UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

CENTRO TECNOLÓGICO

COLEGIADO DO CURSO DE ENGENHARIA DE COMPUTAÇÃO

Wallace Kruger Junior

GanttBox - Sistema de Gestão de Projetos

Vitória, ES

2022

Wallace Kruger Junior

GanttBox - Sistema de Gestão de Projetos

Monografia apresentada ao Curso de Engenharia de Computação do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Bacharel em Engenharia de Computação.

Universidade Federal do Espírito Santo – UFES

Centro Tecnológico

Colegiado do Curso de Engenharia de Computação

Orientador: Prof. Dr. Vítor E. Silva Souza

Vitória, ES

2022

Wallace Kruger Junior
GanttBox - Sistema de Gestão de Projetos/ Wallace Kruger Junior. – Vitória, ES, 2022-
70 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Vítor E. Silva Souza

Monografia (PG) – Universidade Federal do Espírito Santo – UFES
Centro Tecnológico
Colegiado do Curso de Engenharia de Computação, 2022.

1. Palavra-chave1. 2. Palavra-chave2. I. Souza, Vítor Estêvão Silva. II. Universidade Federal do Espírito Santo. IV. GanttBox - Sistema de Gestão de Projetos

CDU 02:141:005.7

Wallace Kruger Junior

GanttBox - Sistema de Gestão de Projetos

Monografia apresentada ao Curso de Engenharia de Computação do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Bacharel em Engenharia de Computação.

Trabalho aprovado. Vitória, ES, 23 de março de 2022:

Prof. Dr. Vítor E. Silva Souza
Orientador

Profª. Drª. Monalessa Perini Barcellos
Universidade Federal do Espírito Santo

Jussara Teixeira
Instituto de Tecnologia da Informação e Comunicação do Espírito Santo

Vitória, ES
2022

Agradecimentos

Agradeço primeiramente a Deus, que me permitiu chegar até este momento com saúde física e mental. Em seguida, agradeço a minha família, meus pais, Daniele e Wallace, que nunca mediram esforços para me proporcionar a melhor educação, serei eternamente grato. Aos meu avós, Maria e Mario, que juntamente aos meus pais, também contribuíram muito amorosamente e financeiramente até que eu pudesse ganhar meu próprio sustento. Ao meu tio Demetrius, que me influenciou a escolher a área de tecnologia. A minha irmã Tanya e a minha namorada Paolla, todos estes formam a base da minha estrutura emocional e dão suporte para enfrentar as dificuldades da vida, muito mais do que eu merecia como ser humano.

Agradeço também a todos os professores que fizeram parte e contribuíram de alguma forma na minha formação acadêmica. Um agradecimento especial aos excelentes professores da graduação: Rosane, Monalessa, Eduardo, Edileia, Raquel, dentre outros que não só exigem como professores de universidade federal, mas entregam um conteúdo compatível e se esforçam a cada semestre para entregar melhor. Ainda dentro dos excelentes professores, um agradecimento especial ao meu professor e orientador Vitor, que além de todos ensinamentos passados ao longo das disciplinas de graduação, principalmente aquelas voltadas a área de programação, foi um protagonista para que esse projeto pudesse ser conclído dentro do prazo, sempre a disposição e boa vontade na orientação e ainda com ágeis revisões, fundamentais dentro dos períodos mais curtos que aconteceram no EARTE (Ensino a distância em meio a pandemia da COVID-19).

Agradeço também a todos colegas e amigos que fiz ao longo destes anos de UFES, tornando o tempo de universidade muito melhor e mais leve, afinal, são as pessoas que conhecemos e os momentos que vivemos com elas, que guardaremos na nossa memória.

Agradeço também ao meu ex chefe e amigo Silas, que foi um professor de programação na minha vida fora da sala de aula e inspirou a ideia desta monografia.

Por fim, agradeço a mim mesmo, por me manter diligente durante todos esses anos de graduação, pois só quem finalizou um curso de tecnologia ou engenharia numa universidade federal sabe a dificuldades enfrentadas, especialmente quando se trabalha paralelamente ao longo da graduação.

Resumo

Qualificar o resultado final de um produto, está intimamente ligado a otimizar a qualidade do seu processo de produção. Adotar metodologias ágeis, visando tornar mais fluido um fluxo de trabalho, removendo possíveis impeditivos e priorizando com inteligência tarefas específicas, poderá melhorar e qualificar a entrega de qualquer equipe, independentemente da área de conhecimento. Neste texto, voltamos a nossa atenção para o cenário de uma empresa de tecnologia que necessita de um sistema para melhorar seus processos administrativos.

O objetivo almejado deste trabalho foi então, adotar um processo sistemático de engenharia, para projetar e implementar um software de gestão que solucionasse a problemática de uma emergente organização, que por falta de um processo gerencial metódico, "estacionou" sua expansão. A ferramenta de gestão que está sendo proposta nesta monografia, busca formas de mediar as ações dos usuários para entregar maior conhecimento e facilitar o trabalho dos gestores de uma empresa.

Para o desenvolvimento deste sistema, foram utilizados precedimentos da Engenharia de Software, como a escrita dos documentos de Especificação de Requisitos, Projeto de Sistema, além de adoção de arquiteturas de software que são pontes para uma boa implementação e seguimentos dos princípios e boas práticas da programação.

Os resultados obtidos neste projeto serão apresentados ao longo do texto, por meio de demonstrativos de códigos e capturas de telas, formando uma base estruturada para futuros trabalhos que se proponham a evoluir e progredirem a ferramenta para o cumprimento integral das necessidades iniciais percebidas.

Palavras-chaves: Gestão de Projetos, Aplicação *Web*, Engenharia de Software, FrameWeb, *Clean Architecture*, *Multitenancy* e Javascript.

Lista de ilustrações

Figura 1 – Arquitetura proposta pela *Clean Architecture* (MARTIN; GRENNING; BROWN, 2018). 22
Figura 2 – Arquitetura Proposta pelo FrameWeb (CAMPOS; SOUZA, 2017). . . . 25
Figura 3 – Diagrama de Pacotes e os Subsistemas Identificados. 31
Figura 4 – Diagrama de Casos de Uso do Subsistema SalesManagement. 32
Figura 5 – Diagrama de Casos de Uso do Subsistema ProjectManagement. 34
Figura 6 – Diagrama de Casos de Uso do Subsistema UserManagement. 35
Figura 7 – Diagrama de classes do Subsistema SalesManagement. 36
Figura 8 – Diagrama de classes do Subsistema ProjectManagement. 37
Figura 9 – Diagrama de classes do Subsistema UserManagement. 38
Figura 10 – Modelo de Entidades do Subsistema UserManagement. 40
Figura 11 – Modelo de Entidades do Subsistema SalesManagement. 41
Figura 12 – Modelo de Entidades do Subsistema ProjectManagement. 41
Figura 13 – Modelo de Aplicação da Entidade Organization do Subsistema UserManagement. 42
Figura 14 – Modelo de Aplicação da Entidade User do Subsistema UserManagement. 42
Figura 15 – Modelo de Persistência do Subsistema SalesManagement. 43
Figura 16 – Modelo de Navegação da Entidade Project do Subsistema ProjectManagement. 43
Figura 17 – Estrutura dos Diretórios no Back-End 44
Figura 18 – Parte do Código da Classe Validators 45
Figura 19 – Parte do Código de routes.ts 46
Figura 20 – Parte do Classe CreateLeadUseCase 46
Figura 21 – Parte da Classe Lead 47
Figura 22 – Parte da Classe FunnelSlotDTO 48
Figura 23 – Parte da Interface IClientRepository 48
Figura 24 – Parte da Classe ClientRepository 49
Figura 25 – Parte do Código de lead.routes.ts 49
Figura 26 – Parte da Classe LeadController 50
Figura 27 – Parte do Código de lead.routes.ts 50
Figura 28 – Estrutura de Diretórios do Front-End 51
Figura 29 – Variáveis de ambiente de desenvolvimento no Front-end 52
Figura 30 – Estutura de Diretórios dos Subsistemas no Front-end 52
Figura 31 – Arquivo HTML do Componente LeadFormModal 53
Figura 32 – Arquivo Typescript do Componente LeadFormModal 53
Figura 33 – Parte do Código da Classe LeadService 54

Figura 34 – Tela de Login 55
Figura 35 – Tela de Criar Conta 55
Figura 36 – E-mail Criação de Conta 56
Figura 37 – Modal Conta Criada 56
Figura 38 – Tela de Primeiro Acesso 57
Figura 39 – Tela Trocar Senha 57
Figura 40 – E-mail Senha Alterada 58
Figura 41 – Modal Esquecer Senha 58
Figura 42 – E-mail Esquecimento de Senha 58
Figura 43 – Tela Redefinir Senha 59
Figura 44 – Tela Funil de Vendas 59
Figura 45 – Modal Criar Lead 60
Figura 46 – Modal Editar Lead 60
Figura 47 – Modal Criar Proposta 61
Figura 48 – Modal Visualizar Lead no Slot Proposta 61
Figura 49 – Modal Criar Contrato 62
Figura 50 – Modal Editar Contrato 62
Figura 51 – PDF Exportado 63
Figura 52 – Modal Declarar Ganho/Perda 63

Lista de tabelas

Tabela 1 – Subsistemas identificados e suas interdependências. 31
Tabela 2 – Descrição dos atores envolvidos nos casos de uso. 32

Lista de abreviaturas e siglas

UML	Unified Modeling Language
DAO	Data Access Object
ARPA	Advanced Research Projects Agency
CERN	European Center for Nuclear Research
TCP	Transmission Control Protocol
HTTP	Hyper Text Transfer Protocol
URL	Uniform Resource Locator
DNS	Domain Name System
HTML	HyperText Markup Language
SPA	Single-page application
Saas	Software as a Service
B2B	Business to Business
WIS	Web Information System
IPMA	International Project Management Associationhttps
DTO	Data Transfer Objects
CRUD	Create, Read, Update, Delete
API	Application Programming Interface
SRP	Single-responsibility principle
JSON	JavaScript Object Notation
TI	Tecnologia da Informação
ABNT	Associação Brasileira de Normas Técnicas

Sumário

1 INTRODUÇÃO 12
1.1 Objetivo 13
1.2 Metodologia 13
1.3 Organização da Monografia 14

2 FUNDAMENTAÇÃO TEÓRICA 15
2.1 Engenharia de Software 15
2.1.1 Especificação de Requisitos 16
2.1.2 Projeto 17
2.1.3 Implementação e Teste 17
2.1.4 Entrega e Manutenção 18
2.2 Desenvolvimento Web 18
2.2.1 Arquitetura 19
2.2.2 Cliente 20
2.2.3 Angular 20
2.2.4 Node.js 21
2.3 Arquitetura de Software 21
2.3.1 Clean Architecture 22
2.3.2 Multitenancy 24
2.4 FrameWeb 25
2.5 Gestão de Projetos 26
2.6 Metodologias Ágeis 27

3 MODELAGEM DE SOFTWARE 29
3.1 Especificação de Requisitos 29
3.1.1 Conjuntura 29
3.1.2 Vendas 30
3.1.3 Projetos 30
3.1.4 Subsistemas 31
3.1.5 Casos de Uso 32
3.1.5.1 Subsistema SalesManagement 32
3.1.5.2 Subsistema ProjectManagement 33
3.1.5.3 Subsistema UserManagement 35
3.1.6 Modelo Estrutural 36
3.1.6.1 Subsistema SalesManagement 36
3.1.6.2 Subsistema ProjectManagement 37

3.1.6.3 Subsistema UserManagement 38
3.2 Projeto de Sistema 39
3.2.1 Tecnologias Utilizadas 39
3.2.2 Modelos FrameWeb 39
3.2.2.1 Camada de Negócio 40
3.2.2.2 Camada de Acesso a Dados 41
3.2.2.3 Camada de Apresentação 42

4 IMPLEMENTAÇÃO E APRESENTAÇÃO 44
4.1 Implementação do Back-end 44
4.2 Implementação do Front-end 50
4.3 Apresentação do Sistema 54
4.3.1 UserManagement 54
4.3.2 SalesManagement 57
4.3.3 ProjectManagement 60

5 CONCLUSÃO 64
5.1 Considerações Finais 64
5.2 Trabalhos Futuros 65

REFERÊNCIAS 68

APÊNDICES 70

1 Introdução

Desde a revolução industrial até o atual ritmo da inovação tecnológica, tornou-se evidente a facilitação do trabalho humano, essencialmente no afastamento da força muscular das tarefas de produção (DORF; BISHOP, 2001). Pela necessidade de comunicação, o surgimento da Internet no século passado trouxe ao mundo o poder do acesso e interatividade da informação. Esse novo meio, somado ao advento de novas ferramentas e aplicações inseridas nesse mesmo nicho, ocasionaram inúmeras mudanças na vida das pessoas, facilitando e moldando a maneira que vivemos. A forma como nos relacionamos com nossos amigos, criamos novos relacionamentos, fazemos compras e até trabalhamos era impensável há 30 anos.

Mais do que nunca, a informação é a chave para a sobrevivência em nossa sociedade informatizada. Compreender sua natureza e significado é o primeiro passo para podermos controlá-la e utilizá-la para o progresso social e individual (GOULART, 2004). A manipulação e organização dos dados que tornaram-se mais acessíveis a cada dia, poderão ser formadores da informação e consolidadores do conhecimento, guias fundamentais em deliberações seguras e elementais para produção de uma sociedade desenvolvida.

“Os dados agora podem ser capturados de forma cada vez mais rápida e precisa, usando sensores diversos capazes de capturarem quaisquer dos estímulos percebidos pelos cinco sentidos humanos e ainda conectar e construir informações em uma linguagem universal a partir deles como nunca antes” (MOONEEGAN, 2016).

Com o surgimento de algumas das principais teorias da administração, a partir da década de 50, as estruturas organizacionais das empresas começaram a recompor a antiga ideia da clara definição de papéis como um fator determinante da eficiência de trabalho. As aceleradas inovações tecnológicas ocorridas nos últimos anos demandam que a implementação das mudanças dos esqueletos organizacionais ocorram de forma ágil. A inópia dessas melhorias se encaixam perfeitamente com as características principais da Gestão de Projetos que é ser peculiar, exclusivo, não repetitivo e único (DINSMORE; NETO, 2006).

Nessa conjuntura, adentrando num ambiente de gerência de projetos para um pequeno empreendimento, tarefas manuais como: preenchimento de planilhas desmedidas, repetidamente, com infinidade de dados de seus colaboradores, lembretes para organizar suas tarefas e prioridades, já são incabíveis para o atual cenário tecnológico e sistema econômico que vivemos. Conduzir a rotina de um negócio dessa maneira, desperdiça tempo e esforço que poderiam ser delegados a outros afazeres. Com um mercado cada vez mais competitivo, a falta de organização nos processos gerenciais podem expor um negócio a

consequências severas. Em contrapartida, num ecossistema empresarial estruturado, além da economia de tempo de execução de tarefas manuais, a organização e o cruzamento de informações serão essenciais na construção do conhecimento do próprio negócio, este conhecimento, poderá ser a quididade nas tomadas de decisões. Emergindo restritamente em empresas de tecnologia de pequeno e médio porte, podemos redigir o objetivo.

1.1 Objetivo

Desenvolver um sistema Web de gestão de projetos, promovendo um ambiente que organiza o fluxo de trabalho dos projetos em andamento, disponibilizando recursos visuais para clientes e colaboradores, como a gerência e progresso de tarefas. Além de um controle de vendas, acompanhando e estruturando as negociações dos novos eventuais clientes da organização. A ferramenta será aplicada em uma empresa de tecnologia com cerca de 40 colaboradores, que ainda desempenha grande maioria de suas tarefas gerenciais manualmente. O sistema visa automatizar tarefas agregando valor aos processos de gestão de projetos e desafogando tempo do setor administrativo da corporação. Especula-se ainda, num futuro próximo, escalar outras empresas de mesmo porte, com ambientes semelhantes e também com deficiências e necessidades gerenciais parecidas.

1.2 Metodologia

- Revisão da literatura em diferentes assuntos abordados ao longo da graduação, tais como: Engenharia de Software, Desenvolvimento Web, Arquitetura de Software, Gestão de Projetos e Metodologias Ágeis (*Scrum*).
- Elaboração do documento de especificação de requisitos de sistema: apresentação do propósito do sistema, de seu minimundo, definição dos requisitos funcionais e não funcionais, modelos de casos de uso, modelo estrutural e glossário do projeto;
- Elaboração da documentação da arquitetura do sistema: produção do Documento de Projeto de Sistema, seguindo método FrameWeb (SOUZA, 2020);
- Estudo das tecnologias utilizadas para o desenvolvimento de APIs e interfaces para Web, tais como as linguagens de programação JavaScript, TypeScript, a plataforma Node.js, o *framework* Angular, o banco de dados relacional PostgreSQL, a ferramenta de *container* Docker, dentre outras;
- Implementação do sistema segundo os princípios da programação orientada a objetos, SOLID, segundo boas práticas, padrões de codificação limpas postuladas pelos padrões de uma empresa de tecnologia fundamentadas nos *Design Patterns* e contando com testes unitários, automatizados e de integração em todas as fases do projeto;

- Implantação: hospedagem da aplicação em um servidor Web da própria empresa onde se possa averiguar o resultado da aplicação produzida.
- Escrita da monografia: produzida em LaTeX utilizando o editor online Overleaf,[1] atendendo os requisitos das normas da ABNT (Associação Brasileira de Normas Técnicas) para elaboração de documentos técnicos e científicos brasileiros.

1.3 Organização da Monografia

Além desta introdução, esta monografia é composta por outros quatro capítulos:

- O Capítulo 2: apresenta uma revisão da literatura acerca de temas relevantes ao contexto deste trabalho, a saber: Engenharia de Software, FrameWeb, desenvolvimento Web, Arquitetura de Softwares e Gerência de Projetos;
- O Capítulo 3 apresenta uma exposição dos requisitos do sistema, com diagramas de classes e casos de uso, além disso, exibe ainda uma perspectiva do projeto do sistema, segundo arquiteturas de softwares utilizadas para o desenvolvimento da ferramenta, em especial o método FrameWeb;
- O Capítulo 4 apresenta detalhes da implementação do sistema, bem como capturas de tela do sistema em funcionamento;
- O Capítulo 5 apresenta as considerações finais do trabalho, os resultados obtidos até o momento e propostas para o prosseguimento do projeto.

[1] <https://www.overleaf.com/>

2 Fundamentação Teórica

Este capítulo apresenta os principais conceitos teóricos que fundamentam o desenvolvimento da ferramenta de Gestão de Projetos e está organizado em 6 seções. A Seção 2.1 apresenta os conceitos básicos e fundamentais da Engenharia de Software. A Seção 2.2 descreve sucintamente o desenvolvimento Web. A Seção 2.3 apresenta as arquiteturas de softwares adotadas para o desenvolvimento da ferramente proposta. A Seção 2.4 apresenta o método FrameWeb. A Seção 2.5 introduz conceitos e a evolução da Gestão de Projetos. Por fim, a Seção 2.6 aborda rapidamente definições das Metodologias Ágeis, em especial, o Scrum e o Kanban.

2.1 Engenharia de Software

Para um usuário, cliente ou novato da área da Computação, desenvolver software é, muitas vezes, confundido com programação. Essa visão, para situações de baixa complexidade, pode ser suficiente. Porém, chega-se a um ponto em que, dado o tamanho ou a complexidade do problema que se pretende resolver, essa abordagem não é mais indicada. Analogamente, em outras áreas de conhecimento, como na Engenharia Civil, um bom pedreiro poderá ser capaz de construir uma casinha de cachorro sozinho, sem projetos com plantas baixas ou terraplanagens. Talvez não seja a melhor solução, mas o resultado poderá corresponder a expectativa inicial. Entretanto, para se construir uma ponte ou mesmo um edifício, faz-se necessário um estudo aprofundado, incluindo análises do solo, cálculos estruturais, isto é, uma abordagem de engenharia. Num cenário semelhante surge a Engenharia de Software, visando melhorar a qualidade dos produtos de software e maximizar a produtividade no processo de desenvolvimento (FALBO, 2014).

A Engenharia de Software engloba processos, métodos e ferramentas que possibilitam a construção de sistemas complexos baseados em computador dentro do prazo e com qualidade (PRESSMAN, 2011).

Para melhorar a qualidade do software, pilar da Engenharia de Software, é preciso notar que qualidade é um conceito de múltiplas facetas. Para um usuário do sistema, um bom software é o que atende suas necessidades, sendo fácil de usar, eficiente e confiável. Por uma outra perspectiva, para um desenvolvedor, um produto de boa qualidade é o de fácil manutenção. Já para um cliente, o produto deve agregar valor ao seu negócio. O que há de comum nas várias perspectivas apresentadas é que todas são focadas no produto final de software. Entretanto para que tais características sejam atingidas é necessário que a qualidade seja incorporada ao longo do processo de desenvolvimento do produto (FALBO, 2014).

Nas subseções a seguir, procurou-se apresentar uma visão geral e sucinta dos principais processos da Engenharia de Software.

2.1.1 Especificação de Requisitos

Para começarmos as discussões dessas etapas, inicialmente, podemos definir requisito de software, segundo a norma IEEE-90, como sendo:

1. Uma capacidade que um usuário necessita para resolver um problema ou atingir um objetivo;
2. Uma capacidade que deve ser atendida ou possuída por um sistema ou componente de um sistema para satisfazer um contrato, padrão, especificação ou outro documento formalmente imposto;
3. O conjunto de todos os requisitos que formam a base para o desenvolvimento subseqüente de um software ou componentes de um software;

Em outras palavras, os requisitos são as descrições do que o sistema deve fazer, os serviços e processos que atendam as necessidades do cliente ou restrições operacionais da organização. O processo de descobrir, analisar, documentar e verificar esses serviços e restrições é chamado de Engenharia de Requisitos (SOMMERVILLE, 2011).

De maneira geral, o processo de Engenharia de Requisitos começa pelo levantamento de requisitos. Nesta fase, os usuários e clientes trabalham junto com o engenheiro de requisitos para entender as necessidades que o software deve atender e os problemas e deficiências dos sistemas atuais (FALBO, 2014).

Uma vez identificados requisitos, é possível iniciar atividades de análise, que visam o acordo na negociação entre usuários para resolver conflitos detectados, resultando em requisitos consistentes e sem ambiguidades. A etapa de análise é essencialmente uma atividade de modelagem, preocupando-se apenas com o domínio do problema e não com soluções técnicas. Diferentes modelos podem ser construídos para representar diferentes perspectivas (FALBO, 2014).

Tanto no levantamento de requisitos quanto na análise, é importante documentar requisitos e modelos. Para estas etapas utiliza-se o documento de especificação de requisitos: contendo uma lista dos requisitos de usuário levantado, junto a uma visão geral do problema a ser resolvido, além do registro de vários diagramas resultantes do trabalho de análise, como o modelos de casos de uso e o modelo estrutural. O documento produzido é, então, verificado e validado. Caso clientes, usuários e desenvolvedores estejam de acordo com os requisitos, o processo de desenvolvimento pode avançar; caso contrário, deve-se retornar à atividade correspondente para resolver os problemas identificados (FALBO, 2014).

2.1.2 Projeto

O objetivo da fase de projeto é produzir uma solução para o problema identificado modelado durante o levantamento e análise de requisitos, incorporando a tecnologia aos requisitos e projetando o que será construído na implementação (FALBO, 2014).

O projeto de software encontra-se no núcleo técnico do processo de desenvolvimento de software e corresponde à primeira atividade que leva em conta aspectos tecnológicos (PRESSMAN, 2011). Independentemente do paradigma adotado, a fase de projeto deve produzir: um projeto de arquitetura do software, que visa definir os grandes componentes estruturais do software e seus relacionamentos, projeto de dados, que tem por objetivo projetar a estrutura de armazenamento de dados necessária para implementar o software, projeto de interfaces, que descreve como o software deverá se comunicar dentro dele mesmo, com outros sistemas e com pessoas que o utilizam e o projeto detalhado, que tem por objetivo refinar e detalhar a descrição dos componentes estruturais do software (FALBO, 2014).

2.1.3 Implementação e Teste

Uma vez projetado o sistema, o estágio mais crítico desse processo é, naturalmente, a implementação do sistema. A implementação pode envolver o desenvolvimento de programas em alto ou baixo nível de linguagens de programação, bem como customização e adaptação de sistemas de prateleira, para atender aos requisitos específicos de uma organização (SOMMERVILLE, 2011).

É muito importante que haja padrões organizacionais para a fase de implementação. Esses padrões devem ser seguidos por todos os programadores e devem estabelecer, dentre outros, padrões de nomes de variáveis, formato de cabeçalhos de programas e formato de comentários, recuos e espaçamento, de modo que o código e a documentação a ele associada sejam claros para quaisquer membros da organização (FALBO, 2014).

Alguns aspectos de implementação particularmente importantes para a Engenharia de Software são: o reúso, os softwares são construídos por meio de reuso de componentes existentes ou sistemas, o gerenciamento de configuração, gerenciamento do versionamento de componentes de software e desenvolvimento *host-target*, separação de ambiente de desenvolvimento e produção (SOMMERVILLE, 2011).

Teste de software é o processo de executar um programa com o objetivo de encontrar defeitos (MYERS et al., 2004). Teste é uma atividade de verificação e validação do software e consiste na análise dinâmica do mesmo, isto é, na execução do produto de software com o objetivo de verificar a presença de defeitos no produto e aumentar a confiança de que o mesmo está correto (ROCHA et al., 2001).

Dada sua importância, testes não devem ser tratados apenas como uma atividade

no ciclo de vida do software, mas sim como um processo. O processo de teste deve ocorrer em paralelo com outras atividades do processo de desenvolvimento de software (análise de requisitos, projeto de software e implementação) e envolve também atividades de planejamento (FALBO, 2014).

Os testes devem ser feitos em diversos níveis, começando pelos teste unitários, em que as unidades individuais de programa ou classes de objetos são testadas individualmente, testes de integração, em que várias unidades individuais são integradas para criar componente compostos e teste de sistema, em que alguns ou todos os componentes de um sistema estão integrados e o sistema é testado com um todo (SOMMERVILLE, 2011).

2.1.4 Entrega e Manutenção

Uma vez concluídas as etapas anteriores e o sistema aceito e instalado, se está chegando ao fim do processo de desenvolvimento de software. A entrega não é meramente uma formalidade. No momento em que o sistema (ou uma versão dele) é instalado no local de operação e devidamente aceito, é necessário, ainda, ajudar os usuários a entenderem e a se sentirem mais familiarizados com o sistema. Neste momento, duas questões são cruciais para uma transferência bem-sucedida: o treinamento, para que os usuários e operadores possam operar o sistema adequadamente e a documentação, como material de referência para a solução de problemas ou como informações adicionais (FALBO, 2014).

O desenvolvimento de um sistema termina quando o produto é entregue para o cliente e entra em operação. A partir daí, deve-se garantir que o sistema continuará a ser útil e atendendo às necessidades do usuário, o que pode demandar alterações no mesmo. Começa, então, a fase de manutenção ou evolução (SANCHES, 2001).

Há muitas causas para a manutenção, dentre elas (SANCHES, 2001): falhas no processamento devido a erros no software e etc., levando à necessidade de modificações em funções existentes ou de inclusão de novas capacidades no sistema. O processo de manutenção é semelhante, mas não igual ao processo de desenvolvimento, e pode envolver atividades de levantamento de requisitos, análise, projeto, implementação e testes, agora no contexto de um software existente. Essa semelhança pode ser maior ou menor, dependendo do tipo de manutenção a ser realizada (FALBO, 2014).

2.2 Desenvolvimento Web

A Internet nasceu de um projeto de pesquisa militar (ARPA: *Advanced Research Projects Agency*), no período da guerra fria, no final dos anos cinquenta e início dos anos sessenta. Este projeto surgiu como resposta do governo americano ao lançamento do Sputnik, pela ex-União Soviética. Através da rede, o governo norte-americano se protegeria e garantiria a fluência das comunicações, caso a guerra fria e os momentos posteriores ao

evento histórico fossem favoráveis à ascensão da União Soviética. Mas a linguagem utilizada nos computadores ligados em rede era muito complicada, por isso, na época, o potencial de alastramento da Internet não podia ser imaginado (MERKLE; RICHARDSON, 2000).

A *World Wide Web*, mais lembrada nos dias atuais como, simplesmente, Web, começou em 1989 no *European Center for Nuclear Research* (CERN), apresentando uma estrutura arquitetônica que permite o acesso a documentos vinculados espalhados por milhões de máquinas na Internet. Desde então, a Web cresceu de forma desproporcional e descontrolada. Em 1994, por exemplo, havia menos de 3.000 sites online. Em 2014, havia mais de 1 bilhão. Isso representa um aumento de 33 milhões por cento em 20 anos (LAFRANCE, 2015).

A Web pode ser analisada do ponto de vista da arquitetura ou do cliente, conforme almejou-se apresentar sumariamente nas próximas duas subseções a seguir.

2.2.1 Arquitetura

A Web pode ser vista como uma vasta coleção mundial de documentos, comumente chamados de páginas. As páginas, majoritariamente, são visualizadas com o auxílio de um software chamado navegador, pode-se destacar os navegadores mais usados em 2021, segundo A 4GNEWS, como sendo o Google Chrome, Safari e Mozilla Firefox.

O navegador exibe uma página Web na máquina do cliente quando envia uma solicitação a um ou mais servidores, que respondem com o conteúdo da página. O protocolo de solicitação-resposta para buscar páginas é simples, baseado em texto, que roda sobre o *Transmission Control Protocol* (TCP). Ele é chamado *Hyper Text Transfer Protocol* (HTTP). A especificação, elaborada pelo W3C, o define como:

"HTTP é um protocolo de nível de aplicação para sistemas de hipermídia, colaborativos e distribuídos. É um protocolo genérico, sem estado e orientado a objetos que pode ser usado para diversas tarefas, tais como servidores de nomes e sistemas de gerenciamento de objetos distribuídos, através da extensão de seus métodos de requisição" (FIELDING, 1997).

O protocolo HTTP é o ingrediente mais básico sobre o qual a Web está fundamentada. Através dele, o cliente (navegador ou dispositivo que fará a requisição) pode requisitar recursos disponíveis a um servidor remoto. O conteúdo pode simplesmente ser um documento que é lido de um disco, ou o resultado de uma consulta de banco de dados. Quanto à página, ela pode ser uma página estática, se apresentar o mesmo documento toda vez que for exibida, ou uma página dinâmica, se ela foi gerada sob demanda por um programa ou se contém um programa. Uma página dinâmica pode se apresentar de forma diferente toda vez que for exibida (TANENBAUM, 2011). Como esses recursos estão distribuídos ao longo de toda a Internet, é necessário um mecanismo de identificação que

permita localizá-los e acessá-los. O identificador usado para referenciar esses recursos é uma URL (CASTELEYN et al., 2009).

2.2.2 Cliente

Cada página recebe um *Uniform Resource Locator* (URL), que efetivamente serve como o nome identificador da página. As URLs têm pelo menos três partes: o protocolo, também chamado de esquema, usado para transferência de arquivos, o *host* ou nome *Domain Name System* (DNS) da máquina em que a página está localizada e o caminho que identifica exclusivamente a página específica. No caso geral, o caminho tem um nome hierárquico que modela uma estrutura de diretório de arquivo. Porém, a interpretação do caminho fica a critério do servidor; ele pode ou não refletir a estrutura de diretório real.

O *HyperText Markup Language* (HTML), é uma linguagem de marcação utilizada na construção de páginas na Web, marcando os arquivos de texto lidos pelos navegadores e enviados pelos servidores. Não se trata de uma linguagem de programação. Um arquivo HTML é um arquivo de texto comum recheado de marcadores que se destacam do texto pelos caracteres especiais "<" e ">". Existem dois tipos de extensões do lado do cliente, os componentes que funcionam como extensões executadas como se fossem parte do navegador e os scripts que estendem a linguagem HTML. Os scripts são interpretados enquanto o navegador carrega a página. O próprio código HTML é um script que é interpretado pelo navegador para definir a estrutura da página (ROCHA, 1999). Um grande exemplo de script em alto nível, é a linguagem de programação interpretada JavaScript. Atualmente é a principal linguagem de programação Web no lado do cliente. Entretanto, também é muito utilizada no lado do servidor por meio de ferramentas como, por exemplo, o Node.js.

Nas próximas subseções, serão apresentados as plataformas Web baseadas no JavaScript, de forma breve, que serão utilizadas no desenvolvimento da ferramenta de gestão de projetos.

2.2.3 Angular

O Angular é um *framework* Javascript, criado pelos desenvolvedores do Google, não apenas para construção de interface de usuário, mas também para aplicações únicas do lado do cliente, sejam elas para a Web, *mobile* ou *desktop*.

O Angular possui alguns elementos básicos que tornam essa construção mais interessante, podemos destacar os componentes, serviços, módulos, diretivas e ferramentas de infraestrutura que automatizam tarefas, como a execução de testes unitários. Conta com uma ferramenta chamada de Angular CLI, ferramenta de linha de comando que facilita a criação desses elementos.

O *framework* é ideal para construção de *single-page applications* (SPAs), aplicações

nas quais o usuário não precisa aguardar o recarregamento de toda a página esperando uma atualização completa ou parcial, melhorando principalmente a usabilidade da aplicação.

2.2.4 Node.js

Node.js é uma tecnologia de execução assíncrona orientada a eventos, usada para executar código JavaScript fora do navegador. Com ele podemos construir aplicações Web em geral, desde *websites* até APIs e microsserviços. Isso é possível pela união do ambiente de execução de JavaScript fornecido pelo próprio Node.js e o motor de interpretação e execução de JavaScript presente no Google Chrome, chamado de V8, que foi criado com o objetivo de permitir redes escaláveis. Node.js usa um modelo de E/S (Entrada e Saída) orientado a eventos e não bloqueante, o que o torna leve e eficiente, ideal para aplicações em tempo real com troca intensa de dados executadas em dispositivos distribuídos (NODEJS.ORG, 2021).

Uma das principais características de um servidor feito com Node.js é o fato de sua execução ser *single-threaded*: a cada conexão, o retorno de chamada é disparado, mas se não houver trabalho a ser feito, o Node.js ficará inativo. Isso contrasta com outros servidores tradicionais que possuem uma execução *multi-threaded*, no qual *threads* do sistema operacional são empregadas. A rede baseada em *thread* é relativamente ineficiente e muito difícil de usar. Além disso, os usuários do Node.js estão livres da preocupação de travar o processo. Quase nenhuma função no Node.js realiza E/S diretamente, portanto, o processo nunca é bloqueado. Como nada bloqueia, sistemas escaláveis são muito razoáveis para desenvolver em Node.js (NODEJS.ORG, 2021).

Porém, o Node.js, não possui suporte nativo para tarefas comuns no desenvolvimento Web, como o tratamento para métodos HTTP, ou especificações de rotas. Ainda assim, resolve-se facilmente utilizando um *framework*. Na ferramenta de gestão de projetos, foco desse texto, será utilizado o *framework* mais popular de Node.js, o Express.js. Este *framework*, fornece um conjunto robusto de recursos, completando o arsenal Node e possibilitando o desenvolvimento de um sistema Web escalável.

2.3 Arquitetura de Software

A seguir, serão apresentadas as arquiteturas de software que foram escolhidas para o desenvolvimento do sistema GanttBox. Em cada subseção, tentou-se apresentar de forma sucinta e instrutiva as arquiteturas, de forma que também expusessem sua vantagens e os motivos da escolha.

2.3.1 Clean Architecture

A arquitetura de software do sistema de gestão de projetos baseia-se primordialmente na *Clean Architecture*, ou Arquitetura Limpa, proposta idealmente por Robert Cecil Martin (MARTIN; GRENNING; BROWN, 2018). A Figura 1 mostra a arquitetura do sistema.

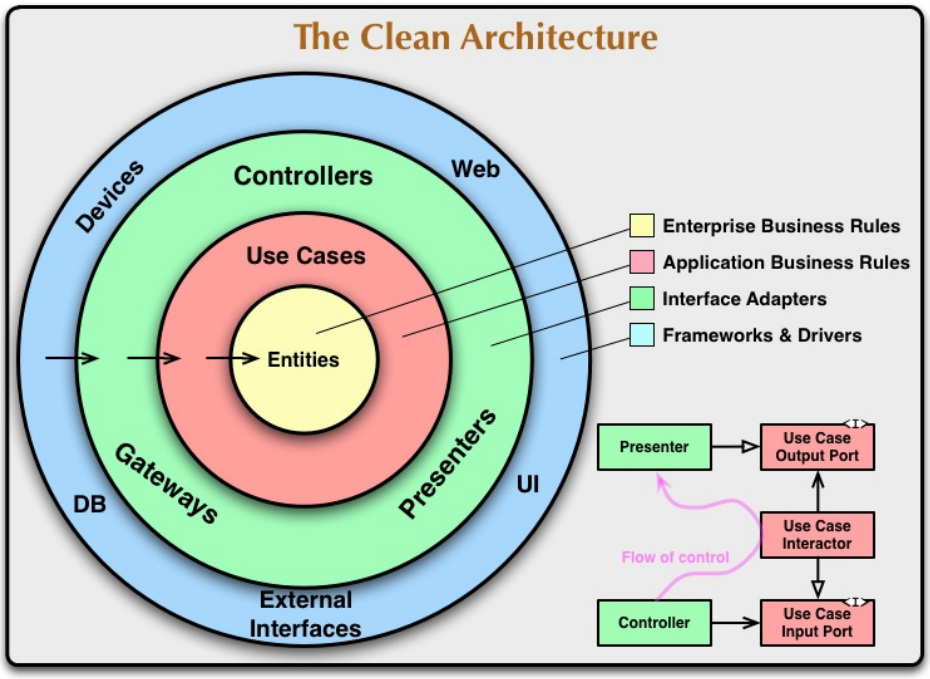


Figura 1 – Arquitetura proposta pela *Clean Architecture* (MARTIN; GRENNING; BROWN, 2018).

O núcleo, camada mais interna do diagrama, é a camada de Entidades (*Entities*), as classes modelos, que representam as tabelas ou *collections* do banco de dados dentro do projeto. Numa camada acima, de comunicação direta com a entidades, observamos os Casos de Uso (*Use Cases*), onde se encontram todas as regras de negócio da aplicação. Em seguida, a camada dos Adaptadores (*Adapters*), esta camada é a camada dos controladores, que se comunicam com as requisições advindas dos elementos externos. Concluímos com a camada mais externa, *frameworks* externos, onde se encontram todos os elementos externos que necessitam se comunicar com a nossa camada de entidades, como bancos de dados, *frameworks* e interfaces.

São vários os princípios das boas práticas de programação seguidos na obra para adoção da arquitetura limpa que, quando bem aplicada, permite, entre vários benefícios, escalar o projeto de software com facilidade. Um dos principais padrões que podemos pontuar é o SOLID, segundo Martin, Grenning e Brown (2018), um acrônimo para os seguintes princípios:

- Princípio da responsabilidade única (*Single-responsibility principle*): uma classe deve

ter um, e somente um, motivo para mudar;

- Princípio aberto-fechado (*Open–closed principle*): objetos ou entidades devem estar abertos para extensão, mas fechados para modificação;
- Princípio de substituição de Liskov (*Liskov substitution principle*): uma classe derivada deve ser substituível por sua classe base;
- Princípio de segregação de interfaces (*Interface segregation principle*): uma classe não deve ser forçada a implementar interfaces e métodos que não irão utilizar;
- Princípio da inversão de dependência (*Dependency inversion principle*): dependa de abstrações e não de implementações.

Existem inúmeras vantagens em se utilizar uma arquitetura dividida em camadas, em especial, projetos de software que adotam a arquitetura limpa com maestria apresentam as seguintes características:

- Independência de qualquer agente externo: a arquitetura não depende de nenhum *framework* ou biblioteca, permitindo o uso das ferramentas como apoio ao desenvolvimento ao invés de limitar o projeto com seus respectivos padrões;
- Independência de banco de dados: as regras de negócio não estão ligadas as particularidades de um determinado banco, a troca de um banco Oracle para PostgreSQL ou MongoDB, pode ser feita sem impacto no funcionamento do negócio;
- Independência da interface com o usuário: a interface do usuário pode ser facilmente substituída sem impactar o restante do sistema, uma interface console pode ser trocada por uma Web, sem o impactar as regras de negócio;
- Testabilidade: como as camadas estão bem definidas, as regras de negócio podem ser testadas sem a interface do usuário, banco de dados, servidores Web ou qualquer outro elemento externo.

Numa arquitetura bem definida, as dependências apontam para o nível mais alto da aplicação, analogamente, na *clean architecture*, as dependências apontam para as camadas mais internas. As entidades não devem saber nada sobre a camada de Casos de Uso, assim como a camada de Casos de Uso não deve saber nada sobre a camada mais externa e assim sucessivamente. Na prática, a alteração de um controlador não deve interferir em nada no comportamento dos casos de usos, igualmente nada nas entidades, as camadas mais externas não influenciam no comportamento das camadas mais internas. Uma arquitetura separada em camadas poupará muito esforço dos desenvolvedores com futuro problemas de manutenção do código. Um problema de regra de negócio específico estará na classe

especialista daquele assunto. A regra de dependência torna o sistema completamente testável. Assim, quando uma biblioteca ou banco de dados se torna obsoleto, a troca do mesmo poderá ser feita de forma hábil, garantindo integridade do sistema como um todo.

2.3.2 Multitenancy

Softwares empresariais tradicionais são instalados em computadores e precisam de um administrador para mantê-los. Esta maneira tradicional de operação pode limitar um negócio quando diferentes departamentos precisam trabalhar a partir de uma mesma base da dados, ou simplesmente pelo custo de infraestrutura. Com a computação em nuvem, atualmente, um modelo de disponibilizar serviços de software vem ganhando espaço no mercado, os chamados Saas (*Software as a Service*), onde uma empresa disponibiliza uma solução tecnológica pela Internet. Com este modelo, a empresa que contrata o serviço desta empresa fornecedora da solução só precisará de acesso à Internet para utilizar os recursos desejados, isto é, uma empresa utilizando serviços de outra empresa, numa relação B2B (*Business to Business*).

Nesse contexto, faz-se o projeto do sistema GanttBox, um sistema que será desenvolvido oferendo serviços de gestão, inicialmente implantado dentro de um cenário conhecido, mas intencionalmente pensado para escalar vários inquilinos. Diante disso, optou-se pela escolha da adoção da arquitetura *multitenant*. *Multitenant* ou *multitenancy* é uma arquitetura de software projetada para que uma única instância da aplicação centralizada, seja executada em um servidor e que atenda vários clientes (KREBS; MOMM; KOUNEV, 2012).

Nos softwares tradicionais ou no modelo tradicional de arquitetura *singletenant*, para cada novo cliente haveria uma instância da aplicação distinta servindo cada cliente de maneira isolada, nesse cenário, quando o software precisa-se ser atualizado, seria necessário também atualizar os inúmeros servidores, havendo risco de um ou mais clientes ficarem sem a devida atualização. O fato é, aplicando a arquitetura *multitenant*, conseguimos diminuir a complexidade de gestão do software, pois tendo uma única instância de software não haverá risco de um pontual cliente ficar desatualizado. Além disso, podemos levantar a economia com infraestrutura, mesmo precisando de uma instância mais robusta para atender *multitenants* será menos custosa de manter pensando em um número razoável de clientes.

Assumindo esta arquitetura como mais adequada para este sistema, caímos na questão da segregação dos dados: como existirá uma única instância de software, precisamos separar os dados dos diferentes *tenants* da nossa aplicação. A abordagem que optou-se para o desenvolvimento do sistema de gestão foi a segregação dos dados via esquemas diferentes. Nesta estratégia, existe uma única instância de banco de dados, mas essa instância possui múltiplos conjuntos de tabelas separados por cada *tenant*. Na aplicação que está sendo

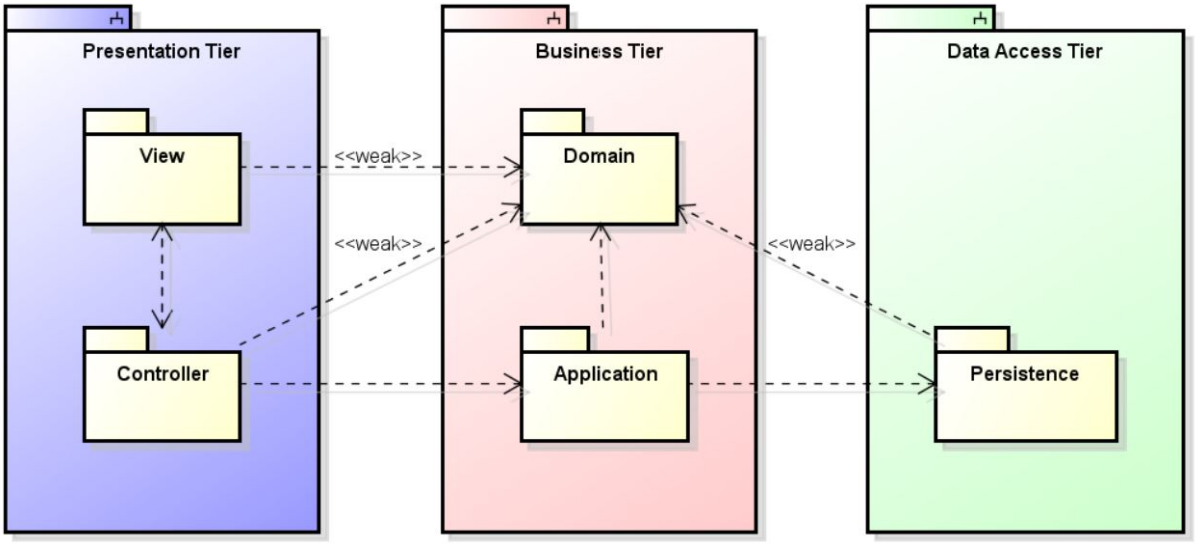


Figura 2 – Arquitetura Proposta pelo FrameWeb (CAMPOS; SOUZA, 2017).

proposta nesse texto, toda vez que um cliente criar uma conta como representação de uma instituição, será criado um novo esquema de banco dados exatamente com as mesma modelagem de tabelas. Esta estratégia é mais barata, pois se usará uma única instância de banco e ainda garantirá o isolamento lógico dos dados de cada cliente por esquemas.

2.4 FrameWeb

FrameWeb surgiu como um método de projeto para construção de sistemas de informação Web (*Web Information Systems* – WISs) baseado em *frameworks*. Em linhas gerais, assume-se que determinados tipos de *frameworks* serão utilizados durante a construção da aplicação, define uma arquitetura básica para o WIS e propõe modelos de projeto que se aproximam da implementação do sistema usando esses *frameworks* (SOUZA, 2007).

Tanto o método FrameWeb quanto sua ferramentas, estão em constante desenvolvimento, pesquisadores seguem trabalhando em diferentes aspectos da proposta. Atualmente, o método dedica suas atenções estritamente na fase de projeto arquitetural, sendo elas:

- Definição de uma uma arquitetura lógica padrão para WISs, que divide o sistema em camadas, baseada no padrão arquitetônico *Service Layer* (Camada de Serviço), proposto por Randy Stafford em (FOWLER, 2002). Segundo Souza (2020), a arquitetura é composta por três camadas conforme a Figura 2:
  - O pacote Visão contém arquivos relacionados exclusivamente com a exibição de informações ao usuário, como páginas Web, folhas de estilo, imagens e scripts que executam do lado do cliente. O pacote Controle, envolve classes que controlam as requisições vindas do usuário. Juntos formam a primeira camada, **Apresentação**, que tem por objetivo prover interfaces gráficas com o usuário;

  - O pacote de Domínio contém classes que representam conceitos do domínio do problema identificados e modelado pelos diagramas de classes na fase de requisitos e refinados durante o projeto. O pacote Aplicação, tem a responsabilidade de implementar os casos de uso definidos na especificação de requisitos, provendo uma camada de serviços independente da interface com o usuário. Juntos, esses pacotes consolidam a lógica de negócio e formam a segunda camada, **Negócio**;
  - A terceira e última camada, **Acesso de Dados**, contém apenas o pacote Persistência, que é responsável pelo armazenamento dos objetos persistentes em mídia de longa duração, como banco de dados, arquivos, serviços de nome, etc.

- A partir da arquitetura definida, é proposto um conjunto de modelos de projeto que trazem conceitos utilizados pelos *frameworks* para esta fase do processo mediante a criação de um perfil, baseado no diagrama de classes da UML, que aproxima a fase de implementação, engenhando o desevolvimento com base nesses modelos. Esses modelos, segudo (CAMPOS; SOUZA, 2017), se dividem em quatro tipos:

  - **Modelo de Entidades**: representa os objetos de domínio do problema e seu mapeamento para a persistência em banco de dados;
  - **Modelo de Aplicação**: representa as classes de serviço, responsáveis pela implementação das funcionalidades do WIS, e suas dependências;
  - **Modelo de Persistência**: representa as classes Data Access Object (DAO) existentes, responsáveis pela persistência das instâncias das classes de domínio;
  - **Modelo de Navegação**: representa os diferentes componentes que formam a camada de apresentação, como páginas Web, formulários HTML, etc.

2.5 Gestão de Projetos

Projeto é um esforço temporário empreendido para criar um produto, serviço ou resultado único (CRUZ, 2013). Assim como a Engenharia de Software nasceu com o intuito de prestar um tratamento mais sistemático no processo de desenvolvimento de software, numa linha muito semelhante, a gestão de projetos se apresenta com a estruturação da forma como o projeto é planejado, executado, monitorado e controlado, independentemente da área de aplicação, a fim de otimizá-lo e proporcionar uma série de benefícios.

A gestão de projetos atual se desenvolveu para incluir indústrias, como setores de produção de energia, siderúrgicas e quaisquer outros ramos, possuindo uma tendência ágil, com características de equipes auto gerenciadas e autônomas para tomar decisões (CONFORTO et al., 2014). Entretanto, a gestão de projetos existe há milhares de anos. Construções como a pirâmide de Gizé e a muralha da China indagam arquitetos e engenheiros ainda hoje sobre como conseguiram tal feito. Ao longo da história, vários foram

os fatores que contribuíram para evoluir e moldar a gestão de projetos como articulada nos dias atuais.

Devemos lembrar de Henry Gantt, considerado por muitos o fundador da gestão de projetos. Na primeira metade do século 20, os líderes empresariais começaram a enfrentar os desafios das leis e regulamentos trabalhistas do governo. Gantt desenvolveu técnicas de planejamento e controle para ajudar as empresas a terem sucesso e cumprirem os regulamentos.

Outro marco importante a se citar é a Associação Internacional de Gestão de Projetos (IPMA), iniciada em Viena no ano de 1965, visando promover a gestão de projetos e liderar o desenvolvimento da profissão.

Como já vem sendo discutido ao longo desse texto, o avanço da Computação está contribuindo de forma notável e imediata nas estratégias de negócios das organizações, moldando a gestão de projetos. Hoje, a automação dessa gestão em um empreendimento não é mais um luxo, e sim, uma necessidade para a empresa se manter competitiva no mercado.

A ideia que motiva esse trabalho não é revolucionária, nem mesmo novidade. O mercado de hoje está recheado de ferramentas que articulam a gerência de projetos, porém a grande maioria dessas ferramentas desempenha papéis pontuais no meio do processo de gestão ou são financeiramente inacessíveis para pequenas empresas. A ferramenta de gestão que está sendo pensada e proposta nesse trabalho mira neste alvo e tenta abranger a grande maioria de funcionalidades desses softwares que acredita-se ser suficientes para gerir um pequeno negócio.

2.6 Metodologias Ágeis

Como vem sendo mencionado ao longo deste texto, à medida que se aumenta a complexidade de um produto, especificamente aqui, de um *software*, se faz necessário seguir metodologias para estruturar e sistematizar toda a complexidade do negócio envolvida na solução.

Especialmente na década de 1990, a indústria de desenvolvimento de *software* enfrentava dificuldades nos seus projetos por tentar imitar estratégias de gestão que funcionavam bem em outras áreas da Engenharia, porém não se enquadravam perfeitamente para projetos de *softwares*. No decorrer na década, foram surgindo métodos que se mostravam mais assertivos. Assim, em 2001 representantes destes métodos se reuniram em Utah, Estados Unidos, percebendo que os métodos possuíam filosofias fundamentais em comum, desta forma, durante este encontro, foram definidos vários valores e princípios que hoje são conhecidos como “O Manifesto Ágil” (FOWLER; HIGHSMITH et al., 2001).

Métodos que seguem estas filosofias são então chamados de métodos ágeis.

Scrum e Kanban são dois métodos ágeis muitos utilizados atualmente. Na prática, estes métodos e customizações dos mesmos acabam sendo combinados para melhor atender as necessidades do negócio. O mesmo ocorre nas estratégias de otimizações gerenciais propostas na ferramenta produzida neste trabalho.

O Scrum é um método ágil, que define um processo de desenvolvimento de projetos focado nas pessoas da equipe. O nome "Scrum" surgiu da rápida reunião que ocorre quando os jogadores do esporte *rugby* vão iniciar um lance. A analogia entre jogadores e desenvolvedores foi usada pois cada time do esporte age em conjunto como uma unidade integrada, cada membro desempenha um papel específico e todos se ajudam em busca de um objetivo comum. Ele baseia-se em algumas características (SCHWABER, 1997): flexibilidade dos resultados, flexibilidade dos prazos, times pequenos, revisões frequentes e colaboração.

O Kanban é um método ágil, que visa identificar possíveis gargalos, realizando as devidas correções para que haja fluidez nas atividades de uma empresa. O nome "Kanban" é de origem japonesa e significa "sinalização" ou "cartão". Sendo um método visual que busca gerenciar o trabalho conforme a movimentação de cartões, que nesta monografia chamamos de *cards*. A proposta do uso de cartões somado as suas movimentações, visa acompanhar o andamento da produção de uma atividade dentro de um fluxo de trabalho de uma equipe (HUANG; KUSIAK, 1996).

3 Modelagem de Software

Este capítulo apresenta os resultados das etapas de modelagem do software GanttBox, com o objetivo de explicitar as características e comportamentos almejados, facilitando a compreensão geral do sistema de gestão proposto neste texto. Na Seção 3.1, é apresentada de forma sintética a especificação de requisitos de sistema. Na Seção 3.2, é apresentado o projeto de software. A íntegra dos resultados produzidos nestas etapas da engenharia de software, pontuadas nas seções 2.1.1 e 2.1.2, podem ser encontradas no apêndice desta monografia pelos documentos de especificação de requisitos e projeto de sistema.

3.1 Especificação de Requisitos

Em nossa opinião, em qualquer área do conhecimento, qualificar o resultado final de um produto, está intimamente ligado a otimizar a qualidade do seu processo de produção. Dependendo da complexidade do bem almejado, somada à calorosa disputa de mercado que nosso sistema econômico molda, um processo de engenharia sistemático se faz necessário para se atingir o resultado esperado e manter-se diligente no mercado.

O objetivo da ferramenta de gestão de projetos é unir os conhecimentos da área de gestão, junto com a Engenharia de Software, para produzir uma aplicação que irá integrar várias funcionalidades de gerência em um só lugar, almejando suprir, principalmente, as necessidades dos gestores de uma pequena empresa de tecnologia no município de Vitória, ES.

3.1.1 Conjuntura

A cada dia mais cresce a procura por soluções na área de TI (Tecnologia da Informação). Neste ecossistema, habita uma empresa de tecnologia, que retém um quadro de funcionários qualificados, com um ambiente e equipamentos favoráveis para atender esta demanda do mercado. Entretanto, a equipe administrativa do negócio leva a rotina de forma manual, como o preenchimento de extensas planilhas, tarefas repetitivas, cansativas e até caóticas, contendo a expansão da instituição.

No cenário da pequena instituição de crescimento emergente, onde, cada vez mais, surgem demandas de serviços e contatos de novos clientes, o sistema *GanttBox*, será um mediador das execuções de atividades de cada figura deste quadro, organizando e estruturando os processos com a finalidade de consignar poder e conhecimento do negócio ao seu administrador. Como uma solução mediatária, a ferramenta de gestão propõe disponibilizar vários recursos e funcionalidades para seus usuários dentro do sistema para,

assim, tornar-se a solução que almeja ser. Cada um desses recursos poderá ser encontrado em um dos dois principais menus, descritos na sequência.

3.1.2 Vendas

O colaborador gestor, um dos atores do sistema, possuirá a responsabilidade de registrar a conta para uma particular instituição começar a utilizar o *GanttBox*. Após **criar conta** e se **autenticar**, terá acesso a todas as funcionalidades do sistema. Estritamente o gestor será o único ator que possuirá acesso a seção de vendas.

No menu de vendas, acessará o funil de vendas, ou *Kanban* de vendas, quadro que visa organizar, melhorar e conduzir o fluxo de trabalho. No funil de vendas, o gestor poderá **gerenciar leads**. *Leads* são pessoas físicas, potenciais futuros clientes. O colaborador gestor também poderá **alterar o status da lead no Kanban**, pois cada *lead*, representará um *card* no *Kanban* de vendas. Ainda poderá **gerenciar propostas**, **propor proposta**, **gerenciar contratos**, **firmar contratos**, **exportar contratos** para arquivos PDF, **gerenciar clientes** e **declara ganho/perda para a lead** que na dinâmica do sistema se traduziria em: transformar a *lead* em cliente para declaração de ganho e assim concluir a negociação com sucesso ou declarar perda para a *lead* encerrando a negociação em fracasso. As atividades de gerencia incluem a visualização, criação, edição e exclusão dos mesmos.

3.1.3 Projetos

No menu de projetos, o colaborador gestor poderá **gerenciar os projetos**, visualizando todos os projetos da empresa, bem como **gerenciar setores** e **gerenciar colaboradores**.

Ao selecionar um projeto específico, na lista de projetos, visualizará mais informações e poderá **gerenciar o funil de projetos** e **gerenciar sprints**. *Sprints*, podem ser vistas como etapas, partes ou ciclos do projeto. O gestor poderá ainda **gerenciar tarefas**, **anexar imagens e documentos numa tarefa** e **alterar status de uma tarefa no kanban**, pois cada tarefa, representará um *card* no *Kanban* de projetos. O *card* é o componente central do método *Kanban*, um elemento visual que resume as informações de uma atividade dentro de um fluxo de trabalho.

Todos os usuários do sistema terão acesso à seção de projetos, entretanto possuirão diferentes permissões. O colaborador comum, um segundo ator do sistema, após se **autenticar**, quando alocado dentro de um projeto pelo gestor, conseguirá visualizar todos os projetos que lhe foram atribuídos. Ao selecionar um projeto específico na sua lista, visualizará mais informações e poderá **gerenciar tarefas**, **anexar imagens e documentos numa tarefa** e **alterar status de uma tarefa no kanban**. O colaborador comum ainda deverá **apontar horas** de trabalho em projetos específicos, que dentro do

fluxo da ferramenta será um *click* em "*play/stop*" dentro do *card* que representa uma tarefa no *Kanban* do projeto, dessa forma, suas horas trabalhadas naquela tarefa serão vinculadas para mais tarde o gestor extrair informações desses registros.

Um último ator do sistema, será o cliente. Este usuário, que foi criado após uma negociação e selamento de contrato, após se **autenticar**, poderá visualizar os projetos que contratou com a empresa. Selecionando um projeto na lista, visualizará mais informações e poderá **visualizar tarefas no Kanban**, conseguindo assim, acompanhar o progresso do projeto pelo sistema.

Além de todos estes recursos pontuados todos os atores mencionados (colaborador gestor, colaborador comum e cliente) poderão **trocar senha** e **alterar dados cadastrais**.

3.1.4 Subsistemas

A fim de facilitar a modelagem e organização do projeto, o sistema foi estruturado em 3 subsistemas: ProjectManagement, SalesManagement e UserManagement.

A Figura 3 ilustra os subsistemas e suas interdependências, enquanto a Tabela 1 apresenta uma breve descrição de cada subsistema identificado.

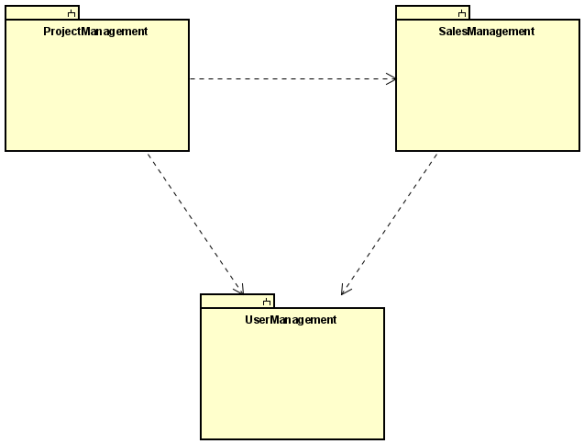


Figura 3 – Diagrama de Pacotes e os Subsistemas Identificados.

Tabela 1 – Subsistemas identificados e suas interdependências.

Subsistema	**Descrição**
SalesManagement	Subsistema responsável pelas funcionalidades relacionadas ao cadastramento, alteração, visualização e gestão de vendas.
ProjectManagement	Subsistema responsável pelas funcionalidades relacionadas ao cadastramento, alteração, visualização e gestão de projetos.
UserManagement	Subsistema responsável pelo gerenciamento e autenticação de usuários no sistema.

3.1.5 Casos de Uso

O modelo de casos de uso corresponde a uma tentativa de descrever a relação das funcionalidades do sistema com cada um de seus atores. Os atores identificados no contexto deste projeto estão descritos na Tabela 2. A tabela completa de requisitos encontra-se no Documento de Especificação de Requisitos, no apêndice desta monografia.

Tabela 2 – Descrição dos atores envolvidos nos casos de uso.

Ator	**Descrição**
Cliente	Pessoa que está contratando o serviço da empresa para desenvolvimento de um ou mais projetos.
Colaborador Comum	Funcionário da empresa que não trabalha diretamente com a gestão do negócio. Exemplo: Desenvolvedor.
Colaborador Gestor	Funcionário gestor administrativo, detentor de todas as permissões de acesso e visualização do sistema.

A seguir, são apresentados os diagramas de casos de uso e descrições associadas, organizados por subsistema.

3.1.5.1 Subsistema SalesManagement

A Figura 4 apresenta o diagrama de casos de uso do subsistema SalesManagement.

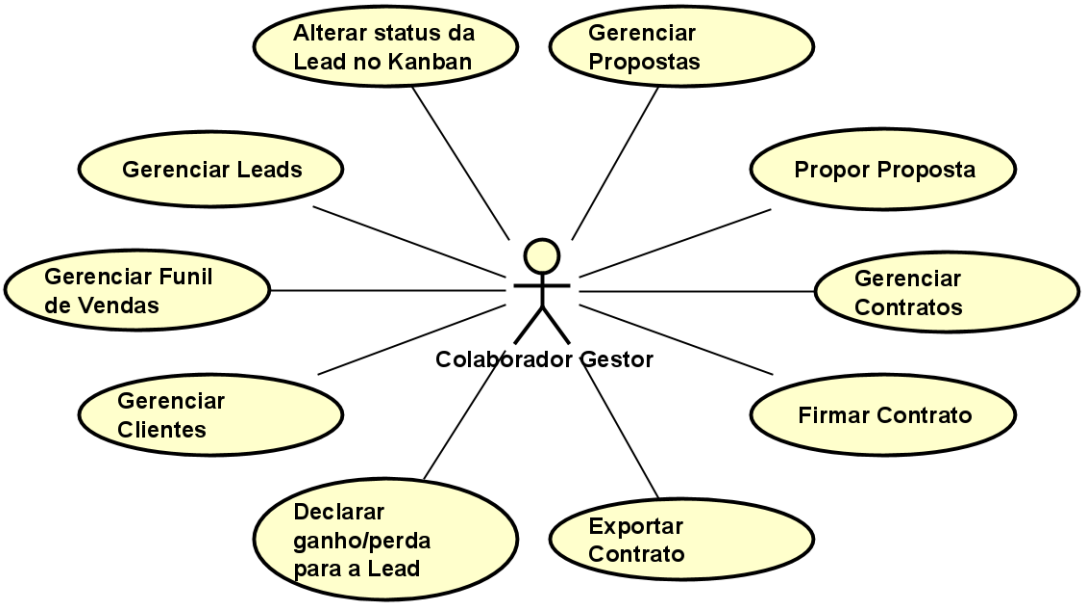


Figura 4 – Diagrama de Casos de Uso do Subsistema SalesManagement.

Os casos de uso **Gerenciar Funil de Vendas**, **Gerenciar Leads**, **Gerenciar Propostas**, **Gerenciar Contratos** e **Gerenciar Clientes** são do tipo cadastrais. Para cada caso de uso cadastral, espera-se que o Gestor possa criar, alterar, consultar e excluir

a classe em questão, passando as informações relativas a ela no momento de sua criação ou alteração.

O caso de uso **Alterar status da Lead no Kanban** permite ao Colaborador Gestor mover o *card* que representa a *lead*, no *Kanban* de vendas, de forma que represente um progresso ou regresso de negociação com a *lead*.

O caso de uso **Propor Proposta** permite ao Colaborador Gestor gerir as propostas dentro do funil de vendas, ao avançar um *card* de uma *lead* até o *status* de tipo proposta, no Kanban, o sistema exigirá a criação da proposta para seguimento do fluxo e progresso do *card*.

O caso de uso **Firmar Contrato** permite ao Colaborador Gestor gerir os contratos dentro do funil de vendas, ao avançar um *card* de uma *lead* até o *status* de tipo contrato, no Kanban, o sistema exigirá a criação do contrato para seguimento do fluxo e progresso do *card*.

O caso de uso **Exportar Contrato** permite ao Colaborador Gestor, após gerar o contrato da *lead*, exportar o contrato gerado em um arquivo formato PDF.

O caso de uso **Declarar ganho/perda para a Lead** permite ao Colaborador Gestor, concluir a negociação com a *lead*. A declaração de ganho só estará permitida caso a *lead* esteja no status de tipo contrato, com contrato gerado, assim o sistema abordará a ação solicitando ao Gestor a inserção dos dados de endereço e outros dados complementares para transformar a *lead* em um cliente. A declaração da perda pode ser feita em qualquer instante depois da criação da *lead*.

3.1.5.2 Subsistema ProjectManagement

A Figura 5 apresenta o diagrama de casos de uso do subsistema ProjectManagement.

Os casos de uso **Gerenciar Funil de Projetos**, **Gerenciar Sprints**, **Gerenciar Setores**, **Gerenciar Projetos** e **Gerenciar Clientes** são do tipo cadastrais. Para cada caso de uso cadastral, espera-se que o Gestor possa criar, alterar, consultar e excluir a classe em questão, passando as informações relativas a ela no momento de sua criação ou alteração.

O caso de uso **Gerenciar Tarefas** também é do tipo cadastral. Espera-se que o Colaborador Gestor e o Colaborador Comum possam criar, alterar, consultar e excluir a classe em questão, passando as informações relativas a ela no momento de sua criação ou alteração.

O caso de uso **Alterar Dados Cadastrais** também é do tipo cadastral. Espera-se que o Colaborador Gestor, Colaborador Comum e o Cliente possam alterar a classe em questão, passando as informações relativas a ela no momento de sua alteração.

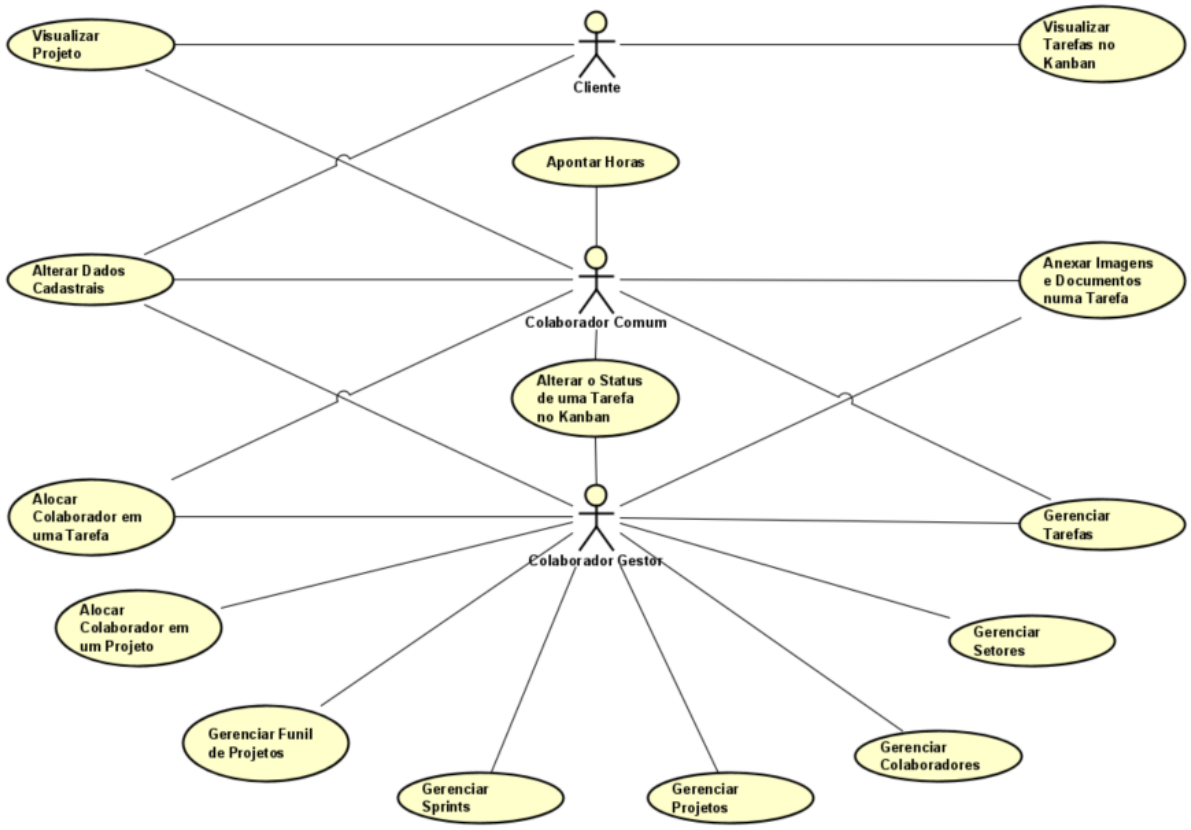


Figura 5 – Diagrama de Casos de Uso do Subsistema ProjectManagement.

O caso de uso **Alterar status da Tarefa no Kanban** permite ao Colaborador Gestor e o Colaborador Comum mover o *card* que representa a tarefa, no *Kanban* de projetos, de forma que represente um progresso ou regresso de trabalho da atividade.

O caso de uso **Visualizar Projeto** permite ao Cliente visualizar um projeto na página de projetos. Espera-se ver de um projeto: nome do projeto, cliente, status, esforço e equipe.

O caso de uso **Visualizar Tarefas no *Kanban*** permite ao Cliente visualizar o quadro na página do *Kanban* com base na busca do seus *slots* com nome e status, com suas devidas tarefas onde se espera ver o nome, responsável, progresso e alocadas em seus status, de suas respectivas *sprints* do respectivo projeto.

O caso de uso **Gerenciar Colaboradores** permite ao colaborador Gestor gerenciar colaboradores na página de colaboradores. A criação de um colaborador passa por algumas etapas: o Gestor acessa a página de colaboradores e solicita criar um colaborador, informando e-mail, nome, estado civil, sexo, RG, CPF, PIS-PASEP, escolaridade, número de dependentes, endereço e dados bancários. Então o sistema gera o usuário de acesso, de acordo com os dados informados e senha automática e envia por e-mail os dados de acesso gerados para o primeiro acesso do Colaborador Comum ao sistema. O Colaborador Comum pode então acessar o sistema. No seu primeiro acesso, o sistema recomendará por

segurança a troca de sua senha.

O caso de uso **Alocar Colaborador em um Projeto** permite ao Colaborador Gestor, alocar um colaborador comum a um projeto, de forma que represente que colaborador comum agora esteja também trabalhando naquele projeto e permitindo ele gerir tarefas dentre outras atividades, dentro do quadro *Kanban* daquele específico projeto.

O caso de uso **Alocar Colaborador em uma Tarefa** permite a um colaborador alocar outro colaborador a uma tarefa, isto é, fazer com que o outro colaborador selecionado seja responsável por aquela específica tarefa.

O caso de uso **Anexar Imagens e Documentos numa tarefa** permite a um colaborador anexar arquivos em uma tarefa, complementado alguma informação se assim julgar necessário.

O caso de uso **Apontar horas** permite a um Colaborador Comum clicar em "*play/stop*" na tarefa do quadro *Kanban* de projeto. O sistema calcula e registra as horas trabalhadas daquele colaborador no respectivo projeto, com base no intervalo de tempo entre os dois cliques ("*play/stop*").

3.1.5.3 Subsistema UserManagement

A Figura 6 apresenta o diagrama de casos de uso do subsistema UserManagement.

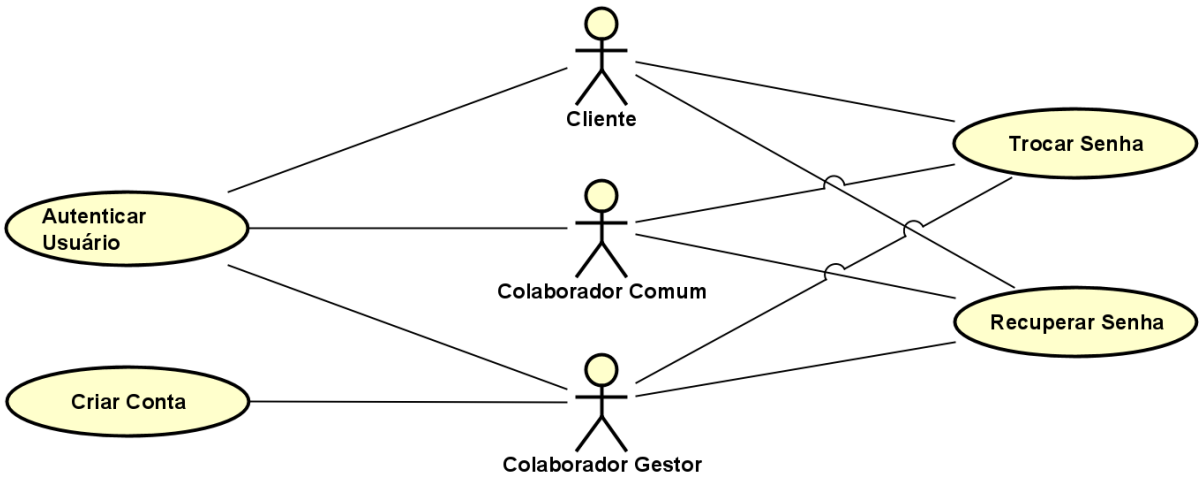


Figura 6 – Diagrama de Casos de Uso do Subsistema UserManagement.

O caso de uso **Criar Conta** permite ao visitante, futuro ator Gestor, criar sua conta e de sua instituição para começar a utilizar o *GanttBox*. Primeiramente, deverá inserir os dados básicos da sua empresa: nome, CNPJ, telefone comercial e e-mail. Em seguida, informar os seus dados básicos de cadastro de colaborador: nome, e-mail, RG, CPF e celular. Logo após, o sistema gera o código da empresa e cria um novo esquema no banco de dados com múltiplos conjuntos de tabelas, para segregar os dados de outras instituições. Em seguida, o sistema gera o usuário de acesso, de acordo com os dados

informados e senha automática, e então envia por e-mail os dados de acesso gerados para o primeiro acesso do Gestor ao sistema. O Colaborador Gestor pode então acessar o sistema. No seu primeiro acesso, o sistema recomendará por segurança a troca de sua senha.

O caso de uso **Autenticar Usuário** permite ao atores, efetuar login no sistema ao inserir suas credenciais de acesso: nome de usuário e senha.

O caso de uso **Recuperar Senha** permite ao atores, recuperarem a senha ao inserirem seu e-mail de cadastro no modal de recuperação de senha, receberem o e-mail com o link para redefinição de senha e, por fim, inserirem a nova senha e a confirmação da nova senha de acesso ao sistema.

O caso de uso **Trocar Senha** permite aos atores, após terem efetuado login no sistema, trocar a senha, na página de troca de senha, inserindo a senha atual, a nova senha e a confirmação da nova senha de acesso.

3.1.6 Modelo Estrutural

O modelo conceitual estrutural visa capturar e descrever as informações (classes, associações e atributos) que o sistema deve representar para prover as funcionalidades descritas nos casos de uso, especificados na Subseção 3.1.5.

A seguir, são apresentados os diagramas de classes de cada um dos subsistemas identificados no contexto deste projeto.

3.1.6.1 Subsistema SalesManagement

A Figura 7 apresenta o diagrama de classes do Subsistema SalesManagement.

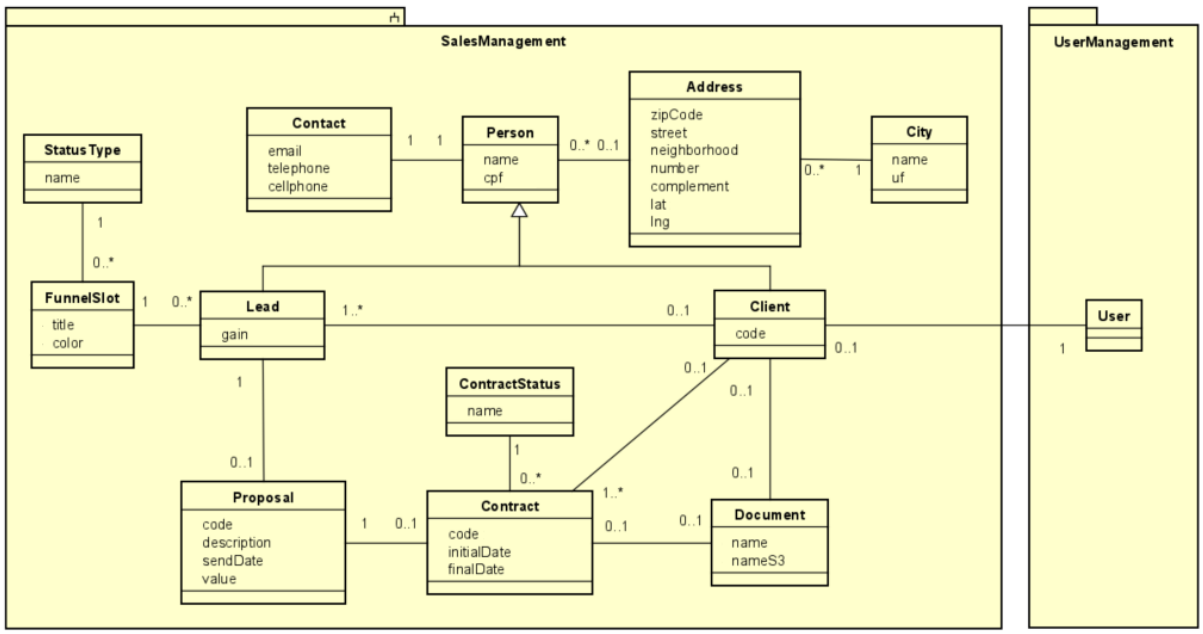


Figura 7 – Diagrama de classes do Subsistema SalesManagement.

A classe **Person**, representa uma pessoa física, que por sua vez possui um contato, representado pela classe **Contact** e um endereço representado pela classe **Address**. Todo endereço possui uma cidade, classe **City**.

Uma *lead* é uma pessoa física representada pela classe **Lead**. *Leads* estão alocados em um *slot* do funil de vendas representado pela classe **FunnelSlot**, que por sua vez possui um status representado pela classe **StatusType**.

A *lead* pode eventualmente receber uma proposta, representada pela classe **Proposal** que por sua vez pode originar um contrato representado pela classe **Contract**, contratos possuem um status, representado pela classe **ContractStatus**. A *lead*, quando sela um contrato, dá origem a um cliente da empresa, representado pela classe **Client**. Clientes também são pessoas físicas.

Por último a classe **Document** representa todo e qualquer arquivo que será armazenado na nuvem, como um documento anexado a uma tarefa ou uma possível foto de um colaborador ou cliente.

3.1.6.2 Subsistema ProjectManagement

A Figura 8 apresenta o diagrama de classes do Subsistema ProjectManagement.

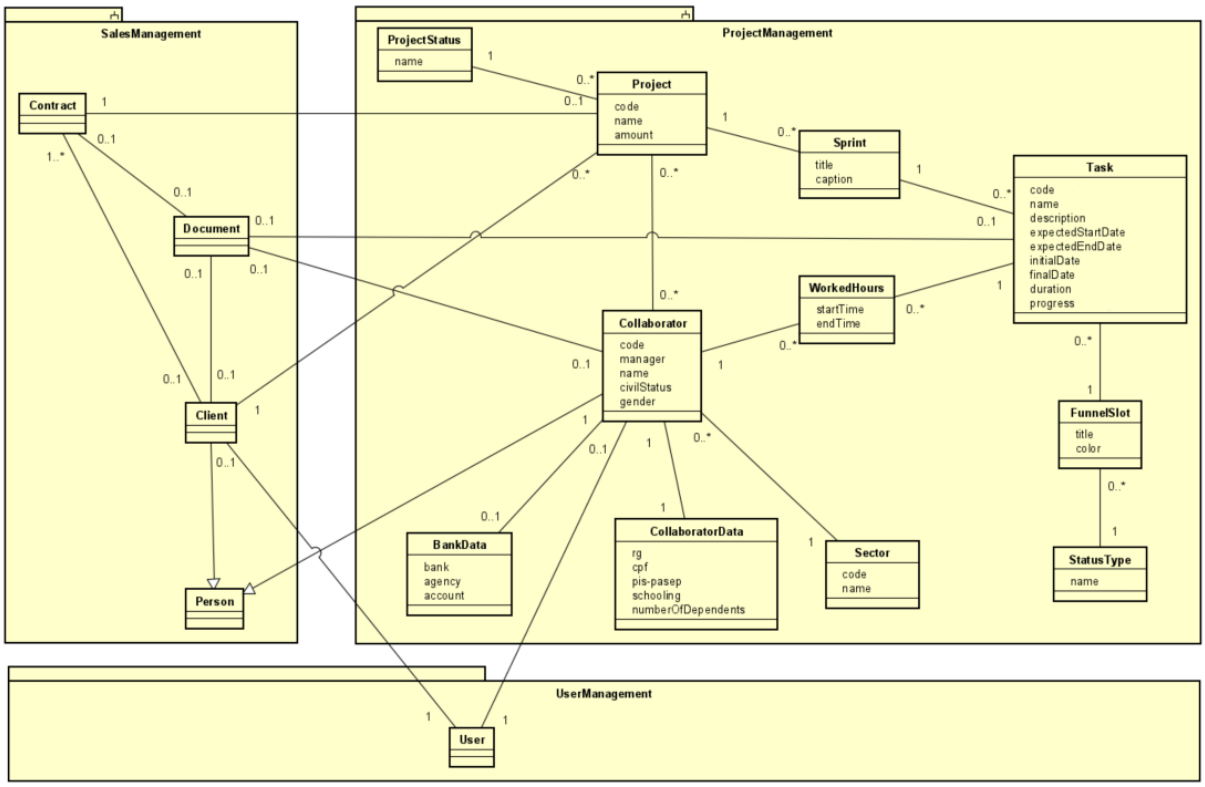


Figura 8 – Diagrama de classes do Subsistema ProjectManagement.

A classe **Collaborator**, representa todo e qualquer colaborador/funcionário da instituição, colaboradores possuem dados bancários representados pela classe **BankData**,

são organizados por setores representado pela classe **Sector** e possuem outras importantes informações pessoais representadas pela classe **CollaboratorData**.

Colaboradores são alocados em um ou mais projetos. Os projetos representam o empreendimento a ser realizado pela empresa para desenvolver a solução proposta ao cliente no contrato. Os projetos são representados pela classe **Project**, que por sua vez possuem **Sprints**, e cada *Sprint* possui diversas tarefas representadas pela classe **Task**.

Tarefas estão alocadas em um *slot* do funil de projetos representado pela classe **FunnelSlot**, que por sua vez possui um status representado pela classe **StatusType**. Por fim a classe **WorkedHours**, representando o tempo de trabalho de um colaborador na tarefa.

3.1.6.3 Subsistema UserManagement

A Figura 9 apresenta o diagrama de classes do Subsistema UserManagement.

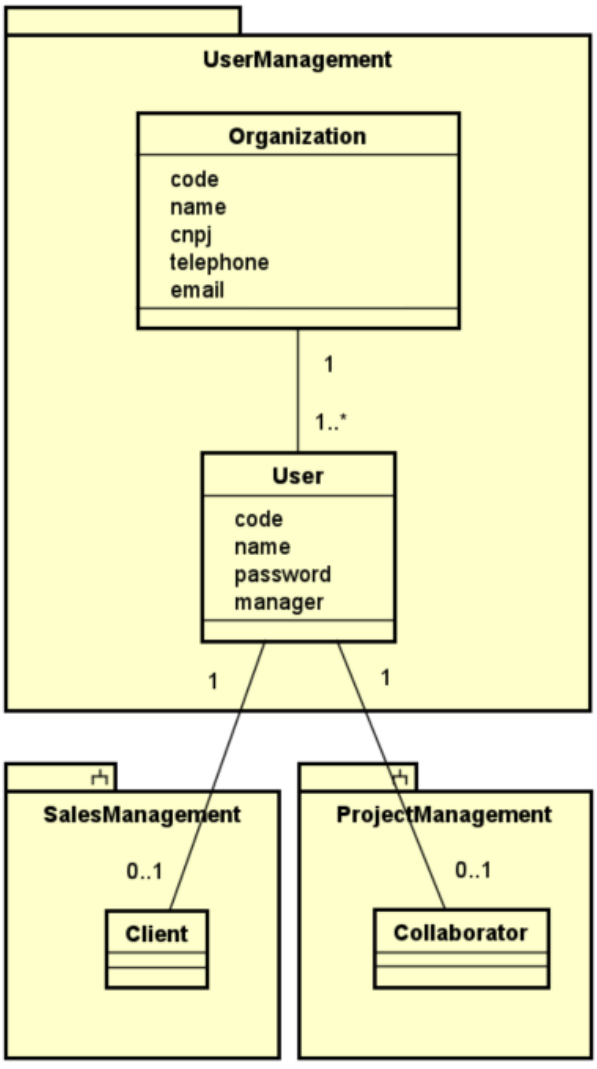


Figura 9 – Diagrama de classes do Subsistema UserManagement.

A classe **User**, representa todo indivíduo que possui acesso para utilização do sistema. Todo usuário está associado a uma empresa, empresas estão representadas pela classe **Organization**.

3.2 Projeto de Sistema

Conforme redigido na Subseção 2.1.2, após a fase de análise de requisitos, a fase de projeto produzirá um modelo de solução para o problema identificado, levando em conta aspectos tecnológicos, a arquitetura do software e componentes estruturais de relacionamentos e armazenamento de dados.

3.2.1 Tecnologias Utilizadas

Foram utilizadas diversas tecnologias para o desenvolvimento da ferramenta de gestão *GanttBox*. No lado do servidor, o *back-end* foi construído utilizando a tecnologia Node.js, que caracteriza-se principalmente, por ser um ambiente de execução *single-threaded*, rodando Javascript no lado do servidor. Com alta capacidade de escalabilidade e boa flexibilidade para o desenvolvedor integrar novos *frameworks* e ferramentas que julgue necessário. O Node.js não possui suporte nativo para tratamento e especificação de rotas e métodos HTTP. Portanto, para esta tarefa foi utilizado o *framework* Express.js, definindo como será a resposta para cada requisição feita pelos usuários para uma URL específica utilizando um dos métodos do protocolo HTTP.

Para o armazenamento dos dados foi utilizado o banco relacional PostgreSQL, e para comunicação entre o servidor e o banco foi utilizado o ORM (*Object/Relational Mapping*) TypeORM, responsável justamente por fazer o mapeamento entre as entidades objetos com as tabelas do banco de dados, facilitando e ainda otimizando *queries* implementadas no lado do servidor.

Para construção do *front-end*, páginas no lado do cliente, foi utilizada a plataforma Angular, *framework* Javascript para construção de *single-page applications*. Para otimizar a produtividade e construir páginas com *design* amigáveis para o usuário, utilizou-se o *framework* Bootstrap, fornecendo classes e componentes responsivos. Utilizou-se ainda o *framework* Syncfusion, fornecendo outras bibliotecas que auxiliaram principalmente nos quadros *Kanban* do sistema e também na interatividade das páginas.

3.2.2 Modelos FrameWeb

Nesta subseção será apresentada uma perspectiva da arquitetura FrameWeb, de forma a expor suas camadas e respectivos modelos pontuados na Seção 2.4 e fazendo um paralelo entre as outras arquiteturas de softwares que amparam o projeto desse texto,

apresentadas na Seção 2.3.

3.2.2.1 Camada de Negócio

A camada central da arquitetura, a camada de negócio, contém o pacote de domínio, onde se encontram todas as classes, entidades, que representam o domínio do problema. As entidades, adaptadas as plataformas de implementação, estão representadas nas figuras 10, 11 e 12, que são os Modelos de Entidades do FrameWeb para este projeto.

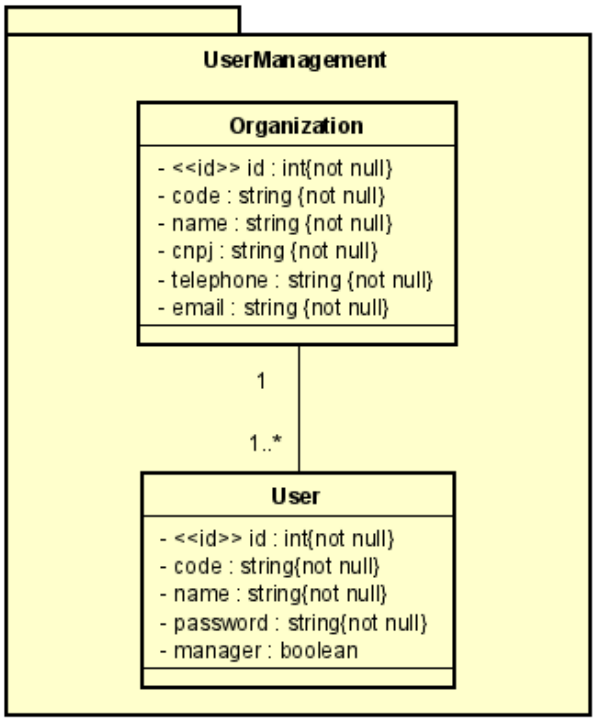


Figura 10 – Modelo de Entidades do Subsistema UserManagement.

Também está presente nesta mesma camada o pacote de aplicação. Este pacote contém os casos de usos da aplicação, consolidando todas as regras de negócio. Podemos fazer uma analogia com a arquitetura de software que também inspira o sistema GanttBox, relatada na Seção 2.3.1: a *Clean Architecture* também centraliza a lógica do negócio em suas duas camadas mais internas e centrais e que também não conhecem as camadas mais externas da arquitetura.

De forma a atingir o exercício da modelagem FrameWeb e também simplificar o trabalho e a quantidade de diagramas representados para entrega desta monografia dentro do prazo, optou-se por escolher módulos e subsistemas específicos para a representação de alguns modelos.

As figuras 13 e 14 representam os Modelos de Aplicação do Subsistema *UserManagement*.

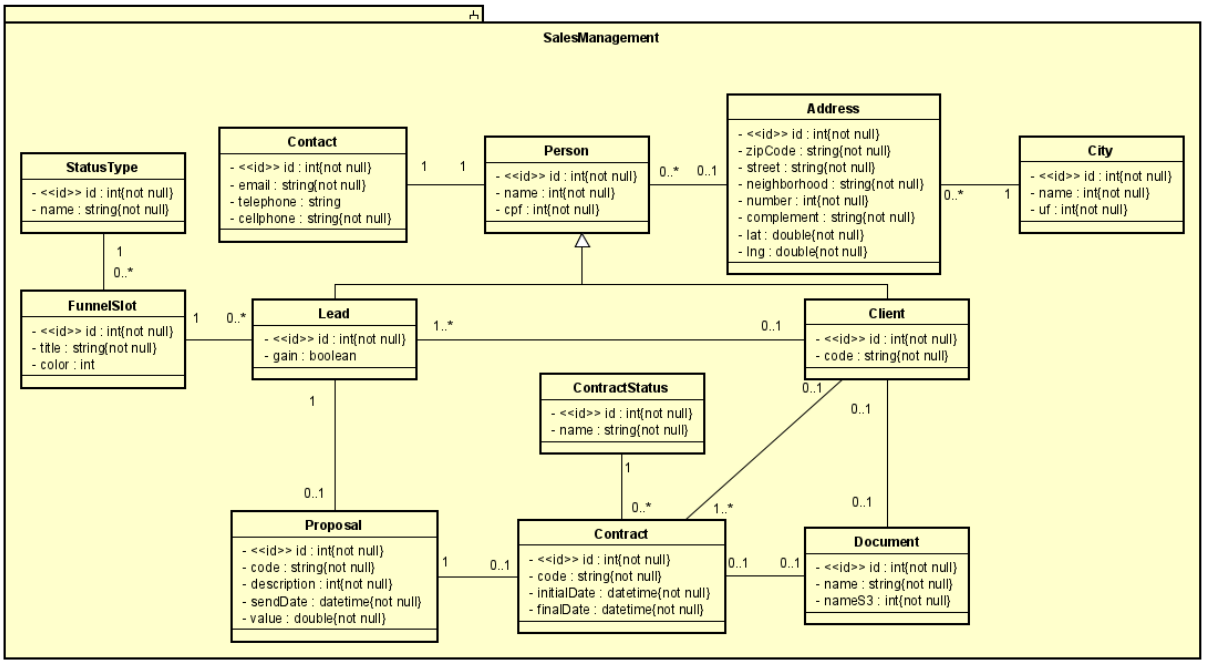


Figura 11 – Modelo de Entidades do Subsistema SalesManagement.

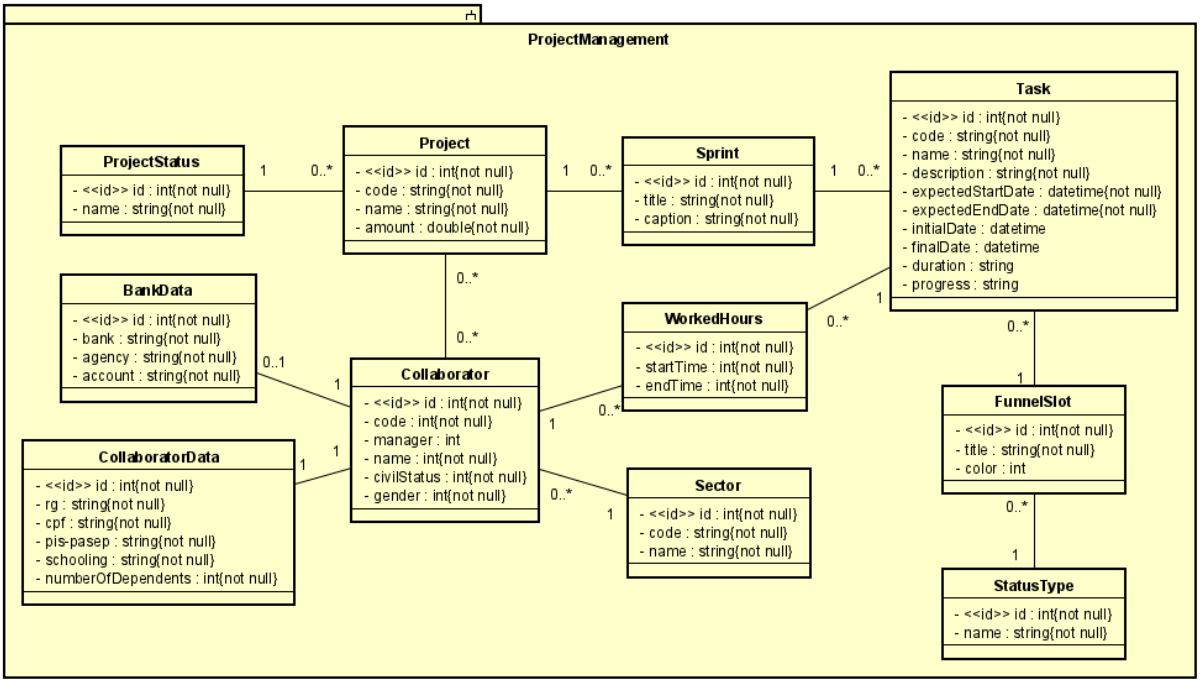


Figura 12 – Modelo de Entidades do Subsistema ProjectManagement.

3.2.2.2 Camada de Acesso a Dados

A camada de acesso a dados contém o pacote de persistência, responsável pelo armazenamento das informações em bancos de dados ou outras mídias de longa duração. No sistema de gestão de projetos utiliza-se DTOs (*Data Transfer Objects*) que são simples classes criadas quando necessárias e que não contém nenhuma regra de negócio, mas são de suma importância para futuras manutenções de código. Além disso, contribuem para

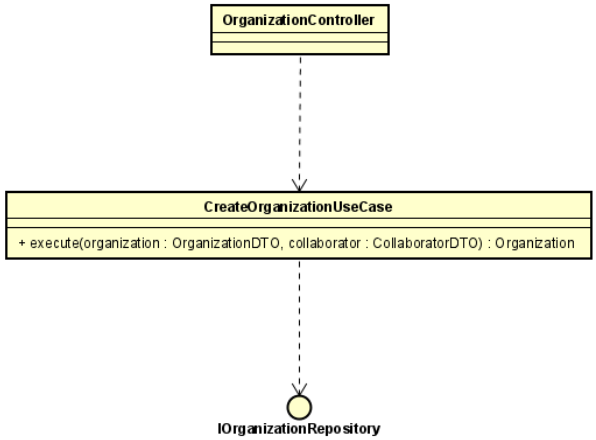


Figura 13 – Modelo de Aplicação da Entidade Organization do Subsistema UserManagement.

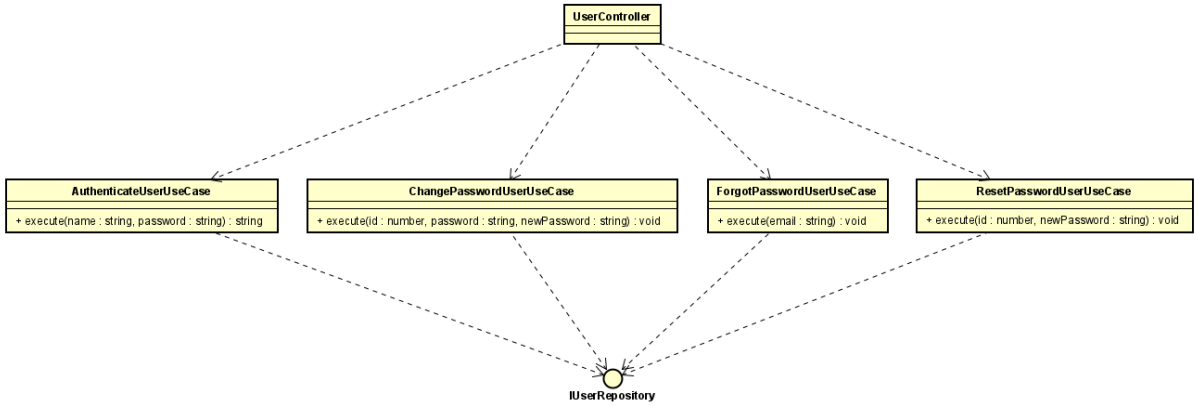


Figura 14 – Modelo de Aplicação da Entidade User do Subsistema UserManagement.

uma comunicação eficiente entre as camadas da arquitetura e aumentam a segurança dos dados de forma geral.

A Figura 15 ilustra o modelo de persistência no subsistema SalesManagement.

3.2.2.3 Camada de Apresentação

Por fim, a camada de apresentação contém o pacote visão, que propõe prover a interface gráfica com o usuário, assim como parte da camada mais externa da arquitetura descrita na Subseção 2.3.1.

Para ilustrar o modelo de navegação dentro das adaptações dessa ferramenta, escolheu-se trabalhar aqui com a página de projetos do subsistema ProjectManagement. A Figura 16 representa este modelo que se espelha para todo CRUD (*Create, Read, Update, Delete*) do sistema, salve devidas adaptações conforme a entidade.

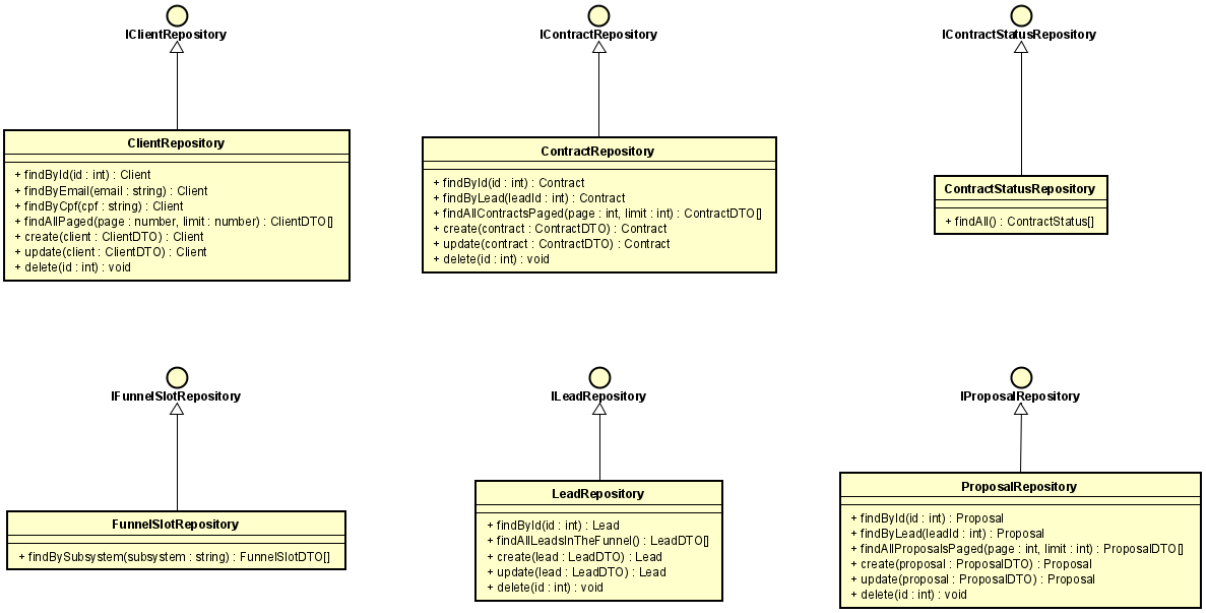


Figura 15 – Modelo de Persistência do Subsistema SalesManagement.

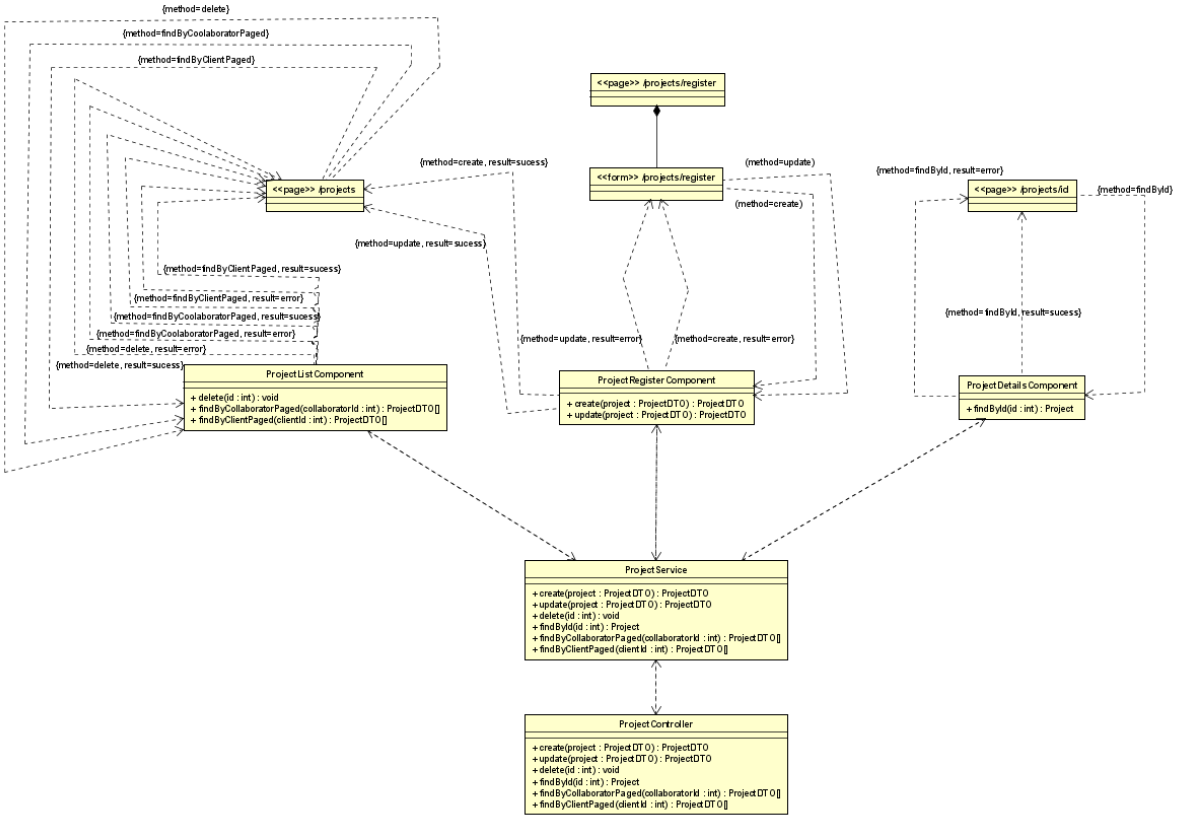


Figura 16 – Modelo de Navegação da Entidade Project do Subsistema ProjectManagement.

4 Implementação e Apresentação

Neste capítulo serão apresentados aspectos da implementação do sistema, assim como os resultados alcançados por meio de capturas de tela. As seções 4.1 e 4.2 abordam aspectos de estrutura e organização do código fonte do *back-end* e do *front-end*, respectivamente, do sistema desenvolvido. A Seção 4.3 apresenta os resultados obtidos através da exposição de funcionalidades e telas.

4.1 Implementação do Back-end

Conforme comentado nesse texto nas subseções 2.3.1, 2.4, 3.2.2 e no Documento de Projeto de Sistema, anexo desta monografia, o sistema *GanttBox* se baseia principalmente nas arquiteturas de softwares *Clean Architecture* (MARTIN; GRENNING; BROWN, 2018) e na arquitetura proposta pelo FrameWeb (SOUZA, 2020). Assim, para organização do *Back-end*, pretendeu-se projetar uma estrutura de diretórios de maneira a se tornar mais próximo possível ao que se é modelado nestes métodos. A Figura 17 apresenta a estrutura de diretórios do *Back-end*.

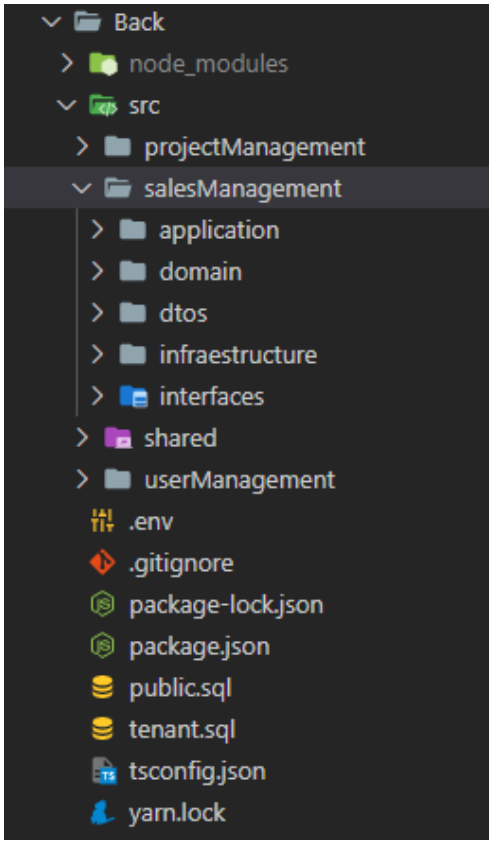


Figura 17 – Estrutura dos Diretórios no Back-End

No diretório **src** econtra-se o código fonte do *Back-end*. Dentro dele, existe um outro diretório chamado **shared**, pasta na qual estão presentes todas as classes, *templates* e interfaces que serão utilizados em vários pontos do código de forma a se componentizar, modularizar e reutilizar código. Em outras palavras, é um diretório de compartilhamento de arquivos e funcionalidades que solucionam reincidentes pequenos problemas, por exemplo, classes de tratamento de erros, funções validadoras e *templates* HTML para envios de e-mail. Na Figura 18 podemos notar parte do código da classe **Validators**, presente em **src/shared/util**.

```
6   class Validators {
7
8   >   public verifyEmail(email: string) {···
21      }
22
23  >   public verifyPassword(password: string) {···
43      }
44
45  >   public verifyDoc(cpf_cnpj: string): ValidatorResponse {···
58      }
59
60      public validateCpf(cpf: string): ValidatorResponse {
61        cpf = cpf.replace(/[\.\-\/]/g, '')
62
63        var firstDigit = 0;
64        for (let i = 0; i < 9; i++) {
65          firstDigit += parseInt(cpf[i]) * (10 - i);
66        }
67
68        var remainder = firstDigit % 11;
69        firstDigit = (remainder < 2) ?
70          0 :
71          11 - remainder;
72
73        if (firstDigit !== parseInt(cpf[9]))
74          return {
75            response: 'CPF inválido',
76            valid: false
```

Figura 18 – Parte do Código da Classe Validators

Conforme pode ser visualizado na figura, a função validadora de CPF será utilizada na criação e edição de uma *lead*, criação e edição de um cliente e criação e edição de um colaborador.

Ainda dentro do diretório **shared**, um importante arquivo que deve ser pontuado é o **routes.ts**, presente em **src/shared/infraestructure/routes**. Conforme pode ser visualizado na Figura 19, este arquivo é a porta de entrada para as requisição HTTP advindas de fora da API (*Application Programming Interface*), utilizando-se as bibliotecas do framework *Express.js*. Logo, para cada requisição que chegar ao *Back-end* com final de URL *'/leads'*, por exemplo, este arquivo direcionará a chamada para o específico arquivo de rotas **leads.routes.ts**, presente em **src/salesManagement/infraestructure/routes** e conforme será comentado mais a frente nesta subseção.

Retomando a observação na Figura 17, podemos ainda notar outros três diretórios inclusos em **src** (**salesManagement**, **projectManagement** e **userManagement**). Estas

```
15   const routes = Router();
16
17   routes.use('/organizations', organizationRoute);
18   routes.use('/users', userRoute);
19
20   routes.use('/funnelslots', funnelSlotRoute);
21   routes.use('/leads', leadRoute);
22   routes.use('/clients', clientRoute);
23   routes.use('/proposals', proposalRoute);
24   routes.use('/contracts', contractRoute);
25   routes.use('/contractsstatus', contractStatusRoute);
```

Figura 19 – Parte do Código de routes.ts

três pastas representam os módulos do sistema ou subsistemas, conforme apresentado na Seção 3.1.4 e no Documento de Especificação de Requisitos. Em cada um desses diretórios, que representam os subsistemas, existe a mesma estrutura de sub-diretórios conforme explicado a seguir:

- **/application**: Este diretório representa o pacote de aplicação dentro da camada de negócios do FrameWeb e a camada de Casos de Uso, na Clean Architecture. Contém todos os casos separados por classes especializadas que conterão todas as regras de negócio do respectivo subsistema.

  Podemos tomar como exemplo a classe **CreateLeadUseCase**, presente em **src/salesManagement/application/Lead**, representada na Figura 20.

```
class CreateLeadUseCase {

    constructor(
        private leadRepository: ILeadRepository,
    ) { }

    public async execute(lead: Lead): Promise<Lead | undefined> {

        lead.person.cpf = lead.person.cpf.replace(/[\.\-\/]/g, '');
        this.verifyCpfValid(lead.person.cpf);
        await this.verifyExistsLeadWithSameCpf(lead.person.cpf);

        this.verifyEmailValid(lead.person.contact.email);
        await this.verifyExistsLeadWithSameEmail(lead.person.contact.email);

        const leadSaved = await this.leadRepository.create(lead);

        return leadSaved;
    }

    private verifyEmailValid(email: string) {
        if (!email) {
            return;
        }
        const validateEmail = validator.verifyEmail(email);
```

Figura 20 – Parte do Classe CreateLeadUseCase

Esta classe é responsável especificamente para a cuidar das regras da criação de uma *Lead* no sistema, isto é, uma classe especializada para a criação da *Lead*. Desta forma, o *Back-end*, tenta colocar em prática um dos princípios postuladas no SOLID, pontualmente neste caso, o SRP (Single-responsibility principle), ou Princípio da

Responsabilidade Única, onde cada classe deve ser especializada em um único assunto e possuir apenas uma responsabilidade dentro da aplicação.

- **/domain**: Este diretório representa o pacote de domínio da camada de negócios do FrameWeb e a camada central de Entidades na Clean Architecture. Contém todas as classes, entidades, do domínio do problema e representam as tabelas do banco de dados.

  Podemos progredir na mesma linha e exemplificar a classe **Lead**, presente em **src/salesManagement/domain/Lead**, representada na Figura 21.

```
@Entity('lead')
class Lead {
  @PrimaryGeneratedColumn()
  id: number;

  @Column('boolean')
  gaint: boolean;

  @Column('int4', { name: "funnel_slot_id" })
  funnelSlotId: number;

  @ManyToOne(type => FunnelSlot, funnelSlot => funnelSlot.leads, { eager: true })
  @JoinColumn({ name: "funnel_slot_id" })
  funnelSlot: FunnelSlot;

  @Column('int4', { name: "person_id" })
  personId: number;

  @OneToOne(type => Person, lead => Lead, { eager: true, cascade: true })
  @JoinColumn({ name: "person_id" })
  person: Person;

```

Figura 21 – Parte da Classe Lead

  Podemos notar os atributos da classe e perceber algumas anotações como *Column*, *ManyToOne*, dentre outras, que serão de suma importância para fazer o mapeamento com as tabelas pelo ORM (Object Relational Mapper) escolhido neste projeto, **TypeORM**.

- **/dtos**: Este diretório como já explicado em algumas partes deste texto, contém os DTOs (*Data Transfer Objects*), que são classes criadas quando necessárias e que contribuem para uma comunicação eficiente entre as camadas da arquitetura e aumentam a segurança dos dados de forma geral.

  Podemos exemplificar a implementação com a classe **FunnelSlotDTO**, presente em **src/salesManagement/dtos**, representada na Figura 22.

  O uso dos DTOs pode trazer vários benefícios práticos. Na segurança, podemos pontuar situações em que não queremos trazer determinadas informações para o *Front-end*, então nesse caso, poderíamos criar uma classe DTO específica, inserindo apenas atributos que queremos levar da resposta da requisição. Estamos implementando o objeto que será utilizado na transferência dos dados, logo, dessa forma, também podemos pontuar casos de melhoria em eficiência, por exemplo, em situações de transferência de grande quantidades de dados, 10 mil, 100 mil ou mais registros, talvez nem todos os atributos da classe entidade são necessários nesta transferência,

```
class FunnelSlotDTO {
    id: string;
    title: string;
    color: string;

    constructor(slotFunnel: FunnelSlot) {
        this.id = slotFunnel.id.toString();
        this.title = slotFunnel.title;
        this.color = slotFunnel.color;
    }
}
```

Figura 22 – Parte da Classe FunnelSlotDTO

logo criamos uma classe DTO, colocando apenas os atributos que serão necessários, fazendo com que a resposta da requisição se torne muito mais leve (tamanho em bytes). Fazendo uma analogia na Figura 22, supondo que em 100 mil registros a informação de cor representada pelo atributo *color*, não fosse necessária, então removeríamos este atributo e o seu mapeamento do construtor da classe de entidade. Dessa forma, economizaríamos 100 mil atributos na transferência dos dados. Podemos ainda imaginar cenários mais caóticos, com muitos atributos, e como o uso dessa abordagem poderia otimizar ainda mais a troca da informação.

- **/infraestructure**: Neste diretório existem ainda mais 2 subdiretórios internos:
  - **/repositories**: Neste diretório estão presente os arquivos de repositórios, classes que fazem as consultas ao banco de dados e suas respectivas interfaces.

    Um exemplo da implementação estão apresentados nas figuras 23 e 24.

```
3   export default interface IClientRepository {
4
5       findById(id: number): Promise<Client | undefined>;
6
7       findByEmail(email: string): Promise<Client | undefined>;
8
9       findByCpf(cpf: string): Promise<Client | undefined>;
```

Figura 23 – Parte da Interface IClientRepository

  - **/routes**: Este diretório contém as rotas específicas de cada *endpoint*, advindo da rota base do sistema que já foi pontuada anteriormente na Figura 19.

    A Figura 25 mostra um exemplo de implementação do arquivo **leads.routes.ts**, presente em **src/salesManagement/infraestructure/routes**, indicando que para cada URL e método HTTP específico será chamado um método do controlador que será comentado no próximo item da seção.

- **/interfaces**: Neste diretório também existem dois subdiretórios internos:

```
class ClientRepository implements IClientRepository {

    private ormRepository: Repository<Client>;

    constructor(private tenant: string) {
        this.ormRepository = getRepository(Client);
    }

    public async findById(id: number): Promise<Client | undefined> {
        await this.ormRepository.query(`SET SCHEMA '${this.tenant}'`);

        const client = this.ormRepository.findOne({ where: { id }});
        return client;
    }

    public async findByEmail(email: string): Promise<Client | undefined> {
        await this.ormRepository.query(`SET SCHEMA '${this.tenant}'`);

        const client = await this.ormRepository.createQueryBuilder('client')
            .innerJoinAndSelect('client.person', 'person')
            .innerJoinAndSelect('person.contact', 'contact')
            .where('contact.email = :email', { email })
            .getOne();

        return client;
    }
```

Figura 24 – Parte da Classe ClientRepository

```
export class LeadRoute {

  public leadRouter: Router;
  public leadController: LeadController;

  constructor() {
    this.leadRouter = Router();
    this.leadController = new LeadController();
  }

  public routes() {
    this.leadRouter.get('', this.leadController.findAll);
    this.leadRouter.post('', this.leadController.create);
    this.leadRouter.put('', this.leadController.update);
    this.leadRouter.put('/changestatus', this.leadController.changeStatus);
    this.leadRouter.put('/declarewinloss', this.leadController.declareWinLoss);
    this.leadRouter.delete('/:id', this.leadController.delete);
```

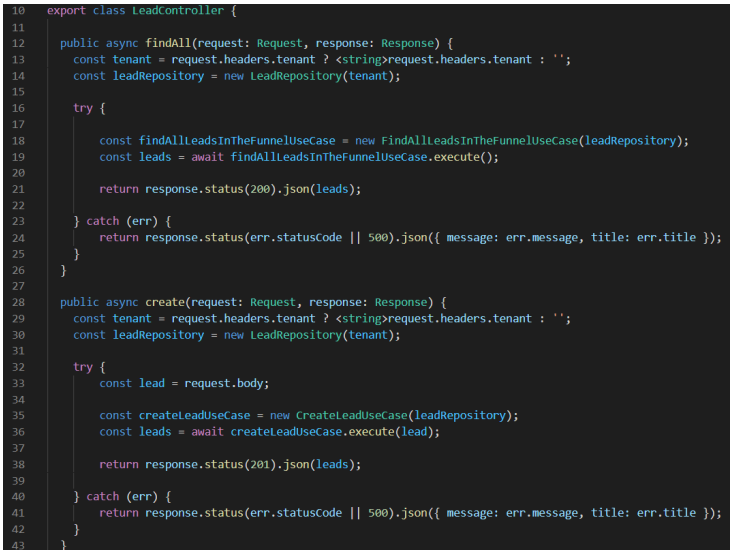
Figura 25 – Parte do Código de lead.routes.ts

– **/controllers**: Neste diretório estão presentes os arquivos controladores do **Back-end**. Estas classes serão responsáveis por recepcionar as requisições passadas pelos arquivos de rota, recepcionar os respectivos dados que possam vir no cabeçalho, corpo, ou outra forma embutida no métopo HTTP e finalmente chamar o correto caso de uso, delegando o papel dentro da arquitetura.

  Um exemplo da implementação dos controladores esta na classe **LeadController**, presente em **src/salesManagement/interfaces/controllers**, representada na Figura 26.

– **/middllewares**: Finalmente, o último repositório que deve ser pontuado dentro da estrutura montada são os *middllewares*, neles estão presentes os arquivos de *middllewares* que funcionam como úteis interfaces entre as rotas e os métodos dos controladores se houver a necessidade de alguma validação, adição ou remoção de informação das requisições advindas dos arquivos de rotas.

  Um bom exemplo para se pontuar aqui é um *middlleware* que, na realidade, esta presente em **src/shared/interfaces/middllewares** mas atende como



```
export class LeadController {

  public async findAll(request: Request, response: Response) {
    const tenant = request.headers.tenant ? <string>request.headers.tenant : '';
    const leadRepository = new LeadRepository(tenant);

    try {

        const findAllLeadsInTheFunnelUseCase = new FindAllLeadsInTheFunnelUseCase(leadRepository);
        const leads = await findAllLeadsInTheFunnelUseCase.execute();

        return response.status(200).json(leads);

    } catch (err) {
        return response.status(err.statusCode || 500).json({ message: err.message, title: err.title });
    }
  }

  public async create(request: Request, response: Response) {
    const tenant = request.headers.tenant ? <string>request.headers.tenant : '';
    const leadRepository = new LeadRepository(tenant);

    try {
        const lead = request.body;

        const createLeadUseCase = new CreateLeadUseCase(leadRepository);
        const leads = await createLeadUseCase.execute(lead);

        return response.status(201).json(leads);

    } catch (err) {
        return response.status(err.statusCode || 500).json({ message: err.message, title: err.title });
    }
  }
```

Figura 26 – Parte da Classe LeadController

exercício de demonstração. O *middlleware* pode ser visualizado na Figura 27, fazendo a validação do *token* e validando se usuário está logado no sistema ou não, isto é, se ele possui permissão para acessar o recurso requerido.

```
export function ensureAuthenticated(request: Request, response: Response, next: NextFunction){
    const token = request.headers.token;

    if(!token) {
        return response.status(401).json({message: "O usuário não esta autenticado."});
    }
```

Figura 27 – Parte do Código de lead.routes.ts

Outros arquivos relevantes que merecem ser pontuados no *Back-end*:

- **/src/shared/infraestructure/server.ts**: Arquivo responsável pela inicialização do servidor e inclusão dos demais arquivos e pacotes que serão utilizados;
- **/src/.env**: Arquivo que armazena informações das variáveis de ambiente e configurações para comunicação com o banco de dados;
- **package.json**: Arquivo JSON (*JavaScript Object Notation*), que exibe algumas informações do projeto, bem como formas de executá-lo e também lista as dependências necessária para execução do mesmo;
- **tsconfig.json**: Arquivo JSON (*JavaScript Object Notation*), que especifica os arquivos e configurações de compilação necessárias para o projeto.

4.2 Implementação do Front-end

De forma análoga ao *Back-end*, tentou-se projetar no *Front-end*, uma estrutura robusta, pensada para escalar o software e seguindo os princípios das boas práticas de

programação orientada a objetos. Podemos visualizar a estrutura de diretórios do *Front-end* através da Figura 28.

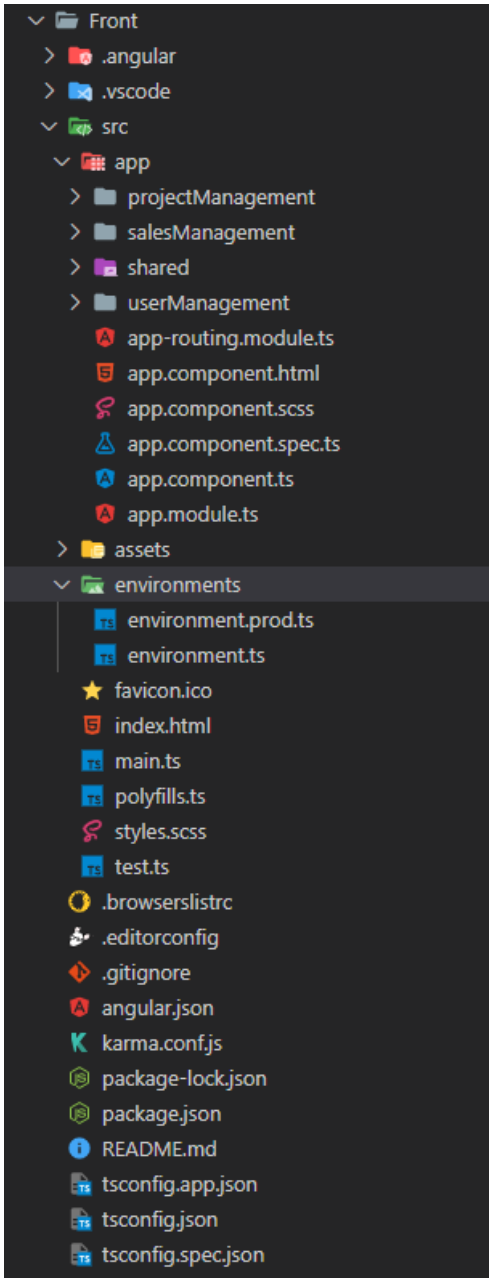


Figura 28 – Estrutura de Diretórios do Front-End

A figura mostra a estrutura padrão de um projeto no *framework* Angular, escolhido para o desenvolvimento das páginas do lado do cliente. Dentro do diretório **src**, existem mais três sub-diretórios. O diretório **assets**, onde estão localizadas todas as imagens utilizadas para a montagem das telas. Além disso, esta presente também o diretório **environments**, onde devem estar os arquivos de variáveis de ambiente do *front*, para exemplificar podemos visualizar a Figura 29.

A Figura 29 mostra as variáveis de ambiente de desenvolvimento do arquivo **environments.ts**, presente em **src/environments/environments.ts**. Nesse caso a variável

```
5  export const environment = {
6    production: false,
7    URL_GB_BACK: 'http://localhost:3000'
8  };
```

Figura 29 – Variáveis de ambiente de desenvolvimento no Front-end

URL_GB_BACK está configurada para apontar para o *Back-end* rodando localmente na porta 3000 em ambiente de desenvolvimento. Outros arquivos de configurações podem ser criados além de **environments.ts** e **environments.prod.ts** presentes neste projeto, conforme necessidade de homologação ou outros ambientes de teste.

Por último e mais importante, um diretório abaixo do **src** está o diretório **app**, nele de fato estão presentes todos os arquivos do *front* codificados no desenvolvimento da ferramenta. Dentro deste diretório **app**, esta presente a pasta **shared** que terá a mesma ideia do diretório de também mesmo nome no *back*, a saber: compartilhamento de recursos, funcionalidades, componentes e serviços que serão utilizadas por todos os outros módulos. Além do diretório **shared**, estão também presentes as pastas **salesManagement**, **projectManagement** e **userManagement**, representando e dividindo componentes, serviços e outros arquivos por subsistema. A seguir, será explicitado a mesma estrutura presente nesses três diretórios, que pode ser visualizada na Figura 30 e conforme feito da forma similar para o *back*.

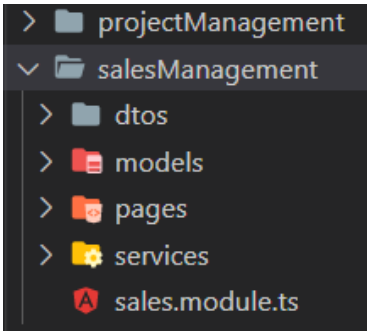


Figura 30 – Estutura de Diretórios dos Subsistemas no Front-end

- **/dtos**: Este diretório conterá as mesmas classes do diretório de também mesmo nome no *Back-end*. De forma, que haja um espelhamento dos atributos das classes e a comunicação entre cliente e servidor ocorra corretamente.
- **/models**: Também conterá espelhamento das classes de entidades do *Back-end*, que em grande parte dos casos também se faz útil na comunicação.
- **/pages**: Este diretório conterá todos os componentes que formarão as páginas do sistema. Um componente no *framework* Angular é composto nesse projeto e na grande maioria de outras aplicações do mercado por um arquivo HTML, sendo o

template do componente, um arquivo Typescript, contento a lógica de programação por trás da página, um arquivo CSS, dando estilização e um arquivo de testes.

Para exemplificação de um componente podemos visualizar as figuras 31 e 32 ilustrando respectivamente os arquivos HTML e Typescript do componente modal de criação de uma Lead.

```
<div class="modal-header">
  <h4 class="modal-title"><span *ngIf='isUpdate'>Editar</span><span *ngIf='!isUpdate'>Cadastrar</span> Lead</h4>
  <button type="button" class="close" aria-label="Close" (click)="activeModal.close(undefined)">
    <span aria-hidden="true">&times;</span>
  </button>
</div>
<div class="modal-body">

  <form [formGroup]="leadForm" class="form mt-3 mb-3">

    <div class="row">

      <div class="col-md-8">
        <div class="form-group mt-3">
          <input formControlName="name" class="form-control" placeholder="Nome" [class.is-invalid]="inputError(leadF
          <span *ngIf="inputError(leadForm, 'name', 'required')" class="text-danger">Campo obrigatório!</span>
        </div>
      </div>

      <div class="col-md-4">
        <div class="form-group mt-3">
          <input formControlName="cpf" class="form-control" placeholder="CPF" mask="000.000.000-00" [class.is-invali
          <span *ngIf="inputError(leadForm, 'cpf', 'required')" class="text-danger">Campo obrigatório!</span>
          <span *ngIf="inputError(leadForm, 'cpf', 'cpfNotValid')" class="text-danger">CPF inválido!</span>
        </div>
```

Figura 31 – Arquivo HTML do Componente LeadFormModal

```
export class LeadFormModalComponent implements OnInit {

  leadForm!: FormGroup;
  lead = new CreateLeadDTO();
  isUpdate = false;

  constructor(
    public activeModal: NgbActiveModal,
    private formBuilder: FormBuilder,
    private leadService: LeadService,
    private toastService: ToastService,
    private modalService: NgbModal,
    ) { }

  ngOnInit(): void {
    this.buildForm();
    this.checkUpdated();
  }

  buildForm(){
    this.leadForm = this.formBuilder.group({
      name: [this.lead?.person?.name, Validators.required ],
      cpf: [this.lead?.person?.cpf, Validators.required ],
      email: [this.lead?.person?.contact?.email, [Validators.required, Validators.email] ],
      cellphone: [this.lead?.person?.contact?.cellphone, Validators.required ],
    }, {
      validators: [ValidatesFormGroup.validatorCpf]
    });
  }

  checkUpdated() {
```

Figura 32 – Arquivo Typescript do Componente LeadFormModal

- **/services**: Este diretório conterá todos os serviços do respectivo subsistema, organizado por entidades. Os serviços de forma geral são usados no Angular para manter a lógica de negócio, compartilhando informações e funcionalidades entre os componentes. O seu maior uso acontece para interação do *front* com o *back* através de chamadas REST pelo protocolo HTTP.

  Podemos exemplificar a implementação dos serviços neste projeto através da Figura 33.

  Podemos retomar atenção novamente na Figura 32 e observar a injeção de dependência deste serviço no construtor.

```
export class LeadService {

  private urlBase = `${environment.URL_GB_BACK}/leads`;

  constructor(
    private http: HttpClient,
  ) { }

  public getLeads() : Promise<LeadsInFunnelDTO[] | undefined> {
    return this.http.get<LeadsInFunnelDTO[]>(`${this.urlBase}/`).toPromise();
  }

  public createLead(lead: CreateLeadDTO) : Promise<Lead | undefined> {
    return this.http.post<Lead>(`${this.urlBase}/`, lead).toPromise();
  }
```

Figura 33 – Parte do Código da Classe LeadService

Outros arquivos relevantes que merecem ser pontuados no *Front-end*:

- **/src/app/shared/services/interceptor.service.ts**: Este arquivo será um interceptador de requisições que chegam e saem do *Front-end*, podendo manipulá-las de forma que adicione ou remova alguma informação, ou ainda execute alguma ação;
- **/src/app/app.component.html**: Este arquivo é o componente pai de todos os outros componentes, ele chamará cada um de acordo com as solicitações de rotas;
- **/src/app/app.module.ts**: Este arquivo é o módulo geral do projeto, que importará todos os outros módulos, componentes e serviços para disponibilização e utilização dos mesmos;
- **/src/app/app-routing.module.ts**: Este arquivo conterá as configurações de rotas de navegação da aplicação;
- **package.json**: Arquivo JSON (*JavaScript Object Notation*), que exibe algumas informações do projeto, bem como formas de executá-lo e também lista as dependências necessária para execução do mesmo;
- **tsconfig.json**: Arquivo JSON (*JavaScript Object Notation*), que especifica os arquivos e configurações de compilação necessárias para o projeto.

4.3 Apresentação do Sistema

Esta seção mostra os resultados da fase de implementação até onde conseguiu-se desenvolver dentro do prazo da entrega desta monografia. O sistema GanttBox será apresentado por meio de uma série de capturas de telas, divididas pelos respectivos subsistemas de forma a instruir melhor e organizar a exposição das telas e funcionalidades.

4.3.1 UserManagement

Ao acessar o sistema pela primeira vez o visitante se deparará com a tela de Login apresentada na Figura 34.

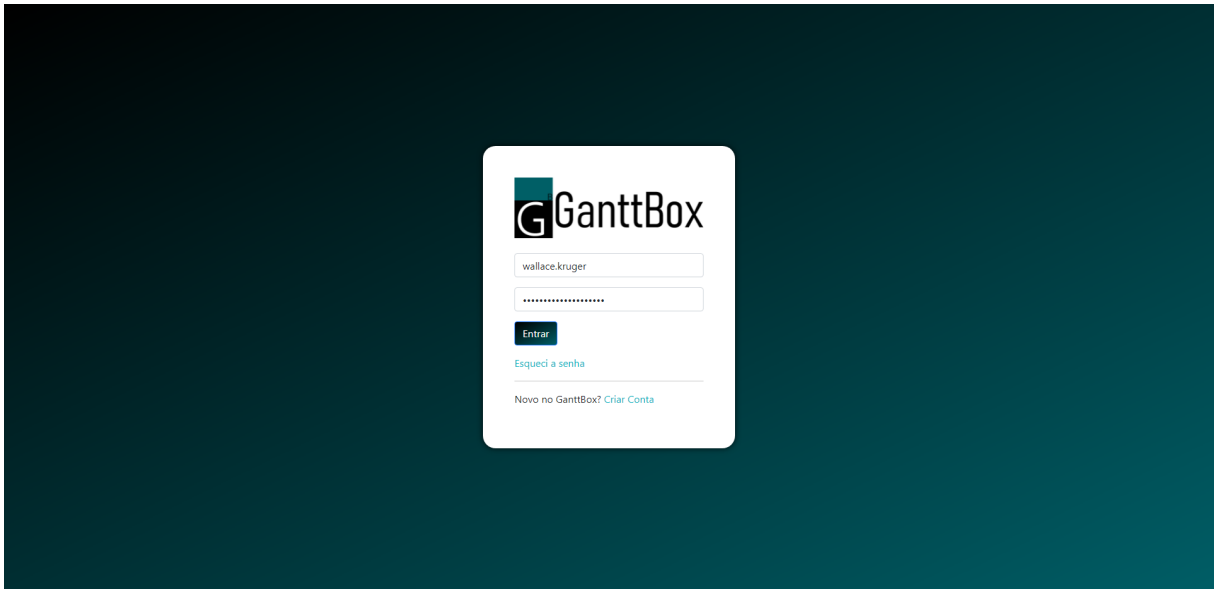


Figura 34 – Tela de Login

Como ele ainda não possuirá acesso ao sistema deverá clicar no *link Criar Conta*, então acessará a tela de cadastro apresentada na Figura 35.

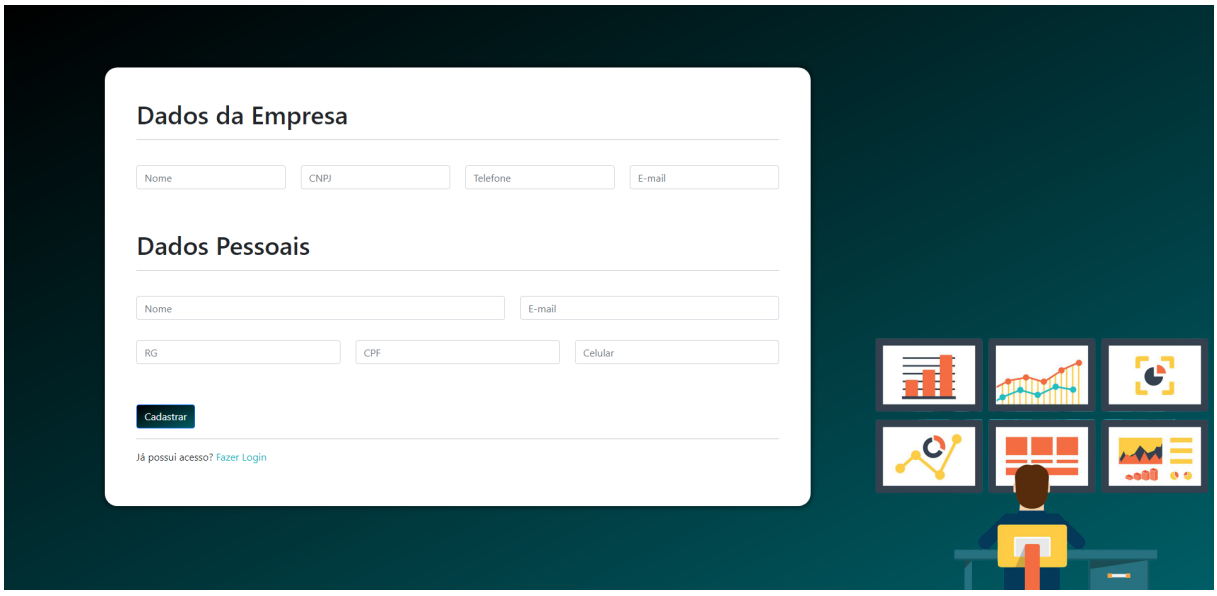


Figura 35 – Tela de Criar Conta

Após preencher todos os seus dados e da sua empresa nos formulários, o sistema criará a conta de Colaborador Gestor e enviará um e-mail com suas credenciais de acessos geradas automaticamente. Posteriormente o sistema o levará novamente para a tela de *login* e avisará que a conta foi criada com sucesso. O e-mail de boas vinda pode ser visualizado na Figura 36 e o aviso de sucesso na criação da conta, pode ser visto na Figura 37.

Assim que o Gestor receber o e-mail com suas credenciais e realizar a autenticação na tela de login, no seu primeiro acesso, receberá uma sugestão de troca de senha conforme pode ser visualizada na Figura 38.

Figura 36 – E-mail Criação de Conta

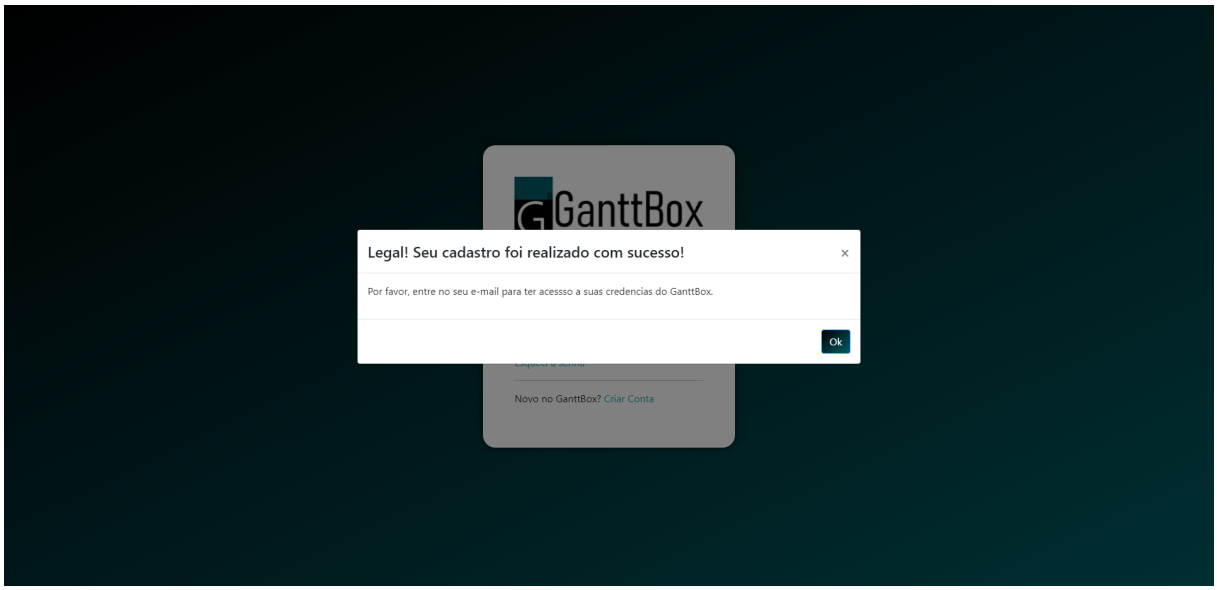


Figura 37 – Modal Conta Criada

O Gestor pode optar por já altear a senha naquele exato momento, ou num segundo momento se assim preferir, acessando o menu *Configurações -> Trocar Senha*. A tela de troca de senha pode ser visualizada pela Figura 39.

Quando realizar a troca senha receberá um e-mail de confirmação que pode ser visualizado na Figura 40.

Uma última funcionalidade a ser pontuada deste subsistema disponível no GanttBox é o esquecimento de senha. O modal de esquecimento de senha pode ser visualizado na Figura 41.

Após o usuário do sistema, preencher o e-mail vinculado à sua conta GanttBox, receberá um email com as instruções para acessar o sistema novamente.

Clicando no botão do e-mail disparado poderá redefinir sua senha e voltar ter acesso ao sistema. A tela de redefinição de senha pode ser visualizada na Figura 43.

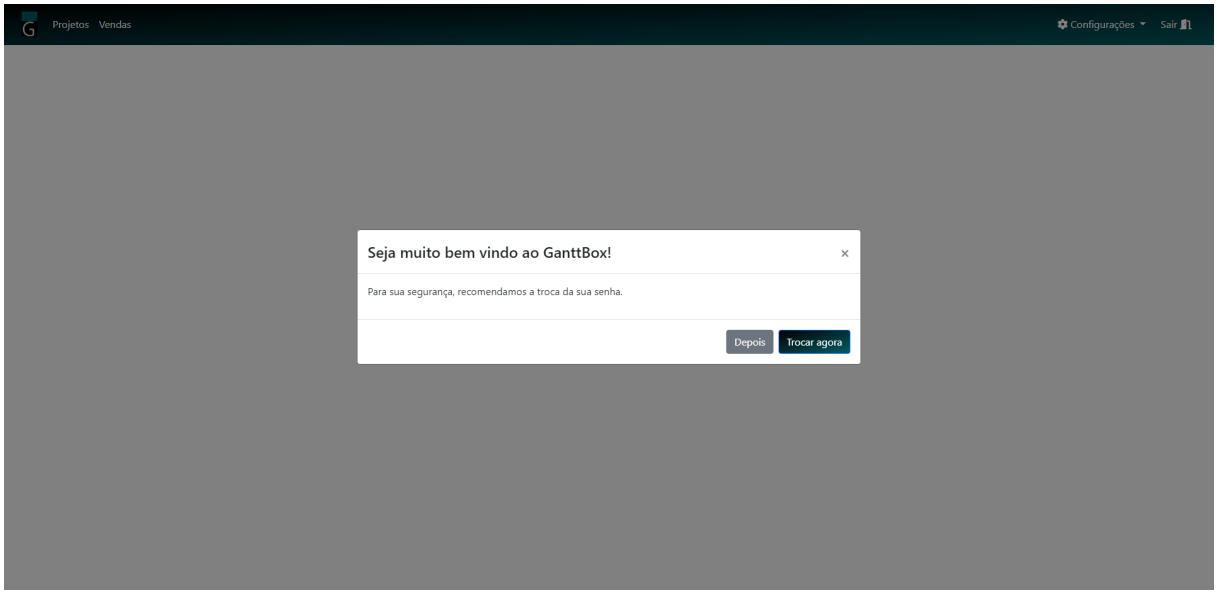


Figura 38 – Tela de Primeiro Acesso

Figura 39 – Tela Trocar Senha

4.3.2 SalesManagement

Como já pontuado em algumas partes deste texto, somente o Gestor terá acesso ao subsistema de vendas. Após se autenticar, o Gestor terá acesso ao menu de vendas clicando em Vendas no menu superior da barra de navegação. Ao entrar no menu vendas, terá a visão do funil de vendas, que num primeiro momento estará vazio, isto é, sem *leads* cadastradas. O funil de vendas num momento inicial pode ser visualizado pela Figura 44.

O Gestor então poderá criar a primeira Lead, clicando no botão “Criar Lead”, logo abaixo do Kanban de vendas. O modal de criação de uma *lead* pode ser visualizado através da Figura 45.

Com uma *lead* criada, o Gestor poderá editar e visualizar informações da mesma

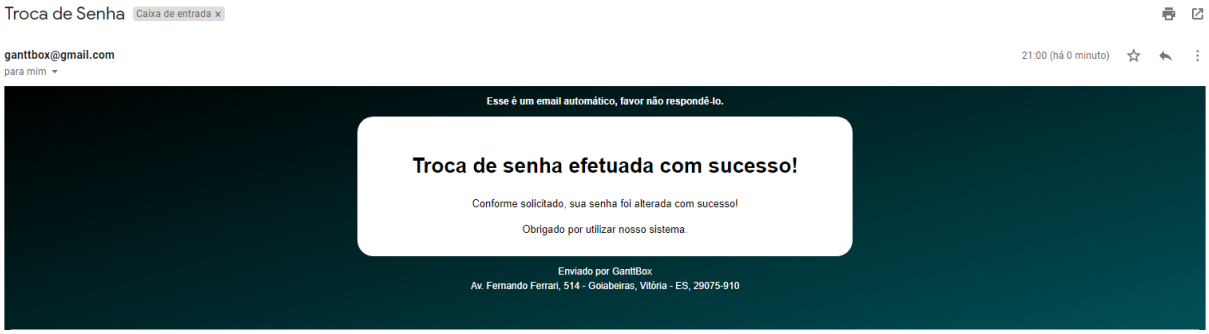


Figura 40 – E-mail Senha Alterada

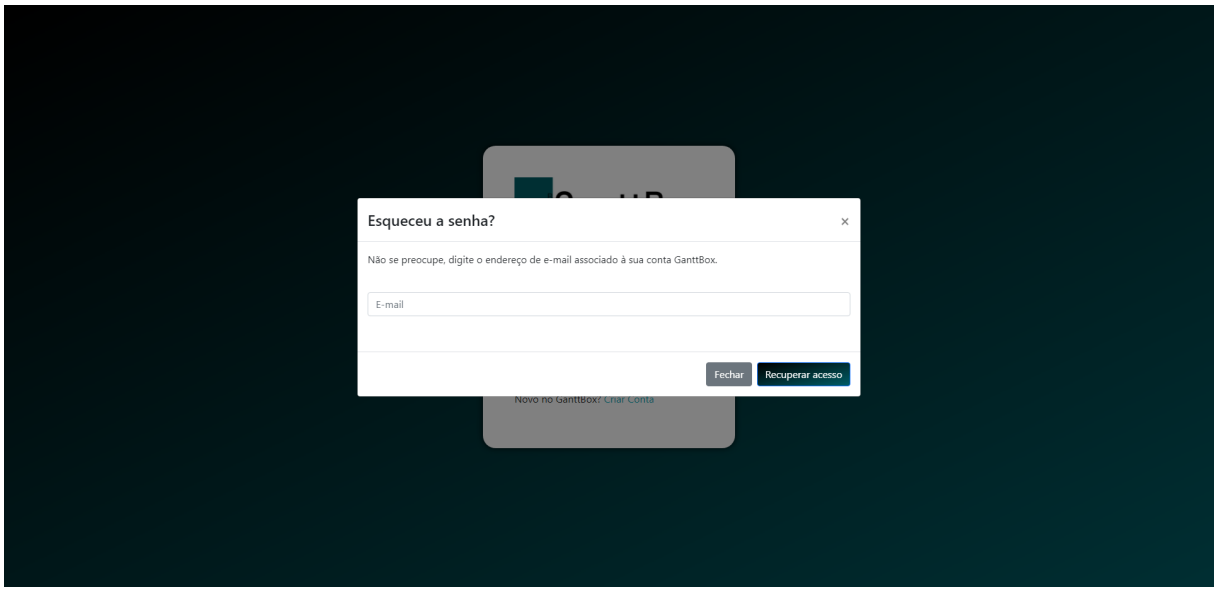


Figura 41 – Modal Esquecer Senha

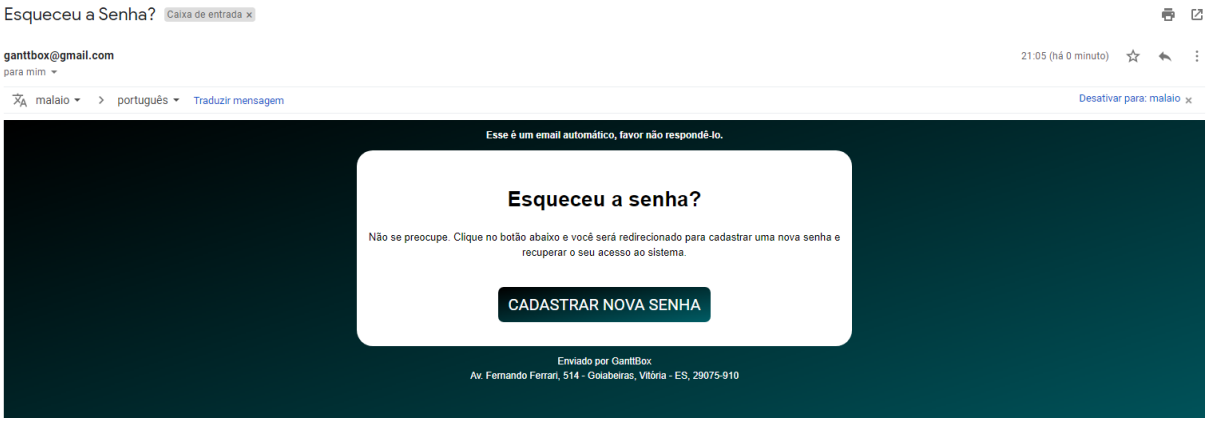


Figura 42 – E-mail Esquecimento de Senha

dando um duplo click sobre o *card* que a representa. A Figura 46 mostra o modal de edição de uma *lead*, bem como as opções de exclusão e declaração de perda, através de botões no mesmo modal.

Quando uma *lead* é movida para o status de tipo “proposta” no Kanban, o sistema exigirá a criação de uma proposta, conforme pode ser visualizado na Figura 47. Caso o

Figura 43 – Tela Redefinir Senha

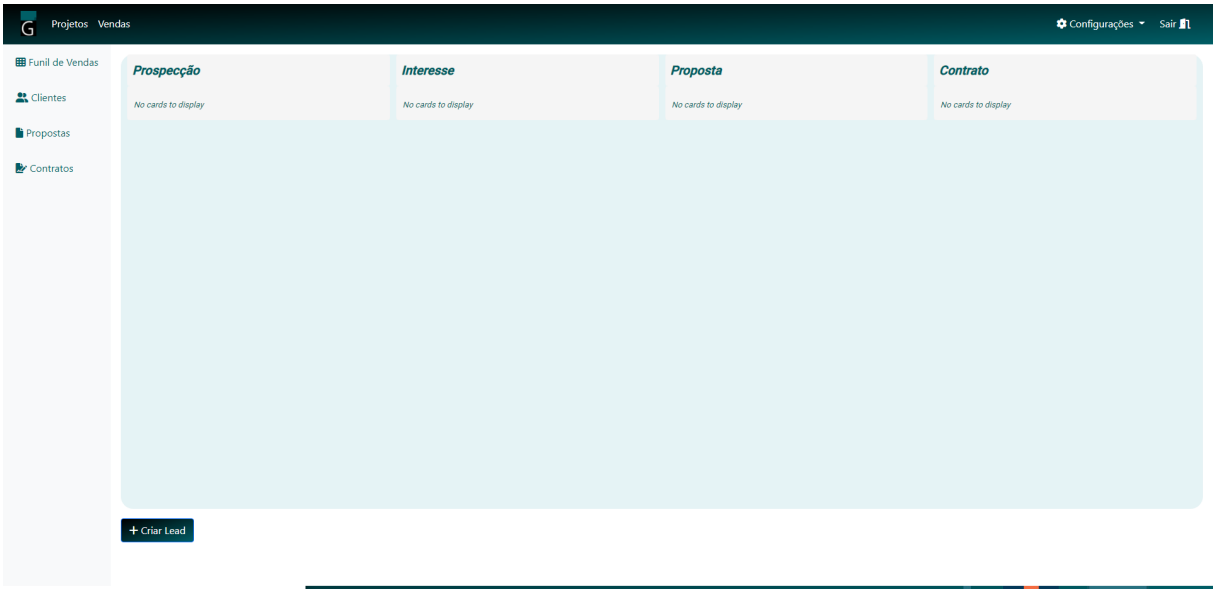


Figura 44 – Tela Funil de Vendas

Gestor cancele a criação da proposta, o *card* que representa a *lead* voltar pra o status de tipo “Interesse”.

Quando criada a proposta o Gestor poderá visualiza-la ou edita-la. A Figura 48 mostra o modal da *lead*, com suas respectivas funções para esta localização do *card*.

Caso a negociação avance, o Gestor moverá o *card* da *lead* para o status de tipo “Contrato”. Ao mover, da mesma forma colocada anterior, o sistema exigirá a criação do contrato, se não retornará a *lead* para o *slot* anterior. O modal de criação do contrato pode ser visualizado na Figura 49.

A Figura 50 mostra o modal de visualização e edição de contrato com suas respectivas funcionalidades.

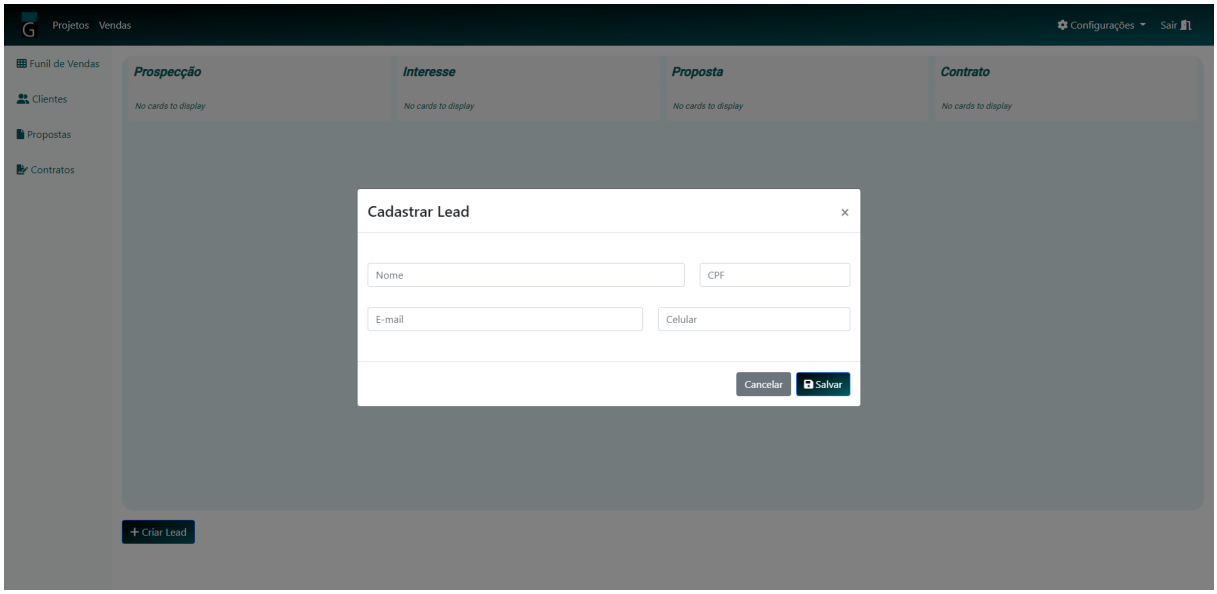


Figura 45 – Modal Criar Lead

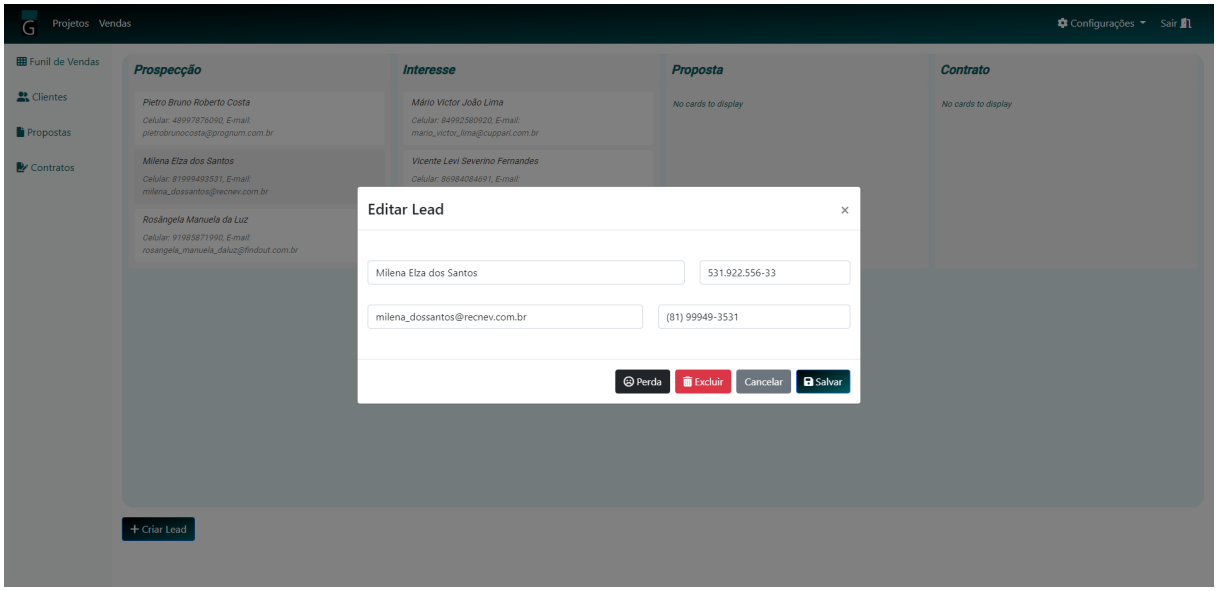


Figura 46 – Modal Editar Lead

Um das funcionalidades do modal de edição de contrato é a possibilidade de exportar o contrato para o arquivo PDF. Ao clicar no botão “Exportar Contrato” o sistema gerará o contrato. Um exemplo de contrato gerado pode ser visualizado na Figura 51.

A Figura 52 mostra a funcionalidade da declaração de ganho, última etapa neste fluxo do Kanban de Vendas.

4.3.3 ProjectManagement

Não foram feitas capturas de telas do subsistema *ProjectManagement* em funcionamento, pois o mesmo, ainda estava na etapa de desenvolvimento até o prazo de entrega desta monografia.

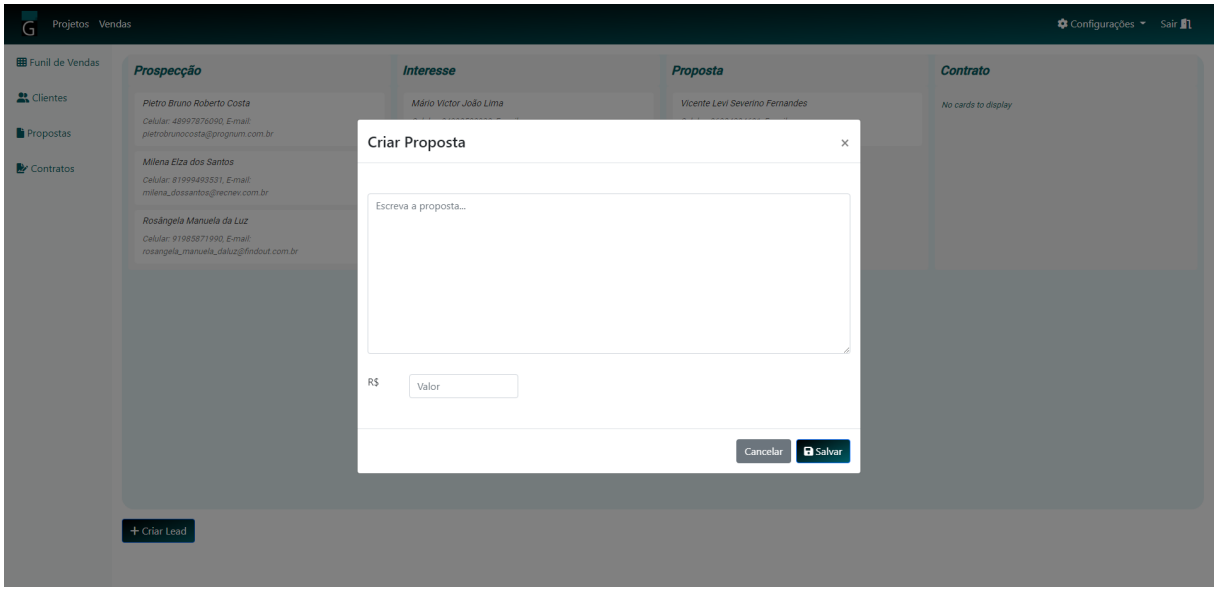


Figura 47 – Modal Criar Proposta

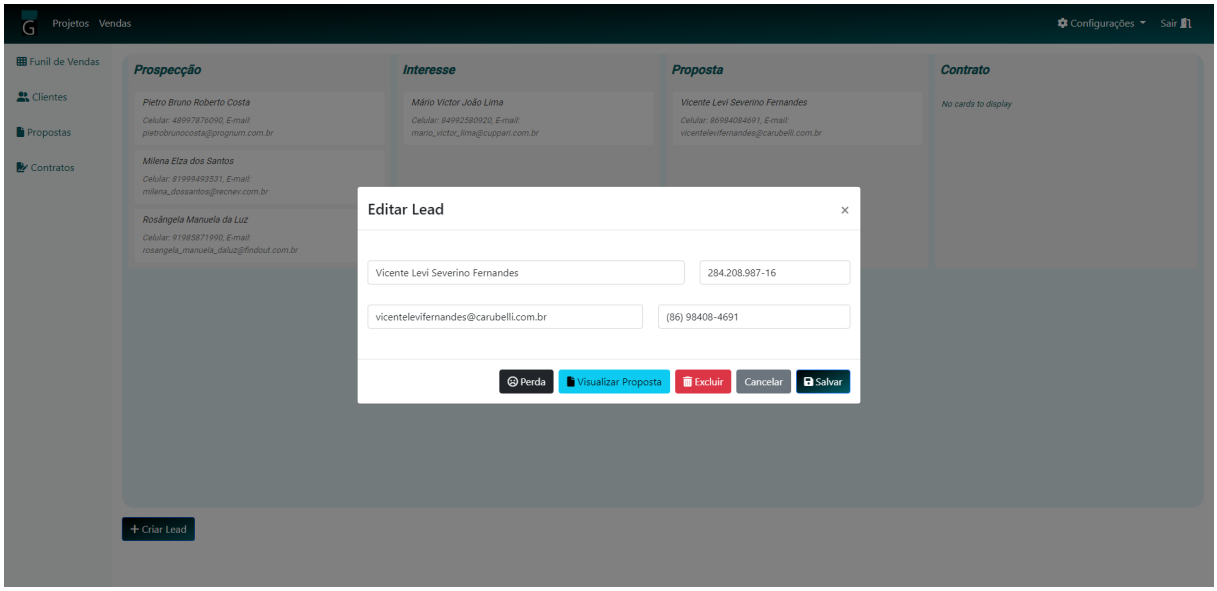


Figura 48 – Modal Visualizar Lead no Slot Proposta

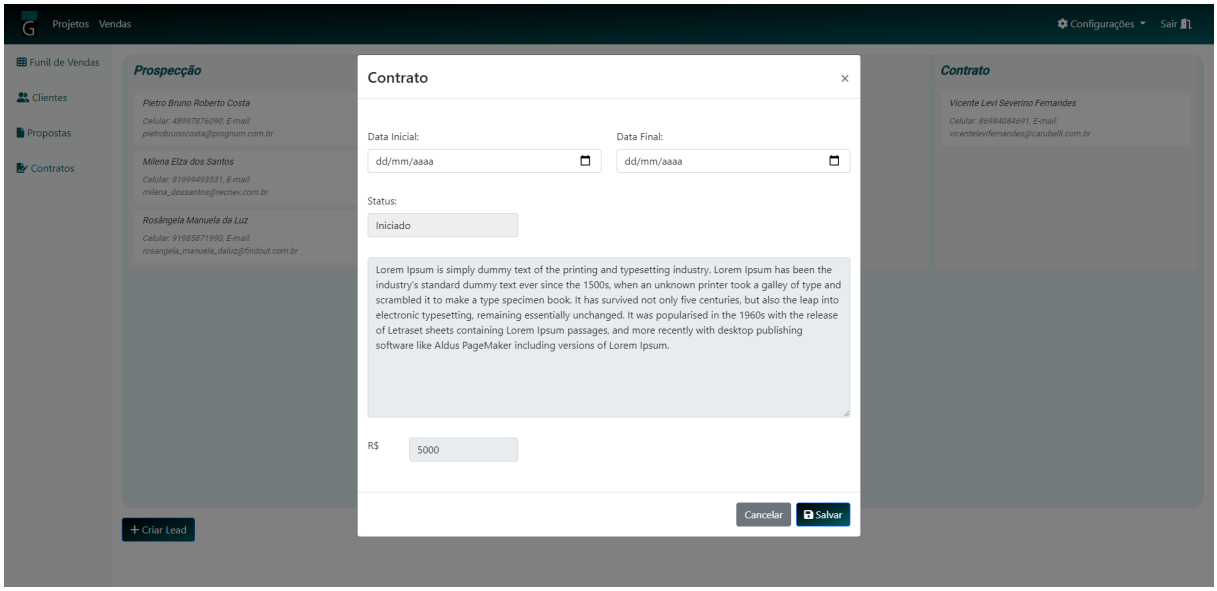


Figura 49 – Modal Criar Contrato

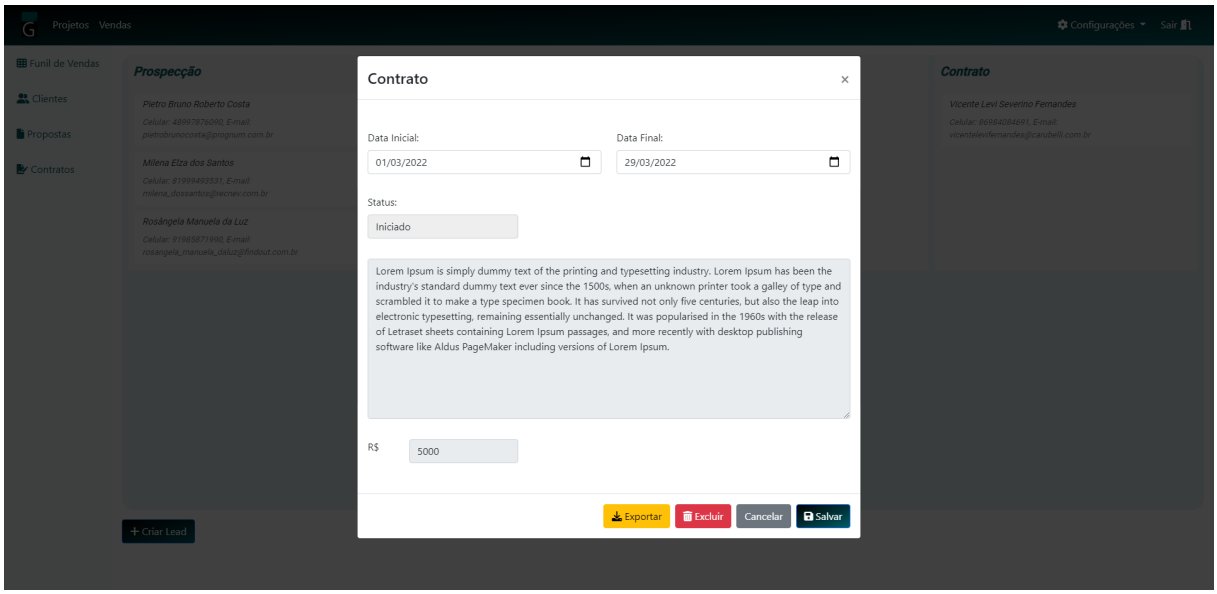


Figura 50 – Modal Editar Contrato

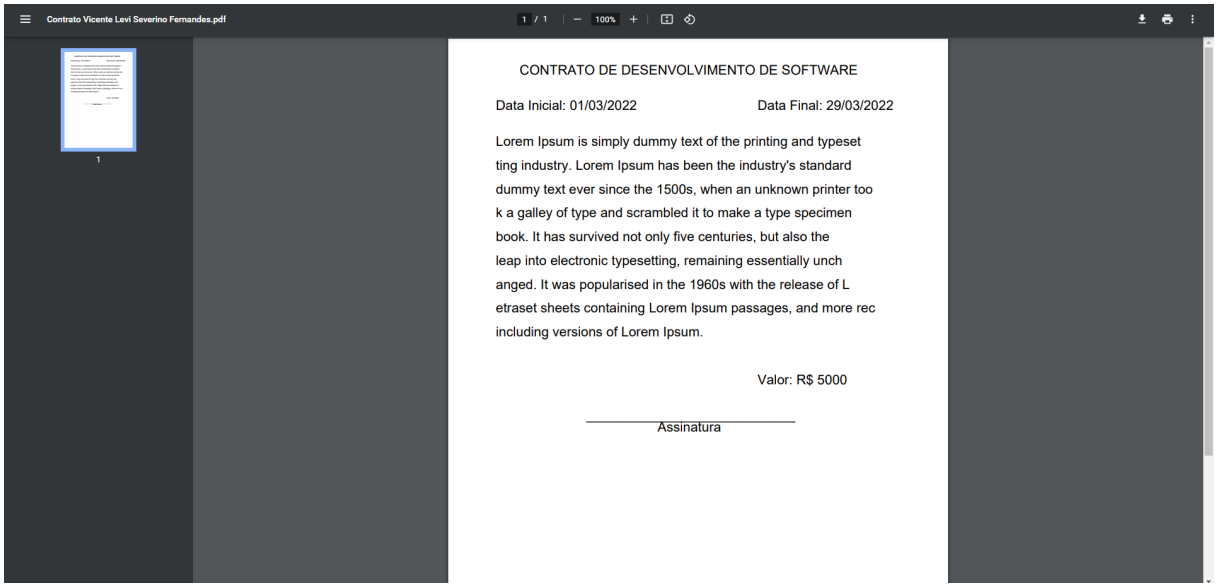


Figura 51 – PDF Exportado

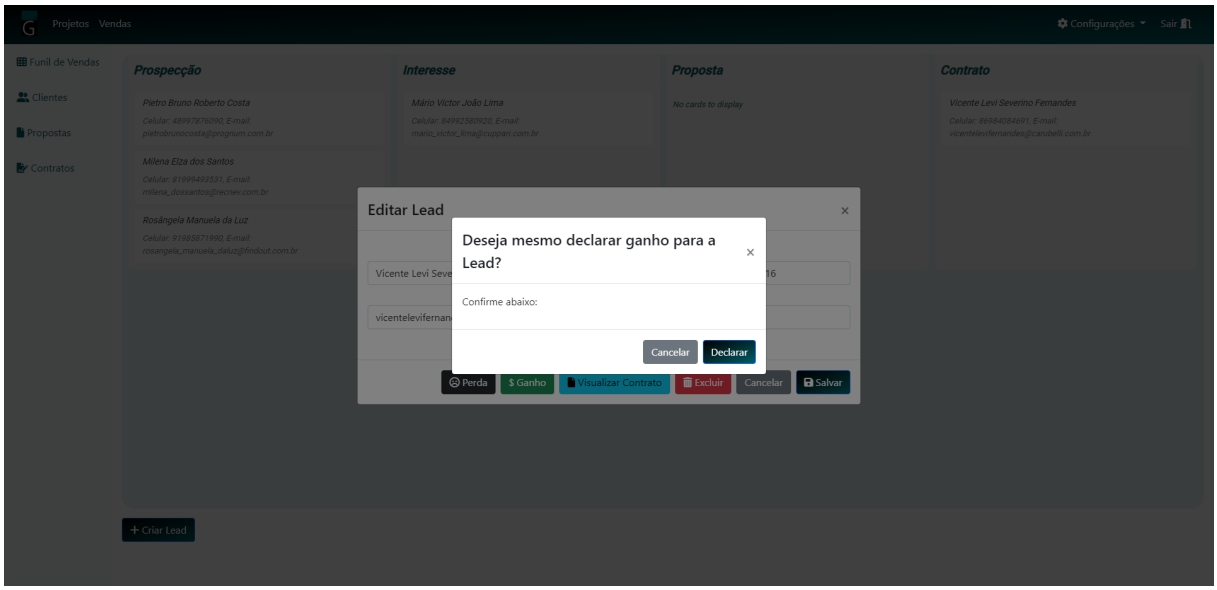


Figura 52 – Modal Declarar Ganho/Perda

5 Conclusão

Neste capítulo serão apresentadas as últimas considerações sobre o projeto desenvolvido, bem como perspectivas de trabalhos futuros. A Seção 5.1, apresenta as contribuições e um paralelo entre os resultados obtidos e o que se havia almejado inicialmente. A Seção 5.2 exibe uma reflexão sobre as limitações da ferramenta, assim como formas que a mesma poderá ser explorada para o procedimento do trabalho.

5.1 Considerações Finais

Uma pequena empresa de tecnologia foi fundada para atender uma necessidade específica dentro de um nicho também específico. Com o avanço da informação e da tecnologia somadas ao capitalismo, cada vez mais surgem demandas e percepções de melhoria de processos, visando o lucro. Neste ecossistema, surgem oportunidades da área de TI (Tecnologia da Informação) prestar as soluções que o mercado está precisando. Diante deste cenário, a pequena empresa de tecnologia, compreende que poderia aumentar seu campo de atuação, de forma a atender a crescente demanda do mercado, visando sua própria ampliação. A empresa, então, propõe o desafio para os seus colaboradores e começa a procura de novos contribuidores e equipamentos para completar o seu "arsenal".

Com o tempo, a empresa consegue evoluir, expandir seu mercado e ainda percebe que, a cada dia, o ritmo da procura por soluções e ferramentas informatizadas está aumentando. Entretanto, esbarra num momento em que os processos gerenciais, administrativos, contábeis e financeiros estão se tornando caóticos, pois não existe um processo sistemático que organiza e atende uma demanda de grande porte. Essas dificuldades estão contento a continuação do seu crescimento. A instituição, começa então a procurar ferramentas que melhorem seus processos e encontra no mercado uma porção de *softwares* que atendem apenas pontuais problemas enfrentados ou não estão dentro do orçamento estipulado. Identifica-se então, a necessidade de construir um próprio sistema de gestão, nascendo a ideia do GanttBox.

Para o desenvolvimento desta ferramenta, inspirada no cenário da empresa de tecnologia, seguiu-se um processo de Engenharia de Software, elaborando os documentos de Especificação de Requisitos e Projeto de Sistema, que são anexos deste texto. Entretanto, notou-se na escrita desses documentos que a ideia pensada inicialmente, almejando controle financeiro, contábil, geração de vários relatórios e *dashboards* para controle do seu Gestor, seria complexa demais pelo seu tamanho, extrapolando o tempo que é projetado para entrega de uma monografia.

Os resultados obtidos até aqui no desenvolvimento da ferramenta de gestão GanttBox podem, então, ser considerados um protótipo de um grande projeto, pois otimizam apenas uma pequena parte dos processos gerenciais da empresa. Ainda assim, o que foi construído até o momento, foi implementado como uma base sólida, projetada para escalar, pois atenderá não só a empresa que inspirou o desenvolvimento desta ferramenta, mas sim outras que também possuem deficiências e necessidades parecidas. Moldado pelas arquiteturas de *software* FrameWeb, *Clean Architecture* e *Multitenancy* e ajustado aos *design patterns*, seguindo os princípios e boas práticas da programação orientada a objetos. Pôde-se então colocar em prática neste projeto, o conhecimento adquirido em várias disciplinas estudadas ao longo da graduação, principalmente as mais ligadas aos processos de Engenharia de Software, programação e banco de dados.

Concluímos expondo que os objetivos premeditados no Capítulo 1 foram apenas parcialmente alcançados, pois foram solucionados apenas parte das necessidades identificadas, conforme pontuadas nos parágrafos anteriores. Observamos, no entanto, que os objetivos específicos de documentar, implementar e implantar o sistema foram alcançados para os módulos que entraram num escopo reduzido para a conclusão deste projeto de graduação.

5.2 Trabalhos Futuros

Como comentado na Seção 5.1, o sistema GanttBox construído até aqui atende apenas uma pequena porção das necessidades gerenciais da empresa, especificamente, a parte de vendas.

Até a entrega desta monografia o subsistema *projectManagement* estava seguindo avanços no seu desenvolvimento. Entretanto, até para este subsistema, o escopo no documento de especificação de requisitos foi reduzido, de forma a se tentar mais ainda, moderar a complexidade para uma monografia. Inicialmente, este subsistema seria muito mais complexo, pois integraria todos os outros módulos e possuiria muitos outros recursos e funcionalidades que agregariam no conhecimento do próprio negócio ao seu gestor.

Podemos levantar para o prosseguimento do desenvolvimento do subsistema *projectManagement*, não pontuados nos documentos anexados, páginas de negociação de estimativas de esforço em cima de *sprints* ou tarefas, em que o cliente e o gestor estipulariam tempo e valor a ser pago pra o desenvolvimento em questão, até que se chegasse em um acordo na negociação. A partir disso, cada tarefa que fosse então criada já teria um tempo estipulado previamente negociado. Podemos ainda considerar como melhorias nesse módulo pesquisas de satisfação enviadas aos clientes para realizar medições. Toda vez que o Gestor quisesse medir o grau de satisfação que a equipe de um projeto estava passando para seu cliente, ele enviaria essa pesquisa de forma a manter contato com seu cliente e também

conseguir retirar informações e tomar decisões a partir delas. Os resultados das medições poderiam ser disponibilizados no sistemas através de *dashboards*, com gráficos interativos, melhorando a experiência do gestor ao utilizar o sistema e facilitando a compreensão daquela informação recolhida.

Além dos subsistemas desenvolvidos, foram pensados inicialmente ainda mais 2 módulos, financeiro e contábil, que prestariam aos administradores, toda uma visualização e gerência facilitada dos recursos monetários que entram e saem dos cofres da empresa, deixando de lado extensas planilhas e gerando outros *dashboards*, que facilitariam e agilizariam muitas análises, como pagamento de tributos e rendimentos de cada colaborador especificados por horas e projetos trabalhados.

Mais uma sugestão que se consegue visar para trabalhos futuros desta ferramenta seria ainda a construção de um módulo administrativo, que a própria equipe de desenvolvimento usaria para gerir permissões de acesso dos usuários ou gerir cadastros das próprias empresas clientes do sistema, de forma que, para um futuro suporte prestado a resoluções de problemas que o sistema possivelmente apresentaria, o módulo administrador poderia ser usado, corrigindo a inconsistência e evitando por exemplo que se mexesse diretamente em um banco de produção.

Por fim, durante a apresentação deste trabalho foram feitas uma série de sugestões pertinentes por parte da banca avaliadora. A maioria das sugestões foram incorporadas nesta versão final, porém registramos como trabalhos futuros aquelas que não puderam ser incorporadas em tempo hábil:

- Melhorias na visualização dos diagramas de casos de uso, por exemplo especificando um ator genérico que seria especializado pelos demais atores para os casos de uso em que todos têm acesso;
- Melhorias no diagrama de classes da fase de requisitos, adicionando nomes aos relacionamentos entre classes e utilizando tipos específicos de domínio;
- Melhorias na funcionalidade de *cards* do Kanban, permitindo saber, por exemplo, o histórico de movimentação dos *cards* (quando um *card* mudou de status);
- Melhorias nas funcionalidades de projetos e *sprints*, permitindo a gestão do tempo de cada uma (registro do início e do fim das mesmas);
- Melhorias na gestão dos cargos dos usuários (*manager*, *collaborator*), permitindo por exemplo gerenciar o histórico de cada um (um colaborador que se torna gestor a partir de uma certa data, por exemplo);
- Melhorias nas funcionalidades de criação de propostas e de contratos, para ficar mais condizente com a realidade das empresas (que costumam redigir tais documentos

primeiro em softwares de edição como Word ou LibreOffice e depois inserir no sistema e não o contrário).

Referências

CAMPOS, S. L.; SOUZA, V. E. S. FrameWeb Editor: Uma Ferramenta CASE para suporte ao Método FrameWeb. In: *Anais do 16º Workshop de Ferramentas e Aplicações, 23º Simpósio Brasileiro de Sistemas Multimedia e Web (WFA/WebMedia 2017)*. Gramado, RS, Brazil: SBC, 2017. p. 199–203. Citado 3 vezes nas páginas 6, 25 e 26.

CASTELEYN, S. et al. *Engineering Web Applications*. 1th. ed. [S.l.]: Springer Publishing Company, Incorporated, 2009. ISBN 3540922008, 9783540922001. Citado na página 20.

CONFORTO, E. C. et al. Can agile project management be adopted by industries other than software development? *Project Management Journal*, SAGE Publications Sage CA: Los Angeles, CA, v. 45, n. 3, p. 21–34, 2014. Citado na página 26.

CRUZ, F. *Scrum e PMBOK unidos no Gerenciamento de Projetos*. [S.l.]: Brasport, 2013. Citado na página 26.

DINSMORE, P. C.; NETO, F. H. S. *Gerenciamento de projetos e o fator humano*. [S.l.]: Qualitymark Editora Ltda, 2006. Citado na página 12.

DORF, R. C.; BISHOP, R. H. Sistemas de controle modernos, 8ª edição. *Editora LTC, Rio de Janeiro*, 2001. Citado na página 12.

FALBO, R. d. A. *Engenharia de Software - Notas de Aula*. [S.l.: s.n.], 2014. Citado 4 vezes nas páginas 15, 16, 17 e 18.

FIELDING, R. *other, HyperText Transfer Protocol (HTTP/1.0)*. [S.l.], 1997. Citado na página 19.

FOWLER, M. *Patterns of Enterprise Application Architecture*. [S.l.: s.n.], 2002. Citado na página 25.

FOWLER, M.; HIGHSMITH, J. et al. The agile manifesto. *Software development*, [San Francisco, CA: Miller Freeman, Inc., 1993-, v. 9, n. 8, p. 28–35, 2001. Citado na página 27.

GOULART, A. Informação: precisamos definir esse termo. *Observatório da Imprensa*, v. 20, 2004. Citado na página 12.

HUANG, C.-C.; KUSIAK, A. Overview of kanban systems. Taylor & Francis, 1996. Citado na página 28.

KREBS, R.; MOMM, C.; KOUNEV, S. Architectural concerns in multi-tenant saas applications. *Closer*, v. 12, p. 426–431, 2012. Citado na página 24.

LAFRANCE, A. How many websites are there. *The Atlantic*, v. 30, 2015. Citado na página 19.

MARTIN, R. C.; GRENNING, J.; BROWN, S. *Clean architecture: a craftsman's guide to software structure and design*. [S.l.]: Prentice Hall, 2018. Citado 3 vezes nas páginas 6, 22 e 44.

MERKLE, E. R.; RICHARDSON, R. A. Digital dating and virtual relating: Conceptualizing computer mediated romantic relationships. *Family Relations*, Wiley Online Library, v. 49, n. 2, p. 187–192, 2000. Citado na página 19.

MOONEEGAN, V. *IoT – Internet of Things*. 2016. Citado na página 12.

MYERS, G. J. et al. *The art of software testing*. [S.l.]: Wiley Online Library, 2004. v. 2. Citado na página 17.

NODEJS.ORG. *About Node.js*. 2021. Disponível em: <https://nodejs.org/en/about/>. Citado na página 21.

PRESSMAN, R. S. *Engenharia de Software - Uma Abordagem Profissional*. 7th. ed. [S.l.]: AMGH EDITORA LTDA, 2011. Citado 2 vezes nas páginas 15 e 17.

ROCHA, A. R. C. d. et al. *Qualidade de software: teoria e prática*. [S.l.]: São Paulo: Prentice Hall, 2001. Citado na página 17.

ROCHA, H. L. S. *Arquitetura da Web*. [S.l.: s.n.], 1999. Citado na página 20.

SANCHES, R. Processo de manutenção. *Qualidade de Software: Teoria e Prática*, 2001. Citado na página 18.

SCHWABER, K. Scrum development process. In: *Business object design and implementation*. [S.l.]: Springer, 1997. p. 117–134. Citado na página 28.

SOMMERVILLE, I. e. a. *Engenharia de Software 9º edição*. 9th. ed. [S.l.]: Pearson, 2011. Citado 3 vezes nas páginas 16, 17 e 18.

SOUZA, V. E. S. *FrameWeb: um Método baseado em Frameworks para o Projeto de Sistemas de Informação Web*. [S.l.], 2007. Disponível em: <https://nemo.inf.ufes.br/wp-content/papercite-data/pdf/souza-masterthesis07.pdf>. Citado na página 25.

SOUZA, V. E. S. The FrameWeb Approach to Web Engineering: Past, Present and Future. In: ALMEIDA, J. P. A.; GUIZZARDI, G. (Ed.). *Engineering Ontologies and Ontologies for Engineering*. 1. ed. Vitória, ES, Brazil: NEMO, 2020. cap. 8, p. 100–124. ISBN 9781393963035. Disponível em: <http://purl.org/nemo/celebratingfalbo>. Citado 3 vezes nas páginas 13, 25 e 44.

TANENBAUM, A. S. *Redes de Computadores*. trad. 5 ed. São Paulo: Pearson, 2011. Citado na página 19.

Apêndices

Documento de Requisitos de Sistema

GanttBox - Sistema de Gestão de Projetos

Vitória, ES

2022

Registro de Alterações:

Versão	Responsável	Data	Alterações
1.0	Wallace Krüger Junior	23/01/2022	Versão inicial.
1.1	Vitor E. Silva Souza	02/02/2022	Primeira Revisão.
1.2	Wallace Krüger Junior	06/02/2022	Ajustes da primeira revisão.
1.3	Vitor E. Silva Souza	11/02/2022	Segunda Revisão.
1.4	Wallace Krüger Junior	13/02/2022	Ajustes da segunda revisão.
1.5	Vitor E. Silva Souza	24/02/2022	Terceira Revisão.
1.6	Wallace Krüger Junior	01/03/2022	Ajustes da teceira revisão.

1 Introdução

Este documento apresenta a especificação dos requisitos do sistema de gestão de projetos GanttBox. Esta especificação foi construída aplicando-se técnicas de levantamento de requisitos, bem como modelagem de casos de uso e de classes utilizando a linguagem UML e foi organizada da seguinte forma: a Seção 2 contém uma descrição do propósito do sistema; a Seção 3 apresenta uma descrição do minimundo apresentando o problema; a Seção 4 apresenta a lista de requisitos de usuários levantados junto ao cliente; a Seção 5 explica a divisão em subsistemas, descrevendo brevemente cada um deles; a Seção 6 apresenta o modelo de casos de uso, incluindo descrições de atores, os diagramas de casos de uso e suas respectivas descrições; a Seção 7 traz os modelos conceituais estruturais do sistema na forma de diagramas de classes. Por fim, a Seção 8 detalha o dicionário do projeto, contendo as definições das classes identificadas.

2 Descrição do Propósito do Sistema

Em qualquer área do conhecimento, qualificar o resultado final de um produto, está intimamente ligado a otimizar a qualidade do seu processo de produção. Dependendo da complexidade do bem almejado, somada à calorosa disputa de mercado que nosso sistema econômico molda, um processo de engenharia sistemático não é uma escolha, mas sim uma necessidade para se atingir o resultado esperado e manter-se diligente no mercado.

O objetivo da ferramenta de gestão de projetos é unir os conhecimentos da área de gestão, junto com a Engenharia de Software, para produzir uma aplicação que irá integrar várias funcionalidades de gerência em um só lugar, suprindo, principalmente, as necessidades dos gestores de uma pequena empresa de tecnologia. A ferramento almeja automatizar tarefas presentes nos processos de gerência de projetos, numa *software house* no município de Vitória, ES, propondo-se a otimizar tempo, além de entregar um maior conhecimento de seu próprio negócio aos seus gestores, monitorando custos, controlando riscos, medições mais assertivas, dentre outros benefícios de gestão.

3 Descrição do Minimundo

A cada dia mais cresce a procura por soluções na área de TI (Tecnologia da Informação). Neste ecossistema, habita uma empresa de tecnologia, que retém um quadro

de funcionários qualificados, com um ambiente e equipamentos favoráveis para atender esta demanda do mercado. Entretanto, a equipe administrativa do negócio leva a rotina de forma manual, como o preenchimento de extensas planilhas, tarefas repetitivas, cansativas e até caóticas, contendo a expansão da instituição. Em vista disso, um processo metódico, inteligente e mecanizado se faz necessário.

No cenário da pequena instituição de crescimento emergente, onde, cada vez mais, surgem demandas de serviços e contatos de novos clientes, o sistema *GanttBox*, será um mediador das execuções de atividades de cada figura deste quadro, organizando e estruturando os processos com a finalidade de consignar poder e conhecimento do negócio ao seu administrador. Três tipos de usuários poderão ter acesso ao sistema: o cliente, o colaborador comum e o colaborador gestor.

O cliente terá acesso ao sistema assim que uma negociação for concluída com sucesso, isto é, após acordar uma proposta e selar um contrato com a empresa, tornando-se de fato um ator que gere lucro. Ao acessar o sistema, este verá o projeto ou a lista de projetos que estão em andamento com a empresa. Com um projeto em andamento, ele poderá visualizar um panorama geral do mesmo, como, por exemplo, quais tarefas que já estão concluídas, atrasadas, o que falta para finalizar uma determinada *sprint* em andamento, dentre outros recursos.

Um colaborador comum, usará o sistema basicamente para organizar e conduzir sua rotina de trabalho. Quando alocado dentro de um projeto, poderá criar e gerir *cards* dentro do *Kanban* de projeto, os *cards* representarão as tarefas, que podem estar sob sua responsabilidade ou não, além disso, o colaborador usará o sistema para "dar *play* ou *stop*" na tarefa que está direcionando sua atenção atualmente, desta forma, o sistema poderá recolher esse tempo e apontar seu esforço dentro daquele projeto.

O colaborador Gestor, terá acesso a todas as funcionalidades do sistema, podendo assim controlar os processos, tarefas de um determinado projeto, medir quais funcionários estão condizentes a suas entregas, ou quanto um funcionário rende por hora trabalhada. Ele poderá medir a satisfação do seu cliente, dentre outras muitas funcionalidades que consolidaram o conhecimento sobre seu negócio e o guiará nas tomadas de decisões.

Como uma solução mediatária, a ferramenta de gestão, propõe-se disponibilizar vários recursos e funcionalidades para seus usuários dentro do sistema, para assim, tornar-se a solução que almeja ser. As subseções a seguir detalham importantes termos que estão inseridos dentro deste universo, de entendimento essencial para correta utilização e funcionamento do *GanttBox*.

3.1 Quadro Funil

Mais conhecido como *Kanban*, método visual muito utilizado dentro das metodologias ágeis, com intuito de organizar, melhorar e conduzir o fluxo de trabalho de uma equipe, limitando e transparecendo o números de atividades em andamentos para maximizar o fluxo atual de entrega. Dentro do sistema de gestão, existirão dois quadros *Kanban*, o quadro funil de vendas, onde somente o gestor terá acesso para gerir negociações de novos potenciais clientes, e o quadro funil de projeto, em que todos os 3 atores terão acessos, porém o cliente terá acesso apenas para visualização e informar-se sobre o andamento do acordado projeto, enquanto o colaborador comum e o colaborador gestor poderão gerir os recursos do quadro como mover os *cards*, que representam as tarefas, atribuir responsáveis, criar novos *cards* entre outros recursos desta seção.

3.2 Slot

O método Kanban, é uma interface que assemelha-se com um quadro, este quadro é dividido em colunas, que representaram etapas dentro de um contexto, estas etapas traduzem um status para uma atividade que esta se movendo dentro do fluxo. Neste texto, adotou-se o nome *slot* para a representação de cada etapa dentro do quadro Kanban.

3.3 Card

O *card* é o componente central do método *Kanban*, um elemento visual que resume as informações de uma atividade dentro de um fluxo de trabalho, como por exemplo, nome, descrição, responsável, prazo e etc. No sistema, no quadro de vendas, um *card* representará uma *Lead*, enquanto no quadro do projeto, um *card* representará uma tarefa.

3.4 Lead

Uma *lead*, dentro do sistema, é qualquer pessoa física que está com uma negociação em andamento com a empresa, com objetivo de selar um contrato para desenvolvimento de um projeto, tornando-se eventualmente um cliente caso esta negociação termine em sucesso.

3.5 Projeto

Um projeto, dentro do sistema, é um empreendimento que a empresa contratada faz para construir uma solução que foi acordada com um cliente, com escopo e prazos bem definidos.

3.6 Sprint

O conceito de *sprint* está intimamente ligado as metodologias ágeis, em especial o *Scrum*. Este método propõe em linhas gerais que um projeto seja dividido em diversos ciclos de atividades, dividindo as entregas em etapas, buscando melhorar dentre outras coisa a produtividade da equipe. Para cada etapa do projeto, chamamos no *GanttBox* de *sprint*.

3.7 Tarefa

A tarefa dentro do cenário da *sprint*, são pequenas atividades a serem feitas por determinados responsáveis para o progresso do projeto.

3.8 Setor

Organização e agrupamento dos colaboradores da empresa por funções, como por exemplo, RH, administrativo e TI.

4 Requisitos de Usuário

Tomando por base o contexto do sistema descrito na Seção 3, foram identificados os seguintes requisitos de usuário.

Tabela 1 – Requisitos Funcionais.

ID	Descrição	Prioridade	Depende
RF-1	O sistema deve permitir que um gestor crie sua conta no primeiro acesso ao sistema, cadastrando seus dados de colaborador e os dados de sua empresa para o uso da aplicação. Uma empresa possui nome, código, cnpj, telefone comercial e e-mail.	Alta	
RF-2	O sistema deve permitir que um usuário efetue login ao inserir nome de usuário e senha de acesso.	Alta	
RF-3	O sistema deve permitir que um usuário recupere sua senha caso ele esteja cadastrado.	Alta	
RF-4	O sistema deve permitir que um usuário troque sua senha.	Alta	

ID	Descrição	Prioridade	Depende
RF-5	O sistema deve permitir gerenciar um quadro funil (Kanban). Um *slot* do funil possui um título e status.	Alta	
RF-6	O sistema deve permitir gerenciar *leads*. Leads possuem nome, CPF, e-mail, telefone, celular e status (etapa do funil).	Alta	RF-5
RF-7	O sistema deve permitir alterar o status de uma *lead*, isto é, mover a *lead* no "Kanban".	Alta	RF-5, RF-6
RF-8	O sistema deve permitir declarar ganho ou perda para uma *lead*.	Alta	RF-6
RF-9	O sistema deve permitir transformar uma *lead* em um eventual cliente, caso se declare ganho para esta *lead*, isto é, a negociação tenha terminado com sucesso.	Alta	RF-6, RF-10
RF-10	O sistema deve permitir a gestão de clientes. Um cliente possui código, nome, nome de usuário de acesso, senha, CPF, e-mail, telefone, celular, endereço (CEP, número, rua, complemento, bairro, cidade, estado, latitude e longitude), contratos e, opcionalmente, uma foto.	Alta	
RF-11	O sistema deve permitir gerenciar propostas. Uma proposta possui código, descrição, data de envio, valor e *lead*.	Alta	
RF-12	O sistema deve permitir gerenciar contratos. Um contrato possui proposta, código, *lead*, data inicial, data final e status (Elaboração, Aguardando, Vigente, Rescindido, Concluído).	Alta	
RF-13	O sistema deve permitir exportar um contrato para arquivo em formato PDF.	Alta	
RF-14	O sistema deve permitir gerenciar colaboradores. Um colaborador possui nome de usuário de acesso, senha, e-mail, nome, código, estado civil, sexo, nome do pai, nome da mãe, cônjuge, número de dependentes, endereço (CEP, número, rua, complemento, bairro, cidade, estado, latitude e longitude), documentação (RG, CPF, PIS/PASEP, escolaridade), dados bancários (banco, agência e conta) e, opcionalmente, uma foto.	Alta	

ID	Descrição	Prioridade	Depende
RF-15	O sistema deve permitir que um usuário altere seus respectivos dados cadastrais.	Alta	
RF-16	O sistema deve permitir gerenciar setores. Um setor tem código e nome.	Baixa	
RF-17	O sistema deve permitir a gestão de projetos. Um projeto possui cliente, contrato, código, nome, valor total, status (Negociação, Aprovado, Em desenvolvimento, Em homologação, Implantado, Termo de aceite enviado, Concluído e Cancelado), Tag (nome, cor), esforço (data inicial prevista, data final prevista, data inicial realizada, data final realizada), colaboradores.	Muito Alta	
RF-18	O sistema deve permitir criar *sprints* para um projeto. Uma sprint possui um projeto, título e, opcionalmente, um subtítulo.	Alta	RF-17
RF-19	O sistema deve permitir criar tarefas para uma *sprint*. Uma tarefa possui código, nome, status, descrição, data inicial prevista, data final prevista, estimativa, data inicial, data final, duração, anexos, progresso, colaboradores responsáveis e *sprint*.	Alta	RF-18
RF-20	O sistema deve ser permitir alocar um colaborador em um projeto.	Alta	
RF-21	O sistema deve ser permitir alocar um colaborador em uma tarefa.	Alta	
RF-22	O sistema deve permitir mudar o status de uma tarefa, isto é, mover a tarefa no “Kanban” do projeto. Um status possui nome e cor.	Alta	RF-5, RF-19
RF-23	O sistema deve permitir dar *play* ou *stop* em uma tarefa.	Alta	RF-19
RF-24	O sistema deve permitir anexar imagens e documentos a uma tarefa.	Alta	
RF-25	O sistema deve ser capaz de disparar e-mails. Os e-mails serão disparados na criação da conta de usuário e na troca/esquecimento da senha auxiliando e garantindo a segurança dos dados do usuário.	Alta	

Tabela 2 – Requisitos Não Funcionais.

ID	Descrição	Categoria	Escopo	Prioridade
RNF-1	A aplicação deve estar disponível pela internet, com compatibilidade aos principais navegadores disponíveis no mercado.	Portabilidade	Funcionalidade	Alta
RNF-2	Uso de Design responsivo nas interfaces gráficas.	Usabilidade	Funcionalidade	Média
RNF-3	O sistema deve ser fácil de manter, de modo a acomodar novas funcionalidades ou até mesmo adaptações para organizações específicas.	Manutenibilidade	Sistema	Alta
RNF-4	O sistema deve controlar o acesso às funcionalidades de gerenciamento por meio de autenticação e autorização.	Segurança	Sistema	Alta
RNF-5	O sistema segregará os dados por diferentes organizações.	Confiabilidade	Sistema	Alta
RNF-6	O sistema deve garantir a consistência de dados, obrigando o usuário a preencher todos os campos essenciais na hora de inserir dados.	Confiabilidade	Sistema	Alta
RNF-7	O sistema deve ter alta disponibilidade, superior a 98% do tempo.	Disponibilidade	Sistema	Alta
RNF-8	O sistema deve possuir mecanismos para se recuperar de falhas e continuar operacional.	Recuperabilidade	Sistema	Alta

ID	Descrição	Categoria	Escopo	Prioridade
RNF-9	O sistema deve se comunicar com os serviços da Amazon S3.	Interoperabilidade	Sistema	Alta
RNF-10	O sistema deverá processar requisições paralelamente em tempo hábil.	Eficiência	Sistema	Alta

5 Identificação de Subsistemas

Para atender aos requisitos elencados na Seção 4 e a fim de facilitar o gerenciamento do projeto, o sistema foi estruturado em 3 subsistemas: ProjectManagement, SalesManagement e UserManagement.

A Figura 1 ilustra os subsistemas e suas interdependências, enquanto a Tabela 3 apresenta uma breve descrição de cada subsistema identificado.

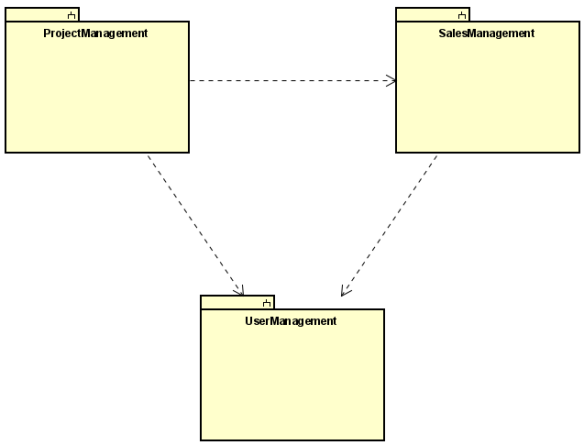


Figura 1 – Diagrama de Pacotes e os Subsistemas Identificados.

Tabela 3 – Subsistemas identificados e suas interdependências.

Subsistema	Descrição
SalesManagement	Subsistema responsável pelas funcionalidades relacionadas ao cadastramento, alteração, visualização e gestão de vendas.
ProjectManagement	Subsistema responsável pelas funcionalidades relacionadas ao cadastramento, alteração, visualização e gestão de projetos.
UserManagement	Subsistema responsável pelo gerenciamento e autenticação de usuários no sistema.

Subsistema	Descrição

6 Modelo de Casos de Uso

O modelo de casos de uso corresponde a uma tentativa de descrever a relação das funcionalidades do sistema com cada um de seus atores. Os atores identificados no contexto deste projeto estão descritos na Tabela 4.

Tabela 4 – Descrição dos atores envolvidos nos casos de uso.

Ator	Descrição
Cliente	Pessoa que está contratando o serviço da empresa para desenvolvimento de um ou mais projetos.
Colaborador Comum	Funcionário da empresa que não trabalha diretamente com a gestão do negócio. Exemplo: Desenvolvedor.
Colaborador Gestor	Funcionário gestor administrativo, detentor de todas as permissões de acesso e visualização do sistema.

A seguir, são apresentados os diagramas de casos de uso e descrições associadas, organizados por subsistema.

6.1 Subsistema SalesManagement

A Figura 2 apresenta o diagrama de casos de uso do subsistema SalesManagement.

A seguir, são apresentadas as descrições de cada um dos casos de uso identificados. Os casos de uso cadastrais de baixa complexidade, envolvendo inclusão, consulta, alteração e exclusão (CRUDs), são descritos na Tabela 5.

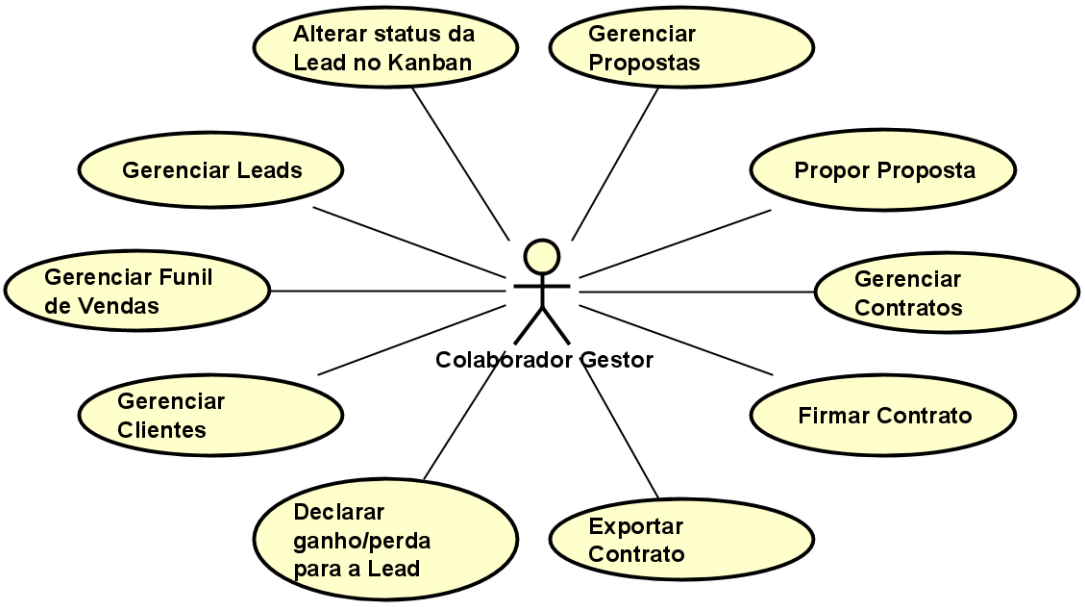


Figura 2 – Diagrama de Casos de Uso do Subsistema SalesManagement.

Tabela 5 – Caso de uso cadastrais do subsistema 01.

Id	Nome	Ações	Requisitos	Classes
UC-1	Gerenciar Funil de Vendas	(I) Informar: título e status. (C) Sem restrições. (A) Sem restrições. (E) Não é permitido excluir um *slot* do funil se houver *leads* alocados.	RF-5	FunnelSlot
UC-2	Gerenciar Leads	(I) No funil de vendas informar: nome, CPF, e-mail, telefone e celular. (C) Sem restrições. (A) Sem restrições. (E) Sem restrições.	RF-6	Lead
UC-3	Alterar status da Lead no *Kaban*	(I) Não disponível. (C) Não disponível. (A) Mover *Lead* entre os status no quadro *Kaban* de Vendas. (E) Não disponível.	RF-7	Lead

Id	Nome	Ações	Requisitos	Classes
UC-4	Gerenciar Clientes	(I) Não disponível. Vide UC-10; (A) Só o próprio cliente poderá alterar seus dados de acesso; (C) Sem restrições. (E) Um Cliente não poderá ser excluído com um projeto em andamento;	RF-10	Client
UC-5	Gerenciar Propostas	(I) Informar: descrição, data de envio e valor. (C) Sem restrições. (A) A Proposta não poderá ser alterada se o *card* que representa a *lead* no *Kaban* estiver no status de tipo Contrato. (E) A Proposta não poderá ser excluída se o *card* que representa a *lead* no *Kaban* estiver no status de tipo Contrato.	RF-11	Proposal
UC-6	Gerenciar Contratos	(I) Informar: data inicial, data final e status. (C) Sem restrições. (A) O Contrato não poderá ser alterado após ter sido declarado "Ganho" ou "Perda". (E) O Contrato não poderá ser excluído após ter sido declarado "Ganho" ou "Perda".	RF-12	Contract

Os casos de uso deste subsistema que não se encaixam na categoria acima são descritos nas páginas subsequentes.

Descrição de Caso de Uso

Projeto: GanttBox - Sistema de Gestão de Projetos
Identificador: UC-7
Nome: Propor Proposta
Descrição: Este caso de uso permite ao Colaborador Gestor, gerir as propostas dentro do funil de vendas.

Tabela 6 – Fluxos de eventos normais para o caso de uso UC-7.

Cenário	Precondição	Descrição
A *lead* aceita a proposta		1. O Gestor move a *lead* até o status de tipo "Proposta"; 2. O sistema solicita a criação de uma proposta; 3. O Gestor cria a Proposta; 4. A *lead* aceita a Proposta; 5. O Gestor move a *lead* até o status de tipo "Contrato";
A *lead* recusa a proposta		1. O Gestor move a *lead* até o status de tipo "Proposta"; 2. O sistema solicita a criação de uma proposta; 3. O Gestor cria a Proposta; 4. A *lead* recusa a Proposta; 5. O Gestor declara "Perda" para a *lead*;

Fluxos variantes:

- No cenário 1, **a Lead aceita a proposta**, a *Lead* pode num primeiro momento recusar a proposta, pedindo ajustes para negociá-la, o Gestor edita a proposta e assim consegue entrar num acordo.

Requisitos relacionados: RF-12, RF-11, RF-8

Classes relacionadas: Lead, FunnelSlot, FunnelStatus, Proposal.

Descrição de Caso de Uso

Projeto: GanttBox - Sistema de Gestão de Projetos
Identificador: UC-8
Nome: Firmar Contrato
Descrição: Este caso de uso permite ao Colaborador Gestor, oficializar um contrato.

Tabela 7 – Fluxos de eventos normais para o caso de uso UC-8.

Cenário	Precondição	Descrição
A *lead* aceita o contrato	A *lead* aceitou a proposta	1. O Gestor move a *lead* até o status de tipo "Contrato"; 2. O sistema solicita a criação de um contrato; 3. O Gestor cria o Contrato; 4. A *lead* aceita o Contrato;
A *lead* recusa o contrato	A *lead* aceitou a proposta	1. O Gestor move a *lead* até o status de tipo "Contrato"; 2. O sistema solicita a criação de um contrato; 3. O Gestor cria o Contrato; 4. A *lead* recusa o Contrato; 5. O Gestor declara "Perda" para a Lead;

Fluxos variantes:

- No cenário 1, **a Lead aceita o contrato**, a Lead pode num primeiro momento recusar o contrato, pedindo ajustes para negociá-lo, o Gestor ajusta o contrato e assim consegue entrar num acordo.

Requisitos relacionados: RF-12, RF-11, RF-8

Classes relacionadas: Lead, FunnelSlot, FunnelStatus, Contract.

Descrição de Caso de Uso

Projeto: GanttBox - Sistema de Gestão de Projetos
Identificador: UC-9
Nome: Exportar Contrato
Descrição: Este caso de uso permite ao Colaborador Gestor, exportar o Contrato para um arquivo PDF.

Tabela 8 – Fluxos de eventos normais para o caso de uso UC-9.

Cenário	Precondição	Descrição
Exportar Contrato	A *lead* aceitou a proposta	1. O Gestor move a *lead* até o status de tipo "Contrato"; 2. O sistema solicita a criação de um contrato; 3. O Gestor cria o Contrato; 4. O Gestor solicita exportar; 5. O sistema exporta o contrato para um arquivo extensão PDF;

Requisitos relacionados: RF-12, RF-13

Classes relacionadas: Lead, FunnelSlot, FunnelStatus, Contract.

Descrição de Caso de Uso

Projeto: GanttBox - Sistema de Gestão de Projetos
Identificador: UC-10
Nome: Declarar ganho/perda para *lead*
Descrição: Este caso de uso permite ao Colaborador Gestor, declarar ganho ou perda para uma *lead* que está em negociação.

Tabela 9 – Fluxos de eventos normais para o caso de uso UC-10.

Cenário	Precondição	Descrição
Declarar Ganho	A *lead* aceita o Contrato;	1. O Gestor declara "Ganho" para a *lead*; 2. O sistema notifica que irá transformar a *lead* em um Cliente; 3. O Gestor confirma a transformação; 4. O Gestor informa nome, documento, e-mail, telefone, celular e endereço; 5. O sistema gera o usuário de acesso, de acordo com os dados informados e senha automática; 6. O sistema envia por e-mail o link e os dados de acesso gerados para o primeiro acesso do cliente ao sistema; 7. O Cliente acessa o sistema; 8. O sistema recomenda através de uma mensagem a troca de senha;
Declarar Perda		1. O Gestor declara "Perda" para a *lead*;

Fluxos variantes:

- No cenário 1, **declarar ganho**, o *card* que representa a *lead* no *Kaban* de vendas, deverá passar por todo fluxo, reforçando a criação de proposta e contrato até enfim ser declarado ganho.
- No cenário 2, **declarar perda**, o *card* que representa a *lead* no *Kaban* de vendas, poderá ser movido diretamente para o status "Perda" a partir de qualquer status de origem.

Requisitos relacionados: RF-8, RF-9, RF-25, RF-10

Classes relacionadas: Lead, FunnelSlot, FunnelStatus, Contract, Client, User.

6.2 Subsistema ProjectManagement

A Figura 3 apresenta o diagrama de casos de uso do subsistema ProjectManagement.

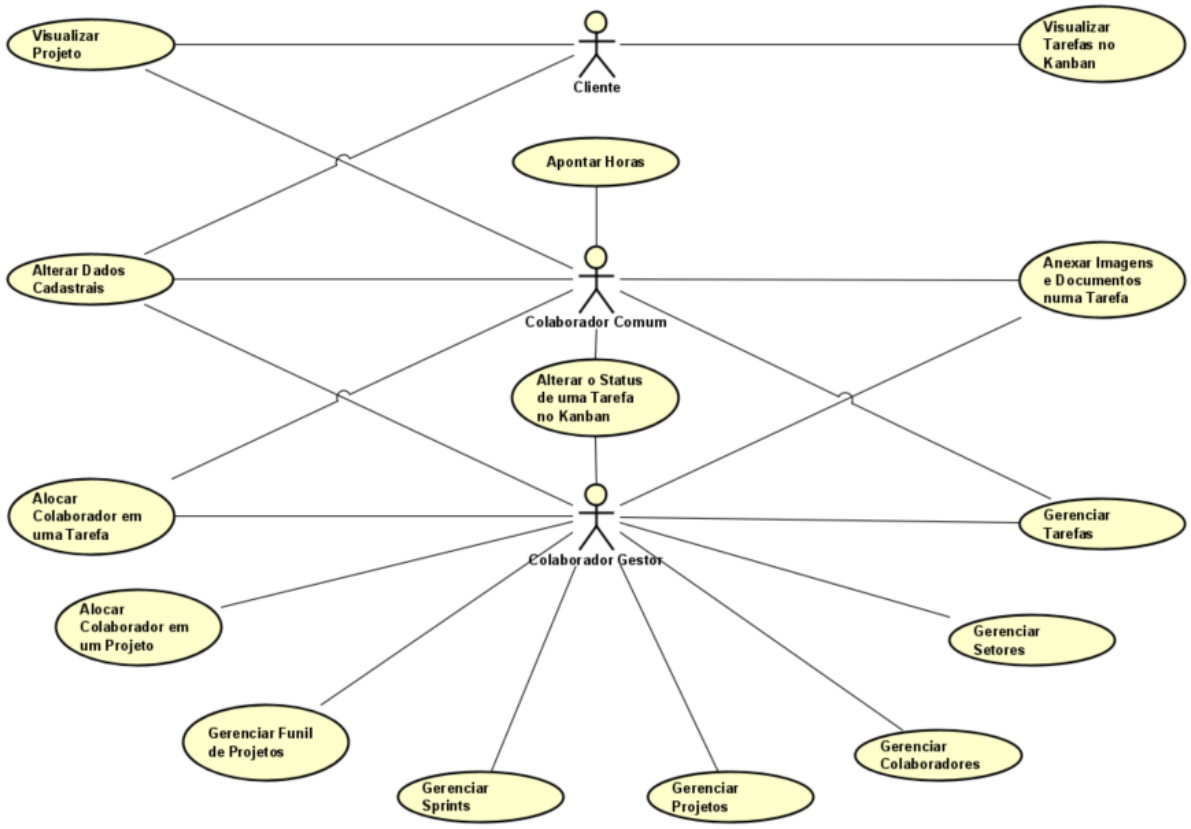


Figura 3 – Diagrama de Casos de Uso do Subsistema ProjectManagement.

A seguir, são apresentadas as descrições de cada um dos casos de uso identificados. Os casos de uso cadastrais de baixa complexidade, envolvendo inclusão, consulta, alteração e exclusão (CRUDs), são descritos na Tabela 10.

Tabela 10 – Caso de uso cadastrais do subsistema 01.

Id	Nome	Ações	Requisitos	Classes
UC-11	Alterar Dados Cadastrais	(I) Não disponível. (C) Não disponível. (A) Sem restrições. (E) Não disponível.	RF-15	Client, Collaborator, Document

Id	Nome	Ações	Requisitos	Classes
UC-12	Gerenciar Funil de Projetos	(I) Informar: título e status. (C) Sem restrições. (A) Sem restrições. (E) Não é permitido excluir um *slot* do funil se houver Tarefas alocadas.	RF-5	FunnelSlot
UC-13	Gerenciar Tarefas	(I) No funil de projetos informar nome, descrição, data inicial prevista, data final prevista. (C) Sem restrições. (A) Sem restrições. (E) A tarefa não poderá ser excluída se estiver sendo executada no momento, isto é, se um colaborador tiver dado "play" na tarefa.	RF-19	Task
UC-14	Alterar status de uma Tarefa no *Kaban*	(I) Não disponível. (C) Não disponível. (A) Mover Tarefa nos status no quadro *Kaban* do Projeto. (E) Não disponível.	RF-22	Task
UC-15	Gerenciar Setores	(I) Informar nome. (C) Sem restrições. (A) Sem restrições. (E) Sem restrições.	RF-16	Sector
UC-16	Gerenciar Projetos	(I) Informar: cliente, contrato, nome, valor total e esforço. (C) Sem restrições. (A) Não disponível. (E) Um projeto não poderá ser excluído se tiver *sprints* alocadas.	RF-17	Project

Id	Nome	Ações	Requisitos	Classes
UC-17	Gerenciar Sprints	(I) Informar: projeto, título e, opcionalmente, um subtítulo. (C) Sem restrições. (A) Sem restrições. (E) Uma *sprint* não poderá ser excluída se tiver tarefas alocadas.	RF-18	Sprint

Os casos de uso de consulta mais abrangente que as consultas a um único objeto, mas ainda de baixa complexidade, tais como consultas que combinam informações de vários objetos envolvendo filtros, estão descritos na Tabela 11.

Tabela 11 – Caso de uso de consultas envolvendo múltiplos objetos do subsistema 02.

Id	Nome	Descrição	Requisitos	Classes
UC-18	Visualizar Projeto	Um projeto pode ser visualizado na página de projetos. Espera-se ver de um projeto: nome do projeto, cliente, status, esforço e equipe.	RF-17	Project
UC-19	Visualizar Tarefas no *Kanban*	O quadro poderá ser visualizado na página do *Kanban* com base na busca do seus *slots* com nome e status, com suas devidas tarefas onde se espera ver o nome, responsável, progresso e alocadas em seus status, de suas respectivas *sprints* do respectivo projeto.	RF-17	Project, Sprint, Task, SlotFunnel, Collaborator

Os casos de uso deste subsistema que não se encaixam nas categorias acima são descritos nas páginas subsequentes.

Descrição de Caso de Uso

Projeto: GanttBox - Sistema de Gestão de Projetos
Identificador: UC-20
Nome: Gerenciar Colaboradores
Descrição: Este caso de uso permite ao Gestor Gerenciar Colaboradores comuns.

Tabela 12 – Fluxos de eventos normais para o caso de uso UC-20.

Cenário	Precondição	Descrição
Criar Colaborador		1. O Gestor acessa a página de colaboradores; 2. O Gestor solicita criar um colaborador; 3. O Gestor informa e-mail, nome, estado civil, sexo, RG, CPF, PIS-PASEP, escolaridade, número de dependentes, endereço e dados bancários; 4. O sistema gera o usuário de acesso, de acordo com os dados informados e senha automática; 5. O sistema envia por e-mail o link e os dados de acesso gerados para o primeiro acesso do Colaborador Comum ao sistema; 6. O colaborador acessa o sistema; 7. O sistema recomenda através de uma mensagem a troca de senha;
Consultar Colaborador		1. O Gestor pode consultar colaboradores na página de colaboradores, espera-se ver de um colaborador: foto, nome, e-mail e projetos alocados;
Excluir Colaborador		1. O Gestor pode excluir um colaborador a qualquer momento na página de colaboradores; 2. O sistema pede confirmação da exclusão; 3. O Gestor confirma; 4. O sistema, remove a associação de projetos e tarefas, exclui o colaborador e o usuário associado do banco de dados;

Requisitos relacionados: RF-14, RF-25

Classes relacionadas: Collaborator, User, Document.

Descrição de Caso de Uso

Projeto: GanttBox - Sistema de Gestão de Projetos
Identificador: UC-21
Nome: Alocar Colaborador em um Projeto
Descrição: Este caso de uso permite ao Gestor alocar um Colaborador da empresa em um projeto.

Tabela 13 – Fluxos de eventos normais para o caso de uso UC-21.

Cenário	Precondição	Descrição
Alocar Colaborador ao Projeto		1. O Gestor acessa o Kanban do projeto; 2. O Gestor seleciona um Colaborador dentre todos cadastrados; 3. O Gestor confirma a alocação; 4. O sistema vincula o colaborador ao projeto;

Requisitos relacionados: RF-14, RF-17, RF-20

Classes relacionadas: Collaborator, Project.

Descrição de Caso de Uso

Projeto: GanttBox - Sistema de Gestão de Projetos
Identificador: UC-22
Nome: Alocar Colaborador em uma Tarefa
Descrição: Este caso de uso permite a um colaborador alocar outro colaborador a uma tarefa, isto é, fazer com que o outro colaborador selecionado seja responsável por aquela específica tarefa.

Tabela 14 – Fluxos de eventos normais para o caso de uso UC-22.

Cenário	Precondição	Descrição
Alocar Colaborador ao Projeto		1. O Colaborador acessa o Kanban do projeto; 2. O Colaborador seleciona um Colaborador dentre todos cadastrados; 3. O Colaborador confirma a alocação; 4. O sistema vincula o colaborador a tarefa;

Requisitos relacionados: RF-14, RF-19, RF-21

Classes relacionadas: Collaborator, Task.

Descrição de Caso de Uso

Projeto: GanttBox - Sistema de Gestão de Projetos
Identificador: UC-23
Nome: Anexar Imagens e Documentos numa Tarefa
Descrição: Este caso de uso permite ao Colaborador anexar imagens e documentos que sejam necessários para a tarefa.

Tabela 15 – Fluxos de eventos normais para o caso de uso UC-23.

Cenário	Precondição	Descrição
Anexar imagens e documentos		1. O Colaborador cria a tarefa; 2. O Colaborador anexa imagem/documento; 3. O sistema salva o arquivo em um bucket S3(Amazon).

Requisitos relacionados: RF-24

Classes relacionadas: Collaborator, Task, Document.

Descrição de Caso de Uso

Projeto: GanttBox - Sistema de Gestão de Projetos
Identificador: UC-24
Nome: Apontar Horas
Descrição: Este caso de uso permite ao Colaborador dar play/stop nas tarefas, apontando suas horas trabalhadas no projeto.

Tabela 16 – Fluxos de eventos normais para o caso de uso UC-24.

Cenário	Precondição	Descrição
Dar play/stop na Terefa	A tarefa estar num status a frente do tipo "Backlog"	1. O Colaborador cria a tarefa; 2. O sistema cria a tarefa com status default do tipo "Backlog"; 3. O Colaborador avança a tarefa para um status de tipo posterior; 4. O Colaborador da "play" na Tarefa; 5. O Colaborador da "stop" na Tarefa; 6. O sistema calcula e registra as horas trabalhadas daquele colaborador no respectivo projeto.

Requisitos relacionados: RF-23

Classes relacionadas: Collaborator, Task.

6.3 Subsistema UserManagement

A Figura 4 apresenta o diagrama de casos de uso do subsistema UserManagement.

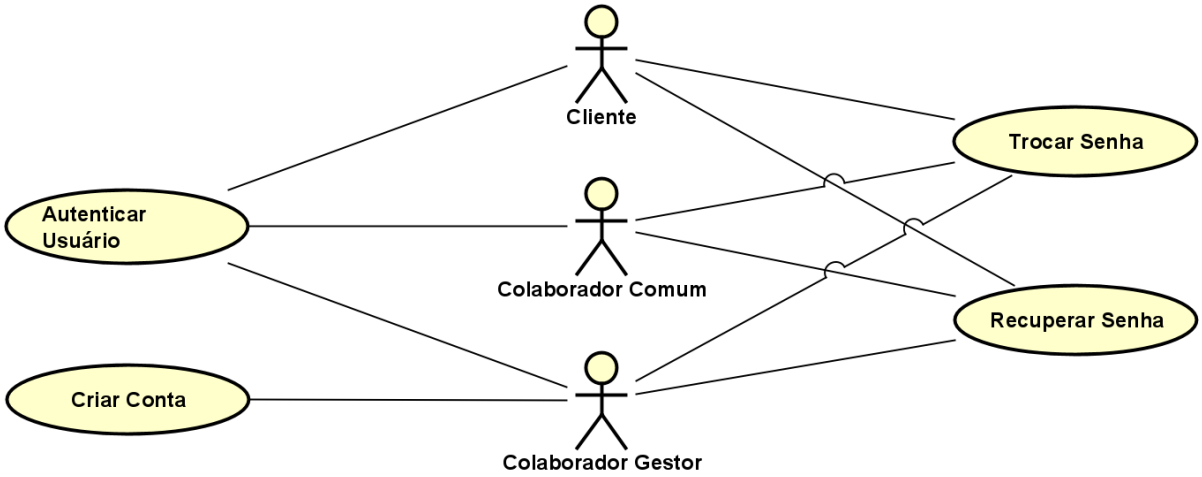


Figura 4 – Diagrama de Casos de Uso do Subsistema UserManagement.

Os casos de uso deste subsistema estão descritos nas páginas subsequentes.

Descrição de Caso de Uso

Projeto: GanttBox - Sistema de Gestão de Projetos
Identificador: UC-25
Nome: Criar Conta
Descrição: Este caso de uso uso permite a um visitante criar uma conta e acessar o sistema pela primeira vez.

Tabela 17 – Fluxos de eventos normais para o caso de uso UC-25.

Cenário	Precondição	Descrição
Criar Conta		1. O visitante, futuro ator Gestor, informa os seus dados básicos de cadastro de colaborador: e-mail, nome, estado civil, sexo, nome do pai, nome da mãe, cônjuge, número de dependentes, endereço, dados bancários; 2. O visitante, futuro ator Gestor, informa os seus dados básicos da sua empresa: nome, cnpj, telefone comercial e e-mail; 3. O sistema gera o código da empresa e cria um novo esquema no banco de dados com múltiplos conjuntos de tabelas, para segregar os dados de outras instituições; 4. O sistema gera o usuário de acesso, de acordo com os dados informados e senha automática; 5. O sistema envia por e-mail o link e os dados de acesso gerados para o primeiro acesso do Gestor ao sistema; 6. O Gestor acessa o sistema; 7. O sistema recomenda através de uma mensagem a troca de senha;

Requisitos relacionados: RF-1, RF-25

Classes relacionadas: Collaborator, User.

Descrição de Caso de Uso

Projeto: GanttBox - Sistema de Gestão de Projetos
Identificador: UC-26
Nome: Autenticar Usuário
Descrição: Este caso de uso permite ao usuário logar no sistema entrando seus dados de acesso.

Tabela 18 – Fluxos de eventos normais para o caso de uso UC-26.

Cenário	Precondição	Descrição
Acessou o sistema	O usuário está cadastrado no sistema	1. O usuário acessa a página de login do sistema; 2. O usuário entra com seu nome de usuário e senha. 3. O sistema certifica que as informações de login estão corretas; 4. O sistema autentica o usuário, e o redireciona para a Home Page; 5. O usuário acessa o sistema;
Não acessou o sistema		1. O usuário acessa a página de login do sistema; 2. O usuário entra com seu nome suário e senha. 3. O sistema certifica que as informações de login estão incorretas; 4. O sistema exibe uma mensagem de erro para o usuário, informando que seus dados de acesso estão incorretos; 5. O usuário não acessa o sistema;

Requisitos relacionados: RF-2

Classes relacionadas: Client, Collaborator, User.

Descrição de Caso de Uso

Projeto: GanttBox - Sistema de Gestão de Projetos
Identificador: UC-27
Nome: Recuperar senha
Descrição: Este caso de uso permite ao usuário recuperar a senha que tenha sido esquecida.

Tabela 19 – Fluxos de eventos normais para o caso de uso UC-27.

Cenário	Precondição	Descrição
Recuperar a Senha	O usuário está cadastrado no sistema	1. O usuário acessa a página de esqueceu a senha do sistema; 2. O usuário entra com seu e-mail de cadastro. 3. O sistema envia um e-mail com um link para cadastro de uma nova senha; 4. O usuário acessa o link, entra com sua nova senha e confirma sua nova senha; 5. O usuário recupera a senha;

Requisitos relacionados: RF-3, RF-25

Classes relacionadas: Client, Collaborator, User.

Descrição de Caso de Uso

Projeto: GanttBox - Sistema de Gestão de Projetos
Identificador: UC-28
Nome: Trocar senha
Descrição: Este caso de uso permite ao usuário trocar sua senha de acesso ao sistema.

Tabela 20 – Fluxos de eventos normais para o caso de uso UC-28.

Cenário	Precondição	Descrição
Trocar a Senha	O usuário está cadastrado no sistema	1. O usuário efetua login sistema; 2. O usuário acessa a página de trocar senha no sistema. 3. O usuário entra com sua nova senha e confirma sua nova senha; 4. O usuário troca a senha;

Requisitos relacionados: RF-4, RF-25

Classes relacionadas: Client, Collaborator, User.

7 Modelo Estrutural

O modelo conceitual estrutural visa capturar e descrever as informações (classes, associações e atributos) que o sistema deve representar para prover as funcionalidades descritas nos casos de uso especificados na Seção 6.

A seguir, são apresentados os diagramas de classes de cada um dos subsistemas identificados no contexto deste projeto. Na Seção 8 — Dicionário de Projeto — são apresentadas as descrições das classes, atributos e operações presentes nos diagramas apresentados nesta seção.

7.1 Subsistema SalesManagement

A Figura 5 apresenta o diagrama de classes do Subsistema SalesManagement.

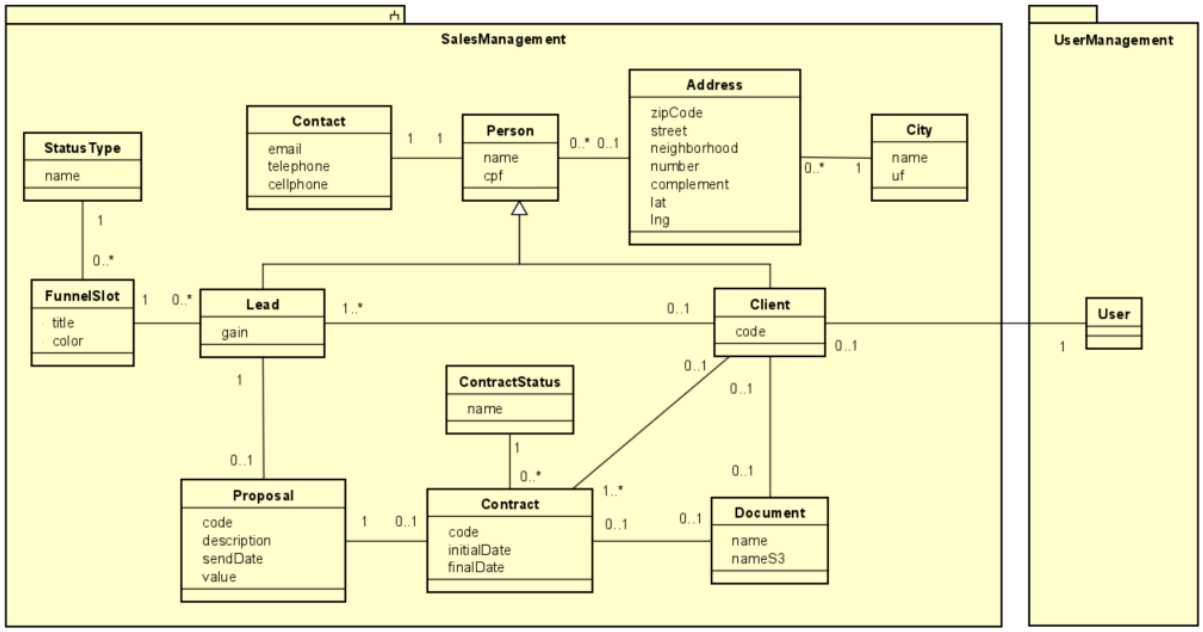


Figura 5 – Diagrama de classes do Subsistema SalesManagement.

7.2 Subsistema ProjectManagement

A Figura 6 apresenta o diagrama de classes do Subsistema ProjectManagement.

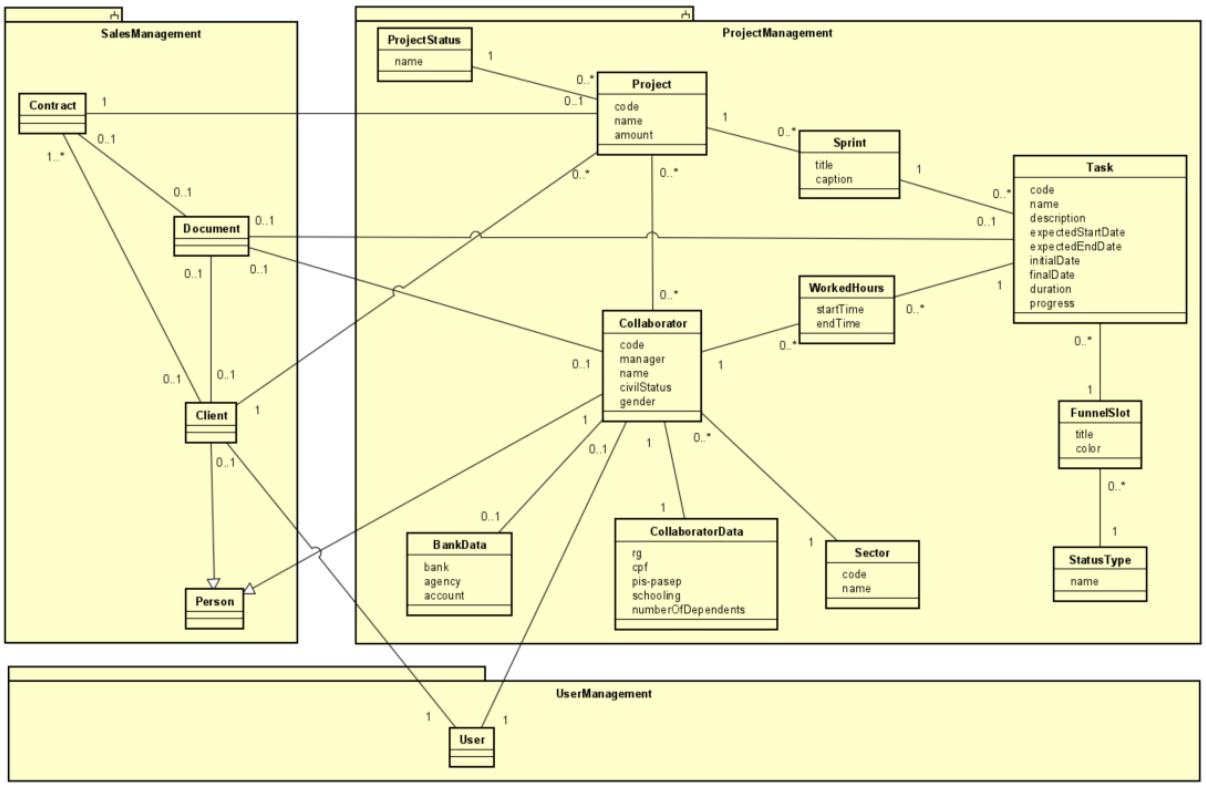


Figura 6 – Diagrama de classes do Subsistema ProjectManagement.

7.3 Subsistema UserManagement

A Figura 7 apresenta o diagrama de classes do Subsistema UserManagement.

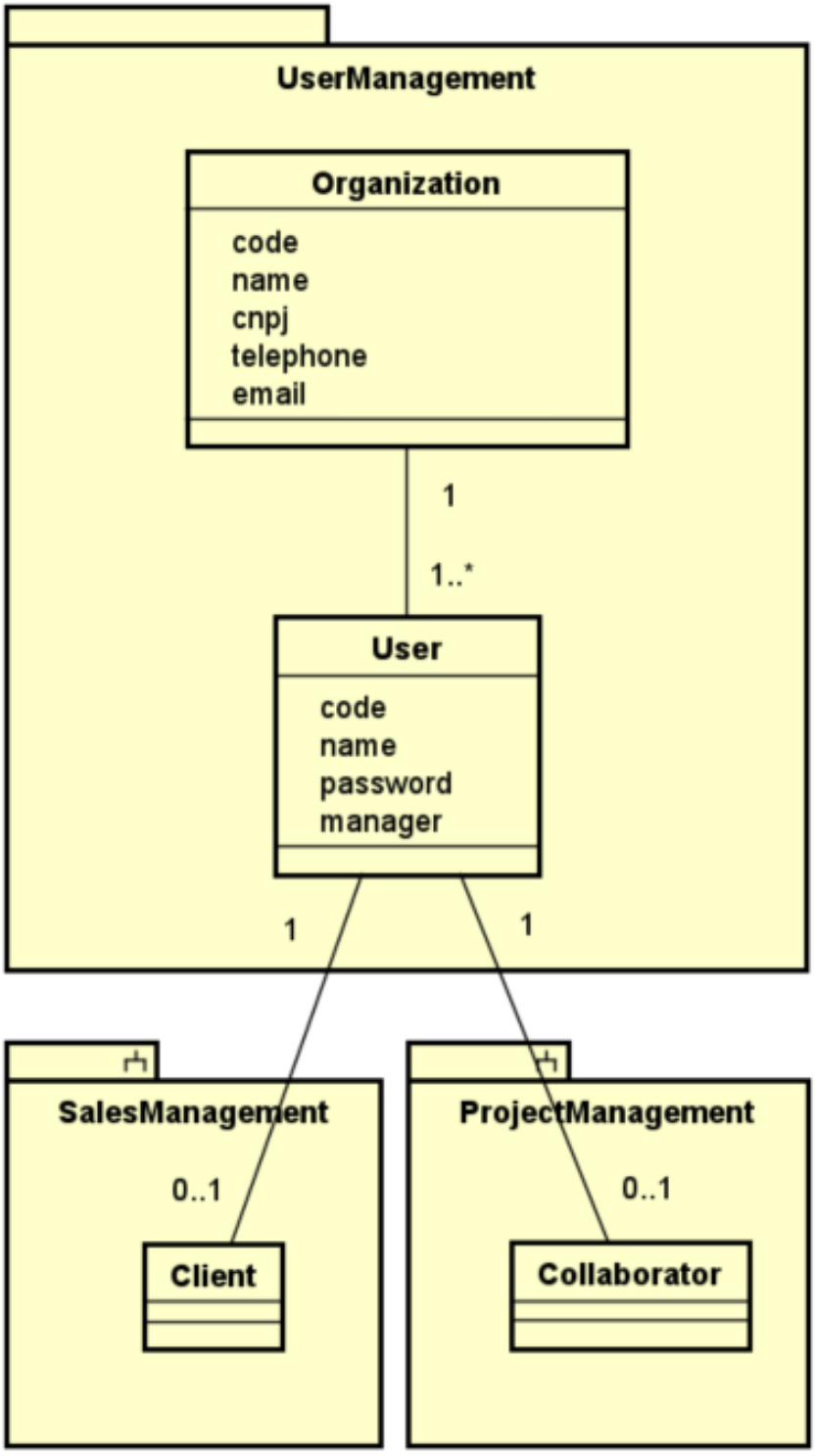


Figura 7 – Diagrama de classes do Subsistema UserManagement.

8 Dicionário de Projeto

Esta seção apresenta as definições detalhadas das classes, descrevendo seus atributos e associações e servindo como um glossário do projeto. As definições são organizadas por subsistema, cada classe sendo apresentada em uma tabela separada. A coluna "Obr.?" indica com um "x" se o atributo é obrigatório (deve possuir um valor para se criar um objeto da classe).

Vale destacar que eventuais operações que estas classes vierem a ter não são listadas e descritas nesta fase do projeto. Além disso, na Seção 7, algumas classes podem ser incluídas nos diagramas de outros subsistemas para ilustrar a relação entre eles. No dicionário de projeto, no entanto, classes são descritas apenas em seus subsistemas de origem.

8.1 Subsistema SalesManagement

Tabela 21 – Detalhamento da classe *Lead*.

Propriedade	**Tipo**	**Obr.?**	**Descrição**
name	String	x	Nome da Lead.
cpf	String	x	Documento CPF da Lead.
gain	Booleano		Indicador de ganho.

Tabela 22 – Detalhamento da classe *FunnelSlot*.

Propriedade	**Tipo**	**Obr.?**	**Descrição**
title	String	x	Título do Slot do Funil.
color	String	x	Cor do Slot do Funil.

Tabela 23 – Detalhamento da classe *StatusType*.

Propriedade	**Tipo**	**Obr.?**	**Descrição**
name	String	x	Nome do Tipo do Status.

Tabela 24 – Detalhamento da classe *Proposal*.

Propriedade	**Tipo**	**Obr.?**	**Descrição**
code	String	x	Código da Proposta.
description	String	x	Descrição da Proposta.
sendDate	Data	x	Data de envio da Proposta.

Propriedade	Tipo	Obr.?	Descrição
value	Número Real	x	valor da Proposta.

Tabela 25 – Detalhamento da classe *Contract*.

Propriedade	Tipo	Obr.?	Descrição
code	String	x	Código do Contrato.
initialDate	Data	x	Data inicial do Contrato.
finalDate	Data	x	Data final do Contrato.

Tabela 26 – Detalhamento da classe *ContractStatus*.

Propriedade	Tipo	Obr.?	Descrição
name	String	x	Nome do Status do Contrato.

Tabela 27 – Detalhamento da classe *Document*.

Propriedade	Tipo	Obr.?	Descrição
name	String	x	Nome da documento.
nameS3	String	x	Nome do documento na nuvem.

Tabela 28 – Detalhamento da classe *Client*.

Propriedade	Tipo	Obr.?	Descrição
code	String	x	Código do Cliente.
name	String	x	Nome do Cliente.
cpf	String	x	Documento CPF do Cliente.

Tabela 29 – Detalhamento da classe *Address*.

Propriedade	Tipo	Obr.?	Descrição
zipCode	String	x	CEP do Endereço.
street	String	x	Rua do Endereço.
neighborhood	String	x	Bairro do Endereço.
number	Inteiro	x	Número do Endereço.
complement	String	x	Complemento do Endereço.
lat	Número Real	x	Latitude do Endereço.
lng	Número Real	x	Longitude do Endereço.

Tabela 30 – Detalhamento da classe *Contact*.

Propriedade	**Tipo**	**Obr.?**	**Descrição**
email	String	x	E-mail do Conato.
telephone	String	x	Telefone do Contato.
cellphone	Booleano	x	Celular do Contato.

Tabela 31 – Detalhamento da classe *City*.

Propriedade	**Tipo**	**Obr.?**	**Descrição**
name	String	x	Nome da Cidade.
uf	String	x	Estado que a cidade pertence.

8.2 Subsistema ProjectManagement

Tabela 32 – Detalhamento da classe *Project*.

Propriedade	**Tipo**	**Obr.?**	**Descrição**
code	String	x	Código do Projeto.
name	String	x	Nome do Projeto.
amount	Valor Real	x	Valor do Projeto.

Tabela 33 – Detalhamento da classe *Sprint*.

Propriedade	**Tipo**	**Obr.?**	**Descrição**
title	String	x	Título da Sprint.
caption	String		Subtítulo da Sprint.

Tabela 34 – Detalhamento da classe *Task*.

Propriedade	**Tipo**	**Obr.?**	**Descrição**
code	String	x	Código da Tarefa.
name	String	x	Nome da Tarefa.
description	String	x	Descrição da Tarefa.
expectedStartDate	Data	x	Data Inicial prevista.
expectedEndDate	Data	x	Data Final prevista.
initialDate	Data		Data Inicial da Tarefa.
finalDate	Data		Data Final da Tarefa.
duration	String		Duração da Tarefa.
progress	String		Progresso da Tarefa.

Propriedade	Tipo	Obr.?	Descrição

Tabela 35 – Detalhamento da classe *Tag*.

Propriedade	Tipo	Obr.?	Descrição
name	String	x	Nome da Tag.
color	String		Cor da Tag.

Tabela 36 – Detalhamento da classe *ProjectStatus*.

Propriedade	Tipo	Obr.?	Descrição
name	String	x	Nome do Status do Projeto.

Tabela 37 – Detalhamento da classe *Question*.

Propriedade	Tipo	Obr.?	Descrição
exam	String	x	Questão formulada.

Tabela 38 – Detalhamento da classe *Answer*.

Propriedade	Tipo	Obr.?	Descrição
name	String	x	Opção de Resposta.

Tabela 39 – Detalhamento da classe *Collaborator*.

Propriedade	Tipo	Obr.?	Descrição
code	String	x	Código do Colaborador.
manager	Booleano	x	Identificador de Colaborador Gestor.
name	String	x	Nome do Colaborador.
fatherName	String	x	Nome do pai do Colaborador.
motherName	String	x	Nome da mãe do Colaborador.
civilStatus	String	x	Staus civil do Colaborador.
gender	String	x	Gênero do Colaborador.

Tabela 40 – Detalhamento da classe *CollaboratorData*.

Propriedade	Tipo	Obr.?	Descrição
rg	String	x	RG do Colaborador.
cpf	String	x	CPF do Colaborador.
pis-paesp	String	x	PIS-PASEP do Colaborador.

Propriedade	Tipo	Obr.?	Descrição
schooling	String	x	Escolaridade do Colaborador.
numberOfDependents	Inteiro	x	Número de dependentes do Colaborador.

Tabela 41 – Detalhamento da classe *WorkedHours*.

Propriedade	Tipo	Obr.?	Descrição
startTime	Data	x	Momento Inicial do trabalho do colaborador na tarefa.
endTime	Data		Momento Final do trabalho do colaborador na tarefa.

Tabela 42 – Detalhamento da classe *BankData*.

Propriedade	Tipo	Obr.?	Descrição
bank	String	x	Banco.
agency	String	x	Número da Agência.
account	String	x	Número da conta.

Tabela 43 – Detalhamento da classe *Sector*.

Propriedade	Tipo	Obr.?	Descrição
code	String	x	Código do Setor.
name	String	x	Nome do Setor.

8.3 Subsistema UserManagement

Tabela 44 – Detalhamento da classe *User*.

Propriedade	Tipo	Obr.?	Descrição
code	String	x	Código do Usuário.
name	String	x	Nome do Usuário.
passwword	String	x	Senha do Usuário.
admin	Booleano		Indicador de Usuário Gestor.

Tabela 45 – Detalhamento da classe *Organization*.

Propriedade	Tipo	Obr.?	Descrição
code	String	x	Código da Empresa.
name	String	x	Nome da Empresa.

Propriedade	**Tipo**	**Obr.?**	**Descrição**
cnpj	String	x	Documento CNPJ da Empresa.
telephone	String	x	Telefone da Empresa.
email	String	x	E-mail da Empresa.

Documento de Projeto de Sistema

GanttBox - Sistema de Gestão de Projetos

Registro de Alterações:

Versão	Responsável	Data	Alterações
1.0	Wallace Krüger Junior	23/01/2022	Versão Inicial
1.1	Vitor E. Silva Souza	09/03/2022	Primeira Revisão
1.2	Wallace Krüger Junior	12/03/2022	Ajustes da primeira Revisão
1.3	Vitor E. Silva Souza	14/03/2022	Segunda Revisão
1.4	Wallace Krüger Junior	15/03/2022	Ajustes da segunda Revisão

Vitória, ES

2022

1 Introdução

Este documento apresenta o projeto (*design*) do sistema *GanttBox*. O sistema que visa estruturar uma perspectiva única para o Gestor, tornando a vida administrativa mais simples e prática, elucidando e transparecendo conhecimento do seu negócio.

Além desta introdução, este documento está organizado da seguinte forma: a Seção 2 apresenta a plataforma de software utilizada na implementação do sistema; a Seção 3 apresenta a especificação dos requisitos não funcionais (atributos de qualidade), definindo as táticas e o tratamento a serem dados aos atributos de qualidade considerados condutores da arquitetura; a Seção 4 apresenta a arquitetura de software; por fim, a Seção 5 apresenta os modelos FrameWeb que descrevem os componentes da arquitetura.

2 Plataforma de Desenvolvimento

Na Tabela 1 são listadas as tecnologias utilizadas no desenvolvimento da ferramenta, bem como o propósito de sua utilização.

Tabela 1 – Plataforma de Desenvolvimento e Tecnologias Utilizadas.

Tecnologia	**Versão**	**Descrição**	**Propósito**
Angular	13.2.1	Plataforma *front-end*, usada para construção da interface da aplicação, utilizando HTML, CSS e JavasScript.	Ganho de produtividade no desenvolvimento e otimização da interatividade do sistema, construindo *Single-Page Applications* com qualidade.
Node.js	17.4.0	Plataforma *back-end*, usada para construção de aplicações *server-side*, utilizando JavaScript.	Utilizar a mesma linguagem no lado do cliente e servidor, construindo aplicações de alta escalabilidade.
JavaScript	8	Linguagem de programação interpretada para páginas Web.	Tornar páginas Web dinâmicas e interativas.
TypeScript	4.5.2	Linguagem de programação superset de JavaScript utilizada para páginas Web.	Super conjunto JavaScript, adicionando funcionalidades que reduzem o esforço no desenvolvimento, com tipagem de dados e orientação a objetos.
HTML	5	Linguagem de marcação para páginas Web.	Criação de páginas Web.
CSS	3	Linguagem utilizada para estilização de páginas Web.	Permitir criar páginas responsivas e esteticamente amigáveis para o usuário.
Npm	8.3.1	Gerenciador de pacotes do Node.	Permitir a instalação de módulos e bibliotecas de forma fácil e ágil.
Yarn	1.22.4	Gerenciador de pacotes.	Permitir a instalação de módulos e bibliotecas de forma fácil e ágil.

Tecnologia	Versão	Descrição	Propósito
Express.js	4.17.2	*Framework* para construção de páginas Web e APIs.	Fornecer um conjunto robusto de ferramentas facilitando o desenvolvimento.
TypeORM	0.2.41	ORM (*Object-Relational Mapping*) para Node, intermediador entre a API e base de dados.	Simplicidade e agilidade nas operações.
Bootstrap	5.1.3	*Framework* para construção de interfaces de usuário.	Fornecer um conjunto de bibliotecas CSS com integração JavaScript, aumentando produtividade no desenvolvimento de páginas com design amigável e responsivo.
Syncfusion	19.4.52	*Framework* para construção de interfaces de usuário.	Fornecer um conjunto de bibliotecas CSS com integração JavaScript, aumentando produtividade no desenvolvimento de páginas com design amigável e responsivo.
PostgreSQL	14.2	Sistema Gerenciador de Banco de Dados Relacional.	Armazenamento de dados de forma segura e otimizada.
DBeaver	21.3.3	Ferramenta de Gestão de Banco de Dados	Robusta e intuitiva facilitando manipulação do banco de dados.
Docker	5	Plataforma de gerenciamentos de containers.	Garantir similaridade independentemente de ambiente de execução.

Na Tabela 2 vemos os softwares que apoiaram o desenvolvimento de documentos e também do código fonte.

Tabela 2 – Softwares de Apoio ao Desenvolvimento do Projeto

Tecnologia	Versão	Descrição	Propósito
Visual Studio Code	1.63	Ambiente de desenvolvimento (IDE) com suporte ao desenvolvimento TypeScript.	Implementação do código.
Astah UML	8.4.1	Ferramenta de modelagem UML	Criação dos diagramas na fase de modelagem do sistema.
Overleaf		Editor de texto online para escrita de documentos LaTeX.	Escrita da documentação do software e monografia.

3 Requisitos Não Funcionais

A Tabela 3 apresenta a especificação dos requisitos não funcionais identificados no Documento de Especificação de Requisitos.

Tabela 3 – Especificação de Requisitos Não Funcionais.

RNF-1 – A aplicação deve estar disponível pela Internet, com compatibilidade aos principais navegadores disponíveis no mercado.	
Categoria:	Portabilidade
Tática / Tratamento:	Utilizar módulos e bibliotecas capazes de prover as devidas funcionalidades, independentes da forma de renderização do navegador; utilizar conversores de código de estilização online para diferentes navegadores; utilizar uma estrutura de arquitetura limpa, inspirada no padrão MVC (*Model, View, Controller*).
Medida:	Diferenças de *layout* de tela e comportamento do sistema entre os 3 navegadores, identificadas pelos desenvolvedores nas fases de implementação e manutenção do sistema e pelo usuários na utilização do mesmo.
Critério de Aceitação:	Nenhuma diferença significativa (que impeça o uso do sistema por parte dos usuários).

RNF-2 – Uso de Design responsivo nas interfaces gráficas.	
Categoria:	Usabilidade
Tática / Tratamento:	Utilizar módulos e bibliotecas com recursos nativamente adaptáveis aos diferentes tamanhos de tela, implementar estilização (CSS) para diferentes tamanhos de tela.
Medida:	Aplicação de pesquisa de satisfação com grupos de 10 a 20 usuários do sistema, indicados pelos gestores das diferentes empresas clientes. Os grupos serão divididos por diferentes dispositivos, com diferentes resoluções e tamanhos de tela, com a finalidade de atribuírem notas para as diferentes funcionalidades e interfaces entregues pelo sistema para aquele formato.
Critério de Aceitação:	Média das avaliações na pesquisa de satisfação igual ou superior a 90%.

RNF-3 – O sistema deve ser fácil de manter, de modo a acomodar novas funcionalidades ou até mesmo adaptações para organizações específicas.	
Categoria:	Manutenibilidade
Tática / Tratamento:	Uso das boas práticas de programação orientados a objetos, inspiradas no SOLID; uso de uma arquitetura limpa e escalável; arquitetura *multi-tenant*.
Medida:	Quantidade de *bugs* e *code smells*, identificados por ferramentas que monitoram a qualidade do software, antes de cada *deploy*, durante toda a "vida" da aplicação.
Critério de Aceitação:	Inexistência de bugs e code smells.

RNF-4 – O sistema deve controlar o acesso às funcionalidades de gerenciamento por meio de autenticação e autorização.	
Categoria:	Segurança
Tática / Tratamento:	Identificar usuários por meio de nome do usuário e senhas criptografadas; identificar usuários logados por *tokens* expiráveis nos cabeçalhos das requisições.
Medida:	Testes de stress de injeção de conteúdos maliciosos e bots.
Critério de Aceitação:	Não quebrar para os testes realizados.

RNF-5 – O sistema segregará os dados por diferentes organizações.	
Categoria:	Confiabilidade

Tática / Tratamento:	Arquitetura *multi-tenant*.
Medida:	Completude, acurácia, consistência e validade dos dados, percebidos por testes de stress com injeção de dados intencionalmente projetados com inconsistências.
Critério de Aceitação:	Recuperação do sistema e do banco de dados em transações má sucedidas ocasionadas por dados contestáveis e também consistência dos dados com completude e acurácia para informações validas.

RNF-6 – O sistema deve garantir a consistência de dados, obrigando o usuário a preencher todos os campos essenciais na hora de inserir dados.	
Categoria:	Confiabilidade.
Tática / Tratamento:	Validar o preenchimento dos formulários, garantindo preenchimento de campos obrigatórios e suas específicas validações.
Medida:	Completude, acurácia, consistência e validade dos dados, percebidos por testes de stress com injeção de dados intencionalmente projetados com inconsistências.
Critério de Aceitação:	Dados válidos consistidos sem apresentar erros e retorno de respostas claras para casos de injeção de dados incoerentes, elucidando para o usuário, a inconsistência no preenchimento de um formulário.

RNF-7 – O sistema deve ter alta disponibilidade, superior a 98% do tempo.	
Categoria:	Disponibilidade.
Tática / Tratamento:	Utilizar uma arquitetura limpa e bem mapeada para reagir operando em casos extraordinários.
Medida:	Tempo de disponibilidade da aplicação na *web*, medido por softwares de monitoramento.
Critério de Aceitação:	Disponibilidade superior a 98% do tempo.

RNF-8 – O sistema deve possuir mecanismos para se recuperar de falhas e continuar operacional.	
Categoria:	Recuperabilidade.
Tática / Tratamento:	Utilizar ferramentas que permitam a recuperação de falhas.
Medida:	Tempo de recuperação do sistema em caso de falhas remetentes a infraestrutura de banco de dados e servidor, capturado através da utilização de ferramentas de monitoramento e alarme.
Critério de Aceitação:	Tempo de recuperação de falhas inferior a 2 segundos.

RNF-9 – O sistema deve se comunicar com os serviços da Amazon S3.	
Categoria:	Interoperabilidade
Tática / Tratamento:	Criação do *bucket* no S3 comunicando via requisições HTTP, conforme documentação da Amazon.
Medida:	Número de requisições bem sucedidas, verificadas através de gráficos de utilização dos servidores disponibilizados pela própria Amazon.
Critério de Aceitação:	Número de requisições bem sucedidas superior a 99%.

RNF-10 – O sistema deverá processar requisições paralelamente em tempo hábil.	
Categoria:	Eficiência.
Tática / Tratamento:	O sistema deve utilizar mecanismos para otimizar as requisições HTTP.
Medida:	Tempo de resposta das requisições em horas do dia que a aplicação seja mais usada, medido por ferramentas de monitoramentos e análises de dados.
Critério de Aceitação:	Obter tempo de resposta inferior a 25% a mais das respectivas requisições em horas de baixa utilização da aplicação.

4 Arquitetura de Software

A seguir, serão apresentadas as arquiteturas de software que foram escolhidas para o desenvolvimento do sistema GanttBox.

4.1 Clean Architecture

A arquitetura de software do sistema de gestão de projetos baseia-se primordialmente na *Clean Architecture*, ou Arquitetura Limpa, proposta idealmente por Robert Cecil Martin (MARTIN; GRENNING; BROWN, 2018). A Figura 1 mostra a arquitetura do sistema.

O núcleo, camada mais interna do diagrama, é a camada de Entidades (*Entities*), as classes modelos, que representam as tabelas ou *collections* do banco de dados dentro do projeto. Numa camada acima, de comunicação direta com a entidades, observamos os Casos de Uso (*Use Cases*), onde se encontram todas as regras de negócio da aplicação. Em seguida, a camada dos Adaptadores (*Adapters*), esta camada é a camada dos controladores, que se comunicam com as requisições advindas dos elementos externos. Concluímos com a camada mais externa, *frameworks* externos, onde se encontram todos os elementos externos que necessitam se comunicar com a nossa camada de entidades, como bancos de dados, *frameworks* e interfaces.

4.2 Multitenancy

Multitenant ou *multitenancy* é uma arquitetura de software projetada para que uma única instância da aplicação centralizada, seja executada em um servidor e que atenda vários clientes. Assumindo esta arquitetura como mais adequada para Saas (*Software as a Service*), caímos na questão da segregação dos dados, como existirá uma única instância

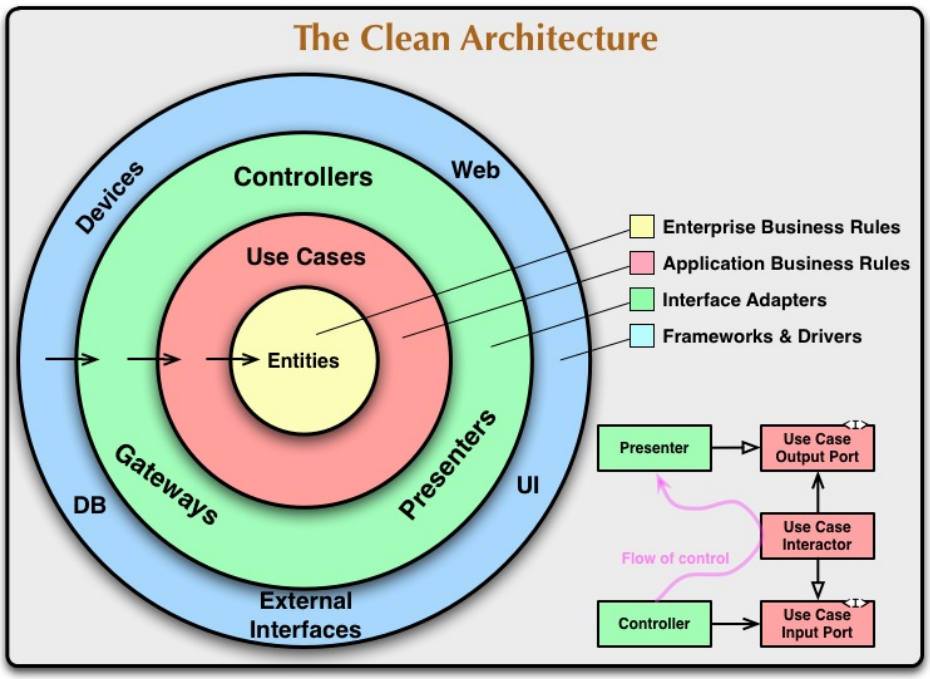


Figura 1 – Arquitetura proposta pela *Clean Architecture* (MARTIN; GRENNING; BROWN, 2018).

de software, precisamos separar os dados dos diferentes *tenants* da nossa aplicação. A abordagem que optou-se para o desenvolvimento do sistema de gestão, foi a segregação dos dados via esquemas diferentes. Nesta estratégia, existe uma única instância de banco de dados, mas essa instância possui múltiplos conjuntos de tabelas separados por cada *tenant*. Na aplicação que está sendo proposta nesse texto, toda vez que um cliente criar uma conta como representação de uma instituição, será criado um novo esquema de banco dados exatamente com a mesma modelagem de tabelas. Esta estratégia é mais barata, pois se usará uma única instância de banco e ainda garantirá o isolamento lógico dos dados de cada cliente por esquemas.

5 Modelagem FrameWeb

Além das arquiteturas de software pontuadas na seção anterior, o *GanttBox* é um sistema Web cuja arquitetura utiliza *frameworks* comuns no desenvolvimento para esta plataforma. Desta forma, o sistema pode ser modelado utilizando a abordagem FrameWeb (SOUZA, 2007).

A Tabela 4 indica os *frameworks* presentes na arquitetura do sistema que se encaixam em cada uma das categorias de *frameworks* que FrameWeb dá suporte.

Tabela 4 – *Frameworks* da arquitetura do sistema separados por categoria.

Categoria de *Framework*	***Framework* Utilizado**
Decorador	Bootstrap e Syncfusion
Controlador Frontal	Node.js e Express
Mapeamento Objeto/Relacional	TypeORM
Segurança	JWT

5.1 Camada de Negócio

A camada central da arquitetura, a camada de negócio, faz referência as 2 camadas mais internas na arquitetura escolhida para o GanttBox (*Clean Architecture*), com as Entidades representadas nas figuras 2, 3 e 4, que são os Modelos de Entidades do FrameWeb para este projeto.

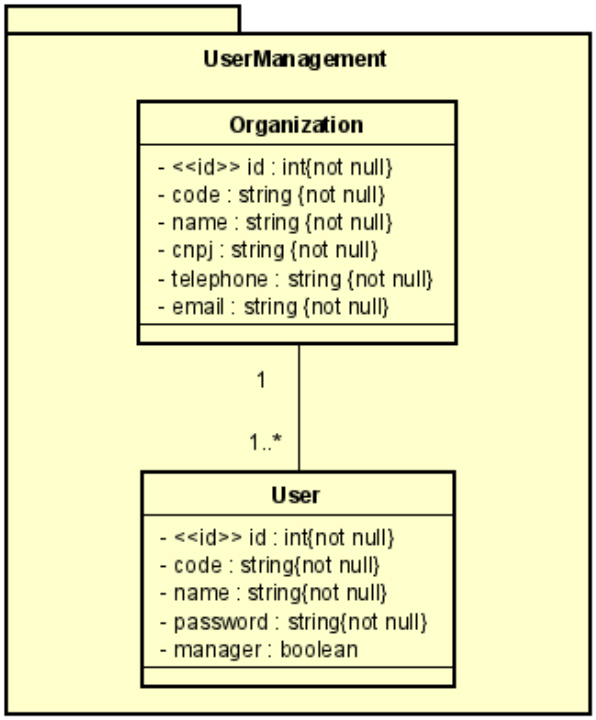


Figura 2 – Modelo de Entidades do Subsistema UserManagement.

Para que o trabalho pudesse ser concluído no prazo, foi reduzido o escopo da documentação para demonstrar os outros modelos UML propostos pelo FrameWeb, dessa forma, foram escolhidos subsistemas específicos, ou parte destes subsistemas, simplificando e reduzindo o número de diagramas.

As figuras 5 e 6 representam os Modelos de Aplicação do Subsistema *UserManagement*.

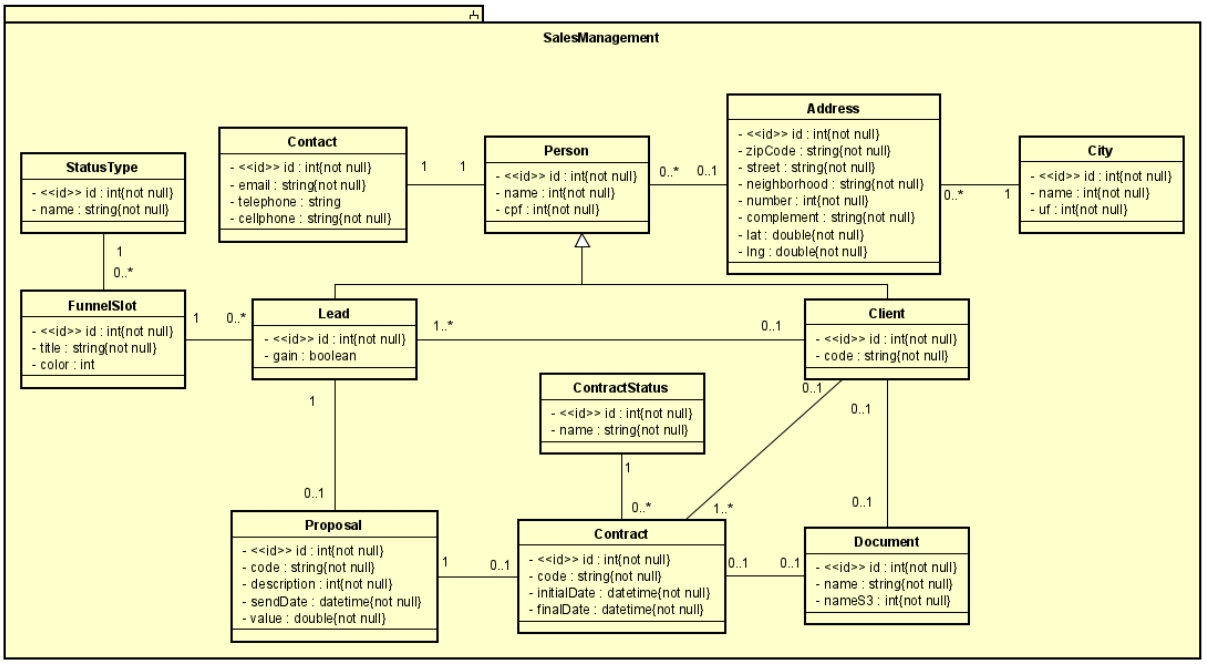


Figura 3 – Modelo de Entidades do Subsistema SalesManagement.

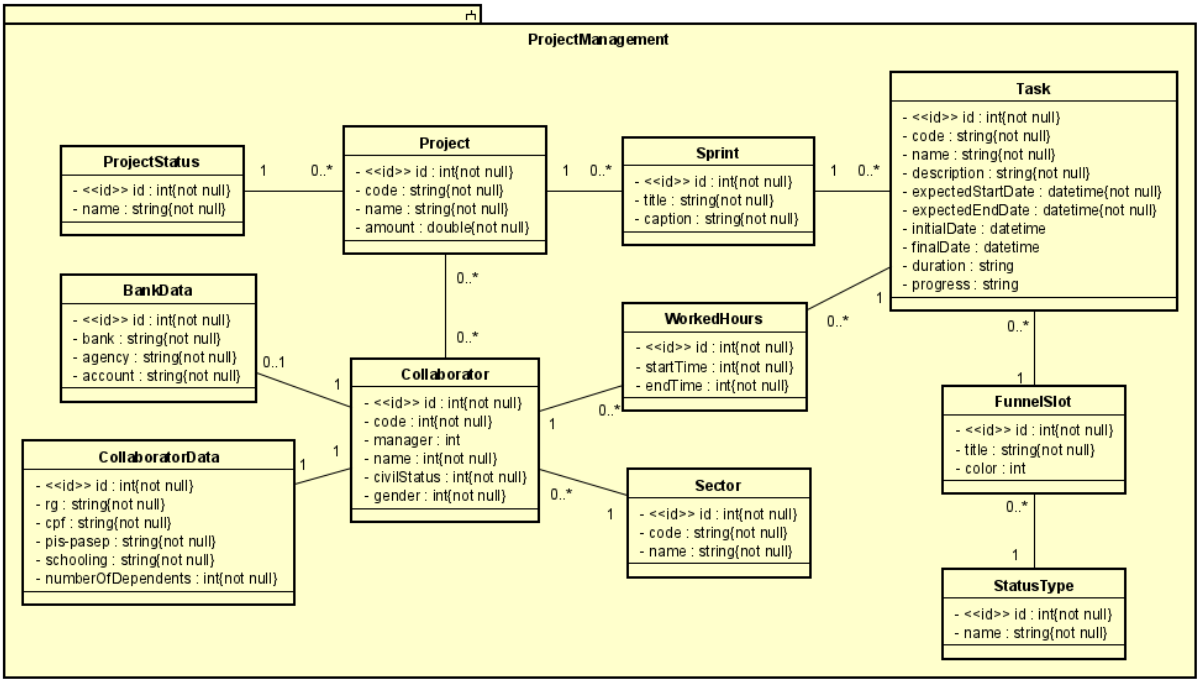


Figura 4 – Modelo de Entidades do Subsistema ProjectManagement.

5.2 Camada de Acesso a Dados

A camada de acesso a dados contém o pacote de persistência, responsável pelo armazenamento das informações em bancos de dados ou outras mídias de longa duração. No sistema de gestão de projetos utiliza-se DTOs (*Data Transfer Objects*) que são simples classes criadas quando necessárias que não contém nenhuma regra de negócio, mas são de suma importância para futuras manutenções de código, além disso, contribuem para uma

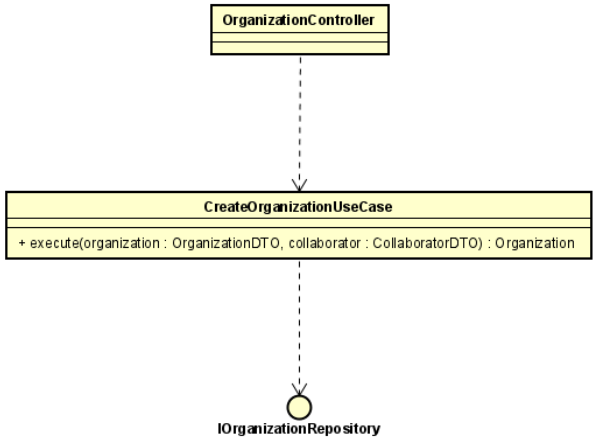


Figura 5 – Modelo de Aplicação da Entidade Organization do Subsistema UserManagement.

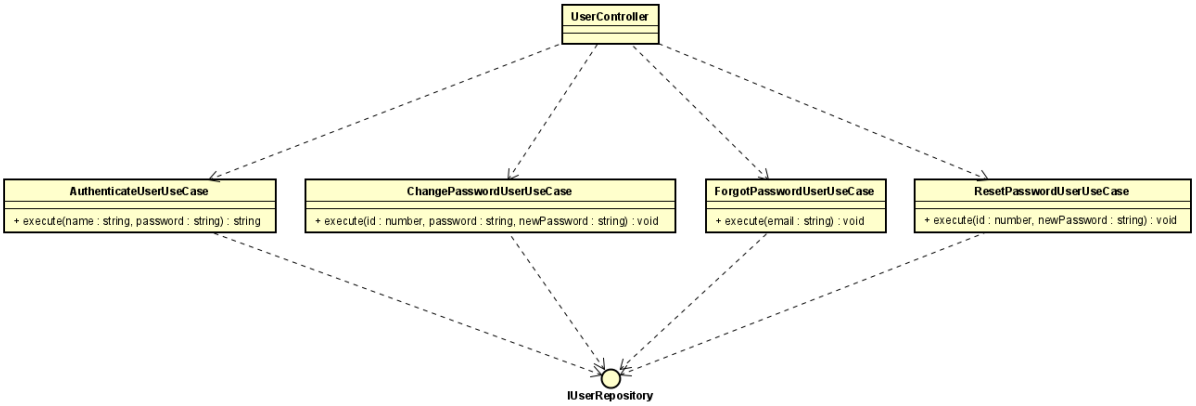


Figura 6 – Modelo de Aplicação da Entidade User do Subsistema UserManagement.

comunicação eficiente entre as camadas da arquitetura e aumentam a segurança dos dados de forma geral.

A Figura 7 ilustra o modelo de persistência no subsistema SalesManagement.

5.3 Camada de Apresentação

A camada de apresentação contém o pacote visão, que propõe prover a interface gráfica com o usuário, assim como parte da camada mais externa da arquitetura descrita na Seção 4.1.

Para ilustrar o modelo de navegação dentro das adaptações dessa ferramenta, escolheu-se trabalhar aqui com a página de projetos do subsistema ProjectManagement. A Figura 8 representa este modelo que se espelha para todo CRUD (*Create, Read, Update, Delete*) do sistema, salve devidas adaptações conforme a entidade.

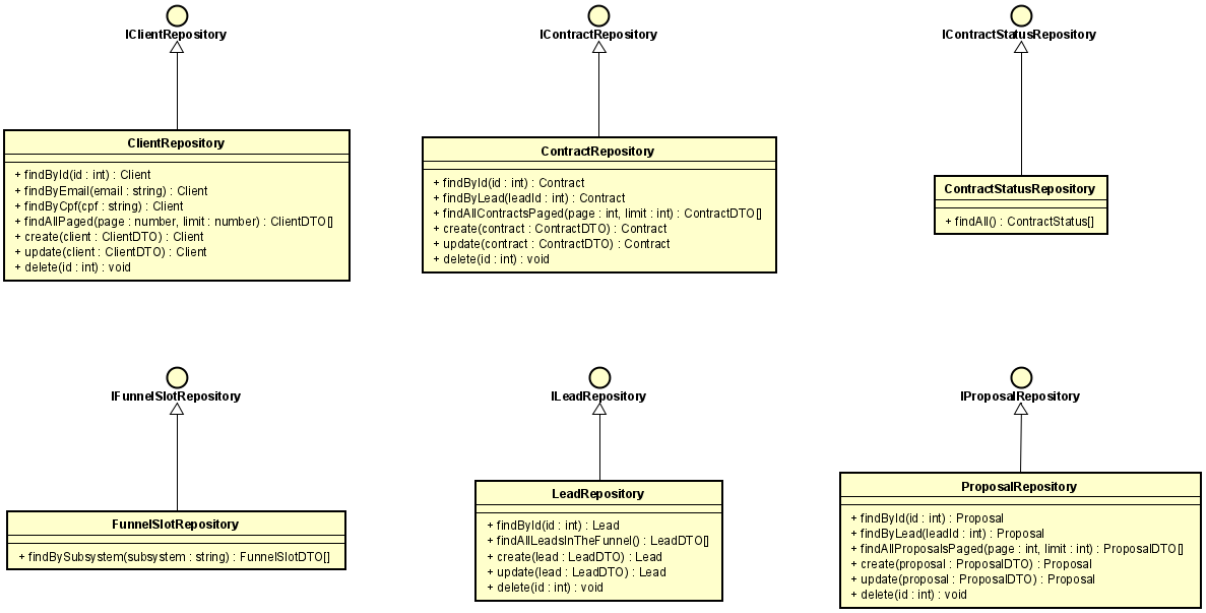


Figura 7 – Modelo de Persistência do Subsistema SalesManagement.

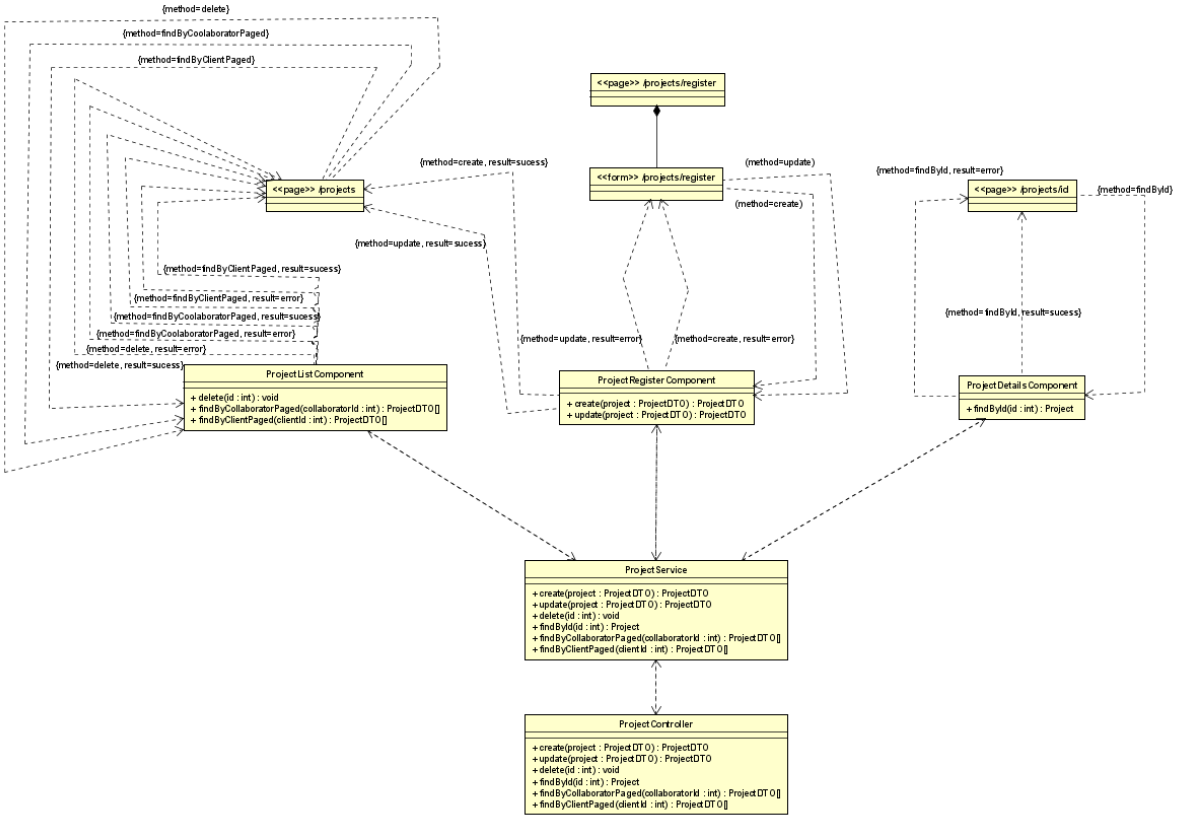


Figura 8 – Modelo de Navegação da Entidade Project do Subsistema ProjectManagement.

Referências

MARTIN, R. C.; GRENNING, J.; BROWN, S. *Clean architecture: a craftsman's guide to software structure and design*. [S.l.]: Prentice Hall, 2018. Citado 2 vezes nas páginas 5 e 6.

SOUZA, V. E. S. *FrameWeb: um Método baseado em Frameworks para o Projeto de Sistemas de Informação Web*. [S.l.], 2007. Disponível em: <https://nemo.inf.ufes.br/wp-content/papercite-data/pdf/souza-masterthesis07.pdf>. Citado na página 6.