INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SUL-RIO-GRANDENSE - CÂMPUS PASSO FUNDO
CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

TIAGO DELLINGHAUSEN LOPES

SISTEMA DE MAPEAMENTO DE DADOS DE ALUNOS COM NECESSIDADES ESPECIFICAS

Jorge Luis Boeira Bavaresco

PASSO FUNDO
2018

TIAGO DELLINGHAUSEN LOPES

SISTEMA DE MAPEAMENTO DE DADOS DE ALUNOS COM NECESSIDADES ESPECIFICAS

Monografia apresentada ao Curso de Tecnologia em Sistemas para Internet do Instituto Federal Sul-rio-grandense, Câmpus Passo Fundo, como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet.

Orientador: Jorge Luis Boeira Bavaresco

Co-orientadora: Vanessa Lago Machado

PASSO FUNDO

2018

TIAGO DELLINGHAUSEN LOPES

SISTEMA DE MAPEAMENTO DE DADOS DE ALUNOS COM NECESSIDADES ESPECIFICAS

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

Banca Examinadora:

_____

Me. Jorge Luis Boeira Bavaresco

_____

Drº. Alexandre Tagliari Lazzaretti

_____

Drº. Josué Toebe

_____

Coordenação do Curso

PASSO FUNDO

2018

AGRADECIMENTOS

Primeiramente agradeço a minha esposa, Flávia Carvalho Calill, e minha filha, Lara Calill Lopes, por seu apoio e compreensão nos momentos complicados.

Agradeço aos meus País, Larry Almeida Lopes e Ângela Dellinghausen Lopes, e meus Irmãos, Thomas Dellinghausen Lopes e Derly Delinghausen Lopes, pela ajuda, apoio, educação que com certeza me ajudaram muito a chegar até aqui.

Agradeço ao professor Jorge Luís Boeira Bavaresco pela sua orientação, conhecimento, atenção e paciência que foram fundamentais.

Por fim agradeço a todos os colegas que tive nesta jornada, aos demais professores, funcionários e estagiários, sem eles está jornada não teria sido possível.

RESUMO

Os sistemas digitais permitem implementar maneiras para facilitar a resolução das necessidades das pessoas, e como estas estão a cada dia mais tempo conectadas ao mundo digital, soluções que estejam disponíveis *online* são necessárias. Levando isto em consideração este trabalho procurou mostrar o desenvolvimento da aplicação que deve ser uma solução para atender as necessidades do NAPNE (Núcleo de Apoio às Pessoas com Necessidades Específicas) e facilitasse o acesso dos colaboradores e alunos ao sistema, tanto para fornecer informações quanto para quem deve busca-las. Esta solução utilizou as tecnologias PostgreSQL para o banco de dados com a JPA controlando a comunicação, a coleção de APIs do Java EE 7 com os *containers* EJB (Enterprise JavaBeans) para passar os dados, e o JSF (JavaServer Faces) permitindo um layout mais amigável para os usuários com o auxílio do Primefaces. No final a aplicação desenvolvida atingiu em parte o seu objetivo permitindo coletar dados junto aos alunos.

Palavras-chave: Questionário. Java. Aplicação. Gerenciamento.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

APIs - Application Programming Interfaces
EJB - Enterprise JavaBeans
IFSUL - Instituto Federal Sul-rio-grandense
Java EE - Java Enterprise Edition
JNDI - Java Naming and Directory Interface
JPA - Java Persistence API
JSF - JavaServer Faces
JSP - JavaServer Pages
JVM - Java Virtual Machine
NAPNE - Núcleo de Apoio às Pessoas com Necessidades Específicas
TCC - Trabalho de Conclusão de Curso
UML - Unified Modeling Language

SUMÁRIO

1 INTRODUÇÃO

A cada dia que passa as pessoas se beneficiam cada vez mais das novas tecnologias para facilitar a solução dos mais variados problemas, e esta tendência deve chegar principalmente a infraestrutura da área de ensino. Ainda se perde muito tempo e recursos de mão de obra especializada com tarefas que poderiam ser facilmente automatizadas, como colher informações dos alunos, e gerenciar estas informações.

Levando isto em conta chega-se a questão: Como melhorar o gerenciamento dos dados coletados dos alunos com necessidade especificas que atualmente estão armazenados em formulários de papel?

Para contribuir com a solução deste problema foi desenvolvido um sistema Web que permite lançar perguntas em um questionário, criar vários questionários e permitir que os alunos respondam estes questionários.

Ao realizar este projeto foram estudadas as tecnologias de desenvolvimento para sistemas web com a linguagem de programação Java que se melhor se adaptariam aos objetivos propostos. Então foi feito um estudo de caso para poder ser feito o levantamento dos requisitos para o projeto, logo em seguida foram aplicadas algumas técnicas de análise de dados para ficar mais claro o projeto. Finalmente foi desenvolvida a aplicação web seguindo a modelagem feita e procurando apresentar uma solução prática para o problema estudado.

Este trabalho é composto das seguintes seções: Referencial teórico, onde serão apresentadas as tecnologias utilizadas, Metodologia, onde pode-se ver como o trabalho foi projetado, Desenvolvimento, que mostra como foi desenvolvido o sistema, e Resultados, que apresenta os resultados finais atingidos pelo trabalho.

1.1 Objetivo geral

Este projeto tem como objetivo desenvolver um sistema informatizado para organizar as informações dos alunos com necessidades especificas para facilitar o acesso e armazenamento destas.

1.2 Objetivos específicos

Os objetivos específicos deste trabalho são:

a) Realizar a modelagem da aplicação, permitindo o lançamento de informações e posterior pesquisa destes dados.
b) Realizar o desenvolvimento da aplicação conforme a modelagem para o gerenciamento dos dados.
c) Criar relatórios que ajudem a visualizar os dados.

2 REFERENCIAL TEÓRICO

Este capítulo contém embasamento teórico dos principais conceitos das tecnologias e ferramentas utilizados para dar suporte no desenvolvimento do sistema.

2.1 JAVA

O Java foi lançado pela Sun Microsystems em 1995, logo chamou a atenção pelo enorme interesse na *Web*. Hoje ele é usado em aplicativos corporativos de grande porte, em servidores *Web* e aplicativos para dispositivos portáteis de consumo popular, e em muitas outras áreas. (DEITEL e DEITEL, 2010).

Ao contrário de linguagens que são compiladas em código de máquina, O Java não depende de Hardware específico, já que é compilado em bytecodes que são portáveis, por serem executados em qualquer plataforma que contém uma *Java Virtual Machine* (JVM). Esta máquina virtual simula um computador ocultando o sistema operacional e o hardware dos programas que ela atende, então as aplicações Java rodam em qualquer plataforma de computador que implemente uma JVM (DEITEL e DEITEL, 2010).

Programas Java são divididos em partes chamadas classes, nas classes declaramos partes menores, os métodos, que são responsáveis por realizar tarefas. Além de criar as suas próprias partes, é possível ainda utilizar as *Java APIs* (*Application Programming Interfaces*), que são bibliotecas de classe Java disponibilizadas por outros programadores (DEITEL e DEITEL, 2010).

“Java é uma poderosa linguagem de programação” (DEITEL e DEITEL, 2010). Sendo assim ele foi escolhido, devido ao tempo que está no mercado, por ter uma grande comunidade de programadores que conhecem a linguagem, pela facilidade de rodar em muitos ambientes de computador e por poder dividir o projeto em camadas, podendo assim separar as soluções em partes menores.

2.2 JAVA EE

O *Java Enterprise Edition* (Java EE) é uma coleção de APIs de baixo nível que podem ser integradas ao projeto conforme a necessidade, permitindo ao

desenvolvedor focar no desenvolvimento da lógica de negócio, enquanto estas APIs solucionam os problemas como transações, troca de mensagens e persistência de dados (GONÇALVES, 2011).

Para poder utilizar as APIs o Java EE é baseado em padrões, estes padrões são abertos o que permite o uso de várias soluções desenvolvidas pela comunidade, e soluções comerciais desenvolvidas por empresas e com código proprietário (GONÇALVES, 2011).

Quanto a arquitetura o Java EE implementa especificações para diferentes containers, que são ambientes de tempo de execução, estes containers oferecem serviços aos componentes que eles hospedam. Os componentes usam os padrões definidos para poder se comunicar com a infraestrutura do Java EE e com outros componentes, desta maneira abstraem a complexidade técnica e melhoram a portabilidade (GONÇALVES, 2011).

Os containers:

- Container de *applets*: estão presentes na maioria dos navegadores, permite ao desenvolvedor focar no aspecto visual da navegação e fornecem um ambiente seguro ao usuário, já que ele não permite ao código acessar recursos do sistema local (GONÇALVES, 2011).
- Container de cliente de aplicação: neste ficam as classes, bibliotecas, e outros arquivos Java necessários para os serviços que foram implementados (GONÇALVES, 2011).
- Container web: oferece o serviço para o gerenciamento e execução de componentes web, fornece as páginas web aos navegadores.
- Container EJB: gerencia os *beans* que contém a lógica funcional da aplicação (GONÇALVES, 2011).

Alguns serviços que são prestados pelos containers aos componentes:

- *Java Persistence API* (JPA): API para o mapeamento objeto-relacional, permite consultar objetos armazenados no banco de dados (GONÇALVES, 2011).
- Validação: permite validar restrições feitas nas classes (Gonçalves, 2011).

- Java Naming na Directory Interface (JNDI): Permite a navegação, em diretórios e serviços, para acessar objetos e acessar os seus dados e métodos de forma mais prática (GONÇALVES, 2011).

2.3 JSF

As tecnologias Web do Java evoluem com o intuito de elevar o nível de abstração e seguir no caminho de separar em camadas o aplicativo, para tornar os sistemas desenvolvidos em Java de fácil manutenção e escalabilidade. Ao separar em camadas permite-se que cada programador foque no desenvolvimento da aplicação na área em que está mais bem preparado (DEITEL e DEITEL, 2010).

JavaServer Faces (JSF) é um framework de aplicativo Web que ajuda a organizar e construir aplicativos. Cria-se a aparência e o funcionamento de uma página (Camada de visão) com o JSF adicionando elementos a um documento *Java Server Pages* (JSP) e manipulando seus atributos. O Comportamento da página fica definido em outra camada através de arquivos Java relacionados (DEITEL e DEITEL, 2010).

Cada JSP representa uma página web, neste arquivo deve-se seguir o padrão *JavaBean*, que é uma classe que contém um construtor padrão (vazio) e métodos *get* e *set* para todas as propriedades da classe. O *bean* de página então vai permitir a interação com os elementos da página, e também controlar o ciclo de vida da página para gerenciar suas tarefas (DEITEL e DEITEL, 2010).

O JSF é um framework baseado em componentes. Para exibir uma tabela formada por linhas e colunas, em vez de utilizar um loop para gerar as *tags* HTML das linhas e colunas você só precisa adicionar a uma página um componente de tabela (GEARY e HORSTMANN, 2010).

Embora os componentes JSF padrão sejam suficientes para a maioria dos aplicativos Web básicos, também é possível criar seus próprios componentes ou importar bibliotecas de componentes disponibilizadas por fornecedores independentes. (DEITEL e DEITEL, 2010).

Sendo assim com JSF ganhamos facilidade e simplicidade ao gerar o código do layout e ainda teremos a possiblidade de escolher entre várias bibliotecas de fornecedores independentes.

2.4 PRIMEFACES

O *PrimeFaces* é uma biblioteca de componentes de interface que tem por características ser leve, sem dependências e não exige configuração extra. Os componentes *PrimeFaces* procuram ocultar a complexidade e manter a flexibilidade no design da página (GEARY e HORSTMANN, 2010).

2.5 JPA

A maioria dos dados utilizados pelas aplicações precisam ser armazenados em bases de dados. Dados persistentes estão por toda parte e geralmente utilizam base de dados relacionais para esta persistência (GONÇALVES, 2011).

Em uma linguagem orientada a objetos, como o Java, manipula-se objetos e suas instâncias. O estado, os dados, o comportamento do objeto só existe enquanto a JVM está rodando e o conteúdo está na memória, neste momento temos uma entidade (GONÇALVES, 2011).

Para persistir um objeto em um banco de dados relacional utilizaremos a JPA que faz o mapeamento objeto-relacional, transformando uma entidade em uma tabela do banco de dados. A JPA utiliza a linguagem *Java Persistence Query Language* (JPQL) para acessar o banco de dados enquanto a aplicação manipula as entidades (GONÇALVES, 2011).

A JPA exige alguns padrões para reconhecer uma classe como uma entidade. É necessário fazer uso de anotações de *Entity* e ID, utilizando a biblioteca *javax.persistence*, a classe tem que ter um construtor vazio. Além disto a JPA precisa de um arquivo de configurações *.xml* que contém as informações necessárias para a conexão com o banco de dados (GONÇALVES, 2011).

A API de validação pode estar integrada a estas classes e fazer as validações antes destas serem persistidas. Assim que a validação for confirmada pode se fazer o mapeamento objeto relacional e os relacionamentos entre as classes com as devidas anotações (GONÇALVES, 2011).

2.6 EJB

Existem nas aplicações alguma operações e interações que devido a suas complexidades não devem ficar na camada de persistência, tão pouco na interface de usuário, por isto no Java EE se implementa uma camada funcional que se chama *Enterprise Java Beans* (EJB's) (GONÇALVES, 2011).

EJB's são componentes do lado servidor que contém a lógica funcional responsável pelas transações e pela segurança. Devida a grande capacidade de interatividade que as EJB's têm com outras tecnologias Java elas ficam responsáveis por esta camada intermediaria (GONÇALVES, 2011).

Os EJB's podem ser de três tipos:

Stateless: o *bean* da sessão não guarda nenhum estado, podendo ser acessado por qualquer cliente (GONÇALVES, 2011).

Stateful: o *bean* da sessão guarda o estado que deve ser único para cada usuário (GONÇALVES, 2011).

Singleton: guarda o estado da sessão que é compartilhado entre todos os usuários, tem suporte a acessos concorrentes (GONÇALVES, 2011).

2.7 POSTGRESQL

O *PostgreSQL* é um sistema de gerenciamento de banco de dados objeto-relacional com código fonte aberto. Começou a ser desenvolvido em 1986, na Universidade da Califórnia em Berkeley. Mas o projeto só teve um grande avanço em 1996, quando um grupo que não fazia parte da Universidade se uniu ao projeto, então o projeto se tornou *open-source* e com a ajuda de diferentes desenvolvedores o código do sistema se tornou uniforme e consistente (POSTGRESQL, 2018).

Atualmente, o *PostgreSQL* tem suporte para Windows, Linux e Solaris e MAC. Suporta a maioria dos tipos de dados SQL como inteiros, *strings* e decimais além de poder armazenar sons, imagens e vídeo. Implementa interfaces de comunicação nativas para C, C++, Java, .Net, Perl, Python, Ruby entre outras (POSTGRESQL, 2018).

Como esta aplicação necessita persistir dados, e o *PostgreSQL* por ter esta função, por dar suporte a vários sistemas operacionais, por se comunicar com a linguagem Java e por possuir licença aberta foi o escolhido.

2.8 UML

A *Unified Modeling Language* (UML) é segundo GUEDES “uma linguagem visual para modelar softwares baseados no paradigma de orientação a objetos”, e pode ser utilizada para todos os domínios de aplicação e por vários tipos de processos (GUEDES, 2011).

A UML não é uma linguagem de programação, ela tem por objetivo levantar as características do sistema, a forma como deve funcionar, apontar as lógicas de negócio e até mesmo os recursos físicos necessários para que o sistema funcione corretamente (GUEDES, 2011).

Esta linguagem de modelagem surgiu da união de três métodos de modelagem, que eram muito populares na década de 1990. Então em 1996 com o apoio da *Rational Software*, uma grande empresa de desenvolvimento na época, foi lançada a primeira versão da UML. A UML assim que foi lançada recebeu apoio de vária empresas de modelagem e desenvolvimento de software e já em 1997 foi reconhecia como linguagem-padrão de modelagem pelo *Object Management Group* (OMG). Em 2005 foi lançada a versão 2.0 (GUEDES, 2011).

Deve-se modelar um software para criar uma imagem deste software antes de começar a produzi-lo de fato. Isto ajuda a produzir um software com menos erros, com menores custo de produção, que vão gerar menos defeitos no futuro e menor manutenção. É também a partir da modelagem que vamos chegar as necessidades técnicas para o projeto, custos e prazo. (GUEDES, 2011).

Na prática a UML utiliza vários diagramas do sistema que se está planejando para que cada diagrama complemente o outro sob vários pontos de vista. Desta maneira consegue-se descobrir falhas antes que elas ocorram e diminuam a ocorrência de erros no futuro. Alguns exemplos de diagramas (GUEDES, 2011):

- Diagrama de Casos de Uso
- Diagrama de Classes
- Diagrama de Objetos
- Diagrama de Sequência
- Diagrama de Atividades
- Diagrama de Tempo

2.9 PESQUISAS RELACIONADAS

O trabalho “SISTEMA PARA ELABORAÇÃO E DISPONIBILIZAÇÃO DE QUESTIONÁRIOS DE AVALIAÇÃO” é um exemplo de aplicação desenvolvida para gerenciar questionários (DA ROSA, 2015).

Este aplicativo foi desenvolvido para a composição e a disponibilização de questionários de avaliação de atividades, para implementar um editor de questionários, para realizar o controle de respondentes do questionário e permitir o acesso aos dados obtidos pelas repostas dos questionários

O trabalho foi desenvolvido para o curso de especialização em tecnologia JAVA da Universidade Tecnológica Federal do Paraná.

Este trabalho foi feito para gerar questionários específicos e permitir a coleta dos dados de forma anônima, já o trabalho aqui apresentado pretende coletar dados específicos de alunos pré-selecionados com necessidades específicas e permitir a sua identificação.

2.10 METODOLOGIA

Neste capítulo são detalhadas todas as etapas realizadas durante o desenvolvimento do trabalho.

Inicialmente foi realizada uma entrevista com o representante do NAPNE, a partir desta reunião foi feito o levantamento de requisitos, uma modelagem UML, e então as pesquisas bibliográficas foram realizadas para obter conhecimento das tecnologias utilizadas para o desenvolvimento.

2.11 ESTUDO DE CASO

No primeiro momento foi realizada uma entrevista com o representante do NAPNE, que apresentou o questionário padrão: “FORMULÁRIO DE NECESSIDADES EDUCACIONAIS E DE SAÚDE DO ESTUDANTE” (Anexo A), utilizado atualmente e discorreu sobre alguns problemas do atual sistema de coleta e gerenciamento dos dados

2.12 MÉTODO ATUAL

Atualmente após o aluno cumprir o processo de matricula ele é encaminhado para uma avaliação de saúde, momento onde responde o questionário (Anexo A), que procura levantar quaisquer necessidades especificas que o aluno possua, além de obter informações que não estão disponíveis em outros cadastros do IFSUL.

Após o questionário ser preenchido ele é armazenado em um arquivo que contém questionários de todos os alunos que estudam ou já estudaram no IFSUL, inclusive contendo mais de um questionário para o mesmo aluno se este se matriculou mais de uma vez no IFSUL.

A partir do relatado nota-se que o método atual apresenta vários problemas para gestão da informação contida nos formulários. Ao utilizar um questionário em meio físico para armazenar estes dados torna-se o trabalho de pesquisa das informações nesses contidas muito trabalhosa além de não atender situações de emergência, já que estão lá presentes informações que seriam necessárias nestes casos.

Não há nenhuma forma de controle de medidas posteriores que foram tomadas a partir das informações coletadas nos questionários.

O gerenciamento das informações na maneira atual também impossibilita o cruzamento entre várias respostas para se tentar uma análise mais profunda para encontrar necessidades especificas que nem mesmo o aluno saiba que possui.

2.13 PROPOSTA

Para solucionar esta situação, a primeira mudança foi na forma como o questionário é respondido. Agora é feito de forma eletrônica, via um sistema web o que permite armazenar os dados em um banco de dados digital.

Com essa nova abordagem procurou-se facilitar o acesso a estas informações, pelas pessoas autorizadas, tornando possível encontrar em tempo hábil informações necessárias em situações de emergência.

Além disto pode-se a partir do tratamento destas informações, criar-se novas situações para qualificar os resultados do NAPNE, como gerar estatísticas das necessidades especificas encontradas e como essas foram atendidas dentro do IFSUL, com o cadastro de cada ação que foi tomada.

2.14 LEVANTAMENTO DE REQUISITOS

Avaliando o método atual e o questionário apresentado (Anexo A), foi realizado uma identificação dos requisitos do sistema e uma modelagem usando a linguagem UML. Então foi realizado o estudo das tecnologias necessárias para realizar o projeto.

2.14.1 Requisitos Funcionais

- Manter usuários.
- Manter alunos.
- Manter perguntas.
- Manter respostas.
- Manter ações posteriores.

- Manter necessidades especificas.
- Ter um sistema de *login* e senha para usuários administradores.
- Ter um sistema de *login* utilizando apenas o cpf do aluno.
- Deve gerar relatórios.

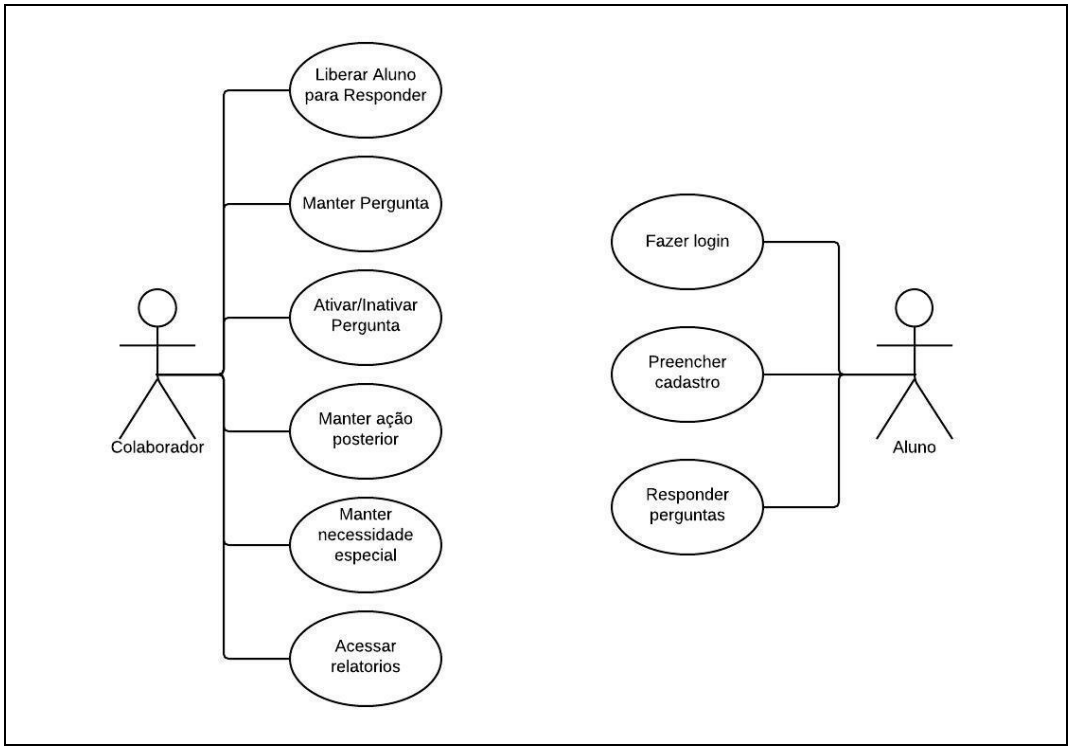
2.14.2 Requisitos não funcionais

- A aplicação deve ser acessível em diferentes sistemas operacionais.
- O sistema deve persistir seus dados em sistema gerenciador de banco de dados.

2.15 DIAGRAMAS DE CASOS DE USO

O diagrama de caso de uso procura demonstrar em uma única imagem todas as ações dos atores no sistema, do ponto de vista dos usuários. O diagrama a seguir foi criado a partir dos requisitos levantados pelo estudo de caso.

Figura 1 - Diagrama de casos de uso



Fonte: Do Autor

2.16 DESCRIÇÃO DOS CASOS DE USO

Nesta sessão procurou-se descrever os casos de uso vistos na Figura 1, apresentando em detalhes cada interação do ator com o caso de uso.

2.16.1 Caso de uso liberar aluno para responder

Tabela 1 – Documentação do caso de uso liberar aluno para responder

Nome do caso de uso	**Liberar aluno para responder**
Caso de uso geral	
Ator principal	Colaborador
Atores secundário	
Resumo	Caso de uso descreve como é liberado para um novo aluno criar o seu cadastro.
Pré-condições	Deve ter o CPF válido do aluno
Pós-condições	Aluno estará apto para acessar o sistema
Fluxo principal	
Ações do ator	**Ações do sistema**
1.Informar o CPF do aluno	
	2.Validar o CPF e gravar se for válido
Restrições/Validações/Regras de negócio	O CPF deve ser valido.
Fluxo alternativo	
Ações do ator	**Ações do sistema**
	1.Informar se o CPF for inválido
2.Informar o CPF do aluno	

Fonte: Do Autor.

2.16.2 Caso de uso manter pergunta

Tabela 2 - Documentação do caso de uso manter pergunta

Nome do caso de uso	**Manter pergunta**
Caso de uso geral	
Ator principal	Colaborador
Atores secundário	
Resumo	Caso de uso descreve como incluir uma pergunta no sistema
Pré-condições	Ter a pergunta formulada
Pós-condições	Pergunta estar inserida no sistema
Fluxo principal	
Ações do ator	**Ações do sistema**
1.Solicitar para inserir uma nova pergunta	
	2.Exibir o formulário de inclusão de pergunta
3.Inserir a pergunta	
Restrições/Validações/Regras de negócio	
Fluxo alternativo	
Ações do ator	**Ações do sistema**

Fonte: Do Autor.

2.16.3 Caso de uso ativar/inativar pergunta

Tabela 3 - Documentação do caso de uso ativar/inativar pergunta

Nome do caso de uso	**Ativar/Inativar pergunta**
Caso de uso geral	
Ator principal	Colaborador
Atores secundário	
Resumo	Caso de uso descreve como alterar o estado de uma pergunta entre ativa e inativa.
Pré-condições	Saber a pergunta que vai ser alterada.
Pós-condições	Pergunta teve seu estado alterado.
Fluxo principal	

Ações do ator	**Ações do sistema**
1.Solicitar para editar uma pergunta.	
	2. Exibir o formulário de edição de pergunta.
2.Alterar o estado ativo/inativo.	
Restrições/Validações/Regras de negócio	
Fluxo alternativo	
Ações do ator	**Ações do sistema**

Fonte: Do Autor.

2.16.4 Caso de uso manter ação posterior

Tabela 4 - Documentação do caso de uso manter ação posterior

Nome do caso de uso	**Manter ação posterior**
Caso de uso geral	
Ator principal	Colaborador
Atores secundário	
Resumo	Caso de uso descreve como inserir as informações de uma "ação posterior".
Pré-condições	Ter os dados da "ação posterior".
Pós-condições	Ação posterior incluída no sistema
Fluxo principal	
Ações do ator	**Ações do sistema**
1.Solicitar a inclusão de uma ação posterior	
	2.Exibir a tela de inclusão de ação posterior
3.Inserir os dados na tela de inclusão de ação posterior	
Restrições/Validações/Regras de negócio	
Fluxo alternativo	
Ações do ator	**Ações do sistema**

Fonte: Do Autor.

2.16.5 Caso de uso manter necessidade especial

Tabela 5 - Documentação do caso de uso manter necessidade especial

Nome do caso de uso	**Manter necessidade especial**
Caso de uso geral	
Ator principal	Colaborador
Atores secundário	
Resumo	Caso de uso descreve como inserir uma necessidade especial no sistema
Pré-condições	Saber a necessidade especial
Pós-condições	Necessidade especial incluída no sistema
Fluxo principal	
Ações do ator	**Ações do sistema**
1. Solicitar a inclusão de uma necessidade especial	
	2. Exibir a tela de inclusão de necessidade especial
3. Inserir os dados na tela de inclusão de necessidade especial	
Restrições/Validações/Regras de negócio	
Fluxo alternativo	
Ações do ator	**Ações do sistema**

Fonte: Do Autor.

2.16.6 Caso de uso acessar o sistema

Tabela 6 - Documentação do caso de uso acessar o sistema

Nome do caso de uso	**Fazer login**
Caso de uso geral	
Ator principal	Aluno
Atores secundário	
Resumo	Caso de uso descreve como se dá o acesso do aluno ao sistema.
Pré-condições	Aluno estar apto para acessar o sistema
Pós-condições	Aluno liberado para preencher o seu cadastro

Fluxo principal	
Ações do ator	**Ações do sistema**
1.Acessar o sistema	
	2. Exibir a tela de login
3. Inserir o seu CPF	
	4. Mostrar tela de cadastro do aluno.
Restrições/Validações/Regras de negócio	CPF deve ser válido e estar cadastrado no sistema.
Fluxo alternativo	
Ações do ator	**Ações do sistema**
	1.Informar CPF inválido/CPF não cadastrado

Fonte: Do Autor.

2.16.7 Caso de uso preencher cadastro

Tabela 7 - Documentação do caso de uso preencher cadastro

Nome do caso de uso	**Preencher cadastro**
Caso de uso geral	
Ator principal	Aluno
Atores secundário	
Resumo	Caso de uso descreve como o aluno preenche o cadastro
Pré-condições	Aluno estar logado no sistema
Pós-condições	Aluno está apto para responder as perguntas
Fluxo principal	
Ações do ator	**Ações do sistema**
	1.Mostrar a tela de cadastro do aluno após o seu primeiro login, ou se está ainda não estiver respondida.
3. Preencher o cadastro apresentado	
	2.Liberar aluno para responder as perguntas.
Restrições/Validações/Regras de negócio	
Fluxo alternativo	
Ações do ator	**Ações do sistema**

Fonte: Do Autor.

2.16.8 Caso de uso responder perguntas

Tabela 8 - Documentação do caso de uso responder perguntas

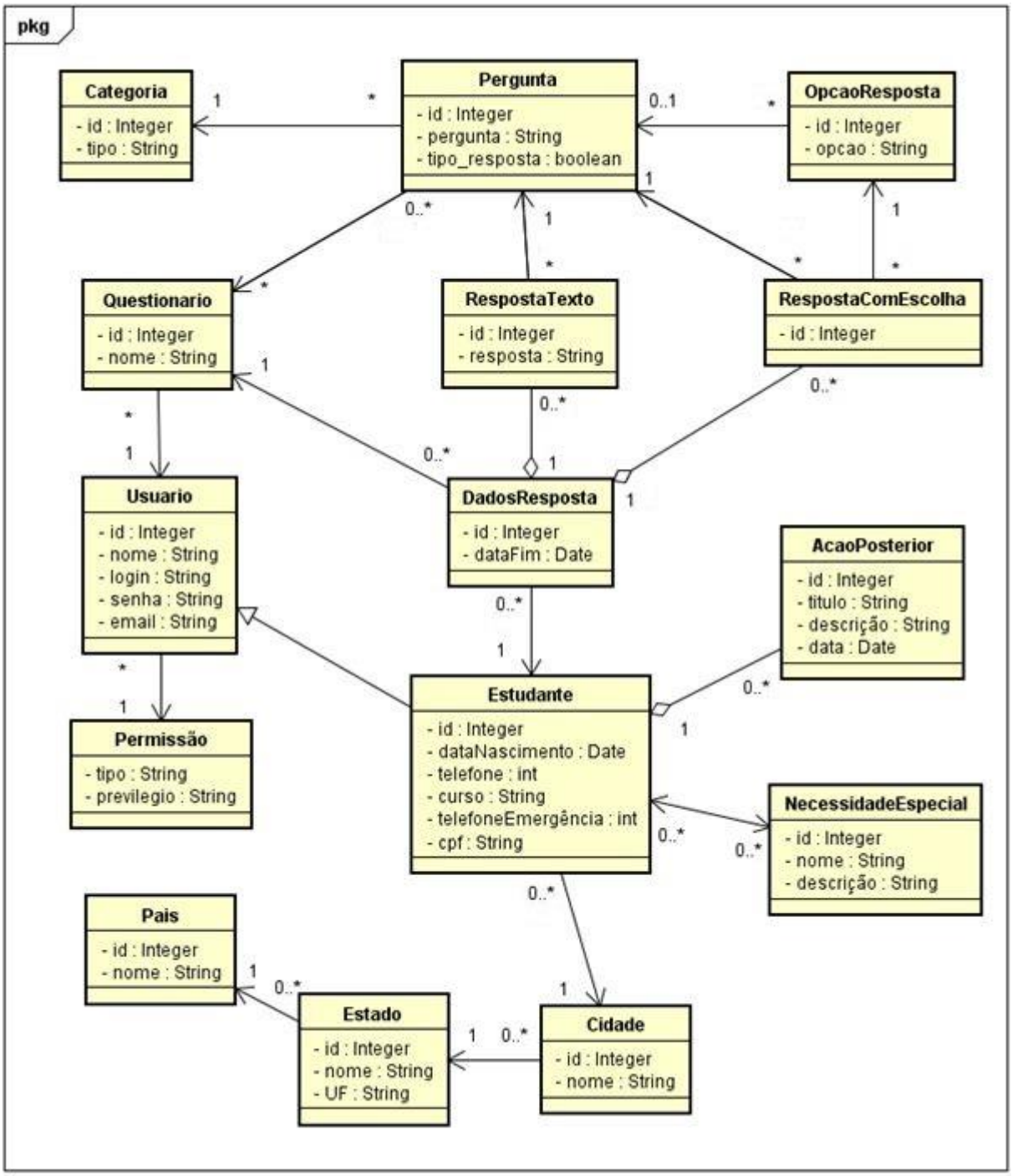
Nome do caso de uso	**Responder pergunta**
Caso de uso geral	
Ator principal	Aluno
Atores secundário	
Resumo	Caso de uso descreve como o aluno responde a pergunta
Pré-condições	Aluno ter respondido o seu cadastro no sistema
Pós-condições	Aluno ter respondido a pergunta
Fluxo principal	
Ações do ator	**Ações do sistema**
1. Solicitar para responder perguntas	
	2. Mostrar tela de responder pergunta com a primeira pergunta configurada que ainda não foi respondida por este aluno
3.Responder a pergunta apresentada	
	4. Mostrar a próxima pergunta
Restrições/Validações/Regras de negócio	
Fluxo alternativo	
Ações do ator	**Ações do sistema**

Fonte: Do Autor.

2.17 DIAGRAMA DE CLASSES

O diagrama de classes, na Figura 2, procura ilustrar a forma que as classes estão dispostas e relacionadas no sistema.

Figura 2 - Diagrama de classes



Fonte: Do Autor

Começando pela classe “Pais”, onde há o atributo da classe que representa os Países, que é o nome do país, em seguida há a classe “Estado” que contém os atributos que representam o nome do estado, UF (Unidade da Federação), que contém a sigla do respectivo estado e o país que este estado pertence. Já a classe “Cidade” contém os atributos nome da cidade que ela mantém e o estado a que esta cidade pertence.

A classe “Estudante” mantém os dados específicos de cada estudante além de ser uma extensão da classe “Usuário”, o que lhe dá acesso a todos os seus atributos como “login” e “senha” que serão necessários para acessar o sistema, a classe “Permissão” contém os dados que definirão quais áreas do sistema cada “tipo” de usuário poderá acessar e quais classes ele poderá acessar.

As classes “AcaoPosterior” e “NecessidadeEspecial” são dependentes da classe “Estudante” e armazenam os dados específicos das necessidades especificas e ações posteriores referentes a cada aluno.

Para poder ter mais de um questionário foi criada a classe “Questionario” que mantém um nome de questionário além de guardar o usuário que a criou.

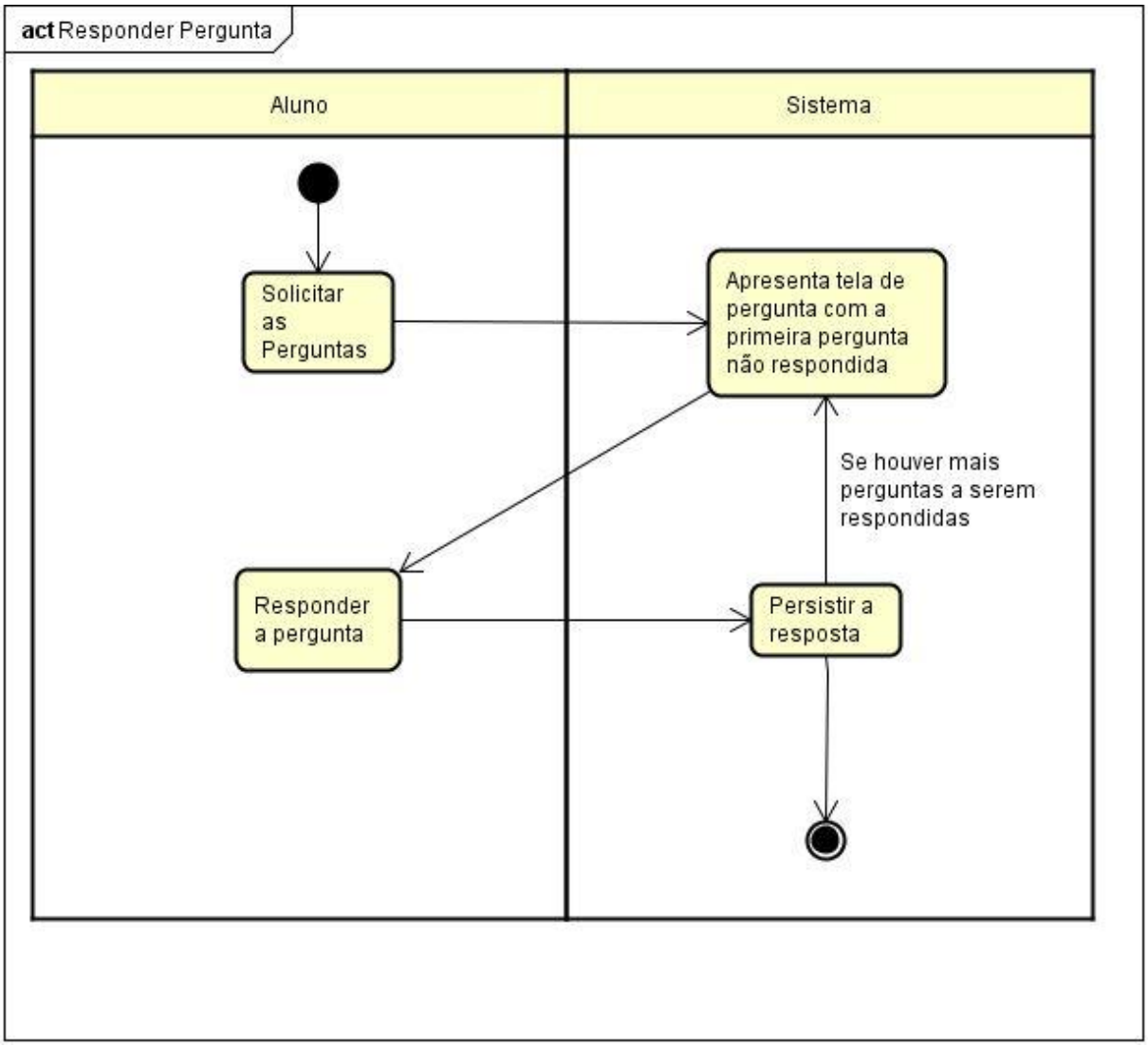
Para possibilitar diferentes tipos respostas a classe “Pergunta” contém além da pergunta um atributo que controla o tipo de resposta que será apresentado na tela para ser respondido. As possibilidades são representadas pela classe “RespostaTexto” que armazenara uma resposta do tipo “String” enquanto que a classe “OpcaoResposta” armazena as opções possíveis de resposta para uma respectiva pergunta e a classe “RespostaComEscolha” armazena a resposta escolhida.

Finalmente a classe “DadosResposta” mantém em um primeiro momento qual estudante pode responder qual questionário e quando o estudante responder o questionário manterá também as respostas e a data em que foi finalizado o questionário.

2.18 DIAGRAMA DE ATIVIDADES

O diagrama de atividades pode mostrar como ocorre um processo e como o sistema responde a algumas ações. O diagrama de atividade, na Figura 3, tem como objetivo demonstrar como o sistema disponibiliza as perguntas para o aluno responder:

Figura 3 - Diagrama de atividade responder pergunta

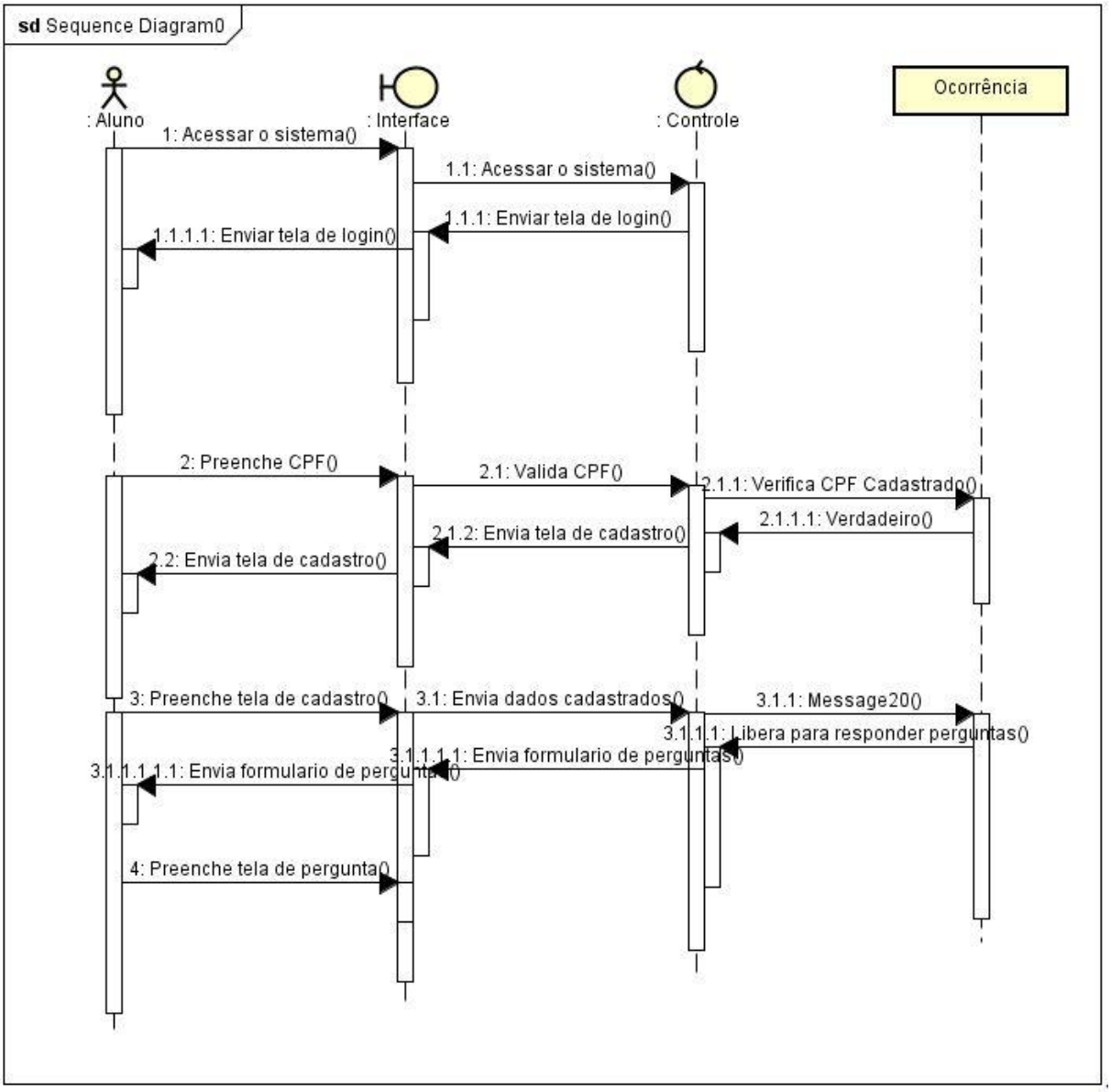


Fonte: Do Autor

2.19 Diagrama de sequência

O diagrama de sequência demonstra a ordem dos acontecimentos.

Figura 4 - Diagrama de sequência



Fonte: Do Autor

Neste diagrama de sequência, da Figura 4, foi demonstrado todo o processo de um aluno desde seu *login* no sistema até responder as perguntas.

3 DESENVOLVIMENTO

Nesta sessão serão apresentados os recursos e técnicas utilizados para o desenvolvimento da aplicação web, a estrutura em camadas da aplicação e os elementos computacionais envolvidos na sua operação.

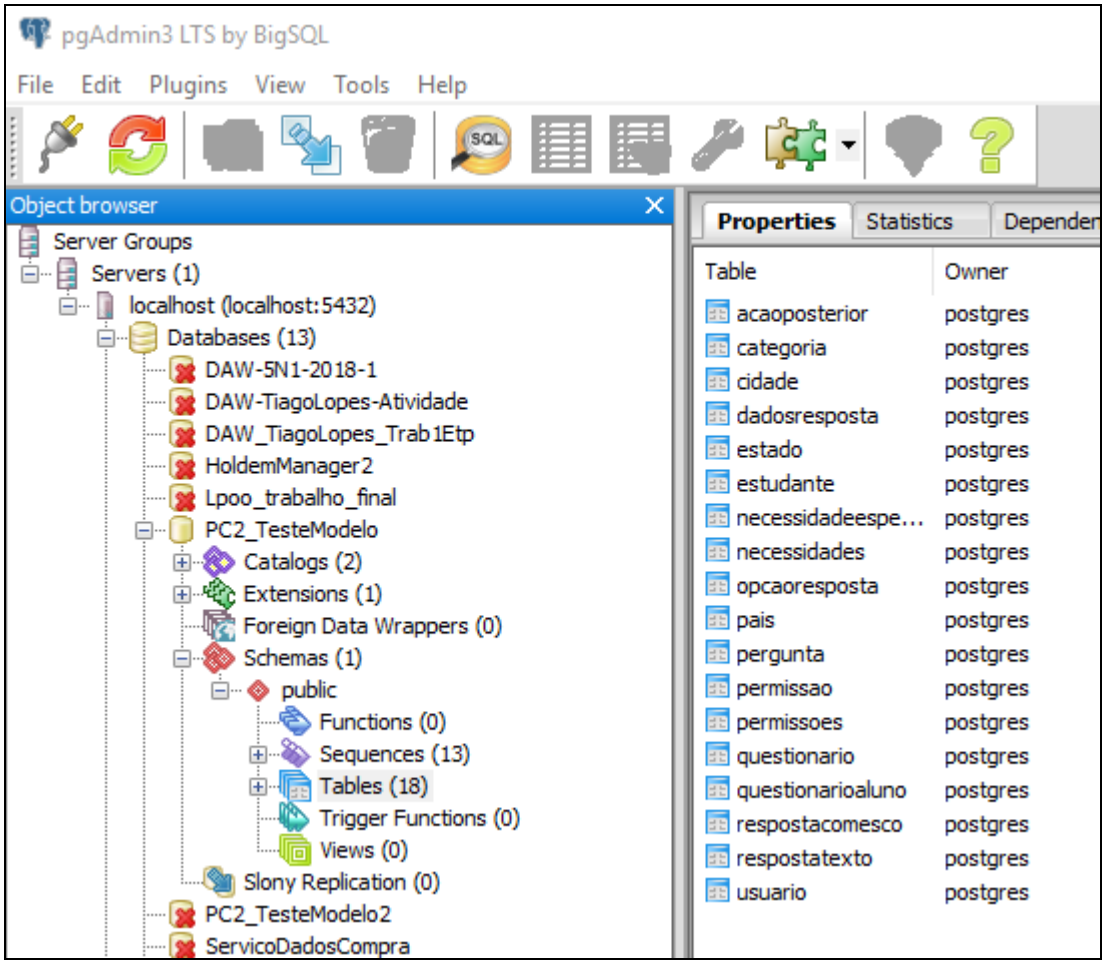
3.1 AMBIENTE DE DESENVOLVIMENTO E RECURSOS

3.1.1 IDE e Servidores

Para desenvolver o sistema de mapeamento de dados de alunos com necessidades especificas, utilizou-se uma aplicação IDE e um servidor, ambos foram necessários para a construção do código e administração do sistema. Como servidor da aplicação utilizou-se o *GlassFish Server* versão 4.1. Esse servidor é mantido pela empresa Oracle, com código fonte aberto e suporta todas as especificações da API *Java Enterprise Edition*. A IDE adotada para implementar a aplicação foi o *NetBeans* IDE 8.2, ferramenta que permite incorporar o servidor *GlassFish*.

Para gerenciar e administrar o banco de dados, utilizou-se o Sistema gerenciador de Banco de Dados *PostgreSQL*. Por intermédio do software gráfico *PGAdmin III* foi possível acessar servidor desse sistema gerenciador tornou-se possível guardar os dados e a comunicar-se com a aplicação web elaborada. Podemos ver a interface do *PGAdmin III* apresentado as tabelas do sistema na Figura 5.

Figura 5 – Estrutura do banco de dados da aplicação



Fonte: Do Autor

3.1.2 Bibliotecas e Frameworks

Para possibilitar o desenvolvimento do sistema de mapeamento de dados de alunos com necessidades especificas, foram utilizados os recursos viabilizados por frameworks e bibliotecas elaborados para a utilização em conjunto com o *JavaServer Faces*. Esses foram aplicados para auxiliar na criação da interface com o usuário, para validar e persistir os dados e tratar da comunicação com a aplicação, entre o banco de dados e o servidor. As bibliotecas utilizadas na aplicação são:

- **Biblioteca Hibernate JPA 4.3.11 JPA 2.1:** A partir desta biblioteca foi possível mapear os dados dos objetos em persistir estes em um banco de dados objeto-relacional.
- **Biblioteca Hibernate Validator 5.2.4:** Esta biblioteca é responsável por implementar a Bean Validation API 1.1, usada para a validar os dados que

integrantes do objeto persistido, impedindo improcedências na aplicação e seu banco de dados.

- **Driver JDBC PostgreSQL versão 42.2.1:** permite que a aplicação se conecte ao banco de dados possibilitando gerenciar e manipular os dados utilizando código Java padrão. É um driver JDBC de software livre escrito em Java.
- **Biblioteca BootsFaces 1.2.0:** A biblioteca BootsFaces é uma biblioteca de recursos gráficos para o JSF baseada no estilo visual da biblioteca Bootstrap. Com a utilização dessa biblioteca no sistema permite a apresentação de um layout responsivo.
- **Biblioteca PrimeFaces 6.1:** Esta biblioteca foi utilizada em conjunto com a Biblioteca BootsFaces. Têm como principal funcionalidade auxiliar na implementação de recursos gráficos.

3.2 ESTRUTURA E LAYOUT DA APLICAÇÃO

Nesta sessão será detalhada a arquitetura em camadas da aplicação e os elementos que constituem a implementação do layout design responsivo.

3.2.1 Camada de Modelo

A camada de modelo do sistema é responsável por manter os dados das classes na forma de objetos instanciados pela aplicação. A biblioteca JPA é responsável pelo mapeamento destes objetos para integrar ao banco de dados objeto-relacional.

Fica sob a responsabilidade da biblioteca JPA criar as relações entre os objetos e as tabelas do banco de dados. Determinam-se estas relações através de anotações nos objetos que indicam as propriedades e as restrições para cada dado nas tabelas persistidas no banco de dados.

Para poder persistir as classes deve-se garantir que estas cumpram o padrão *JavaBeans*, que consiste em implementar na classe a interface *Serializable* e ter todos os atributos encapsulados e sendo acessados apenas pelos seus respectivos métodos *getter* e *setter.* Figura 6 podemos ver um fragmento da classe *Cidade*:

Figura 6 – Exemplo de classe no padrão *JavaBeans*

```
@Entity
@Table(name = "cidade")
public class Cidade implements Serializable {

    @Id
    @SequenceGenerator(name = "seq_cidade", sequenceName = "seq_cidade_id",
            allocationSize = 1)
    @GeneratedValue(generator = "seq_cidade", strategy = GenerationType.SEQUENCE)
    private Integer id;
    @NotNull(message = "O nome não pode ser nulo")
    @NotBlank(message = "O nome não pode ser em branco")
    @Length(max = 40, message = "O nome não pode ter mais que {max} caracteres")
    @Column(name = "nome", length = 40, nullable = false)
    private String nome;
    @NotNull(message = "O estado deve ser informado")
    @ManyToOne
    @JoinColumn(name = "estado", referencedColumnName = "id", nullable = false,
            foreignKey = @javax.persistence.ForeignKey(name="fk_cidade_estado"))
    private Estado estado;

    public Cidade() {
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
```

Fonte: Do Autor

Através das anotações feitas no código utilizamos a biblioteca JPA personalizamos as características do objeto que será persistido no banco de dados. Ao anotar-se *@Entity,* define-se que esta é uma classe que será representada como uma tabela no banco de dados, e logo em seguida foi definido o nome desta tabela com a anotação *@Table(name="nome_tabela").*

As colunas da tabela são representações dos atributos da classe, o primeiro atributo costuma ser o atributo identificador da tabela, a ele define-se a anotação *@Id*, em seguida para gerar o identificador automaticamente no banco de dados usa-se a anotação *@GeneratedValue,* para definir que a estratégia para gerar os valores é sequencial utiliza-se o atributo *strategy = GenerationType.SEQUENCE,* este atributo utiliza a anotação @SequenceGenerator com os atributos que armazenam os controles da sequência.

Para os atributos utiliza-se a anotação *@Column* aonde pode-se definir o nome que coluna terá no banco de dados, além de outras características.

Para controlar relacionamentos pode-se utilizar a anotação @JoinColumn precedida do tipo de relacionamento, como *@ManyToOne*, *@OneToMany* ou *@ManyToMany.*

Para controlar as características dos dados a serem persistidos pode-se utilizar anotações. Ao utilizarmos *@NotNull* determina-se que o atributo não pode ser nulo, @NotBlank não aceita valores em branco para dados tipo *String*, com *@Length* pode-se determinar o número mínimo e máximo de caracteres do atributo. Caso alguma destas regras não seja respeitada pode-se definir a mensagem a ser retornada com o atributo *message="Mensagem_de_erro"* em cada anotação.

Para aplicar a JPA em um sistema é necessário definir os dados referentes a unidade de persistência da aplicação e as configurações necessárias para efetivar a conexão com o banco de dados, para isto utiliza-se o arquivo *persistence.xml.* Na Figura 7 o arquivo persistence.xml com as configurações e as classes da aplicação:

Figura 7 - Conteúdo do arquivo persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
    <persistence-unit name="PC2_Teste_WebPU2" transaction-type="JTA">
        <provider>org.hibernate.ejb.HibernatePersistence</provider>
        <jta-data-source>jdbc/pc2_2</jta-data-source>
        <class>br.edu.ifsul.modelo.AcaoPosterior</class>
        <class>br.edu.ifsul.modelo.Categoria</class>
        <class>br.edu.ifsul.modelo.Cidade</class>
        <class>br.edu.ifsul.modelo.DadosResposta</class>
        <class>br.edu.ifsul.modelo.Estado</class>
        <class>br.edu.ifsul.modelo.Estudante</class>
        <class>br.edu.ifsul.modelo.NecessidadeEspecial</class>
        <class>br.edu.ifsul.modelo.OpcaoResposta</class>
        <class>br.edu.ifsul.modelo.Pais</class>
        <class>br.edu.ifsul.modelo.Pergunta</class>
        <class>br.edu.ifsul.modelo.Questionario</class>
        <class>br.edu.ifsul.modelo.RespostaComEscolha</class>
        <class>br.edu.ifsul.modelo.RespostaTexto</class>
        <class>br.edu.ifsul.modelo.Usuario</class>
        <class>br.edu.ifsul.modelo.Permissao</class>
        <properties>
            <property name="hibernate.hbm2ddl.auto" value="update"/>
            <property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect"/>
            <property name="hibernate.transaction.jta.platform"
                      value="org.hibernate.service.jta.platform.internal.SunOneJtaPlatform"/>
            <property name="hibernate.classloading.use_current_tccl_as_parent" value="false"/>
        </properties>
    </persistence-unit>
</persistence>
```

Fonte - Do Autor

O que torna possível manter os dados diretamente através da aplicação é a criação de um *pool* de conexões e de recursos JDBC no servidor glassfish, as configurações desta tecnologia se dão através do arquivo "glassfish_resouces.xml",

este arquivo possui dados como “databaseName” que apontam o nome da base de dados que será acessada e a “URL” que vai apontar para o endereço de conexão ao banco de dados aonde a base de dados deve se encontra.

O arquivo também possui o elemento “jdbc-resources” que contém principalmente o “jndi-name” que será necessário para fazer uso deste recurso e o “pool-name” utilizado para acessar o pool de conexões. Pode-se ver na Figura 8 o conteúdo do aquivo “glassfish_resouces.xml” utilizado no projeto:

Figura 8 - Conteúdo do arquivo glassfish_resouces.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE resources PUBLIC "-//GlassFish.org//DTD GlassFish Application Server 3.1 Resource Definitions//EN" "http://gl
<resources>
    <jdbc-connection-pool allow-non-component-callers="false" associate-with-thread="false" connection-creation-retry-a
        <property name="serverName" value="localhost"/>
        <property name="portNumber" value="5432"/>
        <property name="databaseName" value="PC2_TesteModelo"/>
        <property name="User" value="postgres"/>
        <property name="Password" value="postgres"/>
        <property name="URL" value="jdbc:postgresql://localhost:5432/PC2_TesteModelo"/>
        <property name="driverClass" value="org.postgresql.Driver"/>
    </jdbc-connection-pool>
    <jdbc-resource enabled="true" jndi-name="jdbc/pc2_2" object-type="user" pool-name="PC2_TesteModelo_postgresPool"/>
</resources>
```

Fonte - Do Autor

3.2.1.1 Classes DAO

Para gerenciar a comunicação com o banco de dados utiliza-se as classes *DAO* (*Data Access Object*), as quais utilizam o recurso chamado EntityManager para realizar esta comunicação, e este recurso permite então acessar a camada de modelo do sistema. O objetivo disto é permitir ao sistema realizar as operações básicas sobre os dados, que são incluir, excluir, edição e leitura.

São estas classes que acessam a Unidade de Persistência configurada no arquivo persistence.xml, através da anotação “@PersistenceContext”.

Para aplicar a reutilização de código implementou-se uma classe “DAOGenerico” que disponibiliza algumas funcionalidades para as classes especializadas como paginação, filtro e ordenação de resultados e pesquisa por registros no banco de dados. Com a implementação deste *DAO* genérico foi possível que as classes *DAO* especializadas ficassem mais simples devido a herança que foi

aplicada entre as classes. Na Figura 9 pode-se visualizar a classe “DAOGenerico” usada no projeto:

Figura 9 - Fragmento do arquivo DAOGenerico.java

```
public class DAOGenerico<TIPO> implements Serializable {

    @PersistenceContext(unitName = "PC2_Teste_WebPU2")
    protected EntityManager em;
    private List<TIPO> listaObjetos;
    protected Class classePersistente;
    protected String ordem = "id";
    protected String filtro = "";
    protected Integer maximoObjetos = 2;
    protected Integer posicaoAtual = 0;
    protected Integer totalObjetos = 0;

    public DAOGenerico() {
    }

    public List<TIPO> getListaObjetos() {
        String jpql = "from " + classePersistente.getSimpleName();
        String where = "";
        filtro = filtro.replaceAll("[';-]", "");
        if (getFiltro().length() > 0) {
            if (getOrdem().equals("id")) {
                try {
                    Integer.parseInt(getFiltro());
                    where += " where " + getOrdem() + " = '" +
                            getFiltro() + "' ";
                } catch (NumberFormatException e) {
                }
            } else {
                where += " where upper(" + getOrdem() + ") like '" +
                        getFiltro().toUpperCase() + "%' ";
            }
        }
        jpql += where;
        jpql += " order by " + ordem;
        totalObjetos = em.createQuery(jpql).getResultList().size();
        return em.createQuery(jpql).
                setFirstResult(posicaoAtual).
                setMaxResults(maximoObjetos).getResultList();
    }
```

Fonte - Do Autor

As classes *DAO* são containers EJB que receberam a anotação “@Stateful” para manterem o estado de cada sessão. Cada classe *DAO* especializada recebe como parâmetro uma das classes da camada de modelo. Quando necessário métodos adicionais podem ser adicionados nestas classes ou sobrescrever-se os métodos da classe genérica. Como podemos ver na Figura 10 a classe “UsuarioDAO” sobrescrevendo o método “getObjectById” e com um método novo “localizaPorNomeUsuario”:

Figura 10 - Conteúdo do arquivo UsuarioDAO.java

```
@Stateful
public class UsuarioDAO<TIPO> extends DAOGenerico<Usuario> implements Serializable {

    public UsuarioDAO() {
        super();
        classePersistente = Usuario.class;
    }

    @Override
    public Usuario getObjectById(Object id) throws Exception {
        Usuario obj = em.find(Usuario.class, id);
        obj.getPermissao().size();
        return obj;
    }

    public Usuario localizaPorNomeUsuario(String nomeUsuario){
        Query query = em.createQuery("from Usuario where upper(login) = :nomeUsuario");
        query.setParameter("nomeUsuario",nomeUsuario.toUpperCase());
        Usuario obj = (Usuario) query.getSingleResult();
        return obj;
    }
}
```

Fonte - Do Autor

3.2.2 Conversores

A camada de visão, formada pelas interfaces, é composta apenas por textos que são interpretados pelos navegadores e exibidos para os usuários. Sendo assim torna-se necessário o uso de conversores de dados para transformar os dados complexos de uma linguagem de programação em *String,* e o caminho contrário de *String* para, neste caso, objetos Java.

Para funcionar corretamente o conversor necessita utilizar uma instância da classe *EntityManager* para poder acessar as estruturas da camada de modelo. O código de um conversor pode ser visto na Figura 11:

Figura 11 - Conteúdo do arquivo ConversorUsuario.java

```
@FacesConverter(value = "conversorUsuario")
public class ConversorUsuario implements Converter, Serializable {

    @PersistenceContext(unitName = "PC2_Teste_WebPU2")
    private EntityManager em;

    @Override
    public Object getAsObject(FacesContext context, UIComponent component, String value) {
        if(value == null || value.equals("Selecione um registro")){
            return null;
        }
        return em.find(Usuario.class, Integer.parseInt(value));
    }

    @Override
    public String getAsString(FacesContext context, UIComponent component, Object value) {
        if (value == null){
            return null;
        }
        Usuario obj = (Usuario) value;
        return obj.getId().toString();
    }

}
```

Fonte - Do Autor

3.2.3 Camada de Controle

A camada de controle tem o objetivo de intermediar a comunicação entre os elementos da camada de visão e os elementos da camada de modelo, é através dos métodos desta camada que a interface consegue realizar as ações básicas nos dados persistidos como incluir, excluir, editar e ler, e também pode-se criar nesta classe outros métodos que sejam necessários para representar a regra de negócio especifica de cada classe.

Para que se possa acessar os métodos desta classe na camada de visão devemos torna-lá uma *ManagedBean,* para isto precisamos fazer a anotação *@Named(value="nomedocontrole")* e definir um ciclo de vida para este controle. Na Figura 12 podemos ver um exemplo de código de um controlador:

Figura 12 - Fragmento do arquivo ControleUsuario.java

```
@Named(value = "controleUsuario")
@ViewScoped
public class ControleUsuario implements Serializable {

    @EJB
    private UsuarioDAO<Usuario> dao;
    private Usuario objeto;
    private Boolean editando;
    @EJB
    private PermissaoDAO<Permissao> daoPermissao;
    private Permissao permissao;

    public ControleUsuario() {
        editando = false;
    }

    public void adicionarPermissao() {
        if (permissao != null) {
            if (!objeto.getPermissao().contains(permissao)) {
                objeto.getPermissao().add(permissao);
                Util.mensagemInformacao("Permissao adicionado com sucesso!");
            } else {
                Util.mensagemErro("Esta permissao já existe na sua lista!");
            }
        }
    }

    public void removerPermissao(int index) {
        permissao = objeto.getPermissao().get(index);
        objeto.getPermissao().remove(permissao);
        Util.mensagemInformacao("Permissao removida com sucesso!");
    }
```

Fonte - Do Autor

No controle de usuário (Figura 12), para cumprir um requisito, foram adicionados métodos “adicionarPermissao” e “removerPermissao”, para definir o que aquele usuário vai ou não poder acessar no sistema.

3.2.4 Camada de Visão

A camada de visão foi modelada com intuito de ser amigavel, para isto foi utilizado o framework *Primefaces*. O uso deste framework abstrai o desenvolvimento nas linguagens CSS e HTML para simplificar o desenvolvimento das interfaces.

Para reutilizarmos os códigos na camada de visão, utilizamos o componente *Facelets* do JSF que permite criar um arquivo “template.xhtml” e neste arquivo podemos criar os menus que irão ser exibidos em todo o sistema através do elemento do *Primefaces* *<p:megaMenu>,* para marcar a área da interface aonde o conteúdo de cada página deverá ser inserido utilizamos o elemento *<ui:insert name="conteudo">* onde *name* é o nome que será dado ao conteúdo que será inserido. Podemos ver uma parte desta configuração na Figura 13:

Figura 13 - Fragmento do arquivo template.xhtml

```
<f:view encoding="ISO-8859-1" contentType="text/html">
    <h:head>
        <title><ui:insert name="titulo"></ui:insert></title>
        <h:outputStylesheet library="css" name="estilos.css"/>
        <h:outputScript library="js" name="scripts.js"/>
    </h:head>
    <p:layout fullPage="true">
        <p:layoutUnit position="north" header="Sistema desenvolvido para a PC2"
                      style="text-align: center">
            <h:form id="menu">
                <p:megaMenu>
                    <p:menuitem value="Inicio" url="/index.xhtml" icon="ui-icon-home"/>
                    <p:submenu ...38 linhas />
                    <p:submenu label="Questionario">
                        <p:column ...7 linhas />
                    </p:submenu>
                    <p:submenu label="Usuario #{controleLogin.usuarioAutenticado != null ?
                                                controleLogin.usuarioAutenticado.nome : ''}">
                        <p:column ...12 linhas />
                    </p:submenu>
                </p:megaMenu>
            </h:form>
        </p:layoutUnit>
        <p:layoutUnit position="center">
            <ui:insert name="conteudo">
            </ui:insert>
        </p:layoutUnit>
        <ui:insert name="dialogos">

        </ui:insert>
        <p:layoutUnit position="south">
            <ui:include src="ajaxstatus.xhtml"/>
            <div align="center">Tiago Dellinghausen Lopes</div>
        </p:layoutUnit>
    </p:layout>
</f:view>
```

Fonte - Do Autor

Ao iniciar o sistema deve carregar o arquivo index.xhtml, neste arquivo foi definida a página inicial do sistema que através do uso do JSF com o elemento

<ui:composition> a página é integrada ao *template* e com o elemento *<ui:define>* foram definidos o título da página e seu conteúdo como podemos ver na Figura 14:

Figura 14 - Fragmento do arquivo index.xhtml

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
    <ui:composition template="/templates/template.xhtml">
        <ui:define name="titulo">Sistema PC2</ui:define>
        <ui:define name="conteudo">

        </ui:define>
    </ui:composition>
</html>
```

Fonte - Do Autor

São apresentadas na interface cada uma das manutenções via um código que está armazenado pelo respectivo arquivo de listagem, estes arquivos acessam via controlador os métodos que permitem listar os dados, filtrar e páginar os resultados. Na Figura 15 podemos ver um exemplo do arquivo listar da classe Cidade:

Figura 15 - Fragmento do arquivo listar.xhtml

```
<ui:composition template="/templates/template.xhtml">
    <ui:define name="titulo">Manutenção de Cidades</ui:define>
    <ui:define name="conteudo">
        <h:form id='formListagem'>
            <p:growl/>
            <p:commandButton value="Novo" actionListener="#{controleCidade.novo()}"
                             icon="ui-icon-plus" oncomplete="PF('dlg').show();" update="formEdicao"/>
            <p:panelGrid columns="11">
                <p:outputLabel value="Ordem:"/>
                <p:selectOneMenu value='#{controleCidade.dao.ordem}'>
                    <f:selectItem itemLabel="ID" itemValue="id"/>
                    <f:selectItem itemLabel="Nome" itemValue="nome"/>
                    <p:ajax update="listagem"/>
                </p:selectOneMenu>
                <p:outputLabel value="Filtro:"/>
                <p:inputText value='#{controleCidade.dao.filtro}' size='15'>
                    <p:ajax update="listagem" event="keyup"/>
                </p:inputText>
                <p:commandButton value="Filtrar" update="listagem" icon="ui-icon-search"/>
                <p:outputLabel value="Resultados por página: "/>
                <p:inputNumber value='#{controleCidade.dao.maximoObjetos}' decimalPlaces="0" decimalSeparator="" thousandSeparator=""/>
                <p:commandButton value="Primeiro" actionListener="#{controleCidade.dao.primeiro()}" icon="ui-icon-seek-first" update="listagem"/>
                <p:commandButton value="Anterior" actionListener="#{controleCidade.dao.anterior()}" icon="ui-icon-seek-prev" update="listagem"/>
                <p:commandButton value="Proximo" actionListener="#{controleCidade.dao.proximo()}" icon="ui-icon-seek-next" update="listagem"/>
                <p:commandButton value="Ultimo" actionListener="#{controleCidade.dao.ultimo()}" icon="ui-icon-seek-end" update="listagem"/>
            </p:panelGrid>
            <p:dataTable value="#{controleCidade.dao.listaObjetos}" var="obj" id='listagem'>
                <f:facet name="footer">
                    <p:outputLabel value="#{controleCidade.dao.mensagemNavegacao}"/>
                </f:facet>
```

Fonte - Do Autor

Já para a inclusão e edição dos dados foram criados modais através do elemento <p:dialog> do Primefaces que permite colocar este código dentro do mesmo arquivo listar, e permite acessar alterar e acrescentar dados sem sair da tela de listagem. Na Figura 16 está a parte referente a este modal ainda no arquivo listar da classe Cidade:

Figura 16 - Fragmento do arquivo listar.xhtml

```
<ui:define name="dialogos">
    <p:dialog widgetVar="dlg" header="Edição de Cidade" modal="true" resizable="false">
        <h:form id="formEdicao">
            <p:growl/>
            <div align="center">
                <p:panelGrid columns="2">
                    <f:facet name="footer">
                        <div align="center">
                            <p:commandButton value="Salvar"
                                             icon="ui-icon-disk"
                                             actionListener="#{controleCidade.salvar()}"
                                             oncomplete="if(!args.validationFailed){PF('dlg').hide();}"
                                             update=":formEdicao :formListagem"/>
                        </div>
                    </f:facet>
                    <p:outputLabel value="ID"/>
                    <p:inputText value="#{controleCidade.objeto.id}" readonly="true" size="5"/>
                    <p:outputLabel value="Nome" for="txtNome"/>
                    <p:inputText id="txtNome" value="#{controleCidade.objeto.nome}" size="40" maxlength="40"/>
                    <p:outputLabel value="Estado" for="selectEstado"/>
                    <p:selectOneMenu id="selectEstado" value="#{controleCidade.objeto.estado}">
                        <f:converter converterId="conversorEstado"/>
                        <f:selectItem itemLabel="Selecione um registro" noSelectionOption="true"/>
                        <f:selectItems value="#{controleCidade.daoEstado.listaTodos}" var="e" itemLabel="#{e.nome}"/>
                    </p:selectOneMenu>
                </p:panelGrid>
            </div>
            <ui:include src="/templates/ajaxstatus.xhtml"/>
        </h:form>
    </p:dialog>
</ui:define>
```

Fonte - Do Autor

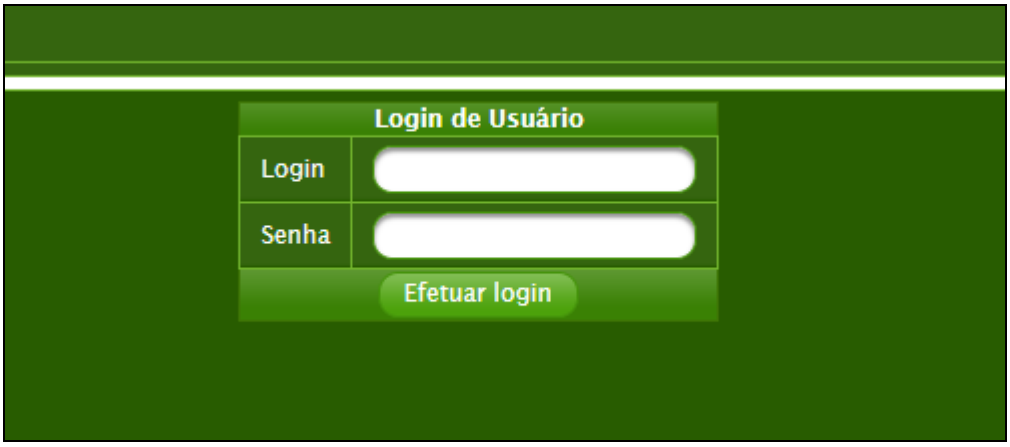
4 RESULTADOS

Com o objetivo de qualificar e facilitar a captação de dados dos alunos do IFSUL foi desenvolvido um sistema web para ajudar neste trabalho. Ao concluir este sistema tornou-se possível colher de forma digital os dados que os alunos preenchem no sistema, e manter estes dados em um banco de dados digital permitindo gerar relatórios a partir destes dados.

O sistema tem validação de segurança para o tipo de pessoa que está acessado o sistema, poderão ser Administrador ou Aluno. O Administrador tem acesso a todas as funcionalidades do sistema e pode alterar as permissões de outros usuários, e acessar os relatórios. Já o aluno poderá apenas preencher seus dados pessoais e o questionário que foi cadastrado para ele responder.

Para um aluno poder responder o questionário, primeiramente este deve acessar o sistema através da tela de *login*. Para obter sucesso no *login* o aluno deve estar previamente cadastrado. Na Figura 17 pode-se ver a tela de *login* do sistema:

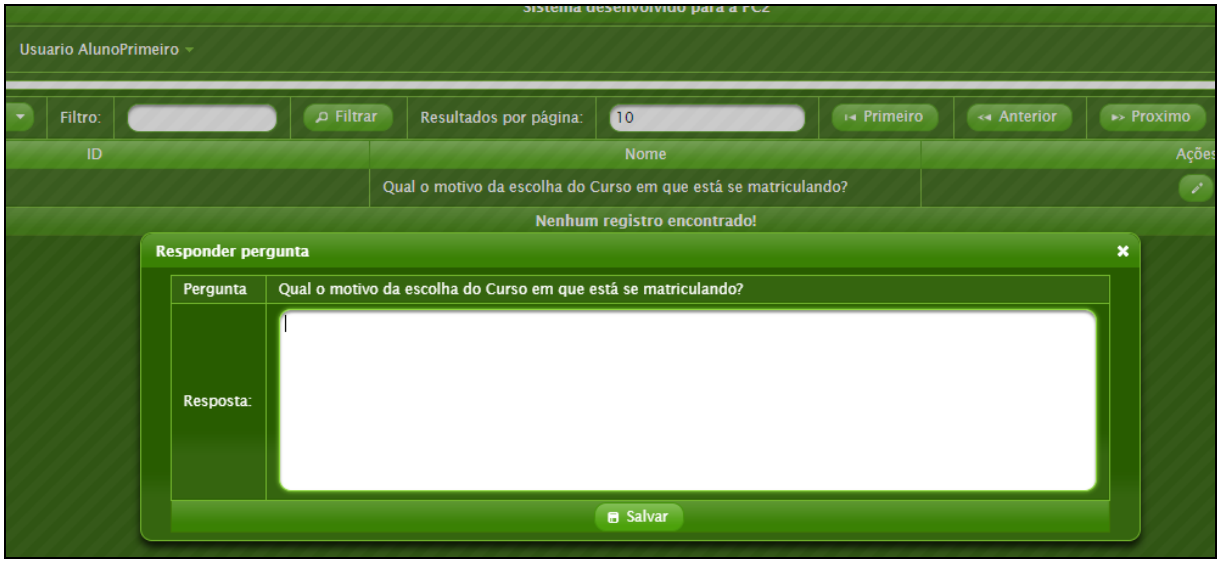
Figura 17 - Tela de Login



Fonte - Do Autor

Assim que faz o *login* o usuário com permissão de estudante tem acesso ao questionário que lhe foi previamente liberado, este questionário tem uma lista de perguntas cadastradas. Na Figura 18 podemos ver o exemplo da tela de pergunta com o campo para resposta:

Figura 18 - Tela de Pergunta/Resposta



Fonte - Do Autor

No caso dos usuários com permissão de administrador, estes têm acesso a toda a área de cadastros conforme podemos ver no menu apresentado na

Figura 19:

Figura 19 - Tela do menu de administrador

Fonte - Do Autor

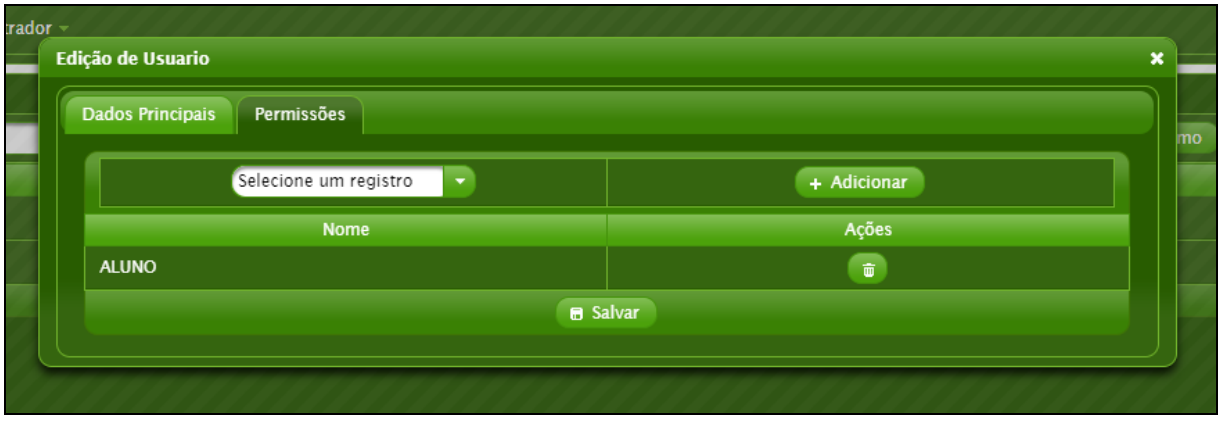
Ao acessar a tela de manutenção de usuários são listados os cadastrados no sistema, e pode acrescentar um novo usuário e editar ou remover um já exigente, pode-se ver a tela de manutenção de usuários na Figura 20:

Figura 20 - Tela de Manutenção de Usuários

Fonte - Do Autor

Ao acessar a área para um novo usuário ou editar um já existente é apresentada a tela de edição de usuários que apresenta os campos para todo os dados pertinentes ao usuário e na aba Permissões é possível adicionar ou remover permissões, esta tela pode ser vista na Figura 21:

Figura 21 - Tela de Usuário/Permissão



Fonte - Do Autor

Uma das telas principais do sistema é a tela de Edição de Estudantes que permite inserir os dados do cadastro do estudante, além dos dados cadastrais esta tela dá acesso a outras duas abas que permitem gerir as necessidades especificas do estudante e as ações posteriores. Na Figura 22, pode-se ver a tela com os dados principais do estudante:

Figura 22 - Tela de cadastro

Fonte - Do Autor

Na aba lista de necessidades é possível adicionar uma necessidade, previamente cadastrada, a o aluno que está sendo cadastrado ou editado, e remover uma necessidade que já havia sido colocada anteriormente. Na Figura 23, pode-se ver a aba que faz esta manutenção:

Figura 23 - Aba de necessidades

Fonte - Do Autor

Já na aba de ações posteriores é possível cadastrar uma ação que tenha sido tomada junto ao aluno que está sendo cadastrado ou editado, ao clicar em nova ação uma nova aba é aberta permitindo inserir os dados da Ação posterior. Pode-se ver na Figura 24 a tela para a adição dos dados da Ação posterior com a tela Ações Posteriores em segundo plano:

Figura 24 - Aba de ações posteriores

Fonte - Do Autor

Para responder uma pergunta o aluno pode acessar o menu questionário e será apresentada a primeira pergunta do questionário que ele estiver inscrito para responder

5 Considerações Finais

Este estudo teve como objetivo o desenvolvimento de uma aplicação de sistema de mapeamento de dados de alunos com necessidades especificas. Esta aplicação procurou resolver a situação da coleta dos dados dos alunos com necessidades especificas e também sua manutenção e acessibilidade. Ao usar o software deve-se diminuir o trabalho manual gerado pela coleta, armazenamento e busca dos dados.

Tendo isto em vista a aplicação desenvolvida atingiu em parte o seu objetivo, conseguindo coletar, e apresentar as respostas descritivas dos portadores de necessidades especificas, além de permitir criar questionários diferentes para posterior aplicação.

Aplicação procurou atender a necessidade do NAPNE aplicando várias tecnologias para o desenvolvimento de uma solução, entre as tecnologias utilizadas estão a o PostgreSQL como banco de dados objeto relacional, a JPA que permite a comunicação com o banco de dados, a coleção de APIs do Java EE 7 que permite utilizar os containers EJB para apresentar os dados na camada de visão e o Primefaces e o BootsFaces para apresentar um layout responsivo.

Então ao atingir parcialmente os objetivos não foi possível testá-lo em um ambiente real, portanto não sendo possível aferir sua real utilidade para aqueles que poderiam vir a utilizá-lo. Pode-se apenas analizá-lo em testes pelo desenvolvedor que conclui que as funcionalidades que foram possíveis de implantar atingiram seu objetivo principal de reduzir o trabalho manual, sendo possível com mais tempo de desenvolvimento chegar ao resultado desejado.

Em trabalhos futuros pretende-se desenvolver as perguntas com múltiplas escolhas, e os relatórios do sistema que permitirá aos funcionários do NAPNE acessar as informações necessárias.

6 REFERÊNCIAS

DA ROSA, Luciano Heleno. **Sistema Para Elaboração E Disponibilização De Questionários De Avaliação**. 2015. 51p. Trabalho de Conclusão de Curso (Especialista e III Curso de Especialização em Tecnologia Java) - Departamento Acadêmico De Informática, Universidade Tecnológica Federal do Paraná, Pato Branco, PR, 2015.

DEITEL, Paul J.; DEITEL, Harvey M. **Java: como programar.** 8. ed. São Paulo, SP: Pearson, 2010. 1144 p. ISBN 8576055631.

GEARY, David; HORSTMANN, Cay. **Core JavaServer Faces.** Rio de Janeiro, RJ: Alta Books, 2012. 636 p. ISBN 9788576086420.

GONCALVES, Antônio. **Introdução à plataforma Java (TM) EE 6 com o glassFish (TM)** 3. 2. ed. Rio de Janeiro: Ciência Moderna, 2011. 563 p. ISBN 9788539900961.

GUEDES, Gilleanes T. A. **UML 2: uma abordagem prática**. 2. ed. São Paulo, SP: Novatec, 2011. 484 p. ISBN 9788575222812.

POSTGRESQL. **About.** Disponível em https://www.postgresql.org/about/. Acesso em 15 de maio de 2018.

ANEXO A

FORMULÁRIO DE NECESSIDADES EDUCACIONAIS E DE SAÚDE DO ESTUDANTE

Objetivo: Este formulário tem por objetivo conhecer os estudantes de nossa instituição, visando realizar as adequações necessárias para melhorar o processo de aprendizagem.

1. Dados de identificação do estudante:

Nome

Cidade que reside: Telefone:

Data de Nascimento: Curso:

Em caso de emergência avisar (nome e telefone):

2. Responda as seguintes questões de acordo com sua vivência escolar até hoje:

a) Onde cursou o Ensino Fundamental?

( ) Todo em escola pública ( ) Todo em escola particular

( ) Maior parte em escola pública ( ) Maior parte em escola particular

b) Em alguma fase de sua vida interrompeu seus estudos?

( ) Não ( ) Sim, Por quê? _____

c) Apresentou dificuldades em alguma disciplina em seu processo escolar?

( ) Não ( ) Sim, Qual/ quais? _____

Por quê? _____

d) Você já precisou repetir alguma etapa no seu processo escolar antes de ingressar no IFSUL?

( ) Não ( ) Sim, Qual /quais disciplinas: _____

Em que série? _____

e) Qual o motivo da escolha do Curso em que está se matriculando?

_____

_____

f) Qual é a sua rotina de estudos?

( ) Diária ( ) Algumas vezes por semana ( ) Somente na hora das provas

g) Quem mais o incentiva a estudar? _____

h) Você possui alguma dificuldade em:

( ) leitura e/ou escrita ( ) atenção e/ou concentração ( ) aprendizagem

i) Você possui diagnóstico de Altas habilidades/Superdotação?

( ) Não ( ) Sim

Caso possua altas habilidades, apresenta alguma outra particularidade associada?

( ) Não ( ) Sim, qual? _____

3. Você possui alguma deficiência?

Se sim, marque abaixo:

( ) Visual
Caso possua alguma dificuldade visual, você precisa de algum recurso para melhorar a visão (óculos, lentes...)? ( ) Não ( ) Sim, qual? _____ Ainda, caso possua alguma dificuldade visual, necessita de alguma adaptação curricular? Como material ampliado, cor específica de tela, áudio descrição do material, braile, entre outros. Qual? _____
( ) Auditiva
( ) faz uso parelho? ( ) usa sistema libras?
( ) Física/motora
Necessita de alguma adaptação curricular? Qual? _____
( ) Intelectual/mental
( ) Aprende melhor a partir de materiais mais visuais. ( ) Aprende melhor ouvindo. ( ) Aprende melhor fazendo
( ) Múltipla
Nesse caso identifique-as: _____

( ) Outra necessidade específica
Nesse caso identifique-as: _____

a) Você possui algum diagnóstico médico associado a alguma das deficiências acima?

( ) Não ( ) Sim, qual: _____

4. Quanto a sua saúde, responda:

a) Você possui algum problema de saúde?

( ) Não ( ) Sim, qual ordem:

( ) Cardiovascular/Problemas do coração
( ) Oncológica/Câncer
( ) Saúde Mental
( ) Diabetes
( ) Hipertensão
( ) Doenças respiratórias
( ) Alergias
( ) Doenças neurológicas
( ) Outras, quais: _____

b) **Utiliza medicamentos contínuos?**

( ) Não ( ) Sim, quais: _____

c) **Utiliza alguma destas substâncias SEMPRE OU FREQUENTEMENTE?**

( ) Álcool ( ) Cigarro ( ) Outras Drogas ( ) Não utilizo

d) **Quando necessita de atendimento de saúde, utiliza:**

( ) SUS ( ) Serviço Particular ( ) Plano de Saúde, qual: _____

5. OUTRAS INFORMAÇÕES RELEVANTES: