SÃO LUCAS EDUCACIONAL DE JI-PARANÁ

FABRÍCIO GUSTAVO WAGOMACKER ROCHA

DESENVOLVIMENTO DE APLICATIVO PARA GERENCIAMENTO FINANCEIRO DE UMA EMPRESA CORRETORA DE IMÓVEIS NA CIDADE DE VILHENA-RO

Ji-Paraná
2021

SÃO LUCAS EDUCACIONAL DE JI-PARANÁ

FABRÍCIO GUSTAVO WAGOMACKER ROCHA

DESENVOLVIMENTO DE APLICATIVO PARA GERENCIAMENTO FINANCEIRO DE UMA EMPRESA CORRETORA DE IMÓVEIS NA CIDADE DE VILHENA-RO

Monografia apresentada à Banca examinadora da disciplina de Trabalho de Conclusão de Curso II do 9º período do curso de Sistemas de Informação do Centro Universitário São Lucas Ji-Paraná, como parte dos requisitos para obtenção do título de Bacharel em Sistemas de Informação, sob a orientação do Prof. Willian Fachetti.

Ji-Paraná
2021

SÃO LUCAS EDUCACIONAL DE JI-PARANÁ

Dados Internacionais de Catalogação na Publicação - CIP

R672d Rocha, Fabrício Gustavo Wagomacker.

Desenvolvimento de aplicativo para gerenciamento financeiro de uma empresa corretora de imóveis na cidade de Vilhena-RO. / Fabrício Gustavo Wagomacker Rocha. – Ji-Paraná, 2021.

111 p., il.

Monografia (Curso de Sistemas de Informação) – Centro Universitário São Lucas Ji-Paraná, 2021.

Orientador: Prof. Esp. Willian Alves de Oliveira Fachetti

1. Desenvolvimento de software. 2. Gestão de imóveis. 3. Gestão financeira. 4. Transformação digital. 5. Tecnologia da Informação. 6. Gerenciamento Imobiliário. I. Fachetti, Willian Alves de Oliveira. II. Título.

CDU 004.4:332.721.07

Ficha Catalográfica Elaborada pelo Bibliotecário Giordani Nunes da Silva CRB 11/1125

SÃO LUCAS
EDUCACIONAL

ATA DE TRABALHO DE CONCLUSÃO DE CURSO

ATA Nº 05/2020 DE TRABALHO DE CONCLUSÃO DE CURSO

No vigésimo terceiro dia do mês de junho de 2021, no horário das 18h às reuniram-se o(a) Orientador(a) professor(a) Esp. William Alves Fachetti e os(as) professores(as) Prof. Esp. Hailton Alves dos Reis e Prof. Me. Maigon Nacib Pontuschka para comporem Banca Examinadora de Trabalho de Conclusão de Curso, sob a presidência do(a) primeiro(a), para analisarem a apresentação do trabalho **"Desenvolvimento de Aplicativo para Gerenciamento Financeiro de uma Empresa Corretora de Imóveis na Cidade de Vilhena-RO"**. Após arguições e apreciação sobre o trabalho exposto foi atribuída à menção como nota do Trabalho de Conclusão de Curso dos(a) acadêmicos(a): Fabricio Gustavo Wagomacker Rocha.

Obs: Trabalho de Conclusão de Curso (X)aprovado ou ( )reprovado com nota total de 9,2 (nove virgula dois) pontos, sendo atribuídos o valor 9,0 (nove) pontos ao trabalho escrito e 9,5 (nove virgula cinco) à apresentação oral.

FABRICIO GUSTAVO WAGOMAKER ROCHA

Prof. Esp. Hailton Alves dos Reis

Prof. Esp. William Alves Fachetti
Orientador

Prof. Me. Maigon Nacib Pontuchska

Prof. Me. Thyago Bohrer Borges
Coord. Sistemas de Informação

São Lucas Educacional Ji-Paraná
Av. Eng. Manfredo Barata Almeida da Fonseca, 542
Jd. Aurélio Bernardi | Ji-Paraná | RO | CEP 76907-438

DEDICATÓRIA

Dedico este trabalho aos meus pais, Marlene Wagomacker e Gessi da Rocha, por nunca medirem esforços para me proporcionar ensino que me traria edificação. Seria impossível esquecer o trabalho de pós-graduação de minha mãe, que me motivou na qualidade deste trabalho, tampouco o esforço exaustivo de meu pai que nunca me negou um centavo ao investir em cursos, livros e materiais para estudo.

À minha noiva, Fábia Laissa Piovesan, que comigo passou horas em pesquisa e me motivando, sempre se preocupando com meu bem-estar e descanso, me preparando o que comer e beber enquanto eu lia e redigia. Certamente, teria sido bem mais difícil e cansativo sem ela ao meu lado, me elogiando e me cuidando com o carinho e amor que só ela tem para comigo.

Ao meu amigo e tutor Clayton Ferraz de Andrade e sua esposa Ellen Ferraz, os quais não consigo encontrar palavras para definir o carinho e consideração que tenho por eles. Como pais que amparam um filho, Ellen sempre visando meu empenho e Clayton sempre me ensinando tudo que podia sobre sistemas de informação, me abrindo as portas de sua casa para qualquer momento que pudéssemos nos ver.

Ao meu coordenador de curso Thyago Borges, que nunca, absolutamente nunca fez vista grossa ou desdém para qualquer situação que eu o tenha levado, muito pelo contrário, tratou de cada uma com zelo e muito apreço. É uma das pessoas mais competentes que já conheci, que realmente se importa com cada aluno de seu curso, fico muito feliz em ter cursado na instituição em que ele era o coordenador de curso, e sempre carregarei comigo a satisfação de cada auxílio prestado.

Dedico também a todos os meus amigos e pessoas próximas, que sempre me incentivaram e confiaram em minha competência, me fazendo sempre me sentir o melhor. Juntos, vencemos esta etapa.

AGRADECIMENTOS

Em primeiro lugar, agradeço a Deus por toda a guia e conduta até aqui e pelo que ele tem planejado para mim. Pelas pessoas que ele colocou no caminho o qual ele tem me guiado. Pela força e amparo nas provações, bênçãos e lições que tenho recebido. Sem ele, eu de nada seria capaz.

Agradeço a meus pais, avós e tios por toda a educação que me foi dada. O carinho e ensinamentos que recebi de minha família são impagáveis e espero um dia poder retribuir tudo que fizeram por mim. Também a meus irmãos, que sempre me encorajaram e motivaram a fazer tudo que pudesse para ser feliz.

Agradeço também a minha noiva, que me incentivou em momentos difíceis e compreendeu minha ausência enquanto eu me dedicava à realização deste trabalho e outros projetos a ele correlacionados, visando um futuro para nós. Também a sua família por todo o carinho, acolhimento e suporte prestados a mim no período em que me mudei para sua cidade, não há como não amar estas pessoas.

Agradeço ainda aos meus professores, e coordenadores de curso, que no decorrer dessa jornada me ensinaram e guiaram nos caminhos do conhecimento, não me deixando faltar nenhuma orientação para que eu alcançasse este objetivo final, ponte para meu futuro.

Também a empresa que me permitiu coletar os dados e me deu a oportunidade de desenvolver um software que atendesse às suas necessidades de negócio, me proporcionando aprendizado e experiência na área, e ao meu sogro que me apresentou à empresa, sendo crucial para a elaboração deste trabalho. É um enorme prazer trabalhar com vocês.

Não poderia esquecer de meus amigos, que sempre me apoiaram e comigo se angustiaram e alegraram nos momentos de minha vida. O nome deles estará sempre em minha memória como a família que ultrapassa as barreiras do sangue.

Por último, mas não menos importantes, meus colegas de curso, os quais tive o prazer de trabalhar e realizar atividades conjuntas, foram momentos proveitosos para aprendizado e divertidos para a construção de amizades duradouras.

RESUMO

A Tecnologia da Informação (TI) foi gradualmente incorporada ao cotidiano das empresas ao decorrer do tempo desde o seu conceito inicial, sendo aprimorada com a evolução da história e hoje se faz presente nas rotinas. Ao passo que a TI evoluiu, as corporações dispunham de mais ferramentas para lidar com seus negócios, ou com um departamento interno de TI ou terceirizando o serviço. Independente da forma, a informática colabora para a agilidade dos processos e ganhos no negócio. Num nicho tecnológico que envolve gerenciamento de contratos, controle de movimentação financeira de determinada empresa e tomada de decisões com base em dados pertinentes ao negócio, fez-se um estudo de caso de uma empresa corretora imobiliária, apontando os prejuízos e riscos do negócio quando não se utilizava uma forma sistematizada tecnológica das informações e processos da mesma. Foram realizadas entrevistas com os atores e coletados dados com questionários respondidos pelos funcionários da empresa sobre a expectativa e necessidade do serviço prestado. Vamos abordar neste projeto os requisitos para que a empresa evolua sua gestão com participação da tecnologia de sistemas de informação, com os parâmetros considerados mais importantes para a relação das partes da melhor maneira possível, e desenvolver um software utilizando os requisitos levantados nas entrevistas para garantir uma gestão ágil e organizada de uma forma que a empresa ainda não usufrui, visto a precariedade do conhecimento em TI da empresa candidata, e estudar os resultados da transformação digital obtida com a implantação do sistema, buscando comprovar fins benéficos para a organização.

Palavras-chave: Desenvolvimento de software, Gestão de imóveis, Gestão financeira, Transformação digital, Tecnologia da Informação.

ABSTRACT

Information Technology (IT) was gradually incorporated into the daily lives of companies over time since its initial concept, being improved with the evolution of history and today it is present in routines. As IT evolved, corporations had more tools to handle their business, either with an internal IT department or by outsourcing the service. Regardless of the form, information technology contributes to the agility of processes and business gains. In a technological niche that involves contract management, control of the financial movement of a given company and decision-making based on data relevant to the business, a case study of a real estate brokerage company was carried out, pointing out the losses and risks of the business when not a systematized technological form of information and processes was used. Interviews were carried out with the actors and data were collected with questionnaires answered by the company's employees about the expectation and need for the service provided. In this project, we will address the requirements for the company to evolve its management with the participation of information systems technology, with the parameters considered most important for the relationship of the parties in the best possible way, and develop software using the requirements raised in the interviews to ensure an agile and organized management in a way that the company does not yet enjoy, given the precariousness of the candidate company's IT knowledge, and to study the results of the digital transformation obtained with the implementation of the system, seeking to prove beneficial purposes for the organization.

Keywords: Software Development, Property Management, Financial Management, Digital Transformation, Information Technology.

LISTA DE QUADROS

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

ANSI	American National Standards Institute (Instituto Americano de Padrões Nacionais)
API	Algorithm Programing Interface (Interface de Programação de Algoritmo)
AWT	Advanced Windows Toolkit (Ferramentas Avançadas do Windows)
CSS	Cascading Style Sheets (Folhas de Estilo Cascata)
DAO	Data Access Object (Objeto de Acesso a Dados)
DBMS	Data Base Management System (Sistema Gerenciador de Banco de Dados)
DCL	Data Control Language (Linguagem de Controle de Dados)
DDL	Data Definition Language (Linguagem de Definição de Dados)
DER	Diagrama Entidade-Relacionamento
DETRAN	Departamento Estadual de Trânsito
DI	Dependencies Injection (Injeção de Dependências)
DML	Data Manipulation Language (Linguagem de Manipulação de Dados)
EE	Enterprise Edition (Edição Empresarial)
FK	Foreign Key (Chave Estrangeira)
FXML	Format Extensible Markup Language (Formato de Linguagem Extensível de Marcação)
IBM	International Business Machine (Máquinas Empresariais Internacionais)
IDE	Integrated Development Environment (Ambiente de Desenvolvimento Integrado)
IoC	Inversion of Control (Inversão de Controle)
JAR	Java Archive (Arquivo Java)
JDK	Java Development Kit (Kit de Desenvolvimento Java)
JIT	Just in Time (Exatamente na Hora)
JPA	Jakarta Persistence API (API de Persistência Jakarta)

JPQL	Java Persistence Query Language (Linguagem de Consulta da Persistência Java)
TI	Tecnologia da Informação
JRE	Java Runtime Environment (Ambiente de Tempo de Execução Java)
JSR	Java Specifications Request (Requisição de Especificações Java)
JVM	Java Virtual Machine (Máquina Virtual Java)
KPI	Key Performance Indicator (Indicador de Desempenho)
LTS	Long Therm Support (Suporte de Longo Período)
MVC	Model-View-Controller (Modelo-Visualização-Controlador)
OMG	Object Management Group (Grupo de Gerenciamento de Objetos)
OOAD	Object-Oriented Analysis and Design (Análise e Projeto Orientado a Objetos)
PC	Personal Computer (Computador Pessoal)
PK	Primary Key (Chave Primária)
POM	Project Object Model (Projeto Modelo de Objeto)
POO	Programação Orientada a Objetos
SE	Standard Edition (Edição Padrão)
SGBD	Sistema Gerenciador de Banco de Dados
SQL	Structured Query Language (Linguagem Estruturada de Consulta)
STS	Spring Tools Suite (Suíte de Ferramentas Spring)
UML	Unified Modeling Language (Linguagem Unificada de Modelagem)
UUID	Universal Unique Identifier (Identificador Único Universal)
WAR	Web application Archive (Arquivo de aplicação Web)
WIP	Work in Progress (Trabalho em Andamento)

SUMÁRIO

1 INTRODUÇÃO

Os gastos com Tecnologia da Informação (TI) ainda são um mistério para certas organizações. A falta de conhecimento e valorização do que um sistema pode agregar leva algumas entidades a postergar a implantação de uma TI estruturada. Essas empresas por vezes perdem tempo e controle do negócio por não integrarem tecnologia a gestão do negócio (MACEDO; PEDRON; CATELA, 2014).

Ainda que não tão pertinente ao projeto, é interessante que a empresa busque seus Indicadores de desempenho, também chamados de KPI (*Key Performance Indicator*). Segundo Doyle, 2018, Estes são métricas que relacionam a performance obtida com a performance desejada. Esta mensuração do desempenho encontra aplicação em avaliar a capacidade da empresa em gerenciar suas áreas críticas e em proporcionar uma visão das condições projetadas para o futuro. No atual contexto competitivo, é importantíssimo que a empresa se encontre com seus indicadores. A empresa candidata ao estudo de caso realizou sua avaliação e buscou alternativas para solucionar alguns de seus desafios de gestão.

Este projeto visa a transformação digital de uma empresa. Em minha mudança de cidade, conheci a empresa candidata através do pai de minha namorada. Sabendo de meu curso superior na área de sistemas de informação, me contou, em uma de nossas conversas, a situação da empresa em que trabalha e quais seriam as medidas possíveis para atualizar a empresa e melhorar a gestão. Após algumas observações, percebe-se que atualmente a organização usa um modelo antigo e pessoal de controle de seus imóveis e financeiro que não provê controle sistematizado e processamento adequado da informação. Será descrito as práticas de elaboração do projeto e a estratégia de desenvolvimento de um software que atenda a necessidade de organização desta empresa, de forma personalizada, e disponibilizando suporte futuro para operações críticas do negócio.

A empresa candidata é uma corretora imobiliária, foi fundada em 2016, na cidade de Vilhena - RO. Atua primariamente no ramo de vendas e locações de imóveis, mas também presta serviços como confecção de contratos para terceiros, avaliações de imóveis e intermediação de vendas para terceiros.

2 PROBLEMATIZAÇÃO

A empresa dispõe de um modelo arcaico de armazenamento de informações referentes ao negócio. Não há um armazenamento sistematizado das informações. Os contratos são armazenados em pastas de arquivos físicos, e não há registros de informações dos clientes. O controle de fluxo de entrada e saída da empresa é feito utilizando documentos e planilhas criadas por funcionários antigos da empresa, o que tem impacto direto com renovação de funcionários e seus diferentes estilos. A digitação nem sempre é fiel ao caso, e as fórmulas das planilhas por vezes não compreendem os valores inseridos, havendo necessidade de constante correção dos dados e fórmulas, bem como os documentos que dispõem de tabelas manuais, com cálculos imprecisos e sujeitos a erros.

Todos os relatórios, contratos e recibos são redigidos manualmente. Quando é necessária alguma informação, se verifica no contrato dos clientes, nos documentos e planilhas arquivados, ou no mensageiro instantâneo WhatsApp, onde ficam as conversas entre a empresa e os clientes salvas. Esta é uma condição genérica, com pouquíssimo gerenciamento e tratamento dos dados e informações necessárias. Ademais, as conversas podem ser apagadas, arquivos perdidos, registros podem ser removidos acidentalmente ou maliciosamente, entre outras fatalidades.

A rotatividade de funcionários, que caracteriza uma diferença na forma de trabalho entre os operadores do negócio, isto é, a forma que criam, editam e gerenciam os arquivos, aliado à falta de armazenamento apropriado de informações, acarreta em insatisfações ao buscar um dado necessário e não o encontrar, havendo de buscar de outras formas a informação desejada, correndo risco de não a recuperar.

Energia e tempo que poderiam ser direcionados a outras áreas do negócio são gastos de forma ineficiente para solucionar problemas humanos que seriam facilmente contornados utilizando soluções sistematizadas, onde as informações e dados úteis para a gerência do negócio estariam centralizadas e disponíveis para análise e ajuda na tomada de decisão.

3 HIPÓTESES

É possível adaptar as planilhas e documentos a fim de melhorar a organização, evitar reedições e garantir a confiabilidade dos dados, e agilizar o acesso a informação. Também disponibilizar em uma página web para visualização e download os arquivos mais solicitados pelos usuários.

A empresa poderia comprar um software disponível no mercado. Estes softwares dispõem de vários módulos e riqueza em opções para gestão e análise de dados e ferramentas de controle. Ou até mesmo utilizar múltiplas plataformas, cada uma para atender cada necessidade do negócio.

Há ainda o processo de desenvolver uma aplicação com os módulos desejados: Controle de Clientes e imóveis, Controle Financeiro e Gestão de contratos. Com base nas regras do negócio, é possível desenvolver um software que seja capaz de realizar os cálculos e gerenciamentos necessários para boa gestão empresarial, e até aprimorando tomadas de decisões com base nos dados disponibilizados.

4 OBJETIVOS

4.1 OBJETIVO GERAL

Desenvolver um sistema de gestão empresarial baseado nas entrevistas e necessidades do candidato ao estudo de caso e apontar seu sucesso de gestão.

4.2 OBJETIVOS ESPECÍFICOS

Desenvolver um módulo para controle de clientes e suas informações;

Desenvolver um módulo para controle de imóveis, seus proprietários e suas informações;

Desenvolver uma ferramenta que gere contratos de locações e vendas com dados inseridos no sistema;

Disponibilizar relatórios rápidos para apreciação de informações do negócio;

Estabelecer os pontos benéficos obtidos com a sistematização de dados.

5 JUSTIFICATIVA

Trabalhar com inserção dos registros manuais, criar os contratos a partir de completa inexistência e lidar com muitos arquivos para se trabalhar toma tempo e requer conhecimento de edição de documentos e planilhas por parte do funcionário. É uma solução simples, mas não tão simples quanto parece ser. Não seria suficiente para corrigir efetivamente o problema. Ainda demandaria pesquisar o disco rígido por arquivos específicos, sem saber assim quais dados dentro do arquivo estão sendo procurados, sem dinâmica no agrupamento e refinamento dos dados desejados.

Adquirir um software ou vários softwares existentes no mercado é uma hipótese muito viável do ponto de vista de opções, porém inviável relacionando aos recursos necessários que a empresa dispõe. Fragmentar a gestão entre diferentes aplicativos também não é atrativo, pois além da ineficácia das opções gratuitas ou até mais baratas no mercado em gestão parcial ou completa necessária, requer cadastrar os mesmos dados em várias plataformas diferentes, ocasionando dados incoerentes e desatualizados entre as plataformas.

A proposta escolhida para o projeto, que consiste em desenvolver um software específico, propõe armazenamento e acesso eficiente às informações dos clientes, imóveis e proprietários do negócio, centralizando as informações neste software, para que esteja disponível sempre que necessário para o operador. Será oferecido um módulo de geração automática de contratos com base nas informações destas entidades, cadastradas e validadas, com modelos predeterminados, ajustados com métodos de *data binding* (ligação de dados), eliminando edição manual de cada contrato e falhas de edição e possibilitando tratamento especializado com programação e geração de relatórios eficientes. Por último, o controle financeiro configurado especificamente para as regras de negócio implantadas na empresa, tais como as mensalidades de locação e acordo entre proprietários e administradores.

6 REFERENCIAL TEÓRICO

6.1 TECNOLOGIA E GESTÃO EMPRESARIAL

Os modelos de gestão se moldam de acordo com o contexto histórico que a humanidade se encontra. Durante a revolução industrial, o processo mecânico de produção foi o modelo mental para moldar muitos processos da sociedade, desde modelos educativos até modelos de gestão empresarial. (MORAES; TERENCE; FILHO, 2004)

Para que uma empresa mantenha sua competitividade, ela precisa acompanhar a contínua evolução tecnológica, que por sua grande disseminação, afetou completamente todas as atividades do nosso cotidiano, e nos traz uma incerteza sobre como será o futuro. (MORAES; TERENCE; FILHO, 2004)

6.2 ARQUITETURA DESKTOP

O Desenvolver tecnológico possibilitou diversas formas de criação de sistemas. Diferentes arquiteturas, estruturas, hospedagem, e tratamento de dados foram criados durante a história, e hoje temos uma migração exponencial de aplicações desktop ou em rede para serviços em nuvem, porém nem sempre essa é a solução necessária.

De acordo com o Xgen (2021), devido ao aumento de uso de navegadores e redes sociais, é comum ouvir sobre a morte de aplicações para desktop. Temos que tomar cuidado ao correr tal afirmação, pois pode ser equívoca. Uma aplicação desktop é um programa que precisa estar instalado em sua máquina, independente de qual seja sua função. Estas aplicações podem ser usadas por muitos usuários através de ambientes de rede e podem funcionar com nada ou quase nada de uso de internet e tendem a ser fortes em seu nível de segurança pela sua estrutura e armazenamento, estando menos vulneráveis a ataques e obtenção indevida de informações. Portanto, uma aplicação desktop pode ainda ser uma ótima escolha em um momento em que há um debate sobre privacidade e segurança de informação no mundo inteiro.

Figura 1 – Comparativo básico entre os tipos de aplicação

Fonte: https://xgen.com.br/blog/aplicacao-web-ou-desktop-qual-a-melhor-solucao

6.3 METODOLOGIAS ÁGEIS DE DESENVOLVIMENTO

Segundo Groffe (2012), para que o projeto alcance sucesso, é crucial definir o paradigma e metodologia de desenvolvimento. As primeiras metodologias eram muito formais e proporcionavam boa auditoria, mas eram rígidas e pouco suscetíveis a mudanças no projeto. Sistemas podem ter requisitos variáveis e alteração em sua demanda, bem como mudanças em seu design, requerendo assim uma rápida adaptação diante destas situações imprevistas. Tais situações promoveram o desenvolvimento de práticas mais flexíveis para o desenvolvimento de softwares, com diversas destas sendo conhecidas como “metodologias ágeis”.

> A evolução da área de sistemas foi acompanhada pelo surgimento de diversas metodologias, com estas últimas normalmente englobando um conjunto de diretrizes e conceitos criados com o intuito de nortear o processo de construção de um software. Como de praxe, não há uma fórmula mágica para se chegar ao resultado final, ou seja, uma aplicação que atenda às expectativas dos usuários e seja capaz de funcionar dentro de uma série de parâmetros considerados como aceitáveis. Partindo de um conjunto de técnicas e diretrizes com eficácia já comprovada, os profissionais envolvidos empreendem esforços no sentido de adaptar um modelo para a realidade na qual se encontram inseridos. (GROFFE, 2012)

6.3.1 Metodologia Ágil Scrum

De acordo com Groffe, 2012, O método Scrum surgiu na década de 90, resultado dos esforços de especialistas em sistemas, com destaque para Ken Schwaber e Jeff Sutherland. O termo “Scrum” é originário do jogo de *rugby* e é um termo que significa o reinício da partida sempre que uma infração leve ocorre. As ideias principais do Scrum são representadas pelos pilares:

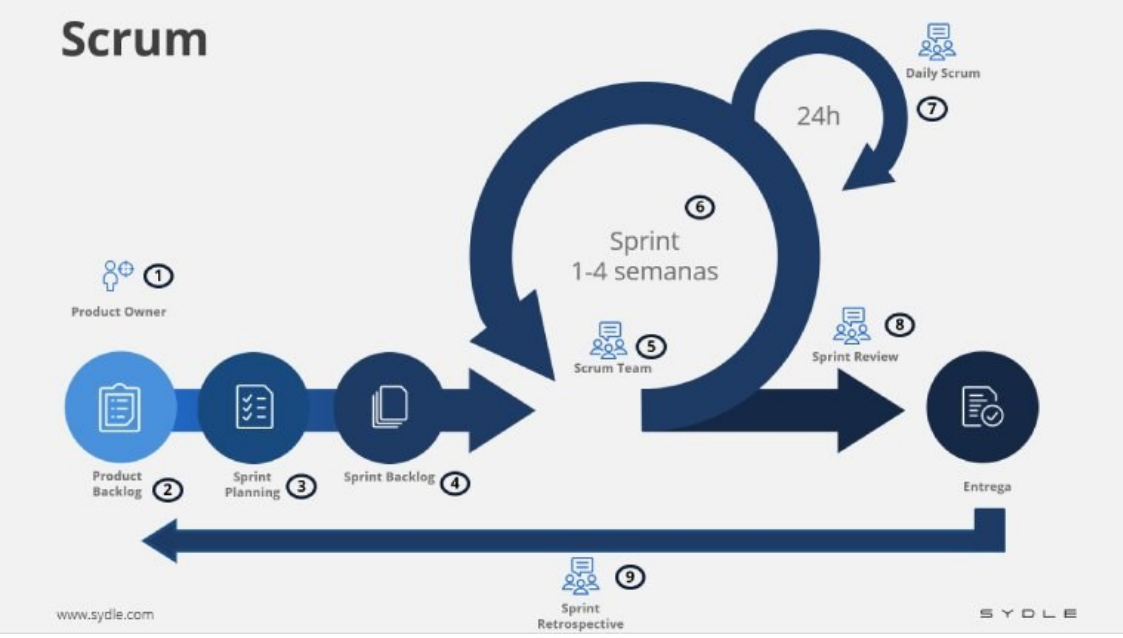
- Transparência, que significa um consenso entre todos os participantes do projeto
- Inspeção de tudo que é produzido para determinar que o projeto segue como planejado
- Adaptação diante de mudanças, pois não são difíceis os casos de alterações solicitadas por um cliente.

Os principais atores de um projeto Scrum são:

- *Product Owner*: O cliente ou solicitante do software
- Scrum Master: O líder do projeto e que vai atuar como gerente
- Equipe de desenvolvimento: Os desenvolvedores das áreas do software.

6.3.1.1 Termos do Scrum e metodologias ágeis

Figura 2 - Ciclo de vida do Scrum



Fonte: https://www.sydle.com/br/blog/metodologia-agil-scrum%20-5f6dc45f320703787497f887/

6.3.1.1.1 Product Backlog (Backlog do produto)

O Backlog do produto representa as funcionalidades desejadas do sistema, podendo ser representadas pelo levantamento de requisitos. Também é levantada através de histórias contadas pelo *product owner* de como seria o uso do sistema. (SYDLE, 2020)

6.3.1.1.2 Sprint Backlog

O *sprint backlog* é o planejamento da equipe para o próximo sprint programado, podendo variar geralmente de 1 a 4 semanas o tempo de cada sprint. (SYDLE, 2020)

6.3.1.1.3 Incremento

Ainda que não seja a fase final do produto, o incremento é o resultado de cada sprint e deve agregar valor do software para o cliente, como nova funcionalidade. (SYDLE, 2020)

6.3.1.1.4 Release Burndown

Um gráfico que mensura o andamento do projeto. Deve ser sempre atualizado para acompanhar o progresso de acordo com o que foi planejado. (SYDLE, 2020)

6.3.2 Metodologia Ágil KANBAN

Kanban é um método em ascensão e muito utilizado por equipes de software ágil e DevOps, mas ela vem já de tempos atrás, quando a Toyota, no fim da década de 40, otimizou seus processos com base no modelo JIT (*Just In Time* – No momento) de atuação. Isso acontecia de maneira que quando algo faltasse na linha de produção, um “kanban”, que significa um cartão, era passado para o depósito e então a reposição acontecia. Hoje, as equipes de software se aproveitam dessa prerrogativa para combinar o JIT com WIP (*Work In Progress* – Trabalho em andamento) para definir as etapas de desenvolvimento com base em cartões, como post-its, para representar o que é necessário implementar no software e entregar ao cliente. (RADIGAN, 2021)

6.3.2.1 O quadro Kanban

Às vezes é difícil perceber o que está causando problemas, e definir a quantidade de tarefas que precisam ser concluídas antes de ter algo concreto para disponibilizar como atualização para o cliente. Quando se monta o quadro Kanban, que nada mais é que um quadro com colunas que descrevem as prioridades, o que fazer e o que está feito, para o seguimento do projeto, os insights do projeto se tornam visíveis, possibilitando direcionar os esforços para resolver um desafio. (SOLOVYOVA, 2019)

Figura 3 - Quadro Kanban

Fonte: https://www.bitrix24.com.br/blogs/dicas/o-que-o-quadro-kanban-e-como-us-lo.php

Diferente do Scrum, que define *sprints* semanais e reuniões da equipe, O quadro Kanban busca simplificar o processo de desenvolvimento adotando colunas para que a equipe saiba o que está acontecendo no projeto instantaneamente, apenas de olhar para o quadro. São necessárias basicamente três colunas: Lista de tarefas, Em andamento, e Concluídas. Isso pode variar de acordo com a necessidade da equipe. O Kanban maximiza a eficiência da equipe, concentrando esforços para reduzir o tempo do projeto do início ao fim. (REHKOPF, 2021).

No Kanban, assim que uma funcionalidade é concluída, ou um “kanban” é finalizado, ele já é disponibilizado para o cliente, diferente do Scrum, que requer o fim do Sprint para liberar as atualizações. Teoricamente, no Kanban, não há prazos para entrega. Se uma tarefa for concluída antes do planejado, ela pode ser lançada

conforme necessário para o cliente. Não há um mestre de projetos, sendo responsabilidade conjunta da equipe colaborar com as entregas descritas no quadro Kanban. (REHKOPF, 2021)

> Segundo Rehkopf, 2021, um fluxo de trabalho Kanban pode ser alterado a qualquer momento. Itens de trabalho novos podem ser adicionados ao backlog e os cartões existentes podem ser bloqueados ou removidos todos juntos com base na priorização. Além disso, se a capacidade da equipe mudar, o limite de WIP pode ser recalibrado e os itens de trabalho ajustados conforme o necessário. No Kanban deve haver flexibilidade.

Ademais, o Kanban não se trata de várias pessoas executando várias tarefas sem planejamento ou ordem alguma, simplesmente pegando o que tem no quadro a fazer. Para isso, é determinado o WIP. O WIP define quais as tarefas serão executadas, com base na história ou requisito atrelada ao kanban do quadro. Geralmente, se define 3 tarefas para cada WIP. Destas 3 tarefas, cada equipe responsável realizará as tarefas vinculadas a este WIP, para que não façam tudo de uma vez e não seja produzido nada. (SOLOVYOVA, 2021).

6.4 UNIFIED MODELING LANGUAGE (UML)

Segundo Costa (2001), A UML (Linguagem de Modelagem Unificada) surgiu da união de métodos de análise de sistemas orientados a objetos, e em 1997 foi reconhecida como potencial modelagem a partir de notações de múltiplas perspectivas de sistemas de informações pela OMG (*Object Management Group* – Grupo de gerenciamento de Objetos).

A UML é uma linguagem sólida para desenvolvimento de sistemas. Ela fornece notações e diagramas que representam os requisitos levantados. Há um excesso de cuidados na criação dos diagramas e notações UML, mas garante que haja a detecção de contradições e omissões no projeto. A UML não é um método de desenvolvimento de sistemas, ela serve para modelar o software, bem como visualizar e documentar (SALMON, 2017).

A UML é dividida em três grupos: Diagramas estruturais, diagramas comportamentais e diagramas de interação.

6.4.1 Diagramas Estruturais

Mostra a estrutura estática do sistema, as partes em níveis de abstrações e suas relações. Os diagramas deste grupo estão descritos no quadro 1. (SALMON, 2017).

Quadro 1 - Diagramas Estruturais

Diagrama	Função
Diagrama de Classes	Permite a visualização de um conjunto de classes, detalhando atributos e operações (métodos) presentes nesta última, assim como prováveis relacionamentos entre essas estruturas. Este tipo de representação pode incluir ainda definições de interfaces.
Diagrama de Componentes	Apresenta diferentes componentes de um sistema, além de possíveis dependências entre tais elementos. A ideia de componente refere-se a uma parte ou um módulo de uma aplicação, englobando assim uma séria de outras estruturas relacionadas (como classes, interfaces etc.).
Diagrama de Pacotes	Descreve as dependências entre diferentes namespaces/pacotes que compõem uma aplicação. Dentro da plataforma **.**NET, um namespace costuma conter classes, interfaces e outros elementos, atuando como uma forma de agrupamento lógico destes elementos.
Diagrama de Objetos	Apresenta o estado de instâncias de objetos dentro de um sistema, levando em conta para isto um intervalo de tempo específico.
Diagrama de Estrutura Composta	Utilizado para demonstrar a estrutura interna de uma classe, incluindo referências que apontam para outras partes de um sistema.
Diagrama de Instalação	Empregado para demonstrar a estrutura de hardware adotada para a implantação de uma aplicação em um ambiente. Pode envolver dispositivos como servidores de

	aplicação, servidores de banco de dados, terminais de usuários etc.
Diagrama de Perfil	Possibilita a definição de novos elementos UML, permitindo assim estender os diagramas existentes com a inclusão de estruturas customizadas para uma determinada necessidade.

Fonte: https://www.devmedia.com.br/modelagem-de-sistemas-atraves-de-uml-uma-visao-geral/27913

6.4.2 Diagramas Comportamentais

Mostra como o sistema se comporta com determinadas ações ou com alterações ao decorrer do tempo. Os Diagramas deste grupo estão descritos no quadro 2. (GROFFE, 2013)

Quadro 2 - Diagramas Comportamentais

Diagrama	Função
Diagrama de Casos de Uso	Voltado à apresentação de funcionalidades e características de um sistema, assim como de que forma tais elementos se relacionam com usuários e entidades externas envolvidas num determinado processo.
Diagrama de Atividades	Contempla as diversas tarefas desempenhadas na execução de uma atividade, sendo utilizado geralmente na representação de processos dentro de uma empresa/organização.
Diagrama de Transição de Estados	Detalha os diferentes estados pelos quais pode passar um objeto, tomando por base a execução de um processo dentro do sistema que se está considerando.

Fonte: https://www.devmedia.com.br/modelagem-de-sistemas-atraves-de-uml-uma-visao-geral/27913

6.4.3 Diagramas de Interação

Considerados um subgrupo dos diagramas comportamentais, são utilizados para representar interações entre objetos de uma aplicação. Os diagramas deste grupo estão descritos no quadro 3. (GROFFE, 2013)

Quadro 3 - Diagramas de Interação

Diagrama	Função
Diagrama de Sequência	Demonstra as interações entre diferentes objetos na execução de uma operação, destacando ainda a ordem em que tais ações acontecem num intervalo de tempo. A sequência em que as diversas operações são executadas ocorre na vertical, de cima para baixo.
Diagrama de Interatividade	Espécie de representação híbrida, com uma estrutura similar à de diagramas de atividade. O que diferencia este tipo de representação está justamente no fato do equivalente a uma atividade ser representada por outro diagrama, sendo o de sequência um exemplo de uso válido neste último caso.
Diagrama de Colaboração ou Comunicação	Similar aos diagramas de sequência, é também empregado na modelagem de interações entre vários objetos dentro de um determinado contexto. Este tipo de representação difere de um diagrama de sequência por não possuir uma estrutura rígida para demonstrar a comunicação entre objetos, ou seja, estes elementos podem ser dispostos na melhor ordem que se julgar necessária, sem a obrigatoriedade de exibir as diferentes operações na vertical uma após a outra.
Diagrama de Tempo	Corresponde a um tipo específico de diagrama de sequência, descrevendo mudanças de estado e interações entre objetos dentro de intervalos de tempo tomados como parâmetro.

Fonte: https://www.devmedia.com.br/modelagem-de-sistemas-atraves-de-uml-uma-visao-geral/27913

6.5 LINGUAGENS DE PROGRAMAÇÃO

Programadores escrevem código de máquina em várias linguagens de programação. Algumas delas são diretamente lidas pela máquina, outras já precisam de passar por uma tradução para que a máquina compreenda e execute. Podemos dividir as linguagens em três grupos principais: Linguagens de máquina, linguagens *Assembly*, e linguagens de alto nível. (DEITEL e DEITEL, 2016)

Qualquer computador só consegue entender a linguagem de máquina. As linguagens de máquina geralmente são compostas por *strings* de números. Elas são complicadas para a compreensão do ser humano. Já as linguagens *assembly* foi criada para resolver paradigmas da linguagem de máquina, que incluíam sua lentidão e ser muito tediosa. Elas usam termos em inglês para representar as operações. Embora seu uso seja mais compreensível para humanos, precisam ser traduzidos para serem executados pela máquina. Então, o uso de computadores cresceu muito rápido, mas os programadores ainda precisavam de muito tempo e usar várias instruções para executar tarefas mais simples, então foram desenvolvidas as linguagens de alto nível, que usam termos e operações mais compreensíveis, mas que precisam de compiladores para traduzi-los em linguagem de máquina. O quadro 4 mostra exemplos de um programa de folha de pagamento que controla horas extras. (DEITEL e DEITEL, 2016)

Quadro 4 - Exemplos de programação entre os tipos de linguagens

Linguagem de máquina	Linguagem Assembly	Linguagem de alto nível
+1300042774 +1400593419 +1200274027	load basepay add overpay store grosspay	grossPay = overpay + basepay

Fonte: DEITEL e DEITEL, 2016. Java: Como programar, p. 7-8.

Nota: O quadro foi elaborado pelo autor com base nos trechos encontrados na fonte.

> “A contribuição mais importante até agora da revolução dos microprocessadores é que ela permitiu o desenvolvimento de computadores pessoais. Os microprocessadores estão tendo um impacto profundo em dispositivos eletrônicos inteligentes de consumo popular”. (DEITEL e DEITEL, 2016)

Notamos então que nestas linguagens mais antigas, não há muita separação entre os elementos de um código. Funções, atributos, era tudo programado em um único bloco, onde para se conseguir criar fluxos diferentes de execução era necessário o uso do temido comando ‘goto’. O código tinha pouca legibilidade e era muito acometido de *boilerplate* (repetição de trecho de código), o que dificultava o desenvolvimento e manutenção. Foi daí que surgiu a expressão ‘código macarrônico’, que representa a dificuldade de ler e compreender o fluxo de execução desse tipo de código. (GUERRA, 2014)

Então houve o advento da programação estruturada. Neste paradigma, vieram as estruturas de controle que permitiam execuções com condições, como o ‘if’ e o ‘switch’, e comandos iterativos, como o ‘for’ e ‘while’. Também vieram as funções, que por sua vez dividiam a lógica do código. Isso facilitava o uso das funções em outras aplicações. Mas este paradigma tinha suas dificuldades e desafios. Era difícil lidar com o comportamento dos dados. Eram criadas variáveis globais que com facilidade causavam problemas por terem valores inconsistentes. (GUERRA, 2014)

De acordo com Guerra (2014), surge então, como uma grande evolução da linguagem estrutural, a POO (Programação Orientada a Objetos). Os dados e a lógica dos dados agora podiam ser estruturados e modelados, bem como os conceitos do software. Então, era possível desacoplar componentes, permitindo um maior reuso e flexibilidade maior de código.

6.5.1 Conceitos da Orientação a Objetos

A busca por softwares novos cada vez mais poderosos faz com que a construção rápida, econômica e correta de software seja um objetivo indefinido. Desenvolvedores podem modular um projeto e implementar orientação a objetos para alcançar maior produtividade do que as técnicas anteriores de programação estruturada. Programas orientados a objetos são mais legíveis e mais fáceis de corrigir e prestar manutenção. (DEITEL e DEITEL, 2016)

6.5.1.1 Classes e Objetos

Em orientação a objetos, podemos criar novos tipos através das classes, e estes tipos podem ser instanciados criando objetos. Um objeto é a representação real,

uma entidade concreta, enquanto a classe é a abstração dos conceitos desse objeto. Por exemplo, caso haja uma classe ‘Gato’, Garfield, 4 anos, cor laranja e Frajola, 6 anos, cor preta, seriam os objetos dessa classe. (GUERRA, 2014)

> Para Deitel e Deitel (2016), “[...]. Quase qualquer substantivo pode ser razoavelmente representado como um objeto de software em termos dos atributos (por exemplo, nome, cor e tamanho) e comportamentos (por exemplo, calcular, mover e comunicar) ”.

No momento de projetar um sistema orientado a objetos, é importante definir as classes e como elas irão contribuir na implementação dos requisitos da aplicação. Se uma classe não representa uma abstração de um objeto e apenas oferece funções e métodos, o projeto na verdade usa o paradigma de programação estruturada. (GUERRA, 2014)

6.5.1.2 Instanciação

Assim como uma montadora tem que fabricar um carro a partir dos desenhos de projetos antes de poder dirigi-lo, precisamos construir um objeto de uma classe antes que as funções e tarefas sobre esse objeto possam acontecer, ou seja, precisamos construir o objeto de uma classe antes de usá-lo. O nome desse processo é Instanciação. Um objeto é uma instanciação de sua classe. (DEITEL e DEITEL, 2016)

6.5.1.3 Atributos

Uma classe possui características, um objeto possui atributos. Considerando como exemplo uma pessoa. Uma classe é uma pessoa, que possui suas características, como nome, idade, altura, peso e gênero. Um objeto é uma representação de uma classe construída, ou seja, O Objeto Funcionário é a construção de uma classe pessoa, com o nome Fulano da Silva, 35 anos, 1,75 metros, 65 quilos, do gênero masculino. Os atributos são as propriedades já definidas de um objeto já construído. (PERRY, 2010)

6.5.1.4 Reutilização

Assim como o projeto de um carro pode ser reutilizado muitas vezes para fabricar muitos carros, podemos usar uma classe diversas vezes para instanciar vários objetos. Isso poupa tempo e trabalho do desenvolvedor e contribui para a construção de sistemas confiáveis e eficientes, já que os objetos serão construídos por uma classe definitiva que já foi testada e aprovada. (DEITEL e DEITEL, 2016)

6.5.1.5 Encapsulamento

“Sempre que falo sobre encapsulamento, inicio com a frase ‘A ignorância é uma benção! ’. [...] com a complexidade das coisas com que lidamos nos dias de hoje, é realmente muito bom não precisar saber como elas funcionam para utilizá-las”. (GUERRA, 2014)

É ótimo usarmos o controle remoto para mudar o canal da televisão, controlar o volume e não precisar entender como essas coisas funcionam, elas simplesmente funcionam. A POO traz um conceito parecido. Antes, um programador era obrigado a saber os detalhes de implementação do código: seu funcionamento interno, como as variáveis estão sendo acessadas e atualizadas. O encapsulamento separa o comportamento interno de uma classe da interface que ela dispõe para o seu usuário. Isso permite que um desenvolvedor precise apenas saber como interagir com ela, sem precisar conhecer seu comportamento interno. (GUERRA, 2014)

Métodos *getters* e *setters* são exemplos de encapsulamento. Eu não preciso saber como o objeto carro é pintado, como a tinta é escolhida ou onde ele foi pintado. Apenas utilizo um método como ‘getCor()’ para receber a cor do carro, sem que me importe a implementação, e um método como ‘setCor()’ para alterar a cor deste carro. (GUERRA, 2014)

6.5.1.6 Herança

Podemos desenhar uma classe nova a partir de outra classe utilizando o conceito de herança. Ela (chamada de subclasse) começa com as características de uma classe já existente (que ganha o nome de superclasse), e pode ser personalizada

e adicionar características próprias. Um exemplo é que existe uma superclasse ‘carro’, e uma subclasse ‘conversível’, que é um carro, mas que o teto pode ser retraído ou expandido. (DEITEL e DEITEL, 2016)

6.5.1.7 Interfaces

Deitel e Deitel (2016) indicam que há também a implementação de interfaces. Interfaces são conjuntos de métodos que informam a um objeto o que fazer, mas não como fazer. Utilizando a analogia do carro, você pode dirigir um carro que pode conter direção hidráulica, mas outro carro tem direção mecânica e outro carro tem direção elétrica. Tudo isso se refere ao modo que você dá direção ao carro, mas não como essa direção é passada para o carro. Uma interface de *backup* pode fornecer os métodos ‘salvar()’ e ‘restaurar()’, mas não diz como elas devem ser implementadas, apenas que elas precisam ser implementadas. Não fala como, mas sim o quê.

Agora vamos a um confronto: Quando devemos elaborar uma interface e quando devemos elaborar uma classe abstrata? É uma dúvida pertinente, afinal, são comportamentos parecidos. A diferença é que uma classe constrói um objeto, algo conceitual, e uma interface infere um comportamento a algo. Vamos usar um exemplo um jogo que possui uma nave que se move. Se a abstração representa uma nave, então estamos representando um objeto, um conceito, logo, devemos usar uma classe abstrata. Mas se a abstração representa algo que se move, independente do que seja, apenas informando o comportamento de movimento, então estamos falando de uma interface, pois não estamos abstraindo um objeto, estamos abstraindo um comportamento. (GUERRA, 2014)

6.5.1.8 Polimorfismo

Segundo Guerra (2014), o polimorfismo decorre da utilização de herança e interfaces. Não faz sentido usar herança para criar novas abstrações se o objeto não pudesse ser visto como uma dessas abstrações. Nessa perspectiva, não faz sentido existir uma interface se a classe que a implementar não pudesse ser tratada como algo que tem o comportamento dessa interface. O polimorfismo é o conceito que permite que um objeto possa ser visto como qualquer uma de suas abstrações.

6.5.1.9 Comunicação e coordenação de objetos

Objetos se comunicam através de mensagens (também conhecidas como métodos). Em uma aplicação orientada a objetos, o código coordena as ações entre objetos para executar tarefas dentro de um contexto específico. (PERRY, 2010)

6.5.1.10 Estado e comportamento

O estado e comportamento das classes estão diretamente ligados com seus atributos e métodos. Este estado pode ser definido pelo valor de um de seus atributos, e os comportamentos deste objeto são definidos pelos seus métodos. (GUERRA, 2014)

6.5.2 Projetos orientados a objetos

De acordo com Deitel e Deitel (2016), existem programas – de pequeno porte – que permitem que um desenvolvedor simplesmente ligue o computador, abra seu editor e comece a digitar código. Pode até funcionar, mas se caso precisemos desenvolver programas estruturados e com funções bem definidas, como garantimos um software consistente e sustentado pelos pilares de desenvolvimento? Como seria o trabalho em equipe? Como novos desenvolvedores poderiam compreender o software? Para criar boas soluções, precisamos seguir um processo de análise e levantamento de requisitos do projeto e desenvolver um design que atenda a estes requisitos. Este processo chama-se OOAD (*Object-Oriented Analysis and Design* – Análise e projeto orientado a objetos).

Deitel e Deitel (2016) afirma que embora existam muitas formas de realizar o OOAD, existe uma linguagem gráfica que veio a ser utilizada em aplicações de amplo espectro, vindo a ser a linguagem mais utilizada para conduzir a elaboração destes projetos. Estamos falando da UML, estudada anteriormente no capítulo 6.4.

6.5.3 A linguagem Java

Reconhecendo o crescimento do número de utilizadores de PC (*Personal Computer* – Computador Pessoal), a *Sun Microsystems* financiou, em 1991, um

projeto comandado por James Gosling, que resultou em uma linguagem de programação orientada a objetos que a empresa batizou de Java. (DEITEL e DEITEL, 2016)

Perry (2010) afirma que assim como as outras linguagens de programação, Java tem sua própria estrutura e paradigma, este último baseado no conceito de POO. Java deriva da linguagem C, assim, vemos semelhanças entre as regras de sintaxe entre ambas, como a modularização dos blocos de código em métodos, delimitação de blocos por chaves ({ }) e variáveis declaradas antes de serem usadas. Um projeto em Java é composto por pacotes, que é um mecanismo de *namespace* da linguagem Java. Dentro de cada pacote, encontramos as classes, e dentro das classes, os métodos, variáveis, constantes e outros.

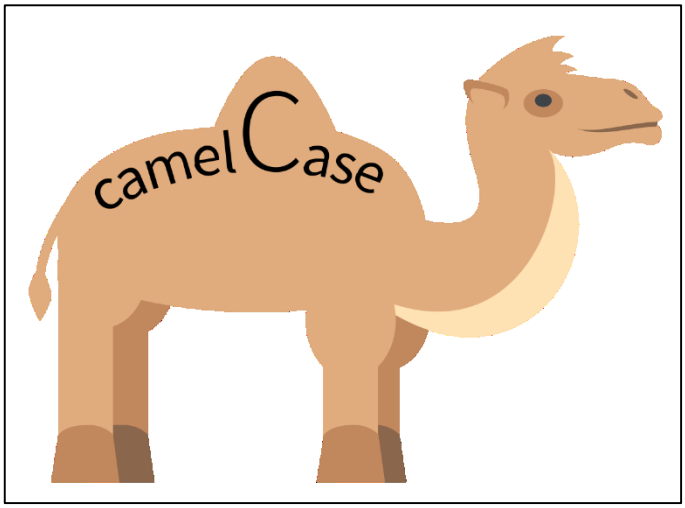
> Com a linguagem Java, é possível criar objetos de primeira classe, mas nem tudo na linguagem é um objeto. Duas qualidades diferenciam a linguagem Java das linguagens puramente orientadas a objetos, como Smalltalk. Primeiro, a linguagem Java é uma mistura de objetos e tipos primitivos. Segundo, com Java, é possível escrever código que expõe os trabalhos internos de um objeto a qualquer outro objeto que o usa.
> A linguagem Java fornece a você as ferramentas necessárias para seguir os princípios de som do OOP e produzir código orientado a objetos de som. Como Java não é puramente orientada a objetos, é necessário ter alguma disciplina sobre como escrever o código A linguagem não o força a fazer a tarefa corretamente, portanto, você deve fazê-la por si mesmo. (PERRY, 2010)

6.5.4 Principais características do Java

6.5.4.1 Convenção de nomenclatura

Java utiliza o padrão de nomenclatura *camelCase*. Este padrão determina que o nome de declarações de variáveis e métodos seja escrito sem separadores de espaços ou caracteres especiais, de maneira que a primeira palavra do nome inicie com minúsculo e cada palavra seguinte no nome tenha sua inicial em maiúsculo. Para nomear classes, deve-se capitalizar a letra inicial do nome da classe. É altamente recomendável não utilizar números nas nomenclaturas, exceto se forem realmente necessários. (PERRY, 2010)

Figura 4 - Ilustração do padrão camelCase



Fonte: https://pt.wikipedia.org/wiki/CamelCase#/media/Ficheiro:CamelCase_new.svg

6.5.4.2 Variáveis

Segundo Perry (2010), as variáveis são membros de uma classe que podem ser um atributo, um estado ou uma instanciação de objeto. Em Java, uma variável possui:

- *acessEspecifier* (Especificador de acesso): determina a visibilidade da variável. Seus possíveis valores são:
  - *public*: Qualquer objeto em qualquer pacote tem acesso a variável.
  - *private*: apenas a classe dona da variável pode vê-la.
  - *protected*: Qualquer objeto no mesmo pacote ou subpacote tem acesso a variável.
  - Nenhum identificador: apenas objetos no mesmo pacote podem ver a variável.
- *dataType* (Tipo de dado): informa o tipo de dado da variável. Pode ser um tipo primitivo, uma classe ou um objeto.
- *variableName* (Nome da variável): determina o nome da variável, recomenda-se utilizar a convenção de nomenclatura.
- *initialValue* (Valor de inicialização): Isto é opcional, você pode inicializar ou não uma variável.

Exemplo de declaração de uma variável: private int minhaIdade;

6.5.4.3 Comentários

Perry (2010) afirma que Java permite que se escreva no código apenas como uma observação, através dos comentários. Os comentários podem ser escritos com a notação vista na figura 5:

Figura 5 - Exemplos de notações para comentários

```
1  //Este é um comentário
2  /* Este é um comentário */
3  /* Este é um comentário
4  De várias linhas */
```

Fonte: PERRY, 2010. Fundamentos da Linguagem Java: Programação orientada a objetos na plataforma Java

Nota: Código original, formatado pelo autor.

6.5.4.4 Tipos de dados primitivos

O Quadro 5 exibe os tipos de dados primitivos Java, o seu tamanho e o seu valor padrão de inicialização. (PERRY, 2010)

Quadro 5 - Tipos de dados primitivos Java

Tipo	Tamanho	Valor padrão	Faixa de valores
boolean	n/a	false	true ou false
byte	8 bits	0	-128 a 127
char	16 bits	(não designado)	\u0000′ \u0000′ a \uffff’ ou 0 a 65535
short	16 bits	0	-32768 a 32767
int	32 bits	0	-2147483648 a 2147483647
long	64 bits	0	-9223372036854775808 a 9223372036854775807
float	32 bits	0,0	1.17549435e-38 a 3.4028235e+38
double	64 bits	0,0	4.9e-324 a 1.7976931348623157e+308

Fonte: https://developer.ibm.com/br/tutorials/j-introtojava1/

6.5.4.5 Operador ternário

Segundo Perry (2010), Java oferece um operador para verificações simples de if/else. Sua sintaxe é: (condição) ? estatutoSeVerdadeiro : estatutoSeFalso;

Figura 6 - Exemplo de operador ternário

```
String palavra;
int numero = 1;
palavra = (numero==1) ? um : nenhum;
```

Fonte: PERRY, 2010. Fundamentos da Linguagem Java: Programação orientada a objetos na plataforma Java

Nota: Código original, formatado pelo autor.

6.5.4.6 Coleções de Dados

Java dispõe de vários tipos de coleções. Se uma dessas coleções implementa Iterable, quer dizer que ela oferece suportes a repetições. As coleções de dados Java incluem Lists, Sets, Maps, Hashs, HashMaps, entre outros. (PERRY, 2010)

6.5.4.7 Lambdas

De acordo com Silveira e Turini (2014), a partir do Java 8, entrou o operador '->', que no Java chamamos de lambda. Como exemplo a figura 7, como a interface Consumer possui apenas um método, que é o default, com apenas uma assinatura, o Java identificou o tipo de dado pertinente à assinatura e o método a executar.

Figura 7 - Instanciando um Consumer com lambdas

```
//Sem Lambdas
Consumer<Usuario> mostrador = new Consumer<Usuario>() {
    public void accept(Usuario u) {
    System.out.println(u.getNome());
    }
};

//Com Lambda
Consumer<Usuario> mostrador = u -> System.out.println(u.getNome());
```

Fonte: SILVEIRA e TURINI, 2014. Java 8 Prático: Lambdas, Streams e os novos recursos da linguagem, p. 8, 9.

Nota: Código original, formatado pelo autor.

“Simplificando bastante, um lambda no Java é uma maneira mais simples de implementar uma interface que só tem um único método”. (SILVEIRA e TURINI, 2014)

6.5.4.8 Streams

Segundo Silveira e Turini (2014), a partir do Java 8, foi inserido a Stream, que trouxe para o Java uma maneira mais funcional de trabalhar com as coleções, além de não interferir na coleção original que está sendo manipulada. Ou seja, os resultados do uso de métodos streams não alteram os objetos de uma coleção. É possível realizar filtragens, seleções, e criar novas listas a partir de resultados de streams.

6.5.4.9 Sobrecarga de métodos

Segundo Guerra (2014), Java também permite sobrecarga de métodos. A sobrecarga de métodos permite que uma classe tenha métodos com exatamente o mesmo nome e especificador de acesso, desde que possua assinaturas diferentes. Assim, na chamada do método, o compilador sabe qual método chamar a partir do tipo de dado passado para ele.

6.5.5 Ambiente de Desenvolvimento Java

Deitel e Deitel (2016) nos mostra os passos para criação e execução de uma aplicação Java. Geralmente, este processo passa por cinco etapas: edição, compilação, carregamento, verificação e execução. Vamos ver estas etapas a seguir.

6.5.5.1 Etapas da criação até a execução de softwares Java

6.5.5.1.1 Edição

A criação de software Java se inicia com a criação e edição do código fonte, a partir de um arquivo do tipo ‘.java’, utilizando linguagem de alto nível. Geralmente, é feita utilizando uma IDE (*Integrated Development Environment* – Ambiente de Desenvolvimento Integrado). Abordaremos IDEs mais à frente neste projeto. (DEITEL e DEITEL, 2016)

6.5.5.1.2 Compilação em bytecodes

De acordo com Deitel e Deitel (2016), concluída a criação do código, podemos compila-lo para execução. Para compilar um arquivo, devemos chamar o compilador em um terminal, para que ele compile o arquivo, através do comando 'javac MeuCodigo.java'. As IDEs tipicamente possuem menus específicos para compilar e distribuir o código pronto para execução chamando o 'javac' para você. O compilador gera então os *bytecodes*, em um arquivo MeuArquivo.class. *Bytecodes* são a representação das tarefas que serão executadas na fase 5, que são executadas pela JVM (*Java Virtual Machine* – Máquina Virtual Java). A JVM simula um computador a partir da técnica chamada de virtualização. Isso quer dizer que, uma vez que um aplicativo é desenvolvido em Java, basta que esteja instalado a JVM no SO (Sistema Operacional) que o programa será executado. Diferente das instruções de máquina, que são totalmente dependentes da plataforma, instruções *bytecode* não dependem da plataforma, podendo executar então o mesmo *bytecode* em qualquer plataforma ou máquina compatível, desde que instalada a JVM. Logo, um programa Java é portátil e pode ser executado em qualquer plataforma compatível.

6.5.5.1.3 Carregamento na memória

Na fase 3, a JVM carrega o arquivo MeuArquivo.class na memória. Ele também carrega as dependências e outros arquivos .class fornecidos pelo Java que a sua aplicação usa. Os arquivos '.class' podem ser carregados de qualquer lugar, abertos no disco ou empacotados em arquivos JAR (*Java Archive* – Arquivo Java). (DEITEL e DEITEL, 2016)

6.5.5.1.4 Verificação de bytecode

Segundo Deitel e Deitel (2016), o verificador de *bytecode* verifica os *bytecodes* para garantir que eles são válidos e não oferecem riscos à segurança. O Java faz marcação pesada de segurança para certificar que os arquivos executados não causem danos ao hardware ou software.

6.5.5.1.5 Execução

Por fim, o programa é executado. Nas primeiras versões, a JVM era uma interpretadora de *bytecodes* e sua execução era lenta, pois apenas um *bytecode* era executado por vez. Com o passar do tempo, atualizações permitiram a execução de vários *bytecodes* por vez, além de que o Java implementou a compilação JIT, que basicamente, executa o código no momento exato de carregamento. Além disso, a JVM conta com o compilador HotSpot da Oracle, que identifica 'pontos quentes' do código, ou seja, por onde passa o maior fluxo de execução e os prioriza para execução mais rápida. Portanto, os programas Java passam por duas etapas de compilação: Uma em que o código é convertido para *bytecodes*, e uma em que os *bytecodes* são traduzidos para linguagem de máquina em que o programa é executado. (DEITEL e DEITEL, 2016)

6.5.5.2 Pré-Requisitos

Antes de começar a programar em Java, precisamos cumprir algumas etapas e configurar o ambiente de desenvolvimento, garantindo que a execução acontecerá corretamente em todas as etapas do processo. Para isso, devemos conferir requisitos de software, e de hardware. Vamos lista-los a seguir.

6.5.5.2.1 Kit de Desenvolvimento Java

A primeira é baixar e instalar o JDK (*Java Development Kit* – Kit de Desenvolvimento Java). Há várias versões de trabalho do JDK, as mais conhecidas hoje são as versões 11 LTS (*Long Therm Support* – Suporte de Longo Período) e a versão 8. O JDK pode ser obtido através dos links:

- JDK 11 LTS: <https://www.oracle.com/br/java/technologies/javase-jdk11-downloads.html>
- JDK 8: <https://www.oracle.com/br/java/technologies/javase/javase-jdk8-downloads.html>

O JDK já conta com o JRE (*Java Runtime Environment* – Ambiente de Tempo de Execução Java), que é o software que executa as aplicações Java.

Também é comum vermos os termos Java SE (*Standart Edition* – Edição Padrão) e Java EE (*Enterprise Edition* – Edição Empresarial). O quadro 6 demonstra a diferença entre as siglas pertinentes ao universo Java.

Quadro 6 - Termos técnicos e siglas Java

Termo	Java RE	Java SE	Java EE	JDK
Quem precisa	Usuários que executam aplicativos Java.	Desenvolvedores que criam apps comuns com a tecnologia Java.	Desenvolvedores que criam apps corporativas com a tecnologia Java.	Desenvolvedores que criam apps com a tecnologia Java.
O que é	Um ambiente necessário para execução de aplicativos java.	Um kit de desenvolvimento de software para criar apps que utilizam Java.	Ambiente para criar soluções corporativas utilizando a tecnologia Java.	Kit necessário ao desenvolvimento de sistemas que utilizam a tecnologia Java.

Fonte: https://www.java.com/pt-BR/download/help/techinfo_pt-br.html

Nota: O quadro foi criado pelo autor utilizando informações originais contidas na fonte.

6.5.5.2.2 Variáveis de ambiente

É possível que haja erros de execução ao tentar executar algum programa em java mesmo com a JDK instalada. Caso veja uma mensagem de erro como '*Exception in thread "main" java.lang.NoClassDefFoundError: SuaClasse',* é necessário ajustar a variável de ambiente CLASSPATH em seu sistema. Na plataforma Windows, localize a variável CLASSPATH e edite o valor dela para incluir o diretório local. No Windows, adicione '.;' ao início do valor CLASSPATH, sem espaços antes ou depois destes caracteres. Em outras plataformas, utilize o separador de caminho adequado – geralmente substituindo ';' (ponto e vírgula) por ':' (dois pontos). (DEITEL e DEITEL, 2016).

Deitel e Deitel (2016) sugere que para outros erros, é possível que seja necessário configurar a variável JAVA_HOME. Crie uma variável de ambiente com o nome JAVA_HOME, e para seu valor, defina o caminho do subdiretório *bin* do diretório de instalação do JDK.

6.5.5.2.3 Ambientes de Desenvolvimento Integrados

Já é possível desenvolver aplicações com estas ferramentas até aqui listadas, mas a comunidade de desenvolvimento desfruta de funcionalidades adicionais proporcionadas pelas IDEs. (PERRY, 2010)

Segundo Deitel e Deitel (2016), as três IDEs mais conhecidas são NetBeans, Eclipse e IntelliJ IDEA. Estas IDEs podem ser obtidas através dos links:

- NetBeans: <https://netbeans.apache.org/download/index.html>
- Eclipse: <https://www.eclipse.org/downloads/>
- IntelliJ IDEA: <https://www.jetbrains.com/pt-br/idea/download/>

6.5.5.2.4 Requerimentos de Sistema

Segundo Perry (2916), Java é suportado pelos sistemas operacionais Linux, Windows, Solaris e Mac OS X. A configuração do sistema requerida está demonstrada no quadro 7.

Quadro 7 - Requerimentos de hardware para Java

Memória mínima	**Memória recomendada**	**Espaço em disco mínimo**	**Espaço em disco recomendado**
1 GB	2 GB	250 MB livres	500 MB livres

Fonte: https://docs.oracle.com/cd/E19226-01/821-1337/abpaj/index.html

Nota: Adaptado pelo autor.

6.6 BANCO DE DADOS

Houve um tempo onde a definição de banco de dados não existia, tampouco DBMSs (*Data Base Management System* - Sistema Gerenciador de Banco de Dados), também chamados de SGBDs em nossa língua, para este fim. Os dados eram armazenados em planilhas e documentos salvos no computador ou em fitas ou disquetes. Para é época, esta era uma evolução, mas ainda eram muito suscetíveis a erros e falhas e isso trazia grandes problemas, como dados duplicados, informações inconsistentes e até arquivos corrompidos, sem falar na degradação das fitas e disquetes quer seja pela ação do tempo ou de mau uso. Outra dificuldade é que os

dados não se relacionavam, então havia pouco o que se fazer para utilizá-las nas estratégias de negócio. Era complicado cruzar informações do negócio para medir produtividade ou obter auxílio à tomada de decisão. (GONÇALVES, 2014)

Podemos ter uma noção de como eram as coisas antes do surgimento dos bancos de dados, cujo objetivo era justamente trazer organização e consistência, algo que não acontecia no passado. Com uma implementação utilizando técnicas para modelagem dos dados e uma linguagem própria de consultas, foi possível criar um ambiente com segurança, confiabilidade, disponibilidade e desempenho. (GONÇALVES, 2014)

Um banco de dados é um conjunto de dados estruturados e organizados de maneira a garantir acesso e gerenciamento fácil e rápido. O modelo mais conhecido e utilizado é o banco de dados relacional, onde os dados são armazenados em tabelas, estas por sua vez incluem registros e campos. Como exemplo, podemos ter uma tabela de alunos que tem os campos nome, sobrenome, curso, notas, entre outros. Podemos pesquisar, ordenar, e manipular os dados da maneira que precisam ser processadas. Como exemplo, uma faculdade pode usar dados do banco de dados dos alunos para combinar com dados de outros cursos, ou fazer buscas no banco de dados por dados específicos desejados, etc. (DEITEL e DEITEL, 2016)

Um banco de dados relacional incorpora as características da modelagem relacional, como representação de entidades, atributos e relacionamentos. As entidades são representadas no banco de dados por tabelas. Cada linha de uma tabela contém suas colunas, ou campos, que são os atributos dessa entidade. Para que haja relacionamentos entre as entidades, precisamos instituir o conceito de chaves. Temos duas chaves principais em um banco de dados: PK (*Primary Key* – Chave primária) e FK (*Foreign Key* - Chave estrangeira). (GONÇALVES, 2014)

Segundo Gonçalves (2014), quando uma coluna é definida como PK, nenhuma linha desta coluna pode conter valores repetidos de outra linha. Esta PK é geralmente o identificador de cada linha em uma tabela. As chaves PK devem se adequar ao conceito de minimalidade, ou seja, quanto menos PKs em uma tabela, melhor. Geralmente, é necessário apenas uma PK para identificar as linhas de uma tabela, mas podem conter mais PKs. Já as FKs são uma coluna ou colunas que o seu valor faz referência a uma PK de outra tabela. É através de uma FK que criamos os relacionamentos de uma tabela para com outra em bancos de dados relacionais.

Para garantir a integridade das informações e do cumprimento das regras de bancos de dados, é necessário um SGBD. Ele é o sistema que controla e manipula todos os acessos do banco de dados. (GONÇALVES, 2014)

6.6.1 Linguagem SQL

A SQL (*Structured Query Language* – Linguagem de consulta estruturada) é a linguagem padrão criada para as comunicações com bancos de dados. Foi desenvolvida nos laboratórios da IBM (*International Business Machines* – Máquinas Empresariais Internacionais) em 1970, chamada de SEQUEL, e depois alterada para SQL. Em 1986, a ANSI (*American National Standards Institute* – Instituto Americano de Padrões Nacionais) estabeleceu um padrão chamado SQL1, recebendo vários aprimoramentos até chegar hoje no padrão SQL3, que trouxe como inovações a criação de funções, regras, procedimentos, armazenamento de imagens e áudios em tabelas, gatilhos, códigos em outras linguagens, entre outros. (GONÇALVES 2014)

6.6.2 Comandos SQL

Segundo Carvalho (2015), há três categorias de comandos SQL. São elas:

- DML (*Data Manipulation Language* – Linguagem de Manipulação de Dados)
- DDL (*Data Definition Language* – Linguagem de Definição de Dados)
- DCL (*Data Control Language* – Linguagem de Controle de Dados)

6.6.2.1 DML

Comandos usados para que o SGBD execute uma ação como inserir, excluir, recuperar e editar registros no banco de dados. São eles, respectivamente: ‘INSERT’, ‘DELETE’, ‘SELECT’ ou ‘LOCK’ e ‘UPDATE’. (CARVALHO, 2015)

6.6.2.2 DDL

Comandos DDL são usados para manipular as entidades e objetos do banco de dados. São eles: ‘CREATE TABLE’, ‘CREATE INDEX’, ‘ALTER TABLE’, ‘DROP TABLE’, ‘DROP VIEW’ e ‘DROP INDEX’. (CARVALHO, 2015)

6.6.2.3 DCL

Utilizados para controlar os acessos, sessões e transações do SGBD. Alguns deles são: ‘COMMIT’, ‘ROLLBACK’, ‘GRANT’ e ‘REVOKE’. (CARVALHO, 2015)

6.6.3 Funções SQL

De acordo com Carvalho (2015), os SGBDs possuem funções que possibilitam fazer vários tipos de operações em comandos SQL, como cálculos, manipular Strings, trabalhar com datas, entre outras funções. Alguns SGBDs possuem funções nativas, que não podem ser utilizadas em outro SGBD. Algumas funções SQL são:

- ‘GROUP BY’: Utilizado para agrupar dados obtidos em tabelas relacionadas com base em critérios;
- ‘MAX’ e ‘MIN’: Utilizados para obter o valor máximo e mínimo de uma consulta;
- ‘SUM’: Utilizado para obter a soma de valores de uma consulta;
- ‘AVG’: Utilizado para obter uma média de valores de uma consulta;
- ‘LENGTH’: Utilizado para obter o comprimento de uma *string*;
- ‘CONCAT’: Utilizado para concatenar strings de dois ou mais campos;
- ‘LOWER’: Utilizado para obter as strings em caixa baixa;
- ‘ROUND’: Utilizado para arredondar o valor de um decimal para a quantidade de casas decimais desejada;
- ‘TRUNCATE’: Utilizado para omitir as casas decimais;

6.6.4 PostgreSQL

O PostgreSQL é um SGBDOR (Sistema de Gerenciamento de Banco de Dados de Objeto Relacional) que é tido como um dos mais avançados já criados. Nascido em 1982, na Universidade de Berkerley, na Califórnia, Estados Unidos, como parte do projeto Postgres, sua primeira versão foi entregue ao mercado em 1989. Desde então foram lançadas muitas outras versões mantendo o projeto como *open-source*. Suas principais vantagens são relacionadas à economia proporcionada e alto desempenho. (UOL MEU NEGÓCIO, 2020)

De acordo com o UOL Meu Negócio (2020), como caso de sucesso, podemos citar o DETRAN (Departamento Estadual de Trânsito) do Ceará, Brasil, que migrou seu banco de dados para o PostgreSQL e alcançou a marca de 1,7 milhão de reais em economia com licenças e terceirização de serviços de suporte técnico. O SBGD também é o oficial do governo do Ceará.

“PostgreSQL é um banco de dados poderoso, orientado a objetos e de código aberto que usa e estende a linguagem SQL combinada com vários recursos que armazena e escala dados com segurança nos mais complicados fluxos de dados”. (POSTGRESQL, 2021. Traduzido por Fabrício Gustavo)

Segundo Caton (2018), a comunidade de desenvolvedores concorda que PostgreSQL vem se consolidando cada vez mais como a melhor opção graças a vários fatores que o colocam neste patamar. Vamos listar aqui alguns motivos:

- Transações a nível de tabela: Transações são muito úteis para realizar ações perigosas, como apagar registros. PostgreSQL oferece transações a nível de tabela, permitindo realizar transações mais específicas em entidades.
- Algumas funcionalidades: PostgreSQL oferece suporte a buscas do tipo *full-text*, baseadas completamente textuais. Também dispõe de um recurso exclusivo entre SBGDs chamado UUID (*Universal Unique Identifier* – Identificador Único Universal), que é um identificador de informações em sistemas de computação.
- Encoding padrão: A codificação padrão do PostgreSQL já é a UTF-8.
- Tipos de dados: PostgreSQL oferece colunas que guardam tipos de dados específicos de maneira consistente, além de permitir que sejam criados tipos de dados. Como exemplo, um dado booleano é realmente salvo como tipo *boolean* no PostgreSQL, ao passo que em outros SGBDs são gravados com adaptações para outros tipos, como *integer*, sendo representados por 1 e 0 para verdadeiro ou falso.
- Comunidade e licença: PostgreSQL sempre foi *open-source*, apesar de possuir uma licença própria.
- Custos: PostgreSQL oferece todo seu software e suas funcionalidades por um total de 0 custos.

Estes são apenas alguns fatores que tornam o PostgreSQL a escolha para diversos programadores quando é chegada a hora de escolher um SGBD. (CATON, 2018)

Os manuais, tutoriais, comandos e funções SQL e outras informações sobre banco de dados podem ser obtidas através da documentação do PostgreSQL, disponível em <https://www.postgresql.org/docs/>. (POSTGRESQL, 2021)

6.7 CONTROLE DE VERSÃO

Programar é uma tarefa que busca equilibrar o tempo com a qualidade. Há sempre uma demanda por atualizações, mas temos problemas com alterar código, mesmo que ele não tenha mais utilidade. O problema é que se removermos uma parte do código mesmo que desnecessária, perderemos parte do histórico do código, e manter cópias do projeto para comparação pode trazer confusão e acidentes, sem falar na poluição de código. Além disso, ainda há o desafio com o trabalho em equipe. Construir um sistema em equipe já tem suas dificuldades, como a necessidade do código se integrar de forma transparente e não ter emendas, contar com detecção de conflitos, pois anotar tudo em planilhas ou outros documentos é muito trabalhoso e propenso a erros. (AQUILES; FERREIRA, 2014)

Existem ferramentas que funcionam como histórico de código e mecanismo de integração para a equipe. Com elas, podemos acompanhar todas as alterações do código, detectar e mesclar alterações no mesmo arquivo e identificar os conflitos, tudo de forma automática. Elas se chamam sistemas de controle de versão. Cada mudança, versão ou atualização que geramos no código devemos armazenar no repositório desse sistema para manter este controle de versionamento. Alterações de colegas ao código são integradas automaticamente sempre que possível e conflitos são gerenciados se necessário cada vez que há colaborações de colegas de equipe. (AQUILES; FERREIRA, 2014)

O Git é uma ferramenta que executa estas tarefas com excelência. Surgiu em 2005, criado por Linus Torvalds, por estar descontente com os controles de versão disponíveis na época. Logo em 2008, foi criado o GitHub, que é uma aplicação web que permite armazenar repositórios Git online, além de ser uma rede social para desenvolvedores. Muitos projetos conhecidos, como jQuery, Node.js, Spring, jUnit e outros são armazenados no GitHub. (AQUILES; FERREIRA, 2014)

Segundo Aquiles e Ferreira (2014, p. 3), “Atualmente, conhecer bem como utilizar o Git é uma habilidade importante para uma carreira bem-sucedida no desenvolvimento de software. ”

O Git possui um repositório local onde são gravadas as informações de controle de versão. Para que o projeto seja disponibilizado online, é necessário criar um repositório remoto no GitHub, que receberá as atualizações do repositório local. O GitHub está disponível pelo endereço www.github.com. (AQUILES; FERREIRA, 2014)

6.7.1 Configurando um repositório Git

É importante lembrarmos que Git é o sistema de controle de versões que interagimos por linha de comando, através do *GitBash*, que é o seu terminal. GitHub é a rede social para desenvolvedores onde podemos armazenar os repositórios de forma online. (AQUILES; FERREIRA, 2014)

Para começarmos com o Git, devemos, pelo *GitBash*, navegar até o diretório de nosso projeto. Ao chegarmos nele, usamos o comando ‘git init’ para que o repositório seja iniciado a partir daquele diretório. Todos os arquivos, diretórios e subdiretórios dentro deste diretório serão adicionados ao repositório Git. (AQUILES; FERREIRA, 2014)

Vamos verificar o estado do repositório com o comando ‘git status’. Caso haja arquivos com o estado *untracked*, significa que ele não está rastreado pelo Git, ou seja, não será incluso nas modificações salvas. Para adicionarmos o projeto completo ao repositório, no diretório raiz do projeto, usamos o comando ‘git add .’. O ponto representa todos os arquivos que não estão rastreados no diretório. Para adicionarmos um arquivo específico, navegamos até seu diretório e usamos o comando ‘git add nomedoarquivo’. (AQUILES; FERREIRA, 2014)

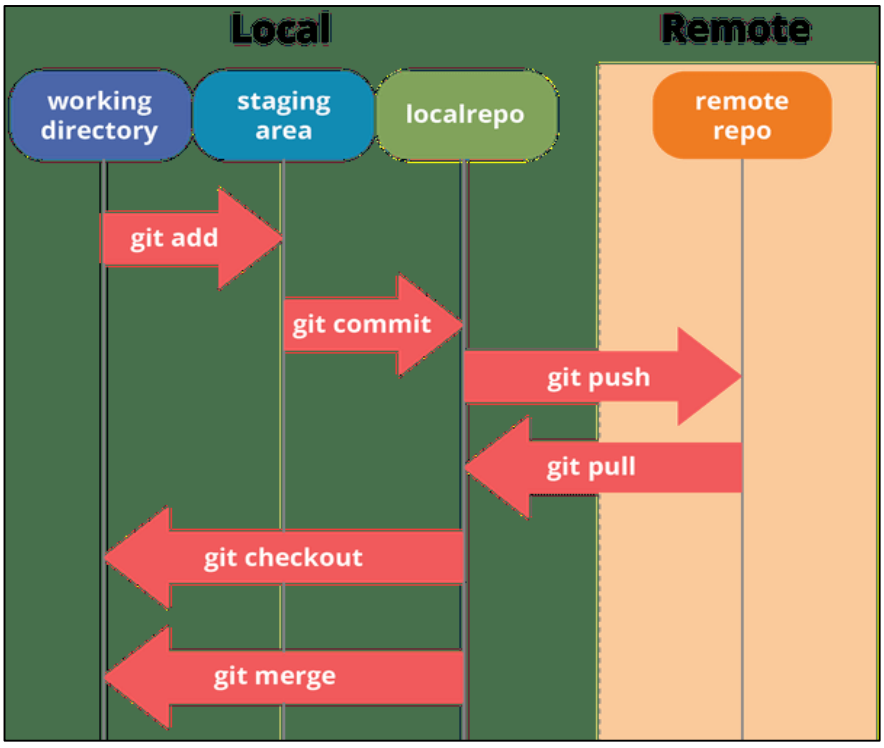
6.7.2 Principais comandos e fluxo de trabalho do Git e GitHub

Na figura 8, podemos acompanhar o fluxo principal de atividades com o Git e GitHub. Na área de trabalho Local, temos o *working directory*, que representa o diretório do projeto, *staging area*, que é onde o Git armazena temporariamente as modificações antes de serem passadas ao *localrepo*, que é onde ficam salvas as informações de cada versão. Representado pelo *Remote*, vemos o repositório remoto

do GitHub, que vai manter nossos arquivos online. Quando informamos ao Git que há modificações na estrutura ou no código com o comando ‘git add’, que também é utilizado para adicionar informações ao *commit*, o Git salva essas informações no *stage*. Quando todas as atualizações e inserções forem passadas para o *stage*, é hora de usarmos o comando ‘git commit -m “nome do *commit*”’. Isso salvará as mudanças no repositório local. (AQUILES; FERREIRA, 2014)

Para vermos o histórico de mudanças (*commits*), utilizamos o comando ‘git log’. Cada *commit* é representado por um código hash. Para vermos as diferenças entre arquivos de diferentes versões, utilizamos o comando ‘git diff hashv1..hashv2’. Teremos uma saída no terminal com as diferenças entre estes arquivos desde a versão hashv1 até a versão hashv2. (AQUILES; FERREIRA, 2014)

Figura 8 - Fluxo de trabalho e comandos do Git e GitHub



Fonte: https://ilegra.com/blog/do-zero-git-github-como-criar-primeiro-repositorio-e-subir-seu-primeiro-projeto/

Podemos enviar o repositório para o GitHub. Primeiro, precisamos informar o Git as configurações de acesso ao GitHub. Após realizar o *commit*, devemos ir ao site do GitHub e criar uma conta. Após criar a conta, acessamos o perfil e criamos o repositório clicando no botão ‘Novo Repositório’. Após criar o repositório, o GitHub nos dará o endereço para inserirmos no GitBash e configurar a conexão. Para configurar

a conexão, utilizamos o comando ‘git remote add origin linkDoRepositorio’. O Git irá solicitar a autenticação com seu usuário e senha, e a conexão está configurada. Hora de enviar os arquivos para o repositório remoto. Para isso, nós utilizamos o comando ‘git push -u origin main’. Pronto. Chegamos ao fim do ciclo de armazenar uma atualização nos repositórios do Git e GitHub. (AQUILES; FERREIRA, 2014)

Também podemos usar o comando ‘git push’ para recuperar uma versão do repositório remoto para o repositório local, e o comando ‘git checkout’ para trazer a versão do repositório git para o nosso diretório de trabalho caso necessário. O comando ‘git merge’ vai combinar vários *commits* em uma versão única e trazer para o nosso diretório de trabalho. (AQUILES; FERREIRA, 2014)

O Git também permite baixar um *commit* do GitHub diretamente para o diretório de trabalho vazio, caso queiramos começar a trabalhar com um projeto já em andamento. Para isso, configuramos o repositório remoto e usamos o comando ‘git clone’. Após isso, o diretório conterá os arquivos da última versão salva no GitHub. (AQUILES; FERREIRA, 2014)

Embora seja possível utilizar o Git exclusivamente por linha de comando, também podemos usar uma interface gráfica para isso. Existem diversas aplicações visuais para o Git disponíveis para *download* que facilitam o trabalho para quem não está acostumado a utilizar terminais. (AQUILES; FERREIRA, 2014)

6.8 SPRING FRAMEWORK

Spring Framework é um framework desenvolvido pela Pivotal para atender a demanda empresarial de desenvolvimento de software na plataforma Java, baseado nos conceitos de IoC (*Inverson of Control* – Inversão de Controle) e DI (*Dependencies Injection* - injeção de dependências). (WEISSMANN, 2014)

> Mais que um framework para desenvolvimento de aplicações corporativas, vejo o Spring como uma ferramenta disciplinadora. Conforme o desenvolvedor vai se habituando ao seu modo de trabalho começa a valorizar ainda mais qualidades como uma melhor modularização do sistema, escrita de código mais simples, reaproveitamento de código legado e tecnologias já existentes, além da criação de interfaces mais significativas. (WEISSMAN, 2014).

Para Walls (2016, traduzido por Fabrício Gustavo), Spring Framework é uma ferramenta que os desenvolvedores Java anseiam já de longos tempos atrás. Ele contém recursos que são fáceis de aplicar e operações “fora da caixa” que fazem com que programar em Java seja divertido.

Há quatro dificuldades essenciais que se aplicam ao desenvolvimento de todo software: Conformidade, que aponta que um software deve ser compatível com qualquer ambiente que será executado; Invisibilidade, que apesar de haver notações como a UML, é difícil visualizar um software a menos que seja a documentação de seu projeto; Mutabilidade, que se refere a alteração constante dos requisitos de software e readaptação do software; e a Complexidade, que está diretamente relacionada aos limites do nosso intelecto, que faz referência a complexidade das regras de negócio e integrações que o software necessita ao longo de seu desenvolvimento. (BROOKS, 1986, apud WEISSMANN, 2014)

Então, chegamos ao Spring Boot. Por mais que ao decorrer da história do Spring Framework houvessem lançamentos maravilhosos de recursos novos, Spring Boot foi uma novidade empolgante, como uma mágica. Spring Boot é capaz de detectar qual tipo de aplicação estamos desenvolvendo e realizar toda sua configuração automaticamente, bem como os componentes necessários para sua execução. Não é necessária configuração explícita por parte do desenvolvedor, o Spring Boot configurará tudo para você. O Spring Framework fica cada dia melhor, simplificando o processo de desenvolvimento. (WALLS, 2016, traduzido por Fabrício Gustavo)

6.8.1 Inversão de Controle e Injeção de Dependências

“[...] Módulos de alto nível não devem depender de módulos de baixo nível, apenas devem depender de abstrações e Abstrações não devem depender de detalhes e sim detalhes devem depender de abstrações”. WEISSMAN (2014, p. 9)

Módulos de alto nível são as partes do código que realmente importam para o projeto, as regras de negócio. Os módulos de operação no banco de dados estão em nível mais abaixo, pois não se referem exatamente a lógica do negócio, dando suporte a classe principal. (WEISSMAN, 2014)

Para Weissman (2014), a Inversão de Controle se refere ao benefício do Spring Boot que ao invés de o desenvolvedor haver de configurar todos os componentes do

sistema, o próprio Spring Boot já os configura, reconhecendo de forma inteligente os componentes e configurações, daí o termo inversão de controle, o controle passa do programador para o framework. Mas na verdade, todo framework realiza a inversão de controle, porém o que o Spring Framework faz que o torna referência, é a especialização dessa inversão de controle através da injeção de dependências. Tomemos como exemplo o código da figura 9:

Figura 9 - Trecho de código para análise de IoC e DI

```
1  public class Integrador {
2
3      public Integrador() throws Exception {
4          Properties configuracao = new Properties();
5          InputStream stream = new FileInputStream("configuracao.properties");
6          configuracao.load(stream);
7
8          String nomeOrigem = configuracao.getProperty("nome_origem");
9          String nomeDestino = configuracao.getProperty("nome_destino");
10
11         ServiceLocator locator = new ServiceLocator()
12         setOrigem(locator.getOrigem(nomeOrigem);
13         setDestino(locator.getDestino(nomeDestino);
14     }
15     //Demais métodos
16 }
```

Fonte: WEISSMAN, 2014. Vire o jogo com Spring Framework, p. 11.

Nota: Código original, formatado pelo autor.

Não precisamos nos preocupar em como o ServiceLocator encontra as instâncias, ele simplesmente as encontra. Podemos ver aqui o conceito de injeção de dependências, onde o Spring Boot injeta todas as dependências necessárias ao código, através da inversão de controle que lhe é atribuída como framework.

6.8.2 Ambiente de Desenvolvimento Integrado

A Pivotal oferece uma IDE com várias ferramentas de desenvolvimento, entre elas, interface gráfica de operação no Git e GitHub, gerenciamento de dependências com Maven, servidor web embarcado Tomcat, entre outros. A IDE se chama STS (*Spring Tools Suite* – Suíte de Ferramentas Spring), desenvolvida a partir da perspectiva da IDE Eclipse. No entanto, o STS pode ser utilizado como complemento em outras IDEs, como o Microsoft Visual Studio Code e no Theia IDE. (SPRING, 2021)

“Spring Tools 4 é a próxima geração de ferramentas Spring para o seu ambiente de desenvolvimento favorito”. (SPRING, 2021)

6.8.3 Acesso a dados

A esmagadora maioria das aplicações requerem alguma forma de banco de dados. Ainda que essa seja uma interação cotidiana dos desenvolvedores, a implementação do acesso ao banco de dados traz algumas dificuldades e consome várias horas de trabalho e desenvolvimento. O Spring oferece várias ferramentas de acesso a dados, e uma delas é a JPA (*Java Persistence API (Algorithm programing interface* – Interface de programação de algoritmo) – API Java de Persistência). (WEISSMAN, 2014)

JPA faz parte da especificação JSR (*Java Specifications Request* – Requisição de Especificações Java) 220, que tem como objetivos principais simplificar a criação, gerenciamento e armazenamento de Beans (componentes de software), e a JPA é uma API que auxilia na persistência de dados. Os principais benefícios de utilizar a JPA são:

- Não ser mais necessário criar complexos DAOs (Data Access Object – Objeto de Acesso a Dados);
- A API ajuda no gerenciamento das transações;
- Os padrões de código se aplicam a qualquer banco de dados, não sendo necessário usar código específico de marcas;
- Pode se usar tanto SQL quanto JPQL (Java Persistence Query Language – Linguagem de Consulta de Persistência Java);
- Utilizar a API para aplicações Desktop. (O’CONNER, 2007, traduzido por Fabrício Gustavo.)

O Spring conta em seu framework com o Spring Data, que fornece interfaces de repositórios que acessam o banco de dados através da JPA, com métodos que geram automaticamente as queries, sem a necessidade de criar as consultas manualmente, mas se necessário, é possível criar queries com a anotação @Query. Estes métodos estão representados no quadro 8. SPRING (2021, traduzido por Fabrício Gustavo)

Quadro 8 - Estatutos SQL suportados em métodos Spring Data

SQL	**Método Exemplo da interface JPArepository**
Distinct	findDistinctByLastnameAndFirstname
And	findByLastnameAndFirstname
Or	findByLastnameOrFirstname
Is, Equals	findByFirstname, findByFirstnameIs, findByFirstnameEquals
Between	findByStartDateBetween
LessThan	findByAgeLessThan
LessThanEqual	findByAgeLessThanEqual
GreaterThan	findByAgeGreaterThan
GreaterThanEqual	findByAgeGreaterThanEqual
After	findByStartDateAfter
Before	findByStartDateBefore
IsNull, Null	findByAge(Is)Null
IsNotNull, NotNull	findByAge(Is)NotNull
Like	findByFirstnameLike
NotLike	findByFirstnameNotLike
StartingWith	findByFirstnameStartingWith
EndingWith	findByFirstnameEndingWith
Containing	findByFirstnameContaining
OrderBy	findByAgeOrderByLastnameDesc
Not	findByLastnameNot
In	findByAgeIn(Collection<Age> ages)
NotIn	findByAgeNotIn(Collection<Age> ages)
True	findByActiveTrue()
False	findByActiveFalse()
IgnoreCase	findByFirstnameIgnoreCase

Fonte: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#reference

O Spring cria automaticamente as tabelas e relacionamentos com base em anotações nas classes de domínio. As classes entidades devem ser anotadas com @Entity. Cada classe de domínio anotada com @Entity se transformará em uma

tabela. As relações são estabelecidas com as anotações @OneToMany, @ManyToOne e @OneToOne. (O'CONNER, 2007, traduzido por Fabrício Gustavo.)

6.8.4 Apache Maven

De acordo com o Apache Maven Project (2021), Apache Maven é um *plugin* do Spring que gerencia a estrutura do projeto. Ele utiliza o conceito POM (*Project Object Model* – Projeto Modelo de Objeto) para gerir os objetos, dependências externas, documentação, plug-ins, informações do projeto e de *build*, tudo através do arquivo pom.xml. Através da inserção de dependências neste arquivo, o Maven é capaz de adicionar os arquivos necessários da dependência ao projeto, disponibilizando-os para uso na estrutura do projeto. Os benefícios de uso do Maven como gerenciador de projetos incluem:

- Configuração simples de um projeto, os quais já incluem boas práticas;
- Consistente para uso entre vários projetos ao mesmo tempo;
- Gerenciamento superior de dependências, que inclui suas atualizações e gestão de dependências transitivas;
- Um repositório riquíssimo em ferramentas e bibliotecas de grande variedade;
- Extensível, permitindo desenvolvimento de plug-ins em Java ou scripts;
- Acesso instantâneo às novas ferramentas instaladas com pequena ou zero configuração extra;
- Capacidade de gerar projetos executáveis como JAR, WAR (*Web application Archive* – Arquivo de aplicação Web), ou outras formas de distribuição, sem necessidade de scripts adicionais na maioria dos casos;
- Gerenciamento automático de dependências: Maven faz download automático de todos os arquivos necessários de uma dependência de seu repositório padrão, e disponibiliza uso de outros repositórios.

6.9 JAVAFX

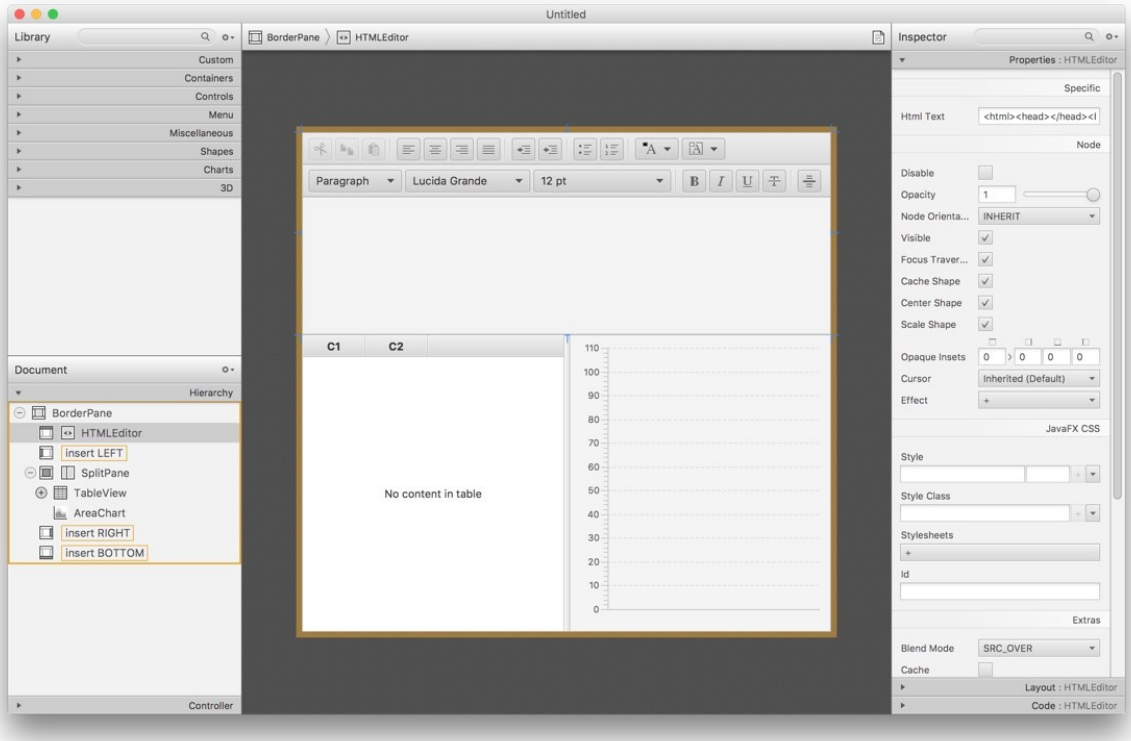
Segundo Grinev (2018. Traduzido por Fabrício Gustavo), Java dispõe das bibliotecas Swing e AWT (*Advanced Windows Toolkit* – Ferramentas avançadas do Windows) para criação de uma interface gráfica de sistemas, mas elas já se provaram

ultrapassadas e com problemas de compatibilidade bem conhecidos. JavaFX é um conjunto de bibliotecas adicionadas ao Java para criação de interfaces gráficas modernas e ricas em recursos, com uma variedade de formas, controles e objetos gráficos.

> JavaFX é uma plataforma de código aberto, a próxima geração de aplicações para área de trabalho, móveis e sistemas embutidos criados em Java. É um esforço conjunto de vários indivíduos e companhias com o objetivo de produzir um conjunto de ferramentas moderno, eficiente e completo em recursos para desenvolver aplicações ricas para clientes. (JAVAFX, 2021, traduzido por Fabrício Gustavo).

JavaFX, por ser escrito como uma API, é capaz de acessar e referenciar qualquer outra API desenvolvida para Java, podendo acessar capacidades nativas do sistema e se conectar a outros serviços. A interface criada no JavaFX pode ser completamente personalizada de forma separada do desenvolvimento da estrutura, utilizando Folhas de estilos cascata, conhecidas como CSS (*Cascading Style Sheets*). Ainda é possível desenvolver uma aplicação separada do *back-end* ou até mesmo uma arquitetura monolítica utilizando arquivos de linguagem de marcação FXML (*Format Extensible Markup Language* – Formato de Linguagem Extensível de Marcação) e utilizar Java para implementar a lógica. É possível utilizar o programa *Scene Builder* para editar os FXML. *Scene Builder* conta com ferramentas para editar todas as formas e elementos gráficos de uma aplicação JavaFX, e também configurar a ligação entre os métodos dos controladores para chamar as funções pertinentes aos requisitos e regras de negócio. (JAVA DOCUMENTATION, 201-?, traduzido por Fabrício Gustavo)

Figura 10 - Scene Builder em ação



Fonte: https://gluonhq.com/products/scene-builder/

6.10 PADRÕES DE DESENVOLVIMENTO DE SOFTWARE

Christopher Alexander, engenheiro civil, criou o que consideramos o primeiro padrão de projeto na década de 1970. Um padrão de projeto é uma solução documentada que possa auxiliar na resolução de problemas pertinentes à estrutura de projetos de variados tipos e escopos. Foi a partir do trabalho de Alexander que vários desenvolvedores e a comunidade de software criaram padrões de projetos e os documentaram para que fossem utilizados pelo restante dos profissionais e estudantes da área. (SOUZA, 2011)

“O uso de um padrão de projeto tem como intuito seguir um modelo pré-determinado de desenvolvimento que busca a melhor adaptação às necessidades dos desenvolvedores. ” (SOUZA, 2011)

É comum desenvolvedores entrarem em um primeiro contato com um determinado padrão e ver que já fez uso da solução anteriormente, até exclamando ‘*eu já usei isso e nem sabia que era um padrão! ’.* Isso é normal, o próprio nome ‘padrão’ quer dizer que essa é uma solução utilizada que trouxe sucesso para diversos contextos. (GUERRA, 2014)

Vamos abordar alguns padrões de desenvolvimento que guiaram algumas etapas do desenvolvimento de software nesta seção.

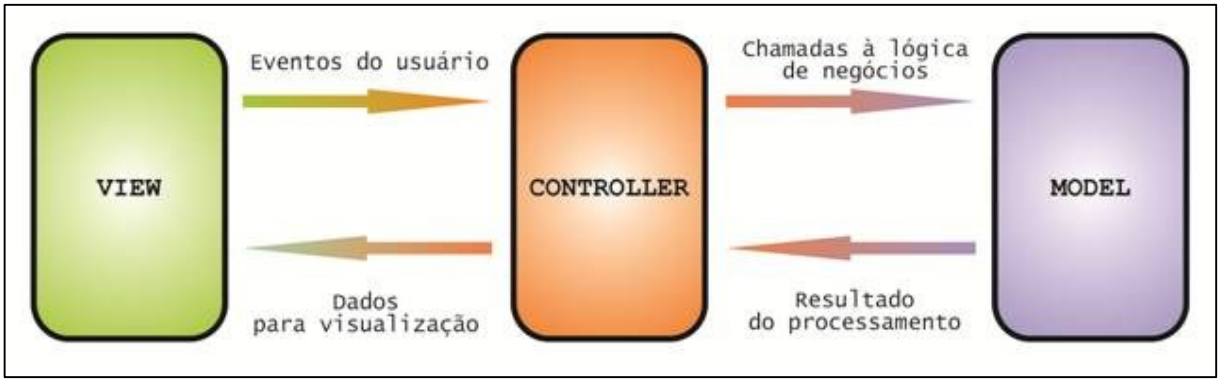
6.10.1 O padrão MVC

Segundo Souza (2011), em 1979, Trygve Reenskaug concebeu o que viria a ser o Padrão MVC (*Model-View-Controller* – Modelo-Visualização-Controlador). A idéia dele, apresentada no artigo "*Applications Programming in Smalltalk-80: How to use Model-View-Controller*", criou um padrão que tinha como objetivo separar o projeto em três camadas independentes, como o nome sugere. Assim, o código fica mais limpo, reduz o acoplamento de classes e aumenta a coesão delas. Isso faz com que seja muito mais fácil dar manutenção no código e evitar boilerplate, aprimorando a reutilização de código.

> De acordo com Souza (2011), acoplamento: é o grau em que uma classe conhece a outra. Se o conhecimento da classe A sobre a classe B for através de sua interface, temos um baixo acoplamento, e isso é bom. Por outro lado, se a classe A depende de membros da classe B que não fazem parte da interface de B, então temos um alto acoplamento, o que é ruim. Coesão: quando temos uma classe elaborada de forma que tenha um único e bem focado propósito, dizemos que ela tem uma alta coesão, e isso é bom. Quando temos uma classe com propósitos que não pertencem apenas a ela, temos uma baixa coesão, o que é ruim.

O padrão MVC pode ser utilizado em diversos tipos de projetos, como por exemplo aplicações *Desktop*, *web*, *mobile*, *cross platform*. O padrão MVC, tem como seu principal benefício isolar as regras de negócio e dados da lógica de visualização, a interface do usuário. Isso permite várias interfaces com o usuário sem que seja necessário reescrever regras de negócio, tornando o código mais flexível e promovendo reutilização entre as classes. (SOUZA, 2011).

Figura 11 - Diagrama do fluxo MVC



Fonte: https://www.devmedia.com.br/padrao-mvc-java-magazine/21995

Analisando a figura 11, compreendemos a lógica de funcionamento do padrão MVC. Mas não poderíamos conectar a *view* diretamente ao *model*, sem passar por mais uma camada *controller*? Sim, mas isso não é recomendado e quebraria a proposta do MVC. Se conectar direto a *view* ao *controller*, funções que precisariam ser utilizadas em outra interface deveriam ser reescritas, além de que o código da classe *view* comportaria mais responsabilidades, além da visualização, também lidaria com a lógica de dados. Isso aumentaria o acoplamento das classes. Assim, a camada *controller* deixa o código mais flexível e distribuído, deixando as responsabilidades de cada parte claras e específicas. (SOUZA, 2011)

6.10.1.1 Model

Segundo Souza (2011), o *model* é o motivo pela qual a aplicação foi construída. É nos pacotes de *model* que definimos as entidades, as regras e lógica de negócio e conexões com o banco de dados. O *model* terá ciência apenas dos dados armazenados no sistema e a lógica que atua sobre eles. Estes dados podem estar bem como em um banco de dados quanto em arquivos no sistema de arquivos do computador, como arquivos xml, txt, entre outros.

6.10.1.2 View

A *view* é a camada do sistema que interage com o usuário. É aqui que haverá a entrada dos dados e onde os dados serão visualizados através das interfaces. A

view não sabe a lógica de negócios, então todo processamento é feito pela *model*, então a resposta é passada para a *view* através do controlador. (SOUZA, 2011)

6.10.1.3 Controller

Já vimos que o usuário utiliza o sistema interagindo com a *view*, e que os dados e lógica do negócio é operado pelo *model*. O *Controller* é a ponte entre as duas outras camadas. As Views recebem a requisição do usuário, as passa para o *controller*, e o *controller* aciona as *models*. As *models* devolvem a requisição ao *controller*, que as repassa para a *view*. Graças a inexistência de dependência entre *view* e *model*, é possível reaproveitar a camada de modelo em outras interfaces e sistemas através do *Controller*. (SOUZA, 2011)

6.10.1.4 Vantagens e desvantagens

Segundo Souza (2011), o padrão MVC oferece muito mais vantagens do que desvantagens, ainda assim elas não podem ser ignoradas. Vamos repassar algumas das vantagens de utilização do padrão MVC:

- Separação bem definida entre as camadas do projeto;
- Manutenção fácil do sistema;
- Reaproveitamento de código;
- Alterações em camadas *view* não afetam as regras de negócio;
- Permite desenvolver e testar funções de cada camada isoladamente;
- Projeto mais organizado;
- Torna o projeto mais fácil de ser entendido por outros programadores;

Suas desvantagens, embora que condicionais, incluem:

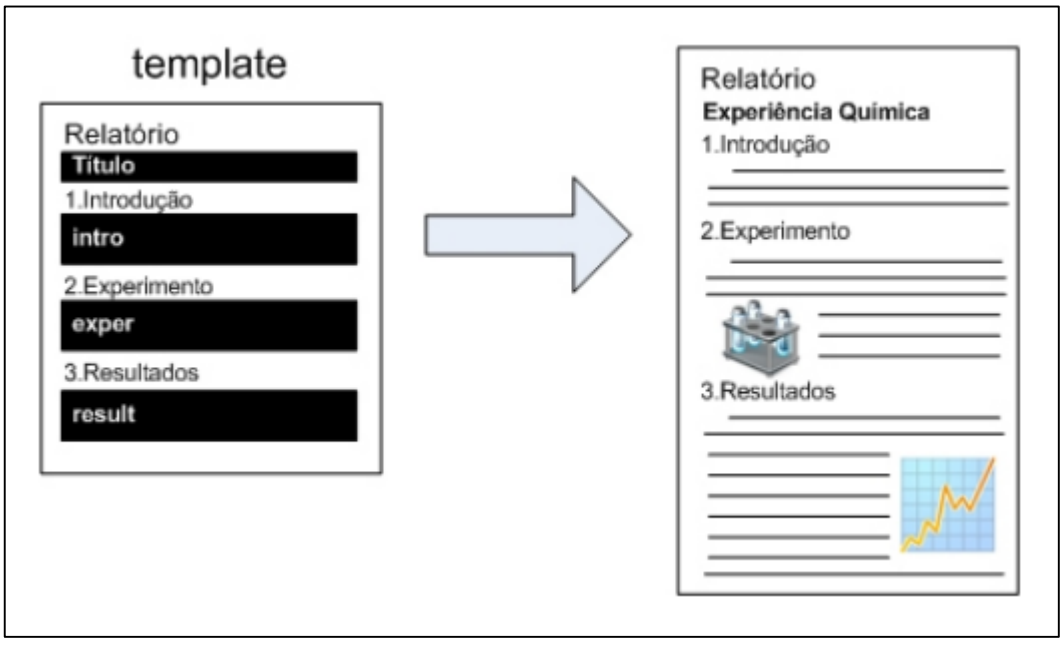
- Em sistemas de pequeno porte, o padrão MVC pode gerar uma complexidade desnecessária;
- Requer extrema disciplina do desenvolvedor no que tange à separação das camadas;
- Exige um tempo adicional na modelagem do sistema.

6.10.2 Template Method

O Padrão *Template Method* (Método de modelos) é bem utilizado quando desejamos estabelecer um algoritmo geral que cumpra uma sequência de etapas para cumprir um requisito da aplicação), porém estes passos podem variar e é necessária uma estrutura que permita que alguns deles sejam substituídos ou alterados. (GUERRA, 2014).

Um modelo de documento é o exemplo perfeito do *Template Method*. Definimos algumas partes que são fixas, não se alteram, e outras partes que serão introduzidas quando for necessário que uma função do sistema gerar o documento. Essas lacunas que precisam ser preenchidas, e que variam de documento para documento, são completadas com funções de *data binding* (ligação de dados). Estes dados podem ser passados para o *template* (modelo) através de métodos mensageiros de objetos sempre que o documento for gerado, inserindo cada dado no campo que lhe for destinado. Vemos o exemplo na figura 12. (GUERRA, 2014)

Figura 12 - Uso do padrão *Template Method* para documentos



Fonte: GUERRA, 2014, p. 30

"[...] *Template Method* é um modelo de algoritmo que possui partes fixas e algumas variáveis. As partes variáveis são lacunas que precisam ser completadas para que o algoritmo faça realmente sentido". (GUERRA, 2014)

6.10.3 Correto uso dos padrões

Podemos levantar uma questão: Quanto mais padrões eu utilizar, melhor vai ficar meu código? A resposta é: Não. Um padrão é uma solução para um problema. Aplicar um padrão onde o problema não existe irá gerar um problema. Padrões também possuem consequências que podem ser negativas e superar as vantagens em alguns casos. É bom conhecer padrões variados para utilizar o mais adequado ao projeto e evitar utilizações desnecessárias, que podem impactar negativamente o desenvolvimento e manutenção de um software. (GUERRA, 2014)

Para Guerra (2014, p. 17), “[...]. Isso seria como para uma pessoa que acabou a aprender a usar um martelo, ver todos os problemas como se fossem um prego. Por mais que um parafuso possa parecer com um prego, a solução é bem diferente”.

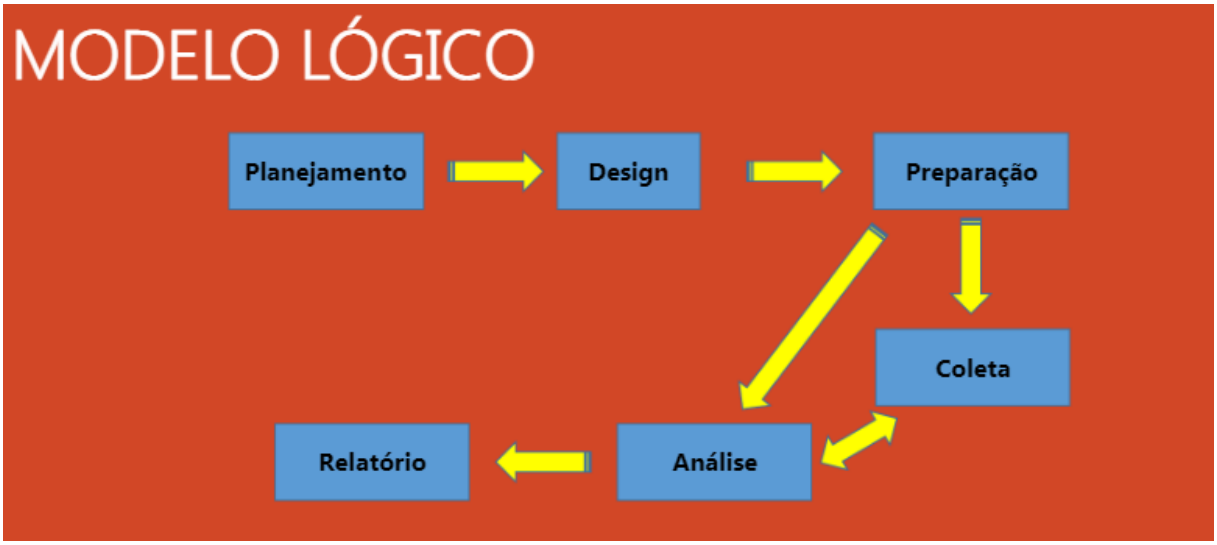
7 MATERIAL E MÉTODOS

O método piloto seguido nesta pesquisa foi o estudo de caso (YIN, 2003). Este acontece em uma empresa brasileira de médio porte, sediada em Vilhena, Rondônia. A empresa é uma corretora de imóveis, realizando serviços de locação e venda de imóveis e outros serviços relacionados.

Segundo Yin (2003), o método de estudo de caso é apropriado para responder a perguntas do tipo “como” e “por que”. Ele indica que este método contempla coleta de dados de diferentes fontes, para que seja possível fazer triangulação de dados. No presente trabalho, a coleta de dados caracteriza-se por ser quantitativa, pois foi realizado um levantamento de quantidades de locações ativas, lançamentos de receitas e despesas, e ao mesmo tempo, qualitativa, pois foi analisada documentação da empresa, recolhida informação através da observação direta e algumas entrevistas com os atores diretos, como corretores, secretárias e gerentes, para entender os processos de geração dos contratos, armazenamento de informações e política de gestão financeira.

Levantadas essas informações, foi projetado o software e então desenvolvido para entrega à empresa candidata para utilização e aplicação na rotina da empresa.

Figura 13 - Fluxo lógico de estudo de caso



Fonte: https://www.researchgate.net/publication/280938344_Estudos_de_casos_-_metodologia_baseada_em_Yin_2003_-_2011

As entrevistas tinham por finalidade verificar o conhecimento da empresa sobre gestão e o apoio da tecnologia à gestão. Foi elaborado um questionário avaliativo para entrevista aos atores diretos. Estes parâmetros, mais tarde, serviriam para guiar o levantamento de requisitos para desenvolvimento do sistema.

7.1 ESTRUTURA DA METODOLOGIA

A estrutura da metodologia adotada é composta por:

a) Referencial teórico: Levantamento das informações referentes ao projeto;
b) Elaboração e aplicação de questionário: Definir os conceitos de apoio tecnológico à gestão para a corretora;
c) Análise dos dados: realização da análise dos dados para sua posterior publicação;
d) Definir as estratégias para obter os requisitos do software e a metodologia para desenvolvimento do software.
e) Planejamento e Desenvolvimento do sistema e implantação na empresa candidata;
f) Elaboração e aplicação de questionário: Obter os resultados obtidos com a organização sistematizada de informações do negócio;
g) Resultado final e conclusões.

7.1.1 Do Referencial Teórico

O referencial teórico resultou no conhecimento necessário para a compreensão e escolha das linguagens e ferramentas necessárias para o desenvolvimento de um software capaz de auxiliar a empresa corretora candidata a ser mais eficiente em sua gestão com apoio da tecnologia. Parte dos resultados desta etapa estão refletidos nas próximas sessões deste trabalho.

7.1.2 Da Elaboração dos questionários

A elaboração do questionário inicial buscou identificar a empresa e entender os problemas referentes ao armazenamento e recuperação de informações necessárias ao negócio, entender o seu processo de trabalho e fluxo de eventos. O segundo

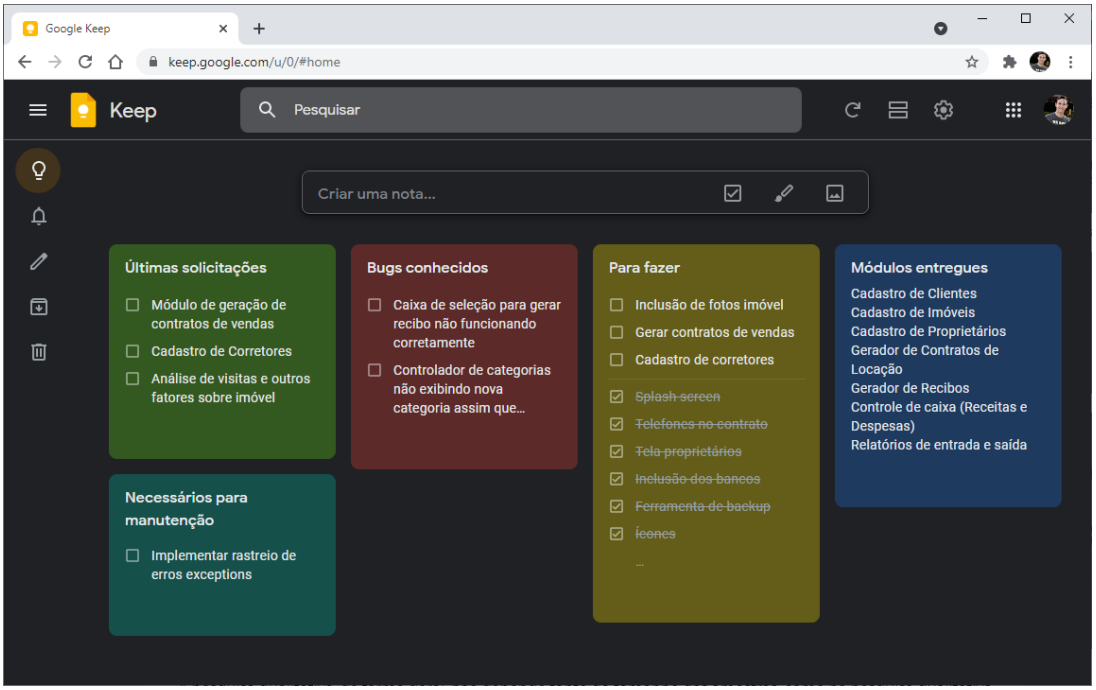
questionário obtém um *feedback* sobre a qualidade do software entregue e em que ele atuou para a melhoria da gestão.

7.1.3 Do Planejamento de Desenvolvimento

Após as entrevistas de definição e entendimento dos problemas, ocorreu o planejamento do desenvolvimento e em seguida iniciou-se o planejamento desenvolvimento do software. Foram realizadas entrevistas para levantamento dos requisitos e desenvolvidos os diagramas UML para documentação do software. As ferramentas, IDEs e Frameworks escolhidos estão escritas no Quadro 11.

Baseado no conhecimento adquirido com o referencial teórico, nos problemas enfrentados pela empresa candidata e o modelo solicitado de entrega, optou-se pelo método ágil Kanban para desenvolvimento. Durante a fase de desenvolvimento, o cliente esteve em constante contato com o desenvolvedor, ajustando seus requisitos e recebendo atualizações sempre que prontas para utilização. Como quadro Kanban, utiliza-se o Google *Keep*, ferramenta de gerenciamento de notas. O projeto será desenvolvido aplicando o padrão MVC.

Figura 14 - Interface do Google Keep



Fonte: Conta do Google do Autor

7.2 ESTUDO DE CASO

7.2.1 Delineamento da Pesquisa

O objetivo desta pesquisa era compreender como a tecnologia da informação poderia ajudar uma empresa corretora de imóveis a melhorar sua gestão e ter controle de seu negócio. A pesquisa é adequada ao estudo de caso por que é um fato atual e é investigado no seu âmbito real, observando a rotina da empresa e obtendo informações do modelo atual de organização. A pesquisa busca entender por que as decisões foram tomadas, como foram implementadas e os resultados obtidos.

7.2.2 Desenho da Pesquisa

As prerrogativas foram levantadas a partir de que existem empresas que não buscaram evoluir sua gestão e organização, ainda presas em modelos passados de organização e armazenamento de dados do negócio e da suposição de que os softwares atuais do mercado não atendem as necessidades de negócio de maneira clara e específica. Softwares de prateleira são pré-moldados e genéricos, não atendendo necessidades específicas, ou apresentando recursos de mais para o utilizador, recursos estes que serão pagos sem se dar utilidade.

Definidas estas prerrogativas, a atividade seguinte era definir um caso. Através de contatos por relações construídas pelo pesquisador, a empresa candidata demonstrou interesse e disponibilidade para realizar uma reunião informal, onde foram marcadas novas reuniões, visitas e entrevistas para compreensão do caso.

7.2.3 Coleta de Dados

Primeiro, o pesquisador buscou entender os softwares disponíveis no mercado e levantar informações sobre as operações de corretoras de imóveis, levantando o máximo de informações que auxiliassem no preparo do estudo de caso. Definiu-se os escopos de estudo como: atividades da empresa, fluxo e armazenamento de informação e análise da TI no apoio as atividades imobiliárias.

As entrevistas e visitas foram agendadas no local de atuação da empresa, sem duração definida. O roteiro das entrevistas dava liberdade para que os entrevistados pontuassem quaisquer outras observações além das perguntas predefinidas.

7.2.4 Análise e Resultados parciais

A pesquisa foi dividida em duas fases: Anterior e posterior à instalação do sistema, para comparar os processos da empresa sem o apoio da TI e os processos da empresa com o apoio da TI, para obtenção dos resultados desejado.

Ao obter os dados dos atores da empresa, estes dados foram confrontados para serem interpretados e cruzados. Algumas dúvidas restantes das visitas foram solucionadas em contatos com os entrevistados. Todas as informações extraídas foram confirmadas pelos entrevistados.

O quadro 9 demonstra os principais resultados da pesquisa que levantou informações atuais de gestão e organização, com o motivo de preparar o projeto e desenvolvimento do software na próxima etapa.

Quadro 9 - Identificação e Análise da coleta de dados

Objetivos	Compreender os processos da empresa sem o apoio da TI		
Premissa	Identificar as ferramentas utilizadas e métodos aplicados		
Elementos de Análise	Atividades na Empresa	Armazenamento e leitura de informações	Métodos de documentação
Entrevistado 1	Secretária, realiza confecção de contratos e mensalidades, lança as entradas e saídas da empresa.	Contratos físicos são armazenados em arquivos, não há armazenamento digital. Informações das entidades de negócio são mantidas no mensageiro instantâneo WhatsApp. Informações financeiras são guardadas em documentos e planilhas digitais em um computador.	Os contratos são feitos manualmente sempre que um novo negócio é fechado, usando o anterior como modelo para o próximo. Não há cálculos automáticos de valores e nem controle de disponibilidade de imóveis. Não há monitoria sobre o estado dos contratos e das mensalidades de clientes. Recibos e comprovantes são obtidos através da utilização de um site gratuito de geração de recibos.

Entrevistado 2	Contador, faz angariação dos imóveis para administração, realiza visitas com potenciais clientes, realiza vistorias em imóveis e intermedia negócios para terceiros	Todas as informações são passadas para a secretária para que ela cuide dos processos, armazenamento e manutenção desta função. Caso necessário alguma informação, esta é solicitada à secretária ou contador administrador.	Todas as informações são passadas para a secretária para que ela cuide dos processos, armazenamento e manutenção desta função. Caso necessário alguma informação, esta é solicitada à secretária ou contador administrador.
Conclusão	As informações de negócio da empresa são totalmente descentralizadas, por vezes confusas e não confiáveis. Não há armazenamento de informações de forma segura, São mantidas onde ocorreu o contato e lá são visualizadas. Informações financeiras são armazenadas de maneira diferente e ineficiente, mal formatadas, sem cálculo preciso, dependendo do fator humano para ser correto. Com a rotatividade de funcionários, é possível que haja erros nos cálculos ou diferenças nos métodos de trabalho, acarretando num processo heterogêneo e sem padrões determinados.		

Fonte: Próprio autor.

Pontos em comum da análise são a falha no processo de armazenamento das informações e sua descentralização. A documentação gerada pelos processos da empresa também não é totalmente confiável devido a não padronização dos processos. Estes fatores já resultaram na perda de negócios importantes para a empresa, deixando de ganhar dinheiro pela desorganização de suas informações.

Entendidos os processos sem o apoio da TI, começou os estudos para levantar os requisitos desejados para o sistema com base em visitas e entrevistas e observações diretas dos processos. As próximas sessões deste projeto abordarão a fase de planejamento de desenvolvimento do software para dar início a segunda fase da pesquisa.

7.3 LEVANTAMENTO DE REQUISITOS

7.3.1 Requisitos Funcionais

7.3.1.1 RF001: Cadastro de Clientes

Registro de clientes e suas informações, contendo nome, CPF/CNPJ, RG, data de nascimento, profissão, estado civil, telefones e e-mail, e por opcionais observações, dados bancários e cadastro de endereço.

7.3.1.2 RF002: Cadastro de Imóveis

Registro de imóveis e suas informações, contendo o nome do proprietário, o endereço do imóvel, o laudo de vistoria do imóvel, data de angariação, data do laudo, descrição, e corretor administrador.

7.3.1.3 RF003: Cadastro de Proprietários

Registro de proprietários dos imóveis e suas informações, contendo nome, CPF, RG, data de nascimento, profissão, estado civil, telefones e e-mail, e por opcional, observações, dados bancários e cadastro de endereço.

7.3.1.4 RF004: Cadastro de Corretores

Registro dos corretores atuando na empresa, contendo nome, CPF, RG, data de nascimento, nº de registro CRECI, estado civil, telefones e e-mail, e por opcionais observações, dados bancários e cadastro de endereço.

7.3.1.5 RF005: Gerenciamento de contratos de locação

O sistema deve ser capaz de criar e manter registro dos contratos firmados pela empresa. Estes contratos devem conter os dados do locador, do administrador do imóvel, informar os valores e prazos, e cláusulas de negócio predefinidas.

7.3.1.6 RF006: Gerenciamento de receitas e despesas

Registro das despesas com seus respectivos valores, vencimentos, categoria, descrição e agendamentos, tais como suas receitas com os mesmos parâmetros.

7.3.1.7 RF007: Gerenciamento de mensalidades

O sistema deve exibir as mensalidades das locações de cada cliente, representadas por seu respectivo contrato de locação, e calcular os juros pertinentes caso necessário. Estas mensalidades serão geradas automaticamente no ato de geração do contrato, e lançando automaticamente suas receitas, onde a primeira mensalidade gera uma comissão de 50% para a corretora e as demais mensalidades geram uma comissão de 10% para a corretora.

7.3.1.8 RF008: Geração de recibos

O sistema deve ser capaz de gerar recibos para receitas e mensalidades, como desejado pelo operador do sistema.

7.3.1.9 RF009: Notificações

O sistema deve contar uma janela que mostre as mensalidades vencidas e que estão a vencer nos próximos dias, e mostrar quais os contratos a vencer nos próximos dois meses.

7.3.2 Requisitos não funcionais

7.3.2.1 NF001: Validação de dados de entrada

Evitar erros de digitação previsíveis e validar dados antes de persistir.

7.3.2.2 NF002: Formatos de arquivos

Os relatórios, contratos e dados de entidades a serem exportados devem ser escritos em formato pdf (*Portable document format* – Formato de documento portátil).

7.3.2.3 NF003: Armazenamento

O sistema de arquivos deve conter pastas para armazenar os recibos, contratos, imóveis e informações adicionais para o funcionamento do sistema.

7.3.2.4 NF004: Minimalismo

O sistema deve adotar um design minimalista, com a menor quantidade de botões necessárias para o funcionamento do programa.

7.3.2.5 NF005: Plataforma

O sistema deve ser desenvolvido em ambiente Desktop, em estrutura monolítica.

7.3.2.6 NF006: Implementação

Os módulos solicitados devem ser incluídos na versão do cliente sempre que a atualização estiver pronta.

7.3.2.7 NF007: Padrões

O sistema deve adotar os padrões de cores e logotipos da franquia.

7.3.2.8 NF008: Segurança e integridade dos dados

O sistema deve possuir um sistema de backup para garantir a permanência de seus dados.

7.3.2.9 NF009: Testemunhas para contrato

O sistema deve aceitar variação de testemunhas para lavrar os contratos.

7.3.3 Pré condições

7.3.3.1 PRC001: Inicialização do sistema

O banco de dados deve estar em execução antes do sistema iniciar para que ele demonstre sua funcionalidade.

7.3.3.2 PRC002: Efetivação de contrato

Para que o sistema crie o contrato de locação ou venda, é necessário que já estejam cadastrados o cliente, o proprietário e o imóvel pertencentes ao acordo.

7.3.4 Fluxo Principal de Eventos

O usuário inicia o sistema. Ele verifica o cadastro de clientes para saber se é necessário um novo cadastro do cliente que deseja alugar um imóvel. Em seguida, ele verifica se o imóvel está cadastrado. Caso não esteja, prossegue para o cadastro do imóvel e seu proprietário. Satisfeitas estas condições descritas no PRC002, inicia a confecção do contrato, selecionando o cliente, o imóvel, a data de início do contrato, a quantidade de meses de locação, o valor de cada mensalidade, e se é necessário ajuste na data da primeira parcela. Quando o contrato é finalizado, ele fica disponível para visualização e impressão.

7.3.5 Pós condições

7.3.5.1 PSC001: Cadastro de receitas

Após geração do contrato, as mensalidades devem ser cadastradas no perfil do cliente em questão, e as receitas provenientes desta locação devem ser lançadas no fluxo de entrada de caixa da empresa com seus respectivos vencimentos.

7.3.6 Exceções ou fluxo secundário de eventos

7.3.6.1 Fluxo secundário 1

O usuário pode acessar o sistema para verificar o status das mensalidades de um determinado cliente através do painel de clientes ou do visualizador de notificações.

7.3.6.2 Fluxo secundário 2

O usuário pode iniciar o sistema para extrair relatórios de despesas e receitas de cada mês ou por um período desejado de tempo, através do painel de receitas ou despesas.

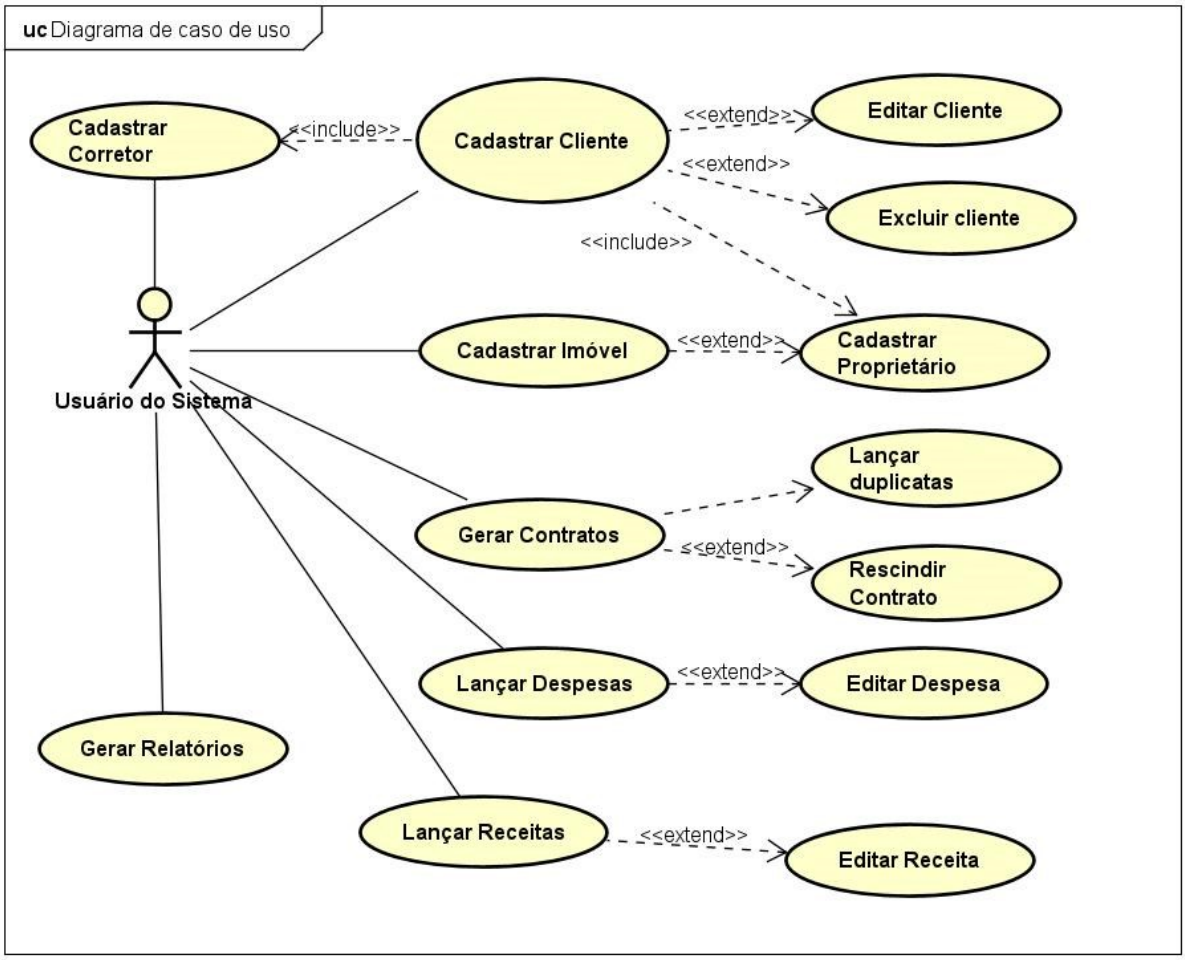
7.4 MODELAGEM DO SOFTWARE

Levantados os requisitos, inicia-se a etapa de modelagem do software. Após algumas reuniões e observações diretas das operações do software, criaram-se as histórias para o quadro Kanban, descrevendo o que o cliente deseja para cada requisito.

Devido ao tamanho dos diagramas, a nível de impressão, pode haver dificuldade para leitura e visualização. Por este motivo, estão disponíveis todos os diagramas deste projeto no seguinte repositório: <https://drive.google.com/drive/folders/1W_wYieL3u8EcyqxXHVViGHLvDaHhxet3?usp=sharing>.

7.4.1 Diagrama de Caso de Uso

Figura 15 - Diagrama de caso de uso



Fonte: Próprio autor

7.4.1.1 Descrição do caso de uso

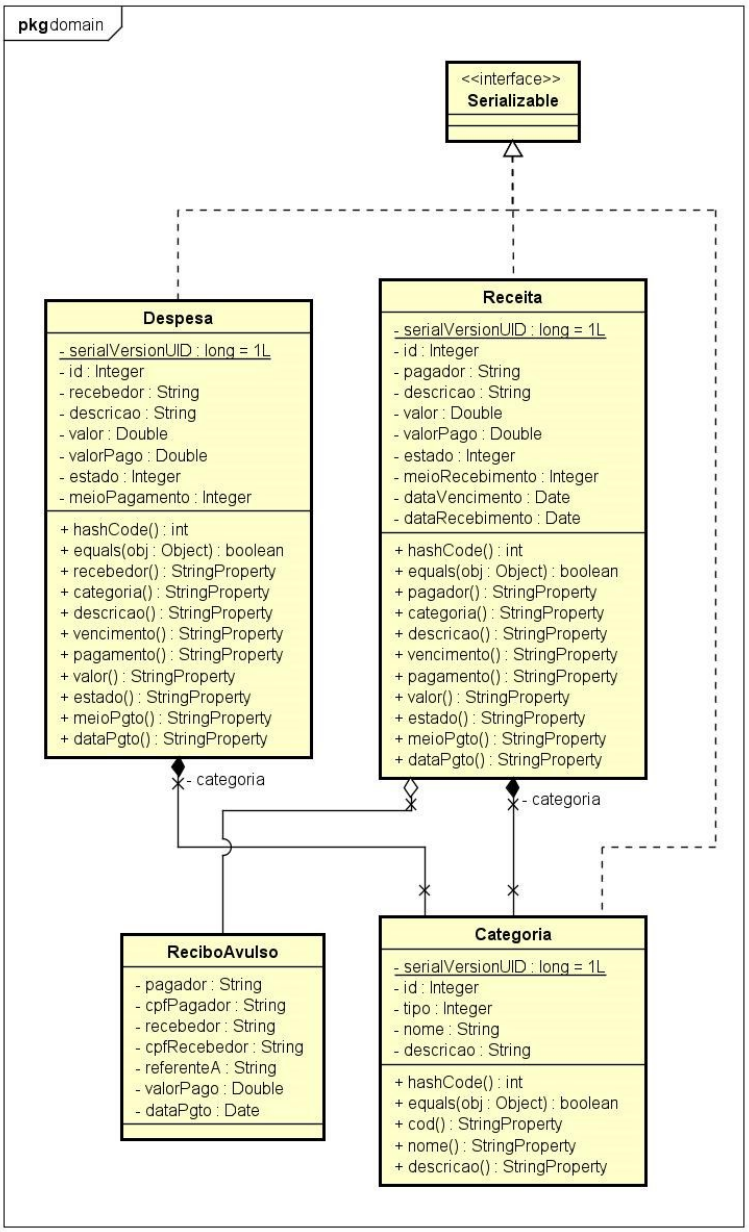
Demonstra as utilidades solicitadas para o operador do sistema.

7.4.1.2 Atores

Usuário do Sistema: Qualquer usuário que tenha permissão de uso do sistema.

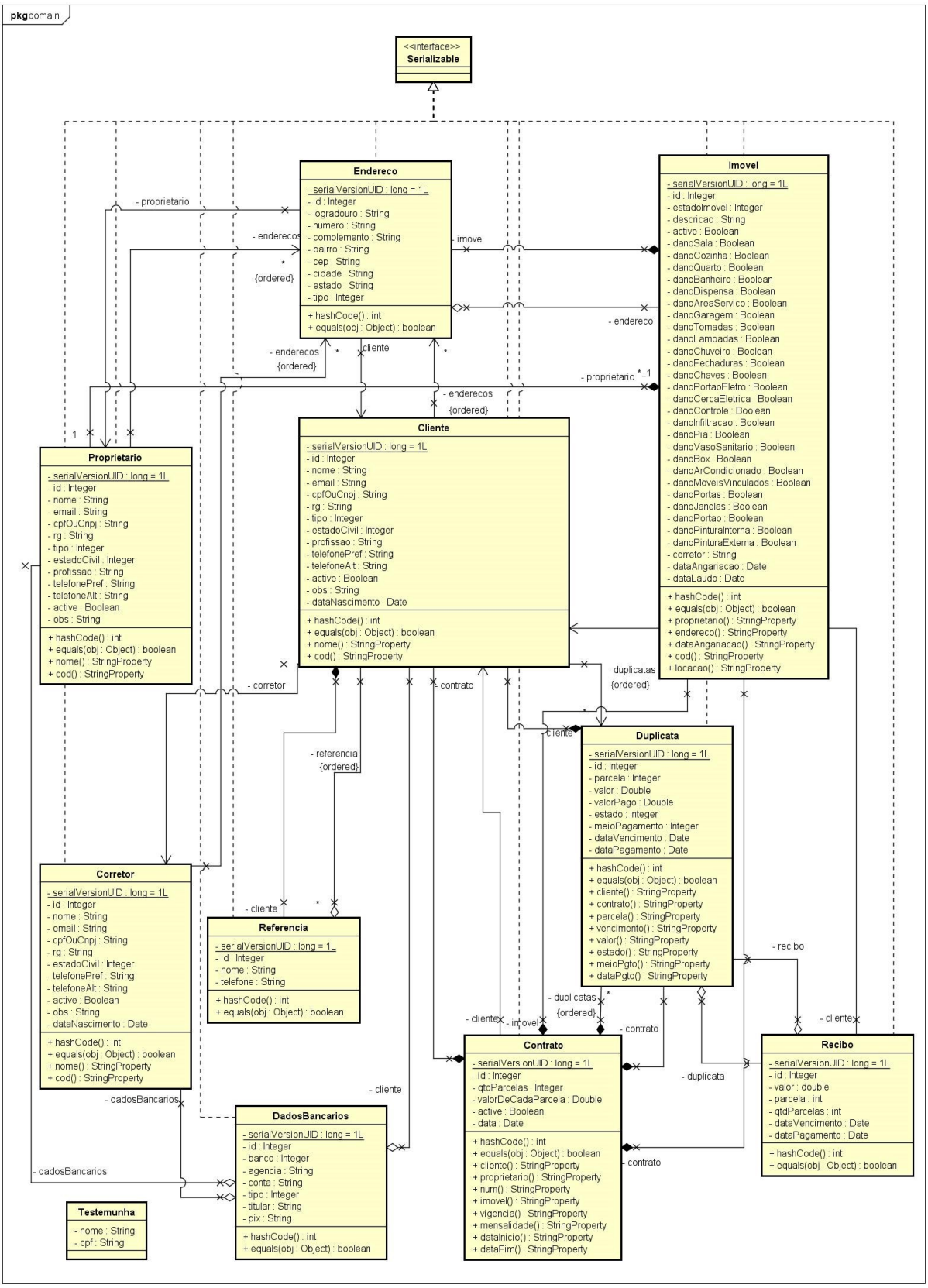
7.4.2 Diagrama de Classes

Figura 16 - Diagrama de Classes (parte 1)



Fonte: Próprio autor

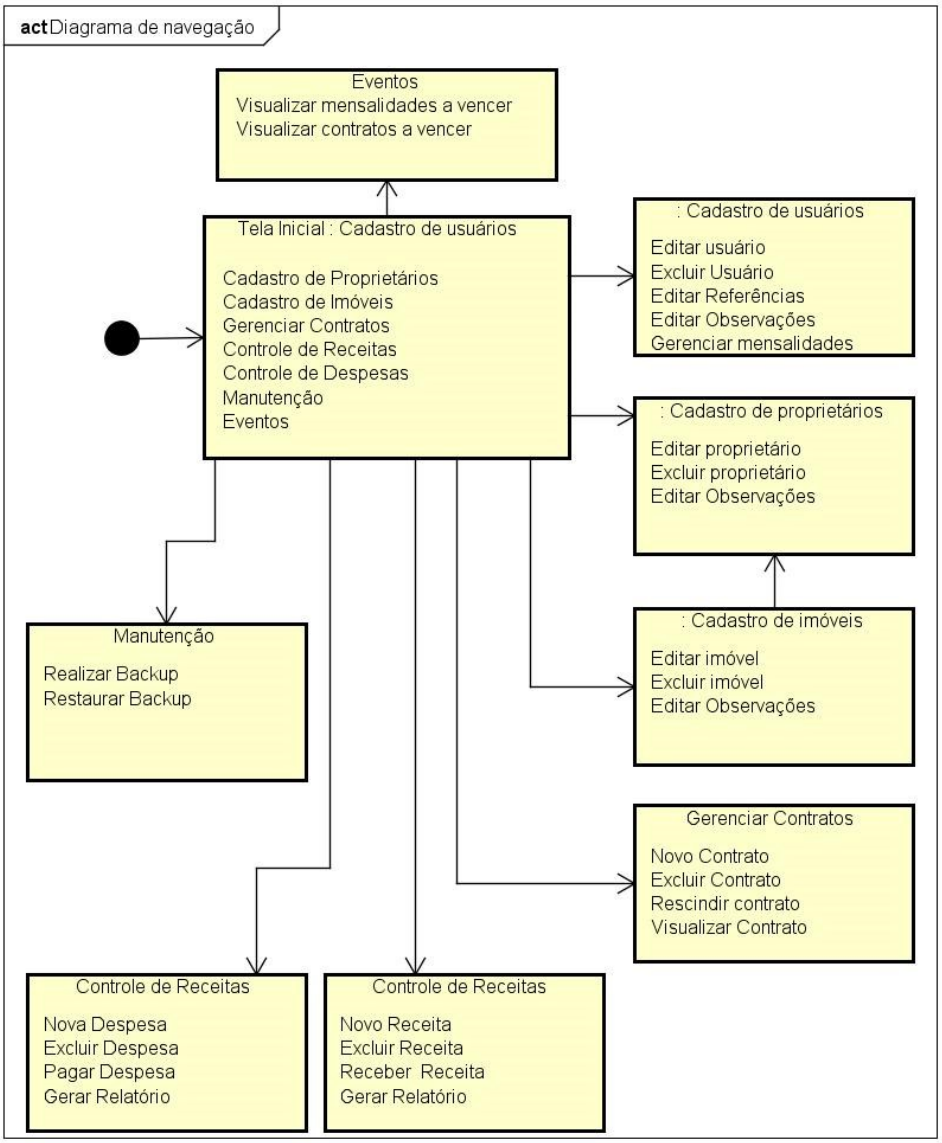
Figura 17 - Diagrama de classes (parte 2)



Fonte: Próprio autor

7.4.3 Diagrama de navegação

Figura 18 - Diagrama de navegação

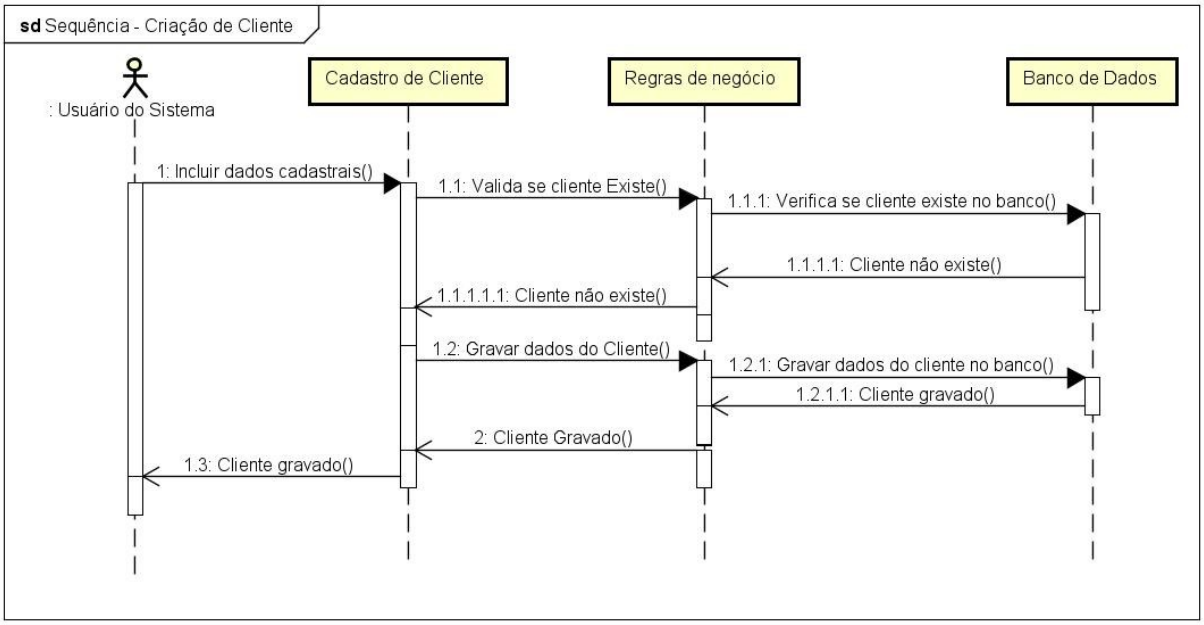


Fonte: Próprio autor

7.4.4 Diagramas de Sequência

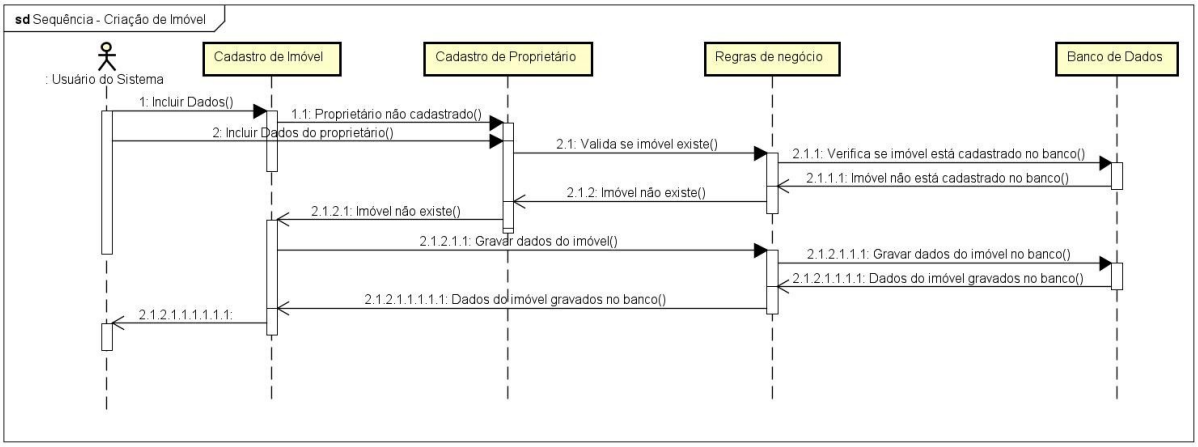
Os Diagramas de sequência para Proprietário e Corretor respeitam a mesma sequência da sequência de criação de clientes.

Figura 19 - Diagrama de sequência - criação de cliente



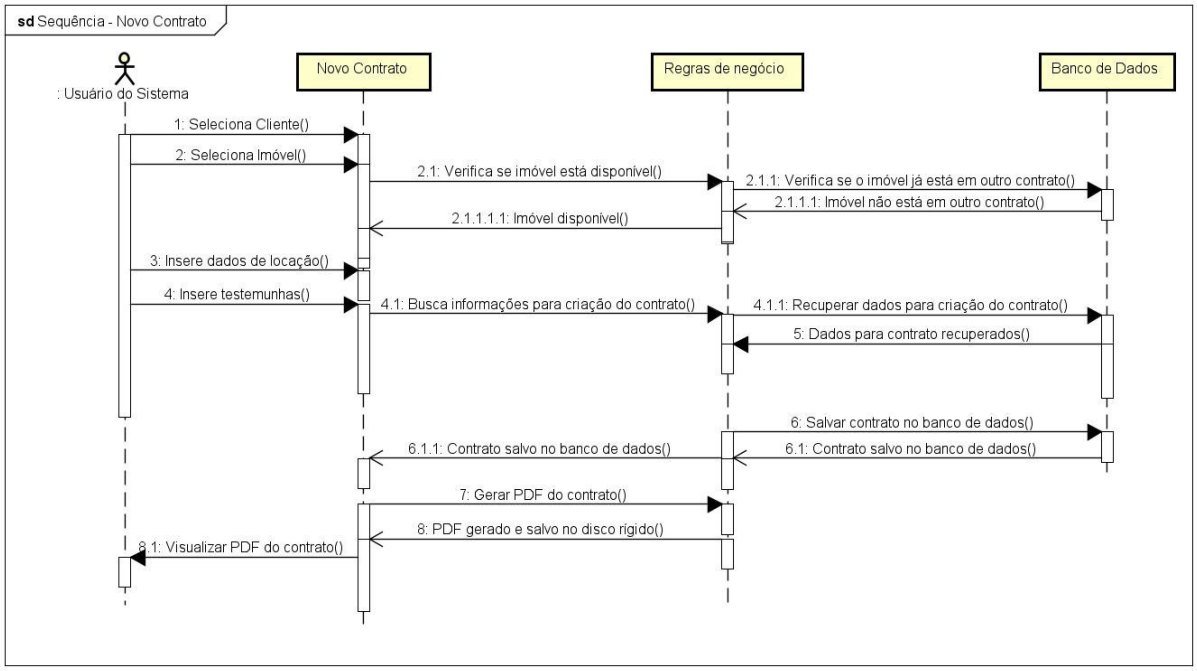
Fonte: Próprio autor

Figura 20 - Diagrama de sequência - criação de imóvel



Fonte: Próprio autor

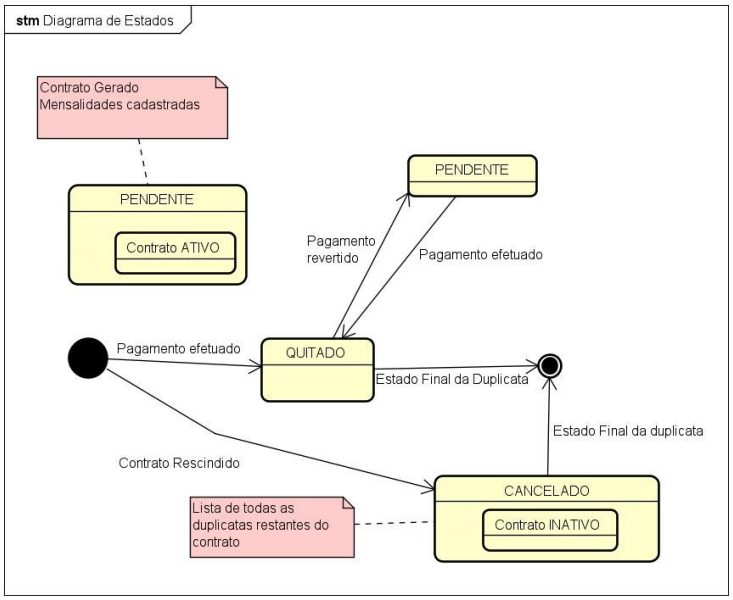
Figura 21 - Diagrama de sequência - novo contrato



Fonte: Próprio autor

7.4.5 Diagrama de estados

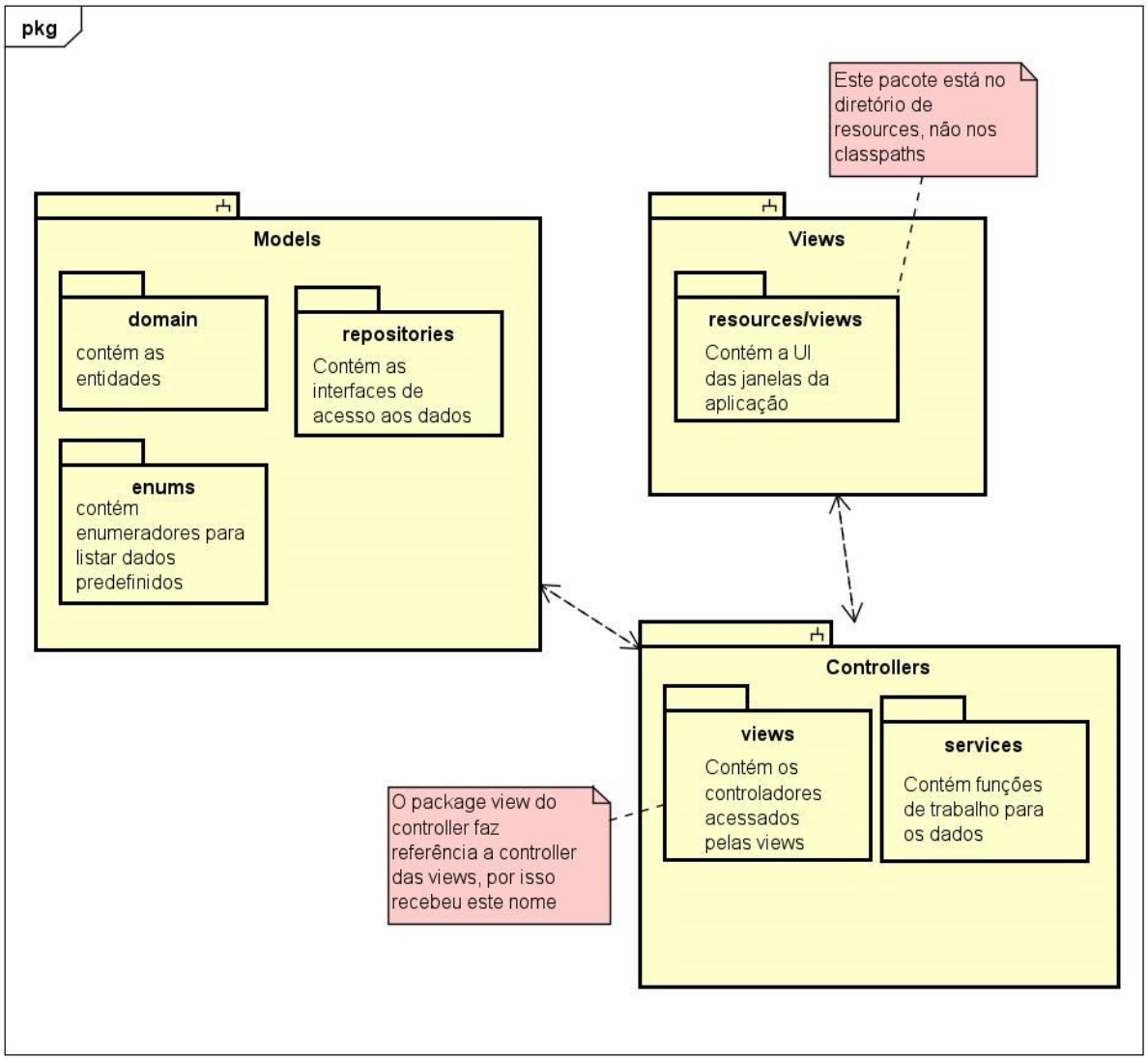
Figura 22 - Diagrama de Estados



Fonte: Próprio autor

7.4.6 Diagrama de pacotes

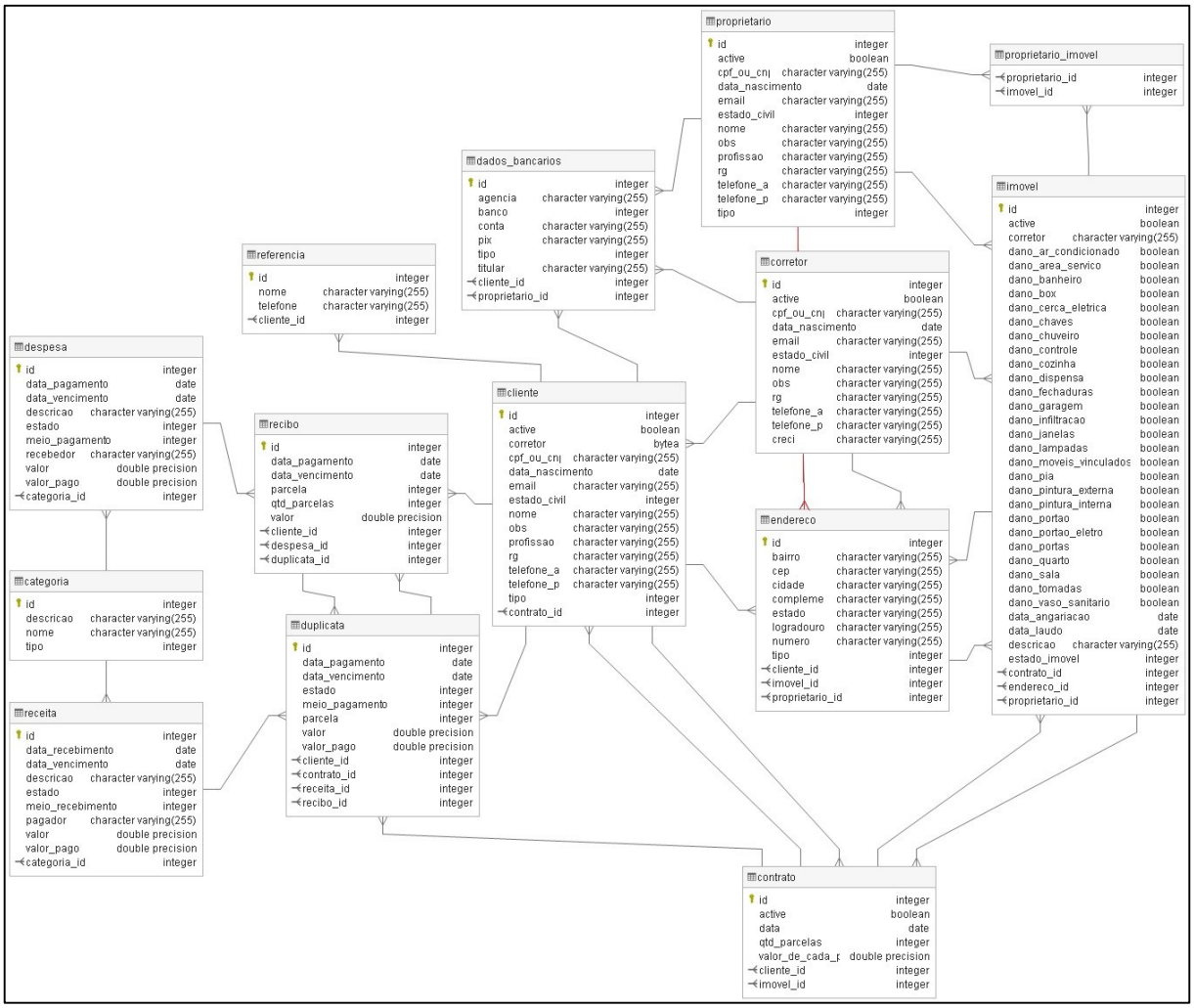
Figura 23 - Diagrama de pacotes seguindo o padrão MVC



Fonte: Próprio autor

7.4.7 Diagrama entidade relacionamento (DER)

Figura 24 - Diagrama Entidade Relacionamento



Fonte: Próprio autor

7.4.8 Dicionário de Dados

Quadro 10 - Dicionário de Dados

Entidade	Atributo	Tipo de Dado	Descrição
Categoria	id	integer	PK - id da categoria no banco de dados
Categoria	descricao	character varying(255)	informa uma descrição que caracteriza a categoria
Categoria	nome	character varying(255)	determina o nome da categoria
Categoria	tipo	integer	determina o tipo da categoria, se receita ou despesa, com base em um enumerador
Cliente	id	integer	PK - id do cliente

Cliente	active	boolean	determina se o usuário está ativo
Cliente	cpf_ou_cnpj	character varying(255)	determina o número de CPF ou CNPJ do cliente
Cliente	data_nascimento	date	determina a data de nascimento do cliente
Cliente	email	character varying(255)	determina o endereço de e-mail do cliente
Cliente	estado_civil	integer	determina o estado civil atual do cliente, com base em um enumerador
Cliente	nome	character varying(255)	determina o nome completo do cliente
Cliente	obs	character varying(255)	observações pertinentes a um cliente
Cliente	profissao	character varying(255)	determina a profissão do cliente
Cliente	rg	character varying(255)	determina o número do RG do cliente
Cliente	telefone_alt	character varying(255)	determina o número de telefone alternativo do cliente
Cliente	telefone_pref	character varying(255)	determina o número de telefone preferencial do cliente
Cliente	tipo	integer	determina o tipo de cliente, se pessoa física ou jurídica, com base em um enumerador
Cliente	contrato_id	integer	faz referência ao id do contrato atual ativo do cliente
Cliente	corretor_id	integer	faz referência ao corretor vinculado ao cliente
Contrato	id	integer	PK - id do contrato no banco de dados e código referência para número do contrato
Contrato	active	boolean	determina se o contrato está ativo
Contrato	data	date	informa a data de início do contrato
Contrato	qtd_parcelas	integer	informa a quantidade de mensalidades para o tempo de vigência do contrato
Contrato	valor_de_cada_parcela	double precision	informa o valor das mensalidades do contrato
Contrato	cliente_id	integer	referência para o cliente locador do contrato
Contrato	imovel_id	integer	referência para o imóvel locado no contrato
Corretor	id	integer	PK - id do corretor

Corretor	active	boolean	determina se o correetor está ativo
Corretor	cpf_ou_cnpj	character varying(255)	determina o número de CPF ou CNPJ do corretor
Corretor	data_nascimento	date	determina a data de nascimento do corretor
Corretor	email	character varying(255)	determina o endereço de e-mail do corretor
Corretor	estado_civil	integer	determina o estado civil atual do corretor, com base em um enumerador
Corretor	nome	character varying(255)	determina o nome completo do corretor
Corretor	obs	character varying(255)	informa alguma observação pertinente ao corretor
Corretor	rg	character varying(255)	determina o número do RG do corretor
Corretor	telefone_alt	character varying(255)	determina o número de telefone alternativo do corretor
Corretor	telefone_pref	character varying(255)	determina o número de telefone preferencial do corretor
Corretor	creci	character varying(255)	informa o número do registro CRECI do corretor
Dados_bancarios	id	integer	PK - id dos Dados Bancários no banco de dados
Dados_bancarios	agencia	character varying(255)	informa o número da agência bancária
Dados_bancarios	banco	integer	informa o nome do banco que pertence a conta
Dados_bancarios	conta	character varying(255)	informa o número da conta bancária
Dados_bancarios	pix	character varying(255)	informa a chave de transferência pix da conta bancária
Dados_bancarios	tipo	integer	determina o tipo de conta, com base em um enumerador
Dados_bancarios	titular	character varying(255)	informa o nome do titular da conta bancária
Dados_bancarios	cliente_id	integer	faz referência ao cliente pertencente da conta bancária
Dados_bancarios	proprietario_id	integer	faz referência ao proprietário pertencente da conta bancária
Dados_bancarios	corretor_id	integer	faz referência ao corretor pertencente da conta bancária

Despesa	id	integer	PK - id da despesa no banco de dados
Despesa	data_pagamento	date	informa a data de pagamento da despesa
Despesa	data_vencimento	date	informa a data de vencimento da despesa
Despesa	descricao	character varying(255)	informa uma descrição da despesa
Despesa	estado	integer	informa o estado do pagamento, com base em um enumerador
Despesa	meio_pagamento	integer	informa a forma de pagamento, com base em um enumerador
Despesa	recebedor	character varying(255)	informa o nome de quem recebeu o pagamento da despesa
Despesa	valor	double precision	informa o valor inicial da despesa
Despesa	valor_pago	double precision	informa o valor pago da despesa
Despesa	categoria_id	integer	faz referência ao tipo de categoria da despesa
Duplicata	id	integer	PK - id da duplicata no banco de dados
Duplicata	data_pagamento	date	informa a data de pagamento da duplicata
Duplicata	data_vencimento	date	informa a data de vencimento da duplicata
Duplicata	estado	integer	informa o estado do pagamento, com base em um enumerador
Duplicata	meio_pagamento	integer	informa a forma de pagamento, com base em um enumerador
Duplicata	parcela	integer	informa qual parcela do contrato que esta duplicata representa
Duplicata	valor	double precision	informa o valor inicial da duplicata
Duplicata	valor_pago	double precision	informa o valor pago da duplicata
Duplicata	cliente_id	integer	referência para o cliente locador do contrato o qual as duplicatas pertencem
Duplicata	contrato_id	integer	faz referência ao id do contrato o qual as duplicatas pertencem
Duplicata	receita_id	integer	faz referência à receita que foi gerada a partir desta duplicata

Duplicata	recibo_id	integer	faz referência ao recibo gerado quando esta duplicata for paga
Endereco	id	integer	PK - id do endereço no banco de dados
Endereco	bairro	character varying(255)	informa o bairro
Endereco	cep	character varying(255)	informa o CEP
Endereco	cidade	character varying(255)	informa a cidade
Endereco	complemento	character varying(255)	informa o complemento
Endereco	estado	character varying(255)	informa o Estado UF
Endereco	logradouro	character varying(255)	informa o nome do logradouro
Endereco	numero	character varying(255)	informa o número da instalação no logradouro
Endereco	tipo	integer	determina o tipo de endereço, com base em um enumerador
Endereco	cliente_id	integer	faz referência ao cliente instalado neste endereço
Endereco	imovel_id	integer	referência para o imóvel instalado neste endereço
Endereco	proprietario_id	integer	faz referência ao proprietário de imóvel instalado neste endereço
Endereco	corretor_id	integer	faz referência ao corretor instalado neste endereço
Imovel	id	integer	PK - id do imóvel no banco de dados e código referência para cadastro do imóvel
Imovel	active	boolean	determina se o imóvel está ativo
Imovel	corretor	character varying(255)	informa o corretor administrador do imóvel
Imovel	dano_ar_condicionado	boolean	informa se há dano no ar condicionado do imóvel
Imovel	dano_area_servico	boolean	informa se há dano na área de serviço do imóvel
Imovel	dano_banheiro	boolean	informa se há dano no banheiro do imóvel
Imovel	dano_box	boolean	informa se há dano no box do banheiro do imóvel
Imovel	dano_cerca_eletrica	boolean	informa se há dano na cerca elétrica do imóvel
Imovel	dano_chaves	boolean	informa se há dano nas chaves do imóvel
Imovel	dano_chuveiro	boolean	informa se há dano no chuveiro do imóvel
Imovel	dano_controle	boolean	informa se há dano nos controles do imóvel

Imovel	dano_cozinha	boolean	informa se há dano na cozinha do imóvel
Imovel	dano_dispensa	boolean	informa se há dano na dispensa do imóvel
Imovel	dano_fechaduras	boolean	informa se há dano nas fechaduras do imóvel
Imovel	dano_garagem	boolean	informa se há dano na garagem do imóvel
Imovel	dano_infiltracao	boolean	informa se há dano de infiltrações do imóvel
Imovel	dano_janelas	boolean	informa se há dano nas janelas do imóvel
Imovel	dano_lampadas	boolean	informa se há dano nas lâmpadas do imóvel
Imovel	dano_moveis_vinculados	boolean	informa se há dano nos móveis vinculados do imóvel
Imovel	dano_pia	boolean	informa se há dano na pia do imóvel
Imovel	dano_pintura_externa	boolean	informa se há dano na pintura externa do imóvel
Imovel	dano_pintura_interna	boolean	informa se há dano na pintura interna do imóvel
Imovel	dano_portao	boolean	informa se há dano no portão do imóvel
Imovel	dano_portao_eletro	boolean	informa se há dano no portão elétrico do imóvel
Imovel	dano_portas	boolean	informa se há dano nas portas do imóvel
Imovel	dano_quarto	boolean	informa se há dano no quarto do imóvel
Imovel	dano_sala	boolean	informa se há dano na sala do imóvel
Imovel	dano_tomadas	boolean	informa se há dano nas tomadas elétricas do imóvel
Imovel	dano_vaso_sanitario	boolean	informa se há dano no vaso sanitário do imóvel
Imovel	data_angariacao	date	informa a data de angariação do imóvel pela corretora
Imovel	data_laudo	date	informa a data do último laudo realizado que constata os danos no imóvel
Imovel	descricao	character varying(255)	informa uma descrição do imóvel
Imovel	estado_imovel	integer	informa o estado do imóvel, baseado em um enumerador
Imovel	contrato_id	integer	faz referência ao id do contrato atual ativo do imóvel locado

Imovel	endereco_id	integer	faz referência ao endereço que o imóvel está instalado
Imovel	proprietario_id	integer	faz referência ao proprietário deste imóvel
Imovel	corretor_id	integer	faz referência ao corretor administrador deste imóvel
Proprietario	id	integer	PK - id do proprietário de imóvel no banco de dados e código referência para cadastro do proprietário
Proprietario	active	boolean	determina se o proprietário está ativo
Proprietario	cpf_ou_cnpj	character varying(255)	determina o número de CPF ou CNPJ do proprietário
Proprietario	data_nascimento	date	determina a data de nascimento do proprietário
Proprietario	email	character varying(255)	determina o endereço de e-mail do proprietário
Proprietario	estado_civil	integer	determina o estado civil atual do proprietário, com base em um enumerador
Proprietario	nome	character varying(255)	determina o nome completo do proprietário
Proprietario	obs	character varying(255)	informa alguma observação pertinente ao proprietário
Proprietario	profissao	character varying(255)	determina a profissão do proprietário
Proprietario	rg	character varying(255)	determina o número do RG do proprietário
Proprietario	telefone_alt	character varying(255)	determina o número de telefone alternativo do proprietário
Proprietario	telefone_pref	character varying(255)	determina o número de telefone preferencial do proprietário
Proprietario	tipo	integer	determina o tipo de proprietário, se pessoa física ou jurídica, com base em um enumerador
Proprietario_imovel	proprietario_id	integer	faz referência ao proprietário de imóvel
Proprietario_imovel	imovel_id	integer	faz referência ao imóvel do proprietário
Receita	id	integer	PK - id da receita no banco de dados
Receita	data_recebimento	date	informa a data de recebimento da receita
Receita	data_vencimento	date	informa a data de vencimento da receita
Receita	descricao	character varying(255)	informa uma descrição da receita

Receita	estado	integer	informa o estado do pagamento, com base em um enumerador
Receita	meio_recebimento	integer	informa a forma de recebimento, com base em um enumerador
Receita	pagador	character varying(255)	informa o nome do pagador da receita
Receita	valor	double precision	informa o valor inicial da receita
Receita	valor_pago	double precision	informa o valor pago da receita
Receita	categoria_id	integer	faz referência ao tipo de categoria da receita
Recibo	id	integer	PK - id do recibo no banco de dados
Recibo	data_pagamento	date	informa a data de pagamento da duplicata que este recibo representa
Recibo	data_vencimento	date	informa a data de vencimento da duplicata que este recibo representa
Recibo	parcela	integer	informa qual parcela do contrato da duplicata que este recibo representa
Recibo	qtd_parcelas	integer	informa a quantidade de mensalidades para o tempo de vigência do contrato
Recibo	valor	double precision	informa o valor pago por este recibo
Recibo	cliente_id	integer	referencia o cliente que este recibo pertence
Recibo	despesa_id	integer	referencia a despesa que este recibo pertence
Recibo	duplicata_id	integer	referencia a duplicata que este recibo pertence
Referencia	id	integer	PK - id da referência pessoal no banco de dados
Referencia	nome	character varying(255)	determina o nome completo da referência pessoal
Referencia	telefone	character varying(255)	informa o número de telefone da referência pessoal
Referencia	cliente_id	integer	referencia o cliente que esta referência pessoal pertence

Fonte: Próprio autor

7.5 ARQUITETURA DO SISTEMA

O Sistema será desenvolvido em duas camadas – cliente/servidor, capaz de aceitar pelo menos 4 conexões simultâneas. A configuração mínima ideal para o servidor é:

Processador 2.0GHz – 6 MB cache.

4 GB Memória RAM.

Espaço livre em disco de armazenamento de 5 GB.

Quadro 11 - Relação de mecanismos de desenvolvimento

Mecanismo de Análise	Mecanismo de Design	Mecanismo de implementação
Modelagem	Ferramentas de modelagem	Astah Community, Dataedo
Persistência	Banco de dados relacional	PostgreSQL
Front-End	GUI – (*Graphic User Interface* - Interface Gráfica de Usuário)	JavaFX
Back-End	Frameworks e Processamento do sistema	Spring Boot, Java 11, Hibernate, JPA, Spring Data
Build	Programação da IDE e Código Fonte	Java 11, JavaFX, SQL
Deploy	Desenvolvimento do software	Spring Tools Suite 4, PgAdmin 4, Scene Builder
Versionamento	Controle de versão	Git e GitHub
Metodologia de desenvolvimento	Método ágil Kanban	Google Keep

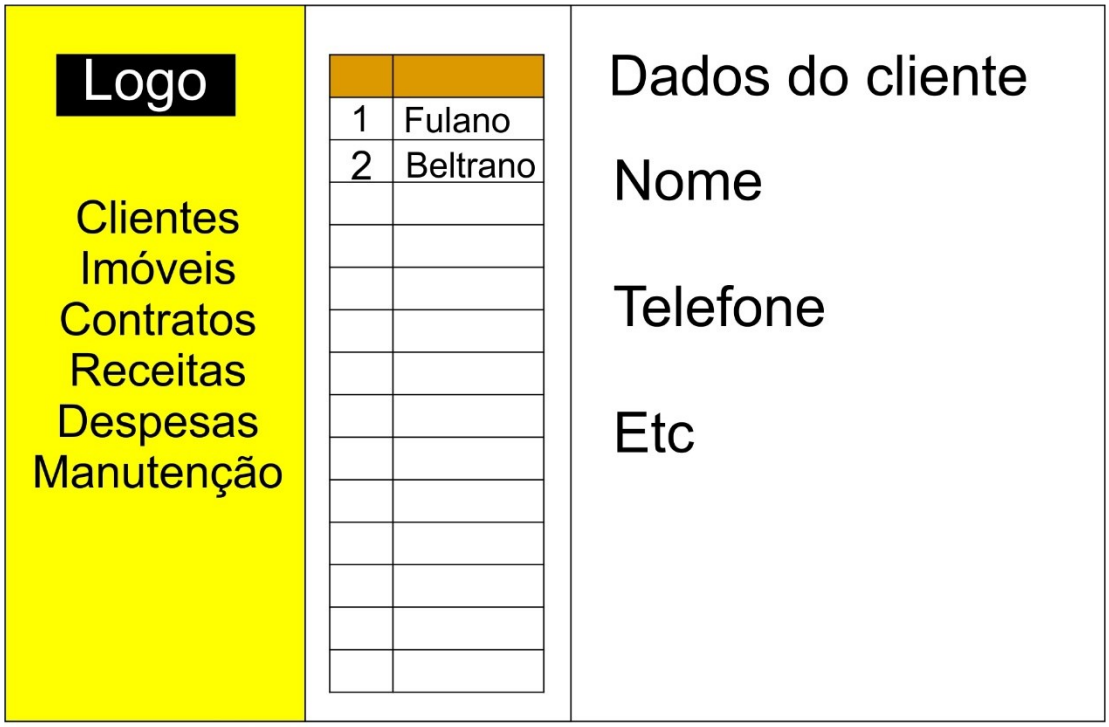
Fonte: Próprio autor

7.6 PROTÓTIPO

7.6.1 Telas

Figura 25 - Modelos de prototipagem das telas.

(a) Dados de Clientes, Proprietários e corretores; (b) Visualização de receitas e despesas



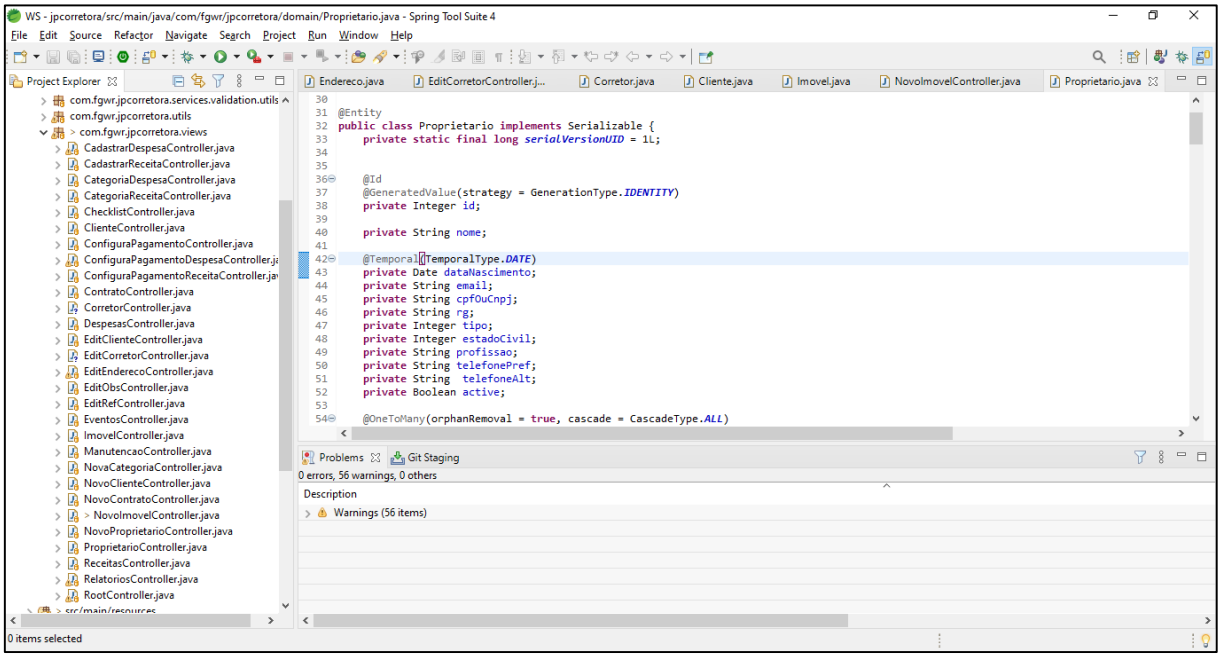
Fonte: (a) Próprio autor; (b) Próprio autor

7.7 DESENVOLVIMENTO DO SOFTWARE

Com base na modelagem descrita neste projeto, inicia-se então o desenvolvimento do software na forma de código e os seus testes de execução, para posterior distribuição e implementação no ambiente de produção do cliente.

Para preservação da imagem da empresa candidata, algumas partes das imagens podem conter borrões com a finalidade de omitir informações da mesma.

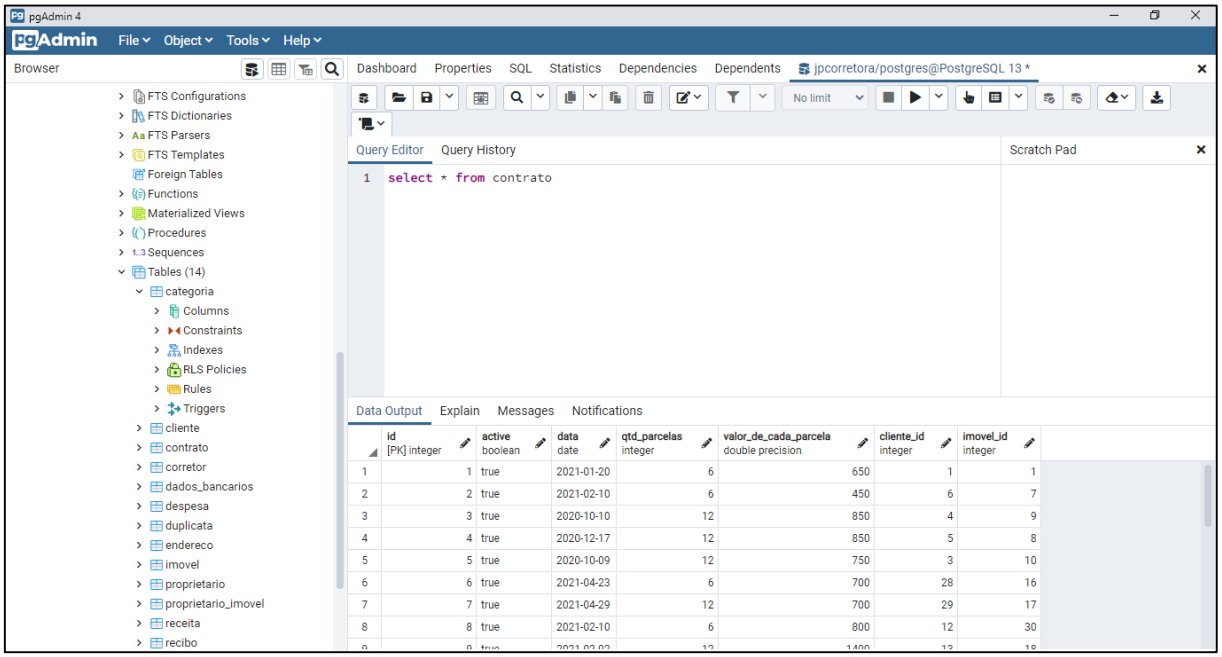
Figura 26 - Spring Tools Suite 4 em execução com exemplo de uma classe



Fonte: Próprio autor

Observando a figura 26, podemos notar alguns recursos da linguagem e do framework, como as anotações. Na figura 26 estamos observando a classe Proprietario, anotada com @Entity, que a define como uma Entidade que será convertida em tabela no banco de dados. A anotação @Id informa que o atributo id será uma PK, e a anotação @GeneratedValue(strategy = GenerationType.IDENTITY) informa que ela será auto incrementada com a estratégia de geração de valor do tipo identidade sempre que persistido um objeto do tipo Proprietario no banco de dados. Mais abaixo observamos uma anotação de relação @OneToMany, definindo que esta classe tem uma relação de Um para Muitos imóveis, que será refletido nas relações criadas no banco de dados pelo framework.

Figura 27 - pgAdmin 4 em execução com exemplo de consulta no SGBD



Fonte: Próprio autor

Logo na primeira execução do programa, o framework configura o código para gerar automaticamente as tabelas e relações no banco de dados. Na figura 27, temos uma visualização do conteúdo da tabela contrato, com os atributos da classe contrato e suas relações com cliente e imóvel, ambas criadas automaticamente pelo framework. Cada inicialização do programa verifica se o banco de dados está de acordo com o código, e caso seja lançada uma atualização que modifique as relações e entidades, o framework irá atualizar automaticamente as tabelas e relações do banco de dados para atingir a conformidade com o modelo do software.

Cada atualização entregue, funcionalidade nova ou correção de bugs concluída e testada era salva no repositório local do Git e enviado para o repositório remoto do GitHub. A utilização deste sistema de controle de versão permitiu várias comparações de código e recuperação de trechos necessários graças ao mecanismo de visualização de códigos do GitHub. Sem dúvidas, uma ferramenta indispensável.

8 RESULTADOS E DISCUSSÃO

8.1 METODOLOGIA E PADRÕES ESCOLHIDOS

A metodologia ágil escolhida se mostrou eficiente para o desenvolvimento do software aplicado ao estudo de caso. Os pontos observados durante o desenvolvimento do software que confirmaram a escolha da metodologia como adequada foram:

- Capacidade de definir as histórias do usuário e gerar os requisitos a partir deles;
- Criar os kanbans no Google *Keep*, controlando através desses kanbans, ou notas, o que era necessário fazer, o que já havia sido feito, os bugs encontrados e os testes realizados;
- Permitiu que cada contato entre a empresa e o desenvolvedor gerassem kanbans ou atualizações de kanbans no quadro, alterando de forma rápida a manutenção e atualização do software;
- Permitiu que cada módulo novo, função nova ou atualização concluída fosse implementada em caráter JIT para o cliente utilizar o mais rápido possível de suas solicitações;
- Permitiu que o fluxo de trabalho fosse bem definido e priorizado;
- Simplificou o processo de desenvolvimento e o fluxo de trabalho.

Seguindo o padrão MVC, obtemos um código organizado e de fácil manutenção se comparado a uma estrutura mais acoplada. Foi enxugado e perceptível a redução de *boiilerplate*, ainda que seja um projeto de pequeno porte, não consideramos que a complexidade do MVC foi um obstáculo ou um atraso no projeto. Pelo contrário, possibilitou que ele escale e que haja atualizações mais rápidas.

8.2 SOFTWARE DESENVOLVIDO

O software obtido conta com os módulos solicitados no levantamento de requisitos. O software foi personalizado através do esquema de cores da marca do

negócio, exibição da logo da empresa e ícones desenhados de acordo com a logo da empresa. Abaixo, temos a lista dos módulos e suas funções:

- **Cadastro de Clientes**
  - Dados Pessoais
  - Dados Bancários
  - Gerenciamento de mensalidades
  - Referências pessoais e observações
  - Inserção, exclusão e edição dos dados de um cliente
- **Cadastro de Imóveis**
  - Dados de localização
  - Descrição do imóvel
  - Dados do proprietário
  - Laudo de vistoria e observações
  - Inserção, exclusão e edição dos dados de um imóvel
- **Cadastro de Proprietários**
  - Dados Pessoais
  - Dados Bancários
  - Referências pessoais e observações
  - Gerenciamento de imóveis
  - Inserção, exclusão e edição dos dados de um proprietário
- **Visualizador de Eventos**
  - Mensalidades vencidas
  - Mensalidades a vencer nos próximos dias
  - Contratos a expirar nos próximos 60 dias
- **Gerenciador de Contratos**
  - Listagem dos contratos ativos
  - Visualização de contratos
  - Exclusão de contratos
  - Criação do PDF do contrato
  - Rescisão de contratos
  - Geração de novos contratos

- **Gerenciamento de Despesas**
    - Visualização de despesas por mês competente
    - Cadastro de despesas
    - Pagamento de despesas
    - Exclusão de despesas
    - Relatório de despesas
- **Gerenciamento de Receitas**
    - Visualização de receitas por mês competente
    - Cadastro de receitas
    - Pagamento de receitas
    - Exclusão de receitas
    - Relatório de receitas
- **Manutenção**
    - Ferramenta de geração de backup
    - Ferramenta de restauração de backup

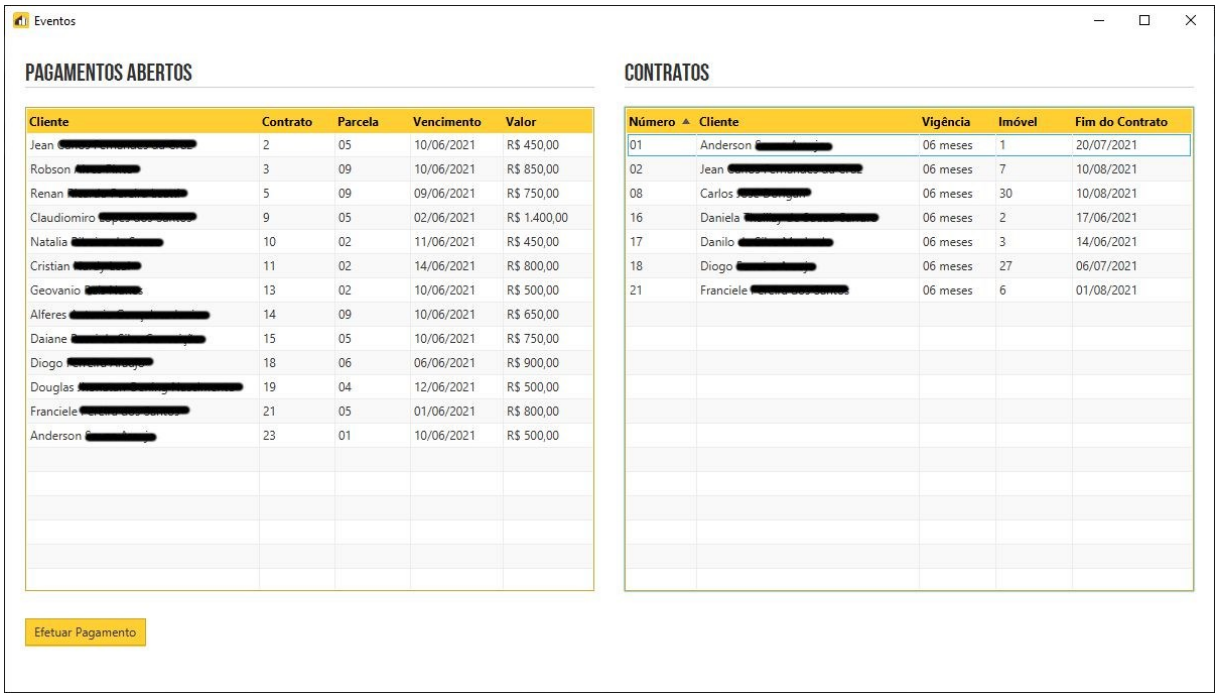
O sistema atende a todas as lógicas de negócios solicitadas pela empresa candidata. Através do levantamento de requisitos, projeção e desenvolvimento do sistema, percebe-se que o negócio flui baseado nos contratos estabelecidos entre os clientes e os proprietários, intermediado pelo corretor administrador.

8.3 INTERFACE DO USUÁRIO

A empresa possui um padrão de cores envolvendo as cores amarelo, preto e branco. O design das interfaces levou em consideração essas cores. Foi feito um menu lateral esquerdo simples para acesso de cada área do sistema. A aplicação de estilos e personalização das janelas foi feita com CSS e com configurações predefinidas disponíveis no Scene Builder, manipulando os arquivos FXML de cada tela.

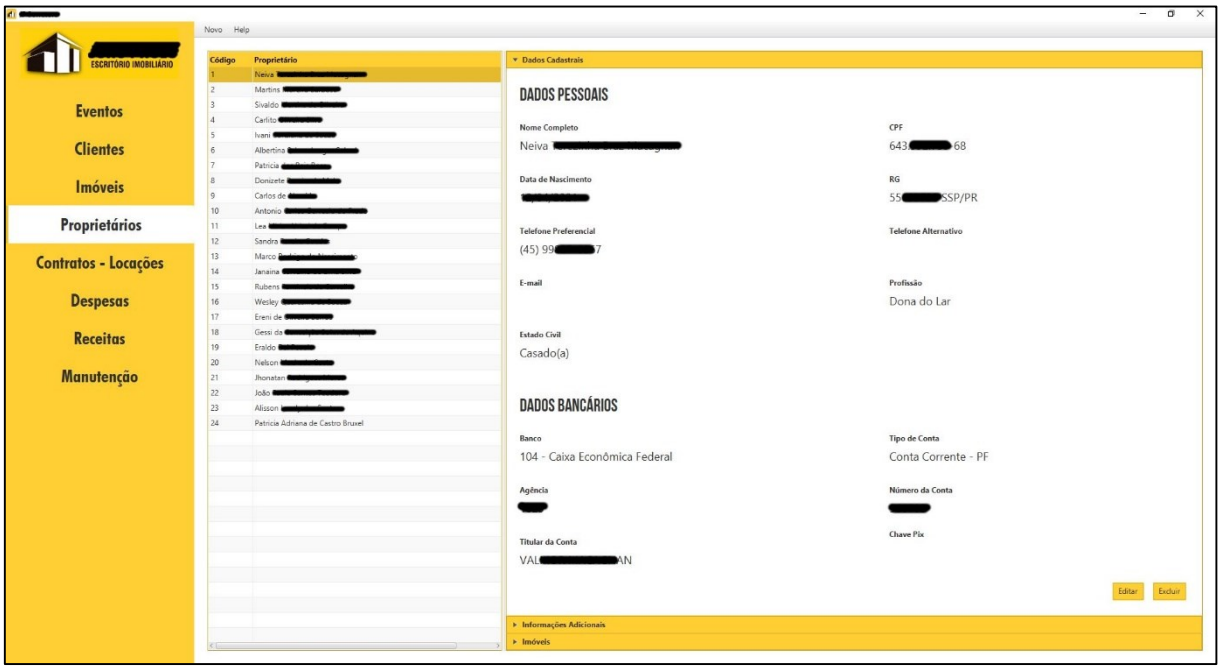
A seguir, veremos imagens das principais telas e funções da empresa em funcionamento. Para preservação da imagem da empresa candidata e das informações de seus clientes, algumas partes das imagens podem conter borrões para alcançar este objetivo.

Figura 28 - Interface da tela de Eventos



Fonte: Próprio autor

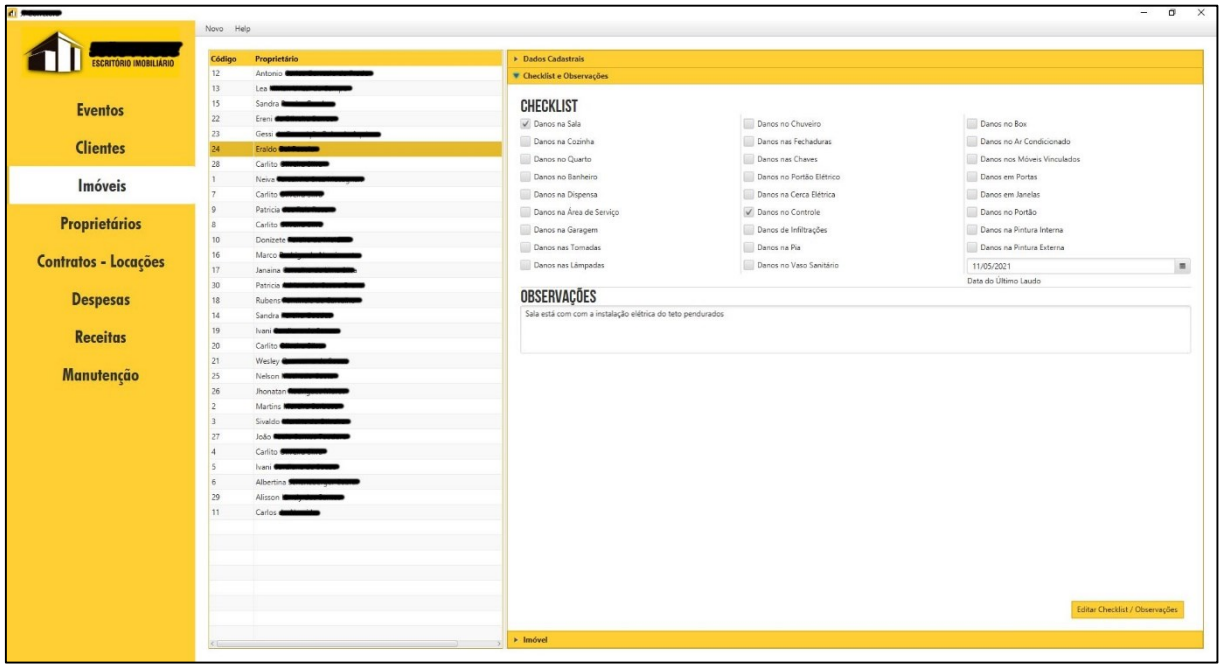
Figura 29 - Tela de cadastro de proprietários



Fonte: Próprio autor

Nota: A tela do cadastro de clientes, imóveis e corretores segue o mesmo latout

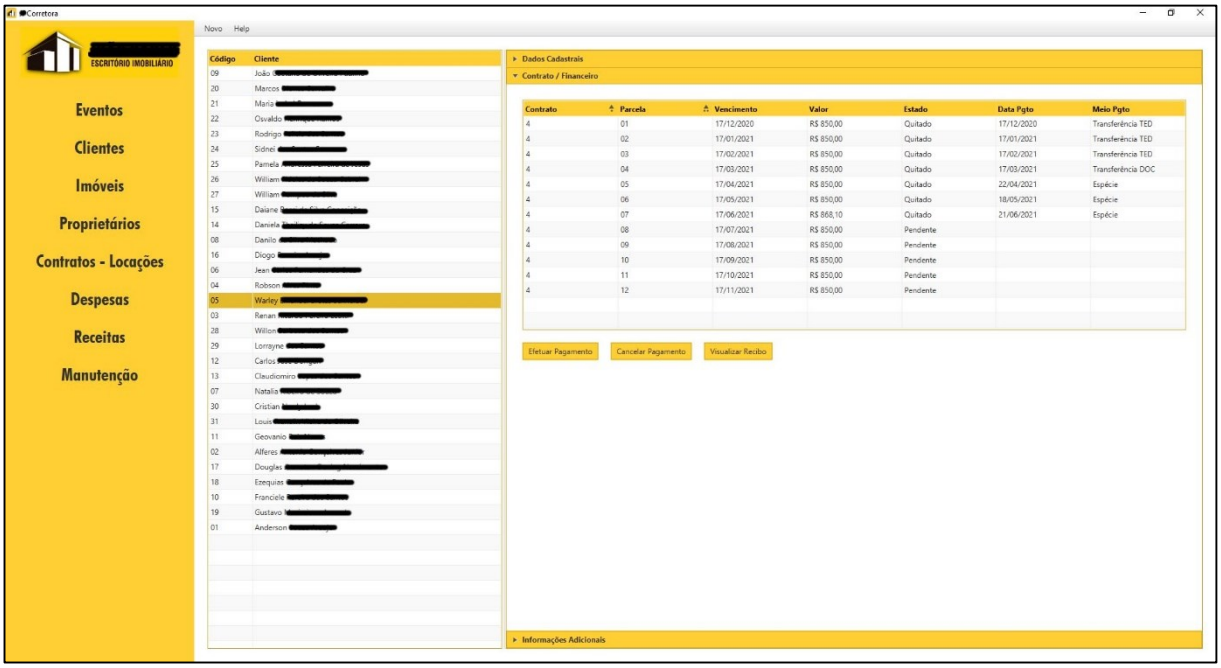
Figura 30 - Tela de cadastro de Imóveis



Fonte: Próprio autor

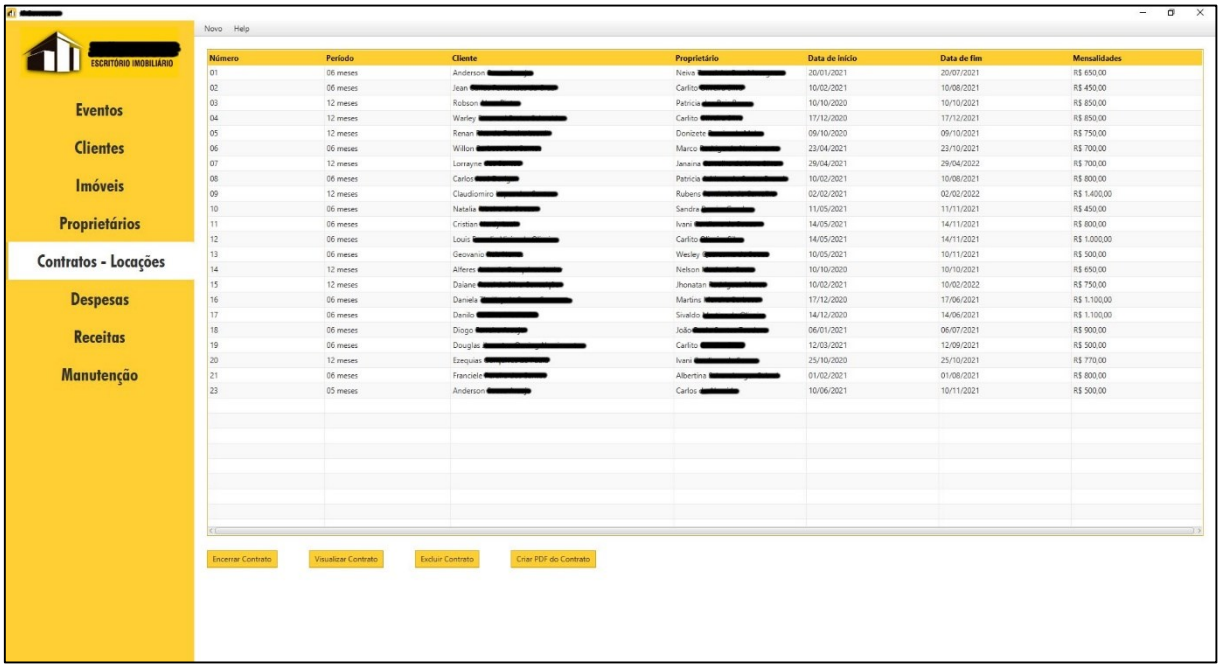
Nota: Os imóveis são listados utilizando o nome do seu proprietário.

Figura 31 - Tela de cadastro de Clientes



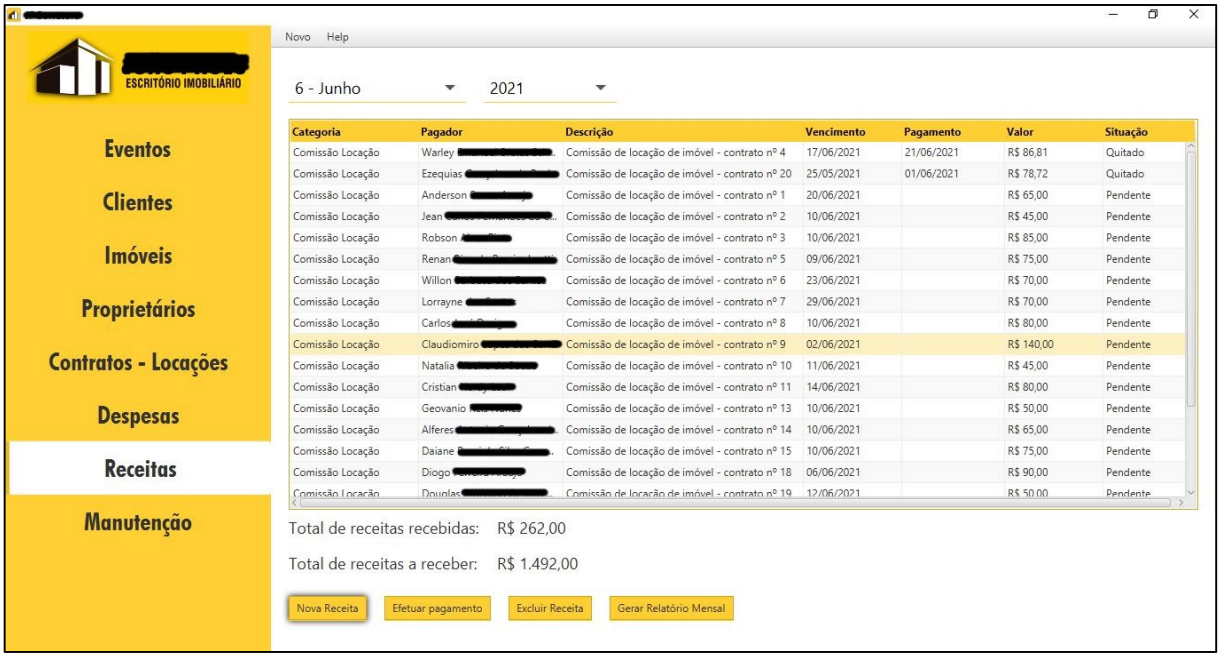
Fonte: Próprio autor

Figura 32 - Tela de gerenciamento de contratos



Fonte: Próprio autor

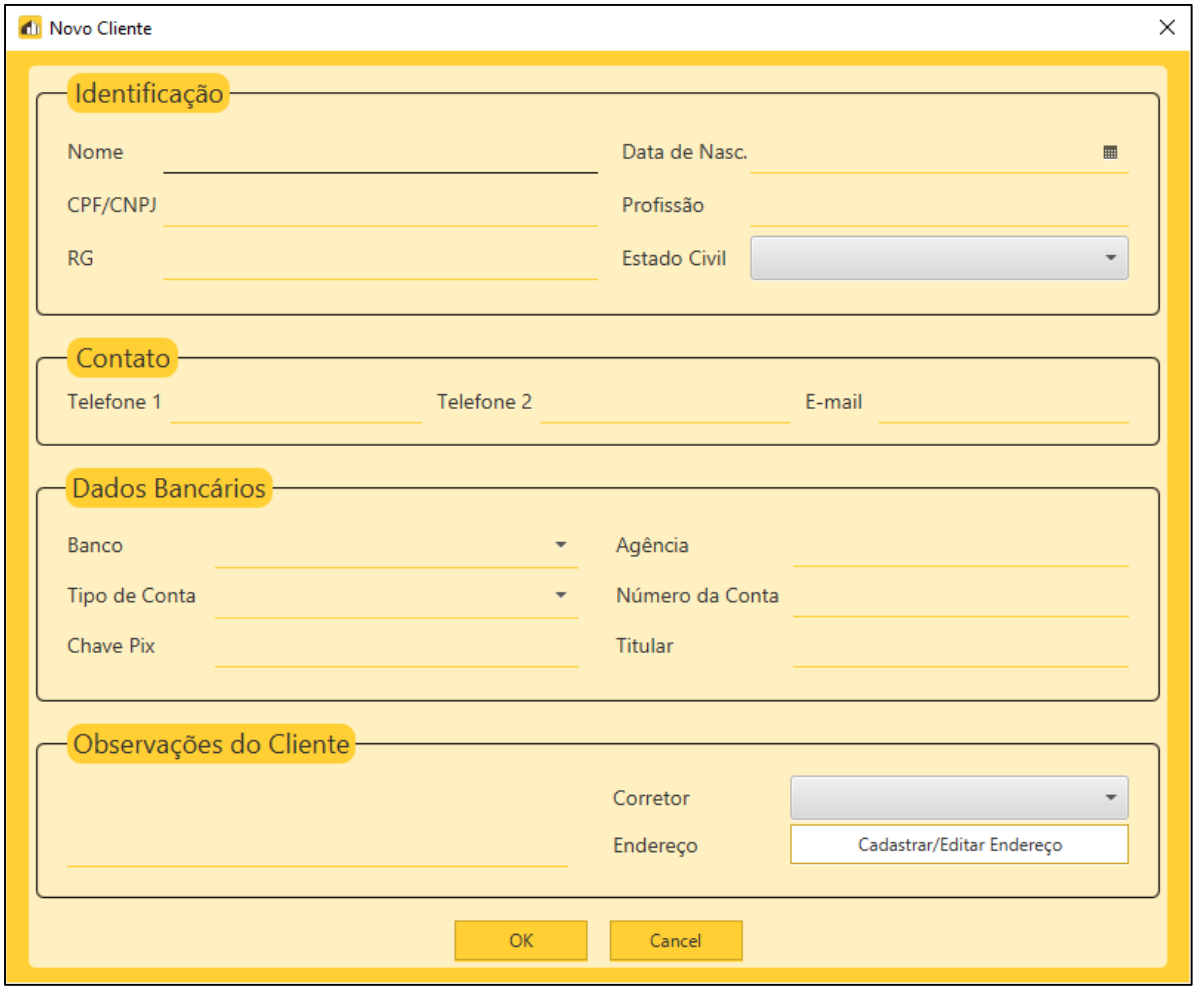
Figura 33 - Tela de gerenciamento de receitas



Fonte: Próprio autor

Nota: A tela de gerenciamento de despesas segue o mesmo layout e ferramentas

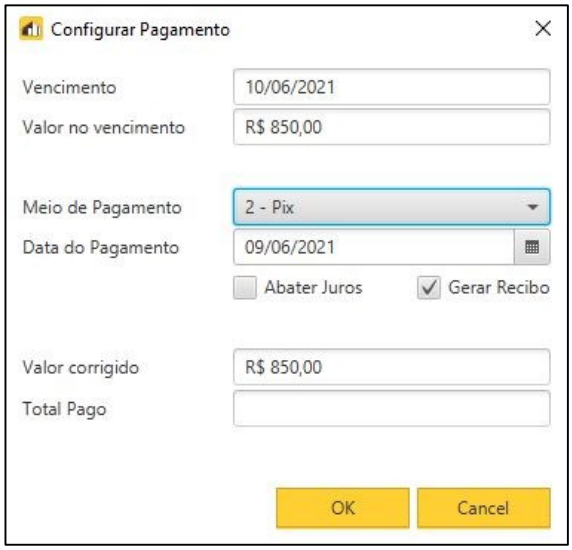
Figura 34 - Tela de cadastro de novo cliente



Fonte: Próprio autor

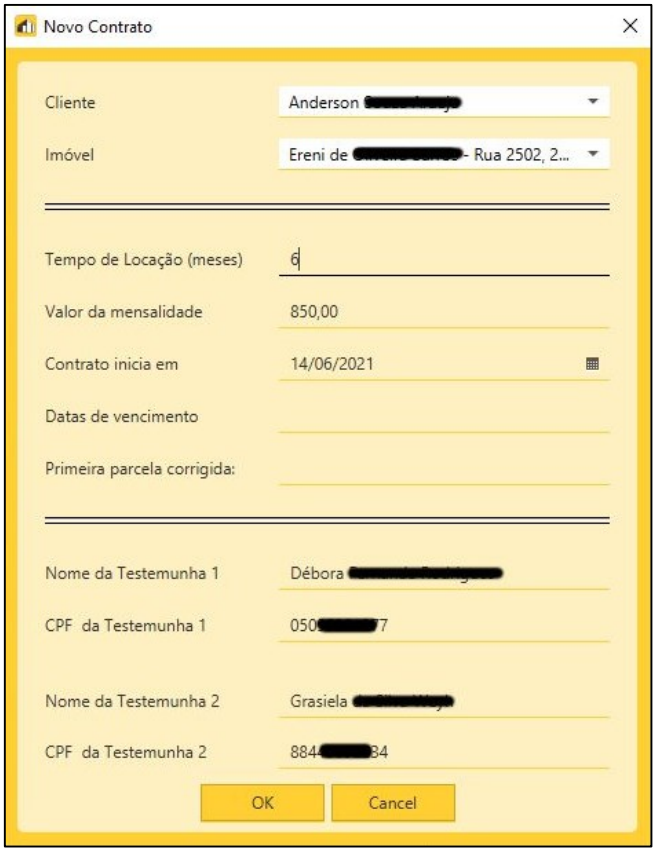
Nota: A tela de cadastro de novo proprietário e novo corretor seguem o mesmo layout

Figura 35 - Tela para baixa de pagamentos



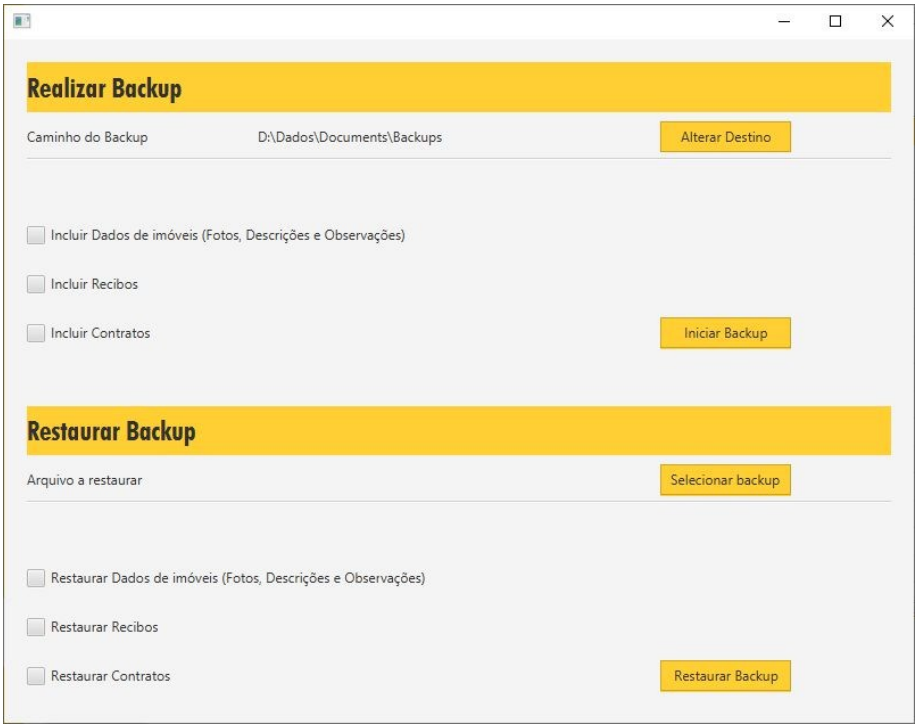
Fonte: Próprio autor

Figura 36 - Tela de geração de novo contrato



Fonte: Próprio autor

Figura 37 - Tela para manipular backups



Fonte: Próprio autor

8.4 DOCUMENTOS GERADOS

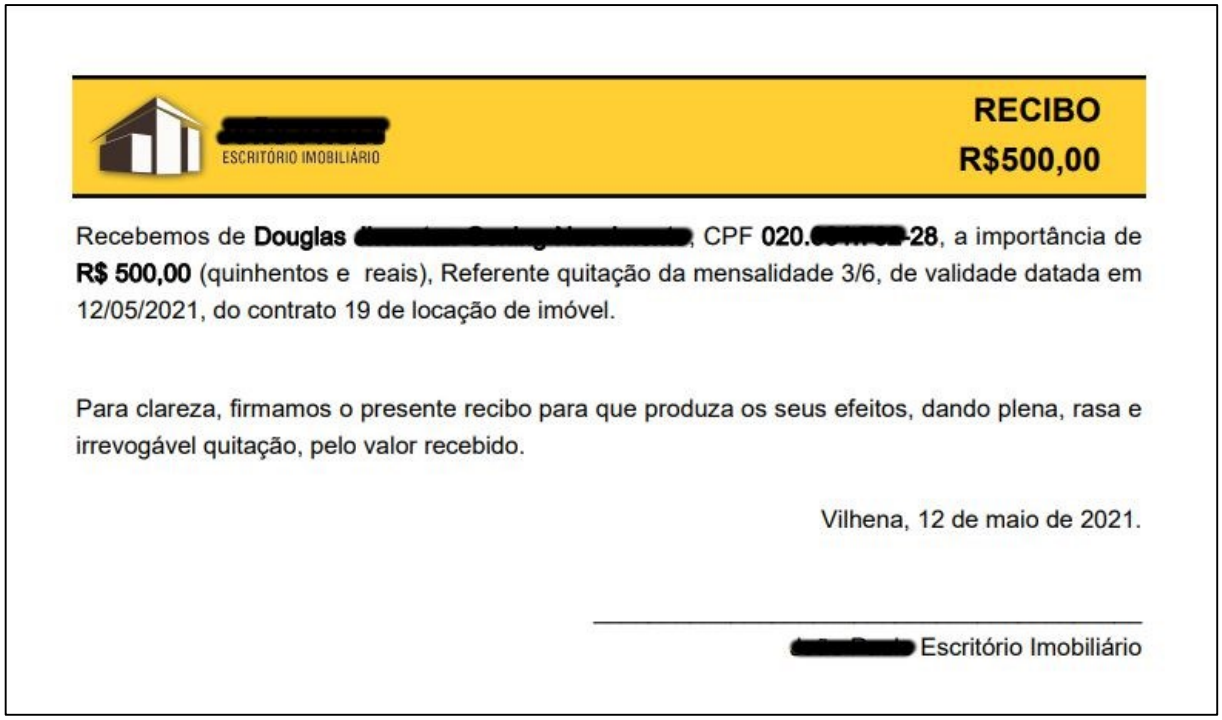
Nesta seção veremos exemplos de documentos gerados pelo sistema.

Os layouts dos documentos são configurados através de uma classe que armazena todas as informações predefinidas, como fonte, texto, e outros elementos que não se alteram, que são modelo. Os dados que se alteram, como informações das entidades envolvidas, valores e datas, são passados para estas classes através de métodos portando as informações necessárias para o *binding* de informações a concatenar.

O modelo dos documentos foi criado a partir de documentos adquiridos da empresa candidata. Alguns modelos dos relatórios foram propostos pelo desenvolvedor, pois não havia na empresa documentação útil para servir de base de criação de modelos.

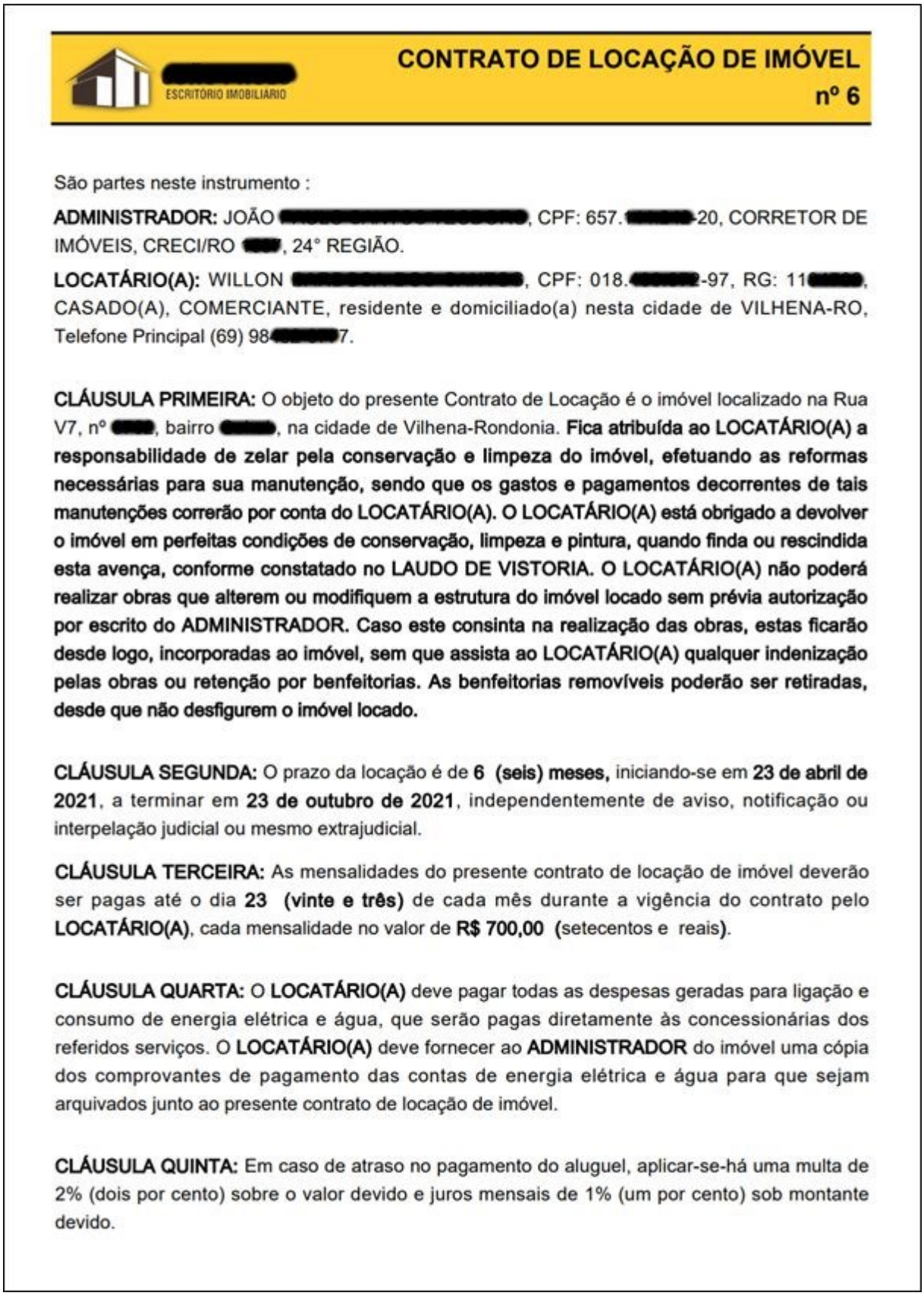
Visualizamos os modelos de Recibos, Contratos e Relatório de receitas e despesas nas figuras 38, 39 e 40, respectivamente.

Figura 38 - Recibos



Fonte: Próprio autor

Figura 39 - Primeira página de um contrato



[redacted] ESCRITÓRIO IMOBILIARIO

CONTRATO DE LOCAÇÃO DE IMÓVEL nº 6

São partes neste instrumento :

ADMINISTRADOR: JOÃO [redacted], CPF: 657.[redacted]20, CORRETOR DE IMÓVEIS, CRECI/RO [redacted], 24° REGIÃO.

LOCATÁRIO(A): WILLON [redacted], CPF: 018.[redacted]-97, RG: 11[redacted], CASADO(A), COMERCIANTE, residente e domiciliado(a) nesta cidade de VILHENA-RO, Telefone Principal (69) 98[redacted]7.

CLÁUSULA PRIMEIRA: O objeto do presente Contrato de Locação é o imóvel localizado na Rua V7, nº [redacted], bairro [redacted], na cidade de Vilhena-Rondonia. **Fica atribuída ao LOCATÁRIO(A) a responsabilidade de zelar pela conservação e limpeza do imóvel, efetuando as reformas necessárias para sua manutenção, sendo que os gastos e pagamentos decorrentes de tais manutenções correrão por conta do LOCATÁRIO(A). O LOCATÁRIO(A) está obrigado a devolver o imóvel em perfeitas condições de conservação, limpeza e pintura, quando finda ou rescindida esta avença, conforme constatado no LAUDO DE VISTORIA. O LOCATÁRIO(A) não poderá realizar obras que alterem ou modifiquem a estrutura do imóvel locado sem prévia autorização por escrito do ADMINISTRADOR. Caso este consinta na realização das obras, estas ficarão desde logo, incorporadas ao imóvel, sem que assista ao LOCATÁRIO(A) qualquer indenização pelas obras ou retenção por benfeitorias. As benfeitorias removíveis poderão ser retiradas, desde que não desfigurem o imóvel locado.**

CLÁUSULA SEGUNDA: O prazo da locação é de **6 (seis) meses,** iniciando-se em **23 de abril de 2021**, a terminar em **23 de outubro de 2021**, independentemente de aviso, notificação ou interpelação judicial ou mesmo extrajudicial.

CLÁUSULA TERCEIRA: As mensalidades do presente contrato de locação de imóvel deverão ser pagas até o dia **23 (vinte e três)** de cada mês durante a vigência do contrato pelo **LOCATÁRIO(A)**, cada mensalidade no valor de **R$ 700,00 (**setecentos e reais**)**.

CLÁUSULA QUARTA: O **LOCATÁRIO(A)** deve pagar todas as despesas geradas para ligação e consumo de energia elétrica e água, que serão pagas diretamente às concessionárias dos referidos serviços. O **LOCATÁRIO(A)** deve fornecer ao **ADMINISTRADOR** do imóvel uma cópia dos comprovantes de pagamento das contas de energia elétrica e água para que sejam arquivados junto ao presente contrato de locação de imóvel.

CLÁUSULA QUINTA: Em caso de atraso no pagamento do aluguel, aplicar-se-há uma multa de 2% (dois por cento) sobre o valor devido e juros mensais de 1% (um por cento) sob montante devido.

Fonte: Próprio autor

Figura 40 - Exemplo de relatório de receitas (entradas) da empresa

[redacted] ESCRITÓRIO IMOBILIÁRIO

Relatório de Receitas
Junho

Categoria	Descrição	Vencimento	Pagamento	Valor
Comissão Locação	Comissão de locação de imóvel - contrato nº 4	17/06/2021	21/06/2021	R$ 85,00
Comissão Locação	Comissão de locação de imóvel - contrato nº 20	25/05/2021	01/06/2021	R$ 77,00
Comissão Locação	Comissão de locação de imóvel - contrato nº 1	20/06/2021		R$ 65,00
Comissão Locação	Comissão de locação de imóvel - contrato nº 2	10/06/2021		R$ 45,00
Comissão Locação	Comissão de locação de imóvel - contrato nº 3	10/06/2021		R$ 85,00
Comissão Locação	Comissão de locação de imóvel - contrato nº 5	09/06/2021		R$ 75,00
Comissão Locação	Comissão de locação de imóvel - contrato nº 6	23/06/2021		R$ 70,00
Comissão Locação	Comissão de locação de imóvel - contrato nº 7	29/06/2021		R$ 70,00
Comissão Locação	Comissão de locação de imóvel - contrato nº 8	10/06/2021		R$ 80,00
Comissão Locação	Comissão de locação de imóvel - contrato nº 9	02/06/2021		R$ 140,00
Comissão Locação	Comissão de locação de imóvel - contrato nº 10	11/06/2021		R$ 45,00
Comissão Locação	Comissão de locação de imóvel - contrato nº 11	14/06/2021		R$ 80,00
Comissão Locação	Comissão de locação de imóvel - contrato nº 13	10/06/2021		R$ 50,00
Comissão Locação	Comissão de locação de imóvel - contrato nº 14	10/06/2021		R$ 65,00
Comissão Locação	Comissão de locação de imóvel - contrato nº 15	10/06/2021		R$ 75,00
Comissão Locação	Comissão de locação de imóvel - contrato nº 18	06/06/2021		R$ 90,00
Comissão Locação	Comissão de locação de imóvel - contrato nº 19	12/06/2021		R$ 50,00
Comissão Locação	Comissão de locação de imóvel - contrato nº 20	25/06/2021		R$ 77,00
Comissão Locação	Comissão de locação de imóvel - contrato nº 21	01/06/2021		R$ 80,00
Comissão Locação	Comissão de locação de imóvel - contrato nº 12	14/06/2021	30/05/2021	R$ 100,00
Comissão Locação	Comissão de locação de imóvel - contrato nº 23	10/06/2021		R$ 250,00
Total de receitas recebidas				R$ 262,00
Total de receitas ainda a receber				R$ 1.492,00

Fonte: Próprio autor

8.5 AVALIAÇÃO DO SOFTWARE

Utilizando o formulário de perguntas constante no Apêndice B, buscamos entender os processos após a implementação do software e os resultados obtidos para a segunda fase da pesquisa.

O quadro 12 mostra os resultados percebidos.

Quadro 12 - Resultados da segunda fase da pesquisa

Objetivos	Compreender os processos da empresa com o apoio da TI		
Premissa	Identificar as mudanças percebidas com a implantação do sistema		
Elementos de Análise	Atividades na Empresa	Armazenamento e leitura de informações	Métodos de documentação
Entrevistado 1	Secretária, realiza confecção de contratos e mensalidades, lança as entradas e saídas da empresa.	Todas as informações foram centralizadas no sistema, dados de pessoas e de imóveis, os contratos e mensalidades, o fluxo de caixa da empresa, são informações bem compreensíveis.	Os contratos são gerados com base nas informações inseridas na janela de geração, com todos os cálculos de mensalidades realizados. Os recibos são gerados pelo sistema automaticamente no momento de quitação, e agora há relatórios do fluxo de caixa da empresa.
Entrevistado 2	Contador, faz angariação dos imóveis para administração, realiza visitas com potenciais clientes, realiza vistorias em imóveis e intermedia negócios para terceiros	Agora com acesso direto aos dados através do sistema, consulta de qualquer dado e informação desejada de forma centralizada, onde tudo que é preciso está no sistema.	Os contratos e recibos são gerados automaticamente em cada operação que eles dependem. Não há mais erros ortográficos e os erros de cálculos praticamente desapareceram. Também há agora o relatório do fluxo de caixa.
Conclusão	As informações agora são centralizadas, únicas e confiáveis. As informações do negócio são todas mantidas no sistema, desde os dados de clientes até as entradas e saídas da empresa. Graças ao padrão estabelecido pelo sistema, não importa mais qual funcionário está utilizando o sistema pois todos usam da mesma forma. Os documentos são confiáveis e os dados são armazenados garantindo sua integridade.		

Fonte: Próprio autor.

Fica muito claro a positividade advinda da implementação do sistema. Dados de clientes, pessoas e imóveis que antes eram buscados em lugares diversos, como as mensagens trocadas entre a empresa e pessoas através de mensageiros como o WhatsApp, ou outras anotações não arquivadas ou em contratos anteriores de clientes, agora são registradas e encontradas em um só lugar e de maneira rápida, já que não é mais necessário ficar procurando em vários lugares nem em locais não indexados, levando mais tempo ainda para encontrar as informações.

Dados financeiros da empresa, como o fluxo de caixa contendo suas receitas e despesas, que antes eram arquivadas pelos cupons e notas fiscais, recibos, entre outros, propensos a erros de cálculos e eram anotados manualmente em planilhas e documentos sem formatação e de visualização cansativa, agora são corretamente representadas no sistema, calculadas e lançadas. Os documentos físicos citados continuam sendo arquivados, mas além disso, agora contam com relatórios e facilidade de gerenciamento pelo sistema.

Estes relatórios, somando a documentação os contratos e recibos, agora são também gerados automaticamente como parte do processo de utilização do sistema. Antes, cada recibo era gerado em uma plataforma separada, sendo necessário nova digitação e entrada de dados. Cada contrato era feito em documento eletrônico repetidas vezes, procurando sempre por erros de ortografia ou de digitação. Com o sistema estes erros praticamente desapareceram, uma vez que a entrada de dados ocorre apenas uma vez para o processo e sub processos, alcançando assim agilidade e confiabilidade das informações.

Além disso, a organização proporcionada pelo sistema trouxe um ambiente de trabalho mais leve e agradável. O motivo para tal é que a diferença entre as experiências antes e depois da implementação teve um impacto positivo em todos os atores do negócio, percebendo-se uma satisfação com os resultados obtidos pelo sistema no que tange a organização, a disponibilidade, a confiabilidade e velocidade de obtenção dos dados e informações.

Cada atualização lançada era recebida com empolgação pela novidade oferecida e pela facilidade imposta pelo software no tratamento de informações e geração de documentos que antes não havia sem ele. Através da pesquisa de satisfação, constatou-se plena satisfação da equipe da empresa com o software entregue.

9 CONCLUSÃO

Usar as informações geradas durante o tempo de atividade da empresa processadas de forma estruturada e adequada a favor de sua tomada de decisões e das estratégias de alavancagem de negócio, mesmo que no século XXI, ainda é negligenciado por alguns empresários. Empresas que recebem a tecnologia respondem melhor à competitividade de mercado, direcionam sua energia e tempo para aumentar a sua produtividade e retornos, graças a sistemas construídos para suprir a demanda de trabalho de dados e transformá-los em informações que são usadas para impactar positivamente os trabalhos de um negócio. Na contramão, vemos empresas que subestimam a tecnologia, atrasados em aderir um sistema que dá mais controle ao negócio. Tempo perdido procurando informações por difícil acesso e armazenamento incorreto, desconforto desnecessário entre a equipe por falha em processos e perda de recursos financeiros e intelectuais por erros humanos cruciais que seriam evitados com a utilização de um software são alguns dos produtos de uma falha de gestão alinhada à tecnologia.

É possível que este desinteresse seja fruto de produtos de mercado ineficientes, produtos muito caros, quer seja pela quantidade de recursos oferecidos ou por outros fatores que impactam em seu preço ou por produtos incompletos. Sistemas de prateleira estão sujeitos a tais percepções. Um sistema modelado com consultoria e levantamento dos requisitos desejados para um negócio específico é capaz de atender a essa individualidade de processos e métodos de forma eficaz quando bem executado desde o projeto até a entrega de atualizações ao cliente, permitindo até mesmo a distribuição a outros negócios graças aos padrões que podem ser implementados que favorecem a manutenção e atualização de um projeto.

A escolha da linguagem Java para o desenvolvimento, dos frameworks e ferramentas de auxílio ao desenvolvimento e interface gráfica atenderam às expectativas percebidas pelo desenvolvedor. A elevação da abstração percebida através do uso do framework Spring Boot reduziu drasticamente a carga de trabalho e tempo com desenvolvimento, já que não foi necessária configuração manual do projeto devido a autoconfiguração do framework, nem criação de classes DAO para

acesso aos dados no banco de dados graças à JPA, nem de download manual de bibliotecas adicionais graças ao gerenciador de dependências do Maven. Ainda que este processo seja todo automatizado, o desenvolvedor pode supervisionar o trabalho do framework e realizar adequações se necessário. Quanto a parte gráfica, a utilização do JavaFX trouxe uma série de benefícios. Com o recurso de utilização de arquivos FXML para construção das interfaces, poupou-se tempo de aprendizado de código graças a edição destes com o Scene Builder, manipulando os elementos gráficos de forma visual, e não por código. A estilização via CSS também permitiu personalizar a aplicação manipulando cada detalhe de cada elemento gráfico, trazendo satisfação e identificação do cliente e sua marca com o software entregue.

Um passo além da teoria, percebeu-se na prática como a tecnologia pode oferecer à empresa suporte aos seus desafios e solucionar problemas. Traz organização, integridade, disponibilidade e velocidade de acesso aos dados e informações pertinentes ao negócio, Documentos propensos a erros ortográficos ou de digitação tem sua superfície de falhas muito reduzida por modelos revisados e predefinidos, onde as falhas são reduzidas unicamente à entrada incorreta de dados. Cálculos propensos ao mesmo tipo de falhas também passam a ser confiáveis com a retirada do fator humano. Por fim, instala-se uma padronização dos processos instaurada pelo software utilizado, que se estende a todos os funcionários futuros da empresa.

Foi uma experiência única aplicar os conhecimentos obtidos nestes anos de estudo ao desenvolvimento deste trabalho e deste projeto. O conteúdo consumido através do referencial teórico pesquisado e a experiência obtida com o processo de desenvolvimento do software abriram a mente e os olhos para horizontes ainda mais profundos e além do que se espera para uma carreira promissora. É preocupante perceber que existe uma classe de pessoas em quem ainda não despertou o interesse pelas maravilhas oferecidas pela tecnologia, e como desenvolvedor de software, fico feliz em fazer parte de uma comunidade que está empenhada em uma missão que tem o seu objetivo principal bem definido: entregar soluções de problemas e desafios.

Espero que tudo isso que motiva alcance ainda mais pessoas para este mundo da tecnologia, onde a rede de conhecimento nunca para, a cada dia que passa nos levando a níveis mais elevados de intelecto, onde nem o céu é mais um limite.

10 REFERÊNCIAS BIBLIOGRÁFICAS

APACHE MAVEN PROJECT. **Welcome to Apache Maven**. Página Inicial. Disponível em: <http://maven.apache.org/>. Acesso em 06 de maio de 2021.

AQUILES, Alexandre; FERREIRA, Rodrigo. **Controlando versões com Git e GitHub**. Casa do Código, 2014.

CARVALHO, Vinícius. **MYSQL**: Comece com o principal banco de dados open source do mercado. Casa do Código, 2015.

CATON, Lucas. 6 Motivos para usar PostgreSQL em vez do MySQL. **Lucas Caton**, janeiro de 2018. Disponível em <https://www.lucascaton.com.br/2018/01/31/6-motivos-para-usar-o-postgresql-em-vez-do-mysql>. Acesso em 15 de abril de 2021.

COSTA, Carlos. A Aplicação da Linguagem de Modelagem Unificada (UML) para o Suporte ao Projeto de Sistemas Computacionais Dentro de Um Modelo de Referência. **Gestão e Produção**, Caxias do Sul, vol. 8, n. 1, p. 19-36, 2001.

DEITEL, Paul; DEITEL, Harvey. **Java**: Como programar. 7. Ed. Pearson Universidades, junho de 2016.

DOYLE, Daniella. Metodologias de Gestão: Exemplos de indicadores de desempenho e como usar cada um deles. **Siteware**, 2018. Disponível em <https://www.siteware.com.br/blog/metodologias/exemplos-de-indicadores-de-desempenho-nas-empresas/>. Acesso em: 20 de março de 2021.

GONÇALVES, Eduardo. **SQL:** Uma abordagem para banco de dados Oracle. Casa do Código, 2014.

GRINEV, Sergey. **Mastering JavaFX 10**. Packt Publishing, maio de 2018.

GROFFE, Renato. Modelagem de Sistemas através de UML: Uma visão Geral. **DEVMEDIA**, 2013. Disponível em: <https://www.devmedia.com.br/modelagem-de-sistemas-atraves-de-uml-uma-visao-geral/27913>. Acesso em 15 de março de 2020.

GROFFE, Renato. Desenvolvimento ágil com Scrum: uma visão geral. **DEVMEDIA**, 2012. Disponível em: < https://www.devmedia.com.br/desenvolvimento-agil-com-scrum-uma-visao-geral/26343>. Acesso em 20 de março de 2020.

GUERRA, Eduardo. **Design Patterns com Java**: Projeto orientado a objetos guiado por padrões. Casa do Código, 2014.

JAVA DOCUMENTATION. JavaFX: Getting Started with JavaFX. **Oracle**, 201-?. Disponível em <https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm#JFXST784>. Acesso em 19 de abril de 2021.

JAVAFX. **JavaFX**. Página Inicial. Disponível em <https://openjfx.io/index.html>. Acesso em 06 de maio de 2021.

MACEDO, Bruno Armindo; PEDRON, Cristiane Drebes; CATELA, Miguel Ferreira. Service level agreement em cloud computing: um estudo de caso em uma empresa portuguesa. **Universitas: Gestão e TI**, Brasília, ano 2014, v. 4, n. 1, p. 13-23, 17 maio 2014.

MORAES, Giseli; TERENCE, Ana; FILHO, Edmundo. A Tecnologia da Informação como Suporte à Gestão Estratégica da Informação na Pequena Empresa. **Revista de Gestão da Tecnologia e Sistemas de Informação**, São Paulo, vol. 1, n. 1, p. 27-43, 30 de maio de 2004.

PERRY, J. Steven. Fundamentos da Linguagem Java: Programação orientada a objetos na plataforma Java. **IBM Developer**, 19 de julho de 2010. Disponível em: <https://developer.ibm.com/br/tutorials/j-introtojava1/>. Acesso em 23 de abril de 2021.

POSTGRESQL. **PostgreSQL**: The World's Most Advanced Open Source Relational Database. Disponível em <postgresql.org>. Acesso em 15 de abril de 2021.

RADIGAN, Dan. Kanban: Como a metodologia Kanban é aplicada ao desenvolvimento de software. **Atlassian Agile Coach**, 2021. Disponível em: <https://www.atlassian.com/br/agile/kanban>. Acesso em 26 de fevereiro de 2021.

REHKOPF, Max. Kanban vs Scrum: que tipo de ágil é você? **Atlassian Agile Coach**, 2021. Disponível em: <https://www.atlassian.com/br/agile/kanban/kanban-vs-scrum>. Acesso em 26 de fevereiro de 2021.

SALMON, Ariana Zoila Oliveira. **Modelagem e Análise de Requisitos de Sistemas Automatizados Usando UML e Redes de Petri.** 2017. 162 p. Tese (Doutorado em Engenharia de Controle e Automação Mecânica) - Escola Politécnica da Faculdade de São Paulo, São Paulo, 2017.

SILVEIRA, Paulo; TURINI, Rodrigo. **Java 8 Prático**: Lambdas, Streams e os novos recursos da linguagem. Casa do Código, 2014.

SOLOVYOVA, Viktoria. O que é o quadro Kanban e como usá-lo. **Bitrix24**, 2019. Disponível em <https://www.bitrix24.com.br/blogs/dicas/o-que-o-quadro-kanban-e-como-us-lo.php>. Acesso em 26 de fevereiro de 2021.

SOUZA, Marcio B. Padrão MVC: Java Magazine. **DEVMEDIA**, 2011. Disponível em: <https://www.devmedia.com.br/padrao-mvc-java-magazine/21995>. Acesso em 20 de março de 2020.

SPRING IO. **Spring Data JPA** - Reference Documentation. Disponível em: <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#reference>. Acesso em 05 de maio de 2021.

SYDLE. Metodologia Ágil Scrum: como trabalhar com Scrum? **SYDLE**, 2020. Disponível em: <https://www.sydle.com/br/blog/metodologia-agil-scrum%20-5f6dc45f320703787497f887/>. Acesso em 04 de dezembro de 2020.

TOTVS. Kanban: Conceito, como funciona, vantagens e implementação**. TOTVS**, 2021. Disponível em: <https://www.totvs.com/blog/negocios/kanban/>. Acesso em: 24 de fevereiro de 2021.

UOL MEU NEGÓCIO. Conheça o banco de dados PostgreSQL. **UOL MEU NEGÓCIO**, junho de 2020. Disponível em: <https://meunegocio.uol.com.br/blog/conheca-o-banco-de-dados-postgresql/>. Acesso em 15 de abril de 2021.

WALLS, Craig. **Spring Boot in Action**. New York, Manning Publications, Co., 2016.

O'CONNER, John. Using the Java Persistence API in Desktop Applications. **Oracle**, 2007. Disponível em <https://www.oracle.com/technical-resources/articles/javase/persistenceapi.html>. Acesso em 04 de maio de 2021.

WEISSMAN, Henrique L. **Vire o Jogo com Spring Framework**. Casa do Código, 16 de abril de 2014.

XGEN. Aplicação Web ou Desktop: qual a melhor solução? **XGEN**, 2020. Disponível em <https://xgen.com.br/blog/aplicacao-web-ou-desktop-qual-a-melhor-solucao>. Acesso em: 25 de março de 2021.

YIN, Robert K. **Case Study Research and Applications: Design and Methods**. 6ª edição. Los Angeles, SAGE Publications, 2018.

APÊNDICE A – Formulário de Levantamento de Informações

Função do entrevistado:

Data de coleta:

1 – Qual a data de abertura da empresa?
R:

2 – Quando iniciou o gerenciamento de contratos?
R:

3 – Como eram armazenadas as informações das entidades relacionadas ao negócio? (Considere entidades como clientes, proprietários, imóveis, corretores, todos os atores participantes do negócio)
R:

4 – Quantos contratos tem arquivado do passado?
R:

5 – Existe um plano de tomada de decisão para aumentar a eficiência da empresa? (Informações que influenciam, por exemplo, no valor do imóvel)
R:

6 – Como são armazenadas hoje as informações das entidades relacionadas ao negócio?
R:

7 – A empresa hoje dispõe de algum controle sistematizado de suas despesas e receitas?
R:

8 – Como são organizadas as informações de negócios e fluxo de caixa da empresa?
R:

9 – Qual o produto que a empresa idealiza para evoluir a gestão do negócio?
R:

10 – A empresa já tomou prejuízo ou deixou de fechar um negócio por falha com as informações de clientes, imóveis, contratos ou mensalidades?
R:

11 – Há outras observações que gostaria de fazer?
R:

APÊNDICE B – Formulário de Avaliação do Software

Função do entrevistado:

Data de coleta:

1 – A implementação do software correu como planejado do início ao fim ou houve novas solicitações de funções ou alterações?
R:

2 – As atualizações solicitas eram entregues de que forma em com que prazos?
R:

3 – As atualizações lançadas supriam as expectativas para o que era solicitado?
R:

4 – De que maneira são armazenadas agora as informações de entidades do negócio?
R:

5 – De que maneira são armazenados agora os contratos gerados pela empresa?
R:

6 – Como são gerados agora os contratos e recibos pela empresa?
R:

7 – Como são visualizadas as informações desejadas pelo operador agora?
R:

8 – De 0 a 5, qual o grau de satisfação com os seguintes itens do sistema?

a) Facilidade de uso:
b) Visual da aplicação:
c) Segurança das informações:
d) Relatórios desejados:
e) Fluxo de trabalho:
f) Atualizações e prazos:

9 – Cite os resultados obtidos com a implantação do sistema e outras observações.
R:

ANEXO A – Licença de Armazenamento não exclusivo

SÃO LUCAS
EDUCACIONAL

LICENÇA DE ARMAZENAMENTO E DISTRIBUIÇÃO NÃO EXCLUSIVA

Autor: Fabrício Gustavo Wagomacker Rocha

RG.: 11611 SESDEC/RO CPF: 012.842.572-58 E-mail: fabricio.gwr@gmail.com

Orientador: Willian Fachetti Coordenação: Thyago Borges

Título do documento: DESENVOLVIMENTO DE APLICATIVO PARA GERENCIAMENTO FINANCEIRO DE UMA EMPRESA CORRETORA DE IMÓVEIS NA CIDADE DE VILHENA-RO

TERMO DE DECLARAÇÃO

Declara que o documento entregue é seu trabalho original, e que detém o direito de conceder os direitos contidos nesta licença. Declara também que a entrega do documento não infringe, tanto quanto lhe é possível saber, os direitos de qualquer outra pessoa ou entidade.

Declara que, se o documento entregue contém material do qual não detém os direitos de autor, obteve autorização do detentor dos direitos de autor para conceder à Faculdade São Lucas os direitos requeridos por esta licença, e que esse material cujos direitos são de terceiros está claramente identificado e reconhecido no texto ou conteúdo do documento entregue. Se o documento entregue é baseado em trabalho financiado ou apoiado por outra instituição que não a Faculdade São Lucas, declara que cumpriu todas as obrigações exigidas pelo respectivo contrato ou acordo. Termo de Autorização

Na qualidade de titular dos direitos de autor do conteúdo supracitado, autorizo que: a Biblioteca Dom João Batista Costa da Faculdade São Lucas pode converter e disponibilizar gratuitamente em seu repositório institucional a obra em formato eletrônico de acordo com a licença pública Creative Commons CC BY-NC-ND; que pode manter mais de uma cópia da obra depositada para fins de segurança, back-up e/ou preservação.

A obra continua protegida por Direito Autoral e/ou por outras leis aplicáveis. Qualquer uso da obra que não o autorizado sob esta licença ou pela legislação autoral é proibido.

Ji-Paraná, 30 de junho de 2021.

Fabricio

ASSINATURA DO AUTOR E/OU DETENTOR DOS DIREITOS AUTORAIS

PLANO DE TRABALHO DE ORIENTAÇÃO – REGISTRO

Docente: WILLIAN ALVES DE OLIVEIRRA FACHETTI

Discentes: FABRÍCIO GUSTAVO WAGOMACKER ROCHA

WILLIAN ALVES DE OLIVEIRA FACHETTI:2044137

Assinado de forma digital por WILLIAN ALVES DE OLIVEIRA FACHETTI:2044137
Dados: 2021.06.13 14:41:53 -04'00'

HORA/DATA	ATIVIDADE DESENVOLVIDA	CORREÇÕES	ASSINATURAS
19h00 13/03/21	Revisão TCC I e levantamento alterações para TCC II		Fabrício
19h00 27/03/21	Revisão dos Materiais e Método		Fabrício
19h00 10/04/21	Revisão Requisitos Funcionais	Finalização Materiais e Método	Fabrício
19h00 17/04/21	Revisão Diagramas UML		Fabrício
19h00 08/05/21	Finalização dos Resultados e Discussão		Fabrício
19h00 22/05/21	Organização Resultados e Discussão		Fabrício
19h00 05/06/21	Revisão Resultados e Discussão	Resultado e Discussão	Fabrício
19h00 12/06/21	Revisão Final TCC II	Revisão Final TCC II	Fabrício

Obs.: Neste documento serão designados os pontos a serem desenvolvidos no TCC, que deve ter como prazo final o estabelecido pela coordenação do curso.

ANEXO B – Plano de Trabalho de Orientação

ANEXO C – Carta de Aceite de Orientação

CARTA DE ACEITE DE ORIENTAÇÃO

Eu, **WILLIAN ALVES DE OLIVEIRA FACHETTI**, professor efetivo do curso de **SISTEMAS DE INFORMAÇÃO,** aceito orientar o trabalho de conclusão de curso do aluno **FABRÍCIO GUSTAVO WAGOMACKER ROCHA** de acordo com a linha de pesquisa abaixo, cumprindo os deveres e atuações do orientador previsto no Regulamento de Trabalho de Conclusão de Curso do Centro Universitário São Lucas.

Linha de pesquisa:

DESENVOLVIMENTO DE APLICATIVO PARA GERENCIAMENTO FINANCEIRO DE UMA EMPRESA CORRETORA DE IMÓVEIS NA CIDADE DE VILHENA-RO

Assim, comprometo-me a desenvolver um plano de trabalho com o orientando que será executado pelo mesmo e conduzido por minha atuação profissional e científica.

Ji-Paraná, 10 de março de 2021.

Assinatura do professor orientador: Assinado por: WILLIAN ALVES DE OLIVEIRA FACHETTI:2044137

Assinatura do Orientando: _____

São Lucas Educacional Ji-Paraná
Av. Eng. Manfredo Barata Almeida da Fonseca, 542
Jd. Aurélio Bernardi | Ji-Paraná | RO | CEP 76907-438