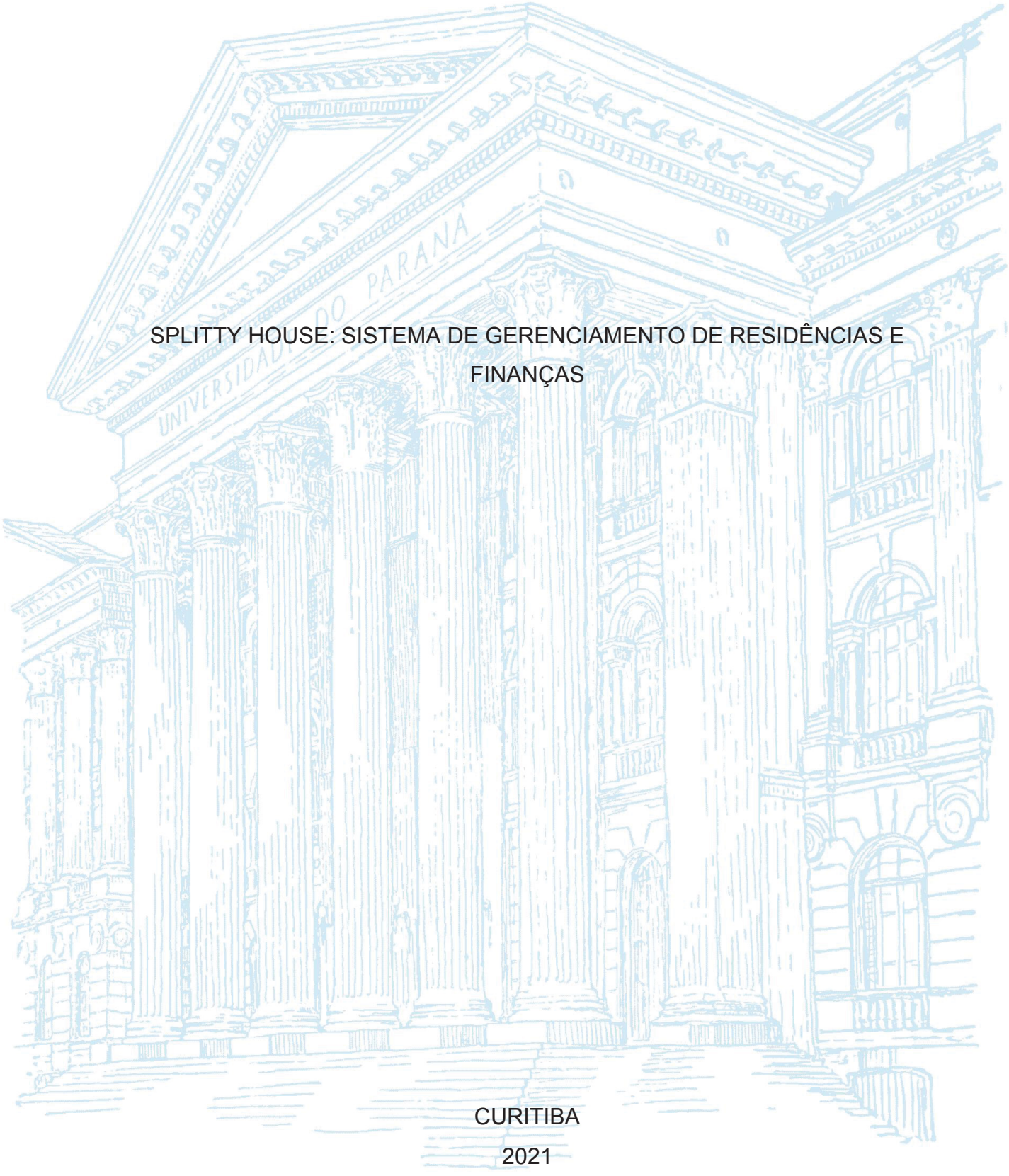
UNIVERSIDADE FEDERAL DO PARANÁ

BRUNO FELIPE RIBEIRO WOSIAK



SPLITTY HOUSE: SISTEMA DE GERENCIAMENTO DE RESIDÊNCIAS E FINANÇAS

CURITIBA

2021

BRUNO FELIPE RIBEIRO WOSIAK

SPLITTY HOUSE: SISTEMA DE GERENCIAMENTO DE RESIDÊNCIAS E FINANÇAS

Monografia apresentada como requisito parcial à obtenção do grau de Especialista em Engenharia de Software, no Curso de Pós-Graduação em Engenharia de Software, Setor de Educação Profissional e Tecnológica, da Universidade Federal do Paraná.

Orientador: Prof. Dr. Razer Anthom Nizer Rojas Montaño

CURITIBA

2021

MINISTÉRIO DA EDUCAÇÃO
SETOR DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
UNIVERSIDADE FEDERAL DO PARANÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
CURSO DE PÓS-GRADUAÇÃO ENGENHARIA DE SOFTWARE - 40001016231E1

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em ENGENHARIA DE SOFTWARE da Universidade Federal do Paraná foram convocados para realizar a arguição da Monografia de Especialização de **BRUNO FELIPE RIBEIRO WOSIAK** intitulada: **SPLITTY HOUSE: SISTEMA DE GERENCIAMENTO DE RESIDENCIAS E FINANCAS**, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua Aprovação no rito de defesa.

A outorga do título de especialista está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

Curitiba, 10 de Junho de 2021.

RAZER ANTHOM NIZER ROJAS MONTAÑO
Presidente da Banca Examinadora (UNIVERSIDADE FEDERAL DO PARANÁ)

JAIME WOJCIECHOWSKI
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

RESUMO

Devido à grande demanda por residências no Brasil, muitos jovens com baixo poder aquisitivo buscam colaboradores para dividir uma residência e mitigar os custos. Esse cenário, em conjunto com a falta de interesse em finanças pessoais, resulta em diversos problemas relativos às relações pessoais, principalmente devido a problemas com as finanças coletivas do imóvel e de responsabilidades coletivas do cotidiano.

Mesmo havendo diversas soluções que podem auxiliar os usuários nessas situações, as aplicações são dispersas e solucionam os diversos problemas apenas individualmente. Com isso, através dos conhecimentos obtidos durante a realização da pós-graduação foi possível desenvolver o Splitty House, um sistema que centraliza as resoluções desses problemas.

O objetivo atingido deste trabalho foi desenvolver um *software* que gerencie uma residência compartilhada que promove a organização dos moradores em conjunto com as atividades e finanças coletivas e individuais, propondo transparência financeira e fomentando o bom convívio em comunidade.

Palavras-chave: Moradia. Finanças. Custos. Moradia dividia. Splitty House.

ABSTRACT

Due to the high demand for housing in Brazil, many young people with low purchasing power are looking for roommates to share a house and mitigate costs. This scenario, together with the lack of interest in personal finances, results in several issues related to social interactions, mainly because of problems with the collective finances of the property and common daily responsibilities.

Even though there are several solutions that can help users in these situations, the applications are dispersed and only solve the different problems individually. Thus, through the knowledge obtained during the postgraduate course, it was possible to develop the Splitty House, a system that centralizes the resolution of these problems.

The goal of this work was to develop a software that manages a shared residence and promotes the organization of the roommates to deal with collective and individual activities and finances, proposing financial transparency and stimulating good community living.

Keywords: Housing. Finances. Costs. Shared housing. Splitty house.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

HTML	-	HyperText Markup Language
JVM	-	Java Virtual Machine
CSS	-	Cascading Style Sheets
SPC	-	Serviço de Proteção ao Crédito
IBGE	-	Instituto Brasileiro de Geografia e Estatística
UML	-	Linguagem de modelagem unificada
SPA	-	Single Page Application
API	-	Application Programming Interface
AOP	-	Aspect Oriented Programming
XML	-	Extensible Markup Language
DOM	-	Document Object Model
SQL	-	Structured Query Language
IDE	-	Integrated Development Environment
SGBD	-	Sistema de gerenciamento de banco de dados
CRUD	-	Create, read, update e delete
PDF	-	Portable Document Format
REST	-	Representational State Transfer
HTTP	-	Hypertext Transfer Protocol

SUMÁRIO

1 INTRODUÇÃO

O mercado imobiliário no Brasil nas últimas décadas foi instável, passando por diversos momentos de crescimento, assim como, de baixa produtividade. Isso se deve porque o setor é diretamente afetado pela economia do país. Tendo como perspectiva o período dos últimos doze anos, é possível observar que essas oscilações estão diretamente ligadas às instabilidades políticas que ocorreram nesse intervalo de tempo no país (ISTOÉ, 2020).

Essa volatilidade do mercado de imóveis afeta diretamente a população, principalmente em relação aos proprietários que não conseguem vender os seus imóveis. Por consequência disto, o setor de locação de residências recebe uma enorme quantidade de propriedades disponíveis, resultando em uma queda no preço do aluguel, inclusive com reajustes abaixo da inflação (GAZETA DO POVO, 2020).

Tendo em vista esse cenário sensível do mercado imobiliário, a população se adequou criando padrões de consumo, para contornar as dificuldades financeiras. Umas das alternativas que cresceram em tempos de crise foi o compartilhamento de residências. Buscando morar em localizações privilegiadas e em imóveis em boas condições, as pessoas se reúnem para dividir moradias com o objetivo de reduzir ainda mais os custos. Além disso, compartilham as atividades correlacionadas à manutenção dos ambientes, e do bom convívio em comunidade. Buscam também, criar vínculos mais afetivos, se afastando da ideia mais tradicional das repúblicas (ESTADÃO, 2017).

Por outro lado, há também uma perspectiva diferente da econômica, a qual leva as pessoas a dividirem suas casas: busca por companhia e sustentabilidade. Há aumento no movimento de pessoas, em uma faixa etária na média dos 30 anos, que buscam ter um convívio social mais ativo em conjunto com princípios sustentáveis, para reger uma vida com maior qualidade em grandes centros urbanos (FOLHA DE SÃO PAULO, 2016).

De acordo com o censo demográfico do IBGE (Instituto Brasileiro de Geografia e Estatística) de 2010, é possível confirmar essa tendência. Houve um crescimento considerável dos lares dos quais as pessoas sem um grau de parentesco convivem. Em torno de 400 mil residências possuem esse padrão no Brasil.

1.1 PROBLEMA

No Brasil, a população no geral não tem um cuidado com suas finanças particulares. De acordo com uma pesquisa do SPC (Serviço de Proteção ao Crédito) realizada em 2016, foi constatado que cerca de 46% dos brasileiros não cuidam de forma correta do seu dinheiro. Este fato reflete no bom convívio comunitário em residências compartilhadas. Pois a falta de apreço com o dinheiro pode refletir na vida de terceiros, tendo em vista que as responsabilidades são compartilhadas, deixando o convívio pouco saudável e gerando conflitos, ou até mesmo inviabilizando a convivência (ABRIL, 2017).

Em contrapartida, há outros problemas, que não envolvem dinheiro, que ocorrem em um convívio comunitário. Compartilhar o espaço com pessoas diferentes, é uma tarefa que exige paciência e tolerância. Questões como tarefas diárias, relacionadas às áreas e itens de uso comum, em conjunto com a privacidade dos indivíduos limitada, podem gerar desconfortos entre os moradores. Por esses motivos, as pessoas devem fazer sacrifícios relacionados a sua liberdade individual, com a finalidade de manter o ambiente de convivência o mais agradável possível (UOL, 2013).

Contudo, com a tecnologia é possível prevenir a grande parte desses problemas. Atualmente, no mercado existem diversas soluções que ajudam nas questões relacionadas a divisão de uma residência, ou com finanças pessoais. Desde planilhas complexas, para gerenciamento de custos, a aplicativos de celulares, com as mais diversas finalidades, como controle de tarefas, gastos, divisão de custos etc.

Porém, o grande problema é que essas soluções tecnológicas são dispersadas através de diversas aplicações diferentes, e consequentemente, para o usuário gerir sua vida pessoal, em companhia com a organização da vida coletiva, ele precisa gerenciar diversas aplicações diferentes, gerando um acréscimo de trabalho.

É possível observar esse comportamento no ambiente *mobile* por exemplo. O brasileiro em média nunca utilizou por volta de 25% dos aplicativos que tem instalados em seu *smartphone*, e aproximadamente 23% abandonam após a primeira utilização (OLHAR DIGITAL, 2018). Ou seja, com um maior número de

sistemas necessários para gerir um conjunto de tarefas, maior a probabilidade de eles não serem utilizados ou descontinuados.

1.2 OBJETIVOS

Nessa seção são apresentados os objetivos gerais e específicos do projeto.

1.2.1 Objetivo Geral

O objetivo geral do trabalho é desenvolver um software que possibilite o gerenciamento de uma residência compartilhada, assim como finanças pessoais dos usuários.

1.2.2 Objetivo Específicos

Os objetivos específicos são:

a) Possibilitar aos usuários cadastrarem sua residência e que possam convidar outros moradores a ingressarem em seu imóvel;

b) Possibilitar a divisão dos custos do imóvel de forma igual e transparente;

c) Possibilitar a divisão das tarefas essenciais para o funcionamento e manutenção da residência;

d) Possibilitar que os usuários definam e compartilhem as regras do imóvel a serem seguidas por todos;

e) Possibilitar os usuários a cadastrarem o inventário do imóvel, distinguindo item por item e referenciando seu proprietário;

f) Possibilitar que os usuários possam gerir suas finanças pessoais, com os custos da divisão do imóvel integrado.

1.3 JUSTIFICATIVA

Considerando todos os fatos apresentados, a proposta do presente trabalho é de desenvolver um sistema que centralize as atividades necessárias para um controle financeiro pessoal em conjunto com ferramentas para realizar a gestão de uma residência compartilhada.

Com isso, facilitando a vida dos usuários responsáveis pelo imóvel, assim como para os demais moradores, resultando em um ambiente justo e transparente, focando em educação financeira e incentivando uma boa convivência em comunidade.

No próximo capítulo serão apresentados a fundamentação teórica que fundamenta o desenvolvimento deste presente trabalho.

1.4 ESTRUTURA DO DOCUMENTO

Este documento apresenta todas as etapas da elaboração e desenvolvimento do projeto conforme estrutura descrita a seguir.

O Capítulo 2 tem como propósito descrever a fundamentação teórica, na qual complementa os tópicos apresentados no Capítulo 1.

O Capítulo 3 descreve a metodologia aplicada pelo desenvolvedor para o desenvolvimento e gerenciamento do projeto.

O Capítulo 4 ilustra a arquitetura do sistema desenvolvido, assim como a descrição das funcionalidades do software, detalhes técnicos relevantes e suas telas implementadas.

No Capítulo 5 é discutido se o objetivo geral e os específicos do projeto foram atingidos. Além disso, também é argumentado questões relativas ao processo de desenvolvimento e trabalhos futuros.

Ao final do documento encontra-se os apêndices com os diagramas confeccionados para a modelagem do sistema.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão apresentados conceitos e tecnologias primordiais para concepção, estruturação e desenvolvimento do sistema proposto.

2.1 FUNDAMENTAÇÃO TEÓRICA CONCEITUAL

A seguir, serão descritos conceitos utilizados para a modelagem, concepção e arquitetura do sistema.

2.1.1 Moradia compartilhada e colaborativa

O conceito de moradia compartilhada é uma prática que ocorre há muito tempo no mundo. Na perspectiva do Brasil, esse tipo de opção de moradia surgiu no período de pós abolição da escravidão, devido à alta demanda por habitação. Fato que resultou na criação de cortiços, repúblicas de estudantes, estalagens entre outros (NUNES; VIEIRA, 2019).

Já no período urbano moderno do país, com a verticalização das moradias, esses conceitos foram expandidos para compartilhamentos de áreas comuns em edifícios e apartamentos. Desta maneira, dando espaço para novas ideias relacionadas à economia compartilhada e colaborativa, como *coworking*, *cohousing* e *coliving* (NUNES; VIEIRA, 2019).

O termo *coliving* é utilizado atualmente para se referir à prática de divisão de uma residência alugada por indivíduos, que geralmente não se conhecem. Tendo como objetivo otimizar custos e possibilitar o acesso a uma habitação de maior qualidade, sendo fatores como localidade e praticidade de mobilidade urbana, determinantes na escolha do imóvel (NUNES; VIEIRA, 2019).

Além disso, há também fatores humanos relativos à convivência, que possibilitam a troca de experiências de vida e profissionais entre os moradores. Pois, geralmente o público-alvo desse tipo de modalidade de moradia são jovens solteiros da geração *Millennials* (jovens nascidos entre 1980-1990), os quais são mais conectados com as tecnologias e desapegados a bens materiais, e que por muitas vezes buscam uma vida social mais agitada (NUNES; VIEIRA, 2019).

Por fim, fica claro que essa modalidade de habitação é uma tendência crescente em grandes centros urbanos no Brasil. Além dos fatores já citados anteriormente, há a adição das facilidades burocráticas para realizar o contrato de aluguel, abrindo uma opção de mercado para empresas que gerenciam e oferecem esse tipo de serviço no país (FOLHA DE SÃO PAULO, 2018).

2.1.2 A importância do planejamento financeiro

Gerenciar finanças pessoais ou profissionais, sempre foi um grande desafio para o homem. Manipular e planejar estrategicamente números é uma tarefa difícil para muitas pessoas. Dado que, para se obter sucesso nesta tarefa, é necessário dedicação, planejamento e pensamento estratégico (ANDREBONA, 2018).

Eventualmente o cuidado com finanças pessoais é indevidamente associado a uma atividade chata, complexa e complicada, na qual envolve muitos cálculos e matemática em geral. Porém essa imagem deve ser alterada, pois na verdade essa atividade é responsável pelas grandes conquistas e realizações pessoais dos indivíduos (ANDREBONA, 2018).

Segundo Gitman (2010), o planejamento financeiro pessoal deve ser iniciado com as definições de metas, de forma realista e cuidadosa e de acordo com seus objetivos. Adicionalmente, também deve ser definido as estimativas de custo, nível de prioridade e o prazo no qual os propósitos devem ser realizados, podendo ser a curto, médio e longo prazo.

Há também ferramentas e práticas, que auxiliam a pessoa a gerir e executar da melhor forma seu planejamento financeiro. Como softwares de controle de gastos, quitação de dívidas, poupar dinheiro regularmente, realizar bons investimentos, obter um reserva de emergência e compreender e acompanhar gastos com cartões de crédito (ANDREBONA, 2018).

2.1.3 Postulados de *design SOLID*

As linguagens de programação orientadas a objetos são bem consolidadas atualmente, tanto no âmbito profissional como no ambiente acadêmico. Entretanto,

apesar da sua popularidade, esse paradigma possui conceitos complexos que são de difícil aplicabilidade no mundo real dos projetos. Por consequência disto, é gerado por parte dos profissionais mais inexperientes, muito código procedural disfarçado de orientado a objeto (ANICHE, 2020).

Em orientação a objeto, o mais importante na prospecção do código é o desenvolvimento estrutural das classes, o que define como elas se encaixam e como serão estendidas para solucionar algum problema proposto. E por muitas vezes, a elaboração dessas estruturas são difíceis de serem criadas, devido à complexidade do processo (ANICHE, 2020).

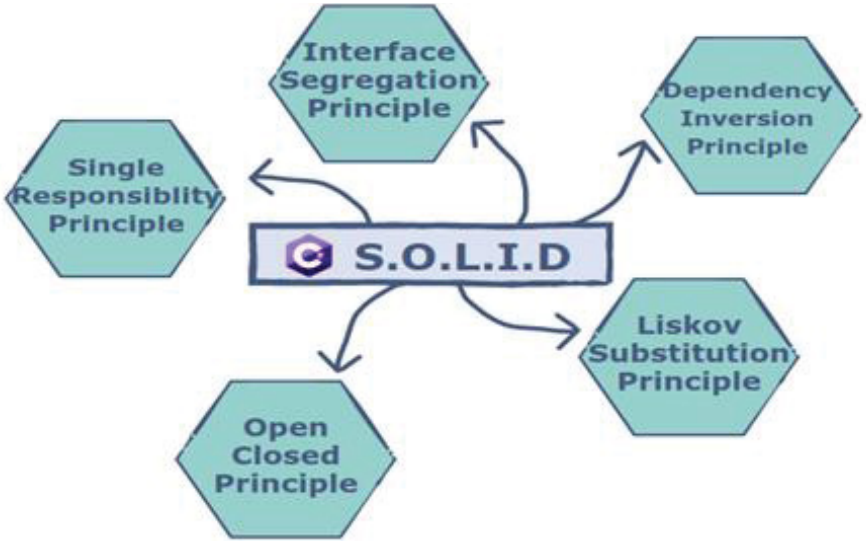
Em consequência disto, softwares podem obter um *design* ruim, o que fica claro, principalmente quando há a necessidade de realizar alguma alteração no código. Sendo esta, uma ação que pode quebrar o sistema em diversos locais, devido a propagação das alterações entre as classes do projeto (ANICHE, 2020).

Por outro lado, há padrões de projetos já bem estabelecidos que auxiliam os programadores na criação de uma estrutura de código para linguagens de programação orientada a objetos. Tendo como objetivo criar um software flexível, claro e de fácil manutenção, no qual conforme cresce em tamanho, diminui em complexidade (BAELDUNG, 2019).

Contudo, é possível condensar alguns conceitos de padrões de projetos, visando facilitar o desenvolvimento de um bom código. Um exemplo desse compilado são os postulados de design *SOLID*. A palavra *SOLID* é um acrônimo criado por Michael Feathers, no qual reúne 5 princípios de *design* criados e introduzidos por Robert C. Martin em seu trabalho acadêmico publicado no ano de 2000 chamado: *Design Principles and Design Patterns* (BAELDUNG, 2019).

Os postulados que formam o acrônimo, representados na FIGURA 1, são: *Single Responsibility Principle, Open/Close Principle, Liskov Substitution Principle, Interface Segregation Principle* e *Dependency Inversion Principle.* Princípios que serão explicados a seguir.

FIGURA 1 - PRINCÍPIOS DO ACRÔNIMO SOLID



FONTE: (EDPRESSO, 2020)

Single Responsibility Principle: O princípio de responsabilidade única é resumido com uma frase: Uma classe deve ter uma única razão para ser alterada. Em outras palavras, uma classe deve ter uma única função no código. A razão para a separação das responsabilidades em entidades únicas, tem como objetivo deixar o código mais coeso (MARTIN; MARTIN, 2006).

Pois quando uma classe possui mais de uma responsabilidade, consequentemente ela vai possuir mais de uma razão para ser alterada. Esse tipo de *design* deve ser evitado, pois ele é frágil, porque leva a quebras inesperadas no projeto conforme as mudanças ocorrem (MARTIN; MARTIN, 2006).

Open/Close Principle: O princípio do Aberto/Fechado é definido como: Classes devem estar abertas para extensão, porém, fechadas para modificação. Em cenários quando uma alteração em uma parte do código, propaga como uma cascata de modificações subsequentes em outros locais do programa, significa que há um problema de *design* no projeto no qual esse postulado pode resolver (MARTIN; MARTIN, 2006).

Quando a solução Aberto/Fechado é aplicada de forma adequada ao projeto, através de abstrações e polimorfismo, alterações de software são realizadas através de adição de código e não de modificação dos dados já existentes e funcionais. Isso é possível, pois quando uma classe é aberta para extensão, significa que o seu comportamento pode ser estendido, com isso novas classes podem adicionar novos

comportamentos que podem satisfazer possíveis necessidades de alterações de projeto. Com isso, não sendo necessária a modificação da classe inicial, satisfazendo a regra de ser fechada para alteração (MARTIN; MARTIN, 2006).

Liskov Substitution Principle: O princípio de substituição de Liskov é estabelecido pelo seguinte conceito: subtipos de um objeto podem ser substituídos pelo seu tipo base. Em outros termos, é possível substituir uma classe filha por sua classe pai, e classes pais podem ser substituídas por qualquer outra classe que seja sua derivada, tendo como resultado o funcionamento correto da aplicação em ambas as situações. A definição parece simples, porém a sua importância fica bem clara quando esse princípio é violado (MARTIN; MARTIN, 2006).

A violação do princípio de Liskov, leva a reflexão do uso correto da herança. Nem sempre a sua utilização é simples e correta, devendo o programador ficar atento se realmente o uso da herança faz sentido na situação desejada. Com isso, sempre verificando se o contrato estabelecido com a classe pai não é quebrado na classe herdada, evitando assim erros inesperados durante a execução do programa. Tendo como alternativa o uso da composição entre classes para solucionar os possíveis problemas nesse contexto (ANICHE, 2020).

Interface Segregation Principle: O princípio da segregação de interfaces é o conceito de divisão de responsabilidade em diversas interfaces, evitando o acúmulo em uma única entidade. Esta regra é aplicada quando há interfaces não coesas no código, ou seja, quando há interfaces grandes e com mais de um motivo para existir no código do software (MARTIN; MARTIN, 2006).

O grande problema de grandes interfaces, é a possibilidade de ocorrer implementação de métodos sem utilidades na classe que cumprem o contrato pré-definido. Tendo por consequência a possibilidade de quebrar o código, quando esses métodos sem uso são tratados de forma errada, quebrando o contrato herdado e gerando erros de compilação em diversas outras partes do sistema. Problema explicado anteriormente com o princípio de substituição de Liskov (ANICHE, 2020).

Entretanto, ao aplicar a segregação das responsabilidades em outras interfaces, com a finalidade de distribuir as regras do contrato, as entidades filhas podem herdar quantas interfaces forem necessárias para suprir a sua necessidade de execução. Desta forma, as interfaces ficam mais coesas, tendo como resultado a

facilidade de reuso de código, aumentando estabilidade e impedindo a criação de improvisos paliativos no código (ANICHE, 2020).

Dependency-Inversion Principle: O princípio da inversão de dependência pode ser resumido em dois pontos (MARTIN; MARTIN, 2006):

- Módulos de alto nível não devem depender de módulos de baixo nível. Mas, ambos devem depender de abstrações;
- Abstrações não devem depender de implementações, e sim implementações devem depender de abstrações.

O grande fator desse princípio é a busca por estabilidade. Se uma classe x depende de outra classe y, a classe y deve ser mais estável que a classe x. Ou seja, as classes devem tender à estabilidade, dependendo sempre de módulos mais constantes do que ela (ANICHE, 2020).

Esse objetivo pode ser alcançado quando se leva em consideração que abstrações são mais estáveis que implementações. Com isso, se classe deve depender de um outro módulo, é ideal que esse módulo seja abstrato. Assim como, se um módulo abstrato depende de outro módulo, este também deve ser abstrato. Desta maneira, as dependências das classes são invertidas, agora dependendo de abstrações em vez de implementações e resultando em códigos com baixo acoplamento e com maior estabilidade (ANICHE, 2020).

Em suma, os padrões de projeto de desenvolvimento de software, quando aplicados de forma correta, tem um papel fundamental na resolução de problemas recorrentes na programação, resultando em códigos com design mais adaptável, mais coeso e menos acoplado (DEVMEDIA, 2020).

2.2 FUNDAMENTAÇÃO TEÓRICA DAS TECNOLOGIAS

Nessa seção, é apresentado conceitos relacionados as tecnologias utilizadas para o desenvolvimento do software.

2.2.1 Engenharia de software

O desenvolvimento de software nos anos entre 1950 e 1960 era uma atividade caótica. Os projetos nessa época, eram desenvolvidos sem padrões ou métodos para conduzir o andamento da codificação. Isso era resultado de uma filosofia seguida pelos profissionais, alguns com pouco conhecimento, ou com vícios de trabalho inadequados, que tinham como princípios (O'REGAN, 2017):

- O código finalizado sempre terá muitos erros;
- O software deve ser concluído o mais rápido possível, para que os erros esperados sejam corrigidos;
- Padrões são criados conforme a forma que o indivíduo codifica.

Por consequência disso, muitos projetos não obtinham sucesso, ou ficavam insustentáveis, devido aos altos índices de falhas. Fato que, por muitas vezes resultavam no cancelamento do projeto. Com esse cenário, ficou claro a urgência e a necessidade de uma mudança no desenvolvimento de software no mercado da época, sendo preciso criar padrões e métodos sólidos de programação, com objetivo de gerar produtos de alta qualidade. Com isso, nasceu a Engenharia de Software (O'REGAN, 2017).

A Engenharia de Software pode ser definida como um estudo de abordagens. Essas abordagens são referentes aos processos de desenvolvimento de software, sob uma perspectiva sistemática, quantificável e disciplinada. Conceitos que englobam processos, métodos de gerenciamento e ferramentas. Entretanto, esse conjunto de princípios tem como alicerce, e principal foco, a qualidade. Isso, em conjunto com uma filosofia de constante aperfeiçoamento das concepções gera, ao passar do tempo, conceitos cada vez mais eficazes (PRESSMAN; MAXIM, 2016).

Conforme ilustrado na FIGURA 2, a engenharia de software é uma tecnologia em camadas, organizadas sobre a máxima referente à qualidade.

FIGURA 2 - CAMADAS DA ENGENHARIA DE SOFTWAREE

FONTE: (PRESSMAN; MAXIM, 2016, p. 16.)

Já a camada referente ao processo (FIGURA 2) é a base principal da tecnologia. Com ela, é possível desenvolver uma aplicação coesa e de forma racional. Sendo os processos uma base para a engenharia, composta pelo controle e gerenciamento do projeto, desenvolvimento dos artefatos, e demais processos que garantem, sobretudo, a qualidade do código e reações coerentes em cenários de alterações de projeto (PRESSMAN; MAXIM, 2016).

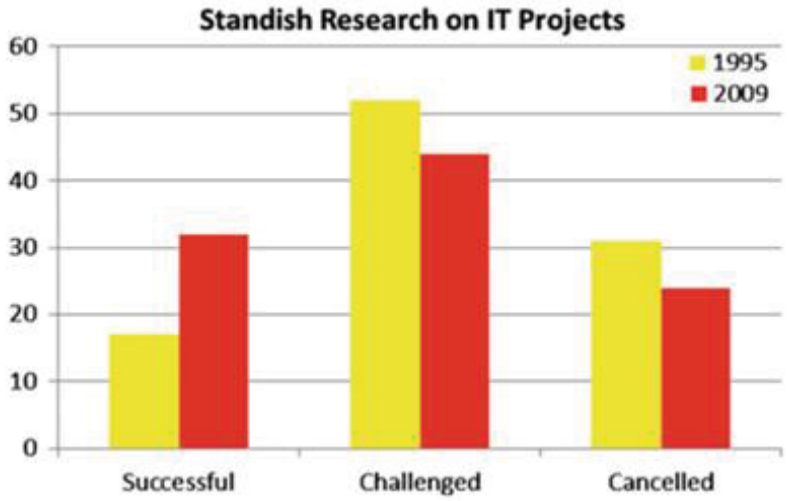
Observando a FIGURA 2, também se nota a camada referente aos métodos. Essa seção é composta pelos processos técnicos que auxiliam o processo de codificação. Estes métodos são compostos por comunicação, análise de requisitos, modelagem de projeto, testes, construção de programa, suporte e demais técnicas descritivas (PRESSMAN; MAXIM, 2016).

Por fim, a FIGURA 2 ilustra a última camada como ferramentas. Esse tópico fornece suporte a automatização completa ou parcial para as camadas de processos e métodos, nas quais quando integradas geram informações que podem ser usadas por outras ferramentas, com isso, é criado um sistema de suporte ao desenvolvimento (PRESSMAN; MAXIM, 2016).

A introdução desses conceitos no âmbito do desenvolvimento de software, foi primordial para a evolução dos programas modernos. De acordo com O'REGAN (2017) é evidente o crescimento de projetos de software bem-sucedidos no período

de 1995 a 2009, conforme ilustrado na FIGURA 3. Consequência das técnicas cada vez mais bem estabelecidas e consolidadas no mercado de tecnologia.

FIGURA 3 - IMPACTO DA ENGENHARIA DE SOFTWARE



FONTE: (O'REGAN, 2017, p. 3).

Desta forma, fica claro a importância da aplicação das técnicas de Engenharia de Software no processo de desenvolvimento de software, com o objetivo de melhorar a qualidade e reduzir o custo em programas cada vez mais focados no negócio empresarial (REZENDE, 2005).

2.2.2 Linguagem de modelagem unificada – UML

A linguagem de modelagem unificada, é uma linguagem de modelagem utilizada para arquitetar sistemas complexos de software. Com ela, é possível utilizar padrões preestabelecidos para a modelagem e especificação de artefatos necessários para a construção de um sistema. Além disso, essa linguagem também proporciona a preparação de modelos de arquitetura de software, assim como bases conceituais relacionadas as regras do negócio, funções do sistema, estrutura de classes, componentes e esquemas de banco de dados (BOOCH; RUMBAUGH; JACOBSON, 2005).

Segundo Wazlawick (2011), a linguagem de modelagem unificada pode ser classificada em três famílias de diagramas:

- Diagramas Estruturais: Essa classificação é composta pelos diagramas de classe, pacotes, objetos, estrutura composta, componentes e distribuição;
- Diagramas Comportamentais: Essa classificação é composta pelos diagramas de casos de uso, atividades e máquinas de estado;
- Diagramas de Interação: Essa classificação é composta pelos diagramas de comunicação, sequência, tempo e de visão geral de integração.

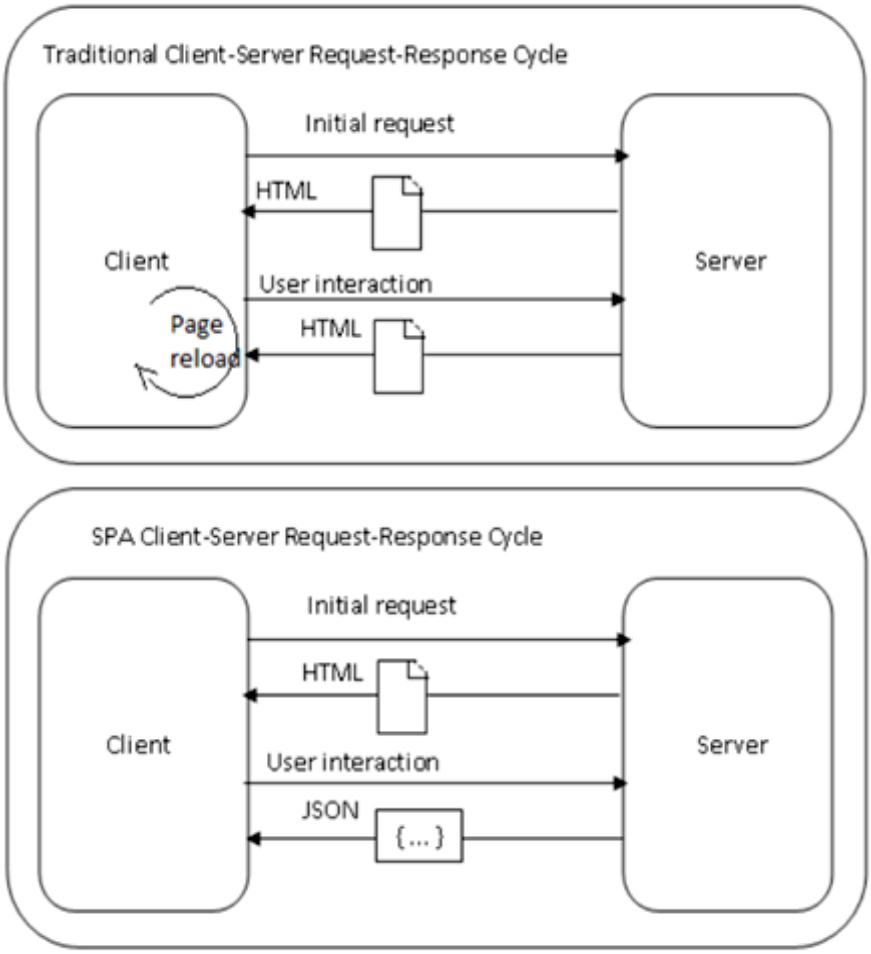
Contudo, apesar das vantagens que a UML pode prover ao desenvolvimento de software, é importante frisar que ela é somente uma notação de padrão de diagramação e não substitui a importância do conhecimento de projetar e pensar em termos de objetos (LARMAN, 2007).

2.2.3 Single Page Application

Uma das grandes tendências do mercado tecnológico atual são as aplicações de uma única página (do inglês *single page application* SPA). Esse tipo de aplicação proporciona uma maior fluidez para o usuário final, devido ao carregamento completo da aplicação ser realizado uma única vez, tendo as interações dinâmicas realizadas pelo usuário, processadas e renderizadas conforme a necessidade. Resultando em um aspecto próximo aos programas de *desktop* (FINK; FLATOW; GROUP, 2014).

Uma aplicação de página única pode ser definida como um software que é composto por um grupo de componentes individuais. Esses componentes podem ser substituídos ou atualizados de forma independente, sem a necessidade de recarregar ou atualizar a página web inteira a cada ação do usuário no sistema. Esse ciclo é ilustrado e comparado como uma aplicação tradicional na FIGURA 4 (JADHAV; SAWANT; DESHMUKH, 2015).

FIGURA 4 - SPA x APLICAÇÃO



FONTE: (JADHAV; SAWANT; DESHMUKH, 2015)

Aprofundando mais nas características desse modelo, é possível separá-lo em quatro grandes conceitos (JADHAV; SAWANT; DESHMUKH, 2015):

- *Componentes Individuais*: A única página HTML do software é dividida em vários componentes que interagem entre si;
- *Trocar/Atualizar*: Através das requisições do usuário, um componente, ou uma região ou alguma parte de uma página, podem ser trocados ou atualizados conforme as respostas das ações realizadas;
- *Atualizar/Recarregar*: A página nunca é recarregada inteiramente de uma só vez, somente suas pequenas partes nas quais ela é dividida;
- *Ações do usuário*: Uma aplicação de página única é muito fluída, devido a ela ser responsável por todas as ações que usuário produz na página.

Como cliques em botões, links, entradas de teclado, ações de navegação etc.

Esse tipo de aplicação permite lidar com os dados de forma mais elegante e flexível, resultando em interfaces mais interativas, o que consequentemente levam o software a ter uma melhor experiência para o usuário (JADHAV; SAWANT; DESHMUKH, 2015).

2.2.4 Scrum

Scrum é um *framework* de gerenciamento ágil de projetos, com ele é possível reduzir consideravelmente o tempo de execução de um projeto e o seu custo. Resultando em produtos com mais qualidade, tendo como resultado uma maior satisfação dos clientes (LAYTON; MORROW, 2019).

Essas vantagens são obtidas devido as características do *framework*, que proveem de uma mentalidade baseada na observação visual e adaptação baseada na experiência. Em conjunto com aspectos como maior visibilidade de desempenho, *feedback* regular dos clientes e das partes interessadas, modelos de comunicação, integração dos negócios e do desenvolvimento de habilidades (LAYTON; MORROW, 2019).

O *Scrum* foi criado devido à necessidade de mudanças no gerenciamento de projetos tradicionais na indústria de tecnologia dos anos 90. Nesse período, os projetos de software eram gerenciados com modelos do tipo cascata, resultando em um processo lento, imprevisível e por muitas vezes o produto resultante não era o que o cliente desejava, ocasionando frustrações e retrabalho (SUTHERLAND, 2016).

Além disso, problemas financeiros eram frequentes, devido aos constantes atrasos em relação ao cronograma. Desta maneira, o orçamento era terrivelmente afetado, gerando complicações desastrosas para as empresas (SUTHERLAND, 2016).

No entanto, apesar da popularidade do *framework* não significa que ele é a solução para todos os problemas dos projetos de desenvolvimento de software. O

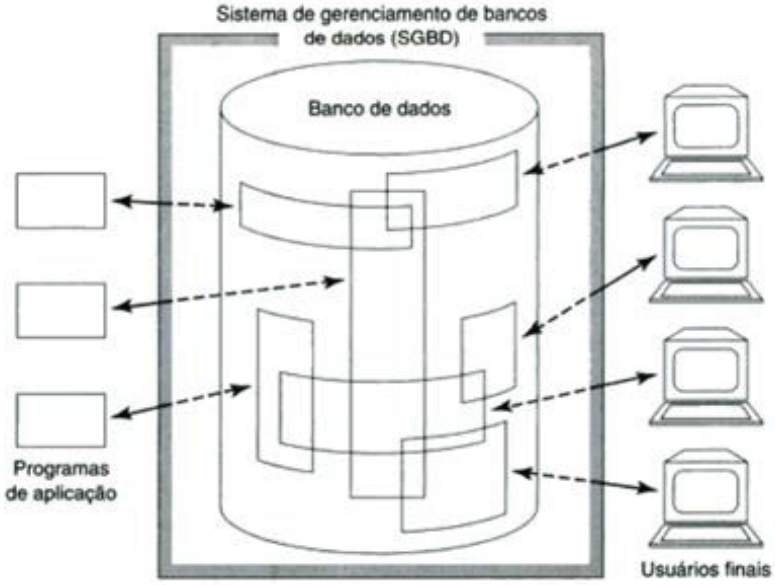
Scrum é apenas uma ferramenta que, para trazer os benefícios na condução dos projetos, deve ser utilizada corretamente em conjunto com outras técnicas e ter suas práticas testadas e adaptadas a realidade do trabalho a ser desenvolvido (SABBAGH, 2020).

2.2.5 Banco de dados

Um banco de dados é um sistema computadorizado de manutenção de registros. Em outras palavras, é um sistema que se comporta como um repositório de coleções de dados computadorizados, que podem ser manipulados conforme a necessidade do usuário. Esse manuseio de informações, pode ser através de ações como, inserção de dados novos, atualização de informações já existentes, exclusão e consultas de registros (DATE, 2004).

Entretanto, essa manipulação de registros geralmente é realizada com o auxílio de ferramentas especializadas, que são denominados sistemas de gerenciamento de banco de dados (SGBD). Conforme ilustrado na FIGURA 5, um SGBD é composto por 4 estruturas: dados, hardware, software e usuários. Desta forma, é possível armazenar e, principalmente, extrair informações provenientes das manipulações dos dados existentes (DATE, 2004).

FIGURA 5 - VISÃO GERAL DE UM SGBD



FONTE: (DATE, 2004, p. 6)

Essa distinção entre dados e informações é importante nesse contexto. Pois, apesar desses termos serem sinônimos, no ponto de vista prático são tratados de forma diferente. Sendo dados, os registros persistidos no banco de dados e informação sendo o significado que esses dados retornam de acordo com a análise realizada neles pelos usuários no SGBD (DATE, 2004).

Em resumo, os sistemas de bancos de dados desempenham um papel fundamental nas aplicações de software. Eles resolvem diversos problemas e resultando em diversas vantagens, como: baixa densidade para um grande volume de dados, velocidade de acesso as informações, eliminação de trabalhos manuais de manutenção de informações, dados precisos e segurança (DATE, 2004).

2.2.5.1 PostgreSQL

O PostgreSQL é um sistema de gerenciamento de banco de dados relacional, *open source* e desenvolvido há mais de 30 anos, proveniente de um projeto realizado na University *of California.* O gerenciador utiliza e estende a linguagem SQL (*Structured Query Language*), combinada com ferramentas que auxiliam no salvamento de dados complexos (POSTGRESQL, 2020).

O sistema PostgreSQL é extremamente confiável, principalmente devido a grande contribuição que a comunidade tem realizado nos últimos anos. Com isso, obtendo alta integridade de persistência de dados, reusabilidade, estendibilidade, um robusto conjunto de ferramentas e arquitetura sólida (POSTGRESQL, 2020).

A popularidade do SGBD atualmente é alta, há grandes empresas que utilizam o sistema e um grande apoio da comunidade opensource. Fatores que contribuem para a sua popularização e confirmam que softwares de código abertos podem concorrer de forma igualitária com programas proprietários (LEARNSQL, 2020).

2.2.6 Versionamento de código

A rotina dos profissionais no desenvolvimento de software é complicada, principalmente devido às pressões que o progresso nos projetos resulta no

cotidiano. Em consequência disso, as alterações referentes ao código devem ser realizadas com muito cuidado e de forma bem organizada. Pois, alterações mal feitas podem gerar problemas graves, como por exemplo, gerar *bugs* em código anteriormente funcionais (AQUILES; FERREIRA, 2018).

Além do mais, é comum que as alterações no código sejam feitas por equipes, agravando a possibilidade de ocorrer erros. Tornando as alterações em equipes desafiadoras, uma vez que as mudanças precisam ser sincronizadas de forma transparente entre os membros. Para solucionar esses problemas, foi criado o versionamento de código (AQUILES; FERREIRA, 2018).

As ferramentas de versionamento de código funcionam como um repositório, com o objetivo de armazenar todo o histórico das alterações e auxiliar na integração das modificações. Desse modo, possibilitando o acompanhando das modificações, tratando possíveis conflitos entre as alterações e permitindo o acesso a versões anteriores do código (AQUILES; FERREIRA, 2018).

Com isso, é nítido que desenvolver softwares sem utilizar sistemas de versionamento de código é correr riscos desnecessários, isto é, para que haja segurança, agilidade e eficiência de desenvolvimento e escalabilidade de equipes, essas ferramentas são imprescindíveis (ATLASSIAN, 2020).

2.2.7 Linguagem de programação Java

A linguagem de programação Java começou a ser desenvolvida nos anos noventa na *Sun Microsystems,* em um projeto chamado *Green Project.* Esse novo projeto tinha como objetivo desenvolver uma linguagem que integrasse computadores a equipamentos eletrônicos e eletrodomésticos. O resultado desse programa foi uma linguagem chamada *Oak* e um produto na forma de um controle remoto universal chamado *StarSeven*, que possuía uma interface interativa (LUCKOW; MELO, 2014).

Entretanto, o controle remoto estava muito a frente do seu tempo, e não conseguiu despertar o interesse no mercado. Com isso, a *Sun Microsystems* resolveu direcionar os esforços para a internet, e com o programador James Gosling

liderando o projeto, a linguagem Oak foi adaptada para a internet, assim nascendo o a plataforma Java em 1995 (LUCKOW; MELO, 2014).

A plataforma Java é composta pela linguagem de programação Java, em conjunto com sua *Java Virtual Machine* (JVM) e pela *Application Programming Interface* (API). Esse grupo de tecnologias possuem as seguintes características (MENDES, 2009):

- Uma linguagem simples: os criadores projetaram o Java para ser o mais simples possível, retirando da responsabilidade do programador o gerenciamento de operadores e ponteiros, assim como se preocupar com os detalhes de hardware, tendo em vista que a linguagem permite o desenvolvimento em variados sistemas operacionais;
- Linguagem orientada a objetos: o Java foi criado dentro do paradigma da orientação a objeto, dessa forma disponibilizando ao programador ferramentas como herança, polimorfismo e encapsulamento;
- Programação concorrente: nessa tecnologia é possível desenvolver softwares que implementam conceitos de *multithread,* sendo possível utilizar aplicações paralelas de forma eficiente;
- Linguagem Interpretada: após a compilação de um código Java é gerado um arquivo no formato *bytecode,* o qual pode ser executado em qualquer tipo de sistema operacional que tenha uma JVM instalada. A transcrição do *bytecode* para o código de máquina local é realizado e gerenciado totalmente pela máquina virtual (FIGURA 6);
- Portabilidade: a máquina virtual Java é a responsável por essa característica da tecnologia. Pois, estando ela disponível em diversos tipos de arquiteturas, os arquivos compilados Java podem ser portados para diferentes sistemas operacionais. Desta maneira sendo necessário manter apenas um único código Java (FIGURA 7);
- Alto desempenho: apesar de a tecnologia ser frequentemente comparada a linguagens compiladas, o Java mesmo sendo um código interpretado consegue ter um bom desempenho. Um dos elementos que ajudam no seu bom desempenho é o *garbage collector,* que nada mais é que uma

thread que roda em segundo plano que tem como objetivo ir liberando a memória que não está sendo usada pelo software;

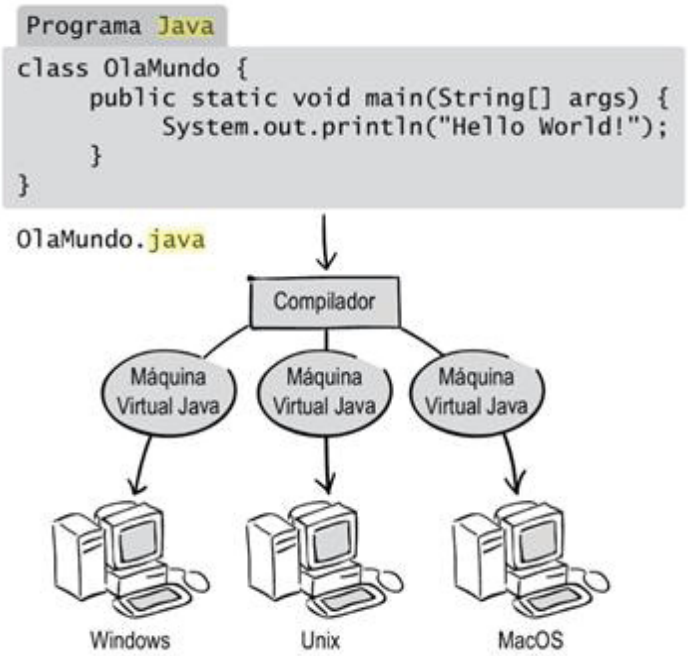
- Robustez: o Java foi projetado para gerar softwares e sistemas com altos índices de confiabilidade. Isso se deve, principalmente a capacidade da tecnologia detectar erros em tempo de compilação. Além disso, também possibilita a manipulação de exceções e é fortemente tipada;
- Segurança: como essa tecnologia foi criada para operar em ambientes distribuídos, a segurança foi muito bem implementada. A máquina virtual Java não permite que nenhum código escrito em outra linguagem seja executado, de forma sorrateira, em seu ambiente.

A linguagem Java é uma das responsáveis por reformular a essência da programação. Com o seu lançamento, ocorreu um rápido e grande impacto, revolucionando o meio tecnológico da época e definindo novos padrões de *design* para linguagens de computador (SCHILDT, 2015).

FIGURA 6 - COMPILAÇÃO DE CÓDIGO JAVA

FONTE: (MENDES, 2009, p. 21)

FIGURA 7 - EXECUÇÃO DE CÓDIGO JAVA



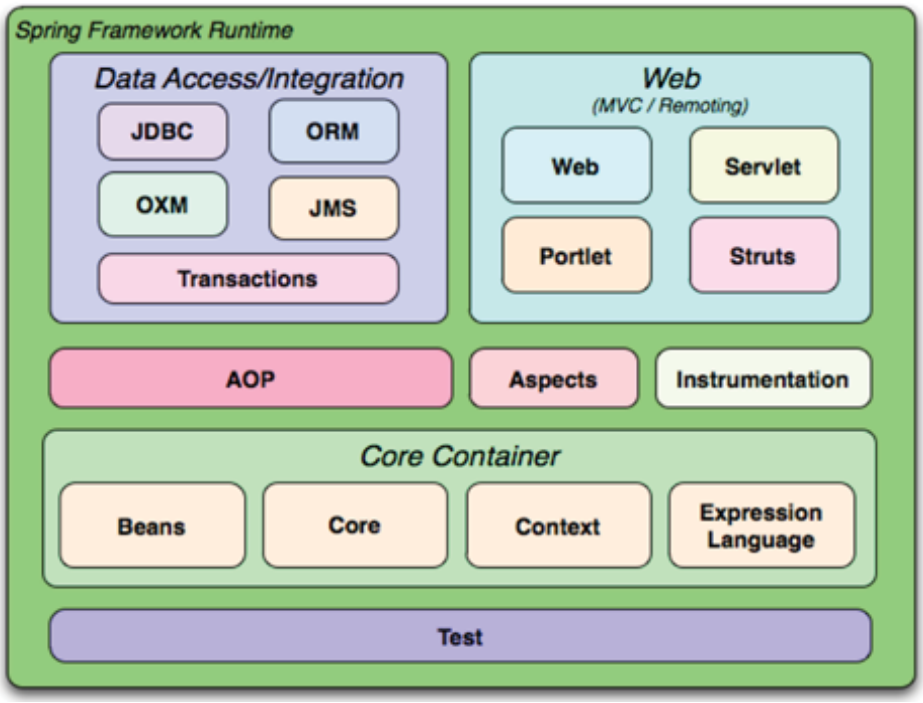
FONTE: (MENDES, 2009, p. 21)

2.2.8 Spring Framework

O *framework Spring* é uma plataforma que prove suporte para a infraestrutura de desenvolvimento em aplicações Java. Tendo disponível esse suporte, o desenvolvedor não precisa se preocupar com essas questões do projeto, podendo focar somente na criação lógica da aplicação (SPRING, 2020).

A plataforma *Spring* é uma solução leve e modular, dando a possibilidade de o desenvolvedor utilizar somente os módulos que são necessários para a implementação do seu projeto. O *framework* é composto por 20 módulos (FIGURA 8), que são divididos entre módulo principal, acesso de dados e integração, *web, AOP* (*Aspect Oriented Programming*), instrumentação e testes (SPRING, 2020).

FIGURA 8 - VISÃO GERAL DO SPRING FRAMEWORK



FONTE: (SPRING, 2020)

As principais características do *Spring* são que ele não é uma plataforma intrusiva, ou seja, seus módulos dificilmente se tornarão dependências da sua lógica de domínio. Além disso, o *framework* foi criado baseado nos padrões de projeto inversão de controle e injeção de dependência. Desta forma, é possível criar softwares robustos e de fácil de manutenção (SPRING, 2020).

Em resumo, a popularidade do *Spring* se deve pela demanda de desenvolvimento com qualidade de software em conjunto com produtividade, sendo para a linguagem Java uma ferramenta fundamental, e cada vez mais popular, para suprir esses objetivos (IGTI, 2017).

2.2.9 Spring Boot

O *Spring Boot* foi criado tendo como objetivo facilitar as configurações iniciais de um projeto utilizando a plataforma *Spring*. Tendo como principais objetivos (SPRING, 2020):

- Prover uma experiência de inicialização de projetos rápida e altamente acessível para iniciar projetos na plataforma *Spring*;

- Possui uma característica opinativa em relação a como configurar os projetos, entretanto é bem evasivo quando os requerimentos começam a sair dos seus padrões;
- Provê um conjunto de requisitos não funcionais já configurados e de uso comum para muitas classes do projeto, como servidor embarcado, segurança, métricas etc.;
- Remove qualquer necessidade de utilizar arquivos de configurações via XML (*Extensible Markup Language*) e não gera nenhum tipo de código automático.

Com isso, o maior benefício de utilização do *Spring Boot* é a liberdade que o desenvolvedor usufrui para pensar com maior foco e tempo na lógica da aplicação do que com suas configurações e publicações.

2.2.10 Angular Framework

O Angular é um *framework* desenvolvido pelo Google, que tem como objetivo prover suporte ao desenvolvimento de aplicações de página única, utilizando HTML (*Hypertext Markup Language*) e *Typescript*. A plataforma dispõe de um conjunto de bibliotecas, com as quais o desenvolvedor importa em sua aplicação conforme a sua necessidade (ANGULAR, 2020).

A arquitetura do angular é formada, de forma resumida, pelos itens representados na FIGURA 9, sendo seus conceitos básicos descritos a seguir (ANGULAR, 2020):

- Componentes definem *views*, que são um conjunto de elementos na tela que o Angular pode escolher entre elas e modificar de acordo com a lógica de dados;
- Componentes usam *services*, os quais proveem funcionalidades específicas não diretamente relacionadas as *views*. *Service providers* podem ser injetados em componentes como dependências, deixando o seu código modular, reusável e eficiente;

- Ambos *components* e *services* são classes simples, com *decorators* que marcam os seus tipos e providenciam *metadata*, que informa ao Angular como usá-los;
- A *metadata* para uma classe *component*, associa ele com um *template*, no qual define uma *view*. Um *template* combina qualquer HTML com as diretivas do Angular em conjunto com *binding markup*, o que permite ao Angular modificar o HTML antes de renderizar ele para visualização;
- A *metadata* para uma classe s*ervice*, providencia a informação que o Angular precisa para deixar o *service* disponível para os componentes através da Injeção de Dependência;

- As aplicações Angular são modulares, e o Angular tem o seu próprio sistema de modularidade, chamado NgModule**.** Esses módulos são *containers* para blocos de códigos coesos dedicados ao domínio de uma aplicação, *workflow* ou um conjunto de recursos intrinsecamente relacionados. Eles podem conter *components*, *services*, *providers* e outros arquivos de código nos quais o escopo é definido pelo NgModule que o contém. Eles podem importar funcionalidades que são exportadas de outros NgModules, e exportar funcionalidades para outros módulos usarem;

- Um *template* combina HTML com marcações Angular, que podem modificar elementos HTML antes deles serem apresentados em tela. *Template directives* proveem a lógica do programa e a marcação *binding* conecta seus dados da aplicação com o DOM (*Document Object Model*). Há dois tipos de *data binding*:

    - *Event Binding*: Permite o seu *app* responder a entradas do usuário no ambiente, atualizando os dados da aplicação;
    - *Property Binding:* Permite você interpolar valores que são computados dos dados de sua aplicação para o HTML;

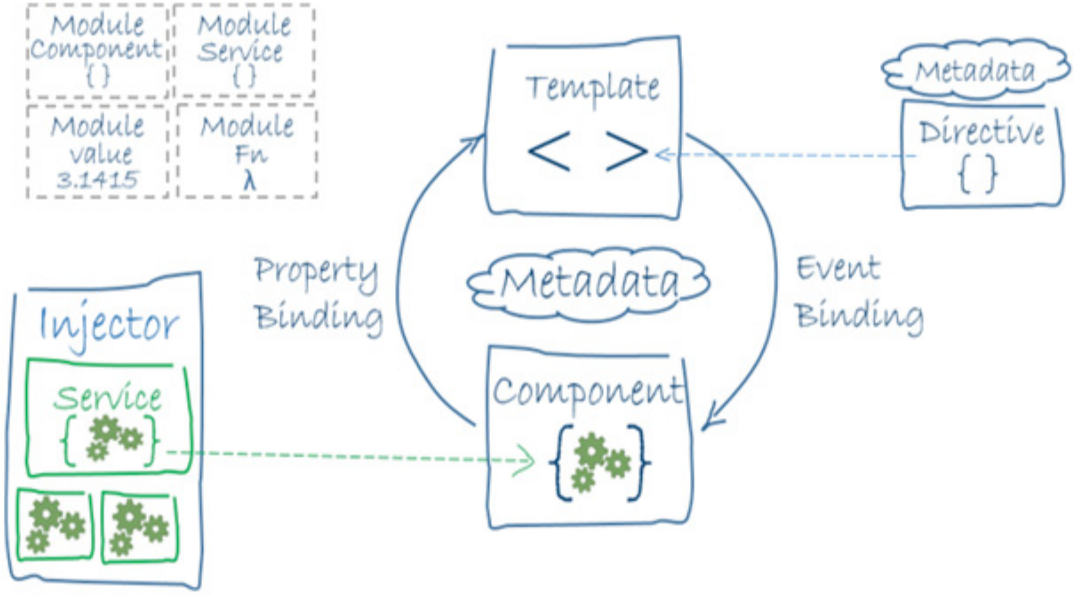
- Antes da *view* ser apresentada, o Angular checa as diretivas e resolve as sintaxes de *binding* no *template* para modificar o HTML e DOM, de acordo com a lógica do seu programa. O Angular suporta o *two-way-*

databinding, o que significa que as alterações no DOM, assim como as escolhas do usuário, refletem nos dados do seu programa;

- *Service* é uma ampla categoria que abrange qualquer valor, função ou *feature* que o *app* precise. Um *service* é tipicamente uma classe com propósito restrito e bem definido. Ele deve fazer algo específico e de forma eficiente;
- A Injeção de dependência está ligada ao *framework* Angular, e é usada por toda a parte para prover novos componentes com *services* ou outros dados que porventura sejam necessários. Componentes consomem *services*, isto é, você pode injetar um *service* em um componente, dando ao componente acesso a classe de serviço

Em síntese, o Angular é um *framework* prático para o desenvolvimento, possibilita uma organização nos seus arquivos de projeto, resultando em um código modular e fragmentado com estrutura simples, compreensível e de fácil manutenção (TOTVS, 2020).

FIGURA 9 - CONCEITOS BÁSICOS DO FRAMEWORK ANGULAR



FONTE: (ANGULAR, 2020)

2.2.10.1 *Typecript*

O *Typescript* é uma linguagem criada pela Microsoft, na qual é definida como um super conjunto do JavaScript. Nessa nova tecnologia foram adicionadas funcionalidades que eram difíceis de serem implementadas ou que não estavam disponíveis nativamente no *Javascript*. Como por exemplo, tipagem de dados e suporte a orientação a objeto (TYPESCRIPT, 2021).

Em virtude disso, aplicações modernas que utilizam JavaScript como tecnologia principal podem usufruir das vantagens do *Typescript*, e criar aplicações com arquitetura mais sólida e com um código mais organizado (DEVMEDIA, 2016).

Como também, os desenvolvedores podem aplicar melhores práticas de programação, uma vez que a sintaxe é mais simples e clara, além de aproveitar da possibilidade de aplicar padrões de projetos referente ao paradigma da orientação a objeto (DEVMEDIA, 2016).

2.2.11 Arquitetura REST

A arquitetura REST, acrônimo para Transferência de Estado Representacional (*Representational State Transfer*), foi definida no ano de 2000 por Roy Fielding em sua tese de doutorado (FIELDING, 2000).

Essa arquitetura foi concebida para ser uma abstração da camada web, tendo como objetivo reunir princípios e definições para criação de serviços web com interfaces bem estabelecidas (TOTVS, 2020).

O modelo REST possui as seguintes características (PERL MONGERS, 2010):

- Os recursos são definidos na requisição através de URIs (*Uniform Resource Identifier*);
- Ações podem ser realizadas nos recursos através dos 8 métodos disponíveis do protocolo HTTP;
- O tipo de conteúdo da negociação é identificado através do cabeçalho da requisição;

- As requisições não possuem estado, ou seja, elas devem conter toda a informação para serem interpretadas pelo servidor.

Com essas propriedades presentes na arquitetura REST, é possível construir aplicações escalonáveis e confiáveis, usufruindo do benefício de não ser necessário persistir estados e possibilitando a criação de elementos como cache e camadas hierárquicas nas aplicações, resultando em maior desempenho, eficiência e escalabilidade nos sistemas (PERL MONGERS, 2010).

3 MATERIAIS E MÉTODOS

Esse capítulo apresenta os procedimentos realizados para o desenvolvimento e gerenciamento do projeto do software, em conjunto com as tecnologias, cronograma de atividades e testes.

3.1 METODOLOGIA SCRUM

A metodologia Scrum foi escolhida para gerenciar o desenvolvimento do projeto. Dessa maneira, as atividades relacionadas a execução do projeto foram divididas em 21 *Sprints* com duração de uma semana. Cada ciclo de interação foi dividido em modelagem ou desenvolvimento de código, finalizando com uma revisão do conteúdo e checagem dos testes unitários criados.

Além disso, ao final de uma *Sprint*, foi realizada uma análise completa do que foi produzido. Reajustando e reorganizando as etapas subsequentes de acordo com atrasos recorrentes, provenientes de dificuldades técnicas ou de tempo insuficiente. Essa ação sempre foi realizada analisando os riscos presentes no gerenciamento do projeto.

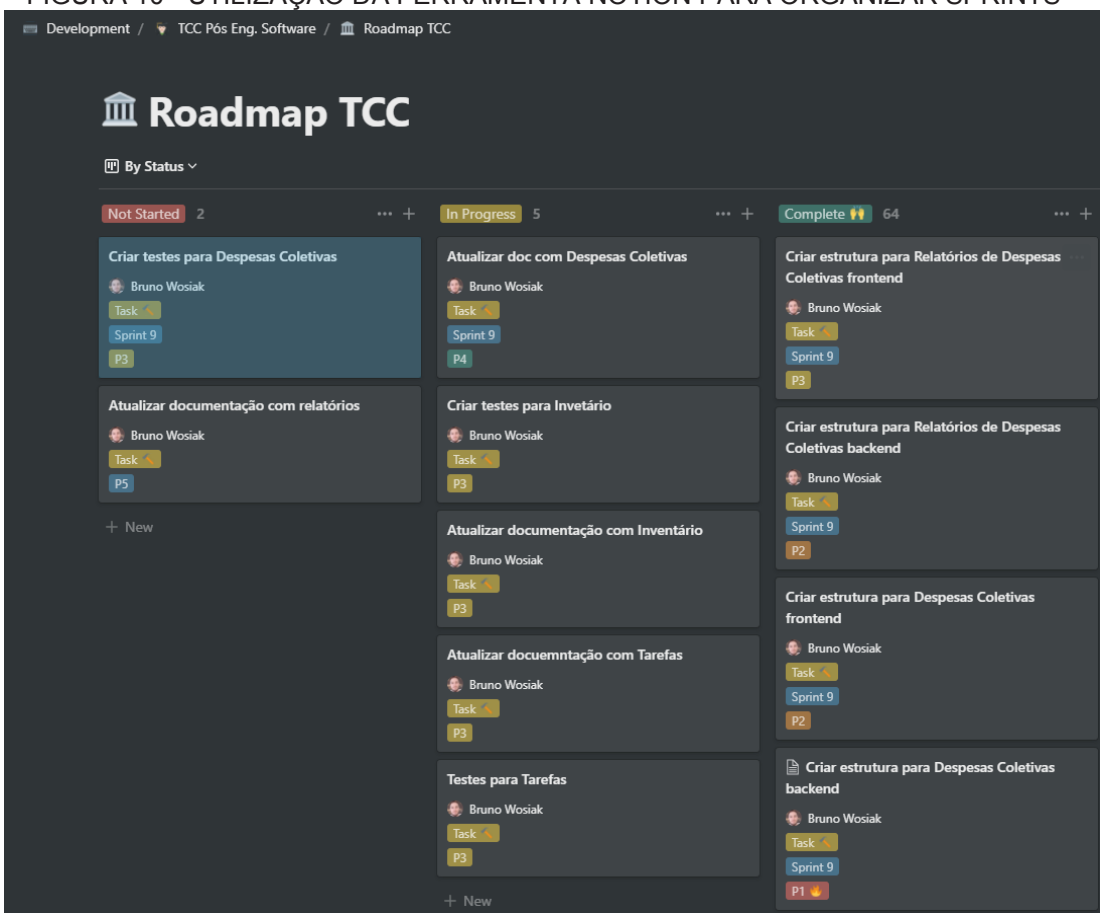
Também foram combinadas reuniões semanais com o orientador, via ferramenta de videoconferência. Esses encontros, tinham como objetivo ajustes, validações e acompanhamento do andamento do projeto. Adicionalmente, foram utilizadas diversas ferramentas em conjunto com o *framework* Scrum para auxiliar o desenvolvimento do software, que serão apresentadas a seguir.

3.1.1 Notion

Para o gerenciamento das atividades relacionadas as *Sprints* do *framework* Scrum foi escolhido a ferramenta Notion. Essa ferramenta possibilita a criação de cartões de atividades que podem ser alterados, movidos e analisados. Com isso, dando uma visão macro de todo o progresso do projeto, em relação ao que está concluído, sendo executado e o que está pendente (NOTION, 2021).

Além disso, a aplicação permite gerenciar as atividades com uma visão de banco de dados, possibilitando fazer filtros e pesquisas complexas, assim como organizar toda a documentação do que está sendo desenvolvido (NOTION, 2021).

FIGURA 10 - UTILIZAÇÃO DA FERRAMENTA NOTION PARA ORGANIZAR SPRINTS



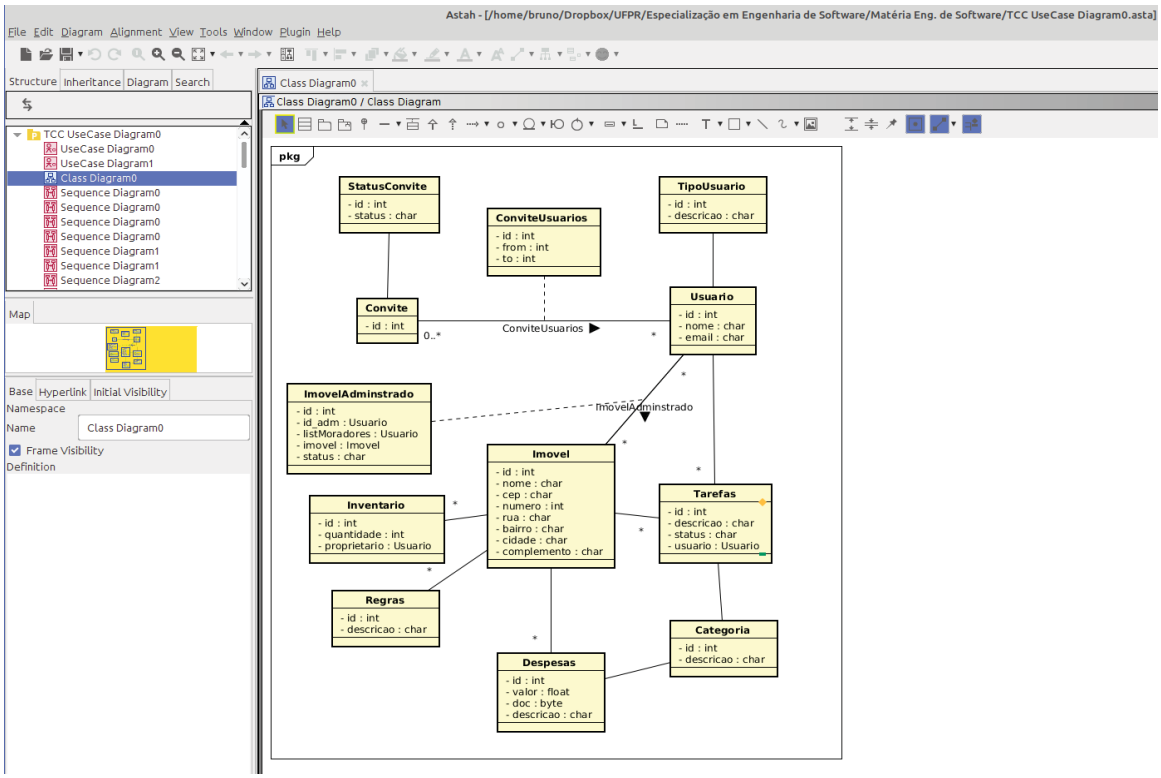
FONTE: O Autor (2020)

3.1.2 Astah

Astah foi a ferramenta selecionada para confeccionar os diagramas UML do projeto. Com esta ferramenta é possível criar diagramas como (ASTAH, 2020):

A FIGURA 11 ilustra a utilização da ferramenta Astah na modelagem do projeto:

FIGURA 11 - FERRAMENTA ASTAH PARA MODELAGENS UML



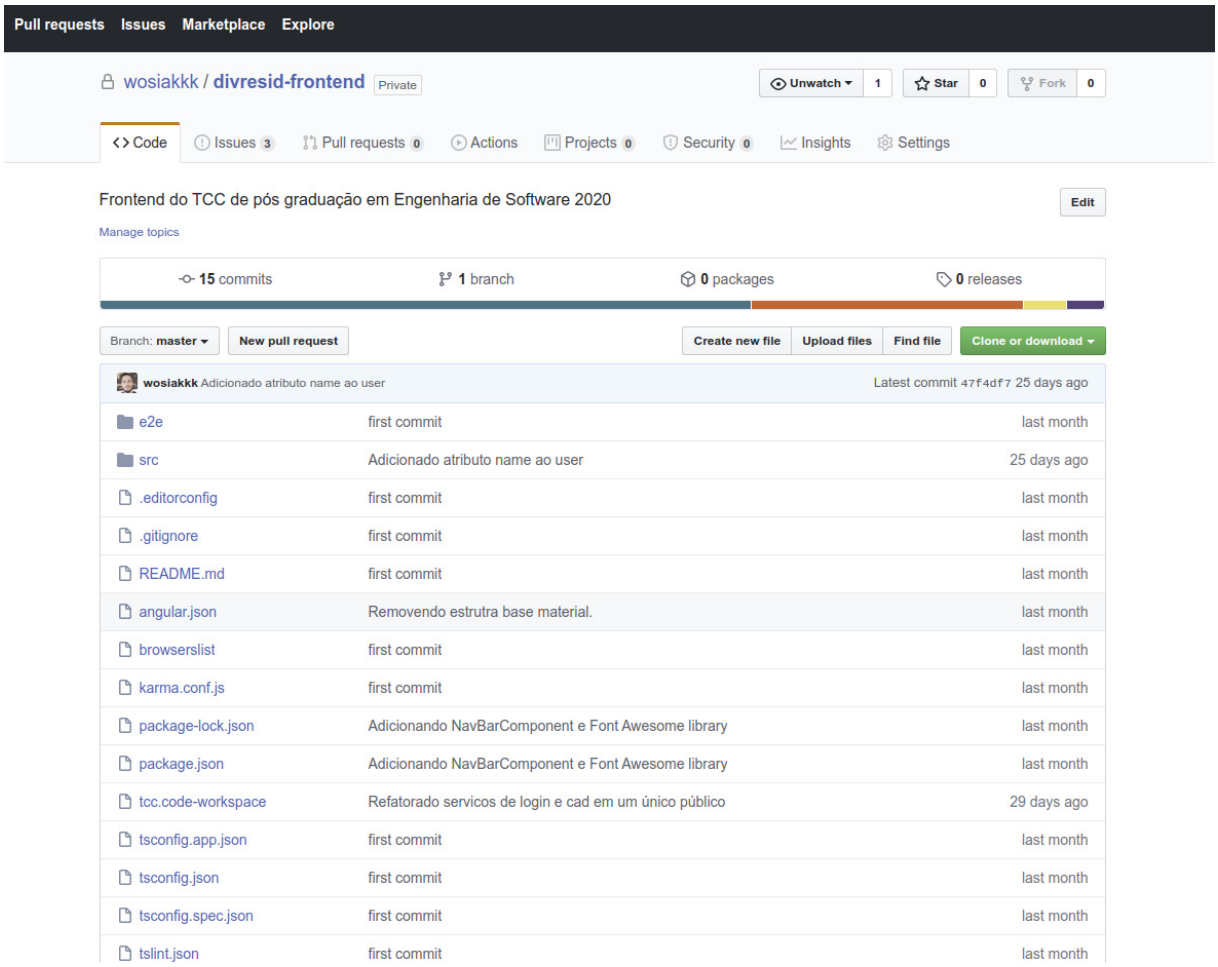
FONTE: O Autor (2020)

3.1.3 Github

Para o versionamento do código do software, o Github foi escolhido como ferramenta principal. O Github é uma rede social colaborativa para desenvolvedores que gerencia e armazena as versões de código fonte, dando a possibilidade de automatizar ações relativas ao versionamento de código (GITHUB, 2020).

A FIGURA 12, ilustra o repositório do código do projeto hospedado no GitHub.

FIGURA 12 - FERRAMENTA GITHUB PARA CONTROLE DE VERSÃO DE CÓDIGO



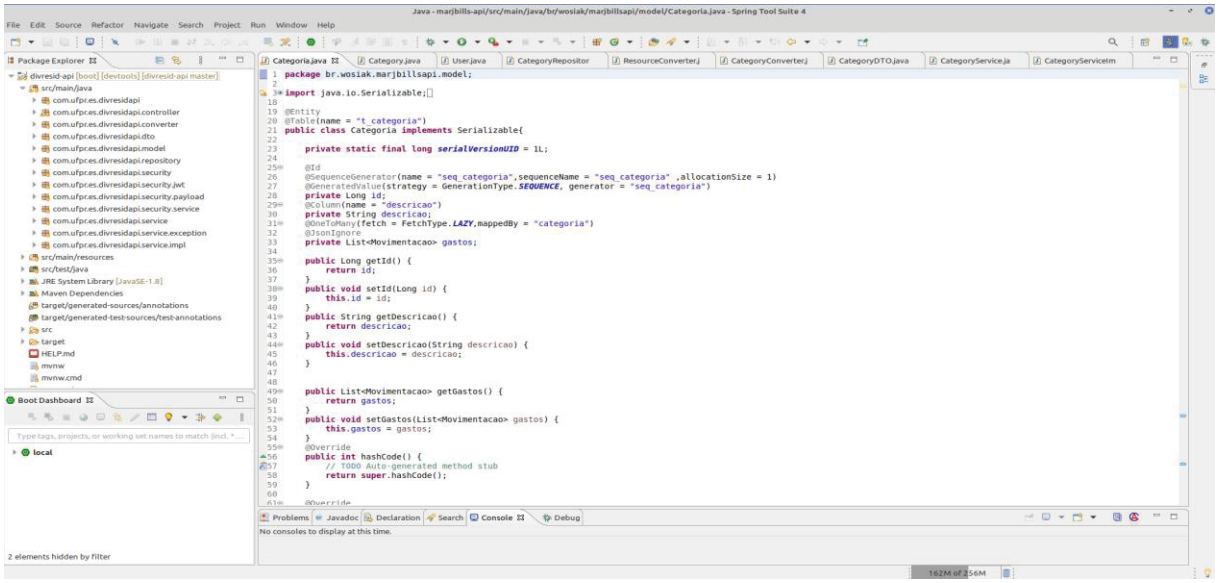
FONTE: Autor (2020)

3.1.4 Spring Tool Suite 4

A IDE (*Integrated Development Environment*) Spring Tool Suite 4, baseada no Eclipse, foi utilizada como ferramenta para realizar a programação do *backend* Java do projeto. Além disso, essa ferramenta possibilitou o acesso rápido e simples as tecnologias relacionadas ao *framework* Spring (SPRING, 2020).

A FIGURA 13 demonstra a utilização da ferramenta no código *backend* do projeto.

FIGURA 13 - IDE SPRING TOOL SUITE 4 PARA DESENVOLVIMENTO JAVA

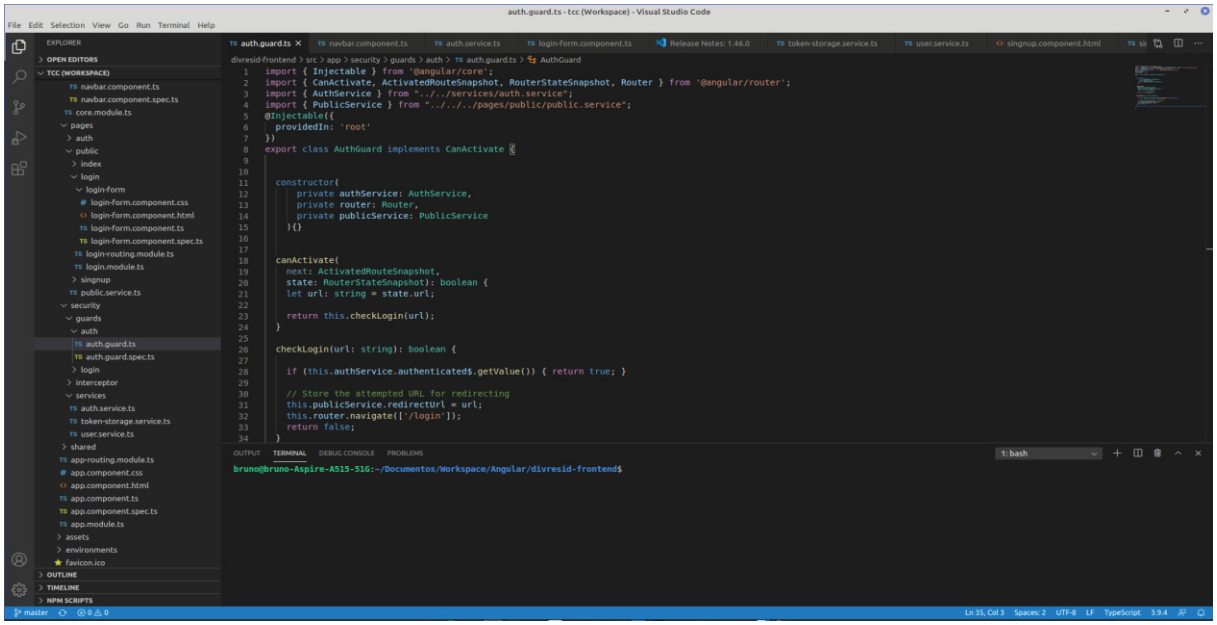


FONTE: O Autor (2020)

3.1.5 Visual Studio Code

A IDE Visual Studio Code foi utilizada para o desenvolvimento do código TypeScript para o *frontend* Angular do projeto, conforme representado na FIGURA 14 (CODE, 2020).

FIGURA 14 - VISUAL STUDIO CODE PARA DESENVOLVIMENTO DO FRONTEND



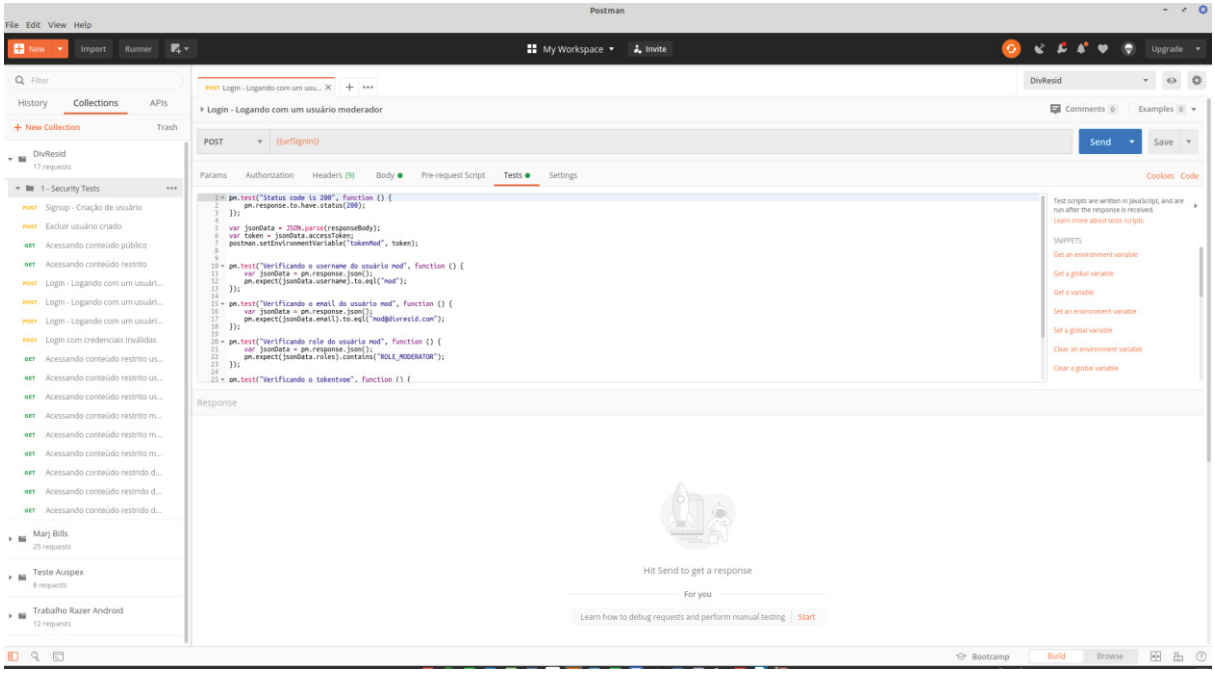
FONTE: O Autor (2020)

3.1.6 Postman

A ferramenta Postman foi utilizada para realizar testes da API do *backend* Java do projeto, testando todos os *endpoints* disponíveis da aplicação. Além de criar a documentação da sua utilização (POSTMAN, 2020).

A figura 15 apresenta a utilização da ferramenta.

FIGURA 15 - FERRAMENTA POSTMAN PARA TESTES EM API



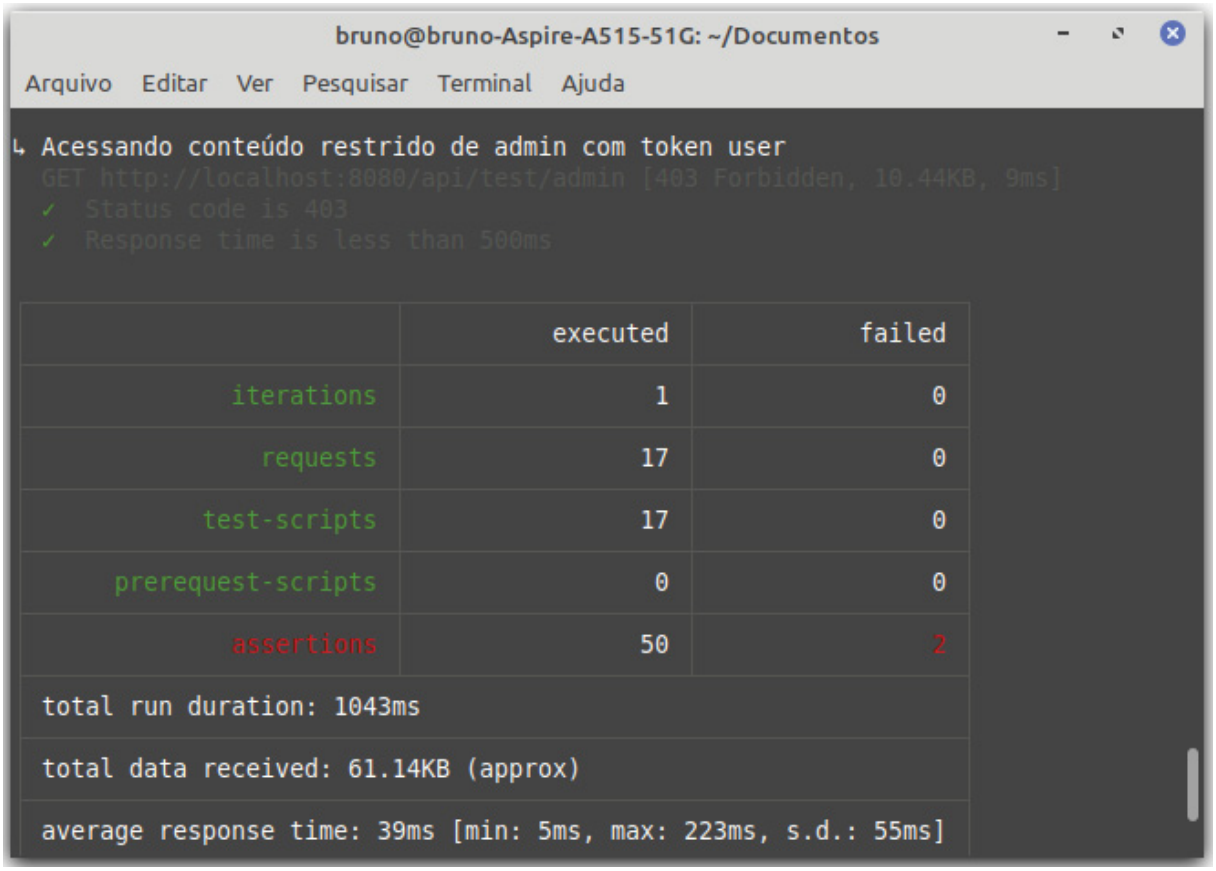
FONTE: O Autor (2020)

3.1.7 Newman

A ferramenta Newman foi utilizada para automatizar e facilitar os testes dos *endpoints* do *backend* realizados pelo Postman. Com ela, é possível executar os testes criados de forma sequencial e simples através de linha de comando (POSTMAN, 2020).

A FIGURA 16 ilustra a utilização do Newman, e o resultado de diagnóstico gerado pela ferramenta a cada fim de ciclo de teste.

FIGURA 16 - UTILIZAÇÃO DA FERRAMENTA NEWMAN PARA TESTES

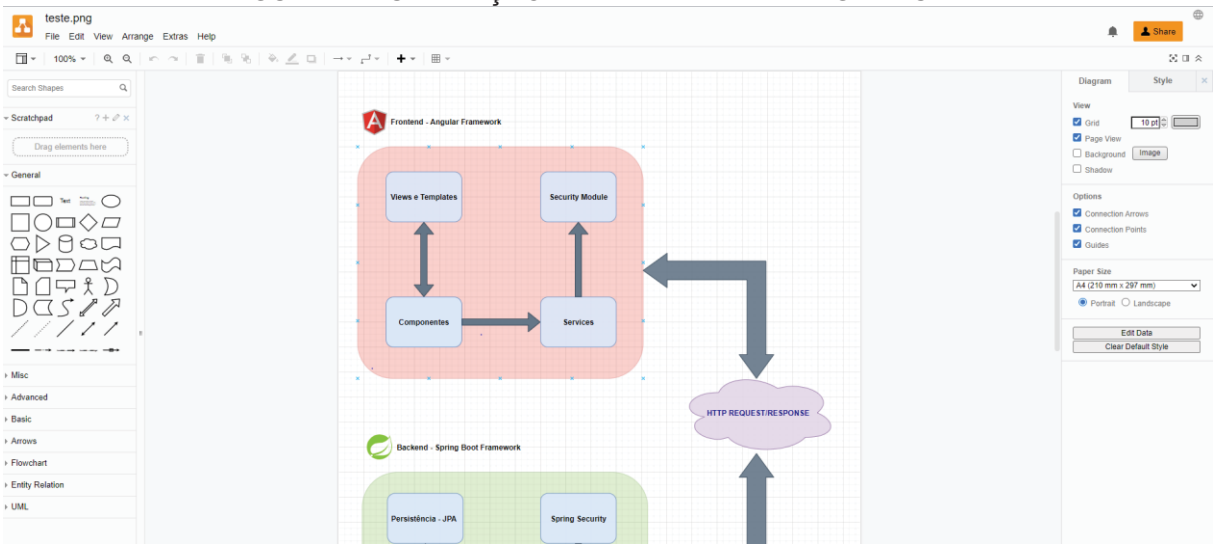


FONTE: O Autor (2020)

3.1.8 Diagrams.net

A ferramenta *web* Diagrams.net foi utilizada para o desenvolvimento de diagramas que ilustram a arquitetura do sistema *backend* e *frontend.* O Diagrams.net é uma aplicação *open source* executada no navegador que possibilita a criação de diversos tipos de diagramas (DIAGRAMS, 2021).

FIGURA 17 - UTILIZAÇÃO DA FERRAMENTA DIAGRAMS.NET



FONTE: O Autor (2021)

3.2 INFRAESTRUTURA DE DESENVOLVIMENTO

Para o desenvolvimento do projeto, foi utilizado um notebook com a seguinte especificação técnica:

- Nome da máquina: bruno-Aspire-A515-51G;
- Proprietário: Bruno Felipe Ribeiro Wosiak;
- Fabricante: Acer;
- Sistema Operacional: Linux Mint 19.3 Tricia;
- Memória RAM: 12GB;
- Processador: Intel Core i7-8550U 1.8GHz;
- Placa de Vídeo: NVIDIA GeForce MX130 2GB;
- Armazenamento: SSD 256GB.

3.3 PLANO DE RISCO

O gerenciamento de risco é uma etapa do gerenciamento de projeto que define os riscos que podem ocorrer durando a execução da proposta. Desta maneira, foi definido as ações que devem ser tomadas em caso de imprevistos, levando em consideração o seu impacto no projeto versus a sua probabilidade de acontecer.

A QUADRO 1, representa o plano de ação elaborado para identificar, quantificar, responder e controlar os riscos encontrados.

QUADRO 1 - QUADRO DE GERENCIAMENTO DE RISCOS DO PROJETO

Número	Descrição	Ação	Probabilidade	Impacto
1	Deficiência técnica do programador em relação a tecnologia	Realizar estudos durante períodos de folga	Baixa	Grave
2	Falta de tempo suficiente para a conclusão do projeto	Reorganização das atividades, alterando a prioridade para os pontos mais críticos	Média	Grave
3	Alteração de requisitos no projeto	Verificar a necessidade da alteração no projeto; Caso a alteração ocorra, priorizar as atividades dependentes da alteração	Média	Moderado
4	Esforço estimado de forma incorreta para determinada atividade	Priorizar as atividades que foram prejudicadas em relação ao seu tempo de execução; Reorganizar as prioridades das atividades subsequentes	Média	Moderado
5	Requisitos imprecisos	Realizar revisões dos requisitos imprecisos e replanejar o cronograma;	Baixa	Leve
6	Cronograma não realista	Identificar as atividades que possam ser realizadas em paralelo	Alta	Grave

FONTE: O Autor (2020)

3.4 RESPONSABILIDADES

Responsabilidades associadas ao integrante do desenvolvimento do projeto. Conforme mostra a QUADRO 2.

QUADRO 2 - QUADRO DE RESPONSABILIDADES

Número	Integrante	Responsabilidades
1	Bruno Felipe Ribeiro Wosiak	Desenvolvimento do *backend*;

		Desenvolvimento do *frontend*; Testes de *software*; Modelagem de banco de dados; Diagramas UML; Documentação do projeto;

FONTE: O Autor (2020)

3.5 CRONOGRAMA DE ATIVIDADES

Na fase de concepção e planejamento do projeto, foi estabelecido o seguinte cronograma de atividades, divididos em *sprints*, conforme QUADRO 3.

QUADRO 3 - CRONOGRAMA DE ATIVIDADES

Sprint nº	**Data de início**	**Atividades**
1	01/06/2020	Definição da ideia do projeto; Definição dos requisitos; Definição dos casos de uso; Definição da tecnologia utilizada; Definição para layout e arquitetura base para o software; Prototipação inicial do software.
2	08/06/2020	Criação dos diagramas UML; Modelagem do banco de dados para login e cadastro; Criação de estrutura base abstrata rest na api; Revisão de Sprint.
3	15/06/2020	Definição da tecnologia de segurança de autenticação; Implementação da tecnologia de segurança escolhida; Modelagem para a segurança em banco de dados; Implementação de login e cadastro; Implementação de testes; Especificação de casos de uso para login e cadastro; Revisão de Sprint.
4	22/06/2020	Modelagem do banco de dados para categorias; Implementação do crud de categorias; Implementação de testes; Especificação de caso de uso para categorias; Revisão de Sprint.
5	29/06/2020	Criação de lógica para despesas pessoais; Modelagem de banco de dados para despesas pessoais; Implementação de crud para despesas pessoais; Implementação de testes; Especificação de caso de uso para despesas pessoais; Revisão de Sprint.
6	06/07/2020	Modelagem de banco de dados para imóvel; Implementação do crud de imóvel; Implementação de testes; Especificação de caso de uso para imóveis; Revisão de Sprint.
7	13/07/2020	Criação de lógica para convidar moradores; Modelagem de banco de dados para moradores; Modelagem de banco de dados para lógica de convite; Implementação de crud de moradores;

		Implementação de uso de convites por moradores; Implementação de testes; Especificação de caso de uso de moradores e convite; Revisão de Sprint.
8	20/07/2020	Criação de lógica para tarefas; Modelagem de banco de dados para tarefas; Implementação de crud para tarefas; Implementação de testes; Especificação de caso de uso para tarefas; Revisão de Sprint.
9	27/07/2020	Criação de lógica de despesas coletivas; Criação de estrutura para upload de arquivos; Modelagem de banco de dados para despesas coletivas; Implementação de crud para despesas coletivas; Implementação de testes; Especificação de caso de uso para despesas coletivas; Revisão de Sprint.
10	03/08/2020	Modelagem de banco de dados para regras; Implementação de crud regras; Implementação de testes; Especificação de caso de uso para regras; Revisão de Sprint.
11	17/08/2020	Modelagem para banco de dados para inventário; Implementação de crud para inventário; Implementação de testes; Especificação de caso de uso para inventário; Revisão de Sprint.
12	07/09/2020	Criação de lógica para tela com gráfico de lançamentos; Implementação tela com gráfico; Especificação de caso de uso para gráfico; Revisão de Sprint.
13	28/09/2020	Criação de lógica para carregamento preguiçoso; Implementação de carregamento preguiçoso; Revisão de Sprint.
14	12/10/2020	Criação de lógica para exibição da tela inicial não autenticada; Implementação de tela inicial não autenticada; Revisão de Sprint.
15	26/10/2020	Modelagem de banco de dados para metas; Implementação de crud metas; Implementação de testes; Especificação de caso de uso para metas; Revisão de Sprint.
16	09/11/2020	Criação de lógica para exibição da tela inicial autenticada; Implementação de tela inicial autenticada; Revisão de Sprint.
17	23/11/2020	Criação de lógica para exibição de perfil de imóvel ativo; Implementação de tela de perfil de imóvel ativo; Revisão de Sprint.
18	07/12/2020	Criação de lógica para exibição de perfil de usuário; Implementação de tela de perfil de usuário; Revisão de Sprint.
19	21/12/2020	Criação de lógica para troca de senha de usuário; Implementação de tela de troca de senha; Revisão de Sprint.

20	04/01/2021	Testes no sistema e correção de bugs.
21	18/01/2021	Testes no sistema e correção de bugs.
22	01/02/2021	Testes no sistema e correção de bugs.
23	15/02/2021	Conclusão do Capítulo 1 e 2 da documentação.
24	01/03/2021	Conclusão do Capítulo 3 da documentação; Revisão do Capítulo 1 e 2 da documentação.
25	15/03/2021	Conclusão do Capítulo 4 e 5 da documentação; Revisão do Capítulo 3 da documentação.
26	29/03/2021	Revisão da documentação por completa; Realização da previa da apresentação.
27	31/05/2021	Entrega final da documentação e do software.
28	08/06/2021	Defesa do Trabalho de Conclusão do Curso.

FONTE: O Autor (2020)

3.5.1 *Sprint* 1

Nessa primeira *sprint* foi definida a ideia geral de como seria o sistema. Foi analisado todos os dados pesquisados anteriormente, que foram utilizados para a confecção dos textos da introdução, objetivos e justificativas. Com isso, foi solidificado como seria o escopo e as funcionalidades do *software* do projeto.

Além disso, foram levantados os requisitos que o *software* deve abranger para atingir os objetivos propostos, desta maneira foi criado também a primeira versão do diagrama de caso de uso. A versão final do diagrama de caso de uso pode ser vista no Apêndice A.

Também nessa primeira iteração, houve a definição final das tecnologias que foram utilizadas no desenvolvimento do projeto, assim como a arquitetura do sistema, que foi escolhida para ser uma solução desacoplada entre um *frontend* e *backend* independentes. Bem como seu *layout* inicial, e padrões de *design* a serem seguidos através uma prototipação base temporária.

Detalhes da arquitetura final implementada do sistema podem ser vistas no Apêndice C.

3.5.2 *Sprint* 2

Nessa s*print* foram confeccionados os diagramas UML parciais e a modelagem inicial do banco de dados para suporte inicial de autenticação e cadastro de usuários. Com isso, já consolidando a base inicial para criar o primeiro cenário funcional para implementar o primeiro CRUD.

Em adição, também foi iniciado a programação da estrutura abstrata da *API* resultando em uma base sólida de arquitetura para o *backend,* provendo contratos e regras predefinidas para as futuras implementações das regras de negócio do sistema interagirem com os métodos *HTTP*.

Além do mais, também foi planejado a estrutura para a execução de testes nos *endpoints* do *backend,* sendo assim possível testar as funcionalidades adicionadas no sistema.

Detalhes da arquitetura final implementada do sistema podem ser vistas no Apêndice C, assim como a estratégia de testes disponíveis no Apêndice E.

3.5.3 *Sprint* 3

Nessa *sprint*, iniciou-se a implementação da tela de *login* e cadastro de usuários do sistema em conjunto com a modelagem de banco de dados para essas funcionalidades. Além disso, também foi realizada uma pesquisa para definição da estratégia para a implementação da segurança no sistema.

Por conseguinte, a tecnologia de autenticação por *token* JWT (*JSON Web Tokens*) foi escolhida para a implementação no sistema (JWT, 2020). E por fim, foi também implementado *scripts* de testes para assegurar as operações de CRUD e login de usuários na *API*, assim como as suas especificações de casos de uso, disponíveis no Apêndice B.

Também é possível observar com detalhes a implementação da segurança do sistema no Apêndice D.

A versão final da tela de criação, edição e exclusão de usuários e de *login* do sistema, podem ser vistas no Capítulo 4 onde o sistema é apresentado por completo.

3.5.4 *Sprint* 4

Nessa *sprint,* foi modelado e implementado a entidade categoria para o sistema. Essa classe tem como finalidade classificar outros objetos no sistema. Além da modelagem de banco de dados e de classe, também foi desenvolvido a sua tela para operações de cadastro e listagem.

Além disso, também foi implementado o *script* de testes para essas operações na API assim como a especificação desse caso de uso, disponível no Apêndice B.

A versão final da tela de criação, edição e exclusão de categorias, podem ser vistas no Capítulo 4 onde o sistema é apresentado por completo.

3.5.5 *Sprint* 5

Nessa *sprint*, foi planejado a lógica para controle de despesas pessoais dos usuários. Chegou-se à conclusão que seria controlado valores de entrada e saída, os quais seriam classificados com categorias e que também possuem estados de pagamento, com o qual cálculos de saldo são obtidos.

Com isso, foi modelado e implementado no sistema a entidade lançamento, em conjunto com seu *script* de testes, tela com listagem de dados, filtro de buscas e operações de cadastro, assim como a especificação de caso de uso, disponível no Apêndice B.

A versão final da tela de lançamentos, podem ser vistas no Capítulo 4 onde o sistema é apresentado por completo.

3.5.6 *Sprint* 6

Nessa *sprint* foi modelado e implementado a entidade imóvel para o sistema. Essa classe é o primeiro objeto que faz parte da lógica, ainda a ser planejada, que associará moradores ao sistema administrativo do usuário responsável.

Também foi implementado o *script* de testes para essas operações relacionados ao cadastro de imóveis na *API* assim como a especificação desse caso de uso, disponível no Apêndice B.

A versão final da tela de criação, edição e exclusão de categorias, podem ser vistas no Capítulo 4 onde o sistema é apresentado por completo.

3.5.7 *Sprint* 7

Nessa *sprint* foi modelado e implementado a estrutura para a realização de convite para usuários serem relacionados como moradores de determinado imóvel do administrador.

Além disso, também foi modelado e implementado a lógica para permissões, criando níveis de acesso para Morador e Moderador, que tem como papel restringir ações no sistema.

Também foi implementado o *script* de testes para essas operações relacionados aos convites e níveis de acesso no sistema na *API*, assim como, a especificação desse caso de uso que está disponível no Apêndice B.

A versão final da tela de criação, edição e exclusão de convites e de alteração de níveis de acesso, podem ser vistas no Capítulo 4 onde o sistema é apresentado por completo.

3.5.8 *Sprint* 8

Nessa *sprint* foi modelado e implementado a estrutura para a realização de cadastros de tarefas e atividades para os moradores do imóvel. Assim como,

implementado a lógica de permissões de acesso para a realização desse procedimento.

Também foi implementado o *script* de testes para essas operações relacionadas as atividades e tarefas do imóvel, em conjunto com a especificação desse caso de uso que está disponível no Apêndice B.

A versão final da tela de criação, edição, exclusão e finalização de atividades e tarefas, podem ser vistas no Capítulo 4 onde o sistema é apresentado por completo.

3.5.9 *Sprint* 9

Nessa *sprint* foi modelado e implementado a estrutura para a realização de cadastros de despesas coletivas. Para essa funcionalidade, foi criado também uma lógica de rotina para que a partir de uma despesa coletiva, sejam criadas despesas individuais, resultando em uma divisão igualitária.

Além disso, foi criado o *script* de testes para essas operações de lógica e criação de despesas coletivas, assim como a confecção da especificação desse caso de uso, a qual está disponível no Apêndice B.

A versão final da tela de criação, edição e exclusão de despesas coletivas, em conjunto com a geração de despesas individuais, podem ser vistas no Capítulo 4 onde o sistema é apresentado por completo.

3.5.10 *Sprint* 10

Nessa *sprint* foi modelado e implementado a estrutura para a realização de cadastro de regras do imóvel ativo. O cadastro das regras utiliza um componente completo de postagem de textos, possibilitando o administrador cadastrar textos com formatações, links e imagens.

Além do mais, foi criado o *script* de testes para essas operações de lógica e criação de regras do imóvel, assim como a confecção da especificação desse caso de uso, a qual está disponível no Apêndice B.

A versão final da tela de criação, edição e exclusão de regras do imóvel podem ser vistas no Capítulo 4 onde o sistema é apresentado por completo.

3.5.11 *Sprint* 11

Nessa *sprint* foi modelado e implementado a estrutura para cadastro do inventário do imóvel. Nesse cadastro é possível controlar os itens do imóvel e os seus respectivos donos.

Também foi criado o *script* de testes para as operações de lógica e criação de itens do inventário do imóvel ativo.

A versão final da tela de criação, edição e exclusão de inventário do imóvel podem ser vistas no Capítulo 4 onde o sistema é apresentado por completo.

3.5.12 *Sprint* 12

Nessa *sprint* foi realizado a implementação da estrutura lógica para a exibição de um gráfico para contextualizar os lançamentos salvos do usuário. As despesas são ilustradas conforme um período entre datas e categorizadas entre despesas e entradas separadas por categorias.

A versão final da tela de exibição do gráfico de lançamentos do usuário pode ser vista no Capítulo 4 onde o sistema é apresentado por completo.

3.5.13 *Sprint* 13

Nessa *sprint* foi realizado a implementação do carregamento preguiçoso (*lazy loading*) das listagens das entidades do sistema. Essa programação tem como objetivo reduzir o custo de banda ao carregar os dados do *backend,* resultando em um software mais ágil.

Detalhes dessa implementação podem ser vistas no capítulo 4 onde é descrito a arquitetura do sistema.

3.5.14 *Sprint* 14

Nessa *sprint* foi realizado a implementação da página inicial do sistema para usuários não autenticados. Esta página tem como objetivo introduzir as funcionalidades do sistema para novos usuários.

A versão final da tela inicial do sistema pode ser vista no Capítulo 4 onde o sistema é apresentado por completo.

3.5.15 *Sprint* 15

Nessa *sprint* foi realizado a implementação da estrutura e lógica para a criação de metas financeiras para os usuários. Com isso, sendo possível os usuários informarem um valor a ser atingido em determinado tempo, levando em consideração os lançamentos de despesa e entrada.

Além do mais, foi criado o *script* de testes para essas operações de lógica e criação de regras de metas financeiras, assim como a confecção da especificação desse caso de uso, a qual está disponível no Apêndice B.

A versão final da tela de criação, edição e exclusão de metas financeiras podem ser vistas no Capítulo 4 onde o sistema é apresentado por completo.

3.5.16 *Sprint* 16

Nessa *sprint* foi realizado a implementação da estrutura e lógica da página inicial do sistema após a autenticação do usuário. Nessa tela, é carregado o resumo de informações referente às finanças pessoais, metas financeiras e do imóvel ativo no momento.

Também nessa tela, é carregado um calendário que apresenta as atividades e tarefas pendentes dos moradores no mês atual.

A versão final da tela inicial do usuário autenticado pode ser vista no Capítulo 4 onde o sistema é apresentado por completo.

3.5.17 *Sprint* 17

Nessa *sprint* foi implementado a opção de visualização do imóvel ativo no momento. Com isso, os usuários podem visualizar as informações de forma rápida no menu lateral do sistema.

A versão final da tela de perfil de imóvel pode ser vista no Capítulo 4 onde o sistema é apresentado por completo.

3.5.18 *Sprint* 18

Nessa *sprint* foi implementado a opção de visualização do perfil do usuário autenticado. Com isso, o usuário pode visualizar as suas informações cadastradas de forma rápida pelo menu lateral.

A versão final da tela de perfil de usuário pode ser vista no Capítulo 4 onde o sistema é apresentado por completo.

3.5.19 *Sprint* 19

Nessa *sprint* foi implementado a estrutura para dar suporte a troca de senha para o usuário autenticado. Com isso, o usuário pode realizar a manutenção do seu acesso ao sistema.

Também foi implementado um *script* de teste dessa operação de atualização de senha do usuário.

3.5.20 *Sprint* 20

Nessa *sprint* foi realizado testes e corrigidos erros na exibição de atividades e tarefas, perfil e de imóvel quando o usuário não possui nenhum imóvel ativo no momento.

3.5.21 *Sprint* 21

Nessa *sprint* foi realizado ajustes no *layout* da tela inicial após a autenticação do usuário, melhorando o visual e corrigindo o idioma do calendário para português brasileiro.

3.5.22 *Sprint* 22

Nessa *sprint* foi realizado testes gerais no sistema, tendo como objetivo corrigir diversos erros relacionados a exibição do conteúdo de HTML e CSS. Além de melhorias visuais aplicadas para deixar o *layout* como um todo mais agradável.

3.5.23 *Sprint* 23

Nessa *sprint* os Capítulos 1 e 2 foram criados e finalizados, com isso a documentação foi enviada para análise do orientador.

3.5.24 *Sprint* 24

Nessa *sprint* o Capítulo 3 foi criado e finalizado. Também foi corrigido as observações apontadas pelo orientador referente as Capítulos 1 e 2. Um novo envio para o orientador foi realizado após os ajustes.

3.5.25 *Sprint* 25

Nessa *sprint* o Capítulo 4 e 5 foram criados e finalizados. Também foi corrigido as observações apontadas pelo orientador referente ao Capítulo 3. UM novo envio para o orientador foi realizado após os ajustes

3.5.26 *Sprint* 26

Nessa *sprint* a documentação foi revisada por completa de acordo com o retorno do orientador. Também foi configurado e revisado o ambiente de execução do *software* para a defesa do trabalho. Com isso, uma prévia da apresentação foi realizada para ser exibida ao orientador.

3.5.27 *Sprint* 27

Nessa *sprint* a versão final da documentação e do *software* foram entregues ao orientador.

3.5.28 *Sprint* 28

Nessa última *sprint* ocorreu a defesa do Trabalho de Conclusão de Curso com a banca julgadora.

4 APRESENTAÇÃO DO SPLITTY HOUSE

Nesse Capítulo é apresentado a arquitetura completa do sistema em conjunto com todas as funcionalidades do software.

4.1 ARQUITETURA DO SISTEMA

O sistema Splitty House foi desenvolvido utilizando duas tecnologias distintas com o objetivo de deixar o *frontend* e *backend* em projetos separados e independentes. A visão geral dos projetos e como eles interagem entre si pode ser observada na FIGURA 18.

Desse modo, no projeto do *frontend* foi utilizado o *framework* Angular com a finalidade de manipular o HTML, CSS e dados, assim como realizar requisições para a API, conforme as iterações do usuário com as telas.

Já no projeto do *backend* foi utilizado a linguagem de programação Java com o suporte dos módulos do Spring Boot, tendo como finalidade responder as requisições REST feitas por clientes.

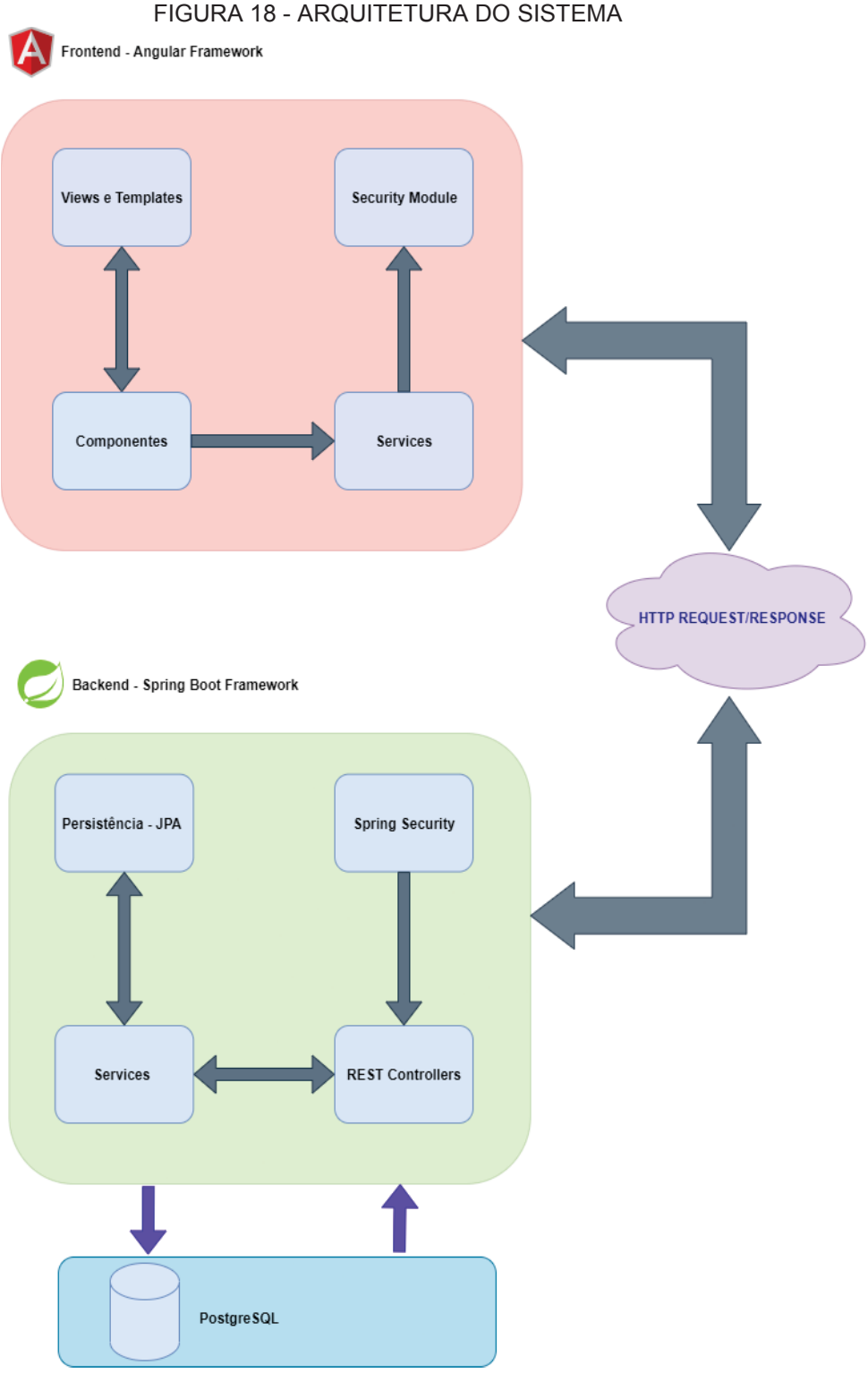
Um fluxo completo de operação de um dado no *frontend* seria iniciado através de ações pelo usuário em uma determinada informação contida em um *template* HTML*,* resultando no envio para o componente específico da tela, no qual contém a lógica de negócio. Após o processamento lógico o dado é enviado ao *servisse* para a realização do envio do elemento via HTTP.

Por outro lado, um fluxo completo no projeto do *backend* é iniciado pela captura da requisição HTTP pelo REST *Controller.* Assim sendo o dado filtrado e redirecionado para o serviço correspondente do protocolo e caminho informando na requisição. Em virtude disso, o *service* processa a informação de acordo com a lógica estabelecida e envia o resultado para a camada de persistência. E por fim, a requisição é respondida para o cliente com o resultado da ação.

Salienta-se que em ambos os projetos todas as requisições geradas são interceptadas por módulos de segurança antes do seu envio à rede. Esta ação tem como propósito verificar a autenticidade da chamada. Todo esse processo é ilustrado na Figura 18.

Diante disso, cada projeto também possui uma arquitetura interna bem estruturada através de abstrações, interfaces e implementações. Detalhes desses modelos de arquitetura podem ser visto no Apêndice C.

FIGURA 18 - ARQUITETURA DO SISTEMA

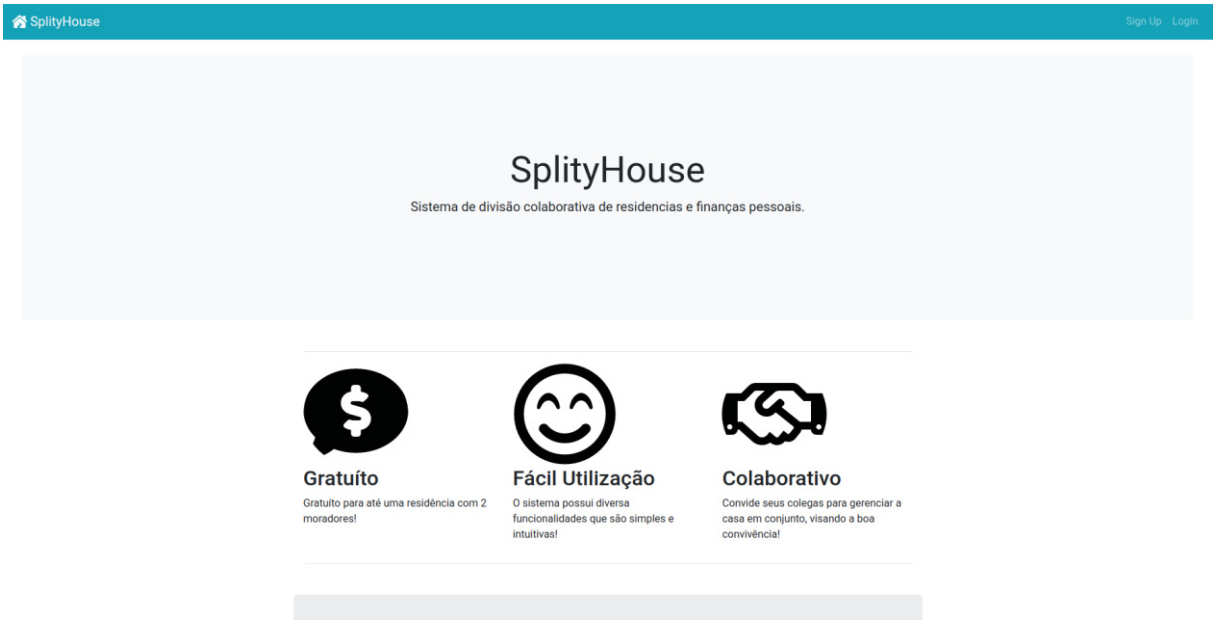


FONTE: Autor (2021)

4.2 PRIMEIRO ACESSO

Ao acessar o sistema pela primeira vez, é apresentado para o usuário a tela inicial, conforme a FIGURA 19.

FIGURA 19 - PRIMEIRO ACESSO



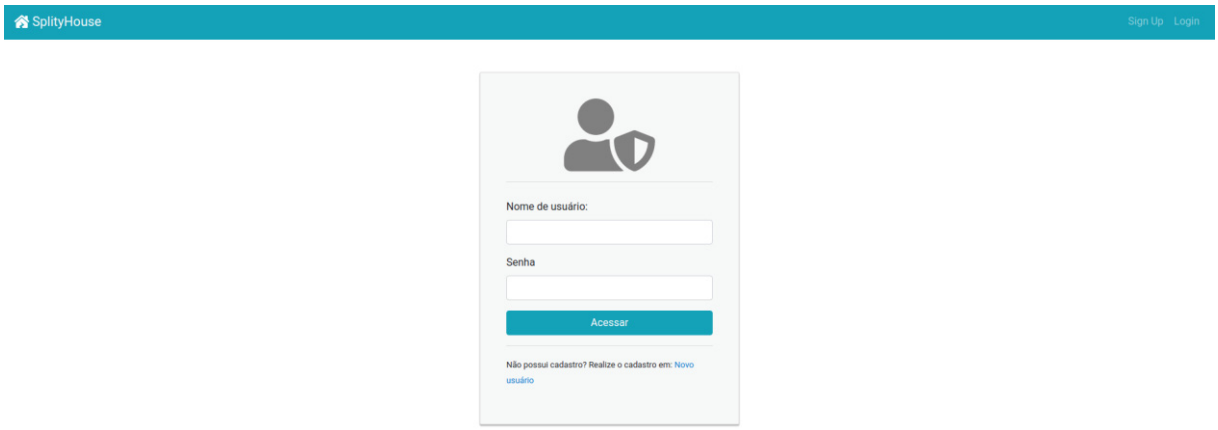
FONTE: Autor (2021)

Esta tela possibilita o usuário a escolher se cadastrar ou efetuar o login, caso já tenha um cadastro efetuado previamente.

4.2.1 Realizar Login

Após o usuário acessar pela primeira vez o sistema, a opção de efetuar o login estará disponível conforme FIGURA 20.

FIGURA 20 - LOGIN



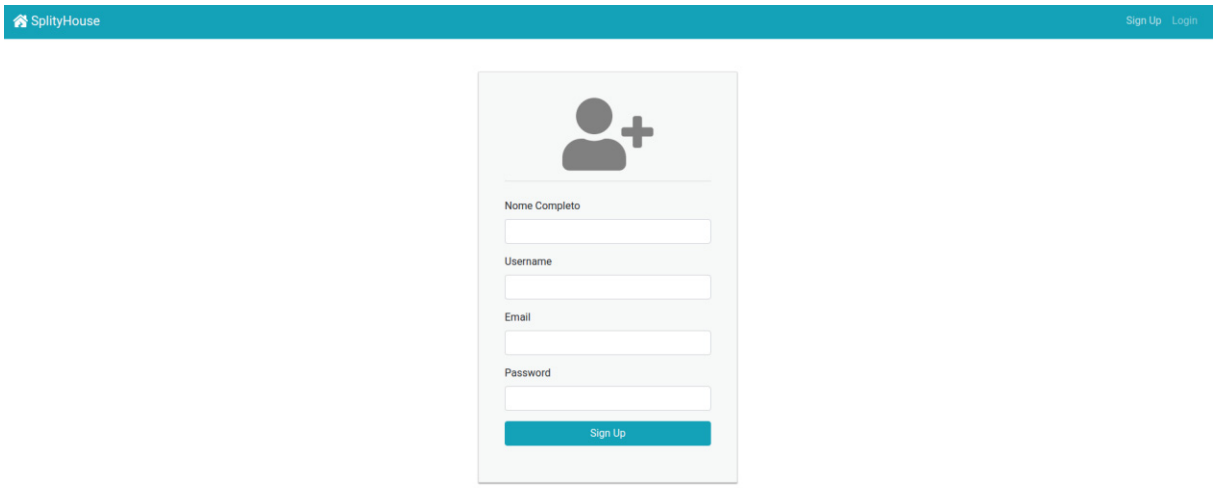
FONTE: O Autor (2021)

O usuário deve preencher os campos “Nome de usuário” e “Senha” para efetuar o login no sistema. Caso o usuário não possua uma conta previamente cadastrada no SplityHouse, o link de cadastro pode ser acessado através do clique em “Novo usuário”.

4.2.2 Cadastro

Para ter acesso as funcionalidades do sistema, o usuário deve possuir um cadastro. E para realizar o cadastro, o usuário deve preencher os campos ilustrados na Figura 21.

FIGURA 21 - CADASTRO



FONTE: O Autor (2021)

4.3 USUÁRIO ADMINISTRADOR

Ao efetuar o login pela primeira vez, todo usuário é considerado um administrador até que este status seja modificado com a ação de se tornar um morador.

FIGURA 22 - Tela Inicial

FONTE: O Autor (2021)

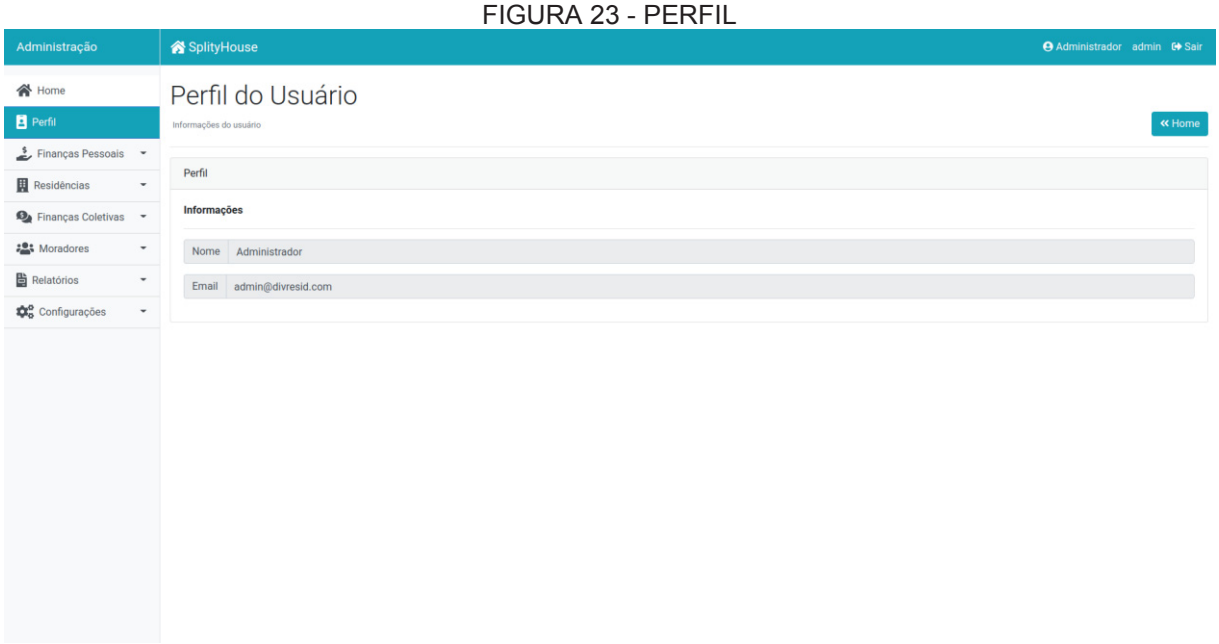
A tela inicial (Figura 22) exibe informações resumidas do usuário autenticado, como residência, moradores atuais, resumo financeiro, tarefas agendadas e metas, quando há dados cadastrados e imóvel ativo no momento.

Como responsável, o usuário possui as seguintes ações no sistema: controle de finanças pessoais e coletivas, controle de cadastro de imóveis e moradores. Funcionalidades estas que serão descritas em seguida.

4.3.1 Perfil

O usuário pode acessar os dados da sua conta clicando na opção “Perfil” no menu lateral. Esta tela apresenta apenas dados simples do usuário, conforme Figura 23.

FIGURA 23 - PERFIL



FONTE: O Autor (2021)

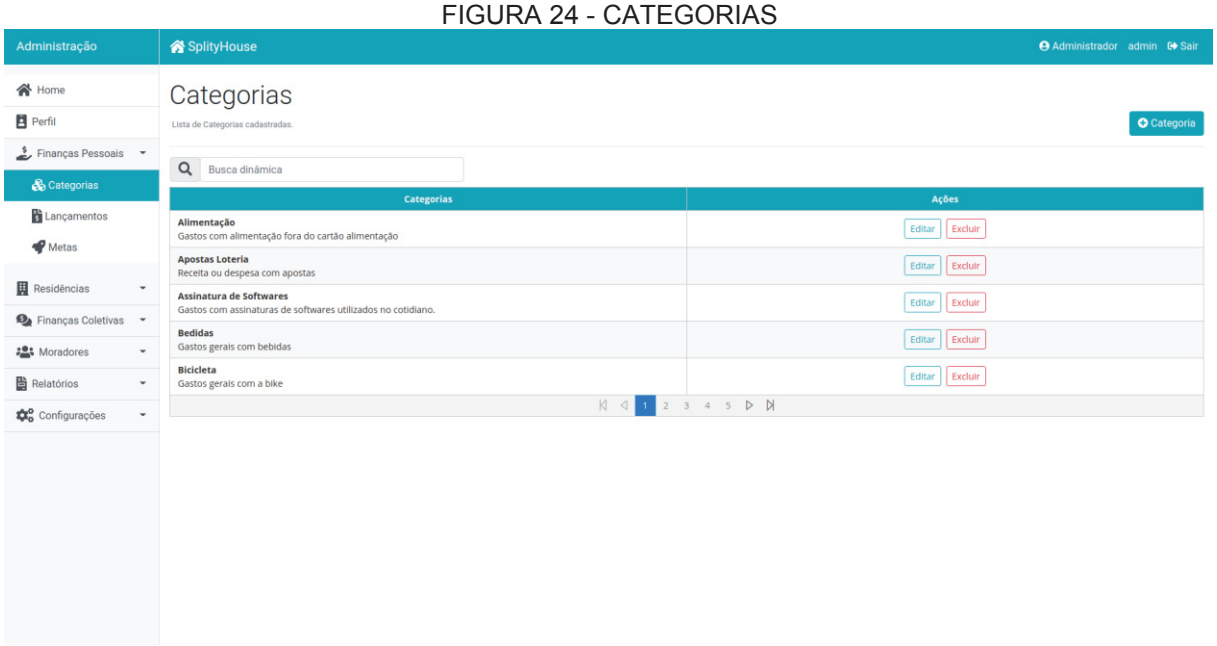
4.4 FINANÇAS PESSOAIS

As opções relacionadas as finanças pessoais do usuário serão demonstradas a seguir.

4.4.1 Categorias

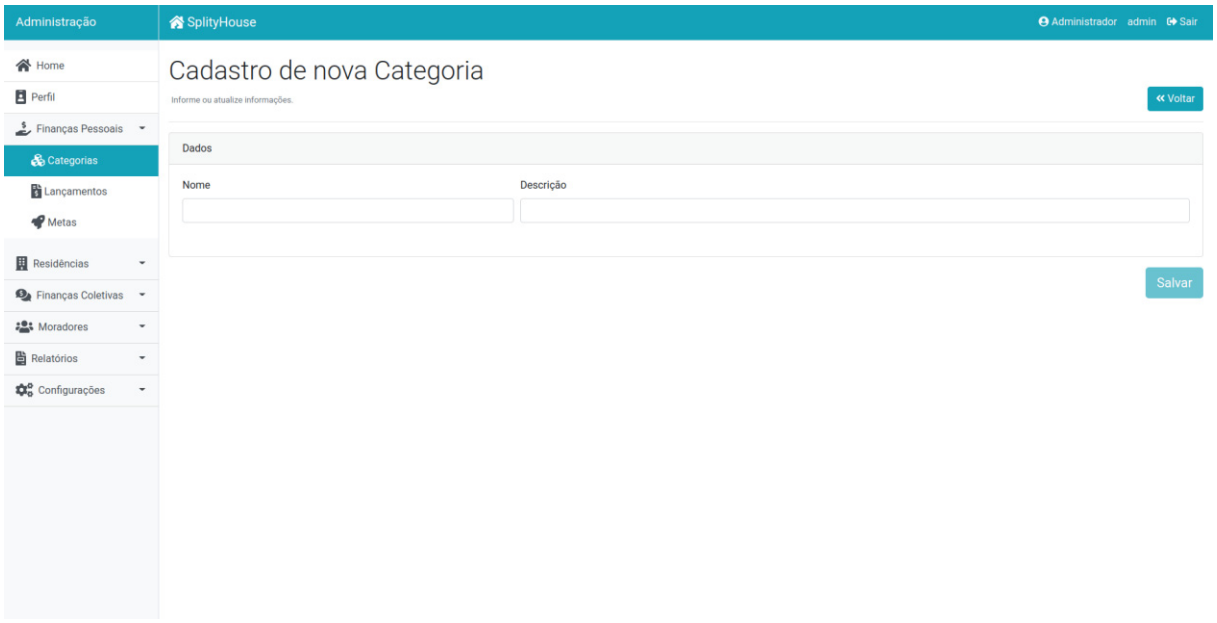
O usuário pode cadastrar categorias que são associadas aos lançamentos financeiros pessoais. As categorias são listadas, como observado na Figura 24, possibilitando ações de cadastro, edição e exclusão. Para cadastrar uma categoria nova, os campos “Nome” e “Descrição” representados na Figura 25 devem ser preenchidos.

FIGURA 24 - CATEGORIAS



FONTE: O Autor (2021)

FIGURA 25 - NOVA CATEGORIA



FONTE: O Autor (2021)

4.4.2 Lançamentos

O usuário pode cadastrar lançamentos referente a operações financeiras pessoais, podendo ser valores de entrada ou de saída. Os dados cadastrador aparecem listados, assim como opções de filtro de pesquisa e ações de edição e exclusão. Opções que podem ser observadas na Figura 26.

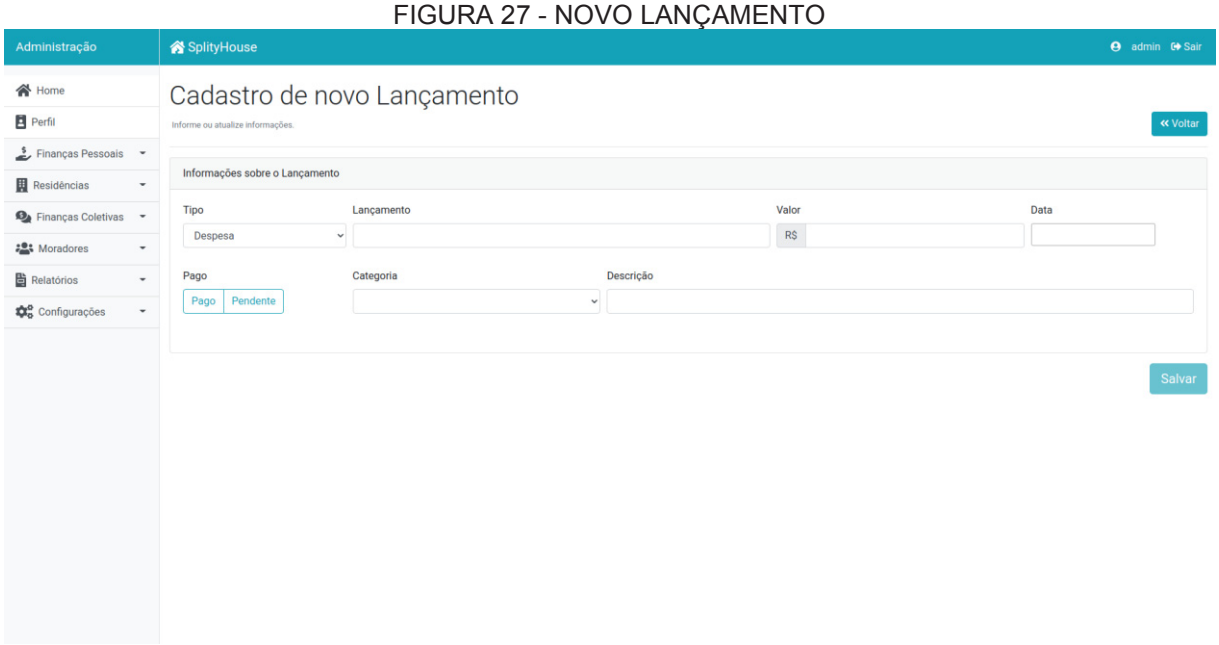
Para o usuário cadastrar um novo lançamento, ele deve selecionar o campo “Tipo”, tendo como alternativas “Despesa” e “Receita, selecionar o campo “Pago”, tendo como opção “Pago” e “Pendente”, selecionar a categoria de acordo com o que já está cadastrado no sistema. E por fim preencher os campos “Lançamento”, “Valor”, “Data” e “Descrição”.

Estas opções podem ser observadas na Figura 27.

FIGURA 26 - LANÇAMENTOS

FONTE: O Autor (2021)

FIGURA 27 - NOVO LANÇAMENTO



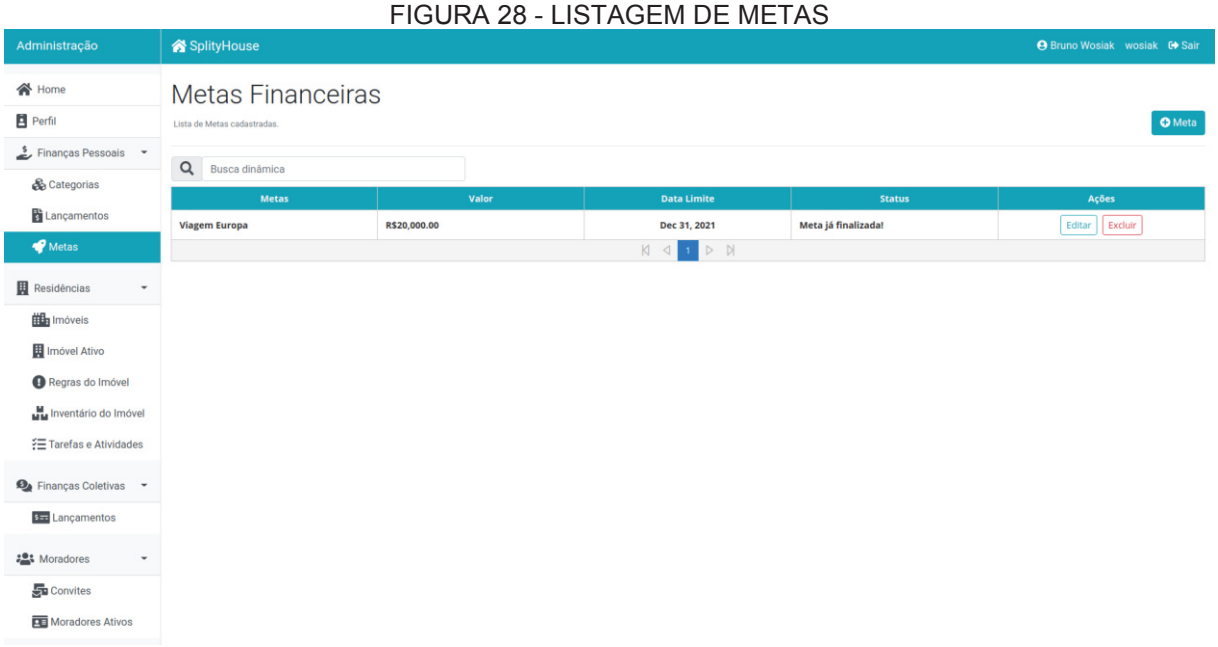
FONTE: O Autor (2021)

4.4.3 Metas

O usuário pode cadastrar metas financeiras no sistema, com o objetivo de visualizar o seu saldo em contraste com algum valor de objetivo perante um prazo. Ao acessar o menu de metas, o sistema exibe todos os dados cadastrados, como demonstrado na Figura 28.

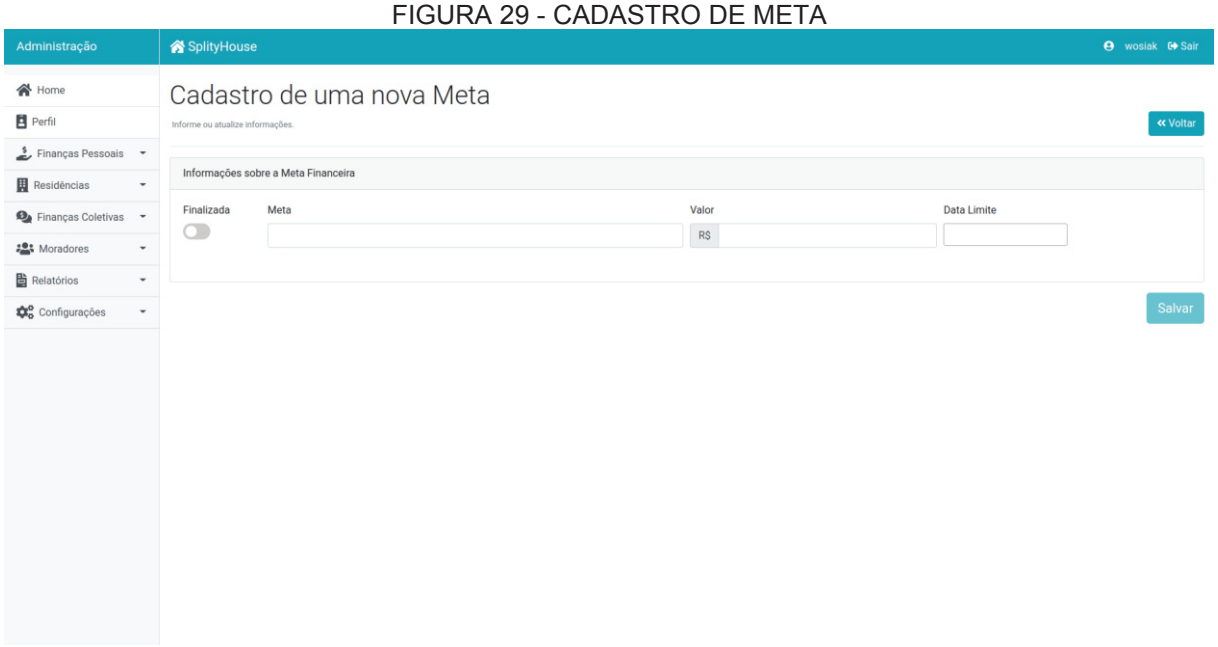
Para realizar um cadastro de uma nova meta, o usuário deve preencher os campos “Meta”, “Valor” e “Data Limite” (Figura 29). Também é possível na edição de uma meta modificar o menu “Finalizada” para concluir uma meta.

FIGURA 28 - LISTAGEM DE METAS



FONTE: O Autor (2021)

FIGURA 29 - CADASTRO DE META



FONTE: O Autor (2021)

4.5 RESIDÊNCIAS

As opções relacionadas as residências do usuário serão demonstradas a seguir.

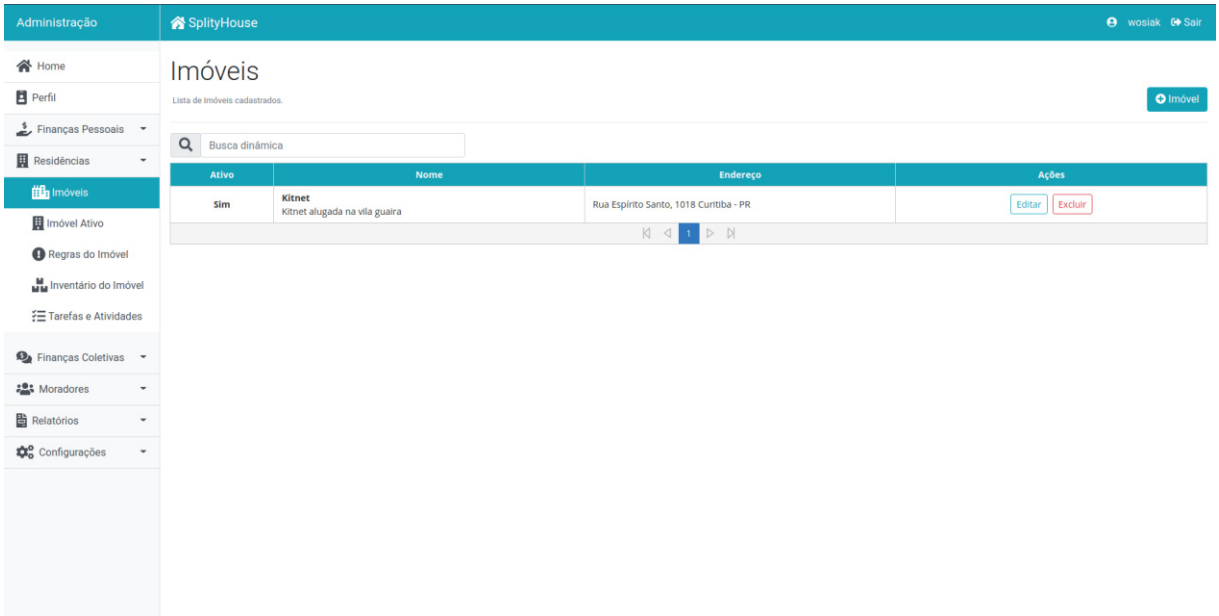
4.5.1 Imóveis

O usuário pode cadastrar imóveis no sistema, com isso, possibilitando todo o controle relacionado a sua residência atual. Ao acessar o menu de residências o sistema exibe os imóveis já cadastrados, como ilustra a Figura 30.

Para um imóvel novo ser cadastrado o usuário deve preencher os seguintes campos demonstrados na Figura 31: “Nome”, “Descrição”, “Proprietário”, “Telefone Proprietário”, “Cep”, “Rua”, “Número”, “Complemento”, “Cidade” e “Estado”.

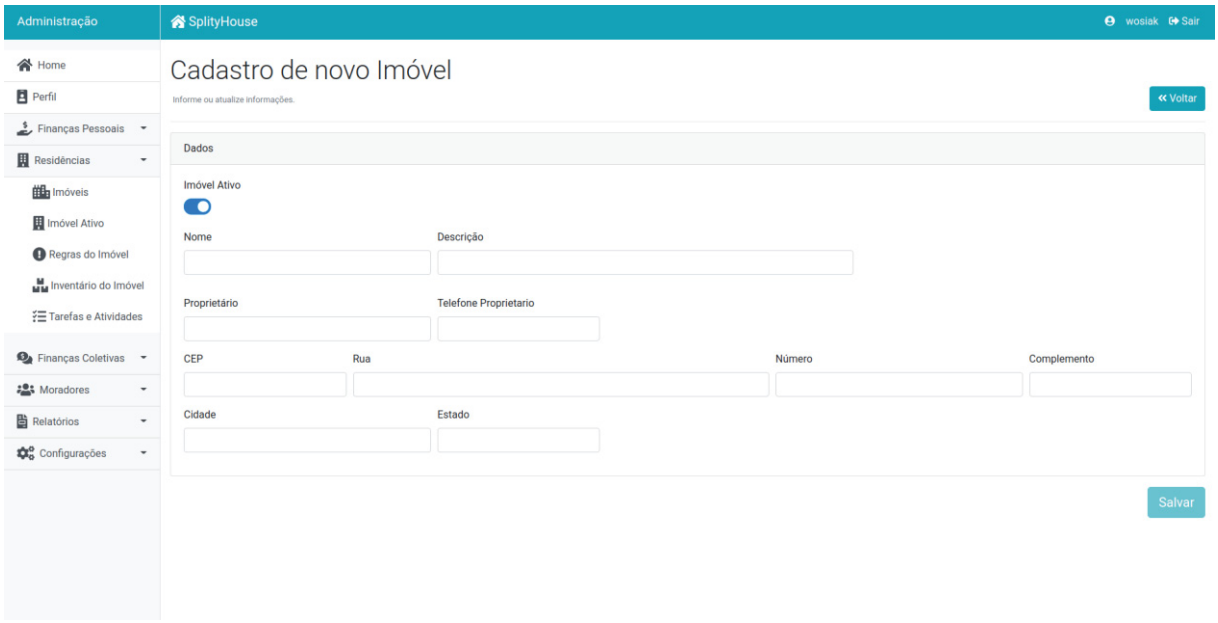
Também nas opções de edição de Imóvel, é possível alterar o *status* de uma residência de ativa para inativa e vice versas. Assim alterando todos os dados carregados no sistema de acordo com o imóvel ativo no momento.

FIGURA 30 - LISTAGEM DE IMÓVEIS



FONTE: O Autor (2021)

FIGURA 31 - CADASTRO DE IMÓVEL

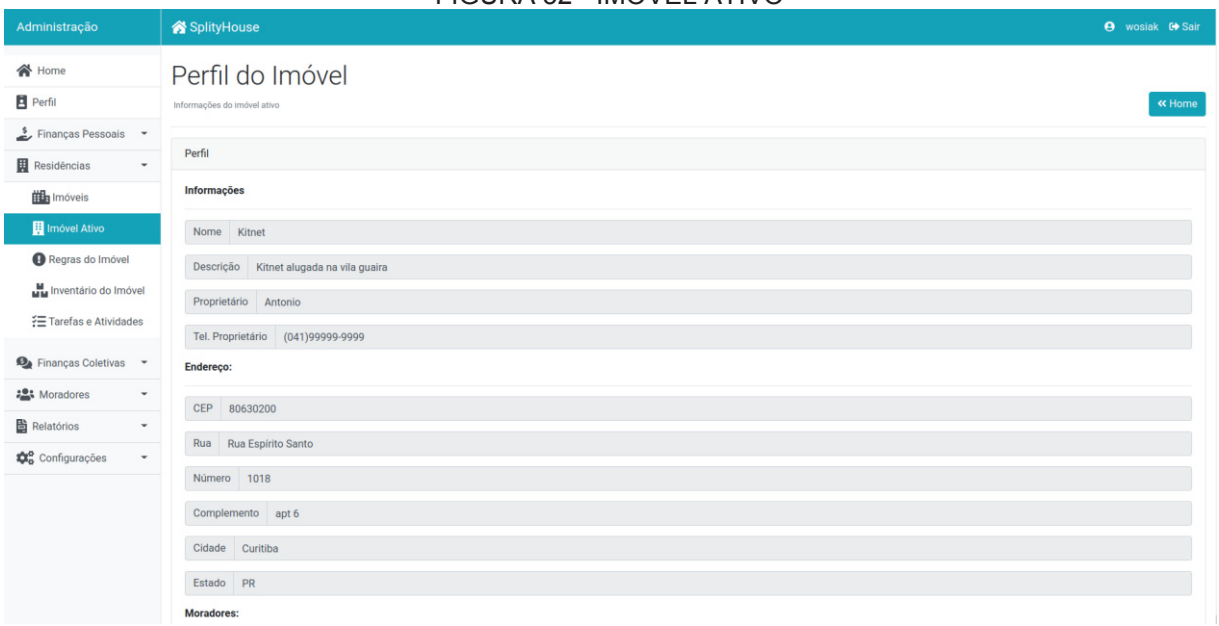


FONTE: O Autor (2021)

4.5.2 Imóvel Ativo

Nesta tela o usuário pode visualizar todas as informações do imóvel no qual está ativo no momento no sistema. A visualização dessas informações pode ser observada na Figura 32.

FIGURA 32 - IMÓVEL ATIVO

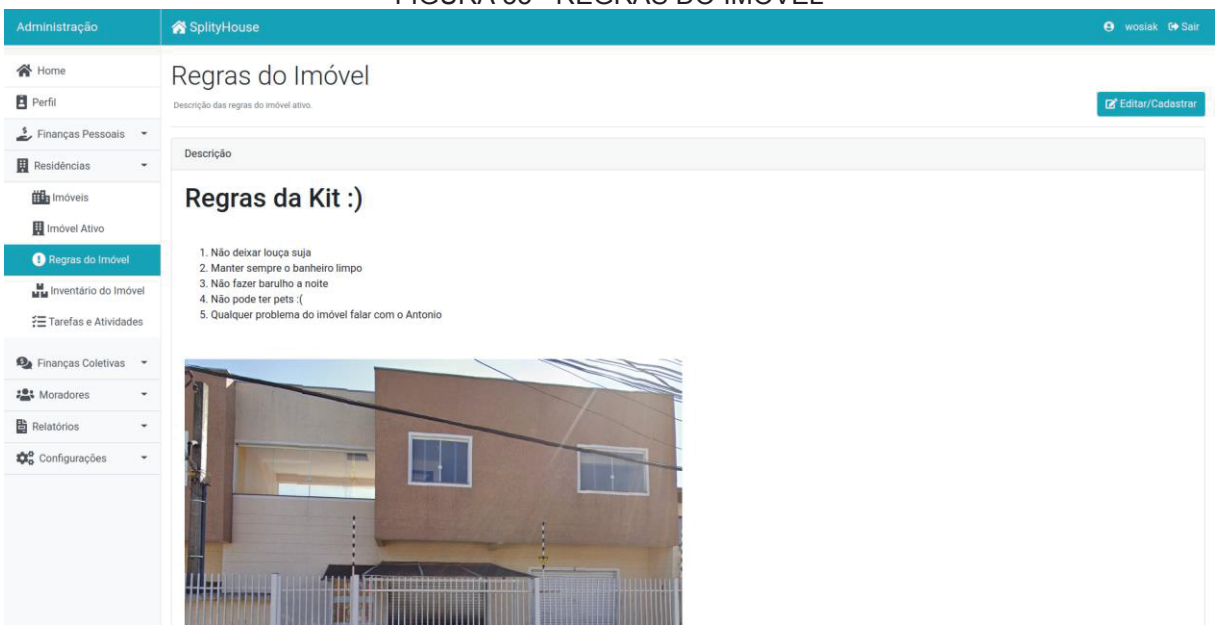


FONTE: O Autor (2021)

4.5.3 Regras do Imóvel

Nesta opção o usuário pode cadastrar as regras do imóvel ativo. Esse cadastro dispõe de muitas opções de edição de texto, inserção de imagens e links, para que as regras fiquem bem claras. O cadastro é ilustrado na Figura 33.

FIGURA 33 - REGRAS DO IMÓVEL



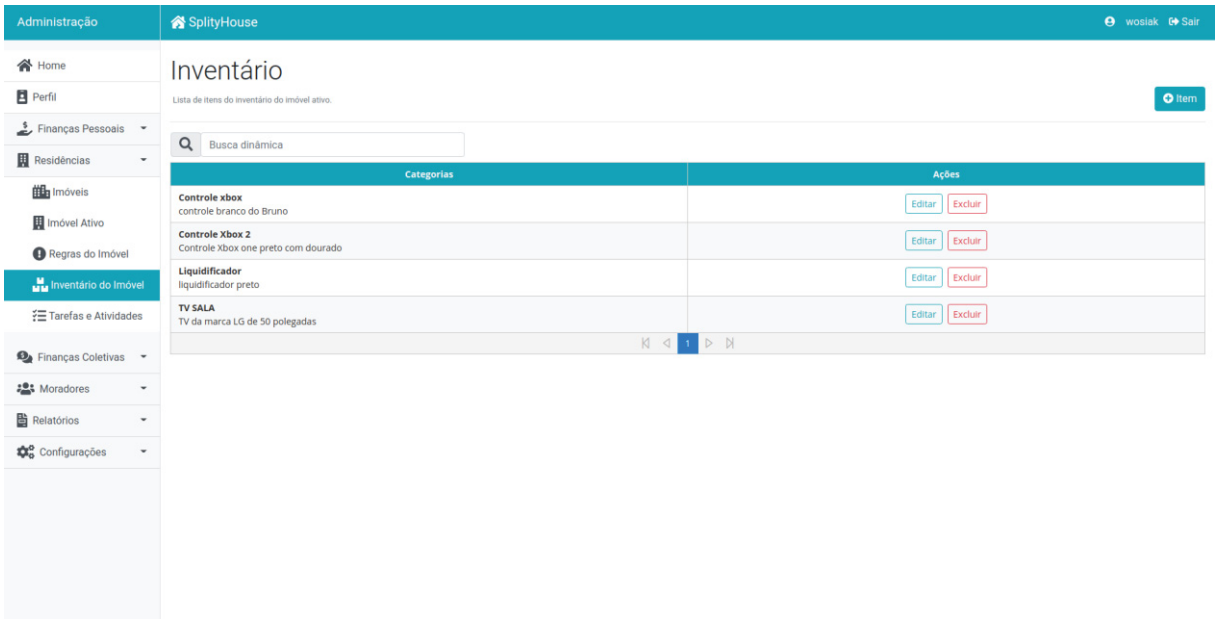
FONTE: O Autor (2021)

4.5.4 Inventário do Imóvel

Nesta opção, o usuário tem a possibilidade de realizar o controle dos itens do imóvel ativo, associando a eles informações como descrição e dono. Ao acessar o menu correspondente ao inventário, o sistema mostra a listagem dos dados cadastrados, como pode ser observado na Figura 34.

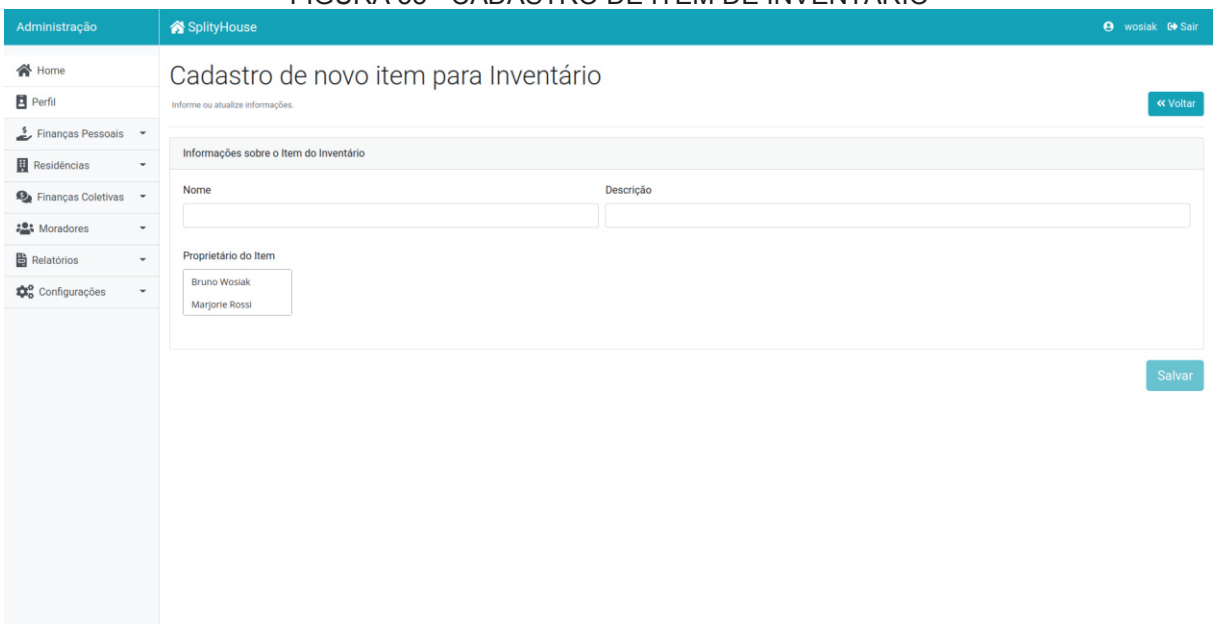
Para ser cadastrado um novo item de inventário, o usuário deve preencher os seguintes campos: “Nome”, “Descrição” e selecionar o dono do item no menu “Proprietário do Item”. O cadastro pode ser visto na Figura 35.

FIGURA 34 - LISTAGEM DE ITENS DE INVENTÁRIO



FONTE: O Autor (2021)

FIGURA 35 - CADASTRO DE ITEM DE INVENTÁRIO



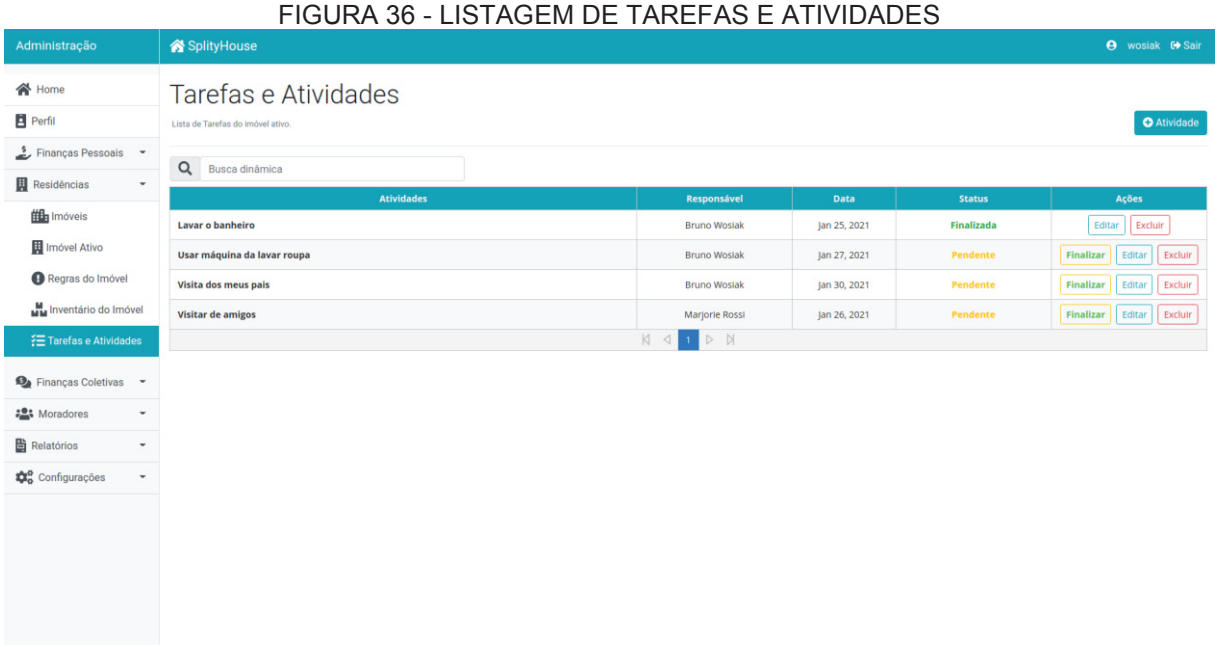
FONTE: O Autor (2021)

4.5.5 Tarefas e Atividades

Nesta opção do sistema o usuário pode cadastrar tarefas e atividades relacionadas ao imóvel ativo. Essas atividades são listadas conforme ilustra a Figura 36, e além disto, as atividades cadastradas e associadas a um morador aparecem no calendário da página inicial do sistema, como pode ser observado na Figura 37.

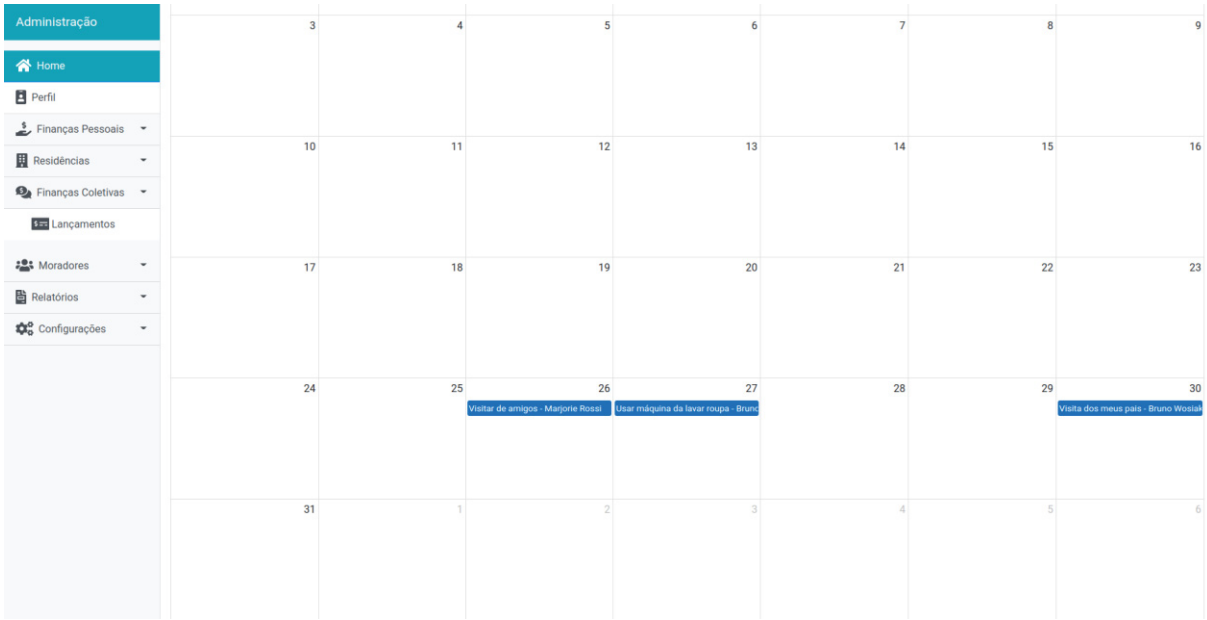
Para uma atividade ser cadastrada o usuário deve preencher os seguintes campos: “Nome”, “Data” e associar a tarefa a um morador no campo “Responsável”. A atividade possui o campo de “Atividade Finalizada” que pode ser modificado tanto no cadastro como na edição (Figura 38) e na tela de listagem. Ao alterar a atividade para atividade finalizada, ela deixa de ser exibida no calendário do imóvel.

FIGURA 36 - LISTAGEM DE TAREFAS E ATIVIDADES



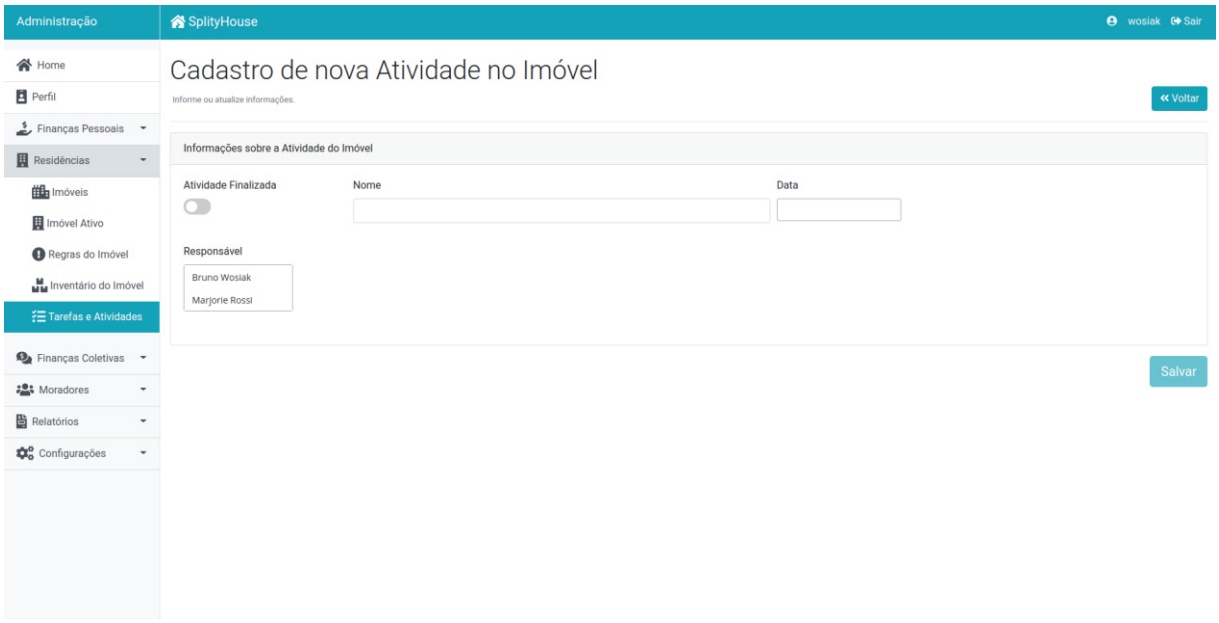
FONTE: O Autor (2021)

FIGURA 37 - TAREFAS E ATIVIDADES EM CALENDÁRIO



FONTE: O Autor (2021)

FIGURA 38 - NOVA ATIVIDADE



FONTE:O Autor (2021)

4.6 FINANÇAS COLETIVAS

As opções relacionadas as finanças coletivas dos usuários do imóvel serão demonstradas a seguir.

4.6.1 Lançamentos

O usuário administrador por cadastrar um lançamento coletivo, com isso o sistema irá gerar uma divisão do valor entre os moradores por igual. Essa operação gera um lançamento individual para cada usuário envolvido no imóvel.

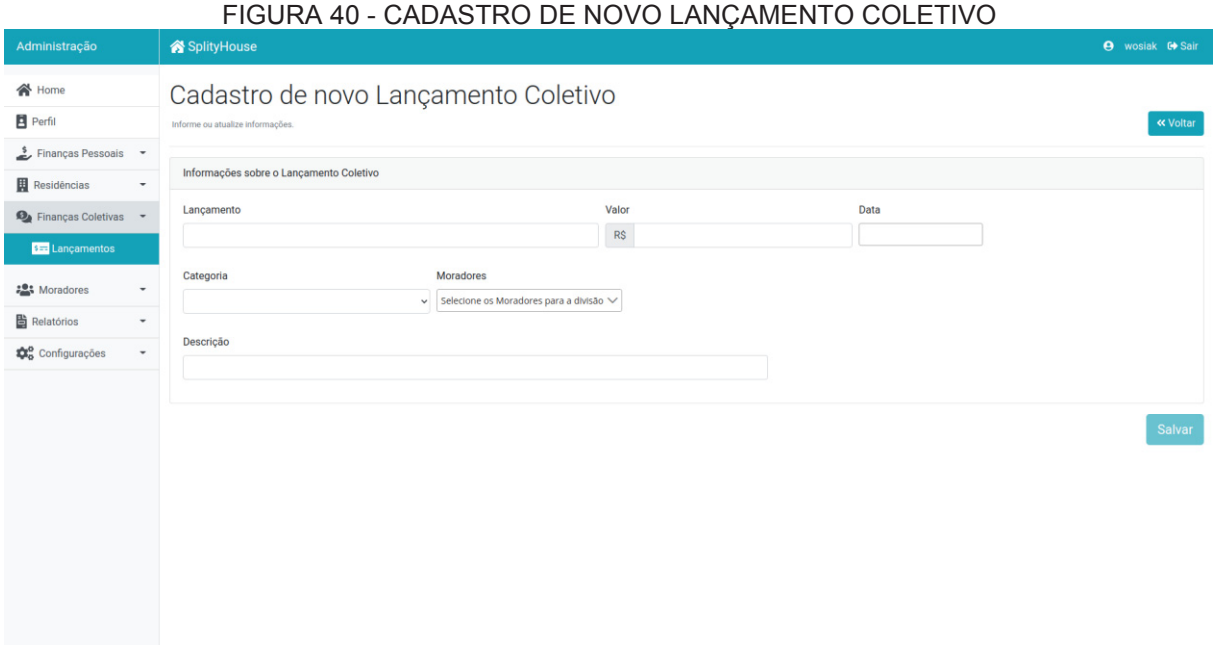
Os lançamentos coletivos podem ser observados na tela de listagem demonstrada pela Figura 39. Além disso o usuário também pode filtrá-los por data e realizar operações de edição e exclusão.

Para um lançamento coletivo novo ser cadastrado o usuário deve preencher os seguintes campos: “Lançamentos”, “Valor”, “Data”, selecionar a categoria na opção “Categoria”, selecionar os moradores da divisão e inserir a “Descrição”. A tela de cadastro é demonstrada na Figura 40.

FIGURA 39 - LISTAGEM DE LANÇAMENTOS COLETIVOS

FONTE: O Autor (2021)

FIGURA 40 - CADASTRO DE NOVO LANÇAMENTO COLETIVO



FONTE: O Autor (2021)

4.7 MORADORES

As opções relacionadas aos moradores do imóvel serão demonstradas a seguir.

4.7.1 Convites

O usuário responsável por um imóvel pode enviar convites para outros usuários cadastrados no sistema. Esta ação tem como o objetivo de convidar um usuário a se juntar ao seu imóvel como um morador.

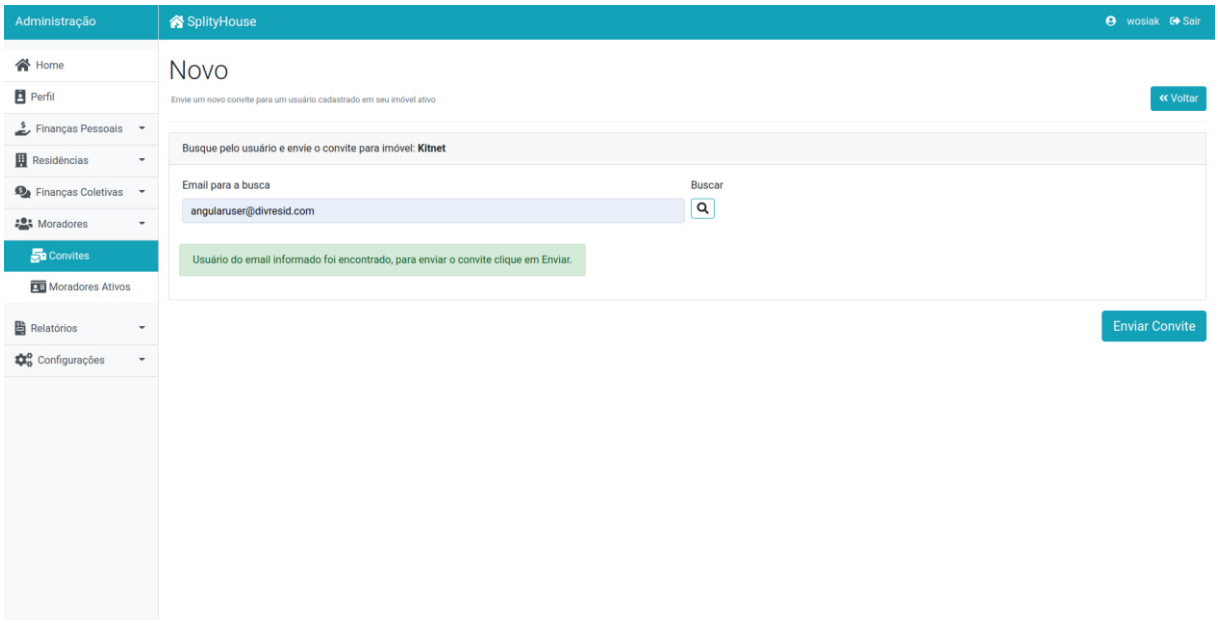
A listagem de convites realizados pelo usuário é mostrada na Figura 41. Nela é possível ver os convites aceitos e pendentes, assim como excluir ou enviá-los novamente.

Para cadastrar um novo convite o usuário inicialmente pesquisar um outro usuário já existente no sistema, como demonstrado na Figura 42. Após a mensagem de confirmação que usuário encontrado, ao pressionar o botão “Enviar Convite” o convite é enviado.

FIGURA 41 - LISTAGEM DE CONVITES

FONTE: O Autor (2021)

FIGURA 42 - NOVO CONVITE



FONTE: O Autor (2021)

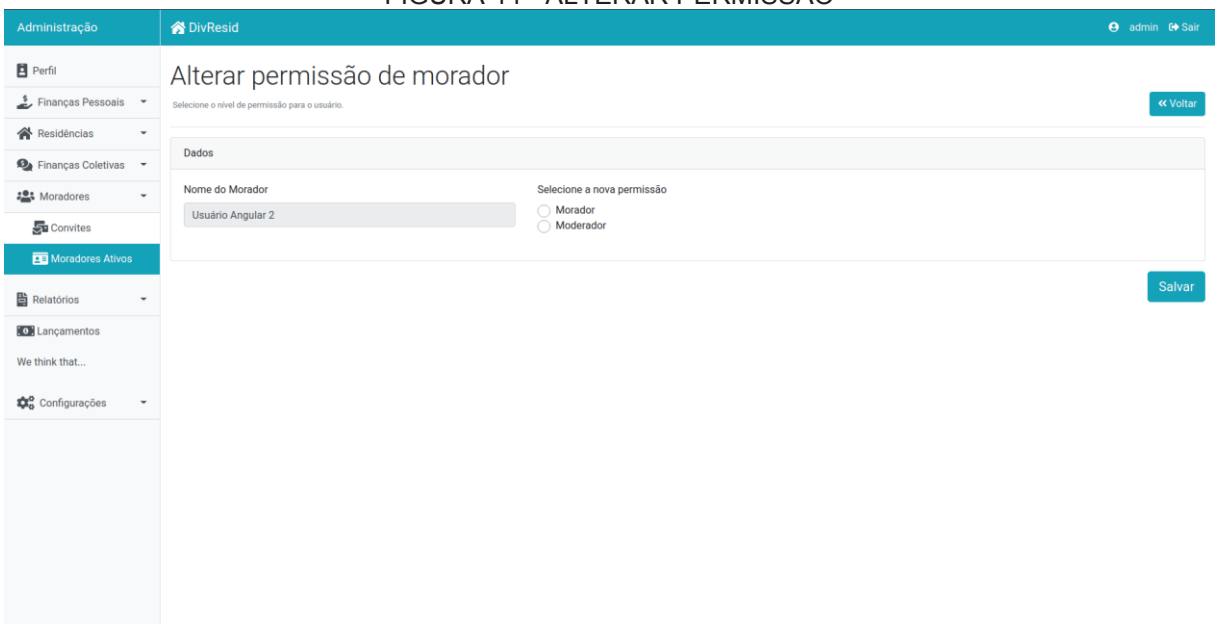
4.7.2 Moradores Ativos

O usuário pode administrar os moradores do seu imóvel ativo na listagem de moradores ativos representada na Figura 43. Nesta tela é possível editar a permissão de um morador (Figura 44) e remover o usuário do seu imóvel.

FIGURA 43 - LISTAGEM DE MORADORES

FONTE: O Autor (2021)

FIGURA 44 - ALTERAR PERMISSÃO



FONTE: O Autor (2021)

4.8 RELATÓRIOS

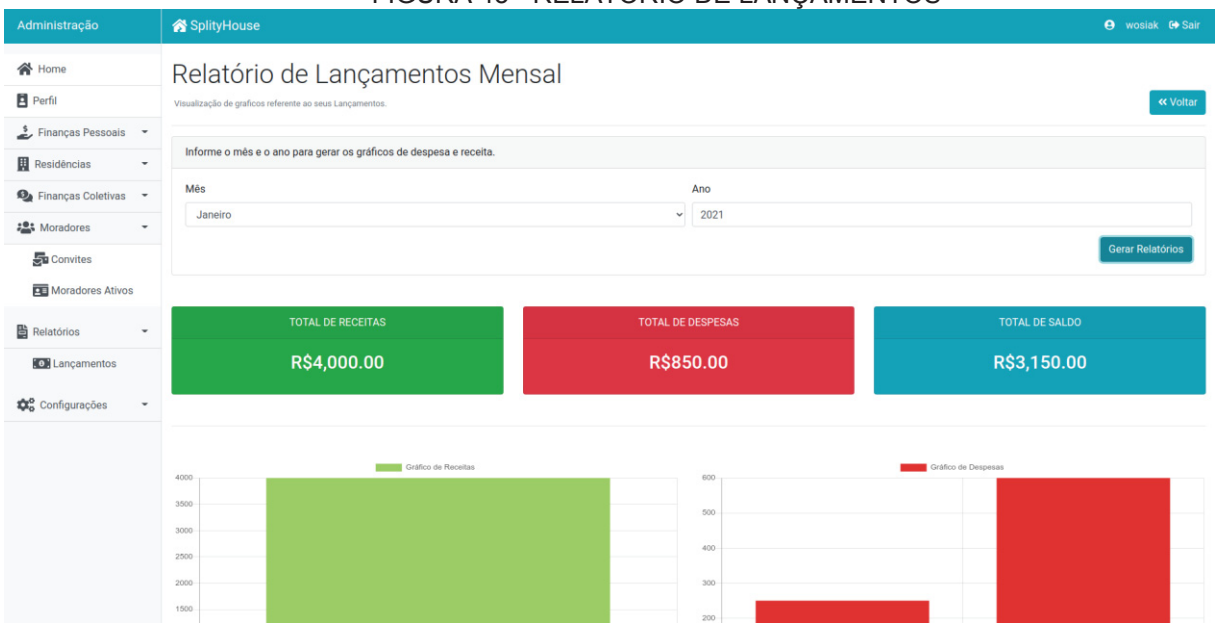
A opção relacionada ao relatório do sistema será demonstrada a seguir.

4.8.1 Lançamentos

O usuário pode gerar um relatório visual em relação ao balanço financeiro entre os lançamentos de entrada e saída. O relatório é gerado de acordo com a data selecionada, bastando o usuário seleciona o campo “Mês” e preencher o campo “Ano”.

A geração dos gráficos do relatório pode ser observada na Figura 45.

FIGURA 45 - RELATÓRIO DE LANÇAMENTOS



FONTE: O Autor (2021)

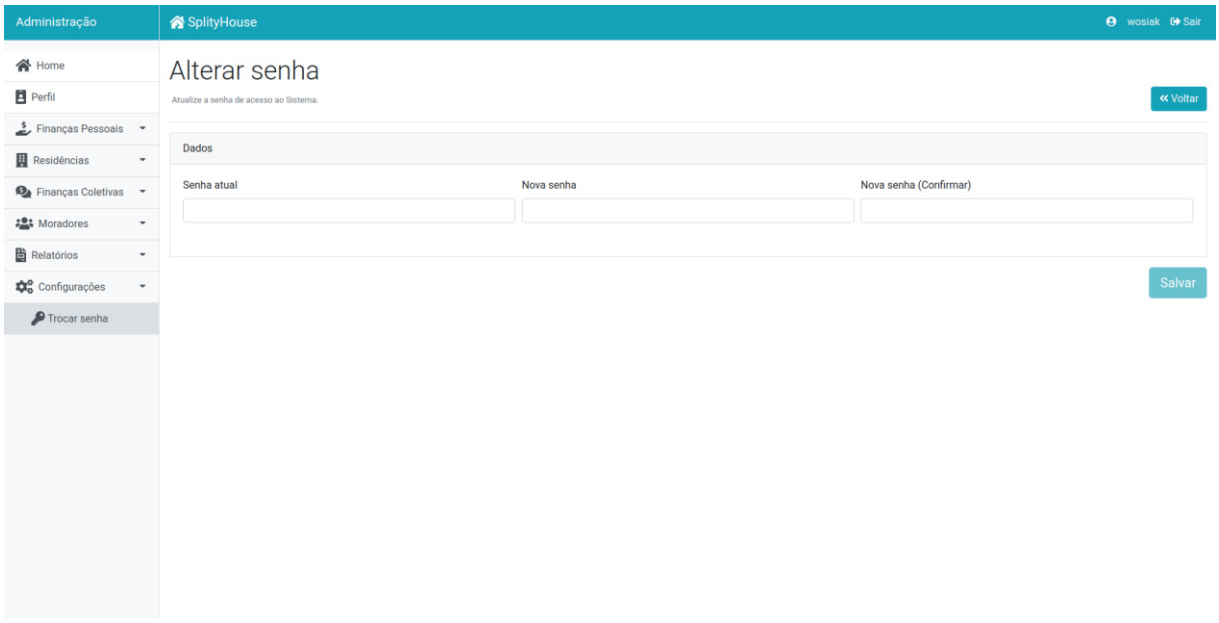
4.9 CONFIGURAÇÃO

A opção relacionada a configuração do sistema será demonstrada a seguir.

4.9.1 Trocar Senha

Nesta tela (Figura 46) o usuário pode alterar a sua senha, preenchendo os campos “Senha atual”, “Nova senha” e “Nova senha (Confirmar)”.

FIGURA 46 - ALTERAR SENHA



FONTE: O Autor (2021)

5 CONSIDERAÇÕES FINAIS

Neste trabalho foi apresentado um sistema de controle de financeiro e de imóvel compartilhado. Este sistema teve como objetivo suprir as necessidades dos usuários que tem como foco gerenciar uma casa compartilhada com outras pessoas em conjunto com suas finanças.

O processo de desenvolvimento desse software teve diversos desafios, pois nele foram utilizados conceitos avançados de programação orientada à objeto e tecnologias de *frontend,* a qual o autor possuía pouco conhecimento. Outro desafio do projeto foi devido a constantes mudanças de funcionalidades, em conjunto com adiamento de prazos devido a pandemia no ano de 2021, que consequentemente gerou um cronograma extenso e volátil.

Contudo, mesmo com as adversidades o software atingiu os objetivos propostos nesse trabalho, resultando em uma implementação satisfatória e resultando em grandes aprendizados relacionados a programação, gerenciamento e modelagem de projetos de software.

5.1 RECOMENDAÇÕES PARA TRABALHOS FUTUROS

Mesmo com a conclusão desse projeto, é evidente que é necessário realizar melhorias e implementações que são essenciais quando o software é colocado em uma perspectiva de vida futura e a longo prazo. Essas melhorias são descritas a seguir.

5.1.1 Melhorias em Filtros de dados

Os filtros implementados são engessados devido à falta de habilidade na tecnologia utilizada. Alguns deles o usuário deve preencher campos em conjunto com seleções para realizar um filtro por período. Isso deve ser eliminado e substituído por uma busca mais dinâmica e direta, com isso favorecendo a usabilidade do sistema.

5.1.2 Perfil de usuários

O perfil de usuário implementado possui pouca informação e é limitado somente ao usuário autenticado. Essa funcionalidade para justificar a sua existência deve ser melhorada para exibir mais informações úteis e possibilitar a visualização das mesmas informações de outros usuários.

5.1.3 Divisão de lançamentos automática

A divisão de lançamentos coletivos é feita de forma manual para cada gasto cadastrado no sistema. Porém há gastos que são constantes por um período pré-definido. Isso gera trabalho repetitivo para o usuário que está realizando está ação. O sistema deve ter a possibilidade de fixar gastos fixos e gerar essas despesas automaticamente.

5.1.4 Relatórios

O sistema possui atualmente somente um relatório relacionado as entradas e saídas do usuário autenticado. O sistema deve ter mais opções de relatórios de gerência, principalmente em relação a imóveis. Também seria necessário possibilitar a impressão deles que não foi implementada devido a dificuldades com a tecnologia utilizada.

5.1.5 Desenvolvimento de *frontend mobile*

A aplicação foi desenvolvida somente para computadores. E mesmo que haja a implementação de responsividade para celular no sistema, isso não supre a necessidade de uma utilização agradável e nativa para esse tipo de equipamento.

6 REFERÊNCIAS

ABRIL. 4 grandes dicas para evitar conflitos por causa de dinheiro. Disponível em: https://claudia.abril.com.br/sua-vida/4-grandes-dicas-para-evitar-conflitos-por-conta-de-dinheiro/. Acesso em: 9 jul. 2020.

ANDREBONA. A importância de administrar finanças pessoais e manter um bom planejamento financeiro. Disponível em: https://andrebona.com.br/importancia-de-administrar-financas-pessoais-e-manter-um-bom-planejamento-financeiro. Acesso em: 13 jun. 2020.

ANGULAR. Introduction to Angular concepts. Disponível em: https://angular.io/guide/architecture. Acesso em: 11 jun. 2020.

ANICHE, Maurício. **Orientação a Objetos e SOLID para Ninjas**: Projetando classes flexíveis. 1. ed. São Paulo: Casa do Código, 2020.

AQUILES, Alexandre; FERREIRA, Rodrigo. **Controlando versões com Git e Github**. 1. ed. São Paulo: Casa do Código, 2018.

ASTAH. About Astah by ChangeVision. Disponível em: https://astah.net/about/. Acesso em: 20 ago. 2020.

ATLASSIAN. What is version control. Disponível em: https://www.atlassian.com/git/tutorials/what-is-version-control#:~:text=Version%20control%20helps%20teams%20solve,working%20at%20the%20same%20time. Acesso em: 17 ago. 2020.

BAELDUNG. A Solid Guide to SOLID Principles. Disponível em: https://www.baeldung.com/solid-principles. Acesso em: 20 mai. 2020.

BIBLIOTECA IBGE. Censo Demográfico 2010. Disponível em: https://biblioteca.ibge.gov.br/visualizacao/periodicos/97/cd_2010_familias_domicilios_amostra.pdf. Acesso em: 15 mai. 2020.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML Guia do Usuário**. 2. ed. São Paulo: ELSEVIER, 2005.

CODE. Code editing. Redefined. Disponível em: https://code.visualstudio.com/. Acesso em: 20 ago. 2020.

DATE, Christopher J. **Introdução a sistemas de bancos de dados**. 8. ed. Rio de Janeiro: Elsevier, 2004.

DEVMEDIA. A importância dos Padrões de Projeto. Disponível em: https://www.devmedia.com.br/a-importancia-dos-padroes-de-projeto/22549#:~:text=Os%20padr%C3%B5es%20de%20projeto%2C%20tamb%C3%A9m,mais%20flex%C3%ADvel%2C%20elegante%20e%20reus%C3%A1vel. Acesso em: 4 ago. 2020.

DEVMEDIA. Introdução ao TypeScript. Disponível em: https://www.devmedia.com.br/introducao-ao-typescript/36729. Acesso em: 1 jun. 2020.

EDPRESSO. What are the SOLID principles in C#?. Disponível em: https://www.educative.io/edpresso/what-are-the-solid-principles-in-c-sharp. Acesso em: 20 mai. 2020.

ESTADÃO. Na crise, casas compartilhadas podem ser opção. Disponível em: https://economia.estadao.com.br/blogs/radar-imobiliario/na-crise-casas-compartilhadas-podem-ser-opcao/. Acesso em: 13 mai. 2020.

FIELDING, R. T. Architectural Styles and the Design of Network-based Software Architectures. 2000. Dissertação (Doutorado em Informação e Ciência da Computação) – University of California, 2000. Disponível em: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm. Acesso em: 06 abr. 2021.

FINK, Gil; FLATOW, Ido; GROUP, Sela. **Pro Single Page Application Development**. 1. ed. New York: Apress, 2014.

FOLHA DE SÃO PAULO. Companhia e sustentabilidade motivam divisão de moradias. Disponível em: https://www1.folha.uol.com.br/sobretudo/morar/2016/08/1807337-companhia-e-sustentabilidade-motiva-compartilhamento-de-moradias.shtml. Acesso em: 14 mai. 2020.

FOLHA DE SÃO PAULO. Novo estilo de moradia compartilhada, coliving é tendência em São Paulo. Disponível em: https://www1.folha.uol.com.br/sobretudo/morar/2018/11/1984613-novo-estilo-de-moradia-compartilhada-coliving-e-tendencia-em-sao-paulo.shtml. Acesso em: 11 jun. 2020.

GAZETA DO POVO. Crise aposentou "regra de ouro" da rentabilidade do aluguel. Disponível em: https://www.gazetadopovo.com.br/economia/crise-aposentou-regra-de-ouro-de-rentabilidade-do-aluguel-ac7wdbtrqhji27tp12yiy7rhl/. Acesso em: 14 mai. 2020.

GITHUB. GitHubis how people build software. Disponível em: https://github.com/about. Acesso em: 20 ago. 2020.

GITMAN, Lawrence J. **Princípios de administração financeira**. 12. ed. São Paulo: Pearson, 2010.

IGTI. O que o Spring pode fazer por você hoje? Disponível em: https://www.igti.com.br/blog/o-que-o-spring-pode-fazer-por-voce-hoje/. Acesso em: 19 ago. 2020.

ISTOÉ. Mercado imobiliário: a tendência de expansão vai se confirmar? Disponível em: https://www.istoedinheiro.com.br/mercado-imobiliario-a-tendencia-de-expansao-vai-se-confirmar/. Acesso em: 13 mai. 2020.

JADHAV, Madhuri A.; SAWANT, Balkrishna R.; DESHMUKH, Anushree. Single Page Application using AngularJS. International Journal of Computer Science and Information Technologies, Navi Mumbai, v. 6, n. 1, p. 2876-2879, mai./2015. Disponível em: http://www.ijcsit.com/docs/Volume%206/vol6issue03/ijcsit20150603195.pdf. Acesso em: 29 mai. 2020.

JWT. JSON Web Tokens are an open, industry standard RFC 7519 method for representing claims securely between two parties. Disponível em: https://jwt.io/. Acesso em: 24 ago. 2020.

LARMAN, Craig. **Utilizando UML e Padrões**: Uma introdução à análise e ao projeto orientados a objeto e ao desenvolvimento iterativo. 3. ed. Porto Alegre: Bookman, 2007.

LAYTON, Mark C.; MORROW, David. **Scrum para leigos**. 2. ed. Rio de Janeiro: Alta Books, 2019.

LEARNSQL. **Which Major Companies Use PostgreSQL? What Do They Use It for?** Disponível em: https://learnsql.com/blog/companies-that-use-postgresql-in-business/. Acesso em: 19 ago. 2020.

LUCKOW, Décio Heinzelmann; MELO, A. A. D. **Programação Java para web**: Aprenda a desenvolver uma aplicação financeira pessoal com as ferramentas mais modernas da plataforma Java. 6. ed. São Paulo: Novatec, 2014.

MARTIN, Robert C.; MARTIN, Micah. **Agile Principles, Patterns and Pratices in C#**. 1. ed. Massachusetts: Prentice Hall, 2006.

MENDES, Douglas Rocha. **Programação Java**: com ênfase em Orientação a Objetos. 1. ed. São Paulo: Novatec, 2009.

NOTION. All-in-one workspace: One tool for your whole team. Write, plan, and get organized. Disponível em: https://www.notion.so/. Acesso em: 13 fev. 2021.

NUNES, Denise Vianna; VIEIRA, Larissa Tavares. MODOS DE HABITAR A CIDADE CONTEMPORÂNEA: Moradia compartilhada e colaborativa. Associação Nacional de Pós-graduação e Pesquisa em Planejamento Urbano e Regional, Natal, v. 1, n. 1, p. 1-12, mai./2019. Disponível em: http://anpur.org.br/xviiienanpur/anaisadmin/capapdf.php?reqid=571. Acesso em: 11 jun. 2020.

O'REGAN, Gerard. **Concise Guide to Software Engineering**: From Fundamentals to Application Methods. 1. ed. Switzerland: Springer, 2017.

OLHAR DIGITAL. Como reengajar o usuário que desinstalou o seu aplicativo. Disponível em: https://olhardigital.com.br/colunistas/marcus_imaizumi/post/como_reengajar_o_usuario_que_desinstalou_o_seu_aplicativo/78907. Acesso em: 15 mai. 2020.

POSTGRESQL. What is PostgreSQL? Disponível em: https://www.postgresql.org/about/. Acesso em: 11 jun. 2020.

POSTMAN. About Postman. Disponível em: https://www.postman.com/about-postman/. Acesso em: 20 ago. 2020.

POSTMAN. What is Newman? Disponível em: https://support.postman.com/hc/en-us/articles/115003710329-What-is-Newman-. Acesso em: 20 ago. 2020.

PRESSMAN, Roger S.; MAXIM, Bruce R. **Engenharia de Software**: Uma abordagem profissional. 8. ed. São Paulo: AMGH, 2016.

REZENDE, Denis Alcides. **Engenharia de Software e Sistemas de Informação**. 3. ed. São Paulo: Brasport, 2005.

SABBAGH, Rafael. **Scrum**: Gestão Ágil para Projetos de Sucesso. 1. ed. São Paulo: Casa do Código, 2020.

PERL MONGERS. Arquitetura REST e o serviço web 'RESTful'. Disponível em: http://sao-paulo.pm.org/pub/arquitetura-rest-e-o-servico-web-restful-#:~:text=REST%20%C3%A9%20um%20termo%20definido,%22%20(Representational%20State%20Transfer)-. Acesso em: 04 mai. 2021.

SCHILDT, Herbert. **Java para iniciantes**. 6. ed. Porto Alegre: Bookman, 2015.

SPC BRASIL. 46% dos brasileiros não controlam seu orçamento, revela pesquisa do SPC Brasil. Disponível em: https://www.spcbrasil.org.br/pesquisas/pesquisa/1201. Acesso em: 15 mai. 2020.

SPRING. Spring Boot Reference Documentation. Disponível em: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/. Acesso em: 1 jun. 2020.

SPRING. Spring Tools 4 for Eclipse. Disponível em: https://spring.io/tools. Acesso em: 20 ago. 2020.

SPRING. Spring framework reference. Disponível em: https://docs.spring.io/autorepo/docs/spring-framework/3.2.18.BUILD-SNAPSHOT/spring-framework-reference/pdf/spring-framework-reference.pdf. Acesso em: 31 mai. 2020.

SUTHERLAND, Jeff. **Scrum**: A arte de fazer o dobro do trabalho na metade do tempo. 2. ed. São Paulo: Leya, 2016.

TOTVS. Por que o Angular é um framework tão poderoso? Disponível em: https://www.totvs.com/blog/developers/angular/. Acesso em: 19 ago. 2020.

TOTVS. Arquitetura REST: Saiba o que é e seus diferenciais. Disponível em: https://www.totvs.com/blog/developers/rest/. Acesso em: 04 mai. 2021.

TRELLO. About Trello What's behind the boards**.** Disponível em: https://trello.com/about. Acesso em: 20 ago. 2020.

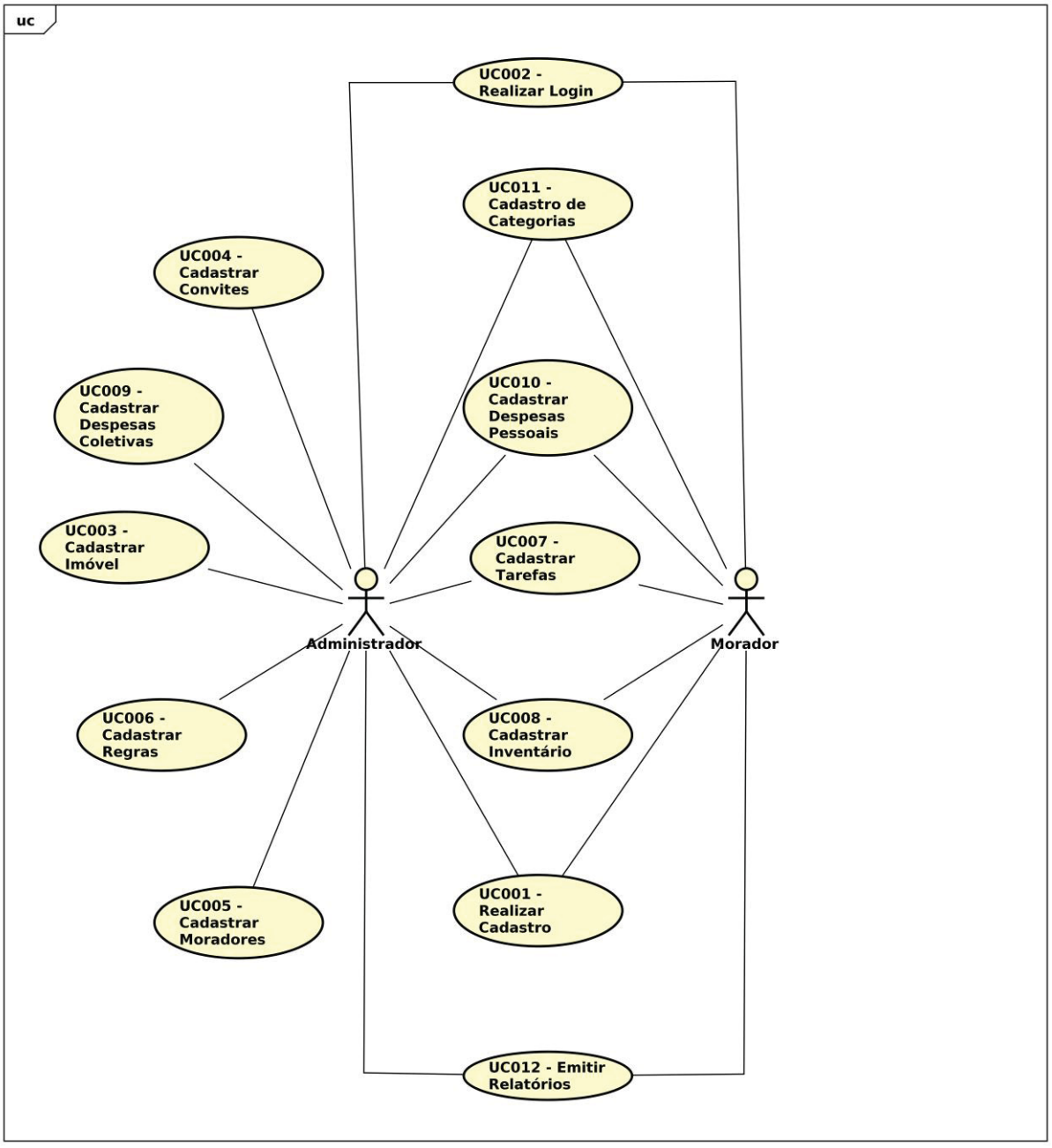
TYPESCRIPT. Typed JavaScript at Any Scale. Disponível em: https://www.typescriptlang.org. Acesso em: 25 mai. 2021.

UOL. Para dividir um lar com alguém não basta ter amizade**.** Disponível em: https://www.uol.com.br/universa/noticias/redacao/2013/11/15/para-dividir-um-lar-com-alguem-nao-basta-ter-amizade-veja-o-que-analisar.htm. Acesso em: 15 mai. 2020.

WAZLAWICK, Raul Sidnei. **Análise e projeto de sistemas de informação orientados a objetos**. 2. ed. São Paulo: Elsevier, 2011.

APÊNDICE A – DIAGRAMA DE CASO DE USO

FIGURA 47 - DIAGRAMA DE CASO DE USO



FONTE: O Autor (2021)

APÊNDICE B – ESPECIFICAÇÃO DE CASO DE USO

Especificação de Caso de Uso

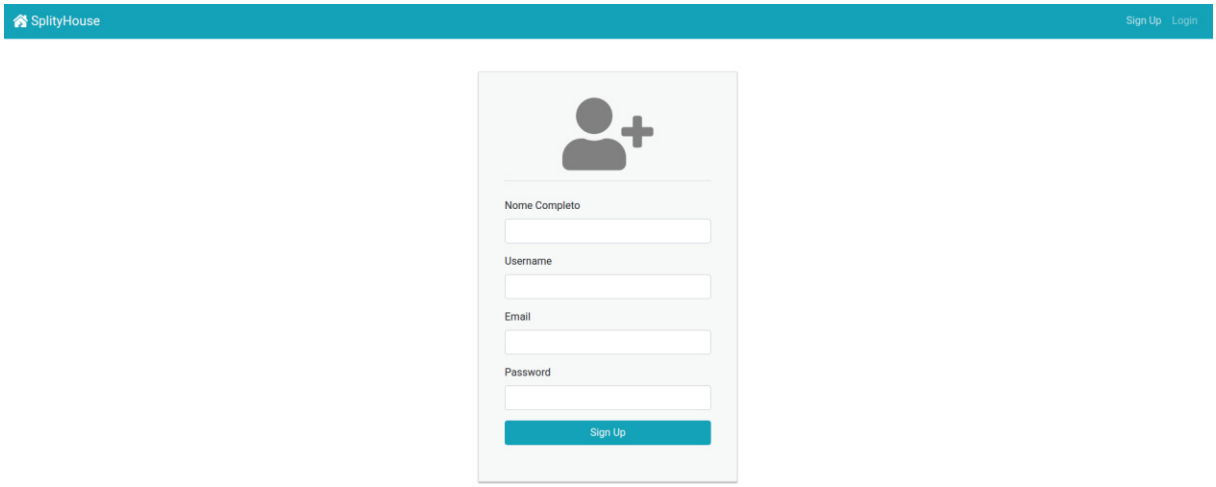
UC001 –Realizar Cadastro

Descrição

Este caso de uso serve para realizar o cadastro de usuários.

Data View

FIGURA 48 - DV001 TELA DE CADASTRO



FONTE: O Autor (2021)

Ator Primário

Administrador

Pré-Condição

O usuário deve estar conectado à internet.

Fluxo de Eventos Principal

1. O sistema apresenta a tela DV001;
2. O usuário preenche o campo “Nome Completo”;
3. O usuário preenche o campo “*Username”;*

4. O usuário preenche o campo “Email”;
5. O usuário preenche o campo “*Password*”;
6. O usuário pressiona o botão “*Sing Up*” **(A1) (A2)**;
7. O sistema valida os dados **(E1) (E2)**;
8. O sistema grava os dados na base;
9. O sistema chama o caso de uso UC002 - Realizar Login;
10. O caso de uso é encerrado;

Fluxos Alternativos

A1: O usuário pressiona no logo do sistema;

1. O sistema redireciona para a tela inicial do sistema;
2. O caso de uso é encerrado;

A2: O usuário pressiona o botão “Login”;

1. O sistema chama o UC002 – Realizar Login;
2. O caso de uso é encerrado;

Fluxo de Exceções

E1: O usuário não preenche algum dos campos;

1. O sistema exibe a mensagem “O campo é obrigatório”;

E2: O usuário preenche o campo Email com um e-mail inválido;

1. O sistema exibe a mensagem “Email deve ser válido”;

Pós-condição:

Ao final deste caso de uso, o usuário terá um cadastro no sistema e a possibilidade de realizar login.

Especificação de Caso de Uso

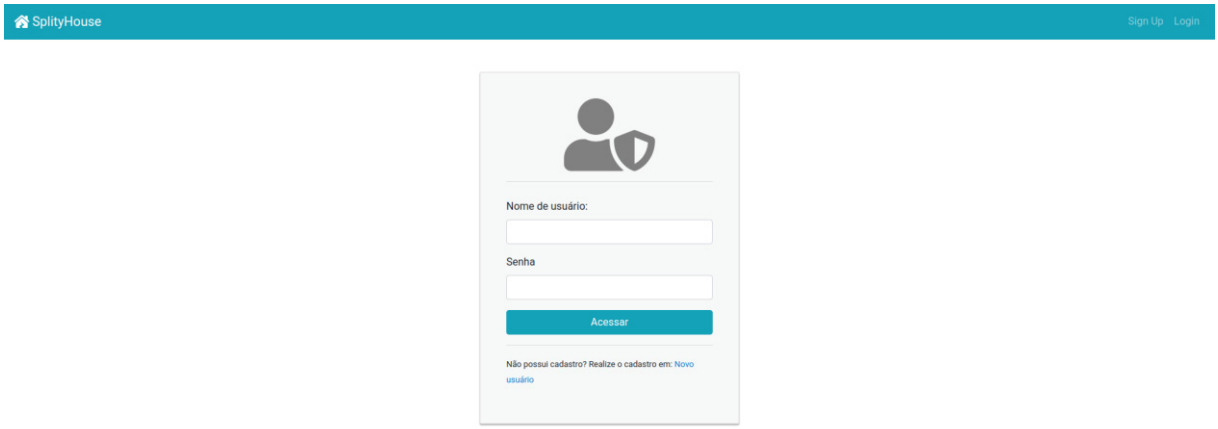
UC002 –Realizar Login

Descrição

Este caso de uso serve para realizar o login de usuários.

Data View

FIGURA 49 - DV002 TELA DE LOGIN DO SISTEMA



FONTE: O Autor (2021)

Ator Primário

Administrador

Pré-Condição

O usuário deve estar conectado à internet e ter cadastro válido no sistema.

Fluxo de Eventos Principal

1. O sistema apresenta a tela DV002;
2. O usuário preenche o campo “Nome de usuário”;

3. O usuário preenche o campo “Senha”;
4. O usuário pressiona o botão “Acessar” **(A1) (A2) (A3)**;
5. O sistema consiste nos dados **(E1) (E2)**;
6. O sistema redireciona o usuário para a tela inicial autenticada;
7. O caso de uso é encerrado;

Fluxos Alternativos

A1: O usuário pressiona no logo do sistema;

1. O sistema redireciona para a tela inicial do sistema;
2. O caso de uso é encerrado;

A2: O usuário pressiona o botão “Sing Up”;

1. O sistema chama o UC001 – Realizar Cadastro;
2. O caso de uso é encerrado;

Fluxo de Exceções

E1: O usuário não preenche algum dos campos;

1. O sistema exibe a mensagem “O campo é obrigatório”;

E2: Login inválido;

1. O sistema exibe a mensagem “Nome de usuário ou senha inválidos”;

Pós-condição:

Ao final deste caso de uso, o usuário estará autenticado no sistema, podendo acessar regiões do software protegidas por autenticação.

Especificação de Caso de Uso

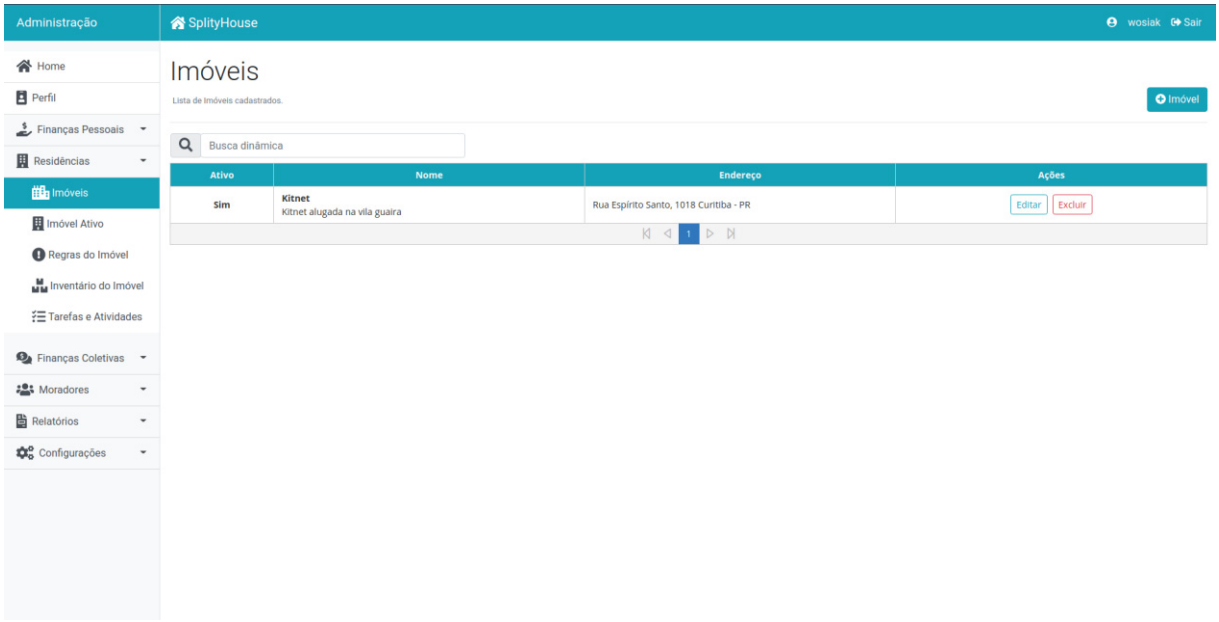
UC003 –Cadastrar Imóvel

Descrição

Este caso de uso serve para realizar cadastro de imóveis no sistema.

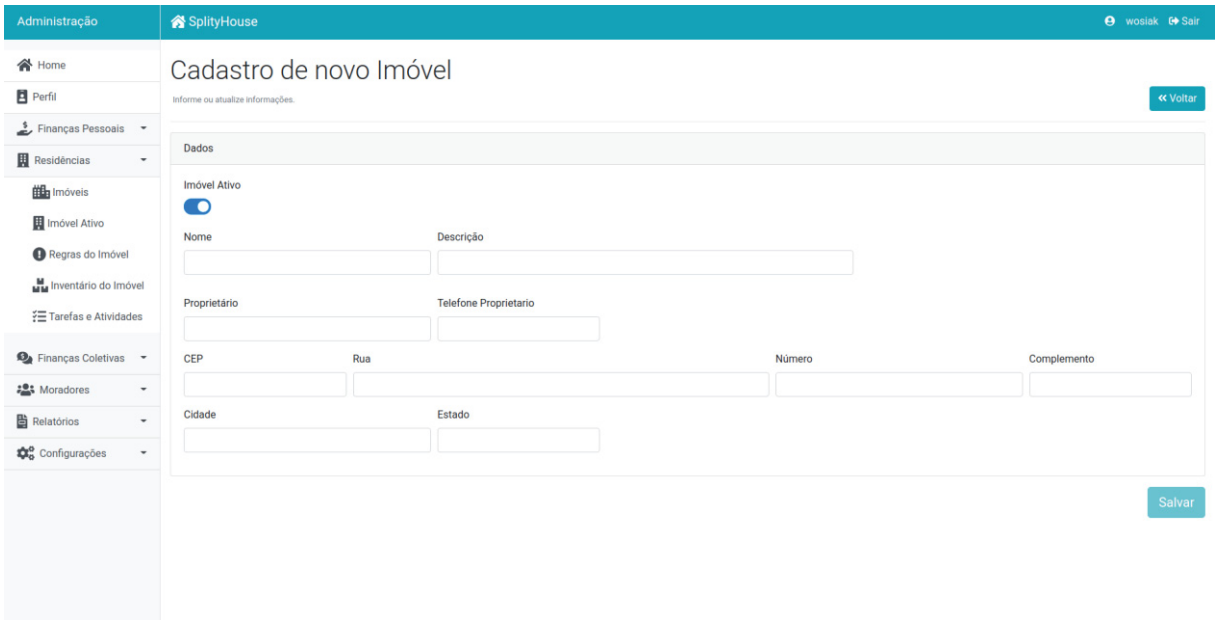
Data View

FIGURA 50 - DV003 TELA DE LISTAGEM DE IMÓVEIS



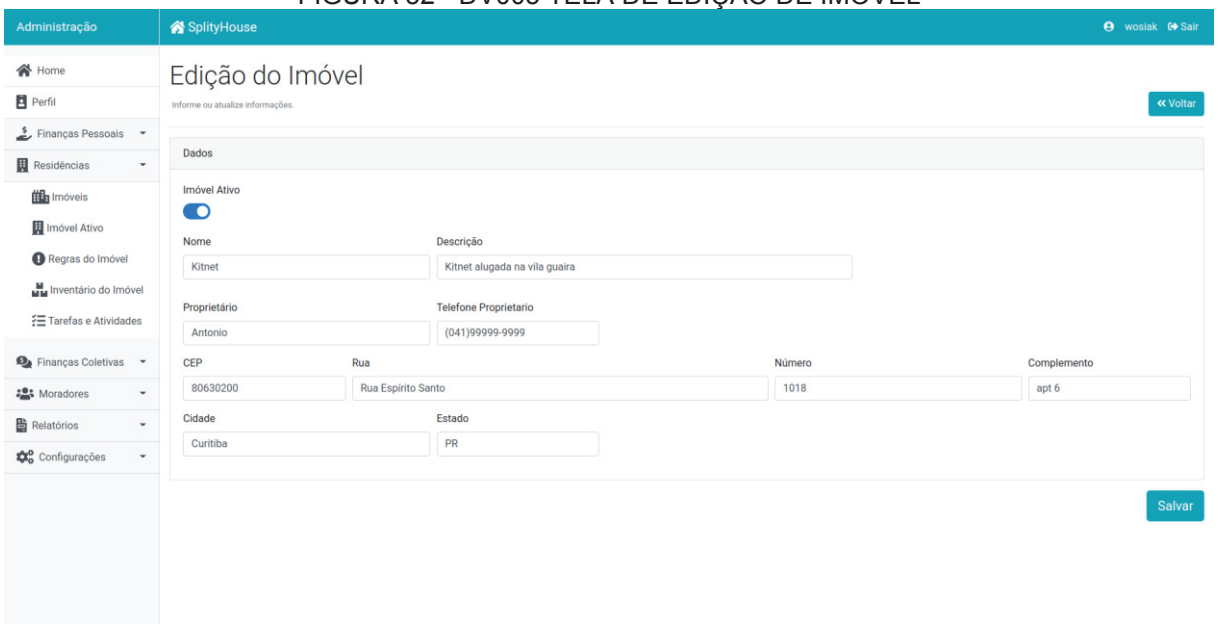
FONTE: O Autor (2021)

FIGURA 51 - DV004 TELA DE CADASTRO DE IMÓVEL



FONTE: O Autor (2021)

FIGURA 52 - DV005 TELA DE EDIÇÃO DE IMÓVEL



FONTE: O Autor (2021)

Pré-Condição

O usuário deve estar conectado à internet e ter cadastro válido no sistema e estar devidamente autenticado.

Fluxo de Eventos Principal

1. O usuário seleciona a opção “Imóveis” no menu “Residências” do sistema;
2. O sistema busca as informações dos imóveis cadastrados na base de dados **(E1)**;
3. O sistema apresenta a tela DV003;
4. O usuário pressiona o botão “+Imóvel” **(A1) (A2) (A3)**;
5. O sistema apresenta a tela DV004;
6. O usuário altera o botão “Imóvel Ativo”;
7. O usuário preenche o campo “Nome” **(E2) (E3)**;
8. O usuário preenche o campo “Descrição” **(E4)**;
9. O usuário preenche o campo “Proprietário”
10. O usuário preenche o campo “Telefone Proprietário”
11. O usuário preenche o campo “CEP”;
12. O sistema busca informações do CEP informado **(E6)**;
13. O sistema preenche os campos “Rua”, “Cidade” e “Estado”;
14. O usuário preenche o campo “Número”;
15. O usuário preenche o campo “Complemento”;
16. O usuário pressiona o botão “Salvar” **(E5)**;
17. O sistema grava os dados na base;
18. O caso de uso é encerrado;

Fluxos Alternativos

A1: O usuário pressiona no botão “Editar” em algum imóvel listado;

1. O sistema carrega as informações do Imóvel escolhido;
2. O sistema preenche os campos com os dados do Imóvel;
3. O sistema apresenta a tela DV005;

4. O usuário altera o botão “Imóvel Ativo”;
5. O usuário atualiza o campo “Nome” **(E2) (E3)**;
6. O usuário atualiza o campo “Descrição” **(E4)**;
7. O usuário atualiza o campo “Proprietário”
8. O usuário atualiza o campo “Telefone Proprietário”
9. O usuário atualiza o campo “CEP”;
10. O sistema busca informações do CEP informado **(E6)**;
11. O sistema atualiza os campos “Rua”, “Cidade” e “Estado”;
12. O usuário atualiza o campo “Número”;
13. O usuário atualiza o campo “Complemento”;
14. O usuário pressiona o botão “Salvar” **(E5) (A4)**;
15. O sistema grava os dados na base;
16. O caso de uso é encerrado;

A2: O usuário pressiona o botão “Excluir” em algum imóvel listado;

1. O sistema apresenta a mensagem de confirmação “Deseja realmente excluir?”;
2. O usuário confirma a exclusão do dado **(A5)**;
3. O sistema remove o dado da base;

A3: O usuário preenche o campo de busca dinâmica;

1. O sistema busca na base dados que contenham o texto informado **(E1)**;
2. O sistema preenche a tabela com os resultados;

A4: O usuário pressiona o botão “Voltar”;

1. O sistema redireciona para a DV003;

A5: O usuário pressiona o para não confinar a exclusão;

1. O sistema fecha a caixa de confirmação e não realiza a alteração;

Fluxo de Exceções

E1: Não tem imóveis previamente cadastrados ou não encontrado na busca;

1. O sistema informa a mensagem “Não foram encontrados Imóveis cadastrados”;

E2: O usuário não preencheu o campo “Nome”;

1. O sistema informa a mensagem “O campo é obrigatório”;

E3: O usuário preencheu o campo “Nome” com menos de 3 caracteres;

1. O sistema exibe a mensagem “O campo nome deve ter no mínimo 3 caracteres”;

E4: O usuário preencheu o campo “Descrição” com mais de 150 caracteres;

1. O sistema exibe a mensagem “A descrição deve ter no máximo 150 caracteres”;

E5: O usuário gerou erro de validação no formulário;

1. O sistema desabilita o botão “Salvar”;

E6: O usuário informou um CEP inválido;

2. O sistema exibe a mensagem “Problema ao consultar o CEP informado”;

Especificação de Caso de Uso

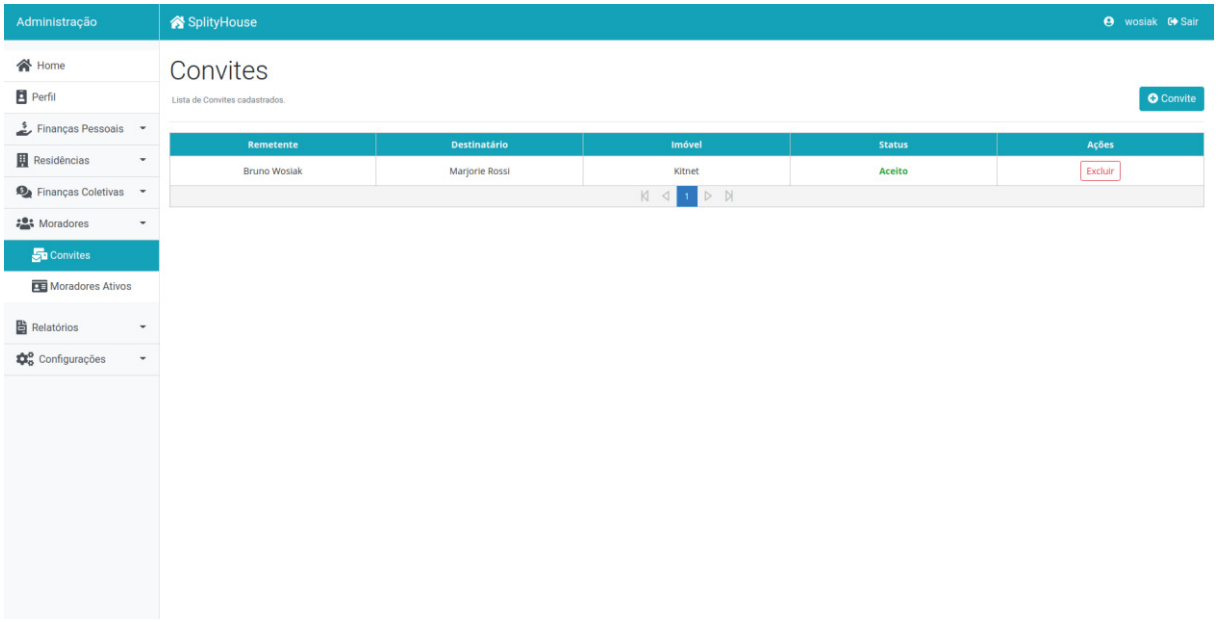
UC004 – Cadastrar Convites

Descrição

Este caso de uso serve para realizar o cadastro de convites para usuários.

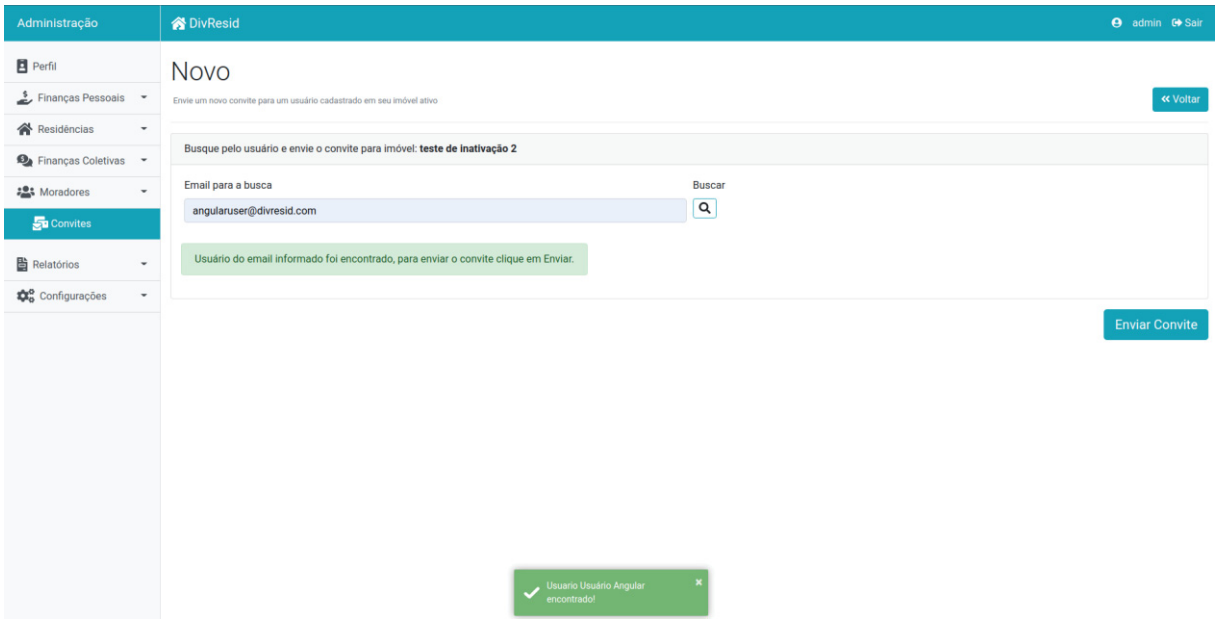
Data View

FIGURA 53 - DV006 TELA DE LISTAGEM DE CONVITES



FONTE: O Autor (2021)

FIGURA 54 - DV007 TELA DE CADASTRO DE CONVITE



FONTE: O Autor (2021)

Ator Primário

Administrador

Pré Condição

O usuário deve estar conectado à internet e ter cadastro válido no sistema e estar devidamente autenticado.

Fluxo de Eventos Principal

1. O usuário seleciona a opção “Convites” no menu lateral
2. O sistema busca as informações dos convites cadastrados na base de dados **(E1)**;
3. O sistema apresenta a tela DV006;
4. O usuário pressiona o botão “+Convite” **(A1) (A2)**;
5. O sistema apresenta a tela DV007;
6. O usuário preenche o campo “Email para a busca” **(E2)**;
7. O usuário pressiona o botão de busca;

8. O sistema busca as informações do usuário de acordo com o e-mail informado **(E3)**;
9. O sistema informa a mensagem “Usuário do email informado foi encontrado, para enviar o convite clique em Enviar.”;
10. O sistema apresenta o botão “Enviar Convite” **(R2)**;
11. O usuário pressiona o botão “Enviar Convite” **(A3)**;
12. O sistema grava os dados na base **(R3)**;
13. O caso de uso é encerrado;

Fluxos Alternativos

A1: O usuário pressiona no botão “Aceitar” em algum convite pendente;

1. O sistema apresenta o botão “Aceitar” **(R1)**;
2. O usuário pressiona o botão “Aceitar”;
3. O caso de uso é encerrado;

A2: O usuário pressiona o botão “Excluir” em algum convite listado;

1. O sistema apresenta a mensagem de confirmação “Deseja realmente excluir?”;
2. O usuário confirma a exclusão do dado **(A4)**;
3. O sistema remove o dado da base;

A3: O usuário pressiona o botão “Voltar”;

1. O sistema redireciona para a DV003;

A4: O usuário pressiona o para não confinar a exclusão;

1. O sistema fecha a caixa de confirmação e não realiza a alteração;

Fluxo de Exceções

E1: Não tem convites previamente cadastradas;

1. O sistema informa a mensagem “Você ainda não cadastrou um Convite”;

E2: O usuário não preencheu o campo “Email para a busca”;

1. O sistema informa a mensagem “O campo é obrigatório”;

E3: Não foi encontrado nenhum resultado para o evento informado;

1. O sistema exibe a mensagem “Usuário não encontrado”;

E4: O usuário preencheu o campo “Descrição” com mais de 150 caracteres;

1. O sistema exibe a mensagem “A descrição deve ter no máximo 150 caracteres”;

E5: O usuário gerou erro de validação no formulário;

1. O sistema desabilita o botão “Salvar”;

Regras de Negócio

R1: O sistema apresenta o botão “Aceitar” em listagem de convites;

1. O botão “Aceitar” só deve ser apresentado quando o convite esteja com o status pendente e o destinatário for o usuário autenticado do sistema;

R2: O sistema apresenta o botão “Enviar Convite” em cadastro de convite;

1. O botão “Enviar Convite” só deve ser apresentado quando a busca por e-mail de usuário seja realizada com sucesso;

R3: Aceitar convite de um usuário;

1. Ao aceitar o convite de um usuário, além de alterar o *status* do convite para aceito, o usuário destinatário é persistido na lista de moradores do imóvel relacionado ao convite;
2. Ao aceitar um convite também é verificado se o usuário em questão já não existe como morador no imóvel, sendo verdadeira essa condição o aceite do convite é ignorado.

Especificação de Caso de Uso

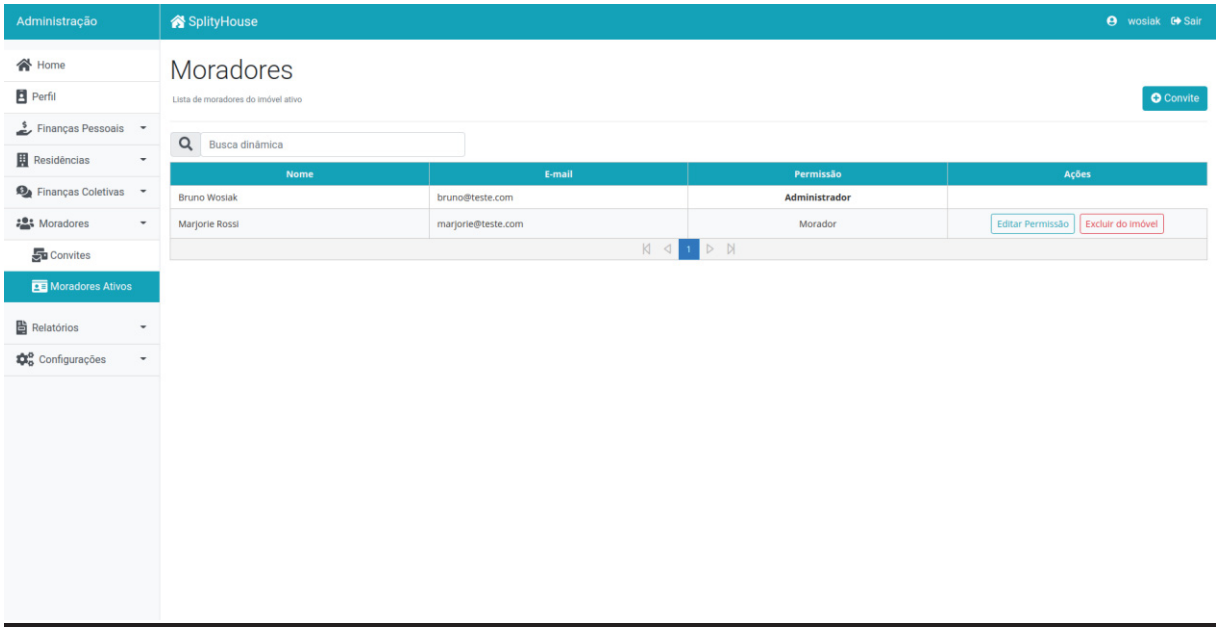
UC005 – Cadastro Moradores

Descrição

Este caso de uso serve para manter os moradores em um imóvel.

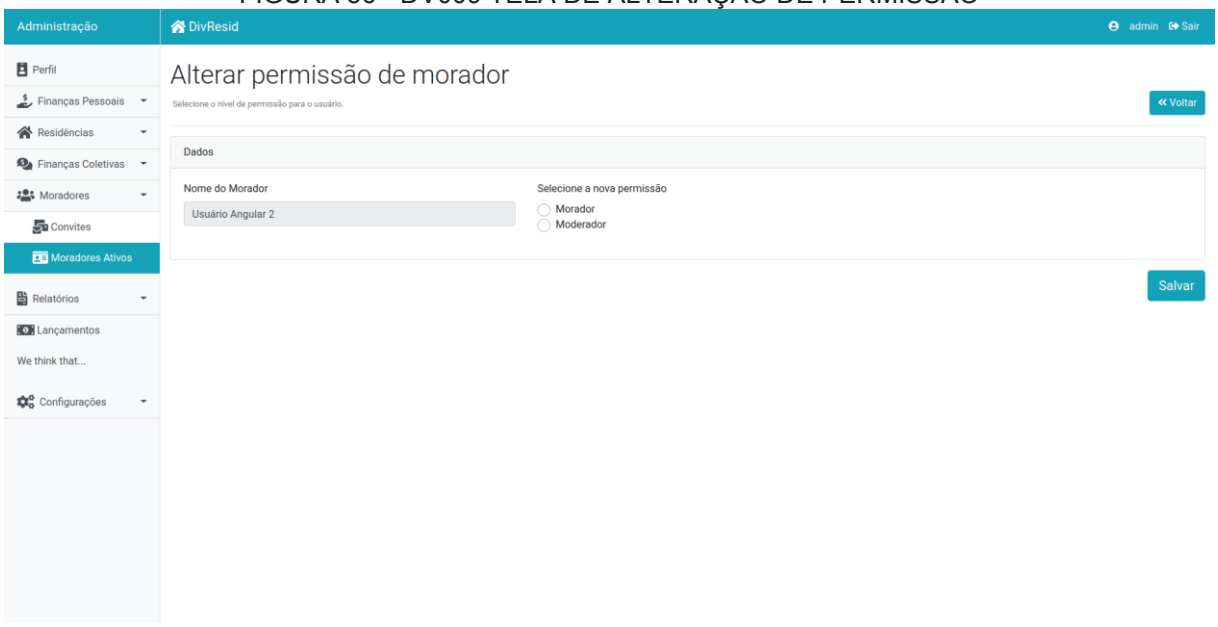
Data View

FIGURA 55 - DV008 TELA DE LISTAGEM DE MORADORES



FONTE: O Autor (2021)

FIGURA 56 - DV009 TELA DE ALTERAÇÃO DE PERMISSÃO



FONTE: O Autor (2021)

Ator Primário

Administrador

Pré-Condição

O usuário deve estar conectado à internet e ter cadastro válido no sistema, gerenciando um imóvel, ter moradores ativos em imóvel ativo e estar devidamente autenticado.

Fluxo de Eventos Principal

1. O usuário seleciona a opção “Moradores Ativos” no menu lateral
2. O sistema busca as informações dos moradores associados ao imóvel ativo cadastrado na base de dados **(E1) (E2)**;
3. O sistema apresenta a tela DV008;
4. O usuário pressiona o botão “Editar Permissão” **(A1) (A2)**;
5. O sistema busca as informações do morador selecionado;
6. O sistema apresenta a tela DV009;

7. O usuário seleciona a nova permissão do usuário;
8. O usuário pressiona o botão “Salvar” **(E3) (A3)**;
9. O sistema grava os dados na base;
10. O caso de uso é encerrado;

Fluxos Alternativos

A1: O usuário pressiona no botão “+Convite”;

1. O sistema chama o caso de uso UC004 – Cadastrar Convites;

A2: O usuário pressiona o botão “Excluir” em algum usuário listado;

1. O sistema apresenta a mensagem de confirmação “Deseja realmente excluir?”;
2. O usuário confirma a exclusão do dado **(A4)**;
3. O sistema remove o dado da base **(R1)**;

A3: O usuário pressiona o botão “Voltar”;

1. O sistema redireciona para a DV008;

A4: O usuário pressiona o para não confinar a exclusão;

1. O sistema fecha a caixa de confirmação e não realiza a alteração;

Fluxo de Exceções

E1: Não tem moradores associados ao imóvel ativo;

1. O sistema informa a mensagem “Você ainda não convidou moradores”;

E2: Não possui um imóvel ativo no momento;

1. O sistema informa a mensagem “Você não possui um imóvel ativo”;

E3: O usuário não selecionou a nova permissão;

1. O sistema informa a mensagem “O campo é obrigatório”;

Regras de Negócio

R1: Removendo usuário de um imóvel;

1. Ao remover um usuário de um imóvel a sua permissão é atualizada para administrador novamente;

Especificação de Caso de Uso

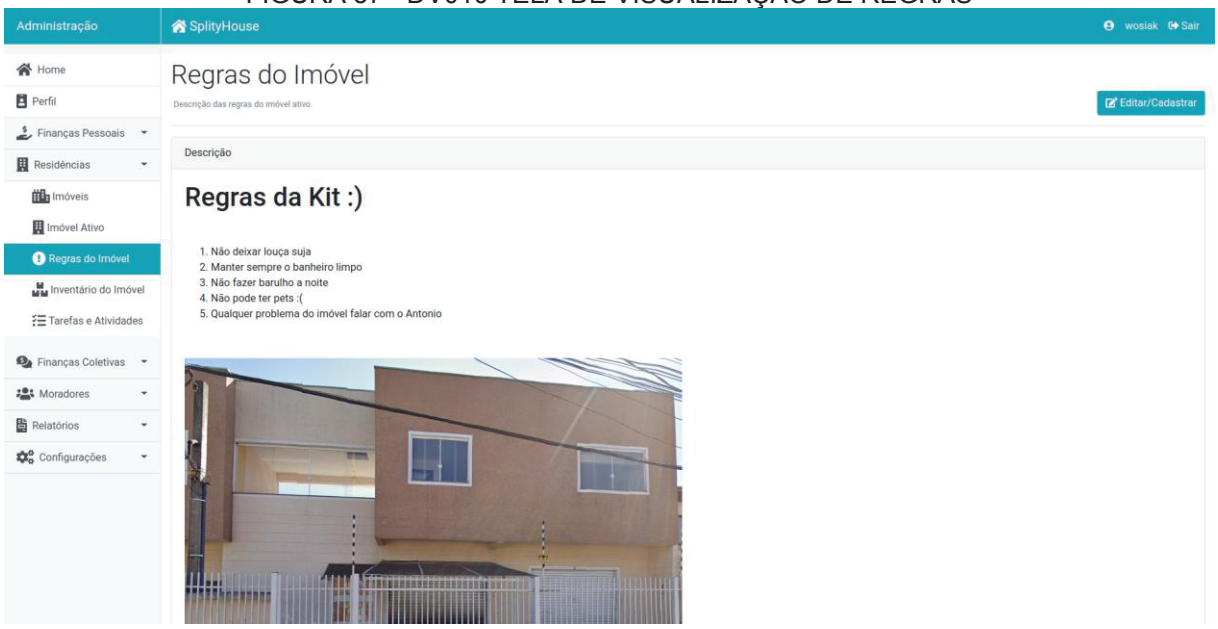
UC006 – Cadastrar Regras

Descrição

Este caso de uso serve para cadastrar regras do imóvel ativo.

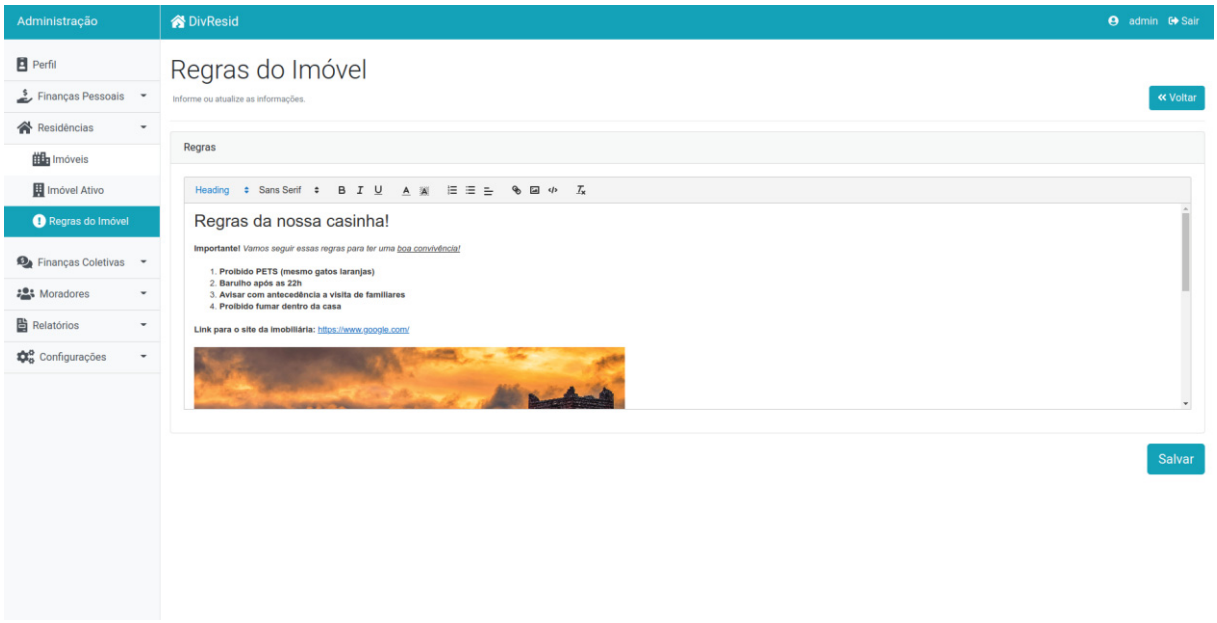
Data View

FIGURA 57 - DV010 TELA DE VISUALIZAÇÃO DE REGRAS



FONTE: O Autor (2021)

FIGURA 58 - DV011 TELA DE EDITAR OU CRIAR REGRAS



FONTE: O Autor (2021)

Ator Primário

Administrador

Pré-Condição

O usuário deve estar conectado à internet e ter cadastro válido no sistema, gerenciando um imóvel e estar devidamente autenticado.

Fluxo de Eventos Principal

1. O usuário seleciona a opção “Regras do Imóvel” no menu lateral
2. O sistema busca as informações do imóvel ativo cadastrado na base de dados **(E1) (E2)**;
3. O sistema apresenta a tela DV010;
4. O usuário pressiona o botão “Editar/Cadastrar”;
5. O sistema busca as informações da regra do imóvel ativo;
6. O sistema apresenta a tela DV011;
7. O usuário preenche o editor com as informações;

8. O usuário pressiona o botão “Salvar” **(A1)**;
9. O sistema grava os dados na base;
10. O caso de uso é encerrado;

Fluxos Alternativos

A1: O usuário pressiona o botão “Voltar”;

1. O sistema redireciona para a DV008;

Fluxo de Exceções

E1: O administrador não possui imóvel ativo;

1. O sistema informa a mensagem “Você não possui imóvel ativo”;

E2: Não possui uma regra cadastrado;

1. O sistema informa a mensagem “Você ainda não definiu as regras para esse imóvel!”;

Especificação de Caso de Uso

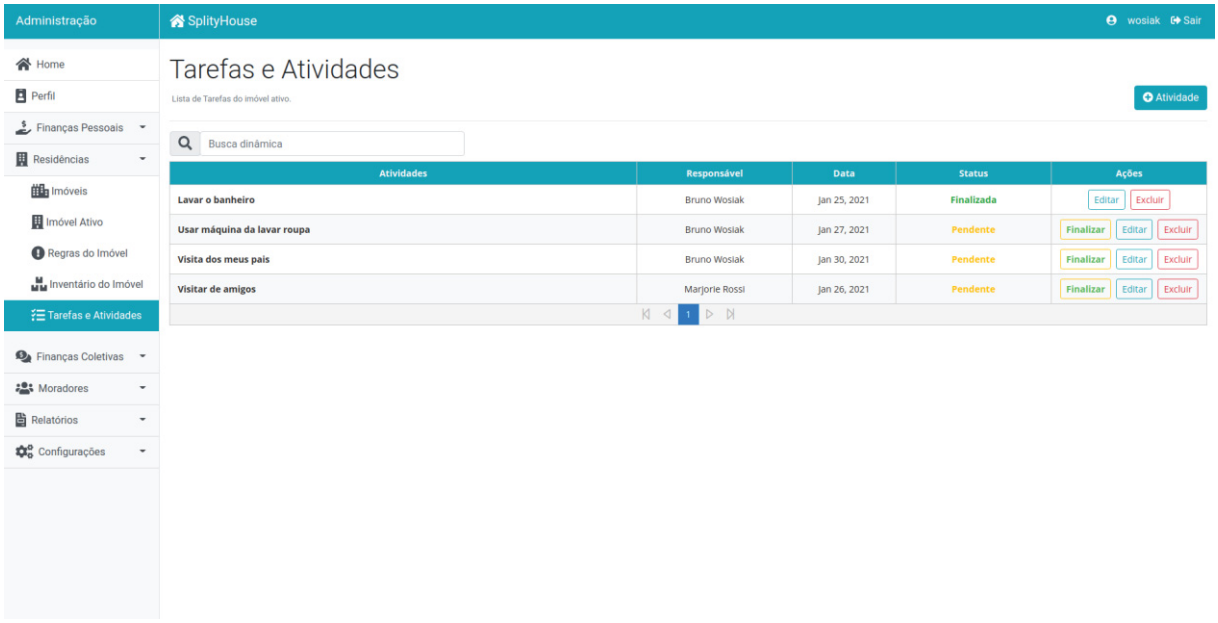
UC007 – Cadastrar Tarefas

Descrição

Este caso de uso possibilita o usuário a manter o cadastro de Tarefas e Atividades no sistema.

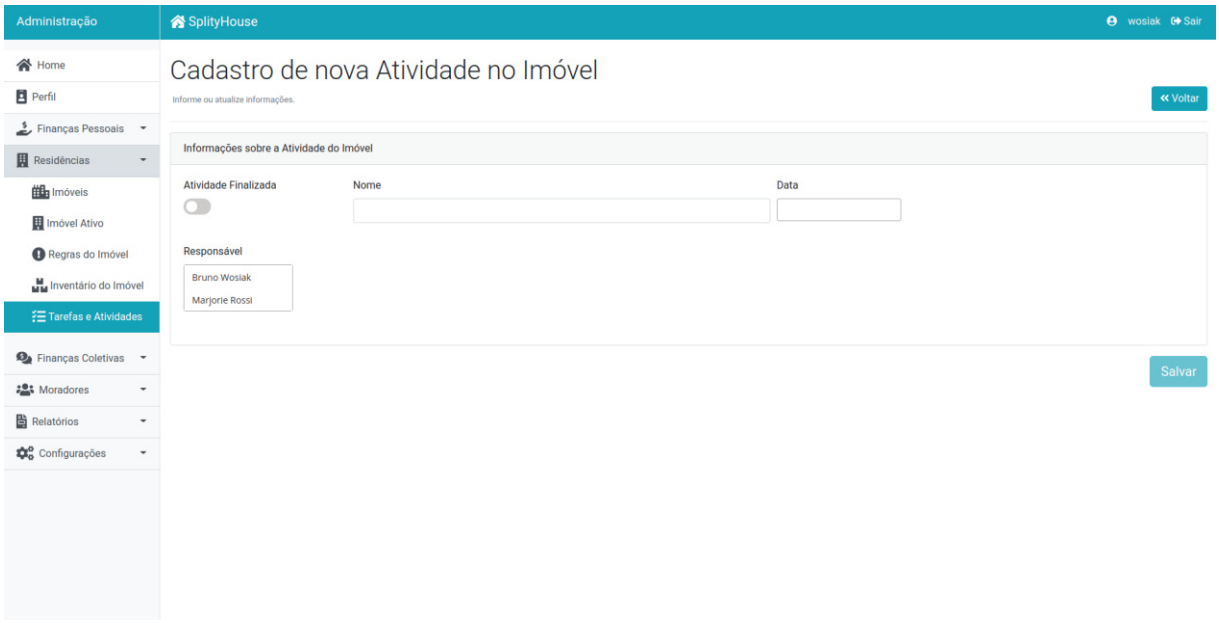
Data View

FIGURA 59 - DV0012 TELA DE LSITAGEM DE TAREFAS



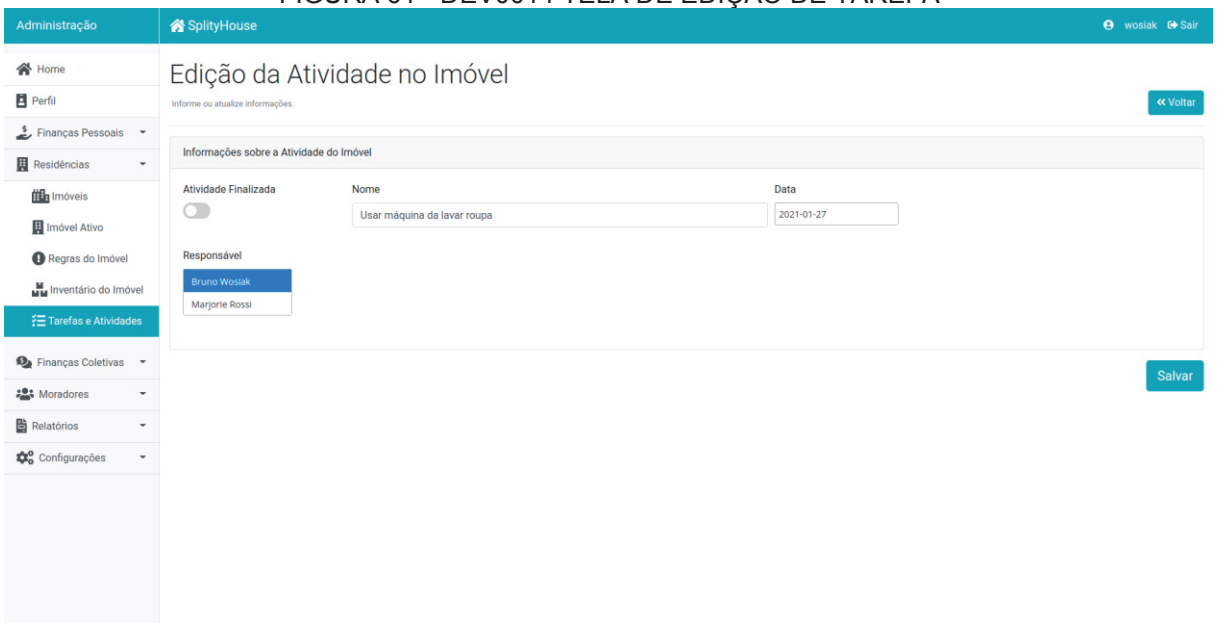
FONTE: O Autor (2021)

FIGURA 60 - DV0013 TELA DE CADASTRO DE TAREFAS



FONTE: O Autor (2021)

FIGURA 61 - DEV0014 TELA DE EDIÇÃO DE TAREFA



FONTE: O Autor (2021)

Ator Primário

Administrador

Pré-Condição

O usuário deve estar conectado à internet e ter cadastro válido no sistema, gerenciando um imóvel, ter moradores ativos em imóvel ativo e estar devidamente autenticado.

Fluxo de Eventos Principal

1. O usuário seleciona a opção “Tarefas e Atividades” no menu do sistema;
2. O sistema busca as informações das tarefas cadastradas na base de dados **(E1)**;
3. O sistema apresenta a tela DV012;
4. O usuário pressiona o botão “+Atividade” **(A1) (A2) (A3) (A4) (A5)**;
5. O sistema apresenta a tela DV013;
6. O usuário preenche o campo “Nome” **(E2) (E3)**;
7. O usuário preenche o campo “Data” **(E4)**;
8. O usuário seleciona um morador no campo “Responsável” **(E6)**;
9. O usuário pressiona o botão “Salvar” **(E5)**;
10. O sistema grava os dados na base;
11. O caso de uso é encerrado;

Fluxos Alternativos

A1: O usuário pressiona no botão “Editar” em alguma tarefa listada;

1. O sistema carrega as informações da Tarefa escolhida;
2. O sistema preenche os campos com os dados da Tarefa;
3. O sistema apresenta a tela DV014;
4. O usuário atualiza o dado do campo “Nome” **(E2) (E3)**;
5. O usuário atualiza o dado no campo “Data” **(E4)**;
6. O usuário seleciona um morador no campo “Responsável” **(E6)**;

7. O usuário pressiona o botão “Salvar” **(E5)**;
8. O sistema grava os dados na base;
9. O caso de uso é encerrado;

A2: O usuário pressiona o botão “Excluir” em alguma categoria listada;

1. O sistema apresenta a mensagem de confirmação “Deseja realmente excluir?”;
2. O usuário confirma a exclusão do dado **(A6)**;
3. O sistema remove o dado da base;

A3: O usuário pressiona o botão “Finalizar” em alguma tarefa listada;

1. O sistema altera o campo “Atividade Finalizada” da tarefa;
2. O sistema persiste a alteração na base de dados;

A4: O usuário preenche o campo de busca dinâmica;

1. O sistema busca na base dados que contenham o texto informado ();
2. O sistema preenche a tabela com os resultados;

A5: O usuário pressiona o botão “Voltar”;

1. O sistema redireciona para a DV008;

A6: O usuário pressiona o para não confinar a exclusão;

1. O sistema fecha a caixa de confirmação e não realiza a alteração;

Fluxo de Exceções

E1: Não tem tarefas previamente cadastradas;

1. O sistema informa a mensagem “Você ainda não cadastrou uma Tarefa”;

E2: O usuário não preencheu o campo “Nome”;

1. O sistema informa a mensagem “O campo é obrigatório”;

E3: O usuário preencheu o campo “Nome” com menos de 3 caracteres;

1. O sistema exibe a mensagem “O campo nome deve ter no mínimo 3 caracteres”;

E4: O usuário não preencheu o campo “Data”;

1. O sistema informa a mensagem “O campo é obrigatório”;

E5: O usuário gerou erro de validação no formulário;

1. O sistema desabilita o botão “Salvar”;

E6: O usuário não selecionou um morador no campo “Responsável”;

1. O sistema informa a mensagem “O campo é obrigatório”;

Especificação de Caso de Uso

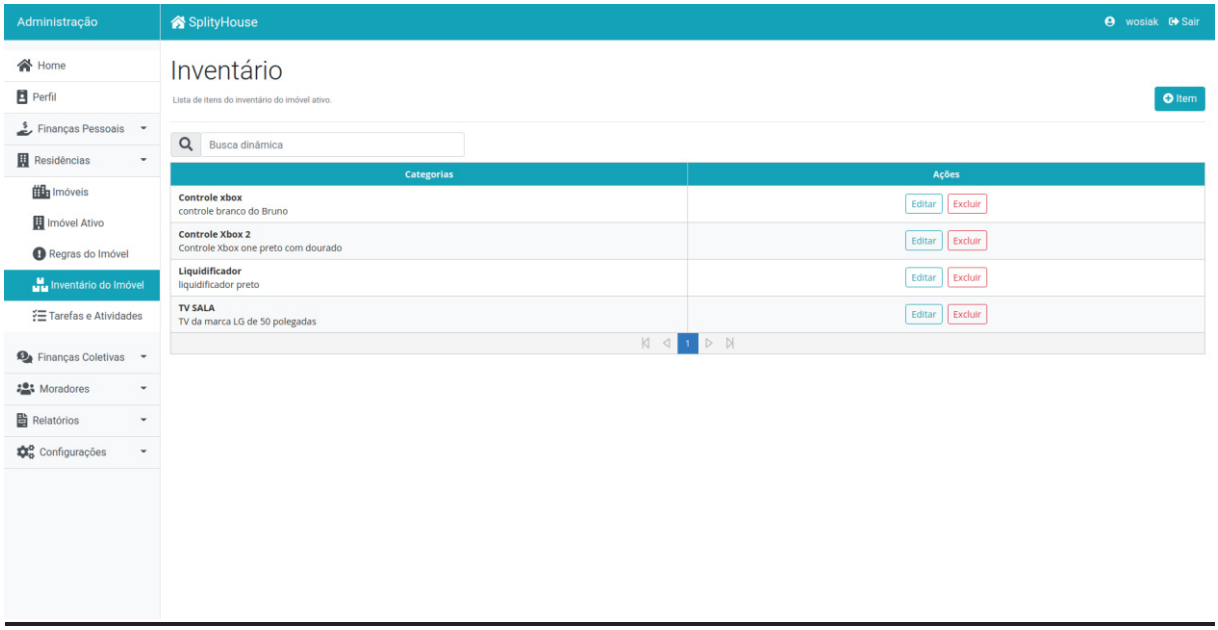
UC008 – Cadastrar Inventário

Descrição

Este caso de uso serve para cadastrar itens de inventário do imóvel ativo.

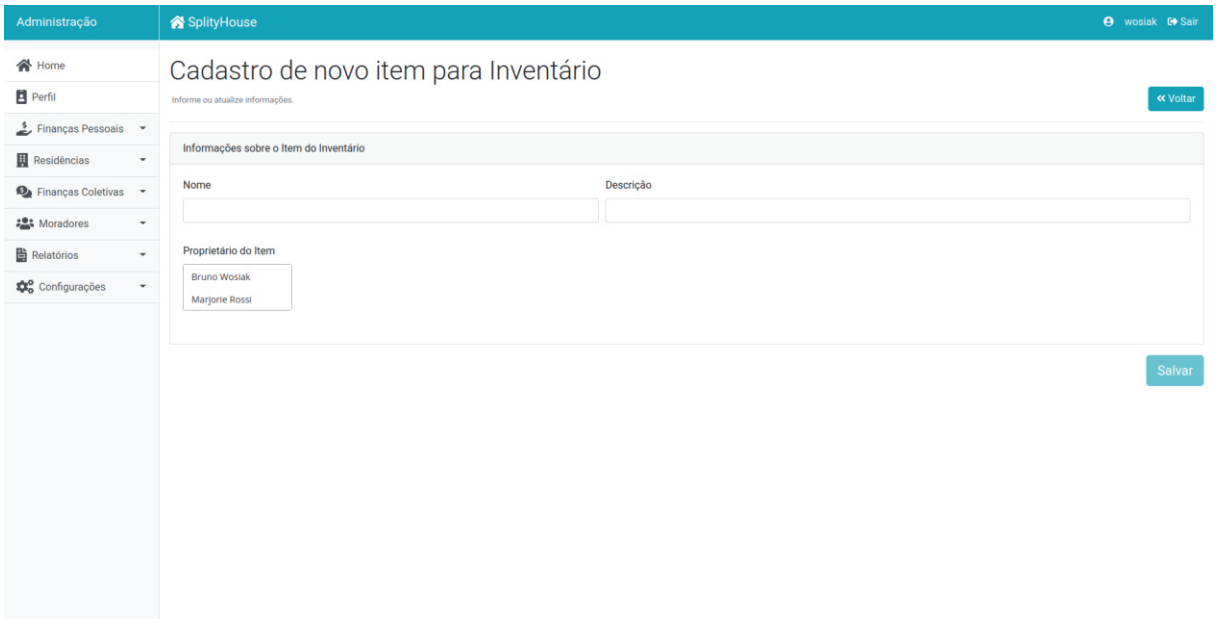
Data View

FIGURA 62 - DV0015 LISTAGEM DE ITENS DE INVENTÁRIO



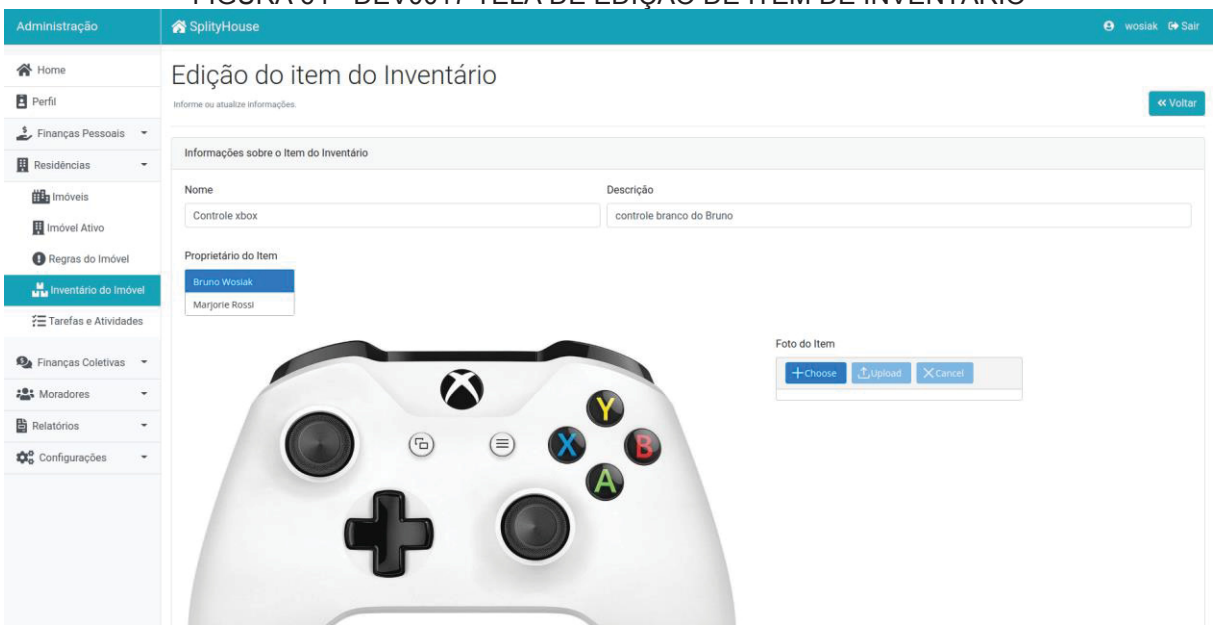
FONTE: O Autor (2021)

FIGURA 63 - DV0016 TELA DE CADASTRO DE ITEM DE INVENTÁRIO



FONTE: O Autor (2021)

FIGURA 64 - DEV0017 TELA DE EDIÇÃO DE ITEM DE INVENTÁRIO



FONTE: O Autor (2021)

Ator Primário

Administrador

Pré-Condição

O usuário deve estar conectado à internet e ter cadastro válido no sistema, gerenciando um imóvel e estar devidamente autenticado.

Fluxo de Eventos Principal

1. O usuário seleciona a opção “Inventário do Imóvel” no menu lateral
2. O sistema busca as informações dos itens do imóvel ativo cadastrado na base de dados **(E1)**;
3. O sistema apresenta a tela DV015;
4. O usuário pressiona o botão “+Item” **(A1) (A2)**;
5. O sistema apresenta a tela DV016;
6. O usuário preenche o campo “Nome” **(E2) (E3)**;
7. O usuário preenche o campo “Descrição” **(E4) (E5)**;
8. O usuário seleciona um morador no campo “Proprietário do Item” **(E6)**;
9. O usuário pressiona o botão “Salvar” **(E7)**;
10. O sistema grava os dados na base;
11. O caso de uso é encerrado;

Fluxos Alternativos

A1: O usuário pressiona no botão “Editar” em algum item listado;

1. O sistema carrega as informações do item escolhido;
2. O sistema preenche os campos com os dados da Item;
3. O sistema apresenta a tela DV017;
4. O sistema libera o botão de adição de foto “Foto do Item” **(R1)**;
5. O usuário atualiza o dado do campo “Nome” **(E2) (E3)**;
6. O usuário atualiza o dado no campo “Descrição” **(E4) (E5)**;
7. O usuário seleciona um morador no campo “Proprietário do Item” **(E6)**;

8. O usuário insere uma fotografia no campo “Foto do Item”;
9. O usuário pressiona o botão “Salvar” **(E7) (A4)**;
10. O sistema grava os dados na base;
11. O caso de uso é encerrado;

A2: O usuário pressiona o botão “Excluir” em algum item listado;

1. O sistema apresenta a mensagem de confirmação “Deseja realmente excluir?”;
2. O usuário confirma a exclusão do dado **(A5)**;
3. O sistema remove o dado da base;

A3: O usuário preenche o campo de busca dinâmica;

1. O sistema busca na base dados que contenham o texto informado;
2. O sistema preenche a tabela com os resultados;

A4: O usuário pressiona o botão “Voltar”;

1. O sistema redireciona para a DV015;

A5: O usuário pressiona o para não confinar a exclusão;

1. O sistema fecha a caixa de confirmação e não realiza a alteração;

Fluxo de Exceções

E1: Não tem itens previamente cadastrados;

1. O sistema informa a mensagem “Você ainda não cadastrou nenhum item”;

E2: O usuário não preencheu o campo “Nome”;

1. O sistema informa a mensagem “O campo é obrigatório”;

E3: O usuário preencheu o campo “Nome” com menos de 3 caracteres;

1. O sistema exibe a mensagem “O campo nome deve ter no mínimo 3 caracteres”;

E4: O usuário não preencheu o campo “Descrição”;

1. O sistema informa a mensagem “O campo é obrigatório”;

E5: O usuário preencheu o campo “Descrição” com menos de 3 caracteres;

1. O sistema exibe a mensagem “O campo nome deve ter no mínimo 3 caracteres”;

E6: O usuário não selecionou um morador no campo “Proprietário do Item”;

1. O sistema informa a mensagem “O campo é obrigatório”;

E7: O usuário gerou erro de validação no formulário;

1. O sistema desabilita o botão “Salvar”;

Regras de Negócio

R1: Inclusão de foto ao item de inventário do imóvel;

1. A adição de foto só é liberada na tela de edição após o item já estar previamente cadastrado;

Especificação de Caso de Uso

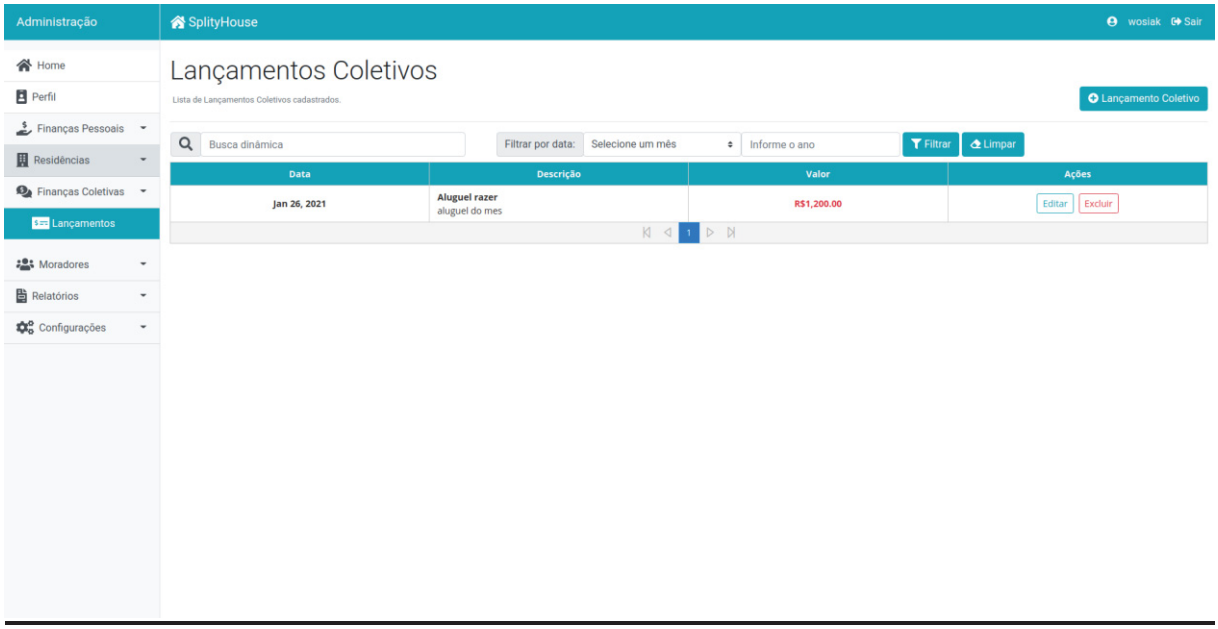
UC009 – Cadastrar Despesas Coletivas

Descrição

Este caso de uso serve para cadastrar despesas coletivas para os moradores do imóvel ativo.

Data View

FIGURA 65 - DV0018 LISTAGEM DE DESPESAS COLETIVAS

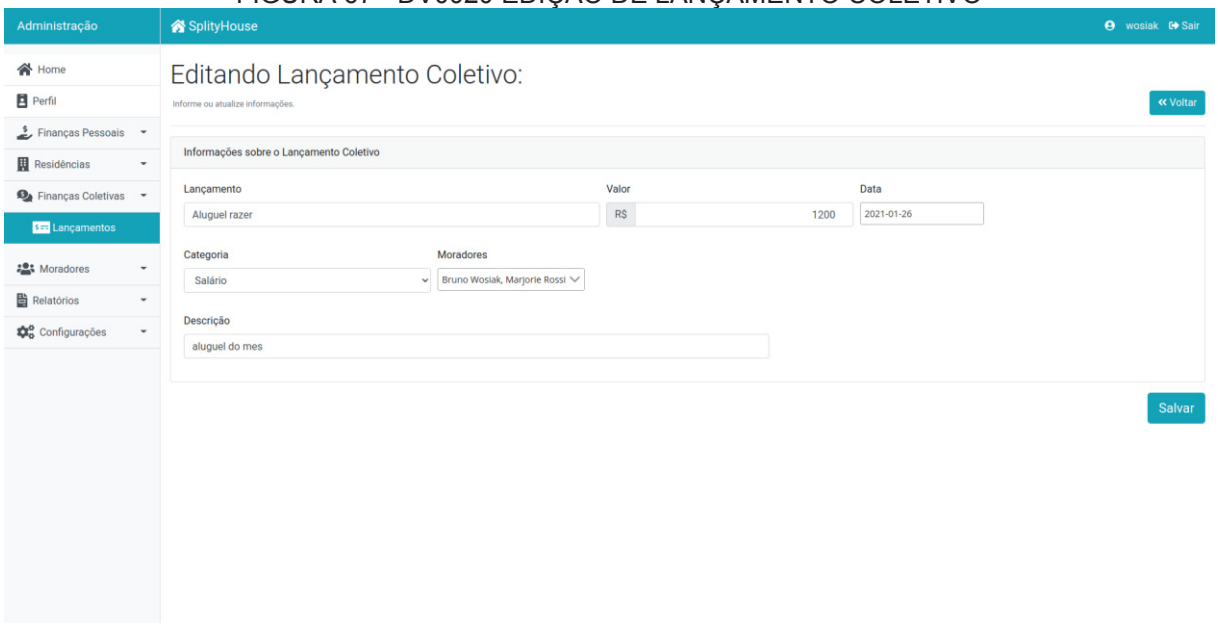


FONTE: O Autor (2021)

FIGURA 66 - DV0019 CADASTRO DE LANÇAMENTO COLETIVO

FONTE: O Autor (2021)

FIGURA 67 - DV0020 EDIÇÃO DE LANÇAMENTO COLETIVO



FONTE: O Autor (2021)

Ator Primário

Administrador

Pré-Condição

O usuário deve estar conectado à internet e ter cadastro válido no sistema, gerenciando um imóvel, ter moradores ativos em imóvel ativo e estar devidamente autenticado.

Fluxo de Eventos Principal

1. O usuário seleciona a opção Lançamentos” no menu lateral
2. O sistema busca as informações dos lançamentos coletivos do imóvel ativo cadastrado na base de dados **(E1)**;
3. O sistema apresenta a tela DV018;
4. O usuário pressiona o botão “+Lançamento Coletivo” **(A1) (A2) (A3) (A4)**;
5. O sistema apresenta a tela DV019;
6. O usuário preenche o campo “Lançamento” **(E2) (E3)**;
7. O usuário preenche o campo “Valor” **(E4)**;
8. O usuário preenche o campo “Data” **(E5)**;
9. O usuário seleciona o campo “Categoria” **(E6)**;
10. O usuário seleciona os moradores no campo “Moradores” **(E7)**;
11. O usuário preenche o campo “Descrição” **(E8) (E9)**;
12. O usuário pressiona o botão “Salvar” **(E10)**;
13. O sistema grava os dados na base;
14. O caso de uso é encerrado;

Fluxos Alternativos

A1: O usuário pressiona no botão “Editar” em algum lançamento coletivo listado;

1. O sistema carrega as informações do item escolhido;
2. O sistema preenche os campos com os dados da Item;
3. O sistema apresenta a tela DV020;

4. O usuário atualiza o dado do campo “Lançamento” **(E2) (E3)**;
5. O usuário atualiza o dado no campo “Valor” **(E4)**;
6. O usuário atualiza o dado no campo “Data” **(E5)**;
7. O usuário atualiza o dado no campo “Categoria” **(E6)**;
8. O usuário atualiza a seleção de moradores no campo “Moradores” **(E7)**;
9. O usuário atualiza o dado no campo “Descrição” **(E8) (E9)**;
10. O usuário pressiona o botão “Salvar” **(E10) (A5)**;
11. O sistema grava os dados na base;
12. O caso de uso é encerrado;

A2: O usuário pressiona o botão “Excluir” em algum lançamento coletivo listado;

1. O sistema apresenta a mensagem de confirmação “Deseja realmente excluir?”;
2. O usuário confirma a exclusão do dado **(A6)**;
3. O sistema remove o dado da base;

A3: O usuário preenche o campo de busca dinâmica;

1. O sistema busca na base dados que contenham o texto informado;
2. O sistema preenche a tabela com os resultados;

A4: O usuário preenche o campo de busca por filtro;

1. O usuário seleciona o mês no campo “Seleciona o mês” **(E11)**;
2. O usuário preenche o campo “Informe o ano” **(E12)**;
3. O sistema busca na base dados que contenham o texto informado;
4. O sistema preenche a tabela com os resultados;

A5: O usuário pressiona o botão “Voltar”;

1. O sistema redireciona para a DV018;

A6: O usuário pressiona o para não confinar a exclusão;

1. O sistema fecha a caixa de confirmação e não realiza a alteração;

Fluxo de Exceções

E1: Não tem lançamentos coletivos previamente cadastrados;

1. O sistema informa a mensagem “Você ainda não cadastrou nenhum lançamento coletivo”;

E2: O usuário não preencheu o campo “Lançamento”;

1. O sistema informa a mensagem “O campo é obrigatório”;

E3: O usuário preencheu o campo “Lançamento” com menos de 3 caracteres;

1. O sistema exibe a mensagem “O campo Lançamento deve ter no mínimo 3 caracteres”;

E4: O usuário não preencheu o campo “Valor”;

1. O sistema informa a mensagem “O campo é obrigatório”;

E5: O usuário não preencheu o campo “Data”;

1. O sistema informa a mensagem “O campo é obrigatório”;

E6: O usuário não selecionou uma categoria no campo “Categoria”;

1. O sistema informa a mensagem “O campo é obrigatório”;

E7: O usuário não selecionou os moradores no campo “Moradores”;

1. O sistema informa a mensagem “O campo é obrigatório”;

E8: O usuário não preencheu campo “Descrição”;

1. O sistema informa a mensagem “O campo é obrigatório”;

E9: O usuário preencheu o campo “Descrição” com menos de 3 caracteres;

1. O sistema informa a mensagem “O campo é obrigatório”;

E10: O usuário gerou erro de validação no formulário;

1. O sistema desabilita o botão “Salvar”;

E11: O usuário não selecionou o campo do filtro por data “Selecione o mês”;

1. O sistema informa a mensagem “O campo é obrigatório”;

E12: O usuário não preencheu o campo do filtro por data “informe o ano”;

1. O sistema informa a mensagem “O campo é obrigatório”;

Especificação de Caso de Uso

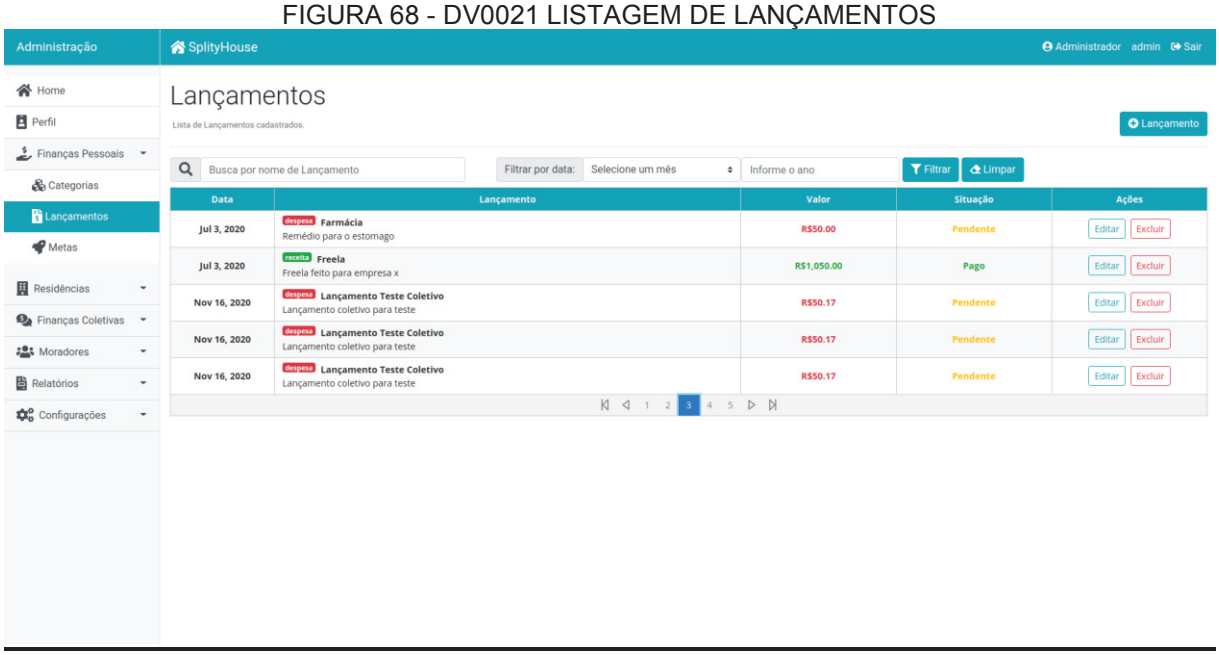
UC010 – Cadastrar Despesas Pessoais

Descrição

Este caso de uso serve para cadastrar despesas do usuário.

Data View

FIGURA 68 - DV0021 LISTAGEM DE LANÇAMENTOS



FONTE: O Autor (2021)

FIGURA 69 - DV0022 CADASTRO DE LANÇAMENTO

FONTE: O Autor (2021)

FIGURA 70 - DV0023 EDITAR LANÇAMENTO

FONTE: O Autor (2021)

Ator Primário

Administrador

Pré-Condição

O usuário deve estar conectado à internet e ter cadastro válido no sistema e estar devidamente autenticado.

Fluxo de Eventos Principal

1. O usuário seleciona a opção Lançamentos” no menu lateral
2. O sistema busca as informações dos lançamentos coletivos do imóvel ativo cadastrado na base de dados **(E1)**;
3. O sistema apresenta a tela DV021;
4. O usuário pressiona o botão “+Lançamento” **(A1) (A2) (A3) (A4)**;
5. O sistema apresenta a tela DV022;
6. O usuário seleciona o campo “Tipo”;
7. O usuário preenche o campo “Lançamento” **(E2) (E3)**;
8. O usuário preenche o campo “Valor” **(E4)**;
9. O usuário preenche o campo “Data” **(E5)**;
10. O usuário seleciona o campo “Pago”;
11. O usuário seleciona o campo “Categoria” **(E6)**;
12. O usuário preenche o campo “Descrição” **(E7) (E8)**;
13. O usuário pressiona o botão “Salvar” **(E9)**;
14. O sistema grava os dados na base;
15. O caso de uso é encerrado;

Fluxos Alternativos

A1: O usuário pressiona no botão “Editar” em algum lançamento listado;

1. O sistema carrega as informações do item escolhido;
2. O sistema preenche os campos com os dados da Item;
3. O sistema apresenta a tela DV023;

4. O usuário atualiza a seleção no campo “Tipo”;
5. O usuário atualiza o dado no campo “Lançamento” **(E2) (E3)**;
6. O usuário atualiza o dado no campo “Valor” **(E4)**;
7. O usuário atualiza o dado no campo “Data” **(E5)**;
8. O usuário atualiza a seleção no campo “Pago”;
9. O usuário atualiza a seleção no campo “Categoria” **(E6)**;
10. O usuário atualiza o dado no campo “Descrição” **(E7) (E8)**;
11. O usuário pressiona o botão “Salvar” **(E9) (A5)**;
12. O sistema grava os dados na base;
13. O caso de uso é encerrado;

A2: O usuário pressiona o botão “Excluir” em algum lançamento listado;

1. O sistema apresenta a mensagem de confirmação “Deseja realmente excluir?”;
2. O usuário confirma a exclusão do dado **(A6)**;
3. O sistema remove o dado da base;

A3: O usuário preenche o campo de busca dinâmica;

1. O sistema busca na base dados que contenham o texto informado;
2. O sistema preenche a tabela com os resultados;

A4: O usuário preenche o campo de busca por filtro;

1. O usuário seleciona o mês no campo “Seleciona o mês” **(E10)**;
2. O usuário preenche o campo “Informe o ano” **(E11)**;
3. O sistema busca na base dados que contenham o texto informado;
4. O sistema preenche a tabela com os resultados;

A5: O usuário pressiona o botão “Voltar”;

1. O sistema redireciona para a DV021;

A6: O usuário pressiona o para não confinar a exclusão;

1. O sistema fecha a caixa de confirmação e não realiza a alteração;

Fluxo de Exceções

E1: Não tem lançamentos coletivos previamente cadastrados;

1. O sistema informa a mensagem “Você ainda não cadastrou nenhum lançamento coletivo”;

E2: O usuário não preencheu o campo “Lançamento”;

1. O sistema informa a mensagem “O campo é obrigatório”;

E3: O usuário preencheu o campo “Lançamento” com menos de 3 caracteres;

1. O sistema exibe a mensagem “O campo Lançamento deve ter no mínimo 3 caracteres”;

E4: O usuário não preencheu o campo “Valor”;

1. O sistema informa a mensagem “O campo é obrigatório”;

E5: O usuário não preencheu o campo “Data”;

1. O sistema informa a mensagem “O campo é obrigatório”;

E6: O usuário não selecionou uma categoria no campo “Categoria”;

1. O sistema informa a mensagem “O campo é obrigatório”;

E7: O usuário não preencheu campo “Descrição”;

1. O sistema informa a mensagem “O campo é obrigatório”;

E8: O usuário preencheu o campo “Descrição” com menos de 3 caracteres;

1. O sistema informa a mensagem “O campo é obrigatório”;

E9: O usuário gerou erro de validação no formulário;

1. O sistema desabilita o botão “Salvar”;

E10: O usuário não selecionou o campo do filtro por data “Selecione o mês”;

1. O sistema informa a mensagem “O campo é obrigatório”;

E11: O usuário não preencheu o campo do filtro por data “informe o ano”;

1. O sistema informa a mensagem “O campo é obrigatório”;

Especificação de Caso de Uso

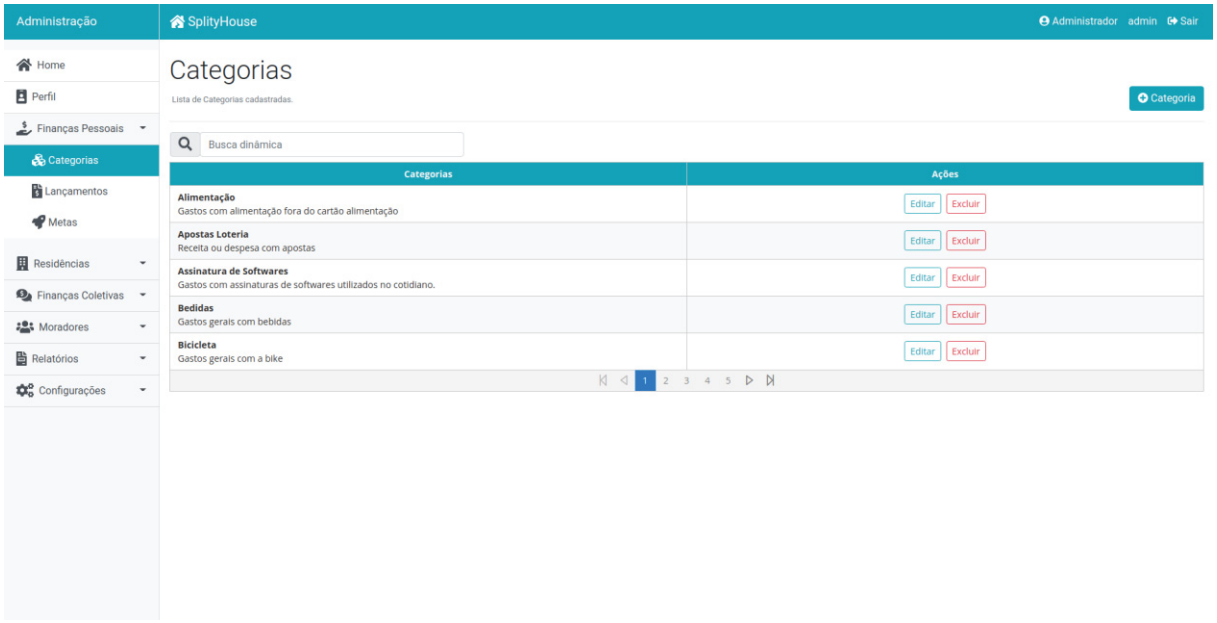
UC011 – Cadastro de Categorias

Descrição

Este caso de uso possibilita o usuário a manter o cadastro de Categorias no sistema.

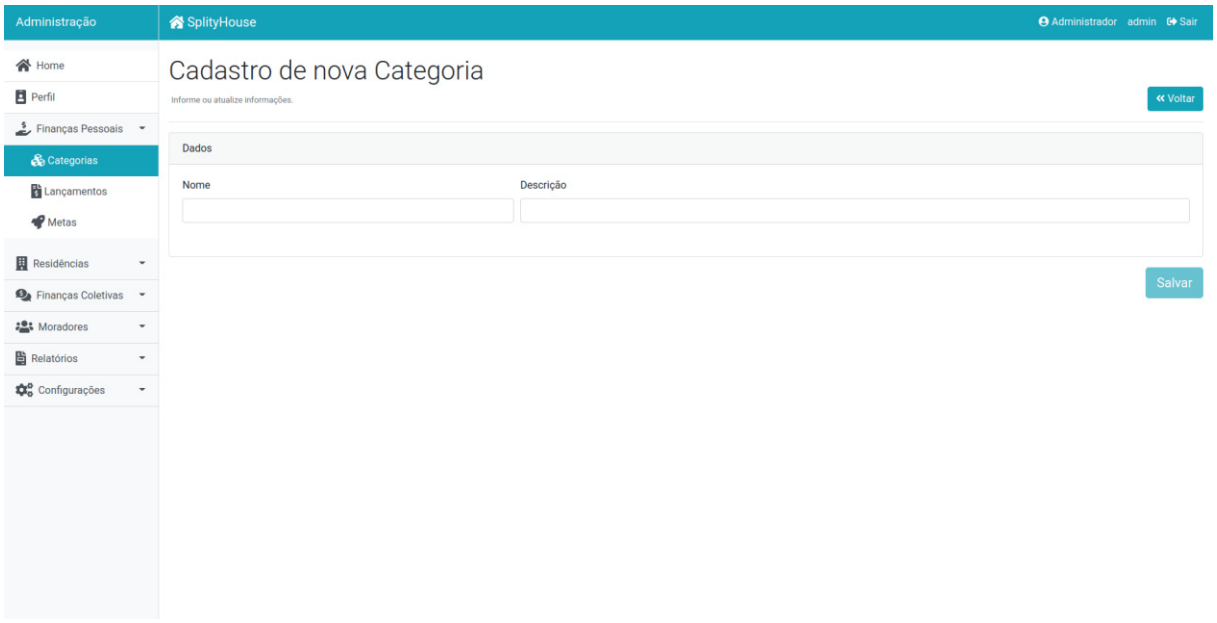
Data View

FIGURA 71 - DV012 LISTAGEM DE CATEGORIAS



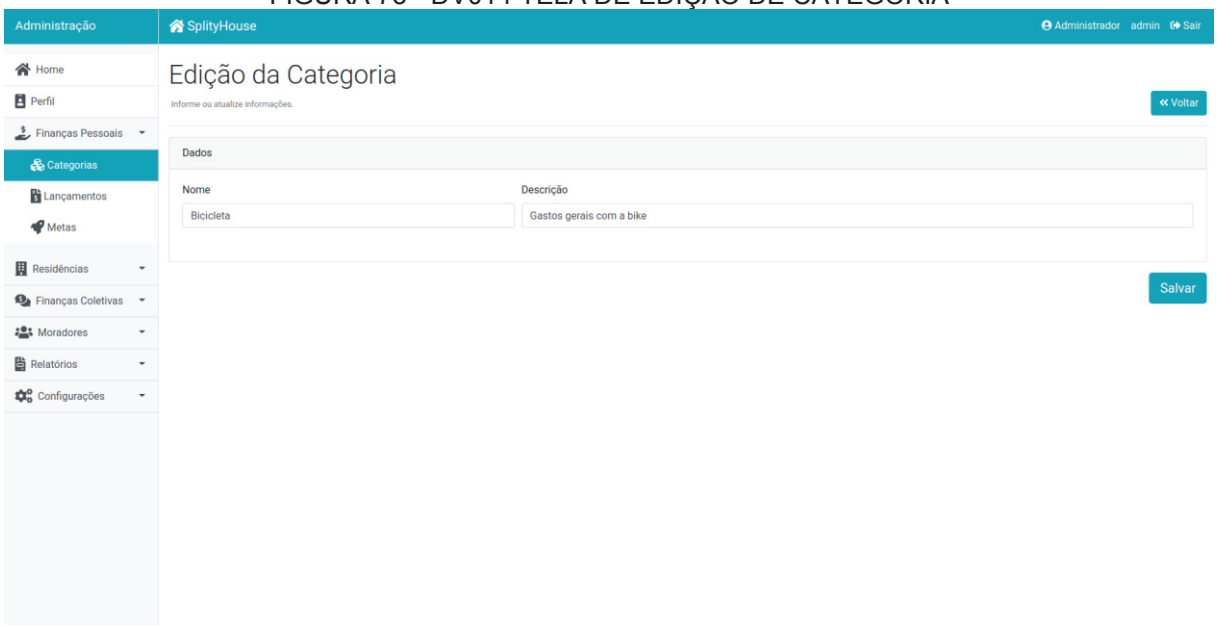
FONTE: O Autor (2021)

FIGURA 72 - DV013 CADASTRO DE CATEGORIA



FONTE: O Autor (2021)

FIGURA 73 - DV014 TELA DE EDIÇÃO DE CATEGORIA



FONTE: O Autor (2021)

Ator Primário

Administrador

Pré-Condição

O usuário deve estar conectado à internet e ter cadastro válido no sistema e estar devidamente autenticado.

Fluxo de Eventos Principal

1. O usuário seleciona a opção “Categorias” no menu do sistema;
2. O sistema busca as informações das categorias cadastradas na base de dados **(E1)**;
3. O sistema apresenta a tela DV008;
4. O usuário pressiona o botão “+Categoria” **(A1) (A2) (A3)**;
5. O sistema apresenta a tela DV009;
6. O usuário preenche o campo “Nome” **(E2) (E3)**;
7. O usuário preenche o campo “Descrição” **(E4)**;
8. O usuário pressiona o botão “Salvar” **(E5)**;
9. O sistema grava os dados na base;
10. O caso de uso é encerrado;

Fluxos Alternativos

A1: O usuário pressiona no botão “Editar” em alguma categoria listada;

1. O sistema carrega as informações da Categoria escolhida;
2. O sistema preenche os campos com os dados da Categoria;
3. O sistema apresenta a tela DV010;
4. O usuário atualiza o dado do campo “Nome” **(E2) (E3)**;
5. O usuário atualiza o dado no campo “Descrição” **(E4)**;
6. O usuário pressiona o botão “Salvar” **(E5) (A4)**;
7. O sistema grava os dados na base;
8. O caso de uso é encerrado;

A2: O usuário pressiona o botão “Excluir” em alguma categoria listada;

1. O sistema apresenta a mensagem de confirmação “Deseja realmente excluir?”;
2. O usuário confirma a exclusão do dado **(A5)**;
3. O sistema remove o dado da base;

A3: O usuário preenche o campo de busca dinâmica;

1. O sistema busca na base dados que contenham o texto informado ();
2. O sistema preenche a tabela com os resultados;

A4: O usuário pressiona o botão “Voltar”;

1. O sistema redireciona para a DV008;

A5: O usuário pressiona o para não confinar a exclusão;

1. O sistema fecha a caixa de confirmação e não realiza a alteração;

Fluxo de Exceções

E1: Não tem categorias previamente cadastradas;

1. O sistema informa a mensagem “Você ainda não cadastrou uma Categoria”;

E2: O usuário não preencheu o campo “Nome”;

1. O sistema informa a mensagem “O campo é obrigatório”;

E3: O usuário preencheu o campo “Nome” com menos de 3 caracteres;

1. O sistema exibe a mensagem “O campo nome deve ter no mínimo 3 caracteres”;

E4: O usuário preencheu o campo “Descrição” com mais de 150 caracteres;

1. O sistema exibe a mensagem “A descrição deve ter no máximo 150 caracteres”;

E5: O usuário gerou erro de validação no formulário;

1. O sistema desabilita o botão “Salvar”;

Especificação de Caso de Uso

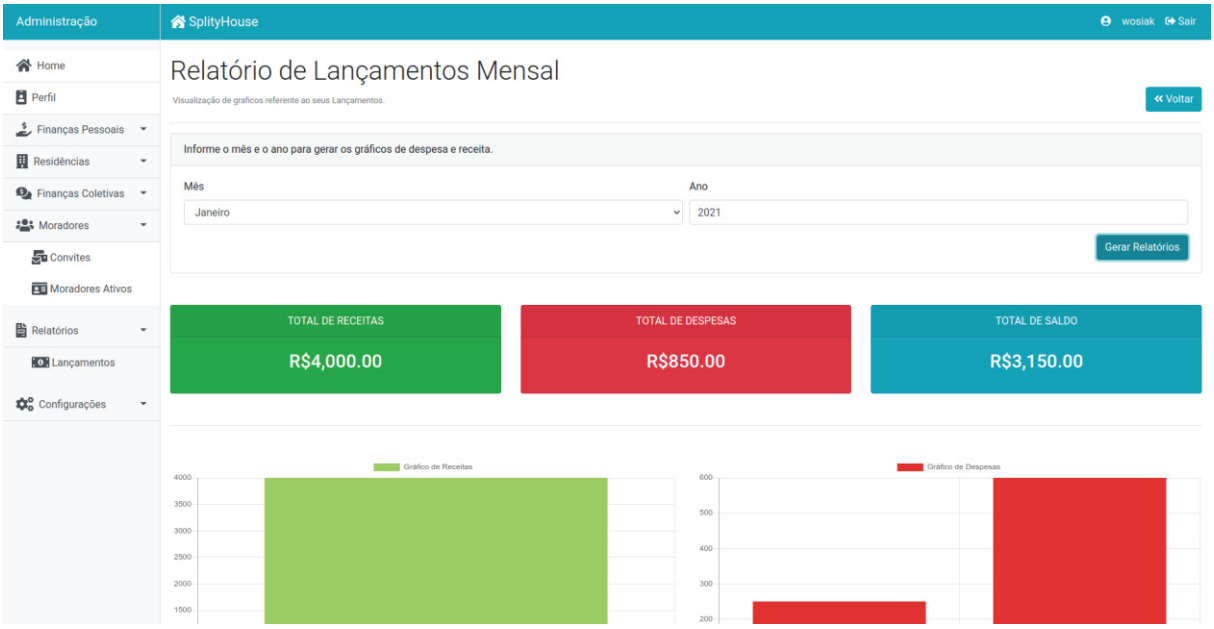
UC011 – Cadastro de Categorias

Descrição

Este caso de uso possibilita o usuário a manter o cadastro de Categorias no sistema.

Data View

FIGURA 74 - DV015 TELA DE RELATÓRIO



FONTE: O Autor (2021)

Ator Primário

Administrador

Pré-Condição

O usuário deve estar conectado à internet e ter cadastro válido no sistema e estar devidamente autenticado.

Fluxo de Eventos Principal

1. O usuário seleciona a opção “Relatórios” no menu do sistema;
2. O sistema apresenta a tela DV015;
3. O usuário seleciona a opção “Mês”;
4. O usuário preenche o campo “Ano” **(E1)**;
5. O usuário pressiona o botão “Gerar Relatórios” **(A1)**;
6. O sistema buscas as informações na base;
7. O sistema preenche e atualiza a tela DV015 com os dados;
8. O caso de uso é encerrado;

Fluxos Alternativos

A1: O usuário pressiona no botão “Voltar” em alguma categoria listada;

1. O caso de uso é encerrado;

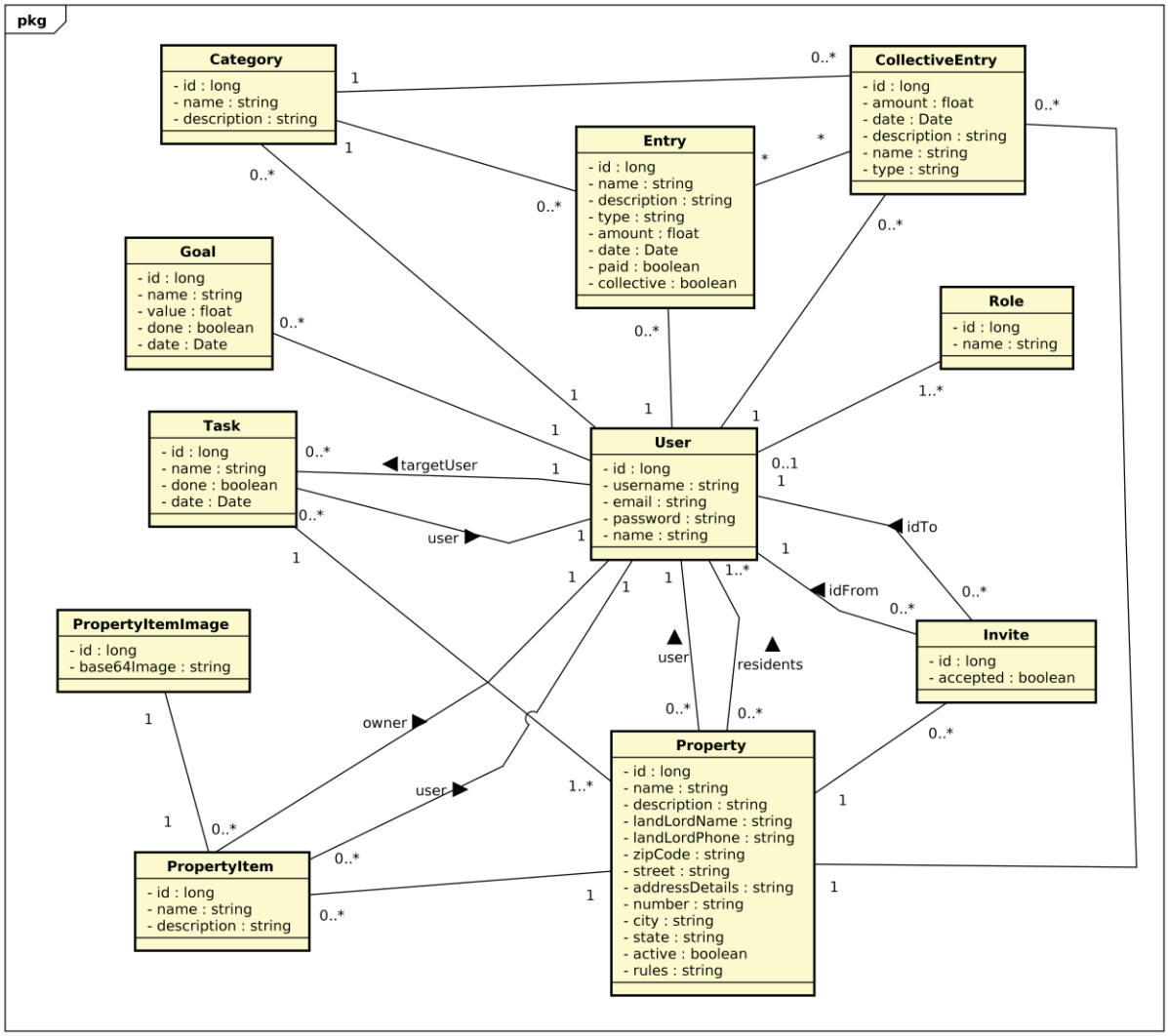
Fluxo de Exceções

E1: O usuário não preencheu o campo “Ano”;

1. O sistema informa a mensagem “O campo é obrigatório”;

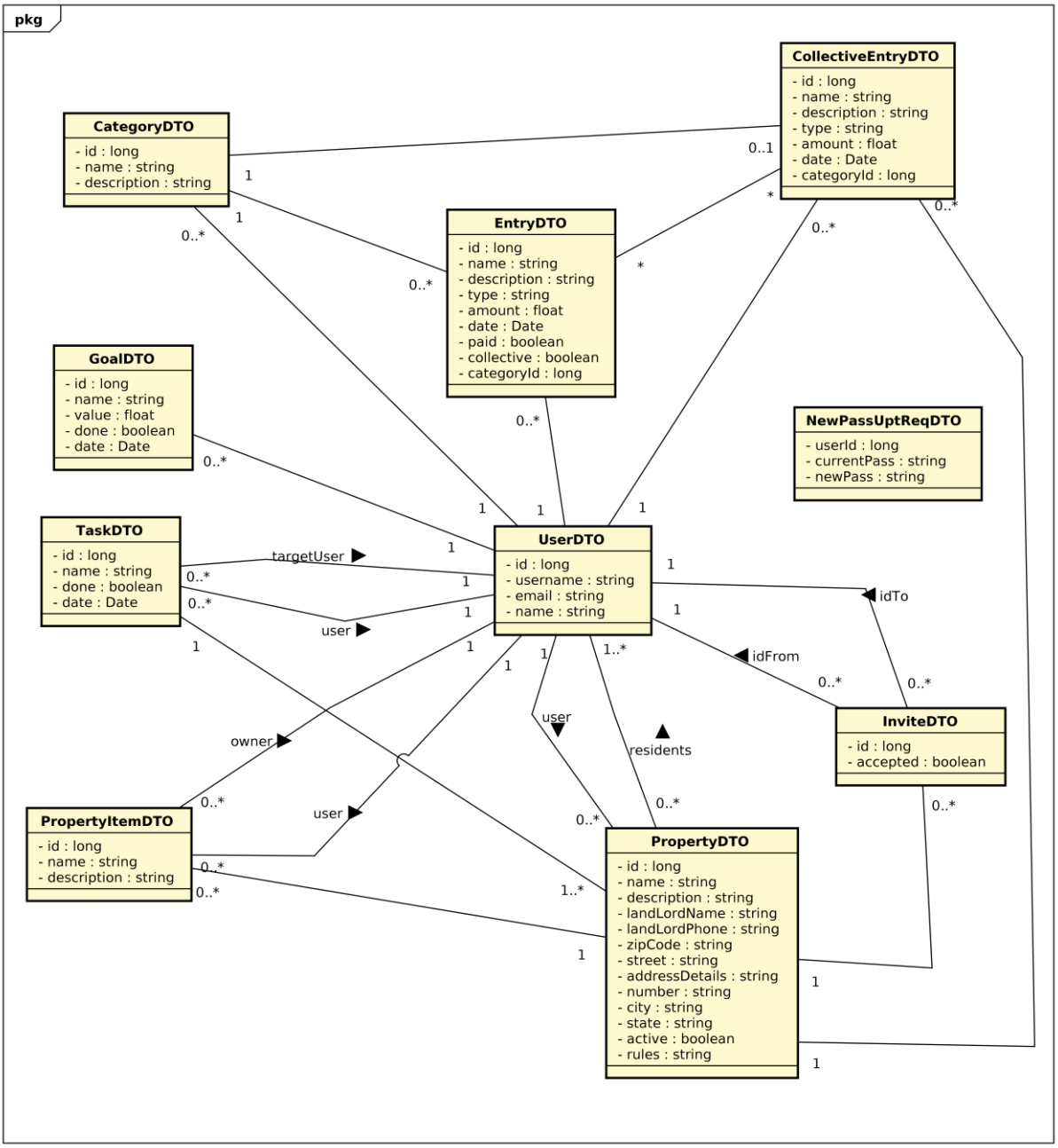
APÊNDICE C – DIAGRAMAS DE CLASSES

FIGURA 75 - DIAGRAMA DE CLASSES BACKEND



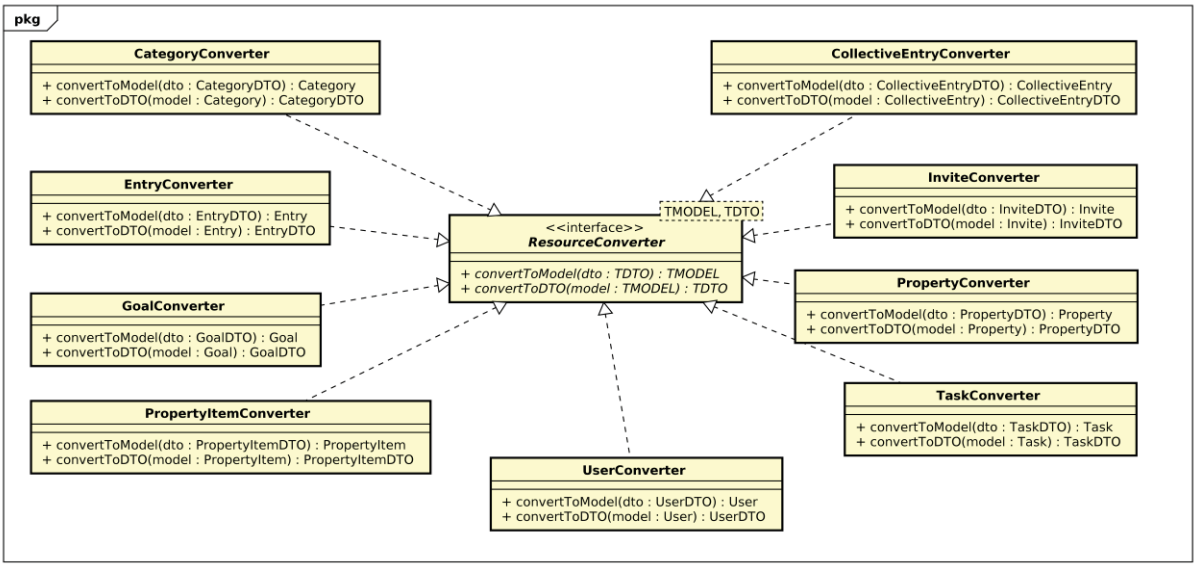
FONTE: O Autor (2021)

FIGURA 76 - DIAGRAMA DE CLASSES DE TRANSFERÊNCIA BACKEND



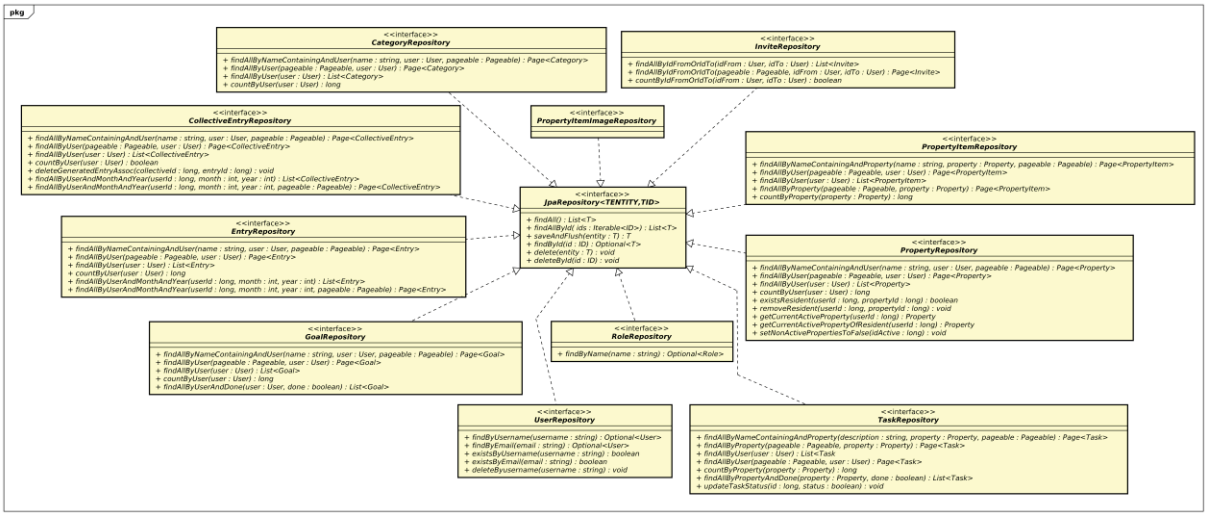
FONTE: O Autor (2021)

FIGURA 77 - DIAGRAMA DE CLASSES CONVERSORAS BACKEND



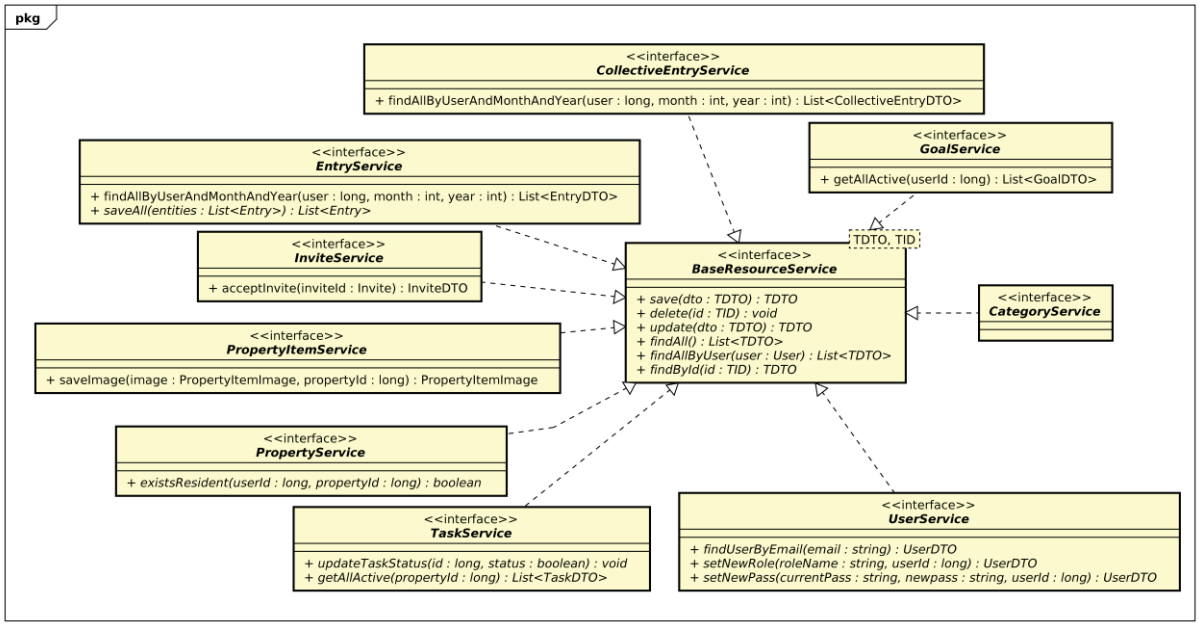
FONTE: O Autor (2021)

FIGURA 78 - DIAGRAMA DE CLASSES REPOSITÓRIOS BACKEND



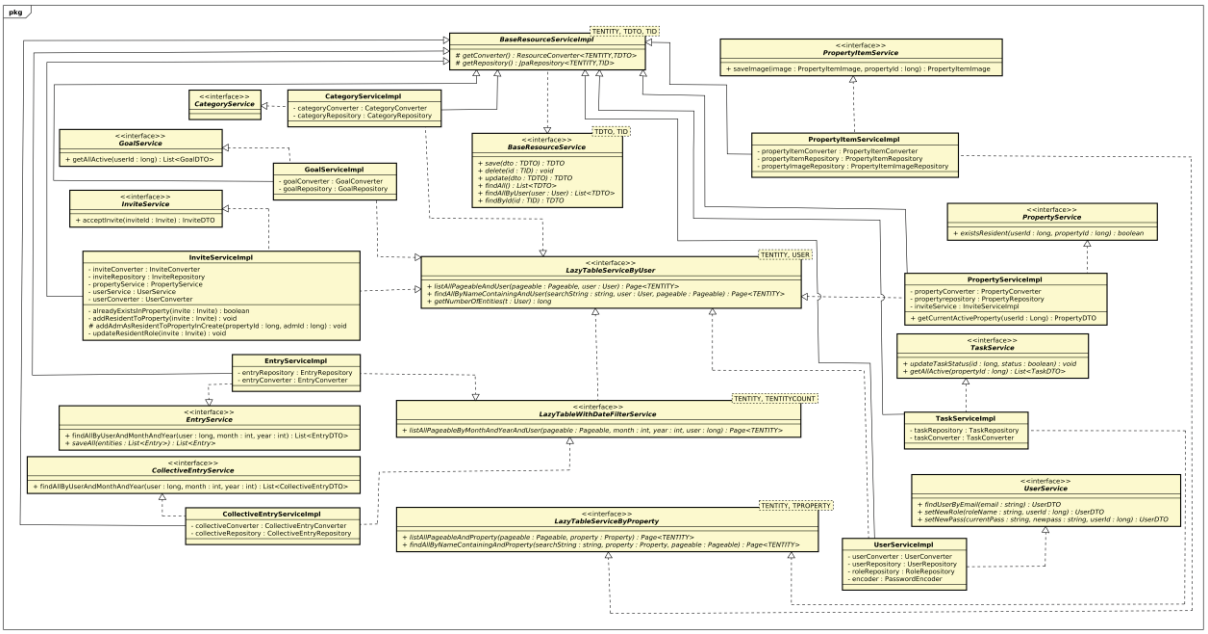
FONTE: O Autor (2021)

FIGURA 79 - DIAGRAMA DE CLASSES DE SERVIÇO BACKEND



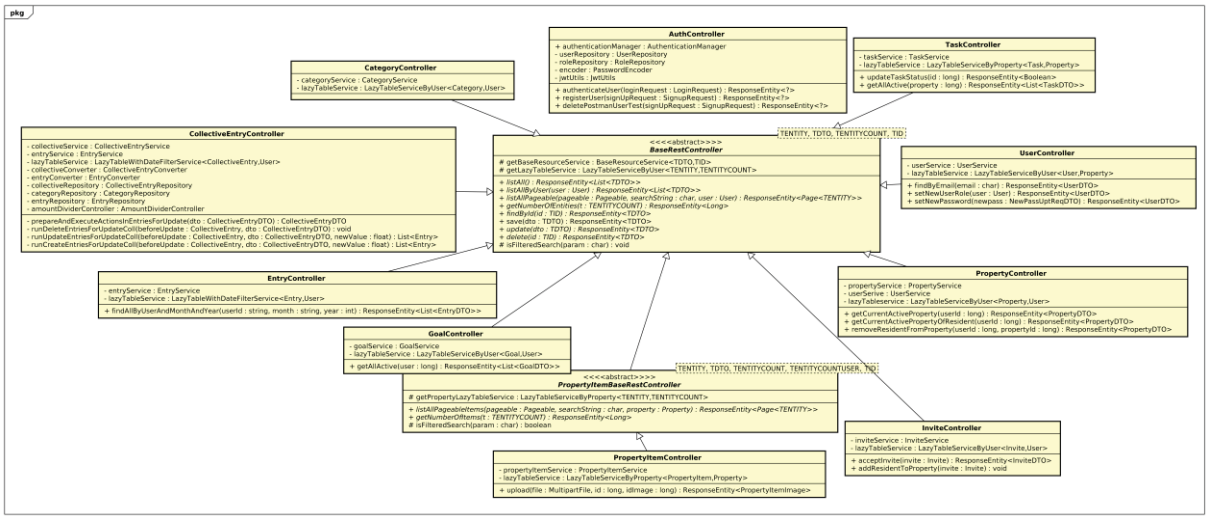
FONTE: O Autor (2021)

FIGURA 80 - DIAGRAMA DE CLASSES DE SERVIÇOS IMPLEMENTADOS BACKEND



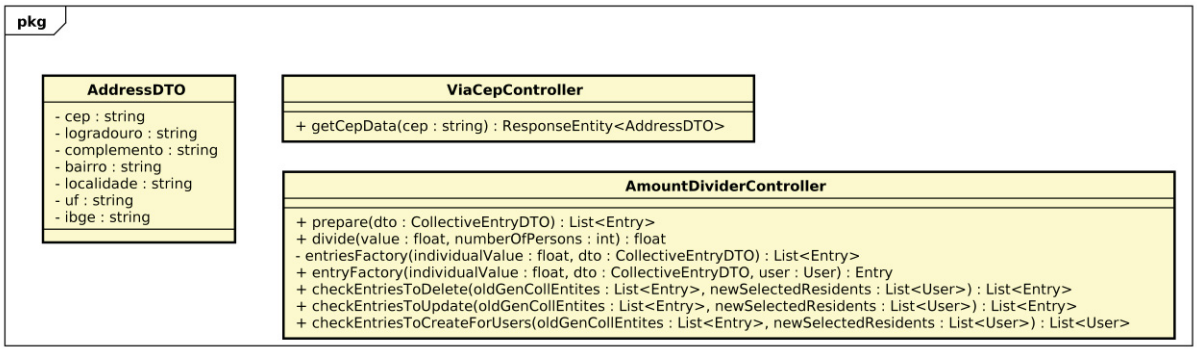
FONTE: O Autor (2021)

FIGURA 81 - DIAGRAMA DE CLASSES CONTROLADORAS BACKEND



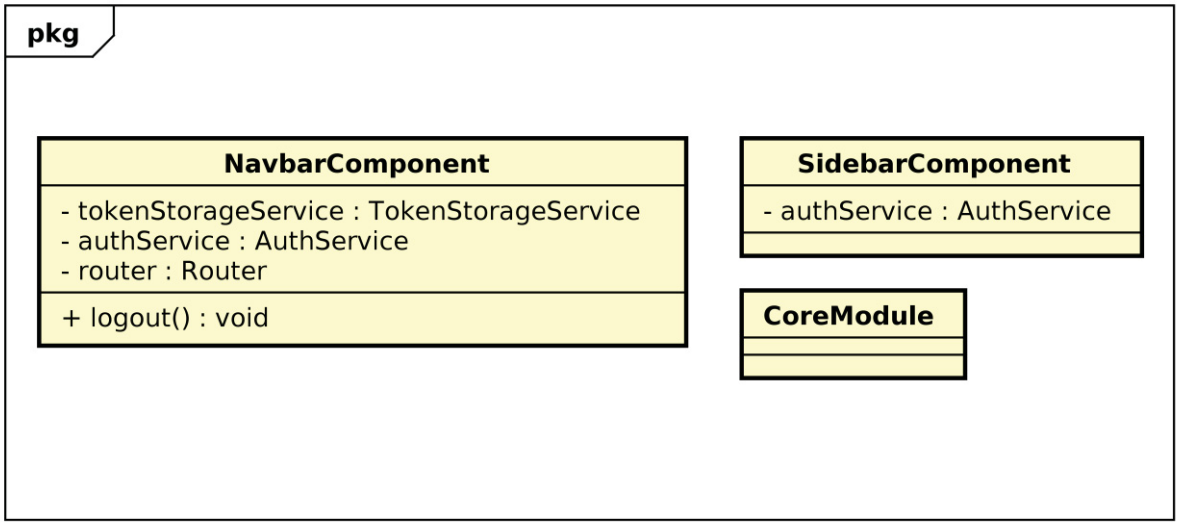
FONTE: O Autor (2021)

FIGURA 82 - DIAGRAMA DE CLASSES UTILS BACKEND



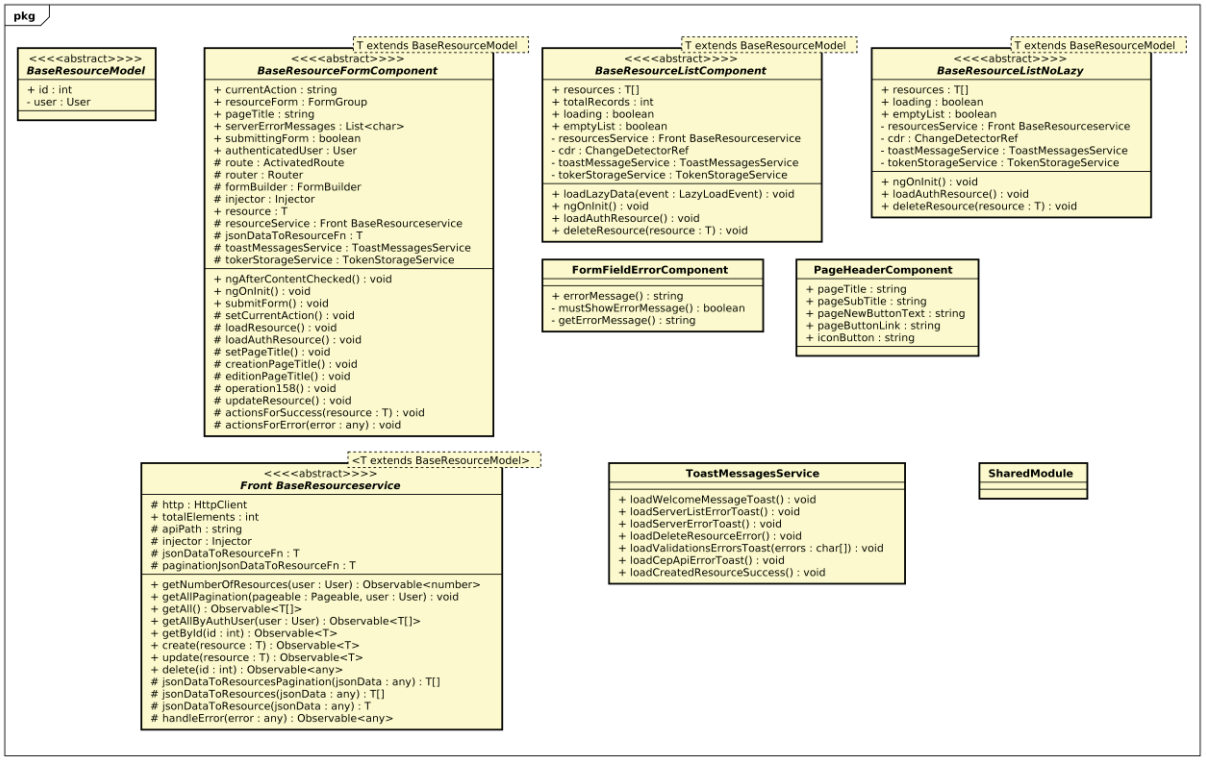
FONTE: O Autor (2021)

FIGURA 83 - DIAGRAMA DE CLASSES CORE FRONTEND



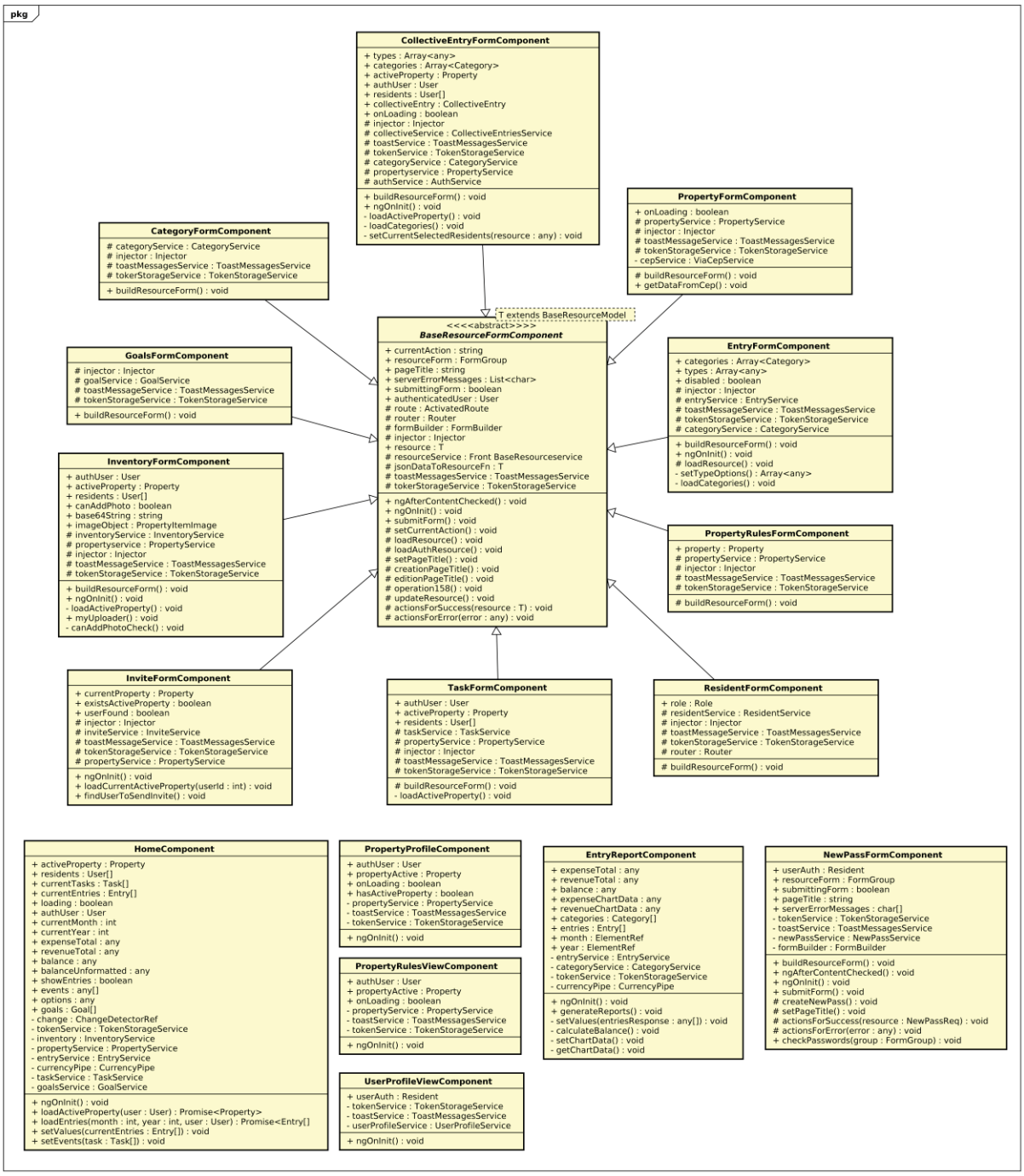
FONTE: O Autor (2021)

FIGURA 84 - DIAGRAMA DE CLASSES COMPARTILHADAS FRONTEND



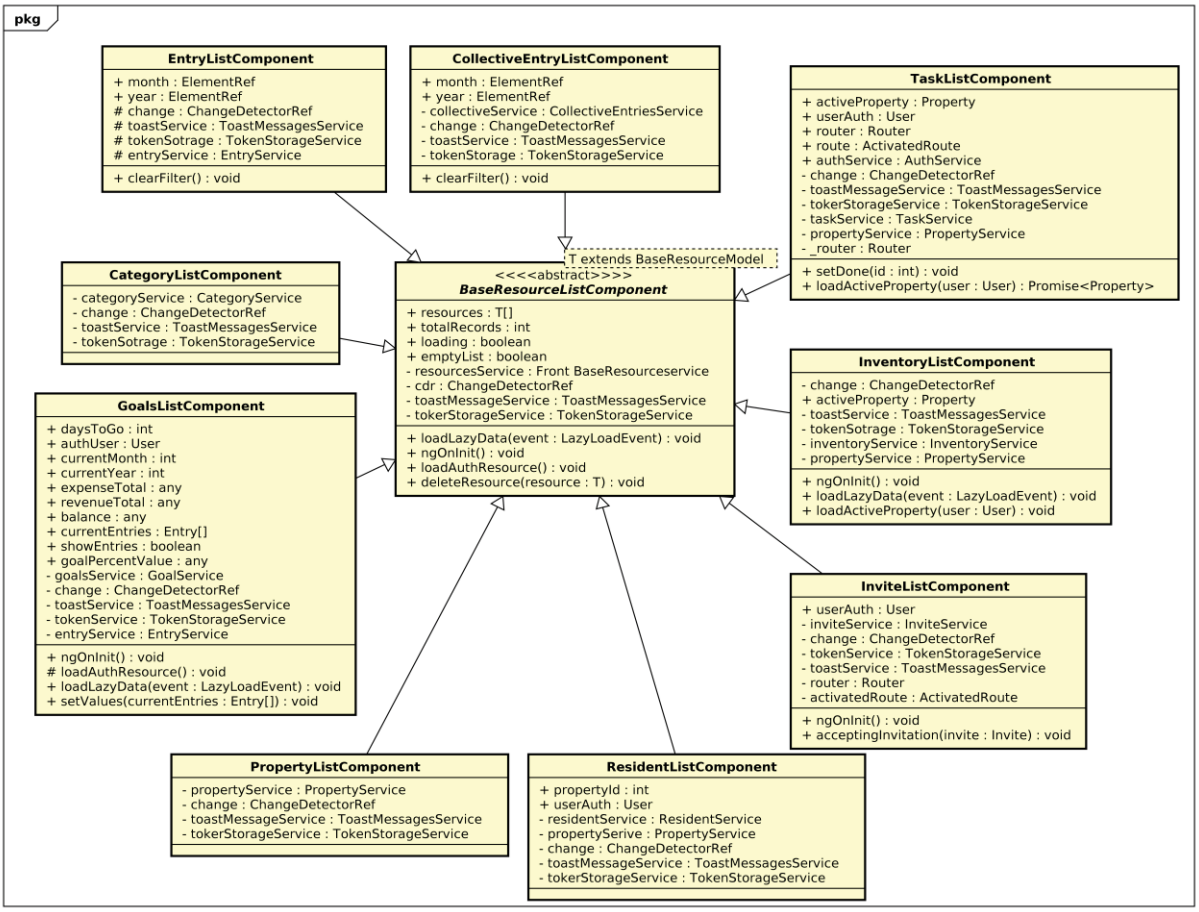
FONTE: O Autor (2021)

FIGURA 85 - DIAGRAMA DE CLASSES DE COMPONENTES DE FORM FRONTEND



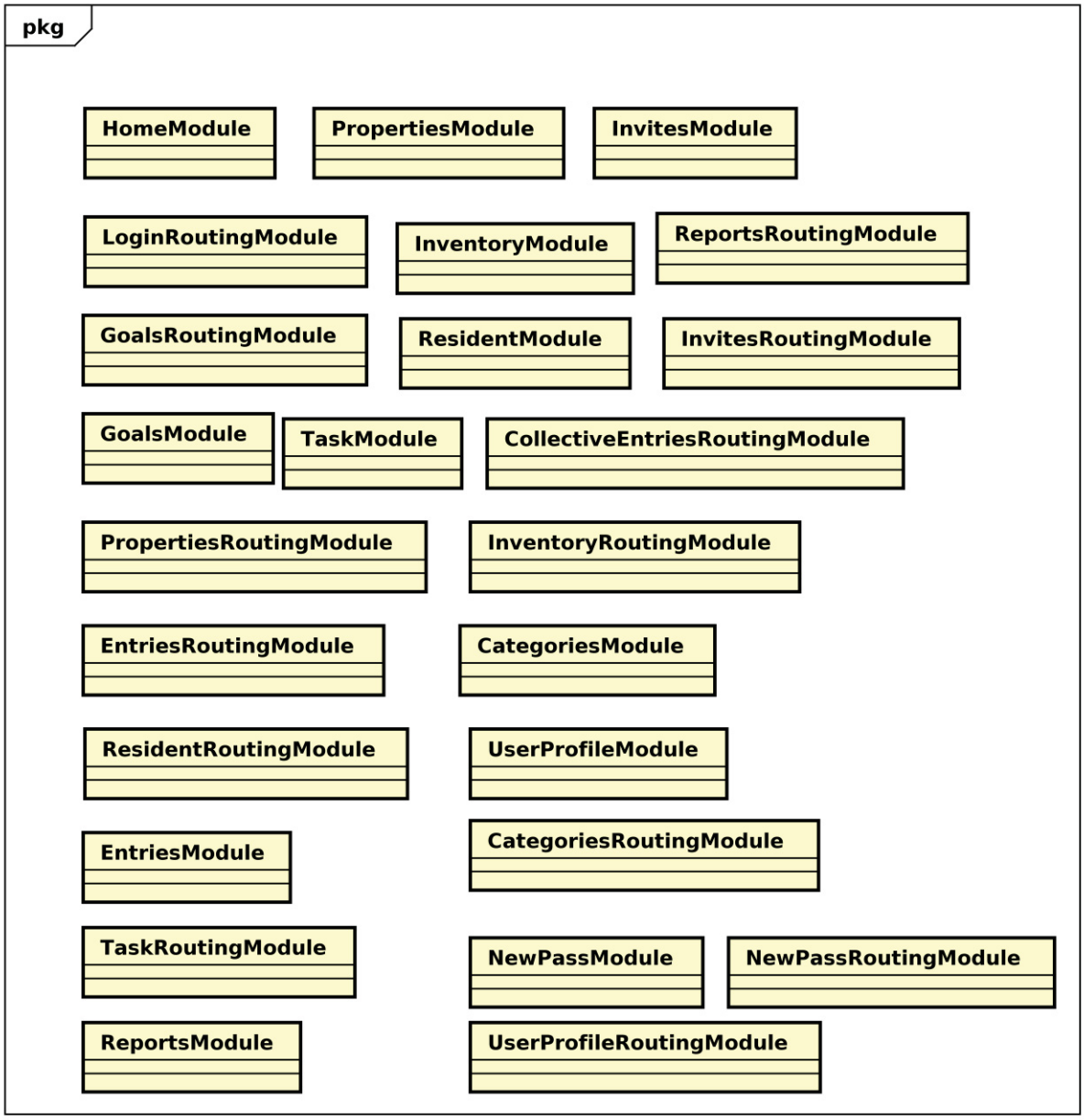
FONTE: O Autor (2021)

FIGURA 86 - DIAGRAMA DE CLASSES DE LISTAGEM FRONTEND



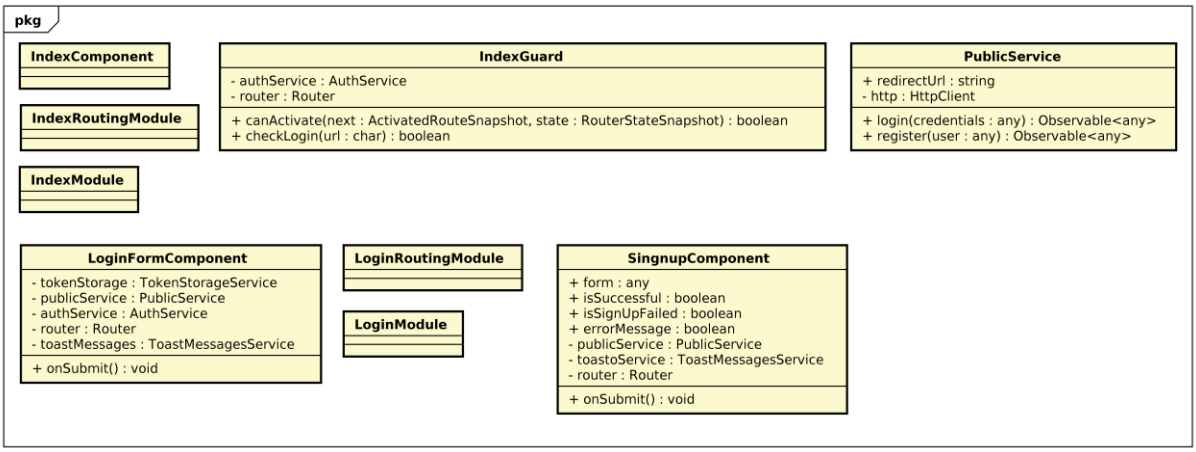
FONTE: O Autor (2021)

FIGURA 87 - DIAGRAMA DE CLASSES DE MODULOS FRONTEND



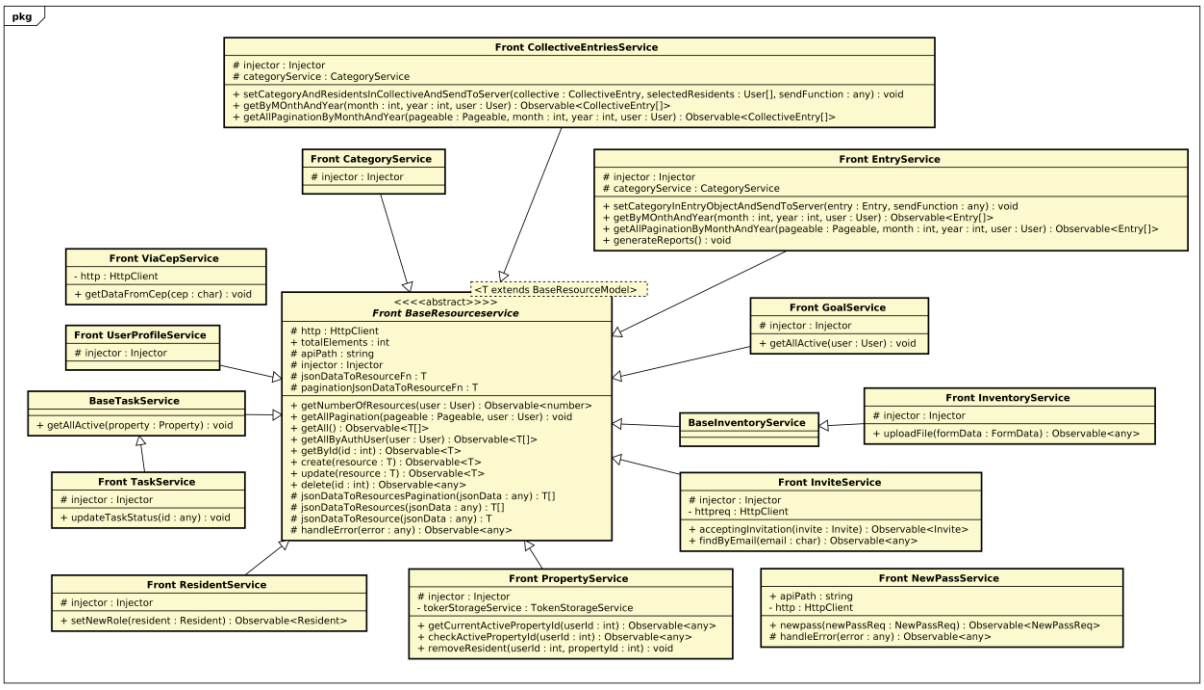
FONTE: O Autor (2021)

FIGURA 88 - DIAGRAMA DE CLASSES DE PÁGINAS PÚBLICAS FRONTEND



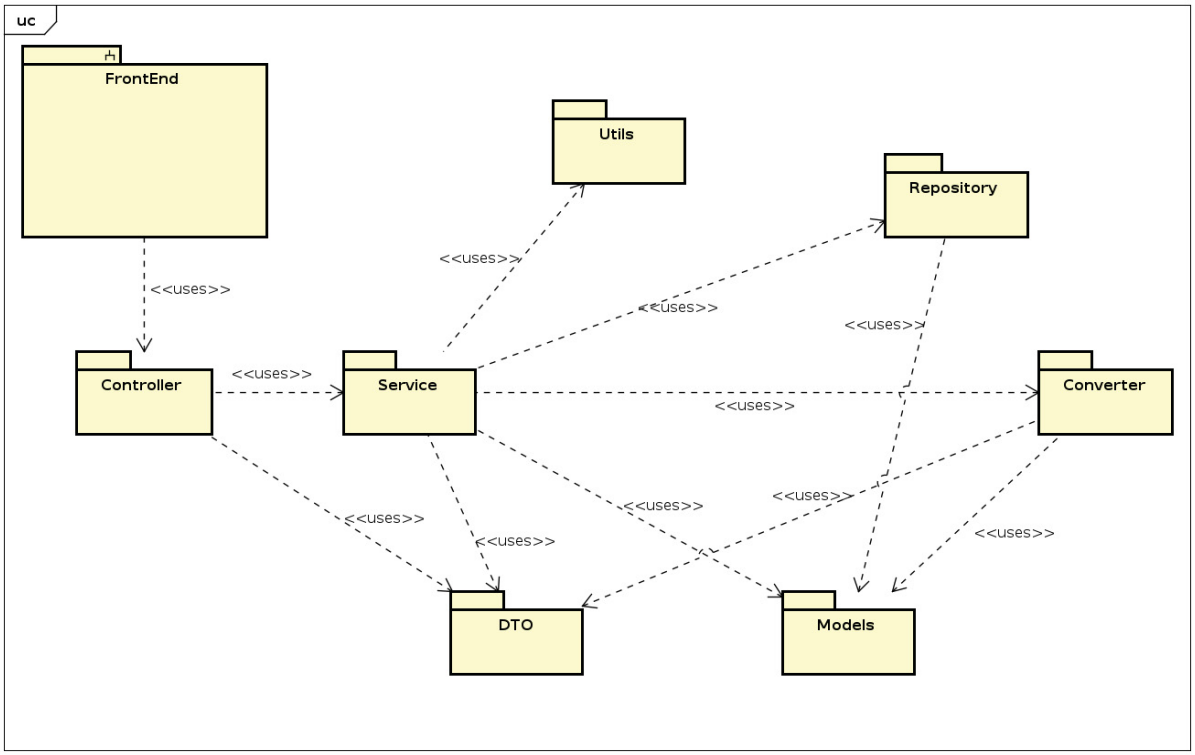
FONTE: O Autor (2021)

FIGURA 89 - DIAGRAMA DE CLASSES DE SERVIÇOS FRONTEND



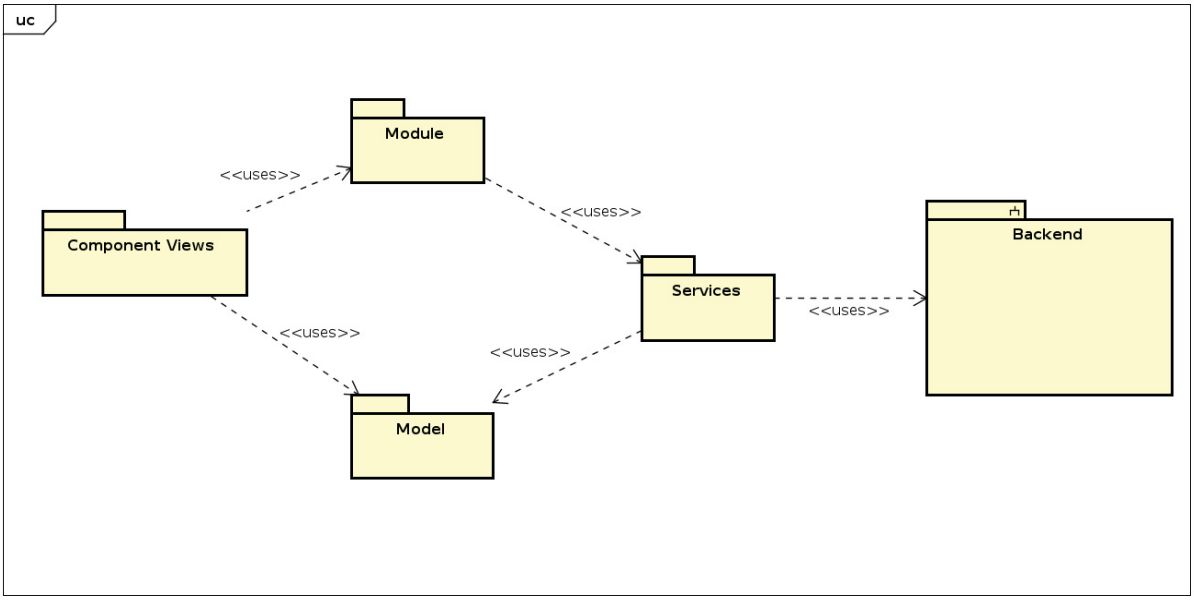
FONTE: O Autor (2021)

FIGURA 90 - DIAGRAMA DE CLASSES PACOTE BACKEND



FONTE: O Autor (2021)

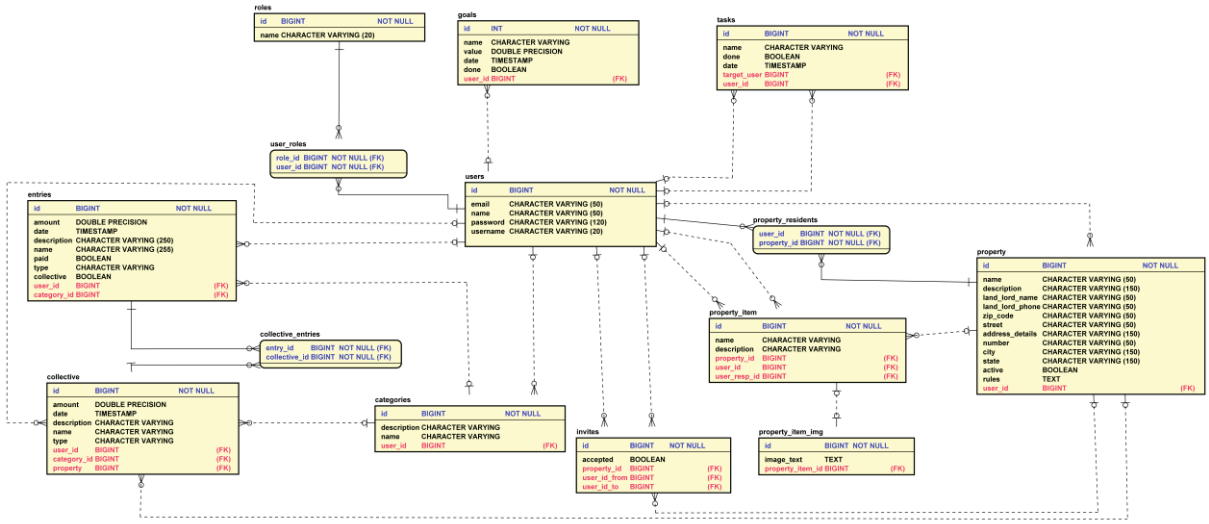
FIGURA 91 - DIAGRAMA DE CLASSES PACOTE FRONTEND



FONTE: O Autor (2021)

APÊNDICE D – DIAGRAMA FÍSICO DO BANCO DE DADOS

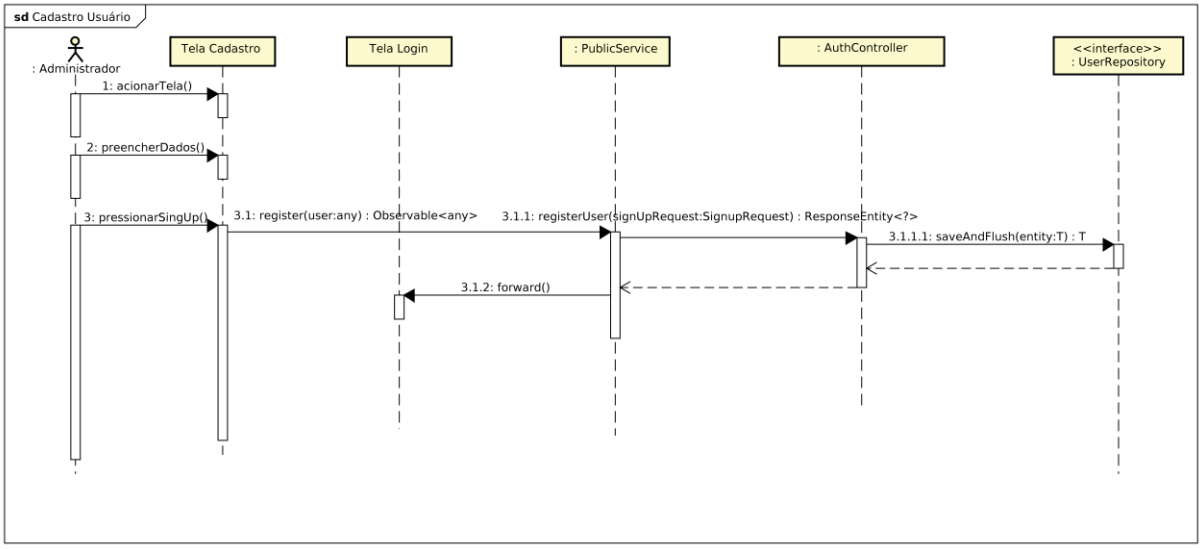
FIGURA 92 - BANCO DE DADOS



FONTE: O Autor (2021)

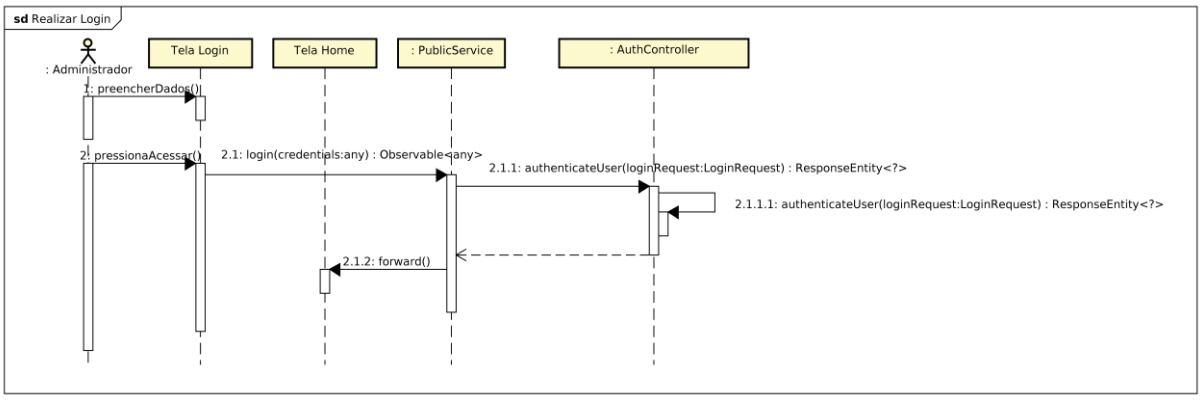
APÊNDICE E – DIAGRAMAS DE SEQUÊNCIA

FIGURA 93 - DIAGRAMA CADASTRAR USUÁRIO



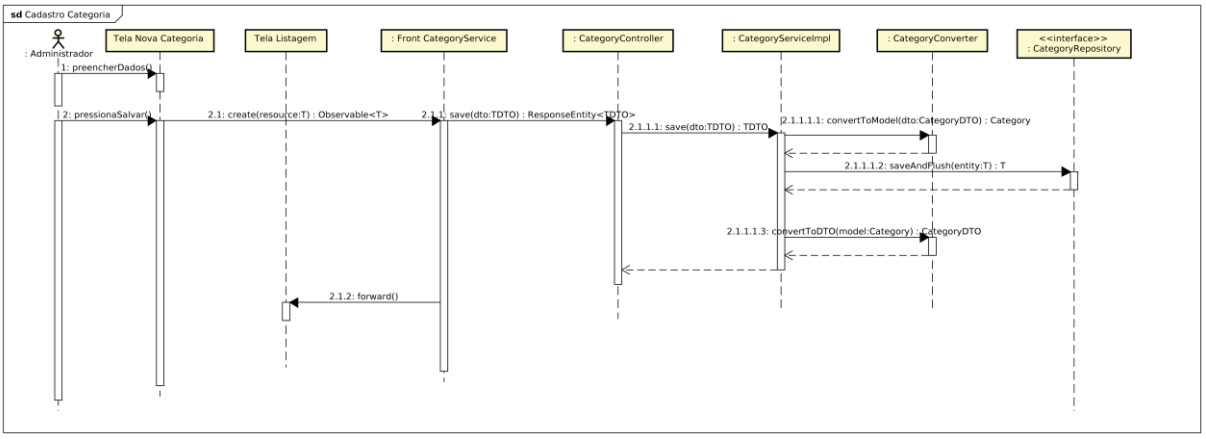
FONTE: O Autor (2021)

FIGURA 94 - DIAGRAMA REALIZAR LOGIN



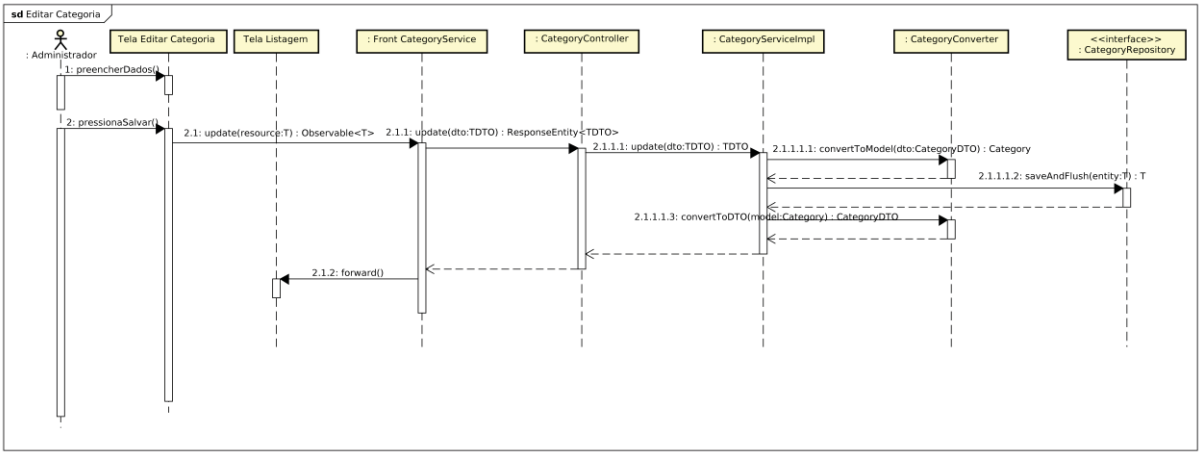
FONTE: O Autor (2021)

FIGURA 95 - DIAGRAMA CADASTRAR CATEGORIA



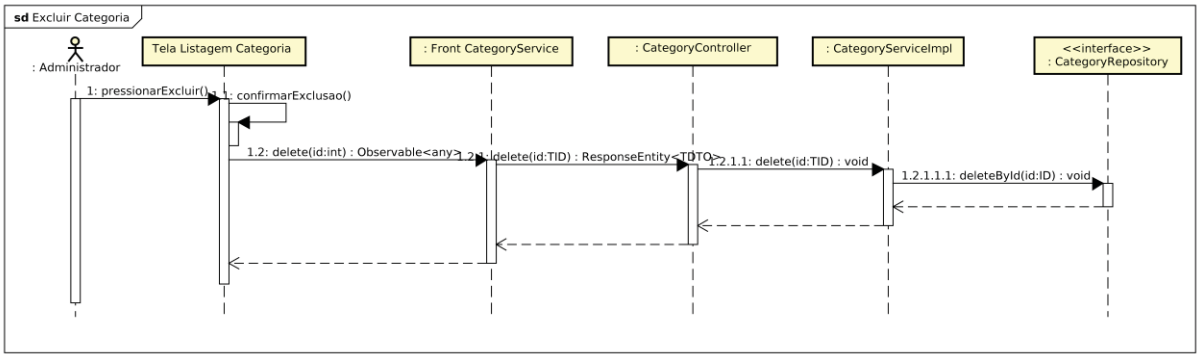
FONTE: O Autor (2021)

FIGURA 96 - DIAGRAMA EDITAR CATEGORIA



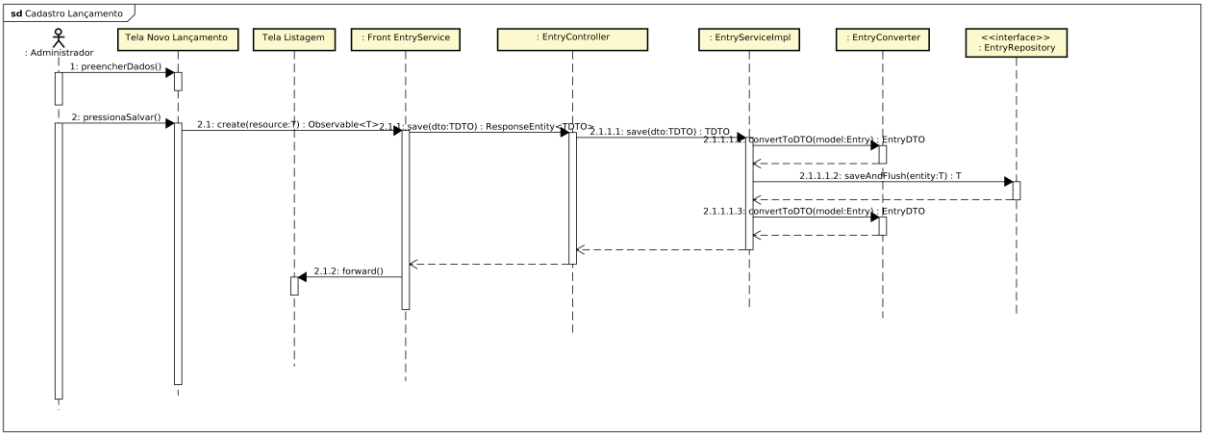
FONTE: O Autor (2021)

FIGURA 97 - DIAGRAMA EXCLUIR CATEGORIA



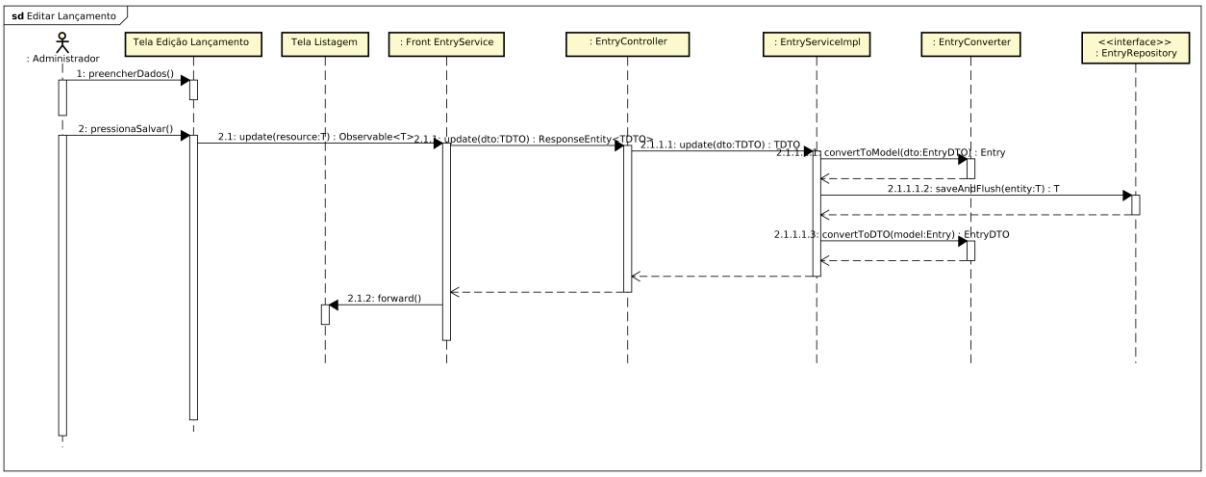
FONTE: O Autor (2021)

FIGURA 98 - DIAGRAMA CADASTRO LANÇAMENTO



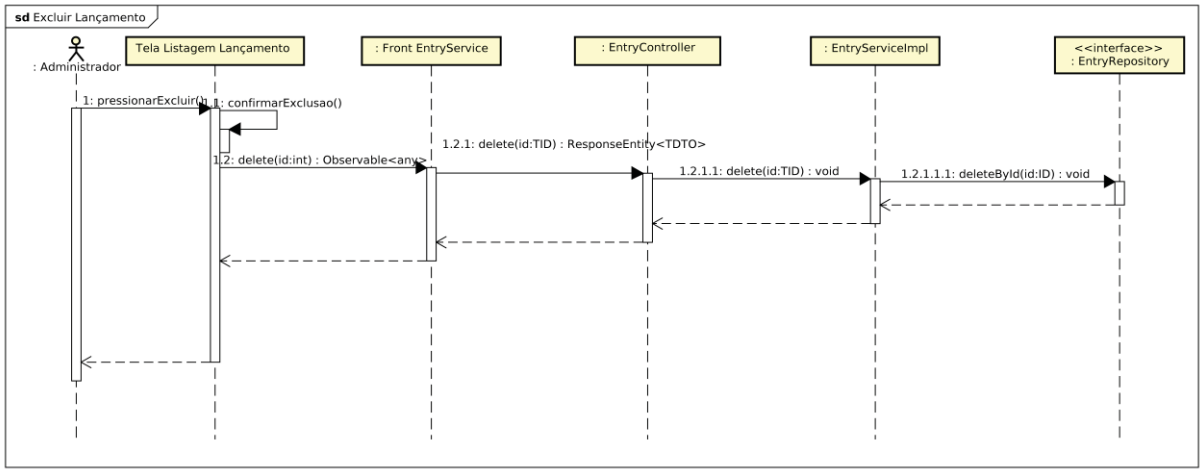
FONTE: O Autor (2021)

FIGURA 99 - DIAGRAMA EDITAR LANÇAMENTO



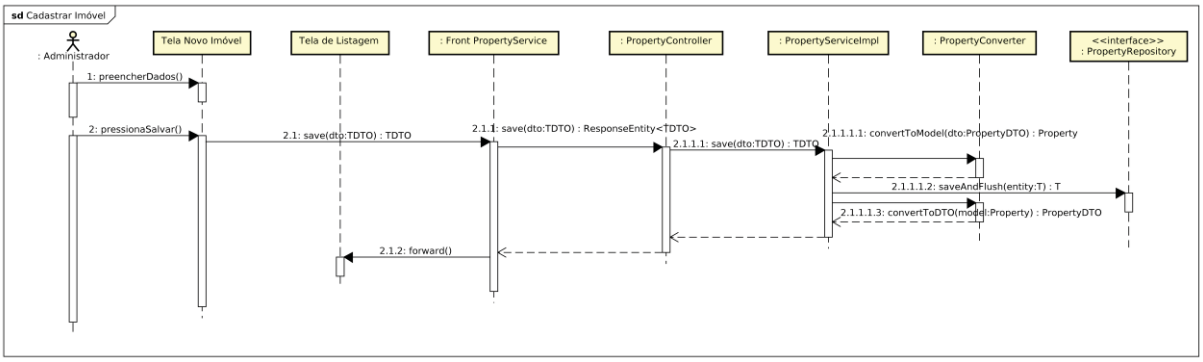
FONTE: O Autor (2021)

FIGURA 100 - DIAGRAMA EXCLUIR LANÇAMENTO



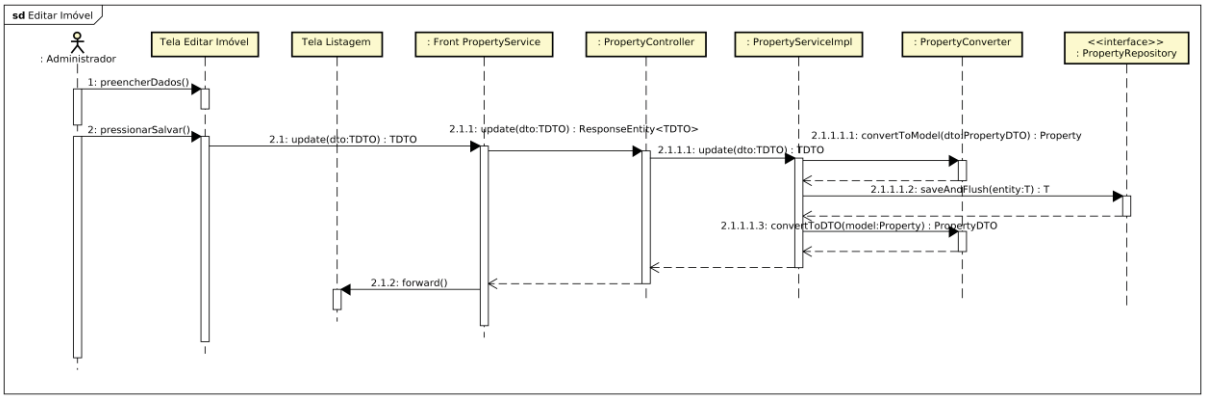
FONTE: O Autor (2021)

FIGURA 101 - DIAGRAMA CADASTRO IMÓVEL



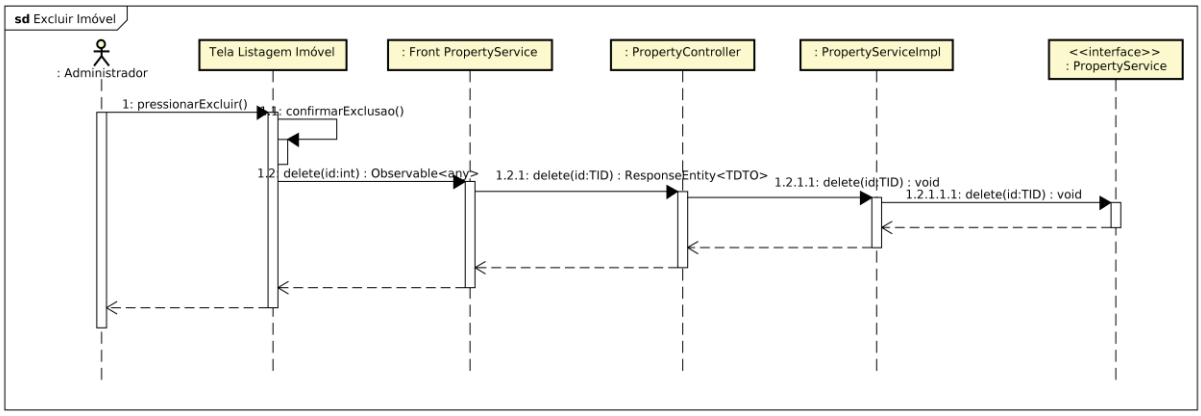
FONTE: O Autor (2021)

FIGURA 102 - DIAGRAMA EDITAR IMÓVEL



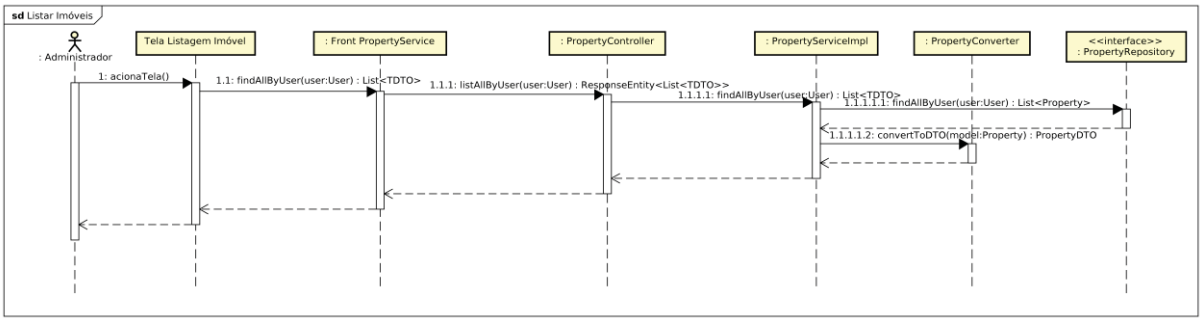
FONTE: O Autor (2021)

FIGURA 103 - DIAGRAMA EXCLUIR IMÓVEL



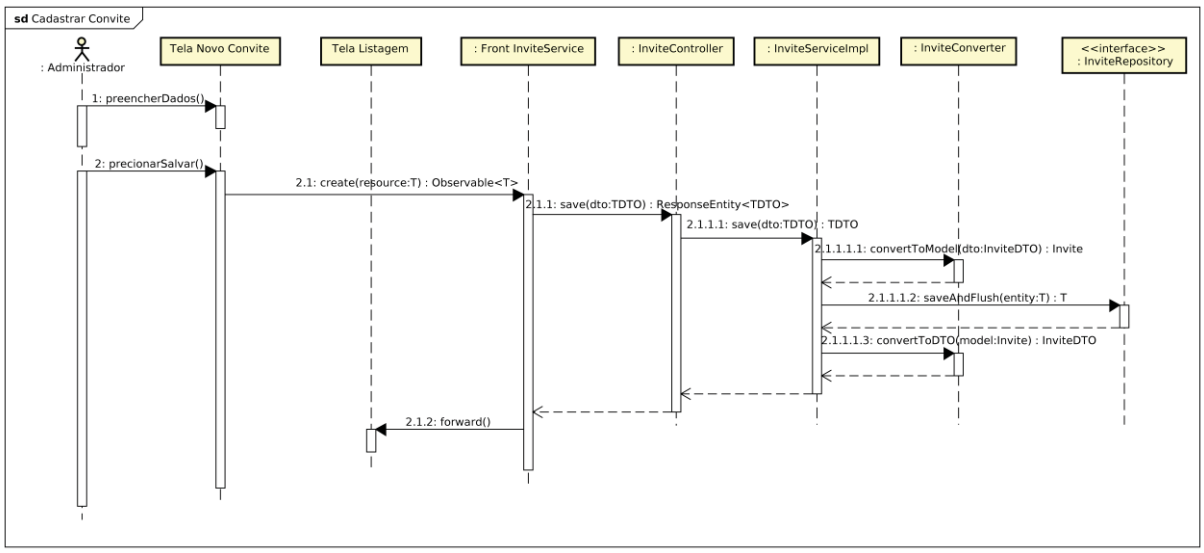
FONTE: O Autor (2021)

FIGURA 104 - DIAGRAMA LISTAR IMÓVEIS



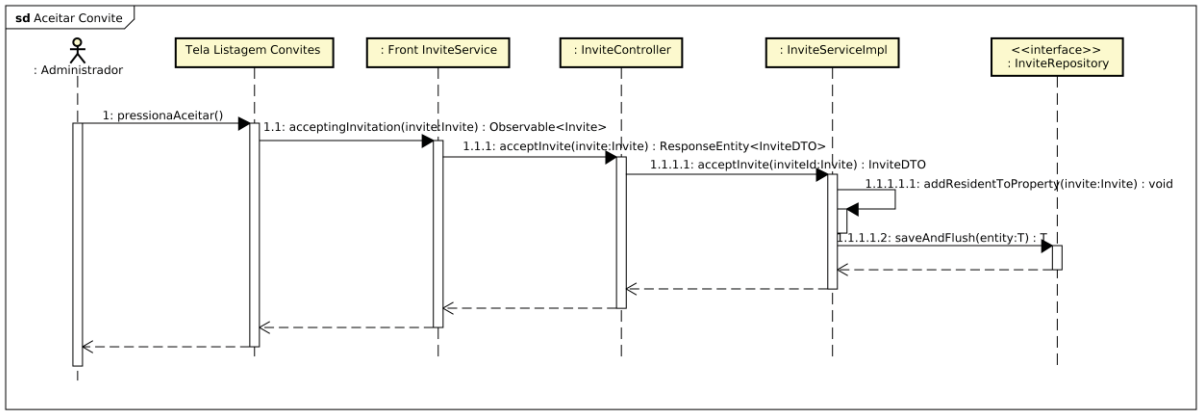
FONTE: O Autor (2021)

FIGURA 105 - DIAGRAMA CADASTRO CONVITE



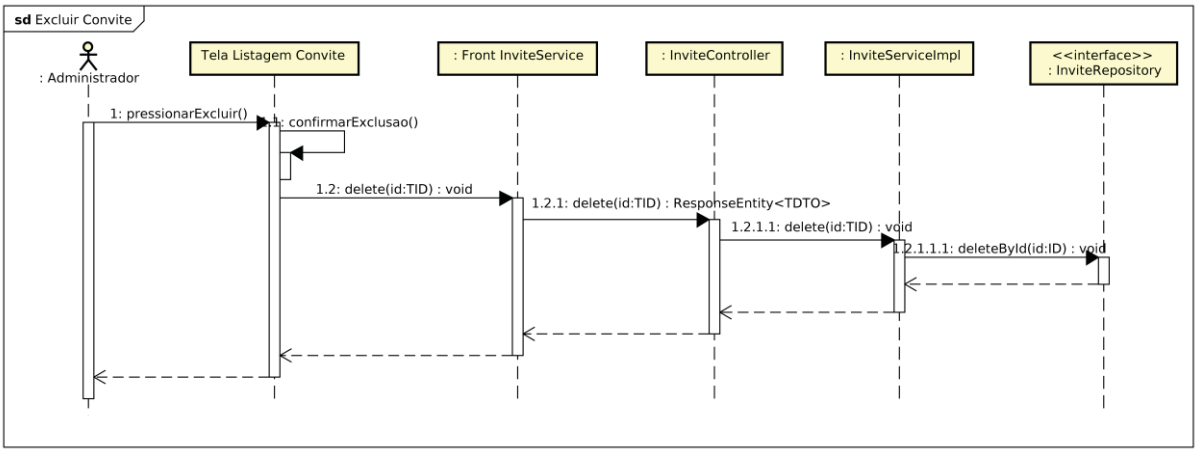
FONTE: O Autor (2021)

FIGURA 106 - DIAGRAMA ACEITAR CONVITE



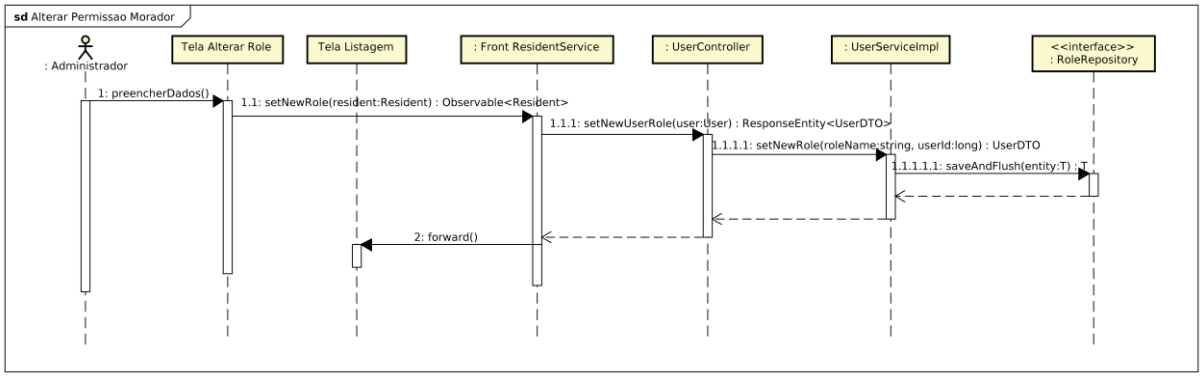
FONTE: O Autor (2021)

FIGURA 107 - DIAGRAMA EXCLUIR CONVITE



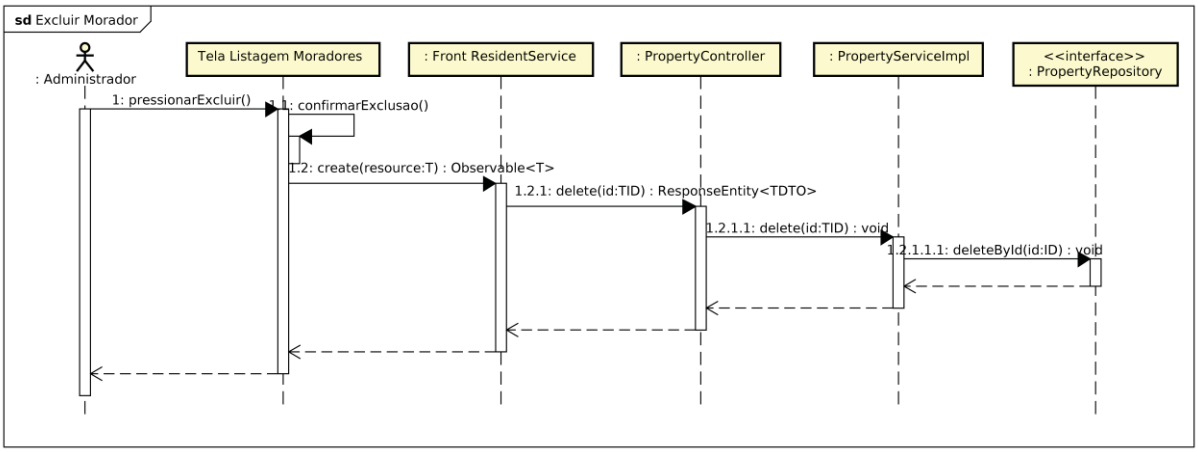
FONTE: O Autor (2021)

FIGURA 108 - DIAGRAMA ALTERAR PERMISSÃO MORADOR



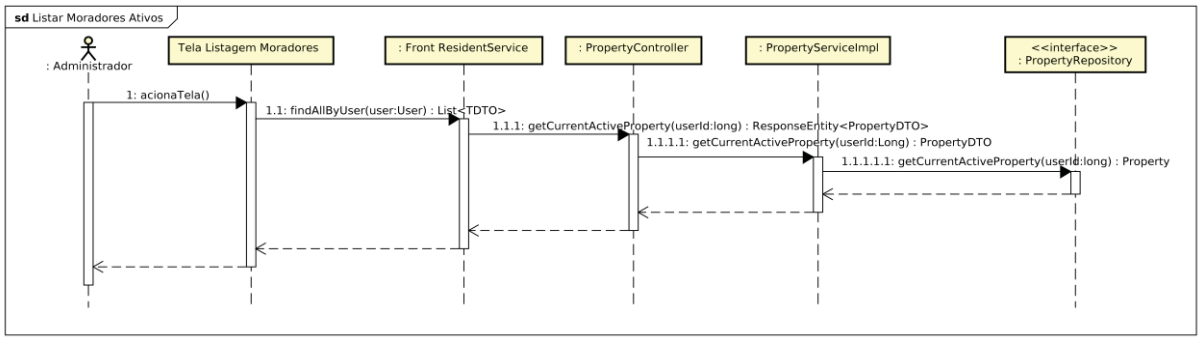
FONTE: O Autor (2021)

FIGURA 109 - DIAGRAMA EXCLUIR MORADOR



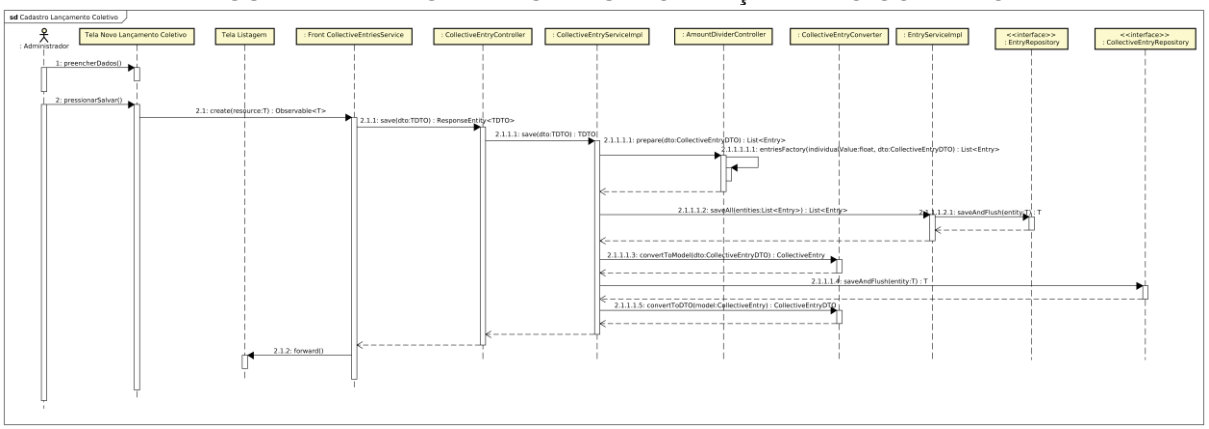
FONTE: O Autor (2021)

FIGURA 110 - DIAGRAMA LISTAR MORADORES



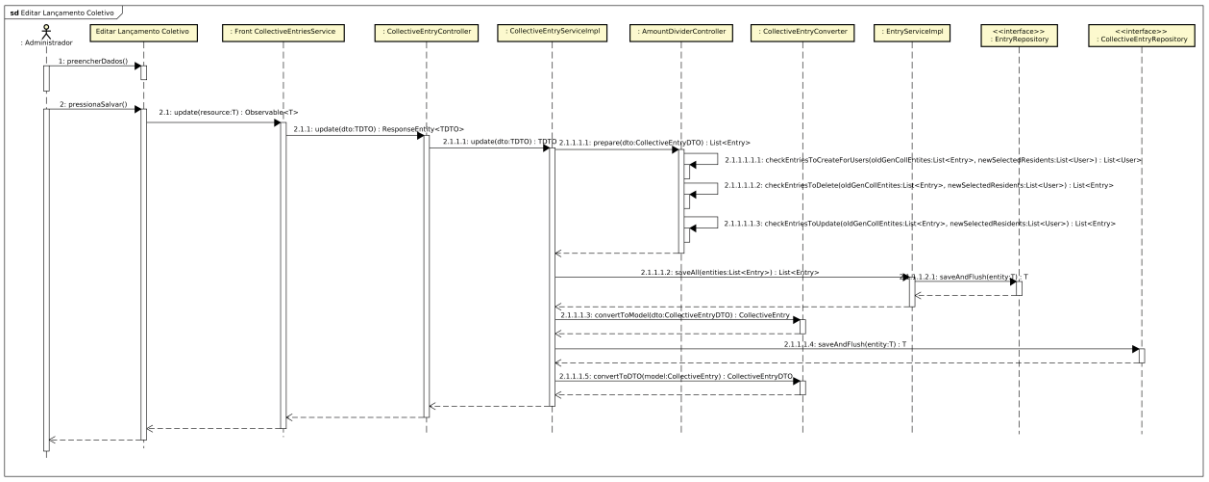
FONTE: O Autor (2021)

FIGURA 111 - DIAGRAMA CADASTRO LANÇAMENTO COLETIVO



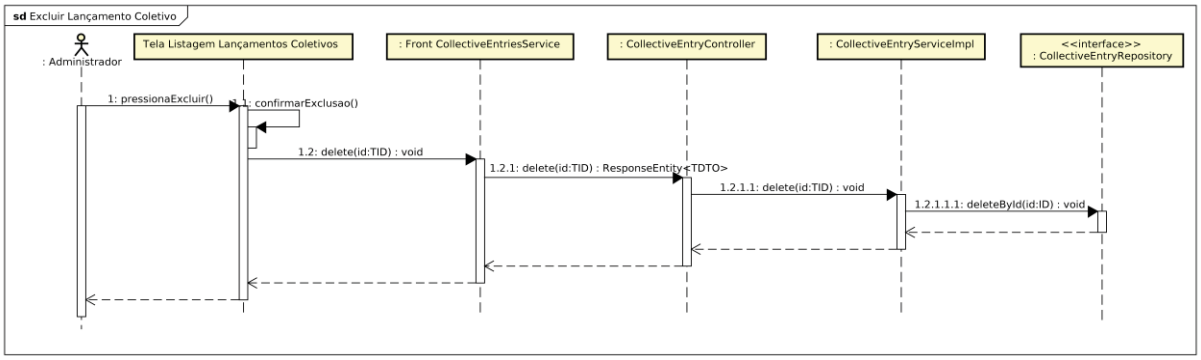
FONTE: O Autor (2021)

FIGURA 112 - DIAGRAMA EDITAR LANÇAMENTO COLETIVO



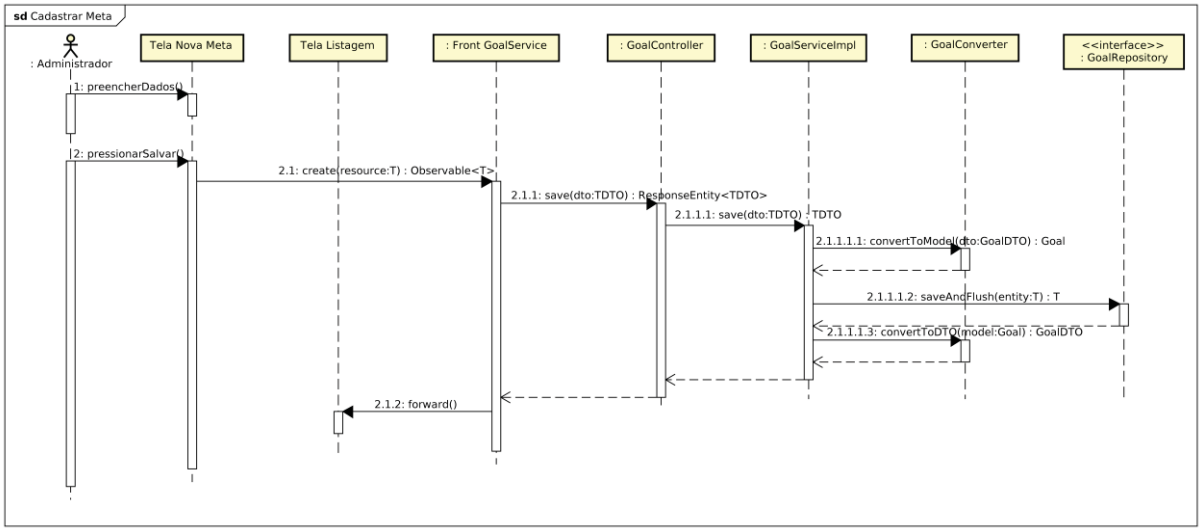
FONTE: O Autor (2021)

FIGURA 113 - DIAGRAMA EXCLUIR LANÇAMENTO COLETIVO



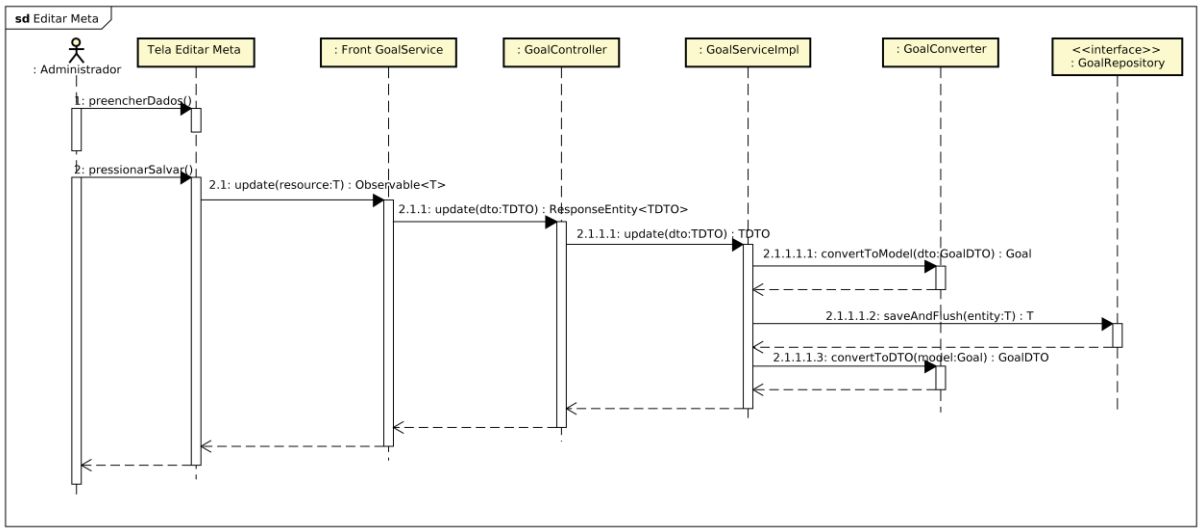
FONTE: O Autor (2021)

FIGURA 114 - DIAGRAMA CADASTRAR META



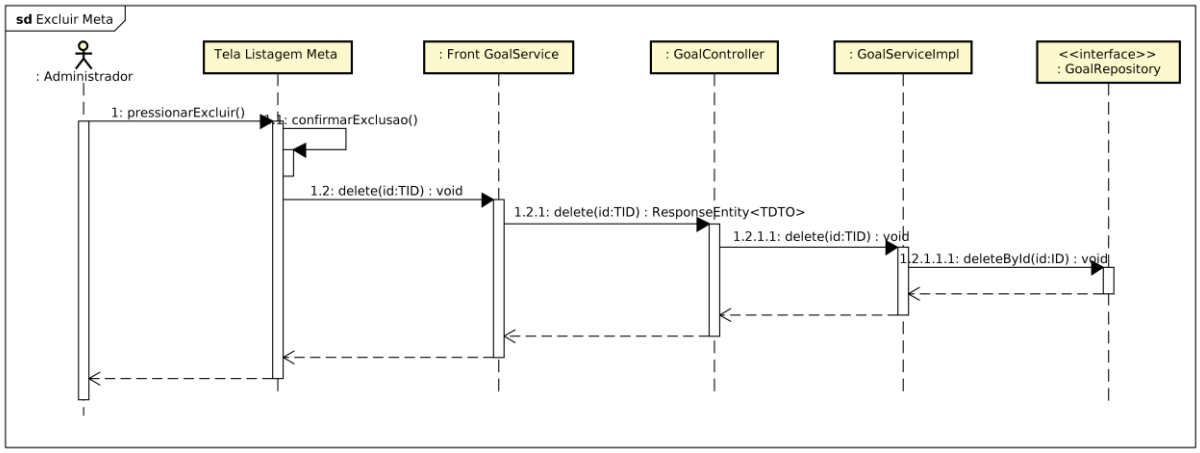
FONTE: O Autor (2021)

FIGURA 115 - DIAGRAMA EDITAR META



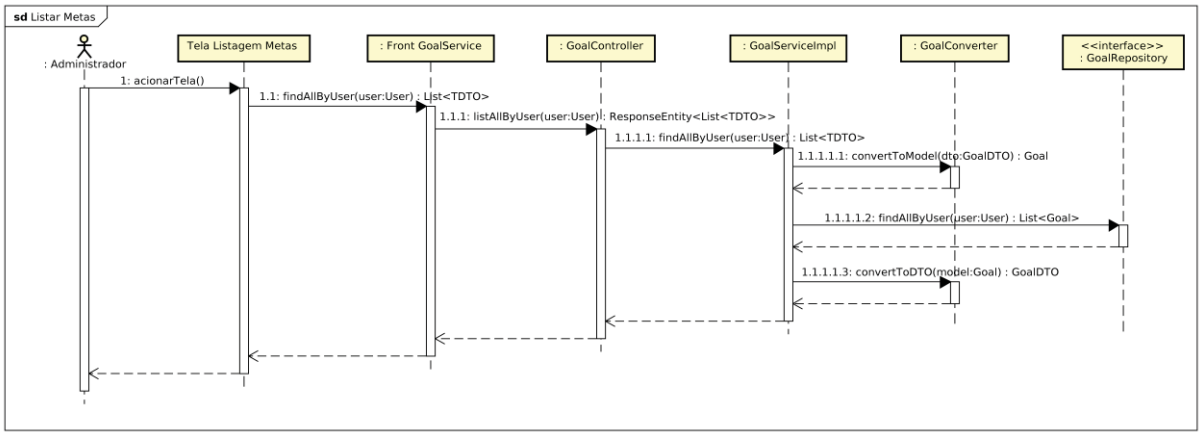
FONTE: O Autor (2021)

FIGURA 116 - DIAGRAMA EXCLUIR META



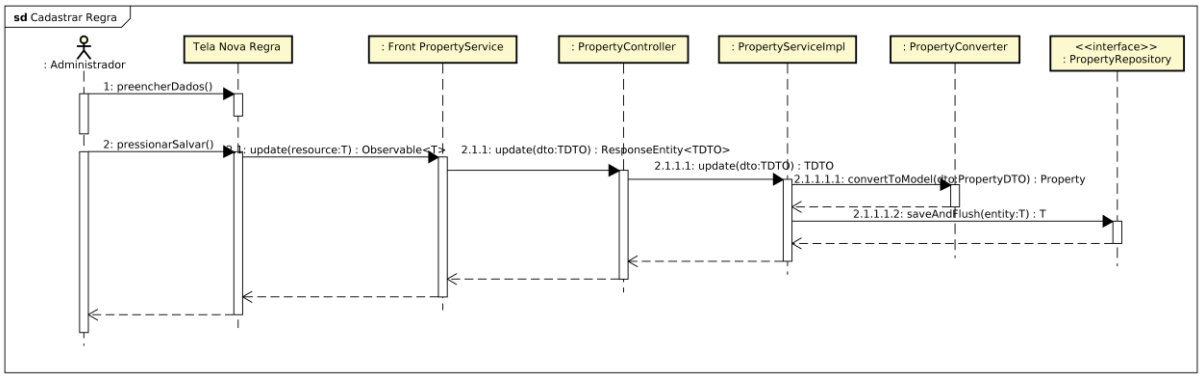
FONTE: O Autor (2021)

FIGURA 117 - DIAGRAMA LISTAR METAS



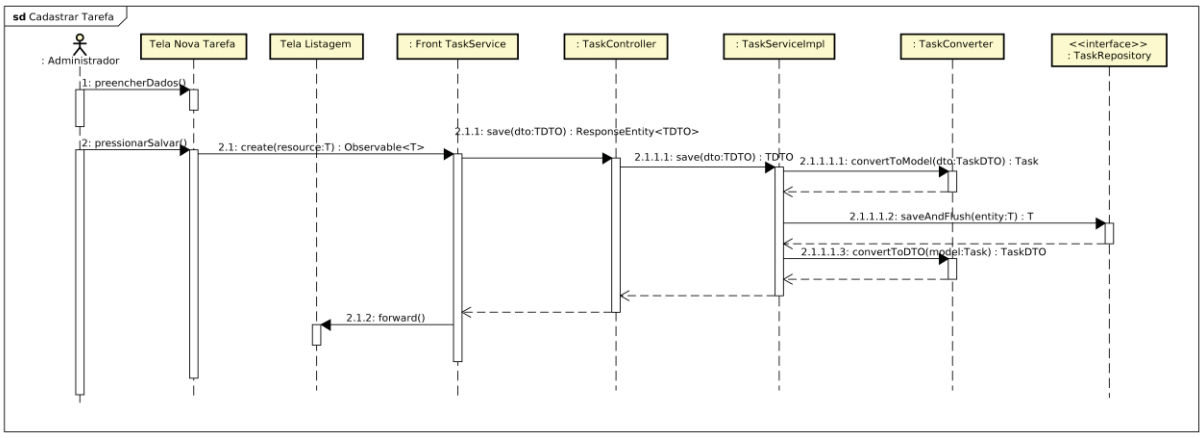
FONTE: O Autor (2021)

FIGURA 118 - DIAGRAMA CADASTRAR REGRA



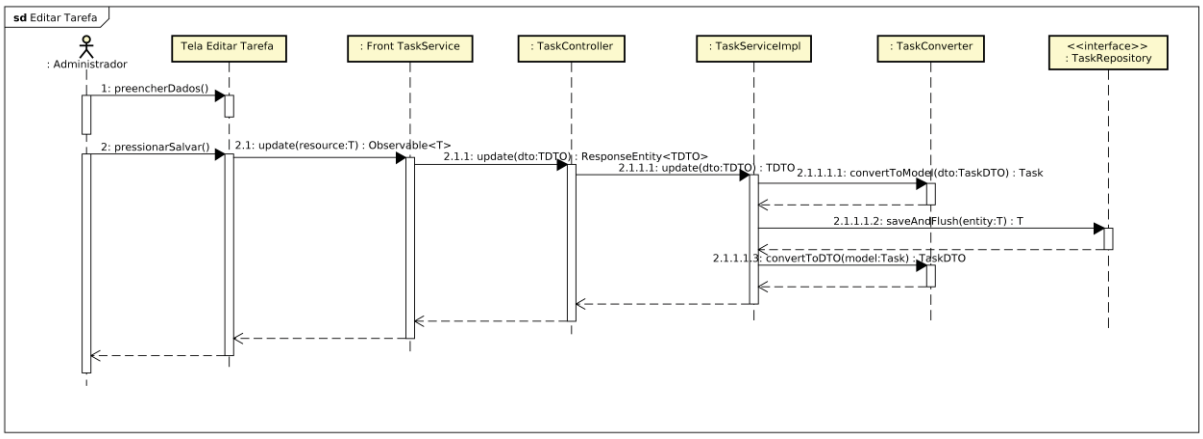
FONTE: O Autor (2021)

FIGURA 119 - DIAGRAMA CADASTRAR TAREFA



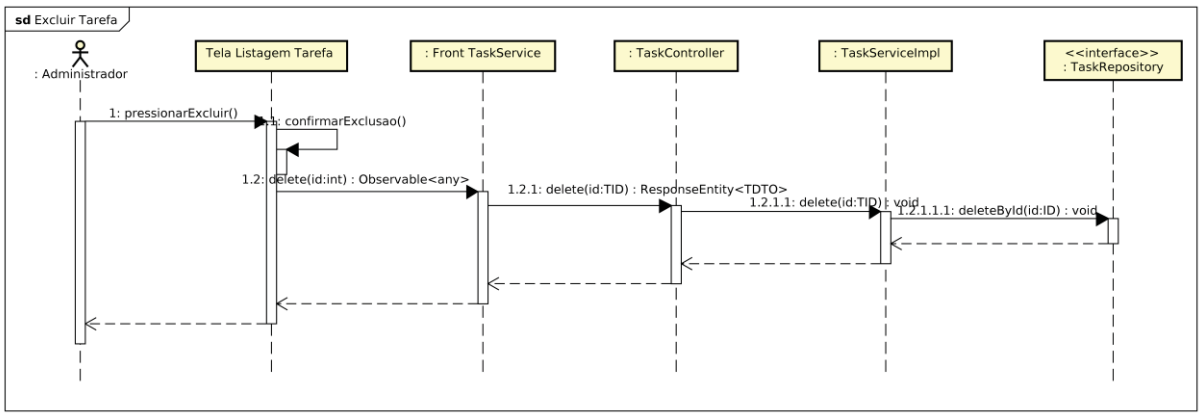
FONTE: O Autor (2021)

FIGURA 120 - DIAGRAMA EDITAR TAREFA



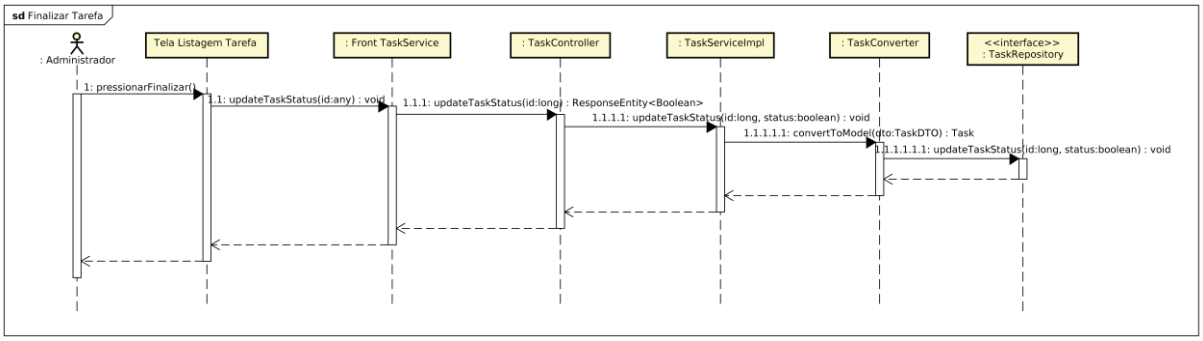
FONTE: O Autor (2021)

FIGURA 121 - DIAGRAMA EXCLUIR TAREFA



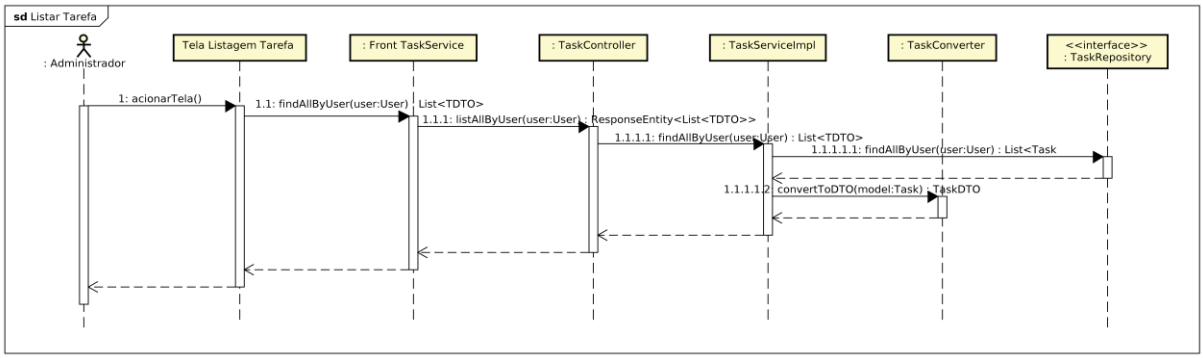
FONTE: O Autor (2021)

FIGURA 122 - DIAGRAMA FINALIZAR TAREFA



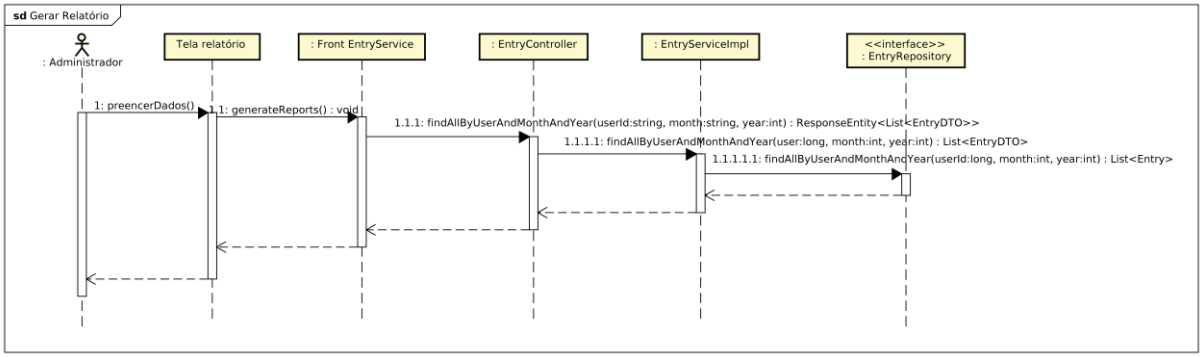
FONTE: O Autor (2021)

FIGURA 123 - DIAGRAMA LISTAR TAREFA



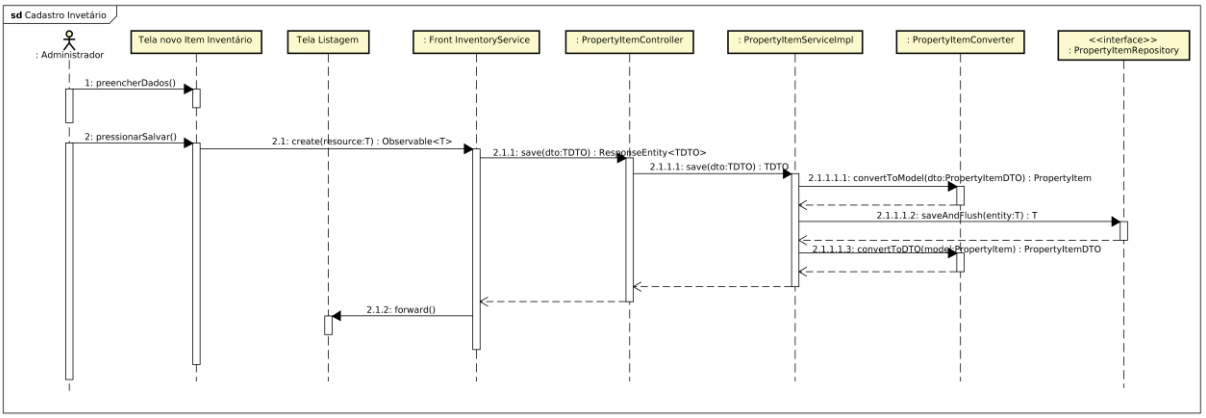
FONTE: O Autor (2021)

FIGURA 124 - DIAGRAMA GERAR RELATÓRIO



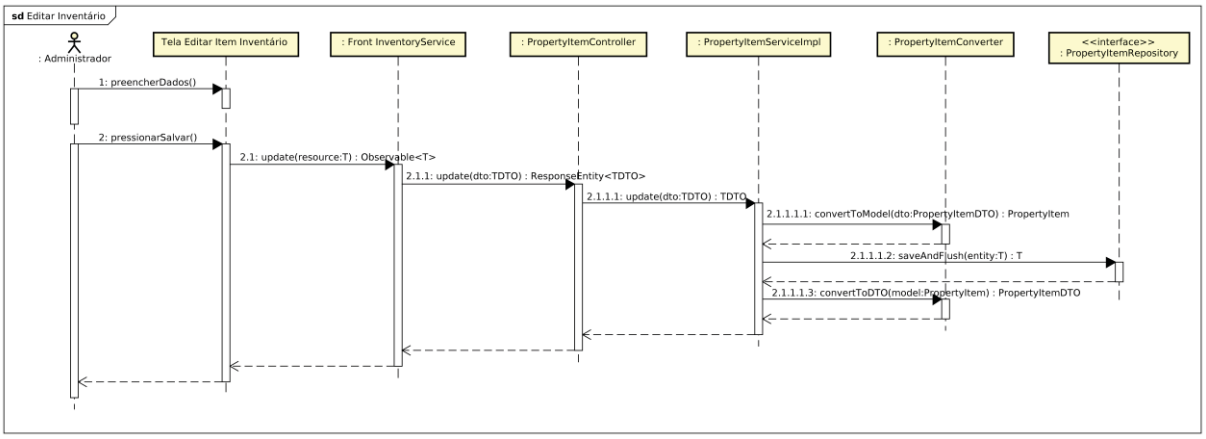
FONTE: O Autor (2021)

FIGURA 125 - DIAGRAMA CADASTRO INVENTÁRIO



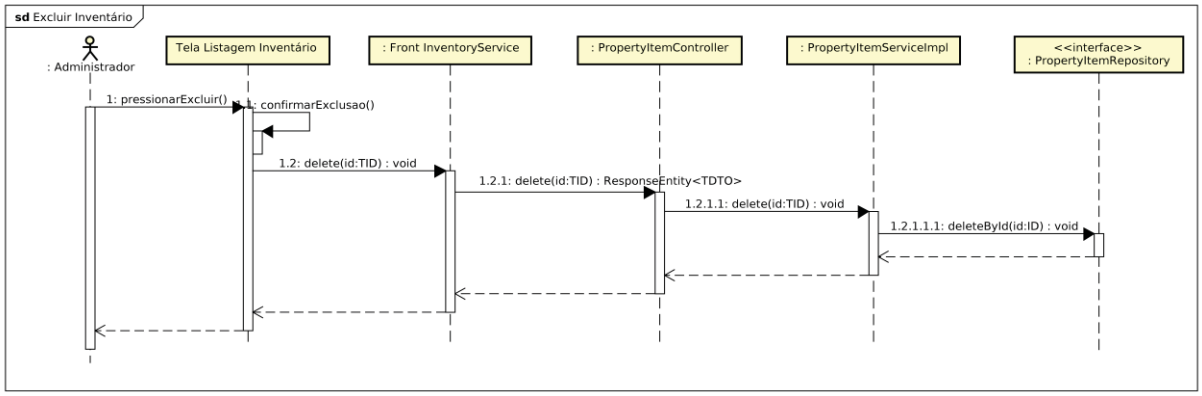
FONTE: O Autor (2021)

FIGURA 126 - DIAGRAMA EDITAR INVENTÁRIO



FONTE: O Autor (2021)

FIGURA 127 - DIAGRAMA EXCLUIR INVENTÁRIO



FONTE: O Autor (2021)