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

RAFAEL BATISTELLA

SISTEMA WEB PARA CONTROLE DE CRÉDITOS EM UM RESTAURANTE UNIVERSITÁRIO

MONOGRAFIA DE ESPECIALIZAÇÃO

PATO BRANCO
2017

RAFAEL BATISTELLA

SISTEMA WEB PARA CONTROLE DE CRÉDITOS EM UM RESTAURANTE UNIVERSITÁRIO

Monografia de especialização apresentada na disciplina de Metodologia da Pesquisa, do Curso de Especialização em Tecnologia Java, do Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Especialista.

Orientador: Profa. Beatriz Terezinha Borsoi

PATO BRANCO
2017

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

UTFPR

TERMO DE APROVAÇÃO

SISTEMA PARA CONTROLE DE CRÉDITOS EM UM RESTAURANTE UNIVERSITÁRIO

por

RAFAEL BATISTELLA

Este trabalho de conclusão de curso foi apresentado em 01 de dezembro de 2017, como requisito parcial para a obtenção do título de Especialista em Tecnologia Java. Após a apresentação o candidato foi arguido pela banca examinadora composta pelos professores Andreia Scariot Beulke e Vinicius Pegorini. Em seguida foi realizada a deliberação pela banca examinadora que considerou o trabalho aprovado.

Beatriz Terezinha Borsoi
Prof. Orientador (UTFPR)

Andreia Scariot Beulke
Banca (UTFPR)

Vinicius Pegorini
Banca (UTFPR)

Robison Cris Brito
Coordenador da IV Especialização em Tecnologia Java

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

RESUMO

BATISTELLA, Rafael. Sistema *web* para controle de créditos em um restaurante universitário. 2017. 54f. Monografia (Trabalho de especialização) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2017.

É comum que instituições de ensino, especialmente superior, ofereçam alimentação aos seus discentes. Essas refeições podem ser usufruídas por docentes e técnicos administrativos. Essas refeições são, geralmente, subsidiadas para os discentes e oferecidas em ambientes próprios denominados restaurantes universitários. É comum que alunos utilizem os restaurantes universitários diariamente. E, assim, o uso de cartões ou alguma forma que permita comprar créditos para refeição e debitá-los à medida que eles são utilizados pode ser bastante útil para usuários frequentes desse tipo de serviço. Visando prover uma maneira de o usuário de um restaurante universitário adquirir créditos e descontá-los quando fizer as refeições, um sistema *web* responsivo foi desenvolvido como resultado deste trabalho. Evitando, assim, que o usuário dispenda muito tempo em fila, comum nesse tipo de ambiente, para pagar a sua refeição. Um sistema pode auxiliar, ainda, para reduzir problemas de troco para os usuários e os operadores desses restaurantes. O sistema desenvolvido é para *web* e foi implementado utilizado a Linguagem Java, juntamente com as tecnologias Intellij IDEA, Spring Data, Spring Security, Spring Boot, Angular e o banco de dados PostgreSQL.

Palavras-chave: Spring Data. Spring Security. Spring Boot. Angular.

ABSTRACT

BATISTELLA, Rafael. System for control of credits in a university restaurant. 2017. 54f. Monografia (Trabalho de especialização) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2017.

It is common that educational institutions, especially universities, provide feed for their students. These meals are usually subsidized for students and offered in their own environments called university restaurants. Students, usually, use these restaurants daily. And they use cards or something similar those allow buying food credits and debit them as they are used. This system can be useful for frequent users of this type of service. In order to provide a way for the user of a university restaurant to acquire credits and discount them when meals are made, a responsive web system was developed as a result of this work. This prevents user must wait in queues, common in this type of environment, to pay for the meal, and to avoid problems of change for the users and operators of these restaurants. The developed system is web and it was implemented using the Java Language, with other technologies such as Intellij IDEA, Spring Data, Spring Security, Spring Boot, Angular and PostgreSQL.

Keywords: Spring Data. Spring Security. Spring Boot. Angular.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGENS DE CÓDIGOS

LISTA DE SIGLAS

CRUD	*Create-Read-Update-Delete*
CSS	*Cascading Style Sheets*
DAO	*Data Access Objects*
FTP	*File Transfer Protocol*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
IDE	*Integrated Development Environment*
JSON	*JavaScript Object Notatio*n
LDAP	*Lightweight Directory Access Protocol*
MVCC	*Multi Version Concurrency Control*
RA	Registro Acadêmico
REST	*Representational State Transfer*
RF	Requisito Funcional
RNF	Requisito não Funcional
SGBD	Sistema Gerenciador de Banco de Dados
SGBDOR	Sistema Gerenciador de Banco de Dados Objeto Relacional
SIAPE	Sistema Integrado de Administração de Recursos Humanos
SPA	*Single Page Application*
SQL	*Structured Query Language*
UML	*Unified Modeling Language*
URL	*Uniform Resource Locator*
WAL	*Write Ahead Logs*

SUMÁRIO

1 INTRODUÇÃO

É comum que as universidades ofereçam a opção de alimentação em restaurante próprios ou terceirizados para os seus alunos e servidores (funcionários e professores). Esses ambientes, geralmente denominados restaurantes universitários, podem oferecer refeições como café da manhã, almoço e jantar, sendo almoço, seguido de jantar, as refeições mais frequentemente oferecidas.

Os preços dos restaurantes universitários são, em geral, subsidiados, fazendo com que o valor cobrado seja menor que de refeições similares em outros estabelecimentos do gênero. Em geral esses restaurantes oferecem refeições com valor reduzido ou subsidiado. A identificação dos usuários que podem usufruir do valor subsidiado, por exemplo, pode ser realizada por meio do crachá individual ou outra maneira que permita assegurar que a pessoa está vinculada à universidade.

O amplo uso de sistemas computacionais para auxiliar em atividades gerenciais e de controle de dados cadastrais, permite afirmar, sem muita margem de erro, que todas as universidades possuem algum sistema computacional para manutenção de cadastro de alunos e de servidores. Os dados contidos nesses cadastros podem ser utilizados para auxiliar na identificação para, por exemplo, acesso a ambientes, realizar empréstimos em bibliotecas, fornecimento de dados pessoais como para o setor de recursos humanos e para utilizar outros serviços oferecidos no ambiente universitário.

Considerando que nas universidades, especialmente as de maior porte, o número de alunos e servidores pode ser bastante grande e que um percentual considerável dessas pessoas realizam refeições no ambiente da universidade, em determinados horários as filas de espera podem ser grandes. Uma forma de reduzir o tempo de fila é agilizar o pagamento da refeição. Ao invés de o usuário pagar no momento de realização de cada refeição, ele poderia adquirir créditos, sendo descontado o valor correspondente da refeição à medida que elas são realizadas. A aquisição de créditos poderia ser realizada por meio de fichas, mas o uso de um sistema computacional facilitaria outros controles, como o número máximo de refeições diárias permitidas e o turno que elas podem ser realizadas, por exemplo. E para os usuários, um controle de saldo e de uso dos créditos.

A agilidade de atendimento atribuída com aquisição de créditos está fundamentada em vários aspectos, como: o usuário pode adquirir os créditos em outros momentos que não o que estaria para realizar a refeição; créditos podem ser adquiridos para diversas refeições (um mês ou um semestre, por exemplo); evitar-se-ia manusear dinheiro no momento de realizar a

refeição; e, além de contribuir para reduzir o problema de troco, que geralmente ocorre nesses locais.

Considerando esse contexto, por meio da realização deste trabalho é proposto um sistema que tem como funcionalidade principal o controle de créditos de refeições realizadas em um restaurante universitário. O sistema é *web* responsivo e, assim, poderá ser utilizado em dispositivos móveis. O sistema permitirá que o usuário acompanhe mais facilmente o histórico da sua conta, ou seja, dados dos créditos adquiridos, dos débitos efetuados e o saldo da conta.

2 FERRAMENTAS, TECNOLOGIAS E PROCEDIMENTOS

Este capítulo apresenta as ferramentas e as tecnologias utilizadas na modelagem e na implementação do aplicativo desenvolvido.

2.1 FERRAMENTAS E TECNOLOGIAS

O Quadro 1 apresenta as ferramentas e tecnologias utilizadas.

Ferramenta / Tecnologia	**Versão**	**Disponível em**	**Aplicação**
Astah Community	7.1	http://astah.net/editions/community	Documentação da modelagem baseada na *Unified Modeling Language* (UML). Ferramenta utilizada para a modelagem do diagrama de casos de uso
Intellij IDEA	2017.1.2	https://www.jetbrains.com/idea/	Ambiente de desenvolvimento de software.
Linguagem Java	JDK 1.8	https://www.oracle.com/br/index.html	Linguagem para desenvolvimento da aplicação.
HTML	5.1	https://www.w3.org/TR/html51/	Linguagem de marcação de textos para desenvolvimento da interface da aplicação.
CSS	2	https://www.w3.org/Style/CSS/	Para estilização das página do projeto.
PgAdmin	4 v1.4	https://www.pgadmin.org/	Gerenciador do banco de dados PostgreSQL. Permite escrever instruções *Structured Query Language* (SQL) simples e complexas.
PostgreSQL	9.6.2	https://www.postgresql.org/	PostgreSQL é um Sistema Gerenciador de Banco de Dados Objeto Relacional (SGBDOR).
Spring Data	Kay-M2	http://projects.spring.io/spring-data/	Efetuar o mapeamento objeto-relacional e a persistência de dados.
Spring Security	5.0.0	https://projects.spring.io/spring-security/	Estrutura que se concentra em fornecer autenticação e autorização para aplicativos Java.
Spring Boot	1.5.3	https://projects.spring.io/spring-boot/	Facilitar o processo de configuração e publicação das aplicações.
Angular	2	https://angular.io/	*Framework* utilizado no desenvolvimento da interface da aplicação.

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

2.1.1 Astah* Community

Astah* Commuity é uma ferramenta para modelagem de sistemas com suporte para a *Unified Modeling Language* na versão 2. Além da modelagem dos diagramas, a ferramenta permite a impressão e a exportação das imagens desses diagramas (ambos com a marca d'água da ferramenta). A ferramenta Astah é apresentada em três versões (ASTAH, 2017):

a) Astah* Community – versão gratuita com suporte aos diagramas básicos da UML.

b) Astah* UML – suporte aos diagramas da UML a mapas mentais. Mapas mentais são diagramas conceituais ou de domínio para representar ideias e a visão geral de um sistema, por exemplo.

c) Astah* Professional – com suporte para os diagramas da UML, diagramas de entidades e relacionamentos, diagrama de fluxo de dados e diagramas de processo, fluxogramas e mapas mentais.

A Figura 2 apresenta a tela inicial da ferramenta Astah* com o objetivo de apresentar a sua interface principal e as partes que a compõe.

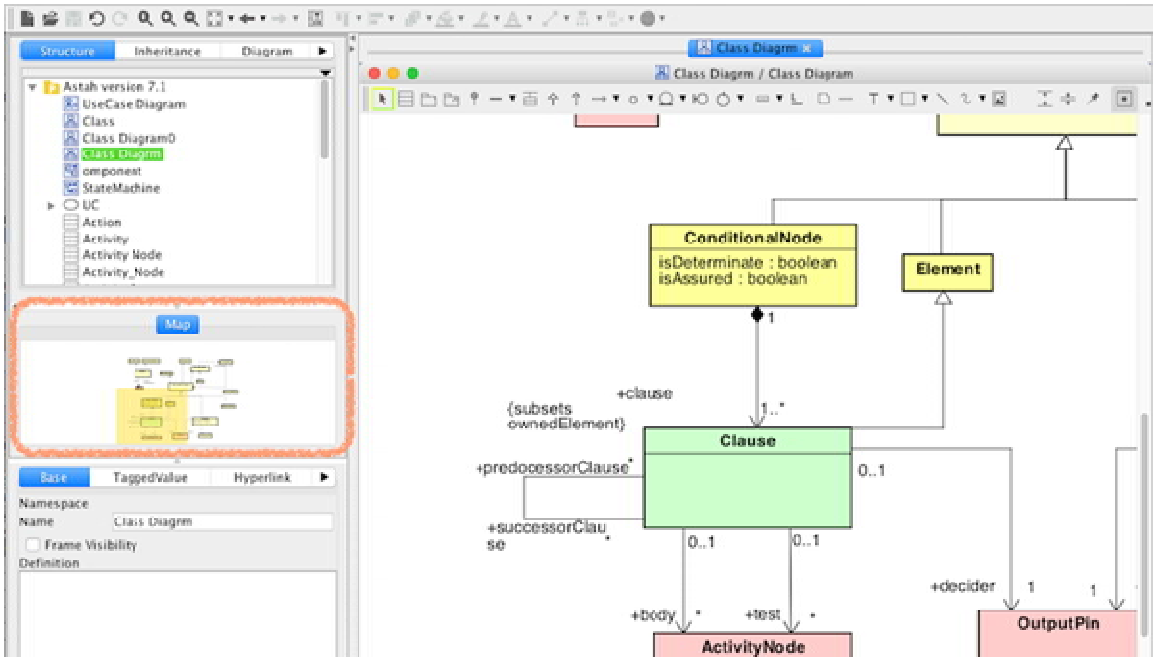

Figura 1 – Ferramenta de modelagem Astah* Community

2.1.2 Intellij IDEA

Intellij IDEA é um ambiente de desenvolvimento de software multilinguagens e compreende um *Integrated Development Environment* (IDE) e *plugins*. Por meio de *plugins*, a IDE Intellij IDEA pode ser usada para desenvolver software em várias linguagens além de Java, como Python, Kotlin, Ruby e PHP (INTELLIJ IDEA, 2017). A Figura 3 apresenta a tela principal da IDE Intellij IDEA.

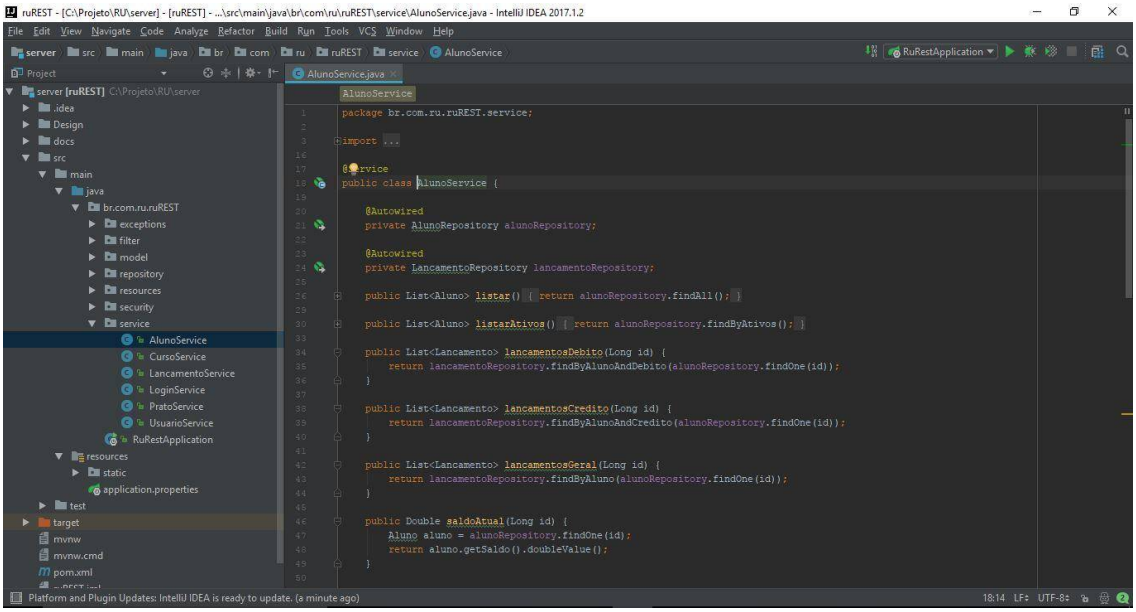

Figura 2 – IDE da ferramenta Intellij IDEA

A seguir são apresentadas algumas funções da ferramenta Intellij IDEA:

a) Barra de ferramentas – por meio das opções dispostas nesta barra é possível fazer o controle de formatação, execução, depuração e navegação no código. Nessa barra, mais à direita, é possível alternar entre as duas perspectivas dessa IDE: depuração e desenvolvimento.

b) Project explorer – permite visualizar o projeto de vários pontos de vista e execute as tarefas, como criar novos itens (diretórios, arquivos, classes, etc.), abrir arquivos no editor e navegar para o fragmento de código de interesse. A maioria das funções nesta janela da ferramenta é acessada como comandos de menu de contexto no painel de conteúdo e atalhos associados.

c) Área de edição – a área de edição contém todos os documentos abertos. É nessa área que o código é criado e editado.

2.1.3 Linguagem Java

A plataforma ou o ambiente de programação Java permite desenvolver aplicativos utilizando qualquer uma das linguagens criadas para a plataforma Java. Uma vantagem dessa plataforma é a de não estar vinculada a um único sistema operacional ou *hardware*, pois seu código gerado executa por meio de uma máquina virtual, denominada *Java Virtual Machine* (JVM), que pode ser emulada em qualquer sistema operacional que suporte à linguagem C++.

Java é uma linguagem orientada a objetos, sendo assim, a maior parte dos elementos de um programa Java são objetos (DEITEL, DEITEL, 2005). Como exceção cita-se os tipos básicos, como o int e o *float*. O código é organizado em classes, que podem estabelecer relacionamentos de herança simples entre si.

Chamadas a funções de acesso remoto (*sockets*) e os protocolos Internet mais comuns (*HyperText Transfer Protocol* (HTTP), *File Transfer Protocol* (FTP), Telnet, etc.) são suportadas em Java, de forma que a implementação de programas baseados em arquiteturas cliente/servidor é facilitada.

Java provê o gerenciamento de memória por meio de *garbage collection* (coleta de lixo). Sua função é a de verificar a memória periodicamente, liberando automaticamente os blocos que não estão sendo utilizados. Esse procedimento pode deixar o sistema de software com execução mais demorada por manter uma *thread* paralela à execução do programa. Porém, esse procedimento evita problemas como referências perdidas e avisos de falta de memória quando há memória disponível na máquina.

2.1.4 HTML

HyperText Markup Language (HTML), que significa Linguagem de Marcação de Hipertexto, é uma linguagem utilizada na construção de páginas na *web*. Documentos HTML podem ser interpretados pelos navegadores *web*. HTML é uma linguagem baseada em marcação que define o quê e como será apresentado na página *web* (W3SCHOOLS.COM, 2017a).

2.1.5 CSS

O *Cascading Style Sheets* (CSS) é uma linguagem que determina a aparência (leiaute) de páginas *web*. O CSS é um mecanismo para adicionar estilo (cores, fontes, espaçamento,

etc.) a um documento *web*. Em vez de colocar a formatação dentro do documento, o CSS cria um *link* para uma página que contém os estilos. As formatações e as alterações de formatação realizadas neste arquivo são refletidas na página *web* que possui o referido arquivo vinculado. CSS tem uma sintaxe simples e utiliza uma série de palavras em inglês para especificar os nomes de diferentes estilos de propriedade de uma página (W3SCHOOLS.COM, 2017b).

Uma instrução CSS consiste em um seletor e um bloco de declaração. Cada declaração contém uma propriedade e um valor, separados por dois pontos (:) e cada declaração é separada por ponto e vírgula (;) (W3SCHOOLS.COM, 2017b).

2.1.6 pgAdmin 4

O pgAdmin é um software gráfico para administração do Sistema Gerenciador de Banco de Dados (SGBD) PostgreSQL disponível para Windows e Linux. A Figura 3 mostra a interface dessa ferramenta. É uma ferramenta desenvolvida pela equipe do PostgreSQL que traz um *help online* e a documentação do PostgreSQL, que é acessado por meio da tecla F1. Ao executar consultas *Structured Query Language* (*query tool*), a tecla F7 permite visualizar graficamente o resultado da consulta na aba Explain ou F5 para executar (PGADMIN, 2017).

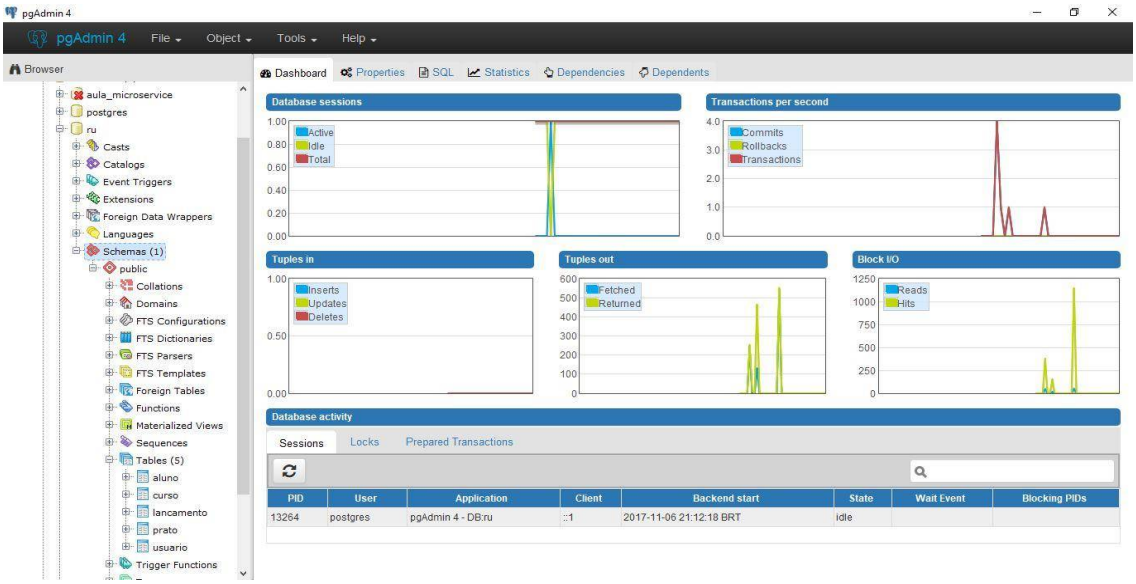

Figura 3 – Tela inicial pgAdmin 4

2.1.7 PostgreSQL

O PostgreSQL (POSTGRESQL, 2017) é um sistema de banco de dados objeto-relacional de código aberto. Pode ser executado na maioria dos sistemas operacionais, incluindo Linux, Mac OS X, Solaris e Windows. Possui suporte total para chaves estrangeiras, *joins, views, triggers* e *stored procedures.* As suas principais características são (SQL MAGAZINE, 2017):

a) Recuperação automática após falha de sistema por meio de *Write Ahead Logs* (WAL);

b) *Multi Version Concurrency Control* (MVCC) ou controle de concorrência de multi versão. Nesse mecanismo, processos de leitura não bloqueiam processos de escrita e vice-versa, reduzindo drasticamente a contenção entre transações concorrentes e paralisação parcial ou completa (*deadlock*) da aplicação;

c) *Commit, rollback* e *checkpoints* de transações;

d) *Triggers* e *stored procedures*;

e) *Foreign keys*;

f) *Backup on-line*;

g) Tamanho ilimitado de registro;

h) Índices em *cluster*. Cada tabela pode suportar um índice em *cluster*. Esse índice classifica fisicamente os dados na sequência especificada pelo índice. Um índice de *cluster* permite maior velocidade na recuperação de dados melhorando o desempenho geral do banco de dados.

2.1.8 Spring Data

O Spring Data é um projeto da Spring Source com proposta de unificar e facilitar o acesso às diferentes tecnologias de armazenamento de dados, como bancos de dados relacionais e os NoSQL. Independentemente da solução de armazenamento, as classes de "repositório", também conhecidas como *Data Access Objects* (*DAO*) normalmente disponibilizam operações *Create-Read-Update-Delete* (CRUD) para um determinado objeto de domínio, além de métodos de pesquisa e funcionalidades de ordenação e paginação (BOTELHO, 2017).

O Spring Data disponibiliza interfaces genéricas para esses aspectos (*CrudRepository*

e *PagingAndSortingRepository*), além de implementações específicas para cada banco de dados. Uma visão geral da arquitetura do Spring Data é apresentada na Figura 4 (TRELLE, 2013).

Figura 4 – Arquitetura do Spring Data

Fonte: Trelle (2013, p.s.n.).

2.1.9 Spring Security

Spring Security é uma estrutura de autenticação e de controle de acesso altamente personalizável. É o padrão de fato para garantir segurança para aplicativos baseados em Spring (CHHATPAR, 2006).

Com algumas poucas configurações é possível realizar autenticação via banco de dados, *Lightweight Directory Access Protocol* (LDAP) ou por memória. Spring Security também suporta várias integrações e permite criar as próprias integrações. Quanto à autorização, ele é bem flexível. Por meio das permissões atribuídas aos usuários autenticados, pode-se proteger as requisições *web* (como as telas do sistema, por exemplo), a invocação de um método e até a instância de um objeto (AFONSO, 2017a).

2.1.10 Spring Boot

O Spring Boot é um projeto da Spring que visa facilitar o processo de configuração e publicação de aplicações. Isso é conseguido pelo favorecimento da convenção sobre a configuração, indicando os módulos que se deseja utilizar (*web*, *template*, persistência, segurança, etc.) que serão reconhecidos e configurados pelo Spring Boot (AFONSO, 2017b).

Os módulos são escolhidos por meio dos *starters* que os inclui no pom.xml do projeto. Os *starters* são, basicamente, dependências que agrupam outras dependências. Esses grupos de dependências, representadas pelo *starter*, ajudam a manter o pom.xml mais organizado.

2.1.11 Angular

Angular é uma plataforma de aplicativos da *web front-end* baseada em código aberto em TypeScript liderada pela Angular Team no Google e por uma comunidade de indivíduos e

corporações. Angular é uma reescrita completa do AngularJS (GUEDES, 2017).

Angular é uma estrutura para criar aplicativos clientes em HTML e JavaScript ou TypeScript que compila JavaScript. Angular combina modelos declarativos, injeção de dependência, ferramentas e práticas recomendadas que são integradas para resolver desafios de desenvolvimento (GUEDES, 2017).

3 RESULTADOS

Este capítulo apresenta o resultado da realização do trabalho que é um aplicativo para controle de créditos para refeições em um restaurante universitário.

3.1 ESCOPO DO SISTEMA

O sistema permite o controle de créditos de refeições realizadas em um restaurante universitário. Essa é a funcionalidade essencial do sistema. O usuário compra créditos de refeições do restaurante universitário que são armazenados na sua conta. Esses créditos são debitados à medida que eles são utilizados. Os créditos podem ser resgatados se não forem utilizados. Nesse caso, o usuário do restaurante retira o valor monetário correspondente aos créditos. Essa operação é realizada pelo usuário do sistema com permissões de administrador.

O sistema possibilita a consulta da conta do usuário do restaurante e isso é realizado por meio de interface responsiva. Assim, o usuário pode realizar consultas utilizando um navegador *web* seja em um computador do tipo *desktop* ou *notebook* ou em dispositivos móveis.

O sistema terá dois perfis de acesso: administrador e usuário do restaurante. O usuário do restaurante poderá consultar apenas dados relacionados aos seus registros, como créditos, débitos (refeições realizadas) e saldo, além de consultar informações relacionadas ao prato do dia oferecido pelo restaurante. O administrador acessará as funcionalidades de cadastro de usuários do restaurante e do sistema, cursos, prato do dia, valor da refeição e a inclusão de créditos e os débitos no sistema que são realizados pelo usuário do restaurante.

3.2 MODELAGEM DO SISTEMA

O Quadro 2 apresenta os requisitos funcionais definidos para o sistema. RF significa Requisito Funcional.

RF	Requisito	Descrição
01	Manter cursos	Manter cadastro de cursos aos quais os usuários do restaurante estão vinculados. Curso é a denominação utilizada para o vínculo da pessoacom a universidade. Curso foi utilizado como nome padrão porque esse tipo de ambiente é mais frequente e tipicamente utilizado por alunos, mas pode incluir o cadastro de

		um departamento ou setor da universidade, para o caso de usuários não discentes.
02	Manter prato	Cadastro e manutenção de cadastro da composição da refeição que será ofertada no dia. Pratos indicam o cardápio do dia. Pratos são cadastrados e o marcado como prato do dia é o disponibilizado para visualização na tela inicial do sistema. Isso permite que o usuário do restaurante possa saber o que será servido do dia atual.
03	Manter usuários	Manutenção de cadastro de usuários do restaurante. Esses usuários podem adquirir créditos, debitá-los e resgatá-los, além de visualizar informações na tela principal de acesso ao sistema. Cada usuário terá acesso somente aos dados de histórico da sua respectiva conta.
04	Cadastrar valor das refeições	O cadastro do valor das refeições utilizado como base para desconto do valor de uma refeição.
05	Registrar refeições realizadas	Debitar a refeição do saldo de créditos. A leitura do código de barras do crachá fará o débito na conta do usuário. O sistema deve manter o histórico de data, horário e valor de cada débito realizado.
06	Adquirir créditos	O aluno adquire créditos (deposita valor de cujo saldo serão debitadas as refeições realizadas) e essa quantia é registrada na base de dados.
07	Resgatar valores de créditos	O usuário pode resgatar valores, ou seja, poderá reaver valores que havia em créditos. O atendente devolverá o dinheiro e o sistema atualizará o saldo. O registro do resgate de créditos ocorrerá pela leitura, por código de barras, do crachá do aluno. O sistema manterá armazenados a data, o horário e a quantidade de créditos resgatados.
08	Consultar histórico da conta	O aluno pode visualizar o saldo em créditos e o histórico dos débitos realizados na sua conta.
09	Debitar créditos	O débito do valor corresponde à refeição é realizado no momento que o usuário do restaurante realiza refeição.
10	Receber dinheiro	No momento que os créditos são adquiridos o usuário administrador receberia o valor monetário correspondente.
11	Devolver dinheiro	No momento que o usuário do restaurante resgatar créditos, o usuário administrador fará a devolução do valor monetário correspondente.

Quadro 2 – Requisitos funcionais

Os requisitos não funcionais são apresentados no Quadro 3. Nesse quadro RNF significa Requisito não Funcional.

RNF	Requisito	Descrição
01	Créditos	O saldo existente nas contas é financeiro e não representa refeições. Assim, quando o valor da refeição é atualizado, o saldo corresponderá à quantidade de refeições de acordo com o novo valor.
02	Consulta de dados da conta	A consulta do usuário na sua conta é realizada por um sistema *web* responsivo e fornecerá o histórico de uso, com débitos e créditos realizados e o saldo existente.
03	Identificação do usuário no débito da refeição	O sistema deverá apresentar o nome e a foto, se cadastrada, de quem está sendo debitado o valor correspondente à refeição.

Quadro 3 – Requisitos não funcionais

O diagrama de casos de uso, apresentado na Figura 5, contém os requisitos organizados na forma de casos de uso e vinculados aos dois perfis de acesso (atores) do sistema.

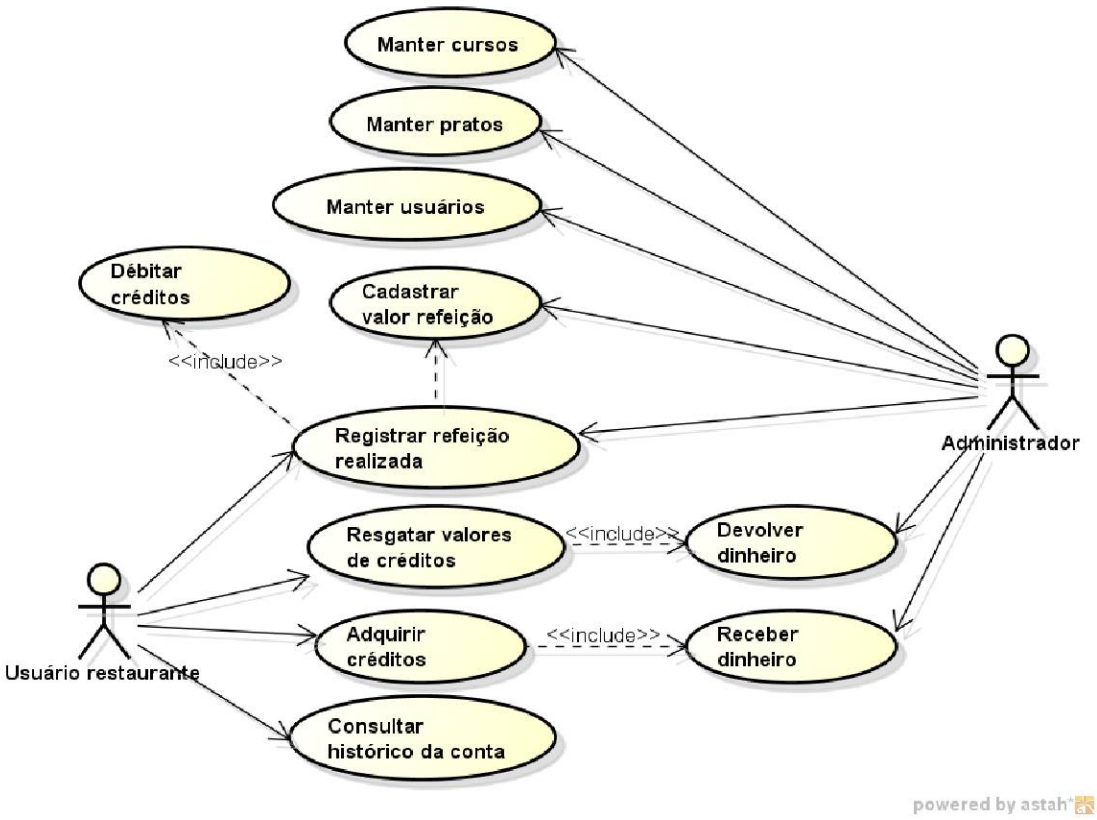

Figura 5 – Diagrama de casos de uso

O Quadro 4 apresenta a expansão do caso de uso registrar refeição realizada. Esse caso de uso ocorre no momento de realização de refeição pelo usuário do restaurante.

Caso de uso: Registrar refeição realizada. **Descrição**: Efetuar o débito da refeição realizada na conta do usuário do restaurante. Esse caso de uso inclui o caso de uso debitar créditos, fazendo com que o valor da refeição seja debitado da conta do usuário e essa operação seja registrada no histórico do respectivo usuário do restaurante. **Evento Iniciador**: Identificação do usuário do restaurante cujo débito será realizado. **Atores**: Administrador. **Pré-condição**: Haver crédito na conta do usuário do restaurante em valor suficiente para realizar o débito da refeição. **Sequência de Eventos**: 1. O usuário se identifica.
2. O sistema apresenta a identificação (nome e foto do usuário do restaurante). 3. O sistema realiza o débito na conta, atualizando o histórico da referida conta.

Pós-Condição: Dados do cadastro inseridos no banco de dados.

Quadro 4 – Caso de uso: registrar refeição realizada

O Quadro 5 apresenta a expansão do caso de uso resgatar valor de créditos. O usuário pode resgatar valores de créditos adquiridos e que não foram ou serão utilizados para realizar refeições.

Caso de uso: Resgatar valor de créditos. **Descrição**: Efetuar a devolução de dinheiro correspondente a créditos de refeição adquiridos. Esse caso de uso inclui o caso de uso devolver dinheiro que é fisicamente realizado pelo administrador do sistema. Em termos de operações, o sistema debita o valor e faz a inclusão no histórico do respectivo usuário do restaurante. **Evento Iniciador**: Identificação do usuário do restaurante que deseja realizar o resgate de créditos. **Atores**: Administrador. **Pré-condição**: Haver créditos na conta do usuário para serem resgatados. **Sequência de Eventos**: 1. Usuário do restaurante se identifica. 2. O sistema apresenta a identificação (nome e foto do usuário) e o valor em créditos existente na sua conta. 3. O administrador registra o valor a ser resgatado, de acordo com informação do usuário. O atendente entrega o valor para o usuário e confirma a entrega no sistema. 4. O sistema registra o resgate de valores no histórico da conta. **Pós-Condição**: Valor resgatado debitado da conta.

Nome do fluxo alternativo (extensão)	Descrição
3 Devolver dinheiro	3.1 Atendente entrega dinheiro para usuário. 3.2 Atendente registra valor entregue a ser debitado na conta do usuário

Quadro 5 – Caso de uso: resgatar valor de créditos

O Quadro 6 apresenta a expansão do caso de uso adquirir créditos. O usuário pode adquirir créditos para serem utilizados na realização das refeições.

Caso de uso: Adquirir créditos. **Descrição**: Efetuar a aquisição (compra) de créditos de refeição. Esse caso de uso inclui o caso de uso Receber dinheiro que ocorre quando o administrador recebe o valor monetário referente aos créditos adquiridos e o sistema faz a inclusão do valor correspondente em créditos na conta do usuário. **Evento Iniciador**: Identificação do usuário do restaurante que está adquirindo os créditos. **Atores**: Administrador. **Pré-condição**: Não há. **Sequência de Eventos**: 1. O usuário se identifica. 2. O sistema apresenta a identificação (nome e foto do usuário). 3. O administrador registra o valor adquirido em créditos, de acordo com o valor fornecido pelo usuário. 4. O sistema registra o valor no histórico da conta do referido usuário do restaurante.

Pós-Condição: Valor de créditos registrado na conta.	
Nome do fluxo alternativo (extensão)	Descrição
3. Receber dinheiro	3.1 Atendente recebe valor fornecido pelo usuário.
	3.2 Atendente registra valor recebido no sistema.

Quadro 6 – Caso de uso: adquirir créditos

O Quadro 7 apresenta a expansão do caso de uso realizar consulta de saldo. O usuário do restaurante pode consultar o saldo e o histórico de débitos e créditos na sua conta. O sistema mantém registro de todas as operações realizadas indicando valor e data de realização.

Caso de uso: Consultar histórico da conta. **Descrição**: Consultar dados de histórico da conta. **Evento Iniciador**: Identificação do usuário pelo Registro Acadêmico (RA) ou pelo Sistema Integrado de Administração de Recursos Humanos (SIAPE) e pela senha previamente indicada pelo aluno. **Atores**: Usuário do restaurante. **Pré-condição**: Não há. **Sequência de Eventos**: 1. O usuário seleciona a opção de consulta. 2. O sistema apresenta os dados de acordo com a consulta. **Pós-Condição**: Dados de histórico de conta apresentados ao usuário.

Quadro 7 – Caso de uso: consultar histórico de conta

Os cadastros de usuários do sistema de restaurante (que são alunos e servidores), cursos e pratos são mantidos pelo administrador. Os Quadros 8 a 11 apresentam a expansão do caso de uso manter desses cadastros. Manter inclui as operações de inclusão, exclusão, consulta e alteração de dados de cadastros.

No Quadro 8 está a expansão da operação de inclusão de cadastros.

Caso de uso: Operação inclusão dos casos de uso manter cadastro. **Descrição**: Incluir dados cadastrais de usuários do sistema e do restaurante, pratos e cursos. **Evento Iniciador**: Acesso à opção de cadastro de usuários do restaurante. **Atores**: Administrador. **Pré-condição**: Não há. **Sequência de Eventos**: 1. O usuário seleciona a opção de cadastro de usuários do restaurante e insere os dados correspondentes a aluno, professor ou funcionários, denominados técnicos administrativos. Aluno possui o RA para individualizar o cadastro e servidores a matrícula SIAPE. Quando d cadastro de usuários do restaurante e do sistema também serão armazenados, nome, curso (coordenação ou departamento se professor ou técnico administrativo), fotografia, se é administrador e se o cadastro está habilitado. 2. O sistema insere os dados no banco de dados e informa que o cadastro foi realizado.

Pós-Condição: Dados de cadastro incluídos no banco de dados.	
Nome do fluxo alternativo (extensão)	Descrição
2. Dados obrigatórios não informados	2.1 Sistema informa que há campos obrigatórios não preenchidos os campos são indicados e o sistema aguarda confirmação de inclusão e realiza nova verificação dos campos.

Quadro 8 – Caso de uso manter: operação inclusão

No Quadro 9 está a expansão da operação de exclusão de cadastros.

Caso de uso: Operação exclusão dos casos de uso manter cadastro. **Descrição**: Excluir cadastro. Se é cadastro de usuários esse será desabilitado, no sentido que o respectivo usuário não mais terá acesso ao sistema, mas os dados serão mantidos (não serão excluídos) e podem ser acessados em consultas posteriores. **Evento Iniciador**: Acesso à opção de exclusão dos cadastros. **Atores**: Administrador. **Pré-condição**: Não há. **Sequência de Eventos**: 1. O usuário seleciona a opção de exclusão e identifica o registro a ser excluído. 2. O sistema exclui ou marca o cadastro como desabilitado (no caso de usuários) não permitindo que o usuário possa utilizar a conta de créditos no restaurante. **Pós-Condição**: Cadastro de usuário do restaurante desabilitado para uso de conta de créditos. Cadastro excluído, se for o caso, ou inativado.	
Nome do fluxo alternativo (extensão)	Descrição
2. Usuário do restaurante sendo desabilitado possui créditos	2.1 Sistema informa que há créditos na conta do usuário.

Quadro 9 – Caso de uso manter: operação exclusão

No Quadro 10 está a expansão da operação de alteração dos cadastros.

Caso de uso: Operação alteração dos casos de uso manter cadastro. **Descrição**: Alterar dados de cadastros. **Evento Iniciador**: Acesso à opção de alteração de dados cadastrais. **Atores**: Administrador. **Pré-condição**: Não há. **Sequência de Eventos**: 1. O usuário seleciona a opção de alteração e identifica o registro no qual quer realizar alterações. 2. O sistema apresenta os dados do cadastro selecionado. 3. O Usuário faz os ajustes nos dados. 4. O sistema salva os dados conforme alterações realizadas. **Pós-Condição**: Dados de cadastro atualizados.	
Nome do fluxo alternativo (extensão)	Descrição
4. Dados obrigatórios não informados	4.1 Sistema informa que há campos obrigatórios não preenchidos os campos são indicados e o sistema aguarda confirmação de inclusão e realiza nova

	verificação dos campos.

Quadro 10 – Caso de uso manter: operação editar

No Quadro 11 está a expansão da operação de consulta de dados de cadastros.

Caso de uso:
Operação consulta dos casos de uso manter cadastro.
Descrição:
Consultar dados de cadastros.
Evento Iniciador:
Acesso à opção de consulta de dados cadastrais.
Atores:
Administrador.
Pré-condição:
Não há.
Sequência de Eventos:
1. O usuário seleciona a opção para consulta de cadastro.
2. O sistema apresenta os dados.
Pós-Condição:
Dados apresentados.

Quadro 11 – Caso de uso manter: operação consultar

No Quadro 12 está a expansão do caso de uso cadastrar valor da refeição.

Caso de uso:
Cadastrar valor da refeição.
Descrição:
O valor da refeição cadastrado é o que será utilizado para debitar créditos na conta do usuário. Quando um novo valor é cadastrado, ele passa a ser utilizado para débito de créditos.
Evento Iniciador:
Acesso à opção de cadastro de valor da refeição.
Atores:
Administrador.
Pré-condição:
Não há.
Sequência de Eventos:
1. O usuário seleciona a opção para cadastrar um novo valor para refeição.
2. O sistema registra o novo valor e finaliza a validade do valor anterior.
Pós-Condição:
Dados de cadastro de valor de refeição incluído.

Quadro 12 – Caso de uso cadastrar valor da refeição

A Figura 6 apresenta o diagrama do banco de dados. A tabela principal do sistema é a de lançamentos na qual são registrados os créditos adquiridos e o valor que vai sendo debitado a cada refeição realizada. O cadastro do prato é realizado para apresentar o cardápio do dia.

A tabela de usuários do restaurante que também é usuário do sistema, mas não possui as permissões de administrador, é denominada aluno. Ressalta-se, porém, que é comum que servidores das Instituições de Ensino também possam utilizar um restaurante universitário.

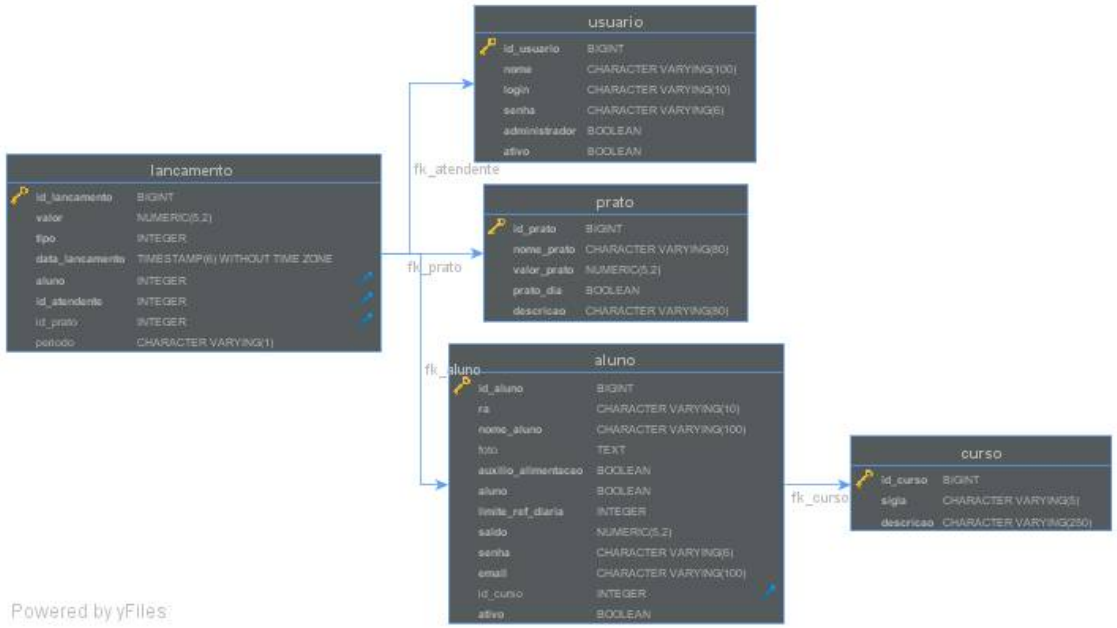

Figura 6 – Diagrama de entidades e relacionamentos do banco de dados

Há um cadastro de usuários (tabela usuários) para os dois perfis do sistema que são o administrador e o usuário do restaurante. O usuário do restaurante pode apenas realizar consulta de histórico dos seus dados e visualizar a descrição do prato do dia e o respectivo valor.

3.3 APRESENTAÇÃO DO SISTEMA

Na Figura 7 está a tela de autenticação para acesso ao sistema realizada por *login* e pela respectiva senha previamente cadastrados no banco de dados.

Figura 7 – Tela de login interno

Para autenticar-se no sistema, o usuário deverá informar seu *login* e sua senha. No caso de informações inválidas no momento de realizar a autenticação, será apresentada a seguinte mensagem: “Login ou senha incorretos”.

Na Figura 8 está a tela de *login* de usuário do restaurante. Para realizar o *login* é necessário informar o número do RA ou SIAPE e a senha. Se o usuário esqueceu a senha, ele pode clicar na opção “Esqueci minha senha”.

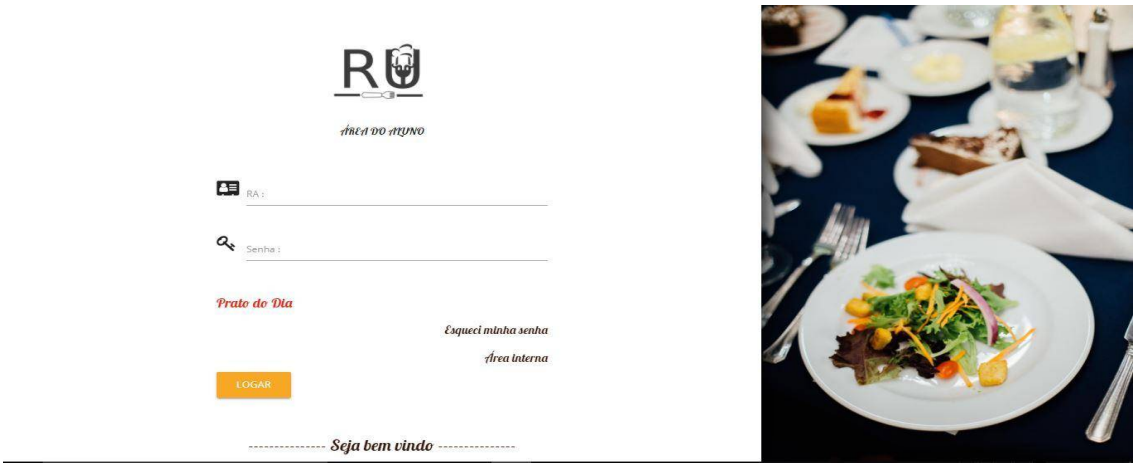

Figura 8 – Tela de login do aluno

Por meio da opção esqueci minha senha, o usuário deve informar seu RA e *email* para receber uma mensagem para poder cadastrar uma nova senha. A Figura 8 apresenta a tela para informação dos dados para recuperação da senha.

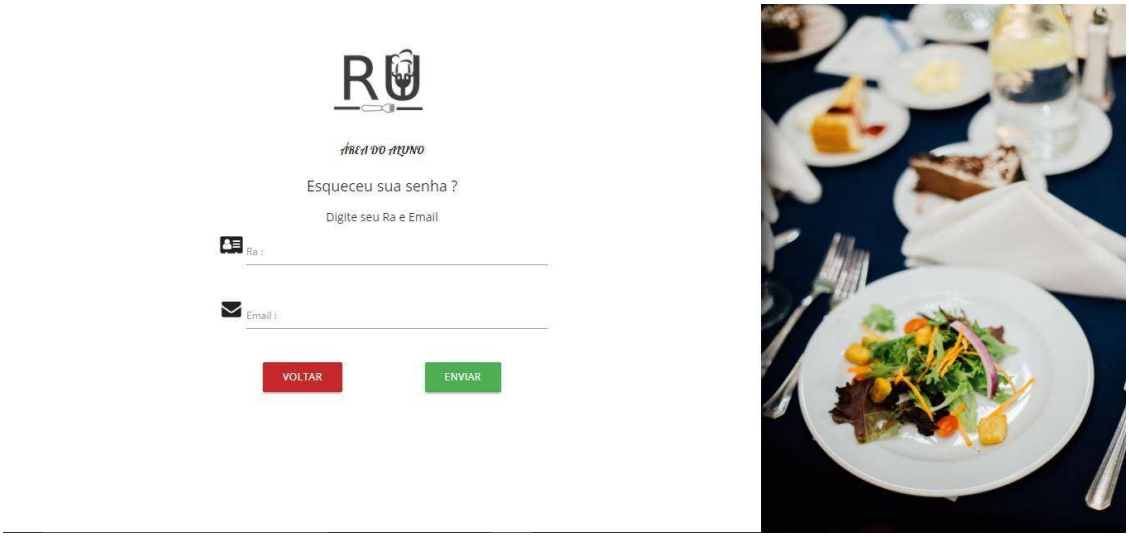

Figura 9 – Tela de Recuperação de Senha

Outra particularidade do sistema é o “Prato do Dia”. Ao clicar sobre a descrição é apresentada uma tela com o nome do prato, o dia que ele será servido, o seu valor e a descrição. Oferecendo, assim, a possibilidade de o usuário do restaurante saber o que será servido na refeição do dia. A Figura 10 apresenta a tela com as informações do prato, que se refere ao cardápio do dia.

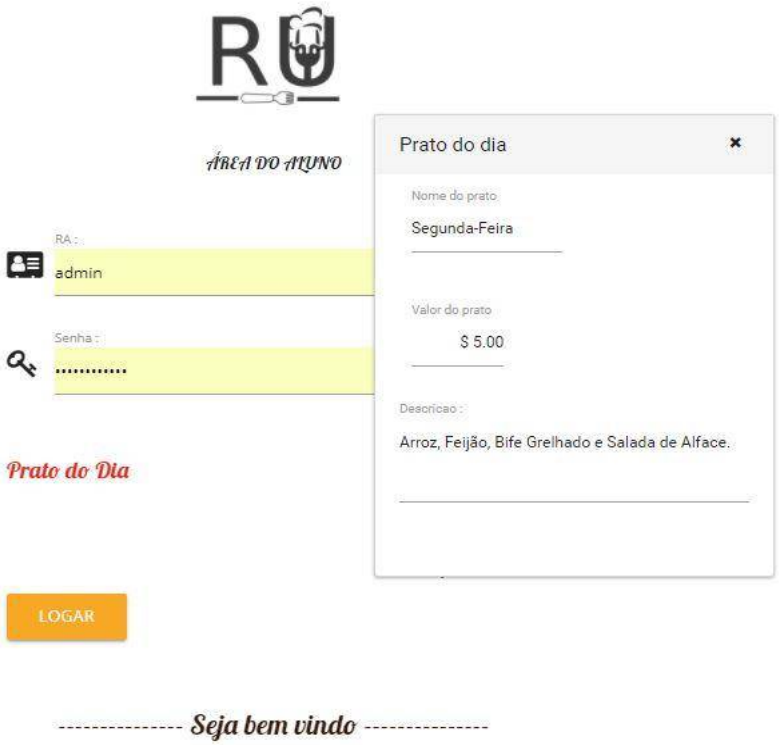

Figura 10 – Tela prato do dia

Após realizada a validação de acesso e todos os dados estarem de acordo com o seu cadastro no banco de dados, o sistema será aberto com a tela inicial. A Figura 11 apresenta a barra de menus do sistema.

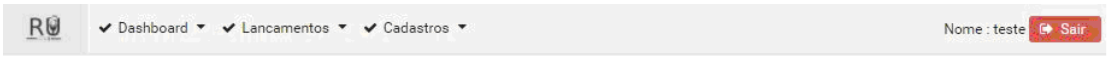

Figura 11 – Tela inicial do sistema interno

Na parte superior da tela da Figura 11 está a opção de “Sair” do sistema. Função essa que permite que o usuário se desconecte. Nesse mesmo local é apresentado o nome do usuário que está autenticado. No topo da tela estão, também, os menus de acesso às funções do sistema.

No menu “Cadastros”, o sistema exibe o cadastro de cursos, alunos, pratos e usuários.

Alunos são denominados como os usuários do sistema, sejam alunos ou servidores. E usuários são os que possuem acesso ao sistema e podem ter o perfil de administrador. Ao acessar a opção de cursos é aberta uma tela que é apresentada na Figura 12, com uma listagem de todos os cursos cadastrados no sistema. Ao ser clicado na linha de cabeçalho dos dados apresentados, os dados listados são ordenados crescente ou decrescente. Cliques sucessivos na mesma coluna alternam entre essas duas possibilidades de ordenação.

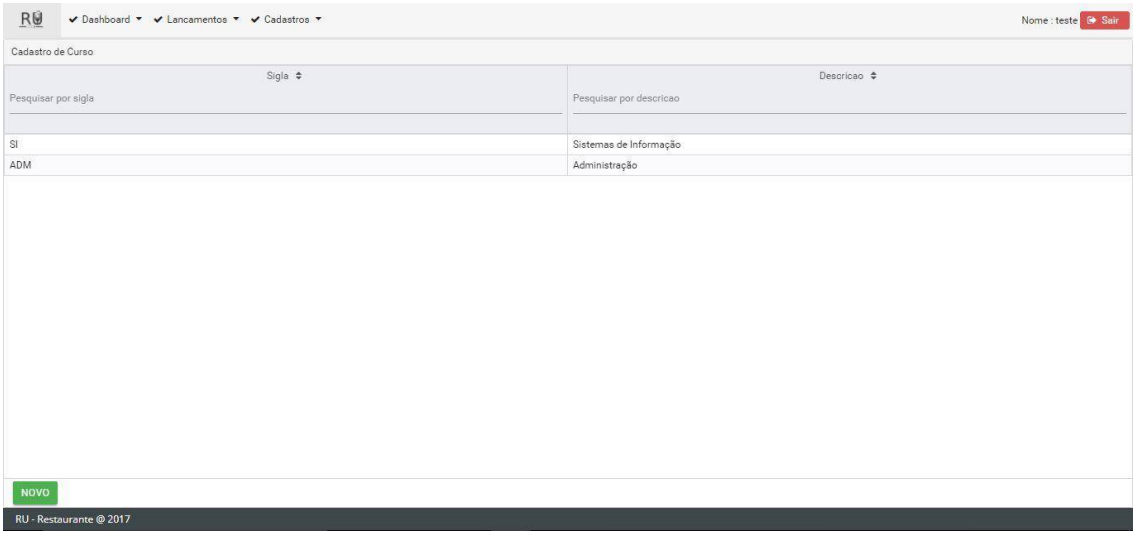

Figura 12 – Tela de listagem de cursos

Cada linha da tabela (como apresentado na Figura 12) representa um curso cadastrado com informações como: sigla e descrição. No canto inferior da tela, está a opção “Novo” para incluir novos cursos no sistema. Ao ser clicado nesse botão será exibido ao usuário um formulário para que seja digitada a sigla e a descrição do curso e também são apresentados os botões para as operações de salvar, excluir e cancelar. Na Figura 13 esses botões são apresentados em destaque.

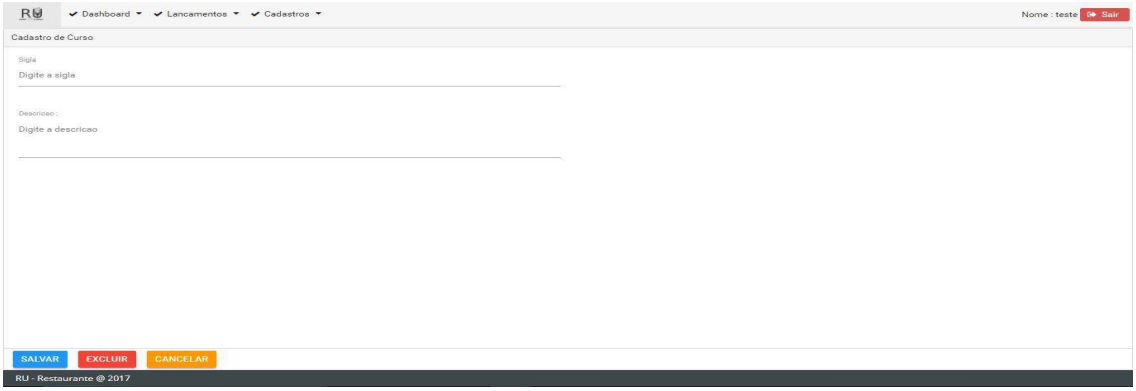

Figura 13 – Tela de cadastro de cursos

Ao acessar a opção de alunos no menu “Cadastros” é aberta uma tela com uma listagem de todos os alunos cadastrados no sistema. Essa tela é apresentada na Figura 14. Cada linha da tabela é o registro que representa um aluno cadastrado com informações como: nome, RA, se aluno, ou SIAPE, se servidor, curso e se ele está ativo. Para cadastrar um novo aluno é utilizado o mesmo procedimento para inserir um novo curso.

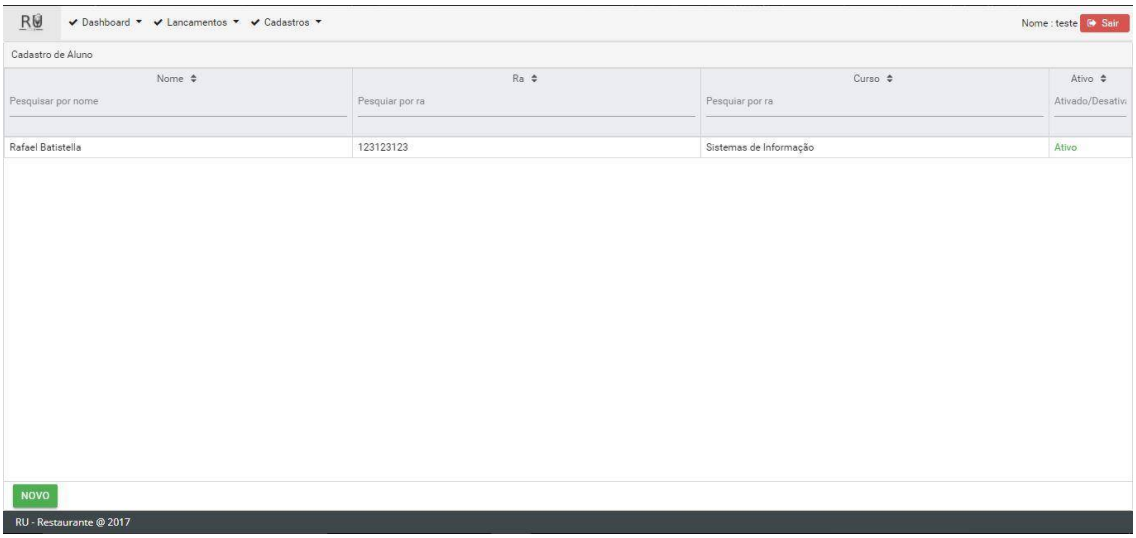

Figura 14 – Tela de listagem de alunos

Ao clicar sobre o cadastro do aluno, o usuário acessará respectivamente a tela de cadastro das informações do aluno, conforme Figuras 15 e 16.

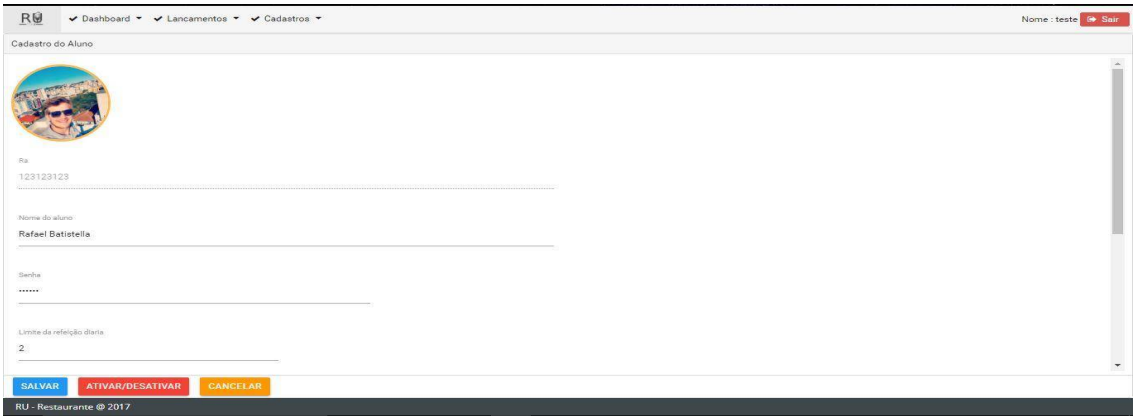

Figura 15 – Tela de cadastro de aluno

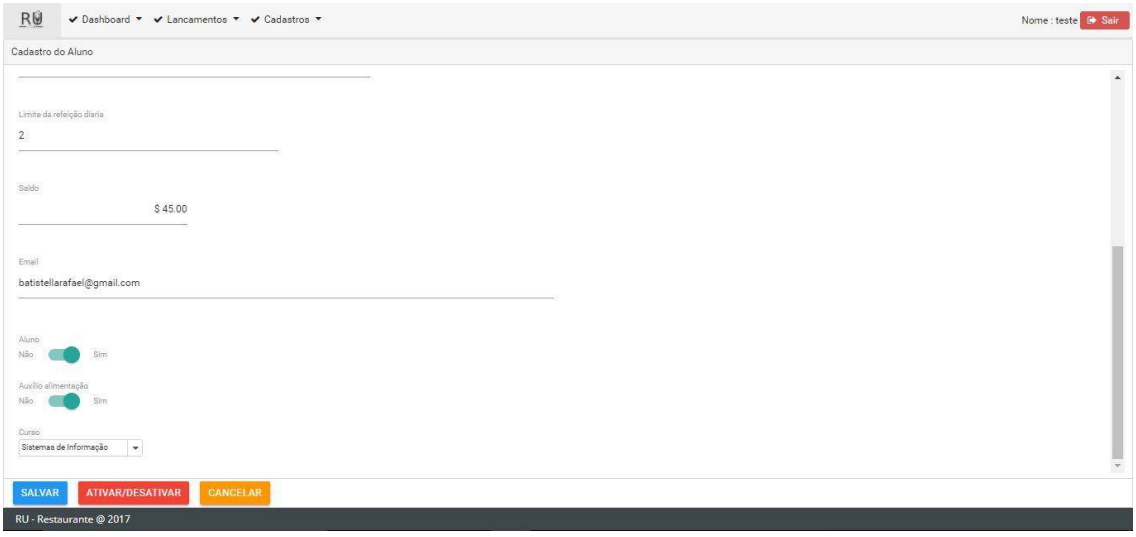

Figura 16 – Tela de cadastro de aluno

O cadastro de pratos, conforme apresenta a tela da Figura 17, permite incluir informações sobre os pratos oferecidos pelo restaurante e apresentar o prato do dia na tela de autenticação do sistema.

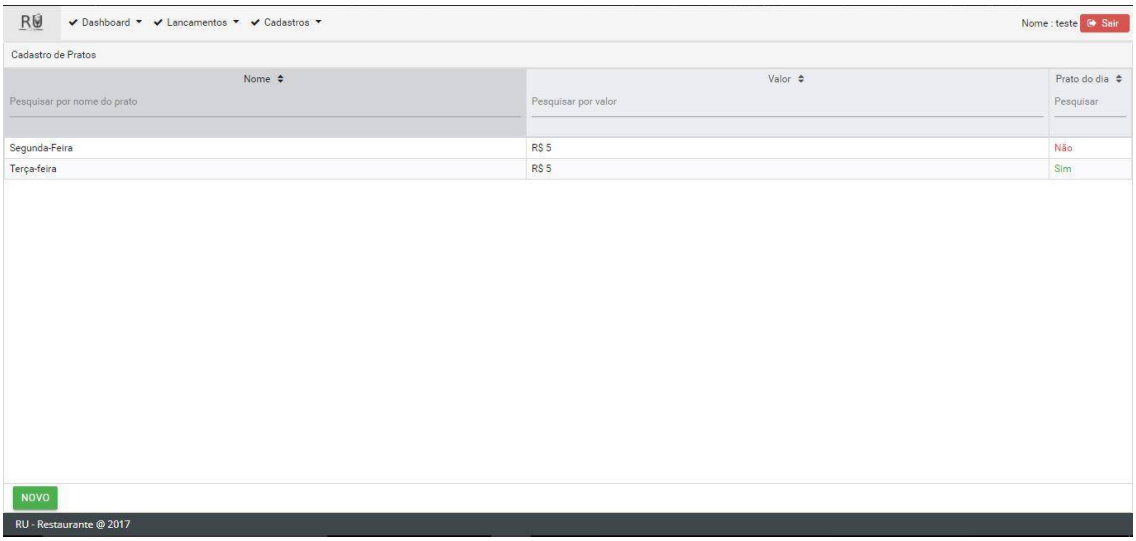

Figura 17 – Tela de listagem de pratos

Ao abrir a tela de cadastro de pratos, uma listagem é apresentada ao usuário com os campos nome, valor e prato do dia. Para realizar o cadastro de um novo prato, o usuário deve clicar no botão novo no canto inferior esquerdo da tela e será aberta a página de cadastro de um prato, conforme apresenta a Figura 18.

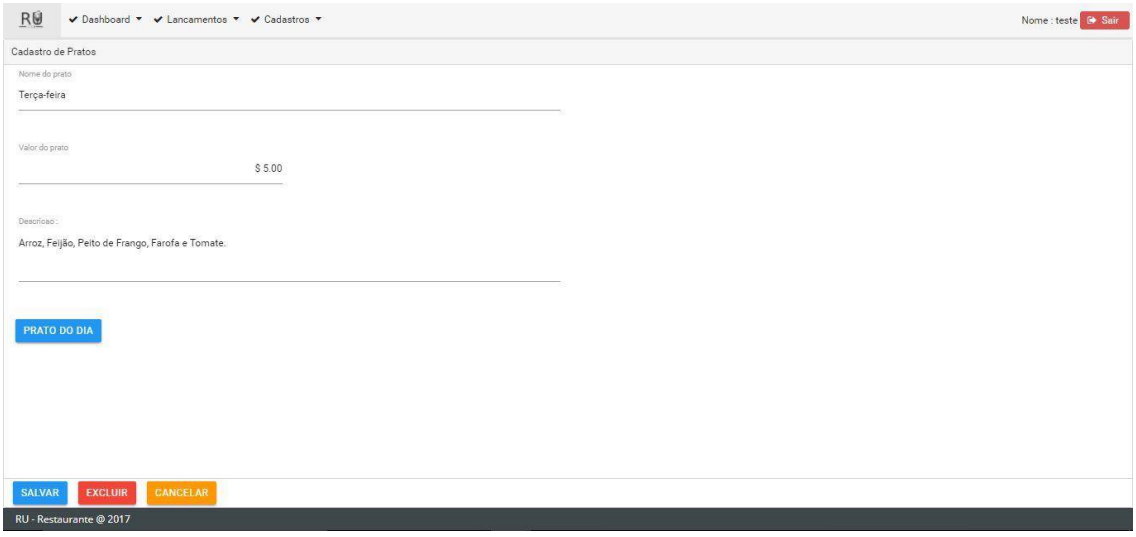

Figura 18 – Tela de cadastro de pratos

No formulário de cadastro de prato, o usuário inserirá o nome do prato, o valor e a descrição correspondentes, selecionando se é o prato do dia. O prato selecionado como o prato do dia será apresentado na tela inicial de interface do usuário do restaurante.

A Figura 19 apresenta a tela de cadastro de usuários do sistema.

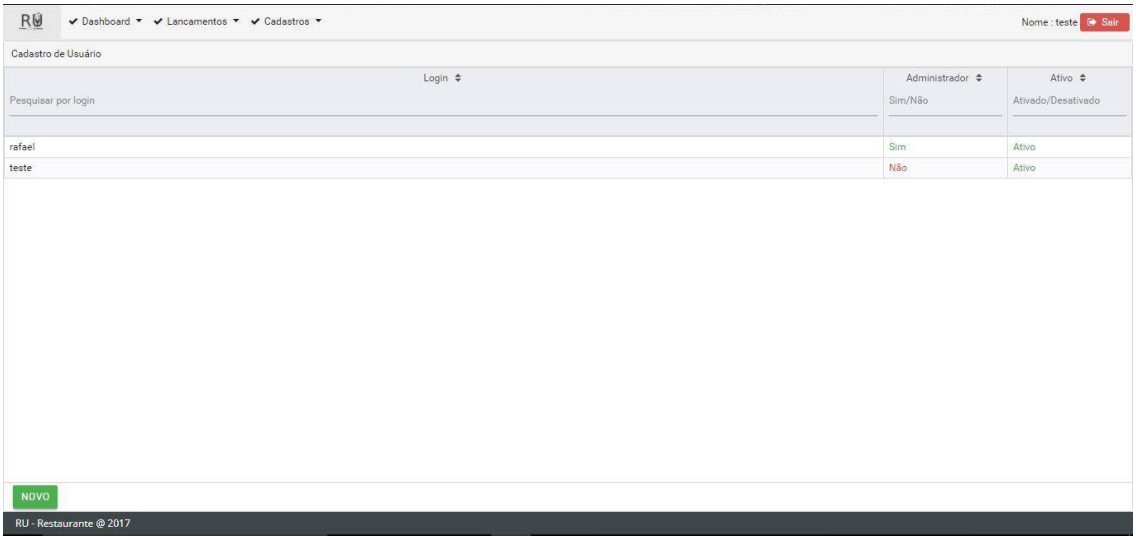

Figura 19 – Tela de listagem de usuários

Na listagem de usuários é apresentado o nome, se ele é administrador e se o cadastro está ativo. O cadastro de um novo usuário segue o padrão dos demais cadastros. Na Figura 20 é apresentada a tela de cadastro de usuários.

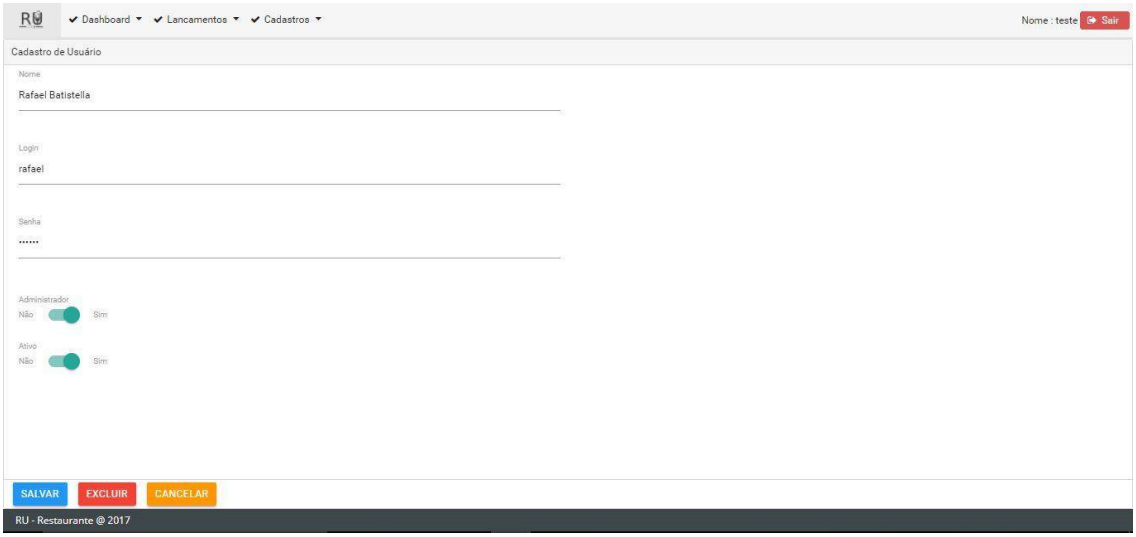

Figura 20 – Tela de cadastro de usuários

Para o cadastro de usuários do sistema é necessário informar nome, *login*, senha, se o usuário terá acesso administrador e se o usuário está ativo. É possível cancelar o cadastro, pressionando o botão “Cancelar”. Essa opção fechará o cadastro de usuários.

Na tela da Figura 21 é apresentado o menu lançamentos. Neste menu, o usuário tem a opção de cadastrar lançamentos e visualizar a listagem de lançamentos realizados.

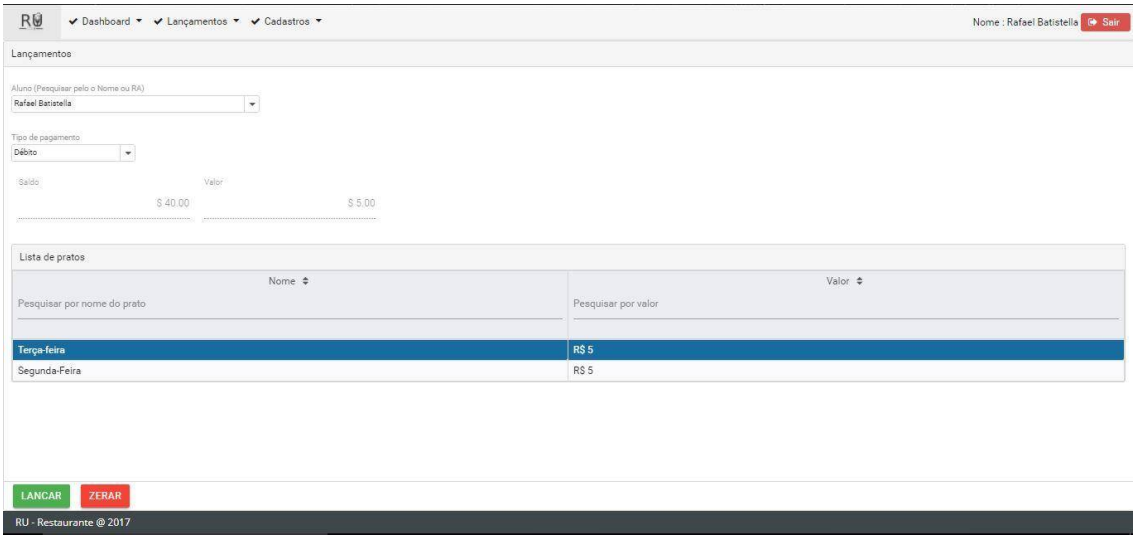

Figura 21 – Tela de lançamentos

A tela de lançamentos, apresentada na Figura 22, receberá as informações de qual usuário do restaurante está realizando a refeição e o tipo de pagamento que está sendo realizado dentro das opções de crédito, débito e devolução. É apresentado o valor de saldo disponível e ao lado deste campo, à direita da tela, é apresentado o valor do prato, conforme o

prato escolhido na listagem de pratos. Ao selecionar esses campos o usuário deverá clicar no botão lançar para que seja confirmado o lançamento. O valor diferente do prato pode ser utilizado para usuários com valores subsidiados ou sem pagamento, como é o caso dos alunos que recebem auxílio alimentação.

Após a realização do lançamento, o usuário poderá verificar a quantidade de lançamentos por meio do caminho: lançamentos --> lista de lançamentos. Nessa listagem é possível visualizar os campos de id, valor, tipo, prato, data, RA e atendente (usuário do sistema que realizou o lançamento). Como nas demais listagens do sistema, os campos podem ser ordenados por meio de um clique sobre o nome do campo.

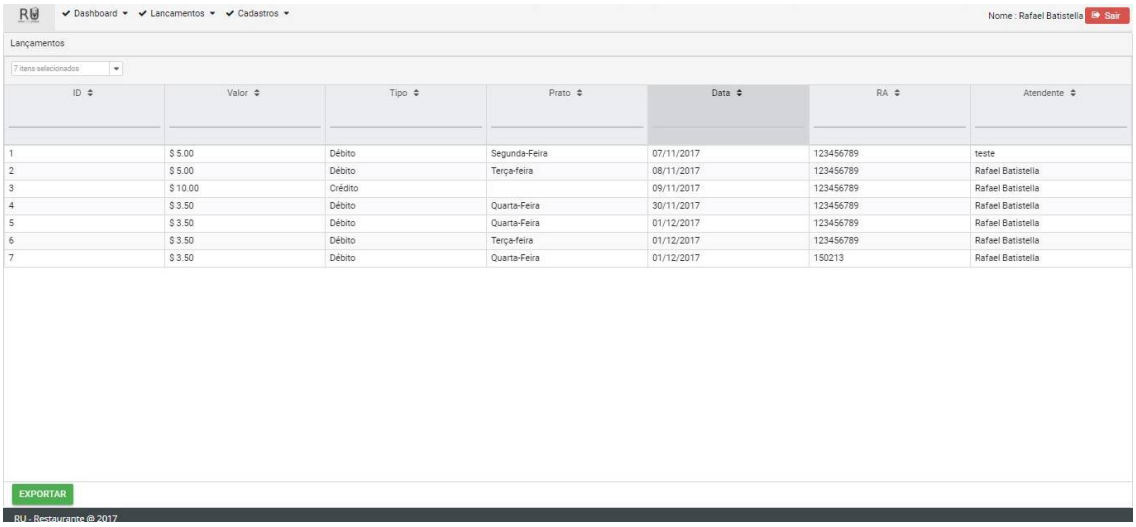

Figura 22 – Tela de listagem de lançamentos

Outra particularidade da tela de listagem de lançamentos é o botão “Exportar”. Ao clicar neste botão, os dados da tela serão exportados em extensão .CSV de *Comma-Separated Values*.

A Figura 23 apresenta o *dashboard* de informações. Nesse *dashboard*, o usuário do restaurante poderá visualizar as movimentações realizadas, os débitos e os créditos, por meio de gráficos, e, ainda, o saldo atual em sua conta. O usuário do restaurante também possui a possibilidade de alterar sua senha, conforme apresentado na área destacada da Figura 23.

Figura 23 – Tela de dashboard do usuário do restaurante

3.4 IMPLEMENTAÇÃO DO SISTEMA

Nesta seção são apresentados códigos com o objetivo de exemplificar a utilização da linguagem Java e suas tecnologias associadas para o desenvolvimento de aplicações *web* e o *framework* Angular2. O processo de autenticação do sistema, ou seja, a validação de um usuário com seu *login* e senha no sistema, é feita pela classe "*LoginComponent*" no *front-end* apresentada na Listagem 1.

```
import { Component, OnInit } from '@angular/core';
import { NgForm } from '@angular/forms';
import { Injectable } from '@angular/core';
import { GenericServico } from '../generic.servico';
import { Router, RouterModule } from '@angular/router';
import { Message, DialogModule} from 'primeng/primeng';
import * as CryptoJS from 'crypto-js';

@Component({
 selector: 'app-login',
 templateUrl: './login.component.html',
 styleUrls: ['./login.component.scss']
})
export class LoginComponent implements OnInit {
 usuario: any;
 msgs: Message[] = [];
 pageSenha:boolean=false;
 display: boolean = false;
 prato:any;
 constructor(private router: Router,private servico: GenericServico) {
 }

 ngOnInit() {
   this.prato = [];
```

```
   this.servico.listar("pratos/pratoDia")
   .toPromise()
   .then(dados => this.prato = dados)
 }

 logar(f:NgForm){
   localStorage.removeItem("jahsdioasdpp1sd");
   if((f.value.ra.length != 0) && (f.value.senha.length != 0)){

this.servico.post("login/externo",{ra:f.value.ra,senha:f.value.senha})
     .toPromise()
     .then(dados =>{
       var dbCript =
CryptoJS.AES.encrypt(JSON.stringify(dados.json()),'vns@2503');
       localStorage.setItem("jahsdioasdpp1sd",dbCript.toString());
       this.router.navigate(["principal"]);
     },erro => {
       if(erro.status === 403){
         this.msgs.push({severity:'error', summary:'Login', detail:'Ra ou
senha incorreto'});
       }else{
         this.msgs.push({severity:'error', summary:'Login',
detail:'Problemas ao logar'});
       }
     });
   }else{
     this.msgs.push({severity:'error', summary:'Login', detail:'Ra ou
senha vazio'});
   }
 }
 showDialog() {
       this.display = true;
   }

 enviarEmail(e:NgForm){
    if(e.value.email.length != 0){
        this.servico.post("esqueciSenha",{ra:e.value.ra,
email:e.value.email})
       .toPromise()
       .then(dados =>{
          this.msgs.push({severity:'success', summary:'Email',
detail:'Enviado com sucesso'});
          this.pageSenha=false;
       },erro => {
           this.msgs.push({severity:'error', summary:'Email', detail:'Não
foi possivel mandar email'});
       });
     }else{
       this.msgs.push({severity:'error', summary:'Email', detail:'Campo
de email vazio'});
     }
 }
}
```

Listagem 1 - Classe de validação de usuário

O código apresentado na Listagem 2 é a classe CadpratoComponent. Essa classe possui os métodos *Hypertext Transfer Protocol* (HTTP) para busca e envio de informações de e para o servidor.

```
import { Component, OnInit } from '@angular/core';
import { GenericAction } from '../generic.action';
import { GenericServico } from '../generic.servico';

@Component({
 selector: 'app-cadprato',
 templateUrl: './cadprato.component.html',
 styleUrls: ['./cadprato.component.scss']
})
export class CadpratoComponent extends GenericAction implements OnInit {

 constructor(private servico: GenericServico) {
   super();
   this.model = "pratos";
 }

 ngOnInit() {
   this.carregar();
 }

 carregar(){
   this.servico.listar(this.model)
   .toPromise()
   .then(dados => this.listagem = dados)
   .then(() =>{
     this.tabelaVisivel=true;
   });
 }

  salvar(){
    if(this.novo){
      this.selecionado.pratoDoDia=false;
      this.servico.post(this.model,this.selecionado)
      .subscribe(dados => {
          this.mensagenShow("success","Atualizado","Atualizado com
sucesso !");
          this.novo=false;
          this.carregar();
      },erro=>{
        this.mensagenShow("error","Problemas","Não foi possivel atualizar
!");
      });
    }else{

this.servico.put(this.model,this.selecionado.idPrato,this.selecionado)
      .subscribe(dados => {
          this.mensagenShow("success","Atualizado","Atualizado com
sucesso !");
          this.carregar();
      },erro =>{
        this.mensagenShow("error","Problemas","Não foi possivel atualizar
!");
      });
    }
  }
```

```
  pratoDia(opc){
     if(opc==1){

this.servico.put(this.model,this.selecionado.idPrato+"/pratoDia",this.sel
ecionado)
      .subscribe(dados => {
          this.mensagenShow("success","Atualizado","Atualizado com
sucesso !");
          this.novo=false;
          this.carregar();
      },erro=>{
        this.mensagenShow("error","Problemas","Não foi possivel atualizar
!");
      });
    }else{

this.servico.delete(this.model,this.selecionado.idPrato+"/pratoDia")
      .subscribe(dados => {
          this.mensagenShow("success","Atualizado","Atualizado com
sucesso !");
          this.novo=false;
          this.carregar();
      },erro=>{
        this.mensagenShow("error","Problemas","Não foi possivel atualizar
!");
      });
    }
  }

  excluir(opc){
     this.servico.delete(this.model,this.selecionado.idPrato)
    .subscribe(dados => {
        this.mensagenShow("success","Excluir","Excluido com sucesso !");
        this.carregar();
    },erro =>{
      this.mensagenShow("error","Problemas","Não foi possivel excluir
!");
    });
  }
}
```

Listagem 2 - Classe CadpratoComponent responsável pelas requisições de pratos

A classe PratoService é apresentado na Listagem 3. Essa classe é responsável pela lógica de negócio da aplicação, evitando, assim, que os *controllers* acessem diretamente a camada de dados (*Repository*). Essa é uma boa prática para desacoplamento de código.

```
package br.com.ru.ruREST.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.dao.EmptyResultDataAccessException;
import org.springframework.stereotype.Service;

import br.com.ru.ruREST.exceptions.EntityExistsException;
import br.com.ru.ruREST.exceptions.EntityNotFoundException;
```

```java
import br.com.ru.ruREST.model.Prato;
import br.com.ru.ruREST.repository.PratoRepository;

@Service
public class PratoService {

   @Autowired
   private PratoRepository pratoRepository;

   public List<Prato> listar() {
       return pratoRepository.findAll();
   }

   public Prato buscarPorId(Long id) {
       Prato prato = pratoRepository.findOne(id);

       if (prato == null) {
           throw new EntityNotFoundException("Prato não encontrado!");
       }

       return prato;
   }

   public Prato salvar(Prato prato) {
       Long count = verificaPratoExistente(prato.getNomePrato());
       System.out.println(count);
       if (count == 1) {
           throw new EntityExistsException("Prato já cadastrado!");
       }
       prato.setIdPrato(null);
       return pratoRepository.save(prato);
   }

   private Long verificaPratoExistente(String nomePrato) {
       return pratoRepository.countByNomePrato(nomePrato);
   }

   public void remover(Long id) {
       try {
           pratoRepository.delete(id);
       } catch (EmptyResultDataAccessException e) {
           throw new EntityNotFoundException("Prato não encontrado!");
       }
   }

   public void atualizar(Prato aluno) {
       verificarExistencia(aluno.getIdPrato());
       pratoRepository.save(aluno);
   }

   public void setaPratoDia(Long id) {
       Prato prato = pratoRepository.findOne(id);

       if (prato == null) {
           throw new EntityExistsException("Prato não encontrado");
           //estora exception
       }
       pratoRepository.findByPratoDoDia().stream().map((p) -> {
           p.setPratoDoDia(false);
           return p;
```

```
        }).forEach((p) -> {
            pratoRepository.save(p);
        });
        prato.setPratoDoDia(true);
        pratoRepository.save(prato);
    }

    public Prato buscaPratoDia() throws Exception {
        List<Prato> pratos = pratoRepository.findByPratoDoDia();
        if (pratos.isEmpty()) {
            throw new Exception("Nenhum prato encontrado!");
        }
        return pratos.get(0);
    }

    private void verificarExistencia(Long id) {
        buscarPorId(id);
    }
}
```

Listagem 3 - Classe PratoService

Aplicações desenvolvidas em Angular utilizam o conceito de *Single Page Application* (SPA). Portanto, para que seja possível que o usuário navegue entre as páginas do sistema foi necessário implementar um arquivo de configuração de rotas, no qual foi definida a *Uniform Resource Locator* (URL) das páginas e qual componente será responsável por ela, como pode ser visualizado na Listagem 4.

```
import { ModuleWithProviders } from '@angular/core';
import { Routes,RouterModule } from '@angular/router';
import { AppComponent } from './app.component';
import { LoginComponent } from './login/login.component';
import { PrincipalComponent } from './principal/principal.component';
import { CadcursoComponent } from './cadcurso/cadcurso.component';
import { CadalunoComponent } from './cadaluno/cadaluno.component';
import { CadusuarioComponent } from './cadusuario/cadusuario.component';
import { CadpratoComponent } from './cadprato/cadprato.component';
import { LancamentoComponent } from './lancamento/lancamento.component';
import { CadlancamentoComponent } from
'./cadlancamento/cadlancamento.component';
import { LogininternoComponent } from
'./logininterno/logininterno.component';

const APP_ROUTES : Routes = [
    {path : '', component:LoginComponent},
    {path : 'principal', component:PrincipalComponent},
    {path : 'cadalunos', component:CadalunoComponent},
    {path : 'cadusuarios', component:CadusuarioComponent},
    {path : 'cadpratos', component:CadpratoComponent},
    {path : 'lancamentos', component:LancamentoComponent},
    {path : 'cadlancamentos', component:CadlancamentoComponent},
    {path : 'interno', component:LogininternoComponent},
    {path : 'cadcursos', component:CadcursoComponent}
];

export const Routing : ModuleWithProviders =
```

```
RouterModule.forRoot(APP_ROUTES);
```

Listagem 4 - Arquivos responsável pelas rotas da aplicação

O desenvolvimento do sistema foi realizado em um serviço utilizando o Spring Boot. A aplicação denominada ruREST contém todas as entidades que persistem informações no banco de dados. Nesse serviço, também, estão implementadas as interfaces *service*, *repository* e *controllers* de cada entidade. Essa aplicação foi implementada para receber as requisições via *Representational State Transfer* (REST) do sistema administrativo em Angular.

Na Listagem 5 é possível visualizar a configuração no arquivo pom.xml da aplicação ruREST.

```
<?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>br.com.ru</groupId>
  <artifactId>ruREST</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>ruREST</name>
  <description>Restaurante Universitário - WS-REST</description>
  <parent>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-parent</artifactId>
     <version>1.5.6.RELEASE</version>
     <relativePath/> <!-- lookup parent from repository -->
  </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-data-jpa</artifactId>
     </dependency>
     <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-solr</artifactId>
     </dependency>
      <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
     </dependency>
     <dependency>
        <groupId>org.springframework.boot</groupId>
```

```
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
     <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-test</artifactId>
        <scope>test</scope>
    </dependency>
  </dependencies>

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

</project>
```

Listagem 5 – Configuração de arquivos no pom.xml da aplicação ruREST

A aplicação *back-end* ruRest, que é responsável por receber as requisições via REST do sistema, foi desenvolvida em Angular. Para isso, é necessário realizar algumas configurações para que o projeto receba esse tipo de requisição. A primeira configuração, exibida na Listagem 6, mostra a dependência incluída no arquivo pom.xml para tornar a aplicação *web*.

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.5.6.RELEASE</version>
  <relativePath/> <!-- lookup parent from repository -->
</parent>
```

Listagem 6 - Dependência spring boot starter web

Cada requisição recebida via REST acessa uma URL disponível no sistema. Para que isso seja possível foi necessário implementar *controllers* com algumas anotações como é

possível visualizar na Listagem 7. A anotação *@RestController* define que todos os métodos disponíveis retornarão as requisições via *JavaScript Object Notation* (JSON) e a anotação *@RequestMapping(*"/alunos") define a URL em que as requisições serão atendidas.

```
package br.com.ru.ruREST.controller;

import br.com.ru.ruREST.exceptions.EntityExistsException;
import br.com.ru.ruREST.exceptions.EntityNotFoundException;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import br.com.ru.ruREST.model.Aluno;
import br.com.ru.ruREST.model.Lancamento;
import br.com.ru.ruREST.service.AlunoService;
import java.net.URI;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import
org.springframework.web.servlet.support.ServletUriComponentsBuilder;

@RestController
@RequestMapping(value = "/alunos")
public class AlunoController {

   @Autowired
   private AlunoService alunoService;

   @RequestMapping(method = RequestMethod.GET)
   public ResponseEntity<List<Aluno>> listar() {
       return
ResponseEntity.status(HttpStatus.OK).body(alunoService.listar());
   }

   @RequestMapping(value ="/ativos" ,method = RequestMethod.GET)
   public ResponseEntity<List<Aluno>> listarAtivos() {
       return
ResponseEntity.status(HttpStatus.OK).body(alunoService.listarAtivos());
   }

   @RequestMapping(value = "/{id}", method = RequestMethod.GET)
   public ResponseEntity<?> buscar(@PathVariable Long id) {
       Aluno aluno = null;
       try {
           aluno = alunoService.buscarPorId(id);
       } catch (EntityNotFoundException e) {
           return ResponseEntity.notFound().build();
       }
       return ResponseEntity.status(HttpStatus.OK).body(aluno);
   }

   @RequestMapping(method = RequestMethod.POST)
   public ResponseEntity<Void> salvar(@RequestBody Aluno aluno) {
       try {
```

```
            aluno.setAtivo(true);
            aluno = alunoService.salvar(aluno);
        } catch (EntityExistsException e) {
            return ResponseEntity.status(HttpStatus.FORBIDDEN).build();
        }
        URI uri =
ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}").buildAndEx
pand(aluno.getIdAluno())
                .toUri();

        return ResponseEntity.created(uri).build();
    }

    @RequestMapping(value = "/{id}", method = RequestMethod.DELETE)
    public ResponseEntity<Void> deletar(@PathVariable Long id) {
        Aluno a = alunoService.buscarPorId(id);
        try {
            a.setAtivo(false);
            alunoService.atualizar(a);
        } catch (EntityNotFoundException e) {
            return ResponseEntity.notFound().build();
        }
        return ResponseEntity.status(HttpStatus.OK).build();
    }

    @RequestMapping(value = "/{id}/ativar", method = RequestMethod.PUT)
    public ResponseEntity<Void> ativar(@PathVariable Long id) {
        Aluno a = alunoService.buscarPorId(id);
        try {
            a.setAtivo(true);
            alunoService.atualizar(a);
        } catch (EntityNotFoundException e) {
            return ResponseEntity.notFound().build();
        }
        return ResponseEntity.status(HttpStatus.OK).build();
    }

    @RequestMapping(value = "/{id}", method = RequestMethod.PUT)
    public ResponseEntity<Void> alterar(@RequestBody Aluno aluno,
@PathVariable Long id) {
        aluno.setIdAluno(id);
        System.out.println(aluno.toString());
        try {
            alunoService.atualizar(aluno);
        } catch (EntityNotFoundException e) {
            return ResponseEntity.notFound().build();
        }
        return ResponseEntity.status(HttpStatus.OK).build();
    }

    @RequestMapping(value = "/{id}/lancamentosD", method =
RequestMethod.GET)
    public ResponseEntity<List<Lancamento>>
lancamentosDebito(@PathVariable Long id) {
        return
ResponseEntity.status(HttpStatus.OK).body(alunoService.lancamentosDebito(
id));
    }

    @RequestMapping(value = "/{id}/lancamentosC", method =
```

```
RequestMethod.GET)
   public ResponseEntity<List<Lancamento>>
lancamentosCredito(@PathVariable Long id) {
       return
ResponseEntity.status(HttpStatus.OK).body(alunoService.lancamentosCredito
(id));
   }

   @RequestMapping(value = "/{id}/lancamentos", method =
RequestMethod.GET)
   public ResponseEntity<List<Lancamento>> lancamentosGeral(@PathVariable
Long id) {
       return
ResponseEntity.status(HttpStatus.OK).body(alunoService.lancamentosGeral(i
d));
   }

   @RequestMapping(value = "/{id}/saldo", method = RequestMethod.GET)
   public ResponseEntity<Double> saldoAtual(@PathVariable Long id) {
       return
ResponseEntity.status(HttpStatus.OK).body(alunoService.saldoAtual(id));
   }

}
```

Listagem 7 - Classe AlunoController responsável pela requisição de alunos

Os dados retornados por meio dos *controllers*, estão armazenados em uma base de dados PostgreSQL, que também necessita de configurações para que possa ser acessada. Na Listagem 8, é possível visualizar a dependência incluída por meio do Maven, do *driver* necessário para conexão com a base de dados.

```
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <scope>runtime</scope>
</dependency>
```

Listagem 8 - Dependência do driver PostgreSQL

Na Listagem 9 estão as configurações incluídas no arquivo *application.properties* de acesso ao banco de dados.

```
spring.jpa.database=POSTGRESQL
spring.jpa.show-sql=true
spring.datasource.driver-class-name=org.postgresql.Driver
spring.datasource.url=jdbc:postgresql://localhost:5432/ru?autoReconnect=t
rue&useSSL=false
spring.datasource.username=postgres
spring.datasource.password=********
```

Listagem 9 - Propriedades para conexão com a base de dados

Para buscar as informações na base dados foi utilizado um *framework* do Spring, chamado Spring Data JPA. Para que fosse possível a sua utilização foi necessário incluir uma dependência por meio do Maven, conforme exibe a Listagem 10.

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

Listagem 10 - Dependência Spring Data

Nos repositórios referentes a cada entidade, foi necessário estender uma interface do Spring Data que possui métodos padrão para serem utilizados sem a necessidade de implementação, como, por exemplo, o método *save*. Na Listagem 11 é possível visualizar um *repository* estendendo essa interface e, também, contendo métodos personalizados de busca que não existem na interface *JpaRepository*. A utilização do Spring Data faz com que não seja necessário implementar consultas *Structured Query Language* (SQL), sendo necessário apenas descrever por meio do nome do método o que deseja buscar e quais filtros se deseja utilizar, como o método findByRaAndSenha, que busca os dados contendo o parâmetro no atributo RA ou senha.

```
package br.com.ru.ruREST.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import br.com.ru.ruREST.model.Aluno;
import java.util.List;

public interface AlunoRepository extends JpaRepository<Aluno, Long>{

   Long countByra(String ra);
   Aluno findByra(String ra);
   Aluno findByRaAndSenha(String ra, String senha);
   List<Aluno> findByAtivos();
}
```

Listagem 11 - AlunoRepository utilizando Spring Data para retornar informações da base de dados

Algumas interfaces do sistema necessitam de autenticação. Para isso, foi necessário implementar um arquivo de configuração e os tipos de requisição que podem ser acessados sem estar autenticado no sistema, como apresentado na Listagem 12.

```
package br.com.ru.ruREST.security;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.builders.Au
thenticationManagerBuilder;
import
```

```
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWe
bSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecur
ityConfigurerAdapter;
import org.springframework.security.web.AuthenticationEntryPoint;

@Configuration
@EnableWebSecurity
public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {

  @Autowired
  private AuthenticationEntryPoint authEntryPoint;

  @Override
  protected void configure(HttpSecurity http) throws Exception {
     http.csrf()
        .disable()
        .authorizeRequests()
        .anyRequest()
        .authenticated()
        .and()
        .httpBasic()
        .authenticationEntryPoint(authEntryPoint);
  }

  @Autowired
  public void configureGlobal(AuthenticationManagerBuilder auth) throws
Exception {

auth.inMemoryAuthentication().withUser("admin").password("admin").roles("
USER");
  }

}
```

Listagem 12 - Arquivo de configuração do Spring Security

Na Listagem 13 é possível visualizar o *template* principal.component que é responsável pelo *dashboard* do usuário do restaurante.

```
<cabecalho></cabecalho>
<p-growl [(value)]="mensagensCadastro" sticky="sticky"></p-growl>

<section class="container-conteudo animated fadeIn">
   <img *ngIf="!usuario.ra" class="centro-img"
src="/assets/img/logo2.png" alt="logo" />

   <div class="container-aluno" *ngIf="aluno.ra">

       <div class="caixa dados-aluno animated fadeInLeft">
            <div class="filtro-dados">
                <img *ngIf="aluno.foto" src="{{aluno.foto}}" class="img-
aluno animated fadeIn" width="100%" alt="FOTO" />
                <h4>Nome : {{aluno.nomeAluno}}</h4>
                <span>RA : {{aluno.ra}}</span> |
```

```
                <span>Curso : {{aluno.idCurso.descricao}}</span>
                <div class="saldo">
                    <div class="input-field">
                      <i class="material-icons fa fa-dollar"></i>
                      <input disabled class="saldo-input"  type="text"
currencyMask [(ngModel)]="aluno.saldo"  class="validate">
                    </div>
                  </div>
            </div>
            <ul>
                <li (click)="tblvisibe=1" style="background: rgba(64,
71, 74, 0.25);">
                  <i class="fa fa-user"></i>
                  <span>Usuario</span>
                </li>
                <li (click)="tblvisibe=2" style="background: rgba(64,
71, 74, 0.45);">
                  <i class="fa fa-bar-chart" aria-hidden="true"></i>
                  <span>Débitos</span>
                </li>
                <li (click)="tblvisibe=3" style="background: rgba(64,
71, 74, 0.65);">
                  <i class="fa fa-line-chart" aria-hidden="true"></i>
                  <span>Créditos</span>
                </li>
                <li (click)="tblvisibe=4" style="background: rgba(64,
71, 74, 0.85);">
                  <i class="fa fa-pie-chart" aria-hidden="true"></i>
                  <span>Todos</span>
                </li>
            </ul>
            <div class="horario">
              <i class="fa fa-clock-o"></i>
              <span>{{hora}}</span>
            </div>
      </div>
      <div class="caixa">
          <div class="container-dados-aluno animated fadeIn"
*ngIf="tblvisibe==1">

            <img class="centro-img" src="/assets/img/logo.png"
alt="logo" />

            <p-panel header="Alterar senha" [collapsed]="true"
toggleable="true" [style]="{'width':'100%','max-width':'300px'}">
                <div class="row">
                  <div class="input-field">
                   <i class="material-icons prefix fa fa-key"></i>
                    <input #novaSenha type="password" ngModel
name="senha" />
                    <label for="senha">Senha : </label>
                  </div>
                </div>
                <p-footer>
                    <button pButton label="Alterar"
(click)="alterarSenha(novaSenha)" ></button>
                </p-footer>
            </p-panel>
            <section class="container-conteudo">
              <div class="grafico-lancamento">
```

```
                    <canvas baseChart
                        [datasets]="barChartData"
                        [labels]="barChartLabels"
                        [options]="barChartOptions"
                        [legend]="barChartLegend"
                        [chartType]="barChartType">
                    </canvas>
                </div>
                <div class="grafico-lancamento">
                    <canvas baseChart
                      [data]="pieChartData"
                      [labels]="pieChartLabels"
                      [chartType]="pieChartType">
                    </canvas>
                </div>
              </section>

            </div>

            <div *ngIf="tblvisibe==2" class="animated fadeIn">
                <p-dataTable  [value]="lancamentoDebito"
[responsive]="true" styleClass="tabela" scrollable="true"
scrollHeight="calc(100vmin - 185px)">
                    <p-header>Lista de Débitos</p-header>
                    <p-column field="idPrato.nomePrato" header="Prato"
[filter]="true" filterPlaceholder="Pesquisar" [sortable]="true"></p-
column>
                    <p-column field="idPrato.valorPrato" header="Valor
Prato" [filter]="true" filterPlaceholder="Pesquisar" [sortable]="true">
                      <ng-template let-col let-car="rowData"
pTemplate="body">
                          <span>$ {{valorPrato(car)}}</span>
                      </ng-template>
                    </p-column>
                    <p-column field="dataLancamento" header="Data"
[filter]="true" filterPlaceholder="Pesquisar" [sortable]="true">
                      <ng-template let-col let-car="rowData"
pTemplate="body">
                          <span>{{formatarData(car[col.field])}}</span>
                      </ng-template>
                    </p-column>
                    <p-column field="idAtendente.nome" header="Atendente"
[filter]="true" filterPlaceholder="Pesquisar" [sortable]="true"></p-
column>
                    <p-column field="valor" header="Valor" [filter]="true"
filterPlaceholder="Pesquisar" [sortable]="true">
                      <ng-template let-col let-car="rowData"
pTemplate="body">
                          <input type="text"
style="border:none;padding:0px;margin:0px;height: auto;" currencyMask
[(ngModel)]="car[col.field]" name="valor" />
                      </ng-template>
                    </p-column>
                </p-dataTable>
            </div>

            <div *ngIf="tblvisibe==3" class="animated fadeIn">
                <p-dataTable  [value]="lancamentoCredito"
[responsive]="true" styleClass="tabela" scrollable="true"
scrollHeight="calc(100vmin - 185px)">
```

```
                    <p-header>Lista de Créditos</p-header>
                   <p-column field="dataLancamento" header="Data"
[filter]="true" filterPlaceholder="Pesquisar" [sortable]="true">
                     <ng-template let-col let-car="rowData"
pTemplate="body">
                         <span>{{formatarData(car[col.field])}}</span>
                     </ng-template>
                   </p-column>
                   <p-column field="idAtendente.nome" header="Atendente"
[filter]="true" filterPlaceholder="Pesquisar" [sortable]="true"></p-
column>
                   <p-column field="valor" header="Valor" [filter]="true"
filterPlaceholder="Pesquisar" [sortable]="true">
                     <ng-template let-col let-car="rowData"
pTemplate="body">
                          <input type="text"
style="border:none;padding:0px;margin:0px;height: auto;" currencyMask
[(ngModel)]="car[col.field]" name="valor" />
                     </ng-template>
                   </p-column>
               </p-dataTable>
           </div>

            <div *ngIf="tblvisibe==4" class="animated fadeIn">
               <p-dataTable  [value]="lancamentos" [responsive]="true"
styleClass="tabela" scrollable="true" scrollHeight="calc(100vmin -
185px)">
                   <p-header>Lista de Todos os Lancamentos</p-header>

                   <p-column field="idPrato.nomePrato" header="Prato"
[filter]="true" filterPlaceholder="Pesquisar" [sortable]="true"></p-
column>

                   <p-column field="idPrato.valorPrato" header="Valor
Prato" [filter]="true" currencyMask filterPlaceholder="Pesquisar"
[sortable]="true">
                     <ng-template let-col let-car="rowData"
pTemplate="body">
                          <span>$ {{valorPrato(car)}}</span>
                     </ng-template>
                   </p-column>

                   <p-column field="dataLancamento" header="Data"
[filter]="true" filterPlaceholder="Pesquisar" [sortable]="true">
                     <ng-template let-col let-car="rowData"
pTemplate="body">
                         <span>{{formatarData(car[col.field])}}</span>
                     </ng-template>
                   </p-column>

                   <p-column field="idAtendente.nome" header="Atendente"
[filter]="true" filterPlaceholder="Pesquisar" [sortable]="true"></p-
column>

                   <p-column field="valor" header="Valor" [filter]="true"
filterPlaceholder="Pesquisar" [sortable]="true">
                     <ng-template let-col let-car="rowData"
pTemplate="body">
                          <input type="text"
style="border:none;padding:0px;margin:0px;height: auto;" currencyMask
```

```
[(ngModel)]="car[col.field]" name="valor" />
                    </ng-template>
                </p-column>

                <p-column field="tipo" header="Tipo"
filterPlaceholder="Sim/Não" [filter]="true" [style]="{'width':'150px'}"
[sortable]="true">
                    <ng-template let-col let-car="rowData"
pTemplate="body">
                        <span *ngIf="car[col.field] == 1"
style="color:#4CAF50;">Crédito</span>
                        <span *ngIf="car[col.field] == 2"
style="color:#f44336;">Débito</span>
                        <span *ngIf="car[col.field] == 3"
style="color:blue;">Devolução</span>
                    </ng-template>
                </p-column>
            </p-dataTable>
        </div>
    </div>
</div>

</section>
```

Listagem 13 - Arquivo principal.component responsável pelo dashboard do aluno

4 CONSIDERAÇÕES FINAIS

O objetivo deste trabalho foi realizar o desenvolvimento de um sistema *web* que possibilita o controle de crédito para restaurante universitário. Diante das tecnologias disponíveis para desenvolvimento *web*, entre as utilizadas no desenvolvimento do sistema, destacam-se Angular e Spring. A junção dessas tecnologias possibilitou o desenvolvimento de um sistema de fácil manutenção devido à estrutura implementada.

Visando facilidade, agilidade e segurança foi desenvolvida uma interface visando simplicidade e intuitividade, no sentido de que o usuário saiba o que está fazendo, o que efetivamente pretende fazer e tendo claro os resultados das suas ações.

Durante o processo de desenvolvimento foram encontradas algumas dificuldades devido à falta de experiência com algumas das tecnologias utilizadas. Para superar esses desafios foram consultadas bibliografias, tutoriais e fóruns para buscar informações e experiências vivenciadas por outros desenvolvedores, além dos conhecimentos adquiridos em sala de aula. Assim, a prática do desenvolvimento deste trabalho proporcionou conhecimento em novas tecnologias, como Angular.

As maiores dificuldades enfrentadas no desenvolvimento do trabalho ocorreram na configuração inicial e na estruturação dos projetos. Inicialmente foi criado o projeto *back-end* em Java utilizando a arquitetura do Apache Maven e configurado o arquivo de dependências. Posteriormente ocorreu a parametrização do *framework* Spring com Spring Security. Destaca-se ainda o conhecimento necessário na linguagem Java, JavaScript e CSS.

Como trabalho futuro, pode-se citar a implementação de um *web service* que busque diretamente as informações dos alunos e servidores no banco de dados da universidade. E, ainda, a possibilidade de realizar a leitura do código de barras do crachá do aluno ou servidor para maior praticidade na hora de realizar a refeição.

REFERÊNCIAS

AFONSO, Alexandre. **O que é Spring Security?** 2017a. Disponível em: <http://blog.algaworks.com/spring-security/>. Acesso em: 20 nove. 2017.

AFONSO, Alexandre. **O que é Spring Boot?** 2017b. Disponível em: <http://blog.algaworks.com/spring-boot/>. Acesso em: 20 nove. 2017.

ASTAH. Disponível em: <http://astah.net>. Acesso em: 10 nov. 2017.

BOTELHO, Cristiano. **Conhecendo Spring Data JPA**. 2017. Disponível em: <http://igti.com.br/blog/conhecendo-spring-data-jpa/>. Acesso em: 28 out. 2017.

CHHATPAR, Arun. **Apache Geronimo e o Spring Framework**. **Parte 1: metodologia de desenvolvimento**. 2006. Disponível em <http://www.ibm.com/developerworks/br/library/os-ag-springframe1/authors.html>. Acesso em: 20 nove. 2017.

DEITEL, Paul; DEITEL, Harvey. **Java como programar**. Bookman, 2005.

GUEDES, Thiago. **Crie aplicações com Angular**: o novo framework da Google. Casa do Código. Disponível em: <https://books.google.com.br/books?id=GN7QDgAAQBAJ>. Acesso em: 19 nov. 2017.

INTELLIG IDEA. Disponível em: <https://www.jetbrains.com/idea/>. Acesso em 18 nov. 2017.

PGADMIN. Disponível em: <https://www.pgadmin.org/>. Acesso em: 19 nov. 2017.

POSTGRESQL. Disponível em: <https://www.postgresql.org/?node=4>. Acesso em: 05 nov. 2017.

SQL MAGAZINE. **SQL developer**. Disponível em: <http://www.sqlmagazine.com.br/artigos/postgre/01_Caracteristicas.asp>. Acesso em: 18 out. 2017.

TRELLE, Tobias. **Spring Data: a solução mais geral para persistência?** 2013. Disponível em: <https://www.infoq.com/br/articles/spring-data-intro>. Acesso em: 18 out. 2017.

W3SCHOOLS.COM. **HTML5 tutoria**l. 2017a. Disponível em: <https://www.w3schools.com/html/>. Acesso em: 19 nov. 2017.