

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

CASSIO MARIOTT

**APLICATIVO MÓVEL DE AUXÍLIO NO ATENDIMENTO DE BARES,  
LANCHONETES E RESTAURANTES**

MONOGRAFIA DE ESPECIALIZAÇÃO

PATO BRANCO  
2017

CASSIO MARIOTT

**APLICATIVO MÓVEL DE AUXILIO NO ATENDIMENTO DE BARES,  
LANCHONETES E RESTAURANTES**

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



---

## TERMO DE APROVAÇÃO

APLICATIVO MÓVEL DE AUXÍLIO NO ATENDIMENTO DE BARES,  
LANCHONETES E RESTAURANTES

por

CASSIO MARIOTT

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

MARIOTT, Cassio. Aplicativo móvel de auxílio no atendimento de bares, lanchonetes e restaurantes. 2017. 42f. Monografia (Trabalho de especialização) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2017.

Existem diversos aplicativos de atendimento ao cliente para bares, lanchonetes e restaurantes, tanto para dispositivos móveis quanto para aplicações *desktop* e *web*. É comum que esses aplicativos não atendam a mais de uma plataforma, exigindo que o usuário se adapte a uma plataforma específica. Pode, ainda, surgir a necessidade de reescrever um projeto inteiro para outra linguagem, o que pode ser incômodo e demorado. A diversidade de aparelhos e plataformas operacionais que hoje uma SoftHouse precisa se preocupar em atender quando desenvolvida uma aplicação, levou ao objetivo deste trabalho que tem como objetivo principal o desenvolvimento de um aplicativo que possibilite com um mesmo projeto, compilar para as principais plataformas móveis disponíveis hoje no mercado, mantendo uma aplicação com características nativas, reduzindo o tempo e os custos.

**Palavras-chave:** Android. IOS. Aplicativos para bares.

## ABSTRACT

MARIOTT, Cassio. Mobile application to assist bars, snack bars and restaurants. 2017. 42f. Monografia (Trabalho de especialização) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2017.

There are several customer service applications for bars, cafeterias and restaurants, for both mobile and *desktop* applications, but most can not serve more than one platform, requiring you to separate and adhere to a specific platform, or the need to rewrite an entire project to another language, which can be cumbersome and time-consuming. A variety of handsets and operating platforms that today a SoftHouse needs to be concerned with when developing an application, has led to the goal of this work. This objective is the development of an application that allows with the same project, compile for the main mobile platforms available in the market, maintaining an application with native characteristics, reducing the time and cost.

**Keywords:** Android. IOS. Hybrid. Bar Service.

## LISTA DE FIGURAS

Figura 1 – Comando para instalação do Apache Cordova.....	12
Figura 2 – Comando para instalação do Ionic Framework .....	12
Figura 3 – Comando para criação de aplicação Iônica .....	12
Figura 4 – Testando aplicação Ionic no browser.....	13
Figura 5 – Adicionando plataforma Android .....	13
Figura 6 – Executar aplicação no dispositivo.....	13
Figura 7 – Gerar versão executável da aplicação .....	13
Figura 8 – Diagrama de caso de uso .....	17
Figura 9 – Diagrama de atividade.....	18
Figura 10 – Modelagem do banco de dados.....	20
Figura 11 – Modelagem do banco de dados.....	21
Figura 12 – Tela de mesas/comandas .....	22
Figura 13 – Opções da mesa/comanda ocupada.....	23
Figura 14 – Tela de produtos da mesa/comanda .....	24
Figura 15 – Tela de categoria de produtos .....	25
Figura 16 – Tela de produtos da categoria .....	26
Figura 17 – Tela do produto a ser adicionado na mesa.....	27
Figura 18 – Tela de configuração do produto .....	28
Figura 19 – Telas para montagem da pizza.....	29
Figura 20 – Telas da montagem da pizza montada e suas opções.....	30
Figura 21 – Tela de transferência de pedidos entre mesas.....	30
Figura 22 – Tela de agrupamento de mesas .....	31

## LISTA DE QUADROS

Quadro 1 – Tecnologias e ferramentas utilizadas na modelagem e na implementação do aplicativo .....	11
Quadro 2 – Requisitos funcionais .....	16
Quadro 3 – Requisitos não funcionais .....	16

## LISTAGENS DE CÓDIGOS

Listagem 1 – Arquivo application.properties .....	14
Listagem 2 – Classe Main.java.....	32
Listagem 3 – Classe Empresa.java .....	32
Listagem 4 – Classe EmpresaRepository.java.....	32
Listagem 5 – Classe EmpresaController.java.....	33
Listagem 6 – Código da Service.ts .....	35
Listagem 7 – Provedores registrados na aplicação .....	36
Listagem 8 – Código da CategoriaService.ts .....	36
Listagem 9 – Código da Categorias.ts .....	37
Listagem 10 – Código da CategoriaService.ts .....	38
Listagem 11 – Código da CategoriaItem.html .....	38
Listagem 12 – Bibliotecas de segurança adicionadas ao pom.xml .....	39
Listagem 12 – Código da WebSecurityConfiguration.java .....	40
Listagem 13 – Código da CategoriaController.java .....	40



## LISTA DE SIGLAS

ABIA	<i>Associação Brasileira das Indústrias da Alimentação</i>
API	<i>Application Programming Interface</i>
CLI	<i>Command Line Interface</i>
CMD	<i>Windows Command Prompt</i>
ERP	<i>Enterprise Resource Planning</i>
GET	<i>Groupe des Ecoles des Télécommunications</i>
HTML	<i>Hyper Text Markup Language</i>
IDE	<i>Integrated Development Environment</i>
IOS	<i>iPhone Operating System</i>
IP	<i>Internet Protocol</i>
JSON	<i>JavaScript Object Notation</i>
JWT	<i>Json Web Token</i>
MacOS	<i>Macintosh Operating System</i>
NPM	<i>Node Package Manager</i>
REST	<i>Representational State Transfer</i>
SDK	<i>Software Development Kit</i>
SGBD	<i>Sistema Gerenciador de Banco de Dados</i>
URL	<i>Uniform Resource Locator</i>

## SUMÁRIO

<b>1 INTRODUÇÃO</b> .....	<b>10</b>
<b>2 FERRAMENTAS, TECNOLOGIAS E PROCEDIMENTOS</b> .....	<b>11</b>
2.1 FERRAMENTAS E TECNOLOGIAS .....	11
2.2 PROCEDIMENTOS TÉCNICOS.....	12
<b>3 RESULTADOS</b> .....	<b>15</b>
3.1 ESCOPO DO SISTEMA.....	15
3.2 MODELAGEM DO SISTEMA.....	16
3.3 APRESENTAÇÃO DO SISTEMA .....	21
3.4 IMPLEMENTAÇÃO DO SISTEMA .....	31
<b>4 CONSIDERAÇÕES FINAIS</b> .....	<b>41</b>
<b>REFERÊNCIAS</b> .....	<b>42</b>

## 1 INTRODUÇÃO

Com o crescimento da população mundial é, também, crescente a demanda por alimentos. No Brasil, os estabelecimentos formais que têm como principal atividade a produção no setor de alimentos somavam, em 2011, 667,5 mil, representando 19% do total de estabelecimentos, contribuindo com quase 13% dos empregos formais (SEBRAE, 2011). Dados mais recentes continuam sustentando o crescimento desse setor. De acordo com dados da Associação Brasileira das Indústrias da Alimentação (ABIA), o setor de alimentação encerrou 2016 com faturamento de R\$ 614,3 bilhões, representando crescimento nominal de 9,3% em relação ao ano de 2015 (REBOUÇAS, 2017).

O crescimento do setor de alimentos no Brasil está associado também ao de estabelecimentos que oferecem alimentação pronta, como bares, restaurantes, lanchonetes e outros do gênero. Cada vez mais as pessoas realizam suas refeições fora de casa porque não há tempo para deslocar-se até em casa para almoçar, por exemplo, e pela praticidade, entre diversos outros fatores. Esses estabelecimentos são utilizados, também, para formas de socialização e entretenimento. Há ainda os que oferecem entrega (*delivery*) de pratos prontos, como as tradicionais pizzas, que têm se tornado cada vez mais comuns.

Nesse contexto, um software de gestão de bares, lanchonetes, restaurantes é uma solução que visa auxiliar no atendimento e na gestão das empresas do ramo. Verificou-se, assim, a oportunidade de desenvolver um aplicativo que pudesse auxiliar nos processos do ramo alimentício. Visando fornecer ao administrador uma visão melhor do que está realmente ocorrendo ou em qual processo necessita de um acompanhamento técnico especializado a fim de melhorar processos e garantir excelência em qualidade.

O aplicativo desenvolvido e apresentado neste texto é para dispositivos móveis e se integra a um *Enterprise Resource Planning* (ERP) existente.

## 2 FERRAMENTAS, TECNOLOGIAS E PROCEDIMENTOS

Este capítulo apresenta os materiais (*software*) utilizados para o desenvolvimento do aplicativo foco deste trabalho.

### 2.1 FERRAMENTAS E TECNOLOGIAS

As tecnologias e as ferramentas utilizadas para a modelagem e a implementação do aplicativo são apresentadas no Quadro 1.

Ferramenta / Tecnologia	Versão	Disponível em	Aplicação
Android SDK	6.0	<a href="https://developer.android.com/studio">https://developer.android.com/studio</a>	Pacote para desenvolver aplicativos para plataforma Android
Sublime Text 3	Build 3143	<a href="https://www.sublimetext.com/3">https://www.sublimetext.com/3</a>	<i>Integrated Development Environment (IDE)</i> para TypeScript/Ionic
MySql Maestro	12.3.0.1	<a href="https://www.sqlmaestro.com/products/mysql/maestro/">https://www.sqlmaestro.com/products/mysql/maestro/</a>	Sistema gerenciador de banco de dados
Eclipse Neon	Neon.2 Release (4.6.2)	<a href="http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/neon/3/eclipse-jee-neon-3-win32-x86_64.zip">http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/neon/3/eclipse-jee-neon-3-win32-x86_64.zip</a>	IDE Java
Ionic-angular	3.2.	<a href="https://www.npmjs.com/package/ionic-angular">https://www.npmjs.com/package/ionic-angular</a>	<i>Application Programming Interface (API)</i>
Node.js	6.11.1	<a href="https://nodejs.org/en/">https://nodejs.org/en/</a>	Interpretador de código JavaScript no lado do servidor.
TypeScript	2.2.1	<a href="https://www.typescriptlang.org/">https://www.typescriptlang.org/</a>	Linguagem de programação utilizada com o <i>framework</i> Angular
Angular	4.1.0	<a href="https://angular.io/">https://angular.io/</a>	<i>Framework</i> para desenvolvimento de aplicações <i>web</i> e <i>mobile</i>
Microsoft Visio 2010	14.0.4760.1000 (32 bits)	<a href="https://products.office.com/pt-br/microsoft-visio-2010">https://products.office.com/pt-br/microsoft-visio-2010</a>	Modelagem do sistema. Elaboração de diagramas e fluxogramas

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

## 2.2 PROCEDIMENTOS TÉCNICOS

Primeiramente, é necessário ter instalado na máquina o Java e o *Software Development Kit* (SDK) do Android para ser possível fazer o *build* do projeto para o dispositivo móvel, ou então ter o Xcode instalado se a intenção é compilar para o iPhone. Ressalta-se que para criar um projeto IOS é necessário ter um computador macOS.

Para iniciar o desenvolvimento de uma aplicação com *Ionic Framework* é preciso realizar a instalação do *Node.js* (o *download* do código-fonte ou instalador pode ser encontrado em <https://nodejs.org/en/download/>). O Node é uma plataforma construída sobre o motor JavaScript do Google Chrome, para o desenvolvimento de aplicações em rede utilizando apenas código em JavaScript, ideal para aplicações em tempo real com troca intensa de dados pelos dispositivos. O *Node Package Manager* (NPM) que vem incorporado ao pacote é um utilitário de linha de comando que interage com o repositório *online* para publicação de projetos para o Node.js. Ele ajuda na instalação de pacotes, no gerenciamento de versão e de dependências e é por meio dele que são instalados os demais pacotes.

Em seguida, abrindo o *Windows Command Prompt* (CMD), é necessário instalar a versão mais recente do *Apache Cordova*. Essa é a plataforma de desenvolvimento móvel híbrido com APIs que permitem que o desenvolvedor acesse funções nativas do dispositivo, digitando o comando apresentado na Figura 1.

```
C:\Users\Administrador>npm install -g cordova
```

Figura 1 – Comando para instalação do Apache Cordova

E por fim, a instalação do Ionic Framework é feita pela digitação do comando que está na Figura 2.

```
C:\Users\Administrador>npm install -g ionic
```

Figura 2 – Comando para instalação do Ionic Framework

Realizada a instalação dos pacotes, é necessário criar a estrutura inicial de um projeto Iônico. Para isso é utilizado o *Command Line Interface* (CLI) do próprio Ionic Framework, que vem com uma série de comandos que ajudam na criação e na manutenção de projetos. Essa instalação é realizada pelo comando apresentado na Figura 3.

```
P:\SUN\trunk\Ionic>ionic start Panto sidemenu
```

Figura 3 – Comando para criação de aplicação Iônica

Na Figura 3: **Panto** é o nome da aplicação e **sidemenu** é o *template*, pois o comando *start* oferece três tipos de *template*:

- **sidemenu**: adiciona um menu lateral a aplicação;
- **tabs**: cria uma navegação baseada em guias;
- **blank**: cria um projeto básico, sem nenhum *template*;

Para realizar testes na aplicação, não é necessário, obrigatoriamente, realizar o *build* para o celular. Isso porque o Ionic possui um servidor incorporado, o que permite testar a aplicação no *browser*. Para isto é preciso entrar no diretório que foi gerada a aplicação e digitar o comando apresentado na Figura 4.

```
P:\SUN\trunk\Ionic\Panto>ionic serve
```

Figura 4 – Testando aplicação Ionic no browser

Para realizar o *build* para o dispositivo, primeiro é necessário adicionar a plataforma na qual se deseja compilar o aplicativo, digitando o comando da Figura 5.

```
P:\SUN\trunk\Ionic\Panto>ionic cordova platform add android
```

Figura 5 – Adicionando plataforma Android

Será solicitada também a instalação do *plugin @ionic/cli-plugin-cordova* se ele ainda não estiver instalado, sendo necessária apenas confirmação. Ao final da instalação, será criado na raiz do projeto um diretório *platforms*, com um projeto Android dentro.

Com esses procedimentos concluídos, é possível executar a aplicação no celular, digitando o comando apresentado na Figura 6.

```
P:\SUN\trunk\Ionic\Panto>ionic cordova run android --prod
```

Figura 6 – Executar aplicação no dispositivo

E por último, é possível gerar a versão executável da aplicação, utilizando o comando que está na Figura 7.

```
P:\SUN\trunk\Ionic\Panto>ionic cordova build --release android
```

Figura 7 – Gerar versão executável da aplicação

A seguir, são apresentadas as configurações necessárias para a criação da aplicação *WebService*, desenvolvida utilizando *Spring Boot*, *Maven* e *MySQL*. Após a criação do projeto, é necessário adicionar no arquivo *pom.xml* as dependências necessárias para o Maven construir o projeto. Em seguida é necessário especificar no arquivo

*resources/application.properties* os parâmetros de conexão para as credencias do MySQL. A Listagem 1 apresenta o código para essas configurações.

```
4 spring.jpa.show-sql: true
5 spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
6 server.port = 8080
7 spring.datasource.url=jdbc:mysql://localhost/db
8 spring.datasource.username=root
9 spring.datasource.password=root
10 spring.datasource.driverClassName=com.mysql.jdbc.Driver
```

**Listagem 1 – Arquivo *application.properties***

### 3 RESULTADOS

Neste capítulo é apresentado o aplicativo para dispositivos móveis que foi desenvolvido como resultado da realização deste trabalho.

#### 3.1 ESCOPO DO SISTEMA

O aplicativo tem como foco principal de atuação as operações de venda que são realizadas por empresas do ramo alimentício e que preparam refeições, visando fornecer maior agilidade nos pedidos, produção e entrega para os clientes. O aplicativo é um *add-on* para um sistema de automação comercial ERP existente. O aplicativo e o ERP utilizam um banco de dados único que é MySQL.

A aplicação *mobile* desenvolvida e apresentada neste trabalho é um módulo para ser utilizado por garçons, quando realizado o atendimento dos clientes junto ao estabelecimento. O aplicativo auxiliará no gerenciamento de pedidos por mesa ou por comanda.

O sistema deverá, a partir da parametrização no ERP, saber diferenciar e atender diversos segmentos como restaurantes, lanchonetes, pizzarias, padarias, confeitarias e bares. O aplicativo não trabalha de forma *off-line*, sendo necessário a disponibilidade de Wi-Fi para a sua utilização.

Além de gerenciar as mesas, o aplicativo possibilitará alterar a composição de pedidos no ato da venda e efetuar o controle de estoque pela baixa por composição. O processo de produção é automatizado para posteriormente saber quanto de consumo e quais são os produtos mais vendidos. Esses dados auxiliam na realização de compras com mais eficiência visando evitar desperdício.

O sistema permite a criação de produtos compostos para a produção como os lanches e o gerenciamento de produtos simples como refrigerantes.

Para os garçons será possível selecionar os produtos por categoria podendo ser bebidas ou lanches, cozinha ou copa ou outro. Isso facilita a localização dos produtos agilizando o atendimento.

O sistema também armazena os dados do vendedor para posteriormente ser possível realizar comissão de vendas por vendedor.



### 3.2 MODELAGEM DO SISTEMA

Esta seção apresenta a modelagem do aplicativo, abordando a análise dos requisitos, os diagramas utilizados, assim como as funcionalidades definidas. Dentre esses diagramas encontra-se o diagrama de caso de uso e o diagrama de atividade. E, por último, está a modelagem do banco de dados.

O Quadro 2 apresenta os requisitos funcionais definidos para o sistema.

Identificação	Nome	Descrição
01	Registrar pedido	Garçom efetua o registro de um pedido feito por um cliente dentro do estabelecimento.
02	Enviar pedido	Após o pedido realizado e confirmado pelo cliente o garçom efetua o envio.
03	Agrupar mesas/comandas	O garçom pode efetuar o agrupamento de mesas e comandas de acordo com solicitações dos clientes.
04	Trocar mesas/comandas	Será possível transferir os produtos de uma mesa ou comanda para outra.
05	Montar pizza	Será possível definir a composição de pizzas, escolhendo a quantidade de sabores e os respectivos sabores.
06	Excluir produto	Será possível realizar a exclusão do produto da mesa antes de ele ser produzido.

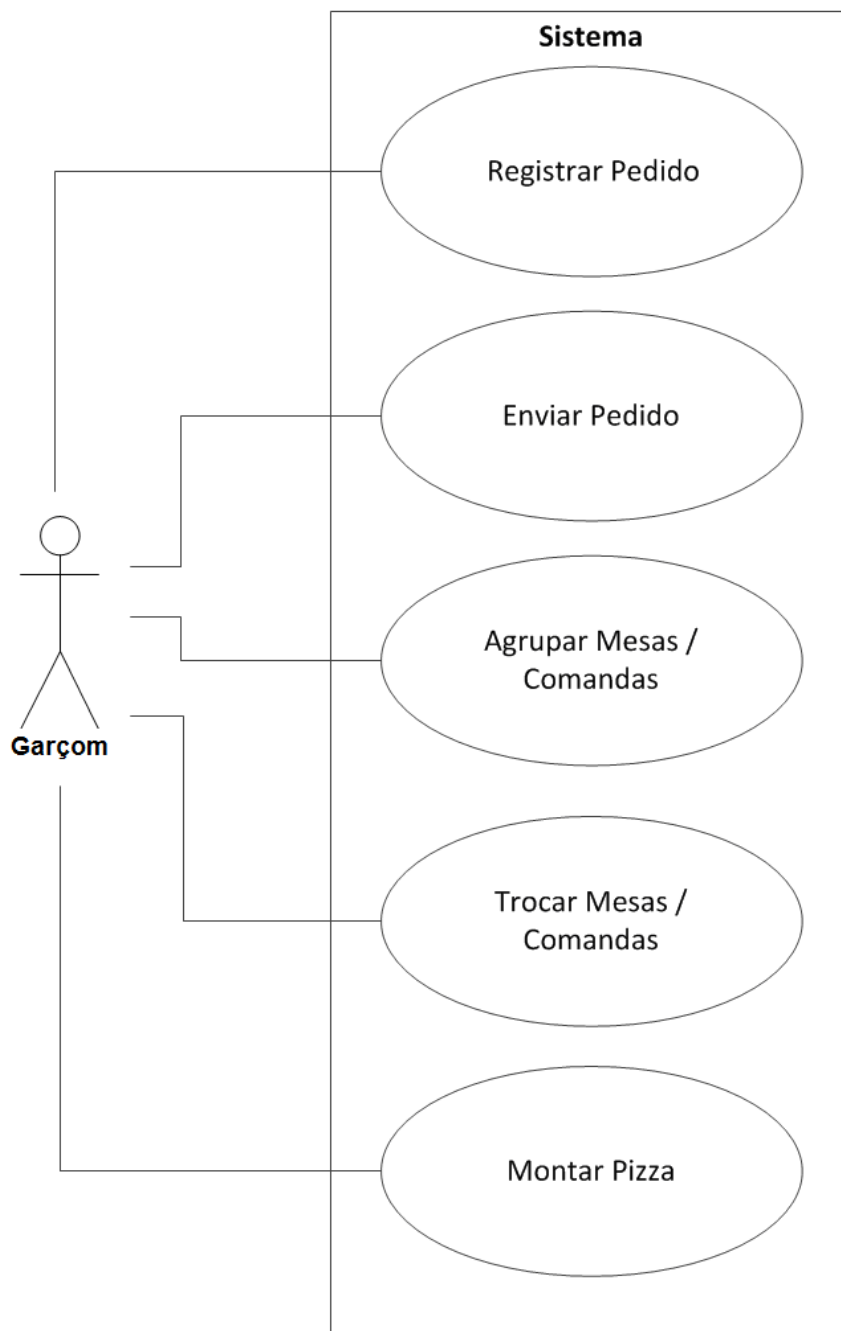
**Quadro 2 – Requisitos funcionais**

Vinculados aos requisitos funcionais estão os não funcionais, apresentados no Quadro 3.

Identificação	Nome	Descrição
01	Padronizar telas	Utilizar padronização de telas para facilitar o entendimento e agilizar o uso do aplicativo.
02	Padronizar nomes de componentes	A padronização de nomes de componentes facilita o desenvolvimento e facilitando manutenção do sistema e o desenvolvimento de novas funcionalidades.
03	Nomes dos campos no banco de dados	Os nomes dos campos no banco de dados devem ser semelhantes com os nomes dos atributos das classes, facilitando o desenvolvimento e entendimento das regras de negócio posteriormente.
04	Versionamento de fontes do projeto (SubVersion)	Utilizar sistema de versionamento de códigos para efetuar o controle de versões geradas no sistema.
05	Servidor de aplicação	Criar servidor de aplicação aonde serão executadas as regras de negócio bem como a persistência dos dados.

**Quadro 3 – Requisitos não funcionais**

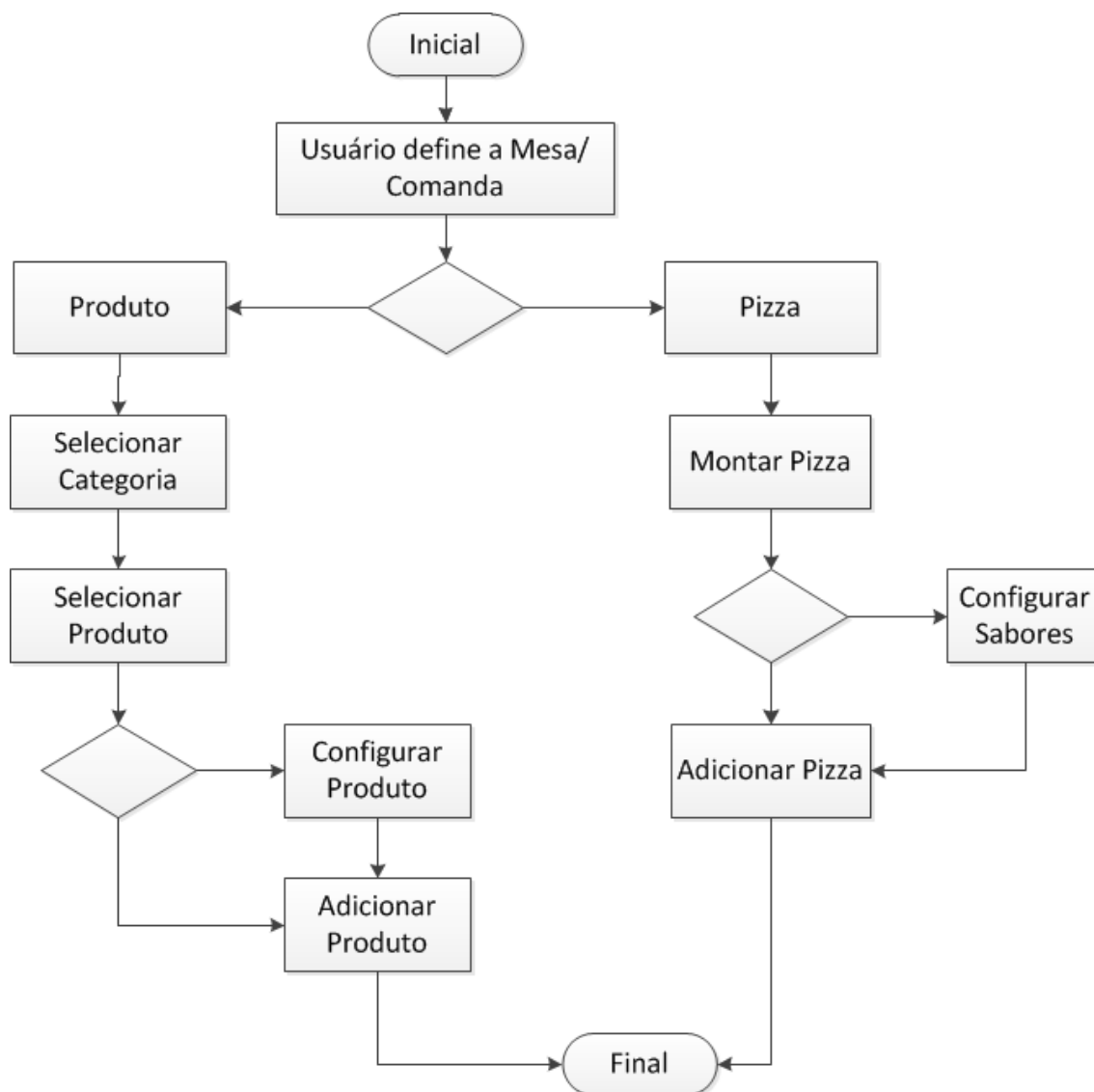
A partir dos requisitos levantados para o sistema, foi possível criar o diagrama de caso de uso, conforme apresentado na Figura 8.



**Figura 8 – Diagrama de caso de uso**

Basicamente, o usuário (o garçon) definirá para qual mesa/comanda deseja realizar o pedido e, posteriormente, selecionará se deseja incluir um produto ou uma pizza, de acordo com o pedido do cliente. Adicionado um item já é possível realizar o envio do pedido para impressão. Além de realizar os pedidos, o usuário pode realizar o agrupamento de mesas/comandas ou realizar a transferência de pedidos entre ambas.

Após a definição dos requisitos e o desenvolvimento do caso de uso, foi o desenvolvimento do diagrama de atividades. Esse diagrama visa auxiliar o desenvolvimento de fluxo entre as telas, assim como a usabilidade do aplicativo. Este diagrama está na Figura 9.



**Figura 9 – Diagrama de atividade**

A modelagem do banco de dados é apresentada nas Figuras 10 e 11. Esse diagrama possui as seguintes tabelas:

- **parametros:** armazena as configurações definidas no ERP e que são necessárias para inclusão de um pedido e o controle de mesa ou comanda.

- empresa: apresenta todos os estabelecimentos cadastrados, obrigando o usuário selecionar um deles ao efetuar o *login*.
- comanda: controle de cadastro das mesas/comandas, que serão apresentados na tela inicial.
- categoria: cadastros do agrupamento de produtos, que serão listados em uma tela para o usuário.
- categoriaitem: produtos vinculados a determinada categoria.
- ecommerce: armazena o pedido gerado após a seleção da mesa/comanda e seja incluído um produto ou pizza, fornecendo uma chave estrangeira com as tabelas ecommerceitem, comanda e ecmmentpedido.
- ecommerceitem: armazena os itens dos pedidos, tanto para produtos quando para pizzas.
- item: armazena todos os produtos cadastrados contendo o nome e a unidade de medida no sistema fornecendo ele a chave estrangeira para a tabela ecommerceitem, pizza, sabor, ecmitemcons, itemconsumo.
- itemempresa: armazena as informações referentes a preço do produto para a tabela item, diferenciando as informações que são distintas para um mesmo produto entre os distintos estabelecimentos cadastrados. Fornece uma chave estrangeira para as tabelas item, itemcomp, itemconsumo e categoriaitem.
- composicao: é a engenharia de um produto composto, fornecendo uma chave estrangeira para tabela item.
- itemcomp: esta tabela contém os itens que compõem determinada engenharia, fornecendo uma chave estrangeira para tabela composicao.
- ecmitemcons: armazena os produtos dos pedidos e seus respectivos subprodutos, como, por exemplo, os sabores de uma pizza. Fornece uma chave estrangeira para tabela itemconsumo. É partir desses registros que são identificados os consumos dos produtos compostos.
- itemconsumo: armazenamento dos detalhes dos produtos composto adicionados ao pedido, sejam eles engenharia padrão ou itens adicionais.
- pizza: vinculada a tabela item. Define quais dos cadastros dos produtos são definidos como pizzas.
- pizzasabor: é nesta tabela que são armazenadas as configurações para montagem da pizza, como, por exemplo, a quantidade de sabores.

- sabor: vinculado a tabela item. Define quais dos cadastros de produto são definidos como sabores de pizza.
- saborpizza: armazena os detalhes de cada sabor de pizza, como a composição do sabor por tamanho de pizza.
- usuario: armazena os usuários cadastrados na aplicação *mobile*, sendo utilizado para validação do Spring Security.
- usuario\_permissoes: tabela que vincula as permissões dos usuários, também é utilizada nas manutenções do Spring Security.
- permissao: tabela para gravar as *roles* dos usuário, também compõe a manutenção de segurança.

peessoa: armazena os colaboradores cadastrados no ERP e que serão disponibilizados para cadastro de *login* para o *mobile*.

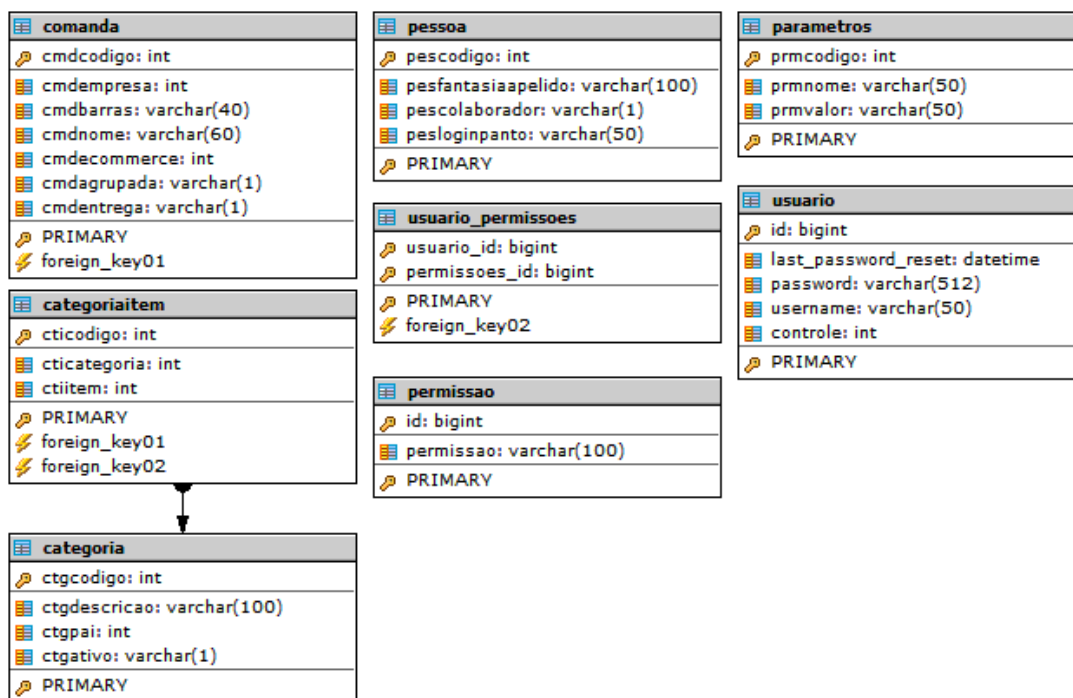


Figura 10 – Modelagem do banco de dados

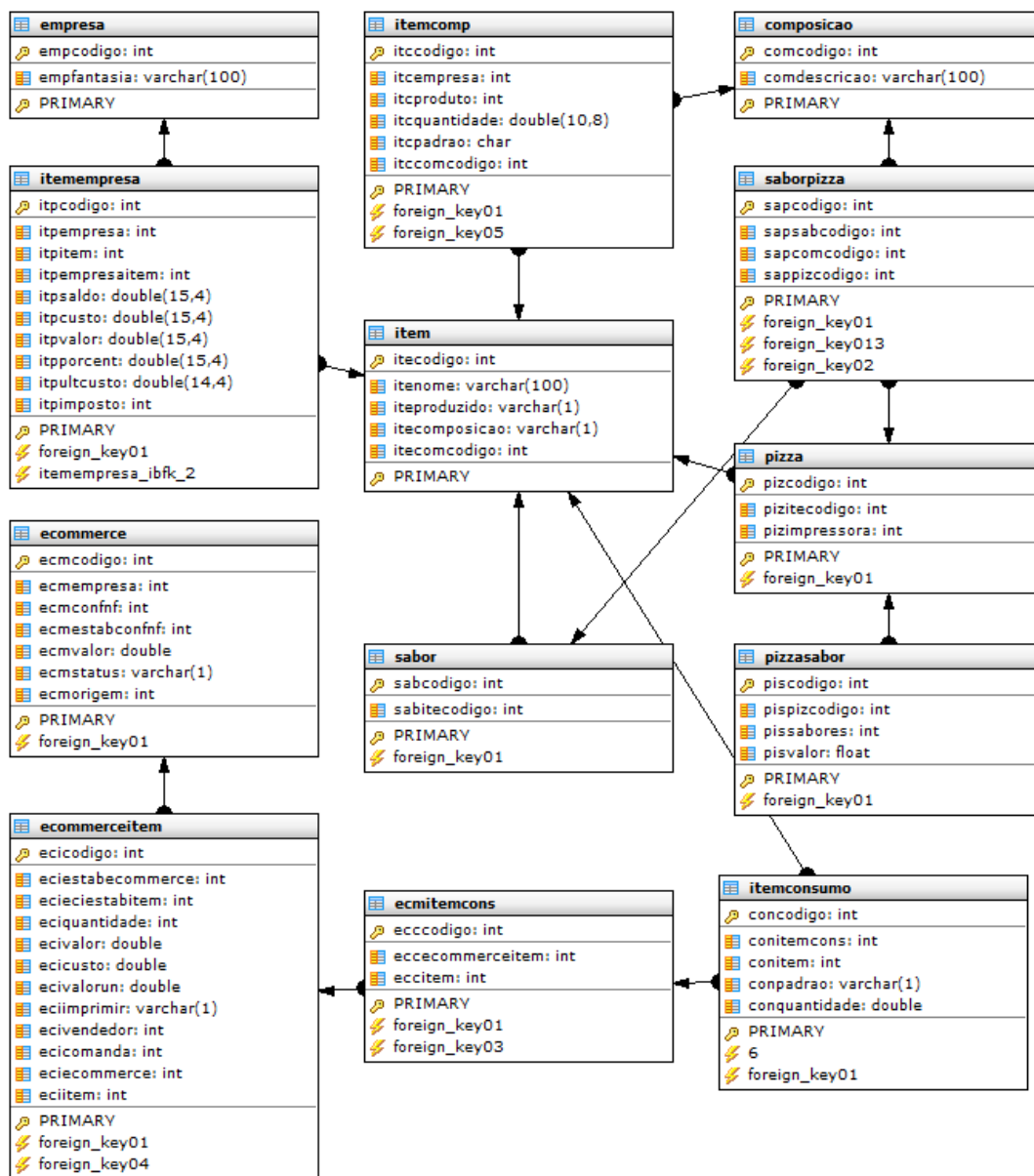


Figura 11 – Modelagem do banco de dados

### 3.3 APRESENTAÇÃO DO SISTEMA

O aplicativo possui uma tela inicial, na qual é solicitado ao usuário a identificação do estabelecimento, usuário e senha, permitindo configurar o *Internet Protocol* (IP) e porta de acesso ao servidor, como também cadastrar um novo usuário.

Após serem efetuadas as validações do usuário é exibida uma tela que apresenta todas as mesas ou comandas cadastradas no ERP. Elas são apresentadas com cores diferentes (azul

significa que a mesa está livre e laranja que está ocupada), conforme o *status* de ocupado ou disponível, permitindo, também, ao usuário realizar uma busca/filtro por código de barras.

A Figura 12 apresenta a tela de mesas/comandas.

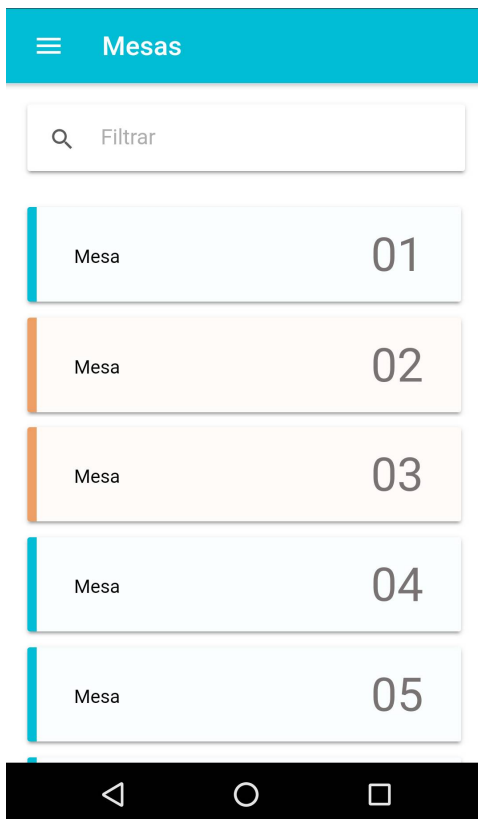


Figura 12 – Tela de mesas/comandas

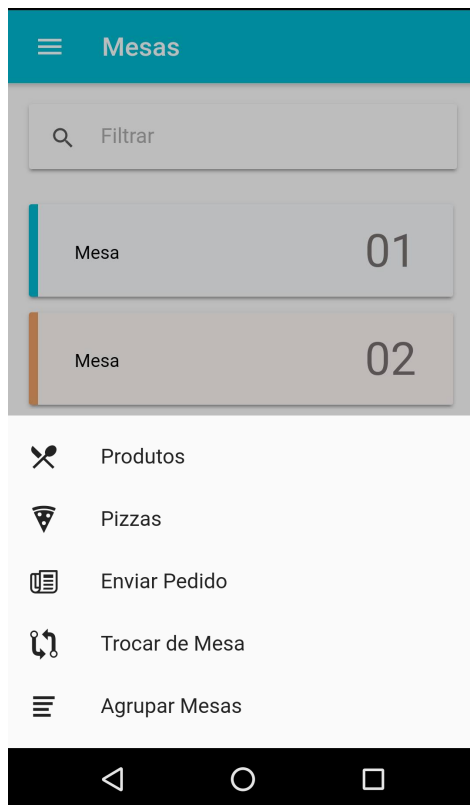
Ao clicar sobre uma mesa/comanda com *status* de disponível, representado pela cor azul, serão apresentadas as opções:

- Produtos
- Pizzas
- Agrupar Mesas

E ao clicar sobre uma mesa/comanda com *status* de ocupada, além das opções já apresentadas anteriormente, serão apresentadas ainda:

- Enviar Pedido
- Trocar de Mesa

A Figura 13 apresenta as opções exibidas quando clicado sobre uma mesa/comanda com *status* de ocupado.



**Figura 13 – Opções da mesa/comanda ocupada**

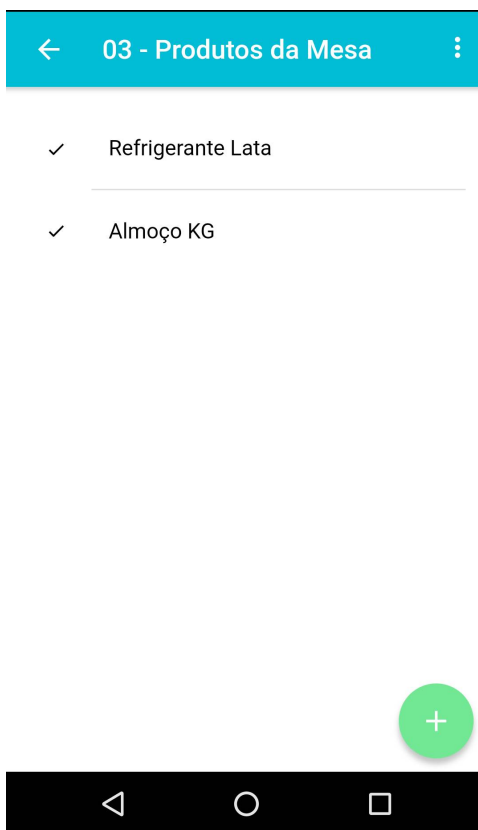
Quando clicado sobre a opção **Produtos** é exibida uma nova tela, sendo apresentados os produtos já adicionados à respectiva mesa/comanda.

Para os clientes que desejam realizar as impressões dos pedidos, é possível realizar a parametrização deles no ERP. Esta impressão é realizada após o usuário adicionar um ou mais produtos à mesa/comanda e, posteriormente, clicar sobre a opção **Enviar Pedido**. Esta opção está disponível tanto na tela inicial das comandas/mesas, quando clicado sobre uma mesa/comanda com *status* de ocupado, quanto na tela de Produtos.

Para permitir ao usuário identificar quais produtos já foram enviados para impressão, um *status* é apresentado ao lado de cada produto da mesa/comanda. Esse status é representado por um ícone no formato de “v”. Os produtos ainda não liberados para impressão, se clicado sobre o mesmo, é possível excluí-los, caso contrário, apenas a consulta dos dados cadastrais. Nesta tela é possível também realizar a inclusão de novos produtos.

A Figura 14 apresenta a tela de produtos vinculados a uma determinada mesa ou comanda.

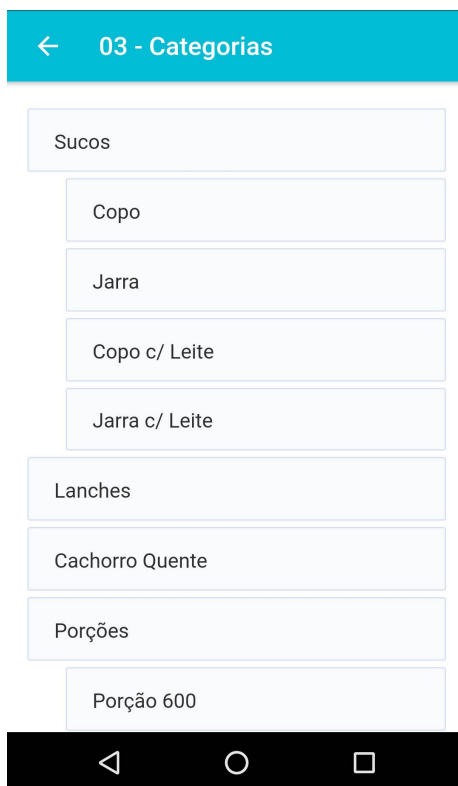




**Figura 14 – Tela de produtos da mesa/comanda**

Se clicado para adicionar um novo produto, a nova tela apresentada será a de categorias. Estas categorias são cadastros que permitem ao usuário definir grupos de produtos, possibilitando uma maior agilidade na seleção destes quando realizado o atendimento às mesas.

A Figura 15 apresenta a tela de categorias.

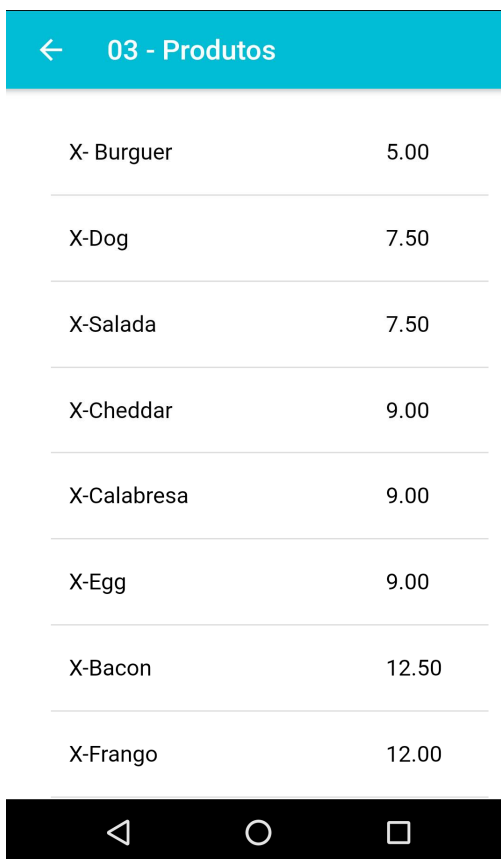


**Figura 15 – Tela de categoria de produtos**

Quando selecionada a categoria desejada serão apresentados todos os produtos vinculados a ela, com a descrição e o valor. No entanto, não serão apresentados todos os produtos cadastrados no ERP, será carregado apenas os que possuírem alguma categoria vinculada em seu cadastro.

É também no cadastro de categorias que é realizada a parametrização de impressões. Um estabelecimento pode ser dividido em setores específicos para enviar os produtos de um mesmo pedido, como, por exemplo, cozinha e copa. Permitindo assim, ser configurado para que os produtos de determinadas categorias sejam impressos em lugares distintos, mesmo estando em um mesmo pedido.

A Figura 16 apresenta a tela de produtos vinculados à determinada categoria. As informações apresentadas estão exibidas em uma lista simples, sem maiores opções, apenas possibilitando clicar sobre o produto que se deseja adicionar a determinada comanda/mesa.



← 03 - Produtos	
X- Burguer	5.00
X-Dog	7.50
X-Salada	7.50
X-Cheddar	9.00
X-Calabresa	9.00
X-Egg	9.00
X-Bacon	12.50
X-Frango	12.00

**Figura 16 – Tela de produtos da categoria**

A tela da Figura 15 é apresentada quando clicado sobre um produto para inclusão. Essa tela apresenta o código de controle do produto no ERP, a descrição, o valor unitário, um campo para vínculo da comanda ou mesa, conforme parametrização do ERP, e a quantidade do produto solicitada pelo cliente. As únicas informações editáveis nesta tela são a quantidade do produto e a comanda ou mesa. Esta informação e a comanda ou a mesa são validadas conforme configurações definida no ERP, respeitando a seguinte condição:

- Quando definido que o controle de pedidos do estabelecimento será por comanda, é exibido ao usuário o campo mesa, permitindo vincular determinado produto de uma comanda a uma mesa.
- Quando definido que o controle de pedidos do estabelecimento será por mesa, é exibido ao usuário o campo comanda, permitindo vincular determinado produto de uma mesa a uma comanda.

Esta configuração permite, posteriormente, maior agilidade no acerto (fechamento ou pagamento) dos pedidos, permitindo também um maior controle de pedidos agrupados em

uma mesma mesa, por exemplo. Nas imagens apresentadas a seguir, o controle está por comanda.

A Figura 17 apresenta a tela do produto selecionado para inclusão na mesa.

← 03 - Adicionar produto

Código: 51

Descrição: X-Salada

- Qtde: 1 +

Comanda:

Valor: 7,5

ADICIONAR

CONFIGURAR

Figura 17 – Tela do produto a ser adicionado na mesa

Quando definido no cadastro do produto que possui composição, é exibida, também, nesta telada (Figura 16) opção **Configurar**, que abrirá uma nota na tela, permitindo a alteração da composição padrão do produto, como, também, vincular itens adicionais.

No cadastro da composição no ERP é possível definir quais produtos são padrão, ou seja, sempre fazem parte da composição, como também, produtos que não são padrão da composição, mas podem ser incluídos como itens adicionais. Caso seja necessário incluir ao produto algum item adicional, e este não esteja cadastrado na composição, é disponibilizada na tela de configuração do produto uma opção para busca e adição de produtos configurados como produtos adicionais de uma composição.

Quando retirado um item padrão da composição, ou incluído um item adicional, esses são impressos no pedido como informações adicionais.

A Figura 18 apresenta a tela de configuração de produto.

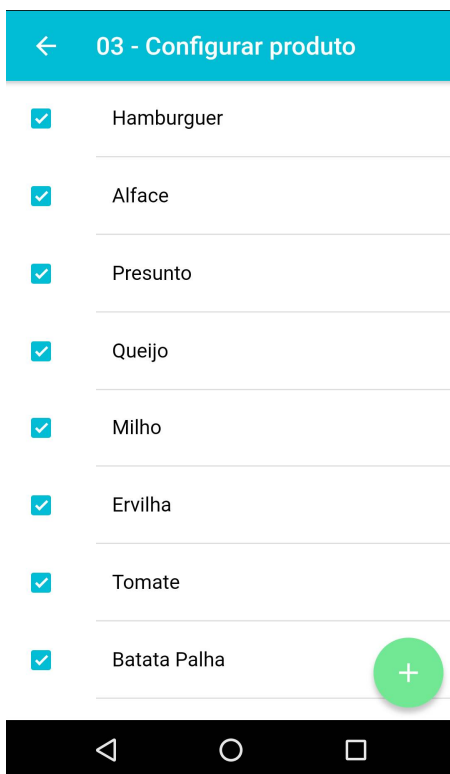


Figura 18 – Tela de configuração do produto

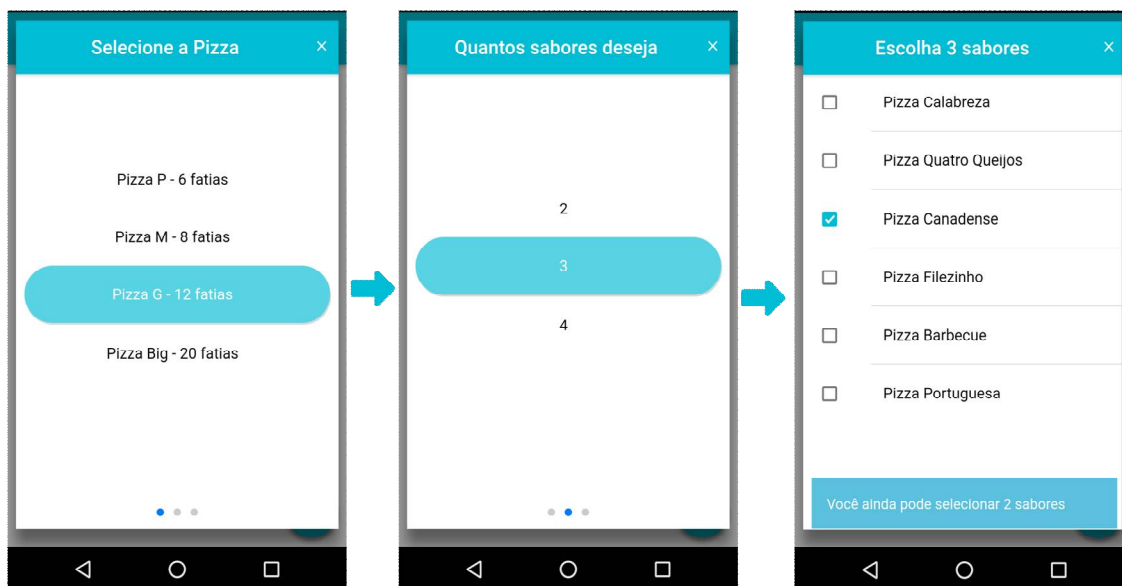
Posteriormente, é necessário clicar sobre a opção *Adicionar* e o produto será vinculado a mesa/comanda e o usuário será direcionado novamente para a lista de produtos.

Quando clicado sobre a opção *Pizzas* é exibida uma nova tela que solicita que ao usuário para selecione uma das pizzas cadastradas. É neste cadastro que são definidos a quantidade de sabores que a pizza possuirá.

Selecionada a opção pizza, o usuário será direcionado para a opção de seleção de quantidades de sabores disponíveis, para posteriormente montar a pizza. Definido a quantidade de sabores, a última etapa de montagem da pizza é a seleção dos sabores. O usuário será direcionado para a tela com a pizza montada e selecionará a quantidade total dos sabores.

É no cadastro de sabores de pizza que são definidos os ingredientes a serem utilizados na montagem da pizza, o que permite, posteriormente à montagem, configurar também os sabores, retirando ingredientes, ou incluindo itens adicionais.

A Figura 19 apresenta as telas para montagem da pizza.



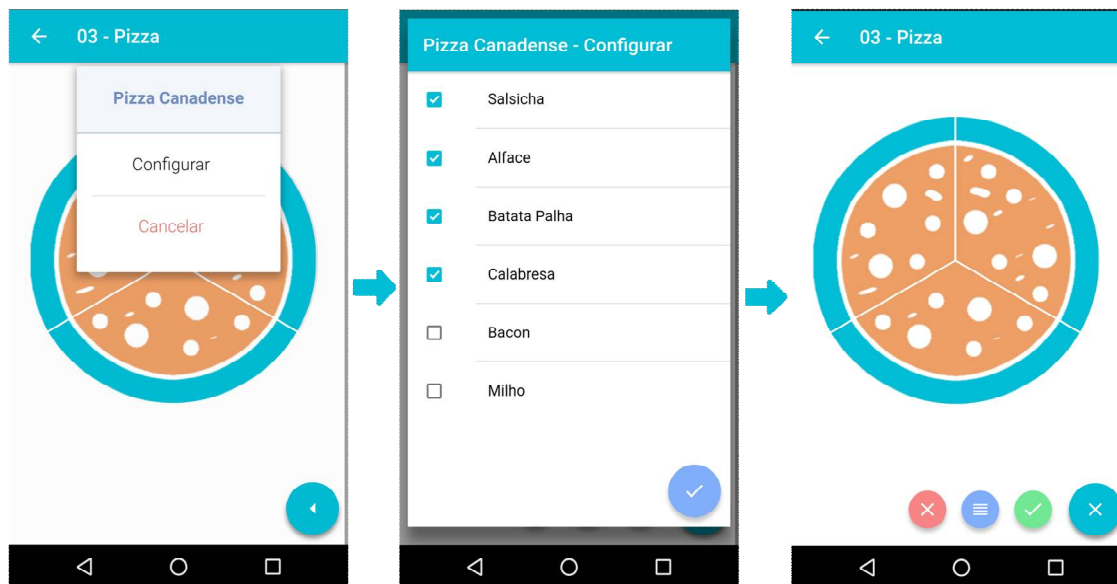
**Figura 19 – Telas para montagem da pizza**

A quantidade de sabores por tamanho da pizza é definido pelo dono do estabelecimento por meio de uma tela de configurações. Após selecionados todos os sabores, o usuário será redirecionado para a tela com a pizza montada. Nesta tela, se clicado sobre um dos sabores, será disponibilizada a opção de configuração, exibindo os ingredientes padrão, juntamente com os itens configurados como adicionais, mantendo a mesma funcionalidade já implementada para os demais produtos. As demais opções disponíveis são:

- Incluir a pizza à mesa/comanda
- Cancelar a inclusão
- Voltar para o processo de montagem

Realizada a inclusão, é apresentada na tela de produtos, permitindo a exclusão apenas antes do pedido ser enviado para impressão e o usuário será redirecionado para a tela de mesas/comandas. A impressão do pedido de pizza é realizada individualmente, pela quantidade de informações que podem estar vinculadas.

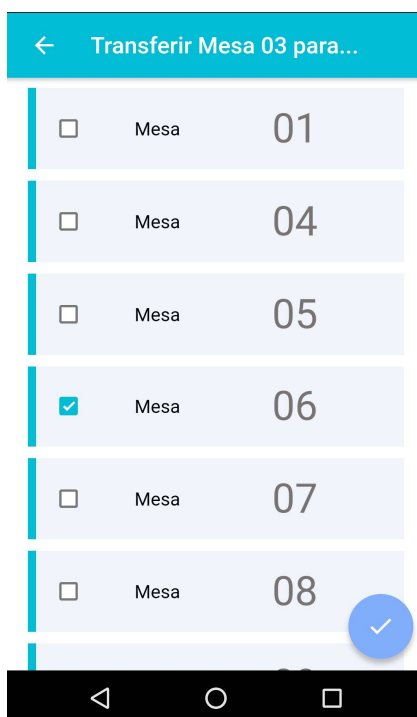
A Figura 20 apresenta as telas da pizza montada e suas opções.



**Figura 20 – Telas da montagem da pizza montada e suas opções**

Quando clicado sobre a opção *Trocar de Mesa*, é apresentada uma nova tela, listando todas as mesas/comandas cadastradas e com *status* disponível. Nesta lista, é possível selecionar apenas uma mesa/comanda para transferência. Após a seleção, se clicado sobre a opção de confirmação, o pedido é transferido de mesa/comanda, alterando o seu *status*.

A Figura 21 apresenta a tela para transferência de pedidos entre mesas.



**Figura 21 – Tela de transferência de pedidos entre mesas**

Quando clicado sobre a opção **Agrupar Mesas**, é apresentada uma nova tela, listando todas as mesas/comandas, independente de seus *status*. Nesta lista é possível selecionar uma ou mais mesas/comandas para realizar o agrupamento.

Após a seleção, se clicado sobre a opção de confirmação, todas as mesas/comandas serão agrupadas, gerando um único pedido. Portanto, se entre as mesas/comandas selecionadas, mais que uma já possui produtos vinculados, ambos serão agrupados em um novo pedido.

A Figura 22 apresenta a tela para agrupamento de mesas.

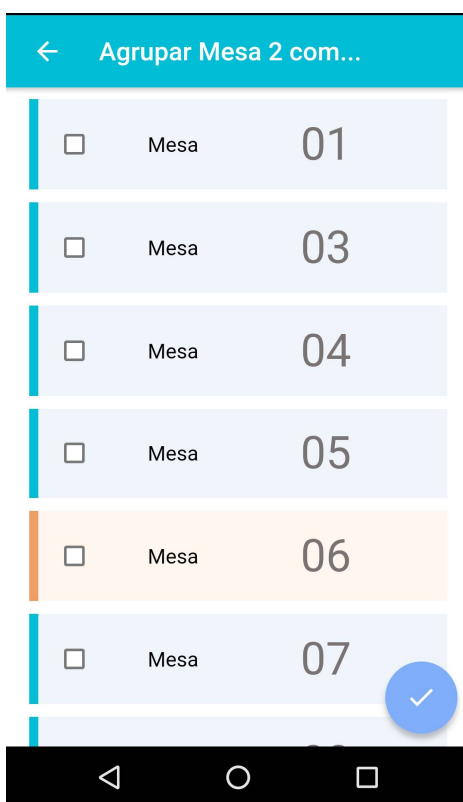


Figura 22 – Tela de agrupamento de mesas

### 3.4 IMPLEMENTAÇÃO DO SISTEMA

Para finalizar as configurações do projeto, é necessário configurar a classe principal (nesse caso, a classe *Main*), que será o posto de partida do projeto *Spring Boot*. O método *main()* usa o *Spring Boot SpringApplication()* para iniciar o serviço, portanto, é necessário a anotação *@SpringBootApplication*. A Listagem 2 apresenta o código dessas configurações.



```

12 @SpringBootApplication
13 public class Main extends SpringBootServletInitializer{
14
15     public static void main(String[] args) {
16         SpringApplication.run(Main.class, args);
17     }
18
19     @Override
20     protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
21         return application.sources(Main.class);
22     }
23
24 }

```

#### Listagem 2 – Classe Main.java

Com o projeto configurado, podem ser criadas as classes de mapeamento das entidades de banco. Neste caso, é necessária a anotação `@Table` para saber qual entidade está sendo referenciada e para os atributos é utilizada a anotação `@Column` para realizar o mapeamento das colunas do banco de dados. Estão sendo utilizadas também as anotações `@Getter` e `@Setter`, ambas herdadas do *lombok*, adicionado anteriormente no arquivo *pom.xml*, não sendo necessária, assim, a criação manual dos métodos *getters* e *setters* para os atributos das classes. A Listagem 3 apresenta a implementação da classe *Empresa.java*.

```

14 @Entity
15 @Getter
16 @Setter
17 @EqualsAndHashCode(of = "id")
18 @Table(name = "empresa")
19 public class Empresa {
20     @Id
21     @GeneratedValue(strategy = GenerationType.IDENTITY)
22     @Column(name = "empcodigo")
23     private Integer id;
24
25     @Column(name = "empfantasia", nullable = false)
26     private String nome;
27
28 }
29

```

#### Listagem 3 – Classe Empresa.java

O próximo passo é a criação da interface de acesso aos dados. Para isto, é necessário fazer a herança da interface *JpaRepository*, definindo também qual entidade está sendo gerenciada e o tipo do identificador dela. Nesse caso, o atributo da classe definido com a anotação `@Id`, como apresentado no código da Listagem 4.

```

5 public interface EmpresaRepository extends JpaRepository<Empresa, Integer> {}

```

#### Listagem 4 – Classe EmpresaRepository.java

Para finalizar as implementações de gerenciamento de manipulação das entidades do banco de dados, é criada a classe controladora, que definirá as *Uniform Resource Locator* (URL) que serão consumidas aplicação Iônica. Neste momento, é necessário fazer uso da anotação *@RestController* que define o retorno REST dos serviços implementados.

Outra anotação necessária é a *@RequestMapping*, que define o prefixo de todas as requisições realizadas pelo cliente. No exemplo da Figura 12, todas as requisições de acesso ao *WebService* terão o prefixo */login*.

Para finalizar, é necessário fazer a injeção de dependência da interface *repository*, que é quem disponibiliza os métodos de manipulação dos registros conforme possível chamada da API. A Listagem 5 apresenta a classe *EmpresaController*.

```

15 @RestController
16 @RequestMapping("/login")
17 public class EmpresaController {
18
19     @Autowired
20     private EmpresaRepository empresaRepository;
21
22     @GetMapping("/empresa/{codigo}")
23     public Empresa visualizar(@PathVariable Integer codigo){
24         return empresaRepository.findOne(codigo);
25     }
26
27     @RequestMapping(method = RequestMethod.GET, value = "/empresa")
28     public List<Empresa> colaboradores(){
29         return empresaRepository.findAll();
30     }
31
32     @GetMapping("/confweb/valida")
33     public String valida(){
34         return "Conectado";
35     }
36
37 }

```

**Listagem 5 – Classe *EmpresaController.java***

Para o desenvolvimento da aplicação *front-end/mobile* foi utilizado o *Ionic*, que é um *framework* para desenvolvimento de aplicações para dispositivos móveis. Esse *framework* que visa o desenvolvimento de aplicação híbridas e agilizar esse desenvolvimento. Ele é uma pilha de componentes e outros *frameworks*.

Esses componentes são:

- Cordova: é uma plataforma de desenvolvimento móvel com APIs que permitem que o desenvolvedor acesse funções nativas do dispositivo. Quando

se desenvolve com o Cordova, um aplicativo híbrido é criado e o código pode ser compilado para diversas plataformas, como Android, *iPhone Operating System* (IOS) e Black Berry.

- Angular: é um *framework* para desenvolvimento de aplicações *web*, que utiliza JavaScript, mas, também, é possível escrever as aplicações em Angular utilizando TypeScript, totalmente baseado em componentes.
- Ionic Module: é o responsável pela inicialização das aplicações em Ionic. Ao passar por um componente raiz, o Ionic Module garantirá que todos os componentes, diretrizes e provedores da estrutura sejam importados.
- Ionic CLI: é a interface de linha de comando para o desenvolvimento de aplicações iônicas. Com a CLI do Ionic Framework é simples iniciar um projeto, como, também, compilar, executar e emular aplicações.

Com a utilização do Angular é possível optar pelo desenvolvimento com TypeScript. Portanto, o desenvolvimento da aplicação *front-end* foi realizado utilizando esta linguagem.

Na Listagem 6 está o código do Service.ts, que é responsável por disponibilizar para aplicação as funções bases de acesso e manipulação de dados que necessitem autenticação do usuário.

```

@Injectable()
export class Service {

  constructor(private http: Http, private config: Config) {}

  getHeader(){
    let headers = new Headers({'X-Auth-Token':this.getToken()});
    headers.append('Content-Type', 'Application/json');
    let options = new RequestOptions({headers:headers});
    return options;
  }

  getToken(){
    let currentUser =
JSON.parse(sessionStorage.getItem('currentUser'));
    let token = currentUser.token;
    return token ? token : '';
  }

  findAll(url): Observable<any[]> {
    return this.http.get(this.config.apiUrl + url, this.getHeader())
      .map(resposta => resposta.json())
      .catch(erro => {throw new Error(erro.message)});
  }

  findOne(url): Observable<any> {
    return this.http.get( this.config.apiUrl + url )
      .map(resposta => resposta.json())
      .catch(erro => {throw new Error(erro.message)});
  }
}

```

```

    }

    save(url, obj): Observable<any> {
        return this.http.post(this.config.apiUrl + url,
            JSON.stringify(obj), this.getHeader())
            .map(resposta => resposta.json())
            .catch(erro => { throw new Error(erro.message)});
    }

    delete(url): Observable<any>{
        return this.http.delete(this.config.apiUrl + url, this.getHeader())
            .map(resposta => resposta.json())
            .catch(erro => {throw new Error(erro.message)});
    }
}

```

**Listagem 6 – Código da Service.ts**

Como pode-se observar na Listagem 6, o método *getToken()*, responsável por obter da sessão o *token* de validação do usuário, que é armazenado ao realizar o *login*, juntamente com o *getHeader()*, são os responsáveis por montar o cabeçalho das requisições. Neste caso, foi utilizado o *sessionStorage* para armazenamento do *token*, pois ele armazena apenas durante a sessão do usuário ou enquanto a aba/navegador não for fechada. Para alguns casos foi utilizado o *localStorage*, como, para armazenar o último estabelecimento e usuário validado, as configurações de endereço de IP e porta de acesso ao servidor. Isso porque com o *LocalStorage* os dados ficam armazenados no dispositivo, vinculados ao domínio do usuário, permitindo recuperar essas informações em um próximo acesso.

Para os demais métodos, observa-se a definição *Observable*, que é uma coleção que funciona de forma unidirecional, ou seja, emite notificações sempre que ocorre uma mudança em um de seus itens e a partir disso é possível executar uma ação. Como o desenvolvimento a partir do Angular 4 foi construído pensando em aplicações reativas, o *Observable* foi adotado como padrão, sendo abandonado o uso de *Promises*.

Juntamente com o *Observable*, está o retorno das requisições que foi definido como *any*. Isto permite tornar as funções dinâmicas, uma vez que o retorno pode ser de qualquer um, deixando para essa atribuição o tipo de retorno ser definida em cada serviço dos componentes.

A definição *@Injectable* define a classe como um serviço injetável. Esses provedores registrados com o injetor raiz do aplicativo, uma vez criada, uma instância do serviço vive durante a vida do aplicativo e o Angular injeta essa instância em cada classe que é necessário. Na Listagem 7 estão os provedores registrados na aplicação desenvolvida.

```

providers: [{provide: ErrorHandler, useClass: IonicErrorHandler},
            AppService, AuthenticationService, Config, Service, Utils,

```

```

UserService, RegisterService, ProdutosService,
CategoriaService,
CategoriaItemService, PedidoItemService, PizzaService,
ComandaService, ConfiguracoesService, AgrupamentoService]

```

#### Listagem 7 – Provedores registrados na aplicação

A seguir, as Listagens 8 e 9 apresentam um dos componentes fazendo uso do *Service.ts* e tratando o retorno de uma requisição realizada com *Observable*.

Na Listagem 8 está a classe *CategoriaService.ts*, que apenas define qual será o tipo de dados retornado ao nosso componente, no caso, um *array* de categoria, fazendo uso dos métodos disponibilizados pelo *Service.ts*.

```

import { Injectable } from '@angular/core';
import { Observable } from 'rxjs/Rx';
import 'rxjs/add/operator/map';

import { Service } from '../pages/common/service';
import { Categoria } from '../models/categoria';

@Injectable()
export class CategoriaService {

    constructor(private service: Service) {}

    findAll(): Observable<Categoria[]> {
        return this.service.findAll('categorias');
    }
}

```

#### Listagem 8 – Código da CategoriaService.ts

Na listagem 9 está o componente *Categorias*, que é quem utiliza e manipula os dados retornados do *WebService*. Um dos operadores mais importantes do *Observable* é o *subscribe*, que é o equivalente ao *then* de uma *promise*. Na Listagem 4 está sendo chamada a função *findAll()* criada anteriormente em *CategoriaService.ts* e foi definido um *subscribe*, ou seja, assim que a resposta da requisição vier e for transformada em *JavaScript Object Notation* (JSON) no *Service.ts*, haverá uma notificação e o *array categorias* receberá a lista de categorias, ou caso ocorra um erro, ele poderá ser tratado.

```

@Component({
    templateUrl: 'categorias.html',
})
export class Categorias {
    nodes = [];

    categorias: Categoria[];
    ecommerce: number;
    comanda: number;
    barras: string;
}

```

```

    constructor(public nav: NavController,
                 private utils: Utils,
                 public navParams: NavParams,
                 public loadingCtrl: LoadingController,
                 private service: CategoriaService) {
        this.ecommerce = navParams.get('ecommerce');
        this.comanda = navParams.get('comanda');
        this.barras = navParams.get('barras');
        service.findAll().subscribe(
            categorias => {
                this.categorias = categorias;
                barras => this.barras;
            },
            error => {
                this.utils.AlertError('Erro ao carregar as categorias: ' +
error.message)
            });
        }
    }
}

```

**Listagem 9 – Código da Categorias.ts**

A definição *@Component* é quem define a classe como um componente angular. Cada componente é uma parte isolada do aplicativo, responsável por sua própria interface de usuário e com uma interface programática bem definida, permitindo controlar seu comportamento e seus ciclos de vida.

Um componente deve pertencer a um *NgModule* para que ele possa ser utilizado pelo aplicativo ou por outro componente. Para definir que um componente é membro de um *NgModule*, é necessário adicioná-lo as *declarations* do mesmo. Na Listagem 10 estão todos os componentes criados e utilizados pelo aplicativo.

```

@NgModule({
  declarations: [
    MyApp,
    Users,
    Configuracoes,
    Register,
    Comandas,
    Produtos,
    Popover,
    Categorias,
    ProdutosCategoria,
    PedidoItem,
    ProdutoCons,
    CompAdicional,
    Troca,
    Agrupamento,
    Pizzas,
    PopoverPizza,
    Composicao,
    Sabor
  ],
  ...

```

### Listagem 10 – Código da CategoriaService.ts

Na Listagem 11 está o *template* de um componente. Apenas um *template* pode ser definido por componente, podendo ser este um arquivo externo, como, no exemplo, utilizando-se da diretiva *templateUrl*, ou então a diretiva *template*, na qual o modelo é criado no próprio componente.

```
<ion-header>
  <ion-navbar>
    <button ion-button menuToggle>
      <ion-icon name="menu"></ion-icon>
    </button>
    <ion-title>{{ barras }} - Produtos</ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding class="im-wrapper scroll-content ionic-scroll has-
header">
  <ion-list>
    <ion-item *ngFor="let categoriaItem of categoriaItens"
(click)="setProduto(categoriaItem, ecommerce)">
      <ion-label>
        {{ categoriaItem.produto.produto.nome }}
      </ion-label>
      <ion-label class="ionQtd">
        {{categoriaItem.produto.valor|number:'.2'}}
      </ion-label>
    </ion-item>
  </ion-list>
</ion-content>
```

### Listagem 11 – Código da CategoriItem.html

Na implementação apresentada na Listagem 11 é possível perceber que o Angular permite também trabalhar com diretivas na *view*, como é o caso do *ngFor*, que permite criar listas e tabelas para apresentação dos dados no *Hyper Text Markup Language* (HTML).

Todas as requisições para o *Webservice*, manipulação de retorno e apresentação dos dados para o usuário seguiram esse padrão. As requisições funcionando de forma assíncrona permitem o trabalho com múltiplos eventos, sendo que a ordem de execução não é um fator determinante para o funcionamento, não tendo operações bloqueantes.

A seguir é apresentado sobre a implementação do *Webservice*. Para o seu desenvolvimento foi utilizada a linguagem Java, juntamente com o framework *Spring Boot* e o gerenciador de dependências *Maven*, que a partir das bibliotecas descritas no *pom.xml*, se encarrega de fazer os *downloads* e gerenciá-los. O serviço vai, basicamente, disponibilizar os JSONs para consumo na aplicação *mobile* e realizar a persistência dos pedidos, seus itens e sua composição.

O *Spring Boot* é uma ferramenta de desenvolvimento de aplicações que utilizam o framework *Spring*, sem que seja necessária praticamente nenhuma configuração, sendo capaz de identificar quais as principais características da aplicação e fazer automaticamente as configurações, utilizando todos os recursos existentes no *Spring*.

Para segurança da aplicação foi utilizado a *Spring Security* e *JSON Web Token (JWT)*. O JWT é um padrão aberto para transmissão de informações de forma segura entre partes. A grande vantagem é que toda informação referente ao usuário autenticado é guardada em um *token*. Assim, toda vez que uma requisição é realizada para o *Webservice* esse *token* é enviado, não sendo necessário ir ao banco de dados buscar informações sobre o usuário autenticado. O *Spring Security* é um *framework* que provê autenticação, autorização e diversas outras funcionalidades para aplicações Java e JavaEE.

Na listagem 12 estão as bibliotecas do *Spring Boot* e *JWT* adicionados ao *pom.xml*.

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

<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt</artifactId>
  <version>0.6.0</version>
</dependency>
```

**Listagem 12 – Bibliotecas de segurança adicionadas ao pom.xml**

O *Spring Security* permite definir quais rotas da aplicação obrigam autenticação e quais estão disponíveis mesmo sem um usuário autenticado. Para definir essas restrições de acesso foi criado a classe *WebSecurityConfiguration*, que é uma classe filha de *WebSecurityConfigurerAdapter* do *Spring Security*.

Na Listagem 12 pode ser observado que todos podem acessar */auth*, */create*, */register* e */login* sem realizar autenticação e para as demais rotas, a autenticação passa a ser necessária.

```
@Override
protected void configure(HttpSecurity httpSecurity) throws Exception {
    httpSecurity.csrf().disable().exceptionHandling().authenticationEntryPoint(
        authenticationEntryPoint).and().sessionManagement().sessionCreationPolicy(
        SessionCreationPolicy.STATELESS).and().authorizeRequests().antMatchers(
        HttpMethod.OPTIONS, "/*").permitAll()
        .antMatchers("/auth/*").permitAll()
        .antMatchers("/create/*").permitAll()
        .antMatchers("/register/*").permitAll()
        .antMatchers("/login/*").permitAll()
        .anyRequest().authenticated();

    httpSecurity.addFilterBefore(authenticationTokenFilterBean(),
        UsernamePasswordAuthenticationFilter.class);
```



```
}

```

#### Listagem 12 – Código da WebSecurityConfiguration.java

A rota `/auth` será utilizada para realizar a autenticação do usuário e sua respectiva senha. Enquanto que `/create` será utilizada para realizar o cadastro de novos usuários. As rotas `/register` e `/login` serão requisições GET, que disponibilizam a lista dos estabelecimentos, dos usuários já cadastrados e dos colaboradores que ainda não possuem uma autenticação definida.

A Listagem 13 apresenta um *controller* responsável por retornar uma categoria ou um *array* de categorias utilizados posteriormente no componente `Categorias.ts`.

```
@RestController
@RequestMapping("/SudoSoft")
public class CategoriaController {

    @Autowired
    private CategoriaRepository categoriaRepository;

    @GetMapping("/categorias/{id}")
    public Categoria visualizar(@PathVariable Integer id){
        return categoriaRepository.findOne(id);
    }

    @GetMapping("/categorias")
    public List<Categoria> todos(){
        return categoriaRepository.findAll();
    }
}

```

#### Listagem 13 – Código da CategoriaController.java

## 4 CONSIDERAÇÕES FINAIS

O objetivo do desenvolvimento deste trabalho foi a implementação de um sistema *mobile* que auxilie o atendimento dos garçons em bares, lanchonetes, restaurantes e estabelecimentos do gênero. O aumento da oferta de sistemas nesta área torna os usuários cada vez mais exigentes, buscando um sistema que possibilite que o usuário possa realizar suas tarefas de maneira fácil, rápida e satisfatória. A interface do aplicativo foi desenvolvida utilizando conceitos e padrões de usabilidade, o que permite ao usuário saber o que está fazendo, quais as ações ele deve tomar e tendo claros os resultados das ações realizadas.

Para o desenvolvimento *mobile* foi utilizada uma tecnologia híbrida para aplicações empresariais. Como o seu consumo não atinge um grande volume de usuários, essa tecnologia se mostrou bastante eficiente, uma vez que o mesmo código pode ser compilado para diversas plataformas, permitindo atender maior diversidade de mercado em termos de dispositivos móveis. A utilização das tecnologias Ionic Framework e Angular Framework apresentam bastante facilidade no desenvolvimento, em virtude de possuir documentação extensa e com diversos exemplos de aplicação. E, ainda, por não haver a necessidade do conhecimento em desenvolvimento com JavaScript, uma vez que a utilização do TypeScript se assemelha muito ao desenvolvimento em Java.

Na implementação do projeto *back-end* em Java, a utilização do *Spring Boot* e *Maven* apresentou benefícios e agilidade na configuração do projeto, o que anteriormente era uma das principais dificuldades entre os programadores que iniciavam com a tecnologia Java. Essas tecnologias apresentam, ainda, grande agilidade no desenvolvimento Java para *web* com a utilização do Spring Framework e seus projetos, como, por exemplo, o Spring Data e Spring Security.

Considerando o objetivo deste trabalho, é possível afirmar que o resultado atingiu as expectativas e objetivos propostos. Além de que as tecnologias utilizadas se mostraram com grande potencial para o desenvolvimento, de forma produtiva e com qualidade nos resultados apresentados.

## REFERÊNCIAS

SEBRAE. **Nota conjuntural: setor de alimentos.** 2011. Disponível em: <[https://m.sebrae.com.br/Sebrae/Portal%20Sebrae/UFs/RJ/Menu%20Institucional/Sebrae\\_SE T\\_dez12\\_alim.pdf](https://m.sebrae.com.br/Sebrae/Portal%20Sebrae/UFs/RJ/Menu%20Institucional/Sebrae_SE T_dez12_alim.pdf)>. Acesso em: 23 nov. 2017.

REBOUÇAS, Cibelle. **Faturamento do setor de alimentação sobe 9,3% em 2016, diz associação.** Disponível em: <<http://www.valor.com.br/empresas/4859964/faturamento-do-setor-de-alimentacao-sobe-93-em-2016-diz-associacao>>. Acesso em: 23 nov. 2017.