UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CÂMPUS GUARAPUAVA
CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

VINÍCIUS BAIL ALONSO

EICHEF: SISTEMA PARA AUTOMAÇÃO DE ATENDIMENTO EM BARES E RESTAURANTES

PROJETO DO TRABALHO DE CONCLUSÃO DE CURSO

GUARAPUAVA

2015

VINÍCIUS BAIL ALONSO

EICHEF: SISTEMA PARA AUTOMAÇÃO DE ATENDIMENTO EM BARES E RESTAURANTES

Projeto de Trabalho de Conclusão de Curso de graduação, apresentado a disciplina de Trabalho de Conclusão de Curso 1 do Curso Superior de Tecnologia em Sistemas para Internet - TSI - da Universidade Tecnológica Federal do Paraná - UTFPR - Câmpus Guarapuava, como requisito parcial para obtenção do título de Tecnólogo em Sistemas para Internet

Orientador: Prof. Dr. Diego Marczal

GUARAPUAVA

2015

RESUMO

ALONSO, Vinícius. EICHEF: Sistema para Automação de Atendimento em Bares e Restaurantes. 41 f. Projeto do Trabalho de Conclusão de Curso – Curso de Tecnologia em Sistemas para Internet, Universidade Tecnológica Federal do Paraná. Guarapuava, 2015.

Em muitos bares e restaurantes, os pedidos dos clientes são feitos de forma tradicional. O garçom vai até a mesa com papel e caneta, anota o pedido, repassa aos cozinheiros e após estar tudo pronto o pedido é entregue ao consumidor. Embora esse seja um processo simples alguns fatores podem causar um desconforto ao cliente. Como por exemplo, um pedido anotado errado ou que demora para chegar. Com isso, surge a necessidade de automatizar a solicitação de pedidos de clientes de modo a tentar reduzir as falhas humanas. Isso, pode ser feito por meio de uma aplicação web, que apresentaria o menu do estabelecimento ao consumidor e enviaria seu pedido diretamente aos funcionários que irão prepará-lo. E após o cliente consumar seu pedido, utilizando-se da mesma tecnologia poderia avaliar a qualidade dos pratos servidos e do atendimento prestado. Nesse cenário, o trabalho do garçom seria reduzido a entregar os pedidos. Por meio de um sistema como esse, dados podem ser coletados, como por exemplo, prato mais pedido e melhor avaliado, que posteriormente serão disponibilizados ao gerente na tentativa de auxiliá-lo na tomada de decisões. A aplicação busca melhorar a forma como os consumidores fazem pedidos e diminuir os erros humanos.

Palavras-chave: Atendimento ao cliente. Automatização. Falhas no atendimento. Tomada de decisões.

ABSTRACT

ALONSO, Vinícius. EICHEF: System to Automation of Treatment in Bars and Restaurants. 41 f. Projeto do Trabalho de Conclusão de Curso – Curso de Tecnologia em Sistemas para Internet, Universidade Tecnológica Federal do Paraná. Guarapuava, 2015.

In many bars and restaurants, the orders of clients are made traditionally. The waiter goes to the table with pen and paper, write down the request, passes the cooks and be all set after the request is delivered to the consumer. Although this is a simple process some factors can cause discomfort to the client. As an example, an application noted wrong or it takes to arrive. With that comes the need to automate the request of customer orders in order to try to reduce human error. This can be done through a web application, which present the property to the consumer menu and send your request directly to the employees who will prepare it. And after the client complete his request, using the same technology could evaluate the quality of food served and the service provided. In this scenario, the work of the waiter would be reduced to deliver the order. Through such a system, data can be collected, such as dish request more and better assessed, which will later be made available to the manager in an attempt to assist in decision making. The application seeks to improve the way consumers make requests and reduce human errors.

Keywords: Customer service. Automation. Failures in care. Decision-making.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

API	Application Programming Interface (Em português: Interface de Programação de Aplicações)
CSS	Cascading Style Sheets (Em português: Folha de Estilo em Cascata)
DOM	Document Object Model (Em português: Modelo de Objeto de Documentos)
DRY	Don't Repeat Yourself (Em português: Não se Repita)
HTML	Hypertext Markup Language (Em português: Linguagem de Marcação de Hipertexto)
IHC	Interação Humano Computador
JSON	Javascript Object Notation (Em portugês: Notação de Objeto Javascript)
ORM	Object Relational Mapper (Em português: Mapeamento Objeto Relacional)
POS	Point of Sale (Em português: Ponto de Venda)
QR Code	Quick Response Code (Em portugês: Código de Resposta Rápida)
SPA	Single Page Application (Em português: Aplicação de Página Única)
UML	Unified Modeling Language (Em português: Linguagem de Modelagem Unificada)
XML	Extensible Markup Language (Em português: Linguagem de Marcação Extensiva)

SUMÁRIO

1 INTRODUÇÃO

Um dos principais fatores utilizados por estabelecimentos de consumo para fidelizar seus clientes é o atendimento. Ter qualidade nesse aspecto tornou-se um fator crucial para aqueles que almejam satisfazer seus clientes. Isso também pode ser visto como um grande diferencial perante os concorrentes, pois oferecer um serviço de qualidade, é uma forma de obter vantagem competitiva (GARCEZ et al., 2000).

Segundo OLIVEIRA (2002), em estabelecimentos como restaurantes podem ocorrer diversos tipos de falhas que prejudiquem o serviço ao cliente. Como por exemplo, uma máquina de cartões de crédito pode quebrar, um garçom pode cometer um erro simples ao anotar o pedido ou no momento de entregar o pedido na mesa.

Essas falhas podem fazer com que o estabelecimento perca clientes, pois, estes julgam a credibilidade de um estabelecimento com base em suas experiências anteriores. Alguns dos problemas acima citados são difíceis de prever, porém, os problemas relacionados ao garçom podem ser previstos.

O uso adequado da tecnologia pode ser uma solução, retirando do garçom muitas responsabilidades e com isso evitando que esses profissionais fiquem sobrecarregados. Tarefas como, pedir um prato pode ser passada diretamente a quem vai prepará-lo, pedir uma bebida pode ser passada direto ao bar, evitando assim, que o garçom tenha que andar até a mesa mais vezes do que o necessário.

O principal desafio está em projetar um sistema que automatize essas tarefas sem complicar um processo que já é muito simples. Como também, não alterar muito a rotina dos restaurantes e bares que farão uso desse software.

Outro aspecto que merece ser levado em consideração é que o sistema não deve ter um custo elevado para sua implantação, além disso, necessita-se ter cuidado, para que não acabe atrapalhando as atividades do estabelecimento ao invés de auxiliar.

1.1 OBJETIVOS

1.1.1 OBJETIVO GERAL

Automatizar os pedidos realizados dentro de bares e restaurantes, por um sistema web acessado por dispositivos móveis dos clientes ou ainda oferecido pelo estabelecimento.

1.1.2 OBJETIVOS ESPECÍFICOS

- Auxiliar clientes de bares e restaurantes a fazer seus pedidos de forma eletrônica mediante a um dispositivo móvel, sem a necessidade de chamar um garçom;
- Disponibilizar para os clientes, um menu do estabelecimento, em que será possível avaliar os pratos servidos;
- Fornecer aos garçons e aos cozinheiros a relação de pedidos realizados pelos clientes;
- Fornecer ao gerente relatórios para auxiliar a tomada de decisões estratégicas, como por exemplo pratos mais pedidos;
- Apresentar pratos mais vendidos e melhor avaliados para os clientes, através de um menu interativo.

1.2 DIFERENCIAL TECNOLÓGICO

Neste tópico destacam-se os diferencias tecnológicos do trabalho proposto em relação ao estado da arte. Primeiramente tratando-se de uma aplicação web, o cliente deveria digitar o endereço do site para acessá-lo, porém, isso acabaria gerando desconforto, uma certa demora e dificuldade no acesso ao menu do restaurante. Uma boa alternativa para isso é a utilização de *Quick Response Code* (QR Code).

Segundo Waters (2012), um QR Code é um código de barras bidimensional, em que é possível guardar alguma informação para ser acessada de forma rápida. Essa informação pode ser um vídeo para ser exibido, uma música para ser tocada, entre outras milhares de possibilidades. No sistema EICHEF o QR Code irá abrir o site e informar o número da mesa. Isso, além de possibilitar mais conforto ao cliente, facilita a identificação da mesa que foi feito o pedido.

Outra técnica utilizada será *Single Page Application* (SPA). Que possui algumas vantagens em relação a aplicativos nativos, como por exemplo: é multi-plataforma, dependendo

apenas do browser; permite gerenciar o estado do cliente, favorecendo a aplicações offline; e, por último não necessita de instalação (FINK; FLATOW, 2014).

Como último diferencial, o sistema também possuirá um *ranking* com os pratos mais vendidos. Útil para o cliente, na decisão do pedido. Além disso, o sistema visa fornecer alguns relatórios ao gerente, como por exemplo tempo de espera desde o pedido até a entrega do prato. Essas informações podem ajudá-lo, a definir possíveis melhoras no atendimento e na elaboração de pratos.

2 RESENHA LITERÁRIA

Nesse capítulo serão abordados os trabalhos já desenvolvidos na área do projeto. Assim, como as tecnologias que serão utilizadas futuramente no desenvolvimento do projeto.

2.1 ESTADO DA ARTE

Nesta seção serão abordados trabalhos correlatos, sendo apresentado os mais relevantes a essa pesquisa e desenvolvimento.

O primeiro sistema estudado foi o eZee iMenu (EZEE TECHNOSYS, 2015b), desenvolvido pela empresa eZee Technosys. Este é um aplicativo que possui uma versão para iOS (sistema operacional móvel da Apple, originalmente iPhone OS) e outra para Android (sistema operacional móvel baseado no kernel do Linux), por esses, é possível que o cliente de um estabelecimento faça pedidos do seu próprio celular. Mas não se limitando apenas aos aparelhos dos clientes, caso seja necessário, o estabelecimento também pode adquirir tablets e quando seus clientes estiverem fisicamente no estabelecimento, um tablet pode ser fornecido como se fosse uma carta de menu. Assim, os clientes podem realizar seus pedidos diretamente por ele. O sistema está ilustrado abaixo, na Figura 1.

Figura 1: Ezee iMenu.

Fonte: http://fortle-telecoms.com/products.php

Segundo o site oficial do eZee iMenu[1], este possui as funcionalidades relacionadas abaixo (EZEE TECHNOSYS, 2015a):

- Customização de tema;
- Menu interativo;
- Modificador de items;
- Integração com sistemas *Point of Sale* (POS);
- Modo *offline*;
- *Feedback* do convidado;
- Observações em pedidos ou itens;
- Sincronização;
- Busca rápida;
- Exportação de menu.

Além dessas funcionalidades, o sistema também possui três modos de operação: 1) modo de visualização; 2) modo de pedido do convidado; 3) e, modo do garçom (EZEE TECHNOSYS, 2015b).

O **modo de visualização** é apenas para que o cliente navegue pelo menu interativo, sem poder fazer pedidos. Nesse modo o garçom fica responsável por fazer os pedidos em nome do cliente.

No **modo de pedido**, o cliente pode fazer seus próprios pedidos, com isso a responsabilidade do garçom fica apenas em servir a mesa e depois recolher o valor do pagamento da conta.

O **modo garçom** é utilizado pelo garçom para executar seu trabalho, da mesma forma que faria utilizando papel e caneta, porém, quando ele marcar um pedido o mesmo irá aparecer instantaneamente para as pessoas responsáveis por preparar as refeições na cozinha.

Outro sistema estudado foi o Eped (PEKUS, 2015), um aplicativo desenvolvido pela empresa Pekus Consultoria para ser utilizado em tablets com o sistema operacional Android.

[1] Site Official `http://www.ezeeimenu.com/features.php`

Por esse aplicativo é possível que o cliente faça pedidos por um tablet fixado à mesa ou oferecido por um funcionário.

O Eped é semelhante em alguns aspectos com o eZee iMenu, ambos oferecem um menu interativo para que o usuário possa fazer pedidos, além disso, o Eped possui algumas funcionalidades como por exemplo, ser utilizado sem integração com sistema de caixa do estabelecimento e também ser traduzido para três idiomas português, inglês e espanhol. Abaixo na Figura 2, uma ilustração do sistema.

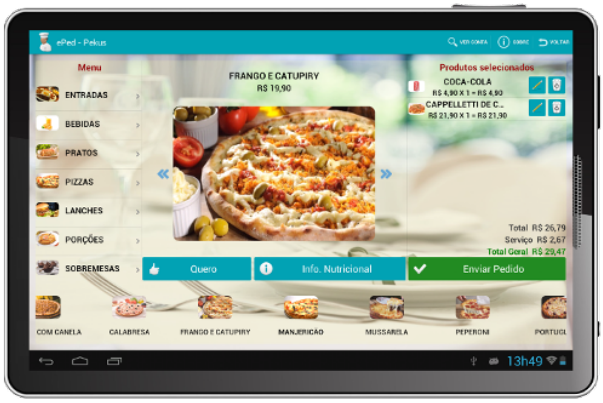

Figura 2: Eped.

Fonte: http://www.pekus.com.br/cardapio.aspx

O último sistema analisado foi o Psiu Garçom (PSIU GARÇOM, 2015), diferente dos outros dois antes mencionados, ele trabalha com o propósito principal de chamar o garçom "[...] é composto por transmissores individuais sem fio posicionados nas mesas do estabelecimento e por um painel de leds, instalado em um local de fácil visualização [...]" (PSIU GARÇOM, 2015, p. principal). Dessa maneira com isso, a mesa do cliente ganha um botão para chamar o garçom, emitindo um sinal sonoro e exibindo o número de sua mesa em um painel eletrônico. O número só irá sair do painel após o garçom que atendeu pressionar o outro botão que está na mesa do cliente. O sistema é ilustrado abaixo, na Figura 3

Figura 3: Psiu Garçom.

Fonte: http://gestaoderestaurantes.com.br/blog/index.php/2009/07/17/psiu-garom-lana-nova-mdia-na-fipan-2009/

Além de facilitar o atendimento, o Psiu Garçom também possui outras duas funcionalidades, ele exibe até 99 (noventa e nove) mensagens publicitárias no painel eletrônico e também cronômetra o tempo de atendimento às mesas fornecendo relatórios para o gerente.

O Psiu Gaçom cumpre bem com funcionalidade de chamar os garçons, porém com relação ao atendimento, está limitado somente a isso. O eZee iMenu e o Eped são bem completos, porém seu design poderia ser mais simples para o usuário, tornando os sistemas mais intuitivos. Isso pode ser feito removendo conteúdo desnecessário da tela de pedidos e deixando apenas o que é preciso no momento de se fazer um pedido. Abaixo na Tabela 1 é possível ver as diferenças mais relevantes a esse projeto.

Tabela 1: Diferenças mais relevantes entre os sistemas

	***Feedback* do Cliente**	**Cronometro da entrega**	**Acompanhamento do pedido**
Ezee iMenu	Possui	Não possui	Não possui
Eped	Não possui	Não possui	Não possui
Psiu Garçom	Não possui	Não possui	Não possui
Eichef	Possuirá	Possuirá	Possuirá

Os sistemas acima citados, cumprem seu propósito, que é facilitar o atendimento e como os pedidos são feitos. E por essa razão o EICHEF, sistema proposto, irá considerar as funcionalidades desses sistemas como base para seu desenvolvimento, como por exemplo, o menu interativo e o *feedback* do cliente. Porém o EICHEF terá alguns diferencias, como por

exemplo, o cliente poderá acompanhar o estado do seu pedido. Também será possível saber o tempo que o pedido demorou para ser entregue. Com essas informações o gerente pode saber em que ponto o estabelecimento pode melhorar seu atendimento.

2.2 REFERÊNCIAL TEÓRICO

2.2.1 SINGLE PAGE APPLICATION

Segundo Fink e Flatow (2014), o SPA é um conceito que busca melhorar a performance das aplicações *web*. Na abordagem tradicional da *web*, um cliente, que pode ser um navegador ou uma outra aplicação, faz uma requisição ao servidor. Em seguida o servidor deverá processar todo o conteúdo estático da página, tais como, *Hypertext Markup Language* (HTML), *Cascading Style Sheets* (CSS), Javascript, entre outros, e retornar ao cliente. O problema nessa abordagem é que o servidor deverá repetir esse ciclo a cada nova requisição.

A abordagem do SPA é um diferente. O servidor deverá responder apenas a primeira requisição com todos os arquivos estáticos. Nas outras requisições deverá responder utilizando *Javascript Object Notation* (JSON), e delegando a responsabilidade de apresentar os dados de uma forma elegante ao cliente (SCOTT, 2015).

Dentre as principais vantagens estudadas, destacam-se:

- **O navegador não precisa recarregar a página**. Ao invés disso, o *framework* do *client-side* pegará o conteúdo do JSON utilizando ajax e irá inseri-lo na página;
- **A lógica envolvida na apresentação dos dados fica a cargo do cliente**. Por meio disso, é possível retirar um pouco da carga do servidor;
- **Transações são feitas no servidor**. O *framework client-side*, deverá enviar os dados para a *server-side*, aonde serão feitas operações que envolverão banco de dados e regras de negócio da aplicação.

Para se conseguir chegar a isso, é necessário dividir as responsabilidades entre o *server-side* e o *client-side*. No *server-side* será necessário que trabalhe como uma API, executando as regras de negócio e respondendo de uma forma adequada. E no *client-side*, será responsável por apresentar os dados. Será preciso utilizar um *framework* adequado para ambas as partes.

2.2.2 FRAMEWORK SERVER-SIDE

Para o desenvolvimento referente ao *server-side*, será utilizado o *framework* Ruby on Rails. Como o projeto será desenvolvido seguindo práticas do *framework* SCRUM, foi escolhido uma ferramenta que utilize conceitos de desenvolvimento ágil.

Ruby on Rails é um *framework* para desenvolvimento web. Criado para tornar o desenvolvimento rápido e manter o código limpo. Para isso, o *framework* segue algumas boas práticas. Como por exemplo, convenção sobre configuração e *Don't Repeate Yourself* (DRY).

A filosofia do DRY, diz respeito a não repetir código. O Rails é um *framework* escrito com a linguagem Ruby, e aproveita-se do poder a linguagem para evitar duplicação de código. Além da linguagem, o padrão MVC implementado no *framework*, permite colocar cada funcionalidade em um lugar especifico evitando duplicações.

Convenção sobre configuração torna o *framework* ágil pelo fato de tornar a configuração para desenvolver uma aplicação miníma. Diferente de outras tecnologias aonde é necessário configurar diversos arquivos *Extensible Markup Language* (XML). O Rails traz várias configurações por *default* e por meio das convenções é possível sobrescrever-las com facilidade (RUBY et al., 2013, p. 21-23).

Na aplicação proposta o *framework* Ruby on Rails, será responsável pelas regras de negócio da aplicação. Seguindo os principios do SPA, o Rails será uma *Application Programming Interface* (API) por meio da qual o *framework frontend* utilizado deverá se comunicar.

2.2.3 FRAMEWORK CLIENT-SIDE

Tratando-se de uma aplicação que segue o conceito SPA. É importante utilizar um *framework* apropriado para o *client-side*. A função desse *framework*, será se comunicar com a API que será desenvolvida utilizando Ruby on Rails e apresentar os dados em tempo real.

Com esse propósito, foram estudados alguns candidatos, entre eles destacam-se o Angularjs e o Emberjs.

2.2.3.1 ANGULARJS

O Angularjs é um *framework* escrito em Javascript para trabalhar no *client-side*, tornando o desenvolvimento mais rápido e fácil. Foi desenvolvido pelo Google[2] que o usa em

[2]Multinacional norte-americana especializada em serviços de internet

projetos como Gmail, Google Maps e Google Calendar.

Até o momento da escrita desse projeto, o AngularJs é mantido pela própria Google e por uma grande comunidade de desenvolvedores. O *framework* foi criado primeiramente para construir aplicações SPA. Além disso, ele abstrai diversas funcionalidades comuns a aplicações web modernas. Como por exemplo:

- Separação da lógica, classes e *views*;
- Serviços com Ajax;
- Injeção de dependências;
- Testes automatizados.

Fora as citadas acima, o Angularjs possui muitas funcionalidades que não foram relacionadas (LERNER, 2013, p. 8-9).

Ele possui diversas vantagens de uso, entre elas destacam-se:

- **Modificar o *Document Object Model* (DOM) de forma simples**, sem o uso de métodos observáveis como os *listeners* do Javascript. Para isso o *framework* usa **diretivas**, ***data-binding*** e **expressões** que se misturam ao código HTML. Dessa forma, o desenvolvedor se preocupa com a apresentação dos dados. Deixando o trabalho de como fazer isso com o *framework* (STARZHINSKY, 2015).
- **É organizado**, consegue serapar muito bem a lógica da apresentação dos dados. Isso porque segue o padrão *Model-View-Controller*. Isso torna o aplicação mais facil para futuras manutenções (LERNER, 2013, p. 8-9).
- **Possui um bom suporte a testes automatizados**. O Angularjs adotou o *framework* para testes Jasmine[3]. Graças a isso, é possível testar uma aplicação, desde testes unitários até testes de integração. Além disso, possui suporte a *mocks*[4]. Garantindo que as funcionalidades implementadas estão funcionando (KOZLOWSKI, 2013, cap. 2).

O *framework* também possui algumas desvantagens. Entre as estudadas, a que mais se destaca é o fato do *framework* ser muito grande e oferecer diversas formas de se chegar ao mesmo resultado. Isso pode ser muito confuso e trabalhoso para um desenvolvedor que está inciando com o Angularjs (STARZHINSKY, 2015).

[3] `http://jasmine.github.io/`

[4] Objetos dublês, usados para simular objetos reais.

2.2.3.2 EMBERJS

Segundo Kelonye (2014) o Emberjs é um *framework client-side* divertido e *open source* escrito em Javascript. É usado para criar aplicações ambiciosas e complexas. Foi construido a partir do SproutCore[5] criado por Yehuda Katz e Tom Dale. O Emberjs usa conceitos como MVC e convenção sobre configuração, o que torna o desenvolvimento mais organizado. Tem sido adotado por empresas do mundo todo, entre elas destacam-se a Apple, Groupon, Square, Zendesk e a Tilde.

Utilizar essa ferramenta pode trazer muitos beneficios ao desenvolvedor. Segundo Cravens e Brady (2014, p. 7-8), entre os beneficios destacam-se:

- **Oferece um ótimo recurso de *Object Relational Mapper* (ORM)**, que permite traduzir dados de um tipo de sistema para outro. Isso permite por exemplo, que dados que estão em objetos Javascript, sejam salvos no Mysql[6] sem complicações.
- **Permite o que é chamado de *developer ergonomics***. Graças as convenções que o *framework* usa, isso permite um desenvolvimento ergonômico. Por exemplo, o desenvolvedor não precisa configurar qual template será carregado. Se o template seguir as convenções ele será carregado automaticamente. Além disso, o Emberjs possui muitos comportamentos por *default*, o que tira um pouco da carga de trabalho da equipe de desenvolvimento.

O Emberjs traz varias funcionalidades e conveções. Isso pode facilidar muito o trabalho de um desenvolvedor. Porém, em contrapartida, também existe um problema. Devido a isso, o *framework* acaba tendo uma curva de aprendizado longa. Dependendo do tempo que se tem para desenvolver o projeto, isso pode tornar o seu uso inviável (SKEIE, 2013, cap. 1).

Apesar de os dois *frameworks* estudados conseguirem cumprir bem o papel no *client-side*. O Emberjs é escolhido, devido a sua boa integração com Ruby on Rails, isso acontece graças ao fato, do Emberjs seguir filosofias comuns. Como por exemplo, convenção sobre configuração.

2.2.4 TEST DRIVEN DEVELOPMENT

O sistema será desenvolvido seguido os príncipios do *Test Driven Development* (TDD). TDD é um ciclo de desenvolvimento, no qual o desenvolvedor deve escrever um teste antes de

[5] *Framework* Javascript criada pela Apple para inovar a experiência do usuário em *web sites*
[6] `https://www.mysql.com`

escrever um código de produção. Ou seja, primeiro é escrito um código que irá testar a nova funcionalidade, antes de escrever a funcionalidade em si (ANICHE; CORBUCCI, 2014).

O TDD baseia-se em três estágios *red-green-refactor*. No estágio ***red***, o teste é escrito, e como a funcionalidade a ser testada não existe, consequentemente o teste irá falhar. Em seguida temos o estágio ***green***, agora a funcionalidade é escrita sem preocupação com conceitos de *clean code*, o importante é apenas passar no teste. Agora com a funcionalidade passando no teste, veem o último estágio ***refactor***. Nessa etapa o desenvolvedor deve refatorar seu código para garantir que fique o melhor escrito possível.

Segundo Aniche e Corbucci (2014, cap. 3) , esse processo pode trazer muitas vantagens para o seu praticante, entre elas destacam-se:

- **Funcionalidade já fica pronta e testada**. Como o desenvolvedor escreve o teste e depois a funcionalidade, isso implica que quando ela estiver pronta já haverá um teste para ela;
- **O desenvolvedor pensa no teste e não na implementação**. Com isso, ele acaba pensando em cenários possíveis que a nova funcinalidade irá encontrar. Cenários nos quais, ele poderia esquecer se estivesse programando a funcinalidade direto;
- ***Feedback* rápido e constante**. Como os testes estão sempre sendo executados a cada ciclo. Caso haja algum problema, poderá ser encontrado e solucionado rapidamente. Graças ao *feedback* fornecido constantemente pelos testes (FREEMAN; PRYCE, 2009).

Embora o TDD traga muitas vantagens, em contrapartida existem desvantagens. A mais importante delas, considerando o projeto proposto é a **experiência**. Em alguns momentos a prática de TDD pode atrapalhar. Nem sempre é recomendado, como por exemplo, para testar *views* ou lógicas muito simples. Porém, o momento de usar ou não depende da experiência do desenvolvedor (ANICHE; CORBUCCI, 2014).

A importância dessa prática, será oferecer *feedback* constante durante o desenvolvimento do projeto proposto. Isso poderá trazer mais segurança para implementar as funcionalidades propostas. Além de facilitar mudanças, caso sejam necessárias.

2.2.5 BEHAVIOR DRIVEN DEVELOPMENT

Behavior Driven Development (BDD) é uma metodologia ágil a qual tem o objetivo de desenvolver sofware melhor e mais rapidamente. Foi criado baseado no TDD e no *Domain*

Driven Development (DDD). Diferente do TDD, que o objetivo é apenas escrever testes e refatorar o código depois. No BDD os testes são escritos para descrever como a aplicação deve se comportar, ou seja, é orientado a comportamento.

Sua principal característica é a possibilidade de descrever o software em uma linguagem natural. Por meio da qual, a equipe de desenvolvimento, os testadores e os *stakeholders*[7], podem se entender com mais facilidade (SMART, 2014).

No BDD as funcionalidades do sistema são descritas em forma de estórias, com participação dos *stakeholders*, por essa razão são escritas em linguagem natural. Com as estórias montadas, é necessário testá-las, para isso são usados cenários, os quais descrevem as situações as quais a funcionalidade será submetida e quais serão suas respostas (ASTELS; CHELIMSKY, 2010).

Dentre as principais vantagens dessa abordagem, destacam-se:

- A nova funcionalidade desenvolvida, fica testada e documentada;
- Promove uma maior interação entre toda a equipe, desenvolvedores, *stakeholders*, entre outros.

Dentre as vantagens acima citadas, a primeira sem dúvidas, é a mais relevante para o projeto proposto. Porque por meio da descrição, será mais fácil de testar as funcionalidades e manter uma boa documentação para futuras extensões ou mudanças.

Os testes serão escritos utilizando a suíte de testes ***RSpec***[8]. É uma biblioteca criada em 2005 por Steven Baker, para escrever testes com pouco código em um contexto controlado. As estórias serão descritas utilizando a biblioteca ***Cucumber***[9], que lê arquivos de funcionalidades, escritos em inglês e os usa para automatizar os testes nos cenários descritos (ASTELS; CHELIMSKY, 2010).

2.2.6 TECNOLOGIAS APRESENTADAS EM FUNÇÃO DO PROJETO

O projeto proposto seguirá as práticas do *framework* SCRUM, sendo assim, as etapas do projeto serão divididas em *sprints*. Seguindo esse princípio, foi escolhido um *framework* para o *server-side* que fosse apropriado para metogolodias ágeis.

[7] São as partes interessadas no software, podem ser clientes, usuários, entre outros.
[8] `http://rspec.info/`
[9] `https://cucumber.io/`

O *server-side* será feito utilizando Ruby on Rails, que funcionará como uma API. Sua função será executar as regras de negócio e responder em JSON para o *framework client-side*, que será o Emberjs. A interação do usuário e a apresentação dos dados, ficará a cargo do Emberjs, que deverá enviar requisições para a API e depois apresentar suas respostas ao usuário.

As funcionalidades serão dividias em histórias, e cada uma será desenvolvida seguindo o ciclo do TDD. Os testes de aceitação e as histórias serão desenvolvidos utilizando a biblioteca *Cucumber*. Para os outros testes, tais como, unitários e de integração, será utilizada a biblioteca *RSpec*.

3 METODOLOGIA

Nesta seção descreve-se a metodologia da solução proposta para o problema apresentado. Os passos metodológicos estabelecidos inicialmente são relacionados e descritos na sequência.

3.1 LEVANTAMENTO DOS REQUISITOS DO SISTEMA

Seguindo os conceitos do BDD, os requisitos do sistema serão descritos em forma de histórias. Essas histórias irão descrever os usuários interagindo com o sistema em alguns cenários. E quais serão as respostas que o sistema deverá retornar em cada cenário. Essas histórias serão escritas em linguagem natural. E não envolverão nenhum conceito de implementação interna.

Ao final dessa etapa, deve-se tem em mãos todas as funcionalidades do sistema descritas, para planejamento e implementação na etapas à diante.

3.2 IDEALIZAÇÃO DA ARQUITETURA DO SISTEMA

Com as histórias em mãos, a próxima etapa será planejar o funcionamento interno do sistema. Visando ter um código modular, a aplicação será dividida em módulos. Cada módulo será responsável por uma parte da aplicação. Conforme a complexidade de cada módulo, poderão ser usados diagramas da Linguagem de Modelagem Unificada[1] (UML) para ajudar no planejamento.

Após terminar essa etapa, o funcionamento interno da aplicação deverá estar planejado.

[1] Do Inglês: Unified Modeling Language

3.3 ESTUDO DAS TECNOLOGIAS À SEREM UTILIZADAS

Nessa etapa serão estudadas as tecnologias a serem empregadas no projeto. Foram reunidos livros para esta etapa. O objetivo dela será criar familiaridade com as tecnologias e práticas sugeridas para o projeto.

3.4 DESENVOLVIMENTO DO SISTEMA

Essa etapa será dividida em *sprints*, planejadas junto com o orientador. As histórias receberão uma nota durante o planejamento das *sprints*, que representará sua importancia para o projeto. Após isso, serão escolhidas quais histórias serão implementadas na *sprint* corrente.

Após uma nova funcionalidade ser finalizada ao final de cada *sprint*, as funcionalidades desenvolvidas serão integradas ao projeto. O código fonte da aplicação ficará em um repositório privado no *Bitbucket*[2]. Além disso, uma versão parcial da aplicação estará hospedada no *Heroku*[3].

Seguindo o ciclo do TDD, durante essa etapa serão escritos testes unitários e de integração. Os testes de aceitação, serão deixados para a próxima etapa.

O objetivo a ser alcançado nessa etapa, é ter a aplicação proposta implementada com as funcionalidades descritas na primeira etapa.

3.5 VALIDAÇÃO DOS REQUISITOS IMPLEMENTADOS

Os requisitos já implementados serão testados *online*. Esses testes serão feitos visando achar possiveis *bugs*[4]. Além disso, será testado como o *layout* ficará em diferentes tamanhos de tela e *browsers*.

O objetivo dessa etapa será testar a aplicação novamente. Afim de garantir que tudo foi implementado corretamente. Os testes de aceitação serão escritos nessa etapa, para forçar o desenvolvedor a revisar as histórias.

[2] `https://bitbucket.org/`
[3] `https://www.heroku.com/`
[4] Um comportamento errado da aplicação, causado por algum erro de lógica do desenvolvedor

3.6 TESTES DE USABILIDADE

Com as funcionalidades implementadas, pretende-se testá-lo com usuários reais, para verificar sua usabilidade, porém será realizado um experimento de caráter mais geral, sem seguir profundamente as diretivas de teste de usabilidade de *Interação Humano Computador* (IHC).

Nessa etapa pretende-se tentar reunir alguns voluntários para usar o sistema e depois deixarem um *feedback*. Isso poderá ser feito na universidade, com alguns servidores públicos de diferentes idades e escolaridade.

O objetivo dessa etapa será analisar possíveis *feedbacks* de diferentes usuários. Por meio disso será possível detectar dificuldades de uso, não percebidas pelo desenvolvedor.

3.7 UTILIZAR A APLICAÇÃO EM AMBIENTE DE PRODUÇÃO REAL

Após desenvolver o sistema pretende-se utilizá-lo ele em um restaurante ou bar. Para isso será necessário entrar em contato com alguns proprietários e verificar se será possível.

O objetivo dessa etapa será utilizar a aplicação em um bar ou restaurante real. Para assim, analisar seu comportamente no ambiente para o qual foi desenvolvida.

3.8 ANALISE DOS RESULTADOS OBTIDOS

Após um tempo de acompanhamento será necessário verificar se o sistema cumpriu corretamente sua proposta inicial. O objetivo dessa etapa será transcrever os resultados obtidos nas etapas anteriores, para relatar o sucesso ou o fracasso da aplicação desenvolvida.

4 RESULTADOS PREMILINARES

Nesse capítulo, será descrito o que foi desenvolvido até o presente momento. Os exemplos referentes diretamente a implementação, serão apresentados em inglês.

4.1 DINÂMICA DE FUNCIONAMENTO

Nessa seção será descrita brevemente a dinâmica de funcionamento da aplicação proposta. Ilustrada abaixo na Figura 4.

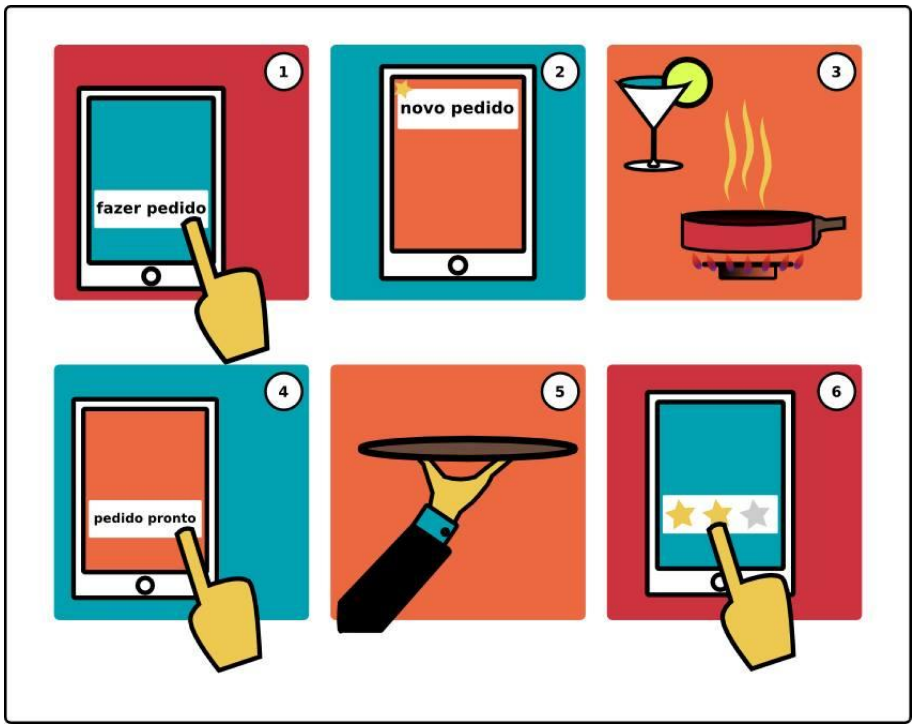

Figura 4: Dinâmica do funcionamento da aplicação proposta.

Fonte: O autor

Na ação número **1** o cliente interage com a aplicação para fazer seu pedido. Após o pedido ser confirmado, o mesmo será separado. As bebidas serão enviadas ao *barman* e a comida ao cozinheiro. Ambos receberão notificações em suas telas, como é demonstrando na

etapa **2**.

Após serem notificados, eles deverão preparar o que foi enviado. Assim como ilustra a etapa **3**. Quando terminarem seu trabalho, eles deverão comunicar no sistema que o produto está pronto para ser entregue ao cliente. Isso é demonstrado na etapa **4**.

Com isso, o garçom recebe uma notificação em seu celular, avisando que o pedido ou parte dele está pronto. Então ele fará a entrega ao cliente, como demonstra a etapa **5**.

Depois que o pedido é entre por completo, o cliente poderá avaliar os produtos que consumiu e o atendimento prestado. Isso é ilustrado na etapa **6**.

4.2 TABELA DE ESTÓRIAS

Na fase de projeto, foram pensandas nas principais funcionalidades do sistema. Para manter a fidelidade ao SCRUM, as funcionalidades serão tratadas e descritas como estórias. Nessa seção será apresentada a tabela com as principais estórias referentes a aplicação proposta.

Tabela 2: Estórias do Sistema Proposto

Estória	Descrição
ES 1	O sistema deve permitir o cadastro e a manutenção de novos usuários
ES 2	O sistema deve permitir o cadastro e a manutenção de novas categorias
ES 3	O sistema deve permitir o cadastro e a manutenção de novos produtos
ES 4	O sistema deve possuir uma interface para fazer novos pedidos
ES 5	O sistema deve permitir que um pedido seja cancelado por completo
ES 6	O sistema deve permitir que apenas alguns itens do pedido sejam cancelados
ES 7	O sistema deve oferecer uma interface para avaliação dos produtos consumidos
ES 8	O sistema deve oferecer uma interface para avaliação do atendimento prestado
ES 9	O sistema deve permitir retirar itens do menu temporariamente
ES 10	O sitema deve permitir alterar o estado do pedido
ES 11	O sistema deve permitir marcar um item ou mais como pronto em um pedido
ES 12	O sistema deve oferecer uma interface de acesso aos relatórios gerenciais
ES 13	O sistema deve oferecer uma interface para marcar uma conta como paga
ES 14	O sistema deve permitir o cadastro e manutenção de itens que não estaram no menu
ES 15	O sistema deve oferecer uma interface para adicionar outros itens a conta do cliente
ES 16	O sistema deve oferecer uma interface para aplicar um valor de desconto na conta

4.3 ATORES E SEUS PAPÉIS NA APLICAÇÃO

A aplicação proposta deverá atuar em bares e restaurantes. Então com base nesse escopo, os atores que vão interagir com o sistema serão os seguintes:

- Cliente;
- Garçom;
- *Barman*;
- Caixa;
- Cozinheiro;
- Gerente.

Seus papéis estão ilustrados na Figura 5:

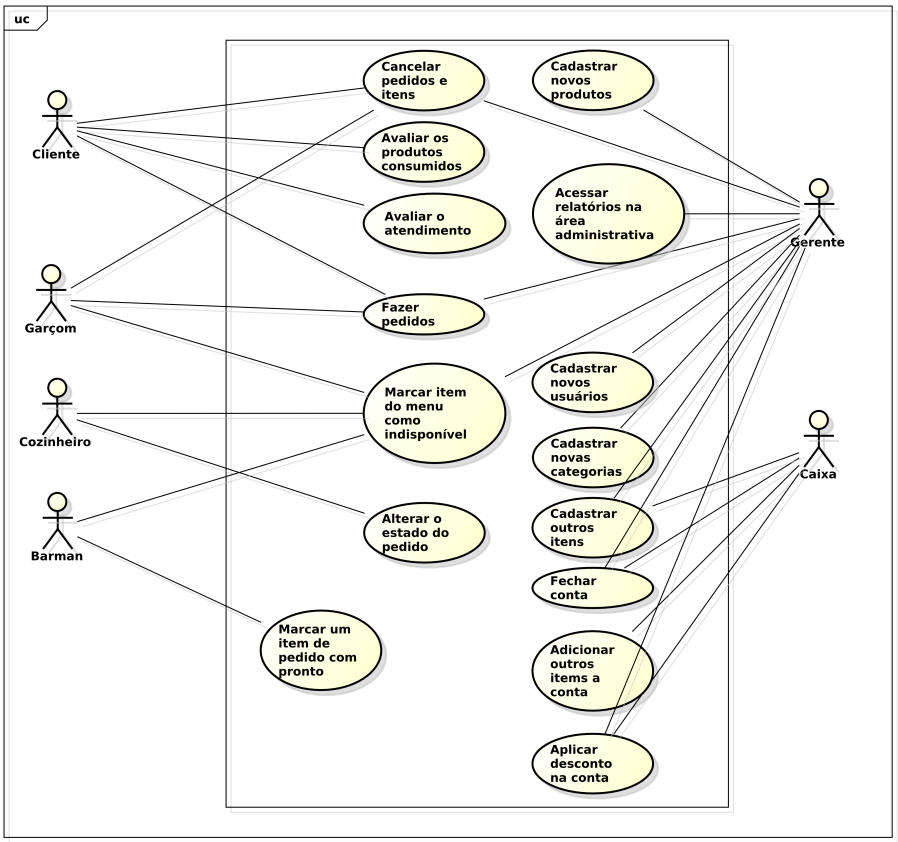

Figura 5: Diagrama de casos de uso.

Fonte: O autor

O papel do **cliente** permitirá a ele fazer os pedidos, avaliar o atendimento e os produtos consumidos e também cancelar pedidos e itens.

O **garçom** poderá fazer pedidos, cancelar itens e pedidos, conforme solicitação do cliente. Também poderá marcar itens do menu como indisponíveis, caso haja necessidade.

Ao ***barman*** será permitido marcar um item do pedido pronto, para quando terminar de preparar uma bebida. E também itens do menu como indisponíveis.

O papel de **caixa** será de cuidar dos pagamentos dos pedidos por parte dos clientes. Porém, essa função envolve algumas outras tarefas como por exemplo, cadastrar outros itens (valor da entrada, *couvert*, entre outros). Além de cadastrar esses itens, poderá adicioná-los a conta e aplicar um desconto. Mas sua principal função será fehar a conta, ou seja, marca-lá como paga.

O **cozinheiro** terá duas funções na aplicação. A primeira será marcar um item como indisponível. A segunda será alterar o estado do pedido, ou seja, é por causa dele que o garçom saberá que o pedido está pronto para entregar ao cliente.

O último papel é o de **gerente**. Ele poderá fazer quase todas as operações do sistema, com exceção das avaliações de pedido e atendimento que são exclusivas do cliente. E também não poderá marcar um pedido e nenhum item como pronto, essas serão atividades destinadas a quem os prepara.

O gerente terá um papel importante na área administrativa. Por meio da qual, poderá cadastrar produtos, usuários e categorias. Também poderá acessar os relatórios da aplicação que estarão disponíveis apenas na área administrativa.

4.4 TELAS DA APLICAÇÃO

Nessa seção serão detalhadas alguns prototipos de telas para a aplicação proposta.

4.4.1 TELA DO PRODUTO

A tela por meio da qual o cliente poderá ver os produtos, é ilustrada abaixo na Figura 3.

Figura 6: Tela de visualização do produto.

Fonte: O autor

Como é possível ver na Figura 6 a tela do produto terá foco na simplicidade. Contendo as informações básicas como nome, preço e uma foto do produto. Para visualizar mais informações e fotos do produto, o cliente deverá clicar no botão com o sinal de adição, situado no canto superior direito. A tela também possui um botão para voltar e adicionar o produto ao pedido.

4.4.2 TELA DO COZINHEIRO E BARMAN

As telas que serão utilizas pelo cozinheiro e *barman* são ilustradas abaixo, nas Figuras 4 e 5.

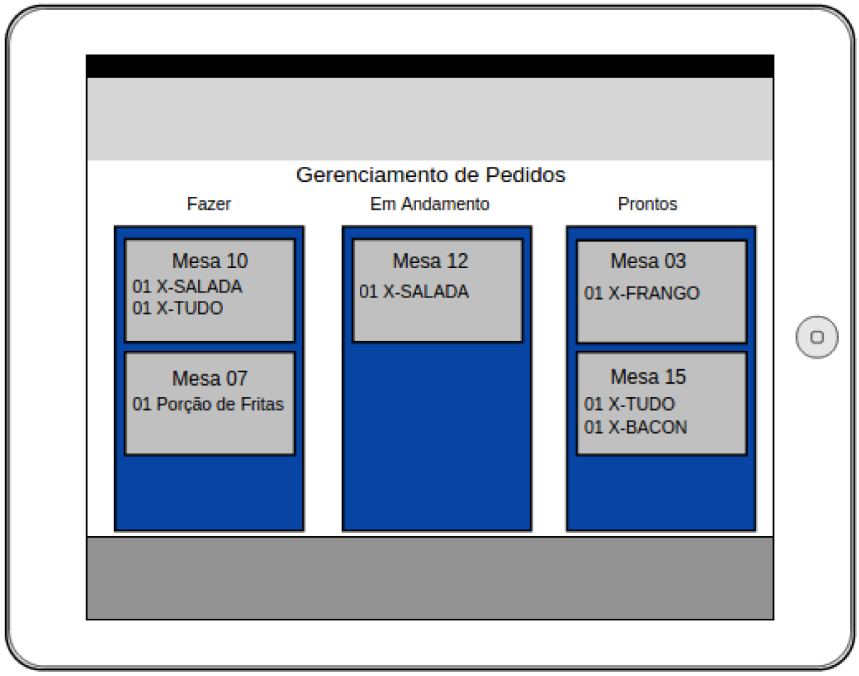

Figura 7: Tela de visualização do cozinheiro.

Fonte: O autor

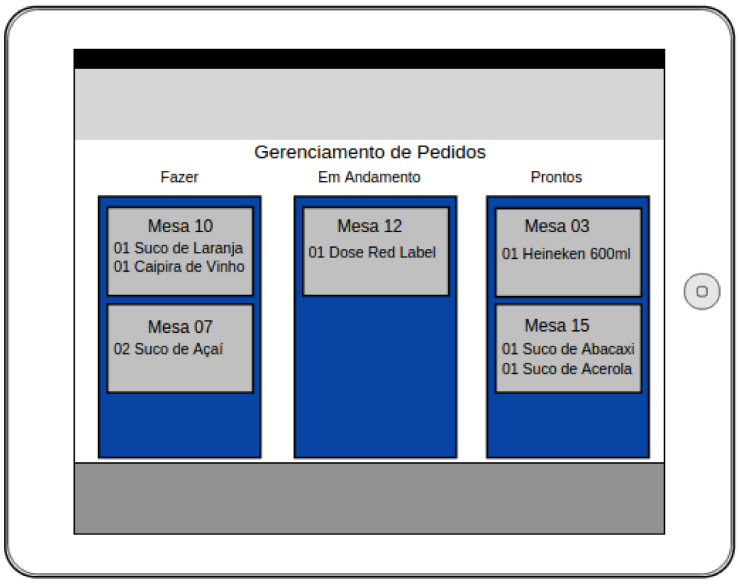

Figura 8: Tela de visualização do *barman*.

Fonte: O autor

As duas telas acima basicamente são idênticas. A única diferença está no conteúdo, a do cozinheiro terá apenas comida e a do *barman* apenas bebida. As telas possuem 3 partes,

pedidos para fazer, pedidos em andamento e pedidos prontos.

Quando o pedido é confirmado pelo cliente ele se junta aos pedidos a fazer. E quando o cozinheiro ou *barman* começar a prepará-lo, deverá tocar na tela e arrastar o pedido para junto dos pedidos em andamento. Quando isso acontecer, o cliente não poderá mais cancelar e nem alterar seu pedido.

Quando o pedido está pronto, quem o preparou deve arrastá-lo para junto dos pedidos prontos. Isso é importante, porque quando o pedido estiver nessa área o garçom será notificado de que está pronto.

4.4.3 TELA DO GARÇOM

Na Figura 9, é ilustrada a tela pela qual o garçom irá interagir com o sistema.

Figura 9: Tela de visualização do garçom.
Fonte: O autor

No topo da tela do garçom será possível visualizar o nome do garçom e um *link* para sair do sistema. Logo abaixo, terá uma listagem dos pedidos prontos, que já poderão ser entregues aos clientes. Quando o garçom entregar um pedido em sua mesa correspondente, deverá

pressionar o botão entregue. Dessa forma será possível marcar o tempo que o pedido demorou desde seu preparo até a entrega.

O garçom também poderá fazer buscas nos pedidos que estão em andamento e para fazer. Isso poderá ser feito por meio do campo para buscas, situado no canto superior esquerdo da tela.

4.5 BANCO DE DADOS

Um dos primeiros itens desenvolvidos foi o banco de dados do projeto. Por meio dele, é possível armazenar e posteriormente extrair as informações necessárias para cumprir os objetivos propostos. Como é apresentado na Figura 10.

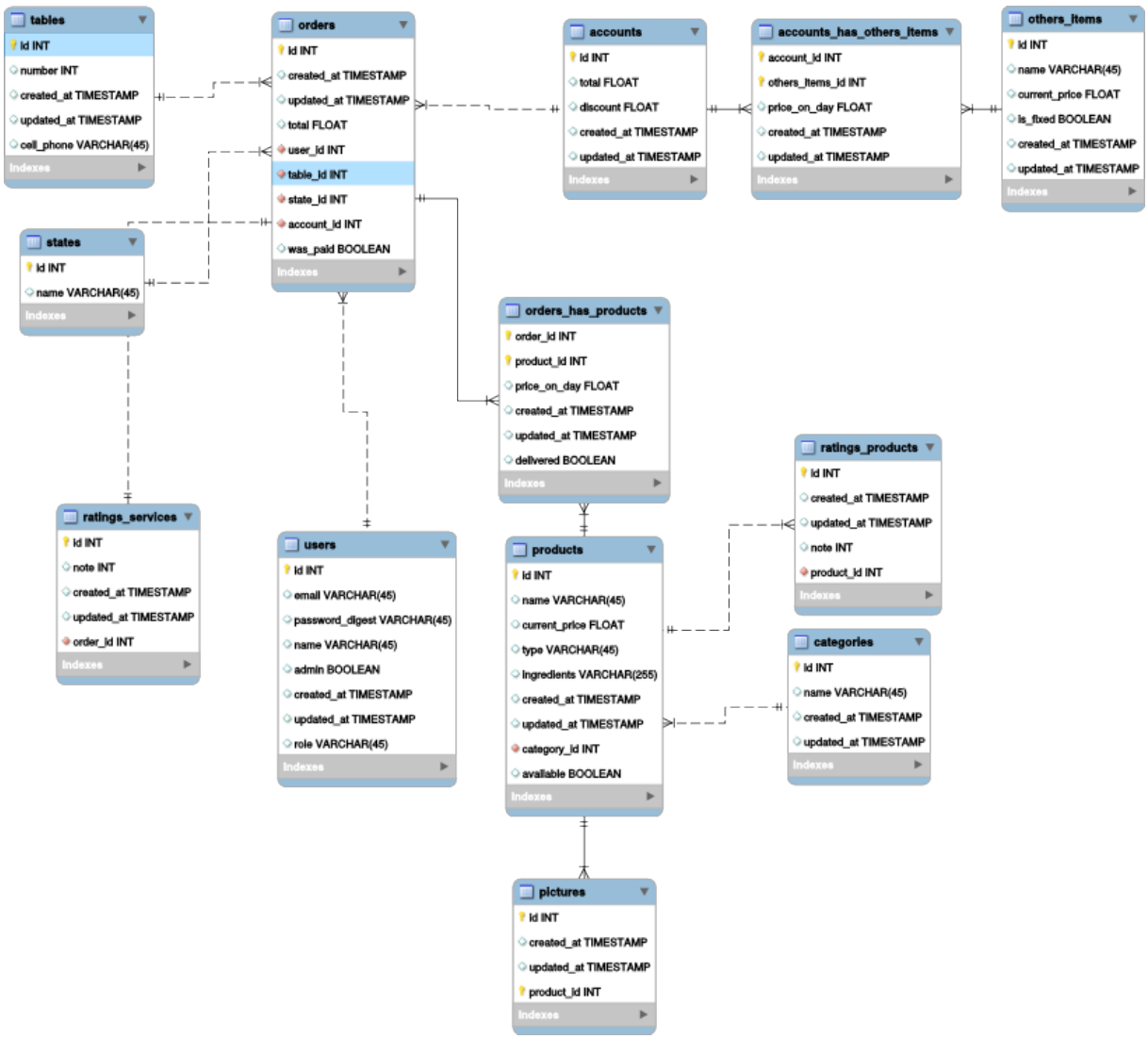

Figura 10: Banco de dados.

Fonte: O autor

À seguir, são enumerados os objetivos específicos do projeto. Com a finalidade de demostrar como o banco de dados proposto, deverá auxiliar no cumprimento de cada um deles.

1. Auxiliar clientes de bares e restaurantes a fazer seus pedidos de forma eletrônica mediante a um dispositivo móvel, sem a necessidade de chamar um garçom;
2. Disponibilizar para os clientes, um menu do estabelecimento, em que será possível avaliar os pratos servidos;
3. Fornecer aos garçons e aos cozinheiros a relação de pedidos realizados pelos clientes;
4. Fornecer ao gerente relatórios para auxiliar a tomada de decisões estratégicas, como por exemplo pratos mais pedidos;
5. Apresentar pratos mais vendidos e melhor avaliados para os clientes, através de um menu interativo.

O primeiro objetivo que deve ser levado em consideração é o número **2**. O cliente deverá visualizar o menu com os produtos para então escolher e fazer o pedido. Os produtos serão separados por categorias no menu.

Para cumprir esse objetivo, foram criadas as tabelas: *products* e *categories*. Na tabela *products*, serão armazenados os dados dos produtos, tais como:

- Nome do produto;
- Preço atual produto;
- Tipo do produto;
- Os ingredientes do produto;
- Categoria do produto;
- Sua disponibilidade.

O relacionamento entre categoria e produto, pemitirá separar os produtos por categoria no menu. Na tabela *categories* que salvará as categorias, será armazenado o nome da categoria. Também foi criada a tabela *pictures*, que deverá salvar o nome das imagens dos produtos,

possibilitando uma galeria de imagens para cada produto. As imagens em si, serão salvas em algum serviço de armazenamento em nuvem, como por exemplo Dropbox[1] ou Google Drive[2].

O cliente também poderá avaliar os produtos, essa avaliação consiste em uma nota de 1 a 5. Para fazer essa avaliação foi criada a tabela *rattings_products*. Que armazenará o uma identificação do produto e a nota recebida.

Para cumprir o objetivo número **3**, é necessário armazenar: os pedidos feitos, quem fez cada pedido e os itens de cada pedido. Para armazenar os pedidos foi criada a tabela *orders*, que contêm as seguintes informações:

- Total do pedido;
- Identificação do usuário que entregou o pedido;
- Desconto;
- Identificação da mesa de onde partiu o pedido;
- Identificação do estado do pedido.

Para identificar quem fez o pedido foi criada a tabela *tables*. E nela serão guardados o número da mesa e o número do celular de quem fez o pedido. No caso dos clientes sentados na mesma mesa resolverem fazer pedidos diferentes, será possível identificar quem fez cada pedido pelo número do celular.

O pedido será controlado por meio de estados. Esses estados, vão representar em qual estágio ele se encontra, esse processo será explicado adiante. Para fazer esse controle, foi criada uma tabela *states* que salvará os estados e a tabela de pedidos terá uma identificação do estado no qual o pedido se encontra.

Na tabela dos pedidos também há uma identificação do usuário. Esse será o garçom que entregará o pedido. Essa identificação possibilitará que o cliente avalie o atendimento. Essa avaliação será feita na tabela *rattings_services* que é equivalente a tabela *rattings_products*.

Os itens de cada pedido serão salvos na tabela *orders_has_products*. Que irá guardar a identificação do pedido e do produto, e também o preço do produto no dia em que foi pedido.

Com o pedido salvo em uma tabela, será possível exibi-los em uma tela para os cozinheiros. Cumprindo assim o objetivo número **3**.

[1] `http://www.dropbox.com`
[2] `https://www.google.com/drive`

Graças a tabela *orders_has_products* o objetivo número **4** e a parte dos produtos mais vendidos do objetivo **5** serão cumpridos. Será necessário, fazer consultas a base de dados. E o *framework* Ruby on Rails possuiu um módulo chamado *ActiveRecord*[3] que fornece uma forma simples para fazer consultas. E para mostrar os pratos mais bem avaliados, será preciso consultar a tabela *rattings_products*.

Para tornar a aplicação funcional, foi adicionada a funcionalidade da conta. Por meio da qual é possível receber os pagamentos dos clientes. As contas são representadas no banco pela tabela *accounts*, que salvam o valor total da conta e do desconto.

Cada pedido terá uma conta e cada conta poderá ter mais de um pedido. Isso possibilitaria que clientes de uma mesma mesa fizessem diferentes pedidos. E depois na hora de pagar a conta, cada um pagaria o que consumiu de uma forma simplificada.

Como podem haver mais itens para serem somados a conta, como por exemplo entrada ou *couvert*. Foi criada a tabela *others_items*, para guardar esses itens e seus valores.

Embora essa última funcionalidade não tenha relação com os objetivos propostos. Para tornar a aplicação funcional, foi necessário adicionar essa funcionalidade extra.

4.6 DIAGRAMA DE SEQUÊNCIA DO PEDIDO

Para ilustrar como ocorre um pedido, foi utilizado um diagrama de sequência seguindo os padrões do *Unified Modeling Language*[4] (UML) 2.0, que pode ser visto na Figura 11.

[3] `http://guides.rubyonrails.org/active_record_basics.html`
[4] `http://www.uml.org/`

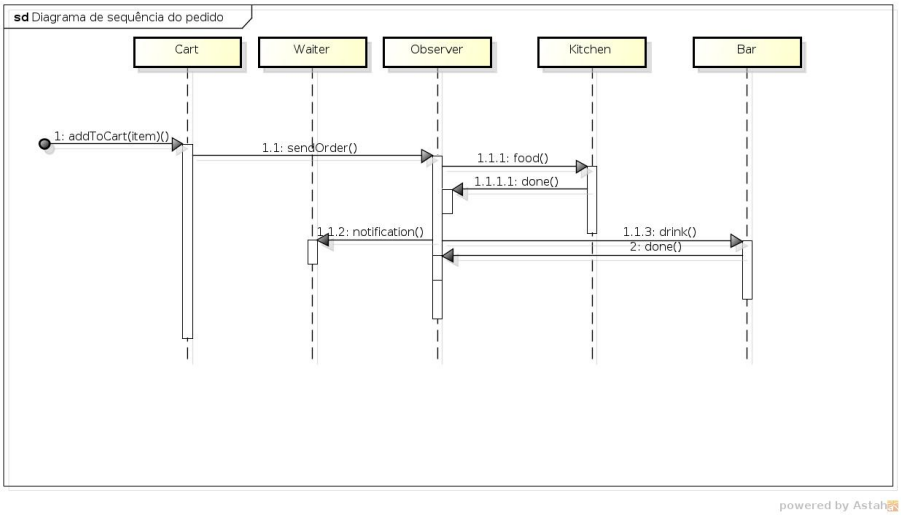

Figura 11: Diagrama de sequência do pedido.

Fonte: O autor

Quando um cliente estiver navegando no menu interativo, poderá selecionar produtos e adicioná-los ao **carrinho**. Que será semelhante aos carrinhos de compra encontrados nas lojas virtuais. A finalidade do carrinho no sistema proposto, será servir como uma área temporária para armazenar os itens, enquanto o cliente ainda não finalizou seu pedido.

No diagrama o carrinho é representado pelo participante *Cart*. Que recebe uma mensagem para adicionar um item e armazená-lo. O carrinho não será salvo no banco de dados, ao invés disso, ele será salvo no *Local Storange*[5] do próprio navegador do cliente. Com isso, será evitado acessos desnecessários a API.

Após o cliente, finalizar seu pedido será enviada uma mensagem *sendOrder* para o participante *Observer*. O *Observer* basicamente, terá duas responsabilidades, a primeira que convém a esse caso, será separar o que for comida e enviar para a cozinha e o que for bebida e enviar para o bar. Como é representado no diagrama pela mensagem *food* enviada ao participante *Kitchen* e *drink* enviado ao participante *Bar*.

O participantes *Kitchen* e *Bar* representam a cozinha e o bar do estabelecimento. Por meio disso que os cozinheiros e *barmans*, receberam informações sobre os pedidos feitos. Após eles terminarem de fazer os pratos e preparar as bebidas, o sistema enviará um aviso ao *Observer* de que os pratos e bebidas estão prontos para serem servidos. Como é representado pelas mensagens *done*.

Agora o *Observer* terá uma segunda responsabilidade que será enviar uma notificação

[5] `http://www.w3schools.com/html/html5_webstorage.asp`

aos garçons. Como é representado pela mensagem *notification* enviadas ao participante *Waiter*. Dessa forma os garçons saberão que os pedidos estão prontos e poderão entregar aos clientes que os solicitaram.

5 CONSIDERAÇÕES FINAIS

Conforme foi levantado na introdução, o principal desafio desse projeto, seria automatizar um processo simples sem torná-lo complexo. A aplicação proposta, se tornará parte da rotina dos usuários. E por isso, é tão importante que ela seja simples.

Partindo dessa ideia, foi necessário pensar no papel de cada usuário no sistema. Uma dificuldade encontrada durante essa elaboração, foi o fato de conhecer o trabalho em bares e restaurantes apenas do ponto de vista de um cliente. Para tentar suprir essa necessidade, foram buscados materias na Internet para entender um pouco mais sobre essa rotina.

Com a modelagem dos papéis dos usuários encerrada, o próximo passo foi elaborar soluções para o objetivo principal e os objetivos específicos. O primeiro passo foi elaborar um banco de dados que conseguisse armazenar os dados necessários para chegar aos objetivos.

Durante a modelagem do banco de dados, o maior desafio foi prever algumas situações que exigissem flexibilidade da aplicação. Como por exemplo, clientes que estão na mesma mesa fazerem pedidos separados. Não é possível prever todos esses cenários durante a modelagem do banco de dados, por isso, esses cenários receberão mais atenção nas fases de desenvolvimento e testes.

O segundo passo para cumprir os objetivos propostos, foi a elaboração das telas principais do sistema. Foram usadas como base algumas interfaces dos sistemas Eped e Ezee iMenu. Porém, as telas da aplicação proposta deveriam ter apenas o conteúdo minimo necessário para cada operação, visando minimizar o esforço dos usuários. A maior dificuldade foi pensar no que seria o minimo necessário para cada operação.

Acredita-se que com o que foi elaborado até a data presente, já é possível desenvolver a aplicação proposta. De forma a cumprir os objetivos propostos inicialmente. Porém, é preciso levar em consideração que devido a falta de experiência prática com o trabalho em bares e restaurantes. A aplicação deverá ser exaustivamente testada, e devido aos possíveis cenários inesperados que provavelmente surgirão, a aplicação deverá sofrer várias mudanças até chegar a uma versão estável.

6 CRONOGRAMA

As atividades (Tabela 3) que compõem a pesquisa serão realizados conforme o cronograma da Tabela 4.

Tabela 3: Atividades Previstas

Atividades	Descrição
1	Levantamento dos Requisitos do Sistema
2	Idealização da Arquitetura do Sistema
3	Estudo das Tecnologias à Serem Utilizadas
4	Desenvolvimento do Sistema
5	Validação dos Requisitos Implementados
6	Testes de Usabilidade
7	Utilizar a Aplicação em Ambiente de Produção Real
8	Analise dos resultados obtidos
9	Redação da Monografia Final
10	Elaboração da Apresentação Final

Tabela 4: Cronograma de Atividades

Atividade	MAR/15	ABR/15	MAI/15	JUN/15	JUL/15	AGO/15	SET/15	OUT/15	NOV/15	DEZ/15
1										
2										
3										
4										
5										
6										
7										
8										
9										
10										

REFERÊNCIAS

ANICHE, M.; CORBUCCI, H. **Test-Driven Development: Test e Design no Mundo Real com Ruby**. [S.l.]: CASA DO CODIGO, 2014. ISBN 9788566250435.

ASTELS, D.; CHELIMSKY, D. **The RSpec Book: Behaviour Driven Development with RSpec, Cucumber, and Friends**. Pragmatic Bookshelf, 2010. (Pragmatic Bookshelf Series). ISBN 9781934356371. Disponível em: <https://books.google.com.br/books?id=0rxoPgAACAAJ>.

CRAVENS, J.; BRADY, T. **Building Web Apps with Ember.js**. O'Reilly Media, 2014. ISBN 9781449370909. Disponível em: <https://books.google.com.br/books?id=dNr-AwAAQBAJ>.

EZEE TECHNOSYS. **Página das features**. 2015. Disponível em: <http://www.ezeeimenu.com/features.php>.

EZEE TECHNOSYS. **Página inicial**. 2015. Disponível em: <http://www.ezeeimenu.com/>.

FINK, G.; FLATOW, I. **Pro Single Page Application Development: Using Backbone.js and ASP.NET**. Apress, 2014. (EBL-Schweitzer). ISBN 9781430266747. Disponível em: <https://books.google.com.br/books?id=ayuLAwAAQBAJ>.

FREEMAN, S.; PRYCE, N. **Growing Object-Oriented Software, Guided by Tests**. Pearson Education, 2009. (Addison-Wesley Signature Series (Beck)). ISBN 9780321699763. Disponível em: <https://books.google.com.br/books?id=QJA3dM8Uix0C>.

GARCEZ, E. M. S.; FACHIN, G. R. B.; JUNIOR, P. P. A. Indicadores da qualidade em restaurantes: Um estudo de caso. **Revista de Ciências da Admnistração**, n. 3, Abril 2000.

KELONYE, M. **Mastering Ember.js**. Packt Publishing, 2014. (Community experience distilled). ISBN 9781783981991. Disponível em: <https://books.google.com.br/books?id=dY3jBAAAQBAJ>.

KOZLOWSKI, P. **Mastering Web Application Development with AngularJS**. Packt Publishing, 2013. (Community experience distilled). ISBN 9781782161837. Disponível em: <https://books.google.com.br/books?id=mZXjwz5X08EC>.

LERNER, A. **Ng-Book - the Complete Book on Angularjs**. Fullstack.io, 2013. ISBN 9780991344604. Disponível em: <https://books.google.com.br/books?id=I7UpnwEACAAJ>.

OLIVEIRA, K. D. **Recuperação de serviço no processo de atendimento em restaurante: Estudo de caso em Porto Alegre**. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Sul, 2002.

PEKUS. **Página inicial**. 2015. Disponível em: <http://www.pekus.com.br/cardapio.aspx>.

PSIU GARÇOM. **Página inicial**. 2015. Disponível em: <http://www.chamagarcom.ind.br/>.

RUBY, S.; THOMAS, D.; HANSSON, D. **Agile Web Development with Rails 4**. Pragmatic Bookshelf, 2013. (Pragmatic Programmers). ISBN 9781937785567. Disponível em: <https://books.google.com.br/books?id=ymiqnAEACAAJ>.

SCOTT, E. **Spa Design and Architecture: Understanding Single Page Web Applications**. Manning Publications Company, 2015. ISBN 9781617292439. Disponível em: <https://books.google.com.br/books?id=v_oXrgEACAAJ>.

SKEIE, J. **Ember.js in Action**. Manning Publications Company, 2013. ISBN 9781617291456. Disponível em: <https://books.google.com.br/books?id=LYxRmwEACAAJ>.

SMART, J. **BDD in Action: Behavior-Driven Development for the Whole Software Lifecycle**. Manning Publications Company, 2014. ISBN 9781617291654. Disponível em: <https://books.google.com.br/books?id=2BGxngEACAAJ>.

STARZHINSKY, A. Blog, **AngularJS advantages and limitations**. 2015. Disponível em: <http://blog.softelegance.com/angularjs/angularjs-advantages-and-limitations/>.

WATERS, J. **QR Codes For Dummies**. Wiley, 2012. (For Dummies). ISBN 9781118370711. Disponível em: <https://books.google.com.br/books?id=Nd_50ehL5EIC>.