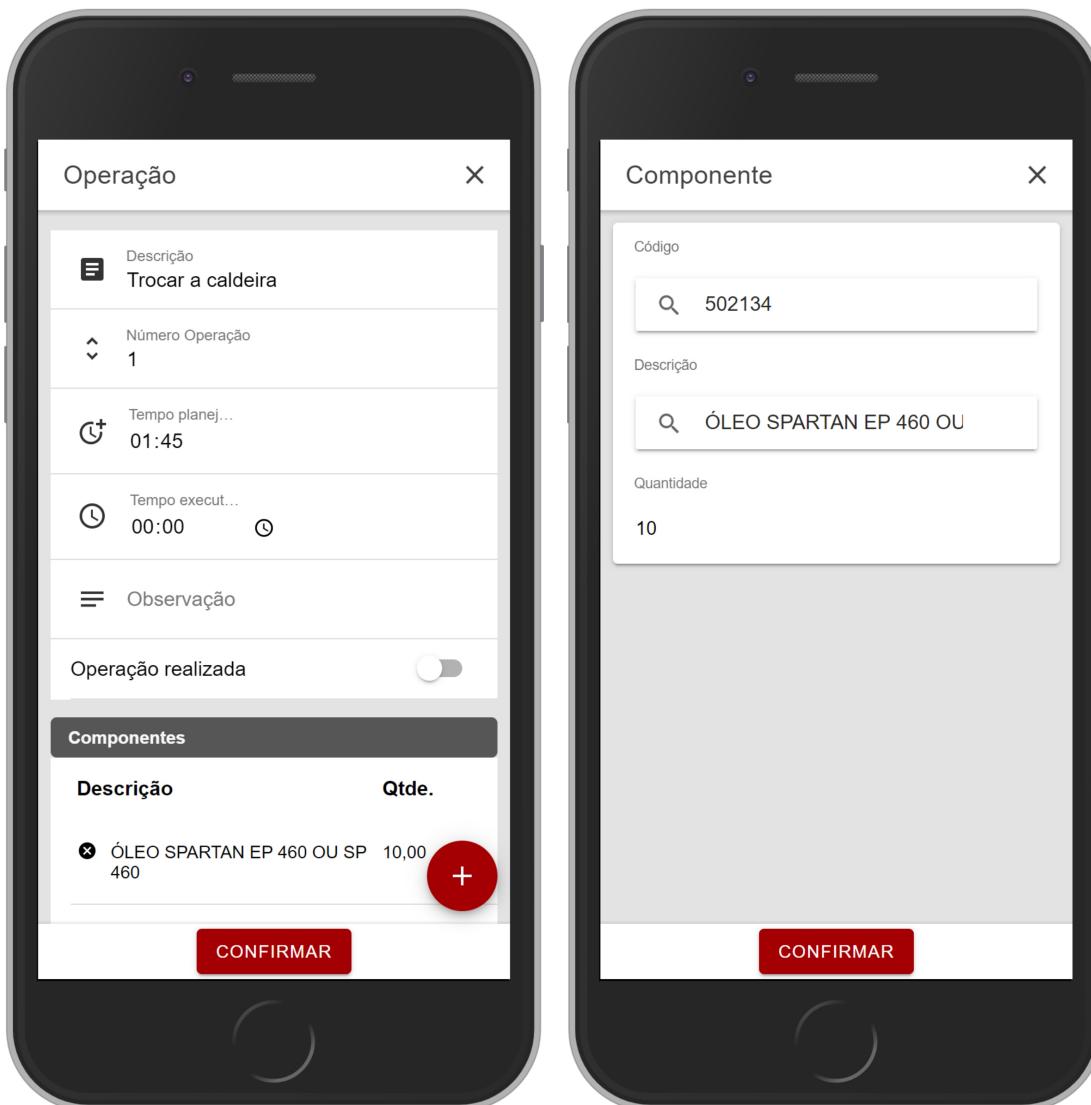


Fonte: os autores (2020)

A figura 8.12 mostra o detalhamento da primeira aba da ordem dos *layouts* lista e rota que é a aba de operações. Nela, tem uma listagem de todos os equipamentos da ordem e suas operações. O técnico pode criar novas operações clicando no botão de “+” ou detalhar/editar uma operação clicando na linha dela. A segunda aba é de apontamentos e a terceira de assinatura, que serão abordadas posteriormente.

8.2.4 Operações da Ordem de Manutenção

Figura 8.13: Ordem de Manutenção Lista e Rota

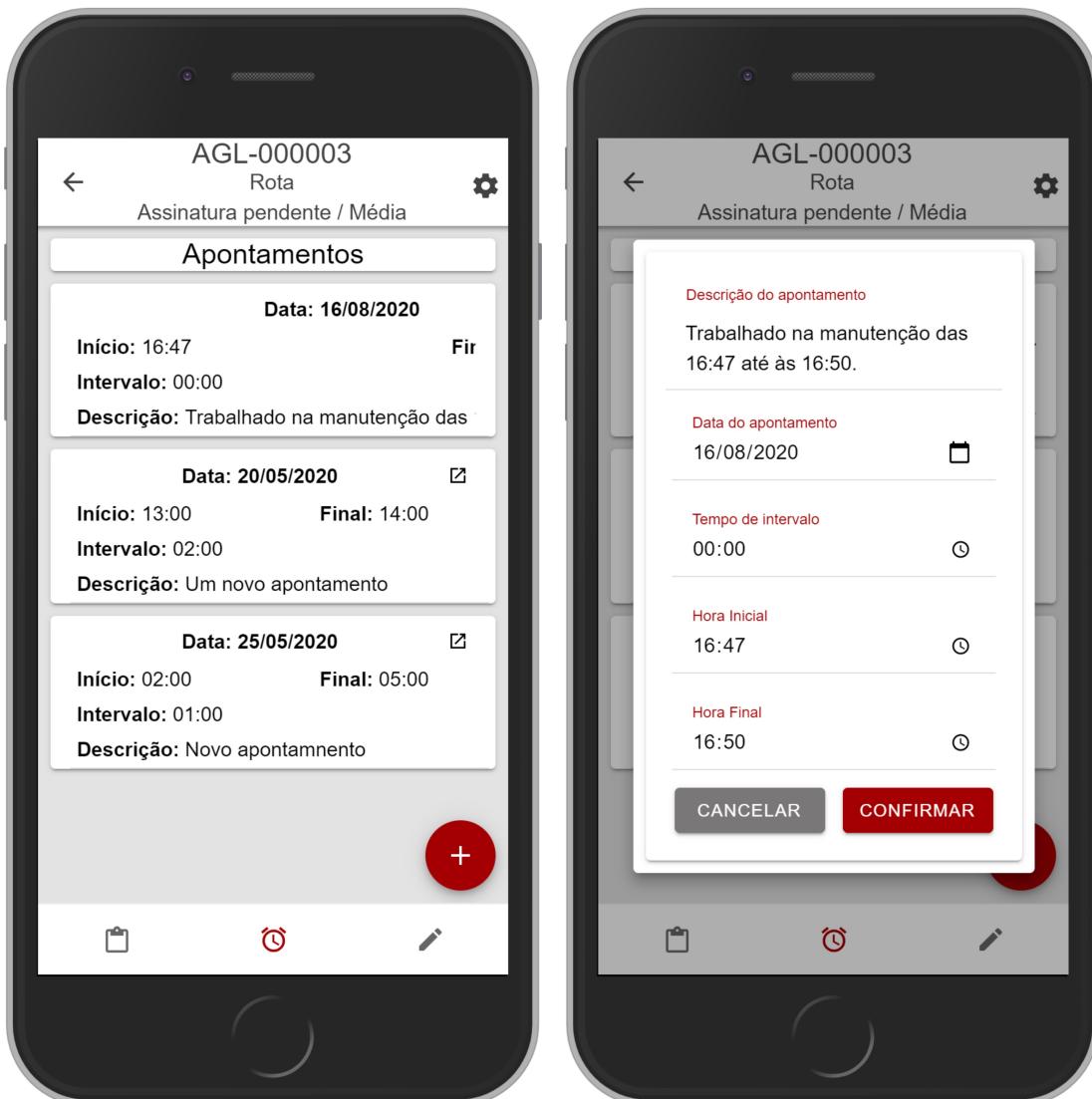


Fonte: os autores (2020)

Ao clicar em uma operação será aberto a tela mostrada na figura 8.13, nela é possível colocar o tempo investido em sua execução, se ela já foi executada e adicionar alguma observação, caso necessário. Para executar determinadas operações, o técnico pode necessitar de alguns insumos, para tanto ele pode adicionar componentes utilizados na operação clicando no botão de “+”, editar clicando na linha do componente e excluir clicando no “x” na primeira coluna da linha. Para adicionar um novo componente, o técnico pode pesquisar por código ou descrição e deve informar a quantidade utilizada.

8.2.5 Apontamento

Figura 8.14: Ordem de Manutenção: Apontamento

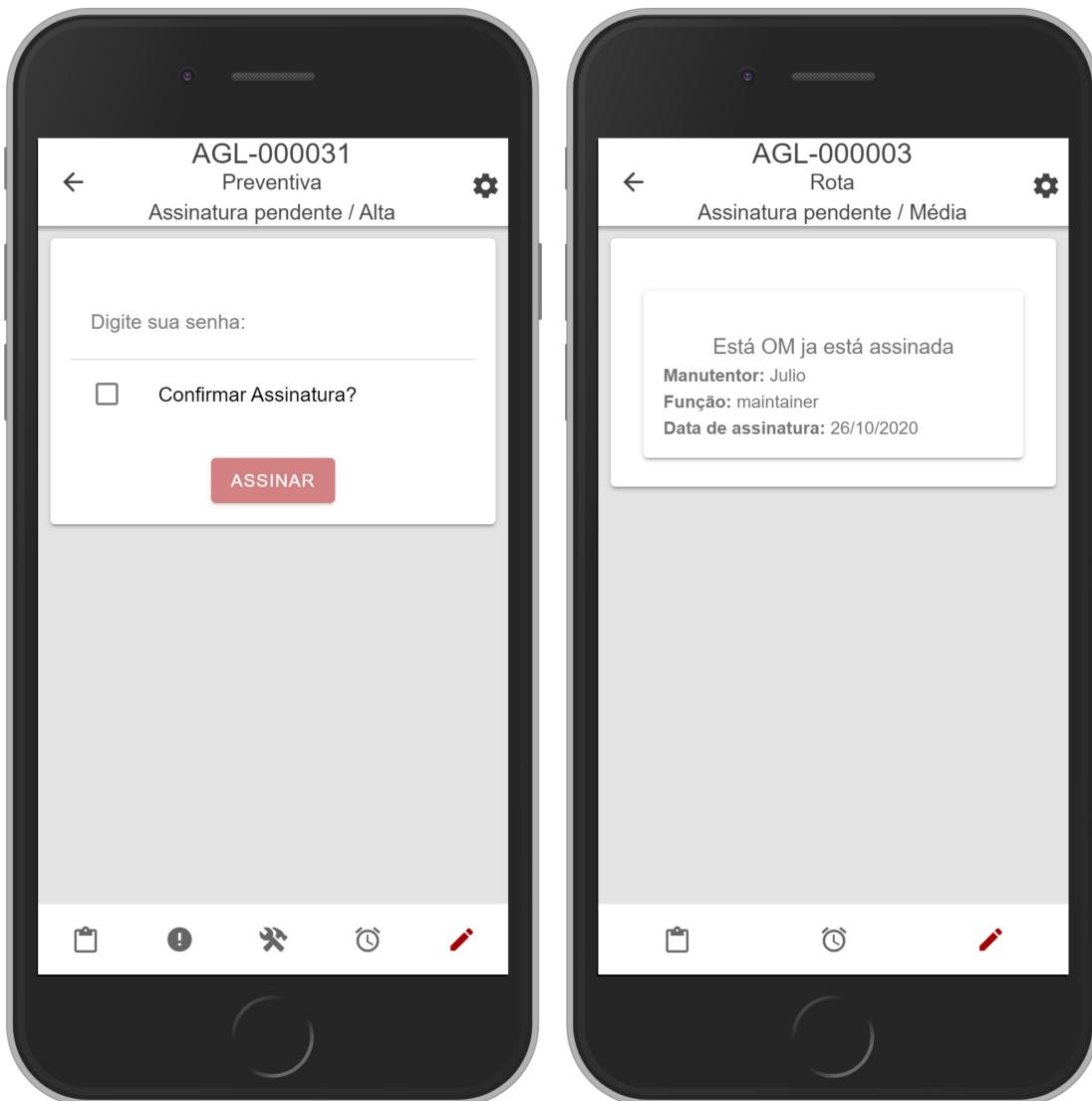


Fonte: os autores (2020)

A figura 8.14 mostra a listagem de apontamentos da OM, nele pode-se editar, excluir e criar um novo apontamento. Ao cadastrar ou editar um registro, o técnico deve informar uma descrição e a data do apontamento, a hora inicial e final. Caso tenha tido algum intervalo, por exemplo almoço, pode-se computar o intervalo no campo tempo de intervalo. Um técnico não pode visualizar apontamentos de outros técnicos na mesma ordem, nem realizar os apontamentos por outro técnico.

8.2.6 Assinatura

Figura 8.15: Ordem de Manutenção: Apontamento



Fonte: os autores (2020)

A figura 8.15 mostra o processo de assinatura da ordem, caso ainda não tenha sido assinada será apresentado para assinatura, no qual o técnico deve informar a senha dele e marcar a caixa para confirmar a assinatura e clicar no botão “Assinar”. Caso já esteja assinada, será informada as assinaturas da ordem contendo os dados do usuário que assinou, sua função e a data da assinatura.

9 TESTES DE USABILIDADE

Uma das definições mais encontradas na literatura é a de que a usabilidade é parte de um projeto mais amplo, que aponta para o desenvolvimento de métodos e técnicas que podem incorporar considerações de ergonomia dentro do processo de design e avaliação da interface homem/computador (BASTIEN; SCAPIN, 1993). A ISO/IEC FCD 9126-1 define usabilidade como a capacidade do software ser compreendido, aprendido, usado e apreciado pelo usuário, dado as condições específicas (GONÇALVES, 2009).

As experiências negativas no uso de interfaces frustam os usuários, fazendo com que se sintam diminuído, culpando-se por não conseguir realizar tarefas que, hipoteticamente, outros usuários conseguem (GONÇALVES, 2009). Quando essa dificuldade é frequente a frustração pode levar o usuário ao estresse e ansiedade, devido à sequência de experiências negativas (CYBIS, 2003).

Para que a revolução digital ocorra, um computador deve representar-se ao usuário, numa linguagem que este comprehenda, por isso, os sistemas computacionais precisam utilizar elementos na interface que sejam intuitivos e de fácil compreensão para que o usuário consiga atingir seus objetivos (JOHNSON, 2001).

Portanto, foi levado em consideração técnicas de UI e UX durante todo o desenvolvimento das aplicações WEB e Mobile com o objetivo de facilitar a usabilidade e deixar a experiência mais fluida possível para o usuário, evitando com que haja frustrações para com o sistema desenvolvido.

9.0.1 Elaboração dos Testes Aplicados

A usabilidade é uma qualidade de uso, ou seja, ela é definida ou medida para um determinado contexto no qual um sistema é operado. Assim, um sistema pode proporcionar boa usabilidade para um usuário experiente, mas péssima para um iniciante, ou vice-versa, ou ainda, pode ser fácil operar se o sistema for usado esporadicamente, mas difícil se for utilizado frequentemente (CYBIS, 2003).

Para isso, estudos como os de (BASTIEN; SCAPIN, 1993), apresentam regras e recomendações para que os sistemas computacionais sejam elaborados de modo a facilitar sua aprendizagem e uso, proporcionando usabilidade.

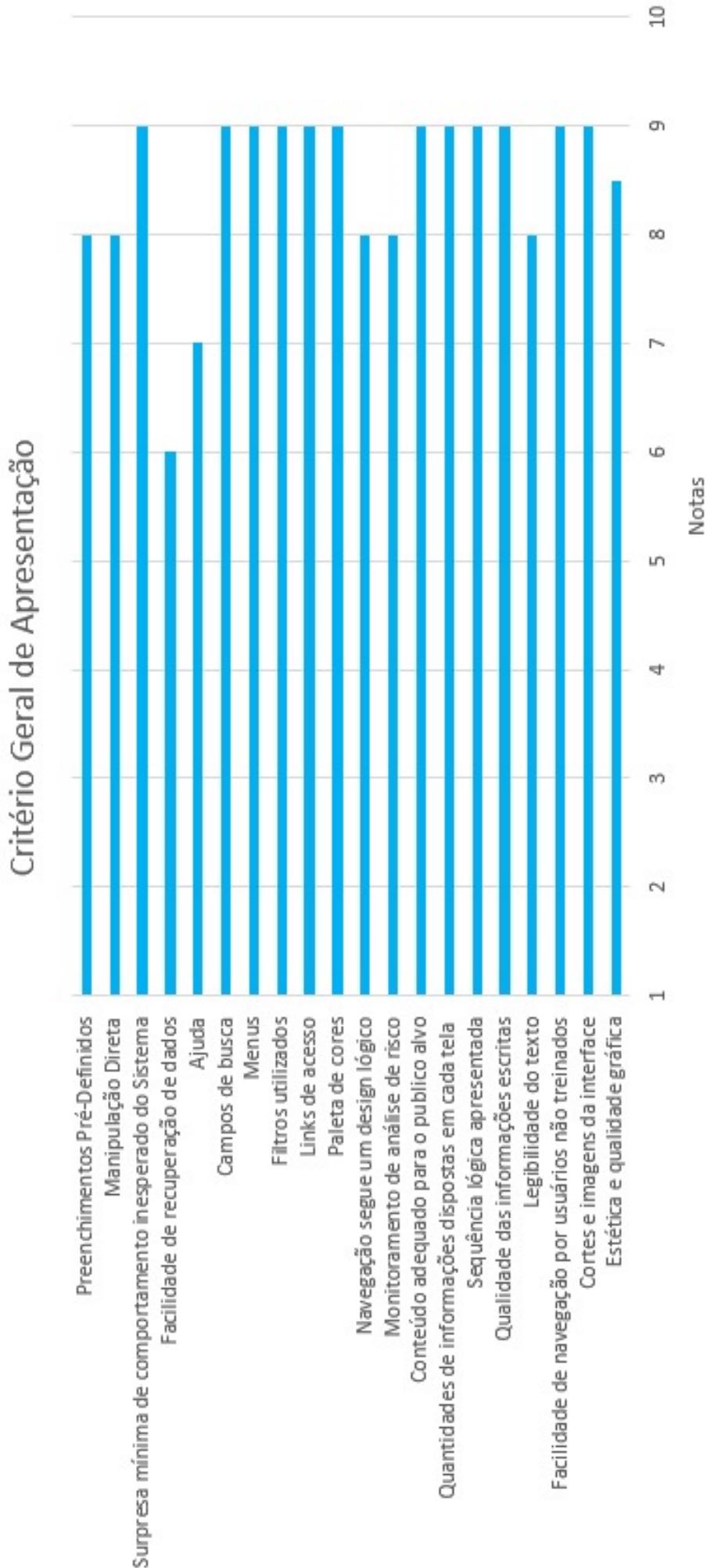
Para a avaliação do projeto, foi elaborado uma metodologia de verificação de alguns requisitos visando a análise e a usabilidade padrão mínimo para a implantação e utilização do web/mobile. Para isso foi criado algumas situações de usabilidade segundo alguns trabalhos científicos como a de (SILVA, 2016), onde verifica-se alguns critérios importantes que foram formulados para a aplicação das avaliações do desenvolvimento dos projetos Integradores. As avaliações aconteceram de forma remota com uma amostragem de aproximadamente 19 usuários, incluindo os colaboradores da empresa Duas Rodas.

Para a elaboração dos resultados, foram realizadas as médias de todas as avaliações e unificadas graficamente, buscando descrever-las o mais transparente possível. Para isso, foi criado 3 módulos de avaliação que são eles: Critério Geral de Apresentação, Critérios Gerais e Erros Comuns, onde respectivamente encontra-se como características de *layout*, características de desenvolvimento e erros que podem ocorrer no projeto.

9.0.2 Discussão dos Resultados

Verificou-se no Gráfico 9.1 de Critérios Gerias de apresentação do projeto Agil.It que o percentual de avaliação teve um aproveitamento de 84,5% e média de todos os resultados avaliados de 8,5, considerando os critérios de apresentação dos *layouts*, onde as notas foram de 1-10 avaliando cada quesito, como observa-se no gráfico.

Figura 9.1: Critério Geral de Apresentação



Para a análise dos Critérios Gerais, observa-se no gráfico 9.2 que o foco principal foi no desenvolvimento e desempenho do software considerando características fundamentais para o sucesso da usabilidade e da aprovação do Usuário. Neste quesito, a avaliação ficou em um aproveitamento de 88,2% com a média de 8,8 dos quesitos avaliados.

Figura 9.2: Critérios Gerais



Quanto a avaliação dos Erros Comuns, foi avaliado com aplica-se ou não este quesito, e observa-se na tabela 9.1 que, o software de gerenciamento do Agil.It, chegou a um percentual médio de aprovação de 92%, sendo assim, considera-se ótimo o percentual obtido na avaliação, na etapa do projeto.

Tabela 9.1: Percentual Médio de Aprovação em Erros Comuns de Usabilidade

	Erros comuns de Usabilidade	Percentual Médio Aprovação
1	Ícones e Menus ambíguos	92%
2	Linguagens que permitem apenas movimentos direcionados de forma única	
3	Limite de entrada e manipulação	
4	Limite de seleção e destaque	
5	Sequência não clara de passos	
6	Mais passos para gerenciar a interface do que para realizar tarefas	
7	Links complexos entre/com aplicações	
8	Confirmações e Feedbacks inadequados	
9	Pouca inteligência e antecipação por conta do sistema	
10	Mensagem de erro, tutoriais ajuda e documentação inadequadas	
11	Poluição Visual	
12	Má Organização da Informação	
13	Componentes Incompreensíveis	
14	Distrações irritantes	
15	Navegação Ineficiente	
16	Sobrecarga de Informações	

Fonte: os autores (2020)

10 CONSIDERAÇÕES FINAIS

Ao longo do desenvolvimento deste projeto, foram levantadas várias necessidades da empresa alimentícia Duas Rodas no que diz respeito ao gerenciamento de todo o seu processo administrativo e operacional das ordens de manutenção. Com base nisso, desenvolveu-se as aplicações abrangendo ao máximo os requisitos especificados pela equipe da empresa.

Sendo assim, o projeto é de suma importância para ambas as partes envolvidas, pois a empresa se beneficiará das funcionalidades do sistema como um todo, usufruindo do seu processo automatizado e totalmente digital enquanto os desenvolvedores alcançaram um crescimento exponencial tanto na parte pessoal, acadêmica e profissional. Para transformar o processo de manutenção dos equipamentos mais viável para a empresa, uma das principais funcionalidades desenvolvidas no sistema foi a parte gerencial da ordem de manutenção, que permite realizar um acompanhamento detalhado e constante das tarefas a serem executadas na ordem de manutenção até que a mesma seja verificada pelo responsável da manutenção.

O resultado de modificar o processo de execução e verificação da manutenção de forma digital gerou resultados positivos, não havendo consumo excessivo com papéis e maior agilidade na busca de manutenções específicas. Em consequência disso, impactos no meio ambiente foram reduzidos, já que não serão utilizados papéis para administrar e acompanhar as manutenções. Almejamos como desafio no final do projeto, desenvolver e integrar o sistema Agil.It com o ERP Alemão SAP.

REFERÊNCIAS BIBLIOGRÁFICAS

- BASTIEN, J. C.; SCAPIN, D. L. Ergonomic criteria for the evaluation of human-computer interfaces. 1993.
- BOFF, L. **Sustentabilidade: o que é - o que não é**. Editora Vozes, 2017. ISBN 9788532656100. Disponível em: <<https://books.google.com.br/books?id=px46DwAAQBAJ>>.
- CARNIELLO, A. **Teste Basead na estrutura de casos de uso**. [S.I.: s.n.], 2003.
- COUTO, J. M. C. **Técnicas de visualização de dados em gerenciamento de projetos de desenvolvimento de software: proposta de extensão do PMBOK**. 108 p. Monografia (Pós Graduação) — Pontifícia Universidade Católica do Rio Grande do Sul, Rio Grande do Sul, 2018.
- COYETTE, A. et al. Sketchxml: towards a multi-agent design tool for sketching user interfaces based on usixml. In: **Proceedings of the 3rd annual conference on Task models and diagrams**. [S.I.: s.n.], 2004. p. 75–82.
- CRERIE, R.; BAIÃO, F. A.; SANTORO, F. M. Identificacao de regras de negocio utilizando mineração de processos. In: ACM. **Companion Proceedings of the XIV Brazilian Symposium on Multimedia and the Web**. [S.I.], 2008. p. 241–246.
- CRUZ, F. **Scrum e PMBOK unidos no Gerenciamento de Projetos**. [S.I.]: Brasport, 2013.
- CUNHA, M. B. Entendendo o uso do git em equipes de desenvolvimento de software. 2018.
- CYBIS, W. d. A. Engenharia de usabilidade: uma abordagem ergonômica. **Florianópolis: Labiutil**, 2003.
- DANTAS, C. M.; CORDULA, F. R.; ARAÚJO, W. J. Análise da representação da informação em modelos entidade relacionamento com base em metadados. **Archeion Online, João Pessoa**, v. 4, n. 1, p. 40–63, 2016.
- DEXTRA. **Prototipação e sua importância no desenvolvimento de software**. 2019. Disponível em: <<https://dextra.com.br/pt/prototipacao-e-sua-importancia-no-desenvolvimento-de-software/>>.
- FAYYAZ, A. R.; MUNIR, M. **Performance Evaluation of PHP Frameworks (CakePHP and CodeIgniter) in relation to the Object-Relational Mapping, with respect to Load Testing**. 2014.
- FILHO, J. R. Definição e implantação de kpis para auxiliar a gestão de uma empresa de softwares. Universidade Federal de Uberlândia, 2017.
- FILHO, W. de P. P. **Engenharia de software**. [S.I.]: LTC, 2003. v. 2.
- FOWLER, M. **UML Essencial**. [S.I.: s.n.], 2005. 104–105 p.
- GASPARINI, B. C. et al. Driv-uml: visualização de não-conformidades arquiteturais em uml no contexto da modernização dirigida a arquitetura. Universidade Federal de São Carlos, 2018.
- GASQUES, J. **O desafio do dízimo**. [S.I.: s.n.], 2002.
- GONÇALVES, M. K. Usabilidade de software: estudo de recomendações básicas para verificação do nível de conhecimento dos alunos dos cursos de design gráfico e sistemas de informação da unesp/bauru. Universidade Estadual Paulista (UNESP), 2009.

- HAUFE, M. I. **Estimativa da produtividade no desenvolvimento de software.** 108 p. Monografia (Mestrado) — Universidade Federal do Rio Grande do Sul, Rio Grande do Sul, 2001.
- HURT, R. L. **Sistemas de informações contábeis.** [S.I.]: AMGH Editora Ltda, 2014.
- INSTITUTE, P. **A Guide to the Project Management Body of Knowledge (PMBOK(R) Guide-Sixth Edition / Agile Practice Guide Bundle (BRAZILIAN PORTUGUESE).** Project Management Institute, 2018. ISBN 9781628255270. Disponível em: <<https://books.google.com.br/books?id=fOlfDwAAQBAJ>>.
- JOHNSON, S. **Cultura da Interface: Como o computador transforma nossa maneira de criar e comunicar.** Zahar, 2001. (Interface). ISBN 9788537810613. Disponível em: <<https://books.google.com.br/books?id=-9sAwQEACAAJ>>.
- KERZNER, H. **Project Management: A Systems Approach to Planning, Scheduling, and Controlling.** Wiley, 2017. ISBN 9781119165378. Disponível em: <<https://books.google.com.br/books?id=VNExDgAAQBAJ>>.
- KILOV, H.; SIMMONDS, I. Business rules: from business specification to design. In: BOSCH, J.; MITCHELL, S. (Ed.). **Object-Oriented Technologies.** Berlin, Heidelberg: Springer Berlin Heidelberg, 1998. p. 188–194. ISBN 978-3-540-69687-2.
- KONNORATE, G. C. et al. A importancia do controle de versões no desenvolvimento de software. **Seminário De Tecnologia Gestão E Educação**, v. 1, n. 2, p. 1–4, 2019.
- MARTINS, J. C. C. **Técnicas para gerenciamento de projetos de software.** [S.I.]: Brasport Livros e Multimidia Ltda, 2007.
- MATTIOLI, A. **Sistemas computacionais.** [S.I.]: Editora Pearson, 2020.
- MICHALSKY, S. Componentes de software no desenvolvimento de aplicações colaborativas para web: Evolução da plataforma groupware workbench. **IME-USP. São Paulo**, 2012.
- MOURA, T. C. e. E. F. R. Condicionantes de sucesso em projetos de software e sua influência nos resultados. **Revista Gestão e Tecnologia**, v. 18, n. 1, p. 61–87, 2018.
- MOURA, T. M. **Análise da Implementação de Práticas de TI Verde em um Instituto Federal de Educação, Ciência e Tecnologia.** 85 p. Monografia (Mestrado) — Universidade Federal de Pernambuco, Pernambuco, 2017.
- NAZÁRIO, M. F. C. et al. Detecting and reporting object-relational mapping problems: An industrial report. In: IEEE. **2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM).** [S.I.], 2019. p. 1–6.
- NOGUERA, V. E. R. et al. Extensão de uma álgebra er para execução de consultas em bancos de dados nosql orientados a documentos. Universidade Federal de São Carlos, 2018.
- PÁDUA, S. I. D. de. Investigação do processo de desenvolvimento de software a partir da modelagem organizacional, enfatizando regras do negócio. 2001.
- PRADO, D. **Gerência de Projetos em Tecnologia da Informação.** [S.I.]: Editora INDG, 1999. ISBN 9788586948176.
- PRIMO, M. O. **Diagrama de Classe de Projeto.** 2014.
- PRODUTTIVO. 2019. Disponível em: <<https://produttivo.com.br>>.
- PRODWIN. 2019. Disponível em: <<https://prodwin.com.br>>.

- QUEIRÓS, R. C. C.; MÉXAS, M. P.; DRUMOND, G. M. Tecnologia da informação verde nas organizações: uma visão estratégica. **Sistemas e Gestão**, v. 15, n. 2, p. 103–112, ago. 2020. Disponível em: <<https://revistasg.emnuvens.com.br/sg/article/view/1629>>.
- REZENDE, D. A. **Engenharia de Software e Sistemas de Informação**. [S.I.: s.n.], 2005. v. 3.
- ROCHA, F. G.; SABINO, R. F.; ACIPRESTE, R. H. L. A metodologia scrum como mobilizadora da prática pedagógica: um olhar sobre a engenharia de software. **FEES 2015**, p. 13, 2015.
- ROCHA, H.; TERRA, R. TerraER: Uma ferramenta voltada ao ensino do modelo de entidade-relacionamento. In: **VI Escola Regional de Banco de Dados (ERBD)**. [S.I.: s.n.], 2010. p. 1–4.
- ROSEMBERG, C. et al. Prototipação de software e design participativo: uma experiência do atlântico. **IHC**, v. 8, p. 312–315, 2008.
- ROSSINI, A. M. Administração de sistemas de informação e a gestão do conhecimento. 2006.
- SABBAGH, R. **Scrum: Gestão ágil para projetos de sucesso**. [S.I.]: Editora Casa do Código, 2014.
- SANTOS, M. A. dos; BALANCIERI, R. Planos de ação para melhoria do processo de produção de software com desenvolvimento distribuído. **Trabalhos de Conclusão de Curso do DEP**, 2017.
- SCHMITZ, E.; ALENCAR, A. **Análise de Risco em Gerência de Projetos - 3ª Edição**. BRAS-PORT, 2012. ISBN 9788574525426. Disponível em: <<https://books.google.com.br/books?id=ubZ7YHtWZ08C>>.
- SELNER, C. et al. Análise de requisitos para sistemas de informações, utilizando as ferramentas da qualidade e processos de software. Florianópolis, SC, 1999.
- SILVA, J. P. S. d. Sasml: a uml based domain specific modeling language for self adaptive systems conceptual modeling. 2018.
- SILVA, V. M.; BARBOSA, R. d. M.; ADAMATTI, D. F. Princípios de usabilidade e a importância do usuário no projeto de interfaces. **sistema**, v. 15, p. 18, 2016.
- SOFTBYTE. 2019. Disponível em: <<http://softbyte.com.br>>.
- SOMASUNDARAM, G.; SHRIVASTAVA, A.; SERVICES, E. **Armazenamento e Gerenciamento de Informações: Como armazenar, gerenciar e proteger informações digitais**. Bo- okman, 2009. ISBN 9788577807642. Disponível em: <<https://books.google.com.br/books?id=d8uCfC46hwsC>>.
- SOMMERVILLE, I. **Software Engineering**. Pearson Education, 2011. ISBN 9780133001495. Disponível em: <<https://books.google.com.br/books?id=fSYrAAAAQBAJ>>.
- SOUZA, A. R. R.; MONTEIRO, L. A.; ALMEIDA, W. H. C. Gerenciamento de projetos ágil na prática: Processos e ferramentas para apoio a gestão. 2017.
- SOUZA, L. F. d. et al. Desenvolvimento de sistema de informação para monitoramento da esclerose múltipla. Universidade Federal da Paraíba, 2017.
- STAIR, R. M. Princípios de sistemas de informação: uma abordagem gerencial. 2008.
- SUTHERLAND, J. **Scrum: a arte de fazer o dobro do trabalho na metade do tempo**. [S.I.]: Leya, 2016.

TORONTO, I. I. de Análise de Negócios de. **O guia para o corpo de conhecimento de análise de negócios.** [S.l.: s.n.], 2005.

VARGAS, R. V. **Manual Prático do Plano de Projeto, Utilizando o PMBOK Guide.** [S.l.: s.n.], 2018.

VAZQUEZ, C.; SIMÕES, G. **Engenharia de Requisitos: software orientado ao negócio.** BRASPORT, 2016. ISBN 9788574527901. Disponível em: <<https://books.google.com.br/books?id=gA7kDAAAQBAJ>>.