INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SUL-RIO-GRANDENSE – IFSUL, *CAMPUS* PASSO FUNDO
CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

LUANA CAROLINE GIRARDI BASSO

SYSADMIN: SISTEMA DE ADMINISTRAÇÃO LAR EMILIANO LOPES

Jorge Luis Boeira Bavaresco

PASSO FUNDO, 2014

LUANA CAROLINE GIRARDI BASSO

SYSADMIN: SISTEMA DE ADMINISTRAÇÃO LAR EMILIANO LOPES

Monografia apresentada ao Curso de Tecnologia em Sistemas para Internet do Instituto Federal Sul-Rio-Grandense, *Campus* Passo Fundo, como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet.
Orientador: Jorge Luis Boeira Bavaresco

PASSO FUNDO, 2014

LUANA CAROLINE GIRARDI BASSO

SYSADMIN: SISTEMA DE ADMINISTRAÇÃO LAR EMILIANO LOPES

Trabalho de Conclusão de Curso aprovado em ____/____/____ como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet

Banca Examinadora:

_____

Prof. Jorge Luis Boeira Bavaresco
Orientador

_____

Prof. Dr. Alexandre Tagliari Lazzaretti
Avaliador

_____

Prof. Me. Roberto Wiest
Avaliador

_____

Prof. Dr. Alexandre Tagliari Lazzaretti
Coordenador do Curso

PASSO FUNDO, 2014

À minha mãe,
pela compreensão e o estímulo
em todos os momentos.

AGRADECIMENTOS

Agradeço, primeiramente, a Deus, por sempre iluminar meu caminho, mostrando que, apesar das dificuldades, tudo acontece em seu tempo certo.

Agradeço à minha mãe, Ieda, por sempre ter me apoiado em busca dos meus objetivos e pelo seu esforço para que eu seguisse meus estudos.

Ao Lar Emiliano Lopes, pela confiança a mim concebida para que esse projeto se tornasse realidade.

Agradeço a todos os professores do Curso Superior de Tecnologia em Sistemas para Internet do IFSul, em especial ao professor Alexandre T. Lazzaretti pelo apoio e pelas oportunidades desde o início do curso, ao meu orientador, Jorge Luis Boeira Bavaresco pelo apoio e confiança durante o desenvolvimento desse trabalho. Aos meus colegas que me acompanharam desde o início do curso, Marina, Alisson, Daniel e Vanessa, pela amizade, paciência e compreensão ao longo desse período.

Por fim, agradeço a todos que, de alguma forma, contribuíram na construção desse projeto.

“A persistência é o menor caminho do êxito”.

Charles Chaplin

RESUMO

O presente trabalho expõe os passos realizados para o desenvolvimento de um protótipo de sistema para uma instituição acolhedora na cidade de Passo Fundo, RS, que realize o cadastro de acolhidos, trazendo uma alternativa ao método manual atualmente utilizado. Através do protótipo, demonstra-se a inserção, edição e exclusão de adolescentes, acolhimentos, visitas e familiares. Para seu desenvolvimento, a tecnologia Java Server Faces foi utilizada, juntamente com a biblioteca Primefaces.

Palavras-chave: Instituição; Protótipo; Java Server Faces; Primefaces.

ABSTRACT

The present paper shows the steps done to the development of a prototype of a system to a host institution in the city of Passo Fundo, RS, that performs the registration of hosted, introducing an alternative to the manual method used currently. Through the prototype, it is shown the insert, edition and exclusion of teenagers, hostings, visits and family. To the development, Java Server Faces technology was used along with Primefaces library.

Keywords: Institution; Prototype; Java Server Faces; Primefaces.

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

CRUD – *Create, Read, Update, Delete* – Criar, Ler, Atualizar, Deletar
EJB – *Enterprise JavaBeans*
HTTP - *Hypertext Transfer Protocol* - Protocolo de Transferência de Hipertexto
HTTPS - *Hypertext Transfer Protocol Secure* - Protocolo de Transferência de Hipertexto Seguro
JDBC – *Java Database Connectivity* – Conexão de Banco de Dados Java
JPA – *Java Persistence API* – API de Persistência Java
JPQL – *Java Persistence Query Language* – Linguagem de Persistência de Consultas Java
JSF – *Java Server Faces*
JTA – *Java Transaction API* – API de Transações Java
MVC – *Model-View-Controller* – Modelo-Visão-Controle
PHP - PHP: *Hypertext Preprocessor*
ODBC - *Open Database Connectivity* – Conectividade aberta ao banco de dados
SQL – *Structured Query Language* – Linguagem de Consultas Estruturada
TCL – *Tool Command Language* – Linguagem de Comandos de Ferramentas
XML – *Extensible Markup Languege* – Linguagem Extensível de Marcação

SUMÁRIO

1 INTRODUÇÃO

No atual mundo informatizado, empresas e instituições possuem necessidade de armazenar cada vez mais dados e de forma mais rápida. Essa demanda, que se torna cada vez maior, faz com que seja imprescindível uma melhor organização dos processos que se tornam sempre mais complexos. Anotações feitas de forma manual funcionam por determinado tempo, mas chega um momento em que é preciso um controle mais rígido sobre a documentação, e então surge a conveniência de sistemas informatizados, que armazenem de forma segura os procedimentos e dados das pessoas envolvidas no processo.

O Lar Emiliano Lopes é uma instituição acolhedora de crianças e adolescentes que conta com acolhidos, funcionários e diretor. O cadastro necessário para acolhimento e permanência dos adolescentes na entidade, além dos relatórios, são criados e atualizados de forma manual, o que consome muito tempo, além de haver a possibilidade de armazenamento de dados imprecisos, já que não há nenhuma forma de se garantir a precisão dos mesmos.

Logo, o presente projeto tem como objetivo a criação, a partir de uma linguagem de programação, do protótipo de um sistema para a entidade acima citada. O protótipo do software deverá permitir o cadastro de acolhidos, além de possuir interface amigável.

A escolha por abordar este tema deu-se pelo fato de a entidade não possuir nenhum tipo de sistema informatizado e, caso o projeto seja aprovado pela instituição, irá trazer grande facilidade no cadastro e posterior controle dos acolhidos.

1.1 OBJETIVOS

Aqui, serão apresentados os objetivos geral e específicos do trabalho.

1.1.1 Objetivo Geral

Desenvolver o protótipo de um software administrativo para uma instituição que realize o cadastro de acolhidos.

1.1.2 Objetivos Específicos

- realizar um estudo sobre a tecnologia Java Server Faces;
- desenvolver um protótipo de um sistema de acolhimentos fazendo uso das tecnologias estudadas.

1.2 ORGANIZAÇÃO DA MONOGRAFIA

O presente trabalho encontra-se dividido em capítulos, sendo que o capítulo 2 aborda o referencial teórico, contendo os principais conceitos referentes às tecnologias utilizadas na realização do sistema, o capítulo 3 contém os passos realizados para a implementação do protótipo, seguidos das considerações finais e referências bibliográficas.

2 REFERENCIAL TEÓRICO

No presente capítulo encontram-se os principais conceitos utilizados durante a realização do trabalho, baseados nas principais literaturas da área.

2.1 PADRÃO

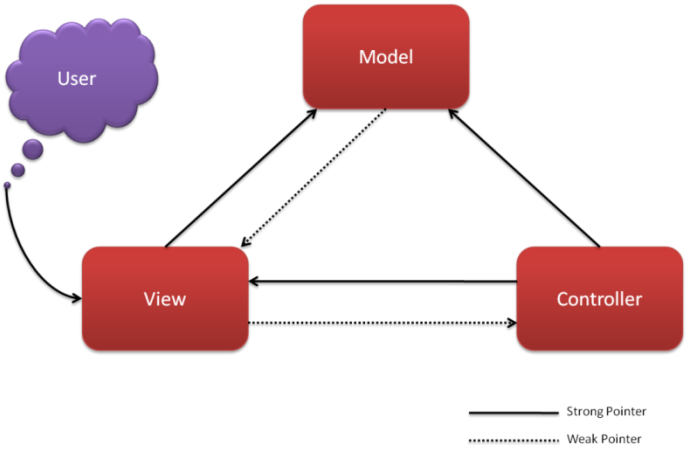
Padrões surgiram com o objetivo de fornecer design reutilizável com restrições em seu uso. Um de seus principais benefícios é o de descrever uma solução generalizada para problemas recorrentes.

O padrão MVC (*Model-View-Controller*) se popularizou devido ao fato de sua principal característica ser a de fornecer componentes reutilizáveis para interfaces interativas, separando as aplicações em três componentes distintos: modelo, visão e controle. Esta separação oferece uma grande vantagem, pois a parte mais volátil de uma aplicação é a interface com o usuário, já que é através dela que eles se comunicam diretamente com o software, e o padrão MVC dissocia os componentes modelo, visão e controle de uma aplicação, fornecendo uma interface uniforme entre eles. (DUDNEY et al., 2004).

O autor descreve os três componentes deste:

- modelo: é considerado o núcleo da aplicação, pois representa o estado e a lógica da mesma. Não possui conhecimento sobre o controle e a visão;
- visão: fornece representações dos dados e comportamento da aplicação e disponibiliza os dados produzidos pelo modelo, gerenciando o que pode ser visto. É a única parte da aplicação com que o usuário interage diretamente;
- controle: é responsável por gerenciar a interação do usuário e do sistema com o modelo, fornecendo o mecanismo pelo qual as mudanças são lá realizadas.

Figura 1 - Padrão MVC



Fonte: http://sngo.me/J9Shx

Quando o usuário interage com o sistema através da camada de visão, eventos são disparados para processamento em um ou mais controles. Se o evento requerer mudanças na camada de modelo, o controle irá manipular os dados lá contidos ou invocar operações específicas. Se o evento requerer mudanças em outros componentes da interface, um controlador irá manipulá-lo diretamente, adicionar novos componentes ou esconder componentes já existentes (DUDNEY et al., 2004).

2.2 JAVA EE

No atual mundo competitivo, percebe-se a constante necessidade de aplicações para acessarem dados e se comunicarem com sistemas externos. Tudo isso deve ser feito com custos reduzidos e uso de tecnologias robustas e que sigam um padrão. *Java Platform Enterprise Edition*, ou simplesmente Java EE surgiu no fim dos anos 1990 trazendo para a linguagem Java uma plataforma robusta para desenvolvimento de softwares (GONCALVES, 2010).

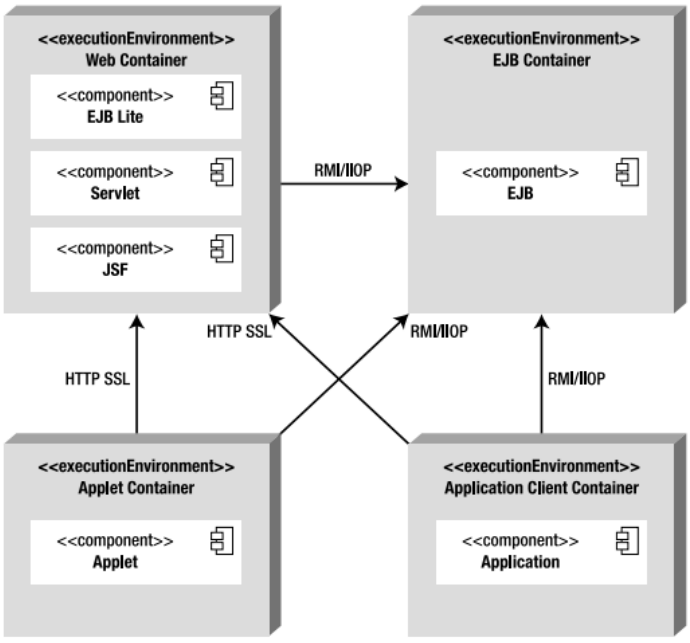
Jendrock et al. (2013), define uma aplicação *enterprise* como aquela que fornece a lógica de negócio para uma empresa. São aplicações gerenciadas centralmente e que normalmente interagem com software de outras empresas. Ainda explica que a plataforma Java EE utiliza um modelo de programação simplificado, pois o desenvolvedor pode simplesmente colocar as informações em forma de anotações diretamente no arquivo java e o servidor Java EE configurará os componentes em tempo de execução, tornando os descritores de implementação XML (*Extensible Markup Language*) opcionais.

A Java EE fornece um padrão para desenvolvimento web e aplicações. Essas aplicações geralmente são multicamadas, com uma camada frontal consistindo de *frameworks* web, uma camada intermediária com as transações e segurança, e a última camada fornecendo conectividade a um banco de dados ou sistema legado. É uma plataforma que define APIs para diferentes componentes, além de alguns serviços adicionais como injeção de dados. (GUPTA, 2012).

Segundo Goncalves (2010), Java EE fornece uma forma padrão para lidar com: transações, com *Java Transaction API* (JTA), mensagens, com o uso de *Java Messages Service* (JMS), e persistência, utilizando o *Java Persistence API* (JPA).

A arquitetura Java EE é, segundo Goncalves (2010), um conjunto de especificações implementado por diferentes *containers*, que são ambientes de execução que fornecem serviços para os componentes por eles hospedados. Os componentes se comunicam com a infraestrutura Java EE e com outros componentes, que precisam ser empacotados de uma maneira padrão. A figura 1 mostra a relação entre *containers*.

Figura 2 – Arquitetura Java EE, composta por c*ontainers*



Fonte: GONCALVES (2010)

Quatro tipos de componentes devem ser implementados por uma aplicação: *Applets* (aplicações GUI executadas em um *browser*), Aplicações (programas executados em um cliente), aplicações Web (executadas em um *container* web e que respondam requisições

HTTP de clientes web) e Aplicações *Enterprise* (executadas em um container EJB). Containers são domínios lógicos que possuem funções específicas, sustentam um conjunto de APIs e oferecem serviços para componentes. Escondem complexidades técnicas e aumentam portabilidade. Containers *Applet* são fornecidos pela maioria dos web *browsers* para executar componentes *Applet*. Usam um *sandbox* de modelo de segurança em que o código que é executado dentro do *sandbox* não é permitido a ser executado fora do mesmo. O *container* de aplicação do cliente contém um conjunto de classes Java, bibliotecas e outros arquivos requeridos para injeção, segurança de gerência e serviço de nomes para aplicações Java SE. O *container* web fornece serviços básicos para gerenciamento e execução de componentes web. É utilizado para alimentar páginas web para *browsers* do cliente, além de ser responsável por instanciar, inicializar e invocar *servlets* e suportar os protocolos HTTP (*Hypertext Transfer Protocol*) e HTTPS (*Hypertext Transfer Protocol Secure*). O *container* EJB se responsabiliza por gerenciar a execução do *enterprise beans* contendo a lógica de serviço para as aplicações Java EE. Além disso, os *containers* fornecem serviços básicos para os componentes implantados (GONCALVES, 2010).

Para serem implantados em um *container*, os componentes devem ser empacotados em um arquivo formatado de maneira padrão.

2.3 JSF

Em 2004, a Sun Microsystems apresentou um *framework* web Java chamado *Java Server Faces*, ou simplesmente JSF. O objetivo deste *framework* era simplificar o desenvolvimento de aplicações web, e ele surgiu como uma evolução do *framework Java Server Pages* (JSP), adicionando a possibilidade de utilizar tecnologias mais modernas com maior facilidade. Além disso, JSF aderiu à arquitetura MVC, pois utiliza arquivos XML para a construção da visão e classes Java para a lógica da aplicação (JUNEAU, 2013).

Geary e Horstmann (2010) definem *Java Server Faces* como a especificação de um *framework* de componentes para desenvolvimento web em Java. Participaram de seu desenvolvimento empresas como IBM, Oracle, Apache, entre outras. Sua primeira versão, JSF 1.0, foi lançada em 2004. Esta primeira versão teve alguns problemas e logo foram lançadas as versões 1.1 e 1.2. Porém, sua especificação original não foi muito bem aceita, sendo acusada de ser muito genérica, o que a tornava pouco interessante na prática. Por outro lado, possuía a vantagem de ser altamente extensível, o que a tornou atrativa para

desenvolvedores de frameworks. Estes desenvolvedores criaram códigos abertos que podiam ser “ligados” ao JSF. Então, a sua segunda versão, JSF 2.0, foi desenvolvida em cima desta experiência com os desenvolvedores de *framework* e, ao contrário da primeira versão, foi muito bem aceita por ser simples e mais fácil de ser integrada com as demais tecnologias Java EE.

JSF possui fácil integração com tecnologias como Ajax e trabalha bem com bancos de dados, utilizando JDBC ou EJB. Além disso, seu *JavaBeans* (componentes reutilizáveis de *software* que podem ser manipulados visualmente com a ajuda de uma ferramenta de desenvolvimento), que é conhecido como JSF *managed beans*, é utilizado na lógica da aplicação e oferece suporte a conteúdos dinâmicos em todas as visões (JUNEAU, 2013).

Segundo Bergsten (2009), *Java Server Faces* é a tecnologia Java mais avançada para aplicações web, e foi desenvolvida baseada nas experiências anteriores adquiridas com *Java Servlets, Java Server Pages* e *frameworks* para aplicações web. Ela simplifica o desenvolvimento de aplicações web sofisticadas, definindo, primeiramente, um modelo de componentes de interface do usuário junto com o processamento do ciclo de vida da requisição. Isso permite que os programadores desenvolvam suas aplicações sem precisarem se preocupar com detalhes relacionados à HTTP e possam integrá-las com a interface do usuário de maneira simples.

Ainda segundo Bergsten (2009), JSF é uma especificação contendo implementações, que define um conjunto de componentes de interfaces do usuário, assim como uma API para estender os componentes padrão ou criar novos componentes. JSF não é limitada a nenhuma linguagem de marcação e todas as suas implementações devem suportar *Java Server Pages* (JSP) como uma camada de apresentação, com seus componentes representados por elementos de ação customizados JSP. Porém, JSF é capaz de suportar outras tecnologias de apresentação além do JSP, a que o torna altamente flexível.

Juntamente com o JSF, pode-se fazer uso de bibliotecas, que fornecem recursos adicionais aos padrões já definidos pela tecnologia. As bibliotecas mais conhecidas são: *Primefaces*, *RichFaces*, *ICEFaces* e *Tomahawk*. Dentre as quatro, *Primefaces*, que será detalhada a seguir, foi a escolhida para ser utilizada no desenvolvimento do sistema.

2.3.1 Primefaces

Primefaces é uma biblioteca de componentes JSF, disponível a partir da segunda versão do Java Server Faces, contendo apenas um arquivo .jar, que não necessita de qualquer configuração e não possui dependências externas. Possui uma grande variedade de componentes com suporte a Ajax, como painéis de layout, botões, tabelas, calendários, entre outros. Todos esses componentes podem ser utilizados com alto grau de usabilidade, flexibilidade e interatividade. Além de tudo isso, também possui suporte para aplicações em dispositivos móveis (HAVLAT, 2013).

Figura 3 – ContextMenu Primefaces

Fonte: Showcase Primefaces

Feitosa (2010), diz que o *framework* possui diversas vantagens, entre as quais diversas possibilidades de criação de *layouts* para aplicações web, evitando a necessidade de uso de outros componentes. Além disso, possui alta facilidade de uso, a versão *Mobile* UI, que é destinada a aplicações web móveis, além de seu amplo conjunto de componentes.

2.4 JPA

A maior parte dos dados manipulados em uma aplicação necessita ser armazenado em um banco de dados. São chamados de dados persistentes aqueles deliberadamente armazenados de formas permanentes em memória *flash*, mídias magnéticas, entre outros. Um objeto que consiga armazenar seu estado para ser reusado posteriormente também é dito persistente (GONCALVES, 2010).

Segundo Goncalves (2010), em Java existem diferentes formas de se persistir um objeto. Uma delas é através da API padrão para acesso a bancos de dados relacionais, o JDBC

(*Java Database Connectivity*), que pode se conectar a um banco de dados, executar comandos SQL (*Structured Query Language*) e retornar um resultado. Outra forma de persistência de dados é através do mecanismo de serialização, que consiste no processo de converter um objeto em uma sequência de *bits*. Para utilizar este processo, basta implementar a interface *java.io.Serializable*. Porém, nenhuma dessas formas consegue abranger todos os requisitos necessários para uma infraestrutura completa de persistência. Então, como solução para os problemas com persistência foi criada, juntamente com o Java EE 5, a *Java Persistence API*, ou simplesmente JPA 1.0, que traz os modelos relacional e orientado a objetos integrados. A partir desta API é possível acessar e manipular dados relacionais de EJBs, componentes *web* e aplicações Java SE. Todas as suas classes e anotações estão contidas no pacote *javax.persistence*. A JPA 2.0 foi lançada juntamente com o Java EE 6 trazendo algumas novas funcionalidades.

Goncalves (2010) descreve os principais componentes da JPA:

- uma API *entity manager* para realizar operações CRUD;
- JPQL, ou *Java Persistence Query Language*, que permite a recuperação de dados com o uso de *queries* orientadas a objetos;
- ORM, que é o mecanismo utilizado para mapeamento de dados armazenados no banco de dados;
- *callbacks* e *listeners*, para ligar a lógica de negócio ao ciclo de vida de um objeto persistente;
- mecanismos de transação e bloqueio quando ocorrer acesso concorrente a dados através da *Java Transaction API* (JTA).

Na JPA 2.0, além dos componentes acima citados, foram adicionadas algumas novas funcionalidades, dentre as quais novas APIs e várias outras funcionalidades como:

- o suporte a mapas foi aumentado, de forma que os mapas agora possuem chaves e valores de entidades ou tipos básicos;
- a possibilidade de manter a ordenação de uma persistência, com a anotação *@OrderColumn*;
- uma nova *Criteria* API foi criada para permitir a construção de consultas de uma maneira orientada a objetos;
- a inclusão de suporte para uma nova API de *cache*;
- algumas propriedades no arquivo *persistence.xml* foram padronizadas, de forma a aumentar a portabilidade das aplicações.

2.5 POSTGRESQL

PostgreSQL é um sistema gerencial de banco de dados objeto relacional que iniciou como um projeto chamado *Ingres*, na universidade da Califórnia em 1977. Em 1986, outro time de desenvolvedores prosseguiu com o código do *Ingres*, criando um banco de dados objeto relacional chamado Postgres. Postgres foi renomeado para PostgreSQL em 1996, devido a uma nova aplicação *open source* que aumentou as funcionalidades do software. O projeto PostgreSQL se encontra em constante aperfeiçoamento por um time de desenvolvedores *open source* e colaboradores (DRAKE e WORSLEY, 2011).

Atualmente, o PostgreSQL é considerado o mais avançado banco de dados *open source* do mundo e fornece uma grande variedade de recursos que geralmente só são encontrados em bancos de dados comerciais. Seguem, a seguir, alguns recursos fornecidos pelo PostgreSQL (DRAKE e WORSLEY, 2011):

- altamente extensível: suporte a operadores definidos pelo usuário, funções, acesso a métodos e datas;
- integridade referencial: oferece suporte à integridade referencial, que é utilizada para assegurar a validade dos dados;
- API flexível: esta flexibilidade têm fornecido suporte ao desenvolvimento facilitado para as RDBMS do PostgreSQL. Estas interfaces incluem Object Pascal, Python, Perl, PHP (PHP: *Hypertext Preprocessor*), ODBC (*Open Database Connectivity*), Java/JDBC, Ruby, TCL (*Tool Command Language*), C/C++ e Pike; (definir siglas)
- linguagens procedurais: possui suporte para linguagens procedurais internas, incluindo uma linguagem nativa, o PL/pgSQL;
- MVCC (*Multi-Version Concurrency Control*): é a tecnologia que o PostgreSQL utiliza para evitar bloqueios desnecessários, ou seja, um leitor nunca será bloqueado por alguém que está escrevendo no banco de dados. Para isso, o PostgreSQL mantém o controle de todas as transações realizadas pelos usuários do banco de dados, e após pode gerenciar estes registros sem causar problemas de bloqueio ao leitor;
- cliente / servidor: o PostgreSQL utiliza uma arquitetura *process-per-user client/server*, que é um processo que fornece conexões adicionais para cada cliente tentando se conectar ao banco de dados.

2.6 UML

Segundo Booch et al. (2005, p. 13), “a UML (Unified Modeling Language) é uma linguagem-padrão para a elaboração da estrutura de projetos de software. Ela poderá ser empregada para a visualização, a especificação, a construção e a documentação de artefatos que façam uso de sistemas complexos de software”.

Após a criação da primeira linguagem orientada a objetos, a Simula-67, em 1967, surgiram, até meados dos anos 80, muitas outras utilizando a orientação a objetos, como C, C++ e Eiffel. Devido a isso, sentiu-se a necessidade de uma linguagem que modelasse isso, pois as aplicações estavam sempre mais complexas e surgiam mais linguagens orientadas a objetos. Por causa dessa maior complexidade nas aplicações, ficava cada vez mais difícil encontrar uma forma de modelagem que atendesse a todas as necessidades dos usuários. Então, novas gerações desses métodos começaram a surgir, dentre os quais se destacaram o método Booch de Booch, o método OOSE (*Objected-Oriented Software Engineering*) de Jacobson, e o método OMT (*Object Modeling Technique*) de Rumbaugh. Os três métodos eram considerados completos, porém cada um continha seus pontos fortes e fracos. As três linguagens evoluíram separadamente até um certo ponto em que seus autores concluíram que a evolução seria maior se as três linguagens fossem unidas e então, em outubro de 1995 foi lançada a versão 0.8 do Método Unificado, que unia os métodos Booch e OMT. A versão 0.9 da UML, que então já unia os três métodos, foi lançada em junho de 1996. Várias empresas começaram a dedicar recursos com o propósito de trabalhar a favor de uma definição mais completa e forte da UML. A versão 1.0 foi lançada em janeiro de 1997. Atualmente, encontra-se na versão 2.5 (BOOCH et al., 2005).

Ainda, segundo Fowler, “a UML (Unified Modeling Language) é uma família de notações gráficas, apoiada por um metamodelo único, que ajuda na descrição e no projeto de sistemas de *software*, particularmente daqueles construídos utilizando o estilo orientado a objetos (OO)” (2005, p. 25).

Segundo Fowler (2005), a UML é utilizada de três modos. O primeiro modo é para esboço, em que ela é utilizada para ajudar a transmitir alguns aspectos de um determinado sistema. Esta é a forma de utilização mais comum da UML. O segundo modo é para projeto, em que a ideia é construir projetos detalhados e suficientemente completos para que um programador o codifique. A terceira forma é a UML como linguagem de programação, em

que, com o uso de ferramentas CASE que geram código, os desenvolvedores desenham diagramas que são compilados diretamente para o código executável, tornando a UML o código fonte.

Treze tipos de diagramas oficiais são descritos pela UML 2.0. A tabela abaixo mostra estes tipos e classifica-os.

Tabela 1 - Tipos de diagrama oficiais da UML

DIAGRAMA	**OBJETIVO**	**LINHAGEM**
Atividades	Comportamento procedimental e paralelo	UML 1
Classes	Classe, características e relacionamentos	UML 1
Comunicação	Interação entre objetos; ênfase nas ligações	Diagrama de colaboração da UML 1
Componentes	Estrutura e conexão de componentes	UML 1
Estruturas compostas	Decomposição de uma classe em tempo de execução	UML 2
Distribuição	Distribuição de artefatos nos nós	UML 1
Visão geral da interação	Mistura de diagrama de sequência e de atividades	UML 2
Objetos	Exemplo de configurações de instâncias	Extraoficialmente na UML 1
Pacotes	Estrutura hierárquica em tempo de compilação	Extraoficialmente na UML 1
Sequência	Interação entre objetos; ênfase na sequência	UML 1
Máquinas de estado	Como os eventos alteram um objeto no decorrer de sua vida	UML 1
Sincronismo	Interação entre objetos;	UML 2

	ênfase no sincronismo	
Casos de uso	Como os usuários interagem com um sistema	UML 1

Fonte: FOWLER (2005)

Para descobrir o que os usuários de um software querem que um sistema faça, a análise de requisitos é um método interessante, pois utiliza técnicas da UML para modelar estes requisitos. Fowler (2005) descreve os principais diagramas utilizados para uma análise de requisitos eficiente. São eles: diagrama de casos de uso, diagrama de classes, diagrama de atividades e diagrama de estados.

- diagrama de casos de uso: um caso de uso descreve o que um sistema deve fazer ou já está fazendo. Segundo Booch et al. (2005), os casos de uso fornecem uma maneira para que haja uma melhor compreensão comum do sistema entre os desenvolvedores e usuários finais. Eles também devem servir para verificação do sistema à medida que ele evolui durante o desenvolvimento. Por isso, diz-se que o diagrama de casos de uso é um instrumento que acompanha um software do seu início até a sua conclusão;
- diagrama de classes: é, segundo Pilone (2006), utilizado para modelar relações estáticas entre os componentes de um sistema. Um único modelo UML pode conter diversos diagramas de classes mostrando o mesmo sistema com diferentes visões. Mostra uma coleção de elementos do modelo, como as classes, seus atributos, seus métodos, seus tipos e seus relacionamentos;
- diagrama de atividades: é o diagrama utilizado para especificar um determinado caso de uso. É utilizado nos casos em que o caso de uso é de difícil compreensão, em cenários mais complexos;
- diagrama de estados: é o diagrama que captura o ciclo de vida de objetos, distinguindo os estados que um objeto pode ter e quais eventos afetam esses estados. Pode ter um ponto de início e vários pontos de encerramento, e cada classe possui somente um diagrama de estados.

3 DESENVOLVIMENTO DO PROTÓTIPO

Este capítulo apresenta o desenvolvimento de cada uma das etapas realizadas para a implementação do protótipo, desde os diagramas de classes, atividades e casos de uso, até a criação do sistema.

3.1 PROJETO

O trabalho proposto tem como objetivo dinamizar as ações realizadas na instituição, trazendo maior facilidade aos membros da entidade que necessitam realizar os processos manualmente. Esta seção irá apresentar a instituição e descrever os processos realizados atualmente.

O Lar Emiliano Lopes é uma Organização Não Governamental cuja missão é acolher crianças, adolescentes e jovens em estado de vulnerabilidade social, órfãos, destituídos de poder familiar ou em processo. Foi criada em julho de 1963 na cidade de Passo Fundo e obtém seus recursos através do governo federal, de uma contribuição monetária do município e, principalmente, de doações do público em geral. Os acolhidos são enviados à instituição somente por ordem judicial. Então, quando é constatado algum tipo de problema familiar muito sério, o caso é enviado ao juizado para análise e, caso seja verificado que não há condições de a criança voltar para casa, ela é enviada para a instituição.

Todos esses processos, desde a chegada da criança até sua saída, além do controle de funcionários, são realizados de forma manual, o que consome bastante tempo, além de não haver segurança na precisão das informações, pois dado obrigatórios podem não ser preenchidos, ou a letra ser incompreensível, dentre outros. Logo, o objetivo deste projeto é a criação de um protótipo que, caso aprovado pela instituição, irá dinamizar o processo de cadastro e controle de acolhidos e funcionários, facilitando esses processos e trazendo maior segurança quanto às informações.

3.2 REQUISITOS

3.2.1 Requisitos Funcionais

O *software* que atenderá o Lar Emiliano Lopes deverá conter funcionalidades específicas para a instituição.

Através de *login* e senha, haverá acesso à área administrativa e haverá restrição de acesso de acordo com a função do funcionário, ou seja, o coordenador será o único que terá acesso a todos os dados, e os demais funcionários poderão acessar somente o que for necessário para realização de seu trabalho.

Haverá um cadastro das crianças, que conterá os dados mais básicos sobre as mesmas. Após o cadastro realizado, será possível acessar, através de menus, os dados cadastrados sobre todos.

Diante das características apontadas, os seguintes requisitos funcionais foram identificados:

- manutenção de familiares;
- manutenção de adolescentes;
- manutenção de acolhimentos;
- manutenção de desacolhimentos;
- manutenção de visitas;
- manutenção de atividades.

3.2.2 Requisitos Não Funcionais

O sistema será implementado fazendo-se uso da tecnologia Java EE, juntamente com o framework Java Server Faces, e então os seguintes itens serão utilizados:

- linguagem de programação Java;
- *framework* Java Server Faces;
- servidor Apache Tomcat;
- sistema de gerenciamento de banco de dados PostgreSQL.

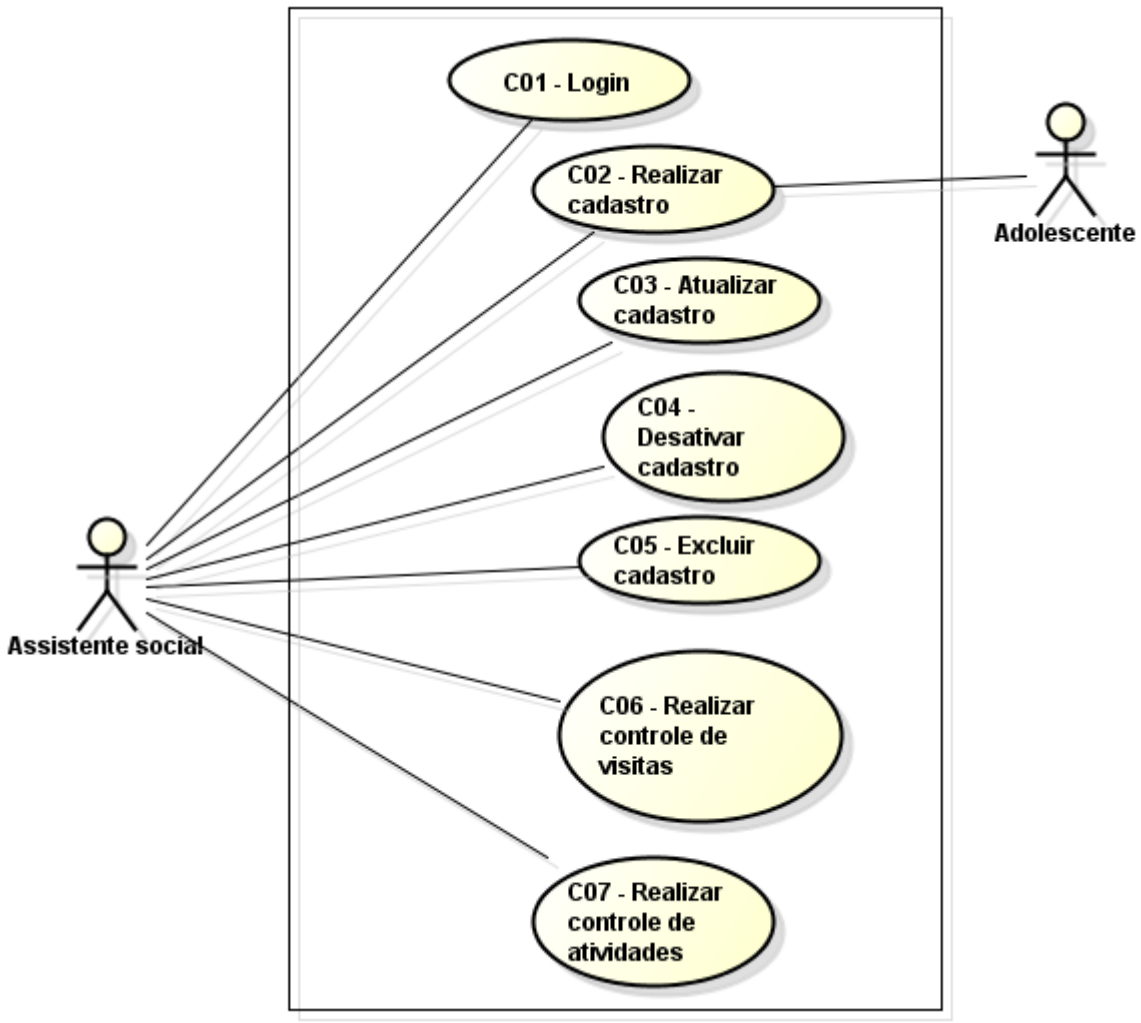
3.3 DIAGRAMAS

Para melhor visualização dos requisitos do sistema, foram criados os diagramas de casos de uso, de atividades e de classes.

3.3.1 Diagrama de Casos de Uso

A partir dos requisitos obtidos, foi construído o digrama de casos de uso (Figura 3), que tem como objetivo demonstrar as funcionalidades que o sistema deverá conter e quem irá interagir com ele, ou seja, seus casos de uso e seus atores.

Figura 4 - Diagrama de casos de uso



Fonte: próprio autor

A assistente social irá realizar *login* no sistema e poderá fazer o cadastro do adolescente acolhido. Esse cadastro poderá ser atualizado ou desativado, posteriormente a sua realização. A assistente social também poderá realizar o controle de visitas e das atividades realizadas pelos acolhidos.

A descrição do Caso de Uso C02 foi detalhada, devido ao mesmo ser considerado o caso de uso mais complexo, para que se possa ter uma melhor visualização. A descrição encontra-se na Tabela 2 abaixo:

Tabela 2 - Descrição caso de uso C02

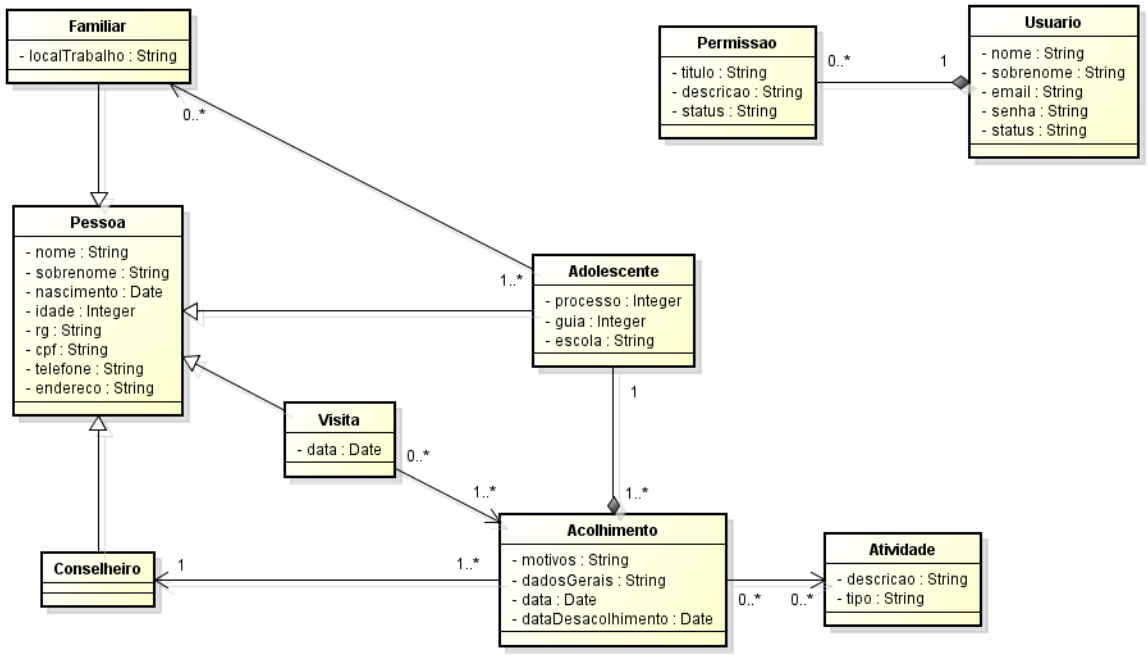
Caso de uso: C02	Realizar cadastro
Objetivo	O objetivo deste caso de uso é realizar o cadastro de adolescentes que são enviados à instituição
Atores	Assistente social, Adolescente
Condições de início	Assistente inicia o programa de manutenção de adolescentes
Fluxo principal	1. Sistema exibe uma tela de listagem dos adolescentes já cadastrados. 2. Assistente social seleciona a opção de novo registro. (A1) (A2) 3. Assistente social informa os dados do adolescente. 4. Assistente social informa os familiares do adolescente. 5. Assistente social seleciona a opção gravar. (A3) 6. Sistema grava o novo adolescente. 7. Caso de uso encerrado.
Fluxo alternativo	(A1) Assistente social seleciona a opção editar 1. Sistema coloca o registro do adolescente selecionado em modo de edição. 2. Assistente social faz as alterações necessárias. 3. Assistente social seleciona a opção gravar. 4. Sistema grava as alterações. 5. Caso de uso encerrado. (A2) Assistente social seleciona a opção excluir 1. Sistema pede a confirmação para exclusão.

	2. Assistente social confirma a exclusão. 3. Sistema exclui o registro. 4. Caso de uso encerrado. (A3) Assistente social seleciona a opção cancelar 1. Sistema cancela a inclusão ou edição do registro. 2. Caso de uso encerrado.

Fonte: próprio autor

3.3.2 Diagrama de Classes

Figura 5 - Diagrama de Classes



Fonte: próprio autor

A classe Usuário, com os atributos nome, sobrenome, email, senha e *status*, deve informar qual permissão possui. Uma Permissão é constituída de título, descrição e *status*.

A classe Pessoa, composta pelos atributos nome, sobrenome, nascimento, idade, rg, cpf, telefone, endereço e *status*, é a superclasse de Conselheiro, Familiar, Visita e Adolescente.

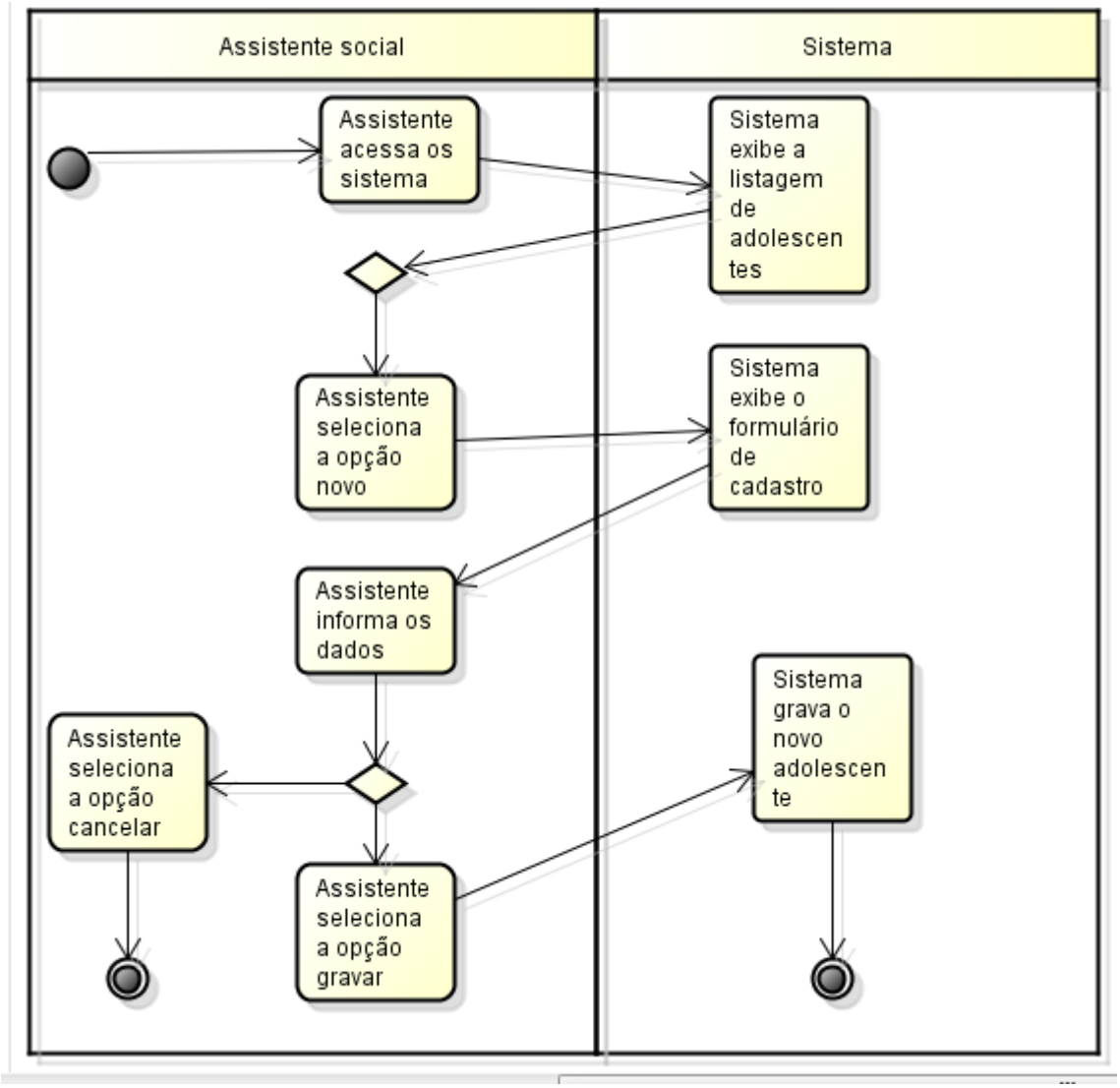
A classe Adolescente, especialização de Pessoa, também possui os atributos processo, guia e escola. O adolescente pode possuir diversos familiares cadastrados. Para um adolescente existir, obrigatoriamente deve estar ligado a um acolhimento. Ao longo de sua

existência, um adolescente pode possuir vários acolhimentos, pois pode sair da instituição e, após algum tempo, retornar, sendo, então, vinculado a um novo código de acolhimento. Um acolhimento necessita estar vinculado a um único conselheiro. Um conselheiro, no entanto, pode possuir vários acolhimentos. O acolhimento também pode conter diversas atividades, e estas serem realizadas em diversos acolhimentos.

3.3.3 Diagrama de Atividades

O diagrama de atividades descreve detalhadamente o caso de uso C02, especificando seus fluxos.

Figura 6 - Diagrama de atividades



Fonte: próprio autor

3.4 IMPLEMENTAÇÃO

3.4.1 Camada de Modelo

A figura 6 mostra a unidade de persistência, que realiza conexão com o banco de dados. O sistema gerencial de banco de dados utilizados foi o PostgreSQL. O nome do banco de dados criado foi sistema.

A *tag* <persistence-unit> é utilizada para definir uma unidade de persistência, onde podemos informar através da propriedade name qual seu nome, que é utilizado quando cria-se um EntityManager e através da propriedade transaction-type qual seu tipo de transação. RESOURCE_LOCAL foi utilizada pois as transações com o banco de dados foram criadas.

Figura 7 - Unidade de persistência

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
  <persistence-unit name="ModeloPU" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <class>beans.Pessoa</class>
    <properties>
      <property name="hibernate.connection.username" value="postgres"/>
      <property name="hibernate.connection.driver_class" value="org.postgresql.Driver"/>
      <property name="hibernate.connection.password" value="[redacted]"/>
      <property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect"/>
      <property name="hibernate.connection.url" value="jdbc:postgresql://localhost:5432/sistema"/>
      <property name="hibernate.cache.provider_class" value="org.hibernate.cache.NoCacheProvider"/>
      <property name="hibernate.connection.autocommit" value="false"/>
      <property name="hibernate.show_sql" value="true"/>
      <property name="hibernate.hbm2ddl.auto" value="update"/>
    </properties>
  </persistence-unit>
</persistence>
```

Fonte: próprio autor

A classe persistente Pessoa, ilustrada na Figura 7, foi criada implementando a interface *Serializable*. Sua chave primária foi mapeada através da anotação @Id. A anotação @SequenceGenerator foi utilizada para definir o uso de uma sequência no banco de dados, que realizará o incremento da mesma. A anotação @GeneratedValue foi utilizada para permitir o uso dessa sequência para definição automática para o valor do código identificador. Após, a anotação @Column foi utilizada para especificar melhor como cada coluna deveria

ser tratada no banco de dados. O atributo *name* definiu o nome de cada coluna mapeada na tabela. O atributo *nullable=false* indicou que nenhuma coluna poderia conter valores nulos. A anotação @Length definiu o número máximo de caracteres que cada coluna deveria conter. A anotação @NotEmpty indicou que a coluna não poderia ficar vazia. A anotação @Temporal foi utilizada para mapear o atributo "nascimento", que era do tipo *Date*.

Figura 8 - Classe Pessoa

```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
@Table(name = "PESSOA")
public class Pessoa implements Serializable{

    @Id
    @Column(name = "ID")
    @SequenceGenerator(name = "SEQ", sequenceName = "SEQ_PESSOA", allocationSize = 1)
    @GeneratedValue(generator = "SEQ", strategy = GenerationType.SEQUENCE)
    private Integer id;

    @Column(name = "NOME", length = 50, nullable = false)
    @Length(max = 50, message = "O nome não deve ultrapassar 50 caracteres")
    @NotEmpty(message = "O nome deve ser preenchido")
    private String nome;

    @Column(name = "SOBRENOME", length = 50, nullable = false)
    @Length(max = 50, message = "O sobrenome não deve ultrapassar 50 caracteres")
    @NotEmpty(message = "O sobrenome deve ser preenchido")
    private String sobrenome;

    @Column(name = "NASCIMENTO")
    @Temporal(TemporalType.DATE)
    @Past(message = "Data de nascimento inválida")
    @NotNull(message = "A data de nascimento não pode ser nula")
    private Calendar nascimento;
```

Fonte: próprio autor

A classe Adolescente, que é uma subclasse de Pessoa, possui relação de Muitos para Um com a classe Acolhimento. Então, foi utilizada a anotação @JoinColumn para identificar qual coluna da tabela Acolhimento seria utilizada para referenciar essa associação.

Esta classe também possui relação de Muitos para Muitos com as classes Familiar e Visita. Para esse mapeamento foram criadas, no banco de dados, as tabelas "adolescente_familiar" e "adolescente_visita". Dessa vez, além da anotação @JoinColumn para indicação de qual coluna da tabela seria referenciada, foi utilizada a anotação @JoinTable, para indicar qual tabela possui essa relação. Também, a anotação

@LazyCollection foi utilizada para definir a estratégia de recuperação da coleção. Nesse caso, a opção Extra foi utilizada, que define que a coleção somente será recuperada quando for acessada.

Figura 9 - Classe Adolescente

```
@ManyToMany
@JoinTable(name = "ADOLESCENTE_FAMILIAR",
        joinColumns =
        @JoinColumn(name = "ADOLESCENTE", referencedColumnName = "ID"),
        inverseJoinColumns =
        @JoinColumn(name = "FAMILIAR", referencedColumnName = "ID"))
@LazyCollection(LazyCollectionOption.EXTRA)
private List<Familiar> familiares = new ArrayList<>();

private Familiar familiar;

@ManyToMany
@JoinTable(name = "ADOLESCENTE_VISITA",
        joinColumns =
        @JoinColumn(name = "ADOLESCENTE", referencedColumnName = "ID"),
        inverseJoinColumns =
        @JoinColumn(name = "VISITA", referencedColumnName = "ID"))
@LazyCollection(LazyCollectionOption.EXTRA)
private List<Visita> visitas = new ArrayList<>();

private Visita visita;

@OneToMany(mappedBy = "adolescente", cascade = (CascadeType.ALL),
        orphanRemoval = true)
@LazyCollection(LazyCollectionOption.EXTRA)
@OrderBy(value = "id asc")
private List<Acolhimento> acolhimentos = new ArrayList<>();
```

Fonte: próprio autor

3.4.2 Camada de Controle

Para gravação e alteração dos dados do Adolescente, foram criados os métodos “gravar” e “alterar”, o último, recebendo como parâmetro o Adolescente. No método de gravação, é feita a comparação: se o objeto for nulo, ele será persistido, caso contrário, será atualizado para o contexto persistente.

Figura 10 - Métodos para gravar e alterar o adolescente

```
public String gravar(){
    try {
        getEm().getTransaction().begin();
        if (getObjeto().getId() != null) {
            getEm().merge(getObjeto());
        } else {
            getEm().persist(getObjeto());
        }
        getEm().getTransaction().commit();
        filtrar();
        return "listar";
    } catch (Exception e) {
        if (getEm().getTransaction().isActive() == false) {
            getEm().getTransaction().begin();
        }
        getEm().getTransaction().rollback();
        FacesMessage msg = new FacesMessage(FacesMessage.SEVERITY_ERROR,
                "Erro ao persitir: " + e.getMessage(), "");
        FacesContext.getCurrentInstance().addMessage(null, msg);
        return "form";
    }
}

public String alterar(Adolescente obj) {
    setObjeto(obj);
    return "form";
}
```

Fonte: próprio autor

Para excluir um adolescente, o objeto é recuperado do banco de dados e removido. Caso ocorra algum erro, é disparada uma mensagem de “Erro ao excluir”.

Figura 11 - Método para excluir o adolescente

```
public String excluir(Adolescente obj) {
    try {
        getEm().getTransaction().begin();
        getEm().remove(obj);
        getEm().getTransaction().commit();
    } catch (Exception e) {
        if (getEm().getTransaction().isActive() == false) {
            getEm().getTransaction().begin();
        }
        getEm().getTransaction().rollback();
        FacesMessage msg = new FacesMessage(FacesMessage.SEVERITY_ERROR,
                "Erro ao excluir: "+e.getMessage(), "");
        FacesContext.getCurrentInstance().addMessage(null, msg);
    }
    filtrar();
    return "listar";
}
```

Fonte: próprio autor

3.4.3 Camada de Visão

A figura 11 mostra a criação do formulário de cadastro e edição de adolescentes. Para criação do menu, que é utilizado em todas as páginas, foi utilizado o recurso *Facelets*, que permite que o mesmo seja encapsulado em um *template*, fazendo com que não seja necessário mudanças individualmente em cada página. Para a utilização do *template*, o mesmo foi indicado em todas as páginas em uma *tag ui:composition*.

Figura 12 - Formulário adolescente

```
<ui:composition template="/templates/template.xhtml">
    <ui:define name="titulo">Cadastro de adolescentes</ui:define>
    <ui:define name="conteudo">
        <h:form id="formEdicao">
            <p:messages/>
            <p:growl/>
            <p:tabView effect="fade" effectDuration="normal">
                <p:tab title="Dados" titletip="Dados gerais">
                    <p:panelGrid columns="2">
                        <f:facet name="header">Cadastro de adolescentes</f:facet>
                        <p:outputLabel value="ID" for="id"/>
                        <p:inputText value="#{controleAdolescente.objeto.id}"
                                     id="id" readonly="true" size="5"/>
                        <p:outputLabel value="Nome" for="nome"/>
                        <p:inputText value="#{controleAdolescente.objeto.nome}"
                                     id="nome" required="true" placeholder="Obrigatório"/>
                        <p:outputLabel value="Sobrenome" for="sobrenome"/>
                        <p:inputText value="#{controleAdolescente.objeto.sobrenome}"
                                     id="sobrenome" required="true" placeholder="Obrigatório"/>
                        <p:outputLabel value="Data de nascimento" for="nascimento"/>
                        <p:calendar value="#{controleAdolescente.objeto.nascimento}"
                                    converter="#{controleAdolescente.converterCalendar}"
                                    id="nascimento" required="true" placeholder="Obrigatório"
                                    pattern="dd/MM/yyyy"/>
                        <p:outputLabel value="Idade" for="idade"/>
                        <p:inputText value="#{controleAdolescente.objeto.idade}"
                                     id="idade" required="true" placeholder="Obrigatório"/>
```

Fonte: próprio autor

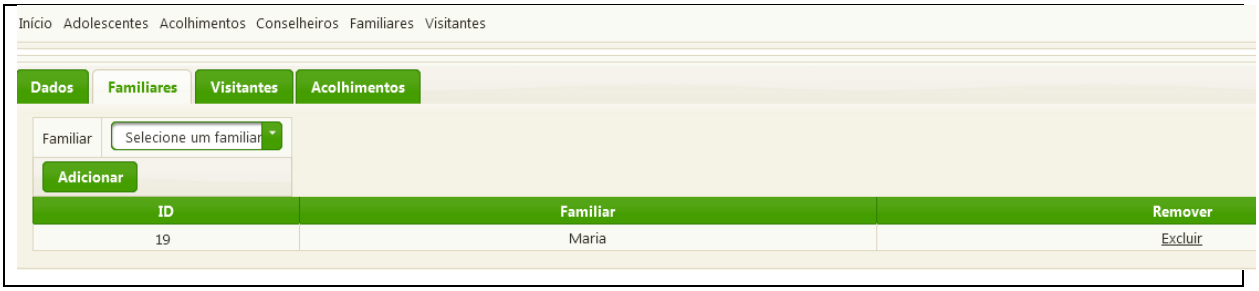
O formulário de cadastro de adolescentes foi criado fazendo-se uso de abas, sendo que na primeira são inseridos os dados cadastrais do adolescente, na segunda, relacionam-se seus familiares, na terceira, seus visitantes, e na última aba são indicados os acolhimentos do adolescente.

Figura 13 - Cadastro de adolescente

Fonte: próprio autor

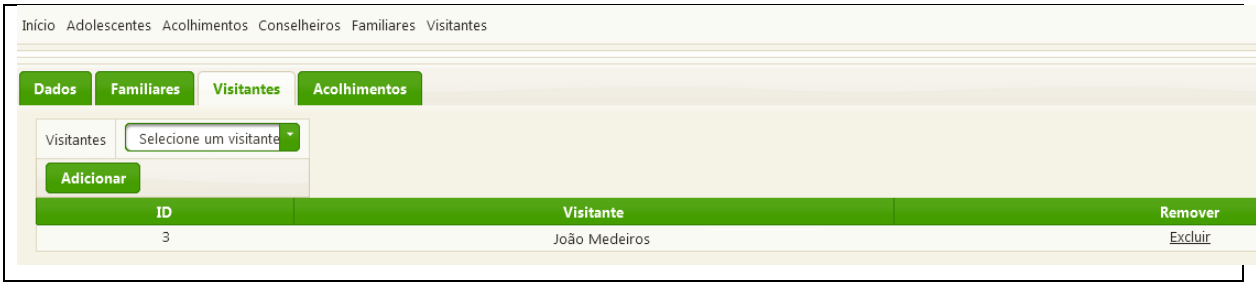
As figuras abaixo demonstram a listagem de todas as informações relacionadas ao adolescente: familiares, visitantes e acolhimentos.

Figura 14 - Listagem de familiares (relação adolescente - familiar)



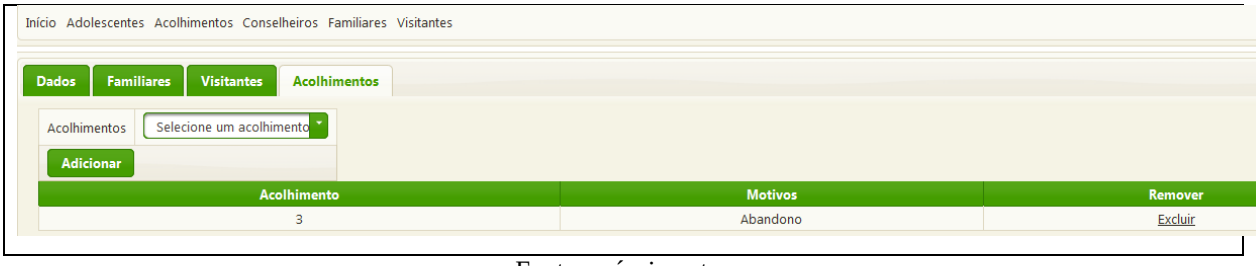
Fonte: próprio autor

Figura 15 - Listagem de visitantes (relação adolescente – visita)



Fonte: próprio autor

Figura 16 - Listagem de acolhimentos (relação adolescente - acolhimento)



Fonte: próprio autor

4 CONSIDERAÇÕES FINAIS

O presente trabalho apresentou os passos necessários à criação de um protótipo para um sistema de cadastro de uma instituição acolhedora. Iniciou-se com o referencial teórico, em que foi realizado um estudo aprofundado sobre todas as tecnologias necessárias para a sua criação. Esse estudo foi de grande importância, pois trouxe maior conhecimento sobre diversas tecnologias que poderão ser utilizadas em projetos futuros.

Após, foi realizada a modelagem do sistema, que possibilitou uma melhor visualização das informações que seriam necessárias no protótipo. O desenvolvimento, implementado com a tecnologia *Java Server Faces*, juntamente com a biblioteca *Primefaces* demonstrou que essas tecnologias são bastante simples de serem utilizadas e possuem funcionalidades muito interessantes, como o fato de não ser necessária maior preocupação com *layout*, pois a biblioteca traz isso de forma simples, facilitando o trabalho do desenvolvedor.

Acredita-se que os objetivos do trabalho foram devidamente atingidos, pois foi realizada toda a análise das funcionalidades necessárias ao sistema para ir ao encontro com as necessidades da instituição, e foi possível a implementação do protótipo para que parte dessas funcionalidades possam ser testadas pela instituição e o sistema possa ser devidamente concluído.

Como trabalho futuro, há o aprimoramento do sistema, aumentando o número de funcionalidades e fazendo com que seja possível realizar todo o processo de acolhimento, da entrada até a saída dos acolhidos, dispensando o método manual em sua totalidade e dinamizando todo o processo, além da implementação da segurança da aplicação, através de *login* e senha, protegendo-a de acessos indevidos. Também, como trabalhos futuros, deverão ser realizados todos os testes e validações necessárias, em conjunto com a instituição, para que o sistema possa ser implementado de forma a auxiliar nas atividades do Lar.

REFERÊNCIAS

BERGSTEN, Hans. **Java Server Faces**. Sebastopol: O'Reilly Media, Inc.; 2009. Disponível em: <http://sngo.me/my5wi>. Acesso em: 11 set. 2014.

BOOCH, Grady. et al. **UML, O guia do usuário**. 2. ed. Rio de Janeiro: Elsevier, 2005. Disponível em: <http://sngo.me/WTjE5>. Acesso em: 22 out. 2014.

DRAKE, Joshua; WORSLEY, John. **Practical PostgreSQL**. Sebastopol: O'Reilly Media, Inc., 2011. Disponível em: <http://sngo.me/8sGcD>. Acesso em: 26 jul. 2014.

DUDNEY, Bill. et al. **Mastering Java Server Faces**. Indianopolis: Wiley Publishing, Inc., 2004. Disponível em: <http://sngo.me/06al5>. Acesso em: 22 jul. 2014.

FEITOSA, Diego Bomfim. **Visão geral sobre Primefaces**. Sergipe, 2010. Disponível em: <http://williamgamers.wordpress.com/2012/06/04/visao-geral-sobre-primefaces/>. Acesso em: 10 nov. 2014.

FOWLER, Martin. **UML Essencial**. 3. ed. Porto Alegre: Bookman, 2005. Disponível em: <http://sngo.me/U3vbI>. Acesso em: 15 ago. 2014.

GEARY, David; HORSTMANN, Cay. **Core Java Server Faces**. Redwood Shores: Oracle, 2010.

GONCALVES, Antonio. **Beginning Java EE Platform with GlassFish 3**. 2. ed. 2010

GUPTA, Arun. **Java EE 6 Pocket Guide**. 1. ed. Sebastopol: O'Reilly Media, Inc., 2012. Disponível em: <http://sngo.me/zgZk6>. Acesso em: 23 set. 2014.

HLAVATS, Ian. **Instant Primefaces Starter**. Packt Publishing, 2013. Disponível em: <http://sngo.me/QD6Kp>. Acesso em: 15 ago. 2014.

JENDROCK, Eric. et al. **The Java EE 6 Tutorial: Advanced Topics**. 4. ed. New Jersey: Pearson Education, Inc., 2013. Disponível em: <http://sngo.me/60M29>. Acesso em: 12 ago. 2014.

JUNEAU, Josh. **Introducing Java EE 7**. 2013. Disponível em: <http://sngo.me/b4THM>. Acesso em: 13 jul. 2014.

JUNEAU, Josh. **Java EE 7 Recipes: A Problem-Solution Approach**. New York: Springer, 2013

KEITH, Mike; SCHINCARIOL, Merrick. **PRO JPA 2.0**. 2013. Disponível em: <http://sngo.me/GVeC7>. Acesso em: 10 out. 2014.

MAKI, Chris. **JPA 101: Java Persistence Explained**. Colorado: SourceBeat, 2007

PILONE, Dan. **UML 2.0 Pocket Reference**. 1. Ed. Sebastopol: O'Reilly Media, Inc., 2006. Disponível em: <http://sngo.me/y7ARY>. Acesso em: 23 out. 2014.

STINSON, Barry. **PostgreSQL Essential Reference.** 1. ed. New Riders Publishing, 2002. Disponível em: <http://sngo.me/Cqr4Q>. Acesso em 15 out. 2014.

ZAMBON, Giulio. **Beginning JSP, JSF and TOMCAT**. 2012. Disponível em: <http://sngo.me/j9RKw >. Acesso em: 17 out. 2014.