

**UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS**

RAFAEL ALVES DE LIMA

APLICATIVO WEB PARA GESTÃO DE PLANOS DE CARREIRA

TRABALHO DE CONCLUSÃO DE CURSO

**PATO BRANCO
2017**

RAFAEL ALVES DE LIMA

APLICATIVO WEB PARA GESTÃO DE PLANOS DE CARREIRA

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Tecnólogo.

Orientadora: Profa. Beatriz Terezinha Borsoi

PATO BRANCO
2017



Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Tecnologia em Análise e Desenvolvimento
de Sistemas



TERMO DE APROVAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO

APLICATIVO WEB PARA GESTÃO DE PLANOS DE CARREIRA

POR

RAFAEL ALVES DE LIMA

Este trabalho de conclusão de curso foi apresentado no dia 11 de dezembro de 2017, como requisito parcial para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, pela Universidade Tecnológica Federal do Paraná. O acadêmico foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Banca examinadora:

Profª Drª Beatriz Terezinha Borsoi
Orientador

Prof. MSc.Vinicius Pegorini

Profª MSc Andreia Scariot Beulke

Prof. Dr. Edilson Pontarolo
Coordenador do Curso de Tecnologia em
Análise e Desenvolvimento de Sistemas

Profª Drª Beatriz Terezinha Borsoi
Responsável pela Atividade de Trabalho de
Conclusão de Curso

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

RESUMO

LIMA, Rafael Alves de. Aplicativo *web* para gestão de planos de carreira. 2017. 63f. Monografia (Trabalho de Conclusão de Curso) - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2017.

Planos de carreira são utilizados pelas empresas como uma forma de os funcionários de serviços públicos ou privados alcançarem promoções e benefícios oferecidos pelas instituições. Para as instituições é uma forma de padronizar as definições necessárias para progressão em termos de cargos e funções e salarial. Para o funcionário serve como uma diretriz da sua trajetória na instituição e como balizador do que ele precisa cumprir para alcançar as progressões oferecidas pela empresa e para traçar o seu plano de carreira institucional. Para o serviço público é mais comum haver planos de carreira institucionalizados e até em escopo de governo (federal, estadual e municipal). Para as instituições, esses planos costumam ser definidos no âmbito de cada instituição. Aplicativos computacionais podem auxiliar no gerenciamento desses planos por parte dos avaliadores e para o funcionário que é avaliado. Considerando esse contexto, por meio deste trabalho foi desenvolvido um sistema web para gestão de planos de carreira. O sistema foi desenvolvido com a linguagem Java e o *framework* Spring MVC. A interface foi desenvolvida utilizando o *framework* Angular e Angular Material, com banco de dados em MySQL.

Palavras-chave: Angular. Java. Spring MVC. MySQL. Angular Material.

ABSTRACT

LIMA, Rafael Alves de. Web application to manager career plans. 2017. 63f. Monografia (Trabalho de Conclusão de Curso) - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2016.

Career plans are used by companies to achieve promotions and benefits offered by institutions. For institutions it is a way of standardizing the definitions needed for progression in terms of positions, functions, and salary. For the employee, this evaluation serves as a guideline to the work trajectory and as a guide of what is needed to meet the progressions offered by the company and to outline the company institutional career plan. For the public service it is more common to have institutionalized career plans and even in the scope of government (federal, state and municipal). For companies, these plans are usually defined in the scope of each institution. Computational applications can help the management of these plans by the evaluators and the employee who is evaluated. Considering this context, it was developed a web system using Java language and Spring MVC framework, the interface was developed using Angular 5 and Angular Material, and the database used was MySQL.

Keywords: Angular. Java language. Spring MVC. MySQL. Angular Material.

LISTA DE FIGURAS

| | |
|---|----|
| Figura 1 - Diagrama de Casos de Uso | 19 |
| Figura 2 - Diagrama de entidades e relacionamentos do banco de dados..... | 24 |
| Figura 3 - Listagem de usuários | 25 |
| Figura 4 - Cadastro de usuário..... | 26 |
| Figura 5 - Mensagem sucesso..... | 26 |
| Figura 6 - Mensagem falha..... | 27 |
| Figura 7 - Login do sistema..... | 28 |
| Figura 8 - Perfil do funcionário | 29 |
| Figura 9 - Listagem de avaliações do funcionário..... | 30 |
| Figura 10 - Cadastro de avaliação: parte 1 | 31 |
| Figura 11 - Cadastro de avaliação: parte 2..... | 32 |
| Figura 12 - Cadastro de avaliação: parte 3..... | 33 |
| Figura 13 - Cadastro de avaliação: parte 4..... | 34 |
| Figura 14 - Efetuar avaliação pelo supervisor..... | 35 |
| Figura 15 - Estrutura do projeto da API..... | 37 |
| Figura 16 - Estrutura de pastas do projeto Angular | 47 |

LISTA DE QUADROS

| | |
|--|----|
| Quadro 1 - Ferramentas e tecnologias utilizadas..... | 15 |
| Quadro 2 - Casos de Uso por requisito | 19 |
| Quadro 3 - Caso de uso cadastrar | 20 |
| Quadro 4 - Caso de uso consultar | 21 |
| Quadro 5 - Caso de uso alterar | 21 |
| Quadro 6 - Caso de uso solicitar avaliação | 22 |
| Quadro 7 - Caso de uso efetivar avaliação | 22 |
| Quadro 8 - Caso de uso cadastrar equipe | 23 |

LISTAGENS DE CÓDIGO

| | |
|---|----|
| Listagem 1 - Arquivo pom.xml | 39 |
| Listagem 2 - Exemplo da classe Cargo | 40 |
| Listagem 3 - Exemplo classe CargoDAO | 41 |
| Listagem 4 - Exemplo classe CargoController..... | 42 |
| Listagem 5 - Propriedades definidas para o Spring | 43 |
| Listagem 6 - Exemplo class AvaliacaoController | 46 |
| Listagem 7 - Componente principal da aplicação | 50 |
| Listagem 8 - <i>Template</i> HTML do componente inicial..... | 51 |
| Listagem 9 - Classe de rotas da aplicação..... | 52 |
| Listagem 10 - <i>Template</i> HTML do componente Perfil..... | 54 |
| Listagem 11 - Arquivo CSS do component Perfil..... | 55 |
| Listagem 12 - Classe TypeScript do componente Perfil..... | 57 |
| Listagem 13 - Classe TypeScript do componente Connection..... | 61 |

LISTA DE SIGLAS

| | |
|------|---|
| API | <i>Application Programming Interface</i> |
| CSS | <i>Cascading Style Sheet</i> |
| IDE | <i>Integrated Development Environment</i> |
| HTML | <i>HyperText Markup Language</i> |
| MVC | <i>Model-View-Controller</i> |
| POO | Programação Orientada a Objetos |
| RIA | <i>Rich Internet Applications</i> |
| SPA | <i>Single Page Application</i> |

SUMÁRIO

| | |
|---|-----------|
| 1 INTRODUÇÃO..... | 10 |
| 1.1 CONSIDERAÇÕES INICIAIS | 10 |
| 1.2 OBJETIVOS | 11 |
| 1.2.1 Objetivo Geral..... | 11 |
| 1.2.2 Objetivos Específicos..... | 11 |
| 1.3 JUSTIFICATIVA | 11 |
| 1.4 ESTRUTURA DO TRABALHO | 12 |
| 2 RICH INTERNET APPLICATIONS | 13 |
| 3 MATERIAIS E MÉTODO | 15 |
| 3.1 MATERIAIS..... | 15 |
| 3.2 MÉTODO | 17 |
| 4 RESULTADO | 18 |
| 4.1 ESCOPO DO SISTEMA..... | 18 |
| 4.2 MODELAGEM DO SISTEMA..... | 19 |
| 4.3 APRESENTAÇÃO DO SISTEMA | 24 |
| 4.4 DESENVOLVIMENTO DO SISTEMA | 36 |
| 4.4.1 API <i>Back-end</i> Java utilizando Spring..... | 36 |
| 4.4.2 Aplicação <i>Front-end</i> em Angular | 46 |
| 5 CONCLUSÃO..... | 62 |
| REFERÊNCIAS..... | 63 |

1 INTRODUÇÃO

Este capítulo apresenta as considerações iniciais, os objetivos e a justificativa de realização deste trabalho. Em seguida, são apresentados os capítulos subsequentes que compõem o texto.

1.1 CONSIDERAÇÕES INICIAIS

Um plano de carreira define as diretrizes de valorização profissional objetivando alcançar os cargos e as funções de acordo com requisitos predefinidos organizações públicas e privadas. Esse plano colabora para a valorização dos profissionais da organização considerando a atuação profissional, o conhecimento, as capacidades e os interesses do funcionário e as perspectivas e as diretrizes da organização (INTELECTUS, 2017).

O plano de carreira é considerado estratégico para a empresa, visto que possibilita uma definição do potencial de cada pessoa e da equipe e, assim, ajuda a empresa a programar o tempo necessário para que os objetivos e as metas definidos no planejamento da organização sejam alcançados (TELMO, 2015).

Para a empresa, o plano de carreira é um instrumento gerencial que visa estabelecer as trajetórias profissionais possíveis de serem trilhadas dentro da organização, construindo perspectivas de desenvolvimento e de ascensão, conciliando as expectativas de carreira das pessoas com as necessidades organizacionais (INTELECTUS, 2017).

A Intellectus (2017) ressalta como benefícios de um plano de carreira:

- a) intensificar o relacionamento da empresa para com os seus funcionários;
- b) facilitar a tomada de decisão sobre o remanejamento de pessoas na organização, tornando a seleção interna de pessoal mais eficiente;
- c) auxiliar no desenvolvimento das pessoas e da organização;
- d) assegurar a transparência dos requisitos para ascensão na carreira.

Considerando a importância do plano de carreira para os funcionários e para a organização, verificou-se a necessidade de um sistema que facilitasse o registro dos dados dos funcionários para as progressões definidas neste plano e que pudesse auxiliar os gestores na tomada de decisão se o funcionário progredirá de acordo com os critérios estabelecidos. É nesse contexto que se insere a proposta deste trabalho que é um aplicativo *web* para que os funcionários possam se autoavaliar verificando o atendimento aos critérios para mudanças de

cargos e funções e os gestores possam utilizar o aplicativo como base para determinar se o funcionário será promovido.

1.2 OBJETIVOS

A seguir estão o objetivo geral e os objetivos específicos deste trabalho.

1.2.1 Objetivo Geral

Desenvolver um aplicativo *web* para a gestão de planos de carreira.

1.2.2 Objetivos Específicos

- Facilitar o processo de avaliação de funcionários que é realizado pelas organizações visando promoções de acordo com um plano de carreira;
- Fornecer dados que visam auxiliar os gestores no processo de tomada de decisão quando de avaliação de promoções de cargos nas empresas;
- Permitir que o próprio funcionário possa realizar autoavaliação visando progressões e a oportunidade de exercer outras funções dentro da empresa.

1.3 JUSTIFICATIVA

A gestão do plano de carreira é importante para a empresa que define critérios de avaliação e os tornam públicos para o progresso dos funcionários, em termos de novos cargos e funções e de remuneração, dentro da empresa. Essa gestão é importante para o próprio funcionário para que ele possa traçar as suas perspectivas profissionais na empresa que está. Além disso, ele pode gerir a forma de alcançar os requisitos para cargos e funções desejados.

É comum que as empresas que possuem um plano de cargos e salários dependem de sistemas distintos para obter as informações necessárias nas avaliações de progressão na carreira. Por exemplo, um sistema para cadastrar as formações e as habilidades de cada

funcionário, outro para gerir os cargos do plano e um software para cadastros dos funcionários, além de envio de *e-mails* e reuniões para realizar as avaliações em cada etapa da hierarquia (gerente, supervisor, recursos humanos, financeiro, etc.) e de *e-mail* com a devolutiva da avaliação.

Verificou-se, assim, a oportunidade de desenvolver um aplicativo *web* que permita o cadastro de cargos contendo os requisitos desejados para cada cargo, a sua hierarquia (*trainee*, júnior, pleno, sênior e outros), cadastro de usuários com as formações e as habilidades, a indicação do funcionário que será o supervisor que fará a avaliação, entre outros dados.

Um aplicativo *web* que permita a gestão de planos de carreiras visa facilitar o acesso e o controle tanto dos supervisores dentro da organização e dos responsáveis pela avaliação das promoções, como o controle do próprio funcionário, das atividades já realizadas e que pretende realizar para obter as promoções desejadas.

1.4 ESTRUTURA DO TRABALHO

Este texto está organizado em capítulos. O Capítulo 2 apresenta o referencial teórico que fundamenta o tipo de aplicação desenvolvido que é uma aplicação *web* caracterizada como rica. Neste capítulo são apresentados alguns conceitos essenciais e a caracterização desse tipo de aplicação. No Capítulo 3 são apresentados os materiais que são as ferramentas e as tecnologias (como ambiente de desenvolvimento material para padronização de interface e desenvolvimento, linguagem de programação e banco de dados) e o método (as principais atividades desenvolvidas para alcançar os objetivos propostos). O resultado obtido com a realização do trabalho (o aplicativo desenvolvido) é apresentado no Capítulo 4. Por fim estão as considerações finais, seguidas das referências bibliográficas.

2 RICH INTERNET APPLICATIONS

Nos últimos anos, com o desenvolvimento das tecnologias e das ferramentas de desenvolvimento e distribuição de aplicações, as aplicações *web* estão transferindo a semântica da sua interface de hipertexto tradicional para a semântica das aplicações *desktop* (PANG et al., 2010). Aplicações *desktop* apresentam vantagens em termos de interface com o usuário, pelos recursos de interatividade, e as aplicações *web* possuem características de desenvolvimento rápido e de baixo custo de distribuição. É vantagem utilizar a Internet ao invés de necessitar de uma rede proprietária, como ocorre com as aplicações *desktop* quando são utilizadas em rede.

Em decorrência das vantagens apresentadas pelas aplicações *desktop* e das vantagens das aplicações, as *Rich Internet Applications* (RIAs), por combinarem as vantagens de ambas, têm se tornado uma nova geração de aplicações *web*. As RIAs combinam áudio, vídeo e tecnologias de comunicação com diálogo em tempo real. Essas tecnologias auxiliam a melhorar a experiência do usuário no uso de aplicações *web*.

Agustín (2015) destaca que a rápida evolução da *web* e a emergência de novas tecnologias como Ajax (CZARNECKI et al., 2005) e interfaces caracterizadas como ricas (MELIÁ, 2008) tem fortemente suportado por novos tipos de aplicações para a Internet.

Diferentemente do desenvolvimento de aplicações *web* tradicionais que consideram *tags HyperText Markup Language* (HTML) e *scripts* de código como elementos básicos, uma nova geração de aplicações para a *web* integram essas novas tecnologias para alcançar um alto grau de interatividade (AGUSTÍN, 2015).

O termo RIA refere-se a uma família heterogênea de soluções, caracterizada por um objetivo comum de adicionar novas funcionalidades ao tipo convencional ou tradicional de aplicações *web* que são as baseadas em hipertexto (FRATERNALI; ROSSI; SÁNCHEZ-FIGUEROA, 2010). Para esses autores, as RIAs combinam a arquitetura de distribuição da *web* com a interatividade de interface e capacidade computacional das aplicações *desktop* e a combinação resultante melhora todos os elementos de uma aplicação *web* (dados, lógica de negócio, comunicação e apresentação).

As RIAs estão oferecendo maior responsividade às aplicações e melhorando a experiência do usuário em relação às aplicações *web* tradicionais (MELIÁ et al., 2010). Tradicionais são consideradas as aplicações *web* baseadas em HTML e formulários simples, com o processamento realizado todo no servidor.

As RIAs são aplicações cliente/servidor que representam a convergência de duas culturas de desenvolvimento de aplicações concorrentes: *web* e *desktop* (MELIÁ et al., 2010). Esse tipo de aplicação proporciona os principais benefícios de distribuição e manutenibilidade das aplicações *web*, ao mesmo tempo em que oferecem uma interface com o usuário muito mais rica. O aspecto de riqueza dessas aplicações está relacionado aos recursos de interface que elas possuem que se assemelham às aplicações *desktop*, oferecendo maior interatividade pela comunicação assíncrona entre cliente e servidor e pela possibilidade de processamento no cliente e funcionalidades como os de arrastar e soltar elementos na tela, botões e menus diferenciados, entre outros. Esses elementos e efeitos são bastante distintos dos existentes nas aplicações *web* tradicionais, construídas com HTML em versões anteriores a 4 e sem o uso de complementos como *Cascading Style Sheet* (CSS) e Ajax, por exemplo.

Além dos aspectos de interface, sempre citados quando se referindo às aplicações RIA que inseriram uma nova característica arquitetural no campo das aplicações *web* tradicionais, por exemplo, a interface do usuário com estado conectado e desconectado, uma comunicação cliente/servidor inteligente com requisições assíncronas (MELIÁ et al., 2010).

3 MATERIAIS E MÉTODO

Este capítulo apresenta os materiais e o método para o desenvolvimento do trabalho.

3.1 MATERIAIS

No Quadro 1 as ferramentas e as tecnologias utilizadas para realizar a análise e o desenvolvimento da aplicação *web* desenvolvida como resultado da realização deste trabalho.

| Ferramenta / Tecnologia | Versão | Referência (site) | Finalidade |
|---|--------|---|--|
| Angular | 5.1 | https://angularjs.org/ | <i>Framework front-end.</i> |
| Angular Material | 5.1 | https://material.angularjs.org/latest/ | Estilização das páginas. |
| Angular Cli | 1.0 | https://cli.angular.io/ | Ferramenta que auxilia no desenvolvimento em Angular. |
| HTML5 | 5.0 | https://www.w3.org/TR/html5/ | Linguagem para criação da estrutura das páginas do site. |
| Visual Studio Code | 1.17.2 | https://code.visualstudio.com/ | <i>Integrated Development Environment (IDE)</i> de desenvolvimento utilizada para o <i>front-end</i> . |
| MySQL | 5.7 | https://www.mysql.com/ | Banco de dados. |
| MySQL Workbench | 6.3 | https://www.mysql.com/ | Ferramenta para o gerenciamento do banco de dados. |
| Java | 1.8 | https://www.java.com/pt_BR/ | Linguagem de programação para desenvolvimento do aplicativo. |
| Spring <i>Model-View-Controller</i> (MVC) | 5.0 | https://spring.io/ | <i>Framework back-end</i> utilizado para desenvolver a <i>Application Programming Interface</i> (API) do aplicativo. |
| Eclipse | 4.7.0 | http://www.eclipse.org/ | IDE de desenvolvimento da API <i>back-end</i> . |
| JSON | | https://www.json.org/ | Formato utilizado para a comunicação entre a API e a aplicação. |
| Maven | 3.5.2 | https://maven.apache.org/ | Ferramenta utilizada para gerenciar as dependências do projeto da API. |

Quadro 1 - Ferramentas e tecnologias utilizadas

Para a criação do *front-end* do projeto foi utilizado o *framework* Angular na versão 5.0.1. Inicialmente chamado de AngularJS na sua primeira versão, após a versão 2.0.0 passou

a ser chamado somente de Angular, para estilização os componentes da aplicação se fez uso do Angular Material que disponibiliza uma biblioteca de componentes feitos em Angular seguindo o padrão do Material Design, que foram importados para a aplicação. O Angular possibilita ampliar o vocabulário HTML utilizado na aplicação (W3SCHOOLS, 2017).

O Angular é um *framework* JavaScript criado pelo Google, que pode ser usado para criação de aplicações *web*, *mobile* e *desktop*, segue o modelo de *Single Page Application* (SPA) centralizando a aplicação em uma única página, executando a aplicação no diretamente navegador. Uma vez carregada a página são buscados os dados do servidor sem a necessidade de atualizar a página inteira para exibir uma nova informação, atualizando somente a parte da página referente à informação sendo atualizada. Tendo a sua estrutura é baseada em diretivas, componentes, serviços e rotas.

Para o desenvolvimento em Angular foi utilizada a linguagem TypeScript, criada pela Microsoft, a feita para atender as deficiências do JavaScript, provendo recursos como a Programação Orientada a Objetos (POO) e a tipagem de variáveis. Uma vez escrito um código em TypeScript ele é compilado em uma arquivo JavaScript, que pode ser executado pelo navegador.

O Angular Material é uma biblioteca de diretivas, componentes desenvolvidos em Angular, voltada para a construção de aplicações que seguem o conceito de *design* do Material Design, criado pela Google.

Angular Material é um *framework* de componentes de interface com o usuário e uma referência de implementação para especificação *design* de interface da Google Material. Esse *framework* provê um conjunto de componentes amplamente testados, reusáveis e acessíveis para interface com o usuário que são desenvolvidos tendo como base o Material Design (ANGULAR MATERIAL, 2017).

Material design é uma linguagem visual para sintetizar os princípios clássicos de um bom *design* de interface considerando as inovações e as possibilidades da tecnologia e da ciência (MATERIAL DESIGN, 2017).

O Spring MVC é um *framework* de desenvolvimento *web*, que possui as principais funcionalidades utilizadas como gerenciar requisições HTTP, persistência de dados, injeção de dependências e controle de acesso. Essas funcionalidades podem ser configuradas para melhor atender as necessidades de cada aplicação.

O Spring é um *framework* Java de código aberto utilizado para facilitar o desenvolvimento de aplicações *web*, provendo diversos módulos para atender diferentes

aspectos da aplicação como: mapeamento de requisições, controle de acesso, acesso ao banco de dados e testes unitários, entre outros.

O Spring Framework é dividido em módulos. As aplicações podem escolher os módulos necessários. Os módulos centrais são os módulos do recipiente do núcleo, incluindo um modelo de configuração e um mecanismo de injeção de dependência. Além disso, o Spring Framework fornece suporte para diferentes arquiteturas de aplicativos, incluindo mensagens, dados transacionais e persistência. Além disso, há a estrutura *web* Spring MVC baseada em *servlet* e, em paralelo, a estrutura *web* reativa da WebFlux Spring (SPRING, 2017).

3.2 MÉTODO

A seguir são apresentadas as principais atividades realizadas para as etapas desenvolvidas para realizar o projeto.

Levantamento de requisitos

O levantamento de requisitos da aplicação foi realizado tendo com base problemas verificados durante a avaliação do plano de carreira realizada na empresa que autor deste trabalho trabalhava.

Análise e projeto

Os requisitos definidos para o aplicativo foram organizados em funcionais e não funcionais, servindo de base para a definição dos dados a serem armazenados e a elaboração dos formulários de cadastro e as consultas.

Desenvolvimento

O desenvolvimento foi realizado utilizando os materiais apresentados no Quadro 1.

4 RESULTADO

O resultado obtido com a realização deste trabalho, representado pelos objetivos, é apresentado neste capítulo. O escopo fornece uma visão geral da funcionalidade de negócio da aplicação. A modelagem apresenta o que foi desenvolvido para representar o problema (as funcionalidades desejadas para a aplicação) e a solução proposta. A apresentação representa as funcionalidades do aplicativo por meio de imagens das suas telas. E o desenvolvimento traz partes de código consideradas relevantes para implementar as funcionalidades do sistema e para exemplificar os recursos das tecnologias empregadas no desenvolvimento da aplicação.

4.1 ESCOPO DO SISTEMA

A aplicação *web*, desenvolvida neste trabalho, permite a gestão do plano de carreira das empresas, possibilitando facilitar o gerenciamento do processo de avaliação dos funcionários, permitindo o cadastro de hierarquia e de ciclos (etapas) do processo de avaliação, bem como, o registro dos resultados das avaliações realizadas. A aplicação conta com cadastro de cargos registrando as funções exercidas e os requisitos para exercer o referido cargo e cadastro de usuários. Esse cadastro possui um painel no qual são registradas as habilidades de cada funcionário e quais requisitos do cargo desejado já foram atingidos. Esses dados possibilitam o acompanhamento pelos supervisores que realizam a avaliação do plano.

O aplicativo possibilita a aplicação de um processo de autoavaliação por meio de um formulário gerado a partir dos dados do cargo desejado. Em seguida, os supervisores fazem suas avaliações com base na autoavaliação do funcionário. Evidenciando, assim, os pontos de divergência entre as diferentes visões, mais especificamente do funcionário que será avaliado e do supervisor que fará a avaliação. Em seguida, o formulário com a avaliação do supervisor segue para a avaliação dos demais usuários definidos no ciclo de avaliação. Gerando, ao final do processo, uma devolutiva para o funcionário, informando-o dos pontos positivos e que podem ser melhorados e se houve a promoção de cargo almejada.

4.2 MODELAGEM DO SISTEMA

O funcionamento do sistema possui três atores denominados deadministrador, funcionários e supervisores que serão os usuários que irão efetivar as avaliações, tendo como principais funções:

- Administrador – esse ator tem acesso aos módulos de cadastro do sistema, que provê as informações que serão utilizadas pelos usuários nas avaliações, assim como o cadastro dos usuários.
- Funcionário – é ator principal do sistema e é um funcionário da organização que pode solicitar ao seu supervisor uma avaliação, para qualquer cargo desejado.
- Supervisor – um usuário com privilégios para efetuar avaliações, finalizando-as ou enviando-as para outro supervisor.

Na Figura 1 é apresentado o diagrama de casos de uso do sistema. Nessa Figura estão os três atores definidos para o sistema e as funcionalidades atribuídas a cada um deles. Essas funcionalidades são representadas no sistema por permissões de acesso.

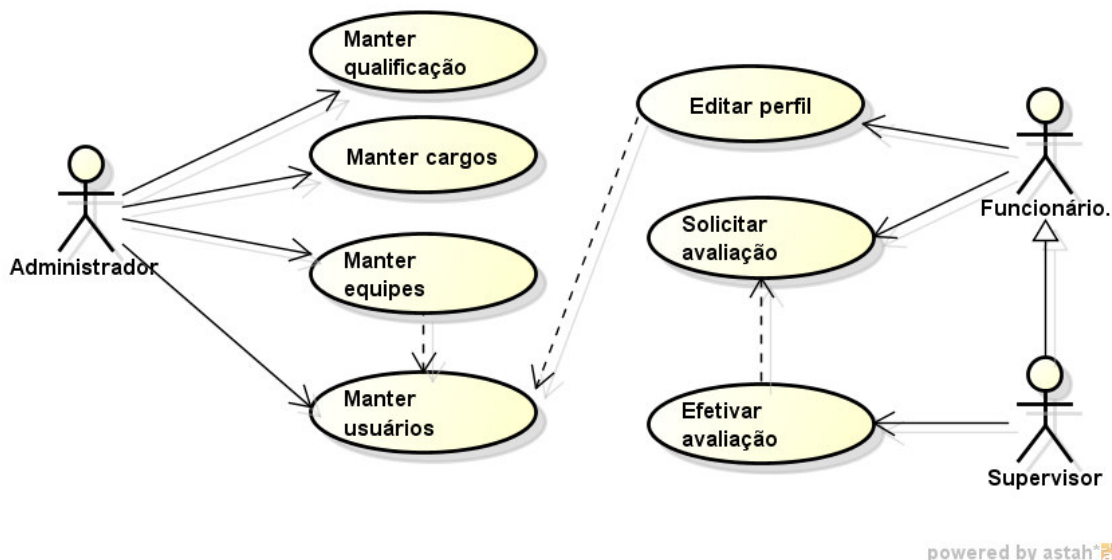


Figura 1 - Diagrama de Casos de Uso

No Quadro 2 são apresentados os requisitos relacionados com os casos de uso que compõem cada funcionalidade do sistema.

| Nro. | Requisito | Objetivo | Casos de uso utilizados |
|------|-------------------------|---|--|
| 1 | Registrar usuários | Permitir a inclusão de novos usuários, bem como a alteração de dados de usuários cadastrados. | Manter cargos. Manter qualificação. |
| 2 | Registrar qualificações | Possibilitar o ajuste e a inclusão de qualificações no sistema, de forma | Manter qualificação. |

| | | | |
|---|----------------------|--|---|
| | | que elas fiquem disponíveis para serem relacionadas com cargos e usuários. | |
| 3 | Registrar cargos | Viabilizar o cadastro de cargos que serão relacionados às qualificações que o referido cargo necessita. | Manter qualificações. |
| 4 | Montar equipes | Disponibilizar um mecanismo para o gerenciamento e a inclusão de equipes, indicando qual será o supervisor da equipe. | Manter usuários. |
| 5 | Gerenciar avaliações | Permitir que o usuário faça a solicitação de uma nova avaliação, bem como disponibilizar o histórico de avaliações do usuário. | Solicitar avaliações. Efetivar avaliação. Manter qualificações. Manter cargos. |
| 6 | Efetivar avaliações | Permitir o ajuste de uma avaliação, resultando no envio para outro supervisor ou a finalização da avaliação. | Manter usuários. Manter qualificações. Manter cargos. |
| 7 | Editar perfil | Possibilitar que o usuário edite as informações básicas do seu cadastro e informe quais as qualificações que ele possui. | Manter qualificações. Manter cargos. |

Quadro 2 - Casos de Uso por requisito

Nos Quadros de 3 a 8 são detalhadas algumas funções básicas do sistema, como inclusão, consulta, alteração de registros, além das funções específicas do processo de avaliação. O Quadro 3 apresenta a expansão da operação de cadastro (inclusão).

| | |
|--|--|
| <p>1. Identificador do requisito: Cadastro. Descrição: Caso de uso que permite ao ator efetuar cadastros. Evento Iniciador: Todas as funcionalidades que visam prover informações para o sistema. Atores: Administrador do sistema. Pré-condição: Apenas o usuário administrador tem acesso a esse tipo de funcionalidade. Fluxo Básico: 1 – Na consulta de dados o Administrador opta por incluir um novo registro. 2 – O Administrador preencher os campos com as informações solicitadas. 3 – Clica na opção de salvar. 4 – O sistema faz a validação dos dados informados. 5 – Após gravar as informações no banco de dados em caso de sucesso retorna para tela de listagem apresentando uma mensagem para o Administrador. Pós-Condição: As informações fornecidas pelo Administrador devem ser validadas.</p> | |
| Fluxo Alternativo (Extensão) | Descrição |
| Tipo de dados incompatíveis. | Caso algum dado informado seja incompatível como esperado para o campo deve ser retornada uma mensagem de erro solicitando a correção do problema. |

Quadro 3 - Caso de uso cadastrar

No Quadro 4 está a expansão da operação de consulta do sistema.

| |
|---|
| <p>2. Identificador do requisito: Consulta. Descrição: Caso de uso que permite que o ator efetue a consulta das informações cadastradas. Evento Iniciador: Todas as funcionalidades que buscam informações da base de dados.</p> |
|---|

| | | |
|--|--------------------|--|
| <p>Atores: Administrador do sistema e Funcionário. Pré-condição: Algumas opções somente são apresentadas de acordo com o tipo de usuário. Fluxo Básico: 1 – Escolhe a opção no menu. 2 – O sistema apresenta uma listagem com os itens cadastrados. 3 – Usuário seleciona o item desejado. 4 – O sistema redireciona para o detalhamento do item selecionado. Pós-Condição: Somente são apresentadas as informações pertinentes ao usuário autenticado.</p> | | |
| Requisitos não funcionais: | | |
| Identificador | Nome | Descrição |
| RNF 2.1 | Filtro por usuário | Somente ficam disponíveis informações referentes ao usuário autenticado. |

Quadro 4 - Caso de uso consultar

O Quadro 5 apresenta a funcionalidade de alteração dos dados do sistema.

| | |
|---|--|
| <p>3. Identificador do requisito: Alteração. Descrição: Caso de uso que permite que o autor efetue a alteração das informações cadastradas. Evento Iniciador: Todas as funcionalidades que visam prover informações para o sistema. Atores: Administrador do sistema. Pré-condição: Apenas o usuário administrador tem acesso à opção de alteração. Fluxo Básico: 1 – Acessar a consulta do item desejado. 2 – Na listagem clicar no item desejado. 3 – O sistema redireciona para os detalhes do item, com as informações atuais. 4 – Editar as informações desejadas. 5 – Clicar em salvar. 6 – O sistema faz a validação dos dados informados. 7 - Após gravar as informações no banco de dados em caso de sucesso retorna para tela de listagem apresentando uma mensagem para o Administrador. Pós-Condição: As informações devem estar de acordo com o esperado pelo sistema.</p> | |
| Fluxo Alternativo (Extensão) | Descrição |
| Tipo de dados incompatíveis | Caso algum dado informado seja incompatível como esperado para o campo deve ser retornada uma mensagem de erro solicitando a correção do problema. |

Quadro 5 - Caso de uso alterar

As funções de solicitação de avaliação e efetivação da avaliação são descritas nos Quadros 6 e 7.

| |
|--|
| <p>4. Identificador do requisito: Solicitar Avaliação. Descrição: Caso de uso permite que o ator solicite uma nova avaliação ao seu supervisor. Evento Iniciador: A funcionalidade de solicitar nova avaliação. Atores: Funcionário e Supervisor. Pré-condição: O usuário autenticado deve ter pelo menos uma equipe definida. Fluxo Básico: 1 – Na opção de avaliações clicar em adicionar. 2 – É solicitado o cargo para qual a avaliação será feita. 3 – São apresentadas as informações do usuário e as qualificações atendidas para o cargo. 4 – Solicita os dados de pontos positivos, aspectos que deve melhorar e a conclusão para a avaliação. 5 – Informar o supervisor que realizará a avaliação.</p> |
|--|

| 6 – O sistema faz a validação dos dados informados. | | |
|--|-------------------|--|
| 7 – Registra a avaliação e encaminha um <i>e-mail</i> para o supervisor, retornando uma mensagem de sucesso para o usuário. | | |
| Pós-Condição: A avaliação fica com <i>status</i> em pendente e é apresentada na listagem para o supervisor relacionado. | | |
| Fluxo Alternativo (Extensão) | | Descrição |
| Falha na validação dos dados informados | | Em caso de falha é apresentada uma mensagem indicando o problema e o sistema permanece na tela de solicitação de avaliação. |
| Requisitos não funcionais: | | |
| Identificador | Nome | Descrição |
| RNF 4.1 | Filtro por equipe | Somente podem ser relacionados como supervisores usuários cadastrados na equipe do usuário autenticado e marcados como supervisor. |

Quadro 6 - Caso de uso solicitar avaliação

| 5. Identificador do requisito: Efetuar Avaliação. | | |
|---|-----------------------|--|
| Descrição: Caso de uso permite que o ator finalize ou encaminhe uma avaliação. | | |
| Evento Iniciador: A funcionalidade de efetuar avaliação. | | |
| Atores: Supervisor. | | |
| Pré-condição: O Supervisor autenticado deve ter sido marcado como supervisor na equipe do funcionário que solicitou a avaliação. | | |
| Fluxo Básico: | | |
| 1 – Na listagem de avaliações clicar na avaliação desejada. | | |
| 2 – São apresentados os dados informados no momento da abertura da avaliação. | | |
| 3 – Os dados podem ser editados com exceção do cargo solicitado. | | |
| 4 – Definir o <i>status</i> da avaliação com a opção de pendente, aprovada ou fechada. | | |
| 5 – Após finalizar a edição a avaliação é gravada. | | |
| 6 – O sistema faz a validação dos dados informados. | | |
| 7 – O resultado é enviado para o <i>e-mail</i> do solicitante e a avaliação fica com o <i>status</i> definido. | | |
| Pós-Condição: A avaliação fica disponível para a consulta do solicitante com <i>status</i> definido. | | |
| Fluxo Alternativo (Extensão) | | Descrição |
| Falha na validação dos dados informados | | Em caso de falha é apresentada uma mensagem indicando o problema e o sistema permanece na tela de solicitação de avaliação. |
| Encaminhamento da avaliação | | Em vez de finalizar avaliação, o supervisor pode encaminhá-la o outro supervisor, alterando o supervisor da avaliação e enviando um <i>e-mail</i> para o solicitante informando que a avaliação foi repassada. |
| Requisitos não funcionais: | | |
| Identificador | Nome | Descrição |
| RNF 5.1 | Filtro por supervisor | Para encaminhar uma avaliação o supervisor deve estar incluído em uma equipe com outros supervisores. |

Quadro 7 - Caso de uso efetivar avaliação

No Quadro 8 está a funcionalidade de montagem de equipes. Essa funcionalidade tem como objetivo indicar quem tem permissão de efetuar avaliações.

| | | |
|---|----------------------------|---|
| <p>6. Identificador do requisito: Cadastrar equipe.</p> <p>Descrição: Caso de uso que permite que ao Administrador do sistema relacionar um grupo de usuários (funcionário e supervisor) em uma equipe.</p> <p>Evento Iniciador: A funcionalidade de incluir equipe.</p> <p>Atores: Administrador do sistema.</p> <p>Pré-condição: O usuário autenticado deve ser o administrador do sistema.</p> <p>Fluxo Básico:</p> <p>1 – Na listagem de equipe utiliza a opção adicionar.</p> <p>2 – São apresentados todos os usuários cadastrados no sistema.</p> <p>3 – Informar o nome da equipe e selecionar os usuários comporão a equipe.</p> <p>4 – Clicar em próximo.</p> <p>5 – São listados os usuários selecionados com a opção de indicar os supervisores da equipe.</p> <p>6 – Ao salvar o sistema retornar para a listagem apresentando uma mensagem de sucesso ao usuário.</p> <p>Pós-Condição: A equipe é apresentada na listagem e os integrantes ficam habilitados a solicitar avaliações para o supervisor.</p> | | |
| Fluxo Alternativo (Extensão) | | Descrição |
| Falha na validação dos dados informados | | Em caso de falha é apresentada uma mensagem indicando o problema e o sistema permanece na tela de solicitação de avaliação. |
| Requisitos não funcionais: | | |
| Identificador | Nome | Descrição |
| RNF 6.1 | Sem limite de equipes | Um mesmo usuário pode ser relacionado em várias equipes. |
| RNF 6.2 | Sem limite de supervisores | Uma equipe não possui limite de supervisores. |

Quadro 8 - Caso de uso cadastrar equipe

A Figura 2 apresenta o diagrama com as entidades e os respectivos relacionamentos entre as tabelas que compõem o banco de dados.

Um usuário pode ser inativado de acesso ao sistema. Um campo da tabela de usuários permite identificar se ele está ativo ou inativo. A tabela de avaliação armazena dados de avaliação de cada funcionário. Na avaliação, o usuário indica o cargo pretendido. A esse cargo estão associadas as qualificações necessárias.

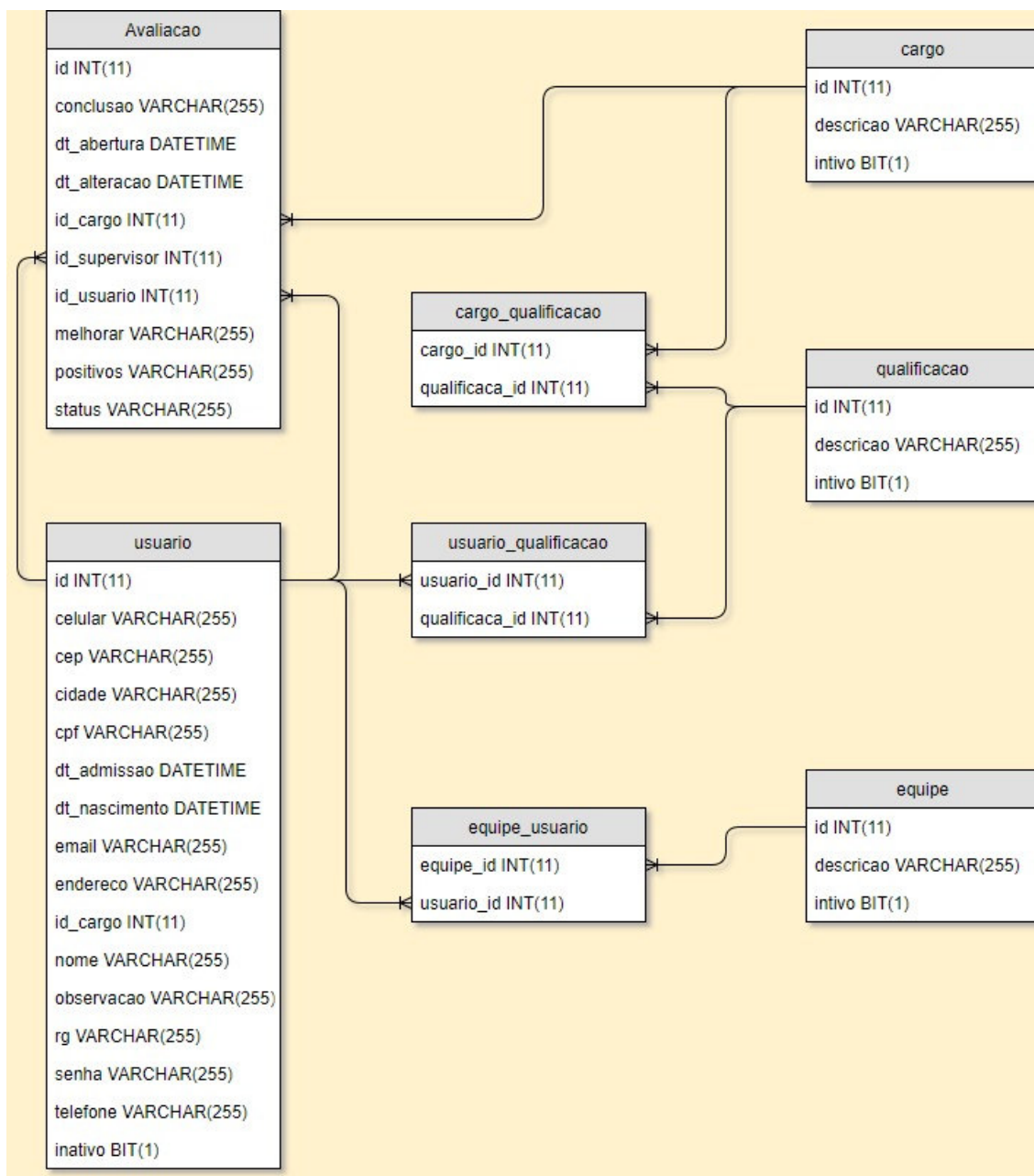


Figura 2 - Diagrama de entidades e relacionamentos do banco de dados

4.3 APRESENTAÇÃO DO SISTEMA

O administrador do sistema tem a função de prover e gerenciar as informações utilizadas no sistema. São informações organizacionais da empresa como cargos e equipes e as informações que serão utilizadas para a avaliação como qualificações e demais usuários do sistema.

A Figura 3 apresenta a tela de listagem de usuários. Todos os cadastros do sistema seguem esse mesmo modelo: são listadas todas as informações cadastradas com as opções de clicar no item para editar ou em adicionar para cadastrar um novo item.

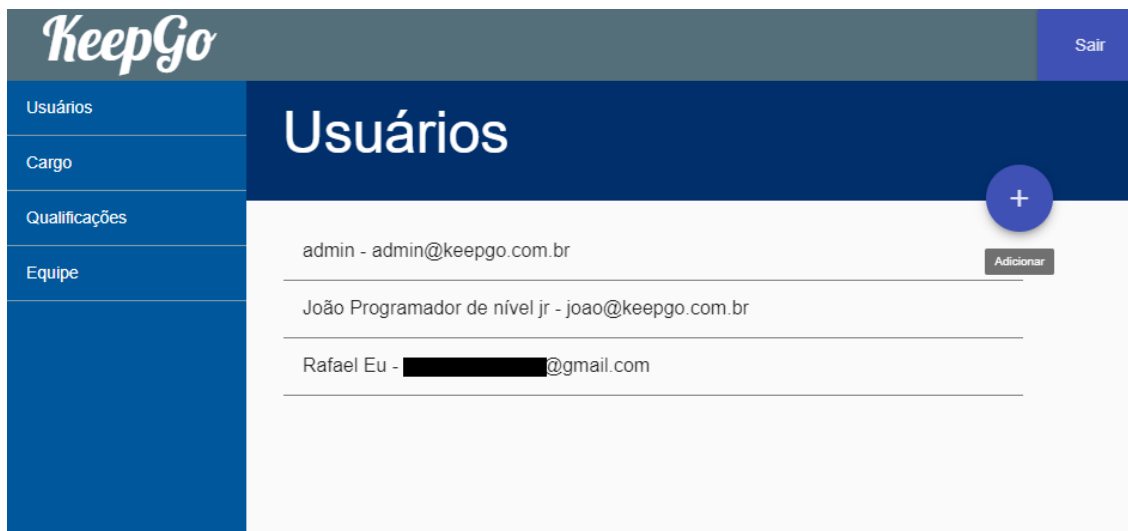


Figura 3 - Listagem de usuários

Clicando em um item da listagem é acessada a tela de edição. A Figura 4 apresenta a tela de usuários, como exemplo de tela de cadastro. Por meio dessa tela o administrador pode editar algumas informações do usuário..

Figura 4 - Cadastro de usuário

As telas de cadastro são utilizadas tanto para a inclusão quanto para a edição das informações, e as informações são gravadas pelo botão salvar.

O resultado da operação é apresentado ao usuário por meio de mensagens na parte inferior da tela. A Figura 5 exemplifica uma mensagem de resposta ao usuário que indica que a operação foi realizada com sucesso. Após a inclusão das informações, o sistema redireciona para a tela de listagem do item que foi cadastrado.

Figura 5 - Mensagem sucesso

Na Figura 6 está um exemplo de mensagem de erro. No caso de falta de informação ou inconsistência dos dados, o sistema apresenta uma mensagem de erro e destaca a informação a ser corrigida.

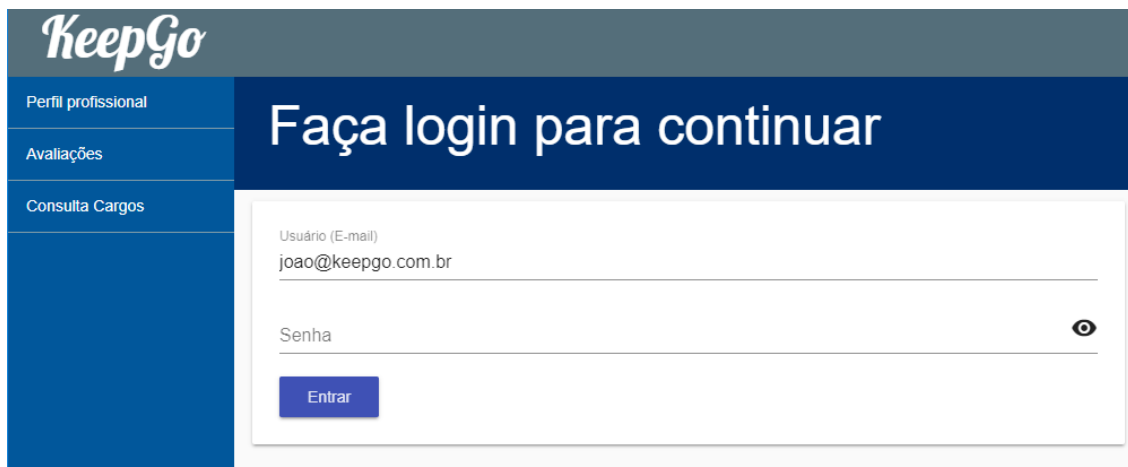
The screenshot shows the 'Cadastro de Usuário' (User Registration) form in the KeepGo system. The form is divided into several sections: 'Perfil profissional', 'Usuários', 'Cargos', 'Qualificações', and 'Equipes'. The 'E-mail' field is highlighted in red, indicating an error. The 'Senha' (Password) field is highlighted in yellow. The form contains the following data:

| | | | |
|--------------------|------------------------------|------------------|------------|
| Nome | João Programador da Silva | | |
| CPF | 012.345.678-90 | RG | 0123456789 |
| Data de Nascimento | 1/31/2000 | Data de Admissão | 11/1/2017 |
| Endereço | Rua dos programadores | | |
| Número | 1022 | Bairro | Centro |
| Cidade | Pato Branco | Estado | Paraná |
| CEP | 85660-000 | Telefone | |
| Celular | (46)99999-8888 | | |
| Observação | 0 / 256 | | |
| Cargo | Programador Java Back-end Jr | | |

A red dashed oval highlights a black error message box at the bottom of the form that says "Informe todos os campos solicitados" (Inform all requested fields) with an information icon.

Figura 6 - Mensagem falha

Após todos os dados serem cadastrados pelo administrador, o usuário com perfil de funcionário pode efetuar o *login* no sistema, por meio da tela apresentada na Figura 7. Sem autenticar-se, o usuário não pode acessar nenhuma tela da aplicação, sendo redirecionado para a tela de *login*.



KeepGo


Perfil profissional

Avaliações

Consulta Cargos

Faça login para continuar

Usuário (E-mail)
joao@keepgo.com.br

Senha 

Entrar

Figura 7 - Login do sistema

Após autenticar-se no sistema, o funcionário é direcionado para a tela do perfil. Nessa tela, ele pode informar seus dados para contato como *e-mail* e telefones. Além dos dados pessoais, ele pode relacionar as qualificações que possui, sendo apresentadas as qualificações do usuário e as do cargo desejado. As qualificações do cargo que ainda não foram alcançadas são apresentadas com os campos desmarcados. Assim, quando determinada qualificação for obtida, o próprio usuário pode adicionar as suas qualificações, que posteriormente são utilizadas na avaliação. A tela de perfil é apresentada na Figura 8.

KeepGo Sair

Perfil profissional

Avaliações

Consulta Cargos

Perfil

Nome
Rafael Lima

CPF 01234567890 RG 0123456789

Data de Nascimento 1/14/1992 Data de Admissão 11/3/2017

Endereço
Rua da minha casa, 1000, Meu Bairro

Cidade Dois Vizinhos CEP 8566000

Telefone Celular 4699757227

Observação
Conclui o ensino superior em 2017
Férias marcadas para o mês 12.


64 / 256

Cargo

Cargo atual
Programador Java Back-end Jr

Qualificações

- Conhecimento em JAVA
- Ensino superior incompleto
- Noções de JavaScript
- Noções de Spring
- Noções de Ruby
- Noções de Angular

Adicionar qualificação 

Salvar

Figura 8 - Perfil do funcionário

Com as qualificações adicionadas, o funcionário pode solicitar uma avaliação. Na opção avaliações são apresentadas todas as avaliações relacionadas ao funcionário autenticado, tanto as suas quanto as enviadas para ele. Como pode ser visto na Figura 9, na listagem de avaliações também há a opção de adicionar uma nova avaliação pelo botão com o símbolo “+”.



| No. | Solicitante | Supervisor | Status | Abertura | Ações |
|-----|---------------------------------------|------------------------------------|----------|------------|---------|
| 1 | João Programador da Silva Rafael Lima | | Pendente | 01/01/2017 | Editar |
| 12 | Rafael Lima | João Programador da Silva Pendente | | 12/02/2017 | Avaliar |
| 13 | Rafael Lima | João Programador da Silva Pendente | | 12/05/2017 | Avaliar |

Figura 9 - Listagem de avaliações do funcionário

Ao clicar no botão adicionar, o funcionário é direcionado para a tela de cadastro da avaliação mostrada na Figura 10. Inicialmente, é solicitado o cargo para qual a avaliação será realizada, trazendo por padrão o cargo do funcionário. Esse cargo é utilizado posteriormente para verificar as qualificações atendidas pelo funcionário.

KeepGo Sair

Perfil profissional

Avaliações

Consulta Cargos

Avaliação

- 1 Informe o cargo
 - Cargo considerado para a avaliação *
 - Programador Java Back-end Jr
 - Próximo
- 2 Avaliação
- 3 Encaminhar
Optional
- 4 Pronto

Figura 10 - Cadastro de avaliação: parte 1

Em seguida, no próximo passo, são apresentadas as informações do funcionário e as qualificações atendidas para o cargo definido, como mostra a tela do sistema apresentada na Figura 11.

KeepGo Sair

Perfil profissional
Avaliações
Consulta Cargos

Avaliação - nova

- Informe o cargo
- Avaliação

Data de abertura: 12/2/2017 Última atualização: 12/2/2017

Informções do usuário

Nome: Rafael Lima

E-mail: exemploemail@gmail.com

Data de Nascimento: 1/14/1992 Data de Admissão: 11/3/2017

Cargo atual: Programador Java Back-end Jr

Observação: Conclui o ensino superior em 2017

Qualificações

- Conhecimento em JAVA
- Ensino superior incompleto
- Noções de JavaScript
- Noções de Spring
- Noções de Ruby
- Noções de Angular

check - usuário atende a qualificação

Dados da avaliação

Pontos positivos

Bom relacionamento com os colegas
Qualidade no desenvolvimento
Disponibilidade para viajar 90 / 256

Pontos a melhorar

Estudar sobre ruby
Chega atrasado 33 / 256

Conclusão

Como estou concluindo a faculdade gostaria de me candidatar para esse cargo. 76 / 256

Status da avaliação: Pendente

Voltar Encaminhar

- Encaminhar
Optional
- Pronto

Figura 11 - Cadastro de avaliação: parte 2

No passo dois (Figura 11) há duas opções: a) voltar, que retorna para o passo um e b) encaminhar, avança para o próximo passo. A Figura 12 apresenta a tela do próximo passo. Nessa tela o supervisor efetuará a avaliação. Esse passo também possui alguns elementos específicos para o supervisor, apresentados na sequência.

KeepGo

Sair

Perfil profissional

Avaliações

Consulta Cargos

Avaliação - nova

- Informe o cargo
- Avaliação
- Encaminhar
- Pronto

Supervisor que irá efetuar a avaliação *

João Programador

Voltar Gravar

Figura 12 - Cadastro de avaliação: parte 3

Nesse ponto, o funcionário pode salvar a avaliação, que ficará com o *status* de pendente. Após a avaliação ser salva, fica disponível na listagem de avaliações do supervisor informado e são enviados *e-mails* para ambos (supervisor e funcionários) com os dados da avaliação.

Na Figura 13 é apresentada a tela de finalização da avaliação, que informa ao funcionário que a avaliação foi gravada e os *e-mails* foram enviados.

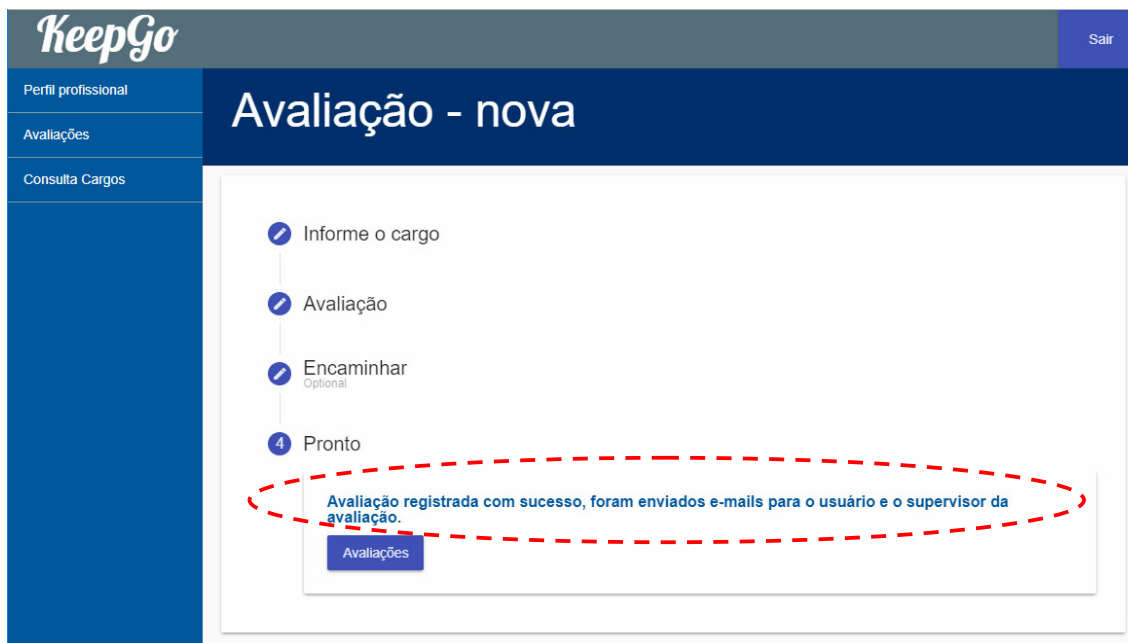



Figura 13 - Cadastro de avaliação: parte 4

Para efetivar a avaliação, o usuário definido como supervisor deve acessá-la, sendo apresentada na mesma tela de cadastro, apenas com algumas opções adicionais no passo dois, como apresentado na Figura 14.


Sair

Perfil profissional
 Avaliações
 Consulta Cargos

Avaliação #12

1

Informe o cargo

2

Avaliação

Data de abertura
12/2/2017
Última atualização
12/2/2017

Informações do usuário

Nome
Rafael Lima

E-mail
exemploemail@gmail.com

Data de Nascimento Data de Admissão
1/14/1992 11/3/2017

Cargo atual
Programador Java Back-end Jr

Observação
Conclui o ensino superior em 2017

Qualificações

- Conhecimento em JAVA
- Ensino superior incompleto
- Noções de JavaScript
- Noções de Spring
- Noções de Ruby
- Noções de Angular

check - usuário atende a qualificação

Dados da avaliação

Pontos positivos
Bom relacionamento interpessoal.
Qualidade no desenvolvimento, testa antes de enviar.
Disponível para viajar. 109 / 256

Pontos a melhorar
Estudar ruby
Não chegar atrasado 32 / 256

Conclusão
Se compromete a solucionar os pontos a melhorar em dois meses.
Esta apto para o cargo. 86 / 256

Status da avaliação
Aprovada

Voltar
Encaminhar
Gravar

3

Encaminhar
Optional

4

Pronto

Figura 14 - Efetuar avaliação pelo supervisor

O processo de efetivação da avaliação ocorre quando o usuário definido como supervisor revisa, juntamente com o funcionário sendo avaliado, os pontos informados pelo funcionário. Os pontos de comum acordo entre ambos são registrados na avaliação e é definindo o *status* como aprovada, indicando que o funcionário está apto para o cargo o qual a avaliação foi submetida e a avaliação é fechada se o funcionário não estiver apto. Uma outra opção é deixar a avaliação como pendente e encaminhar para outro supervisor, de acordo com o processo interno adotado pela empresa.

Da mesma forma que na abertura, ao final do processo de efetivação são enviados *e-mails* para ambos os participantes. Assim, qualquer alteração é informada para todos os envolvidos no processo de avaliação, evitando, assim, que o funcionário e o supervisor possam não ter conhecimento da situação da avaliação.

Com todas as funções apresentadas, o sistema gerencia o processo de avaliação do plano de carreira, centralizando todas as informações necessárias e fornecendo aos supervisores e aos funcionários avaliados uma ferramenta fácil para registrar suas qualificações e solicitar uma avaliação, garantindo transparência e comodidade para o processo.

4.4 DESENVOLVIMENTO DO SISTEMA

O conteúdo desta seção apresenta a estrutura do sistema, que se divide em duas partes: a API de comunicação com o banco de dados e a aplicação desenvolvida em Angular. Também são apresentados alguns exemplos dos principais códigos implementados.

4.4.1 API *Back-end* Java utilizando Spring

A Figura 14 apresenta a estrutura dos arquivos da API na IDE do eclipse.

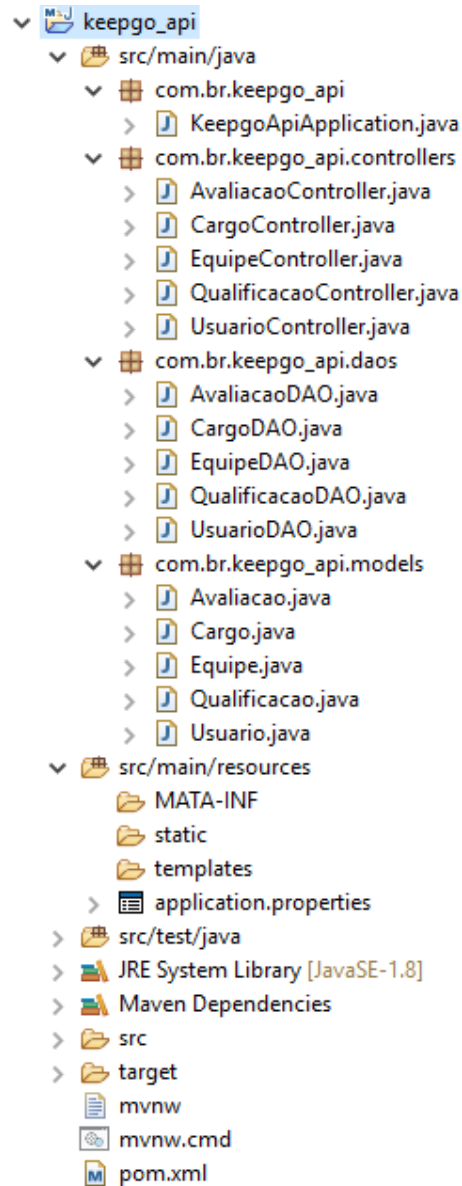


Figura 15 - Estrutura do projeto da API

Para a construção do projeto foi utilizado o Maven, que é uma ferramenta desenvolvida pela Apache utilizada para gerenciar as dependências do projeto e automatizar o processo de *build*. As dependências são configurada em um arquivo chamado pom.xml, a Listagem 1 apresenta o arquivo utilizado no projeto.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.br.teste</groupId>
```

```

<artifactId>keepgo_api</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packaging>

<name>keepgo_api</name>
<description>Demo project for Spring Boot</description>

<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.5.8.RELEASE</version>
  <relativePath/>
</parent>

<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>
  <java.version>1.8</java.version>
</properties>

<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-mail</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web-services</artifactId>
  </dependency>

  <dependency>
    <groupId>com.jayway.jsonpath</groupId>
    <artifactId>json-path</artifactId>
    <scope>test</scope>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>

  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
  </dependency>

  <dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-jasper</artifactId>
  </dependency>

```

```

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-tomcat</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
        </dependency>

    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
        <finalName>KeepGo_api</finalName>
    </build>
</project>

```

Listagem 1 - Arquivo pom.xml

A implementação é basicamente dividida em três tipos de classes: *models*, *Data Access Object* (DAOs) e *controllers*. Cada uma trata do acesso a determinada informação: os *models* servem como modelo para a criação das tabelas no banco e são os objetos utilizados na comunicação da API. Os DAOs são os objetos recebidos e retornados nas requisições. A Listagem 2 apresenta a classe Cargo como exemplo.

```

package com.br.keepgo_api.models;

import java.util.ArrayList;
import java.util.List;

import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToMany;

import org.hibernate.annotations.Where;

@Entity
public class Cargo {

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private int id;

```



```

private String descricao;
private boolean inativo;

@OneToMany(fetch=FetchType.EAGER)
@Where(clause = "not inativo")
private List<Qualificacao> qualificacao= new ArrayList<Qualificacao>();

public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

public String getDescricao() {
    return descricao;
}

public void setDescricao(String descricao) {
    this.descricao = descricao;
}

public boolean isInativo() {
    return inativo;
}

public void setInativo(boolean inativo) {
    this.inativo = inativo;
}

public List<Qualificacao> getQualificacao() {
    return qualificacao;
}

public void setQualificacao(List<Qualificacao> qualificacao) {
    this.qualificacao = qualificacao;
}

@Override
public String toString() {
    return "Cargo [id=" + id + ", descricao=" + descricao + ", inativo="
+ inativo + ", qualificacao="
        + qualificacao + "];"
}
}

```

Listagem 2 - Exemplo da classe Cargo

Os DAOs são responsáveis pelo acesso ao banco de dados. Devem ser anotados com a anotação `@Repository` indicando para o Spring que essa classe pertence à camada de persistência. Todos os DAOs da API possuem basicamente os mesmos métodos de busca, listagem, atualização e inclusão. Na Listagem 3 está um exemplo do DAO de cargos.

```

package com.br.keepgo_api.daos;

import java.util.List;
import java.util.NoSuchElementException;

import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

import org.springframework.stereotype.Repository;

import com.br.keepgo_api.models.Cargo;

@Repository
public class CargoDAO {

    @PersistenceContext
    private EntityManager manager;

    public List<Cargo> getCargos(){

        List<Cargo> cargos = manager.createQuery("SELECT c FROM Cargo c",
Cargo.class).getResultList();

        if(cargos.isEmpty()) {
            throw new NoSuchElementException("Nenhum cargo encontrado.");
        }

        return cargos;
    }

    public Cargo getCargoId(int id) {
        List<Cargo> cargos = manager.createQuery("SELECT c FROM Cargo c where
c.id = :buscaId", Cargo.class)
            .setParameter("buscaId", id).getResultList();

        if(cargos.isEmpty()) {
            throw new NoSuchElementException("Nenhum cargo encontrado com
o id " + id + ".");
        }

        return cargos.get(0);
    }

    public void updateCargo(Cargo cargo) {
        manager.merge(cargo);
    }

    public void setCargo(Cargo cargo) {
        manager.persist(cargo);
    }
}

```

Listagem 3 - Exemplo classe CargoDAO

Por fim, as classes de comunicação, os *controllers*. É por meio dos *controllers* que a API recebe e envia informações, devem ser anotados com `@Controller` ou `@RestController`, indicando para o Spring que essa classe possui métodos que processam as requisições feitas

para a API. A estrutura dos *controllers* é baseada em métodos que utilizam os objetos das classes DAOs, mapeados para cada requisição.

O Spring faz o mapeamento dos métodos utilizando a anotação `@RequestMapping()`. Por meio dessa anotação é possível informar qual endereço o método vai atender e o tipo de requisição, podendo haver dois mapeamentos para o mesmo endereço, mas com tipos de requisição diferentes. São também utilizadas outras anotações para recuperar as informações repassadas nas requisições, como pode ser verificado na Listagem 4.

```

package com.br.keepgo_api.controllers;

import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import com.br.keepgo_api.daos.CargoDAO;
import com.br.keepgo_api.models.Cargo;

@RestController
@CrossOrigin()
@RequestMapping("/cargos")
public class CargoController {

    @Autowired
    private CargoDAO cargoDAO;

    @RequestMapping(method=RequestMethod.GET)
    public List<Cargo> getCargo(){
        return cargoDAO.getCargos();
    }

    @RequestMapping(value="/{cargoId}", method=RequestMethod.GET)
    public Cargo getCargoId( @PathVariable("cargoId") int cargoId) {
        return cargoDAO.getCargoId(cargoId);
    }

    @RequestMapping(value="/{cargoId}", method=RequestMethod.PUT)
    @Transactional
    public void updateCargoId( @PathVariable("cargoId") int cargoId,
@RequestBody Cargo cargo) {
        cargoDAO.updateCargo(cargo);
    }

    @RequestMapping(value = "/new", method=RequestMethod.POST)
    @Transactional
    public void setCargo(@RequestBody Cargo cargo) {
        cargoDAO.setCargo(cargo);
    }
}

```

Listagem 4 - Exemplo classe CargoController

Outro recurso utilizado no projeto é o envio de *e-mails* utilizando o objeto MailSender do Spring, que utiliza as configurações definidas no arquivo application.properties. Nesse arquivo, também podem ser definidas outras propriedades, como as de acesso ao banco de dados, por exemplo. A Listagem 5 mostra as propriedades definidas no arquivo que foram utilizadas no projeto.

```
spring.jpa.hibernate.ddl-auto=none
spring.datasource.url=jdbc:mysql://localhost:3306/Keepgo
spring.datasource.username=root
spring.datasource.password=<<senha do usuário do banco de dados>>

spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=<<e-mail para o envio>>
spring.mail.password=<<senha do email>>
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
spring.mail.properties.mail.smtp.connectiontimeout=5000
spring.mail.properties.mail.smtp.timeout=5000
spring.mail.properties.mail.smtp.writetimeout=5000
```

Listagem 5 - Propriedades definidas para o Spring

A função “EnviaEmailAvaliacao” na Listagem 6 é responsável pelo envio dos *e-mails* para o supervisor e funcionário no final da avaliação.

```
package com.br.keepgo_api.controllers;

import java.text.SimpleDateFormat;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.mail.MailSender;
import org.springframework.mail.SimpleMailMessage;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import com.br.keepgo_api.daos.AvaliacaoDAO;
import com.br.keepgo_api.daos.CargoDAO;
import com.br.keepgo_api.daos.UsuarioDAO;
import com.br.keepgo_api.models.Avaliacao;
import com.br.keepgo_api.models.Cargo;
import com.br.keepgo_api.models.Usuario;

@RestController
@CrossOrigin()
@RequestMapping("/avaliacoes")
public class AvaliacaoController {

    @Autowired
    private AvaliacaoDAO avaliacaoDAO;
```

```

@Autowired
private UsuarioDAO usuarioDAO;

@Autowired
private CargoDAO cargoDAO;

@Autowired
private MailSender sender;

@RequestMapping(value="/user/{usuarioId}", method=RequestMethod.GET)
public List<Avaliacao> getAvaliacoesUsuario(@PathVariable("usuarioId") int
idUsuario){
    return avaliacaoDAO.getAvaliacaoUsuario(idUsuario);
}

@RequestMapping(value="/{avaliacaoId}", method=RequestMethod.GET)
public Avaliacao getAvaliacao(@PathVariable("avaliacaoId") int
idAvaliacao){

    Avaliacao retAvaliacao = avaliacaoDAO.getAvaliacaoId(idAvaliacao);

    return retAvaliacao;
}

@RequestMapping(value="/{avaliacaoId}", method=RequestMethod.PUT)
@Transactional
public void updateAvaliacao( @PathVariable("avaliacaoId") int avaliacaoId,
@RequestBody Avaliacao avaliacao) {
    avaliacaoDAO.updateAvaliacao(avaliacao);
    this.EnviaEmailAvaliacao(avaliacao);
}

@RequestMapping(value = "/new", method=RequestMethod.POST)
@Transactional
public void setAvaliacao(@RequestBody Avaliacao avaliacao) {

    avaliacaoDAO.setAvaliacao(avaliacao);
    this.EnviaEmailAvaliacao(avaliacao);
}

public void EnviaEmailAvaliacao(Avaliacao avaliacao){

    Usuario usuario = usuarioDAO.getUsuarioID(avaliacao.getIdUsuario());
    Usuario supervisor =
usuarioDAO.getUsuarioID(avaliacao.getIdSupervisor());

    Cargo cargo = cargoDAO.getCargoId(avaliacao.getIdCargo());

    SimpleDateFormat formata = new SimpleDateFormat("dd/MM/yyyy HH:mm");

    SimpleMailMessage email = new SimpleMailMessage();

    //Envia e-mail para o usuário
    email.setSubject("Avaliação #" + avaliacao.getId());
    email.setTo(usuario.getEmail());
    email.setText(
        "Avaliação de " + usuario.getNome() + " para o cargo de
" + cargo.getDescricao() + ".\n" +

```

```

        "Encaminhada para " + supervisor.getNome() + ".\n" +
        "\n" +
        "Data de abertura: " + avaliacao.getDtAbertura() + "\n" +
        "Data de alteração" + avaliacao.getDtAlteracao() + "\n" +
        "\n" +
        "Pontos positivos: " + avaliacao.getPositivos() + ".\n"
+
        "\n" +
        "Pontos a melhorar: " + avaliacao.getMelhorar() + ".\n"
+
        "\n" +
        "Conclusão: " + avaliacao.getConclusao()+ ".\n" +
        "Avaliação se encontra " +
this.showStatus(avaliacao.getStatus()) + ".";
        email.setFrom("keepgo@keepgo.com.br");

        sender.send(email);

        //Envia e-mail para o supervisor
        email.setSubject("Avaliação #" + avaliacao.getId() + " recebida");
        email.setTo(supervisor.getEmail());
        email.setText(
            "Avaliação de " + usuario.getNome() + " para o cargo de
" + cargo.getDescricao() + ".\n" +
            "Encaminhada para você.\n" +
            "\n" +
            "Data de abertura: " +
formata.format(avaliacao.getDtAbertura()) + "\n" +
            "Data de alteração" +
formata.format(avaliacao.getDtAbertura()) + "\n" +
            "\n" +
            "Pontos positivos: " + avaliacao.getPositivos() + ".\n"
+
            "\n" +
            "Pontos a melhorar: " + avaliacao.getMelhorar() + ".\n"
+
            "\n" +
            "Conclusão: " + avaliacao.getConclusao()+ ".\n" +
            "Avaliação se encontra " +
this.showStatus(avaliacao.getStatus()) + ".\n" +
            "\n" +
            "Essa avaliação se encontra disponível no seu menu de
avaliações.");

        sender.send(email);
    }

    public String showStatus(String status){
        String descricaoStato;
        switch (status) {
            case "P":
                descricaoStato = "pendente";
                break;
            case "F":
                descricaoStato = "fechada";
                break;
            case "A":
                descricaoStato = "aprovada";
                break;

```

```
                default:
                    descricaoStato = "";
                    break;
            }
            return descricaoStato;
        }
    }
```

Listagem 6 - Exemplo class AvaliacaoController

4.4.2 Aplicação *Front-end* em Angular

A aplicação *front-end* foi desenvolvida em Angular na sua versão 5, utilizando componentes do Angular Material.

Os componentes do Angular são compostos basicamente por um arquivo de classe TypeScript, um arquivo HTML e um arquivo CSS, organizados em uma mesma pasta, como apresentado na estrutura do sistema na Figura 15.

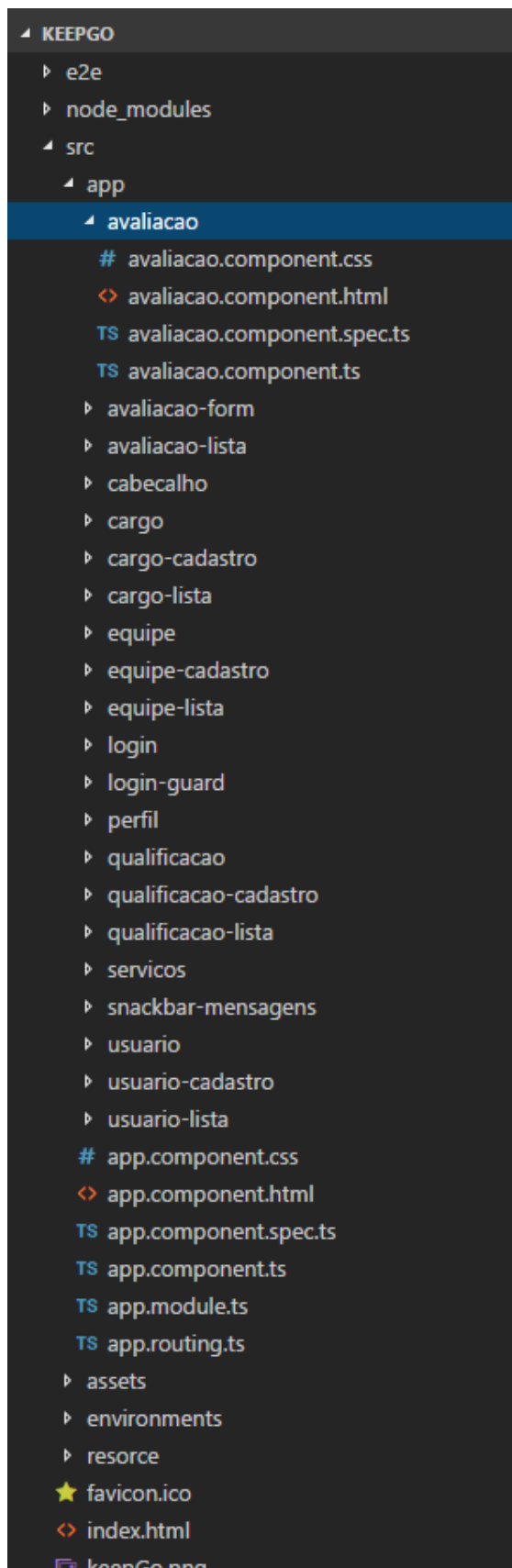


Figura 16 - Estrutura de pastas do projeto Angular

Tendo um componente principal que faz a importação dos demais. Esse componente é anotado com a anotação “@NgModule” que recebe todos os componentes e classes que serão utilizados pela aplicação. Esse componente pode ser verificado na Listagem 7.

```
import { BrowserModule } from '@angular/platform-browser';
import { BrowserAnimationsModule} from '@angular/platform-browser/animations';
import { NgModule } from '@angular/core';
import { FormsModule , ReactiveFormsModule} from '@angular/forms';
import { HttpClientModule } from '@angular/http';
import { RouterModule, Routes} from '@angular/router';
import 'rxjs/add/operator/map';

// Componentes do AngularMaterial
import {MatFormFieldModule} from '@angular/material/form-field';
import {
  MatAutocompleteModule,
  MatButtonModule,
  MatButtonModuleToggleModule,
  MatCardModule,
  MatCheckboxModule,
  MatChipsModule,
  MatDatepickerModule,
  MatDialogModule,
  MatExpansionModule,
  MatGridListModule,
  MatIconModule,
  MatInputModule,
  MatListModule,
  MatMenuModule,
  MatNativeDateModule,
  MatPaginatorModule,
  MatProgressBarModule,
  MatProgressSpinnerModule,
  MatRippleModule,
  MatSelectModule,
  MatSidenavModule,
  MatSliderModule,
  MatSlideToggleModule,
  MatSnackBarModule,
  MatSortModule,
  MatTableModule,
  MatTabsModule,
  MatToolbarModule,
  MatTooltipModule,
  MatStepperModule
} from '@angular/material';
import {MatRadioModule} from '@angular/material/radio';
import {MatRadioGroup} from '@angular/material/radio';

// Componentes criados
import { AppComponent } from './app.component';
import { CabecalhoComponent} from './cabecalho/cabecalho.component';
import { PerfilComponent} from './perfil/perfil.component';
import { LoginComponent} from './login/login.component';
import { UsuarioComponent} from './usuario/usuario.component';
```

```

import { UsuarioListaComponent } from './usuario-lista/usuario-
lista.component';
import { UsuarioCadastroComponent } from './usuario-cadastro/usuario-
cadastro.component';
import { QualificacaoComponent } from './qualificacao/qualificacao.component';
import { SnackbarMensagensComponent } from './snackbar-mensagens/snackbar-
mensagens.component';
import { CargoComponent } from './cargo/cargo.component';
import { CargoListaComponent } from './cargo-lista/cargo-lista.component';
import { CargoCadastroComponent } from './cargo-cadastro/cargo-
cadastro.component';
import { AvaliacaoComponent } from './avaliacao/avaliacao.component';
import { AvaliacaoListaComponent } from './avaliacao-lista/avaliacao-
lista.component';
import { AvaliacaoFormComponent } from './avaliacao-form/avaliacao-
form.component';

// Servicos criados
import { ConnctionService } from './servicos/Connection.service'
import { UsuarioCheckService } from './servicos/usuarioCheckService';

//Componentes de rotas e controle de usuario
import { routing } from './app.routing'
import { LoginGuard } from './login-guard/login.guard';

@NgModule({
  declarations: [
    AppComponent,
    CabecalhoComponent,
    PerfilComponent,
    LoginComponent,
    UsuarioComponent,
    UsuarioListaComponent,
    QualificacaoComponent,
    UsuarioCadastroComponent,
    SnackbarMensagensComponent,
    CargoComponent,
    CargoListaComponent,
    CargoCadastroComponent,
    AvaliacaoComponent,
    AvaliacaoListaComponent,
    AvaliacaoFormComponent
  ],
  imports: [
    BrowserModule,
    BrowserAnimationsModule,
    FormsModule,
    ReactiveFormsModule,
    HttpClientModule,
    MatFormFieldModule,
    MatAutocompleteModule,
    MatButtonModule,
    MatButtonModuleToggleModule,
    MatCardModule,
    MatCheckboxModule,
    MatChipsModule,

```

```

    MatDatepickerModule,
    MatDialogModule,
    MatExpansionModule,
    MatGridListModule,
    MatIconModule,
    MatInputModule,
    MatListModule,
    MatMenuModule,
    MatNativeDateModule,
    MatPaginatorModule,
    MatProgressBarModule,
    MatProgressSpinnerModule,
    MatRadioModule,
    MatRippleModule,
    MatSelectModule,
    MatSidenavModule,
    MatSliderModule,
    MatSlideToggleModule,
    MatSnackBarModule,
    MatSortModule,
    MatTableModule,
    MatTabsModule,
    MatToolbarModule,
    MatTooltipModule,
    MatStepperModule,
    routing
  ],
  entryComponents: [
    SnackbarMensagensComponent
  ],
  providers: [
    LoginGuard,
    ConnctionService,
    UsuarioCheckService
  ],
  bootstrap: [AppComponent]
})
export class AppModule {}

```

Listagem 7 - Componente principal da aplicação

Nessa classe também é definido o componente inicial do sistema, na propriedade “bootstrap”. Nesse componente é encontrada a diretiva “router-outlet” que recebe o componente definido no arquivo de rotas. Nele, são configurados quais componentes devem ser apresentados para cada endereço. Na Listagem 8 está o *template* HTML do componente inicial.

```

<app-cabecalho></app-cabecalho>
<main>
  <mat-sidenav-container class="app-container" >
    <mat-sidenav mode="side" opened="true" class="menu-lateral">
      <ul>
        <li *ngIf="!valida.isAdmin()">
          <a [routerLink]="['/perfil']">Perfil profissional</a>

```

```

        </li>

        <li *ngIf="! valida.isAdmin()">
            <a [routerLink]="['/avaliacoes']">Avaliações</a>
        </li>

        <li *ngIf="! valida.isAdmin()">
            <a [routerLink]="['/cargos']">Consulta Cargos</a>
        </li>

        <li *ngIf="valida.isAdmin()" >
            <a [routerLink]="['/usuarios']"
routerLinkActive="active">Usuários</a>
        </li>

        <li *ngIf="valida.isAdmin()">
            <a [routerLink]="['/cargos']">Cargo</a>
        </li>

        <li *ngIf="valida.isAdmin()" >
            <a href="#">Qualificações</a>
        </li>

        <li *ngIf="valida.isAdmin()">
            <a href="#">Equipe</a>
        </li>

    </ul>
</mat-sidenav>
<mat-sidenav-content>
    <router-outlet></router-outlet>
</mat-sidenav-content>
</mat-sidenav-container>
</main>

```

Listagem 8 - Template HTML do componente inicial

Na Listagem 9 é apresentada a classe de rotas, nela são definidos os endereços que cada componente vai atender, através das propriedades 'path' e 'component', representando o endereço e o componente respectivamente, ainda temos a propriedade opcional 'canActivate' com ela é possível definir um componente que fará o gerenciamento do acesso ao endereço.

```

import {Routes, RouterModule} from '@angular/router';

import {PerfilComponent} from './perfil/perfil.component';
import {LoginComponent} from './login/login.component';
import {UsuarioCadastroComponent} from './usuario-cadastro/usuario-
cadastro.component';
import {UsuarioListaComponent} from './usuario-lista/usuario-lista.component';
import {CargoCadastroComponent} from './cargo-cadastro/cargo-
cadastro.component';
import {CargoListaComponent} from './cargo-lista/cargo-lista.component';
import {AvaliacaoFormComponent} from './avaliacao-form/avaliacao-
form.component';

```

```

import {AvaliacaoListaComponent} from './avaliacao-lista/avaliacao-
lista.component'

import {LoginGuard} from './login-guard/login.guard';

const appRoutes: Routes = [
  {path: 'login', component: LoginComponent},
  {path: 'perfil', component: PerfilComponent, canActivate:[LoginGuard]},
  {path: 'cargos', component: CargoListaComponent,
canActivate:[LoginGuard]},
  {path: 'cargos/:id', component: CargoCadastroComponent,
canActivate:[LoginGuard]},
  {path: 'usuarios', component: UsuarioListaComponent,
canActivate:[LoginGuard]},
  {path: 'usuarios/:id', component: UsuarioCadastroComponent,
canActivate:[LoginGuard]},
  {path: 'avaliacoes', component: AvaliacaoListaComponent,
canActivate:[LoginGuard]},
  {path: 'avaliacoes/:id', component: AvaliacaoFormComponent,
canActivate:[LoginGuard]}
]

export const routing = RouterModule.forRoot(appRoutes);

```

Listagem 9 - Classe de rotas da aplicação

Os componentes listados na constante “appRoutes” na propriedade “component” são incluídos na página principal, encapsulando as funções definidas na classe e as configurações do CSS. A seguir é apresentado um exemplo de um componente com o arquivo de *template* HTML, CSS e o arquivo da classe TypeScript, respectivamente representados pelas Listagens 10, 11 e 12

```

<div class="app-sub-cabecalho">
  <h1>Perfil</h1>
</div>
<div class="app-sub-container">
  <mat-card>
    <form [formGroup]="perfilForm">

      <mat-form-field class="row-100">
        <input matInput formControlName="nome" maxLength="100"
[ngModel]="perfilUsuario.nome" placeholder="Nome" disabled >
      </mat-form-field>

      <mat-form-field class="row-50">
        <input matInput formControlName="cpf"
[ngModel]="perfilUsuario.cpf" placeholder="CPF" disabled>
      </mat-form-field>

      <mat-form-field class="row-50">
        <input matInput formControlName="rg"
[ngModel]="perfilUsuario.rg" placeholder="RG" disabled>
      </mat-form-field>
    </form>
  </mat-card>
</div>

```

```

        <div>
            <mat-form-field>
                <input matInput [matDatepicker]="dtNascimento"
placeholder="Data de Nascimento" [ngModel]="perfilUsuario.dtNascimento"
formControlName="dtNascimento" disabled readonly>
                <mat-datepicker-toggle matSuffix
[for]="dtNascimento"></mat-datepicker-toggle>
                <mat-datepicker #dtNascimento ></mat-datepicker>
            </mat-form-field>

            <mat-form-field>
                <input matInput [matDatepicker]="dtAdmissao"
placeholder="Data de Admissão" [ngModel]="perfilUsuario.dtAdmissao"
formControlName="dtAdmissao" disabled readonly>
                <mat-datepicker-toggle matSuffix [for]="dtAdmissao"></mat-
datepicker-toggle>
                <mat-datepicker #dtAdmissao></mat-datepicker>
            </mat-form-field>
        </div>

        <mat-form-field class="row-100">
            <input matInput [(ngModel)]="perfilUsuario.endereco"
maxLength="100" placeholder="Endereço" formControlName="endereco">
        </mat-form-field>

        <mat-form-field class="row-50">
            <input matInput [(ngModel)]="perfilUsuario.cidade"
placeholder="Cidade" formControlName="cidade">
        </mat-form-field>

        <mat-form-field class="row-50">
            <input matInput [(ngModel)]="perfilUsuario.cep"
placeholder="CEP" maxLength="8" formControlName="cep" >
        </mat-form-field>

        <mat-form-field class="row-50">
            <input matInput [(ngModel)]="perfilUsuario.telefone"
placeholder="Telefone" formControlName="telefone">
        </mat-form-field>

        <mat-form-field class="row-50">
            <input matInput [(ngModel)]="perfilUsuario.celular"
placeholder="Celular" formControlName="celular">
        </mat-form-field>

        <mat-form-field class="row-100">
            <textarea matInput [(ngModel)]="perfilUsuario.observacao"
#observacao placeholder="Observação" matTextareaAutosize
matAutosizeMinRows="1" matAutosizeMaxRows="8" formControlName="observacao"
maxLength="256" >>/textarea>
            <mat-hint align="end">{{observacao.value.length}} / 256</mat-
hint>
        </mat-form-field>

        <mat-card>
            <h2>Cargo</h2>

```

```

        <mat-form-field class="row-100">
            <input matInput [ngModel]="perfilCargo.descricao"
placeholder="Cargo atual" formControlName="idCargo" disabled>
        </mat-form-field>
    </mat-card>

    <mat-card>
        <h2>Qualificações</h2>
        <mat-selection-list>
            <mat-list-option checkboxPosition="before" *ngFor="let
qualificacao of perfilUsuario.qualificacao; let i = index"
selected="{{qualificacao.selecionada}}" (click)="onSeleciona(i)">
                {{qualificacao.descricao}}
            </mat-list-option>
        </mat-selection-list>

        <div class="row-100">
            <mat-form-field class="row-50">
                <mat-select placeholder="Adicionar qualificação"
[[value]]="qualificacaoSelecionada" [disabled]="(listaQualificacoesAdd.length
== 0)? true : false">
                    <mat-option></mat-option>
                    <mat-option *ngFor="let quali of
listaQualificacoesAdd; let i = index" value="{{i}}">{{quali.descricao}}</mat-
option>
                </mat-select>
            </mat-form-field>
            <button mat-fab color="primary"
(click)="addQualificacao(qualificacaoSelecionada)" matTooltip="Adicionar
qualificação" [disabled]="qualificacaoSelecionada? false : true">
                <i class="material-icons">playlist_add</i>
            </button>
        </div>
    </mat-card>
    <div class="app-button-row">
        <button mat-raised-button color="primary" type="submit"
(click)="onCadastrar()">Salvar</button>
    </div>

</form>
</mat-card>
</div>

```

Listagem 10 - Template HTML do componente Perfil

```

.perfil-qualificacoes{
    border: solid 1px gray;
    padding: 10px;
}
.app-check{
    color: black;
}
h2{
    height: 30px;
    font-size: 30px;
    padding-bottom: 10px;
}

```

```
}

```

Listagem 11 - Arquivo CSS do *component Perfil*

```
import { Component, OnInit } from '@angular/core';
import { Form, FormGroup, FormBuilder, Validators, } from '@angular/forms';
import { Router } from '@angular/router';
import { MatSnackBar } from '@angular/material';

import { UsuarioComponent } from '../usuario/usuario.component';
import { Usuario } from '../usuario/usuario';
import { CargoComponent } from '../cargo/cargo.component';
import { ConnctionService } from '../servicos/Connection.service';
import { QualificacaoComponent } from '../qualificacao/qualificacao.component';
import { SnackbarMensagensComponent } from '../snackbar-mensagens/snackbar-
mensagens.component';

@Component({
  selector: 'app-perfil',
  templateUrl: './perfil.component.html',
  styleUrls: ['./perfil.component.css']
})
export class PerfilComponent implements OnInit {

  perfilUsuario: UsuarioComponent = new UsuarioComponent();
  perfilCargo: CargoComponent = new CargoComponent();

  listaQualificacoesAdd: Array<QualificacaoComponent> = [];

  perfilForm: FormGroup;

  qualificacaoSelecionada;

  usuarioLogado: Usuario;

  constructor(
    private connection:ConnctionService,
    private formBuilder: FormBuilder,
    private router: Router,
    public snackBar: MatSnackBar) {

    console.log("inicio");

    this.perfilForm = this.formBuilder.group({
      nome: [{value: '', disabled: true}],
      cpf: [{value: '', disabled: true}],
      rg: [{value: '', disabled: true}],
      dtNascimento: [{value: '', disabled: true}],
      dtAdmissao: [{value: '', disabled: true}],
      endereco: ['', Validators.required],
      cidade: ['', Validators.required],
      cep: ['', Validators.required],
      telefone: [''],
      celular: [''],
      observacao: ['', Validators.maxLength],
      idCargo: [{value: '', disabled: true}],
    });
  }
}
```



```

    cargo: ['', Validators.required]
  })

  // Recupera o usuário Logado.
  this.usuarioLogado = JSON.parse(localStorage.getItem("currentUser"));

  if(this.usuarioLogado.id){
    this.connection.getUsuarioID(this.usuarioLogado.id)
      .subscribe(
        retorno =>{
          this.perfilUsuario = retorno;

          //Coverte as datas do formato JSON
          this.perfilUsuario.dtNascimento = new
Date(this.perfilUsuario.dtNascimento);
          this.perfilUsuario.dtAdmissao = new
Date(this.perfilUsuario.dtAdmissao);

          this.connection.getUsuarioQualificacoes(this.perfilUsuario.id,
this.perfilUsuario.idCargo)
            .subscribe(ret =>{
              this.perfilUsuario.qualificacao = ret;

              this.connection.getQualificacoes()
                .subscribe(
                  retorno =>{
                    this.listaQualificacoesAdd =
this.connection.filterList(retorno, this.perfilUsuario.qualificacao);
                  },
                  erro =>{
                    console.log(erro);
                  }
                );
            }, erro => console.log(erro));

          connection.getCargoId(this.perfilUsuario.idCargo)
            .subscribe(
              retorno => this.perfilCargo = retorno,
              erro => console.log(erro)
            );

          console.log(this.perfilUsuario);

        },erro => {
          console.log(erro);
          this.showMensagem("Falha ao carregar perfil do usuário", false);
        }
      );
  }else{
    this.showMensagem("Falha ao recuperar usuário logado.", false);
    this.router.navigate(['login']);
  }
}

```

```

ngOnInit() {
}

onSeleciona(index:number){
  this.perfilUsuario.qualificacao[index].selecionada =
  !this.perfilUsuario.qualificacao[index].selecionada;
}

onCadastrar(){
  let cadastrarUsuario: UsuarioComponent = this.perfilUsuario;
  cadastrarUsuario.qualificacao = this.perfilUsuario.qualificacao.filter(x=>
{return x.selecionada});

  this.connection.setUsuario(cadastrarUsuario)
    .subscribe(retorno =>{
      this.showMensagem("Perfil atualizado com sucesso", true);
    },
    erro =>{
      this.showMensagem("Erro ao gravar perfil", false);
      console.log(erro);
    });
}

addQualificacao(qualificacaoId:number){
  this.listaQualificacoesAdd[qualificacaoId].selecionada = true;
this.perfilUsuario.qualificacao.push(this.listaQualificacoesAdd[qualificacaoId
]);
  this.listaQualificacoesAdd.splice(qualificacaoId, 1);
  this.qualificacaoSelecionada = 0;
}

showMensagem(mensagem: string, sucesso: boolean){
  this.snackBar.openFromComponent(
    SnackbarMensagensComponent,
    {data: {msg: mensagem, retorno: sucesso},
    duration: 2000})
}
}

```

Listagem 12 - Classe TypeScript do componente Perfil

Na Listagem 12 pode ser observado que o componente é anotado com a anotação “@Component” que possui três propriedades sendo: a) o seletor que é a diretiva a ser utilizada para adicionar o componente no HTML, no caso do componente Perfil basta adicionar a diretiva “<app-perfil></ app-perfil>” que um componente perfil é adicionado na página; b) o *template* indica qual é o arquivo HTML do componente; c) o arquivo CSS que será aplicado somente nesse componente.

Outro tipo de classe implementada no Angular são as classes de serviços, que centralizam recursos utilizados no sistema, para o projeto foi criada uma classe de serviço para a comunicação com a API, apresentada na Listagem 13.

```

import {Injectable} from '@angular/core';
import {Http, Headers} from '@angular/http';
import {Observable} from 'rxjs';
import {UsuarioComponent} from '../usuario/usuario.component';
import {CargoComponent} from '../cargo/cargo.component';
import {QualificacaoComponent} from '../qualificacao/qualificacao.component';
import {EquipeComponent} from '../equipe/equipe.component';
import {AvaliacaoComponent} from '../avaliacao/avaliacao.component';

@Injectable()
export class ConnctionService {

  headers: Headers;
  prefixoUrl: string = 'http://localhost:8080/';

  UrlUsuario: string = this.prefixoUrl + 'usuarios';
  UrlCargo: string = this.prefixoUrl + 'cargos';
  UrlQulificacao: string = this.prefixoUrl + 'qualificacoes';
  UrlEquipe: string = this.prefixoUrl + 'equipes';
  UrlAvaliacoes: string = this.prefixoUrl + 'avaliacoes';

  constructor(private http:Http){
    this.http = http;
    this.headers = new Headers();
    this.headers.append('Content-Type', 'application/json');
  }

  // Usuarios
  ListarUsuarios(): Observable<UsuarioComponent[]>{
    return this.http.get(this.UrlUsuario).map(retorno => retorno.json());
  }

  getUsuarioID(id: number): Observable<UsuarioComponent>{
    return this.http.get(this.UrlUsuario + '/' + id).map(retorno =>
retorno.json());
  }

  setUsuario(usuario: UsuarioComponent): Observable<MensagemResposta>{
    if(usuario.id){
      return this.http.put(this.UrlUsuario + "/" + usuario.id,
JSON.stringify(usuario), {headers: this.headers})
        .map( () => new MensagemResposta("Usuário alterado com
sucesso"));
    }else{
      return this.http.post(this.UrlUsuario + "/new",
JSON.stringify(usuario), {headers: this.headers})
        .map( () => new MensagemResposta('Usuário incluído com
sucesso'));
    }
  }

  getSupervisores(idUsuario: number): Observable<UsuarioComponent[]>{
    return this.http.get( this.UrlUsuario + "/super/" +
idUsuario).map(retorno => retorno.json());
  }
}

```

```

//Cargos
ListaCargos(): Observable<CargoComponent[]>{
    return this.http.get(this.UrlCargo).map(retorno => retorno.json());
}

getCargoId(id: number): Observable<CargoComponent>{
    return this.http.get(this.UrlCargo + '/' + id).map(retorno =>
retorno.json());
}

setCargo(cargo: CargoComponent): Observable<MensagemResposta>{
    if(cargo.id){
        return this.http.put(this.UrlCargo + "/" + cargo.id,
JSON.stringify(cargo), {headers: this.headers})
        .map( () => new MensagemResposta("Cargo alterado com
sucesso"))
    }else{
        return this.http.post(this.UrlCargo + "/new",
JSON.stringify(cargo), {headers: this.headers})
        .map( () => new MensagemResposta('Cargo incluído com
sucesso'));
    }
}

//Qualificações
getQualificacoes(): Observable<QualificacaoComponent[]>{
    return this.http.get(this.UrlQulificacao).map(retorno =>
retorno.json());
}

setQualificacao(qualificacao: QualificacaoComponent):
Observable<MensagemResposta>{
    if(qualificacao.id){
        return this.http.put(this.UrlQulificacao + "/" + qualificacao.id,
JSON.stringify(qualificacao), {headers: this.headers})
        .map( () => new MensagemResposta("Qualificação alterada com
sucesso"))
    }else{
        return this.http.post(this.UrlQulificacao + "/new",
JSON.stringify(qualificacao), {headers: this.headers})
        .map( () => new MensagemResposta('Qualificação incluída com
sucesso'));
    }
}

//Equipes
ListaEquipes(): Observable<EquipeComponent[]>{
    return this.http.get(this.UrlEquipe).map(retorno => retorno.json());
}

getEquipeId(id: number): Observable<EquipeComponent>{
    return this.http.get(this.UrlEquipe + '/' + id).map(retorno =>
retorno.json());
}

setEquipe(equipe: EquipeComponent): Observable<MensagemResposta>{

```

```

        if(equipe.id){
            return this.http.put(this.UrlEquipe + "/" + equipe.id,
JSON.stringify(equipe), {headers: this.headers})
                .map( () => new MensagemResposta("Equipe alterada com
sucesso"))
        }else{
            return this.http.post(this.UrlEquipe + "/new",
JSON.stringify(equipe), {headers: this.headers})
                .map( () => new MensagemResposta('Equipe incluída com
sucesso'));
        }
    }

    // Avaliações
    getAvaliacoesUsuario(idUsuario: number): Observable<AvaliacaoComponent[]>{
        return this.http.get(this.UrlAvaliacoes + '/user/' +
idUsuario).map(retorno => retorno.json());
    }

    getAvaliacaoID(id: number): Observable<AvaliacaoComponent>{
        return this.http.get(this.UrlAvaliacoes + '/' + id).map(retorno =>
retorno.json());
    }

    setAvaliacao(avaliacao: AvaliacaoComponent): Observable<MensagemResposta>{
        if(avaliacao.id){
            return this.http.put(this.UrlAvaliacoes + "/" + avaliacao.id,
JSON.stringify(avaliacao), {headers: this.headers})
                .map( () => new MensagemResposta("Avaliação alterada com
sucesso"))
        }else{
            return this.http.post(this.UrlAvaliacoes + "/new",
JSON.stringify(avaliacao), {headers: this.headers})
                .map( () => new MensagemResposta('Avaliação incluída com
sucesso'));
        }
    }

    //Funções utilitarias
    filterList(lista:Array<QualificacaoComponent>,
remover:Array<QualificacaoComponent>): Array<QualificacaoComponent>{

        let retorno:Array<QualificacaoComponent> = [];

        let listar:boolean = true;

        for (var index = 0; index < lista.length; index++){
            listar = true;
            for(var i = 0; i < remover.length; i++){
                if(remover[i].id == lista[index].id){
                    listar = false;
                }
            }
            if(listar){
                retorno.push(lista[index]);
            }
        }
    }

```

```

    }
    return retorno;
  }

  getUsuarioQualificacoes(idUsuario: number, idCargo:number):
  Observable<QualificacaoComponent[]>{
    return this.http.get(this.UrlUsuario + '/qualificacao/' + idUsuario +
    '/' + idCargo).map(retorno => retorno.json());
  }

  login(usuario: UsuarioComponent): Observable<UsuarioComponent>{
    return this.http.post(this.UrlUsuario + '/access',
    JSON.stringify(usuario), {headers: this.headers}).map( retorno =>
    retorno.json())
  }

  logout(){
    localStorage.removeItem('currentUser');
  }
}

class MensagemResposta{
  constructor(private _mensagem: string){
    this._mensagem = _mensagem;
  }

  get mensagem(): string{
    return this._mensagem;
  }
}

```

Listagem 13 - Classe TypeScript do componente Connection

Os métodos de comunicação com a API implementados neste serviço retornam um objeto do tipo “Observable”. Esse objeto retorna o resultado da requisição de maneira assíncrona. Assim, no componente são definidas as funções de sucesso e falha, como pode ser observado na Listagem 12, página 56, é no método construtor que é utilizado o método “this.connection.getUsuarioID()” do serviço “connection”.

Aplicação foi implementada seguindo os padrões de desenvolvimento do Angular, com o auxílio da ferramenta Angular CLI que disponibiliza funções para a criação do projeto, criação de componentes, e até mesmo um servidor para execução da aplicação em ambiente de desenvolvimento.

5 CONCLUSÃO

A realização deste trabalho teve como objetivo o desenvolvimento de um sistema que atenda as necessidades de uma empresa no gerenciamento do plano de carreira. De forma que o funcionário possa informar de maneira fácil e prática as qualificações obtidas e consultar as qualificações necessárias para outros cargos. E para a empresa é uma ferramenta que gerencie o processo de avaliação, que possa se adequar ao fluxo de avaliação de diferentes empresas.

Utilizando uma API feita em Java com a utilização do *framework* Spring que ajuda no desenvolvimento de aplicações, juntamente com o Maven para gerenciar as dependências do projeto, fazendo com que, uma vez configurados corretamente, o desenvolvimento transcorra de forma adequada, com as funcionalidades da API implementadas em poucas linhas de código. Foram encontrados problemas com configuração dos módulos do Spring referentes à compatibilidade entre eles, após reformular o arquivo pom.xml do Maven os problemas foram solucionados.

Para a aplicação foi utilizando JavaScript com Angular e a biblioteca de componentes do Angular Material, permitindo a criação de uma SPA sem a necessidade de recarregar toda a página para apresentar os dados, propiciando ao usuário do sistema uma usabilidade diferenciada das páginas convencionais, como *leiaute* interativo. Uma dificuldade no desenvolvimento em Angular é o gerenciamento das requisições assíncronas, por vezes as requisições tiveram que ser encadeadas para garantir a integridade dos dados apresentados.

A aplicação desenvolvida atende os principais problemas verificados durante o levantamento de requisitos, como a falta de resposta ao funcionário e a necessidade de vários sistemas para prover as informações necessárias para a avaliação do atendimento a requisições para determinado cargo ou função na empresa.

Como trabalhos futuros, outros recursos podem ser implementados para a utilização em produção da aplicação, como configurações da empresa, mais informações para o usuário, recursos de segurança para evitar ataques ao sistema e a implementação de relatórios sobre os usuários e as qualificações. Esses relatórios serviriam para a empresa ter acesso às informações estratégicas para aplicação de cursos de formação ou contratação de novos funcionários com as qualificações menos atendidas no momento na empresa, por exemplo.

REFERÊNCIAS

AGUSTÍN, José Luis Herrero. **Model-driven web applications**. In: Science and Information Conference, 2015, p. 954-964.

ANGULAR MATERIAL. **What is angular material?** Disponível em: <<https://material.angularjs.org/latest/>>. Acesso em: 17 abr. 2017.

CZARNECKI Krzysztof; HELSEN, Simon; EISENECKER Ulrich W. Staged configuration through specialization and multilevel configuration of feature models. **Software process: improvement and practice**, v.10, n, 2, p.143-169, 2005.

FRATERNALI, Piero; ROSSI, Gustavo; SÁNCHEZ-FIGUEROA, Fernando. Rich Internet Applications. **IEEE Computer Society**. May/june 2010, p. 9-12.

INTELECTUS. **Gestão estratégica: gestão de carreira**. Disponível em: <<http://intelectusconsultoria.com.br/servicos/gestao-de-carreira/>>. Acesso em: 20 mar. 2017.

MATERIAL DESIGN. **Material design: introduction**. Disponível em: <<https://material.io/guidelines/#>>. Acesso em: 17 abr. 2017.

MELIÁ, Santiago; GÓMEZ, Jaime; PÉREZ, Sandy; DIAZ, Oscar. **A model-driven development for GWT-based Rich Internet Applications with OOH4RIA**. IEEE Conference (ICWE'08), USA, 2008, p.13-23.

PANG, Zhen; WEN, Fuan; PAN, Xiwei; LU, Cen. **Migration model for rich internet applications based on PureMVC framework**. In: 2010 International Conference on Computer Design and Applications (ICCD 2010), IEEE, v. 5, p. V5-340-V5-343.

TELMO, Aline. **Plano de carreira nas organizações**. Disponível em: <<http://www.rhportal.com.br/artigos-rh/plano-de-carreira-nas-organizaes/>>. Acesso em: 20 mar. 2017.

W3SCHOOLS. **AngularJS introduction**. Disponível em: <https://www.w3schools.com/angular/angular_intro.asp>. Acesso em: 17 abr. 2017.

SPRING. **Spring framework overview**. Disponível em: <<https://docs.spring.io/spring/docs/current/spring-framework-reference/overview.html#overview>>. Acesso em: 02 nov. 2017.