UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DEPARTAMENTO ACADÊMICO DE INFORMÁTICA

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

GIOVANE AIRES GOMES

THIAGO TRACZYKOWSKI

SOLUÇÃO AUTOMATIZADA PARA GESTÃO DOS PROCESSOS INTERNOS DO CENTRO SOCIAL CASA DO PIÁ

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA

2019

GIOVANE AIRES GOMES

THIAGO TRACZYKOWSKI

SOLUÇÃO AUTOMATIZADA PARA GESTÃO DOS PROCESSOS INTERNOS DO CENTRO SOCIAL CASA DO PIÁ

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, do Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Richard Duarte Ribeiro

PONTA GROSSA

2019

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Ponta Grossa

Diretoria de Graduação e Educação Profissional
Departamento Acadêmico de Informática
Tecnologia em Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

SOLUÇÃO AUTOMATIZADA PARA GESTÃO DOS PROCESSOS INTERNOS DO CENTRO SOCIAL CASA DO PIÁ

por

GIOVANE AIRES GOMES E THIAGO TRACZYKOWSKI

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 11 de junho de 2019 como requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Prof. Dr. Richard Duarte Ribeiro
Orientador(a)

_____
Prof. MSc. Vinícius Camargo Andrade
Membro titular

_____
Prof. MSc. Giancarlo Rodrigues
Membro titular

_____
Prof. MSc. Geraldo Ranthum
Responsável pelo Trabalho de Conclusão de Curso

_____
Prof. Dr. André Pinz Borges
Coordenador do curso

AGRADECIMENTOS

Primeiramente, agradecemos ao Prof. Dr. Richard Duarte Ribeiro, pela paciência e orientação prestada para o desenvolvimento deste trabalho.

Agradecemos ao Centro Social Casa do Piá, pela oportunidade de desenvolver este trabalho em conjunto.

Agradecemos também, as nossas famílias e amigos, que sempre estiveram ao nosso lado nos incentivando e apoiando para nós atingirmos os nossos objetivos.

Aos professores da UTFPR, pelos ensinamentos que contribuíram para nossa formação profissional.

Por fim, agradecemos aos colegas de curso, por todos os momentos que compartilhamos durante esta longa caminhada realizada na UTFPR.

RESUMO

GOMES, Giovane Aires. TRACZYKOWSKI, Thiago. **Solução automatizada para gestão dos processos internos do centro social Casa do Piá**. 2019. Número total de folhas 78f. Trabalho de Conclusão de Curso (Tecnologia em Análise e Desenvolvimento de Sistemas) - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2019.

Este trabalho apresenta um sistema de gestão para o Centro Social Casa do Piá, da cidade de Ponta Grossa, no estado do Paraná. A entidade atende a 150 crianças e adolescentes de 6 a 15 anos, e tem dificuldades para gerenciar toda informação coletada dos seus usuários e dos processos internos do Centro social, por este motivo necessita de uma solução para a organização destas informações e processos internos. Assim sendo, a solução proposta para auxiliar e agilizar este processo foi um sistema *WEB/Mobile.* Para o desenvolvimento do servidor do sistema, foi utilizado o *framework* Laravel, juntamente com banco de dados MariaDB. Para construção do cliente, foi utilizado o framework Vue.js. Por se tratar de uma entidade assistencial e sem fins lucrativos, optou-se pelo uso de tecnologias open-source. O resultado obtido foi um sistema que agiliza o trabalho dos funcionários, evitando a duplicidade dos dados e facilitando o acesso seguro às informações.

Palavras-chave: Sistema *WEB/Mobile*. *Framework* Laravel. *Framework* Vue.js.

ABSTRACT

GOMES, Giovane Aires. TRACZYKOWSKI, Thiago. **Automated solution to make the internal processes of the social center Casa do Piá**. 2019. Total number of sheets 78 p. Course Completion Work (Technology in Analysis and Development of System) - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2019.

This work presents a management system for the Social Center Casa do Piá, at the city of Ponta Grossa, state of Paraná. The entity serves 150 children and teenagers, aged 6 to 15 years old and has difficulties managing all information collected from its users and the internal processes of the social center, for this reason it needs a solution for the organization of this information and internal processes. Therefore, a solution proposed to aid and streamline its processes was a WEB / Mobile system. For the development of the system, the Laravel framework was used, along with the MariaDB database. For the constructionof the client Vue.js. framework was used. As Casa do Piá is a non-profit organization, it was decided to use open source technologies. The result was a system that streamlines the work of employees, avoiding duplication of data and facilitating the secure access to information.

Keywords: System WEB/Mobile. Laravel Framework. Vue.js Framework.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

ABASE	Aliança Brasileira de Assistência Social e Educacional
CSS	*Cascading Style Sheets*
CRAS	Centro de Referência de Assistência Social
CREAS	Centro de Referência Especializado de Assistência Social
DER	Diagrama Entidade Relacionamento
DRY	*Don't Repeat Yourself*
ECA	Estatuto da Criança e do Adolescente
XP	*Extreme Programming*
HTTP	*HyperText Transfer Protocol*
LOAS	Lei Orgânica da Assistência Social
MDS	Ministério do Desenvolvimento Social e Combate à Fome
MER	Modelo entidade relacionamento
ORM	*Object-relational mapping*
REST	*Representational State Transfer*
SPA	*Single-Page Applications*
SUAS	Sistema Único de Assistência Social
UTFPR	Universidade Tecnológica Federal do Paraná
W3C	*World Wide Web Consortium*

SUMÁRIO

1 INTRODUÇÃO

O Centro Social Casa do Piá presta atendimento para crianças, adolescentes e jovens, advindos de uma vasta região da cidade de Ponta Grossa no estado do Paraná, distribuídos no contra turno social em salas de convivência. O atendimento é prestado por meio de orientação pedagógica e educacional, atividades como oficinas e esportes e também consultas com assistente social e psicólogo (informação verbal) [1].

Desde 25 de março de 1998, o Centro Social Casa do Piá faz parte dos Centros Sociais da Aliança Brasileira de Assistência Social e Educacional (ABASE) – Irmãos do Sagrado Coração, que é uma Entidade de Assistência Social e Educacional, sem fins lucrativos (ABASE, 2016, p. 2). O objetivo geral da ABASE, bem como do Centro Social Casa do Piá é acolher e prestar assistência à criança e/ou adolescente e seus familiares com vistas à promoção humana, qualidade de vida, fortalecimento de vínculos e desenvolvimento do protagonismo e da autonomia (ABASE, 2016, p. 2).

A entidade “[...] atende a 150 crianças e adolescentes de 6 a 15 anos de ambos os sexos em situação de vulnerabilidade social, encaminhadas pela rede de proteção do município.” (Instituto dos Irmãos do Sagrado Coração, 2017). A mesma proporciona às crianças e/ou adolescentes um espaço de convivência, formação, participação e cidadania, desenvolvimento do protagonismo e da autonomia, a partir dos interesses, demandas e potencialidades dessa faixa etária. Estes programas estão em acordo às legislações específicas, como o Estatuto da Criança e do Adolescente (ECA) e a Lei Orgânica da Assistência Social (LOAS) (ABASE, 2016, p7).

A organização das equipes que trabalham direta e indiretamente com os jovens são: equipe técnica composta por assistente social, psicóloga, orientador pedagógico e educacional e equipe pedagógica composta por equipe técnica e educadores. Contando ainda com pessoal na área administrativa, de serviços gerais, serviços de limpeza e na área de alimentação (ABASE, 2016, p7).

Os funcionários da Casa do Piá gerenciam os usuários e os processos internos do centro social por meio de fomulários impressos, editor de planilhas e

[1] Informações fornecidas em entrevista, por direção e coordenadores do Centro Social Casa do Piá, em Ponta Grossa, em março de 2019.

serviços de armazenamento e compartilhamento de arquivos. A quantidade reduzida de funcionários, e um grande número de usuários gera dificuldades para registrar e localizar informações (informação verbal)[2].

O Centro Social Casa do Piá participa do programa de inclusão da Universidade Tecnológica Federal do Paraná (UTFPR). Os funcionários, diretor e coordenador, da Casa do Piá, buscaram junto ao professor Richard Duarte Ribeiro, orientador deste trabalho, uma solução para este problema.

Como solução para as dificuldades de gestão que a Casa do Piá enfrenta por meio das ferramentas utilizadas atualmente, propõe-se um sistema *WEB/Mobile*, que centralize a coleta de informações dos usuários e forneça processos informatizados para acompanhamento dos usuários do centro social. Agilizando o trabalho dos funcionários e facilitando a gestão da Casa do Piá.

1.1 MOTIVAÇÃO

Em entrevistas realizadas com a equipe da Casa Do Piá, foi constatada a necessidade de automatizar os processos internos, de forma a uniformizar o recolhimento dos dados, a fim de centralizá-los e protegê-los garantindo o sigilo de informações conforme prevê o Código de Ética dos/as Assistentes Sociais e do Código de Ética Profissional do Psicólogo (ABASE, 2016, p9)**.**

Em observação realizada *in loco*, utilizando a técnica etnográfica**,** a qual possibilita descobrir requisitos de *software* implícitos que refletem os processos reais em que as pessoas estão envolvidas (KOSCIANSKI, 2006, p.182), foi averiguada a constante repetição de informações pessoais, dos usuários atendidos pelo centro, em múltiplos cadastros destinados a relatórios de gestão interna e prestação de contas.

O processo atual demanda o uso de valiosos recursos humanos e financeiros. A coleta e manipulação das informações podem se tornar onerosas e demoradas, além de não garantir a segurança necessária e a consistência dos dados. Isso acontece devido a fragmentação das informações, que podem levar a erros nos relatórios.

---

[2] Informações fornecidas em entrevista, por direção e coordenadores do Centro Social Casa do Piá, em Ponta Grossa, em março de 2019.

1.2 OBJETIVOS

Visando a eficiência e segurança do controle e acompanhamento dos usuários da Casa do Piá, propõe-se a criação de um *software WEB/Mobile* para centralizar as informações e os processos internos, garantindo o sigilo e a privacidade das crianças e adolescentes, suas famílias e funcionários.

1.2.1 Objetivo Geral

Desenvolver um sistema para centralizar as informações e os processos internos da Casa do Piá, garantindo o sigilo e a privacidade de seus usuários.

1.2.2 Objetivos Específicos

- Levantar os requisitos do sistema por meio de técnicas de levantamento de requisitos;
- Utilizar diagramas da UML para realizar a modelagem do sistema;
- Utilizar o diagrama entidade relacionamento para modelagem do banco de dados do sistema;
- Utilizar a metodologia *eXtreme Programming* (XP) para o gerenciamento e desenvolvimento do sistema.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta o referencial teórico utilizado para o desenvolvimento deste trabalho.

2.1 LEVANTAMENTO DE REQUISITOS

A fim de desenvolver uma solução que facilite e agilize os trabalhos da Casa do Piá, será realizado o levantamento de requisitos. Requisitos são operações que o sistema deve realizar e refletem as necessidades dos clientes (funcionários da Casa do Piá) (SOMMERVILLE, 2011, p. 57).

Para atender as necessidades do centro social, será necessário alinhar os processos internos, visando obter um melhor entendimento das dificuldades e problemas enfrentados na gestão atual.

Para tal, será utilizada a técnica de entrevistas, na qual de acordo com Koscianski (2006), o analista propõe um ponto inicial à discussão e questiona de forma simples e pontual sobre o futuro sistema, para então evoluir para as características mais complexas (KOSCIANSKI, 2006, p.182).

Será utilizada a técnica de entrevistas semi-estruturadas, que combina perguntas abertas e fechadas, nesta o entrevistado pode discorrer sobre o tema proposto. O entrevistador deve seguir um conjunto de perguntas pré-definidas, que serão feitas em um contexto parecido ao de uma conversa informal (BONI; QUARESMA, 2005, p. 74).

2.2 MODELAGEM DO SISTEMA

Partindo do levantamento de requisitos, será realizada a modelagem do sistema. Serão utilizados diagramas de casos de uso, diagramas de atividades e o Diagrama Entidade Relacionamento (DER).

2.2.1 Diagrama de casos de uso

O diagrama de caso de uso é um contrato entre o cliente e analista com a intenção de validar as funcionalidades e características que um sistema deve conter (PRESSMAN; MAXIM, 2016). É desenvolvido visando gerar discussões que possam levar a elucidação de requisitos (COCKBURN, 2007).

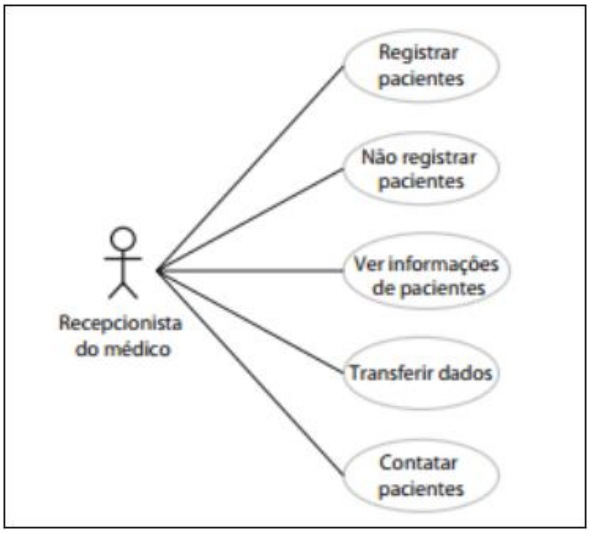
O diagrama de caso de uso é composto por cenário, ator, caso de uso e comunicação. O cenário é o conjunto de funcionalidades que o caso de uso contém, o ator é um tipo de usuário, o caso de uso representa uma funcionalidade do sistema e a comunicação é a associação de um caso de uso ao ator (RIBEIRO, 2012).

Casos de uso podem ser utilizados para documentação do *software*, planejamento de teste, para gerar novos casos de usos e validação dos atributos dos sistemas (WAZLAWICK, 2013).

Casos de uso podem ser relacionados, este relacionamento pode ser realizado por inclusão, o qual é utilizado para integrar um caso de uso em outro caso de uso, tornando-se uma parte lógica desse caso de uso. Ou por extensão, que é utilizado para expressar que um caso de uso será estendido por outro caso de uso em certas circunstâncias (WEILKIENS; OESTEREICH, 2007, p. 108).

A Figura 1 apresenta um exemplo de diagrama de caso de uso de um sistema de registro de pacientes.

Figura 1 - Caso de uso de um sistema de registro de pacientes



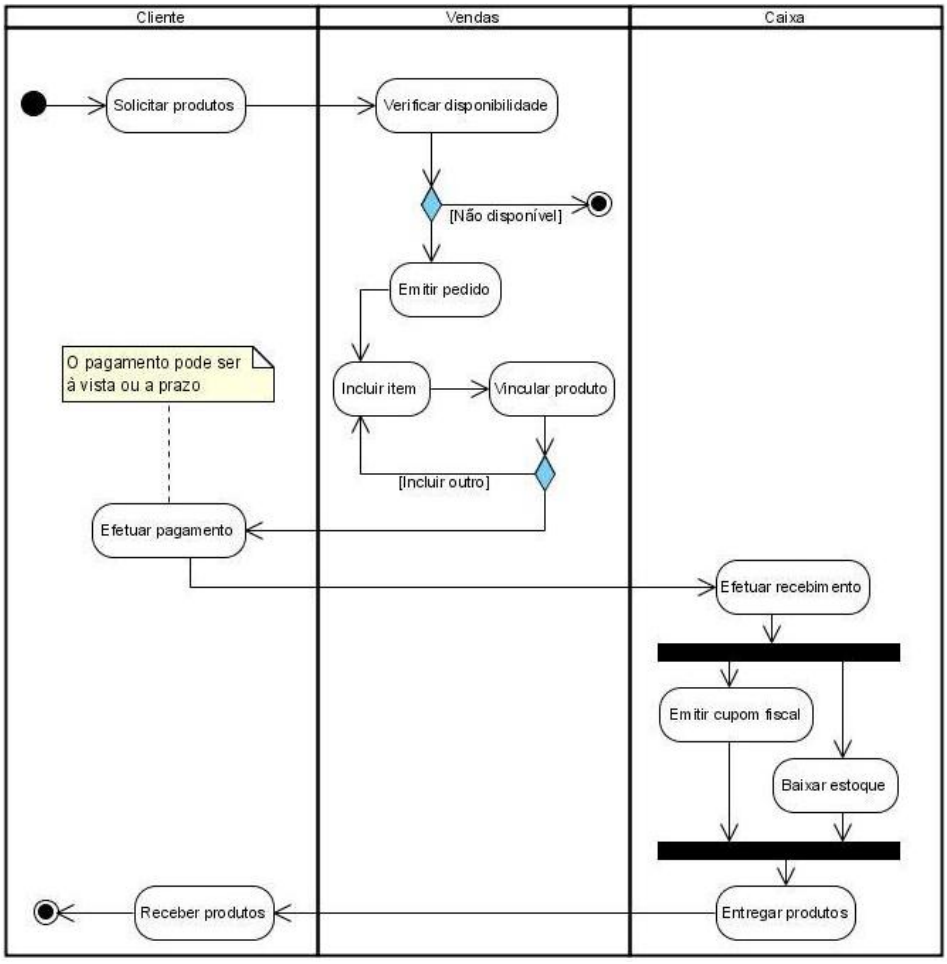
Fonte: Sommerville (2011)

2.2.2 Diagrama de atividades

De acordo com Booch, Rumbaugh e Jacobson (2006), um diagrama de atividade é um gráfico de fluxo, representando o fluxo de uma atividade para outra. O diagrama de atividades é apropriado para descrever interações entre os objetos do sistema (KIM, 2007). Resumidamente pode-se dizer que o diagrama de atividades exibe um fluxo sequencial de como o sistema funciona.

A Figura 2 ilustra um exemplo de diagrama de atividades, no qual são exibidas as atividades de uma venda.

Figura 2 - Exemplo de Diagrama de Atividade



Fonte: Divisão de Processamento de Dados[3]

[3] https://dtic.tjpr.jus.br/wiki/-/wiki/Governan%C3%A7a-TIC/Modelo+de+Atividades/pop_up

O diagrama exibe o fluxo de uma venda no sistema, são descritos todos os passos realizados pelo sistema desde a solicitação de produtos, verificação de disponibilidade, emissão de pedido, registro de pagamento, até a entrega do produto.

2.2.3 Modelo entidade relacionamento (MER)

Segundo Alves (2014), o MER é um modelo de dados de alto nível, utilizado na concepção do esquema conceitual do banco de dados. Foi criado para que projetistas de banco de dados possam representar a semântica que se encontra associada aos dados.

O modelo foi concebido em 1976, por Peter Chen, com base na teoria de bancos de dados relacionais de Edgard F. Codd. O principal conceito por trás do MER, está na definição de dois grupos de objetos que formam um negócio: Entidades e Relacionamentos. Os quais possuem uma ligação tão forte que não é possível tratar de um sem mencionar o outro (ALVES, 2014, p. 92).

2.3 DESENVOLVIMENTO DO SISTEMA

No processo de construção de um *software*, recomenda-se a utilização de um plano que permita gerenciar tópicos pertinentes ao seu desenvolvimento, como qualidade, controle, estabilidade, organização e cumprimento do prazo estabelecido para a entrega (PRESSMAN, 2011). Para o planejamento deste projeto foi utilizada a metodologia de desenvolvimento ágil XP.

O XP é uma metodologia ágil de desenvolvimento de *software*, concebido por Kent Beck, Ward Cunningham e Ron Jeffries, a metodologia é voltada para equipes de pequeno a médio porte, na qual os requisitos são vagos e mudam frequentemente (WILDT; et al., 2015, p. 16), pois seu objetivo é a codificação com foco menor nos processos formais de desenvolvimento e com uma maior disciplina de engenharia ágil de *software* para codificação e testes (WILDT; et al., 2015, p. 16).

O XP valoriza a automatização de testes, é flexível para a mudanças de requisitos e valorizando o *feedback* com o usuário e a qualidade do código-fonte final. O foco principal do XP é a criação de *software* de alta qualidade, abandonando

todo tipo de *overhead* de processo que não suporte diretamente a entrega de valor (WILDT; et al., 2015, p. 16).

2.3.1 Papéis do XP

Um time XP é formado por papéis com objetivos diferentes que se complementam, tais como: o papel de desenvolvedor, do cliente, do gerente, do *coach*, do testador, do *tracker* e do *cleaner* (WILDT; et al., 2015, p. 23).

Uma pessoa pode assumir mais de um papel, mas, é aconselhável que alguns papéis, como *coach* e gerente, não sejam assumidos por uma mesma pessoa, pois seus interesses são conflitantes (WILDT; et al., 2015, p. 23).

Os papéis do XP presentes no desenvolvimento deste projeto foram: desenvolvedor, cliente, testador e *cleaner* (WILDT; et al., 2015, p. 23). E estão descritos a seguir:

- **Desenvolvedor:** O desenvolvedor é um profissional multidisciplinar, capaz de trabalhar em todas as etapas do desenvolvimento de *software*, desde a escrita de histórias de usuário até o *deploy* em produção. No time XP, ele é um programador que estima as histórias de usuários e tarefas, quebra as histórias em tarefas, escreve testes, escreve código, automatiza processos de desenvolvimento e, gradualmente, aprimora o *design* do sistema.

- **Cliente:** O cliente define e prioriza as histórias de usuário, validando o produto desenvolvido por meio de testes de aceitação. É importante que ele esteja o mais próximo possível do time, com disponibilidade suficiente para conversar e tirar as dúvidas habituais.

- **Testador:** testador no time XP auxilia o cliente a escolher e escrever testes de aceitação, para, então, automatizá-los. Para o time de desenvolvimento, o testador não é responsável por detectar erros triviais, isto é papel dos próprios desenvolvedores. Ele pensa no teste e na qualidade do produto como um todo, considerando também os que

rodarão na integração contínua, auxiliando em par os programadores a resolver problemas do sistema.

- **Cleaner:** O *cleaner* é um membro do time que assume o papel de limpar o código, de encorajar os membros a praticar pequenas refatorações e reduzir a complexidade e acoplamento do código, tornando o código cada vez mais enxuto.

2.3.2 Histórias de Usuário

As histórias de usuários são utilizadas para descrever os requisitos de um sistema de forma ágil, por meio da comunicação face a face com o cliente. Para isso, cada história segue um ciclo de vida. Inicia-se com grandes histórias e segue-se o detalhamento da mesma de acordo com a sua prioridade (WILDT; et al., 2015, p. 37). De acordo com WILDT, et al (2015), as histórias de usuários são textuais, não necessitam de ferramenta específica, são de fácil compreensão aos envolvidos e são descritas em cartões de história.

É possível utilizar outras formas de documentação complementares, pois o XP não impede que isso aconteça, desde que sejam úteis e necessárias, tais como protótipos ou diagramas (WILDT; et al., 2015, p. 37).

Os cartões de histórias de usuário são utilizados como lembrete para conversação do time, bem como para confirmação do cliente (WILDT; et al., 2015, p. 39).

Com o intuito de deixar claro o objetivo das histórias de usuário, Ron Jefries criou o modelo 3C[4] (WILDT; et al., 2015, p. 39). No qual, cada letra C representa um aspecto crítico das mesmas:

> Cartão: as histórias de usuário são escritas em cartões (ou no tamanho de um cartão). Ele não conterá toda a informação de um requisito, mas serve para lembrar a todos do que este se trata. As prioridades e os custos também podem ser anotados neles.
>
> Conversação: o requisito é comunicado do cliente ao time por conversação (face a face), podendo ser suplementado por documentos.
>
> Confirmação: a confirmação da história dá-se por exemplos para criar testes de aceitação; ou seja, pelos critérios de aceitação. O cliente dirá ao time

[4] https://ronjeffries.com/xprog/articles/expcardconversationconfirmation/

> como aceitará cada história por meio de critérios de aceitação. Essa é uma ótima forma de suplementar a documentação. A preferência é de que esses critérios sejam automatizados pelo time. A história de usuário estará pronta quando todos eles estiverem passando nos testes de aceitação.

2.3.3 Padrões de Projeto

Cada padrão descreve um problema que ocorre repetidamente e então descreve a solução para aquele problema de forma que se possa utilizar a mesma solução inúmeras vezes, sem nunca repetir (GAMMA; el al., 1995, p. 14). Padrões de projeto têm 4 elementos essenciais:

- Nome: deve ser o mais fiel possível ao problema e sua solução;
- Problema: quando e em qual contexto deve-se aplicar o padrão de projeto;
- Solução: descreve o design da solução, suas concepções e relacionamentos;
- Consequências: descreve os ganhos e perdas ao aplicar um padrão de projeto específico.

Padrões de projeto não são convenções e sim certas soluções que foram repetidas inúmeras vezes e tiveram seu valor aprovado para um ou mais desenvolvedores (GAMMA; el al., 1995, p. 14).

2.4 ESTRUTURA DO TRABALHO

Este trabalho está dividido em sete capítulos. O capítulo 1 apresenta a introdução do trabalho, seus os objetivos e a motivação para o desenvolvimento. O capítulo 2 descreve o referencial teórico utilizado. O capítulo 3 apresenta os sistemas similares e o referencial teórico da legislação. O capítulo 4 apresenta as tecnologias envolvidas no desenvolvimento do trabalho. O capítulo 5 descreve a metodologia utilizada no desenvolvimento do sistema. O capítulo 6 descreve o desenvolvimento do sistema. Já o capítulo 7 apresenta os resultados obtidos no desenvolvimento do sistema. E, por fim, o capítulo 8 apresenta as conclusões e propostas para trabalhos futuros.

3 TECNOLOGIAS SIMILARES E LEGISLAÇÃO

Neste capítulo serão discutidos os seguintes assuntos: sistemas similares ao proposto neste trabalho; a legislação e normas que devem ser respeitadas pelo Centro Social; tecnologias utilizadas no desenvolvimento de sistema.

3.1 SISTEMAS SIMILARES

Realizou-se uma pesquisa no serviço de buscas da Google[5], a fim de encontrar *softwares* que contenham as mesmas soluções propostas por este projeto. Essa pesquisa trouxe alguns possíveis candidatos, dos quais realizou-se uma análise de suas funcionalidades.

O *software* de gestão Gesuas[6], é uma versão digital e *online* do Prontuário Sistema Único de Assistência Social (SUAS)[7] que permite registrar observações sobre os atendimentos, monitorar os encaminhamentos realizados, controlar a frequência em ações coletivas e gerar relatórios solicitados pelo Ministério do Desenvolvimento Social e Combate à Fome (MDS).

Apesar de todas suas funcionalidades, esta solução não possui facilidade de uso como uma das suas diretrizes, tornando a interação com o sistema algo que requer tempo e treinamento.

A solução SintegraSUAS[8], tem as mesmas funcionalidades que o anterior, com algumas funcionalidades extras, como diversos relatórios estatísticos e georreferenciamento[9] das famílias. Apesar de ter uma interface mais agradável, esta é uma solução comercial, não podendo ser utilizada livremente.

Já o *software* IDS Social[10], contém menos funcionalidades que os anteriores. Similarmente aos anteriores, o mesmo é um *software* comercial, limitando-se ao Prontuário SUAS e relatórios simplificados.

---

[5] https://www.google.com/
[6] https://www.gesuas.com.br/
[7] Ferramenta que permite o registro de características e de ações realizadas pelos CRAS e CREAS no atendimento a famílias e indivíduos
[8] https://www.sintegrasuas.com.br/
[9] Georreferenciamento ou georreferenciação de uma imagem ou um mapa ou qualquer outra forma de informação geográfica é tornar suas coordenadas conhecidas num dado sistema de referência
[10] https://www.ids.inf.br/ids-social/

É notável a existência de soluções que atendem certas necessidades, contudo, nenhuma delas é de livre acesso[11]. O foco das ferramentas pesquisadas é em Gestão Pública.

Diante dessas informações, conclui-se que desenvolver um novo *software* focado em facilidade de uso, de livre acesso e especializado em instituições individuais e seus processos internos é de grande utilidade pública.

3.2 LEGISLAÇÃO E NORMAS

Por se tratar de uma instituição de assistência social, faz-se necessário a adequação do sistema proposto a legislação e normas, seguidas pela Casa do Piá. Estas estão listadas no apêndice A.

[11] Software livre é uma expressão utilizada para designar qualquer programa de computador que pode ser executado, copiado, modificado e redistribuído pelos usuários gratuitamente.

4 TECNOLOGIAS ENVOLVIDAS

Nesta seção serão abordadas as tecnologias utilizadas para o desenvolvimento do cliente e do servidor do sistema.

4.1 PROTOCOLO HTTP

HyperText Transfer Protocol (HTTP), em português, Protocolo de Transferência de Hipertexto é um protocolo de comunicação que é responsável pela transferência de informações entre um cliente e um servidor (TANENBAUM, 2003).

Uma requisição HTTP não tem estado, o que significa que cada solicitação é tratada de forma independente. Isso significa que, com uma sequência de solicitação e resposta, a comunicação é concluída (LOPES, 2016, p. 18).

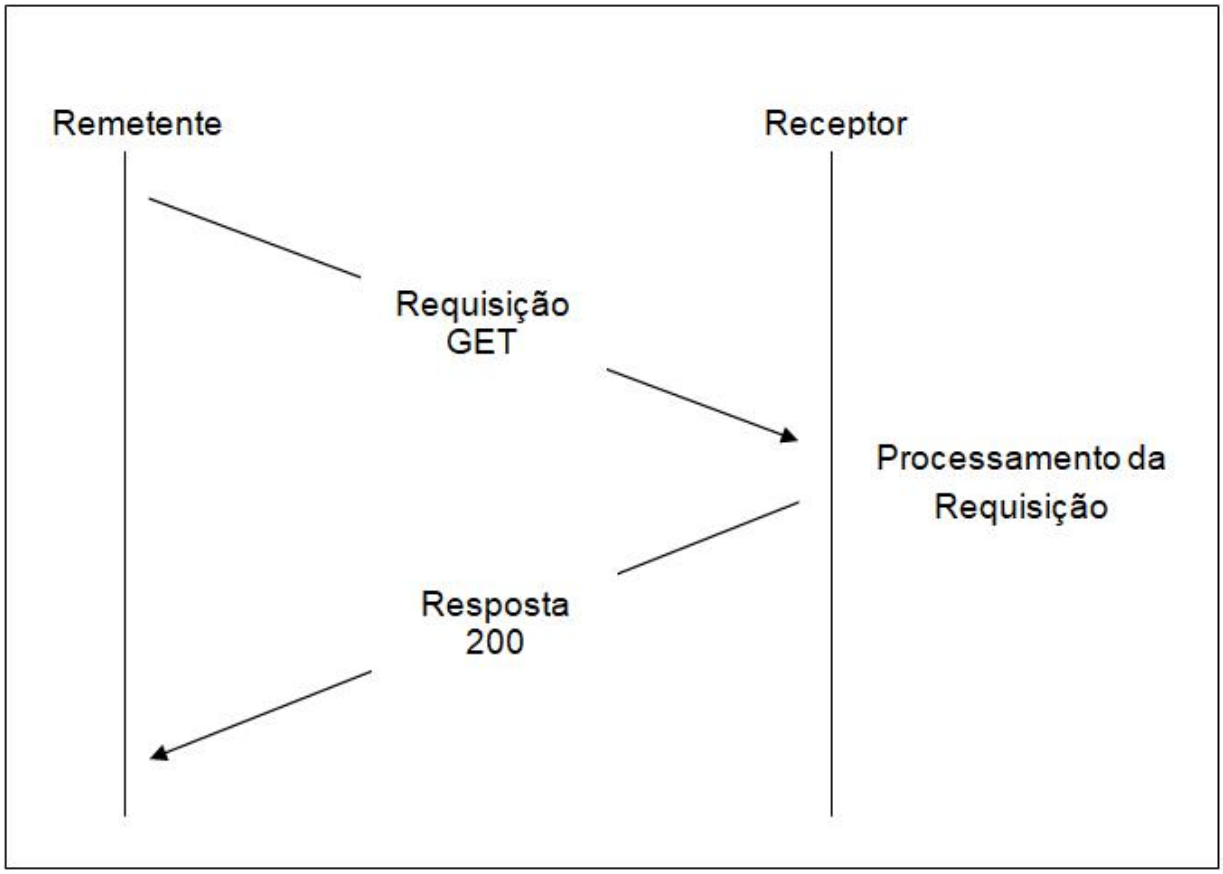
O HTTP na sua versão 1.1 define oito métodos de requisição:

- GET
- POST
- PUT
- DELETE
- OPTIONS
- HEAD
- TRACE
- CONNECT

Cada um desses métodos possui particularidades e objetivos dentro da sua utilização (SAUDATE, 2014, p. 15).

A Figura 3 apresenta um exemplo simples de comunicação por meio do protocolo HTTP, na qual um remetente envia uma requisição GET, o receptor processa a requisição e retorna uma resposta com o o código de *status* 200, indicando sucesso.

Figura 3 - Requisição HTTP



Fonte: Adaptado de (LOPES, 2016)

4.2 CLIENTE

Nesta seção serão descritas as tecnologias utilizadas para o desenvolvimento do cliente da aplicação, responsável por receber as entradas dos usuários e exibir informações aos mesmos. As principais tecnologias utilizadas são: HTML, CSS, Javascript, Vue.js.

4.2.1 HTML

A linguagem foi definida em especificações no ano de 1993 e é mantida pela *World Wide W*eb *Consortium* (W3C)[12]. *HyperText Markup Language* (HTML), é uma linguagem utilizada para criar e representar visualmente uma página *WEB*, sendo o elemento mais básico da *World Wide Web* (W3C, 2016).

De acordo com Marcondes (2000), os comandos de HTML são chamados de *tags*, que significa etiquetas em inglês. Os documentos na linguagem HTML são

[12] https://www.w3.org/

criados com o uso de *tags* e os navegadores identificam as mesmas para saber quais elementos estão na página (MARCONDES, 2000).

O HTML foi desenvolvido para padronizar elementos fragmentados que anteriormente eram compartilhados por seus autores. São exemplos disso as fotos, sons e textos (KENNEDY; MUSCIANO, 2006).

O HTML também possibilitou, por meio de *links*, que documentos fossem referenciados a outros documentos localizados em lugares diferentes (KENNEDY; MUSCIANO, 2006).

Neste trabalho será utilizada a versão mais recente da atualidade, que é HTML5.

4.2.2 CSS

Cascading Style Sheets (CSS) é uma linguagem *stylesheet* que descreve a apresentação de um documento HTML ou XML (MOZILLA, 2016).

O CSS possibilita personalizar os elementos de uma página HTML, ao ser interpretado pelo navegador, adiciona estilo ao documento, como por exemplo: fontes, cores, espaçamento, entre outros (MEYER, 2006).

A aparência do documento estilizado com o CSS é melhor do que o HTML (MEYER, 2006). De acordo com W3C, também é possível com o CSS adequar as páginas a diferentes tamanhos e telas de dispositivos. Neste trabalho será utilizada a versão CSS3.

4.2.3 JavaScript

JavaScript é uma linguagem de programação interpretada, que utiliza funções de primeira classe, onde funções podem ser passadas como parâmetros para outras funções e retornadas como resultado de uma função (MOZILLA, 2016). A linguagem funciona em todos os dispositivos que tenham um navegador *WEB*, seja em computadores, *smartphones* ou *tablets.*

Os navegadores são capazes de executar aplicações desenvolvidas nesta linguagem, por isso, segundo Flanagan (2011), o JavaScript é considerada a linguagem mais ubíqua da história.

A linguagem também é muito utilizada para transformar a interação com páginas na *WEB*, tornando a experiência do usuário mais dinâmica, como por exemplo, a validação de informações introduzidas pelos usuários em formulários

(NEGRINO; SMITH, 2001). Pelo fato de ser uma linguagem de *script* é considerada robusta e eficiente (FLANAGAN, 2011).

Para o desenvolvimento do trabalho será utilizada a versão ECMAScript 6.

4.2.4 VUE.JS

O *Vue.js* é uma biblioteca para o desenvolvimento de interfaces *WEB* interativas. O objetivo do *Vue.js* é ser simples, reativo, baseado em componentes, compacto e expansível (SCHMITZ, 2016, p. 2).

O Vue.Js em si, não é um *framework* completo, ele é focado apenas na camada de visualização de uma aplicação. Isso faz com que seja muito fácil integrar com outras bibliotecas ou projetos já existentes.

Por outro lado, quando utilizado em conjunto de ferramentas adequadas e bibliotecas de suporte, o Vue.js é capaz de criar *Single-Page Applications* (*SPA*)[13], ou aplicações de página única (KYRIAKIDIS; MANIATIS, 2016, p. 240).

Para complementar a utilização do Vue.Js, utilizou-se algumas bibliotecas que serão descritas a seguir:

- **Axios[14]:** é um cliente HTTP baseado em Promises[15]. A biblioteca é uma API que interage com *XMLHttpRequest* possibilitando fazer requisições assíncronas;
- **Vuetifyjs[16]:** é um *framework* que fornece componentes limpos, semânticos e reutilizáveis para o Vue e facilita a criação de um aplicativo.
- **Router[17]:** é o roteador oficial do Vue.js, utilizado para simplificar a criação de aplicações de uma página;
- **Vuex[18]:** é uma biblioteca para gerenciamento de estados para de aplicativos Vue.js, que serve para centralizar todos os componentes em um aplicativo, com regras que garantem que o estado possa ser modificado de maneira previsível.

---

[13] Single-page application, são aplicações web que rodam em uma única página.
[14] https://github.com/axios/axios
[15] *Promise*, em portugûes promessa, é um objeto usado para processamento assíncrono. Um Promise representa um valor que pode estar disponível agora, no futuro ou nunca.
[16] https://vuetifyjs.com/pt-BR/getting-started/why-vuetify
[17] https://router.vuejs.org/
[18] https://vuex.vuejs.org/

4.3 REST

REST significa *Representational State Transfer*, ou em português, transferência de estado representativo. É um método de desenvolvimento de *web services* que teve origem na tese de doutorado de Roy Fielding, que por sua vez, é co-autor do protocolo HTTP (SAUDATE, 2014, p. 4).

Fieldings é co-autor do HTTP, assim, é notável que o protocolo REST seja guiado pelas melhores práticas do HTTP (SAUDATE, 2014, p. 4), como:

- Uso adequado de métodos HTTP;
- Uso adequado de *Uniform Resource Locator* URL;
- Uso de códigos padronizados para mensagens de *status* (sucesso ou falha);
- Uso adequado de cabeçalhos HTTP;
- Interligações entre vários recursos.

O REST tem uma característica conhecida como *stateless*, isto significa que um sistema REST não armazena nenhum dado entre as requisições de um cliente para o servidor. Por este motivo, cada requisição precisa conter toda a informação necessária para compreender a requisição, independente de informações obtidas previamente. (FIELDINGS, 2000, p. 78).

4.4 *BACK-END*

Nesta seção serão conceituadas as tecnologias para desenvolvimento do servidor da aplicação, que são: linguagem de programação *PHP, Framework* Laravel e banco de dados MariaDB, que são ferramentas de código aberto.

4.4.1 Linguagem de programação PHP

O PHP[19], é uma linguagem criada por Ramus Ledorf, em 1995 (MILETTO, BERTAGNOLLI, 2014, p. 162).

O PHP representa um acrônimo recursivo para *Hypertext Preprocessor*, é uma linguagem de *script* de código aberto, que pode ser embutida dentro do HTML e é adequada para o desenvolvimento web (PHP, 2019).

O código PHP é executado no servidor da aplicação e a interpretação dos *scripts* PHP, possibilita a geração de códigos HTML, JavaScript, além de

[19] https://www.php.net/manual/pt_BR/intro-whatis.php

documentos PDF, XML, imagens ou textos, os quais podem ser enviados ao cliente ou simplesmente armazenados no servidor (MILETTO, BERTAGNOLLI, 2014, p. 162).

4.4.2 *Framework* Laravel

O Laravel é um *framework* de desenvolvimento web MVC escrito em PHP. Segundo MCCOOL (2012). O Laravel foi projetado para melhorar a qualidade do *software*, o que implicará na redução de custos de desenvolvimento e manutenção, além de tempo de implementação (MCCOOL, 2012, p. 3).

O Laravel permite modularidade de código por meio de um conjunto de drivers e seu sistema de pacotes. É possível estender facilmente as funcionalidades de *cache*, sessão, banco de dados e autenticação por meio do uso de pacotes. Assim sendo, é possível empacotar qualquer tipo de código, seja para reutilização ou para alimentar a comunidade do Laravel (MCCOOL, 2012, p. 3).

O *framework* também conta com um conjunto avançado de ferramentas para interação com bancos de dados. São eles:

- *Migrations*: em português migrações, permitem projetar e modificar facilmente um banco de dados de maneira independente de plataforma. O Laravel suporta os seguintes bancos de dados: MySQL, PostgreSQL, MSSQL e SQLite (MCCOOL, 2012, p. 3).
- *Eloquent*: é a implementação do *ActiveRecord*, que permite interagir com um banco de dados de uma maneira orientada a objetos, podemos criar, recuperar, atualizar e excluir os registros do banco de dados sem que seja necessário utilizar as instruções SQL. Segundo MCCOOL (2012), o *Eloquent* ORM também fornece gerenciamento de relacionamento poderoso e pode até mesmo lidar com a paginação automaticamente (MCCOOL, 2012, p. 3).

Outro recurso do *framework* Laravel, é o *Artisan*, com o qual é possível interagir com seu aplicativo para executar migrações, testes de unidade e tarefas agendadas (MCCOOL, 2012, p. 4).

E por fim, o *Routing*, que é o sistema de roteamento do Laravel, que permite gerenciar facilmente as URLs do seu *site* (MCCOOL, 2012, p. 4).

4.4.3 MariaDB

MariaDB[20] foi iniciada em 2009 por Monty Widenius, o autor original do MySQL, após o projeto antigo ter sido adquirido pela Oracle. A primeira versão do MariaDB foi baseada no MySQL 5.1, e as melhorias no código base do MySQL são regularmente mescladas no projeto MariaDB.

O MariaDB é um banco de dados relacional rápido, escalável, robusto e de código aberto. Tem a função de transformar dados em informações estruturadas e fornece uma interface SQL para acessar os dados (MARIADB, 2019). A versão utilizada neste trabalho é a 10.3.15.

[20] https://mariadb.org/about/

5 METODOLOGIA DO DESENVOLVIMENTO

Nesta seção será abordada a metodologia utilizada para o desenvolvimento da aplicação. Na subseção 1 será apresentado como foi realizado o levantamento de requisitos. Na subseção 2 será apresentada a modelagem do sistema. A subseção 3 descreverá a metodologia ágil *Extreme Programming.* E, por fim, a subseção 4. apresentará as restrições do projeto.

5.1 LEVANTAMENTO DE REQUISITOS

A Casa do Piá buscou junto a UTFPR, uma forma de otimizar o atendimento dos usuários do Centro Social. Para entender melhor o problema, foram realizadas reuniões com os funcionários da Casa do Piá, aplicando as técnica de entrevistas e a técnica etnográfica para o levantamento de requisitos.

Para entender melhor o problema, foram realizadas reuniões com os funcionários do centro, buscando as documentações que informam sobre os seus processos internos (ABASE, 2016), as orientações do Ministério do Desenvolvimento Social para a construção do prontuário do cadastro (BRASIL, 2014) e como a Casa do Piá atua.

Primeiramente buscou-se entender como é prestado o atendimento para os usuários, o qual é realizado por orientação pedagógica e educacional, atividades como oficinas e esportes e também consultas com assistente social e psicólogo. Estas atividades são realizadas no contraturno das escolas que os usuários frequentam.

Após o entendimento dos serviços prestados, direcionou-se a entrevista para entender as suas dificuldades na gestão por meio de ferramentas utilizadas atualmente. A gestão é dividida entre as seguintes funções do centro social: Assistente Social, Coordenador, Diretor, Psicólogo, e Professores.

A assistente social, faz o cadastro dos usuários e registra as consultas por meio de formulários impressos e depois repassa essas informações para editores de planilha. A psicóloga registra as consultas em editores de texto e editores de planilhas. Os professores registram a presença dos usuários nas atividades por meio de formulários impressos e entregam para o coordenador registrar as listas de presença em editores de planilhas. Estes arquivos são compartilhados entre os

funcionários por meio de um serviço de armazenamento e compartilhamento de arquivos.

A assistente social e a psicóloga tratam de assuntos pessoais dos usuários atendidos pelo centro, estas informações são registradas em arquivos que só elas têm acesso por questões de privacidade, mas o centro social precisa fazer o controle de consultas com assistente social e psicólogo por mês.

O coordenador e o diretor geram relatórios para prestação de contas com a Secretaria Municipal de Assistência Social, relatórios de frequência mensal de usuários e quantidade mensal de usuários, estes relatórios são gerados manualmente consultando as informações das planilhas compartilhadas em um serviço de armazenamento e compartilhamento de arquivos.

A partir da compreensão dos processos do centro social, verificou-se que os funcionários tem bastante retrabalho nos cadastros de usuários e presença nas atividades, e também que a gestão por editores de planilha dificulta o trabalho de consultar informações dos usuários e gerar relatório. Outro problema encontrado foi a segurança das informações sensíveis dos usuários.

Foi idealizada como solução, a criação de um sistema *Web/Mobile*. Com o objetivo de gerenciar os processos internos, referente ao cadastro e acompanhamento das crianças e adolescentes participantes do centro social, no qual cada funcionário tem seu papel e acesso dentro do sistema, limitado a função que exerce no centro. Definiu-se como prioridade no desenvolvimento, garantir a segurança das informações e evitar a repetição desnecessária dos dados.

De acordo com a análise dos processos da Casa do Piá, definiu-se como requisitos mínimos do sistema as seguintes funcionalidades:

- Cadastro de usuários do sistema: cadastro de usuários que poderão acessar o sistema;
- Cadastro de grupos de permissões: por conter informações pessoais e sensíveis dos usuários, é necessário definir perfis que têm acessos diferenciados dentro do sistema, para isso, será necessário o cadastro de grupos de permissões;
- Cadastro de usuários atendidos: cadastros dos usuários que são atendidos pela Casa do Piá, que inclui as informações dos usuários, familiares, escolaridade e histórico de saúde;
- Cadastro de funcionários: cadastro de funcionários do centro social;

- Cadastro de atividades: cadastro das atividades e oficinas que a Casa do Piá oferece para os usuários;
- Registro de presença: registro da presença dos usuários nas atividades ou oficinas;
- Registro de acompanhamento: registro das consultas com assistente social ou psicólogo. Estes cadastros são confidenciais e é necessário garantir a privacidade dos usuários;
- Geração de relatórios: geração dos relatórios “Frequência mensal de usuários” e “Quantidade mensal de usuários”, de acordo com filtros inseridos pelo usuário;

5.2 EXTREME PROGRAMMING

Após o levantamento de requisitos, iniciou-se o planejamento do projeto pela metodologia XP.

5.2.1 Papéis do XP

Primeiramente descreveu-se quem desempenhou cada papel do XP. Conforme apresentado na seção 2.3.1, utilizou-se os seguintes papéis:

- Desenvolvedor: os autores deste trabalho, Giovane e Thiago, foram os desenvolvedores do sistema;
- Cliente: o Centro Social Casa do Piá;
- Testador: o papel de testador ficou a cargo de Giovane;
- *Cleaner*: o papel de testador ficou a cargo de Thiago.

5.2.2 Histórias de Usuários

A partir dos requisitos levantados, criou-se as histórias de usuário, as quais estão descritas no quadro 1:

Quadro 1 – Histórias de Usuário

Histórias de Usuário		
Como um	**Eu quero**	**Para que**
Coordenador	Cadastrar um usuário no sistema	Eu possa disponibilizar acesso ao sistema para novos funcionários
Coordenador	Cadastrar um grupo de permissões	Eu possa controlar os acessos dos usuários dentro do sistema, de acordo com a função desempenhada no Centro Social
Assistente social	Cadastrar um usuário atendido pela Casa do Pía no sistema	Eu possa registrar o usuário no sistema
Coordenador	Cadastrar um funcionário no sistema	Para que eu possa registrar os funcionários no sistema e suas funções dentro da Casa do Piá
Funcionário	Cadastrar no sistema as atividades realizadas no centro social	Eu possa registrar no sistema as atividades ofertadas pelo Centro Social
Professor	Registrar as presenças dos usuários nas atividades	Eu possa ter o controle da participação dos usuários nas atividades ofertadas pela Casa do Piá
Assistente social / Psicólogo	Registrar o acompanhamento do usuário no sistema	Eu possa registrar o acompanhamento de consultas e as ocorrências do usuário dentro do Centro Social
Funcionário	Gerar relatórios	Eu possa consultar as informações do Centro Social de forma rápida e gerar os relatórios para prestação de contas

Fonte: Autoria Própria

5.3 MODELAGEM DO SISTEMA

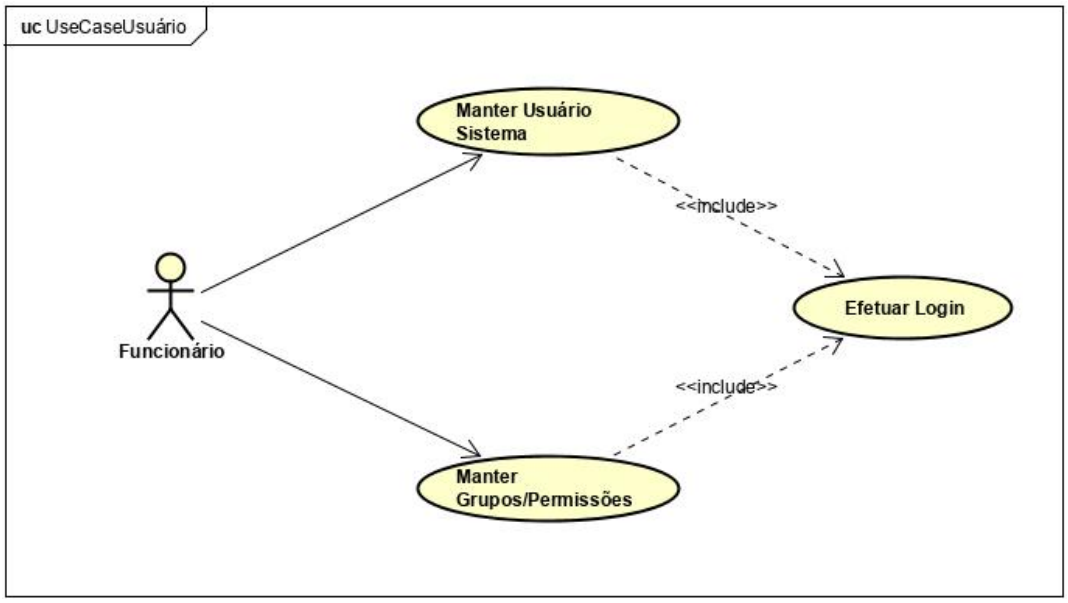
Para o desenvolvimento dos diagramas de caso de uso e diagramas de atividade, foi utilizada a ferramenta Astah[21], que é uma ferramenta de modelagem UML. Para o desenvolvimento do DER foi utilizada a ferramenta BRmodelo[22], que é uma ferramenta de código aberto e totalmente gratuita voltada para ensino de modelagem de banco de dados relacionais.

5.3.1 Diagrama de Caso de Uso

A Figura 4 apresenta o diagrama de caso de uso “UseCaseUsuário”, este foi desenvolvido para o controle de usuários, grupos e permissões do sistema. O ator “Funcionário” pode realizar as seguintes ações no sistema: cadastrar, consultar e editar usuário e cadastrar, consultar e editar grupo de permissões.

[21] http://astah.net/
[22] http://www.sis4.com/brModelo/

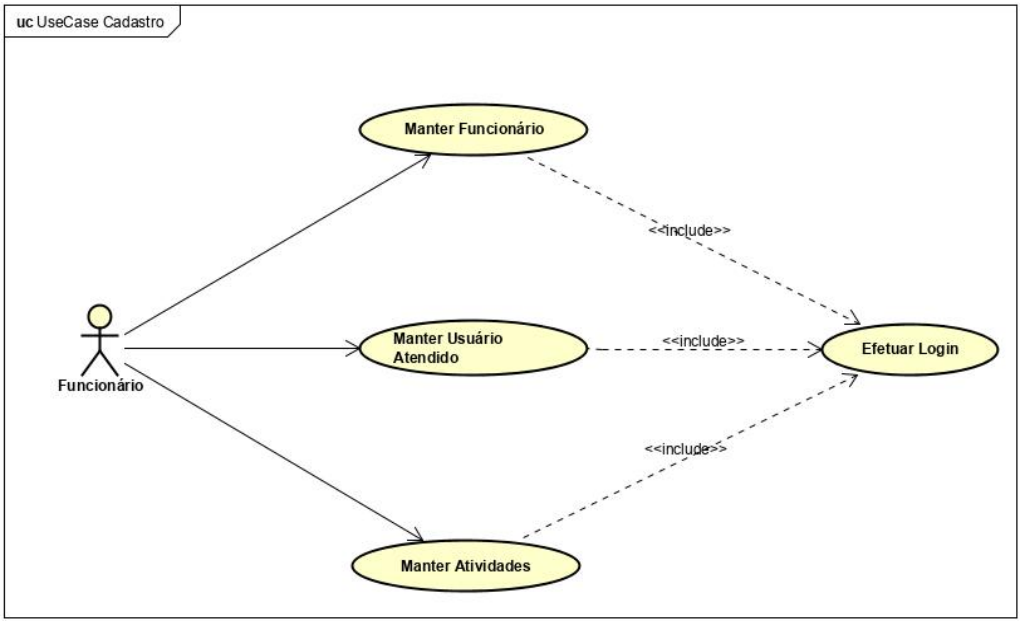
Figura 4 - Diagrama de Casos de Uso de Usuários e Grupos de Permissões



Fonte: Autoria Própria

A Figura 5 apresenta o diagrama de caso de uso “UseCaseCadastro”, este diagrama foi desenvolvido para o controle de cadastros de funcionários, usuários e atividades da Casa do Piá. O ator “Funcionário” pode realizar as seguintes ações no sistema: cadastrar, consultar e editar funcionário, cadastrar, consultar e editar usuário e cadastrar, consultar e editar atividades.

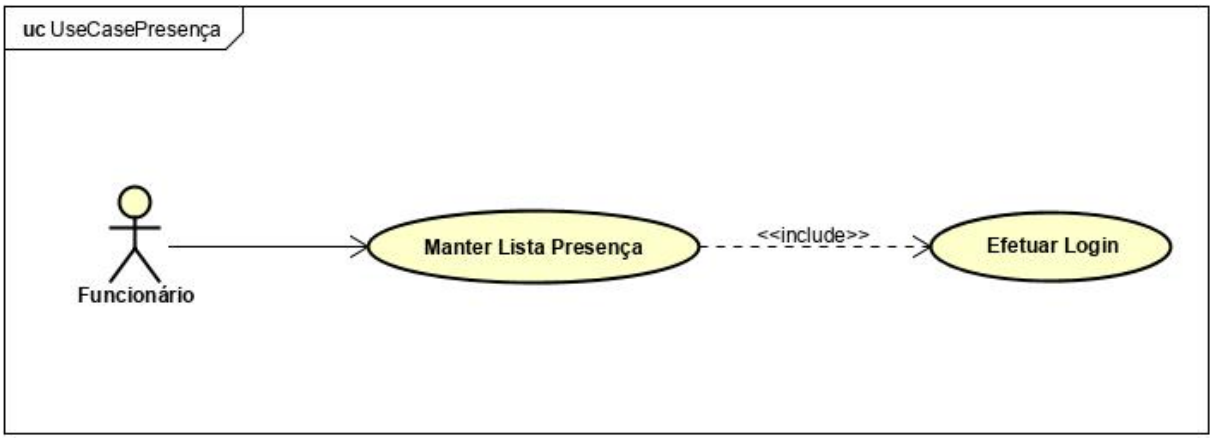
Figura 5 - Diagrama de Casos de Uso do Módulo Cadastro



Fonte: Autoria Própria

A Figura 6 apresenta o diagrama de caso de uso “UseCasePresença”, este diagrama foi desenvolvido para o controle de presença dos usuários nas atividades da Casa do Piá. O ator “Funcionário” pode realizar as seguintes ações no sistema: consultar atividade e registrar presença.

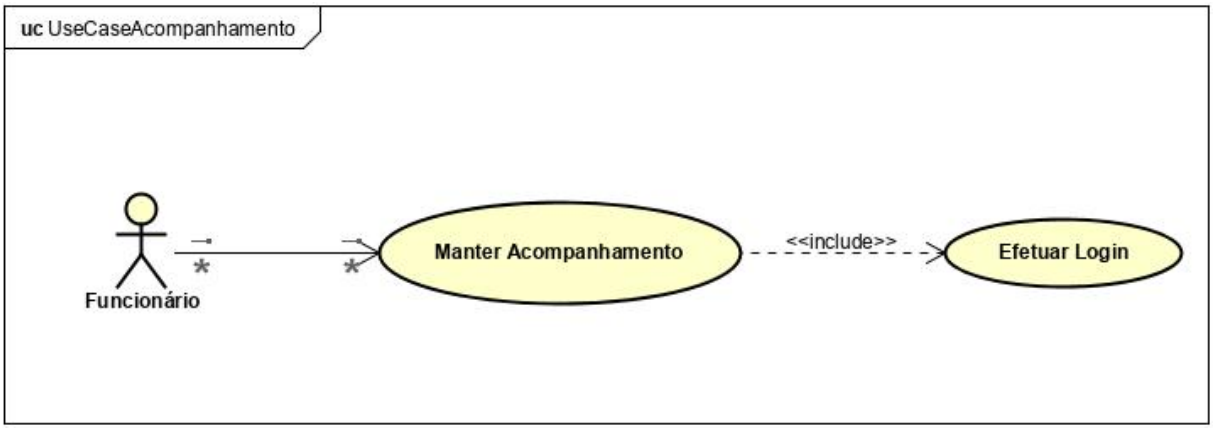
Figura 6 - Diagrama de Casos de Uso do Registro de Presença



Fonte: Autoria Própria

A Figura 7 apresenta o diagrama de caso de uso “UseCaseAcompanhamento”, este diagrama foi desenvolvido para o controle de consultas com assistente social e psicólogo. O ator “Funcionário” pode realizar as seguintes ações no sistema: consultar usuário, registrar atendimento e consultar atendimento.

Figura 7 - Diagrama de Casos de Uso de Registro de Acompanhamento



Fonte: Autoria Própria

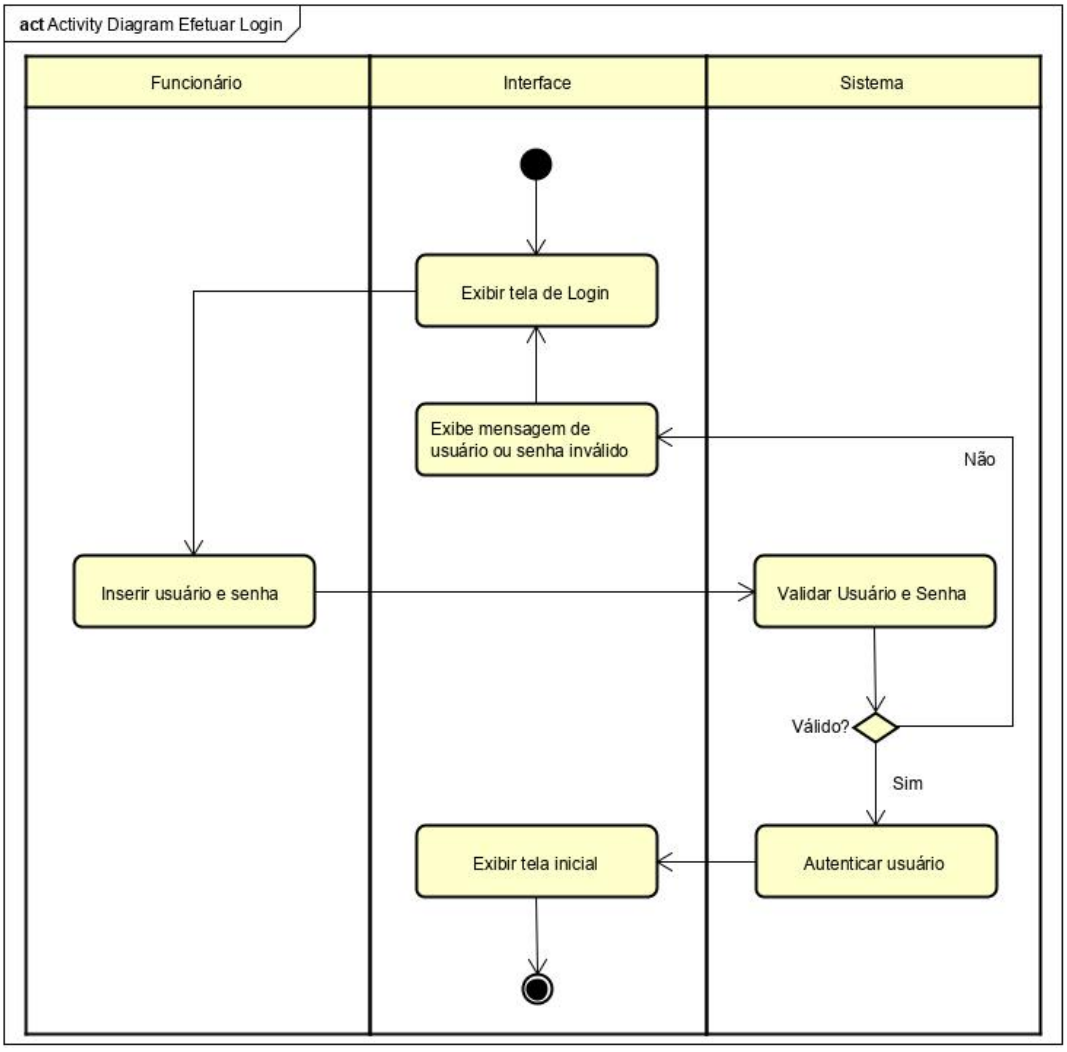
5.3.2 Diagrama de Atividades

Nesta seção descreveu-se os diagramas de atividades desenvolvidos para modelagem do sistema.

5.3.2.1 Diagrama de atividades - Efetuar Login

A Figura 8 apresenta o diagrama de atividades para autenticação do funcionário no sistema. Inicialmente, o sistema exibe a tela de *login*, onde será inserido o usuário e senha para autenticação. O sistema faz a validação dos dados e, caso falhe, mostra a mensagem de usuário ou senha inválido, onde o usuário poderá tentar logar novamente. Caso a validação seja bem sucedida o sistema autentica o usuário e redireciona para a tela inicial do sistema.

Figura 8 - Diagrama de Atividades Efetuar *Login*



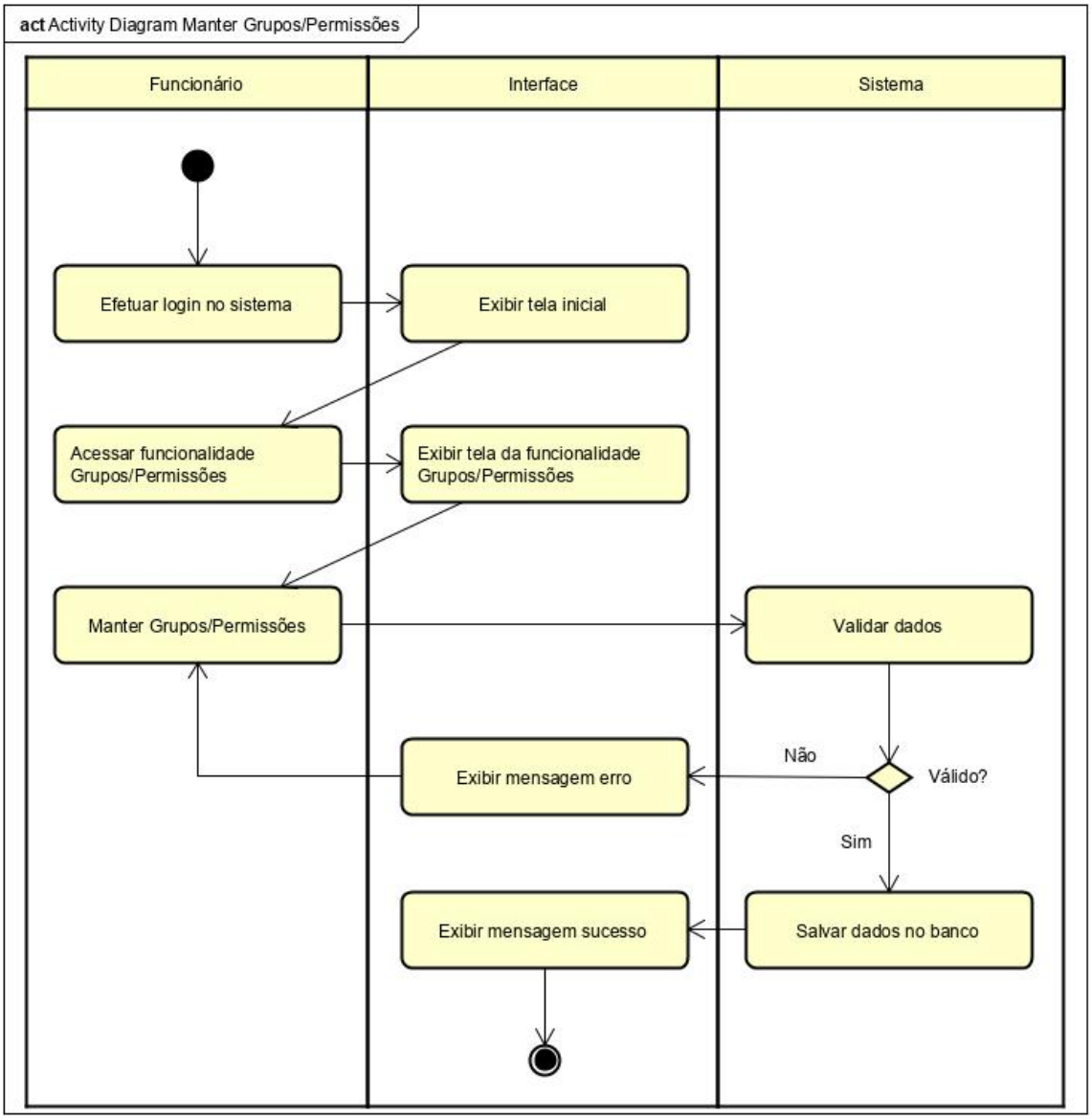
Fonte: Autoria Própria

5.3.2.2 Diagrama de atividades - Manter Grupos/permissões

A Figura 9 apresenta o diagrama de atividades para o cadastro de grupos de permissões. Primeiramente o usuário vai acessar o sistema e logo em seguida será exibida a tela inicial, a qual terá um menu para acessar a funcionalidade “Grupos/Permissões”. Ao acessar a funcionalidade, o sistema irá exibir na tela uma listagem dos grupos já cadastrados.

Nesta tela, será possível também cadastrar um novo ou editar um grupo já existente, ao realizar uma destas ações o sistema valida os dados. Caso a validação falhe, exibirá uma mensagem de erro para o usuário, caso a validação seja bem sucedida, os dados serão gravados no banco e exibirá uma mensagem de sucesso para o usuário.

Figura 9 - Diagrama de Atividades Manter Grupo/Permissões



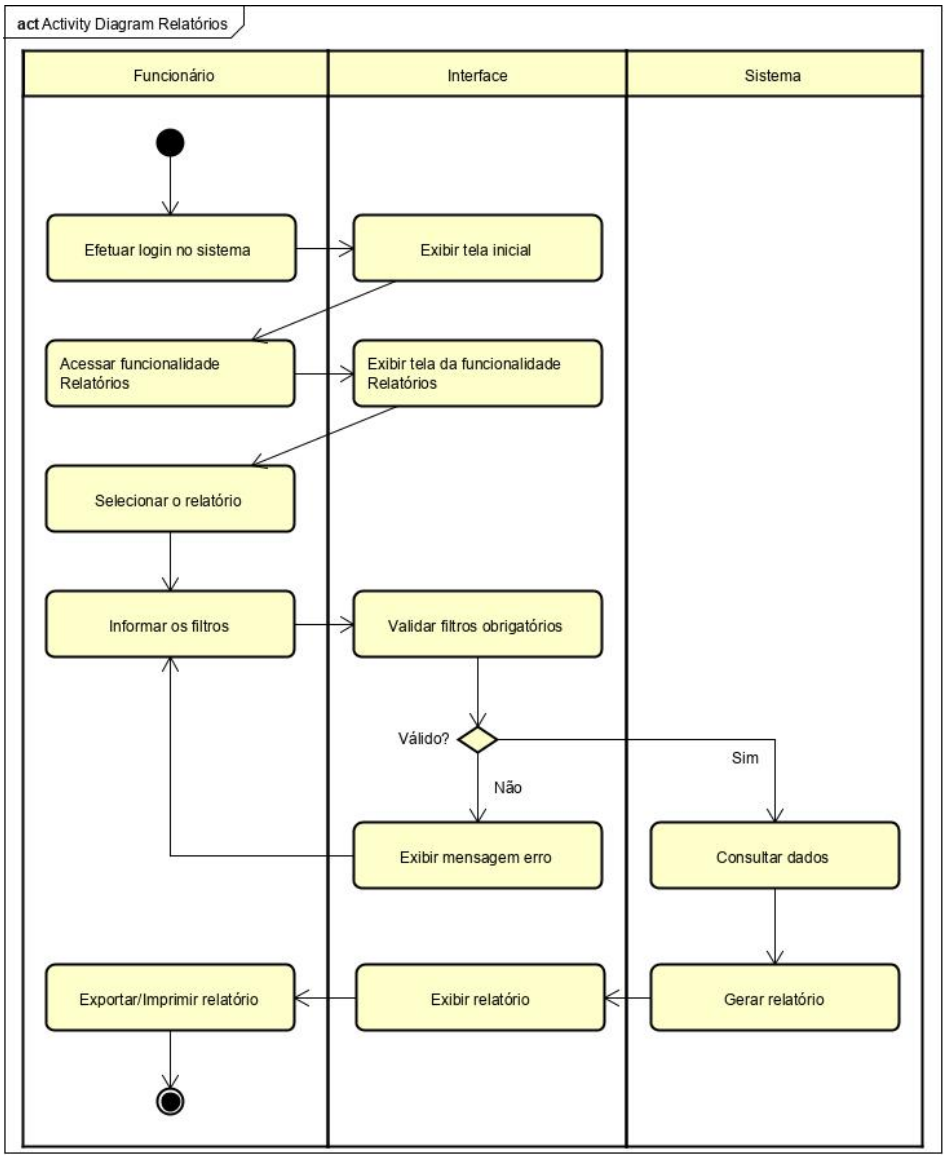
Fonte: Autoria Própria

5.3.2.3 Diagrama de atividades - Relatórios

A Figura 10 apresenta o diagrama de atividades para gerar relatórios. Primeiramente o usuário vai acessar o sistema e logo em seguida será exibida a tela inicial, a qual terá um menu para acessar a funcionalidade “Relatórios”. Ao acessar a funcionalidade, será possível selecionar o relatório que deseja gerar e informar os filtros necessários para a consulta das informações.

Logo em seguida, o sistema validará os campos obrigatórios para geração do relatório. Caso a validação falhe, exibirá uma mensagem de erro para o usuário, caso a validação seja bem sucedida, o sistema irá consultar as informações no banco, gerar o relatório e disponibilizar para o usuário baixar ou imprimir o relatório.

Figura 10 - Diagrama de Atividades Relatórios



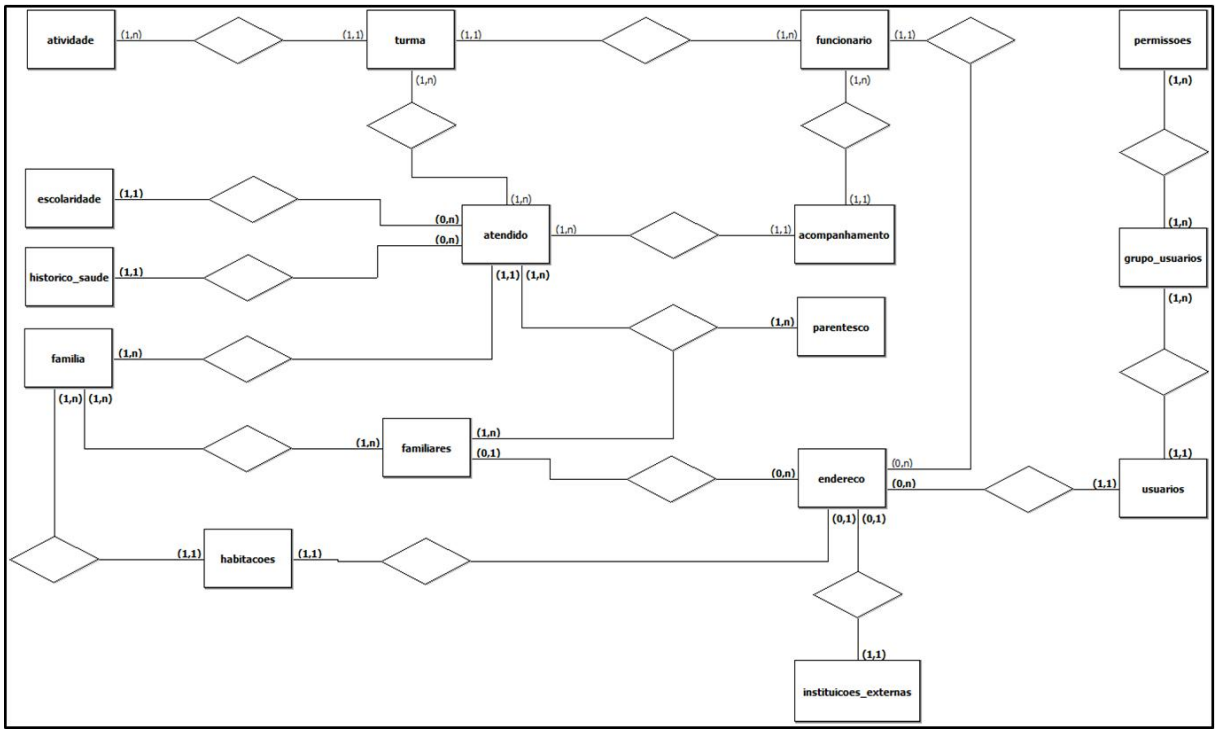
Fonte: Autoria Própria

Foram desenvolvidos os diagramas de atividades para cada caso de uso ilustrado na seção 5.3.1, os quais encontram-se em sua totalidade no apêndice B.

5.3.3 Diagrama Entidade Relacionamento

A Figura 11 apresenta o diagrama entidade relacionamento que foi desenvolvido como modelo para criação do banco de dados.

Figura 11 – Diagrama Entidade Relacionamento



Fonte: Autoria Própria

Com base no diagrama entidade relacionamento, utilizou-se os *Migrations* do *framework* Laravel para criação das tabelas no banco de dados, no qual, cada *Migration* a ser criada é uma classe da linguagem PHP. A figura 12 apresenta um exemplo do *Migration*:

Figura 12 - Classe do Framework Laravel que Representa a Tabela Família

```php
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateFamiliesTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('families', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->integer('nis_familiar');
            $table->string('cras_referencia')->nullable();
            $table->boolean('bolsa_familia');
            $table->unsignedBigInteger('external_institution_id')->nullable();
            $table->foreign('external_institution_id')->references('id')->on('external_institutions');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('families');
    }
}
```

Fonte: Autoria Própria

5.4 RESTRIÇÕES DO PROJETO

Este projeto fornece funcionalidades para realizar a gestão de Centros Sociais. Porém, o *software* não tem um módulo para realizar a gestão financeira da instituição, pois atualmente a Casa do Piá já utiliza outra solução com esta finalidade.

Referente às requisições, são aceitas somente requisições HTTP que estejam de acordo com o modelo de arquitetura REST. A comunicação entre cliente e servidor, se dá por arquivos JSON, não suportando requisições com conteúdo XML.

E por fim, a solução não se compromete a funcionar sem uma conexão com a Internet.

6 DESENVOLVIMENTO DO SISTEMA

Neste capítulo será exposta a estrutura da aplicação, cada uma das suas partes e funcionamento geral de seus domínios.

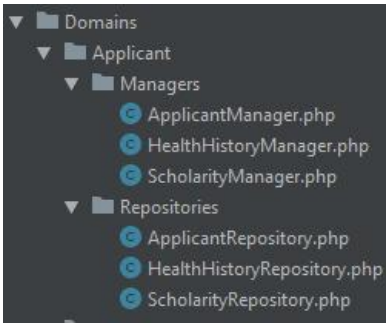
6.1 VISÃO GERAL DA ESTRUTURA DO CÓDIGO

O desenvolvimento do *software* teve início com a estruturação das classes, que consistem na fundação do restante do código. Ao aplicar princípios da programação orientada a objetos e alguns padrões de projetos, chegou-se a uma estrutura reutilizável.

A estrutura do *software* possui roteamento, verificações de autorização e permissão, controladores para gerenciar as requisições *HTTP*, modelos para comunicação entre a aplicação e seu banco de dados, mediadores para gerenciar a construção dos objetos e validar seus dados de entrada, repositórios para controlar os modelos e alguns decoradores para estender funcionalidades.

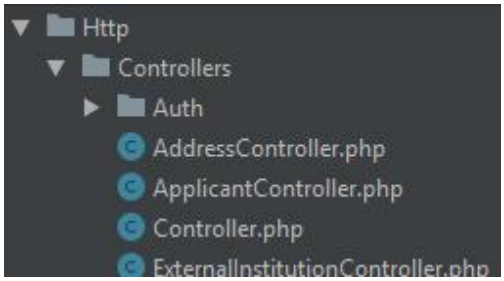
Todos os módulos da aplicação foram organizados pelo seu domínio de conhecimento, sendo eles: Usuário, Família e Instituição. As figuras 13, 14 e 15 apresentam a estrutura do projeto:

Figura 13 - Diretórios de domínios



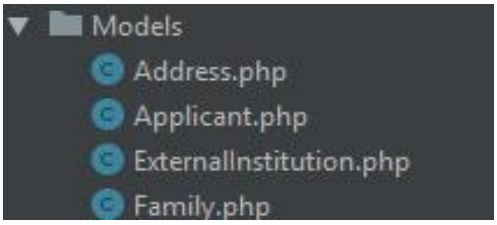
Fonte: Autoria Própria

Figura 14 - Diretórios de controladores



Fonte: Autoria Própria

Figura 15 - Diretórios de modelos



Fonte: Autoria Própria

6.2 ROTEAMENTO

Para realizar o roteamento das requisições, utilizou-se a façade *Route* provida pelo *framework*, classe responsável por servir de ponte entre a aplicação com a real classe responsável pela gerência das rotas dentro do sistema.

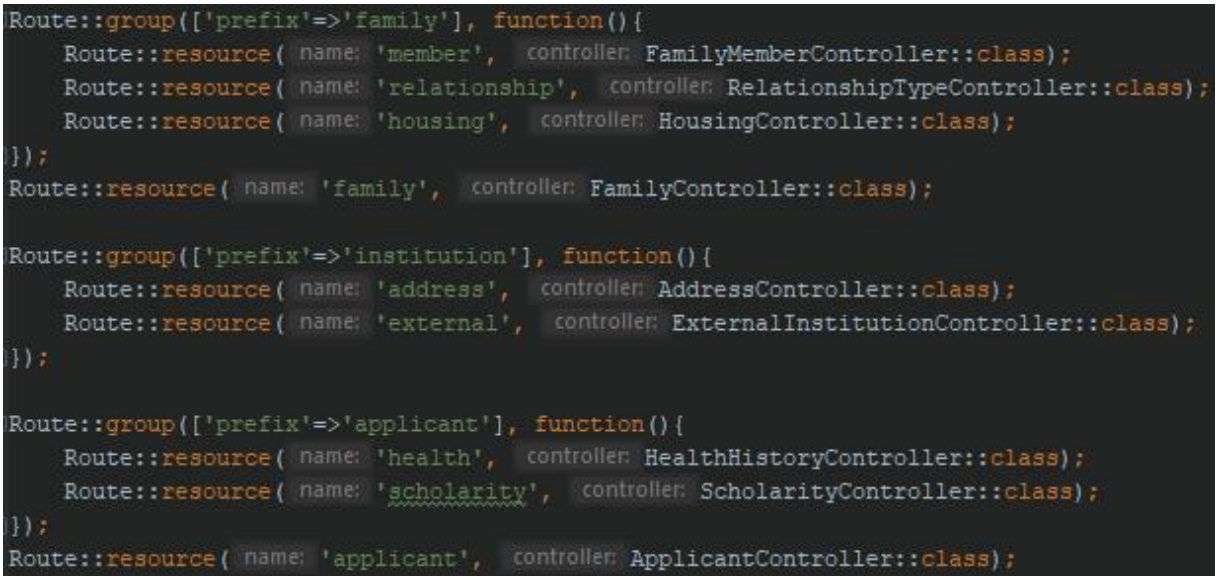
Foram organizadas as rotas de consulta, criação, edição e destruição das entidades de acordo com o domínio e hierarquia.

A façade possui métodos estáticos com os mesmos nomes dos verbos *HTTP*, utilizados para mapear requisições para as funções que devem ser invocadas dos controladores definidos. Ao utilizar o método estático *resource,* é realizado o mapeamento automático pelo *framework*, seguindo as seguintes regras:

- Método *get* para a função *index* do controlador definido;
- Método *get* com o código de identificação do recurso para a função *show* do controlador definido;
- Método *post* para a função *store* do controlador definido;
- Método *put* com o código de identificação do recurso para a função *update* do controlador definido;
- Método *delete* com o código de identificação do recurso para a função *destroy* do controlador definido..

A Figura 16 apresenta como são definidas as rotas e seus controladores.

Figura 16 - Mapeamento de rotas e seus controladores



```
Route::group(['prefix'=>'family'], function(){
    Route::resource( name: 'member',  controller: FamilyMemberController::class);
    Route::resource( name: 'relationship',  controller: RelationshipTypeController::class);
    Route::resource( name: 'housing',  controller: HousingController::class);
});
Route::resource( name: 'family',  controller: FamilyController::class);

Route::group(['prefix'=>'institution'], function(){
    Route::resource( name: 'address',  controller: AddressController::class);
    Route::resource( name: 'external',  controller: ExternalInstitutionController::class);
});

Route::group(['prefix'=>'applicant'], function(){
    Route::resource( name: 'health',  controller: HealthHistoryController::class);
    Route::resource( name: 'scholarity',  controller: ScholarityController::class);
});
Route::resource( name: 'applicant',  controller: ApplicantController::class);
```

Fonte: Autoria Própria

6.3 AUTENTICAÇÃO

Para implementar a autenticação, foi utilizada a biblioteca pertencente ao *framework* Laravel, chamada Laravel Passport que implementa a *framework* de autenticação OAuth 2.0.

A biblioteca provém alguns comandos para a implementação de autenticação por meio de *token* ou usuário e senha.

6.4 PERMISSÃO

O gerenciamento de permissões foi construído a partir do laravel-permission da empresa Spatie, uma biblioteca *Open Source* focada em gerenciar tanto as permissões quanto os papéis dos usuários dentro do sistema.

A biblioteca permitiu a agilidade no desenvolvimento de verificações do que cada usuário pode fazer ou ver dentro da aplicação, o que remove a necessidade de desenvolver uma solução própria para o gerenciamento de permissões.

6.5 CONTROLADOR

Criou-se um controlador base chamado *CrudController*, no qual foi aplicado o princípio de herança definido pela programação orientada a objetos. Este princípio é um mecanismo que permite que características comuns sejam concentradas numa superclasse. A partir desta, podem ser derivadas outras classes, as quais possuem os mesmos métodos e propriedades, no entanto podem ser acrescidas novas funções (RICARTE, 2001, p. 6).

Todos os outros controladores do sistema estenderam este controlador base que é apresentado na figura 17.

Figura 17 - Construtor do controlador base

```php
<?php

namespace App\Http\Controllers;

use ...

class CrudController extends BaseController
{
    use AuthorizesRequests, DispatchesJobs, ValidatesRequests;


    /**
     * @var ManipulationManagerInterface
     */
    protected $manager;
    /**
     * @var RepositoryInterface
     */
    protected $repository;

    /**
     * CrudController constructor.
     * @param ManipulationManagerInterface $manager
     * @param RepositoryInterface $repository
     */
    public function __construct(
        ManipulationManagerInterface $manager,
        RepositoryInterface $repository
    )
    {

        $this->manager = $manager;
        $this->repository = $repository;
    }
```

Fonte: Autoria Própria

O *CrudController* é responsável pela implementação básica de todas as funções mapeadas pelo roteamento. Padroniza o comportamento das entidades e garante a aplicação do princípio de *Don't Repeat Yourself(DRY),* que define que

todo pedaço de conhecimento deve ter uma única, mas não ambígua e autoritativa representação dentro de um sistema (HUNT, THOMAS, 1999).

A figura 18 apresenta as funções mapeadas dos verbos HTTP.

Figura 18 - Funções mapeadas aos verbos HTTP

```php
public function index(Request $request)
{
    $sanitized = $request->all();
    return $this->repository->list($sanitized);
}

/** Store a newly created resource in storage. ...*/
public function store(Request $request)
{
    $sanitized = $request->all();
    return $this->manager->storeOrUpdate($sanitized);
}

/**
 * Display the specified resource.
 *
 * @param  int  $id
 * @return \Illuminate\Http\Response
 */
public function show($id)
{
    return $this->repository->find($id);
}

/**
 * Update the specified resource in storage.
 *
 * @param  \Illuminate\Http\Request  $request
 * @param  int  $id
 * @return \Illuminate\Http\Response
 */
public function update(Request $request, $id)
{
    $sanitized = $request->all();
    return $this->manager->storeOrUpdate($sanitized, $id);
}

/**
 * Remove the specified resource from storage.
 *
 * @param  int  $id
 * @return \Illuminate\Http\Response
 */
public function destroy($id)
{
    return $this->manager->destroy($id);
}
```

Fonte: Autoria Própria

Estendendo o *CrudController* simplificou-se a implementação de todas as outras entidades como mostra a Figura 19. Foi necessário somente customizar o construtor para especializar a entidade e adicionar novos métodos específicos para o funcionamento da funcionalidade.

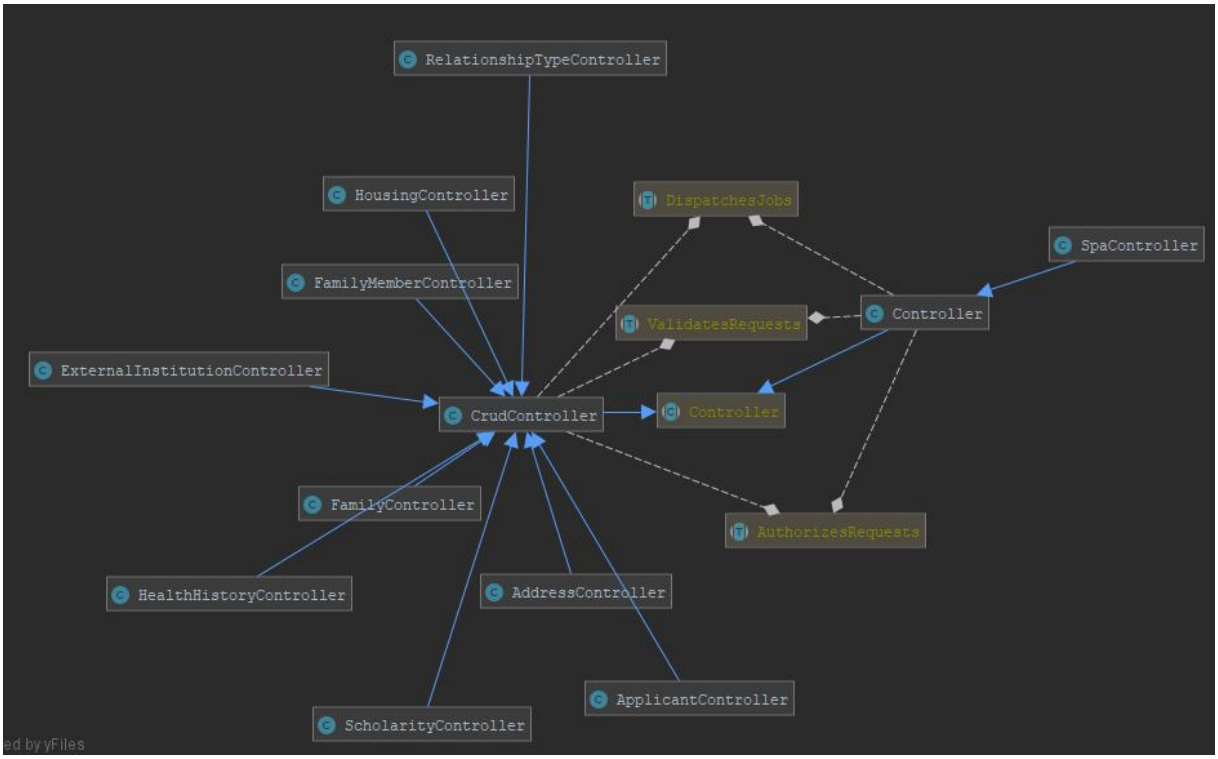
Figura 19 - Construtor sendo sobreposto em uma entidade

```php
<?php

namespace App\Http\Controllers;

use App\Domains\Family\Repository\FamilyMemberRepository;
use App\Domains\Family\Managers\FamilyMemberManager;
use App\Models\RelationshipType;


class FamilyMemberController extends CrudController
{
    public function __construct(
        FamilyMemberManager $manager,
        FamilyMemberRepository $repository
    )
    {
        parent::__construct($manager, $repository);
    }

    public function show($id)
    {
        return $this->repository->find($id, ['address', 'family']);
    }
    public function getFatherFromFamily($family_id){
        $this->repository->applyFilters(
            [
                'family' => $family_id,
                'relationship_type' => RelationshipType::PAI
            ]
        );
        return $this->repository->first();
    }
}
```

Fonte: Autoria Própria

A Figura 20 apresenta as relações entre os controladores e o controlador base.

Figura 20 - Relações entre controladores e o controlador base



Fonte: Autoria Própria

6.6 MODELO

As classes de modelo são a representação da entidade dentro do sistema. São também responsáveis pela interação entre a aplicação e o banco de dados. Foram implementadas utilizando a biblioteca *Eloquent,* a qual é parte integrante do *framework* Laravel.

Essa biblioteca trabalha com mapeamento de objeto-relacional (*ORM, do inglês Object-relational mapping*) que permite acessar e manipular tuplas e tabelas de um banco de dados relacional, por meio de uma API orientada a objetos (DUNGLAS, 2013).

A Figura 21 apresenta um exemplo das classes de modelo.

Figura 21 - Propriedades dos modelos

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Address extends Model
{

    protected $fillable = [
        "cep",
        "tipo_logradouro",
        "logradouro",
        "numero",
        "complemento",
        "bairro",
        "cidade"
    ];

    protected $hidden = [
        "addressable_type",
        "addressable_id"
    ];

    public function addressable()
    {
        return $this->morphTo();
    }
```

Fonte: Autoria Própria

6.7 MEDIADOR

Mediadores são responsáveis pela aplicação da regra de negócio dentro do sistema, realizar validações dos dados de entrada para geração e persistência de novas entidades e por invocar as funções de destruição de recursos na aplicação.

A Figura 22 apresenta o construtor do manipulador base e suas propriedades.

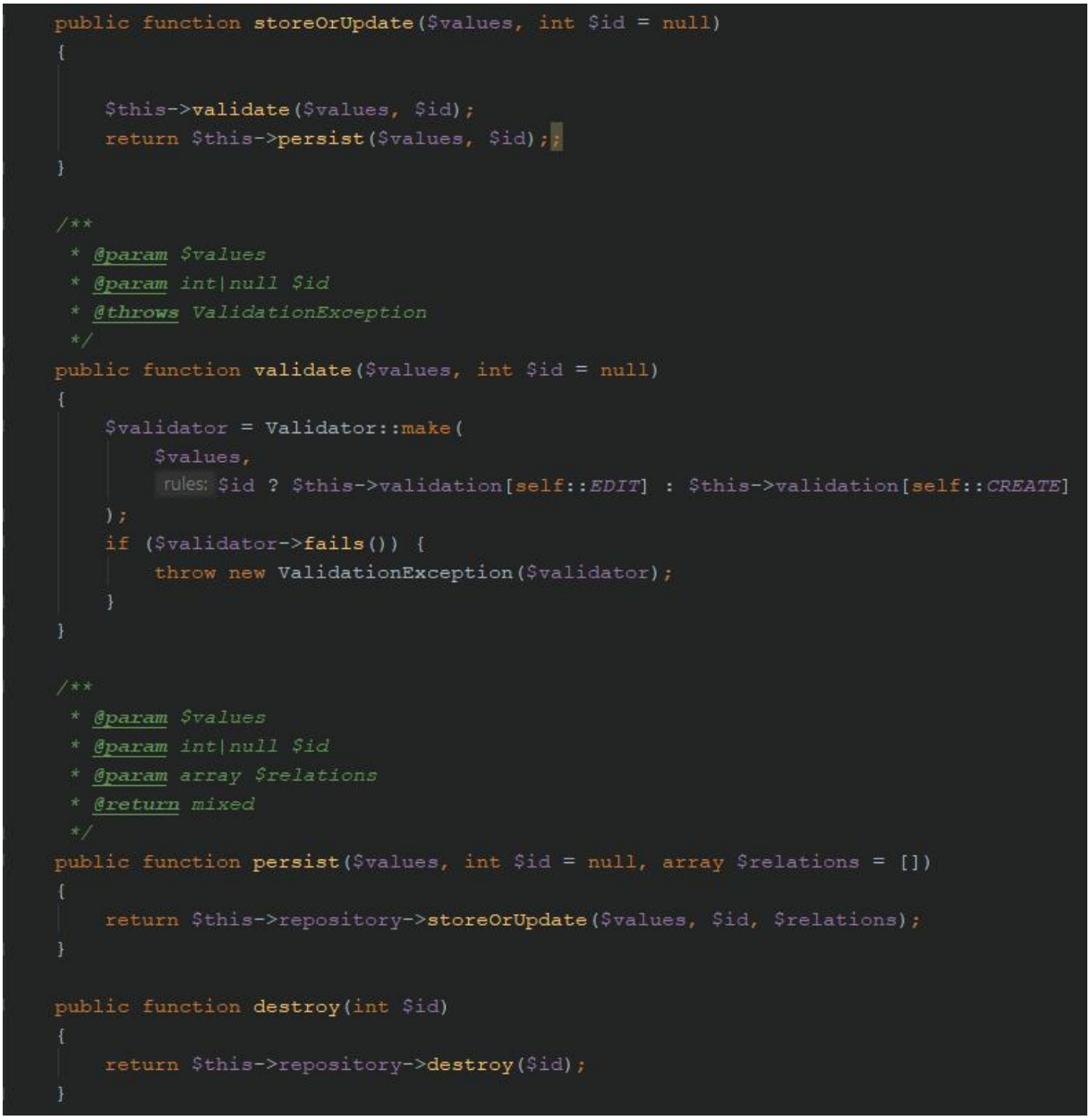
Figura 22 - Construtor do manipulador base e suas propriedades

```php
<?php


namespace App\Extendable;


use App\Interfaces\ManipulationManagerInterface;
use App\Interfaces\RepositoryInterface;
use Illuminate\Support\Facades\Validator;
use Illuminate\Validation\ValidationException;

class ManipulationManager implements ManipulationManagerInterface
{
    const CREATE = 'create';
    const EDIT = 'edit';
    protected $validation = [
        self::CREATE => [],
        self::EDIT => []
    ];
    /**
     * @var RepositoryInterface
     */
    protected $repository;

    /**
     * ManipulationManager constructor.
     * @param RepositoryInterface $repository
     */
    public function __construct(RepositoryInterface $repository)
    {
        $this->repository = $repository;
    }
```

Fonte: Autoria Própria

A partir do mesmo princípio do controlador, aplicou-se o princípio de DRY, que gerou a superclasse *ManipulationManager,* a qual recebeu os métodos padrão de um mediador. A Figura 23 exibe as funções do manipulador base.

Figura 23 - Funções de validação e persistência



```php
public function storeOrUpdate($values, int $id = null)
{

    $this->validate($values, $id);
    return $this->persist($values, $id);;
}

/**
 * @param $values
 * @param int|null $id
 * @throws ValidationException
 */
public function validate($values, int $id = null)
{
    $validator = Validator::make(
        $values,
        rules: $id ? $this->validation[self::EDIT] : $this->validation[self::CREATE]
    );
    if ($validator->fails()) {
        throw new ValidationException($validator);
    }
}

/**
 * @param $values
 * @param int|null $id
 * @param array $relations
 * @return mixed
 */
public function persist($values, int $id = null, array $relations = [])
{
    return $this->repository->storeOrUpdate($values, $id, $relations);
}

public function destroy(int $id)
{
    return $this->repository->destroy($id);
}
```

Fonte: Autoria Própria

Como pode ser visto na Figura 24, é responsabilidade do mediador específico da entidade implementar suas próprias regras de validação e de negócio.

Figura 24 - Validações de uma entidade específica

```php
<?php

namespace App\Domains\Family\Managers;

use App\Domains\Common\Traits\HasRelations;
use App\Domains\Family\Repository\FamilyMemberRepository;
use App\Domains\Family\Repository\FamilyRepository;
use App\Extendable\ManipulationManager;
use App\Models\FamilyMember;

class FamilyMemberManager extends ManipulationManager
{

    use HasRelations;

    protected $validation = [
        self::CREATE => [
            'nome' => 'required|min:2',
            'sobrenome' => 'required|min:2',
            'cpf' => 'required|cpf',
            'idade' => 'required|min:1',
            'profissao' => 'required|min:3',
            'renda_mensal' => 'required|min:1',
            'family' => 'required|exists:families, id',
        ],
        self::EDIT => [
            'nome' => 'min:2',
            'sobrenome' => 'min:2',
            'cpf' => 'cpf',
            'idade' => 'min:1',
            'profissao' => 'min:3',
            'renda_mensal' => 'min:1',
            'family' => 'exists:families, id',
        ]
    ];
```

Fonte: Autoria Própria

Para regras de validação mais complexas, é necessário utilizar outro método do mediador além das suas propriedades protegidas. Método personalizado demonstrado na Figura 25.

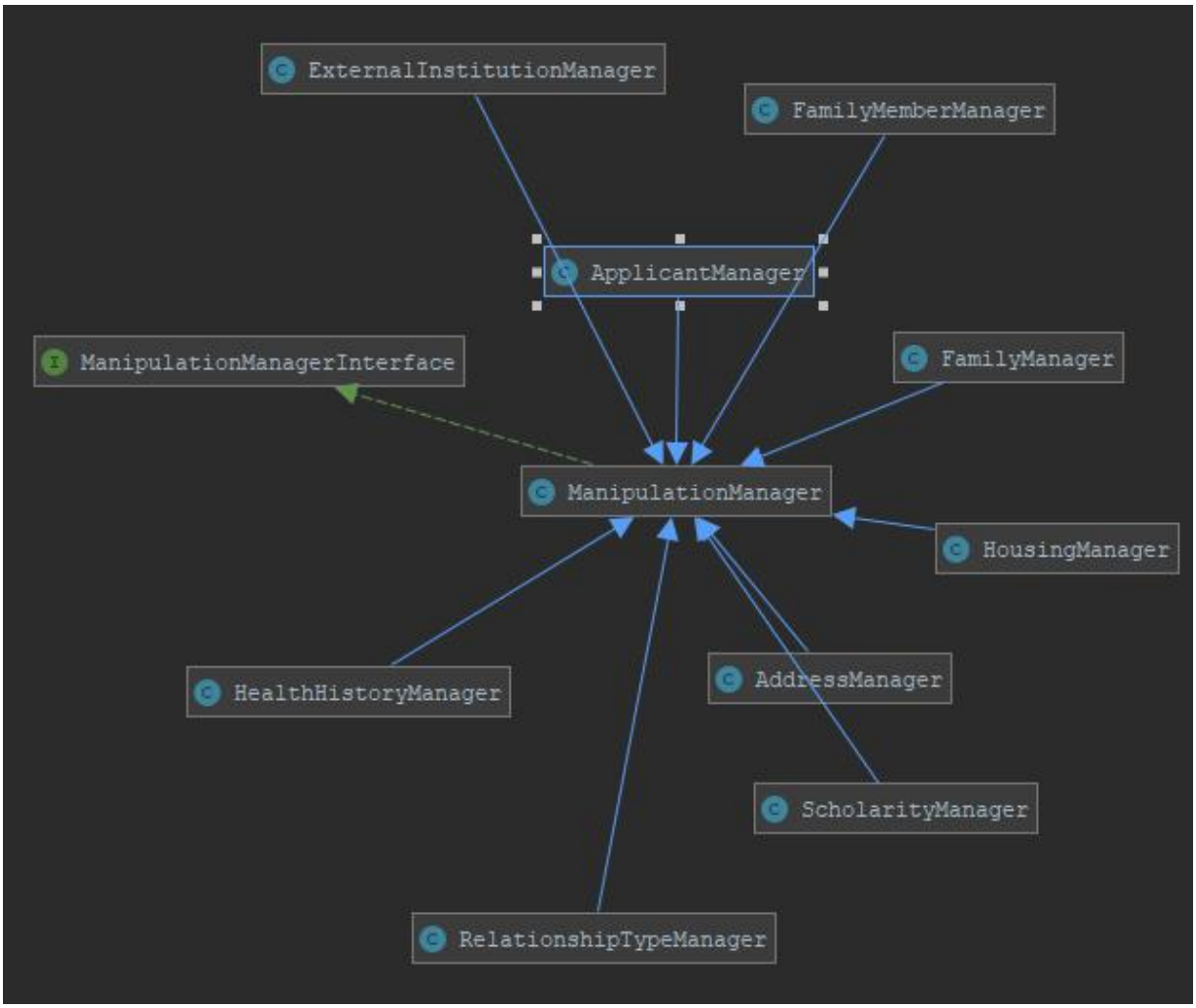
Figura 25 - Validações complexas de uma entidade

```
    /**
     * @var FamilyRepository
     */
    private $family_repository;

    /**
     * FamilyMemberManager constructor.
     * @param FamilyMemberRepository $repository
     * @param FamilyRepository $family_repository
     */
    public function __construct(
        FamilyMemberRepository $repository,
        FamilyRepository $family_repository
    )
    {
        parent::__construct($repository);
        $this->family_repository = $family_repository;
        $validation_create = [
            'estado_civil' => 'required|in:' . implode(
                glue: ',',
                array_keys(
                    FamilyMember::$ESTADO_CIVIL_CHOICES
                )
                ),
            'escolaridade' => 'required|in:' . implode(
                glue: ',',
                array_keys(
                    FamilyMember::$ESCOLARIDADE_CHOICES
                )
                ),
            'genero' => 'required|in:' . implode(
                glue: ',',
                array_keys(
                    FamilyMember::$GENERO_CHOICES
                )
                )
        ];

        $validation_edit = [
```

Fonte: Autoria Própria

A Figura 26 demonstra as relações entre o mediador base e as entidades que o estendem.

Figura 26 - Relações entre o mediador base e as entidades que o estendem



Fonte: Autoria Própria

6.8 REPOSITÓRIO

Os repositórios são as classes mais complexas da aplicação, estas servem como interface entre a camada de classes de serviços, assim como os mediadores e a camada de modelos.

Têm como responsabilidade montar as pesquisas com seus filtros para o banco de dados, trazer de forma paginada os resultados dessas pesquisas, encontrar as tabelas com as quais a entidade se relaciona e gravar o relacionamento.

Da mesma maneira que o controlador e o mediador, também construiu-se uma superclasse *Repository,* a fim de evitar a repetição do código e concentrar toda sua modificação num único lugar. A Figura 27 demonstra suas propriedades.

Figura 27 - Propriedades de um repositório

```php
<?php


namespace App\Extendable;


use App\Interfaces\RepositoryInterface;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\ModelNotFoundException;
use Illuminate\Database\Query\Builder;
use Illuminate\Http\Resources\Json\JsonResource;
use Illuminate\Support\Collection;

class Repository implements RepositoryInterface
{
    /**
     * @var Model
     */
    protected $model = null;
    protected $searchableFields = [];
    /**
     * @var JsonResource
     */
    protected $presenter = null;
    protected $returnable = null;
    /**
     * @var Builder
     */

    protected $query = null;
    /**
     * @var array
     */
    protected $filters;
    private $polymorphic = false;
```

Fonte: Autoria Própria

Foram definidos três métodos genéricos para realizar as pesquisas em banco, cada um responsável por trazer o resultado de maneira diferente.

O método *list* tem como comportamento padrão a busca de todos os registros no banco de dados que coincidirem com os filtros utilizados.

Já o método *first* tem o objetivo de trazer o primeiro resultado do banco de dados que coincida com os filtros.

Como e possível verificar na Figura 28, *find* é realizado o filtro por um código de chave primária, que retorna um único resultado, independente de outros filtros.

Figura 28 - Funções de pesquisa no banco de dados

```php
/**
 * @param array $filters
 * @param array $with
 * @param int $pagination
 * @return \Illuminate\Http\Resources\Json\AnonymousResourceCollection|null
 */
public function list(array $filters = [], array $with = [], $pagination = 45)
{
    $query = $this->newQuery();
    $query->with($with);
    if (!empty($filters)) {
        $this->applyFilters($filters);
        $this->injectFiltersOnQuery();
    }
    $this->returnable = $query->paginate($pagination);
    return $this->present();
}

public function first(array $with = [], array $filters = [])
{
    $this->newQuery();
    $this->applyFilters($filters);
    $this->query->with($with);
    $this->returnable = $this->query->first();
    return $this->returnable;
}
/**
 * @param int $id
 * @param array $with
 * @return \Illuminate\Http\Resources\Json\AnonymousResourceCollection|null
 */
public function find(int $id, array $with = [])
{
    $query = $this->newQuery();
    $query->with($with);
    $this->returnable = $query->findOrFail($id);
    return $this->present();
}

/**
 * @param JsonResource $resource
 */
public function setPresenter($resource = null)
{
    $this->presenter = $resource;
}
```

Fonte: Autoria Própria

Os filtros são injetados em todas as pesquisas de forma automática por meio das funções *applyFilters* e *injectFiltersOnQuery.* É realizada a comparação dos parâmetros passados pela requisição com a propriedade *searchableFields,* a qual é configurada por repositório, e define quais colunas da tabela do banco de dados da entidade podem ser utilizadas para filtros. Na Figura 29 pode ser visto sua implementação.

Figura 29 - Funções de construção dos filtros de pesquisa

```php
/**
 * @param array $filters
 */
public function applyFilters(array $filters = [])
{
    $this->filters = $filters;
}

/**
 *
 */
public function injectFiltersOnQuery()
{
    foreach ($this->searchableFields as $searchableField) {
        if (in_array($searchableField['field'], array_keys($this->filters))) {
            $this->query->where(
                $searchableField['field'],
                operator: $searchableField['operator'] ?? "=",
                $this->filters[$searchableField['field']]
            );
        }
    }
    $this->filters = [];
}
```

Fonte: Autoria Própria

Como é possível verificar nas figuras 30 e 31, para realizar a persistência no banco de dados foram construídas duas funções, uma responsável por persistir o dados da entidade principal e outra função responsável por gravar todas as associações entre a entidade principal e suas relações.

Figura 30 - Funções de persistência do banco de dados dos repositórios

```php
/**
 * @param $values
 * @param int|null $id
 * @param array $relations
 * @return \Illuminate\Http\Resources\Json\AnonymousResourceCollection|null
 */
public function storeOrUpdate($values, int $id = null, array $relations = [])
{
    $presenter = $this->presenter;
    $this->setPresenter( resource: null);

    if ($id) {
        $this->returnable = $this->find($id);
    } else {
        $this->returnable = new $this->model;
    }
    $this->returnable->fill($values);

    $this->persist($relations);

    if ($presenter) {
        $this->setPresenter($presenter);
    }

    return $this->present();
}

/**
 * @param array $relations
 */
public function associate(array $relations)
{
    foreach ($relations as $relation) {
        $relationship = $relation['name'];
        $this->returnable->$relationship()->associate($relation['model']);
    }
}

/**
 * @param int $id
 * @return int
 */
public function destroy(int $id)
{
    return $this->model::destroy($id);
}
```

Fonte: Autoria Própria

Figura 31 - Tratamento da persistência simples e polimórfica

```php
public function persist(array $relations = [])
{
    if ($this->isPolymorphic()) {
        return $this->persistPolymorphic($relations);
    }
    if (!empty($relations)) {
        $this->associate($relations);
    }
    $this->returnable->save();
}

public function persistPolymorphic(array $relations)
{
    $result = false;
    foreach ($relations as $relation) {
        $model = $relation['model'];
        $polymorphic = $relation['polymorphic'];
        $result = $model->$polymorphic()->save($this->returnable);
    }
    return $result;
}
```

Fonte: Autoria Própria

Todos os repositórios do sistema herdam dessa superclasse esses métodos padrão, o que torna sua construção mais simplificada.

Na figura 32, pode-se ver um exemplo de como é um repositório de uma entidade. Sendo somente necessário apontar a qual modelo a entidade corresponde e quais os campos que podem ser utilizados para construção dos filtros.

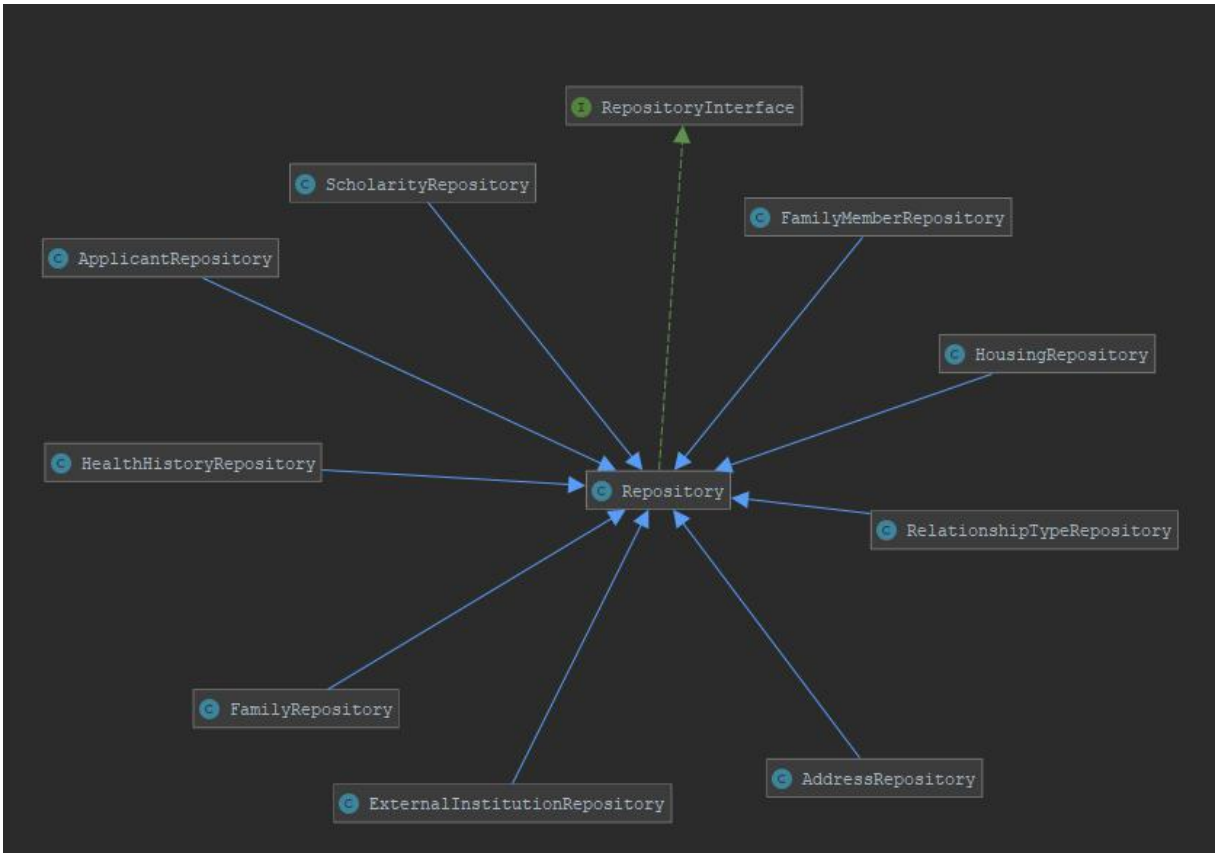
Figura 32 - Implementação de um repositório de uma entidade específica

```php
<?php

namespace App\Domains\Family\Repository;

use App\Extendable\Repository;
use App\Models\Family;

class FamilyRepository extends Repository
{
    protected $model = Family::class;
    protected $searchableFields = [
        ["field" => "nis_familiar"],
        ["field" => "bolsa_familia"]
    ];

}
```

Fonte: Autoria Própria

Na Figura 33 verifica-se a relação de herança entre o Repositório base e as implementações das entidades.

Figura 33 - Relação entre a superclasse Repositório e suas implementações específicas



Fonte: Autoria Própria

6.9 DECORADOR

Apesar da maior parte das funcionalidades serem genéricas o suficiente para serem herdadas de algumas superclasses, existem casos nos quais algumas funções precisam ser reescritas, com o auxílio da propriedade da programação orientada a objetos de sobreposição de funções.

Apesar dessa necessidade ser menor, as novas funções ainda são compartilháveis entre todas as entidades que contêm as mesmas características.

Por isso foi implementado um “decorador”, um tipo de classe responsável por agregar funções à classe a qual o utiliza na forma de *trait* do PHP.

No exemplo da Figura 34, é sobreposta a função *storeOrUpdate* e criada uma nova função *getRelations*.

Figura 34 - Funções que sobrepõem outras funções onde o decorador é utilizado

```php
<?php


namespace App\Domains\Common\Traits;


trait HasRelations
{
    public function getRelations($values)
    {
        $relations = [];
        foreach ($values as $key => $value){
            if(in_array($key, $this->relations)){
                $repository = $key.'_repository';
                $$key = $this->$repository->find($value);

                  $row = ['name' =>$key, 'model' => $$key];
                  if(isset($this->polymorphic) && !empty($this->polymorphic)){
                      $row = array_merge($row, ["polymorphic" => $this->polymorphic]);
                  }
                $relations[] = $row;
            }
        }

        return $relations;
    }

    public function storeOrUpdate($values, int $id = null)
    {
        $this->validate($values, $id);
        $relations = $this->getRelations($values);
        return $this->persist($values, $id, $relations);

    }
}
```

Fonte: Autoria Própria

7 RESULTADOS

7.1 USUÁRIO

No módulo de usuário foram concentrados os cadastros das crianças e adolescentes no sistema, o acompanhamento realizado com cada um e o lançamento de ocorrências.

7.1.1 Cadastro

O cadastro do usuário contém a principal forma de inserir os dados de um usuário dentro do sistema. É possível realizar o cadastro e adicionar dados pessoais, familiares, de contato, escolares e referências externas (encaminhamento de algum CRAS).

A Figura 35 apresenta a tela de cadastro desenvolvida.

Figura 35 - Tela de cadastro do usuário

Fonte: Autoria Própria

7.1.2 Acompanhamento

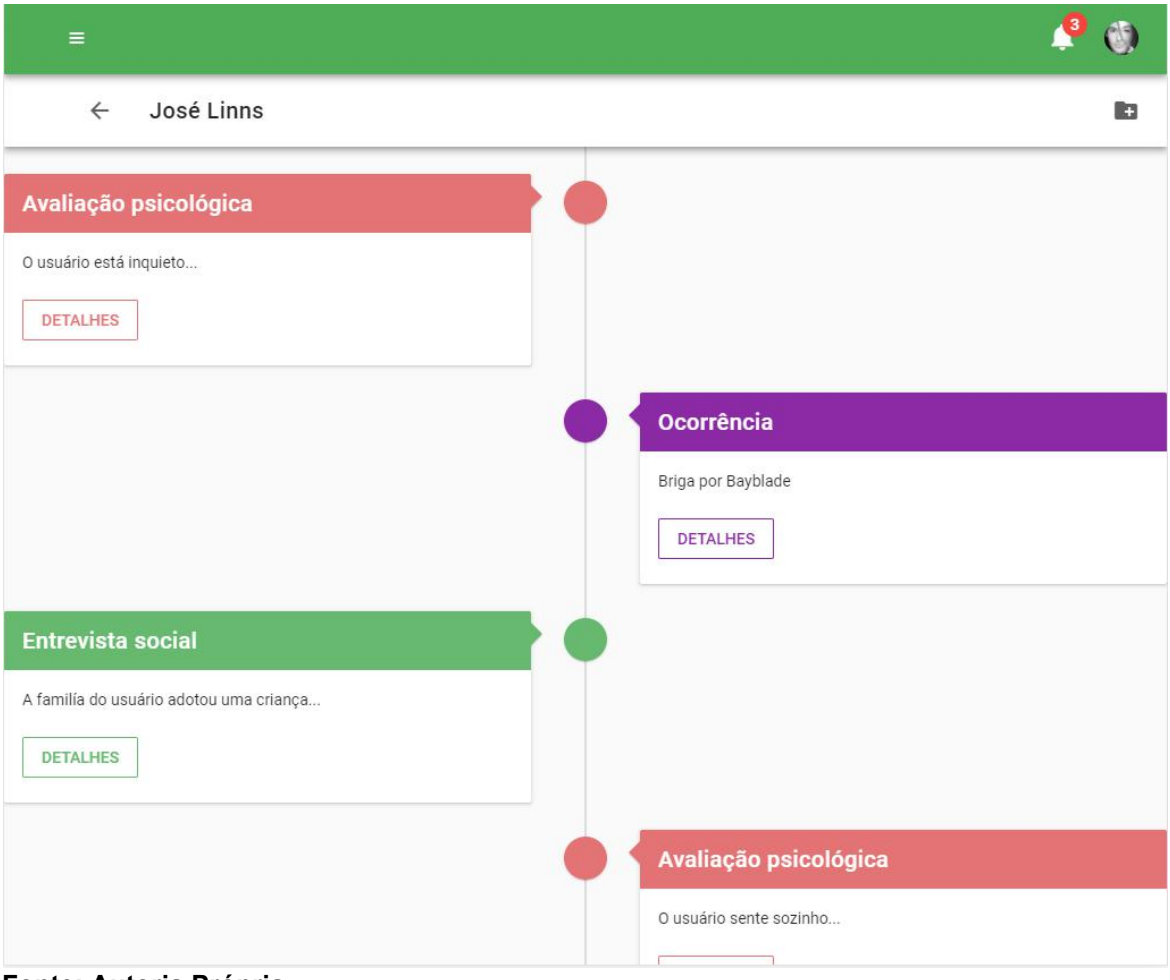
O submódulo de acompanhamento permite ao funcionário visualizar as informações pertinentes ao seu nível de acesso em forma de *timeline*, possibilitando ao mesmo realizar novas entradas em forma de ocorrência.

No corpo principal da funcionalidade é exibida a informação pública registrada pelo funcionário que lançou a ocorrência. Esta informação é disponibilizada a todos que têm acesso a esta funcionalidade.

Foram utilizadas diferentes cores para representar cada tipo de ocorrência lançada no perfil do usuário. Sendo as principais: vermelho para avaliações psicológicas, roxo para ocorrências internas e verde para atualizações pela assistência social.

A Figura 36 apresenta uma *timeline* com as ocorrências dos usuários da Casa do Piá.

Figura 36 - Timeline de acompanhamento



Fonte: Autoria Própria

7.1.3 Ocorrências

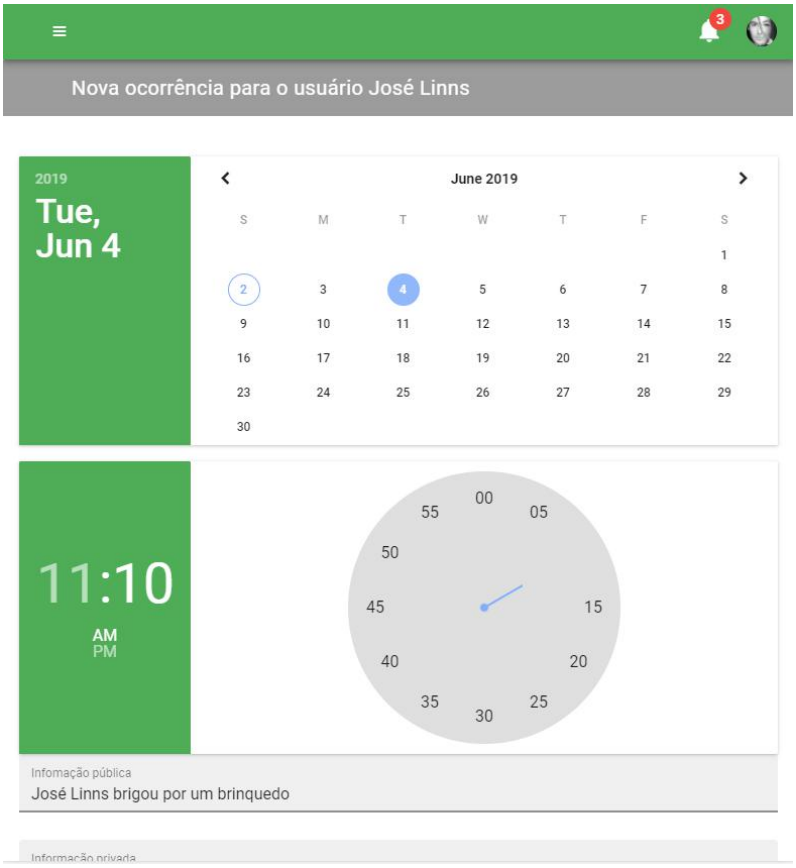
É a forma pela qual os funcionários interagem com o sistema, atualizando o banco de informações de cada usuário. Registrando o acompanhamento que representa a jornada do usuário desde que ele entrou em contato com a instituição.

Consiste na definição da data e hora do ocorrido e uma descrição pública, a qual qualquer funcionário, que tem acesso a funcionalidade de acompanhamento, pode ver, com a intenção de informar e disseminar informações pertinentes para o funcionamento da instituição em relação ao usuário.

Possui também, um campo de descrição privado, onde somente os funcionários com o mesmo ou maior nível de acesso podem ver as informações. Destina-se aos relatórios das avaliações psicológicas e entrevistas sociais.

A Figura 37 apresenta a tela de cadastro de ocorrências.

Figura 37 - Tela de cadastro de ocorrência



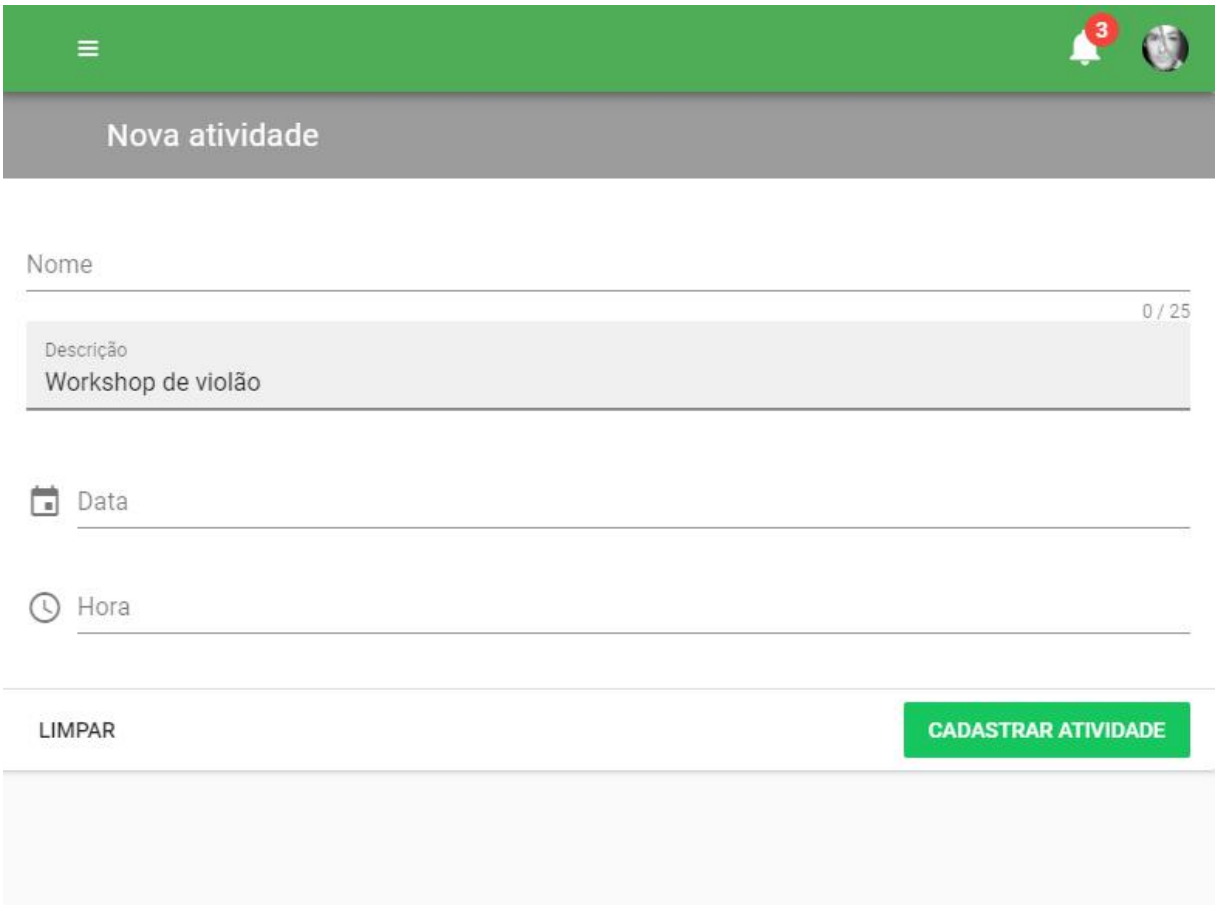
Fonte: Autoria Própria

7.1.4 Atividade

Destina-se ao cadastro das atividades que serão realizadas pelos usuários junto ao Centro Social.

A Figura 38 apresenta a tela de cadastro de atividades.

Figura 38 - Tela de cadastro de atividade



Fonte: Autoria Própria

7.2 FAMÍLIA

No módulo de família, disponibiliza-se a descrição do contexto no qual o usuário se encontra. Registrando os cadastros da família e seus membros, o grau de parentesco entre os usuário e familiares e as informações habitacionais.

7.2.1 Cadastro

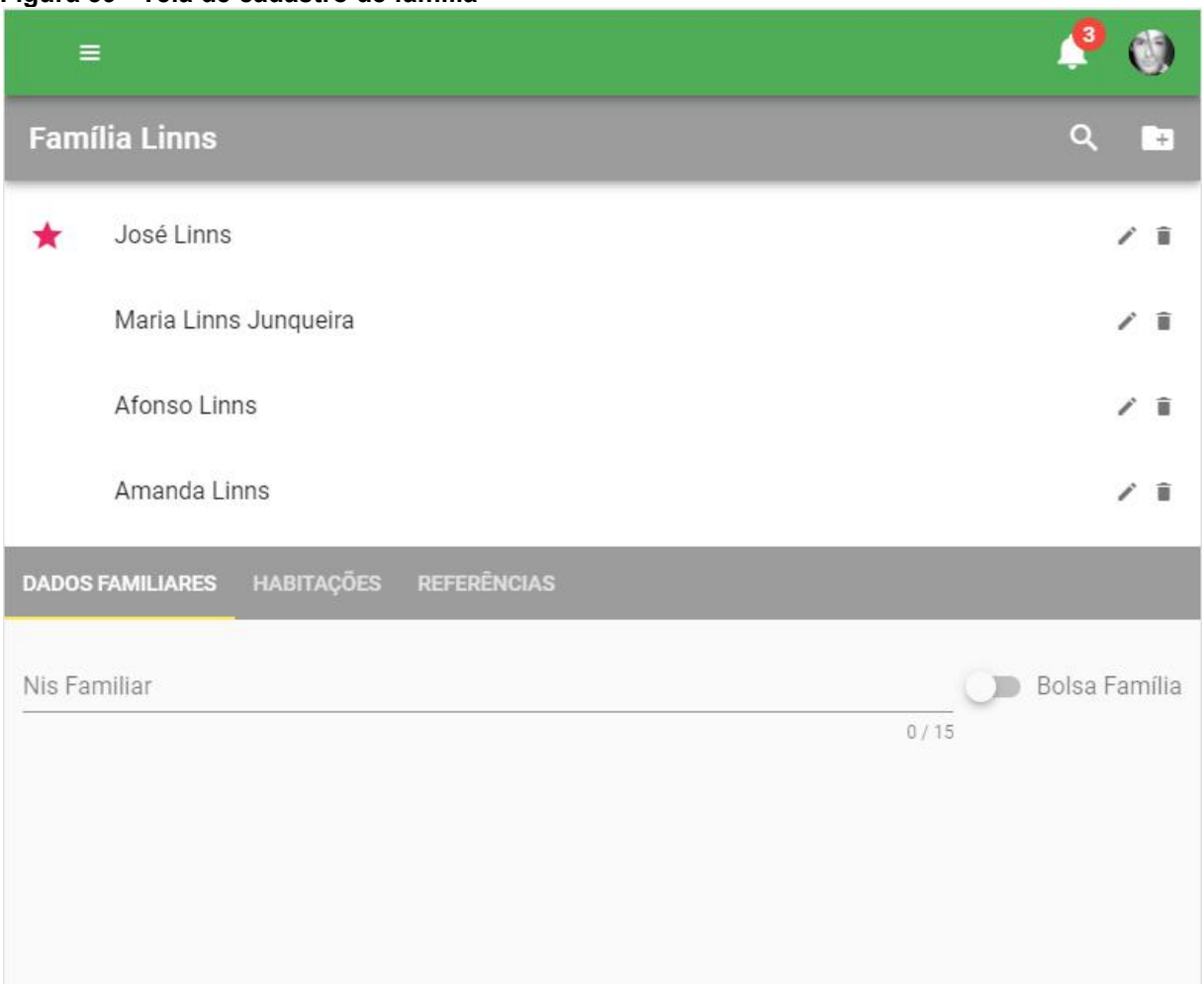
O cadastro familiar constitui-se pela lista de membros da família, seu NIS familiar e se tem ou não cadastro com o auxílio Bolsa Família.

Também contém informações sobre as habitações que a família possui e as referências de instituições externas que a família teve ao realizar o cadastro junto a instituição.

São destacados com uma estrela, os cadastros pertencentes aos usuários a qual a instituição presta atendimento.

A Figura 39 apresenta a tela de cadastro de família.

Figura 39 - Tela de cadastro de família



Fonte: Autoria Própria

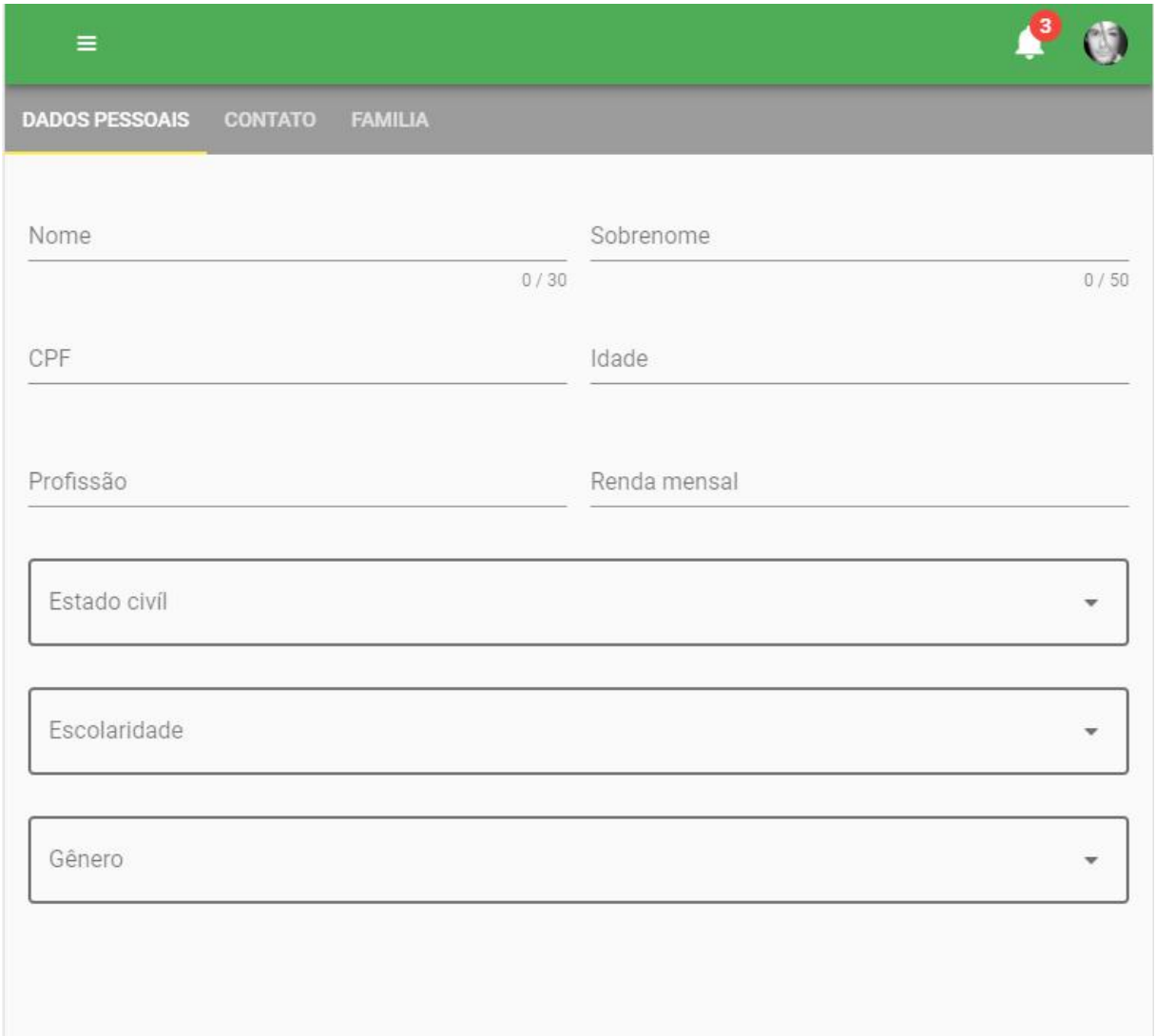
7.2.2 Familiar

O cadastro de familiar destina-se a receber todas informações das pessoas ligadas ao usuário, que não são diretamente atendidas pela instituição.

Contém dados básicos e de contato, com a sua associação em cada família cadastrada no sistema.

A Figura 40 apresenta a tela de cadastro de familiar.

Figura 40 - Tela de cadastro de familiar



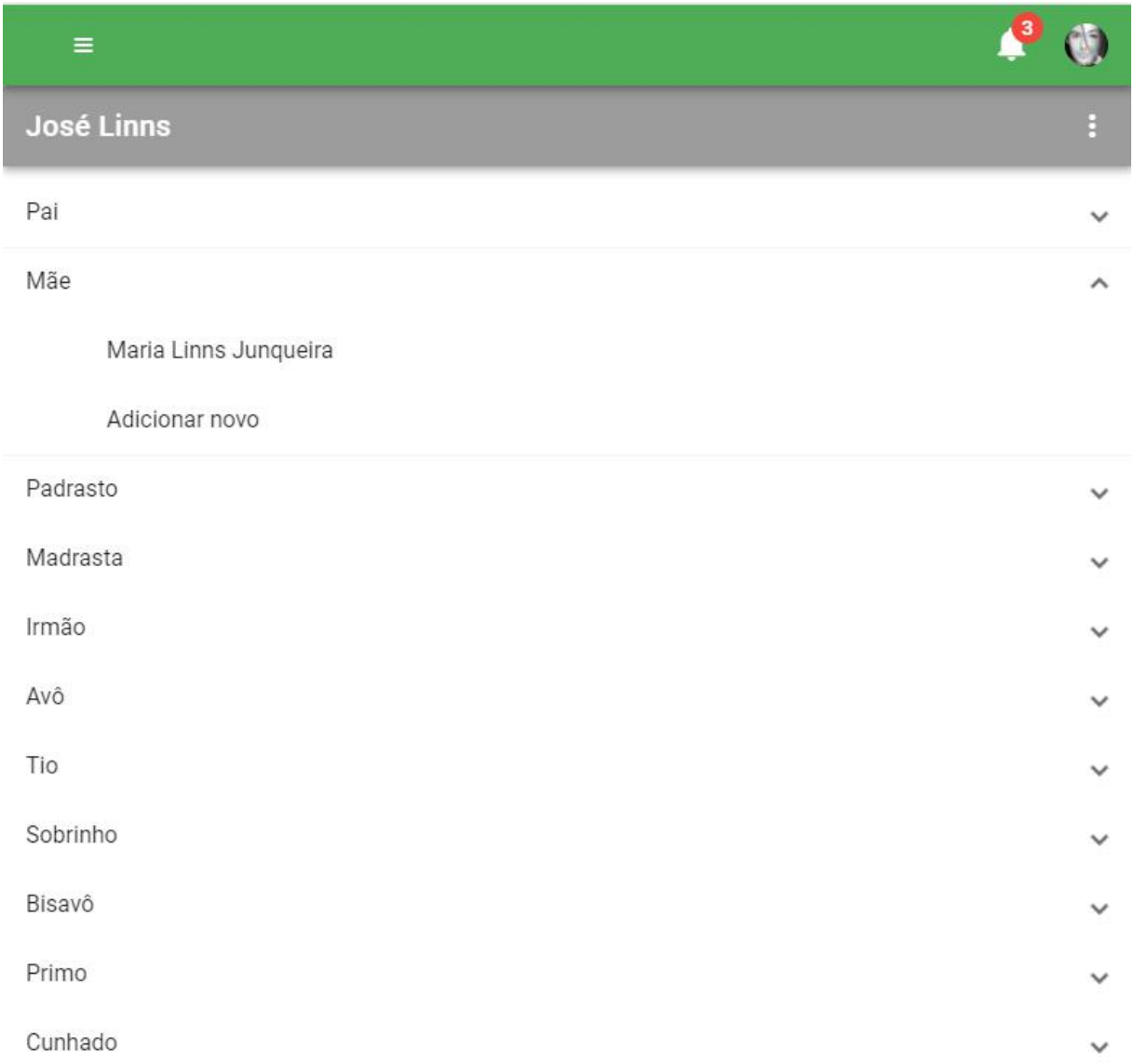
Fonte: Autoria Própria

7.2.3 Grau de parentesco

Funcionalidade destinada a definir a relação entre os membros das famílias e cada um dos usuários cadastrados no sistema, dentro dos graus de parentesco pré-determinados.

A Figura 41 apresenta a dela de associação de um usuário com seus familiares.

Figura 41 - Tela de associação de parentesco do usuário



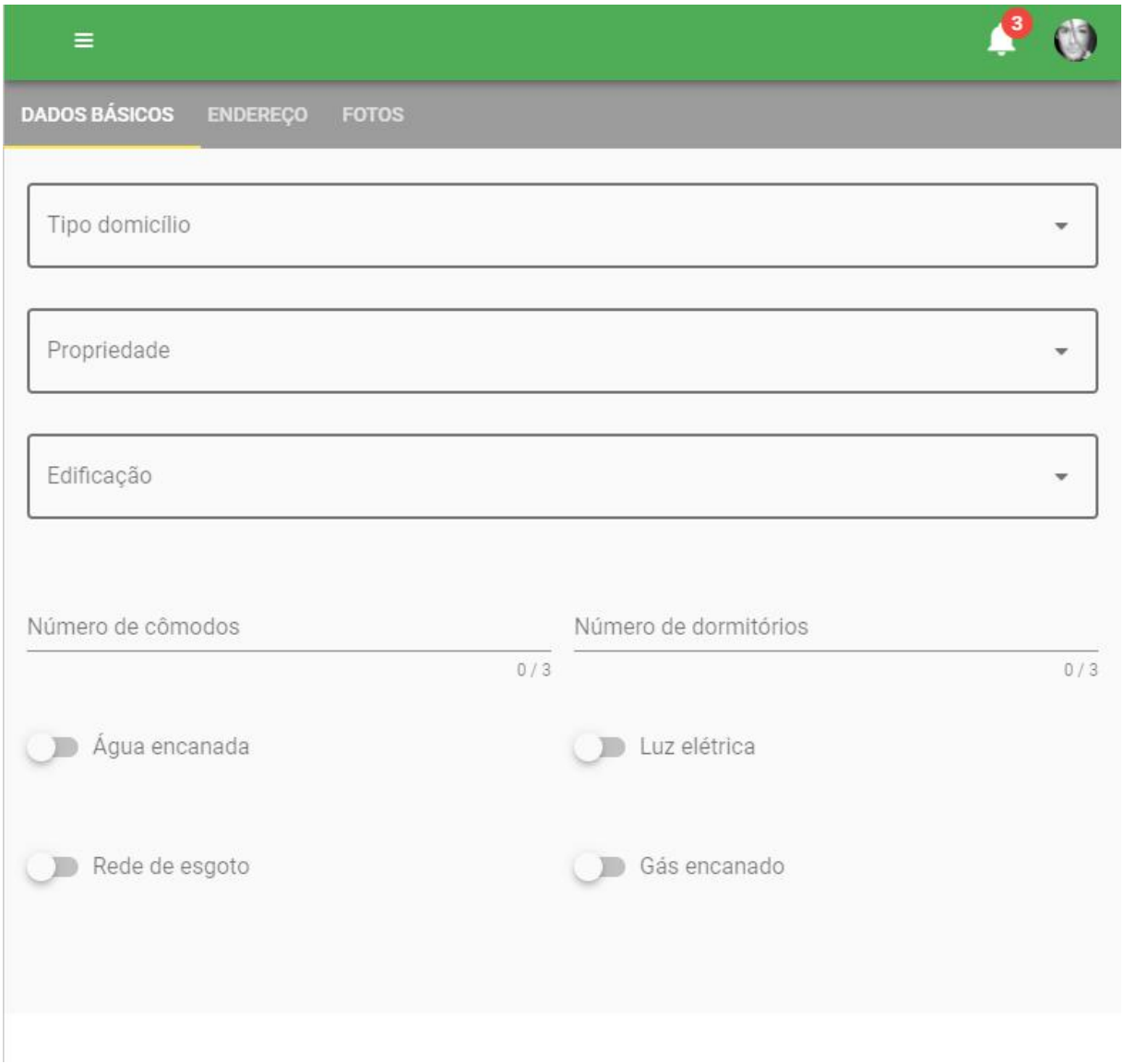
Fonte: Autoria Própria

7.2.4 Habitação

O submódulo de habitação destina-se a guardar as informações sobre as habitações que as famílias possuem, seu tipo, o nível de propriedade, tipo de edificação e algumas informações básicas para contextualizar a vivência do usuário fora da instituição.

A Figura 42 apresenta a tela de cadastro de habitações.

Figura 42 - Tela de cadastro de habitação



Fonte: Autoria Própria

7.3 INSTITUIÇÕES

O módulo de instituições serve para registrar todas as instituições que interagem com o centro social. Contém seus nomes e endereços.

7.3.1 Instituições Externas

Destina-se a receber os dados das instituições que não fazem parte do centro social ou da ABASE.

8 CONCLUSÃO

Este trabalho propôs o desenvolvimento de uma solução para a gestão do Centro Social Casa do Piá, onde era notável a dificuldade nos processos internos para registrar e localizar as informações. O objetivo do sistema foi centralizar as informações e os processos internos da Casa do Piá, garantindo o sigilo e a privacidade dos usuários atendidos pelo Centro Social.

Iniciou-se o trabalho pelo levantamento de requisitos do sistema por meio da técnica de entrevistas. Após a definição dos requisitos com os funcionários da Casa do Piá, realizou-se a modelagem do sistema, desenvolvendo os diagramas de caso de uso, diagramas de atividades e o diagrama entidade relacionamento.

Para o desenvolvimento deste trabalho, foi utilizada a metodologia ágil, de gerenciamento e desenvolvimento, XP, para garantir a qualidade da aplicação desenvolvida. A metodologia de desenvolvimento ágil evolui de forma orgânica junto de cada iteração da aplicação, e traduz de forma eficaz a linguagem do solicitante para os desenvolvedores.

As tecnologias utilizadas para o desenvolvimento foram, o *framework* Vue.js para o desenvolvimento do cliente e o *framework* Laravel para o desenvolvimento do servidor. Estas tecnologias demonstraram ser um conjunto que permite desenvolver bons sistemas, com agilidade, além de facilitar a programação e manutenção do código.

A aplicação desenvolvida permite realizar um cadastro único dos usuários atendidos pela Casa do Piá. Nestes cadastros são mantidas as informações pessoais, familiares, histórico escolar e histórico de saúde dos mesmos. O sistema também conta com uma funcionalidade para registrar os atendimentos de consultas com assistente social e psicólogo. Outra funcionalidade, é o registro de ocorrências dos usuários.

A Casa do Piá, além do acompanhamento com assistente social e psicólogo, também oferece atividades no contraturno das escolas dos usuários. Assim sendo, foi desenvolvida uma funcionalidade para o cadastro dessas atividades no sistema, e também um aplicativo móvel para o registro de presença nestas atividades.

E por fim, por se tratar de um sistema *online*, caso necessário, os funcionários da Casa do Piá podem acessá-lo de qualquer lugar.

Apesar deste sistema não ter sido implantado, é notável que uma aplicação pode automatizar um trabalho burocrático, possibilitando que o usuário tenha tempo para focar nas atividades que são realmente importantes, o que proporciona mais qualidade de vida e agilidade no dia-a-dia.

8.1 TRABALHOS FUTUROS

A realização deste trabalho supriu as principais necessidades do Centro Social Casa do Piá, quanto a centralização das informações e dos processos internos. Porém, pensando em expandir o sistema para outros Centros Sociais, pode-se complementar este trabalho com algumas funcionalidades que serão listadas a seguir:

- Módulo de gestão financeira;
- Notificações no sistema (aniversariantes, agendamento de consultas);
- Integração com órgãos públicos para prestação de contas;
- Elaboração de Plano de Individual de Atendimento.

REFERÊNCIAS

ABASE. **Aliança Brasileira de Assistência Social e Educacional**. Histórico. Ponta Grossa, 2016. 2 p.

ABASE. **Aliança Brasileira de Assistência Social e Educacional**. Regimento interno. Ponta Grossa, 2016. 12 p.

ALVES, W. P. **Banco de dados**. 1. ed. São Paulo: Érica, 2014. 160 p. ISBN 9788536518961

BONI, V; QUARESMA, S. J. **Aprendendo a entrevistar: como fazer entrevistas em Ciências Sociais**. 2005. Disponível em: <https://periodicos.ufsc.br/index.php/emtese/article/view/18027/16976> Acesso em: 16 de Maio 2019.

BRASIL. Ministério do Desenvolvimento Social e Combate à Fome. **Manual de instruções para utilização do prontuário SUAS**. Brasília, DF, 2014.

DUNGLAS, K. **Persistence in PHP with Doctrine ORM**. 1. ed. Birmingham: Packt Publishing, 2013. 114 p. ISBN 9781782164104

FIELDING, Roy T. **Architectural Styles and the Design of Network-based Software Architectures**. Tese de Doutorado, Universidade da California, Irvine, 2000.

FLANAGAN, D. **JavaScript: the definitive guide**. O'Reilly Media, Inc., 2011.

HUNT, A.; THOMAS, D. **The Pragmatic Programmer**. Disponível em <https://www.cin.ufpe.br/~cavmj/104The%20Pragmatic%20Programmer,%20From%20Journeyman%20To%20Master%20-%20Andrew%20Hunt,%20David%20Thomas%20-%20Addison%20Wesley%20-%201999.pdf>. Acesso em 16 de Maio 2019.

KENNEDY, B.; MUSCIANO, C. **HTML & XHTML: The Definitive Guide**. O'Reilly, 2006.

KOSCIANSKI, A. **Qualidade de Software: Aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software**. 2. ed. São Paulo: Novatec Editora, 2007. 395 p. ISBN 9788575221129.

KYRIAKIDIS, A; MANIATIS, K. **The Majesty of Vue.js**. 1. ed. Birmingham: Packt Publishing, 2016. 240 p. ISBN 9781787124370.

LOPEZ, A. **Learning PHP 7: Learn the art of PHP programming through this example-rich book filled to the brim with tutorials every PHP developer needs to know**. 1. ed. Birmingham: Packt Publishing, 2016. 393 p. ISBN 9781785880544

MARCONDES, C. A. **Programando em HTML 4.0**. 4. ed. São Paulo: Editora Érica, 2000. 275 p. ISBN 9788571945791

MARIADB, **About MariaDB**. Disponível em <https://mariadb.org/about/>. Acesso em 16 de Maio 2019.

MCCOOL, S. **Laravel Starter: The definitive introduction to the Laravel PHP web development framework**. 1. ed. Birmingham: Packt Publishing, 2012. 51 p. ISBN 9781782160908

MEYER, E. A. **CSS: The Definitive Guide**. 3. ed. O'Reilly Media, Inc., 2006.

MILETTO, E. A.; BERTAGNOLLI, S. C. **Desenvolvimento de software II: introdução ao desenvolvimento web com HTML, CSS, JavaScript e PHP**. 1. ed. Porto Alegre: Bookman, 2014. 266 p. ISBN 9788582601969.

MOZILLA, **HTML**. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTML>. Acesso em: 22 de Abril 2019.

NEGRINO, T.; SMITH, D. **JavaScript para world wide web**. Pearson Education, 2001.

PRESSMAN, R. S. **Engenharia de software: uma abordagem profissional**. 7.ed. Porto Alegre: AMGH, 2011. 780 p. ISBN 9788563308337.

RAZZOLI, F. **Mastering MariaDB: Debug, secure, and back up your data for optimum server performance with MariaDB**. 1. ed. Birmingham: Packt Publishing, 2014. 100 p. ISBN 9781783981540

RICARTE, I. L. M. **Programação Orientada a Objetos: Uma Abordagem com Java**. Disponível em <http://www.dca.fee.unicamp.br/cursos/PooJava/Aulas/poojava.pdf>. Acesso em 16 de Maio 2019.

SAUDATE, Alexandre. **REST: Construa API's inteligentes de maneira simples**. 1. ed. São Paulo: Casa do Código, 2014. 101 p.

SCHMITZ, D. **Vue.js na prática**. 1. ed. Canada: Leadpub, 2016. 39 p.

SILVA, L. L. B.; PIRES, D. F.; NETO, S. C. **Desenvolvimento de Aplicações para Dispositivos Móveis: Tipos e Exemplo de Aplicação na plataforma Ios**. 2015. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/wicsi/2015/004.pdf> Acesso em: 30 de Maio 2019.

SOMMERVILLE, I. **Engenharia de software**. 9.ed. São paulo: Pearson Prentice Hall, 2011. 529 p. ISBN 9788579361081.

TANENBAUM, A. S. **Redes de computadores**. 4. ed. Amsterdam, 2003. 945 p. ISBN 8535211853

WEILKIENS, T.; OESTEREICH, B. **UML 2 Certification Guide : Fundamental and Intermediate Exams**. 1. ed. Amsterdam: Morgan Kaufmann, 2007. 294 p. ISBN 9780123735858

WILDT, D. **eXtreme Programming: Práticas para o dia a dia no desenvolvimento ágil de software**. 1. ed. São Paulo: Casa do Código. 2015. 128 p.

APÊNDICE A - Legislação e normas

Quadro 2 - Legislação e normas

Legislação/Normas	Descrição
1 - Leis Orgânicas de Assistência Social (LOAS)	Material produzido pelo Ministério do Desenvolvimento Social e de Combate à Fome (MDS), serve para consulta por coordenadores, técnicos e gestores sobre as normas que regulamentam a assistência social.
2 - Regimento Interno do Centro Social A Casa do Piá	Documento redigido pela ABASE, definindo as normas internas da Casa do Piá, a definição das suas operações, organização, deveres e direitos.
3 - LEI Nº 8.662/1993	Lei que prevê as obrigações e direitos da Assistente Social, descrevendo quais atividades deve desenvolver e prezar pelo sigilo das informações de quem atende.
4 - Norma Operacional Básica do SUAS	Norma que normatiza a gestão pública da Assistência Social em concordância com a LOAS e a Constituição Federal Brasileira de 1988.
5 - Orientações Técnicas da Proteção Social Básica do Sistema Único de Assistência Social – SUAS: Centro de Referência de Assistência Social – CRAS	É um conjunto de diretrizes e informações sobre a implantação e funcionamento dos CRAS nos municípios brasileiros.
6 - Orientações Técnicas: Centro de Referência Especializado de Assistência Social (CREAS)	Define as regras para a implantação e funcionamento adequado de uma unidade do CREAS em municípios brasileiros.
7 - MANUAL DE INSTRUÇÕES PARA UTILIZAÇÃO DO PRONTUÁRIO SUAS	É um conjunto de diretrizes que definem como deve ser um Prontuário SUAS, exemplificando quais dados um cadastro deve ter.

Fonte: Documentos fornecidos pela Casa do Piá

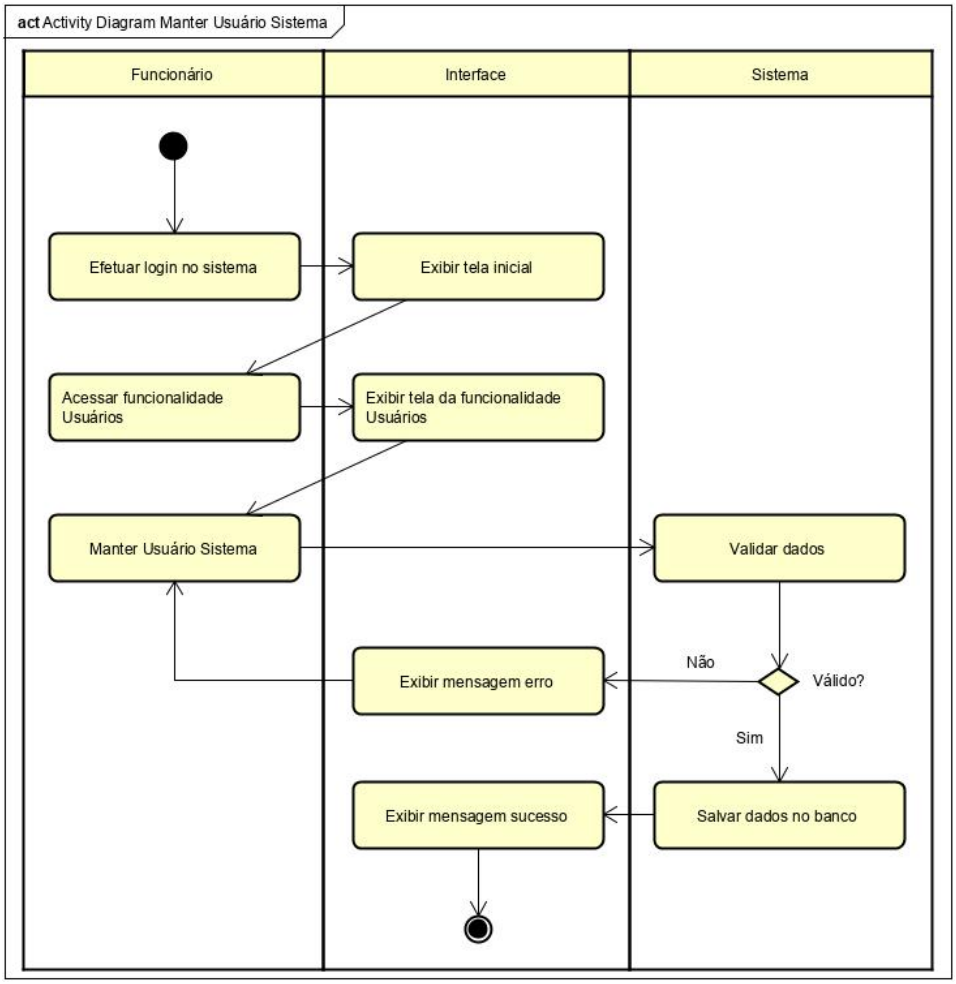
APÊNDICE B - Diagramas de atividades

DIAGRAMAS DE ATIVIDADES DESENVOLVIDOS PARA MODELAGEM DO SISTEMA

A Figura 43 apresenta o diagrama de atividades para o cadastro de usuários do sistema. Primeiramente o usuário vai acessar o sistema e logo em seguida será exibida a tela inicial, a qual terá um menu para acessar a funcionalidade “Usuários”. Ao acessar a funcionalidade, o sistema exibirá na tela uma listagem dos usuários já cadastrados.

Nesta tela, será possível também cadastrar um novo ou editar um usuário já existente, ao realizar uma destas ações o sistema vai validar os dados. Caso a validação falhe, exibirá uma mensagem de erro para o usuário, caso a validação seja bem sucedida, os dados serão gravados no banco e exibirá uma mensagem de sucesso para o usuário.

Figura 43 - Diagrama de Atividades Manter Usuário Sistema

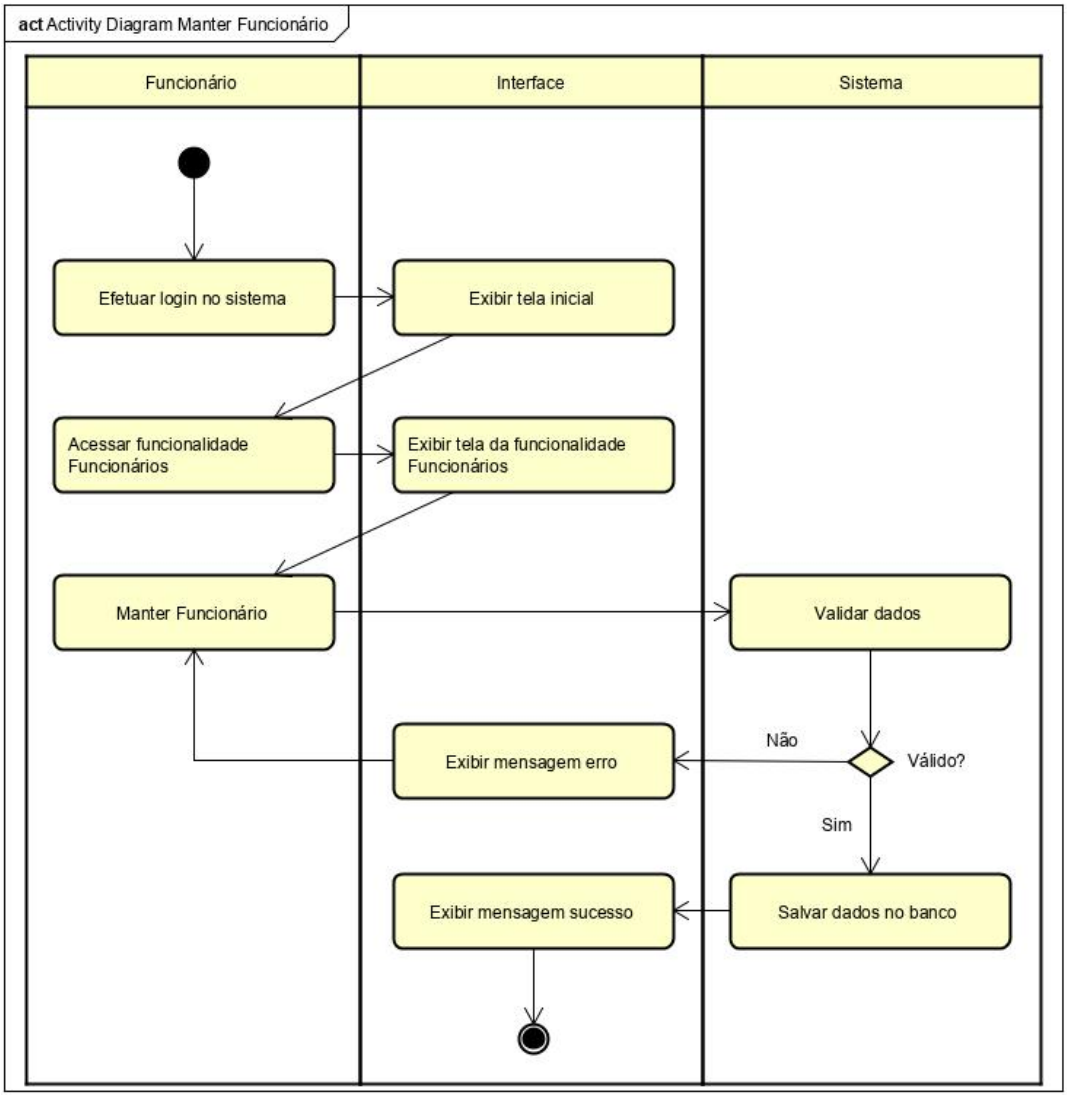


Fonte: Autoria Própria

A Figura 44 apresenta o diagrama de atividades para o cadastro de funcionários no sistema. Primeiramente o usuário vai acessar o sistema e logo em seguida será exibida a tela inicial, em que terá um menu para acessar a funcionalidade “Funcionários”. Ao acessar a funcionalidade, o sistema irá exibir na tela uma listagem dos funcionários já cadastrados.

Nesta tela, será possível também cadastrar novos ou editar funcionários já existentes. Ao realizar uma destas ações o sistema vai validar os dados, caso esta falhe, exibirá uma mensagem de erro para o usuário, caso contrário, os dados serão gravados no banco e exibirá uma mensagem de sucesso para o usuário.

Figura 44 - Diagrama de Atividades do Manter Funcionário

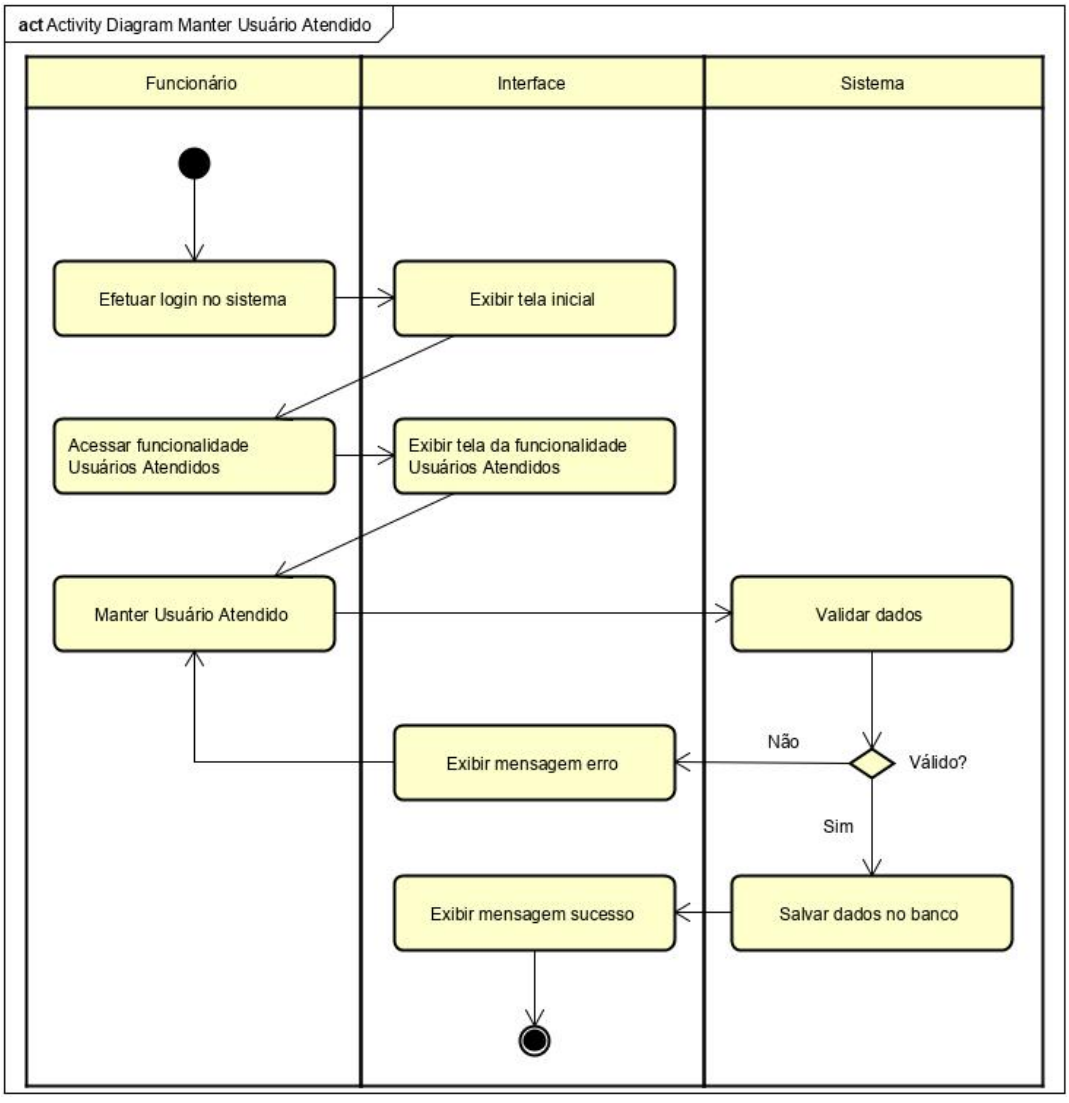


Fonte: Autoria Própria

A Figura 45 apresenta o diagrama de atividades para o cadastro de usuários da Casa do Piá no sistema. Primeiramente o usuário vai acessar o sistema e logo em seguida será exibida a tela inicial, em que terá um menu para acessar a funcionalidade “Usuários Atendidos”. Ao acessar a funcionalidade, o sistema irá exibir na tela uma listagem dos usuários já cadastrados.

Nesta tela, será possível também cadastrar um novo ou editar um usuário já existente, ao realizar uma destas ações o sistema vai validar os dados. Caso a validação falhe, exibirá uma mensagem de erro para o usuário, caso seja bem sucedida, os dados serão gravados no banco e exibirá uma mensagem de sucesso para o usuário.

Figura 45 - Diagrama de Atividades do Manter Usuário Atendido

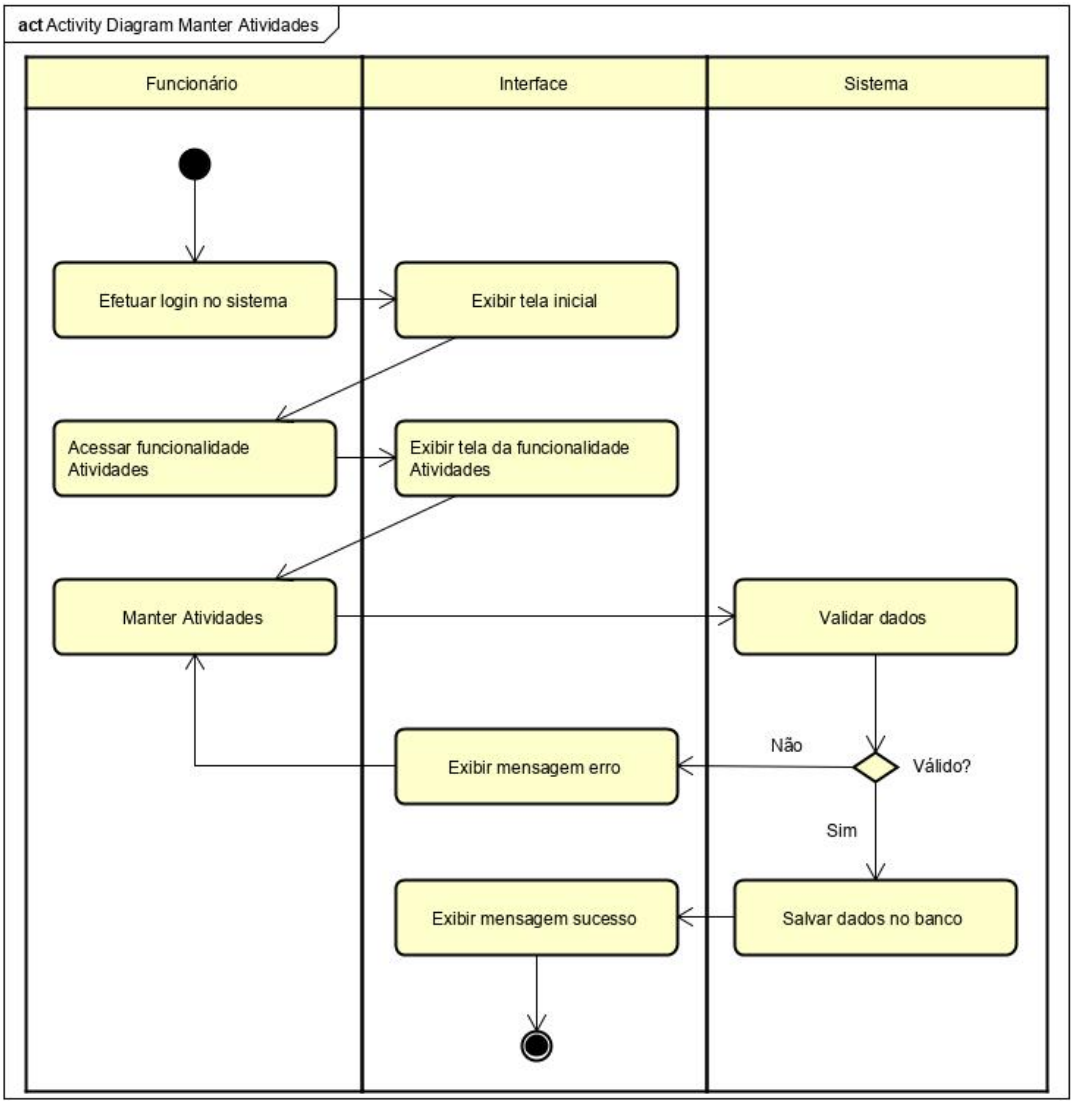


Fonte: Autoria Própria

A Figura 46 apresenta o diagrama de atividades para o cadastro de atividades da Casa do Piá no sistema. Primeiramente o usuário vai acessar o sistema e logo em seguida será exibida a tela inicial, em que terá um menu para acessar a funcionalidade “Atividades”. Ao acessar a funcionalidade, o sistema irá exibir na tela uma listagem das atividades já cadastrados.

Nesta tela, será possível também cadastrar uma nova ou editar uma atividade já existente, ao realizar uma destas ações o sistema vai validar os dados. Caso a validação falhe, exibirá uma mensagem de erro para o usuário, caso seja bem sucedida, os dados serão gravados no banco e exibirá uma mensagem de sucesso para o usuário.

Figura 46 - Diagrama de Atividades Manter Atividades

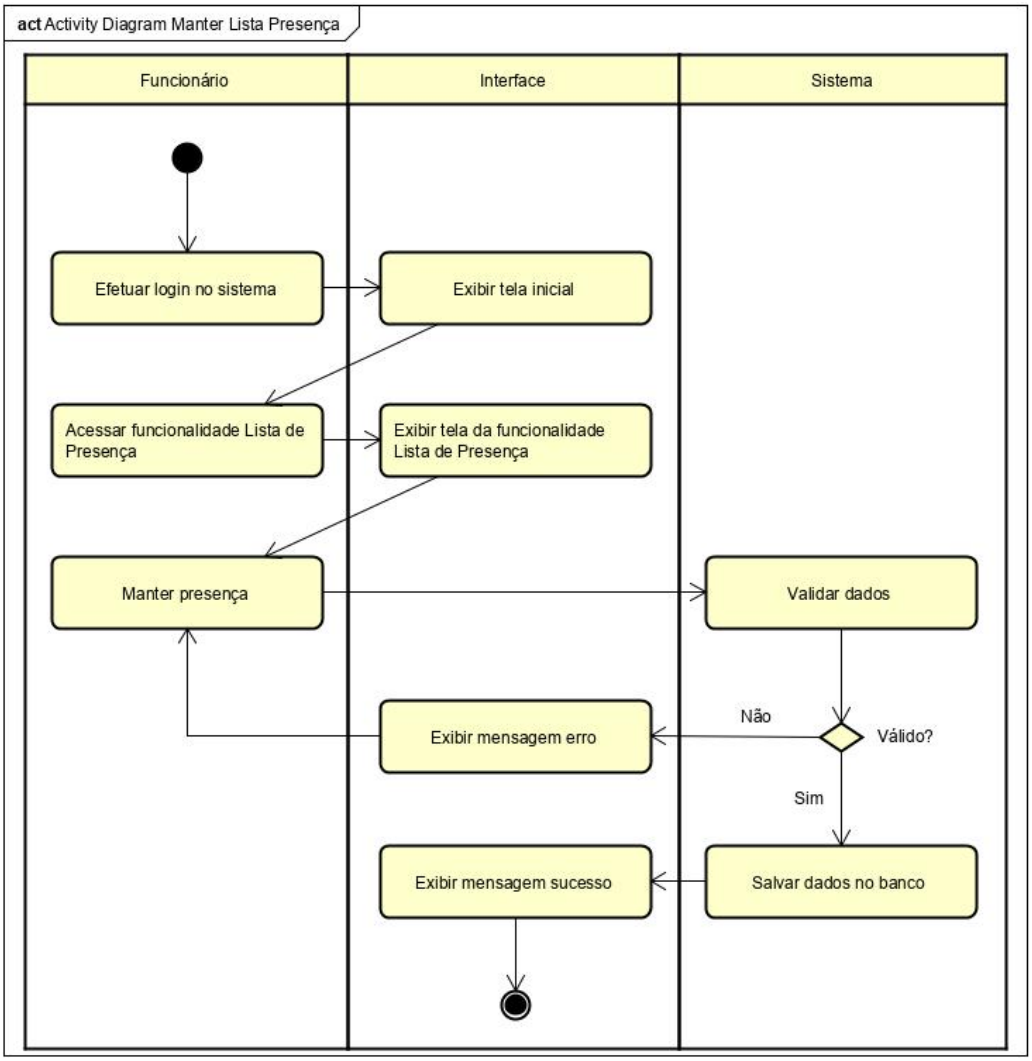


Fonte: Autoria Própria

A Figura 47 apresenta o diagrama de atividades para registrar a presença dos usuários nas atividades realizadas na Casa do Piá. Primeiramente o usuário vai acessar o sistema e logo em seguida será exibida a tela inicial, em que terá um menu para acessar a funcionalidade “Lista de Presença”. Ao acessar a funcionalidade, o sistema irá exibir na tela uma listagem das atividades de acordo com o professor.

Nesta tela, será possível inserir a presença dos usuários em uma atividade, ao realizar esta ação o sistema validará os dados. Caso a validação falhe, exibirá uma mensagem de erro para o usuário, caso seja bem sucedida, os dados serão gravados no banco e exibirá uma mensagem de sucesso para o usuário.

Figura 47 - Diagrama de Atividades Manter Lista Presença

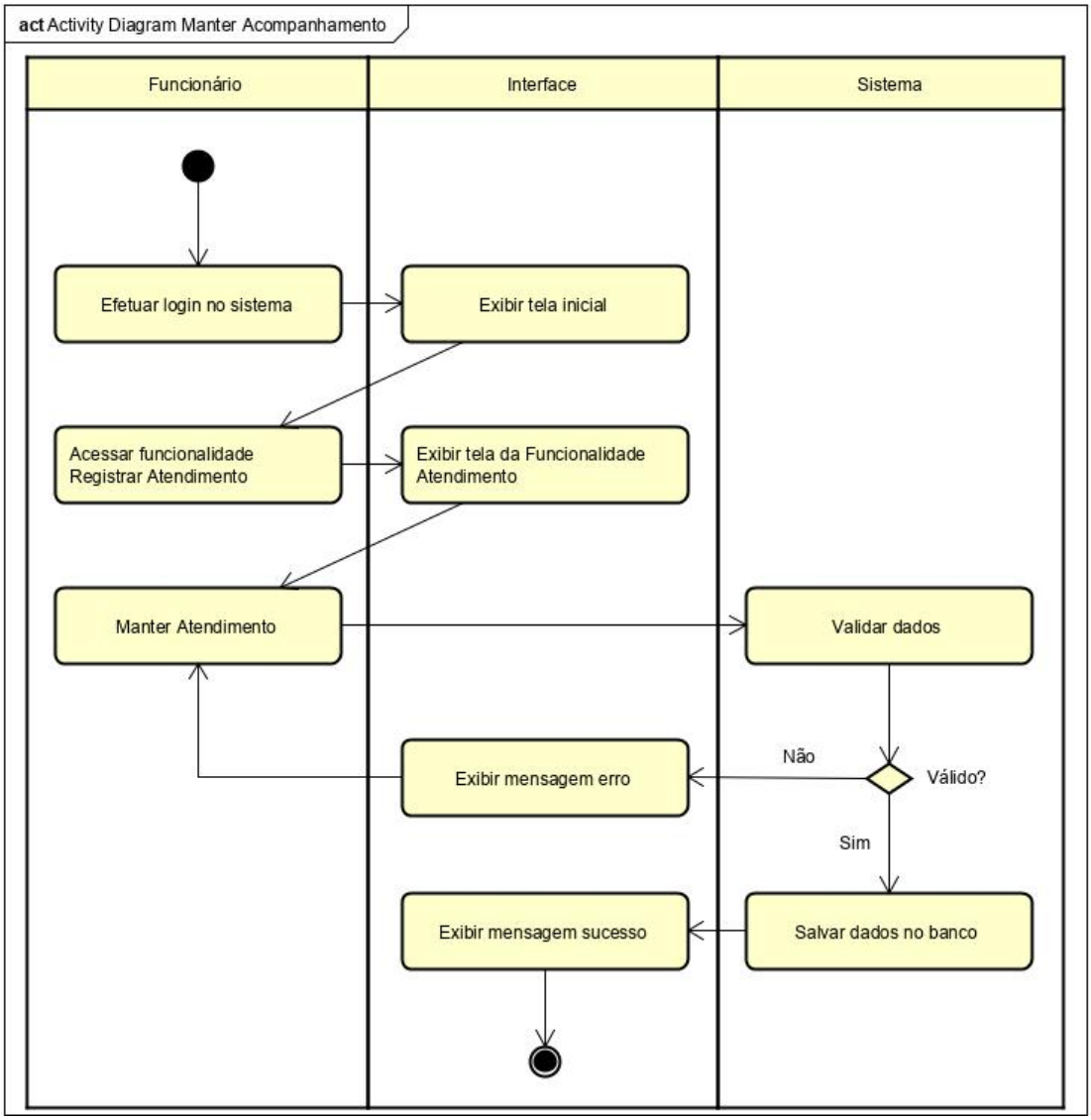


Fonte: Autoria Própria

A Figura 48 apresenta o diagrama de atividades para registrar no sistema as consultas realizadas pelo assistente social ou psicólogo. Primeiramente o usuário vai acessar o sistema e logo em seguida será exibida a tela inicial, a qual terá um menu para acessar a funcionalidade “Atendimento”. Ao acessar a funcionalidade, o sistema irá exibir na tela uma listagem dos atendimentos já cadastrados para visualização.

Nesta tela, será possível inserir um novo atendimento, ao realizar esta ação o sistema vai validar os dados. Caso a validação falhe, exibirá uma mensagem de erro para o usuário, caso a validação seja bem sucedida, os dados serão gravados no banco e exibirá uma mensagem de sucesso para o usuário.

Figura 48 - Diagrama de Atividades do Manter Acompanhamento



Fonte: Autoria Própria