UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

GUSTAVO MACIAS CORRÊA

SISTEMA GERADOR DE DOCUMENTOS PARA A ENTIDADE SOCIAL CASA DO PIÁ

PONTA GROSSA

2022

GUSTAVO MACIAS CORRÊA

SISTEMA GERADOR DE DOCUMENTOS PARA A ENTIDADE SOCIAL CASA DO PIÁ

Document generator system for the social entity Casa do Piá

Trabalho de conclusão de curso de graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação, do Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná (UTFPR).

Orientador: Prof. Dr. Richard Duarte Ribeiro

Coorientador: Prof. MSc. Vinícius Camargo Andrade

PONTA GROSSA

2022

GUSTAVO MACIAS CORRÊA

SISTEMA GERADOR DE DOCUMENTOS PARA A ENTIDADE SOCIAL CASA DO PIÁ

Trabalho de Conclusão de curso apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 04/novembro/2022

Richard Duarte Ribeiro
Doutor
Universidade Tecnológica Federal do Paraná

Vinícius Camargo Andrade
Mestre
Universidade Tecnológica Federal do Paraná

Simone de Almeida
Doutora
Universidade Tecnológica Federal do Paraná

Geraldo Ranthum
Mestre
Universidade Tecnológica Federal do Paraná

PONTA GROSSA

2022

AGRADECIMENTOS

Gostaria de deixar registrado a minha gratidão a todos que me apoiaram nessa fase da minha vida, todos foram importantes para que eu chegasse onde cheguei.

Agradeço aos meus professores orientadores, o Prof. Dr. Richard e o Prof. MSc. Vinícius por me guiarem durante esse processo de elaboração do TCC, e sempre dispostos a me ajudar.

Agradeço aos meus pais, Sandro e Denise, minha madrinha, Nadmar, minha avó e meu irmão pelo apoio demonstrado ao longo de todo o período de tempo em que me dediquei a este trabalho.

À minha noiva, Leticia, por toda ajuda e apoio, e correções no texto do trabalho quando eu precisei.

E aos funcionários de minha equipe na PagSeguro, por compartilharem seus conhecimentos e todo aprendizado e experiência que obtive trabalhando na empresa.

E por fim, mas não menos importante, agradeço à instituição Casa Do Piá, pela oportunidade de colaborar com eles fazendo este projeto.

RESUMO

Entidades sociais são importantes pois oferecem o serviço de assistência social para os mais carentes, e essas instituições são monitoradas pelo Conselho Nacional de Assistência Social, e pelo Conselho Municipal de Assistência Social (CMAS) do seu município. Para a manutenção da inscrição da entidade social com o CMAS, é necessário a entrega de documentos, controle de presenças e relatórios técnicos ou psicológicos sobre os usuários atendidos. Esses documentos eram feitos manualmente pela assistente social da instituição e então os dados eram repassados ao documento no computador para serem impressos. Além de ser trabalhoso, há um problema com a privacidade desses documentos, os quais contêm dados sensíveis. O objetivo deste trabalho foi desenvolver uma aplicação *Desktop* para simplificar e agilizar o processo de criação dos documentos e relatórios. O trabalho tem como foco a geração de documentação para a Casa do Piá, uma entidade social do município de Ponta Grossa – PR, junto com a documentação necessária para o desenvolvimento e manutenção do *software*. O produto deste trabalho foi um *software* desenvolvido em Java que realiza o controle das atividades relacionadas às crianças e jovens atendidas pela instituição. Durante o desenvolvimento utilizou-se ferramentas e *frameworks* populares no mercado de trabalho, como o Spring, Hibernate e Gradle, além da interface gráfica em JavaFX. A intenção é que ele possa ajudar na administração dos usuários da entidade, bem como na diminuição do tempo de trabalho dedicado à confecção dos documentos que devem ser enviados ao CMAS.

Palavras-chave: Java; desenvolvimento; assistência social; software; Ponta Grossa; JavaFX; Spring.

ABSTRACT

Social entities are important because they offer social assistance services to the neediest, and these institutions are monitored by the National Council of Social Assistance, and by the Municipal Council of Social Assistance (CMAS) of their municipality. In order to maintain the registration of the social entity with the CMAS, it is necessary to deliver documents, control attendance and technical or psychological reports on the users served. These documents were made manually by the institution's social worker and then the data were transferred to the document on the computer to be printed. In addition to being cumbersome, there is a problem with the privacy of these documents, which contain sensitive data. The objective of this work was to develop a Desktop application to simplify and speed up the process of creating documents and reports. The work focuses on the generation of documentation for Casa do Piá, a social entity in the municipality of Ponta Grossa - PR, along with the necessary documentation for the development and maintenance of the software. The product of this work was a software developed in Java that performs the control of activities related to children and young people served by the institution. During the development, popular tools and frameworks in the job market were used, such as Spring, Hibernate and Gradle, in addition to the graphical interface in JavaFx. The intention is that it can help in the administration of the entity's users, as well as in the reduction of the work time dedicated to the preparation of documents that must be sent to CMAS.

Keywords: Java; development; social assistance; software; JavaFX; Spring.

LISTA DE FIGURAS

FIGURA 1 – FRAGMENTO DO QUADRO *KANBAN* UTILIZADO 16
FIGURA 2 – MODELO ESPIRAL TÍPICO 18
FIGURA 3 - PROCESSO DE DESENVOLVIMENTO DE PROTÓTIPO 21
FIGURA 4 – ELEMENTOS UML PARA DIAGRAMA DE CASO DE USO 25
FIGURA 5 – EXEMPLO DE DIAGRAMA DE CLASSES 27
FIGURA 6 – EXEMPLO DE DER 28
FIGURA 7 – ESTRUTURA PARA UM REQUISITO NA FERRAMENTA EASYBACKLOG 30
FIGURA 8 – EXEMPLO DE CARD DO EASYBACKLOG 30
FIGURA 9 – INTERFACE DO SPRING INITIALIZR 32
FIGURA 10 – ESTRUTURA DO PROJETO PADRÃO DO SPRING INITIALIZR ..32
FIGURA 11 – REQUISITOS FUNCIONAIS LEVANTADOS PARA A PRIMEIRA VERSÃO 37
FIGURA 12 – CRITERIOS DE ACEITE 38
FIGURA 13 – REQUISITOS NÃO FUNCIONAIS E CRITÉRIOS DE ACEITE DA PRIMEIRA VERSÃO 38
FIGURA 14 – DIAGRAMA DE CASO DE USO DO LOGIN 39
FIGURA 15 – DIAGRAMA DE CASO DE USO DAS FUNCIONALIDADES 40
FIGURA 16 – PROTÓTIPO DA TELA DE LOGIN 41
FIGURA 17 – PROTÓTIPO DA TELA DE CADASTRO DE BENEFICIÁRIO 42
FIGURA 18 – CONTINUAÇÃO PROTÓTIPO DA TELA DE CADASTRO DE BENEFICIÁRIO 43
FIGURA 19 – DIAGRAMA DE CLASSES DA V1 FOCADO NO BENEFICIÁRIO .44
FIGURA 20 – DIAGRAMA DE CLASSES DA V1 FOCADO NO FUNCIONÁRIO..45
FIGURA 21 – DER DO SISTEMA PARA A PRIMEIRA VERSÃO 46
FIGURA 22 – DEPENDÊNCIAS DO SISTEMA DA CASA DO PIÁ 47
FIGURA 23 – EXEMPLO DO ARQUIVO APPLICATION.PROPERTIES 48
FIGURA 24 – TELA DE LOGIN DO SISTEMA 49
FIGURA 25 – TELA PRINCIPAL DA VERSÃO 1 49
FIGURA 26 – TELA DE CADASTRO DE DADOS E ENDEREÇO DO BENEFICIÁRIO 50
FIGURA 27 – TELA DE CADASTRO DE ASPECTOS FAMILIARES DO BENEFICIÁRIO 51
FIGURA 28 – TELA DE CADASTRO DE HABITAÇÃO E SAÚDE DO BENEFICIÁRIO 51
FIGURA 29 – TELA DE CADASTRO DE ASSISTÊNCIA SOCIAL E ESCOLARIDADE DO BENEFICIÁRIO 52
FIGURA 30 – CLASSE DE DOMÍNIO E CLASSE DE ENTIDADE 53
FIGURA 31 – MÉTODO TOENTITY 54
FIGURA 32 – EXEMPLO DE CÓDIGO DA CLASSE DE SERVIÇO 54

FIGURA 33 – MÉTODO DA CLASSE DE CONTROLE 55
FIGURA 34 – CLASSE DE SERVIÇO DE FUNCIONÁRIOS.....56
FIGURA 35 – CARDS DETALHADOS NA VERSÃO 2.....58
FIGURA 36 – TELA DE CADASTRO DE DADOS PESSOAIS DO BENEFICIÁRIO 59
FIGURA 37 – TELA DE CADASTRO DE ENDEREÇO DO BENEFICIÁRIO.....60
FIGURA 38 – TELA DE CADASTRO DE ASPECTOS FAMILIARES DO BENEFICIÁRIO.....61
FIGURA 39 – TELA DE CADASTRO DE DADOS DE HABITAÇÃO DO BENEFICIÁRIO.....61
FIGURA 40 – TELA DE CADASTRO DE DADOS DE SAÚDE DO BENEFICIÁRIO 62
FIGURA 41 – TELA DE CADASTRO DE DADOS DE ESCOLARIDADE DO BENEFICIÁRIO.....62
FIGURA 42 – TELA DE CADASTRO DE DADOS DE ASSISTÊNCIA SOCIAL DO BENEFICIÁRIO.....63
FIGURA 43 – TELA DE DADOS DE INGRESSO DO BENEFICIÁRIO.....63
FIGURA 44 – TELA DE BUSCA DE BENEFICIÁRIOS.....64
FIGURA 45 – TELA DE VISUALIZAÇÃO DE DADOS PESSOAIS 65
FIGURA 46 – MENU PRINCIPAL ATUALIZADO PARA A VERSÃO 2.....66
FIGURA 47 – DIAGRAMA DE CLASSE DA VERSÃO 2 67
FIGURA 48 – DER ATUALIZADO DA VERSÃO 2.....68
FIGURA 49 – PROPRIEDADES ADICIONADAS AO HIBERNATE.....68
FIGURA 50 – EXEMPLO DE POP-UP USADO.....69
FIGURA 51 – CÓDIGO PARA APRESENTAÇÃO DE MENSAGEM POP-UP 69
FIGURA 52 – CAMPOS DESTACADOS COM DADOS INVÁLIDOS 70
FIGURA 53 – APRESENTAÇÃO DE UM CAMPO INVÁLIDO.....70
FIGURA 54 – MÉTODO PARA FILTRAR OS NOMES DOS BENEFICIÁRIOS.....71
FIGURA 55 – MÉTODO PARA BUSCA DE DADOS DE BENEFICIÁRIOS 71
FIGURA 56 – MÉTODOS DE SERVIÇO DO BENEFICIÁRIO.....71
FIGURA 57 – MÉTODO "*FROMENTITY*" 72
FIGURA 58 – MÉTODO PARA ATUALIZAÇÃO DE DADOS DO BENEFICIÁRIO 72
FIGURA 59 – MÉTODO QUE FAZ A CONSULTA NA API DE CEPS.....73
FIGURA 60 – MÉTODO DE GERAÇÃO DE DOCUMENTO DE INSCRIÇÃO 74
FIGURA 61 – TELAS DE RECUPERAÇÃO DE SENHA 75
FIGURA 62 – TELA DE RELATÓRIOS.....77
FIGURA 63 –TELA DE BUSCA DE BENEFICIÁRIOS.....77
FIGURA 64 – TELA DE DESLIGAMENTO DE BENEFICIÁRIOS.....78
FIGURA 65 – CLASSE "BENEFICIARIO" ATUALIZADA NA VERSÃO 3.....79
FIGURA 66 – DER ATUALIZADO COM A ENTIDADE DESLIGAMENTO.....80
FIGURA 67 – MÉTODO PARA ENVIO DE E-MAIL DE RECUPERAÇÃO DE SENHA.....81

FIGURA 68 – CHAMADA DO ENVIO DE E-MAIL 81
FIGURA 69 – CLASSE DE ENTIDADE "DESLIGAMENTO".....82
FIGURA 70 – MÉTODO DE DESLIGAMENTO DO BENEFICIÁRIO 82
FIGURA 71 – QUERYS PARA DADOS DO RELATÓRIO 83
FIGURA 72 – PROTÓTIPO TELA DE BUSCA DE BENEFICIÁRIOS INATIVOS..85
FIGURA 73 – PROTÓTIPO DA TELA DE CADASTRO DE ATIVIDADES 86
FIGURA 74 – PROTÓTIPO DA TELA DE EDIÇÃO DE ATIVIDADES.....86
FIGURA 75 – PROTÓTIPO DA TELA DE PRESENÇA DE BENEFICIÁRIOS 87
FIGURA 76 – DIAGRAMA DE CLASSE DA VERSÃO 4 88
FIGURA 77 – DER ATUALIZADO DA VERSÃO 4.....89
FIGURA 78 – MÉTODO DE BUSCA DE BENEFICIÁRIOS POR STATUS 90
FIGURA 79 – CLASSE DE REPOSITÓRIO DE ATIVIDADES.....90
FIGURA 80 – MAPEAMENTO DE ENTIDADE ASSOCIATIVA COM HIBERNATE 91
FIGURA 81 – BUSCA DE ATIVIDADES DO BENEFICIÁRIO EM UM MÊS.....91

LISTA DE ABREVIATURAS, SIGLAS E ACRÔNIMOS

ACID	Atomicidade, Consistência, Isolamento e Durabilidade
API	*Application Programming Interface*
CNAS	Conselho Nacional de Assistência Social
CMAS	Conselho Municipal de Assistência Social
CRAS	Centro de Referência de Assistência Social
CRUD	*Create, Read, Update e Delete*
DAO	*Data Access Object*
DER	Diagrama Entidade-Relacionamento
HTML	*HyperText Markup Language*
JAR	*Java Archive*
JIT	*Just In Time*
JPA	*Java Persistence API*
JVM	*Java Virtual Machine*
SQL	*Structured Query Language*
UML	*Unified Modeling Language*
WiP	*Work in Progress*

SUMÁRIO

1 INTRODUÇÃO 12
1.1 Objetivos 13
1.1.1 Objetivo geral 14
1.1.2 Objetivos específicos 14
1.2 Estrutura do trabalho 14
2 REFERENCIAL TEÓRICO 15
2.1 Quadro de Kanban 15
2.2 Engenharia de Software 16
2.2.1 Modelo de processo evolucionário 17
2.2.2 Engenharia de requisitos 19
2.2.3 Prototipação 20
2.3 Modelos de Software 22
2.3.1 Unified Modeling Language (UML) 23
2.3.1.1 Diagrama de Caso de Uso 24
2.3.1.2 Diagrama de Classe 26
2.3.2 Diagrama Entidade-Relacionamento (DER) 28
2.4 Ferramentas e Frameworks 29
2.4.1 Ferramentas para o projeto do software 29
2.4.2 Ferramentas para desenvolvimento do software 31
2.4.3 Ferramentas para implantação do software 34
3 DESENVOLVIMENTO 35
3.1 Configuração, Login e Cadastro – Versão 1 35
3.1.1 Comunicação 36
3.1.2 Projeto do Software 43
3.1.3 Desenvolvimento da Versão 1 46
3.1.3.1 Subversão de configuração – Configurações do sistema 46
3.1.3.2 Interfaces de usuário desenvolvidas 49
3.1.3.3 Implementação do código-fonte 52
3.1.4 Feedback da Casa do Piá 56
3.2 Visualização de Beneficiários e Geração do Documento de Inscrição – Versão 2 57
3.2.1 Comunicação com o cliente 57
3.2.2 Projeto de software 66
3.2.3 Desenvolvimento 68
3.2.4 Feedback da Casa do Piá 74
3.3 Versão 3 – Relatórios e desligamento de beneficiários 75
3.3.1 Comunicação com a Casa do Piá 75
3.3.2 Projeto de Software 78
3.3.3 Desenvolvimento 80

3.3.4 Feedback da Casa do Piá 84
3.4 Versão 4 – Registro de atividades e presença de beneficiários 84
3.4.1 Comunicação 85
3.4.2 Projeto de Software 87
3.4.3 Desenvolvimento 89
3.4.4 Feedback da Casa do Piá 91
4 CONCLUSÃO 93
4.1 Trabalhos futuros 94
REFERÊNCIAS 95
ANEXO A – MODELO DO DOCUMENTO DE CADASTRO DA CASA DO PIÁ 97

1 INTRODUÇÃO

A assistência social é definida pelo Art. 1 da Lei nº. 8742, de 7 de dezembro de 1993, como “direito do cidadão e dever do Estado”, e é “realizada através de um conjunto integrado de ações de iniciativa pública e da sociedade, para garantir o atendimento às necessidades básicas” (BRASIL, 1993). Ela é importante, principalmente para a parcela mais necessitada da população, pois garante a eles os seus direitos fundamentais.

A cidade de Ponta Grossa possui várias instituições sem fins lucrativos que possuem como objetivo o proposto pela Lei nº. 8742, sendo uma delas a Casa do Piá. A Casa do Piá oferece atividades no contraturno escolar para cerca de 165 crianças e adolescentes de 6 a 17 anos, atuando no município de Ponta Grossa desde 1998. As atividades oferecidas pela Casa do Piá às crianças e adolescentes atendidos são: refeições, revisão de higiene, curso de informática, apoio escolar, esporte, dança, coral e formação humana e religiosa (CENTRO, 2014).

As entidades sociais são monitoradas pelo Conselho Nacional de Assistência Social (CNAS)[1] e pelo Conselho Municipal de Assistência Social (CMAS)[2] do município. Para continuar em atividade, a Casa do Piá deve entregar anualmente a documentação definida na Resolução CNAS n° 14 de maio de 2014 (BRASIL, 2014). Além disso, anualmente é definida uma resolução no CMAS Ponta Grossa detalhando a documentação a ser entregue pela Casa do Piá. A última resolução detalhando essa documentação é a Resolução n° 2 de fevereiro de 2020 (PONTA GROSSA (PR) 2022), disponível no website do CMAS[2].

Para o funcionamento de uma entidade social, é necessário controle sobre os dados das crianças atendidas, assim como verificação de presença, relatório de atividades, plano de trabalho, e inscrição de novas crianças e/ou adolescentes, por exemplo. Para isso são coletados dados da família, saúde, escolaridade, e origem do inscrito, o qual pode ter sido encaminhado pelo Centro de Referência de Assistência Social (CRAS), ou pode ter buscado o serviço pessoalmente.

[1] Site do CNAS: http://www.mds.gov.br/cnas
[2] Site do CMAS: https://cmas.pontagrossa.pr.gov.br

Atualmente na Casa do Piá, o controle de presença e a escrita de relatórios técnicos ou psicológicos referentes aos jovens atendidos são feitos manualmente e individualmente, sendo em seguida repassados para planilhas e documentos eletrônicos, tornando o processo repetitivo e trabalhoso. Neste contexto, este trabalho propõe o desenvolvimento de um *software* para gerenciar a movimentação e permanência dos jovens na instituição por meio de registros dos dados pessoais de cada um. A partir disso são gerados documentos para o cumprimento da legislação vigente, desta maneira, otimizando o processo e centralizando os dados dos usuários da Casa do Piá na aplicação, garantindo o controle de acesso e a segurança dos dados.

Para o desenvolvimento deste trabalho, obteve-se os requisitos do *software* a partir de uma entrevista aberta inicial seguida da apresentação de um protótipo. Utilizou-se de outras entrevistas fechadas e/ou abertas para levantar novos requisitos, refinar e validar as funcionalidades obtidas nas entrevistas anteriores. Com estes requisitos corretos e completos, realizou-se a modelagem dos diagramas de Caso de Uso e Classe, referentes à *Unified Modeling Language* (UML). Além destes diagramas, modelou-se o Diagrama Entidade-Relacionamento (DER) para projetar o banco de dados. Finalmente, com o sistema modelado, priorizou-se os requisitos listados para dar início ao desenvolvimento do *software*.

O *software* desenvolvido é um sistema *Desktop* voltado para a segurança e privacidade dos dados dos usuários atendidos pela Casa do Piá. A aplicação gera documentos e mantém os relatórios técnicos ou psicológicos sobre os seus usuários. Esses são necessários para que a instituição continue exercendo suas atividades. Com o uso da aplicação, os processos de cadastro e desligamento de usuários foram facilitados e acelerados. Adicionalmente, o controle das atividades e presença dos usuários são registrados no *software*, atingindo, do ponto de vista dos funcionários da Casa do Piá, os requisitos inicialmente solicitados.

1.1 Objetivos

O objetivo geral e os específicos serão descritos a seguir.

1.1.1 Objetivo geral

Desenvolver um *software* que auxilie o acompanhamento dos jovens e adolescentes atendidos pela entidade social Casa do Piá, da cidade de Ponta Grossa, bem como que gerenciar a documentação e produção de relatórios exigidos pela legislação nacional e municipal.

1.1.2 Objetivos específicos

Para atingir o objetivo geral desta pesquisa, os seguintes objetivos específicos foram definidos:

- Realizar o levantamento de requisitos funcionais e não funcionais do projeto junto aos funcionários da Casa do Piá.
- Desenvolver um protótipo para a validação dos requisitos levantados.
- Modelar os diagramas necessários para a documentação e desenvolvimento do projeto.
- Implementar o *software*.
- Validar o *software* desenvolvido a fim de obter um produto completo e correto.

1.2 Estrutura do trabalho

Este trabalho está dividido em 4 capítulos. Sendo este, o Capítulo 1, a introdução do trabalho. O Capítulo 2 aborda o referencial teórico necessário para o entendimento do trabalho. O Capítulo 3 trata dos processos de desenvolvimento do aplicativo. No Capítulo 4 são descritas as conclusões do trabalho e faz referências aos trabalhos futuros.

2 REFERENCIAL TEÓRICO

O referencial teórico foi estruturado em cinco tópicos, de modo a demonstrar a importância de cada etapa dentro do processo de desenvolvimento deste trabalho. A Seção 2.1 aborda o uso do quadro de *Kanban*. A Seção 2.2 descreve a Engenharia de *Software* e tópicos relacionados. A Seção 2.3 aborda tópicos de modelagem de *software*. A Seção 2.4 apresenta ferramentas e *frameworks* usados na implementação do trabalho.

2.1 Quadro de Kanban

Kanban é uma palavra japonesa que pode ser traduzida como *signboard* (em inglês) ou “quadro de avisos" (em português), e se tornou sinônimo de programar por demanda. O *Kanban* tem raízes no início do sistema *Toyota* de produção, e foi desenvolvido para controlar a produção entre os processos, além de implementar a manufatura *Just In Time* (JIT) nas fábricas da empresa nipônica (YASUHIRO, 2015).

No sistema *Toyota* de produção o sistema *Kanban* é sustentado pelos seguintes itens (YASUHIRO, 2015):

- Sincronização da produção;
- Padronização das operações;
- Redução do tempo de preparação;
- Atividades de melhoria;
- Projeto de layout das máquinas; e
- Autonomação[3].

O *Kanban* como é utilizado na Engenharia de *Software* foi concebido por David Anderson e é estabelecido como “um método para definir, gerenciar e melhorar serviços que entregam trabalho de conhecimento” (ANDERSON; CARMICHAEL, 2016, p.1), pois este método facilita visualizar a quantidade de trabalho, limitando a carga que a equipe é capaz de realizar. O método consiste de quadros *kanbans*, que

[3] É a junção das palavras autonomia e automação; No Toyotismo é um “Controle autônomo de defeitos” (YASUHIRO, 2015).

podem ou não ter um limite mínimo e máximo de *Work in Progress* (WiP), sendo estes limites WiP que regulam a carga de trabalho da equipe (ANDERSON; CARMICHAEL, 2016).

A Figura 1 apresenta como exemplo parte do quadro *Kanban* utilizado para organizar as tarefas e reuniões para o desenvolvimento deste TCC, e possibilitando que os interessados neste trabalho pudessem acompanhar o seu progresso.

Por intermédio deste quadro, é possível identificar o progresso das atividades. As tarefas são movidas de um quadro a outro: do "*A fazer*" ao "*Fazendo*" quando iniciadas, e do "*Fazendo*" ao "*Feito*" quando concluídas. No caso deste trabalho não foi definido WiP em nenhum dos quadros *Kanban* pois o desenvolvimento foi feito por apenas uma pessoa, tendo maior controle sobre as tarefas nos quadros.

Figura 1 – Fragmento do quadro *Kanban* utilizado

Fonte: Autoria própria (2022).

2.2 Engenharia de Software

No dicionário, a definição para *software* é: "Programa; reunião dos procedimentos e/ou instruções que determinam o funcionamento de um computador" (SOFTWARE, 2022). *Software* está presente em quase todos os dispositivos usados nos dias de hoje e é a parte do dispositivo que não podemos tocar (PFLEEGER, 2010).

Segundo Pfleeger (2010), *software* é o que controla e regula os dispositivos que usamos, simplificando ou até automatizando diversas tarefas do dia a dia. Para que os dispositivos que usamos sejam funcionais, é necessário que o sistema esteja correto. Para isso, boas práticas da engenharia de *software* garantem que o produto do desenvolvimento faça uma contribuição positiva para nossas vidas.

Pressman (2011) menciona que o *software* não se desgasta, porém se deteriora. Ou seja, com o passar dos anos, novas funcionalidades e alterações no escopo do problema são necessárias. Com a atualização do *software* para suprir as novas demandas, novos erros são inseridos, o que deteriora ainda mais o *software*, minimizando assim sua vida útil.

Para tentar maximizar a vida útil de um sistema, a disciplina de engenharia de *software*, busca analisar um problema genérico e aplicar boas práticas no desenvolvimento do seu produto. Desta maneira, o engenheiro de *software* busca soluções computacionais para o problema de seu cliente, utilizando diversos métodos e recursos para encontrar a melhor solução (PFLEEGER, 2010).

O processo para a produção de um *software* pode ser dividido em quatro atividades fundamentais: (i) especificação do *software*; (ii) desenvolvimento de *software*; (iii) validação do *software*; e (iv) evolução de *software* (SOMMERVILLE, 2011).

Para o desenvolvimento deste trabalho utilizou-se o modelo de processo evolucionário, passando diversas vezes por essas atividades. Para a especificação de *software*, optou-se pela Engenharia de Requisitos e Prototipação. No design da aplicação, utilizou-se os diagramas *Unified Modeling Language* (UML) como maneira de documentar o *software*. A implementação contou com algumas ferramentas e frameworks que serão melhores detalhados na Seção 2.4. Por fim, a validação e evolução do *software* foi realizada por meio de *feedback* com o cliente.

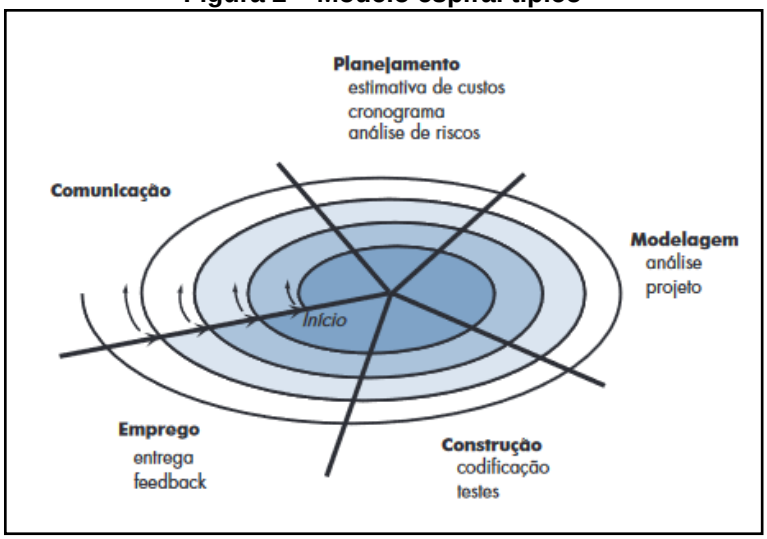
2.2.1 Modelo de processo evolucionário

Segundo Pressman (2011, p. 62 – 66), o modelo de processo evolucionário possibilita a entrega de um produto mais completo a cada versão do *software*. Na literatura há diversos modelos de processos evolucionários, como por exemplo, prototipação e modelo espiral.

O modelo espiral foi originalmente proposto por Barry Boehm, e “fornece potencial para o rápido desenvolvimento de versões cada vez mais completas de *software*.” (BOEHM, 1988). Além disso, é composto de um ciclo de atividades, com cada uma representando um segmento.

A Figura 2 ilustra o modelo espiral. O modelo possui 5 segmentos: comunicação, planejamento, modelagem, construção e entrega. Ao final de cada ciclo é entregue um produto cada vez mais completo, podendo nas primeiras iterações ser um modelo ou protótipo do *software* que evoluirá até que se torne o produto completo (PRESSMAN, 2011).

Figura 2 – Modelo espiral típico



Fonte: Pressman (2011, p. 65).

Pressman menciona que o *software* evolui ao longo do tempo, enquanto o cliente deseja ter suas necessidades supridas. Portanto, não é viável planejar a entrega de um *software* completo de uma só vez, pois levaria muito tempo para ser entregue. Assim, são lançadas diversas versões intermediárias suprindo aos poucos cada uma das necessidades do cliente (PRESSMAN, 2011).

O uso do modelo de processo evolucionário evita também um dos problemas encontrados na Engenharia de Requisitos, como por exemplo, o problema da volatilidade, o qual é definido como a natureza de mudança dos requisitos, ou seja, as necessidades do cliente mudam com o tempo, assim como os requisitos do sistema. Neste caso, com a aplicação de um modelo evolucionário, é possível diminuir o efeito da volatilidade nos requisitos, pois enquanto estiver em um ciclo evolucionário a última versão do *software* estará sempre mais completa e atualizada (CHRISTEL; KANG, 1992).

As duas principais características que distinguem o modelo espiral dos demais modelos são (BOEHM, 1988): (i) a sua abordagem cíclica (é um ciclo evolutivo em cinco etapas: comunicação, planejamento, modelagem, construção e emprego do *software*) que, ao passo que incrementa a definição e a implementação do sistema,

diminui o seu grau de risco; e (ii) pontos âncora de controle (pontos de reflexão sobre o projeto, onde é verificado o andamento do projeto, o que pode ser melhorado, entre outros) que garantem o engajamento dos interessados na busca de soluções satisfatórias e praticáveis.

Conforme mencionado por Pressman (2011), o primeiro ciclo em volta da espiral pode resultar em uma especificação de produto, para isso, pode-se utilizar técnicas da engenharia de requisitos, abordadas na Seção 2.2.2. Os ciclos subsequentes em volta da espiral podem ser utilizados para o refinamento destes requisitos que podem ser realizados por meio da técnica de prototipação, descrita na Seção 2.2.3.

2.2.2 Engenharia de requisitos

Um requisito é uma necessidade que o cliente deseja que seja suprida pelo sistema, levando em conta a viabilidade de sua implementação, e especificando a solução para essa necessidade sem duplicidade. Levantar requisitos nada mais é do que descobrir as necessidades que um sistema deve fornecer para o usuário (PRESSMAN, 2011). Esses requisitos podem ser divididos em funcionais e não funcionais. Os funcionais são funcionalidades do sistema, ou seja, descrevem o que este deve fazer. Enquanto os não funcionais definem algumas características ou restrições para o *software* (SOMMERVILLE, 2011).

Segundo Pressman (2011), a engenharia de requisitos é a primeira etapa para o desenvolvimento do *software*, e é por meio dela que é fornecido um entendimento escrito do problema à todas as partes envolvidas no projeto, podendo ser alcançado por uma série de artefatos como: casos de uso, listas de funções e características. Porém, em muitos casos o cliente não saberá o que é necessário no sistema para solucionar os seus problemas ou não terá um bom entendimento das características e funções que o trarão benefícios, dificultando a tarefa de levantamento de requisitos para o engenheiro de *software*.

Dentro da etapa de engenharia de requisitos há quatro atividades principais (SOMMERVILLE, 2011):

- Estudo de viabilidade: é feita a verificação se as necessidades do cliente podem ser supridas utilizando as tecnologias atuais de hardware e *software*.
- Elicitação de requisitos: nesta atividade os requisitos são levantados. O levantamento dos requisitos pode ser alcançado por meio de entrevistas abertas ou fechadas. A entrevista aberta facilita compreender as necessidades do cliente e explorar o sistema como um todo; e na entrevista fechada o cliente responderá a um conjunto de perguntas predefinidas e ajuda a explorar pontos específicos do sistema.
- Especificação de requisitos: os requisitos de usuário obtidos na atividade anterior devem ser reunidos em um documento que os descreva. Estes podem ser escritos em linguagem natural (linguagem desenvolvida naturalmente pelo ser humano), ou de forma estruturada (como uma tabela, ou um card com a estrutura predefinida), devendo ser escritos de forma que o cliente possa os compreender.
- Validação de requisitos: na qual é verificado se os requisitos refletem o que o cliente realmente quer. Requisitos incorretos implementados no sistema podem tomar um grande tempo para serem corrigidos. Os erros devem ser encontrados antes da implementação, sendo possível encontrá-los por meio de revisões dos requisitos, prototipação ou o uso de casos de teste.

2.2.3 Prototipação

Segundo Pfleeger (2010), por vezes o cliente não sabe exatamente o que necessita em um sistema e essas incertezas fazem com que o time de analistas e desenvolvedores enfrentem problemas ao implementar o projeto. Neste contexto, o desenvolvimento de um protótipo auxilia na validação dos requisitos funcionais junto as partes interessadas, possibilitando que os mesmos apontem problemas e melhorias ao produto que será construído.

Segundo Sommerville (2011), o protótipo ajuda a verificar possíveis mudanças que possam interferir em dois processos: (i) no processo de engenharia de requisitos pode auxiliar na análise e validação de requisitos; e (ii) no processo de

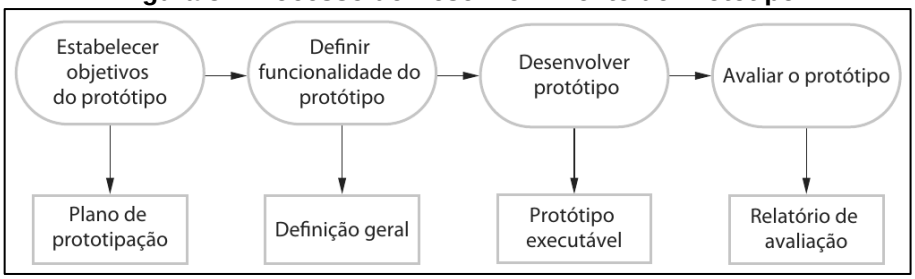
design do sistema pode ser utilizado para analisar soluções específicas do *software* e para apoiar o projeto de interface de usuário.

Mesmo com os requisitos bem definidos e especificados, quando determinadas funções dos sistemas são combinadas com outras, os usuários percebem que sua visão inicial do sistema está errada e que mudanças devem ser realizadas. Isso só é possível por meio de uma avaliação de um protótipo, caso contrário, o erro só será encontrado na versão final do sistema, custando altos valores para ser reparado (SOMMERVILLE, 2011).

Além da utilização da prototipação para validar os requisitos elicitados e verificar a viabilidade do projeto, a prototipação é comumente utilizada como parte essencial de projeto da interface de usuário. Pois, somente a modelagem e descrições textuais não são o suficiente para expressar os requisitos de usabilidade (SOMMERVILLE, 2011).

Sommerville (2011) descreve o processo de desenvolvimento de protótipo (Figura 3) o qual envolve 4 etapas: (i) estabelecer objetivos do protótipo; (ii) definir funcionalidade do protótipo; (iii) desenvolver protótipo; e (iv) avaliar o protótipo.

Figura 3 - Processo de Desenvolvimento de Protótipo



Fonte: Sommerville (2011, p. 30).

Na etapa de objetivos do protótipo são explicitadas as intenções de realizar a prototipação desde o início do processo, como por exemplo, validação de requisitos funcionais, interface de usuário, demonstrar aos gerentes a viabilidade do projeto, entre outros. Caso estes objetivos são sejam definidos, a gerência e/ou os usuários finais podem não compreender a função do protótipo, consequentemente, não obterão os benefícios da realização da prototipação.

A próxima etapa é responsável pela definição de quais funcionalidades o protótipo terá. Esta etapa visa reduzir custos e acelerar o cronograma de entrega. Para isso, algumas funcionalidades podem ser deixadas de fora do sistema prototipado, dando ênfase nos objetivos definidos na etapa anterior.

Por fim, as etapas de desenvolver e avaliar o protótipo. Nestas etapas ocorrem a implementação e avaliação do protótipo junto aos gerentes e/ou clientes do projeto, os quais poderão solicitar modificações a fim de sanar problemas encontrados, sejam de interface de usuário, funcionalidades, cronograma do projeto ou recursos da própria organização.

A implementação de um protótipo exige do time de desenvolvimento a escolha do tipo de protótipo que será construído. Dentre os tipos de protótipos existentes, dois se destacam (PFLEEGER, 2010): descartável e evolutivo. Ambos têm o propósito de validar os requisitos levantados e são desenvolvidos rapidamente, porém o descartável não tem como propósito ser integrado ao sistema final, permitindo que se deixe de lado algumas características como o seu desempenho. Em contrapartida, o evolutivo pode ser integrado ao sistema final, sendo necessário maior cuidado para seu desenvolvimento.

2.3 Modelos de Software

Um modelo é uma simplificação da realidade. Para o desenvolvimento de *software*, modelos são construídos para compreender melhor um determinado problema. Ao dividir um problema complexo, é possível encontrar uma solução para cada aspecto deste e resolvê-lo como um todo. Consequentemente, a escolha dos modelos corretos a serem usados são importantes pois estes esclarecerão o problema, enquanto os modelos incorretos podem gerar incertezas no projeto (BOOCH, 2006).

Para o presente trabalho utilizou-se o paradigma orientado a objetos por prover sistemas manuteníveis e flexíveis à novas funcionalidades. Para modelá-lo, utilizou-se a *Unified Modeling Language* (UML), apresentada na Subseção 2.3.1, uma linguagem de modelagem amplamente utilizada para apoiar a escrita de documentação de *software* e auxiliar no desenvolvimento do mesmo, minimizando as chances de se cometer erros de especificação de requisitos. Para representar graficamente a estrutura lógica do banco de dados, fez-se uso do Diagrama Entidade-Relacionamento, abordado na Subseção 2.3.2.

2.3.1 Unified Modeling Language (UML)

A *Unified Modeling Language* (UML) é uma linguagem que fornece um vocabulário de elementos para a representação dos modelos de um sistema e indicam como criar e ler modelos bem estruturados. Esses modelos podem representar diversas visões do *software* a ser desenvolvido, e cabe ao desenvolvedor escolher os melhores modelos e visões para projetar o *software*.

O vocabulário da UML contém três tipos de blocos de construção (BOOCH, 2006):

i. itens: são os tipos de elementos básicos da UML;
ii. relacionamentos: são elementos que reúnem os itens; e
iii. diagramas: são visões do sistema sobre diferentes perspectivas, são formados por itens e relacionamentos.

Os itens podem ser classificados também em quatro tipos (BOOCH, 2006): estruturais, comportamentais, de agrupamento e anotacionais. Os itens estruturais são a parte mais estática do modelo e representam elementos conceituais ou físicos.

Dentre os itens estruturais estão as classes, representadas por retângulos divididos em três partes: nome, atributos e operações; interfaces, representadas por um círculo quando é fornecida ao mundo externo, e indica uma especificação de operações, mas não suas implementações; e casos de uso, representados por elipse, com seu nome geralmente dentro deste elemento, representando uma funcionalidade do sistema (BOOCH, 2006).

Nos itens comportamentais estão três elementos: interações, máquinas de estados, e atividades. Os três são comportamentos do sistema, o primeiro abrangendo as mensagens trocadas entre os objetos num contexto, o segundo demonstrando a sequência de estados que o objeto percorre como resposta a determinados eventos, e o terceiro indica os passos que um processo realiza (BOOCH, 2006).

Os itens de agrupamento são itens que ajudam a organizar o projeto, existindo apenas um tipo principal de item de agrupamento chamado de *pacote.* Esses itens podem agrupar itens estruturais, comportamentais e até mesmo de agrupamento para que o projeto seja organizado. Por fim, os itens anotacionais são explicações ou comentários no modelo que podem descrever ou esclarecer qualquer elemento do modelo (BOOCH, 2006).

Para o presente trabalho, optou-se por utilizar os Diagramas de Casos de Uso e o Diagrama de Classe. O primeiro para compreender o escopo do projeto, quais as principais funcionalidades compõem o sistema e como os usuários irão interagir com cada funcionalidade. O segundo para definir o modelo a ser seguido durante a implementação do sistema.

As subseções 2.3.1.1 e 2.3.1.2 detalham ambos os digramas utilizados no desenvolvimento deste trabalho.

2.3.1.1 Diagrama de Caso de Uso

Ao projetar um *software*, a sua estrutura deve estar adequada às formas que este será usado e, para isso, são projetados os diagramas de Caso de Uso pois estes permitem fazer uma análise do uso do sistema e planejar a estrutura mais adequada para o *software* a ser desenvolvido (BOOCH, 2006).

Um diagrama de Caso de Uso bem estruturado deve somente explicar o essencial de um sistema ou subsistema. É demonstrado por meio deste diagrama a interação de elementos externos sobre o sistema que está sendo projetado, tendo cada Caso de Uso representando um requisito funcional (BOOCH, 2006).

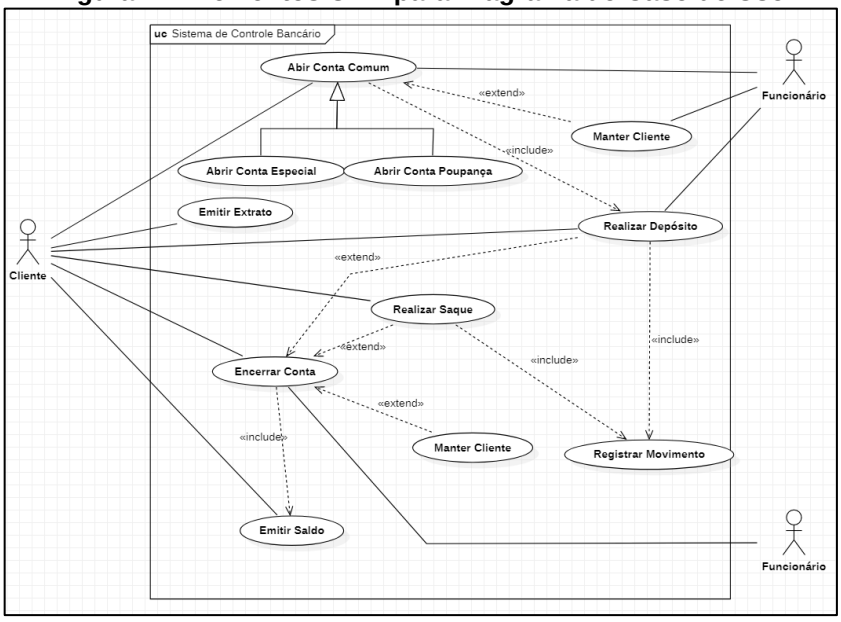
A Figura 4 ilustra um exemplo de um diagrama de caso de uso para um sistema de controle bancário. Pode-se citar os seguintes elementos (GUEDES, 2011):

- Ator: representa um elemento externo que interage com o sistema, podendo ser um usuário do sistema ou um sistema externo. No exemplo há dois atores: Cliente e Funcionário;
- Casos de uso: representa uma sequência de ações do sistema que geram um resultado observável para o ator. No exemplo há ao todo onze casos de uso: Abrir Conta Comum, Abrir Conta Especial, Abrir Conta Poupança, Manter Cliente, Emitir Extrato, Realizar Depósito, Encerrar Conta, Realizar Saque, Manter Cliente, Registrar Movimento e Emitir Extrato.
- Associações: representam as interações ou relacionamentos entre os atores e os casos de uso, ou entre casos de uso e outros casos de uso. As associações podem ser de três tipos:
    - Inclusão: indica a obrigatoriedade de um caso de uso executar um outro caso de uso. A associação de inclusão é identificada pelo estereótipo

<<*include*>> associada a ela. No exemplo, o caso de uso “Realizar Saque” obrigatoriamente irá executar o caso de uso “Registrar Movimento”;

- Extensão: representam eventos opcionais, ou seja, um determinado caso de uso pode ou não executar um outro caso de uso, dependendo de algumas condições. A associação de inclusão é identificada pelo estereótipo <<*extend*>> associada a ela. No exemplo, o caso de uso “Encerrar Conta” pode ou não executar o caso de uso “Realizar Saque”;
- Generalização: associação entre dois casos de uso ou dois atores que possuem características similares, sendo que os casos de uso / atores especializados herdam características e associações de inclusão ou extensão que o caso de uso / ator geral possui. No exemplo, o caso de uso “Abrir Conta Especial” e “Abrir Conta Poupança” especializam o caso de uso “Abrir Conta Comum”.

Figura 4 – Elementos UML para Diagrama de Caso de Uso



Fonte: Adapatado de GUEDES (2011, p. 31).

Para fazer a modelagem de um Diagrama de Caso de Uso é necessário primeiramente identificar os atores que interagem com o elemento a ser modelado e organizá-los com generalizações ou especializações dos atores. Em seguida, deve-se verificar as principais formas que estes atores interagem, indireta ou diretamente com o sistema, e suas alterações de estado. E então, finalmente dispor as

funcionalidades como casos de usos, indicando seus relacionamentos com outros casos de uso e/ou atores (BOOCH, 2006).

2.3.1.2 Diagrama de Classe

Os Diagramas de Classes modelam uma visão estática do projeto do sistema e são importantes para a construção de sistemas executáveis. Este diagrama contém classes, interfaces, colaborações e seus relacionamentos; podendo contribuir para a modelagem de outros diagramas, como o esquema lógico de um banco de dados (BOOCH, 2006).

Cada classe é identificada pelo seu nome, atributos e métodos. Os atributos são responsáveis por armazenar os estados dos objetos, por outro lado, os métodos definem as ações que uma instância pode executar (GUEDES, 2011).

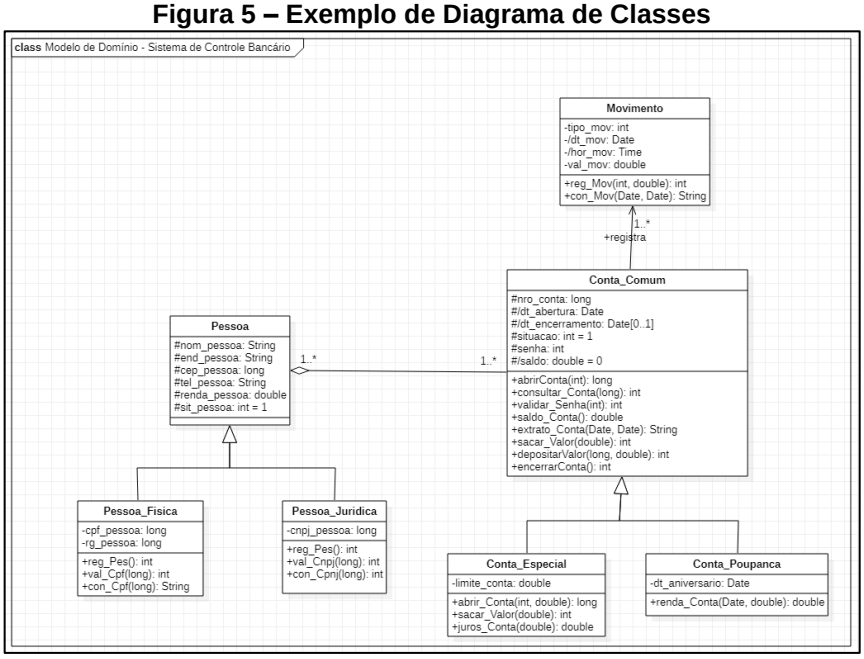
Para que o sistema funcione adequadamente, as classes possuem associações entre si, para que efetuem a comunicação e troca de informações entre os objetos. Os tipos de associações são (GUEDES, 2011):

- Unária/Reflexiva: ocorre quando há um relacionamento de um objeto com outros objetos da mesma classe;
- Binária: ocorre quando há um relacionamento entre objetos de classes diferentes;
- Agregação: ocorre quando há um relacionamento entre dois objetos de classes distintas, porém, este relacionamento demonstra que as informações de um objeto precisam ser complementadas pelas informações contidas no outro objeto. Ou seja, evidencia a relação todo/parte entre os objetos associados.
- Composição: ocorre quando há relacionamento entre dois objetos de classes distintas, porém, na composição, o vínculo é considerado forte entre os objetos-todo e objetos parte, em que ambos não podem coexistir separadamente;
- Generalização/Especialização: representa a ocorrência de herança entre duas ou mais classes. A classe especialista, também conhecida como subclasse ou classe filha, herda as características e comportamentos da classe genérica, também conhecida como superclasse ou classe mãe.

Além das associações, há também questões como navegabilidade e multiplicidade. A primeira diz respeito ao sentido que as informações estão sendo transmitidas entre os objetos e é identificada por uma seta em uma das extremidades da associação. A segunda, especifica quantas instâncias de uma classe podem estar associadas a cada objeto de outra classe, e é identificado por um valor numérico relacionado a associação em questão (GUEDES, 2011).

A Figura 5 ilustra um Diagrama de Classes para um sistema de controle bancário.

Figura 5 – Exemplo de Diagrama de Classes



Fonte: Adaptado de GUEDES (2011, p. 32).

No exemplo apresentado na Figura 5, há uma relação de generalização/especialização entre as classes “Pessoa_Fisica” e “Pessoa_Juridica” para a classe “Pessoa”. Neste caso, as características e comportamentos da classe “Pessoa” são herdadas para as demais classes citadas. A associação de generalização/especialização também ocorre entre as classes “Conta_Comum”, “Conta_Especial” e “Conta_Poupanca”, sendo que as duas últimas herdam as informações de “Conta_Comum”.

A classe “Pessoa” possui uma associação de agregação com “Conta_Comum” e uma multiplicidade de 1 ou N, ou seja, uma pessoa pode possuir uma ou várias contas neste sistema bancário. Por fim, a classe “Conta_Comum” possui uma associação binária de multiplicidade 1 ou N com a classe “Movimento”.

Neste caso, cada conta registra um ou vários movimentos que ocorrem na conta, como por exemplo, saques, depósitos, entre outros.

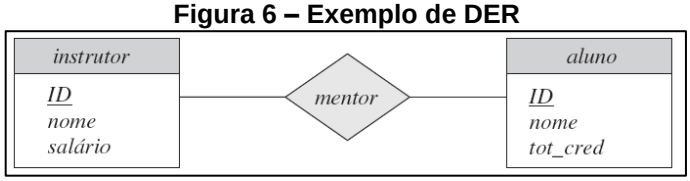
2.3.2 Diagrama Entidade-Relacionamento (DER)

A abordagem Entidade-Relacionamento foi criada para facilitar o projeto de banco de dados. O conceito fundamental nesta abordagem é o conceito de Entidade, que representa um objeto do mundo real que pode ser distinguido de outros objetos. Cada Entidade pode conter um conjunto de atributos que o definem, e um ou mais desses atributos que serão únicos, sendo chamado de “atributo identificador”. Além do conceito de entidades, outro conceito que define essa abordagem são os relacionamentos, sendo estes relacionamentos as associações entre as entidades (SILBERSCHATZ, 2020).

No Diagrama Entidade-Relacionamento (DER) as entidades são representadas por um retângulo com o nome da entidade representada, e os relacionamentos entre essas entidades são apresentadas com um losango ligado por linhas às entidades que têm um relacionamento. É possível identificar nos relacionamentos a sua cardinalidade, que expressa o número de entidades ao qual uma entidade pode se associar por um conjunto de relacionamentos (SILBERSCHATZ, 2020).

A Figura 6 ilustra duas entidades: “instrutor” e “aluno”, o “instrutor” tem como atributos “nome” e “salário”, e atributo identificador “ID”; o “aluno” tem os atributos “nome” e “tot_cred” e atributo identificador “ID”. O losango representa a relação entre as entidades que estão ligadas. Esse relacionamento indica que o “instrutor” tem a associação de “mentor” de “aluno”.

Figura 6 – Exemplo de DER



Fonte: SILBERSCHATZ (2020, p. 135).

2.4 Ferramentas e Frameworks

Para o desenvolvimento de um *software*, o desenvolvedor deve priorizar a reutilização, facilidade de manutenção e extensão. Para isso, faz-se necessário a utilização de padrões de projeto, pois os mesmos diminuem e limitam dependências de plataformas e de outras camadas do sistema (GAMMA *et al*., 2011).

Atualmente há diversos frameworks que auxiliam o desenvolvedor a aplicar as boas práticas de desenvolvimento, bem como o uso de padrões de projetos, facilitando a construção de uma aplicação de forma rápida e consistente (GAMMA *et al*., 2011). Para o desenvolvimento deste trabalho, utilizou-se algumas ferramentas e frameworks que auxiliaram em todas as etapas do projeto. As ferramentas e frameworks foram separadas em três categorias de acordo com qual etapa foi usada: (i) projeto de *software*; (ii) desenvolvimento do *software*; (iii) e implantação do *software*

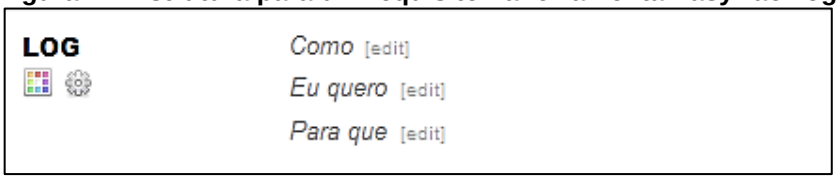
2.4.1 Ferramentas para o projeto do software

Com o intuito de padronizar a escrita dos requisitos levantados na etapa de Engenharia de Requisitos de forma estruturada, utilizou a plataforma EasyBacklog[4] a qual permite ser configurada para gerenciar as demandas e requisitos para o desenvolvimento de *software*.

A Figura 7 apresenta a estrutura que é usada para a escrita de uma demanda na ferramenta. O card é identificado por um código, que tem relação com o tema do cartão, identificado na figura como “LOG”; o campo “*Como*” deve conter quem é o ator/atores que deverão ter acesso a essa funcionalidade; em “*Eu quero*” é escrito a funcionalidade que este requisito representa; e “*Para que*” é a motivação da implementação deste requisito.

[4] Website da ferramenta EasyBacklog: https://easybacklog.com/

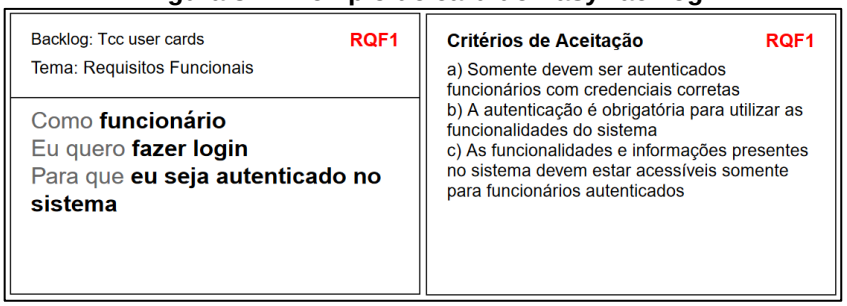
Figura 7 – Estrutura para um requisito na ferramenta EasyBacklog



Fonte: EasyBacklog (2022).

Além da estrutura apresentada, é possível adicionar critérios de aceite e comentários sobre o cartão escrito. Ao final, a ferramenta permite gerar um PDF com todos os requisitos escritos, dispostos em cards semelhantes ao mostrado na Figura 8.

Figura 8 – Exemplo de card do EasyBacklog



Fonte: Autoria própria (2022).

Este exemplo apresenta um requisito do sistema, onde quem tem a permissão de executar essa funcionalidade é a assistente social. Esta deve conseguir gerar um documento que representa o cadastro de um usuário na Casa do Piá, com o propósito de enviar essa documentação para o CMAS. Os critérios de aceitação são pontos que devem ser verificados para que a implementação desse requisito possa ser considerada correta, no caso, o preenchimento com os dados corretos do beneficiário que foi gerada a documentação de cadastro.

Para a escrita da documentação do sistema, uma vez que o mesmo será entregue a instituição filantrópica e futuramente ocorrerão atualizações no sistema, utilizou-se a ferramenta StarUml[5], a qual permite modelar diagramas utilizando os padrões especificados na UML, além de ser possível utilizar, na própria ferramenta, os componentes gráficos, que são dispostos de acordo com o diagrama modelado (MKLABS, 2021).

[5] Website da ferramenta StarUml: https://staruml.io

2.4.2 Ferramentas para desenvolvimento do software

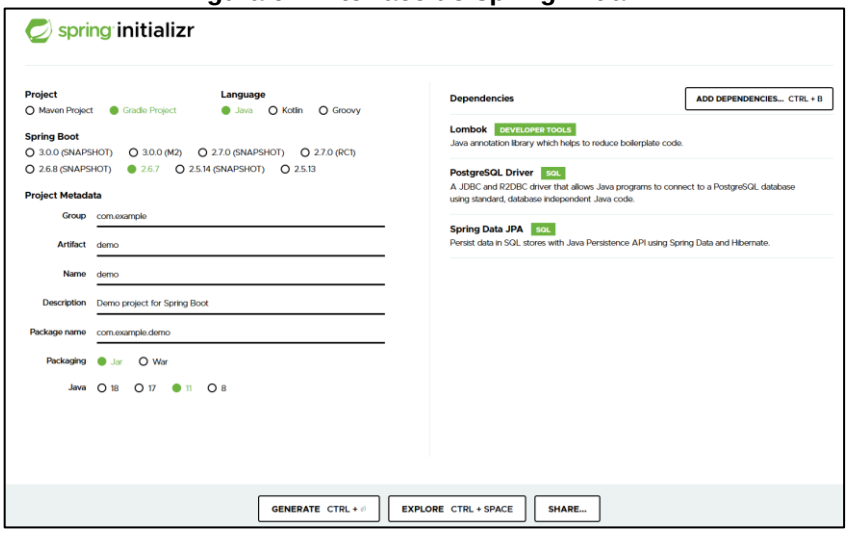
Na etapa de implementação do sistema, utilizou-se o framework *Spring*[6] o qual visa diminuir o tempo gasto com a configuração do ambiente, possibilitando ao programador focar na regra de negócio e implementação. O *Spring* também implementa no projeto a injeção de dependências, fazendo com que a aplicação possa ter baixo acoplamento de uma classe, ou seja, menor dependência de uma classe para o programa funcionar (JUNIOR; AFONSO, 2017).

Além disso, pode se ressaltar que o *Spring Boot*, um dos projetos do *Spring* "analisa o projeto e automaticamente o configura" (JUNIOR; AFONSO, 2017. p. 19). O uso do *Maven*[7] ou do *Gradle*[8] possibilita a inserção de uma dependência no projeto e então, o *Spring Boot* irá configurar essas dependências automaticamente. Com o uso do *Spring*, as classes principais, chamadas de "*beans*", são carregadas pelo gerenciamento de inversão de controle do projeto. Estas são instanciadas, acopladas e gerenciadas pelo *Spring*, tendo elas e suas dependências refletidas na configuração e inicialização do projeto (VMWARE, 2022).

Utilizou-se o *website spring initializr*[9], uma API que facilita a configuração do *Spring Boot* em projetos baseados na *JVM*. Este *website* contém uma interface amigável ao usuário, no qual é possível escolher o tipo de projeto entre *Maven* e *Gradle*, escolher a linguagem usada, a versão do *Spring Boot* que será usada no projeto, adicionar algumas dependências do programa, além de definir alguns metadados que definirão a estrutura do projeto. A interface do *spring initializr* é apresentada na Figura 9 com os metadados preenchidos com o exemplo dado pela ferramenta.

[6] Site Spring: https://spring.io/
[7] Site Apache Maven: https://maven.apache.org/
[8] Site Gradle: https://gradle.org/
[9] Site Spring Initializr: https://start.spring.io/

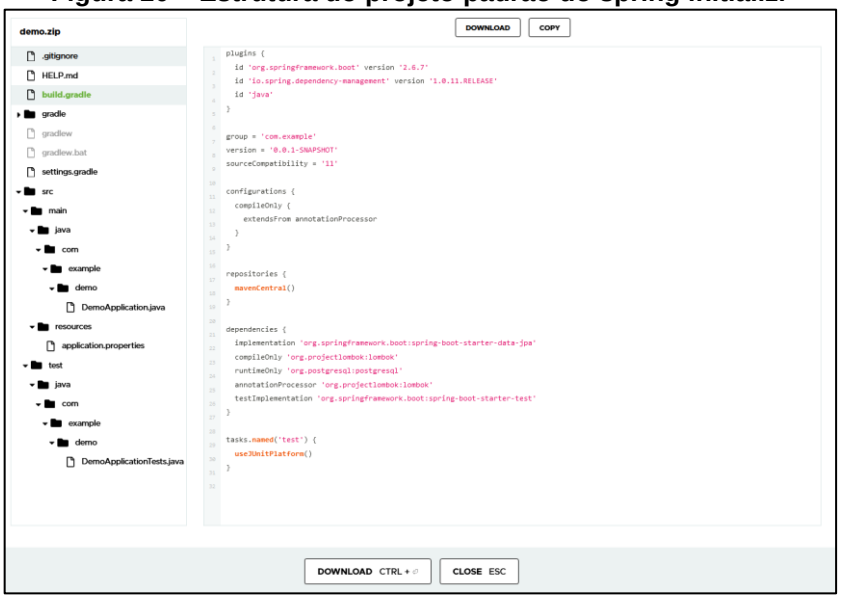
Figura 9 – Interface do spring initializr



Fonte: VMware (2022)

Observa-se na Figura 9 que a opção do tipo de projeto está configurada como *Gradle*, a linguagem em Java, a versão do *Spring Boot* na 2.6.7, e algumas dependências como o *Lombok*, o driver do *PostgreSQL* e o *Spring Data* JPA, os metadados estão preenchidos com os valores padrões da ferramenta, o tipo de empacotamento encontra-se em *Jar* e a versão 11 do Java. Com estas configurações concluídas é possível gerar ou explorar a estrutura, podendo também compartilhar as configurações deste projeto.

Figura 10 – Estrutura do projeto padrão do spring initializr



Fonte: VMware (2022)

Ao lado esquerdo da Figura 10 é exibida a estrutura do projeto. O padrão seguido pela ferramenta é que as pastas que contém códigos devem ficar dentro do

caminho de pastas “*src.main.java*”, seguido de algumas informações preenchidas nos campos de metadados observado na Figura 9. Assim, o restante do caminho de pastas para o código deve ser preenchido com o valor de “*Package name*”, que será a concatenação do caminho de “*Group*” com do “*Artifact*”. Além das pastas que contém códigos, há também: (i) a pasta “*src.resources*” que deve conter recursos utilizados no projeto, como imagens ou até mesmo arquivos de configurações do programa, como o “*application*.*properties*”; e (ii) a pasta “test” que pode conter testes para o sistema ou classes do projeto.

Ao lado direito da Figura 10 encontra-se aberto o arquivo “*build.gradle*” gerado. Este é um arquivo que irá conter informações para compilação e execução do sistema, além de ser onde as dependências são especificadas para que o *Gradle* possa baixa-las e aplicar ao projeto.

Para a persistência de dados, fez-se uso do framework *Hibernate*[10], que é uma implementação do Java *Persistence* API (JPA). Este fornece um mapeamento objeto-relacional para classes Java. Os atributos das classes mapeadas pelo *Hibernate* podem ser mapeados para colunas no banco de dados assim como os relacionamentos entre as classes podem ser mapeadas para os relacionamentos entre as tabelas (SILBERSCHATZ, 2020).

Juntamente com a utilização do Hibernate, foi necessário a escolha de um banco de dados em nuvem. De acordo com a Oracle (2022), este serviço oferece maior agilidade e inovação além de custos e riscos reduzidos, pois podem ser configurados de maneira rápida e fácil. Não há a necessidade de pedir hardware para a hospedagem do banco de dados, estes podem ser configurados e disponibilizados em alguns minutos e a empresa que provém o serviço pode aplicar e atualizar medidas de segurança com menor indisponibilidade para o cliente. Esta tecnologia foi utilizada também para que o cliente possa acessar o *software* em qualquer desktop sem a configuração de um banco de dados local, além de disponibilizar todos os registros do banco de dados para todos usuários do *software* em tempo real.

A primeira opção encontrada foi o “*Amazon DynamoDB*[11]”, um banco de dados não relacional, NoSQL disponibilizado pela Amazon, com 25 GB de armazenamento gratuito. Apesar de essa ser uma opção atrativa, optou-se pela busca

[10] Site Hibernate: https://hibernate.org/
[11] Website Amazon DynamoDB: https://aws.amazon.com/pt/dynamodb

de um banco de dados relacional, com o intuito de se beneficiar das propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade) na transação desta forma de modelagem dos dados. Encontrou-se então o Heroku.

Utilizou-se também na implementação da aplicação o framework Lombok, pois este fornece um conjunto de anotações que elimina uma quantidade considerável de código repetido das classes Java, tornando-as mais limpas, simples e fáceis de manter. O Lombok ajuda também a aplicar padrões de projeto no código, como o *Builder*, com a anotação @*Builder*. É possível até mesmo lidar com métodos *multithread*, utilizando a anotação @*Synchronized* acima do método que teria a palavra reservada *synchronized* no método (LOMBOK, 2021).

É algo comum em um projeto Java conter muitas linhas de código definindo dados simples em uma classe, assim como métodos de acesso a esses atributos. As anotações disponíveis no Lombok são utilizadas para gerar o código de métodos de comportamento padrão, como "*getters*" e "*setters*" (LOMBOK, 2021).

2.4.3 Ferramentas para implantação do software

Com o intuito de facilitar o processo de instalação da aplicação no ambiente real de utilização, há a necessidade de criar um instalador para o *software*. Neste contexto, utilizou-se *Inno Setup*[12], que é uma ferramenta gratuita, o qual auxilia na criação de instaladores de programas para o Sistema Operacional *Windows*. A configuração dos instaladores é realizada por meio de um *script* que será salvo com a extensão “.iss” (*inno setup script*), no qual é possível controlar todos os aspectos da instalação: definir quais arquivos devem ser instalados, atalhos a serem criados, ícones para os atalhos, entre outros. (RUSSEL, 2021).

[12] Website da ferramenta Inno Setup: https://jrsoftware.org/isinfo.php

3 DESENVOLVIMENTO

Este capítulo aborda as etapas de desenvolvimento da aplicação. Os modelos e diagramas evoluíram ao longo das versões de acordo com os requisitos que seriam implementados, assim como o *software*. Cada versão é um novo ciclo do modelo espiral de desenvolvimento de *software*, sendo cada subseção uma etapa da espiral. As subseções passam pelas etapas de: (i) Comunicação, onde há reuniões com o cliente, elicitação e detalhamento de requisitos, incluindo também a etapa de planejamento, onde são definidas as funcionalidades que o restante do ciclo implementará. (ii) Projeto de *software*, nesta etapa há a modelagem dos diagramas necessários para a implementação do sistema. (iii) Desenvolvimento, quando o *software* é implementado de fato e ao fim desta etapa é entregue ao cliente. (iv) *Feedback* e conclusões, nesta etapa o cliente teve contato com a nova versão e tempo para testá-la e apontar melhorias para a próxima versão, além de nesse momento já conter algumas melhorias na forma de desenvolvimento percebidas durante o desenvolvimento.

3.1 Configuração, Login e Cadastro – Versão 1

A primeira etapa para o desenvolvimento deste trabalho foi entrar em contato com os funcionários da Casa do Piá e realizar as atividades descritas na Engenharia de Requisitos. Com os requisitos levantados e especificados, desenvolveu-se um protótipo para a validação dos requisitos que, posteriormente, foi útil para iniciar o projeto do *software* e finalmente o desenvolvimento de sua primeira versão. A Subseção 3.1.1 aborda a comunicação com os funcionários da instituição. A Subseção 3.1.2 apresenta o projeto de *software* da primeira versão do sistema. A Subseção 3.1.3 descreve as configurações necessárias a fim de executar a aplicação. Por fim, a Subseção 3.1.4 discorre sobre a entrega da primeira versão, bem como o *feedback* obtido do usuário.

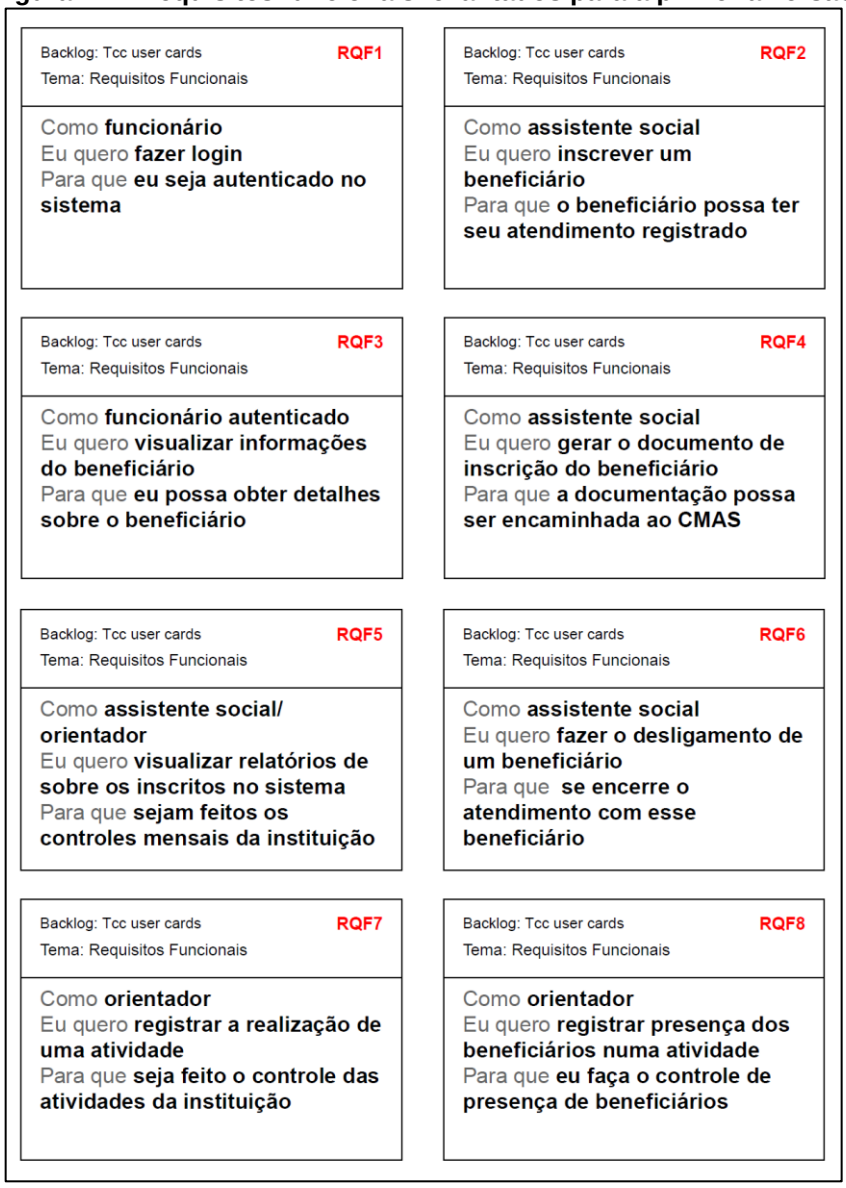
3.1.1 Comunicação

O início da etapa de comunicação se deu com o levantamento do escopo do sistema desenvolvido. Para isso, a primeira reunião ocorreu em forma de uma entrevista aberta, na qual elicitou-se as funcionalidades que o cliente deseja no *software*, e quem tem acesso à estas.

Realizou-se a primeira reunião com os funcionários da Casa do Piá indicados como os principais utilizadores das funcionalidades do sistema. As necessidades levantadas pela assistente social e pelo orientador da instituição são: um sistema que controle o acesso aos dados utilizados nas atividades da instituição; realize a inscrição, disponibilize e produza a documentação com os dados dos beneficiários da Casa do Piá inscritos; elabore relatórios a partir dos cadastros realizados; registre o desligamento de inscritos; e controle as atividades realizadas e registre a presença dos beneficiários.

Destas necessidades, definiu-se para todos os funcionários a autenticação, ou login, provendo a camada de proteção aos dados contidos no *software*, e a visualização dos dados a eles permitidos. Para a assistente social as funcionalidades disponíveis são: o cadastro de beneficiários, gerar os documentos como resultado da inscrição e relatórios sobre os atendidos. Enquanto o orientador deve registrar no software a realização de atividades assim como a presença dos beneficiários participantes, além do acesso aos relatórios sobre os inscritos.

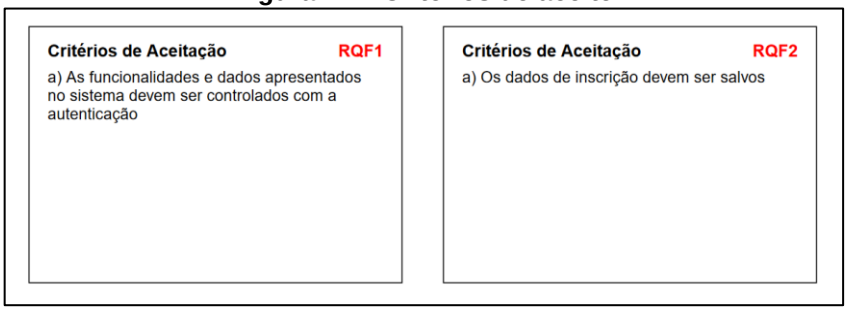
Figura 11 – Requisitos funcionais levantados para a primeira versão



Fonte: Autoria própria (2022).

As necessidades do cliente foram escritas como requisitos do sistema na ferramenta easyBacklog, definindo o seu escopo. Ao longo do desenvolvimento das versões, mais detalhes dos requisitos foram coletados e escritos, complementando-os com critérios de aceite, comentários e diagramas que melhor os definem. A Figura 11 apresenta os requisitos funcionais listados em cards gerados pela ferramenta usada. Para os requisitos funcionais utilizou-se o código RQF juntamente com o número de identificação para este requisito.

Figura 12 – Criterios de aceite

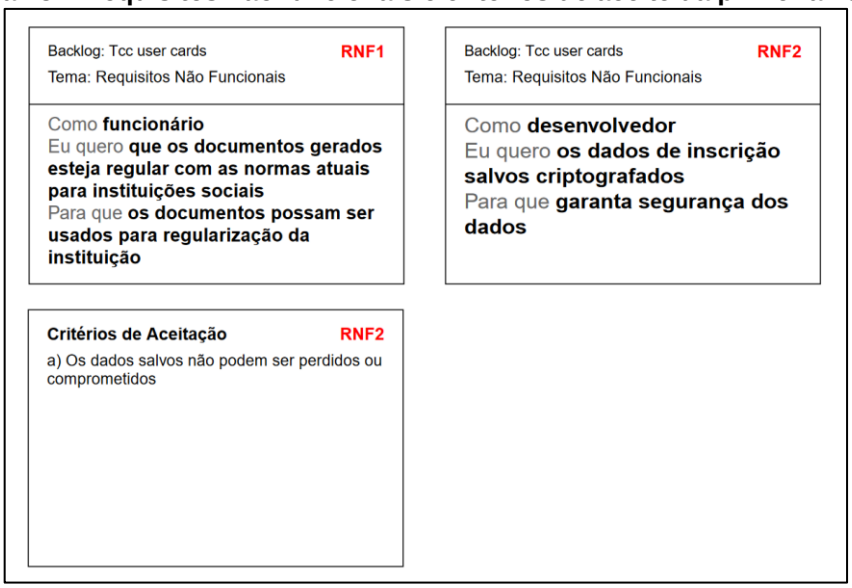


Fonte: Autoria própria (2022)

Em complemento aos requisitos funcionais levantados, critérios de aceite são apresentados na Figura 12. Estes ajudaram a detalhar características das funcionalidades que devem ser implementadas para que sejam consideradas corretas e a documentar os requisitos de forma mais completa. Escreveu-se os critérios de aceite apenas para aqueles detalhados na última comunicação com o cliente, registrando de forma resumida aquilo que foi descrito.

Além dos requisitos funcionais, foi possível extrair das necessidades do cliente alguns requisitos não funcionais, exibidos na Figura 13. Para estes cards, utilizou-se o código RNF juntamente com o número de identificação do requisito. Os requisitos não funcionais escritos puderam ser levantados a partir do escopo do sistema. Estes requisitos foram escritos para garantir a proteção dos dados utilizando a criptografia dos mesmos durante a armazenagem, além de se certificar que a documentação gerada pela aplicação será útil para a Casa do Piá.

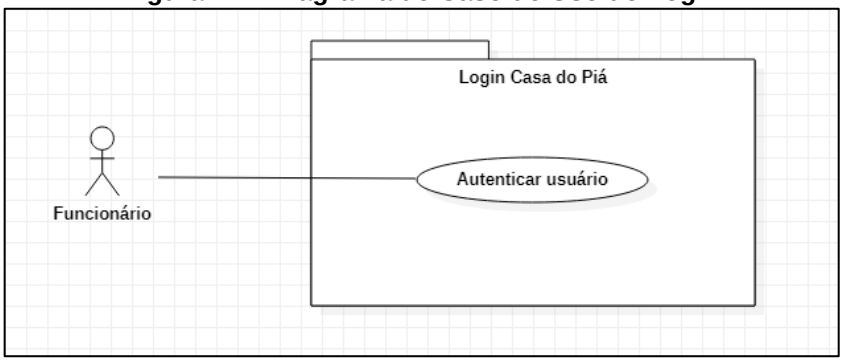
Figura 13 – Requisitos não funcionais e critérios de aceite da primeira versão



Fonte: Autoria própria (2022).

Após a definição do escopo de requisitos do sistema, as necessidades foram ordenadas em função do maior benefício para o cliente, resultando na seguinte ordem de prioridade: (i) login, gerando a camada de proteção e privacidade desejada pelo cliente; (ii) inscrição de beneficiários, pois as principais finalidades do sistema são voltadas para estes; (iii) visualização das informações de beneficiários, para que os dados daqueles inscritos sejam utilizados durante os atendimentos e atividades realizadas; (iv) geração de documento de inscrição, devido a necessidade deste documento após o cadastro; (v) desligamento de beneficiários, concluindo as ações de manipulação dos inscritos; (vi) registro de realização de atividades; (vii) presença de usuários; e (viii) a geração de relatórios sobre os beneficiários.

Figura 14 – Diagrama de Caso de Uso do Login



Fonte: Autoria própria (2022).

Modelou-se o Diagrama de Caso de Uso para documentar os fluxos possíveis para cada usuário do sistema. Primeiramente, o funcionário tem uma interação com a tela de login, onde ele pode somente autenticar-se. Este caso de uso, apresentado na Figura 14 foi desenhado separadamente pois somente há interação com outras funcionalidades do sistema após o sucesso nesse fluxo, sendo pré-requisito para os fluxos apresentados posteriormente.

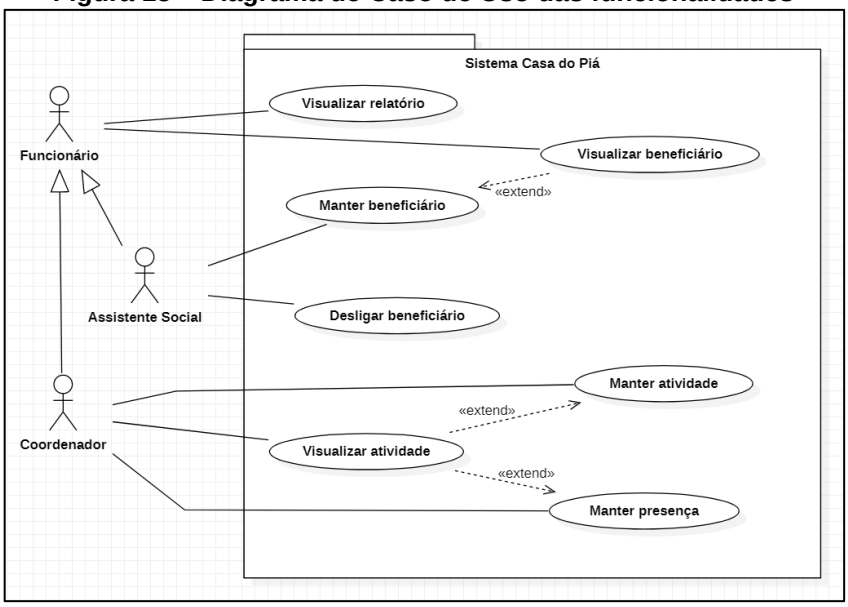
Em seguida, modelou-se na Figura 15 os demais requisitos disponíveis após o login do *Funcionário*. O diagrama contém como principal ator o *Funcionário*, que interage com os casos de uso diretamente ou através de suas especializações: “*Assistente Social*”, e “*Coordenador*”. O *Funcionário*, após a autenticação pode visualizar os relatórios de inscrição ou visualizar os dados dos beneficiários inscritos.

Caso o funcionário seja um “*Assistente Social*”, além das funcionalidades herdadas do *Funcionário* é possível, no caso de uso “*Manter beneficiário*”, fazer a inscrição de um novo beneficiário, ou editar os dados dos inscritos partindo do fluxo

de “*Visualizar beneficiário*”. Este tipo de ator pode também acessar a funcionalidade de “*Desligar beneficiário*”.

O “*Coordenador*” tem acesso aos casos de uso “*Manter atividade*”, “*Visualizar atividade*” e “*Manter presença*”, onde ele pode registrar atividades que foram executadas, visualizar as atividades registradas, podendo editá-las ou preencher a presença dos beneficiários nessas atividades.

Figura 15 – Diagrama de Caso de Uso das funcionalidades



Fonte: Autoria própria (2022).

Com os Diagramas de Caso de Uso modelados, os dois primeiros requisitos priorizados foram selecionados para a implementação na primeira versão. Necessitou-se de uma nova reunião com a Casa do Piá, onde o fluxo modelado nos casos de uso foi validado. Os fluxos apresentados no diagrama eram os esperados pelo cliente e então, foi possível prosseguir para o detalhamento dos requisitos escolhidos para a implementação nesta versão.

Para a funcionalidade de login foi necessário entender se havia preferência pelo uso de algum tipo de nome de usuário ou e-mail para a realização do login, além da senha. Escolheu-se usar um nome de usuário e senha do funcionário para a autenticação.

Para a inscrição de beneficiários, perguntou-se quais os dados dos beneficiários que devem ser coletados e salvos nesse processo. Esta questão foi respondida com a entrega do documento digital que eles preenchem ao final do processo de cadastro, ou seja, na etapa de transcrição para o meio digital. Este

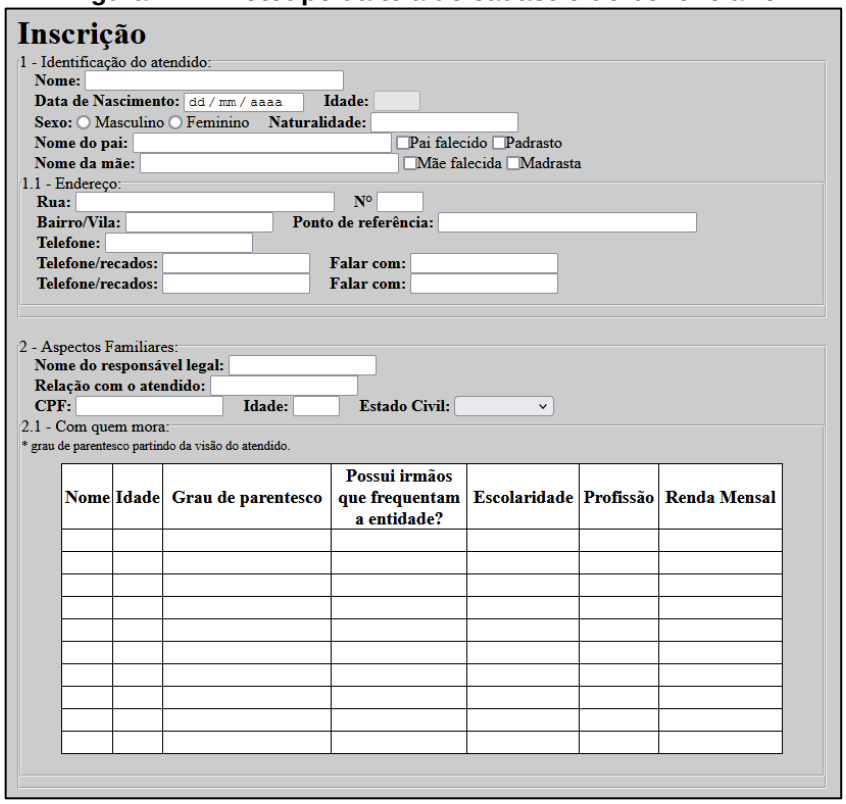
modelo está disponível no Anexo A e foi definido como template para o requisito *RQF4*, de geração do documento de inscrição.

Figura 16 – Protótipo da tela de login

Fonte: Autoria própria (2022).

Para a validação das funcionalidades escolhidas, criou-se protótipos descartáveis dos requisitos selecionados, pois estes seriam mais rápidos de se desenvolver, e é possível utilizar linguagens ou ferramentas que agilize esse processo, pois o protótipo não será integrado ao sistema final. As telas criadas foram implementadas com HTML e CSS, apenas para validar os campos e formatos que estes deveriam ser preenchidos. A Figura 16 apresenta o protótipo da tela de login dos funcionários, onde deve ser preenchido o usuário e a senha pré-definidos de acordo com a função.

Figura 17 – Protótipo da tela de cadastro de beneficiário



Fonte: Autoria própria (2022).

Por ser muito extenso o formulário de inscrição, dividiu-o em duas imagens. As Figuras Figura 17 e Figura 18 apresentam o protótipo utilizado para validar quais campos e seus respectivos formatos seriam necessários no formulário. A Figura 17 apresenta a primeira tela de inscrição do beneficiário. Nesta tela o funcionário deve preencher os dados de identificação, endereço e aspectos familiares do beneficiário.

Figura 18 – Continuação protótipo da tela de cadastro de beneficiário

Fonte: Autoria própria (2022)

A Figura 18 apresenta a continuação da Figura 17 que é apresentado com a rolagem da tela do protótipo. Nesta tela são apresentadas as seções de dados da habitação, saúde e escolaridade do beneficiário, além de dados referentes a assistência social e o ingresso do mesmo na Casa do Piá.

Ao verificar os protótipos o cliente aprovou a tela de login e o formato dos campos de inscrição, apresentando algumas melhorias e alguns novos campos desejados. Na tela de inscrição, o cliente pediu que fosse adicionado um campo de texto ao lado de cada uma das opções na seção de saúde, para ser preenchido caso seja marcado “Sim”, além de um campo de texto para serem adicionadas observações para a escolaridade.

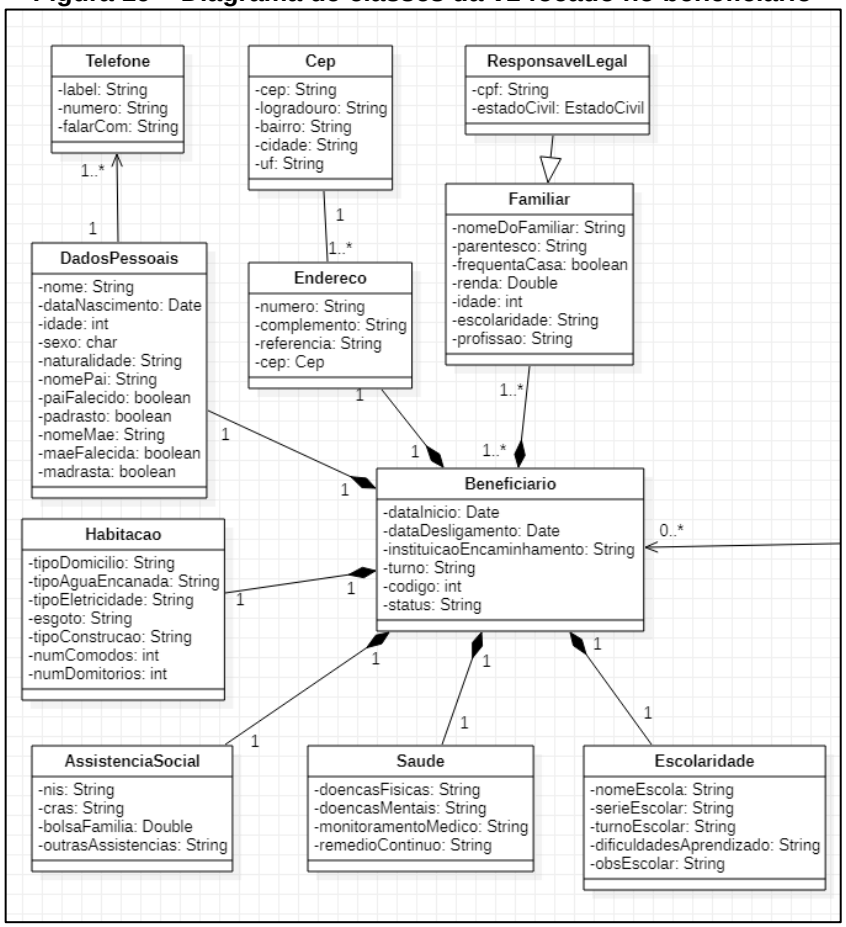
3.1.2 Projeto do Software

O projeto de software ocorreu com o fim da elicitação e validação dos requisitos na etapa de comunicação. Nesta etapa foram modelados os diagramas de

classe, para representar o domínio do sistema desenvolvido, e o diagrama Entidade-Relacionamento, representando o esquema do banco de dados. Ambos os modelos evoluíram ao longo das versões do software e representam as funcionalidades implementadas na versão atual.

O diagrama de classes nesta versão foi apresentado separando-o em duas figuras, para melhor abstração e organização do problema apresentado. A Figura 19 foca no domínio do beneficiário e seus dados, tendo a classe “Beneficiario” ao centro, e seus dados separados em classes que os agrupam, categorizando-os. Enquanto a Figura 20 é focada no funcionário e sua relação com o “Beneficiario”.

Figura 19 – Diagrama de classes da v1 focado no beneficiário

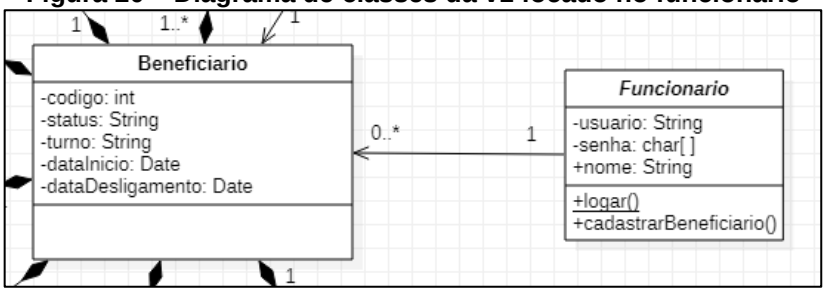


Fonte: Autoria própria (2022).

Criou-se a classe “*Beneficiario*”, onde ficam os dados de cadastro do inscrito. Os dados do beneficiário foram agrupados dentro das classes “DadosPessoais”, “Endereco”, “Familiar”, Saude”, “Escolaridade”, “Habitacao” e “AssistenciaSocial”, criando uma categorização para os dados. As classes de dados do “Beneficiario” são uma composição deste, pois só fazem sentido existirem com uma instância desta

classe. Os dados foram categorizados para melhor organização e controle de acesso a esses dados do beneficiário, possibilitando permissões individuais para cada categoria.

Figura 20 – Diagrama de classes da v1 focado no funcionário

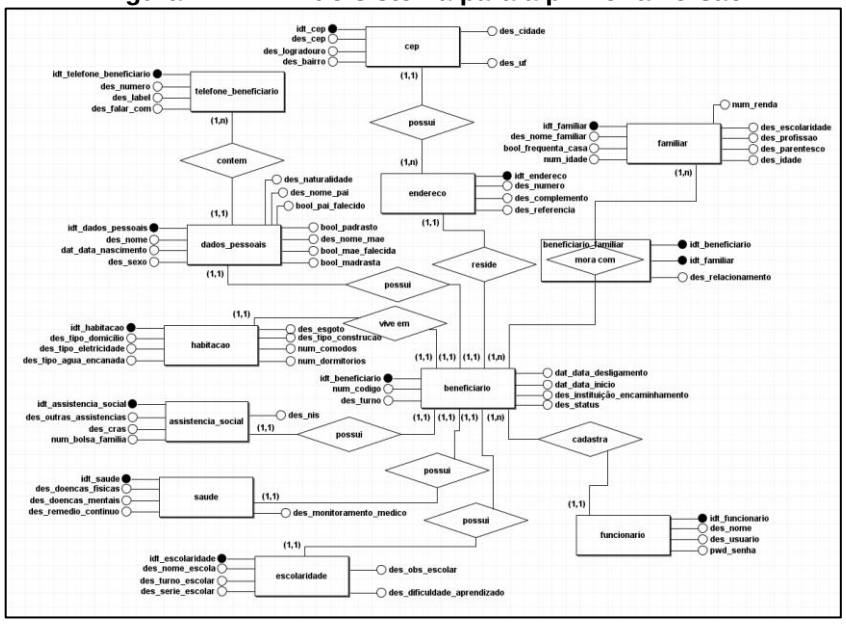


Fonte: Autoria própria (2022).

A classe “Funcionario” foi criada para representar o usuário da Casa do Piá que tem acesso ao sistema. O método estático “logar” deve retornar a instância de “Funcionario” referente ao usuário e senha utilizados na autenticação e, somente com este objeto disponível o usuário terá acesso às interfaces e funcionalidades permitidas. Foi adicionado também o método “cadastrarBeneficiario” que abstrai a possibilidade de esta classe conseguir utilizar esta funcionalidade do sistema.

Além do diagrama de classe, realizou-se também a modelagem do DER para o planejamento e documentação da estrutura criada no banco de dados para esta versão. A Figura 21 apresenta o Diagrama Entidade Relacionamento, na qual a entidade “*beneficiario*” se relaciona às entidades que compõe os dados deles, como por exemplo a entidade “*dados_pessoais*”, “*saude*” e “*escolaridade*”. A entidade “funcionário” contêm usuário e senha para a autenticação, e seu nome.

Figura 21 – DER do sistema para a primeira versão



Fonte: Autoria própria (2022).

O “*beneficiario*” está relacionado também à entidade associativa “*beneficiario_familiar*”. Nesta entidade fica armazenado o parentesco ou relacionamento desse beneficiário com o familiar. Caso algum familiar deste inscrito já esteja cadastrado na Casa do Piá, os familiares serão reaproveitados através deste relacionamento.

3.1.3 Desenvolvimento da Versão 1

Iniciou-se o desenvolvimento da primeira versão do software com a configuração do ambiente de desenvolvimento, contendo as dependências, conexão ao banco de dados e configurações para a implantação do sistema. Em seguida foram feitas as telas com o javaFX que serviriam para o login e cadastro dos beneficiários, desenvolvidas utilizando o *software* Scene Builder. Então foram criadas as classes do domínio do projeto; e finalmente foram implementadas as funcionalidades e atribuídas às telas desenvolvidas.

3.1.3.1 Subversão de configuração – Configurações do sistema

Nesta seção é apresentada a subversão usada para a configuração dos frameworks, ferramentas e dependências utilizadas no desenvolvimento do sistema.

Utilizou-se a ferramenta “*spring initializr*” com as configurações apresentadas na Figura 9, adaptando-se apenas os metadados. Com o projeto gerado, o Springboot e o Gradle configurados pela ferramenta, outros frameworks foram adicionados como dependência do software a ser desenvolvido. A Figura 22 demonstra todas as seções alteradas do arquivo “*build.gradle*” que foram alteradas para a entrega da versão 1 do sistema.

Figura 22 – Dependências do sistema da Casa do Piá

```
plugins {
    id 'application'
    id 'org.openjfx.javafxplugin' version '0.0.9'
    id 'org.springframework.boot' version '2.5.3'
    id 'io.spring.dependency-management' version '1.0.11.RELEASE'
    id 'java'
}

repositories {
    mavenCentral()
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa:2.7.4'
    implementation 'org.springframework.boot:spring-boot-starter-validation:2.7.4'
    implementation 'org.hibernate:hibernate-validator:8.0.0.Final'
    implementation 'net.rgielen:javafx-weaver-spring-boot-starter:1.3.0'
    implementation group: 'org.openjfx', name: 'javafx', version: '16', ext: 'pom'
    compileOnly 'org.projectlombok:lombok:1.18.24'
    annotationProcessor 'org.projectlombok:lombok:1.18.24'
    implementation 'org.postgresql:postgresql:42.5.0'
}

javafx {
    version = "16"
    modules = [ 'javafx.controls', 'javafx.fxml', 'javafx.base']
}

compileJava.options.encoding = 'UTF-8'

tasks.withType(JavaCompile) {
    options.encoding = 'UTF-8'
}

mainClassName = 'gmacias.AppStarter'
```

Fonte: Autoria própria (2022).

Ao começar o desenvolvimento do projeto, realizou-se a configuração da inicialização do JavaFX com o framework Spring, pois as classes devem ser carregadas como “*beans*” e, por padrão, o JavaFX não é carregado pois não contêm as anotações do Spring que as configure como tal. Para contornar essa situação, adotou-se o *framework JavaFX-weaver*, que cria as anotações necessárias nas classes do JavaFX para que estas sejam carregadas devidamente com o *Spring*. A inserção do *JavaFX-weaver* pode ser verificada na Figura 22 como uma dependência do sistema, na qual foi configurada pelo *Gradle*.

Configurou-se também nesta subversão a conexão com o banco de dados. Primeiramente foi testada a conexão com o banco de dados a partir de arquivo de nome “*application.properties*”, no qual são definidas configurações do projeto. Entre as configurações, encontram-se o *drive* de conexão de banco de dados usado, *host*, usuário e senha para a conexão.

Primeiramente foi utilizado um banco de dados configurado localmente, para testar a conexão com o uso do Hibernate e as configurações neste arquivo, e então foi utilizado o banco de dados criado no Heroku.

Figura 23 – Exemplo do arquivo application.properties

```
spring.datasource.url=jdbc:postgresql://database_host:5432/database_name
spring.datasource.username=database_username
spring.datasource.password=database_password
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

Fonte: Autoria própria (2022)

Um exemplo deste arquivo é apresentado na Figura 23, onde os valores “*database_host*”, “*database_name*”, “*database_username*” e “*database_password*” são configurados com o host, o nome, o usuário e a senha do banco de dados, respectivamente. Há também a porta e o driver utilizado na conexão com este banco de dados, além das configurações do Hibernate: a linguagem que será usada para as *querys* geradas pelo framework; se deve ser apresentado nos *logs* as *querys* feitas no banco de dados, e se o Hibernate deve formatá-las para serem mais legíveis. As duas últimas foram configuradas como *true* durante o desenvolvimento para melhor controle do que está sendo feito no banco de dados, mas foram trocadas para o valor “*false*” ao final do desenvolvimento.

Além disso, nesta subversão utilizou-se da ferramenta Launch4j[13], que transforma um arquivo de extensão “.jar” para um arquivo executável com extensão “.exe”, facilitando a execução da aplicação desenvolvida no Sistema Operacional Windows. Após essa etapa, configurou-se um script na aplicação “*Inno Setup*” a qual gera um instalador, permitindo que a aplicação seja executada pelo usuário com todas as dependências para sua correta implantação.

[13] Website da ferramenta Launch4j: http://launch4j.sourceforge.net/

3.1.3.2 Interfaces de usuário desenvolvidas

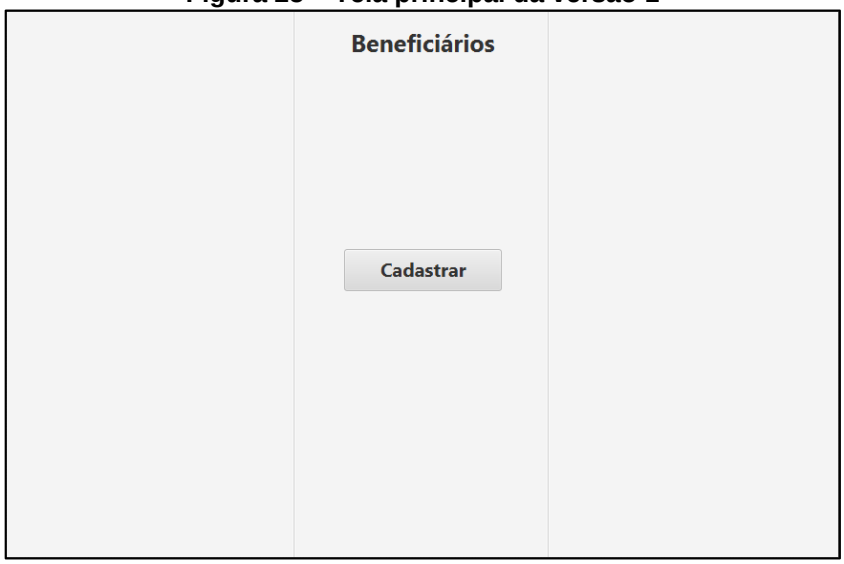
Após a configuração do ambiente de desenvolvimento, as telas foram criadas com o uso do JavaFX. A interface gráfica do sistema se enquadrou nas conversas levantadas durante a prototipação das mesmas, contendo um visual melhorado devido ao uso do JavaFX em seu desenvolvimento.

Figura 24 – Tela de login do sistema

Fonte: Autoria própria (2022).

A Figura 24 ilustra a tela de login do sistema. Nesta tela, a pessoa que está utilizando o sistema deve informar o seu usuário e senha, a fim de validar suas credenciais e possibilitar ao mesmo o acesso às informações cadastradas no software.

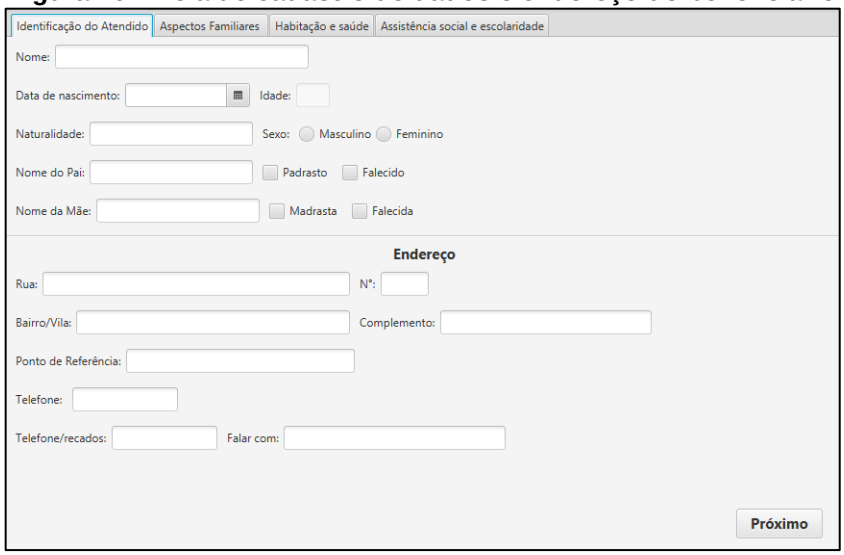
Figura 25 – Tela principal da versão 1



Fonte: Autoria própria (2022).

Logo após ao login, o usuário é direcionado para a tela principal do sistema, que contém botões para as funcionalidades disponíveis para uso. Nesta versão, a tela contém apenas o menu para o cadastro de beneficiários. A tela principal é apresentada na Figura 25 e será preenchida com mais botões de funcionalidades ao logo dos capítulos deste trabalho.

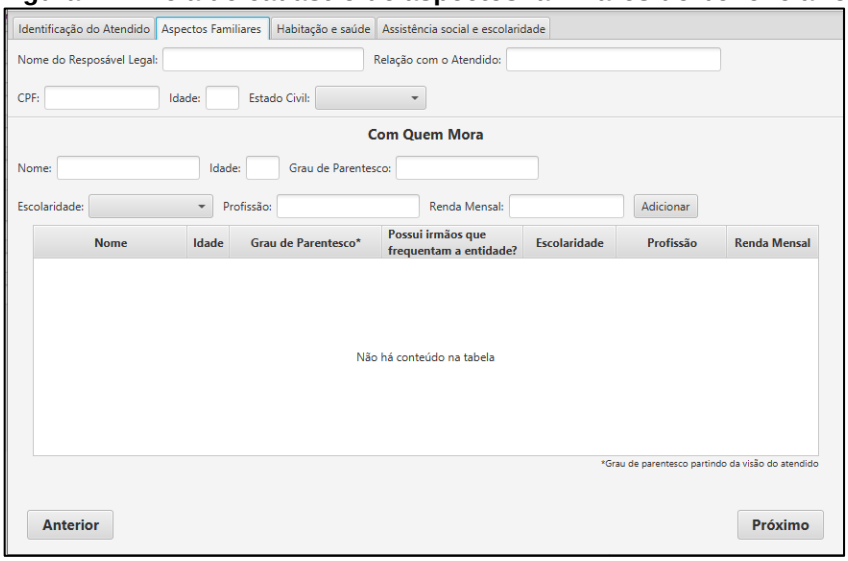
Figura 26 – Tela de cadastro de dados e endereço do beneficiário



Fonte: Autoria própria (2022).

A Figura 26 ilustra a tela de cadastro de usuários, porém focada nos aspectos de identificação do atendido e seu endereço. Os campos nome, data de nascimento, naturalidade, sexo, nome do pai e nome da mãe são obrigatórios para a identificação do usuário, sendo o campo idade calculado automaticamente quando preenchida a data de nascimento. No endereço são obrigatórios os campos rua, número, bairro e o primeiro telefone.

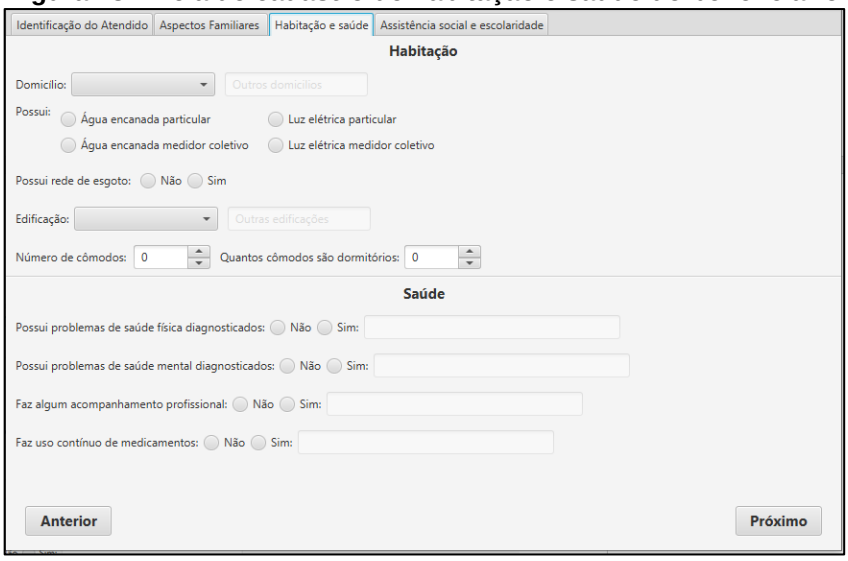
Figura 27 – Tela de cadastro de aspectos familiares do beneficiário



Fonte: Autoria própria (2022).

A Figura 27 ilustra a tela de cadastro de usuários focada nos aspectos familiares do usuário atendido. Nesta tela deve ser preenchido obrigatoriamente os campos de responsável legal, não sendo obrigatório que haja outras pessoas com quem o usuário conviva na mesma residência.

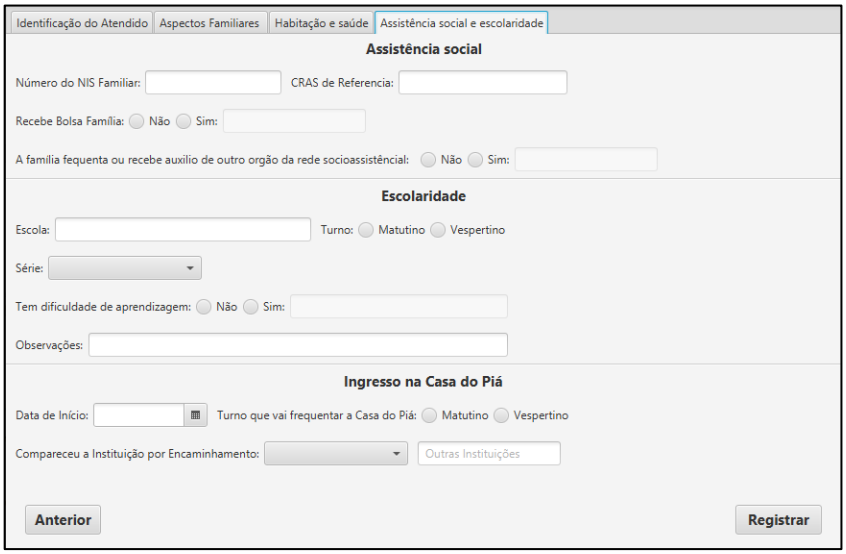
Figura 28 – Tela de cadastro de habitação e saúde do beneficiário



Fonte: Autoria própria (2022).

A Figura 28 ilustra a tela de cadastro de usuários, focada na habitação e saúde do atendido. Todos os componentes habilitados são obrigatórios nesta tela, tendo alguns campos de texto que somente serão habilitados ao selecionar a opção “Outros” em domicílio ou em edificação, ou ao selecionar “Sim” em alguma das opções de saúde.

Figura 29 – Tela de cadastro de assistência social e escolaridade do beneficiário



Fonte: Autoria própria (2022).

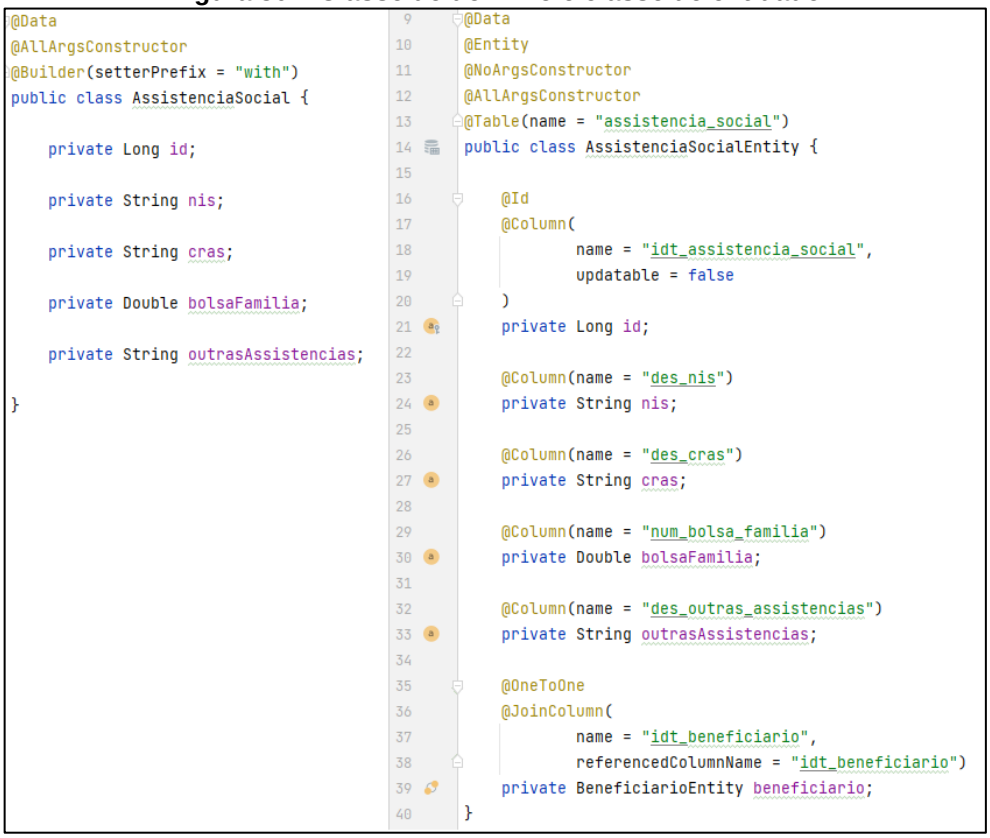
A Figura 29 ilustra a tela de cadastro de usuários, com o foco nos campos de assistência social, escolaridade e alguns dados para o ingresso. Os campos de NIS, CRAS de referência e observação são opcionais, os demais campos habilitados são obrigatórios, tendo alguns componentes de texto habilitados ao selecionar a opção “Sim” em alguns campos, similar ao comportamento da tela representada na Figura 28.

3.1.3.3 Implementação do código-fonte

O desenvolvimento do código fonte teve o início paralelo à criação das telas. Iniciou-se implementando as classes que compõem o domínio do software, ou seja, aquelas modeladas no Diagrama de Classes. Estas são as principais entidades onde os dados do sistema navegam, e têm como função garantir que as regras de negócio serão atendidas, podendo conter validações para este propósito.

Foram criadas também as classes de entidade, que são parecidas classes de domínio, porém específicas para o acesso ao banco de dados. Estas classes contêm anotações do Spring e do Hibernate para que mapeiem as tabelas descritas no DER. Um exemplo de uma classe de domínio ao lado de uma entidade é apresentado na Figura 30.

Figura 30 – Classe de domínio e classe de entidade



```java
@Data
@AllArgsConstructor
@Builder(setterPrefix = "with")
public class AssistenciaSocial {

    private Long id;

    private String nis;

    private String cras;

    private Double bolsaFamilia;

    private String outrasAssistencias;

}
```

```java
@Data
@Entity
@NoArgsConstructor
@AllArgsConstructor
@Table(name = "assistencia_social")
public class AssistenciaSocialEntity {

    @Id
    @Column(
            name = "idt_assistencia_social",
            updatable = false
    )
    private Long id;

    @Column(name = "des_nis")
    private String nis;

    @Column(name = "des_cras")
    private String cras;

    @Column(name = "num_bolsa_familia")
    private Double bolsaFamilia;

    @Column(name = "des_outras_assistencias")
    private String outrasAssistencias;

    @OneToOne
    @JoinColumn(
            name = "idt_beneficiario",
            referencedColumnName = "idt_beneficiario")
    private BeneficiarioEntity beneficiario;
}
```

Fonte: Autoria própria (2022).

A classe à esquerda é onde trafegam os dados de domínio na aplicação, enquanto à direita está a classe de entidade, que auxilia no mapeamento dos objetos do software para o banco de dados. Esta classe contém as anotações “@Table”, “@Entity” e “@Column”, as duas primeiras indicam ao Spring e ao Hibernate que esta classe é uma entidade e o nome da tabela que deve ser usada, enquanto a outra anotação mapeia os nomes de cada atributo para o nome de coluna na tabela. Outro detalhe são as anotações no atributo “beneficiario”, que fazem o mapeamento da coluna que será uma chave estrangeira para relacionar esta tabela com a tabela “beneficiario”, através de seu “id”. É necessário converter a classe de domínio para a de entidade para que sejam salvos os dados de um objeto.

Figura 31 – Método toEntity

```
public static AssistenciaSocialEntity toAssistenciaSocialEntity(AssistenciaSocial assistenciaSocial) {
    if (assistenciaSocial == null)
        return null;
    return new AssistenciaSocialEntity(
            assistenciaSocial.getId(),
            assistenciaSocial.getNis(),
            assistenciaSocial.getCras(),
            assistenciaSocial.getBolsaFamilia(),
            assistenciaSocial.getOutrasAssistencias(),
             beneficiario: null
    );

}
```

Fonte: Autoria própria

Foram criados métodos como o da Figura 31, que recebe como parâmetro um objeto e o converte para a classe que tem o mapeamento para o banco de dados. Após essa conversão é possível salvar os objetos retornados por este método através de interfaces implementadas pelo Hibernate.

Em seguida, foram criadas as classes de serviço. Estas agrupam métodos que operam as classes de domínio do sistema. São as implementações das funcionalidades esperadas do software. As classes de serviço fazem parte do modelo do software, e juntamente com o domínio do sistema, garantem as regras de negócio funcionando.

Figura 32 – Exemplo de código da classe de serviço

```
public boolean cadastrarBeneficiario(Beneficiario beneficiario) throws CadastroBeneficiarioException {
    BeneficiarioEntity beneficiarioEntity = BeneficiarioMapper.toBeneficiarioEntity(beneficiario);
    AssistenciaSocialEntity assistenciaSocialEntity = BeneficiarioMapper.
            toAssistenciaSocialEntity(beneficiario.getAssistenciaSocial());
    assistenciaSocialEntity.setBeneficiario(beneficiarioEntity);
    try {
        beneficiarioRepositorio.save(beneficiarioEntity);
        assistenciaSocialRepositorio.save(assistenciaSocialEntity);
    } catch (IllegalArgumentException e) {
        throw new CadastroBeneficiarioException("Erro ao salvar beneficiario no banco de dados");
    }
    return true;
}
```

Fonte: Autoria própria (2022).

O trecho de código apresentado na Figura 32 exemplifica o uso das estruturas apresentadas anteriormente. Este método recebe uma instância de “Beneficiario” e a partir dela são criadas e salvas as entidades para esta classe e para a “AssistenciaSocial”. Nota-se que a instancia de “BeneficiarioEntity” é necessária para que os dados referentes aos dados de assistência social sejam salvos, para que o framework mapeie a chave estrangeira usada na segunda entidade. Este método faz parte de uma classe de serviço e salva todos os dados do beneficiário, sendo simplificado para melhor entendimento do leitor.

Juntamente o desenvolvimento das interfaces gráficas, as classes de controle foram implementadas. Estas existem para que a GUI e o sistema possam conversar entre si e disponibilizar as funcionalidades contidas nas classes de serviço ao usuário.

Figura 33 – Método da classe de controle

```
@FXML
public void cadastrarBtnOnAction(ActionEvent event) {

    try {
        Beneficiario beneficiario = preencherBeneficiario();
        if(beneficiarioService.cadastrarBeneficiario(beneficiario))
            System.out.println("Cadastro completo!");
        else
            System.out.println("Houve um problema para\n completar o cadastro!");
    } catch (CadastroBeneficiarioException e) {
        System.out.println(e.getMessage() + "Houve um problema para\n completar o cadastro!");
    }
}
```

Fonte: Autoria própria (2022).

O método apresentado na Figura 33 faz parte da classe de controle de cadastro e é executado ao apertar o botão de cadastrar, na última tela. O método “preencherBeneficiario” cria uma instancia de “Beneficiario” com os dados de todos os campos da interface gráfica. Com esta instancia, é usada a classe de serviço para salvar o cadastro.

Para a implementação do login, quando o usuário aperta o botão para entrar no sistema, são validados se os campos de nome de usuário e senha estão vazios ou com apenas espaços. Caso estejam preenchidos os campos citados, é verificado no banco de dados se há um funcionário com os dados informados. Atualmente não há implementada uma funcionalidade de cadastro de usuários, sendo estes cadastrados previamente pela chamada de um método na classe de serviço de funcionários apresentada na Figura 34.

Figura 34 – Classe de serviço de funcionários

```java
@Service
public class FuncionarioService {

    @Autowired
    private FuncionarioRepositorio funcionarioRepositorio;

    public boolean autenticarFuncionario(Funcionario funcionario){
        List<FuncionarioEntity> funcionarioEntities = funcionarioRepositorio.findByUsuario(funcionario.getUsuario());
        if(funcionarioEntities.size()==1){
            FuncionarioEntity funcionarioBanco = funcionarioEntities.get(0);
            return BCrypt.checkpw(funcionario.getSenha(), funcionarioBanco.getSenha());
        }
        return false;
    }

    public boolean cadastrarFuncionario(Funcionario funcionario){

        funcionario.setSenha(BCrypt.hashpw(funcionario.getSenha(), BCrypt.gensalt()));
        FuncionarioEntity funcionarioEntity = FuncionarioMapper.toFuncionarioEntity(funcionario);
        try {
            funcionarioRepositorio.save(funcionarioEntity);
        }catch (IllegalArgumentException e){
            return false;
        }
        return true;
    }

}
```

Fonte: Autoria própria (2022).

O método “cadastrarFuncionario” recebe uma instancia de funcionário e faz a criptografia da senha através da biblioteca jBcrypt, para finalmente salvar o resultado no banco de dados. Para verificar se a senha digitada pelo funcionário e a senha salva são as mesmas, a biblioteca usada tem o método “checkpw” que executa esta função. O uso desta biblioteca é interessante pois é usado a geração de “salt” juntamente com o hash das senhas, garantindo que mesmo que dois usuários tenham a mesma senha, o valor criptografado para ambas será diferente.

3.1.4 Feedback da Casa do Piá

Esta versão foi entregue para o cliente, que teve aproximadamente uma semana para testar. Ao entrar em contato com a Casa do Piá, foram apontados alguns pontos que deveriam ser corrigidos no sistema: (i) o tempo de inicialização do sistema era lento; (ii) quando o cadastro de usuário tinha alguma falha, era retornada a mensagem de que falhou, mas não apresentava o motivo da falha ou o campo que causou o erro; e (iii) não era possível visualizar os usuários que já estavam cadastrados.

Outro ponto levantado pelo cliente é que há alguns campos novos que eles começariam a anotar que não estão presentes do documento de exemplo enviado. São os campos para anotar se o beneficiário já foi repetente na escolaridade e em saúde, se tem alguma alergia.

Além do *feedback* da Casa do Piá, foi constatado que o JavaFX poderia ser usado para a criação dos protótipos nas versões seguintes. O desenvolvimento das telas com a utilização do JavaFX proporcionaram um desenvolvimento rápido da interface gráfica e sem a necessidade da implementação das funcionalidades de cada tela. Desta forma, foram criados protótipos que após a avaliação do cliente e suas correções, pôde ser utilizado na fase de desenvolvimento do software.

3.2 Visualização de Beneficiários e Geração do Documento de Inscrição – Versão 2

Terminado o primeiro ciclo do modelo espiral, iniciou-se o desenvolvimento da segunda versão do software. O processo de comunicação realizou-se juntamente com o *feedback* do ciclo anterior, que resultou em algumas correções necessárias para a entrega desta versão, além do projeto e implementação das novas funcionalidades escolhidas para este novo ciclo de desenvolvimento. As funcionalidades escolhidas para essa versão foram: (i) a visualização de detalhes sobre os beneficiários inscritos e a partir do cadastro criado, (ii) a geração da documentação com os dados deste beneficiário, além das correções propostas durante o *feedback*.

A Subseção 3.2.1 discorre sobre a comunicação com os funcionários da instituição. A Subseção 3.2.2 apresenta o projeto de software atualizado com as novas funcionalidades do sistema. A Subseção 3.2.3 aborda a implementação dos requisitos propostos para esta versão do software. Por fim, a Subseção 3.2.4 descreve o *feedback* obtido do usuário.

3.2.1 Comunicação com o cliente

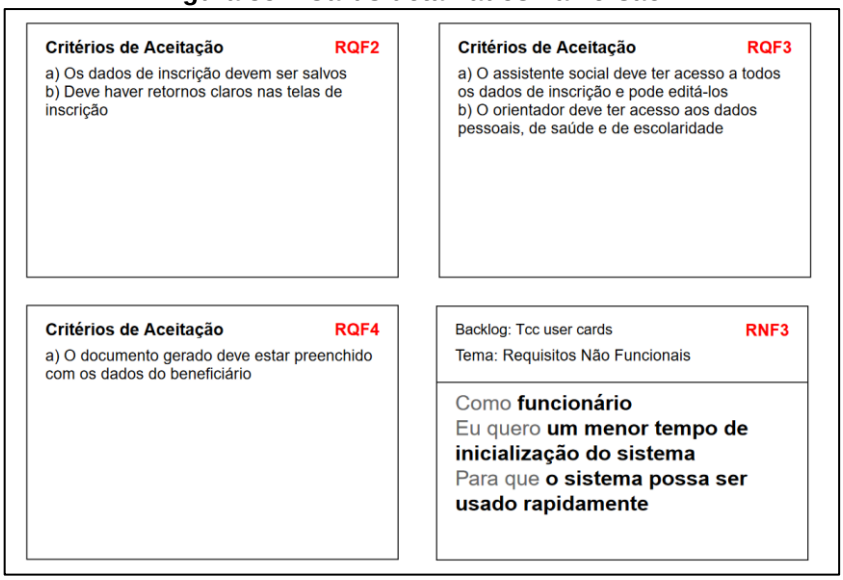
Iniciou-se a comunicação para o desenvolvimento desta versão abordando o *feedback* da versão anterior, pois as correções e melhorias apontadas tiveram seu planejamento e implementação feitas neste capítulo. Durante o *feedback* foi perguntado se haviam problemas nas funcionalidades ou interfaces desenvolvidas na versão testada e, como já abordado no capítulo anterior, o cliente informou os principais problemas: o tempo de inicialização do software; o sistema não informar o motivo que ocasionou a falha no cadastro de um usuário; e não ser possível visualizar

os dados dos beneficiários cadastrados, sendo este último ponto já previsto como um problema e priorizado logo em seguida à funcionalidade de inscrição, corrigindo-o.

Após o recebimento do *feedback*, foi abordado o detalhamento das funcionalidades escolhidas para implementação: (i) apresentação de dados do beneficiário cadastrado; e (ii) geração do documento de inscrição, onde usou-se o template definido em comunicações anteriores. Como especificado, os dados dos inscritos devem ser restritos apenas a quem tem autorização. O assistente social deve ter acesso a todos esses dados, sendo o único com poder de editá-los; enquanto o coordenador deve acessar apenas os dados necessários para o atendimento do beneficiário: dados pessoais, saúde e escolaridade.

Criou-se os critérios de aceite nos cards “RQF3” e “RQF4” respectivamente, referentes aos requisitos discutidos na reunião, assim como para a melhoria dos retornos nas telas de inscrição do “RQF2” e são apresentadas na Figura 35. Juntamente, registrou-se a melhoria no tempo de inicialização do software levantada pelo cliente e abordada como um requisito não funcional a ser implementado nesta versão.

Figura 35 – Cards detalhados na versão 2



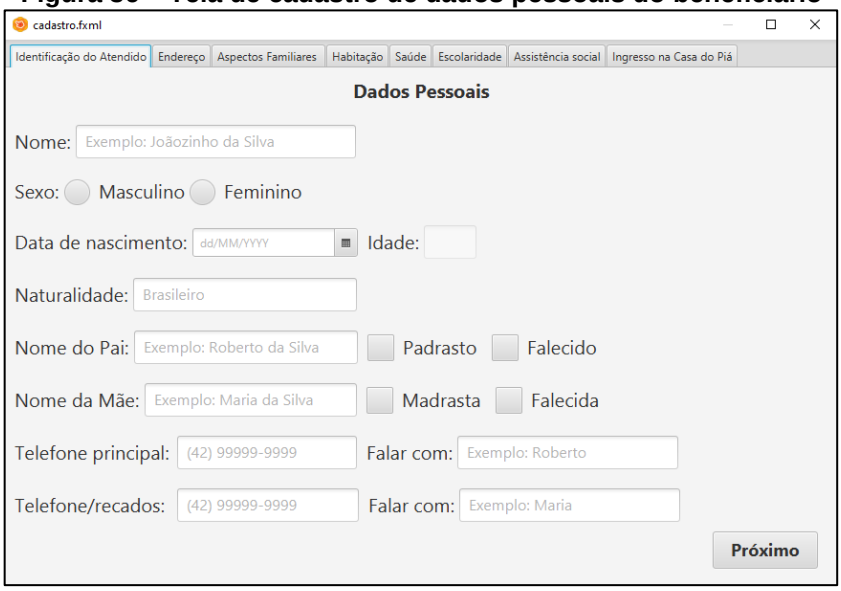
Fonte: Autoria própria (2022)

Informou-se ao cliente que a partir desta versão as telas dos protótipos serão mais próximas das telas finais pois seriam desenvolvidas já com as ferramentas que foram utilizadas para a criação das telas para o software. Iniciou-se a etapa de prototipação para a validação dos requisitos detalhados na comunicação.

Notou-se que para a visualização dos dados do beneficiário seria possível aproveitar as interfaces já existentes de cadastro, apenas desabilitando os campos editáveis das telas. Porém, para melhor controle de acesso aos dados, cada categoria deve ser separada, para que sejam disponibilizadas apenas os detalhes permitidos ao funcionário. Para isso, as telas de cadastro passaram por aprimoramentos, antes de serem usadas para a visualização e edição dos dados. Além da separação das categorias, os campos e fontes das interfaces foram aumentados para melhor aproveitamento da janela. Os campos passaram por um alinhamento, trazendo um aspecto visual mais agradável e elegante, além de facilitar para o usuário a leitura das telas devido à fonte maior.

Os protótipos apresentados a seguir são das telas de cadastro atualizadas, e em seguida são apresentadas as mudanças destas telas para serem utilizadas para visualização dos dados do beneficiário

Figura 36 – Tela de cadastro de dados pessoais do beneficiário



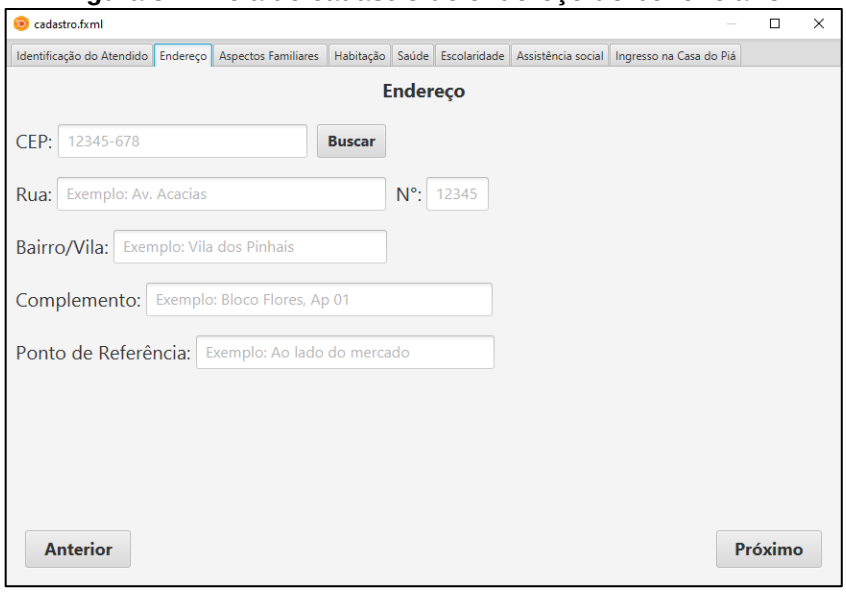
Fonte: Autoria própria (2022).

A Figura 36 apresenta a tela de cadastro atualizada dedicada aos dados pessoais. É possível identificar as categorias de dados separadamente, conforme as classes criadas na versão anterior. A tela com uma quantidade menor de campos para preencher e um espaçamento melhor dos recursos gráficos da interface a tornam mais limpa visualmente e mais direcionada para o dado a ser preenchido.

Nesta tela, os campos de “nome” do beneficiário, e do pai e da mãe, “naturalidade” e “falar com” passam a ser validados para conter apenas letras e

espaços. A data de nascimento e telefone devem conter apenas números, e somente o telefone de recados não é de preenchimento obrigatório.

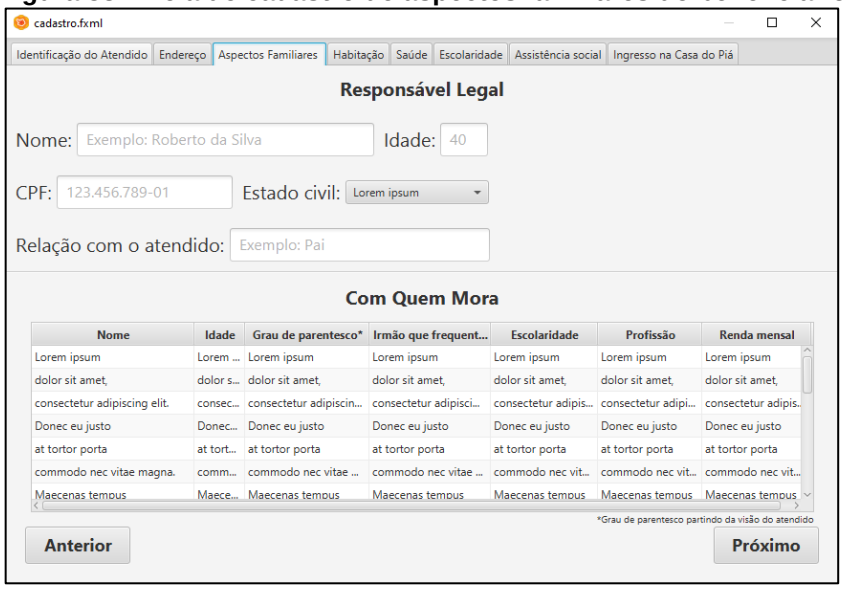
Figura 37 – Tela de cadastro de endereço do beneficiário



Fonte: Autoria própria (2022).

A Figura 37 é a tela atualizada de cadastro de endereço. Nesta tela também foi pensado em aproveitar o uso do CEP para consultá-los no banco de dados e preencher as informações salvas. Caso seja um novo, é consultada uma API que retorna os dados e completa os campos de “Rua” e “Bairro/Vila”. Quando há falhas para buscar o CEP pela API ou no banco de dados, o usuário pode preencher manualmente estes campos. Para evitar dados incorretos, estes campos são validados para não conter caracteres especiais.

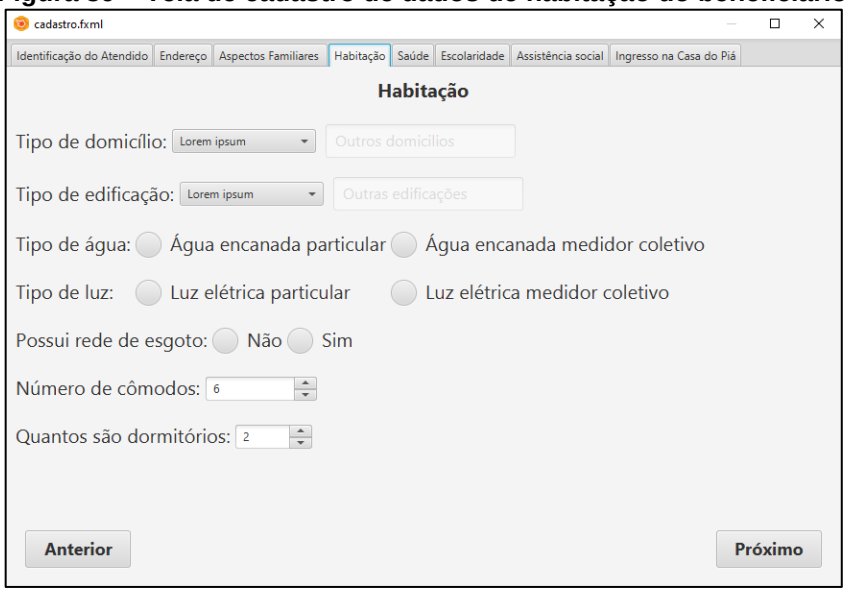
Figura 38 – Tela de cadastro de aspectos familiares do beneficiário



Fonte: Autoria própria (2022).

A tela da Figura 38 possuía campos que eram preenchidos e adicionados à tabela através de um botão “Adicionar”. Estes campos foram removidos para que sejam inseridos os dados diretamente na tabela. A tabela está preenchida com dados de exemplo e todos os campos são editáveis. Nesta tela, os campos de “Nome” e “Relação com o atendido” são validados para receber apenas letras, enquanto o CPF deve conter apenas números. Ao preencher os dados do responsável legal, o mesmo deve aparecer na primeira coluna da tabela.

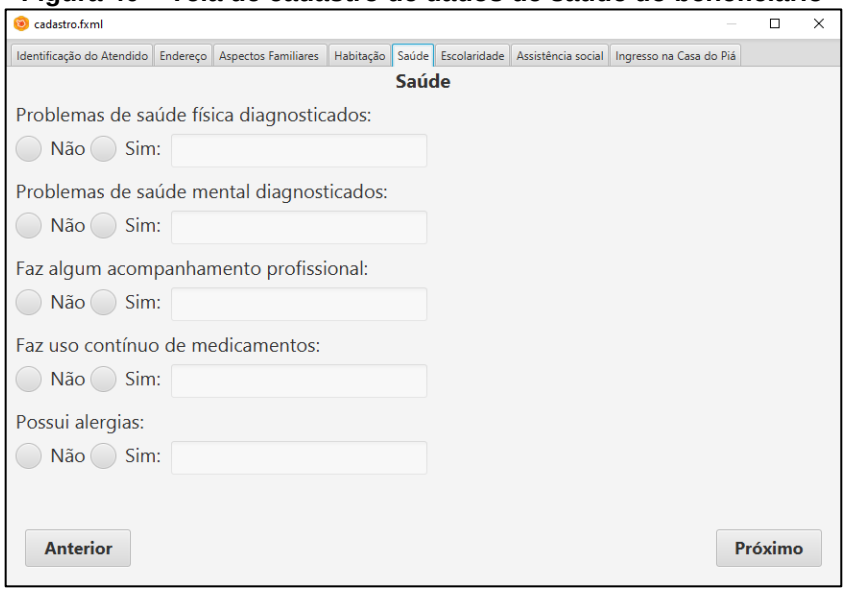
Figura 39 – Tela de cadastro de dados de habitação do beneficiário



Fonte: Autoria própria (2022).

A Figura 39 ilustra a tela de cadastro de dados de habitação do beneficiário. Nesta tela os campos de texto de “Tipo de domicílio” e “Tipo de edificação” permanecem desabilitados, caso o usuário escolha no “ComboBox” a opção “outros”, estes serão habilitados e de preenchimento obrigatório.

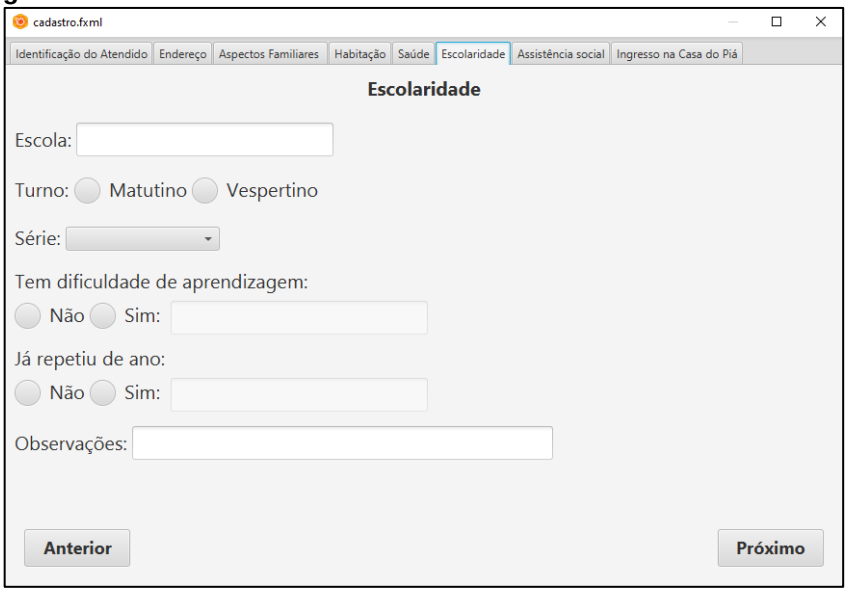
Figura 40 – Tela de cadastro de dados de saúde do beneficiário



Fonte: Autoria própria (2022).

É retratada na Figura 40 a tela onde se preenche o cadastro dos dados de saúde dos beneficiários. Esta tela tem diversos campos de “Sim” ou “Não”, e quando se escolhe a opção positiva os campos de texto ao lado são habilitados individualmente, tornando-se um campo de preenchimento obrigatório.

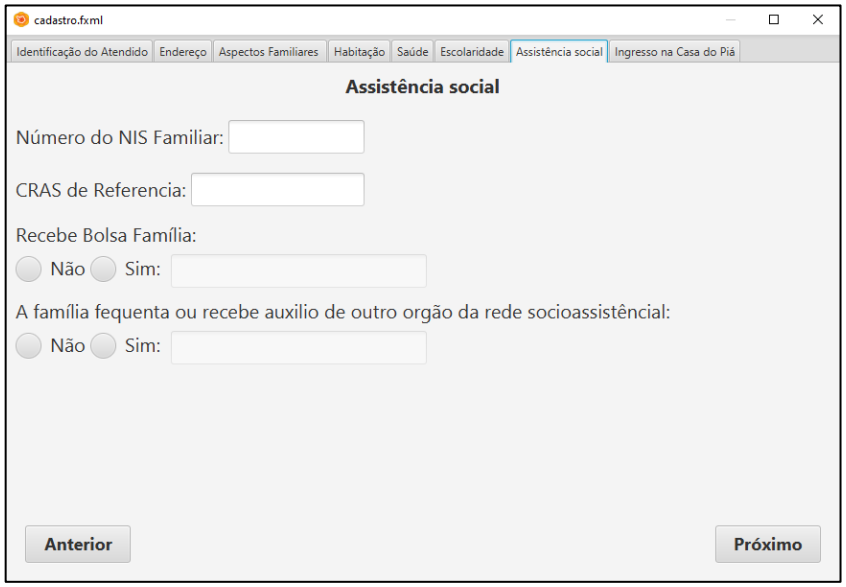
Figura 41 – Tela de cadastro de dados de escolaridade do beneficiário



Fonte: Autoria própria (2022).

Na tela de cadastro de dados de escolaridade do beneficiário apresentado na Figura 41 teve seus campos aumentados e disponibilizados na tela com um espaçamento melhor em comparação à tela antiga. Somente o campo de observações não é obrigatório nesta tela.

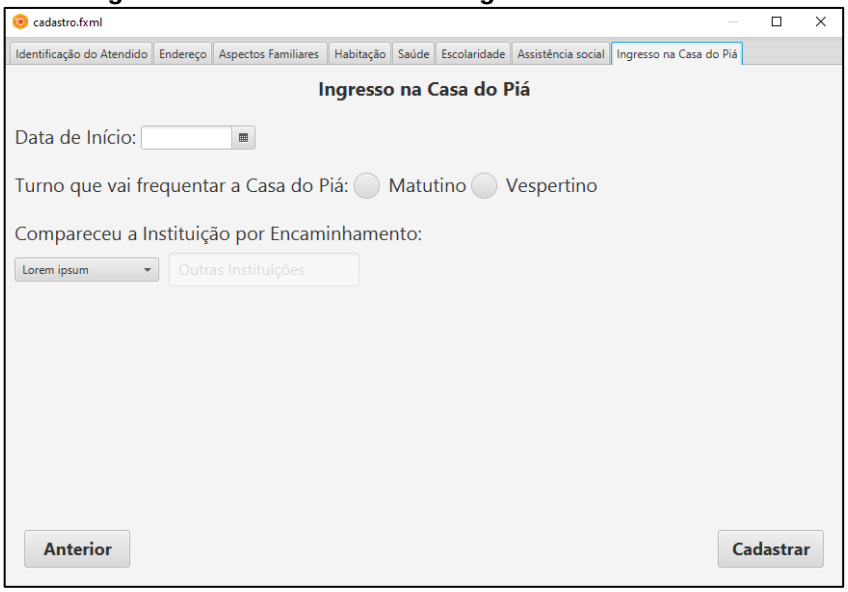
Figura 42 – Tela de cadastro de dados de assistência social do beneficiário



Fonte: Autoria própria (2022).

A interface da Figura 42 contém os dados de assistência social. O número do NIS, se preenchido, deve conter 11 números exatamente. Somente as opções de “Sim” ou “Não” são obrigatórias nessa tela e, caso sejam preenchidos com “Sim”, os campos descritivos ao lado de cada um se tornam mandatórios também.

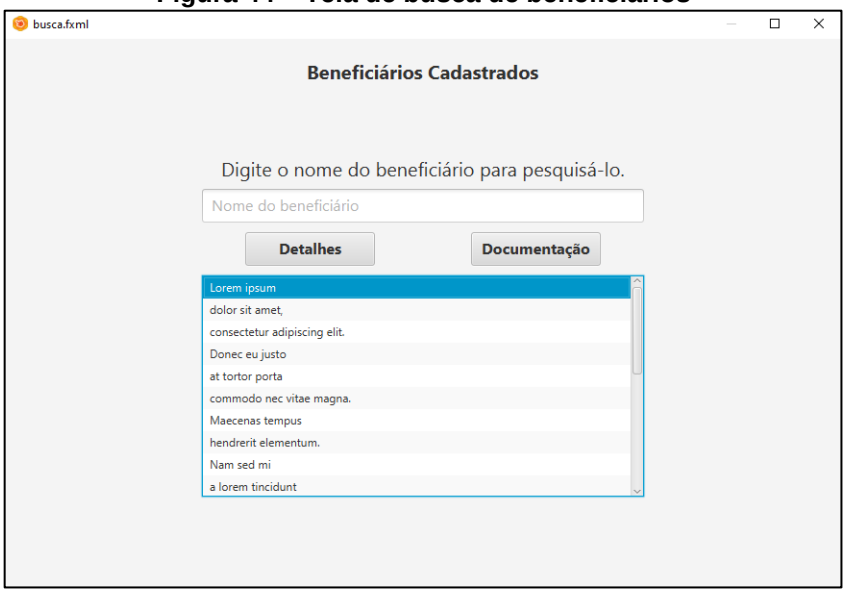
Figura 43 – Tela de dados de ingresso do beneficiário



Fonte: Autoria própria (2022).

Por fim, na tela da Figura 43 são preenchidos os últimos dados para cadastrar o beneficiário. Nesta tela são preenchidos a data de início, o turno e se houve encaminhamento para a instituição. Caso haja algum erro não reportado em alguma tela anterior, ao tentar finalizar a inscrição, será apresentada uma mensagem explicando o erro de cadastro.

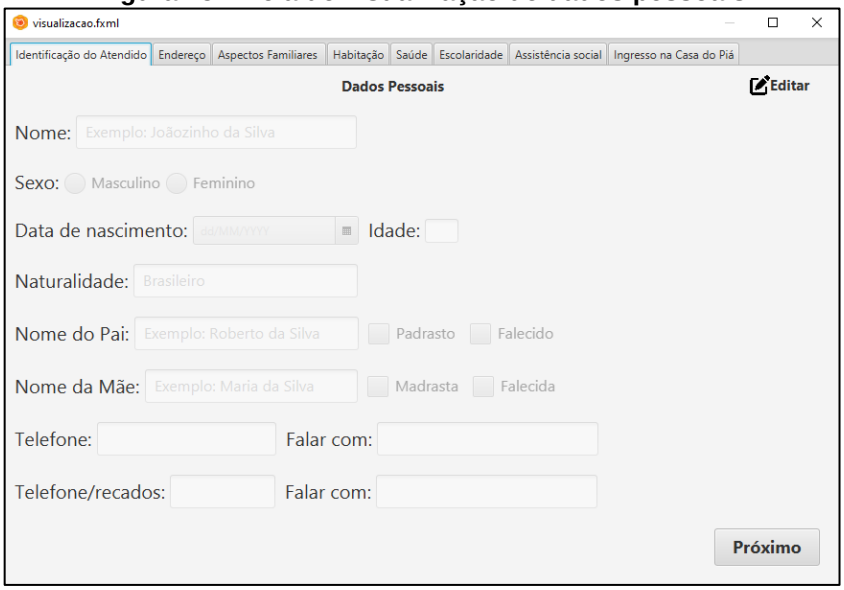
Figura 44 – Tela de busca de beneficiários



Fonte: Autoria própria (2022).

A tela da Figura 44 mostra onde o usuário poderá fazer a busca de beneficiários para visualizar os detalhes daquele que for selecionado na lista. Também é possível através dessa tela gerar a documentação de cadastro do mesmo. A pesquisa pode ser feita somente digitando o nome do beneficiário desejado.

Figura 45 – Tela de visualização de dados pessoais

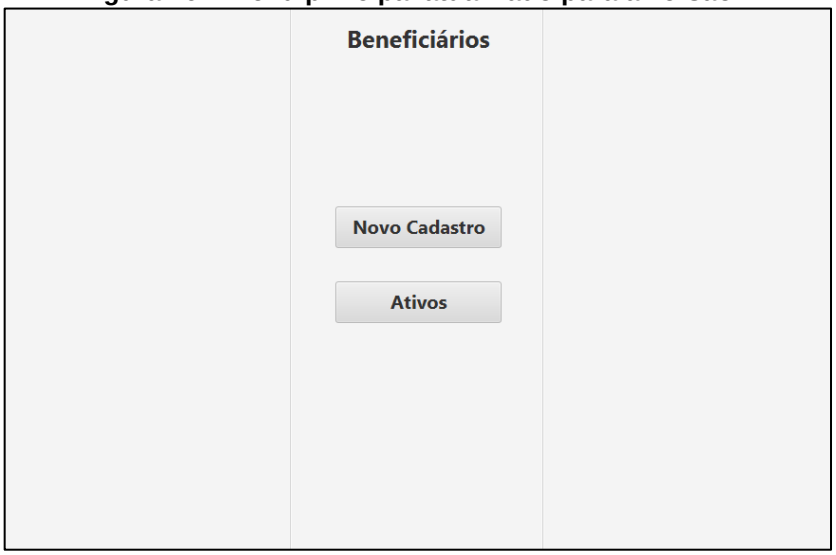


Fonte: Autoria própria (2022).

A tela da Figura 45 mostra como foi aproveitada as telas novas de cadastro para a visualização dos detalhes do beneficiário. Todos os campos foram desabilitados e serão preenchidos com os dados salvos do beneficiário selecionado na tela anterior. Foi adicionado ao canto superior direito um botão para habilitar a edição dos dados.

A partir desta versão o usuário pode cadastrar beneficiários, acessar e editar seus dados, ou gerar a sua documentação. É possível acessar as telas de cadastro através do menu principal. Os dados e a geração da documentação são acessíveis pela tela de busca. Porém, não há um caminho para acessar a tela de busca e por isso o menu principal, mostrado na Figura 46 foi atualizado, incluindo o acesso para a tela de busca com o botão “Ativos”.

Figura 46 – Menu principal atualizado para a versão 2



Fonte: Autoria própria (2022).

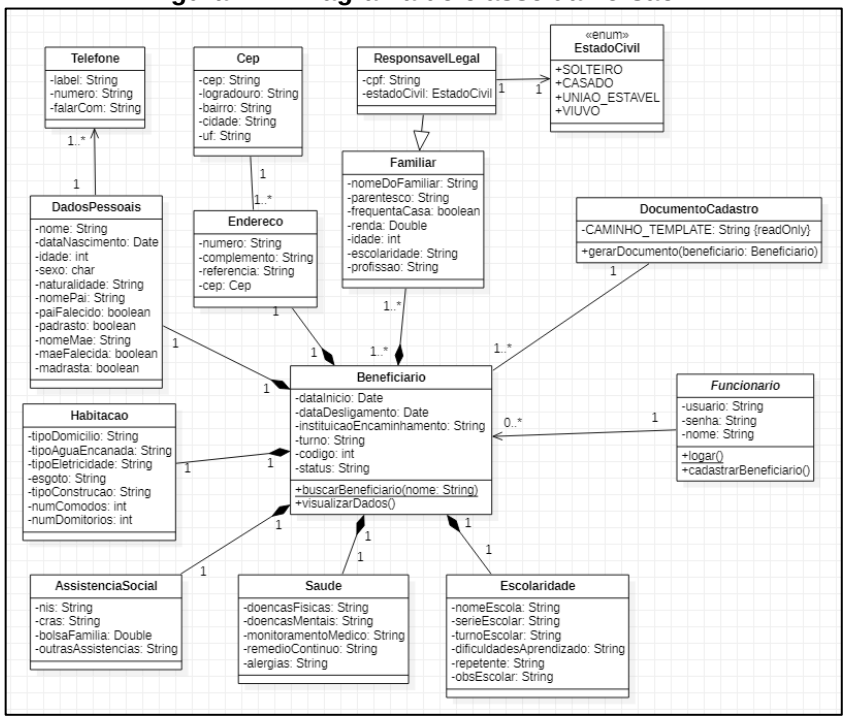
O retorno do cliente sobre os protótipos com as alterações de tela foi bastante positivo, pois as telas tiveram a usabilidade melhorada. A pesquisa e geração de documento também se apresentaram de forma simples e intuitiva, sendo o esperado dessas interfaces.

3.2.2 Projeto de software

A etapa de projeto de software nesta versão reaproveitou quase totalmente os diagramas modelados na versão anterior. Na modelagem de classes apresentado na Figura 47 adicionou-se apenas alguns métodos novos e as classes utilizadas para a geração do documento de inscrição. O DER teve os novos campos adicionados, conforme o *feedback* da versão anterior.

Para o requisito de visualização de dados do beneficiário, foi adicionado à classe “Beneficiario” o método “buscarBeneficiario”. Neste método será recebido o parâmetro “nome” e, ao buscar no banco de dados, é retornada uma lista de objetos desta classe que contêm o valor pesquisado. Outro método adicionado é o “visualizarDados”, responsável por retornar o objeto com os dados permitidos ao funcionário que o utilizar.

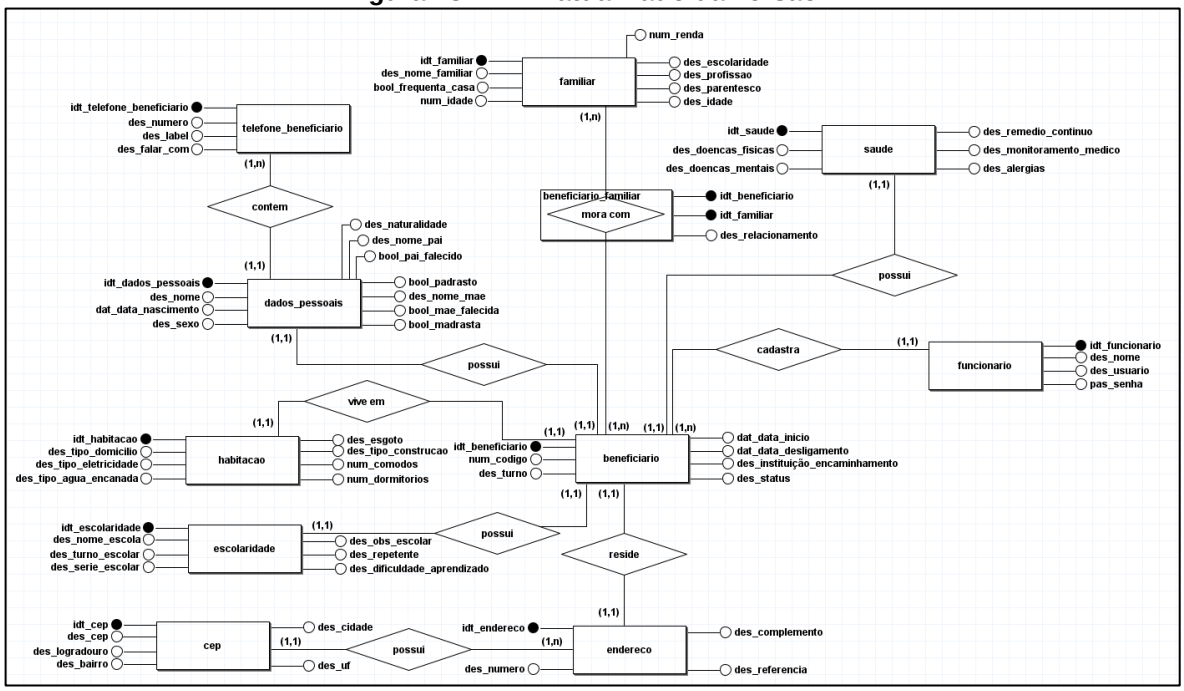
Figura 47 – Diagrama de classe da versão 2



Fonte: Autoria própria (2022).

Foi criada a classe “DocumentoCadastro”, representando a classe que irá implementar a geração de documento de cadastro do beneficiário. Esta classe contém o método “gerarDocumento”, que recebe como parâmetro a instância de “Beneficiario”, e o documento é preenchido a partir dos dados contidos neste objeto. Além disso, há um atributo “CAMINHO_TEMPLATE” que indica a pasta onde está o documento a ser preenchido com os dados recebidos.

Figura 48 – DER atualizado da versão 2



Fonte: Autoria própria (2022).

O DER atualizado é apresentado na Figura 48. Neste diagrama foram adicionados os campos “des_repetente” e “des_alergia” para cumprir o que foi pedido pelo cliente no *feedback*.

3.2.3 Desenvolvimento

Adotou-se como prelúdio ao desenvolvimento, a investigação da razão para o longo tempo de carregamento do sistema. Verificou-se através dos logs gerados pelo Spring que a etapa de inicialização do Hibernate tomava o maior tempo de carregamento, sendo o causador do problema levantado pelo cliente.

A inicialização do Hibernate conta com algumas validações, checagens de metadados e até mesmo a criação tabelas novas caso as mesmas não existam no banco de dados, tendo um alto tempo de carregamento. Para contornar esse problema, as propriedades da Figura 49 foram adicionadas ao arquivo “application.properties”.

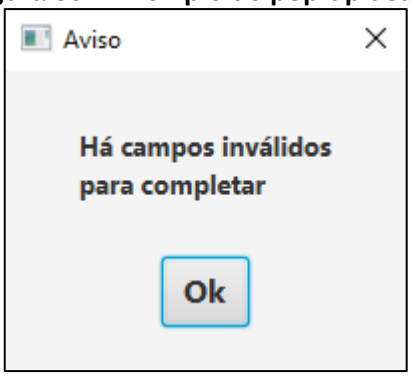
Figura 49 – Propriedades adicionadas ao Hibernate

```
spring.jpa.hibernate.ddl-auto=none
spring.jpa.properties.hibernate.temp.use_jdbc_metadata_defaults=false
```

Fonte: Autoria própria (2022).

A primeira propriedade desativa a validação e verificação de tabelas que devem existir, serem criadas ou atualizadas na estrutura do banco de dados. A segunda propriedade desativa a consulta para a configuração padrão para momentos de indisponibilidade do banco de dados. Ambas as configurações puderam ser implementadas, pois não há previsão de indisponibilidade do banco de dados durante o uso da aplicação e não utilizou-se do gerenciamento da estrutura de tabelas do Hibernate.

Figura 50 – Exemplo de pop-up usado



Fonte: Autoria própria (2022).

A melhoria dos retornos nas telas de cadastro consistiu de aplicar validações em campos preenchidos pelo usuário. Quando o botão de “Cadastrar” é pressionado e há campos que não estão de acordo com as validações, ou campos vazios, é aberta uma pequena janela *pop-up* avisando o usuário que há problemas para prosseguir, como apresentado na Figura 50.

Figura 51 – Código para apresentação de mensagem pop-up

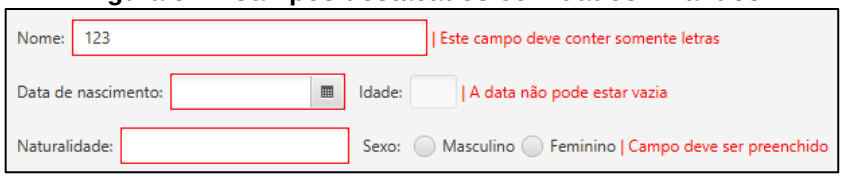
```
public static void showPopup(String message){
    PopupController popupController = JavaFxApplication.fxWeaver.getBean(PopupController.class);
    Stage window = new Stage();
    window.initModality(Modality.APPLICATION_MODAL);
    window.setTitle("Aviso");
    window.setResizable(false);
    VBox popup = (VBox) ScreensEnum.POPUP.getNode();
    popupController.setMessage(message);
    try{
        window.setScene(new Scene(popup, width: 200, height: 150));
    }catch (Exception e) {
        window.setScene(popup.getScene());
    }
    window.showAndWait();
}
```

Fonte: Autoria própria (2022).

Foi criada uma tela padrão para o *pop-up*, e um método estático, apresentado na Figura 51, para facilitar o uso desta interface. A mensagem mostrada nessa tela é alterada com o método “setMessage”. Este método estático pode ser chamado de

qualquer tela, apresentando a mensagem desejada. A criação deste *pop-up* neste formato aprimorou a interação das interfaces com o usuário.

Figura 52 – Campos destacados com dados inválidos



Fonte: Autoria própria (2022).

Além da janela apresentada, para que o usuário possa identificar rapidamente onde está o problema, os campos que não estão em conformidade com as validações são destacados. O contorno dos campos é destacado em vermelho e é apresentada uma mensagem detalhando o problema, como na Figura 52.

Figura 53 – Apresentação de um campo inválido

```
private boolean invalidField(Control field, String errorMessage) {
    if (!field.getStyleClass().contains("invalid-field"))
        field.getStyleClass().add("invalid-field");

    if (mapLabels.containsKey(field))
        mapLabels.get(field).setText(" | " + errorMessage);
    else {
        Label errorLabel = new Label( text: " | " + errorMessage);
        errorLabel.setTextFill(Color.RED);
        mapLabels.put(field, errorLabel);
        ((HBox) field.getParent()).getChildren().add(mapLabels.get(field));
    }

    return false;
}
```

Fonte: Autoria própria (2022).

O trecho de código da Figura 53 aplica uma classe criada em CSS ao campo indicado como inválido, então o JavaFX atualiza a formatação usada obtendo a aparência exemplificada na Figura 52. Além disso, a mensagem informada no parâmetro “errorMessage” é apresentada logo após o campo, explicando o que causou o erro.

Para a visualização dos dados do usuário foi necessária a implementação de duas consultas ao banco de dados. A primeira consulta retorna a lista de todos beneficiários ativos atualmente, enquanto a segunda consulta retorna o beneficiário escolhido para a visualização dos dados.

Figura 54 – Método para filtrar os nomes dos beneficiários

```
public void filterList(KeyEvent event) {
    beneficiariosList.setPredicate(beneficiario ->
            beneficiario.getDadosPessoais().getNome()
                    .toUpperCase().contains(searchBar.getText().toUpperCase()));
}
```

Fonte: Autoria própria (2022).

O trecho de código da Figura 54 recebe a lista de beneficiários retornada pelo banco de dados, compara o texto digitado na barra de pesquisa e filtra os usuários que contém este texto no nome. Ao escolher um dos nomes e clicar no botão de visualizar detalhes, é realizada uma nova busca no banco de dados direcionada ao beneficiário que teve o nome selecionado, retornando os dados que não foram carregados na primeira consulta.

Figura 55 – Método para busca de dados de beneficiários

```
@Repository
public interface BeneficiarioRepositorio extends JpaRepository<BeneficiarioEntity, Long> {

    List<BeneficiarioEntity> findByStatus(String status);
}
```

Fonte: Autoria própria (2022).

A primeira consulta foi desenvolvida com o método apresentado na Figura 55, onde o Hibernate faz com que este método funcione de acordo com o nome e parâmetros recebidos, sem que precise desenvolver a consulta manualmente. Na versão atual este método retorna todos os beneficiários, pois todos estão com o status ativo. Foi implementado desta forma, prevendo o desligamento dos beneficiários, que mudará o valor deste status. A segunda consulta é implementada automaticamente pelo framework ao usar a extensão à classe “JpaRepository”.

Figura 56 – Métodos de serviço do beneficiário

```
public List<Beneficiario> getBeneficiariosAtivos() {
    return beneficiarioRepositorio.findByStatus("ativo")
            .stream()
            .map(BeneficiarioMapper::fromBeneficiarioEntity)
            .collect(Collectors.toList());
}

public Beneficiario getBeneficiarioById(Long id){
    return beneficiarioRepositorio.findById(id)
            .map(BeneficiarioMapper::fromBeneficiarioEntity)
            .orElse( null);
}
```

Fonte: Autoria própria (2022).

Para tratar o retorno das consultas apresentadas acima, foram adicionados na classe de serviço do beneficiário dois novos métodos utilizando, respectivamente o retorno das consultas mostradas na Figura 56. O primeiro método implementa a busca para os inscritos ativos, fazendo os resultados passarem pela conversão dos dados da entidade para a classe de domínio. Enquanto o segundo método retorna apenas uma entidade, referente a um beneficiário selecionado na busca.

Figura 57 – Método "*fromEntity*"

```
public static AssistenciaSocial fromEntity(AssistenciaSocialEntity assistenciaSocialEntity){
    return AssistenciaSocial.builder()
            .withNis(assistenciaSocialEntity.getNis())
            .withCras(assistenciaSocialEntity.getCras())
            .withBolsaFamilia(assistenciaSocialEntity.getBolsaFamilia())
            .withOutrasAssistencias(assistenciaSocialEntity.getOutrasAssistencias())
            .build();
}
```

Fonte: Autoria própria (2022).

Os métodos implementados na classe de repositório retornam objetos de entidade que devem ser convertidos para as classes do domínio. Para isto, foram implementados métodos chamados "fromEntity" que convertem as entidades encontradas para objetos do domínio, que são usados para apresentar os dados para o usuário do sistema. Durante a conversão de entidade para o domínio, também são validados os dados recuperados. Por exemplo, se algum dos atributos que deveriam estar preenchidos e estiverem nulos, é retornado ao cliente que houve um erro para buscar esses dados. Um dos métodos implementados para essa conversão é apresentado na Figura 57.

Figura 58 – Método para atualização de dados do beneficiário

```
@Transactional
public void atualizarAssistenciaSocialBeneficiario(Beneficiario beneficiario){
    Session session = sessionFactory.getCurrentSession();
    AssistenciaSocial assistenciaSocial = beneficiario.getAssistenciaSocial();
    AssistenciaSocialEntity assistenciaSocialEntity = session
            .get(AssistenciaSocialEntity.class, assistenciaSocial.getId());

    assistenciaSocialEntity.setNis(assistenciaSocial.getNis());
    assistenciaSocialEntity.setCras(assistenciaSocial.getCras());
    assistenciaSocialEntity.setBolsaFamilia(assistenciaSocial.getBolsaFamilia());
    assistenciaSocialEntity.setOutrasAssistencias(assistenciaSocial.getOutrasAssistencias());

    session.flush();
    session.close();
}
```

Fonte: Autoria própria (2022).

Para atualizar os dados do beneficiário foram usados métodos como o da Figura 58. No exemplo é usado a anotação "@Transactional" que indica ao programa que se houver falhas durante a atualização dos dados deve-se dar *rollback* na

alteração, ou seja, retornar ao estado anterior à atualização e informar ao usuário a falha para atualizar. Este método atualiza os dados da entidade, e ao fazer a chamada ao “session.flush()” o estado da entidade é atualizada no banco de dados.

Figura 59 – Método que faz a consulta na API de CEPs

```
public Cep buscarCep(String cep) throws Exception{
    try {
        var cliente :HttpClient = HttpClient.newHttpClient();

        String servicoCep = "https://viacep.com.br/ws/{cep}/json/";

        HttpRequest conexao = HttpRequest.newBuilder(URI.create(servicoCep))
                .header( name: "accept", value: "application/json")
                .build();

        var resposta :HttpResponse<Supplier<Cep>> = cliente.send(conexao, new JsonBodyHandler<>(Cep.class));


        if (resposta.statusCode() != codigoSucesso)
            throw new RuntimeException("HTTP error code : " + resposta.statusCode());

        return resposta.body().get();
    } catch (Exception e) {
        throw new Exception("ERRO: " + e);
    }
}
```

Fonte: Autoria própria (2022).

Outro método implementado nesta versão é apresentado na Figura 59. Esta função recebe como parâmetro o número do CEP e faz uma requisição para o serviço “viacep”, que retorna os dados de endereço para este valor. O retorno do serviço é convertido em um objeto com os atributos preenchidos. A implementação deste método agiliza o processo do usuário ao cadastrar o endereço, precisando digitar menos dados para completar o formulário.

Para a geração do documento de inscrição dos beneficiários, foram aproveitados também os métodos implementados para a visualização de suas informações pois, é necessário o preenchimento de todos atributos da classe “Beneficiario” para que o documento seja gerado. A classe “DocumentoCadastro” recebe um objeto de “Beneficiario” como parâmetro do método “gerarDocumento”.

Figura 60 – Método de geração de documento de inscrição

```
public void generateDoc(Beneficiario beneficiario) {
    final String pdfTemplate = "/Casa/template.rtf";
    final String pdfOut = "/Casa/out.rtf";
    String line;

    try {
        File inFile = new File(pdfTemplate);
        File outFile = new File(pdfOut);
        BufferedReader br = new BufferedReader(new FileReader(inFile));
        FileWriter writer = new FileWriter(outFile);
        Pattern pattern = Pattern.compile("(\\$)(.*)(\\$)");
        Matcher matcher;

        while ((line = br.readLine()) != null) {
            matcher = pattern.matcher(line);
            while (matcher.find()) {
                line = line.replace( target: matcher.group(1) + matcher.group(2) + matcher.group(3), map.get(matcher.group(2)));
            }
            writer.append(line);
        }
        writer.close();
    } catch (java.io.IOException ex) {
        throw new RuntimeException("Erro ao gerar o documento");
    }
}
```

Fonte: Autoria própria (2022).

O método apresentado na Figura 60 busca por um padrão definido no código e preenchido no *template*, e faz a substituição destes pelos dados salvos do beneficiário. Desta forma, todos os campos são preenchidos automaticamente com os dados atualizados. Também é lançada uma exceção com mensagem de erro caso ocorra alguma falha durante a criação do documento.

3.2.4 Feedback da Casa do Piá

Algum tempo após o envio da nova versão para a Casa do Piá, foi realizada uma nova reunião. Esta nova comunicação possibilitou que o cliente traga suas conclusões e possíveis melhorias encontradas na versão desenvolvida, tornando o sistema mais próximo das necessidades do cliente.

As alterações nas telas de cadastro foram bem recebidas. De acordo com o cliente, apesar de não ter sido comentado na versão anterior, o tamanho da fonte presente era muito pequena e dificultava a leitura.

Outro ponto que passou despercebido nas versões anteriores foi a recuperação de senha. Não há nenhum mecanismo para o usuário redefinir a senha caso a tenha esquecido e, ao passar por esse problema, o cliente notou e pediu que fosse adicionado essa funcionalidade.

3.3 Versão 3 – Relatórios e desligamento de beneficiários

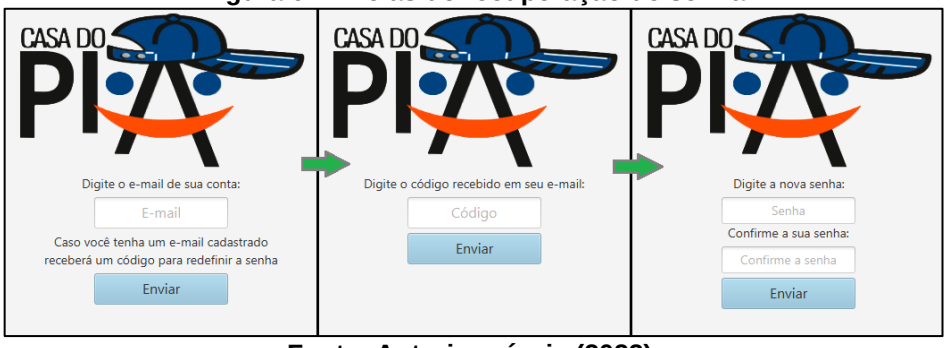
Este novo ciclo do modelo espiral iniciou-se juntamente com o *feedback* da versão anterior. Na etapa de comunicação, obteve-se mais detalhes para o projeto e implementação das novas funcionalidades escolhidas para essa versão, e dos pontos levantados no *feedback* da versão anterior. As funcionalidades desenvolvidas nesta versão foram o desligamento de beneficiário e os relatórios sobre os inscritos, além da recuperação de senha proposta durante o *feedback*.

A Subseção 3.3.1 discorre sobre a comunicação com os funcionários da instituição. A Subseção 3.3.2 apresenta o projeto de software atualizado com as novas funcionalidades do sistema. A Subseção 3.3.3 aborda a implementação dos requisitos propostos para esta versão do software. Por fim, a Subseção 3.3.4 descreve o *feedback* obtido do usuário.

3.3.1 Comunicação com a Casa do Piá

Esta etapa iniciou-se juntamente com a comunicação na qual se realizou o *feedback* da versão anterior. Foi pedido pela Casa do Piá uma forma de recuperar a senha na tela de login. Para cumprir este requisito, foi decidido que no lugar do nome de usuário para o login, será usado o e-mail institucional dos funcionários. Esta mudança possibilita o envio de um e-mail para a redefinição da senha.

Figura 61 – Telas de recuperação de senha



Fonte: Autoria própria (2022).

As telas do fluxo de redefinição de senha são apresentadas na Figura 61. O funcionário, ao clicar na opção de “esqueci a senha”, deverá informar o e-mail que

deseja recuperar. Então o usuário receberá um código para digitar na tela seguinte. Na última tela ele pode digitar sua nova senha.

Após anotados os pontos de melhoria apontados pelo cliente, foram detalhados os requisitos funcionais RQF5 – visualização de relatórios sobre os inscritos no sistema; e RQF6 – desligamento dos beneficiários da instituição. Os funcionários da Casa do Piá desejavam, através do software, ter em mãos os seguintes relatórios:

- quantidade de beneficiários atendidos, classificando-os por sexo;
- quantidade de famílias atendidas;
- quantidade de beneficiários que frequentam cada turno, classificando-os por sexo;
- quantos recebem Auxilio Brasil;
- quantos possuem Cadastro Único;
- quantidade de desligamentos por mês.

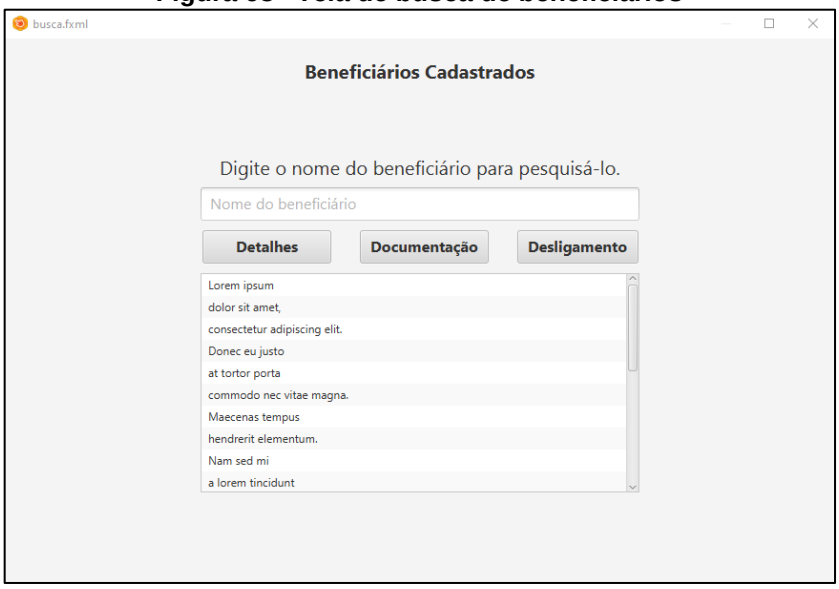
Para isso, é necessária uma nova tela que contenha as informações desejadas. Inicialmente, foi pensada em uma tela com diversos gráficos por exemplo, gráficos de barras ou de pizza. Ao questionar para a Casa do Piá a preferência para o formato dos relatórios, obteve-se a resposta de que inicialmente precisam apenas dos dados numéricos, pois estes são consultados, copiados e usados. O protótipo da Figura 62 foi pensado como uma forma de tentar dispor os dados pedidos pelo cliente de forma organizada e bem espaçada.

Figura 62 – Tela de relatórios

Fonte: Autoria própria (2022).

O desligamento também necessitou de uma nova tela, onde o beneficiário seja selecionado para que este seja desligado. Além da mudança do status do beneficiário para inativo, é necessário registrar o motivo do desligamento. Foi perguntado quais os principais motivos que levam a esta ação, para que sejam considerados no desenvolvimento desta nova interface. Os principais motivos são: (i) mudança de endereço (município/região/bairro), (ii) a pedido do responsável e (iii) por idade. Com o fim desta reunião, iniciou-se a prototipação das novas telas.

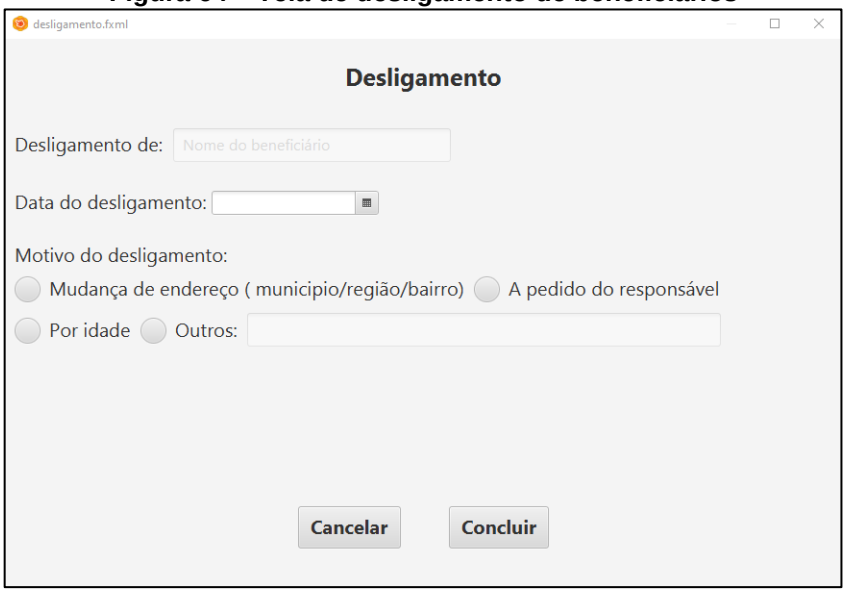
Figura 63 –Tela de busca de beneficiários



Fonte: Autoria própria (2022).

A tela da Figura 63 mostra onde o usuário pode encontrar o acesso à funcionalidade de desligamento. Ao selecionar um dos nomes encontrados pela busca de beneficiários, o usuário deve clicar no botão de desligamento que o encaminha para a nova tela, já mostrando qual é o beneficiário que será desligado.

Figura 64 – Tela de desligamento de beneficiários



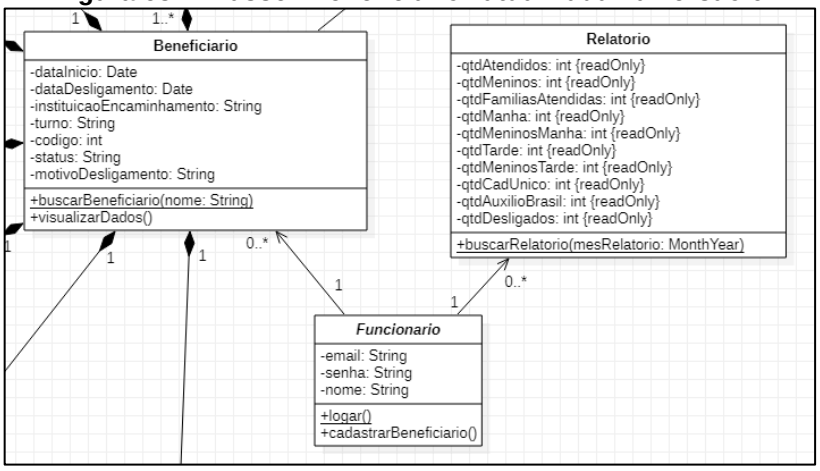
Fonte: Autoria própria (2022).

A nova tela, apresentada na Figura 64, é onde será realizado o desligamento de beneficiários. O primeiro campo mostra o nome do beneficiário a ser desligado, selecionado na tela anterior. Em seguida, é necessário preencher a data do desligamento. Ao final, o usuário deve escolher um dos motivos de desligamento adicionados à tela por serem os mais frequentes, agilizando o trabalho do usuário, e caso não seja uma das opções de motivos de desligamento, o funcionário deve selecionar “Outros” e digitar a razão do desligamento para então concluir o desligamento.

3.3.2 Projeto de Software

Nesta versão foram atualizados tanto o diagrama de classe quanto o DER. Apesar de alterações pequenas, ambos os diagramas tiveram alguns ajustes para a implementação dos requisitos.

Figura 65 – Classe “Beneficiario” atualizada na versão 3

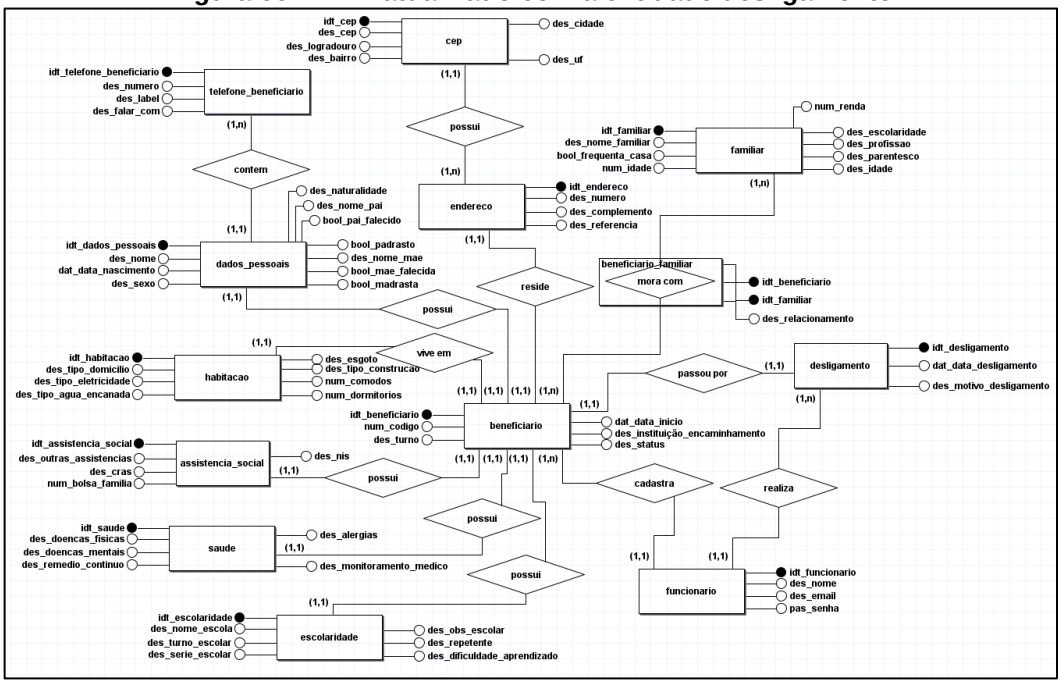


Fonte: Autoria própria (2022).

As alterações do Diagrama de Classe são apresentadas na Figura 65. Na classe “Beneficiario” adicionou-se um novo atributo “motivoDesligamento”, para salvar o valor preenchido para este campo durante o desligamento. Foi alterado também o atributo “usuario” na classe “Funcionario” para cumprir o requisito de recuperação de senha. Além da nova classe “Relatorio” adicionada nesta nova versão do Diagrama de Classe.

A classe “Relatorio” contém todos os dados necessários ao usuário e somente é instanciada pelo método estático “buscarRelatorio”. Este método recebe como parâmetro um “MonthYear”, ou seja, o mês e ano referente ao relatório que deve ser buscado. Com esse parâmetro recebido, é buscado no banco de dados as informações desejadas e instanciado o relatório completo, preenchendo a tela apresentada no protótipo.

Figura 66 – DER atualizado com a entidade desligamento



Fonte: Autoria própria (2022).

O DER apresentado na Figura 66 não teve alterações referente aos relatórios, pois estes são obtidos apenas com consultas ao banco de dados. Porém, uma nova entidade foi criada para representar o desligamento de beneficiários, onde são salvos a data de desligamento e o motivo de desligamento.

Foi criada esta nova entidade apenas no DER e não no Diagrama de Classe, pois a classe “Beneficiario” agrupa os dados referentes ao cadastro, enquanto no banco de dados o desligamento pode ser considerado uma entidade independente. Isto acontece porque quando os dados são cadastrados, o beneficiário não tem um registro nesta tabela, e só passa a ter quando é realizado o desligamento deste. A criação desta entidade tira a dependência das colunas relacionadas ao desligamento dos dados do beneficiário.

3.3.3 Desenvolvimento

Iniciou-se o desenvolvimento fazendo as alterações pedidas no *feedback* da versão anterior. A classe “Funcionario” teve a alteração do atributo “usuário” pelo atributo “email” tanto na classe de domínio quanto na de entidade. Em seguida, foi necessário criar um método na classe de serviço que mandasse um e-mail para o funcionário fazer a recuperação de senha.

Figura 67 – Método para envio de e-mail de recuperação de senha

```
private void createSession() {
    Properties props = new Properties();
    props.put("mail.smtp.host", "smtp-mail.outlook.com");
    props.put("mail.smtp.port", "587");
    props.put("mail.smtp.starttls.enable", "true");
    props.put("mail.smtp.ssl.protocols", "TLSv1.2");
    props.put("mail.smtp.auth", "true");

    this.session = Session.getInstance(props, new Authenticator() {
        @Override
        protected PasswordAuthentication getPasswordAuthentication() {
            return new PasswordAuthentication(emailUsername, emailPassword);
        }
    });
}

public void enviarEmail(String title, String msg, String toEmail) throws MessagingException {

    Message message = new MimeMessage(session);
    message.setFrom(new InternetAddress(this.emailUsername));
    message.setRecipients(
            Message.RecipientType.TO, InternetAddress.parse(toEmail));
    message.setSubject(title);

    MimeBodyPart mimeBodyPart = new MimeBodyPart();
    mimeBodyPart.setContent(msg, type: "text/html; charset=utf-8");

    Multipart multipart = new MimeMultipart();
    multipart.addBodyPart(mimeBodyPart);

    message.setContent(multipart);

    Transport.send(message);
}
```

Fonte: Autoria própria (2022).

O trecho de código da Figura 67 tem dois métodos, o primeiro faz as configurações para o envio de e-mail; o segundo faz o envio do e-mail, após ter a “session” configurada pelo primeiro. O segundo método é chamado pela classe de serviço de beneficiário, onde são passados os parâmetros: (i) “title”, o título do e-mail; (ii) “msg”, o corpo da mensagem; e (iii) “toEmail”, a lista de e-mails de destino.

Figura 68 – Chamada do envio de e-mail

```
public void recuperarSenha(String email) throws MessagingException {
    String titulo="Recuperação de senha Casa do Piá";
    ultimoToken = emailUtils.gerarToken();
    String msg = "O seu token para recuperação de senha é:\n" + ultimoToken;
    emailUtils.enviarEmail(titulo, msg, email);
}
```

Fonte: Autoria própria (2022).

A Figura 68 faz a chamada do método apresentado anteriormente. É criado um token que é salvo e enviado para o e-mail informado. O usuário deve então buscar o e-mail recebido e digitar o token para ser direcionado à tela de redefinição de senha. Foi utilizado este formato para garantir a criptografia já utilizada para as senhas em sua redefinição.

Em seguida, foi implementado o desligamento dos beneficiários. Para este requisito, foi desenvolvida a classe de entidade “Desligamento”, apresentado na Figura 69.

Figura 69 – Classe de entidade "Desligamento"

```java
@Data
@Entity
@NoArgsConstructor
@AllArgsConstructor
@Table(name = "desligamento")
public class DesligamentoEntity {

    @Id
    @Column(
            name = "idt_desligamento",
            updatable = false
    )
    private Long id;

    @Column(name = "dat_data_desligamento")
    private LocalDate dataDesligamento;

    @Column(name = "des_motivo_desligamento")
    private String motivoDesligamento;

}
```

Fonte: Autoria própria (2022).

Esta classe fará o mapeamento dos dados necessários para desligar os beneficiários da Casa do Piá e é instanciada diretamente no serviço que realiza esta ação. O método da Figura 70 faz o uso desta nova entidade, sendo o principal trecho de código para esta funcionalidade.

Figura 70 – Método de desligamento do beneficiário

```java
@Transactional
public void desligarBeneficiario(Beneficiario beneficiario,
                                 LocalDate dataDesligamento,
                                 String motivoDesligamento){
    DesligamentoEntity desligamentoEntity = new DesligamentoEntity(
            id: null,
            dataDesligamento,
            motivoDesligamento);

    desligamentoRepositorio.save(desligamentoEntity);

    Session session = sessionFactory.getCurrentSession();

    BeneficiarioEntity beneficiarioEntity = session
            .get(BeneficiarioEntity.class, beneficiario.getId());
    beneficiarioEntity.setStatus("INATIVO");

    session.flush();
    session.close();
}
```

Fonte: Autoria própria (2022).

No trecho de código acima, é criada a entidade de desligamento com os seus atributos preenchidos e salva no banco de dados. Além disso, o status do beneficiário é alterado para refletir o desligamento. O método apresentado acima utiliza a anotação “@Transactional” para evitar inconsistências na base. Por exemplo, se a entidade de desligamento for salva, mas houver alguma falha para salvar o status do beneficiário como “INATIVO”, será feito o *rollback* e o usuário informado que houve alguma falha. Foi iniciada então a implementação do próximo requisito.

A criação dos relatórios precisou de uma consulta que retorne todos os seus atributos preenchidos. Para isto, utilizou-se o repositório de usuários juntamente com a anotação “@Query”, que aceita *querys* personalizadas ao banco de dados para retornar os dados do relatório, preenchendo seus atributos. Foram utilizadas “nativeQuerys”, ou seja, *querys* na linguagem nativa do banco de dados, em vez do uso da linguagem do Hibernate. Não foi criado um repositório próprio para a classe “Repositorio”, pois esta não tem uma entidade própria na base de dados.

Figura 71 – Querys para dados do relatório

```
@Query(nativeQuery = true,
        value =
                "SELECT COUNT(b.id), dados_pessoais.sexo from beneficiario b " +
                        "INNER JOIN DadosPessoaisEntity dp ON be.id = dpe.id " +
                        "WHERE b.dataInicio >= :dataInicio " +
                        "AND b.dataDesligamento <= :dataFim " +
                        "GROUP BY dp.sexo")
List<Integer> qtdAtendidos(LocalDate dataInicio, LocalDate dataFim);

@Query(nativeQuery = true,
        value =
                "SELECT COUNT(b.id), dados_pessoais.sexo from beneficiario b " +
                        "INNER JOIN DadosPessoaisEntity dp ON be.id = dpe.id " +
                        "WHERE b.dataInicio >= :dataInicio " +
                        "AND b.dataDesligamento <= :dataFim " +
                        "AND b.des_turno='Vespertino' " +
                        "GROUP BY dp.sexo")
List<Integer> qtdAtendidosTarde(LocalDate dataInicio, LocalDate dataFim);
```

Fonte: Autoria própria (2022).

A Figura 71 apresenta algumas das *querys* utilizadas para capturar os atributos da classe “Relatorio”. Ao fazer um “count” em alguma coluna, é retornado o número de ids de beneficiários que foram atendidos e nas *querys* usadas, este foram agrupados por sexo. Isto possibilitou coletar três atributos de “Relatorio” em uma *query*, pois é contado a quantidade de meninos e meninas do período vespertino, e o total é obtido com a soma destes.

3.3.4 Feedback da Casa do Piá

Para os funcionários elaborarem o *feedback*, novamente tiveram um tempo para testar a nova versão do software. Após esse tempo, foi marcada uma nova reunião com a instituição para que trouxessem os pontos de melhorias.

Nesta versão, as funcionalidades e correções implementadas foram consideradas como corretas pelo cliente. Quando um jovem passa pelo desligamento, este deixa de constar entre os beneficiários ativos, assim como esperado. Os relatórios gerados trazem os dados corretos e facilmente utilizáveis pelos funcionários da Casa do Piá. Além da recuperação da senha, que traz uma forma fácil de resolver o problema causado pelo seu esquecimento.

Somente um ponto de melhoria foi levantado pela Casa do Piá, sendo ele, a necessidade de acessar alguns dos dados dos beneficiários, mesmo após desligados. Portanto, foi pedido que estes inscritos inativos sejam disponibilizados em alguma nova tela, separadamente dos ativos.

3.4 Versão 4 – Registro de atividades e presença de beneficiários

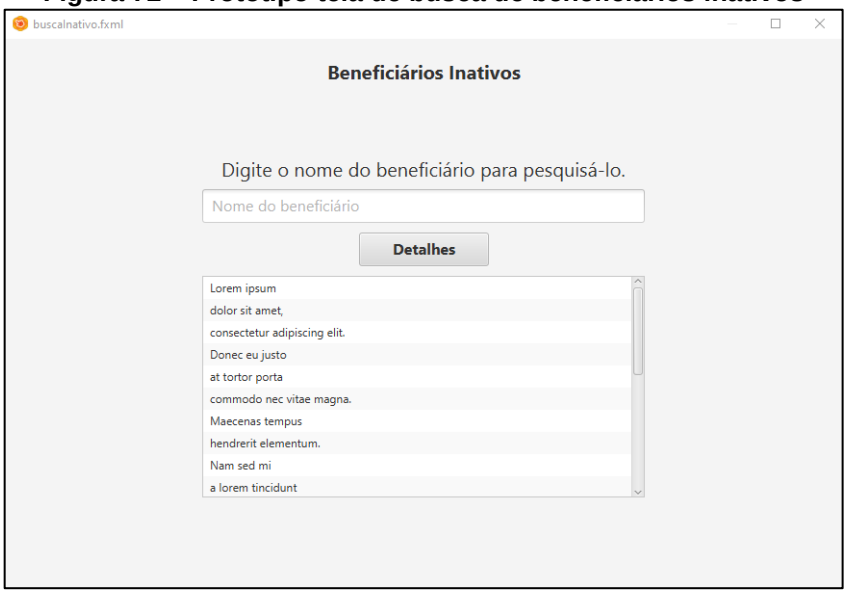
Este novo ciclo do modelo espiral iniciou-se juntamente com o *feedback* da versão anterior. Na etapa de comunicação, obteve-se mais detalhes para o projeto e implementação das novas funcionalidades escolhidas para essa versão, e dos pontos levantados no *feedback* da versão anterior. As funcionalidades desenvolvidas nesta versão foram o registro de atividades e da presença dos beneficiários, além das correções propostas durante o *feedback*.

A Subseção 3.4.1 discorre sobre a comunicação com os funcionários da instituição. A Subseção 3.4.2 apresenta o projeto de software atualizado com as novas funcionalidades do sistema. A Subseção 3.4.3 aborda a implementação dos requisitos propostos para esta versão do software. Por fim, a Subseção 3.4.4 descreve o *feedback* obtido do usuário.

3.4.1 Comunicação

Esta etapa iniciou-se juntamente com a comunicação na qual se realizou o *feedback* da versão anterior. Como citado, é necessário a consulta aos dados de beneficiários inativos. Estes já não podem mais ter interações novas com atividades, ou a geração do documento de inscrição devido ao seu status e, para evitar a confusão do usuário, a nova tela só apresentará estes inscritos inativos e somente com a possibilidade da consulta de seus dados, como no protótipo da Figura 72.

Figura 72 – Protótipo tela de busca de beneficiários inativos

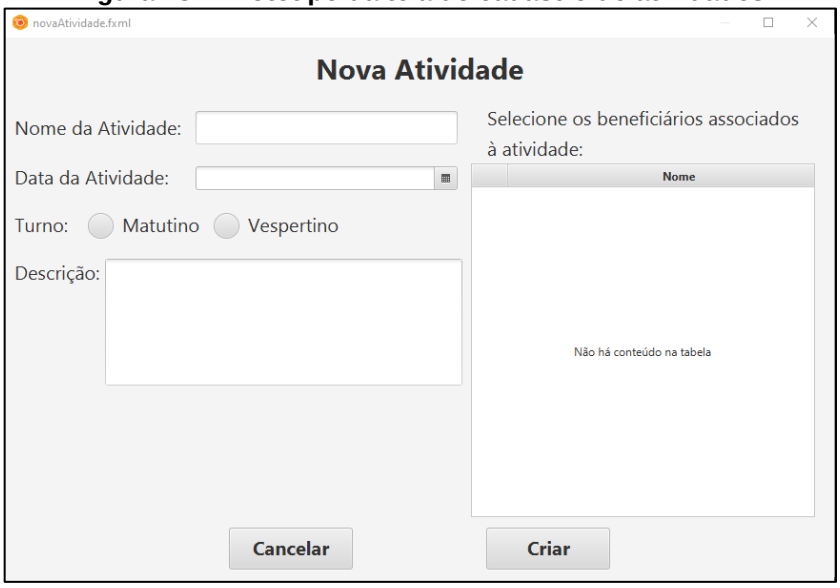


Fonte: Autoria própria (2022).

O protótipo apresentado é similar à busca de beneficiários ativos, tendo as funcionalidades que não cabem aos inativos, sendo possível pesquisá-los para visualizar seus dados. Além disso, a tela de detalhes também não deve permitir mais a edição dos dados do beneficiário cujo dados são apresentados.

Para detalhar o RQF7 – registro da realização de uma atividade, perguntou-se os principais dados que são anotados para as atividades realizadas. Estes são: um nome para a atividade, a data de realização, o turno em que será realizada e uma breve descrição da atividade. Outro ponto importante das atividades é a participação dos beneficiários, que precisam se inscrever naquelas que eles tiverem interesse, necessitando de um campo para que isso seja registrado. O protótipo desta tela é apresentado na Figura 73.

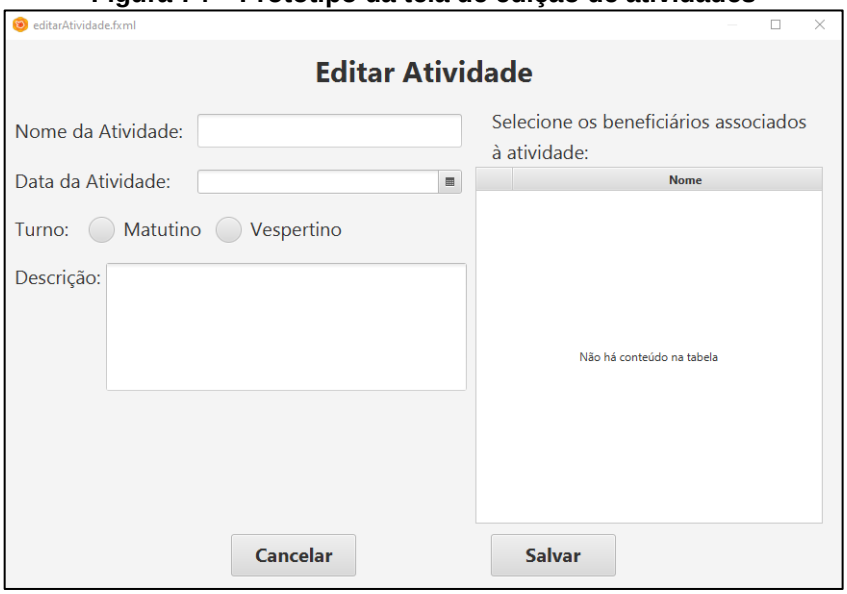
Figura 73 – Protótipo da tela de cadastro de atividades



Fonte: Autoria própria (2022).

O protótipo criado contém os campos “Nome da Atividade”, “Data da Atividade”, “Turno” e “Descrição”, onde são preenchidos os dados necessários para o registro da atividade. Além disso, há uma tabela que será preenchida com os nomes dos beneficiários aptos a participar desta atividade, ou seja, que estejam ativos na data de realização e que frequentem a entidade social no mesmo período da realização da atividade.

Figura 74 – Protótipo da tela de edição de atividades



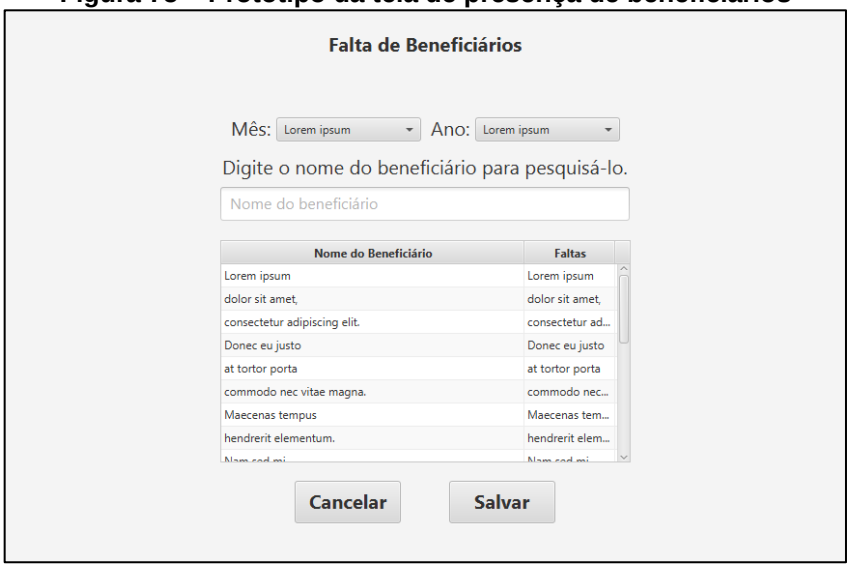
Fonte: Autoria própria (2022).

Além da criação da atividade, foi prototipada a tela da Figura 74. Nesta tela o usuário pode editar os dados da atividade. Foi necessária a criação deste protótipo

pensando que novos beneficiários podem se cadastrar na atividade, ou alguns detalhes na descrição ou até mesmo a data de realização podem mudar inesperadamente.

O RQF8 – presença dos beneficiários necessitou um entendimento de como são registradas as presenças sem o uso do *software*. O cliente explicou que o coordenador registra a quantidade de faltas mensalmente e então é calculado a porcentagem de presença do usuário a partir da quantidade de atividades realizadas e de faltas.

Figura 75 – Protótipo da tela de presença de beneficiários



Fonte: Autoria própria (2022).

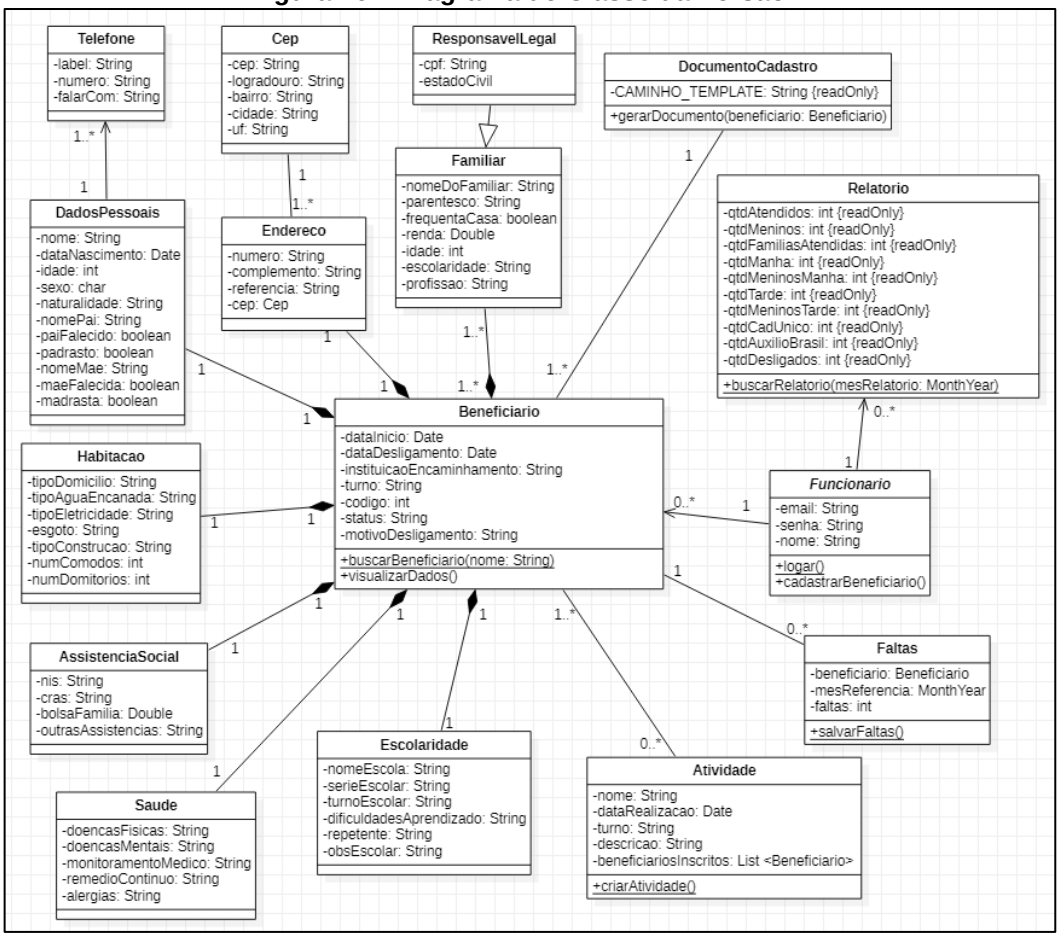
A tela da Figura 75 mostra a tela prototipada para o requisito RQF8. O usuário poderá selecionar o mês e o ano de referência para o registro das faltas dos beneficiários. Usou-se uma tabela que conterá os nomes dos beneficiários e o campo falta será preenchido com o número de faltas. Além disso, caso o usuário necessite alterar a quantidade de faltas de um beneficiário específico, ele pode realizar a pesquisa por nome. Os protótipos foram apresentados à Casa do Piá, que os aprovou.

3.4.2 Projeto de Software

Esta versão teve alterações tanto no Diagrama de Classe quanto no DER. Novas classes foram criadas para o registro de atividades e de faltas dos beneficiários,

utilizando as classes criadas em versões anteriores somente em seus atributos. A Figura 76 apresenta o Diagrama de Classe atualizado.

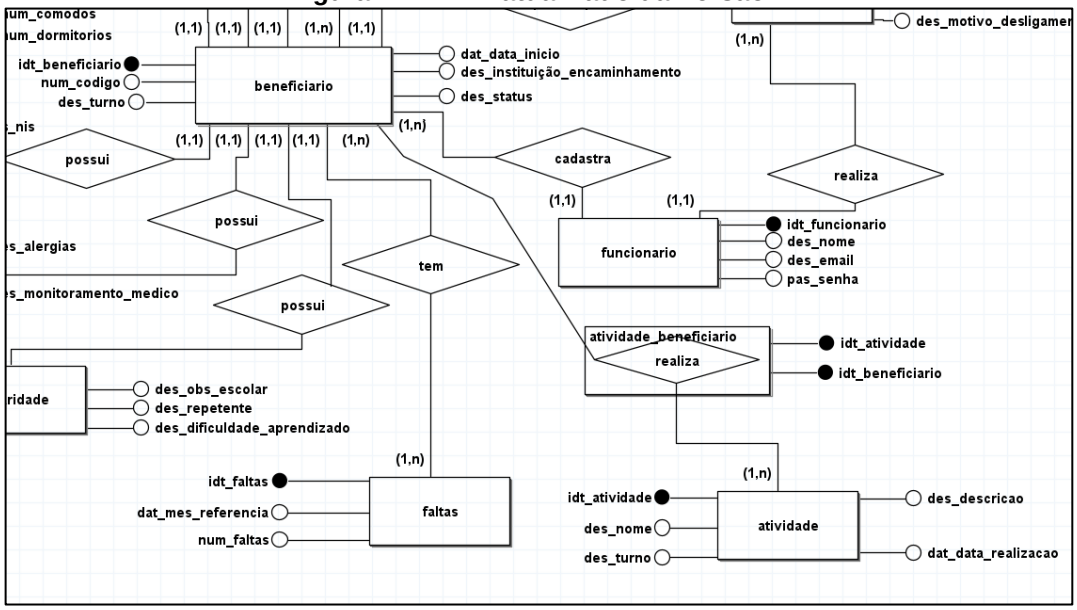
Figura 76 – Diagrama de Classe da Versão 4



Fonte: Autoria própria (2022).

As classes “Atividade” e “Faltas” foram adicionadas ao diagrama para que os requisitos RQF7 e RQF8 sejam implementados, respectivamente. A classe “Atividade” tem o método estático “criarAtividade”, responsável por instanciar e salvar o objeto desta classe.

Figura 77 – DER atualizado da Versão 4



Fonte: Autoria própria (2022).

A Figura 77 mostra o DER cortado, com foco nas entidades que foram criadas e seus relacionamentos. A entidade “faltas” tem o relacionamento com o “beneficiario”, pois é onde as faltas desta outra entidade que serão apresentadas. A entidade “atividade” se relaciona com “beneficiario” através de uma entidade associativa, como resultado da normalização do relacionamento “n para n” entre as duas entidades. Este relacionamento se dá devido à lista de beneficiários que é inscrito em uma atividade.

Não houveram alterações resultantes das correções pedidas pelo cliente na versão anterior. Esta funcionalidade será implementada apenas com uma consulta que será utilizada na nova tela.

3.4.3 Desenvolvimento

Primeiramente foram implementadas as correções pedidas pela Casa do Piá. As buscas implementadas na versão anterior já faziam um filtro para não retornar os beneficiários inativos e, verificou-se que esta busca poderia facilmente ser convertida para retornar os inscritos com o status desejado, sendo ele “ativo” ou “inativo”.

Figura 78 – Método de busca de beneficiários por status

```
public List<Beneficiario> getBeneficiariosAtivos() {
    return beneficiarioRepositorio.findByStatus("ativo") List<BeneficiarioEntity>
            .stream() Stream<BeneficiarioEntity>
            .map(BeneficiarioMapper::fromBeneficiarioEntity) Stream<Beneficiario>
            .collect(Collectors.toList());
}
```

```
public List<Beneficiario> getBeneficiariosByStatus(boolean ativo) {
    String status = ativo ? "ativo" : "inativo";
    return beneficiarioRepositorio.findByStatus(status) List<BeneficiarioEntity>
            .stream() Stream<BeneficiarioEntity>
            .map(BeneficiarioMapper::fromBeneficiarioEntity) Stream<Beneficiario>
            .collect(Collectors.toList());
}
```

Fonte: Autoria própria (2022).

A Figura 78 mostra as alterações que o método de busca passou para cumprir os requisitos desta nova versão. O trecho de código acima da divisória é o método antes da adaptação desta versão, e abaixo da divisória é o resultado. A busca passa a receber um parâmetro *booleano* “ativo”, que representa se os beneficiários buscados serão os ativos, alterando status que será buscado no banco de dados.

Para que as atividades e faltas sejam salvas, criou-se uma estrutura similar ao implementado para a classe “Beneficiario”. A classe “Atividade” tem uma classe de entidade, “AtividadeEntity”, que a mapeia para o banco de dados, uma classe de serviços que é chamada pelo controlador da tela, e uma classe repositório, apresentada na Figura 79, para fazer as chamadas ao banco de dados.

Figura 79 – Classe de repositório de atividades

```
public interface AtividadeRepositorio extends JpaRepository<AtividadeEntity, Long> {

    List<AtividadeEntity> findAtividadeEntityOrderByDataRealizacao();
}
```

Fonte: Autoria própria (2022).

A classe apresentada estende a interface “JpaRepository”, que por sua vez é implementada pelo Hibernate. Foi criado o método com o padrão de nome esperado pelo *framework* “findAtividadeEntityOrderByDataRealizacao”, que busca todas as atividades do banco de dados e as ordena pela data de realização da atividade. O método para salvar um objeto de “Atividade” já é gerado automaticamente pelo Hibernate também.

Figura 80 – Mapeamento de entidade associativa com Hibernate

```
@ManyToMany
@JoinTable(
        name = "atividade_beneficiario",
        joinColumns = { @JoinColumn(name = "idt_atividade") },
        inverseJoinColumns = { @JoinColumn(name = "idt_beneficiario") }
)
private List<BeneficiarioEntity> beneficiarios;
```

Fonte: Autoria própria (2022).

A Figura 80 mostra como é feito o mapeamento do Hibernate para a criação de uma entidade associativa no banco de dados. A anotação “@JoinTable” indica que há uma tabela representando esse relacionamento entre duas entidades, enquanto as anotações “@JoinColumn” indicam a chave estrangeira utilizada para o relacionamento das entidades.

Figura 81 – Busca de atividades do beneficiário em um mês

```
@Query(nativeQuery = true, value = "SELECT COUNT (*) " +
        "FROM atividade_beneficiario ab " +
        "INNER JOIN atividade a ON a.idt_atidade = ab.idt_atividade" +
        "WHERE a.dat_data_realizacao>= :primeiroDia " +
        "AND a.dat_data_realizacao <= :ultimoDia" +
        "AND ab.idt_beneficiario = :beneficiario.id")
Integer countAtividadeEntitiesByBeneficiarios(LocalDate primeiroDia,
                                              LocalDate ultimoDia,
                                              BeneficiarioEntity beneficiario);
```

Fonte: Autoria própria (2022).

Por fim, as faltas seguem a estrutura padrão apresentada anteriormente. Para que seja calculado a porcentagem de faltas de um beneficiário em um mês, é utilizado o método da Figura 81 para contar a quantidade de atividades realizadas. Também é contada a quantidade de faltas num método similar ao apresentado para que o calculo seja realizado.

3.4.4 Feedback da Casa do Piá

Ao fim da implementação, o *software* realiza o cadastro de inscritos que podem ser editados, ter o seu documento de inscrição gerados, ou serem desligados. As atividades podem ser cadastradas no sistema contendo beneficiários associados a elas. As faltas dos beneficiários são contabilizadas no sistema, que também calcula a porcentagem de faltas por atividades realizadas no mês.

Foi gerado então, o arquivo executável e enviado à Casa do Piá, que teve um tempo para testá-lo. Após o período de teste, foi recebido o *feedback* de que o *software* supriu as necessidades elicitadas no escopo do sistema.

4 CONCLUSÃO

Este trabalho propôs o desenvolvimento de um sistema que gere documentos para a Casa do Piá, além de armazenar e proteger os dados dos usuários atendidos pela instituição, evita um processo que sem o uso do *software* é laborioso e repetitivo.

Para o desenvolvimento do trabalho, aplicou-se o fluxo de processo do modelo espiral. Ao início de cada ciclo foi realizada uma comunicação com o cliente, na qual foram obtidos requisitos e/ou correções que foram documentados, priorizados, prototipados, validados, modelados e finalmente implementados nas etapas seguintes deste ou de ciclos posteriores.

Para a implementação do sistema fez-se uso de frameworks e ferramentas que auxiliaram na codificação de um *software* de código fonte limpo, focado na manutenibilidade e de qualidade de *software*. Para a implantação do *software*, foi utilizada uma ferramenta para criar um instalador personalizado, facilitando esta tarefa para o utilizador da aplicação.

Notou-se no desenvolvimento deste trabalho que o uso de princípios da engenharia de requisitos mitigou diversos erros que poderiam ter sido encontrados apenas ao fim da implementação do *software*. Durante a elicitação de requisitos, a padronização das funcionalidades desejadas pelo cliente facilitou a escolha dos requisitos a serem implementados em cada versão. Enquanto o uso da prototipagem ajudou os clientes a validarem em fases iniciais de cada versão se as suas expectativas estavam atendidas.

Durante a implementação do *software* também foi notado que muitas estruturas e padrões criados em versões anteriores puderam ser reutilizados. Isto agilizou a implementação de novas funcionalidades em cada versão. Tal reaproveitamento ainda aconteceria caso toda a implementação do sistema fosse realizada em uma única versão, porém, as atualizações constantes nos diagramas e o foco apenas nas funcionalidades escolhidas ciclo a ciclo, otimizaram a percepção e a estruturação para o reaproveitamento de código fonte.

Além disso, a aplicação do modelo espiral para o desenvolvimento do *software* proporcionou um contato constante com a Casa do Piá, ampliando os benefícios das etapas de engenharia de *software*. O cliente pôde experienciar as mudanças sofridas pelo sistema e manifestar mais claramente as suas necessidades.

4.1 Trabalhos futuros

O desenvolvimento desse trabalho satisfez as necessidades da Casa do Piá, porém outros processos poderiam ser automatizados. A seguir são descritas algumas possibilidades de trabalhos futuros:

- Implementação de módulos como: controle financeiro; controle de funcionários; e controle de materiais e recursos;
- Integrar o sistema desenvolvido com sistemas do governo para sincronizar dados já existentes;
- Transformar o trabalho em uma aplicação personalizável e de código livre para que qualquer instituição de assistência social do Brasil pudesse utilizá-la.
- Melhorias na interface de usuário, utilizando conceitos de IHC

REFERÊNCIAS

ANDERSON, D.; CARMICHAEL, A. **Kanban Essencial Condensado**. 2. ed. Seatle: Lean Kanban University Press, 2016.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML**: guia do usuário. Elsevier Brasil, 2006.

BOEHM, B. W. **A Spiral Model of Software Development and Enhancement**. IEEE Computer, v. 21, n. 5, 1988, p. 61-72

BRASIL. **Lei n. 8.742 de 7 de dezembro de 1993**. Dispõe sobre a organização da Assistência Social e dá outras providências. Disponível em: http://www.planalto.gov.br/ccivil_03/leis/l8742.htm. Acesso em: 10 set. 2020. (Lei Federal).

CENTRO Social Casa do Piá. [S. l.], 1 jan. 2014. Disponível em: http://irsc.org.br/casa-do-pia/conheca.php. Acesso em: 18 jun. 2020.

CHRISTEL, M. G.; KANG, K, C. **Issues in Requirements Elicitation**. Software Engineering Institute, CMU/SEI-92-TR-12 7, Setembro de 1992.

EASYBACKLOG. **Ferramenta EasyBacklog**. [S.l.]: EasyBacklog, 2021. Disponível em: https://easybacklog.com. Acesso em 07 dez. 2021.

GAMMA, E. *et al*. **Padrões de Projetos:** soluções reutilizáveis de software orientado a objetos. Grupo A, 2011. 9788577800469. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788577800469/. Acesso em: 05 dez. 2021.

GUEDES, Gilleanes TA. **UML 2-Uma abordagem prática**. Novatec Editora, 2011.

JUNIOR N.; AFONSO, A. **Produtividade no Desenvolvimento de Aplicações Web com Spring Boot.** AlgaWorks Softwares, Janeiro de 2017.

LOMBOK. **Framework Lombok**. [S.l.]: Lombok, 2021. Disponível em: https://projectlombok.org. Acesso em 07 dez. 2021.

MKLABS. **Ferramenta StarUml.** [S.l.]: MKLabs, 2021. Disponível em: https:// https://staruml.io. Acesso em 07 dez. 2021.

PONTA GROSSA (PR). **Resolução n. 2 de 20 de fevereiro de 2020**. Dispõe sobre a regulamentação da documentação para manutenção das inscrições de entidades e serviços socioassistenciais no CMAS. Disponível em: https://cmas.pontagrossa.pr.gov.br/wp-content/uploads/2020/03/Resolu%C3%A7%C3%A3o-02-documentos-manuten%C3%A7%C3%A3o-inscri%C3%A7%C3%A3o-CMAS-2020.pdf. Acesso em: 15 out. 2022. (Lei Municipal).

PFLEEGER, S. L. **Software Engineering:** Theory and Practice. 4. ed. Nova Jersey, EUA: Pearson Higher Education, 2010.

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

RUSSEL, J. **Ferramenta Inno Setup.** [S.l.]: Jordan Russel, 2021. Disponível em: https://jrsoftware.org/isinfo.php. Acesso em: 07 dez. 2021.

SILBERSCHATZ, A. **Sistema de Banco de Dados**. Grupo GEN, 2020. 9788595157552. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788595157552/. Acesso em: 10 nov. 2021.

SOFTWARE. In: DICIO, Dicionário Online de Português. Porto: 7Graus, 2021. Disponível em: https://www.dicio.com.br/software/. Acesso em: 20 out. 2021.

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

VMWARE, Inc. **Spring:** *Core Technologies*. [S.l.]: VMware, 2021. Disponível em: https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-introduction. Acesso em: 07 dez. 2021

YASUHIRO, M. **Sistema Toyota de Produção**: Grupo A, 2015. 9788582602164. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788582602164/. Acesso em: 17 Mar 2021.

ANEXO A – Modelo do Documento de Cadastro da Casa do Piá

1	**2.1 Aliança Brasileira de Assistência Social e Educacional** **Mantenedora do Projeto Casa do Piá** **Rua Maurício de Nassau, 560. CEP 84.070.330 - Vila Madureira** **2.1.1.1 Ponta Grossa-PR - Fone/Fax (042) 3027.6070** **CNPJ - 62.207.634/0013-00 - CNSS 44513/67**

Inscrição das crianças e/ou adolescentes inseridos no projeto.

INSCRIÇÃO NÚMERO:	**001/21**

1-Identificação do atendido:										
Nome:										
Data de nascimento:				Idade:						
Sexo:	Masculino		Feminino		Naturalidade:					
Nome do pai:										
Nome da mãe:										
1.1-Endereço:										
Rua:							N°:			
bairro/Vila:			Ponto de referência:							
Telefone:		Telefone/recados:			Falar com:					
Telefone/recados:				Falar com:						

2- Aspectos familiares:									
Nome do responsável Legal:									
Relação com o atendido:									
CPF		Idade:		Estado Civil:	Solteiro:		Casado:		
União estável:		Viúvo							

2.1- Com quem mora: * grau de parentesco partindo da visão do atendido.

Nome.	Idade.	Grau de parentesco	Possui irmãos que freqüentam a entidade?	Escolaridade:	Profissão:	Renda mensal:

2.1 Aliança Brasileira de Assistência Social e Educacional

Mantenedora do Projeto Casa do Piá

Rua Maurício de Nassau, 560. CEP 84.070.330 - Vila Madureira

2.1.1.1 Ponta Grossa-PR - Fone/Fax (042) 3027.6070

CNPJ - 62.207.634/0013-00 - CNSS 44513/67

3-Habitação:													
Domicílio	Particular permanente:		Particular provisório:		Particular coletivo:		Outros:						
Possui	Água encanada particular:		Luz elétrica particular:		Luz elétrica medidor coletivo:								
Água encanada medidor coletivo:		Possui rede de esgoto		Sim:		Não:							
Edificação	Alvenaria:		Madeira:		Misto:		Outros:						
Número de cômodos:		Quantos cômodos são dormitórios:											

4-Assistência Social:							
Número do NIS Familiar:							
CRAS de Referência:							
Recebe Bolsa Família:	Sim:			Não:			
A família frequenta ou recebe auxilio de outro órgão da rede socioassistencial:		Sim:			Não:		

5-Saúde:				
Possui problemas de saúde física diagnosticados:	Sim:		Não:	
Possui problemas de saúde mental diagnosticados:	Sim:		Não:	
Faz algum acompanhamento profissional:	Sim:		Não:	
Faz uso continuo de medicamentos:	Sim:		Não:	

Obs:

1	**2.1 Aliança Brasileira de Assistência Social e Educacional** **Mantenedora do Projeto Casa do Piá** **Rua Maurício de Nassau, 560. CEP 84.070.330 - Vila Madureira** **2.1.1.1 Ponta Grossa-PR - Fone/Fax (042) 3027.6070** **CNPJ - 62.207.634/0013-00 - CNSS 44513/67**

6-Escolaridade:							
Escola:							
Série de 1 a 9 anos*:		Turno:	Matutino:		Vespertino:		
Tem dificuldades de aprendizagem?	Sim:					Não:	

7- Ingresso na Casa do Piá.								
Data de início:				Data desligamento:				
Compareceu a instituição por encaminhamento:					CRAS:		CREAS:	
Conselho Tutelar:		Procura espontânea:		Outros:				
Turno que vai frequentar a Casa do Piá:				Matutino:		Vespertino:		

8-Termo de compromisso:		
Eu		Comprometo-me a acompanhar
		Durante o período em que o mesmo estiver
inscrito na Casa do Piá, respeitando as normas do regimento interno, participando das reuniões, **mantendo**		
contato direto e imediato com a coordenação e serviço social para comunicar qualquer fato que		
interfira diretamente no atendimento da criança/adolescente inclusive comunicar desistência, ou		
quando for solicitado minha presença na entidade.		
Ponta Grossa		
Assinatura do responsável:		

1	**2.1 Aliança Brasileira de Assistência Social e Educacional** **Mantenedora do Projeto Casa do Piá** **Rua Maurício de Nassau, 560. CEP 84.070.330 - Vila Madureira** **2.1.1.1 Ponta Grossa-PR - Fone/Fax (042) 3027.6070** **CNPJ - 62.207.634/0013-00 - CNSS 44513/67**

	Carimbo e assinatura do Assistente Social responsável pela inscrição:

Recorte aqui

PARECER DO ASSISTENTE SOCIAL:

CONTRA REFERÊNCIA (encaminhada ao CRAS via Email)

De: Centro Social Casa do Piá**. AO:**

Nome do Responsável familiar atendido:

NIS:

Serviço Ofertado:

Atendimento realizado em:

Resumo do Procedimento:

Responsável: