UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE ESPECIALIZAÇÃO EM TECNOLOGIA JAVA

TALYSSON DE CASTRO

DESENVOLVIMENTO DE UM PORTAL DE EGRESSOS

MONOGRAFIA DE ESPECIALIZAÇÃO

PATO BRANCO
2017

TALYSSON DE CASTRO

DESENVOLVIMENTO DE UM PORTAL DE EGRESSOS

Trabalho de Conclusão de Curso, apresentado ao Curso de Especialização em Tecnologia Java, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Especialista.

Orientadora: Profa. Andreia Scariot Beulke

PATO BRANCO
2017

MINISTÉRIO DA EDUCAÇÃO
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Especialização em Tecnologia Java

UTFPR

TERMO DE APROVAÇÃO

DESENVOLVIMENTO DE UM PORTAL DE EGRESSOS

por

TALYSSON DE CASTRO

Este trabalho de conclusão de curso foi apresentado em 08 de dezembro de 2017, como requisito parcial para a obtenção do título de Especialista em Tecnologia Java. Após a apresentação o candidato foi arguido pela banca examinadora composta pelos professores Beatriz Terezinha Borsoi e Vinicius Pegorini. Em seguida foi realizada a deliberação pela banca examinadora que considerou o trabalho aprovado.

Andreia Scariot Beulke
Profa Orientadora (UTFPR)

Beatriz Terezinha Borsoi
Banca (UTFPR)

Vinicius Pegorini
Banca (UTFPR)

Robison Cris Brito
Coordenador da IV Especialização em Tecnologia Java

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

RESUMO

CASTRO, Talysson de. Desenvolvimento de um portal de egressos. 2017. 62 f. Monografia (Trabalho de especialização) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2017.

Uma Universidade representa um papel significativo na vida de um egresso, pois é por meio dela que o egresso se diplomou e adquiriu os conhecimentos necessários para sua carreira profissional. Por isso, é importante que a Universidade e o egresso mantenham relações por meio de algum sistema com o objetivo de coletar informações sobre sua trajetória na Universidade e na carreira profissional. Nesse contexto, o objetivo desse trabalho foi desenvolver uma aplicação *web*, denominada de portal de egressos, para que a Universidade e o egresso possam manter um vínculo por meio de informações cadastradas no sistema. O sistema possui cadastros para identificar um histórico profissional do egresso, permite a realização de comentários sobre a Universidade por parte do egresso e, também, permite a composição de questionários disponibilizados para que os egressos respondam. O sistema também possibilita realizar consultas no conteúdo dos comentários realizados pelos egressos. O desenvolvimento do sistema foi baseado na linguagem Java, utilizando o *framework* Spring no *banck-end* e o *framework* Angular no *front-end*, juntamente com o *framework* JHispter para auxiliar na criação de classes e integração entre ambas as partes. O JHispter proporciona desenvolvimento rápido resultando em um sistema padronizado tanto na codificação quanto nas telas para a interação do usuário com o sistema.

Palavras-chave: Universidade. Egresso. Angular.

ABSTRACT

CASTRO, Talysson de. A web portal to university graduated. 2017. 62 f. Monografia (Trabalho de especialização) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2017.

The university plays a significant role in the life of a graduated student, because they provide the knowledge necessary for his professional career. Therefore, it is important that the university and the graduated maintain relations through some system with the objective of collecting information about its trajectory in the university and in the professional career. In this context, the objective of this work was to develop a web application, with name of portal de egressos, so the university and the students can maintain a relation through information registered in the system. The system has registers to identify a professional history about the student, allows comments on the University by the student, and also allows the composition of questionnaires available to answer by e=student. It also allows to do some searches on the comments of each egress. The development of the system was based on the Java language, using the Spring framework in the back end and the Angular framework on the front end, along with the JHispter framework to aid in class creation and integration between both sides. The result was a rapid development provided by JHispter and a standardized system both in coding and in screens for user interaction and the system.

Keywords: University. Graduated. Angular.

LISTA DE FIGURAS

Figura 1 – Criação de uma aplicação baseada no JHipster 17
Figura 2 – Diagrama de casos de uso..... 20
Figura 3 – Diagrama entidades e relacionamentos do sistema..... 26
Figura 4 – Diagrama de entidades e relacionamentos do webservice 26
Figura 5 – Tela inicial do sistema..... 27
Figura 6 – Tela de autenticação do sistema..... 27
Figura 7 – Tela inicial do sistema..... 28
Figura 8 – Listagem de tecnologias 29
Figura 9 – Tela de cadastro de tecnologia..... 29
Figura 10 – Tela de edição de uma tecnologia 30
Figura 11 – Tela de listagem de empresas 30
Figura 12 – Tela de inserção e edição de atividade profissional..... 31
Figura 13 – Tela de listagem das atividades profissionais..... 31
Figura 14 – Tela de relacionamento de atividade profissional e tecnologia 32
Figura 15 – Tela de listagem de atividade profissional e tecnologia..... 32
Figura 16 – Tela de inserção ou edição de histórico..... 33
Figura 17 – Tela de listagem de histórico de egressos..... 33
Figura 18 – Tela de inserção ou edição de perguntas do questionário 34
Figura 19 – Tela de listagem de perguntas do questionário..... 34
Figura 20 – Tela de inserção ou edição de alternativas do questionário..... 35
Figura 21 – Tela de listagem de alternativas do questionário 35
Figura 22 – Tela de inserção ou edição de questionário..... 36
Figura 23 – Tela de listagem dos questionários 36
Figura 24 – tela de relacionamento de pergunta e questionário. 37
Figura 25 – Tela de listagem de perguntas e questionários..... 37
Figura 26 – Tela de questionários disponíveis para serem respondidos..... 38
Figura 27 – Tela de pergunta e alternativas de resposta 38
Figura 28 – Tela com mensagem de questionário finalizado..... 38
Figura 29 – Tela de inserção de comentários..... 39
Figura 30 – Tela de listagem de comentários..... 39
Figura 31 – Tela de validação de comentários 39
Figura 32 – Tela de consulta de histórico 40
Figura 33 – Tela de consulta de comentários..... 40
Figura 34 – Tela de consulta de questionários respondidos..... 41
Figura 35 – Tela dos detalhes do questionário respondido 41
Figura 36 – Tela de consulta de questionário não respondidos..... 41
Figura 37 – Tela de inserção e edição de usuário..... 42
Figura 38 - Tela de listagem de usuários 43

LISTA DE QUADROS

Quadro 1 - Tecnologias e ferramentas utilizadas na modelagem e na implementação do aplicativo 14
Quadro 2 – Requisitos Funcionais 19
Quadro 3 – Requisitos não-funcionais 20
Quadro 4 – Operação “incluir” dos casos de uso de cadastro 21
Quadro 5 – Operação “alterar” dos casos de uso de cadastro 22
Quadro 6 – Operação “excluir” dos casos de uso de cadastro 22
Quadro 7 – Operação “consultar” dos casos de uso de cadastro 22
Quadro 8 – Caso de uso “Validar comentários” 23
Quadro 9 – Caso de uso “Realizar comentários” 24
Quadro 10 – Caso de uso “Responder questionário” 24
Quadro 11 – Caso de uso “Disponibilizar questionário” 25

LISTAGENS DE CÓDIGOS

Listagem 1 – Instalação do Node e NPM 16
Listagem 2 – Configuração da classe *SiriusuniversityApp* 44
Listagem 3 - Método *configure* para realizar as configurações de acesso às *APIs* 45
Listagem 4 – Validação do token pelo filter do servlet 45
Listagem 5 – Classe de configuração de serialização e deserialização de objetos 46
Listagem 6 – Configuração para atualização da estrutura de banco de dados 46
Listagem 7 – Arquivos de atualização da estrutura de banco de dados 48
Listagem 8 – Configuração de acesso ao banco de dados 49
Listagem 9 – Classe da entidade Alternativa 49
Listagem 10 – Classe responsável pela comunicação com banco de dados 50
Listagem 11 – Classe responsável por executar regras de negócio 51
Listagem 12 – Classe de exposição de métodos à *API* 53
Listagem 13 – Interceptador de requisições via HTTP 53
Listagem 14 – Método de login do usuário no sistema 54
Listagem 15 – Módulos disponíveis no sistema 55
Listagem 16 – Opções de menu do sistema 56
Listagem 17 – Listagem de registros cadastrados no sistema 57
Listagem 18 – Classe responsável pelo controle da tela de listagem 58
Listagem 19 – Classe com a definição do modelo do objeto 58
Listagem 20 – Rotas disponíveis do sistema 59
Listagem 21 – Métodos para realizar requisições ao back-end 60
Listagem 22 – Definição do módulo da entidade 61

LISTA DE SIGLAS

API	*Application Programming Interface*
CSS	*Cascading Style Sheet*
CRUD	*Create, Retrieve, Update and Delete*
DER	Diagrama de Entidade e Relacionamento
ES6	ECMAScript 6
HTML	*Hypertext Markup Language*
HTTP	*Hypertext Transfer Protocol*
IES	Instituição de Ensino Superior
JAR	Java ARchive
JPA	*Java Persistence API*
JWT	*JSON Web Token*
MVC	*Model View Controller*
NPM	Node Package Manager
ORM	*Object Relational Mapping*
REST	*Representational State Transfer*
RF	Requisito Funcional
RNF	Requisito Não-Funcional.
SPA	*Single Page Application*
URL	*Uniform Resource Locator*

SUMÁRIO

1 INTRODUÇÃO 11
2 MATERIAIS E PROCEDIMENTOS 13
2.1 MATERIAIS 13
2. PROCEDIMENTOS E CONFIGURAÇÕES 16
3 RESULTADOS 18
3.1 ESCOPO DO SISTEMA 18
3.2 MODELAGEM DO SISTEMA 18
3.3 APRESENTAÇÃO DO SISTEMA 27
3.4 IMPLEMENTAÇÃO DO SISTEMA 43
4 CONCLUSÃO 61
REFERÊNCIAS 62

1 INTRODUÇÃO

Uma Instituição de Ensino Superior (IES) é um ambiente organizacional que pode ser público ou privado e tem por finalidade promover ensino, pesquisa e extensão. Essa é uma características mais especificamente das Universidades, cuja denominação caracteriza a indissociabilidade entre esses três pilares (BRASIL, 1988).

De acordo com a Lei nº 9394, de 20 de dezembro de 1996, que estabelece as diretrizes e bases da Educação Nacional, sobre a educação superior, têm, dentre outras diretrizes, estimular a criação cultural e o desenvolvimento do espírito científico e do pensamento reflexivo e formar diplomados nas diferentes áreas de conhecimentos.

Nesse sentido, uma Universidade tem a missão de fomentar, construir e disseminar o conhecimento, contribuindo para a formação do acadêmico e do desenvolvimento humano. Em sua diversidade de cursos, seja de graduação ou pós-graduação, uma Universidade forma um número acentuado de alunos que exercem papel fundamental na sociedade seja em regiões que priorizam as atividades empresariais voltadas para a pesquisa, desenvolvimento e inovação, ou regiões voltadas para o desenvolvimento comercial e industrial que tem a necessidade de incorporar informação e tecnologia nas economias (PEREIRA *et al.* 2016).

Nesse aspecto, é relevante compreender a definição de egresso que na concepção de Guimarães e Salles (s.d.), é toda pessoa que concluiu um curso superior ou qualquer capacitação profissional em uma instituição de ensino. Esses autores afirmam, ainda, que há certa divergência com relação a essa definição, pois alguns estudiosos afirmam que egresso é a denominação de todos os indivíduos que saíram do sistema escolar, com ou sem diploma (desistência, transferência ou jubilados). Para Ferreira (1999), o termo egresso é aplicado a todo indivíduo que cumpriu a grade curricular de um curso de graduação ou pós-graduação. Sob essa mesma ótica, a legislação da área educacional compreende que egresso é a pessoa que efetivamente concluiu seus estudos (BRASIL, 1996).

Sabendo que o egresso representa um importante papel na sociedade e que uma das missões de uma Universidade é contribuir com a formação integral de seus acadêmicos que vai além de competências técnicas exigidas pelo mercado de trabalho, com a consolidação de princípios, valores e comportamentos. Para Guimarães e Salles (s.d.) a diferença entre o que o mercado de trabalho exige e o que a Universidade oferece pode ser visualizada por meio de significativas reformas que vão além da transmissão de conhecimentos. Para esses autores, é interessante que a Universidade tenha uma politica de acompanhamento que visa

disponibilizar informações necessárias (pessoais, acadêmicas e profissionais) de alunos egressos.

Nesse cenário, este trabalho tem por objetivo desenvolver um aplicativo *web* para acompanhamento de egressos. Esse aplicativo funciona como um canal de comunicação entre a Universidade e o mercado de trabalho por meio de informações sobre os egressos, como, por exemplo, empresa em que atua e/ou atuou, tecnologias utilizadas, área de atuação, funções exercidas, histórico, entre outras. Além disso, o sistema permite que o egresso responda questionários e faça comentários sobre sua experiência com a Universidade e o campo de atuação. Essas e outras funcionalidades, o sistema permite que a Universidade não perca o vínculo com seus egressos e obtenha informações relacionadas à formação e atividades que eles desenvolvem.

2 MATERIAIS E PROCEDIMENTOS

Esse capítulo tem por objetivo elencar os materiais utilizados na modelagem e na implementação do aplicativo para o desenvolvimento do trabalho proposto. Também são apresentados os procedimentos técnicos essenciais para o uso das tecnologias empregadas no desenvolvimento do trabalho.

2.1 MATERIAIS

O Quadro 1 apresenta as ferramentas e as tecnologias utilizadas na modelagem e no desenvolvimento do aplicativo.

Ferramenta/ Tecnologia	**Versão**	**Finalidade**	**Disponível em**
Linguagem Java	JDK 8	Linguagem de programação orientada a objetos para a construção do projeto lado servidor.	http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
Apache Maven	3.3.9	Gerenciamento de dependências e *build*.	https://maven.apache.org
Spring Framework	5.0.0	Framework Java para gerenciar processos básicos permitindo foco na codificação da aplicação.	https://spring.io
Postgres	9.6.2	Banco de dados relacional para armazenamento de informações.	https://www.postgresql.org
PgAdmin	1.22.2	Gerenciador de banco de dados.	https://www.pgadmin.org
Hibernate ORM	5.0.12	Mapeamento objeto-relacional para persistência e consulta de informações.	http://hibernate.org/orm/
IntelliJ	2017.1	IDE de desenvolvimento	https://www.jetbrains.com/idea/
HTML	5	Linguagem de marcação de textos para desenvolvimento da interface da aplicação.	https://www.w3schools.com/html/
CSS	3	Mecanismo para adicionar estilização aos elementos *Hypertext Markup Language* (HTML).	https://www.w3schools.com/css/
JavaScript		Linguagem de programação executada no lado cliente da aplicação.	https://www.w3schools.com/js/
Angular	4.0	Plataforma de	https://angular.io

		desenvolvimento para aplicação *Web*.	
SQL Power Architect	1.0.8	Modelagem do banco de dados.	http://www.sqlpower.ca/page/architect
Visual Paradigm	14.1	Modelagem do sistema.	https://www.visual-paradigm.com/download/community.jsp

Quadro 1 - Tecnologias e ferramentas utilizadas na modelagem e na implementação do aplicativo

2.1.1 Angular

Angular (a partir da versão 2.0) é um *framework* que utiliza a linguagem TypeScript desenvolvido e mantido pela Google. Segundo Eis et al. (2014), esse *framework* foi desenvolvido com base no conceito de que a programação declarativa deve ser utilizada no desenvolvimento de interfaces e componentes e a programação imperativa deve ser empregada na implementação das regras de negócio. Almeida (2015) informa que Angular é um *framework* que utiliza o padrão *Model, View* e *Controller* (MVC) que funciona do lado cliente e que trabalha com *Hypertext Markup Language* (HTML), *Cascading Style Sheet* (CSS) e JavaScript no lado cliente. Para Guedes (2017, p.55), Angular é um *framework Single Page Application* (SPA) para desenvolvimento *front-end*, com uma característica importante que é o conceito de modularização. Portanto, a principal característica do Angular é a criação de SPA que é um padrão utilizado para não recarregar a página durante seu uso, carregando todo o JavaScript e CSS de que precisa quando o cliente carregar a aplicação pela primeira vez (ALMEIDA, 2015). Além disso, o Angular é composto por uma série de bibliotecas, sendo a mais importante o *core* que contém todo o código base do Angular (KAZALE, 2015).

Para Pereira (2014), o Angular apresenta uma série de características, como, o uso de diretivas, serviços e injetor de dependências. Guedes (2017) endossa essa afirmação e complementa que uma aplicação desenvolvida com esse *framework* utiliza o conceito de componentes, *templates*, *data binding* e *metadatas*. Pereira (2014) e Guedes (2017) explicitam sobre esses conceitos:

a) Componente: é uma classe que permite controlar a *view*. É no componente que é definida a lógica que será aplicada para controle e ações da *view*.
b) *Template*: define a *view* do componente, pois é criado em HTML e com algumas marcações específicas do Angular.

c) *Metadata*: informa como processar uma classe por meio de um *decorator* que fica responsável por fazer a configuração entre a classe e o *template*.
d) *Data binding*: sincroniza os dados entre classe do componente e o *template*. As informações da classe podem ser mostradas no *template* ou informações do *template* podem ser executadas na classe do componente.
e) Diretiva: permite interação com o *template*, pois interagem dinamicamente com o HTML.

A modularização permite o desenvolvimento em pequenas partes que, quando juntadas, compõem a aplicação (GUEDES, 2017). Cada módulo tem sua responsabilidade e que pode ser facilmente testado, mantendo o código mais simples.

Outro conceito importante do Angular é o serviço. De acordo com Guedes (2017) um serviço é uma classe que por meio do *decorator @Injectable* é possível executar algumas funções, regras de negócio, métodos ou procedimentos que não fazem parte dos componentes e que não se limitam a apenas buscar e enviar dados para o servidor. Assim, como todos os serviços são instanciados, ele só será criado quando for solicitado pela aplicação (EIS *et al.* 2014).

Ao criar um componente, o Angular verifica quais são suas dependências e solicita que sejam injetadas todas as instâncias necessárias. Isso ocorre por meio do gerenciador de injeção de dependências que guarda todas as instâncias do projeto. Guedes (2017) e Almeida (2015) afirmam que é importante declarar todas as dependências no construtor para organizar o código e verificar quais são as dependências que o componente necessita.

Na versão 2 do Angular foram inseridos novos conceitos de desenvolvimento, surgiu o lançamento da linguagem TypeScript e, também, incorporou funcionalidades do ECMAScript 6 (ES6). O TypeScript é uma linguagem para desenvolvimento JavaScript em larga escala. Com TypeScript é possível escrever códigos utilizando estruturas fortemente tipadas e ter código compilado para JavaScript (TYPESCRIPT, 2017). Com TypeScript é possível implementar seguindo os conceitos da Orientação a Objetos e, ainda, acrescenta uma variedade de sintaxe útil e ferramentas sobre a linguagem JavaScript (GUEDES, 2017, p. 14). TypeScript é compilado para JavaScript e, portanto, é interpretado pelos navegadores.

A versão 4 (lançada em março de 2017) apresenta uma significativa atualização na parte de minificação e velocidade. Para Guedes (2017, p. 256) “com as novas atualizações, a nova versão do *build* ficou menor e mais rápida, reduzindo em até 60% o código gerado pelos componentes”. Esse autor cita que as novas *features* de desenvolvimento criadas são

relacionadas à edição do *else* dentro dos *templates*, a implementação de parâmetros no laço *for* e a validação automática de *e-mail* em formulários.

Antes de iniciar um projeto em Angular, é essencial conhecer a sua arquitetura, compreender seu funcionamento e seus conceitos principais. Além disso, é importante saber configurar o ambiente de desenvolvimento do projeto utilizando esse *framework*.

2. PROCEDIMENTOS E CONFIGURAÇÕES

Para o desenvolvimento da aplicação foi utilizado o *framework* JHipster, uma plataforma para gerar, desenvolver e entregar aplicações *web* e microsserviços Spring utilizando Spring Boot e Angular.

A instalação do JHipster é realizada via terminal de comando, utilizando o *Node Package Manager* (NPM) que é um gerenciador de pacotes do Node JS. Para isso, é preciso possuir o Node e o NPM instalados na máquina, que pode ser realizado utilizando os comandos apresentados na Listagem 1:

```
sudo apt-get install –y nodejs
sudo apt-get install npm
```

Listagem 1 – Instalação do Node e NPM

Como o JHipster é baseado no Yoman, sua instalação é um pré-requisito para o funcionamento do JHispter. A instalação é obtida por meio do comando: `sudo npm install –g yo.`

Por fim, a instalação do JHipster é realizada pelo seguinte comando: `npm install –g generator-jhipster.`

Após esses procedimentos, o gerador JHipster está instalado e pronto para ser utilizado. Com isso, é possível criar a aplicação e configurá-la utilizando linha de comando e selecionando as opções disponibilizadas pelo JHispter.

A Figura 1 demonstra a criação de uma aplicação baseada no JHipster, após selecionar as opções desejadas, o gerador irá construir uma base para aplicação utilizando Spring Boot e Angular 4.

Figura 1 – Criação de uma aplicação baseada no JHipster

O *back-end* pode ser executado normalmente como um Java ARchive (JAR), utilizando o comando *java –jar*. O *front-end* é empacotado juntamente com o *back-end*. Contudo, para o desenvolvimento se torna mais prático executar em modo *live reload* por meio da execução do comando `npm start`.

3 RESULTADOS

Este capítulo apresenta o resultado da realização do trabalho. O capítulo inicia com a descrição do escopo do sistema, seguido da modelagem, apresentação e implementação do sistema desenvolvido.

3.1 ESCOPO DO SISTEMA

O sistema desenvolvido como resultado deste trabalho tem por objetivo apresentar um aplicativo *web* para acompanhamento de egressos. Para isso, foram desenvolvidos os cadastros de cidades, estados, empresas, tecnologias, áreas, profissões, funções, históricos de alunos (ano de ingresso e formação). Além disso, permite o controle de atividades profissionais realizadas por egressos em empresas em que atua ou atuou e, também, que seja realizada a composição de questionários com suas respectivas perguntas para ser respondidas por egressos.

O cadastro de alunos é realizado por meio de um *web service* que permite a integração e a comunicação entre os dados da aplicação. A aplicação *web* permite que o egresso possa responder as perguntas e realizar comentários sobre a Universidade e/ou seu curso de formação. Os comentários serão validados pelo administrador e, somente após a validação, serão disponibilizados na página.

O usuário do sistema poderá realizar uma pesquisa para buscar uma informação específica. Isso ocorrerá por meio de filtros que poderá ser por Câmpus, curso, ano de ingresso e de conclusão.

3.2 MODELAGEM DO SISTEMA

O Quadro 2 contém a relação de requisitos funcionais identificados para o sistema. Nesse quadro, a sigla RF significa Requisito Funcional.

Identificação	Nome	Descrição
RF 1	Manter usuários	Cadastro dos usuários do sistema, contendo os dados necessários para identificação e *login*. Esse cadastro será realizado pelo egresso. Os dados de Câmpus, curso e nome são pré-cadastrados e será necessário indicar os campos de *login* e senha para acesso ao sistema.
RF 3	Manter Câmpus	Cadastro de Câmpus de cada Universidade. Esses dados são providos por um *webservice*.
RF 4	Manter departamentos	Cadastro de departamentos de cada Câmpus. Esses dados são providos por um *webservice*.

RF 5	Manter cursos	Cadastro de cursos de cada departamento. Esses dados são providos por um *webservice*.
RF 6	Manter alunos	Cadastro de alunos de cada curso. Esses dados são providos por um *webservice*.
RF 7	Manter histórico de alunos	Cadastro do histórico do aluno. Esse histórico se refere à trajetória acadêmica, de formação (cursos de aperfeiçoamento) e profissional.
RF 8	Manter empresa	Cadastro de empresas.
RF 9	Manter função	Cadastro de função que cada colaborador exerce na empresa que está vinculado, como, por exemplo, programador sênior, programador júnior. As funções serão utilizadas para seleção do egresso ao manter o seu histórico profissional.
RF 10	Manter profissão	Cadastro de profissão que o colaborador possui na empresa à qual está vinculado, como, por exemplo, programador, analista, entre outras. Ar profissões serão utilizadas para seleção do egresso ao manter o seu histórico profissional.
RF 11	Manter áreas	Cadastro de áreas à qual o colaborador está vinculado, como, por exemplo, *Web*, *Mobile*, *Desktop*, entre outras. As áreas serão utilizadas para seleção do egresso ao manter o seu histórico profissional.
RF 12	Manter tecnologias	Cadastro de tecnologias com as quais o colaborador trabalha. As tecnologias serão utilizadas para seleção do egresso ao manter o seu histórico profissional.
RF 13	Manter atividade profissional	Cadastro da atividade profissional do colaborador, incluindo, empresa, área, profissão, tecnologia, função e período que trabalha/trabalhou na empresa. As atividades serão utilizadas para seleção do egresso ao manter o seu histórico profissional.
RF 14	Realizar comentários	O egresso poderá realizar comentários que serão validados pelo administrador.
RF 15	Realizar consultas	Será possível realizar uma pesquisa por filtro de Câmpus, curso, ano de ingresso e de formatura.
RF 16	Manter Cidades	Cadastro de cidades à qual pertence a empresa. As empresas serão utilizadas para seleção do egresso ao manter o seu histórico profissional.
RF 17	Manter Estados	Cadastro de estados ao qual pertence a cidade.
RF 18	Manter Questionários	Refere-se à criação de questionários disponibilizados pelo Câmpus para obter informações de egressos.
RF 19	Manter Perguntas	Refere-se às perguntas que compõem um questionário.
RF 20	Responder perguntas	Será possível o egresso responder às perguntas de determinado questionário.

Quadro 2 – Requisitos Funcionais

No Quadro 3 estão descritos os requisitos não-funcionais do sistema, ou seja, apresenta como serão realizadas as interações no sistema. Definindo, assim, regras de negócio, formas de acesso, requisitos de qualidades e desempenho. Nesse quadro, a sigla RNF significa Requisito Não-Funcional.

Identificação	Nome	Descrição
RNF01	Acesso ao sistema	O acesso ao sistema será realizado por meio de *login* e senha.
RNF02	Validação de campos	O campo *e-mail* deverá ser validado com restrições de formato. O campo confirmar *e-mail* deverá ser preenchido com o mesmo valor do campo *e-mail*. O campo senha deverá ser validado com restrições de quantidade mínima e máxima de caracteres. O campo confirmar senha deverá ser preenchido com o mesmo valor do campo senha.
RNF03	Campos de preenchimento obrigatórios	Os campos que são de preenchimento obrigatório serão validados por meio de uma função do sistema.

RNF04	Campos com máscaras de entrada	Os campos que possuem caracteres especiais e, que não serão armazenados no banco de dados, como, por exemplo, “()”, “.”, “-”, para os campos RG, CPF, telefone, entre outros, serão validados por meio de máscaras de entrada.
RNF05	*Browser*	O sistema deve ser totalmente compatível com o Google Chrome, versão mínima 58.
RNF06	Sistema operacional	Deve ser executado em todos os ambientes operacionais suportados pelos *browsers* especificado (RNF05).
RNF07	Segurança dos dados	Informações como senha do usuário serão armazenadas criptografadas com algoritmo SHA-2.
RNF08	Linguagem	Todas as informações poderão ser apresentadas no idioma espanhol, inglês e português.
RNF09	Modo *off-line*	O sistema não realizará operações de maneira *off-line*.
RNF10	Busca de registros	A busca de informações dispõe da funcionalidade de filtragem dos dados, de acordo com as informações digitadas pelo usuário.

Quadro 3 – Requisitos não-funcionais

O diagrama de casos de uso apresentado na Figura 2 contém as funcionalidades essenciais do sistema realizadas por seus atores, representados pelo administrador e egresso. O administrador é responsável pelos cadastros gerais, validação dos comentários realizados pelos egressos e realizar consultas sobre questionários e comentários. O egresso é responsável por realizar comentários referentes à instituição e aos cursos e também por responder os questionários.

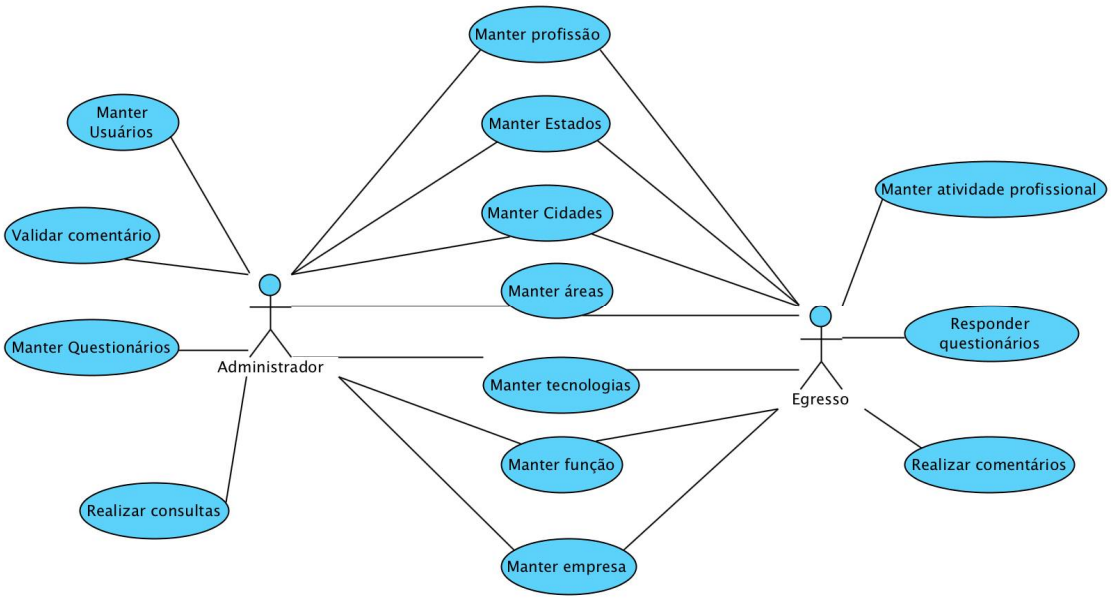

Figura 2 – Diagrama de casos de uso

Os Quadros 4 a 7 apresentam a descrição do caso de uso cadastrar. Essa descrição se refere à operação de inclusão de todos os casos de usos identificados como “manter”.

Caso de uso:
Incluir (refere-se à operação de inclusão nos casos de uso "manter").
Descrição:
Inclusão de informação no sistema.
Evento Iniciador:
Ator solicita inclusão de um registro no sistema.
Atores:
Administrador ou egresso de acordo com suas funções definidas no caso de uso.
Pré-condição:
Acessar a tela de cadastro pelo menu do sistema ou outro *link*.
Sequência de Eventos:
1. Ator acessa a tela para inserir o registro.
2. Ator preenche os campos.
3. Ator clica no botão "Salvar".
4. O sistema insere as informações no banco de dados e retorna mensagem informando que os dados foram inseridos.
5. O sistema é direcionado à tela de pesquisa das informações armazenadas**.**

Pós-Condição:
Informação registrada e disponível para pesquisas ou emissão de relatórios.

Nome do fluxo alternativo (extensão)	**Descrição**
3.1. Campos obrigatórios não preenchidos.	3.1.1. Ator não preenche campos obrigatórios e clica em "Salvar". 3.1.2. Sistema realiza validações e exibe mensagem sobre campos obrigatórios não preenchidos. 3.1.3. Campos obrigatórios não preenchidos são destacados 3.1.4. Retorna ao passo 2 do fluxo principal.
3.2. Campos preenchidos com formato inválido	3.2.1. Ator preenche campos de forma incorreta e clica em "Salvar". 3.2.2. Sistema realiza validações e exibe mensagem sobre campos preenchidos incorretamente. 3.2.3. Campos preenchidos incorretamente são destacados. 3.2.4. Retorna ao passo 2 do fluxo principal.

Quadro 4 – Operação "incluir" dos casos de uso de cadastro

Caso de uso:
Alterar (refere-se à operação de alteração nos casos de uso "manter").
Descrição:
Registro inserido no sistema.
Evento Iniciador:
Ator solicita alteração de um registro no sistema.
Atores:
Administrador ou egresso.
Pré-condição:
Acessar a tela de cadastro pelo menu do sistema ou outro *link*.
Sequência de Eventos:
1. Ator acessa a página inicial de alteração do registro.
2. Ator modifica as informações necessárias.
3. Ator clica no botão "Salvar".
4. O sistema realização a atualização das informações no banco de dados e retorna mensagem de sucesso como *feedback*.
5. O sistema é direcionado à tela de pesquisa das informações armazenadas**.**

Pós-Condição:
Informação registrada e disponível para futuras auditorias ou emissão de relatórios.

Nome do fluxo alternativo (extensão)	**Descrição**
3.1. Campos obrigatórios não preenchidos.	3.1.1. Ator não preenche campos obrigatórios e clica em "Salvar". 3.1.2. Sistema realiza validações e exibe mensagem sobre campos obrigatórios não preenchidos. 3.1.3. Campos obrigatórios não preenchidos são destacados 3.1.4. Retorna ao passo 2 do fluxo principal.

3.2. Campos preenchidos com formato inválido	3.2.1. Ator preenche campos de forma incorreta e clica “Salvar”. 3.2.2. Sistema realiza validações e exibe mensagem sobre campos preenchidos de maneira incorreta. 3.2.3. Campos preenchidos de maneira incorreta são destacados. 3.2.4. Retorna ao passo 2 do fluxo principal.

Quadro 5 – Operação “alterar” dos casos de uso de cadastro

Caso de uso:
Excluir (refere-se à operação de exclusão nos casos de uso “manter”).
Descrição:
Remoção de registro no sistema.
Evento Iniciador:
Ator solicita exclusão de um registro no sistema.
Atores:
Administrador ou egresso de acordo com suas funções definidas no caso de uso.
Pré-condição:
Acessar a tela de pesquisa pelo menu do sistema ou outro link.
Sequência de Eventos:
1. Ator acessa a página inicial de listagem de registros.
2. Ator seleciona o registro que deseja excluir.
3. Ator clica no botão “Excluir”.
4. O sistema remove o registro do banco de dados e da listagem de registros e informa ao usuário que o *status* do procedimento.

Pós-Condição:
Informação excluída do banco de dados e não mais disponíveis em qualquer relatório ou listagem.

Nome do fluxo alternativo (extensão)	**Descrição**
3.1. Registro com relacionamento em outros cadastros.	3.1.1 Ator clica em excluir um registro que possui vínculos no sistema. 3.1.2. O sistema apresenta mensagem que o registro selecionado não pode ser excluído devido ao relacionamento. 3.1.3. Retorno ao passo 1 do fluxo principal.

Quadro 6 – Operação “excluir” dos casos de uso de cadastro

Caso de uso:
Consultar (refere-se à operação de consulta nos casos de uso “manter”).
Descrição:
Consulta de informações armazenadas no banco de dados.
Evento Iniciador:
Ator solicita consultar de informações do sistema.
Atores:
Administrador ou egresso de acordo com suas funções definidas no caso de uso.
Pré-condição:
Acessar a tela de pesquisa pelo menu do sistema ou outro link.
Sequência de Eventos:
1. Ator acessa a página de listagem de registros.
2. Ator preenche os campos referentes aos filtros aplicados à consulta**.**
3. Ator clica no botão “Pesquisar”.
4. Sistema aplica os filtros informados pelo ator.
5. Sistema exibe os registros que atendem as condições especificadas nos campos de filtragem dos dados.

Pós-Condição:
Dados da consulta apresentados ao usuário.

Nome do fluxo alternativo (extensão)	**Descrição**
3.1. Nenhum campo de filtro preenchido.	3.1.1. O sistema deve apresentar registros sem filtros.

Quadro 7 – Operação “consultar” dos casos de uso de cadastro

No Quadro 8 está a especificação do caso de uso denominado validar comentários. Os egressos podem realizar comentários sobre a Universidade ou curso e, posteriormente, esses comentários são validados pelo administrador.

Caso de uso: Validar comentários. **Descrição:** Validar comentários realizados por egressos referentes ao curso e instituição. **Evento Iniciador:** Ator solicita comentários pendentes para aprovação. **Atores:** Administrador. **Pré-condição:** Acessar a tela de validação de comentários de egressos. **Sequência de Eventos:** 1. Ator acessa a página de validação de comentários. 2. Ator seleciona o comentário que deseja validar**.** 3. Ator aceita comentário do egresso. 4. Comentário é vinculado ao curso e instituição. **Pós-Condição:** Comentário fica vinculado ao curso e instituição e disponível para qualquer usuário visualizar.	
Nome do fluxo alternativo (extensão)	**Descrição**
3.1. Ator recusa comentário.	3.1.1. Ator recusa o comentário realizado pelo egresso. 3.1.2. Comentário não fica mais disponível na tela para aceitação e não é vinculado ao curso e instituição. 3.1.2. Encerra caso de uso.

Quadro 8 – Caso de uso "Validar comentários"

No Quadro 9 está a especificação do caso de uso denominado realizar comentários. Esses comentários se referem à instituição ou curso que o egresso frequentou.

Caso de uso: Realizar comentários. **Descrição:** Efetuar comentário referente ao curso e/ou instituição. **Evento Iniciador:** Ator solicita formulário para preencher comentário. **Atores:** Egresso. **Pré-condição:** Acessar a tela de realização de comentários. **Sequência de Eventos:** 1. Ator acessa a página de realização de comentários. 2. Ator seleciona a instituição ou o curso sobre o qual deseja fazer comentário**.** 3. Ator descreve o comentário desejado. 4. Ator clica em "Enviar comentário". 5. Comentário é enviado para aprovação. **Pós-Condição:** Comentário fica pendente para aprovação pelo administrador.	
Nome do fluxo alternativo (extensão)	**Descrição**
4.1. Campos obrigatórios não informados.	4.1.1. Ator não preenche campos obrigatórios. 4.1.2. Sistema realiza validação e exibe mensagem de campos obrigatórios não informados. 4.1.3. Campos obrigatórios não preenchidos são destacados.

	4.1.4. Retorna ao passo 1 do fluxo principal.

Quadro 9 – Caso de uso "Realizar comentários"

O Quadro 10 apresenta o caso de uso denominado responder questionário. Esses questionários são respondidos pelos egressos.

Caso de uso: Responder questionários. **Descrição:** Responder questionários disponibilizados pelo administrador. **Evento Iniciador:** Ator solicita questionário para responder. **Atores:** Egresso. **Pré-condição:** Acessar a tela de questionários. **Sequência de Eventos:** 1. Ator acessa a página de questionários disponíveis. 2. Ator seleciona o questionário que deseja responder. 3. Ator clica em "Iniciar". 4. Sistema exibe perguntas ao ator. 5. Ator responde as perguntas que aparece na tela. 6. Sistema armazena as respostas informadas. **Pós-Condição:** Respostas do questionário ficam disponíveis para possíveis consultas pelos administradores.	
Nome do fluxo alternativo (extensão)	**Descrição**
5.1. Campos obrigatórios não informados.	5.1.1. Ator não preenche campos obrigatórios. 5.1.2. Sistema realiza validação e exibe mensagem de campos obrigatórios não informados. 5.1.3. Campos obrigatórios não preenchidos são destacados. 5.1.4. Retorna ao passo 4 do fluxo principal.

Quadro 10 – Caso de uso "Responder questionário"

O Quadro 11 apresenta a especificação do caso de uso denominado disponibilizar questionário. Esses questionários são disponibilizados para os egressos.

Caso de uso: Disponibilizar questionário. **Descrição:** Disponibilizar questionário para ser respondido por egressos. **Evento Iniciador:** Ator solicita gerenciamento de questionários. **Atores:** Administrador. **Pré-condição:** Acessar a tela de gerenciamento de questionários. **Sequência de Eventos:** 1. Ator acessa a página de disponibilização de questionários. 2. Ator seleciona o questionário que deseja disponibilizar. 3. Ator informa data final que o questionário deve ficar disponível. 4. Ator seleciona qual instituição e curso poderá responder. 5. Ator clica em "Disponibilizar". 6. Sistema torna possível responder esse questionário para os devidos egressos. **Pós-Condição:** Questionário pendente para resposta pelos egressos que atendem as condições do cadastro.	
Nome do fluxo alternativo	**Descrição**

(extensão)	
5.1. Campos obrigatórios não informados.	5.1.1. Ator não preenche campos obrigatórios. 5.1.2. Sistema realiza validação e exibe mensagem de campos obrigatórios não informados. 5.1.3. Campos obrigatórios não preenchidos são destacados. 5.1.4. Retorna ao passo 2 do fluxo principal.
5.2. Campos preenchidos com formato inválido	5.2.1. Ator preenche campos de forma incorreta e clica “Salvar”. 5.2.2. Sistema realiza validações e exibe mensagem sobre campos preenchidos de maneira incorreta. 5.2.3. Campos preenchidos de maneira incorreta são destacados. 5.2.4. Retorna ao passo 2 do fluxo principal.

Quadro 11 – Caso de uso “Disponibilizar questionário”

A Figura 3 representa o Diagrama de Entidades e Relacionamentos (DER) do banco de dados do sistema. A tabela atividade_profissional armazena um histórico profissional do egresso, permitindo detalhamento por data de início e fim e possui relacionamento com as tabelas empresa, funcao, area e profissao.

A tabela historico, tem papel fundamental para registrar as passagens do egresso pela instituição. Os campos id_campus_curso e id_aluno, não possuem chave estrangeira devido à sua informação estar presente em outro banco de dados, fica a responsabilidade da aplicação gravar as informações consistentes.

A tabela questionário, permite o armazenamento de questionários que serão disponibilizados para os egressos responderem e a tabela resposta, armazena as alternativas selecionadas pelo egresso durante o processo de responder o questionário. A tabela comentário armazena as informações dos comentários realizados pelos egressos, possuindo relacionamento com a tabela histórico para identificar qual egresso realizou o comentário.

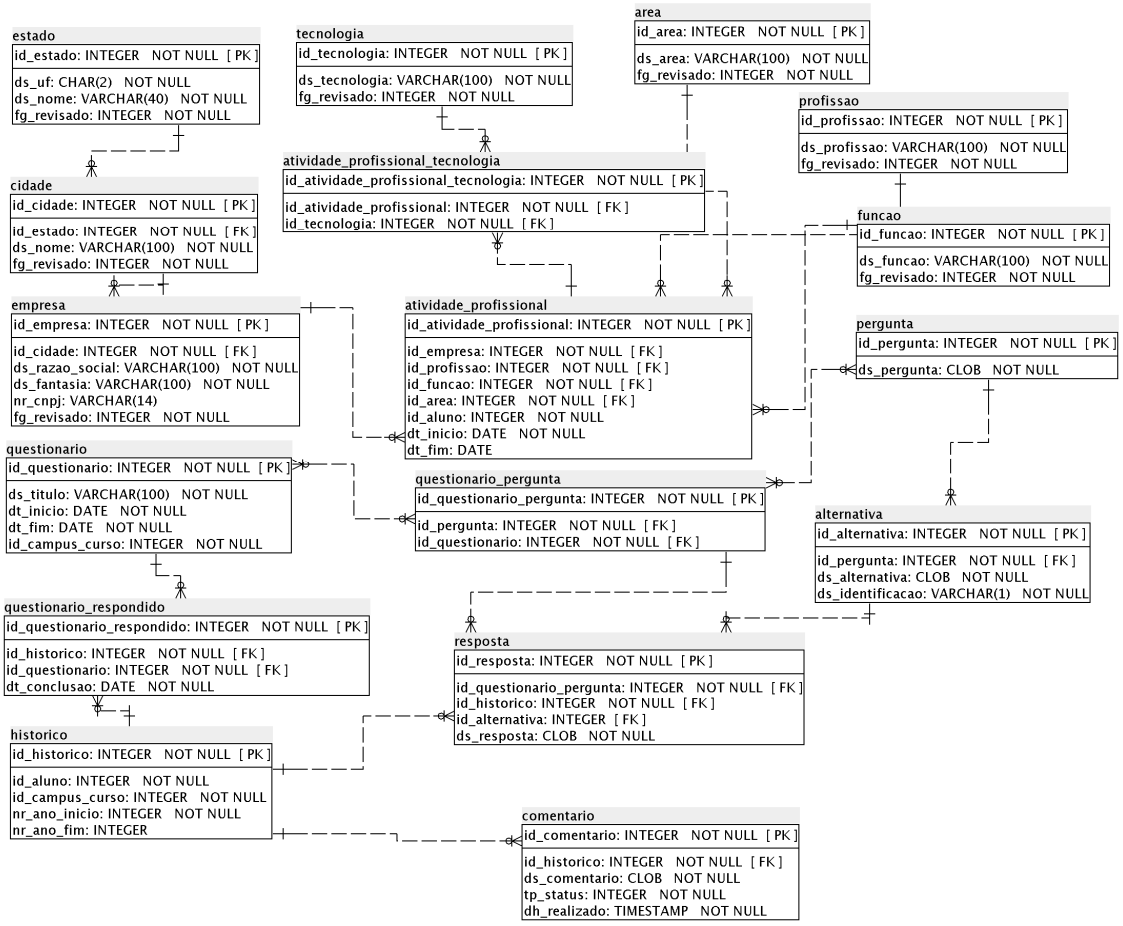

Figura 3 – Diagrama entidades e relacionamentos do sistema

A Figura 4 representa o DER do banco de dados do *webservice*. A tabela campus permite a gravação da descrição do Câmpus. A tabela curso permite gravação do nome do curso. Desta forma, a tabela campus_curso permite relacionar as informações dessas outras duas tabelas e identificar os cursos de Câmpus. A tabela aluno é importante para gravação de dados pessoais referente ao egresso.

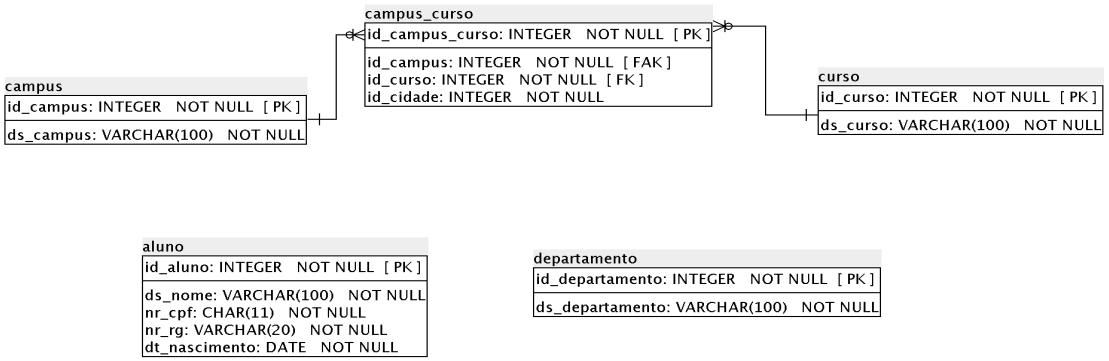

Figura 4 – Diagrama de entidades e relacionamentos do webservice

3.3 APRESENTAÇÃO DO SISTEMA

O sistema possui uma tela inicial com as opções para autenticar-se ou criar uma conta para acessar o sistema. A Figura 5 exibe a tela inicial do sistema com essas opções.

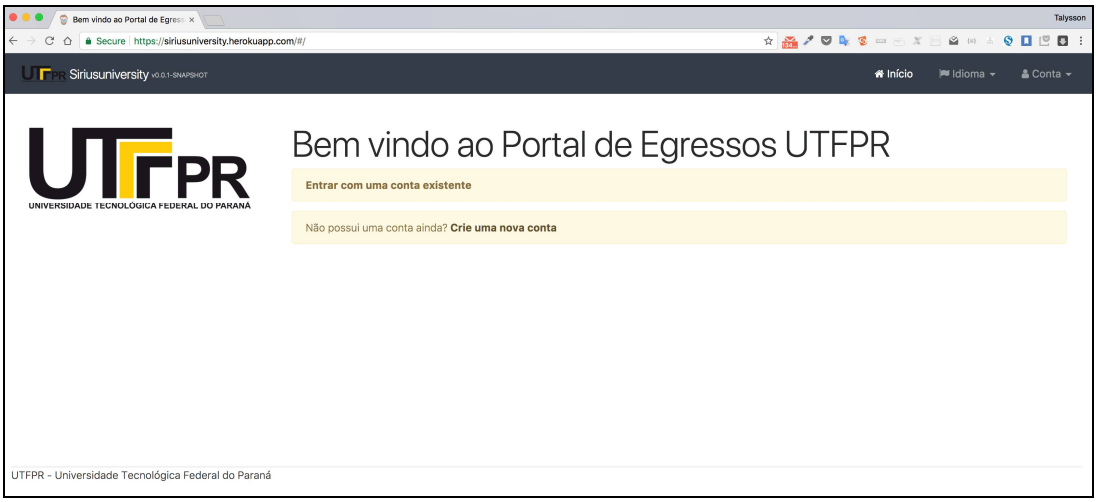

Figura 5 – Tela inicial do sistema

Após autenticar-se no sistema é possível utilizar os recursos que serão apresentados a seguir. A Figura 6 exibe a tela de autenticação do sistema, na qual são solicitados os dados de autenticação (*login* e senha) previamente cadastrados.

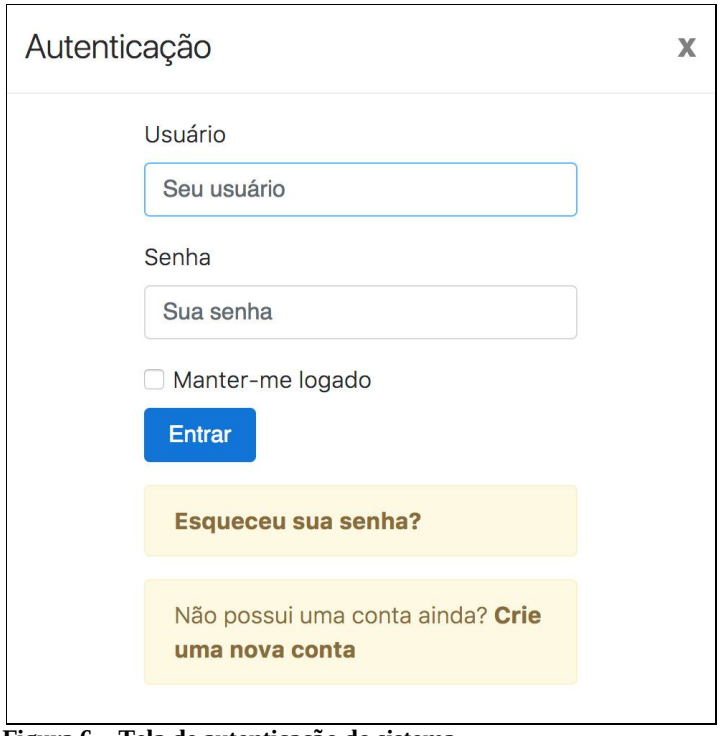

Figura 6 – Tela de autenticação do sistema

A Figura 7 ilustra a tela inicial do sistema após realizar o *login* por um usuário que possui características de administrador ou egresso. Os menus de acesso às funcionalidades do sistema podem ser visualizados no topo da página.

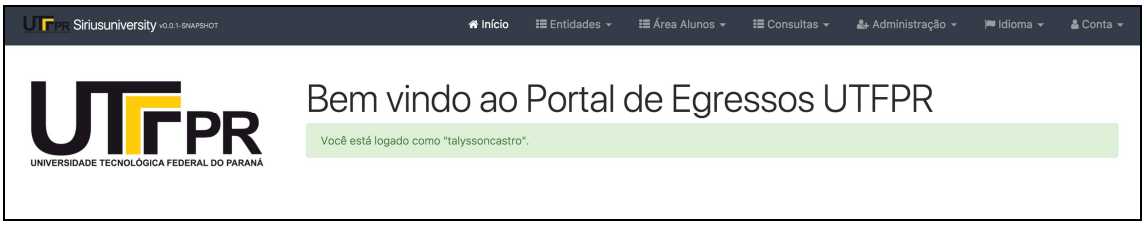

Figura 7 – Tela inicial do sistema

O menu Entidades é disponibilizado para usuários com perfil de administrador ou egresso. Essa opção permite realizar diversos cadastros para alimentar informações sobre os egressos, criar perguntas com suas respectivas alternativas, formulários para disponibilizar para os egressos responderem e também validar comentários realizados pelos egressos sobre os cursos em cada Câmpus.

O menu Área Alunos é disponibilizado para usuários com perfil de egresso. Nessa opção é possível responder os questionários vigentes e também realizar comentários sobre Câmpus e cursos.

O menu de consultas é possível ser acessado por usuários com perfil administrador. Nesse menu é permitido realizar consultas de comentários de Câmpus e curso, de questionários respondidos e não respondidos e de históricos de alunos.

A opção de menu Administração, disponível para usuários com perfil de administrador, permite o cadastro de usuários do sistema, assim como demais gerenciamentos técnicos sobre o sistema.

Na opção idioma, é possível alterar a linguagem que deseja que o sistema seja apresentado. Atualmente é suportado Português, Inglês e Espanhol.

Por meio do menu Conta podem ser realizadas configurações referente à conta do usuário do sistema, além da opção de sair do sistema.

A Figura 8 apresenta a listagem com exemplos de tecnologias cadastradas no sistema.

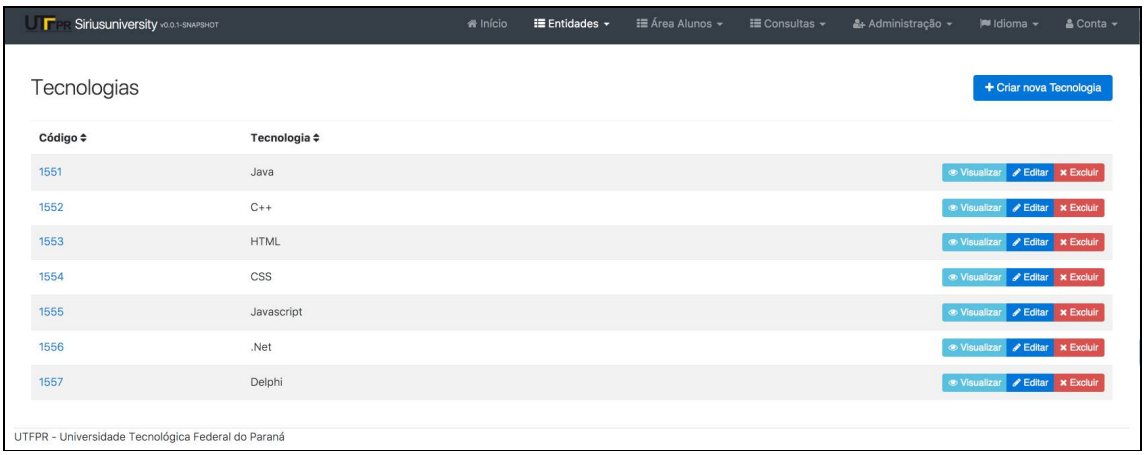

Figura 8 – Listagem de tecnologias

A listagem na Figura 8 apresenta o código e o nome de cada tecnologia, assim como os botões de ações para visualizar, editar ou excluir um registro. A ação de visualizar apresenta os detalhes dos dados da tecnologia. A opção de editar permite realizar a edição de um registro selecionado e a opção de excluir permite realizar a exclusão de registro do sistema. Todas as entidades do sistema possuem essas três ações, que tem o mesmo objetivo apresentado na entidade de tecnologia.

Essa listagem segue o mesmo padrão utilizado na listagem de áreas, funções, profissões, cidades e Estados. A Figura 9 apresenta a tela de cadastro de uma nova tecnologia.

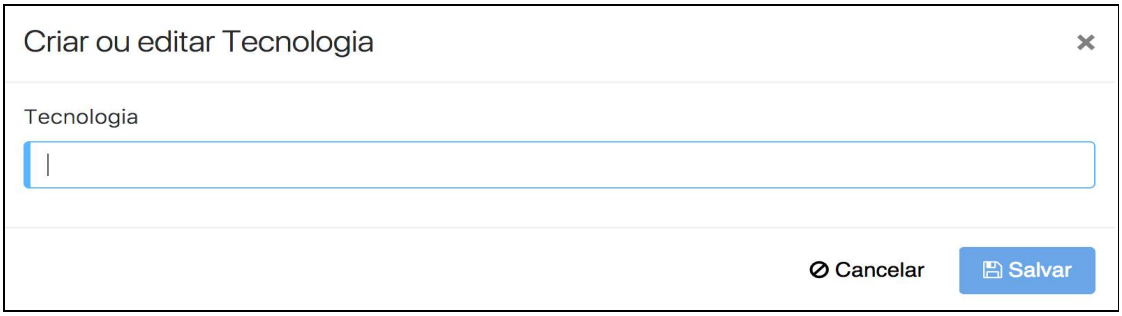

Figura 9 – Tela de cadastro de tecnologia

Para realizar o cadastro de uma nova tecnologia é necessária, apenas, a descrição da tecnologia, pois o código é gerado automaticamente quando o registro é salvo no banco de dados. O mesmo processo é realizado para os cadastros de áreas, funções, profissões, cidades e Estados.

A Figura 13 apresenta a tela de edição de um registro de tecnologia. Ao abrir a tela de edição são carregados os dados armazenados referentes ao registro selecionado, porém é permitido alterar somente a descrição, pois o código é a identificação única do registro e

controlado pelo sistema. O mesmo acontece na edição de registros nas entidades de áreas, funções, profissões, cidades e Estados.

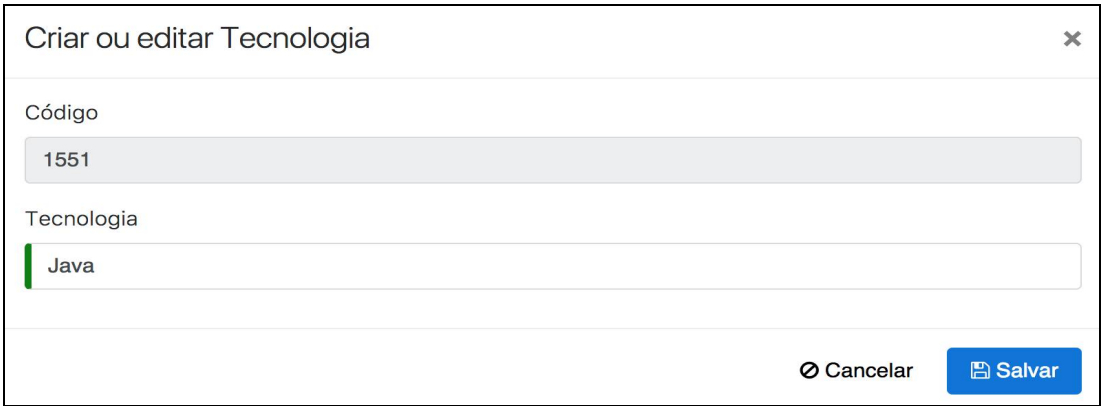

Figura 10 – Tela de edição de uma tecnologia

A Figura 11 apresenta a listagem de empresas exibindo os dados das empresas e as opções para editar ou excluir um registro.

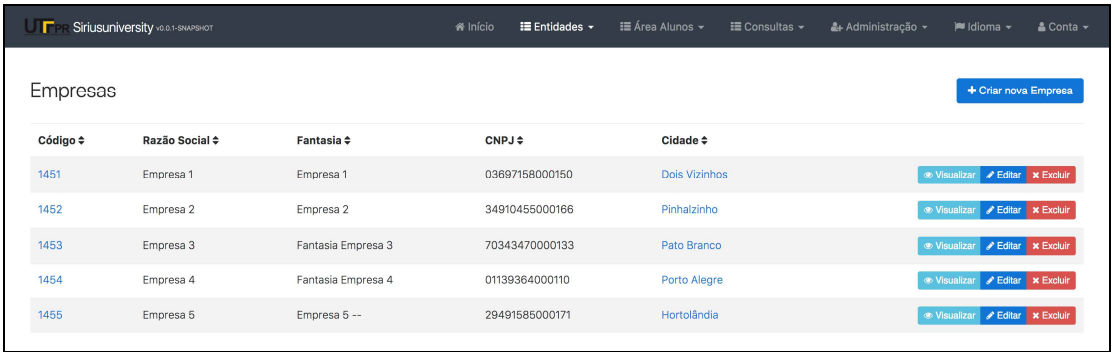

Figura 11 – Tela de listagem de empresas

O sistema permite armazenar as atividades profissionais exercidas por egressos em empresas que atuam ou atuaram. Os dados solicitados são o nome da empresa em que o egresso atua ou atuou, o período de início e fim, a profissão e função exercida e área de trabalho. A Figura 12 apresenta a tela de inserção ou edição de atividade profissional.

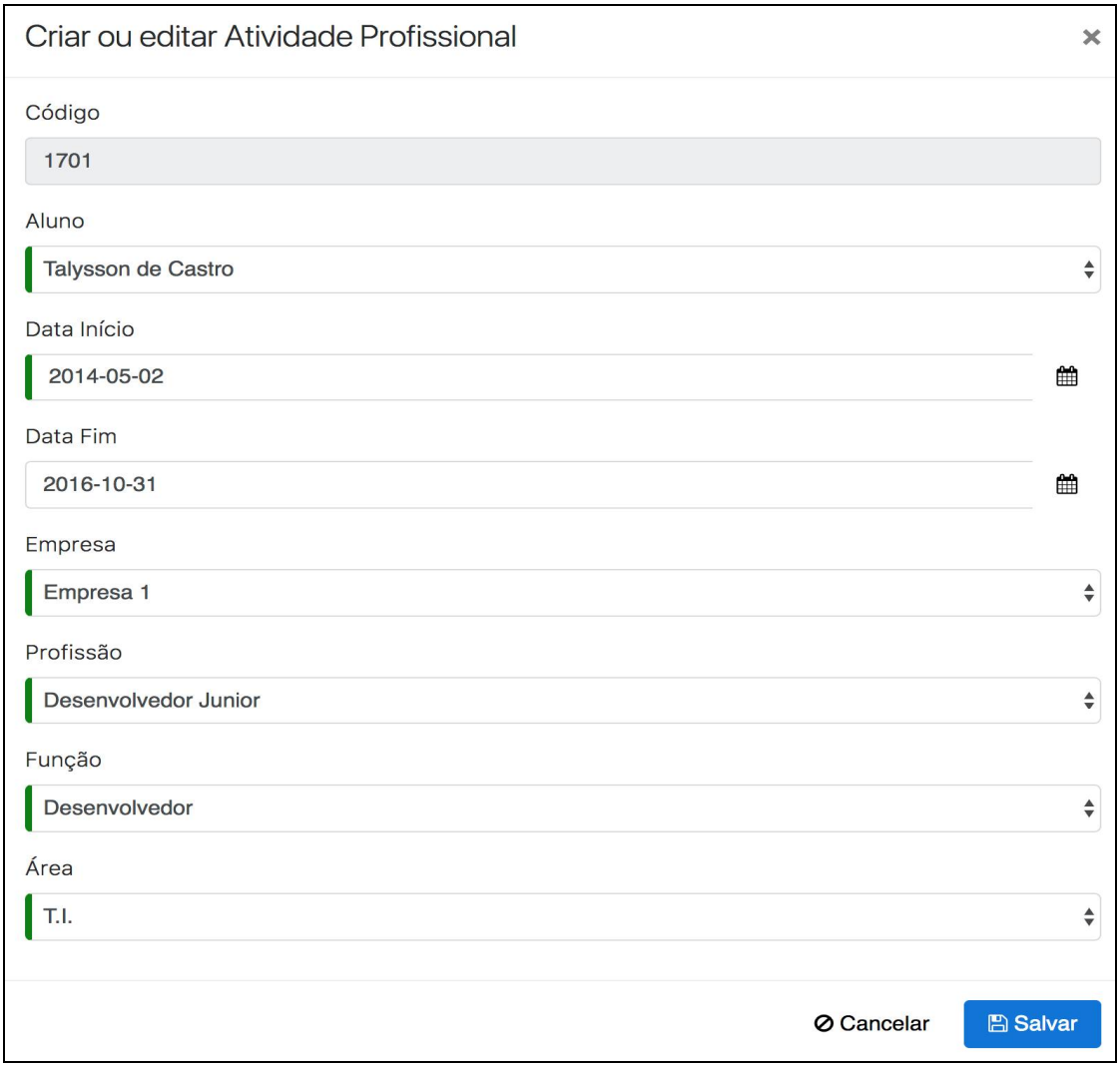

Figura 12 – Tela de inserção e edição de atividade profissional

A Figura 13 apresenta a listagem dos dados das atividades profissionais.

Figura 13 – Tela de listagem das atividades profissionais

Após o cadastro de atividades profissionais exercidas pelo egresso, é possível relacionar as tecnologias que utilizadas por ele na empresa. A Figura 14 apresenta como realizar esse relacionamento.

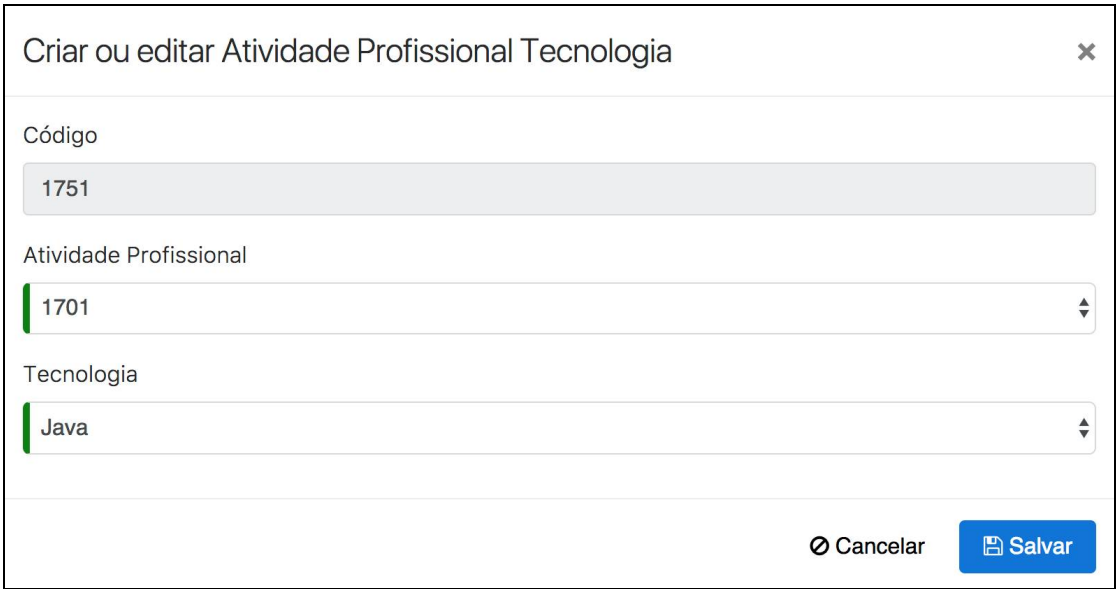

Figura 14 – Tela de relacionamento de atividade profissional e tecnologia

A Figura 15 exibe a listagem das atividades profissionais e das tecnologias relacionadas.

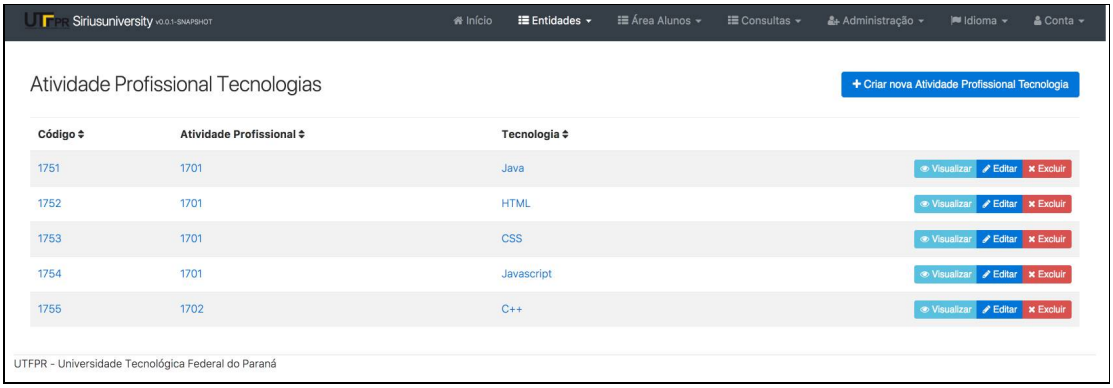

Figura 15 – Tela de listagem de atividade profissional e tecnologia

É possível realizar o cadastro de históricos de egressos, no qual é informado o nome do egresso, o Câmpus e curso de formação, ano de ingresso e de formação. A Figura 16 exibe a tela de inserção ou edição do histórico do egresso.

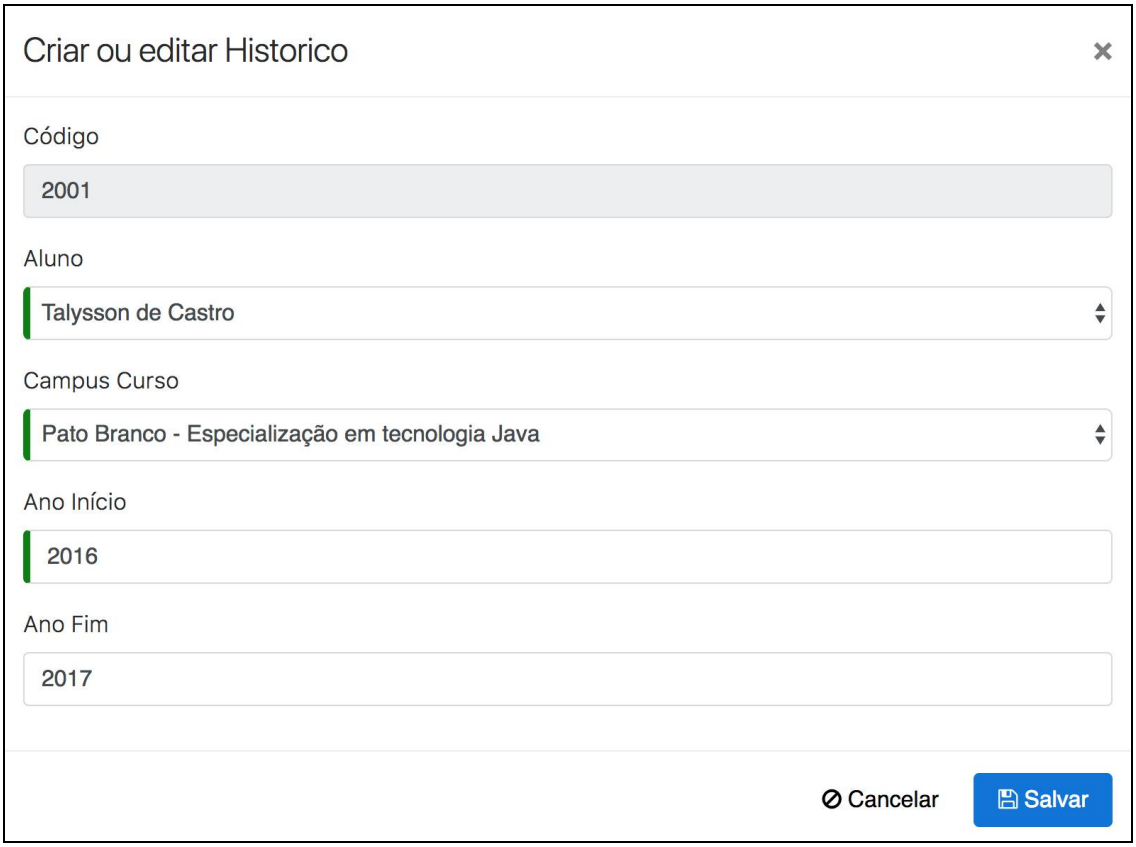

Figura 16 – Tela de inserção ou edição de histórico

A Figura 17 exibe a tela de listagem do histórico dos egressos.

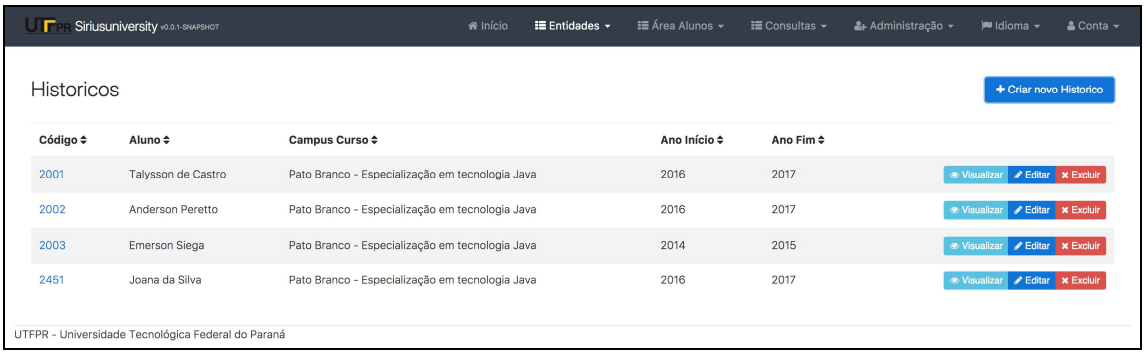

Figura 17 – Tela de listagem de histórico de egressos

Um dos principais objetivos do sistema é permitir a criação de questionários com perguntas e alternativas de respostas. O gerenciamento de perguntas possui apenas a descrição da pergunta e o código que é gerado automaticamente pelo sistema. A Figura 18 exibe a tela de inserção ou edição de perguntas do questionário.

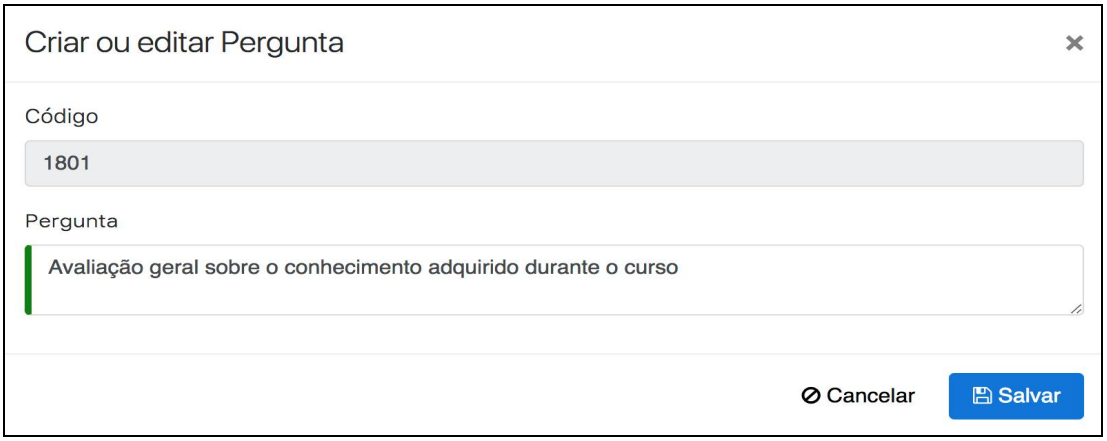

Figura 18 – Tela de inserção ou edição de perguntas do questionário

A Figura 19 exibe a tela de listagem das perguntas registradas no banco de dados.

Figura 19 – Tela de listagem de perguntas do questionário

Pelo gerenciamento de alternativas é possível controlar sua descrição, qual a letra identificadora e à qual pergunta pertence, conforme apresenta a Figura 20. O código é gerado automaticamente pelo sistema.

Figura 20 – Tela de inserção ou edição de alternativas do questionário

A Figura 21 exibe a tela de listagem de alternativas do questionário.

Figura 21 – Tela de listagem de alternativas do questionário

O gerenciamento de questionário permite controlar para qual Câmpus e curso o questionário é direcionado. Permite, também, criar o título, a data inicial e data limite para ser respondido, conforme ilustra a Figura 22.

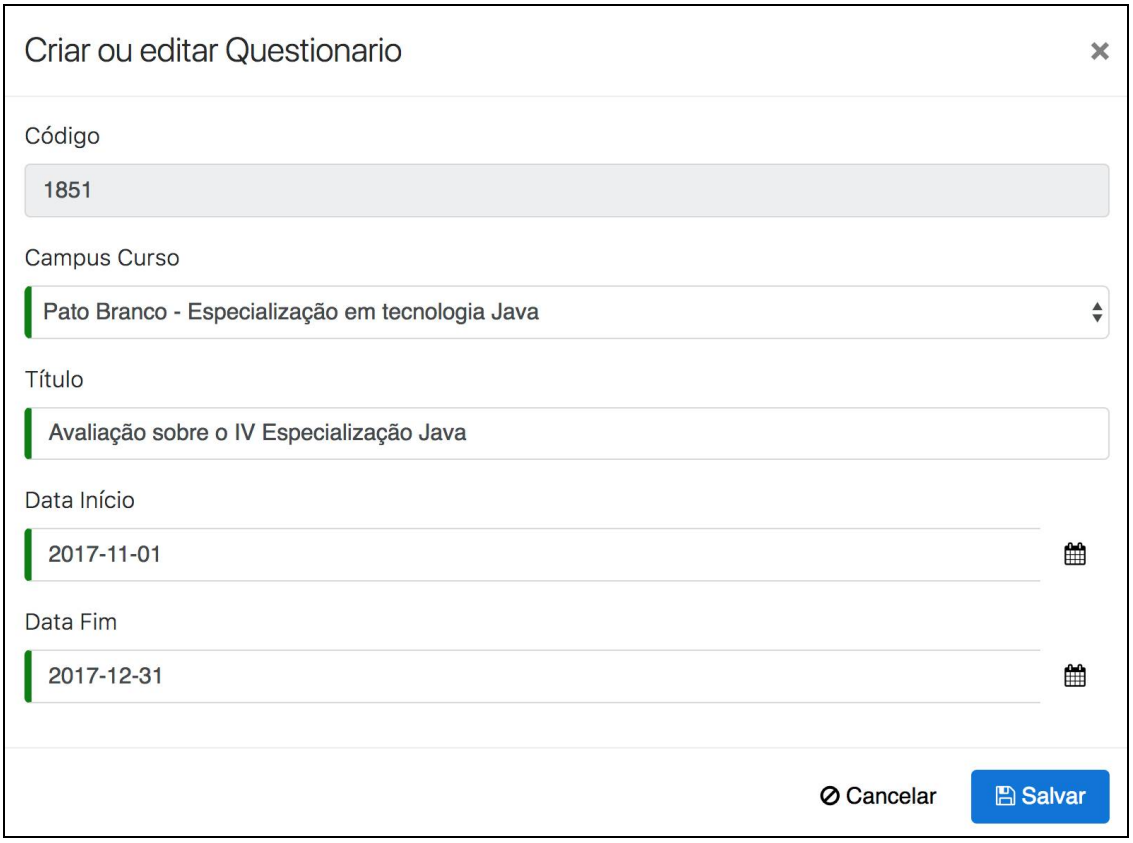

Figura 22 – Tela de inserção ou edição de questionário

A Figura 23 exibe a tela de listagem dos questionários.

Figura 23 – Tela de listagem dos questionários

Após o cadastro de questionários e perguntas com suas respectivas alternativas, é necessário realizar o relacionamento das perguntas disponíveis em cada questionário. A Figura 24 exibe a tela de relacionamentos de perguntas e questionários.

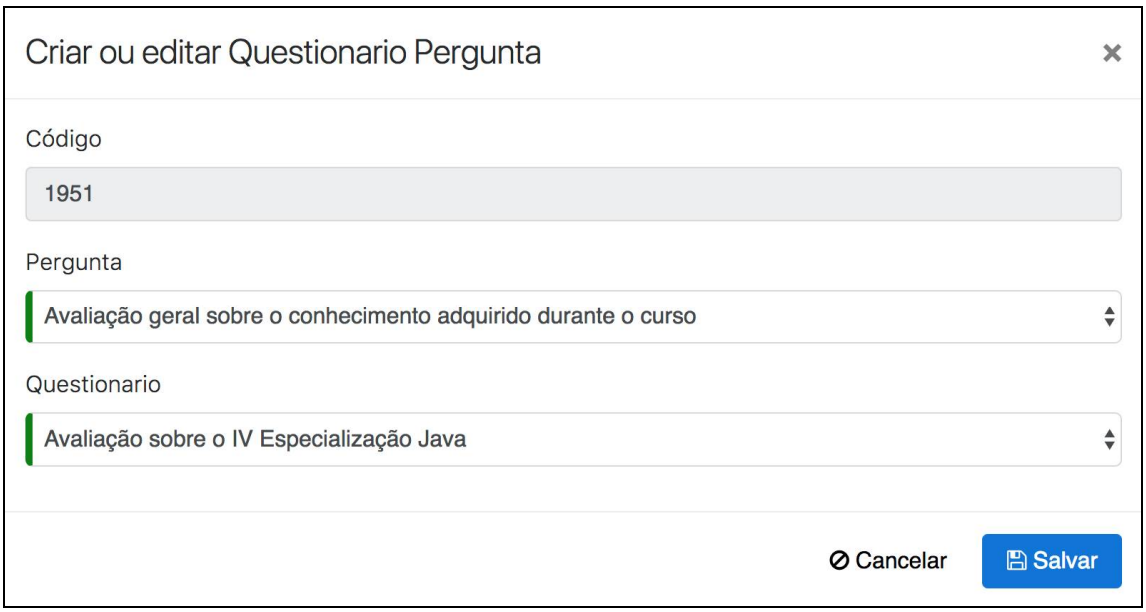

Figura 24 – tela de relacionamento de pergunta e questionário.

A Figura 25 exibe a tela de listagem do relacionamento das perguntas e questionários.

Figura 25 – Tela de listagem de perguntas e questionários

Com essas configurações realizadas, os questionários podem estar disponíveis para ser respondidos pelos egressos que pertencem ao Câmpus e curso correspondente ao informado no cadastro.

Os questionários disponíveis para serem respondidos podem ser acessados por meio do menu Área Alunos, opção Questionários Disponíveis. Na tela inicial são disponibilizados os questionários que ainda não foram respondidos pelo egresso e estão dentro da data de vigência, conforme apresentado na Figura 26.

Figura 26 – Tela de questionários disponíveis para serem respondidos

O botão denominado responder da Figura 26 é disponibilizado para cada questionário. Clicando nesse botão o egresso será direcionado para a tela que possui as perguntas com suas respectivas alternativas. A Figura 27 apresenta a tela com uma pergunta do questionário, na qual é possível selecionar qual a alternativa escolhida e acionar o botão denominado responder para efetivar a resposta, conforme apresentado na Figura 27.

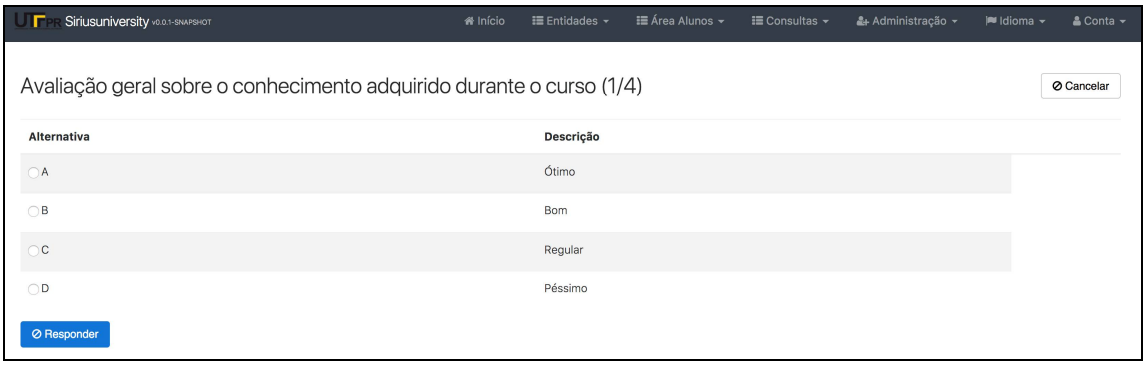

Figura 27 – Tela de pergunta e alternativas de resposta

Após clicar no botão responder (Figura 27), a próxima pergunta do questionário é exibida na tela e esse procedimento é realizado até que o questionário seja finalizado. Após responder todas as perguntas, o sistema exibe uma tela informando que o questionário foi finalizado, com um botão para voltar à listagem de questionários disponíveis para responder, conforme mostra a Figura 28. O questionário que foi respondido não será mais listado.

Figura 28 – Tela com mensagem de questionário finalizado

Outro recurso disponibilizado pelo sistema é a realização de comentários que são realizados pelo egresso em relação ao Câmpus e curso frequentado. É possível realizar esses comentários, informando a descrição no campo denominado comentário da Figura 29.

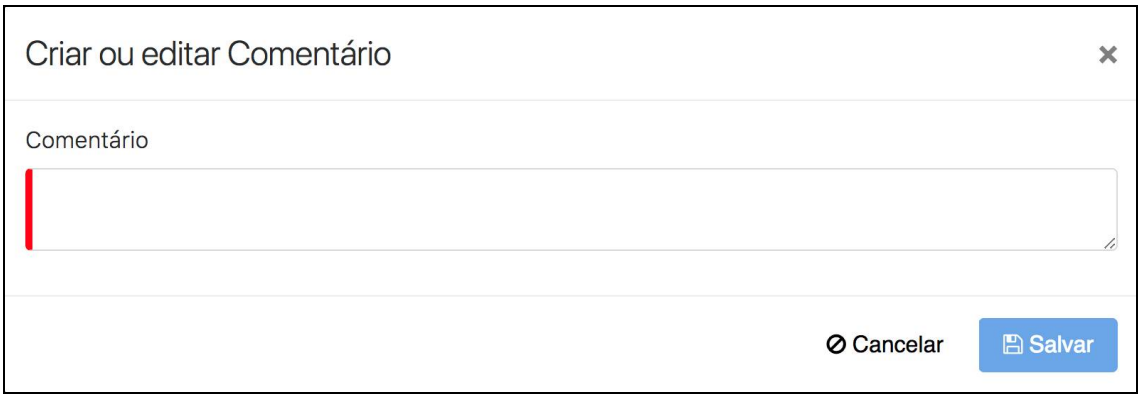

Figura 29 – Tela de inserção de comentários

Esse comentário fica com o *status* de pendente até que o administrador realize a validação desse comentário que pode ser aprovado ou reprovado. Na listagem de comentários realizados pelo egresso, é possível acompanhar a descrição e *status* de cada comentário, também é permitido editar comentário antes que seja aprovado ou recusado, conforme exemplifica a Figura 30.

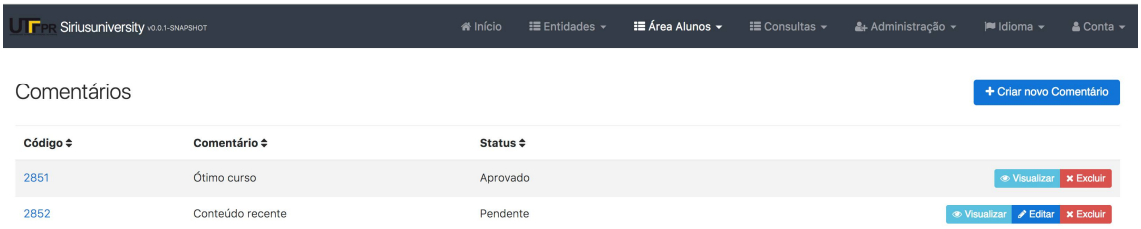

Figura 30 – Tela de listagem de comentários

Para realizar a validação dos comentários é exibida uma listagem de comentários pendentes, com informações como o nome do egresso e o texto informado e as opções de aceitar ou rejeitar o comentário, conforme exemplifica a Figura 31.

Figura 31 – Tela de validação de comentários

O sistema permite realizar consultas referentes aos questionários e históricos dos egressos. A consulta de históricos permite informar o Câmpus e curso, ano de ingresso e de término do curso como filtros para exibir as informações do egresso, conforme apresentado na Figura 32.

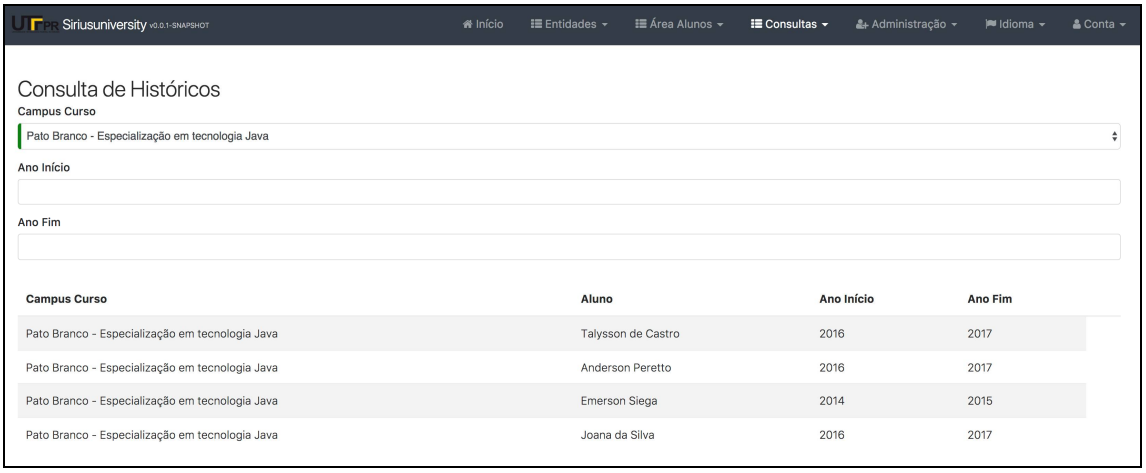

Figura 32 – Tela de consulta de histórico

Também é possível realizar a consulta de comentários realizados por egressos em cada Câmpus e curso. A Figura 33 apresenta a tela que permite filtrar os comentários por Câmpus e curso, exibindo o resultando em uma listagem.

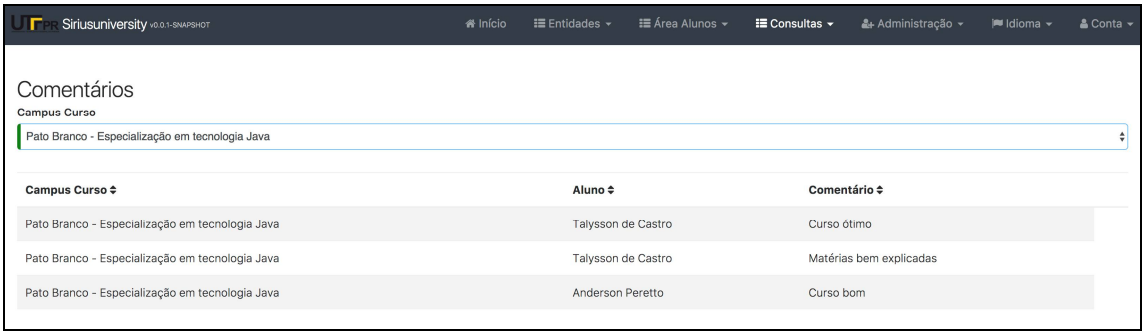

Figura 33 – Tela de consulta de comentários

O sistema também permite realizar a consulta de questionários respondidos, conforme apresentando na Figura 34. Nessa consulta é possível selecionar o Câmpus e curso e também o questionário que deseja visualizar os egressos que responderam.

Figura 34 – Tela de consulta de questionários respondidos

Na listagem de resultados, a ação de visualizar permite exibir as perguntas e a respectiva alternativa selecionada pelo egresso no momento que respondeu o questionário. Ao clicar nesse botão é apresentada a listagem dos detalhes do questionário respondido, conforme apresentado na Figura 35.

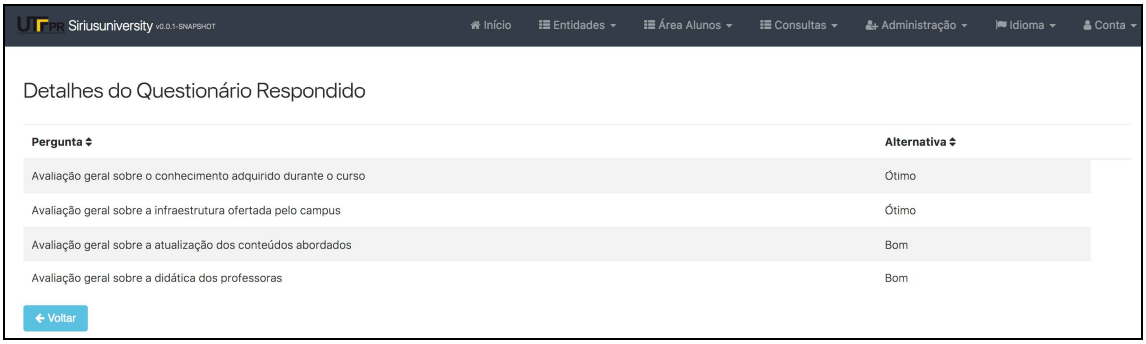

Figura 35 – Tela dos detalhes do questionário respondido

A consulta de questionários não respondidos permite filtrar por Câmpus e curso e também por questionário, para exibir os egressos que não responderam o questionário, conforme apresentando na Figura 36. A consulta também permite selecionar questionários ainda vigentes.

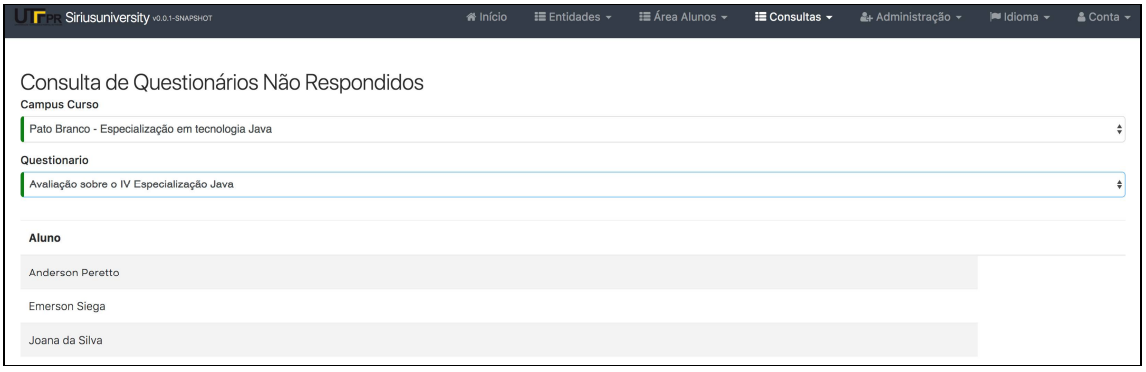

Figura 36 – Tela de consulta de questionário não respondidos

Por meio do menu Administração e da opção gerenciamento de usuário, é possível realizar a manutenção e o cadastro de usuários do sistema. Para realizar o cadastro de um

novo usuário ou alterar um existente é necessário informar os campos de *login*, senha e confirmação da senha, primeiro nome, último nome, *e-mail*, qual o idioma principal e os perfis que o usuário possui, conforme apresentado na Figura 37.

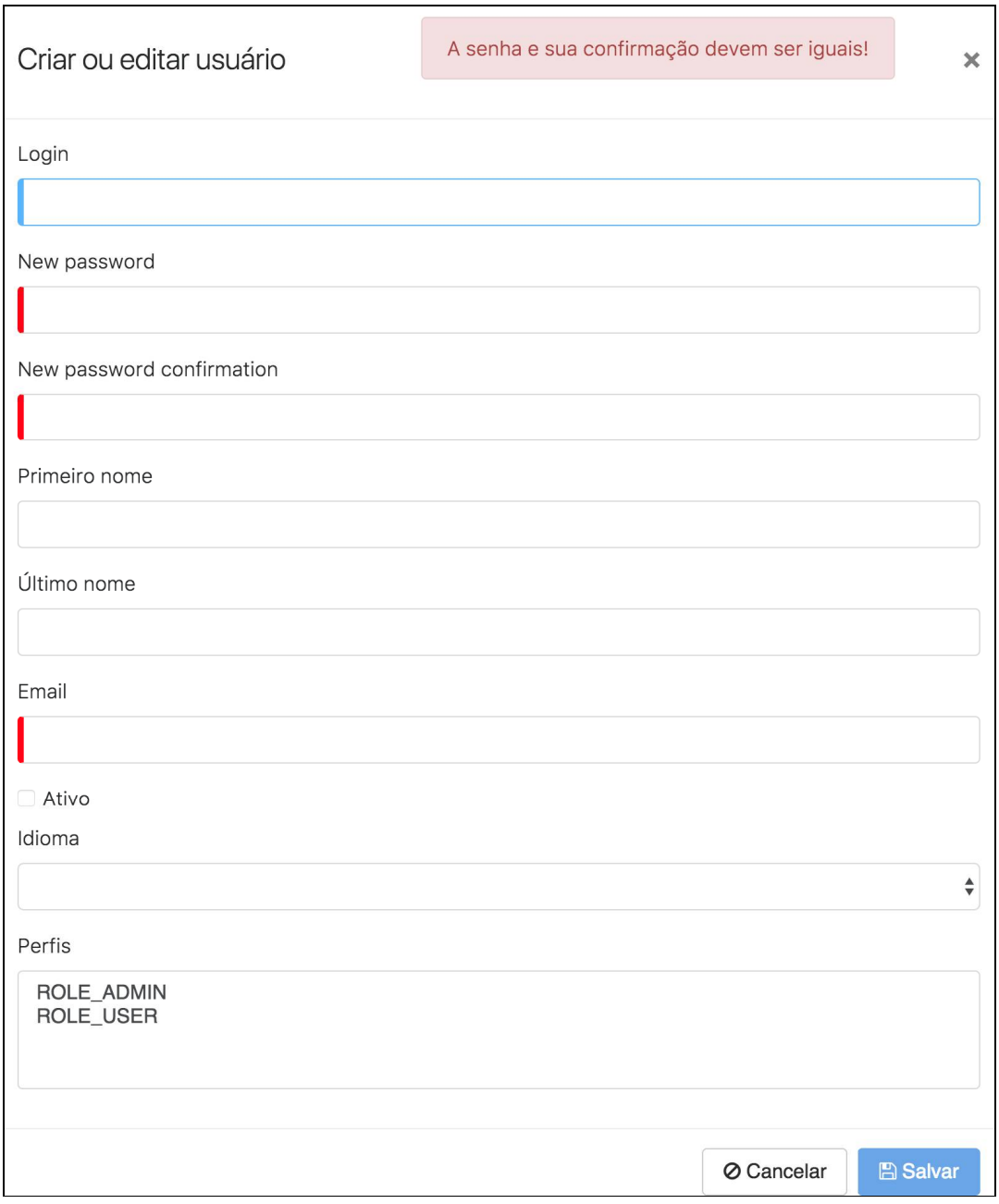

Figura 37 – Tela de inserção e edição de usuário

As principais informações do usuário são exibidas na listagem apresentada na Figura 38.

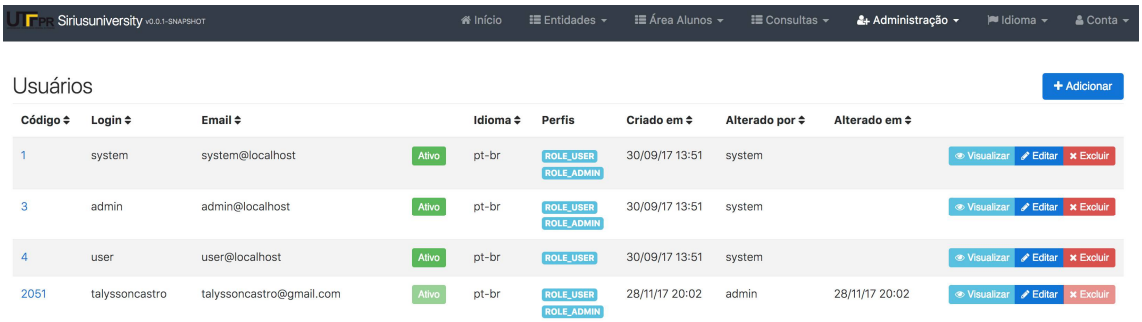

Figura 38 - Tela de listagem de usuários

3.4 IMPLEMENTAÇÃO DO SISTEMA

Para o desenvolvimento *back-end* da aplicação foi utilizado o *framework* Spring. Um dos componentes do Spring é o Spring Boot, que também foi utilizado para o desenvolvimento da aplicação. Para execução da aplicação utilizando Spring Boot é necessário criar uma classe com método *main* para identificar o ponto de início do sistema. Foi criada a classe *SiriusuniversityApp* e configurada conforme mostra Listagem 2.

```
@ComponentScan
@EnableAutoConfiguration(exclude = {MetricFilterAutoConfiguration.class,
MetricRepositoryAutoConfiguration.class})
@EnableConfigurationProperties({LiquibaseProperties.class, ApplicationProperties.class})
public class SiriusuniversityApp {

    /**
     * Main method, used to run the application.
     *
     * @param args the command line arguments
     * @throws UnknownHostException if the local host name could not be resolved into an address
     */
    public static void main(String[] args) throws UnknownHostException {
        SpringApplication app = new SpringApplication(SiriusuniversityApp.class);
        DefaultProfileUtil.addDefaultProfile(app);
        Environment env = app.run(args).getEnvironment();
        String protocol = "http";
        if (env.getProperty("server.ssl.key-store") != null) {
            protocol = "https";
        }
        log.info("\n----------------------------------------------------------\n\t" +
                "Application '{}' is running! Access URLs:\n\t" +
                "Local: \t\t{}://localhost:{}\n\t" +
                "External: \t{}://{}:{}\n\t" +
                "Profile(s): \t{}\n----------------------------------------------------------",
            env.getProperty("spring.application.name"),
            protocol,
            env.getProperty("server.port"),
```

```
            protocol,
            InetAddress.getLocalHost().getHostAddress(),
            env.getProperty("server.port"),
            env.getActiveProfiles());
    }
}
```

Listagem 2 – Configuração da classe *SiriusuniversityApp*

A segurança da aplicação foi realizada por meio da criação da classe *SecurityConfiguration* que estende a classe *WebSecurityConfigurerAdapter*. Nesta classe foi criado o método *configure* para realizar as configurações de acesso às *Application Programming Interface* (*APIs)*, conforme ilustrado na Listagem 3.

```
@Override
    public void configure(WebSecurity web) throws Exception {
        web.ignoring()
            .antMatchers(HttpMethod.OPTIONS, "/**")
            .antMatchers("/app/**/*.{js,html}")
            .antMatchers("/i18n/**")
            .antMatchers("/content/**")
            .antMatchers("/swagger-ui/index.html")
            .antMatchers("/test/**");
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .addFilterBefore(corsFilter, UsernamePasswordAuthenticationFilter.class)
            .exceptionHandling()
            .authenticationEntryPoint(http401UnauthorizedEntryPoint())
        .and()
            .csrf()
            .disable()
            .headers()
            .frameOptions()
            .disable()
        .and()
            .sessionManagement()
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
        .and()
            .authorizeRequests()
            .antMatchers("/api/register").permitAll()
            .antMatchers("/api/activate").permitAll()
            .antMatchers("/api/authenticate").permitAll()
            .antMatchers("/api/account/reset_password/init").permitAll()
            .antMatchers("/api/account/reset_password/finish").permitAll()
            .antMatchers("/api/profile-info").permitAll()
            .antMatchers("/api/**").authenticated()
            .antMatchers("/management/health").permitAll()
            .antMatchers("/management/**").hasAuthority(AuthoritiesConstants.ADMIN)
            .antMatchers("/v2/api-docs/**").permitAll()
            .antMatchers("/swagger-resources/configuration/ui").permitAll()
            .antMatchers("/swagger-ui/index.html").hasAuthority(AuthoritiesConstants.ADMIN)
        .and()
```

```
        .apply(securityConfigurerAdapter());

    }
```

Listagem 3 - Método *configure* para realizar as configurações de acesso às *APIs*

Como pode ser observado na Listagem 3 a maioria das requisições deve ser autenticada. Para isso foi utilizado o padrão *JSON Web Token* (JWT) para composição do *token*. Desta forma, é necessário indicar as classes responsáveis por realizar a verificação se o *token* é válido. Na Listagem 4 é possível observar a validação do *token* pelo *filter* do *servlet* em que a aplicação está sendo executada.

```
public class JWTFilter extends GenericFilterBean {

    private TokenProvider tokenProvider;

    public JWTFilter(TokenProvider tokenProvider) {
        this.tokenProvider = tokenProvider;
    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain
filterChain)
        throws IOException, ServletException {
        HttpServletRequest httpServletRequest = (HttpServletRequest) servletRequest;
        String jwt = resolveToken(httpServletRequest);
        if (StringUtils.hasText(jwt) && this.tokenProvider.validateToken(jwt)) {
            Authentication authentication = this.tokenProvider.getAuthentication(jwt);
            SecurityContextHolder.getContext().setAuthentication(authentication);
        }
        filterChain.doFilter(servletRequest, servletResponse);
    }

    private String resolveToken(HttpServletRequest request){
        String bearerToken = request.getHeader(JWTConfigurer.AUTHORIZATION_HEADER);
        if (StringUtils.hasText(bearerToken) && bearerToken.startsWith("Bearer ")) {
            return bearerToken.substring(7, bearerToken.length());
        }
        return null;
    }
}
```

Listagem 4 – Validação do token pelo filter do servlet

Para troca de informações entre a API e o consumidor, foi utilizada a biblioteca Jackson, com isso é necessário realizar algumas configurações no projeto, conforme exibe a Listagem 5.

```
@Configuration
public class JacksonConfiguration {

    /*
     * Support for Hibernate types in Jackson.
```

```
     */
    @Bean
    public Hibernate5Module hibernate5Module() {      return new Hibernate5Module();   }

    /*
     * Jackson Afterburner module to speed up serialization/deserialization.
     */
    @Bean
    public AfterburnerModule afterburnerModule() {       return new AfterburnerModule();   }
}
```

Listagem 5 – Classe de configuração de serialização e deserialização de objetos

O sistema foi preparado para quando não existir a estrutura do banco de dados criada, executar os *scripts* necessários para criar tabelas e relacionamentos conforme o esperado. Para essa tarefa foi utilizada a biblioteca Liquibase e realizadas as configurações necessárias no projeto, conforme Listagem 6.

```
@Configuration
@EnableJpaRepositories("br.edu.utfpr.sirius.university.repository")
@EnableJpaAuditing(auditorAwareRef = "springSecurityAuditorAware")
@EnableTransactionManagement
public class DatabaseConfiguration {

    private final Logger log = LoggerFactory.getLogger(DatabaseConfiguration.class);

    private final Environment env;

    public DatabaseConfiguration(Environment env) {
        this.env = env;
    }

    @Bean
    public SpringLiquibase liquibase(@Qualifier("taskExecutor") TaskExecutor taskExecutor,
            DataSource dataSource, LiquibaseProperties liquibaseProperties) {

        // Use liquibase.integration.spring.SpringLiquibase if you don't want Liquibase to start asynchronously
        SpringLiquibase liquibase = new AsyncSpringLiquibase(taskExecutor, env);
        liquibase.setDataSource(dataSource);
        liquibase.setChangeLog("classpath:config/liquibase/master.xml");
        liquibase.setContexts(liquibaseProperties.getContexts());
        liquibase.setDefaultSchema(liquibaseProperties.getDefaultSchema());
        liquibase.setDropFirst(liquibaseProperties.isDropFirst());
        if (env.acceptsProfiles(JHipsterConstants.SPRING_PROFILE_NO_LIQUIBASE)) {
            liquibase.setShouldRun(false);
        } else {
            liquibase.setShouldRun(liquibaseProperties.isEnabled());
            log.debug("Configuring Liquibase");
        }
        return liquibase;
    }
}
```

Listagem 6 – Configuração para atualização da estrutura de banco de dados

Os arquivos que realizam a criação da estrutura do banco de dados ficam todos relacionados ao arquivo *master.xml*, encontrado na pasta *resources* do projeto. Sempre que a aplicação é executada no *container*, o Liquibase faz a verificação se é necessário executar alguma alteração na estrutura de banco de dados. A Listagem 7 apresenta os arquivos que possuem definição de estrutura de banco de dados.

```
<?xml version="1.0" encoding="utf-8"?>
<databaseChangeLog
    xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.5.xsd">

    <include file="config/liquibase/changelog/00000000000000_initial_schema.xml"
relativeToChangelogFile="false"/>
    <include file="config/liquibase/changelog/20170926025627_added_entity_Estado.xml"
relativeToChangelogFile="false"/>
    <include file="config/liquibase/changelog/20170926030254_added_entity_Cidade.xml"
relativeToChangelogFile="false"/>
    <include file="config/liquibase/changelog/20170926030939_added_entity_Empresa.xml"
relativeToChangelogFile="false"/>
    <include file="config/liquibase/changelog/20170926031859_added_entity_Area.xml"
relativeToChangelogFile="false"/>
    <include file="config/liquibase/changelog/20170926032019_added_entity_Tecnologia.xml"
relativeToChangelogFile="false"/>
    <include file="config/liquibase/changelog/20170926032113_added_entity_Profissao.xml"
relativeToChangelogFile="false"/>
    <include file="config/liquibase/changelog/20170926032222_added_entity_Funcao.xml"
relativeToChangelogFile="false"/>
    <include file="config/liquibase/changelog/20170926032358_added_entity_Questionario.xml"
relativeToChangelogFile="false"/>
    <include file="config/liquibase/changelog/20170926033215_added_entity_Pergunta.xml"
relativeToChangelogFile="false"/>
    <include file="config/liquibase/changelog/20171017230839_added_entity_Alternativa.xml"
relativeToChangelogFile="false"/>
    <include file="config/liquibase/changelog/20171021203408_added_entity_AtividadeProfissional.xml"
relativeToChangelogFile="false"/>
    <include file="config/liquibase/changelog/20171114105924_added_entity_QuestionarioPergunta.xml"
relativeToChangelogFile="false"/>
    <include file="config/liquibase/changelog/20171114131118_added_entity_AlternativaCorreta.xml"
relativeToChangelogFile="false"/>
    <include file="config/liquibase/changelog/20171115154550_added_entity_AtividadeProfissionalTecnologia.xml"
relativeToChangelogFile="false"/>
    <include file="config/liquibase/changelog/20171115195354_added_entity_Historico.xml"
relativeToChangelogFile="false"/>
    <include file="config/liquibase/changelog/20171115215844_added_entity_HistoricoUsuario.xml"
relativeToChangelogFile="false"/>
    <include file="config/liquibase/changelog/20171115224416_added_entity_Resposta.xml"
relativeToChangelogFile="false"/>
    <include file="config/liquibase/changelog/20171119185749_added_entity_QuestionarioRespondido.xml"
relativeToChangelogFile="false"/>
    <include file="config/liquibase/changelog/20171120235218_added_entity_Comentario.xml"
relativeToChangelogFile="false"/>
    <!-- jhipster-needle-liquibase-add-changelog - JHipster will add liquibase changelogs here -->
```

```
    <include file="config/liquibase/changelog/20170926030254_added_entity_constraints_Cidade.xml"
relativeToChangelogFile="false"/>
    <include file="config/liquibase/changelog/20170926030939_added_entity_constraints_Empresa.xml"
relativeToChangelogFile="false"/>
    <include file="config/liquibase/changelog/20171017230839_added_entity_constraints_Alternativa.xml"
relativeToChangelogFile="false"/>
    <include
file="config/liquibase/changelog/20171021203408_added_entity_constraints_AtividadeProfissional.xml"
relativeToChangelogFile="false"/>
    <include
file="config/liquibase/changelog/20171114105924_added_entity_constraints_QuestionarioPergunta.xml"
relativeToChangelogFile="false"/>
    <include file="config/liquibase/changelog/20171114131118_added_entity_constraints_AlternativaCorreta.xml"
relativeToChangelogFile="false"/>
    <include
file="config/liquibase/changelog/20171115154550_added_entity_constraints_AtividadeProfissionalTecnologia.xml"
relativeToChangelogFile="false"/>
    <include file="config/liquibase/changelog/20171115215844_added_entity_constraints_HistoricoUsuario.xml"
relativeToChangelogFile="false"/>
    <include file="config/liquibase/changelog/20171115224416_added_entity_constraints_Resposta.xml"
relativeToChangelogFile="false"/>
    <include
file="config/liquibase/changelog/20171119185749_added_entity_constraints_QuestionarioRespondido.xml"
relativeToChangelogFile="false"/>
    <include file="config/liquibase/changelog/20171120235218_added_entity_constraints_Comentario.xml"
relativeToChangelogFile="false"/>
    <!-- jhipster-needle-liquibase-add-constraints-changelog - JHipster will add liquibase constraints changelogs here
-->
     <include file="config/liquibase/changelog/20171115204123_refactory_entities_Questionario.xml"
relativeToChangelogFile="false"/>
</databaseChangeLog>
```

Listagem 7 – Arquivos de atualização da estrutura de banco de dados

A comunicação entre a aplicação e o banco de dados é feita utilizando a interface *Java Persistence API* (JPA). Para isso foi utilizado o *framework* Hibernate. Essa configuração pode ser encontrada no arquivo *application-dev.yml*, conforme mostra a Listagem 8.

```
spring:
    profiles:
        active: dev
        include: swagger
    devtools:
        restart:
            enabled: true
        livereload:
            enabled: false # we use gulp + BrowserSync for livereload
    jackson:
        serialization.indent_output: true
    datasource:
        type: com.zaxxer.hikari.HikariDataSource
        url: jdbc:postgresql://localhost:5432/siriusuniversity
        username: siriusuniversity
        password:
    jpa:
        database-platform: io.github.jhipster.domain.util.FixedPostgreSQL82Dialect
```

```
    database: POSTGRESQL
    show-sql: true
    properties:
      hibernate.id.new_generator_mappings: true
      hibernate.cache.use_second_level_cache: false
      hibernate.cache.use_query_cache: false
      hibernate.generate_statistics: true
  mail:
    host: localhost
    port: 25
    username:
    password:
  messages:
    cache-seconds: 1
  thymeleaf:
    cache: false
liquibase:
  contexts: dev
```

Listagem 8 – Configuração de acesso ao banco de dados

Para realizar a comunicação da aplicação com as informações do banco de dados foram criadas classes de entidades para cada tabela existente na estrutura. Conforme exibe a Listagem 9, a classe *Alternativa.java* foi criado seguindo a técnica *Object Relational Mapping* (ORM) suportado pelo JPA, assim como as demais entidades do sistema foram criadas seguindo esse padrão.

```
@Entity
@Table(name = "alternativa")
public class Alternativa implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "sequenceGenerator")
    @SequenceGenerator(name = "sequenceGenerator")
    private Long id;

    @NotNull
    @Lob
    @Column(name = "ds_alternativa", nullable = false)
    private String dsAlternativa;

    @NotNull
    @Size(min = 1, max = 1)
    @Column(name = "ds_identificacao", length = 1, nullable = false)
    private String dsIdentificacao;

    @ManyToOne(optional = false)
    @NotNull
```

Listagem 9 – Classe da entidade Alternativa

A comunicação entre objetos Java e banco de dados é feita pelas *interfaces* presentes no pacote *repository*. Essas interfaces utilizando o Spring Data para montar as instruções SQL, conforme mostrado o arquivo *TecnologiaRepository.java* na Listagem 10.

```
/**
 * Spring Data JPA repository for the Tecnologia entity.
 */
@SuppressWarnings("unused")
@Repository
public interface TecnologiaRepository extends JpaRepository<Tecnologia,Long> {

}
```

Listagem 10 – Classe responsável pela comunicação com banco de dados

As classes responsáveis pela execução de validações, assim como regras de negócio da aplicação ficam localizadas no pacote *service* da aplicação. Na listagem 11 é possível observar os métodos implementados em todas as classes *services*.

```
@Service
@Transactional
public class AreaService {

    private final Logger log = LoggerFactory.getLogger(AreaService.class);

    private final AreaRepository areaRepository;

    public AreaService(AreaRepository areaRepository) {
        this.areaRepository = areaRepository;
    }

    /**
     * Save a area.
     *
     * @param area the entity to save
     * @return the persisted entity
     */
    public Area save(Area area) {
        log.debug("Request to save Area : {}", area);
        return areaRepository.save(area);
    }

    /**
     *  Get all the areas.
     *
     *  @param pageable the pagination information
     *  @return the list of entities
```

```
     */
    @Transactional(readOnly = true)
    public Page<Area> findAll(Pageable pageable) {
        log.debug("Request to get all Areas");
        return areaRepository.findAll(pageable);
    }

    /**
     *  Get one area by id.
     *
     *  @param id the id of the entity
     *  @return the entity
     */
    @Transactional(readOnly = true)
    public Area findOne(Long id) {
        log.debug("Request to get Area : {}", id);
        return areaRepository.findOne(id);
    }

    /**
     *  Delete the  area by id.
     *
     *  @param id the id of the entity
     */
    public void delete(Long id) {
        log.debug("Request to delete Area : {}", id);
        areaRepository.delete(id);
    }
}
```

Listagem 11 – Classe responsável por executar regras de negócio

As exposições das funcionalidades do sistema são todas feitas por comunicação via *Representational State Transfer (*REST). As classes responsáveis por essa operação ficam localizadas no pacote *rest*. Na Listagem 12 é apresentado o contrato de *API* para realizar operações *RESTful* sobre as entidades.

```
@PostMapping("/cidades")
    @Timed
    public ResponseEntity<Cidade> createCidade(@Valid @RequestBody Cidade cidade) throws
URISyntaxException {
        log.debug("REST request to save Cidade : {}", cidade);
        if (cidade.getId() != null) {
            return ResponseEntity.badRequest().headers(HeaderUtil.createFailureAlert(ENTITY_NAME, "idexists",
"A new cidade cannot already have an ID")).body(null);
        }
        Cidade result = cidadeService.save(cidade);
        return ResponseEntity.created(new URI("/api/cidades/" + result.getId()))
```

```
            .headers(HeaderUtil.createEntityCreationAlert(ENTITY_NAME, result.getId().toString()))
            .body(result);
    }
    /**
     * PUT  /cidades : Updates an existing cidade.
     *
     * @param cidade the cidade to update
     * @return the ResponseEntity with status 200 (OK) and with body the updated cidade,
     * or with status 400 (Bad Request) if the cidade is not valid,
     * or with status 500 (Internal Server Error) if the cidade couldn't be updated
     * @throws URISyntaxException if the Location URI syntax is incorrect
     */
    @PutMapping("/cidades")
    @Timed
    public ResponseEntity<Cidade> updateCidade(@Valid @RequestBody Cidade cidade) throws
URISyntaxException {
        log.debug("REST request to update Cidade : {}", cidade);
        if (cidade.getId() == null) {
            return createCidade(cidade);
        }
        Cidade result = cidadeService.save(cidade);
        return ResponseEntity.ok()
            .headers(HeaderUtil.createEntityUpdateAlert(ENTITY_NAME, cidade.getId().toString()))
            .body(result);
    }

    /**
     * GET  /cidades : get all the cidades.
     *
     * @param pageable the pagination information
     * @return the ResponseEntity with status 200 (OK) and the list of cidades in body
     */
    @GetMapping("/cidades")
    @Timed
    public ResponseEntity<List<Cidade>> getAllCidades(@ApiParam Pageable pageable) {
        log.debug("REST request to get a page of Cidades");
        Page<Cidade> page = cidadeService.findAll(pageable);
        HttpHeaders headers = PaginationUtil.generatePaginationHttpHeaders(page, "/api/cidades");
        return new ResponseEntity<>(page.getContent(), headers, HttpStatus.OK);
    }

    /**
     * GET  /cidades/:id : get the "id" cidade.
     *
     * @param id the id of the cidade to retrieve
     * @return the ResponseEntity with status 200 (OK) and with body the cidade, or with status 404 (Not
Found)
     */
    @GetMapping("/cidades/{id}")
    @Timed
    public ResponseEntity<Cidade> getCidade(@PathVariable Long id) {
        log.debug("REST request to get Cidade : {}", id);
        Cidade cidade = cidadeService.findOne(id);
        return ResponseUtil.wrapOrNotFound(Optional.ofNullable(cidade));
    }
    /**
     * DELETE  /cidades/:id : delete the "id" cidade.
```

```
     *
     * @param id the id of the cidade to delete
     * @return the ResponseEntity with status 200 (OK)
     */
    @DeleteMapping("/cidades/{id}")
    @Timed
    public ResponseEntity<Void> deleteCidade(@PathVariable Long id) {
        log.debug("REST request to delete Cidade : {}", id);
        cidadeService.delete(id);
        return ResponseEntity.ok().headers(HeaderUtil.createEntityDeletionAlert(ENTITY_NAME,
id.toString())).build();
    }
```

Listagem 12 – Classe de exposição de métodos à *API*

Para o desenvolvimento *fron-tend* da aplicação foi utilizado Angular na versão 4 e Bootstrap na versão 4. Como a maioria das APIs necessitam de *token* no *header* da requisição, foi implementada uma classe *interceptor* genérica para enviar essa informação toda vez que solicitado, conforme mostrado na Listagem 13.

```
export class AuthInterceptor extends JhiHttpInterceptor {
    constructor(
        private localStorage: LocalStorageService,
        private sessionStorage: SessionStorageService
    ) {
        super();
    }

    requestIntercept(options?: RequestOptionsArgs): RequestOptionsArgs {
        const token = this.localStorage.retrieve('authenticationToken') ||
this.sessionStorage.retrieve('authenticationToken');
        if (!!token) {
            options.headers.append('Authorization', 'Bearer ' + token);
        }
        return options;
    }

    responseIntercept(observable: Observable<Response>): Observable<Response> {
        return observable; // by pass
    }
}
```

Listagem 13 – Interceptador de requisições via HTTP

Para enviar o *token* é necessário que o mesmo esteja armazenado no *local storage* do navegador, para isso é necessário realizar o *login* pelo sistema, permitido por meio do método *login* implementado conforme Listagem 14.

```
login(credentials): Observable<any> {

        const data = {
            username: credentials.username,
            password: credentials.password,
            rememberMe: credentials.rememberMe
        };
```

```
        return this.http.post('api/authenticate', data).map(authenticateSuccess.bind(this));

        function authenticateSuccess(resp) {
            const bearerToken = resp.headers.get('Authorization');
            if (bearerToken && bearerToken.slice(0, 7) === 'Bearer ') {
                const jwt = bearerToken.slice(7, bearerToken.length);
                this.storeAuthenticationToken(jwt, credentials.rememberMe);
                return jwt;
            }
        }
    }
```

Listagem 14 – Método de login do usuário no sistema

O desenvolvimento das entidades foi agrupado em um módulo com nome *SiriusuniversityEntityModule*. Esse arquivo possui o relacionamento com todos os módulos das entidades disponíveis no sistema, conforme exibido na Listagem 15.

```
import...
/* jhipster-needle-add-entity-module-import - JHipster will add entity modules imports here */

@NgModule({
    imports: [
        SiriusuniversityEstadoModule,
        SiriusuniversityCidadeModule,
        SiriusuniversityEmpresaModule,
        SiriusuniversityAreaModule,
        SiriusuniversityAlunoModule,
        SiriusuniversityTecnologiaModule,
        SiriusuniversityProfissaoModule,
        SiriusuniversityFuncaoModule,
        SiriusuniversityQuestionarioModule,
        SiriusuniversityPerguntaModule,
        SiriusuniversityAlternativaModule,
        SiriusuniversityAtividadeProfissionalModule,
        SiriusuniversityQuestionarioPerguntaModule,
        SiriusuniversityCampusCursoModule,
        SiriusuniversityAlternativaCorretaModule,
        SiriusuniversityAtividadeProfissionalTecnologiaModule,
        SiriusuniversityHistoricoModule,
        SiriusuniversityHistoricoUsuarioModule,
        SiriusuniversityQuestionarioDisponivelModule,
        SiriusuniversityRespostaModule,
        SiriusuniversityQuestionarioRespostaModule,
        SiriusuniversityQuestionarioRespondidoModule,
        SiriusuniversityComentarioModule,
        SiriusuniversityComentarioValidarModule,
        SiriusuniversityComentarioConsultaModule,
        SiriusuniversityQuestionarioRespondidoConsultaModule,
        SiriusuniversityQuestionarioNaoRespondidoConsultaModule,
        SiriusuniversityHistoricoConsultaModule,
        /* jhipster-needle-add-entity-module - JHipster will add entity modules here */
    ],
    declarations: [],
    entryComponents: [],
```

```
    providers: [],
    schemas: [CUSTOM_ELEMENTS_SCHEMA]
})
export class SiriusuniversityEntityModule {}
```

Listagem 15 – Módulos disponíveis no sistema

Para o acesso às entidades que é permitido pelo menu, foi criada uma *navbar*, possuindo o *link* para cada entidade por meio da rota definida no módulo. A Listagem 16 tem o exemplo de algumas opções do menu.

```
<li *jhiHasAnyAuthority="'ROLE_ADMIN'" ngbDropdown class="nav-item dropdown pointer"
routerLinkActive="active" [routerLinkActiveOptions]="{exact: true}">
            <a class="nav-link dropdown-toggle" ngbDropdownToggle href="javascript:void(0);" id="entity-
menu">
                <span>
                    <i class="fa fa-th-list" aria-hidden="true"></i>
                    <span jhiTranslate="global.menu.entities.main">
                        Entities
                    </span>
                    <b class="caret"></b>
                </span>
            </a>
            <ul class="dropdown-menu" ngbDropdownMenu>
                <li>
                    <a class="dropdown-item" routerLink="estado" routerLinkActive="active"
[routerLinkActiveOptions]="{ exact: true }" (click)="collapseNavbar()">
                        <i class="fa fa-fw fa-asterisk" aria-hidden="true"></i>
                        <span jhiTranslate="global.menu.entities.estado">Estado</span>
                    </a>
                </li>
                <li>
                    <a class="dropdown-item" routerLink="cidade" routerLinkActive="active"
[routerLinkActiveOptions]="{ exact: true }" (click)="collapseNavbar()">
                        <i class="fa fa-fw fa-asterisk" aria-hidden="true"></i>
                        <span jhiTranslate="global.menu.entities.cidade">Cidade</span>
                    </a>
                </li>
                <li>
                    <a class="dropdown-item" routerLink="empresa" routerLinkActive="active"
[routerLinkActiveOptions]="{ exact: true }" (click)="collapseNavbar()">
                        <i class="fa fa-fw fa-asterisk" aria-hidden="true"></i>
                        <span jhiTranslate="global.menu.entities.empresa">Empresa</span>
                    </a>
                </li>
                <li>
                    <a class="dropdown-item" routerLink="area" routerLinkActive="active"
[routerLinkActiveOptions]="{ exact: true }" (click)="collapseNavbar()">
                        <i class="fa fa-fw fa-asterisk" aria-hidden="true"></i>
                        <span jhiTranslate="global.menu.entities.area">Area</span>
                    </a>
                </li>
                <li>
                    <a class="dropdown-item" routerLink="tecnologia" routerLinkActive="active"
[routerLinkActiveOptions]="{ exact: true }" (click)="collapseNavbar()">
                        <i class="fa fa-fw fa-asterisk" aria-hidden="true"></i>
```

```
                    <span jhiTranslate="global.menu.entities.tecnologia">Tecnologia</span>
                </a>
            </li>
```

Listagem 16 – Opções de menu do sistema

O mesmo padrão de desenvolvimento foi seguido para todas as entidades do sistema. A seguir é apresentado o desenvolvimento de uma entidade, visto que as restantes seguiram a mesma arquitetura e codificação semelhante.

O arquivo *profissao.componente.html* possui a codificação necessária para renderizar a listagem de profissões no *browser*, juntamente com as opções de visualizar, editar e excluir cada profissão. Na tela de listagem, também existe um botão para cadastrar uma nova profissão, conforme exibido na Listagem 17.

```
<div>
    <h2>
        <span jhiTranslate="siriusuniversityApp.profissao.home.title">Profissaos</span>
        <button class="btn btn-primary float-right jh-create-entity create-profissao" [routerLink]="['/', { outlets: {
popup: ['profissao-new'] } }]">
            <span class="fa fa-plus"></span>
            <span  jhiTranslate="siriusuniversityApp.profissao.home.createLabel">
            Create new Profissao
            </span>
        </button>
    </h2>
    <jhi-alert></jhi-alert>
    <div class="row">
    </div>
    <br/>
    <div class="table-responsive" *ngIf="profissaos">
        <table class="table table-striped">
            <thead>
            <tr jhiSort [(predicate)]="predicate" [(ascending)]="reverse" [callback]="reset.bind(this)">
            <th jhiSortBy="id"><span jhiTranslate="global.field.id">ID</span> <span class="fa fa-
sort"></span></th>
            <th jhiSortBy="dsProfissao"><span jhiTranslate="siriusuniversityApp.profissao.dsProfissao">Ds
Profissao</span> <span class="fa fa-sort"></span></th>
            <th></th>
            </tr>
            </thead>
            <tbody infinite-scroll (scrolled)="loadPage(page + 1)" [infiniteScrollDisabled]="page >= links['last']"
[infiniteScrollDistance]="0">
            <tr *ngFor="let profissao of profissaos ;trackBy: trackId">
                <td><a [routerLink]="['../profissao', profissao.id ]">{{profissao.id}}</a></td>
                <td>{{profissao.dsProfissao}}</td>
                <td class="text-right">
                    <div class="btn-group flex-btn-group-container">
                        <button type="submit"
                                [routerLink]="['../profissao', profissao.id ]"
                                class="btn btn-info btn-sm">
                            <span class="fa fa-eye"></span>
                            <span class="hidden-md-down" jhiTranslate="entity.action.view">View</span>
```

```
                </button>
                <button type="submit"
                        [routerLink]="['/', { outlets: { popup: 'profissao/'+ profissao.id + '/edit'} }]"
                        replaceUrl="true"
                        class="btn btn-primary btn-sm">
                    <span class="fa fa-pencil"></span>
                    <span class="hidden-md-down" jhiTranslate="entity.action.edit">Edit</span>
                </button>
                <button type="submit"
                        [routerLink]="['/', { outlets: { popup: 'profissao/'+ profissao.id + '/delete'} }]"
                        replaceUrl="true"
                        class="btn btn-danger btn-sm">
                    <span class="fa fa-remove"></span>
                    <span class="hidden-md-down" jhiTranslate="entity.action.delete">Delete</span>
                </button>
            </div>
        </td>
    </tr>
    </tbody>
</table>
</div>
</div>
```

Listagem 17 – Listagem de registros cadastrados no sistema

O arquivo *profissao.componente.ts* possui a codificação TypeScript para gerenciamento da listagem de profissões. Esta classe é responsável pela chamada de métodos para buscar as informações para exibir em tela, assim como direcionar para outras rotas do sistema quando solicitada alguma ação, como, por exemplo, editar um registro. A Listagem 18 mostra o corpo da classe que faz o gerenciamento da listagem.

```
@Component({
    selector: 'jhi-profissao',
    templateUrl: './profissao.component.html'
})
export class ProfissaoComponent implements OnInit, OnDestroy {

    profissaos: Profissao[];
    currentAccount: any;
    eventSubscriber: Subscription;
    itemsPerPage: number;
    links: any;
    page: any;
    predicate: any;
    queryCount: any;
    reverse: any;
    totalItems: number;

    constructor(
        private profissaoService: ProfissaoService,
        private alertService: JhiAlertService,
        private eventManager: JhiEventManager,
        private parseLinks: JhiParseLinks,
        private principal: Principal
    ) {
```

```
        this.profissaos = [];
        this.itemsPerPage = ITEMS_PER_PAGE;
        this.page = 0;
        this.links = {
            last: 0
        };
        this.predicate = 'id';
        this.reverse = true;
    }

    loadAll() {
        this.profissaoService.query({
            page: this.page,
            size: this.itemsPerPage,
            sort: this.sort()
        }).subscribe(
            (res: ResponseWrapper) => this.onSuccess(res.json, res.headers),
            (res: ResponseWrapper) => this.onError(res.json)
        );
    }
```

Listagem 18 – Classe responsável pelo controle da tela de listagem

Para cada entidade existente é necessário possuir uma classe com a declaração dos atributos existentes, essa classe serve para definir o tipo do objeto que está sendo trafegado nas requisições *Hypertext Transfer Protocol* (HTTP) e também no momento de exibir em tela. O arquivo *profissao.model.ts* mostra os atributos existentes para uma profissão.

```
import { BaseEntity } from './../../shared';

export class Profissao implements BaseEntity {
    constructor(
        public id?: number,
        public dsProfissao?: string,
    ) {
    }
}
```

Listagem 19 – Classe com a definição do modelo do objeto

Cada entidade deve expor as rotas que deseja atender, para isso, é necessário criar um arquivo contendo essa configuração, demonstrando que quando foi informado determinado endereço na *Uniform Resource Locator* (URL), será exibida alguma tela da entidade. A Listagem 20 mostra o arquivo *profissao.route.ts* contendo a configuração necessária para capturar as rotas atendidas pelo módulo.

```
export const profissaoRoute: Routes = [
    {
        path: 'profissao',
        component: ProfissaoComponent,
        data: {
            authorities: ['ROLE_USER'],
            pageTitle: 'siriusuniversityApp.profissao.home.title'
```

```
        },
        canActivate: [UserRouteAccessService]
    }, {
        path: 'profissao/:id',
        component: ProfissaoDetailComponent,
        data: {
            authorities: ['ROLE_USER'],
            pageTitle: 'siriusuniversityApp.profissao.home.title'
        },
        canActivate: [UserRouteAccessService]
    }
];
```

Listagem 20 – Rotas disponíveis do sistema

A comunicação com o *back-end* é realizada sempre por meio do módulo *http* disponibilizado pelo próprio Angular, mas todas essas requisições ao *backend* são centralizadas nos arquivos *services* de cada módulo. Para isso foi criado o arquivo profissao.service.ts possuindo os métodos necessários para requisição REST, conforme Listagem 21.

```
@Injectable()
export class ProfissaoService {

    private resourceUrl = 'api/profissaos';

    constructor(private http: Http) { }

    create(profissao: Profissao): Observable<Profissao> {
        const copy = this.convert(profissao);
        return this.http.post(this.resourceUrl, copy).map((res: Response) => {
            return res.json();
        });
    }

    update(profissao: Profissao): Observable<Profissao> {
        const copy = this.convert(profissao);
        return this.http.put(this.resourceUrl, copy).map((res: Response) => {
            return res.json();
        });
    }

    find(id: number): Observable<Profissao> {
        return this.http.get(`${this.resourceUrl}/${id}`).map((res: Response) => {
            return res.json();
        });
    }

    query(req?: any): Observable<ResponseWrapper> {
        const options = createRequestOption(req);
        return this.http.get(this.resourceUrl, options)
            .map((res: Response) => this.convertResponse(res));
    }

    delete(id: number): Observable<Response> {
```

```
        return this.http.delete(`${this.resourceUrl}/${id}`);
    }

    private convertResponse(res: Response): ResponseWrapper {
        const jsonResponse = res.json();
        return new ResponseWrapper(res.headers, jsonResponse, res.status);
    }

    private convert(profissao: Profissao): Profissao {
        const copy: Profissao = Object.assign({}, profissao);
        return copy;
    }
}
```

Listagem 21 – Métodos para realizar requisições ao back-end

O conceito adotado para desenvolvimento desta aplicação é de que cada entidade é um módulo do sistema, portanto, a codificação da entidade precisa se comportar de forma independente e expor suas funcionalidades sem depender de outras entidades. Para isso é criado o arquivo *profissao.module.ts* possuindo essas configurações, como as rotas que ele suporta e as classes TypeScript que serão expostas junto com o modulo, conforme demonstrado na Listagem 22.

```
const ENTITY_STATES = [
    ...profissaoRoute,
    ...profissaoPopupRoute,
];

@NgModule({
    imports: [
        SiriusuniversitySharedModule,
        RouterModule.forRoot(ENTITY_STATES, { useHash: true })
    ],
    declarations: [
        ProfissaoComponent,
        ProfissaoDetailComponent,
        ProfissaoDialogComponent,
        ProfissaoDeleteDialogComponent,
        ProfissaoPopupComponent,
        ProfissaoDeletePopupComponent,
    ],
    entryComponents: [
        ProfissaoComponent,
        ProfissaoDialogComponent,
        ProfissaoPopupComponent,
        ProfissaoDeleteDialogComponent,
        ProfissaoDeletePopupComponent,
    ],
    providers: [
        ProfissaoService,
        ProfissaoPopupService,
    ],
    schemas: [CUSTOM_ELEMENTS_SCHEMA]
})
```

```
export class SiriusuniversityProfissaoModule {}
```

Listagem 22 – Definição do módulo da entidade

4 CONCLUSÃO

Este trabalho apresentou o desenvolvimento de uma aplicação *web* que visa ser um canal de comunicação da Universidade com os seus egressos. A aplicação oferece recursos de controle de histórico profissional e de formação , gerenciamento de comentários, composição e consulta de questionários respondidos por egressos.

O desenvolvimento do projeto foi realizado utilizando suporte do *framework* JHispter, que auxiliou na produtividade de criação da estrutura básica de cada *Create, Retrieve, Update and Delete* (CRUD), segurança no *back-end* e *front-end* e *templates* no *front-end*.

Durante o desenvolvimento da aplicação, a falta de conhecimento em algumas ferramentas causou dificuldade no processo de criação da aplicação. Com o auxílio de tutoriais, documentação oficial e fóruns foi possível superar as dificuldades e alcançar o objetivo proposto para o projeto. Por fim, pode-se ressaltar o conhecimento adquirido utilizando JHipster, Angular 4, Spring Boot, Spring Data e Liquibase.

Com a utilização do sistema apresentado é possível obter um *feedback* sobre a situação do egresso após a conclusão curso, se está atuando na área, qual sua evolução profissional e como o curso ajudou na sua formação. Também permite a coleta de informações sobre a experiência do egresso durante o decorrer do curso, e mais variadas questões, como, por exemplo, infraestrutura ofertada pelo Câmpus, didática utilizada pelos professores, atualidade das informações apresentadas, entre outras.

Como trabalhos futuros pretende-se criar novas consultas permitindo maior detalhamento das informações pertencentes ao sistema. Também implementar a geração de relatórios e gráficos.

REFERÊNCIAS

ALMEIDA, Flávio. **Mean**: Full stack JavaScript para aplicações web com MongoDB, Express, Angular e Node. São Paulo: Casa do Código, 2015

BRASIL. Lei n.9.394 – 20 dez. 1.996. **Estabelece as diretrizes e bases da educação**. Diário Oficial, Brasília, 23 dez. 1996. p.4-27.

EIS Diego; SHIOTA Eduardo; MARQUES Deivid; GONDIM Caio; GOMES Jaydson; KEPPELEN Giovanni; DE LUNA Bernard. **Coletânea Front-end**: Uma antologia da comunidade front-end brasileira. São Paulo: Casa do Código, 2014.

FERREIRA, Aurélio Buarque Hollanda de. **Novo Aurélio século XXI: o dicionário da língua portuguesa**. 3.ed. Rio de Janeiro: Nova Fronteira, 1999.

GUEDES, THIAGO. **Crie aplicações com Angular**: o novo *framework* do Google. São Paulo: Casa do Código, 2017.

GUIMARÃES Maria Angélica Miranda, SALLES Mara Telles. O acompanhamento de egressos como ferramenta de inserção no mercado de trabalho. **Congresso Nacional de** Excelencia em Gestão.

KAZALE IT, **Curso básico de Angular 2**. Disponível em: <http://kazale.com/curso-basico-angular-2/>. Acesso em: 22 out. 2017.

PEREIRA, Michael Henrique R. **AngularJS**: uma abordagem prática e objetiva. São Paulo: Novatec, 2014.

PEREIRA Giveldna Maria Costa, CASTRO Felipe Nalon, LANZA Luciana Nunes Menolli, LANZA Daniel Caros Ferreira. Panorama de oportunidades para os egressos do ensino superior no Brasil: o papel da inovação na criação de novos mercados de trabalho. **Ensaio: Avaliação e Políticas Públicas Educacionais**. v. 24, n. 90, Rio de Janeiro, 2016.

TYPESCRIPT. Disponível em: < https://www.typescriptlang.org/>. Acesso em: 25 nov. 2017.