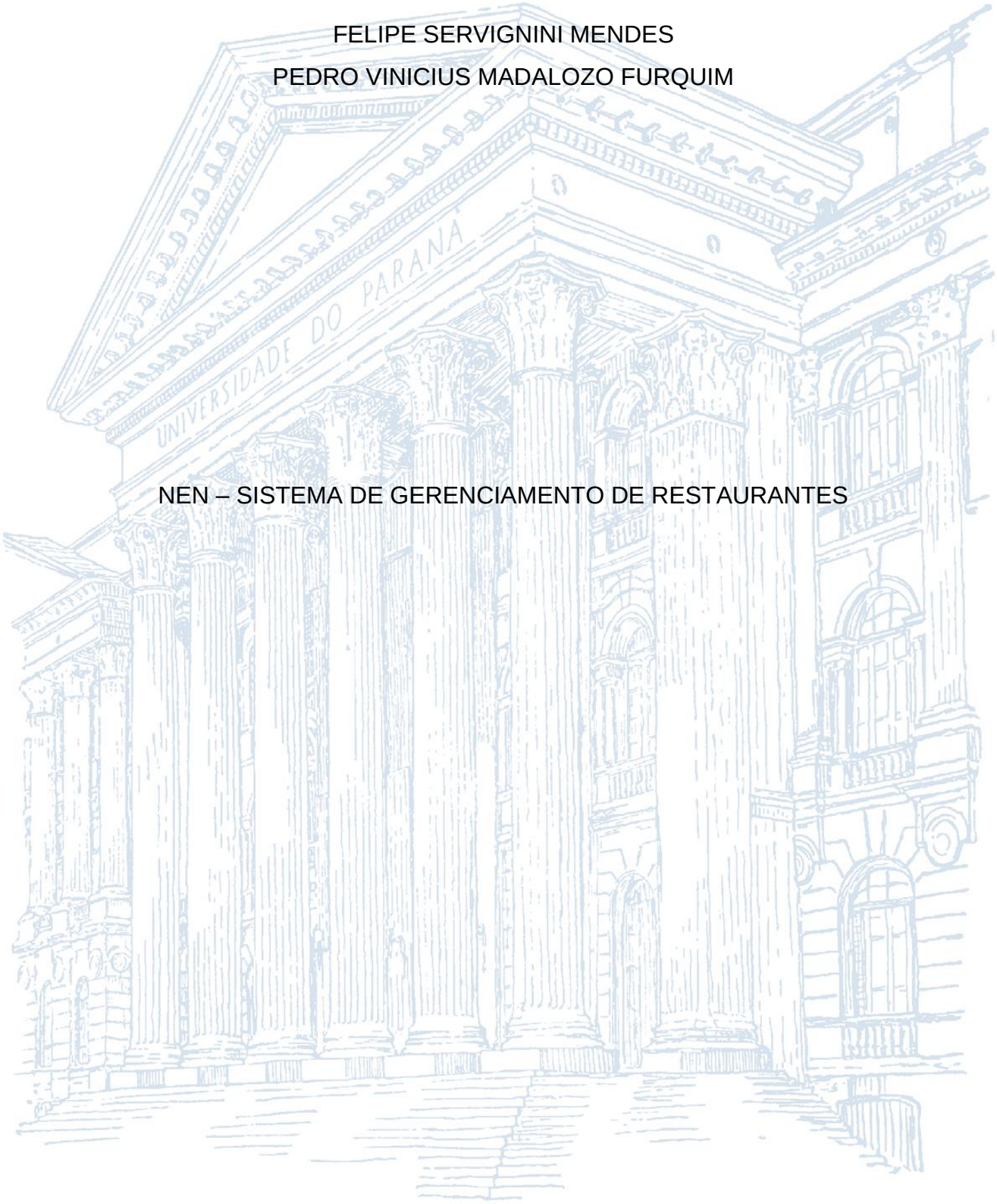
UNIVERSIDADE FEDERAL DO PARANÁ

FELIPE SERVIGNINI MENDES

PEDRO VINICIUS MADALOZO FURQUIM

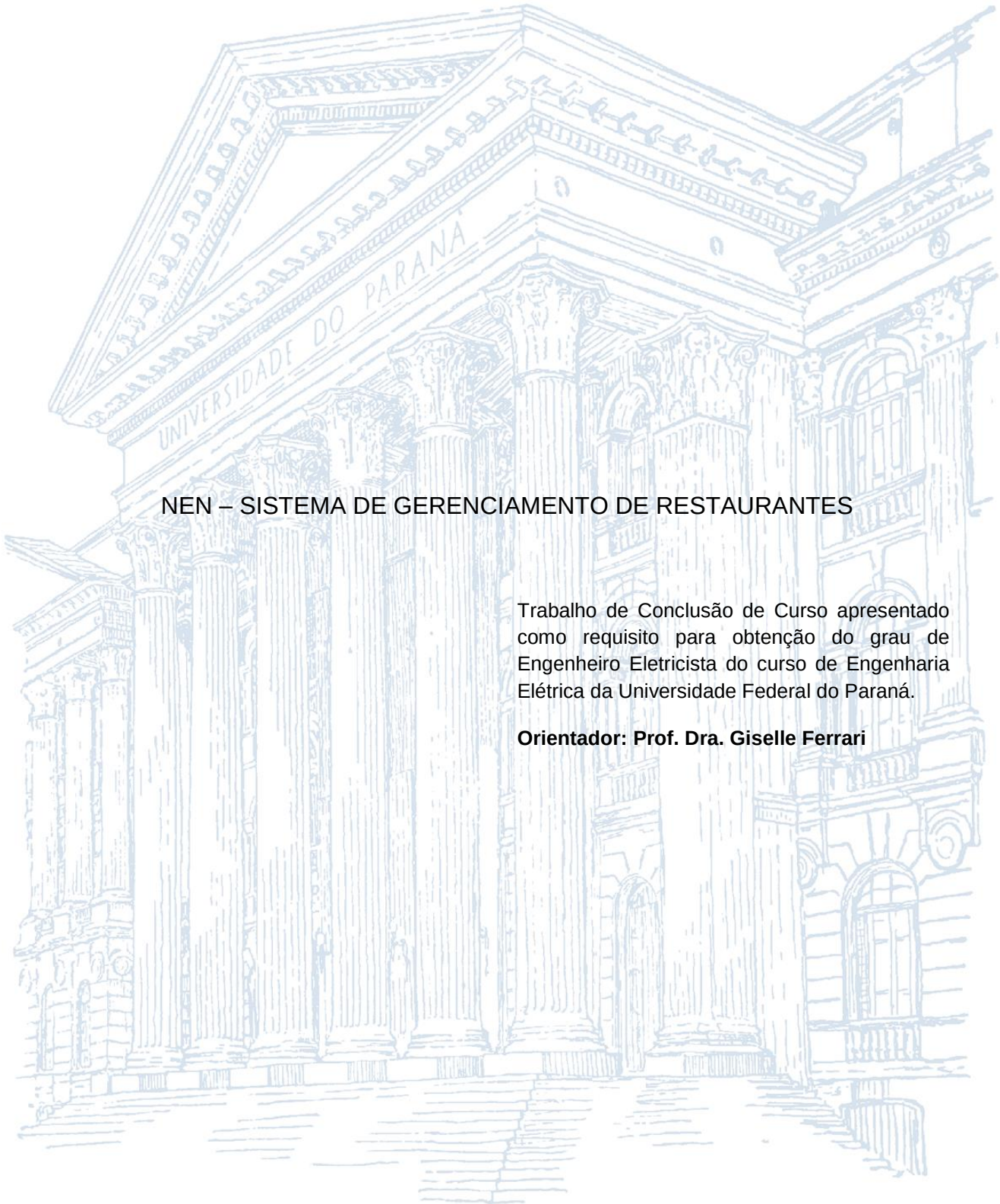


NEN – SISTEMA DE GERENCIAMENTO DE RESTAURANTES

CURITIBA

2018

FELIPE SERVIGNINI MENDES
PEDRO VINICIUS MADALOZO FURQUIM



NEN – SISTEMA DE GERENCIAMENTO DE RESTAURANTES

Trabalho de Conclusão de Curso apresentado como requisito para obtenção do grau de Engenheiro Eletricista do curso de Engenharia Elétrica da Universidade Federal do Paraná.

Orientador: Prof. Dra. Giselle Ferrari

CURITIBA

2018

TERMO DE APROVAÇÃO

FELIPE SERVIGNINI MENDES
PEDRO VINICIUS MADALOZO FURQUIM

NEN – SISTEMA DE GERENCIAMENTO DE RESTAURANTES

TRABALHO DE CONCLUSÃO DE CURSO APROVADO COMO REQUISITO PARCIAL PARA OBTENÇÃO DO TÍTULO DE ENGENHEIRO ELETRICISTA NO CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA, SETOR DE TECNOLOGIA DA UNIVERSIDADE FEDERAL DO PARANÁ, PELA SEGUINTE BANCA EXAMINADORA:

_____

ORIENTADOR: PROF. DRA. GISELLE LOPES FERRARI RONQUE
DEPARTAMENTO DE ENGENHARIA ELÉTRICA, UFPR

_____

PROF. ME. CARLOS GOUVEA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA, UFPR

_____

PROF. DRA. JULIANA LUÍSA MÜLLER IAMAMURA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA, UFPR

CURITIBA, 04 DE DEZEMBRO DE 2018

AGRADECIMENTOS

Agradecemos ao nosso país a oportunidade de estudar em uma instituição pública e com professores altamente qualificados que nos proporcionaram uma formação excelente.

Agradecemos aos nossos pais, que nos deram a vida e cuidaram de nós, nos apoiando e ensinando os caminhos que nos levam a uma vida de sucesso pessoal e profissional.

Agradecemos aos nossos colegas, companheiros e amigos que conhecemos durante esta jornada para nos tornarmos engenheiros eletricistas. Mesmo nos momentos difíceis, sempre nos trouxeram sentimentos de alegria e conforto.

Agradecemos aos nossos amigos anteriores a graduação, que nos acompanharam durante todo o processo, incentivando e torcendo para que pudéssemos levantar o diploma no final do caminho.

Agradecemos profundamente a nossa orientadora, Prof. Dra. Giselle Ferrari, que nos estendeu a mão e acreditou no nosso projeto e no nosso potencial de executa-lo, nos dando todo o suporte acadêmico necessário para o desenvolvimento.

“Quanto mais aumenta nosso conhecimento, mais evidente fica nossa ignorância”

John F. Kennedy

RESUMO

Manter o controle do fluxo de produtos e serviços prestados em um restaurante torna-se uma tarefa árdua quando não se tem disponível uma ferramenta correta para realizar esta tarefa, o que resulta em muitas limitações quanto ao armazenamento e análise de dados, além de mascarar as perdas de recursos. O presente trabalho de conclusão de curso tem a finalidade de desenvolver um software para suprir as necessidades do restaurante Madalozo. Abrangendo todas as suas áreas vitais de funcionamento, fornecendo dados relevantes do desempenho do estabelecimento, auxiliando o administrador a identificar pontos de fraqueza e torna-los mais fortes. O projeto se iniciou com uma análise do modelo de negócio do restaurante, para que os requisitos do projeto ficassem claros para os desenvolvedores. Uma vez estabelecidos os objetivos do projeto, foram criados diagramas de caso de uso, de classe, sequencial e colaborativo dos setores do restaurante. A metodologia utilizada para a elaboração do trabalho é chamada *Scrum*, que é uma metodologia muito difundida no planejamento de softwares devido ao seu aspecto ágil e de constante iteração com o comprador do programa. A plataforma de desenvolvimento escolhida é a web, devido a sua capacidade de boa reponsividade entre os sistemas operacionais existentes, mantendo suas funcionalidades e escalando os objetos conforme o tamanho da tela em uso. Aliada a esta plataforma versátil, foi utilizado o *framework* Angular 6 e o banco de dados noSQL, Angular Firestore. Para a concepção do programa, foram utilizadas 3 linguagens de programação: Typescript, HTML, CSS. A escolha do projeto foi voltada para a preparação dos autores para um mercado de trabalho que demanda engenheiros que saibam programar com excelência, bem como abrir as portas para os colegas mais novos no curso para que se sintam bem-vindos no desenvolvimento de software como engenheiros eletricistas.

Palavras-chave: Angular 6. Firestore. NoSQL. Typescript. Scrum.

ABSTRACT

Keeping track of the flow of products and services rendered in a restaurant becomes an arduous task when the correct tool is not available to accomplish this task, resulting in many limitations on data storage and its analysis, as well as concealing the loss of resources. The present work of course completion has the purpose of developing software to meet the needs of the restaurant Madalozo. Covering all of its vital areas of operation, providing relevant data on facility performance, helping the administrator to identify weaknesses and make them stronger. The project began with an analysis of the restaurant's business model, so that the project requirements were clear to the developers. Once the project objectives were established, use case, class, sequential and collaborative diagrams of the restaurant sectors were created. The methodology used for the elaboration of the work is called Scrum, which is a very widespread methodology in software planning due to its agile aspect and constant iteration with the buyer of the program. The development platform chosen is the web, due to its ability to responsiveness between existing operating systems, maintaining its functionality and scaling objects according to the size of the screen in use. Allied to this versatile platform, we used the Angular 6 framework and the noSQL database, Angular Firestore. To develop this software, 3 programming languages were used: Typescript, HTML, CSS. The choice of this project was aimed at preparing the authors for a job market that demands engineers who know how to program with excellence as well as open the doors to the younger colleagues in the course so that they feel welcome in software development as electrical engineers.

Key-words: Angular 6. Firestore. NoSQL. Typescript. Scrum.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

BSON	BINARY JASON
DNS	DOMAIN NAME SYSTEM
HTTP	HYPER TEXT TRANSFER PROTOCOL
IBM	INTERNACIONAL BUSINESS MACHINES
JS	JAVASCRIPT
JSON	JAVASCRIPT OBJECT NOTATION
POO	PROGRAMAÇÃO ORIENTADA A OBJETO
SGBDs	SISTEMAS DE GERENCIAMENTO DE BANCO DE DADOS
SSL	SECURE SOCKET LAYER
TCC	TRABALHO DE CONCLUSÃO DE CURSO
TCP	TRANSMISSION CONTROL PROTOCOL
TLS	TRANSPORT LAYER SECURITY
UDP	USER DATAGRAM PROTOCOL
UML	UNIFIED MODELING LANGUAGE
VSCODE	VISUAL STUDIO CODE
XML	EXTENSIBLE MARKUP LANGUAGE

SUMÁRIO

1. INTRODUÇÃO

1.1. CONTEXTO

O incentivo ao avanço tecnológico ao redor do mundo força os mais diversos setores da economia a se reinventarem e apropriarem-se das novas ferramentas oferecidas. Como benefício, elas podem melhorar a produtividade do negócio através da minimização dos custos, expondo pontos frágeis na organização, aumentando os lucros.

Mas para que estes pontos frágeis sejam expostos a liderança de um empreendimento, é necessário que existam informações na forma adequada para serem tratadas, ou seja, armazenadas digitalmente. Uma vez existente um arranjo de dados grande o suficiente para que seja tratado, as interpretações e conexões com a realidade no dia a dia da empresa podem ser feitos e trazer resultados.

O software que será apresentado neste trabalho tem como objetivo manter o controle do fluxo de produtos e serviços prestados de maneira organizada e digital. O cliente alvo é um restaurante tradicional de Morretes no estado do Paraná, chamado Madalozo.

Atualmente o Restaurante Madalozo utiliza programas de terceiros para o controle de estoque, porém devido a algumas deficiências apresentadas pelo software, também é necessário o uso de planilhas do Excel bem como planilhas físicas. Como o restaurante possui os dados segregados em diferentes plataformas, torna-se muito difícil relacionar estes dados e gerar relatórios complexos para análise de desempenho do restaurante, o atual sistema carece de opção de gerar relatórios.

Outra carência que se faz presente é a de um sistema multiplataforma que possa ser acessado de qualquer lugar do restaurante. Recorrentemente no recebimento de produtos de fornecedores não há um computador próximo para serem cadastrados os dados do recebimento, esta informação acaba sendo gravada em uma planilha física e somente depois é transferida para o programa, resultando em retrabalho e possibilidade de algum dado ser inserido incorretamente.

Com a intenção de resolver estes problemas apresentados, seguem as propostas do presente trabalho de conclusão de curso (TCC).

1.2. JUSTIFICATIVA

Os principais fatores para a escolha deste tema são a aplicabilidade do trabalho, ajudando o restaurante, e a possibilidade de abrir um negócio após a graduação com o software desenvolvido. Espera-se atingir um nível muito bom de experiência de usuário na utilização do programa, bem como a redução de ineficiências que o administrador do empreendimento poderá identificar através dos dados coletados.

1.3. OBJETIVOS

1.3.1. OBJETIVO GERAL

Desenvolver um sistema de gerenciamento para o Restaurante Madalozo, visando suprir as deficiências de sistema expostas anteriormente, aprimorando assim a operação do estabelecimento.

1.3.2. OBJETIVOS ESPECÍFICOS

O programa deverá apresentar uma forma eficiente e funcional de organizar o fluxo de operações das áreas do restaurante, sendo estas: a recepção, o atendimento às mesas, a cozinha, o bar, o caixa e a administração. Elaborar a documentação referente ao projeto, incluindo o diagrama de caso de uso, o diagrama UML, os diagramas de sequência e diagrama colaborativo;

O sistema deverá ser:

- Robusto: contemplar diversas áreas do restaurante, armazenando e tratando os dados gerados;
- Moderno: utilizar a plataforma web e um banco de dados noSQL;
- Amigável: possuir um layout e ágil e intuitivo.

1.4. ESTRUTURA DO TRABALHO DE CONCLUSÃO DE CURSO

O relatório do TCC está dividido em quatro capítulos. No primeiro capítulo foi apresentado o propósito do trabalho, com a contextualização dele, os objetivos a serem alcançados e a sua justificativa. No segundo capítulo são abordados os conceitos de construção de software, linguagens, métodos e documentações para a

compreensão do restante do trabalho. No terceiro capítulo são apresentados os materiais e métodos utilizados. Por fim, no quinto e último capítulo é apresentada a conclusão do trabalho e também sugeridos temas para trabalhos futuros.

2. FUNDAMENTAÇÃO TEÓRICA

Tendo em vista a proposta de desenvolvimento da aplicação, a seguir será apresentado o embasamento teórico necessário para seu entendimento. No começo do projeto se decide qual a metodologia será utilizada,

2.1. AGILE

Devido aos muitos insucessos obtidos nos projetos de software, surgiram iniciativas que visavam encontrar maneiras mais eficientes de desenvolver sistemas complexos. Esses estudos deram origem a algumas metodologias com foco mais nos fatores humanos e na satisfação do cliente do que na burocracia dos processos (Silva, Souza e Camargo 2013).

Agile é uma forma de pensar, podendo ser vista como uma filosofia ou do inglês, *mindset.* Esta ideia foi pública em forma de um manifesto, especificamente para softwares, em 2001, e ainda pode ser encontrada no seguinte endereço: https://agilemanifesto.org/. Este manifesto, resumidamente, tinha a intenção de tornar mais flexível a entrega dos softwares, valorizando os clientes, a funcionalidade do programa e a capacidade de reatividade alta a mudanças em detrimento de seguir um plano até a data de entrega final de um projeto.

Seguindo a filosofia ágil, a metodologia utilizada para o gerenciamento deste trabalho, chamada Scrum, foi introduzida e aplicada em um projeto de desenvolvimento de software (Sutherland 2004). Seguindo os mesmos princípios, apenas explicitada de uma maneira mais prática, algo que os responsáveis por um projeto pudessem aplicar a filosofia ágil.

2.2. SCRUM

O Time Scrum é composto pelo Product Owner (Dono do Produto), o Time de Desenvolvimento e o Scrum Master (Mestre Scrum). O Dono do Produto é o responsável por determinar os requisitos do produto, de maneira clara, garantindo que o Time de Desenvolvimento consiga executar, ajudando a dissolver os pontos duros quando encontrados. O Time de Desenvolvimento recebe e executa as ações. O

Mestre Scrum é o responsável pelas regras Scrum, papel crítico no aumento da eficácia das aplicações desenvolvidas pelo time (Souza 2014).

A principal marca do Scrum é o Sprint, um período de tempo, geralmente, entre 2 a 4 semanas na qual o Time de Desenvolvimento trabalha em algumas funcionalidades. Durante o Sprint acontecem reuniões diárias de aproximadamente 15 minutos, onde o time mostra ao Mestre Scrum o que foi feito no dia anterior, e o que será feito no dia da reunião.

Como Scrum tem o viés ágil, é sempre possível mudar o escopo do Sprint, mantendo as possibilidades abertas a novas necessidades que surjam pelo Dono do Produto, sendo marcada uma nova reunião com todo o Time Scrum para novas definições do próximo Sprint (Ken Schwaber 2013). Ainda ao final de cada Sprint, ocorre o chamado Sprint Review (Revisão do Sprint), onde é analisado o desempenho da equipe em relação aos resultados, e são propostas melhorias para o próximo ciclo.

Ao final de cada Sprint, o Mestre Scrum decide se quer ou não entregar as funcionalidades implementadas pelo Time de Desenvolvimento ao cliente.

A metodologia Scrum foi escolhida para o presente trabalho devido a sua grande eficiência, segundo a literatura, em entregas com prazos curtos, que é o caso. Embora o Time Scrum não estivesse completo, devido a equipe reduzida, nós adaptamos, seguindo a ideologia ágil, assumindo múltiplos papeis durante o projeto.

2.3. ANÁLISE DO NEGÓCIO

O restaurante é composto por 5 áreas principais, sendo elas a recepção, atendimento as mesas (garçons), cozinha e bar, caixa e administração. Todos estes setores serão detalhados adiante.

2.3.1. RECEPÇÃO

Em um restaurante a recepção sempre faz o primeiro atendimento ao cliente e nesta ocasião ter um processo ágil e funcional pode ser determinante para a satisfação dele.

2.3.2. ATENDIMENTO ÁS MESAS

Após o cliente ser encaminhado à mesa pela recepção, o próximo atendimento será realizado pelo garçom, através de um tablete ou celular.

2.3.3. COZINHA E BAR

Na sequência do atendimento realizado pelo garçom, a cozinha e o bar recebem as informações do pedido, contendo o número de itens, a mesa e quaisquer observações que tenham sido mencionadas.

2.3.4. CAIXA

O papel do caixa, após os clientes estarem satisfeitos é receber o valor devido ao estabelecimento pelos clientes.

2.3.5. ADMINISTRAÇÃO

A administração não tem um papel direto na operação do restaurante quanto a relação com o cliente. O papel dela é gerenciar todos os recursos físicos e humanos do estabelecimento, tomando todas as decisões de grande valor na empresa.

2.4. PROGRAMAÇÃO ORIENTADA A OBJETOS (POO)

O conceito de programação orientada a objeto, que é amplamente utilizado atualmente em inúmeras linguagens de programação, se originou por volta de 1960, mas encontrou muita resistência antes de ganhar momento. Devido as drásticas mudanças de estrutura de código, as empresas se recusavam a refazer softwares que já funcionavam apenas para utilizar uma nova tecnologia (Weisfeld 2009).

Anteriormente a POO, o sistema utilizado se chamava *Procedural* (Processual), em um código existiam, simplificadamente, funções e variáveis que se relacionavam através de um código global. Esta abordagem provocava grande redundância no código, forçando várias linhas de código iguais ou muito semelhantes para executar ações idênticas em variáveis diferentes.

Com a implantação da abordagem de objetos, foram criados alguns conceitos explicados na sequência:

Encapsulamento: este conceito se baseia na associação de elementos que se relacionam diretamente, métodos e variáveis, dispersos em um código global, criando assim a prestigiosa classe, que é um objeto. Criando uma instancia desta classe no código, é possível acessar, conforme os modificadores de acessos, seus atributos e métodos. Tal capacidade agregar informações aumenta a segurança, permitindo que a mudança dos parâmetros seja feita somente através dos métodos de acesso público definidos previamente;

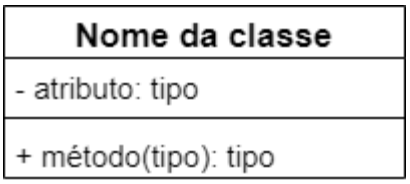
Classes: Uma classe é um compilado de instruções onde estão definidos todos os atributos de um objeto e todas as ações que serão realizadas por aquele objeto através de métodos. Após a declaração da classe, vários objetos poderão ser instanciados usando todos os atributos declarados na classe, sem necessidade de repetir o código (Weisfeld 2009). Na (Figura 1), é ilustrada a estrutura de uma classe através de um diagrama de classe UML, que será detalhado mais a frente, contendo um nome, seus atributos, e métodos. O símbolo que precede atributos e métodos é chamado de visibilidade do item, podendo atribuir visibilidade pública, no pacote, protegida ou privada. A visibilidade, ou modificadores de acesso, diz respeito aos acessos que outros componentes de um sistema têm em relação a classe em questão, conforme a (Tabela 1);

Tabela 1 - MODIFICADORES DE ACESSO

Acesso	Símbolo	Descrição
Público *(public)*	+	O tipo ou membro pode ser acessado por qualquer outro código no mesmo pacote ou outro pacote que faz referência a ele
Privado *(private)*	-	O tipo ou membro só pode ser acessado por código na mesma classe ou estrutura (*struct*)
Protegido *(protected)*	#	O tipo ou membro só pode ser acessado por código na mesma classe ou estrutura (*struct*), ou em uma classe derivada
Interno *(internal)*	~	O tipo ou membro pode ser acessado por qualquer código no mesmo pacote, mas não de outro conjunto.

FONTE: O autor, 2018.

Figura 1 - EXEMPLO DE CLASSE



FONTE: O autor, 2018.

Herança: Na POO é possível muitas vezes evitar o retrabalho de escrever códigos repetidos e a causa disso é a herança. Para entender este conceito é preciso saber que uma superclasse é um tipo especial de classe, que a partir dela serão derivadas novas subclasses que herdam suas propriedades (COLLA 2013). Desta maneira, é possível definir herança como a capacidade de subclasses utilizarem os parâmetros ou métodos que já foram declarados anteriormente em suas classes mães (superclasses) sem precisar repetir código;

Polimorfismo: Após entender o conceito de herança é possível definir o polimorfismo. Existe a possibilidade de fazer alterações nos métodos que foram herdados da superclasse. Muitas vezes os métodos que estão sendo reaproveitados necessitam de parâmetros de entrada diferentes para cada uma das subclasses que o estão herdando, isto pode ser feito com a sobrecarga, do inglês *overload,* que é a nova declaração do construtor do método herdado com os novos parâmetros de entrada na subclasse (Weisfeld 2009).

2.5. DIAGRAMAS UML

A UML (Unified Modeling Language) é uma linguagem visual desenvolvida para modelar negócios, processos, análises e implementações de software.

De acordo com a obra The Unified Modeling Language Reference Manual, segunda edição, esta linguagem tem a finalidade de ser simples e capaz de representar todos os sistemas de viés prático que precisem ser construídos, mesmo através de linguagens e tecnologias modernas (Booch 2004)

Ainda de acordo com o livro, de uma maneira específica ao projeto em questão, a representação visual da UML é feita através de 2 grandes áreas: estrutural na forma dos diagramas de classe, diagramas de colaboração e diagramas de caso de uso), e de comportamento dinâmico através diagramas sequenciais. Que serão detalhados na sequência.

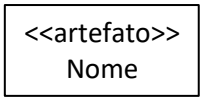
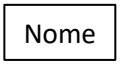
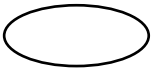
2.5.1. DIAGRAMA DE CASO DE USO

Os diagramas caso de uso, são utilizados para apresentar as interações entre os componentes de um modelo de software ou negócio. Geralmente são utilizados para ilustrar a proposta do software entre o time de desenvolvimento e seu cliente.

Nas Tabelas (Tabela 2 e Tabela 3) estão descritos de maneira concisa alguns elementos que fazem parte do diagrama. A Tabela 2 apresenta os principais elementos para o entendimento do próximo capítulo deste trabalho. A Tabela 3 mostra as possíveis relações entre os elementos da Tabela 2.

Para o melhor entendimento do funcionamento do diagrama de caso de uso, vamos fazer uso do exemplo na Figura 2, onde o limite do sistema está representado no interior do retângulo, contendo dentro dele uma forma oval que representa as funcionalidades que podem ser implementadas no sistema. Ao redor do sistema se encontram os atores, que representam objetos que interagem com o sistema através de uma funcionalidade. A linha solida ligando os atores e a funcionalidade representa uma relação de associação, que de maneira abrangente significa uma interação entre os elementos conectados.

Tabela 2 - CLASSIFICADORES

Objeto	Função	Notação
Ator *(actor)*	Entidade exterior ao sistema	
Artefato *(artifact)*	Uma parte física do sistema	
Classe *(class)*	Um objeto modelado do sistema	
Caso de uso *(use case)*	Um comportamento ou funcionalidade do sistema para interagir com outras entidades	

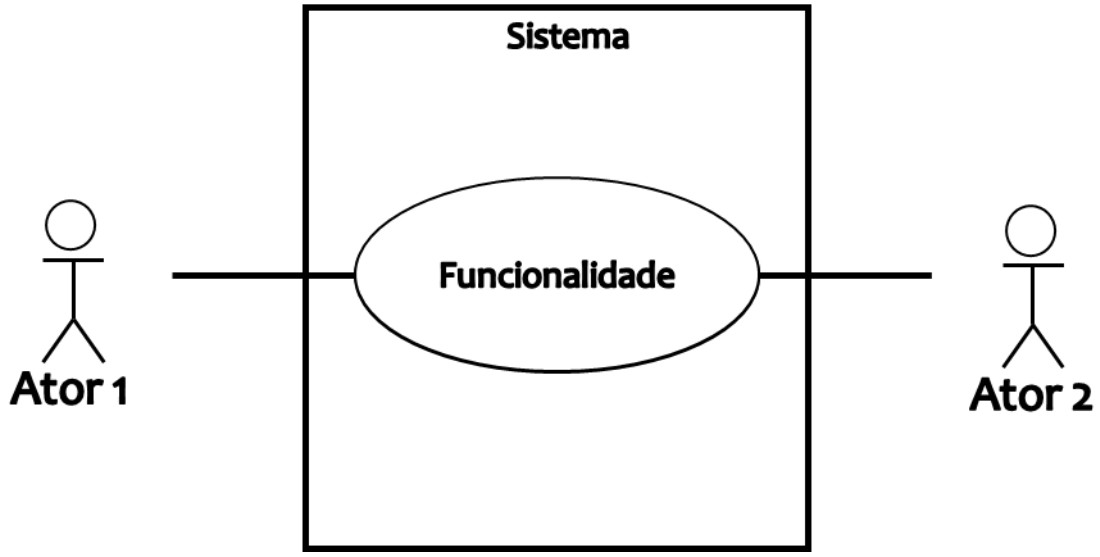
FONTE: Adaptado de (Booch 2004) p.48 – Table 4-1.

Tabela 3 - RELAÇÕES

Relação	Função	Notação
Associação *(association)*	Um comportamento de interação geral entre objetos	
Extensão *(extend)*	Um comportamento dependente de uma condição para ser executado a partir de outra funcionalidade	
Inclusão *(include)*	Um comportamento que é necessariamente executado caso a funcionalidade base ocorra	
Herança *(inheritance)*	Um comportamento entre casos de uso na qual a ponta da flecha indica a funcionalidade que herda atributos	
Agregação (agregation)	Quando um objeto agrega outro	
Realização (realization)	Relação entre a interface e a classe de implementação	

FONTE: Adaptado de (Booch 2004) p.52, p.79 – Table 4-2, 6-1.

Figura 2 - PRIMEIRO EXEMPLO - CASO DE USO

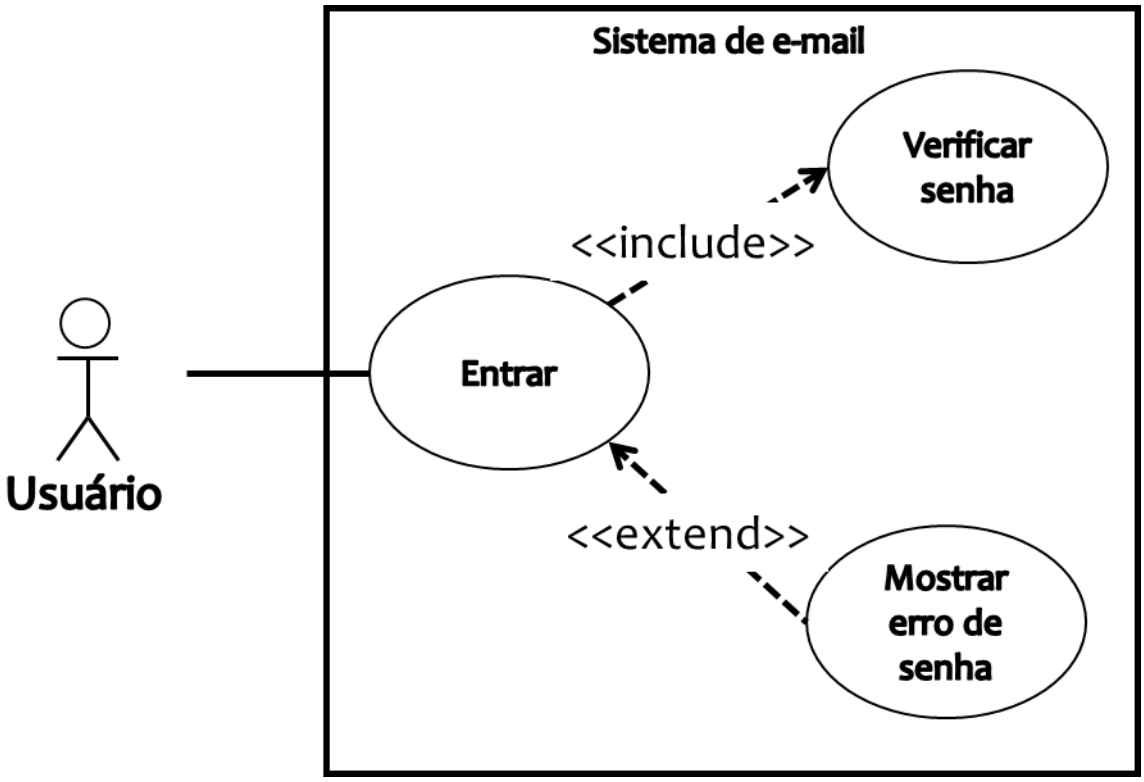


FONTE: O autor, 2018.

Outro tipo de relação que será necessário conhecer para a continuação do projeto são as relações de *include* (inclusão) e *extend* (extensão) (VIEIRA 2003), que serão detalhadas através do segundo exemplo (Figura 3)Um usuário que deseja entrar em um sistema genérico de e-mail irá interagir com o sistema através da

funcionalidade “Entrar”. Esta funcionalidade por sua vez ocorrerá apenas se a outra tarefa chamada “Verificar senha” ocorrer, isto por que existe uma relação de inclusão entre as duas, na qual a primeira somente será executada caso a segunda seja. Já a relação de “Entrar” com “Mostrar erro de senha” é uma relação de extensão, ou seja, ela poderá ou não ocorrer quando a primeira funcionalidade for executada, apenas se ela atender critérios específicos será executada. Neste caso o critério específico seria a senha digitada pelo usuário estivesse incorreta.

Figura 3 - SEGUNDO EXEMPLO – CASO DE USO



FONTE: O autor, 2018.

Abaixo segue uma tabela que sumariza as relações mais importantes bem como as que serão utilizadas nos diagramas de caso de uso deste projeto, ressaltando que existem várias outras relações que não estão sendo tratadas neste trabalho.

2.5.2. DIAGRAMA DE CLASSES

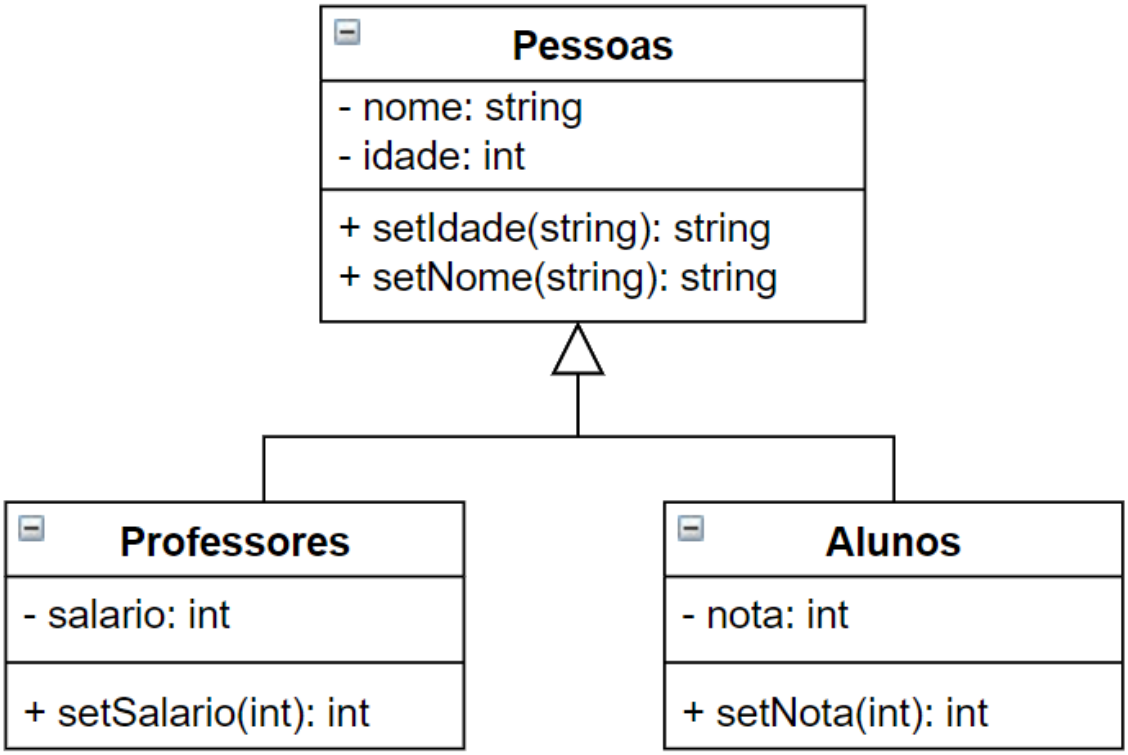
Um diagrama de classes é uma ilustração das relações e das dependências do código-fonte entre as classes. Nesse contexto, uma classe define os métodos e variáveis em um objeto, que é uma entidade específica em um programa ou a unidade

de código que representa essa entidade. Diagramas de classes são úteis em todas as formas de programação orientada a objetos (Weisfeld 2009).

Em um diagrama de classes, as classes são organizadas em grupos que compartilham características comuns. Um diagrama de classes se assemelha a um fluxograma no qual as classes são retratadas como caixas, cada caixa com três retângulos dentro (Figura 1). O retângulo superior contém o nome da classe; o retângulo do meio contém os atributos da classe; o retângulo inferior contém os métodos, também chamados de operações, da classe. As linhas, que podem ter setas em uma ou nas duas extremidades, conectam as caixas. Essas linhas definem os relacionamentos, também chamados de associações, entre as classes.

Na Figura 1 está representado um exemplo de herança entre classes. A classe “Pessoas” é a superclasse que fornece os atributos nome e idade, bem como os métodos “setIdade()” e “setNome()” para as classes derivadas, chamadas “Professores” e “Alunos”.

Figura 4 - EXEMPLO DE HERANÇA



FONTE: O autor, 2018.

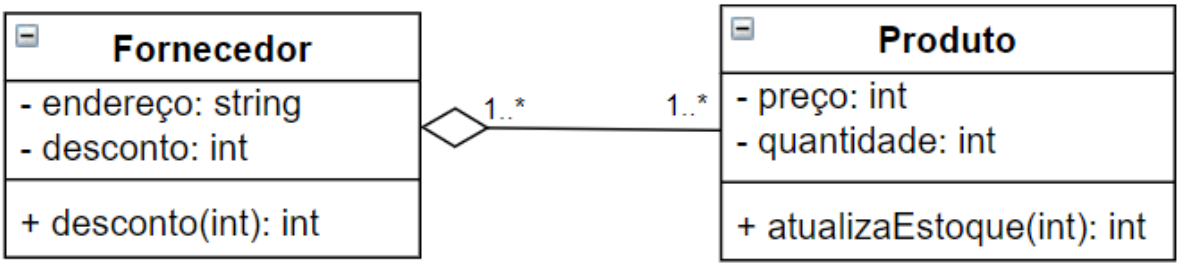
Juntamente com as relações, é possível determinar o número mínimo e máximo de instancias de um determinado elemento descrito no diagrama. A Tabela 4 apresenta as possibilidades de numeração. Os números são colocados nas extremidades das ligações entre os elementos, conforme exemplificado na Figura 5.

Tabela 4 - MULTIPLICIDADE

Multiplicidade	Opção	Cardinalidade
0..0	0	A coleção deve estar vazia
0..1		Uma ou nenhuma instancia
1..1	1	Exatamente uma instancia
0..*	*	Zero ou mais instancias
1..*		Pelo menos uma instancia
5..5	5	Exatamente 5 instancias
m..n		De m a n instancias

FONTE: Adaptado de (Organization, Multiplicity s.d.).

Figura 5 - EXEMPLO DE AGREGAÇÃO E MULTIPLICIDADE



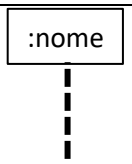
FONTE: O autor, 2018.

2.5.3. DIAGRAMA DE SEQUÊNCIA

Um diagrama de sequência, no contexto da UML, representa a colaboração de objetos e tem por finalidade definir as sequências de eventos entre objetos. Um diagrama de sequência é um componente essencial para se entender a ordem cronológica do uso de um software, por isso, o diagrama de sequência também é conhecido como diagrama de tempo ou de evento (Booch 2004).

Neste tipo de diagrama, existem alguns classificadores específicos, demonstrados na Tabela 5. A dimensão vertical do diagrama, representa o eixo do tempo, de cima para baixo o tempo aumenta. A dimensão horizontal representa a ordem na qual a comunicação entre objetos acontece.

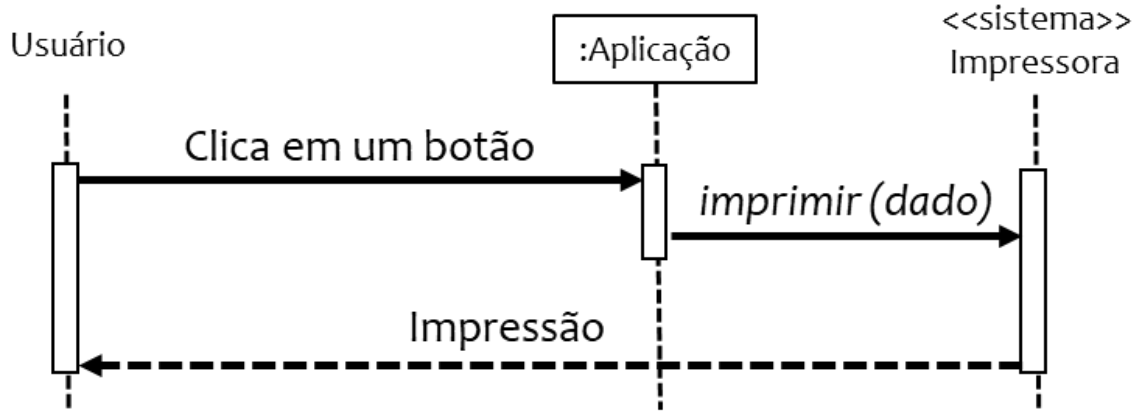
Tabela 5 - CLASSIFICADORES DIAGRAMA DE SEQUÊNCIA

Entidades	Função	Notação
Mensagem *(message)*	Símbolo de troca de mensagens entre objetos	
Mensagem de retorno *(return message)*	Símbolo de retorno de mensagem entre objetos	
Linha de vida *(life line)*	Representação de um único objeto e suas interações	
Ativação *(activation)*	Símbolo que representa a atividade de objeto ou ator	

FONTE: Adaptado de (Organization, Sequence Diagrams s.d.).

O exemplo da Figura 6 mostra a interação entre o usuário, que utiliza uma aplicação genérica para imprimir um dado. A primeira mensagem é enviada do “Usuário” para a “Aplicação”, ela é enviada através do clique de um botão. A aplicação recebe essa mensagem, e envia uma outra mensagem para a “Impressora”. Como retorno, a impressora vai retornar uma impressão ao “Usuário”. A ordem cronológica é extremamente importante neste tipo de diagrama, ou seja, a impressão de um dado não poderia acontecer antes que o “Usuário” clicasse no botão.

Figura 6 – EXE MPLO DE DIAGRAMA SEQUENCIAL



FONTE: O autor, 2018.

2.6. TYPESCRIPT

O TypeScript é uma linguagem de programação de código aberto desenvolvida pela Microsoft, licenciada pela Apache Software Foundation e disponibilizada ao público em outubro de 2012 (Fenton 2014).

Uma linguagem de programação é uma nova língua criada artificialmente que serve para escrever um conjunto de instruções que serão seguidas um por computador ao realizarmos ações específicas (Haverbeke 2011).

Existe uma grande variedade de linguagens de programação, e o typescript se destaca por ser “fortemente tipado” e englobar as diretrizes dos códigos em JS já existentes. Um código “tipado” significa que sua sintaxe é rígida e pode apresentar erros durante a compilação, mas as chances de haverem falhas durante a sua execução são muito baixas (Luis Rei 2007).

2.7. NODE.JS

Durante a construção de uma nova aplicação sempre é necessária a configuração do lado do cliente (*front-end*) e do lado dos servidores (*back-end*). Quando o desenvolvimento do *front-end* era feito em Javascript era necessário escolher uma segunda linguagem de programação para a configuração dos servidores, assim tornando o trabalho mais árduo já que cada linguagem de programação possui a sua sintaxe e suas dependências.

Porém o Node.js surgiu com o conceito de “JavaScript everywhere” que foi uma solução para este problema. Com o Node é possível executar um código em JavaScript em diferentes ambientes de desenvolvimento além do web, até mesmo para fazer a configuração de servidores (Rauch 2012).

O Node também possui uma ferramenta de gerenciamento de pacotes (*Node Package Manager*) que auxilia na instalação de todas as dependências do seu projeto, contando com a maior biblioteca de código aberto do mundo (Patel 2018).

O Node suporta os principais protocolos web existentes, como: DNS, HTTP, TCP, TLS, SSL e UDP. Além dos principais sistemas operacionais, como: Microsoft Windows, MacOS, SmartOS, IBM Aix, Linux e o FreeBSD (Node.js s.d.).

2.8. BANCOS DE DADOS

2.8.1. BANCO DE DADOS SQL

O modelo de banco de dados SQL, ou também chamado de modelo relacional, foi criado por Edgar Codd, um desenvolvedor da IBM, em 1970. Este modelo é suportado por diversos Sistemas de Gerenciamento de Banco de Dados (SGBDs) como o IBM DB2, Microsoft SQL Server, MySQL, Oracle, entre outros. Estes sistemas controlam o armazenamento, a exclusão, a segurança, a integridade e a recuperação dos bancos de dados (Ben-Gan 2012).

Um banco de dados relacional é baseado em tabelas que são compostas por linhas e colunas. Estas linhas são responsáveis por possuir uma chave primária que será usada como um identificador único para cada registro na tabela. Por sua vez cada coluna armazena um dado com um tipo predefinido que será indexado a chave primária da linha em que está sendo escrito.

Para exemplificar o que foi dito anteriormente temos a Tabela 6 com 4 linhas e 7 colunas, onde a primeira coluna é a chave primária que tem um valor único para cada linha e as colunas seguintes são atributos relacionados ao “Id”, cada um destes atributos tem o seu tipo (inteiro, caractere, data, etc.) definidos previamente quando o banco de dados é configurado (SQL Server Documentation 2016).

Tabela 6 - EXEMPLO DE BANCO DE DADOS SQL

Id	Nome	Sobrenome	E-mail	Idade	Cidade	Estado
1	Evelyn	Mendes	evelyn@transnerd.com	35	Porto Alegre	RS
2	John	Doe	john@doe.com	20	Rio de Janeiro	RJ
3	Esmeralda	Silva	esmeralda@email.com	44	Florianópolis	SC
4	João	Silva	joao@email.com	66	Bagé	RS

FONTE: O autor, 2018.

2.4.2 BANCO DE DADOS NOSQL

O modelo de banco de dados NoSQL, também chamado de modelo não relacional, foi proposto pela primeira vez em 1998 por Carlo Strozzi quando estava sendo discutido um banco de dados de código aberto sem uma interface SQL, porém esta ideia foi abandonada pouco tempo depois devido a limitações na tecnologia da época. Em 2009 o conceito de um banco de dados não relacional voltou a ser discutido

quando Eric Evans, um funcionário da Rackspace, organizou um evento para discutir sobre bancos de dados de código aberto (Fowler 2015).

Os bancos de dados NoSQL são divididos em 4 tipos principais: chave-valor, documento, gráfico, famílias de colunas.

• Chave-valor: este modelo apresenta uma relação 1-1 que permite que um atributo seja indexado a uma chave única para cada objeto, como mostrado na figura abaixo. Este modelo oferece grande escalabilidade, é particionável e é um dos modelos mais constantes. O Amazon DynamoDB, Snapchat Stories, Berkeley DB e o Project Voldemort são tecnologias que usam este modelo (Fowler 2015);

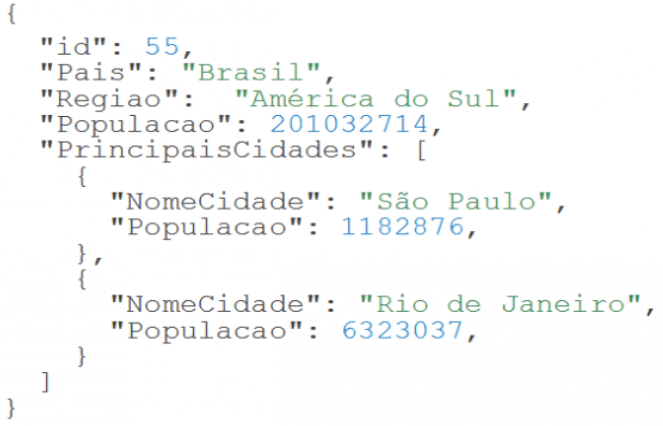
Tabela 7 - EXEMPLO CHAVE-VALOR

Chave	Valor
carro_3345_cor	preto
carro_3345_pneu	17
carro_3365_cor	branco
carro_3365_pneu	15
carro_4560_peso	1215
carro_4715_ano	2016

Fonte: (Micreiros s.d.)

• Documento: este modelo trabalha com arquivos JSON, BSON e XML para a apresentação dos dados no servidor, como mostrado na Figura 7. Estes documentos mostram os dados de forma hierárquica, similar a uma árvore de decisões. Este modelo também facilita para a construção de queries para a extração de dados. O MongoDB, CouchDB e o RavenDB utilizam esta tecnologia (Fowler 2015);

Figura 7 - EXEMPLO DE DOCUMENTO



FONTE: (GROFFE 2016)

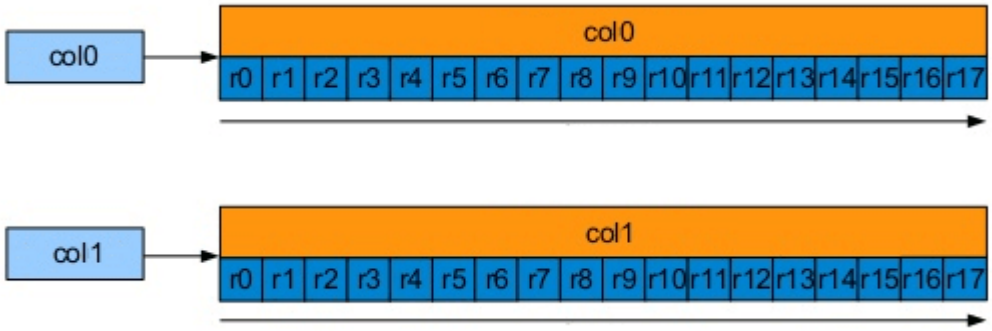
• Gráfico: este modelo é representado por seus objetos e pelas relações existentes entre eles, como mostrado na imagem abaixo. Estas relações são usadas para identificar padrões entre os objetos e para a análise dos dados. Muitos mecanismos de busca e redes sociais utilizam este modelo, assim como o OrientDB, Neo4j e o Giraph (Fowler 2015);

Figura 8 - EXEMPLO DE GRÁFICO

Fonte: (Micreiros s.d.)

• Famílias de colunas: este modelo organiza os dados em múltiplas colunas associadas a uma linha com uma chave única, estas colunas podem ser adicionadas, removidas ou modificadas a qualquer momento, sem afetarem as linhas primárias. É importante notar que as colunas podem ser adicionadas apenas para algumas linhas específicas e não necessariamente para todo o banco de dados. O Amazon DynameDB, Cassandra e HBase e o são exemplos de sistemas que usam este modelo (Fowler 2015).

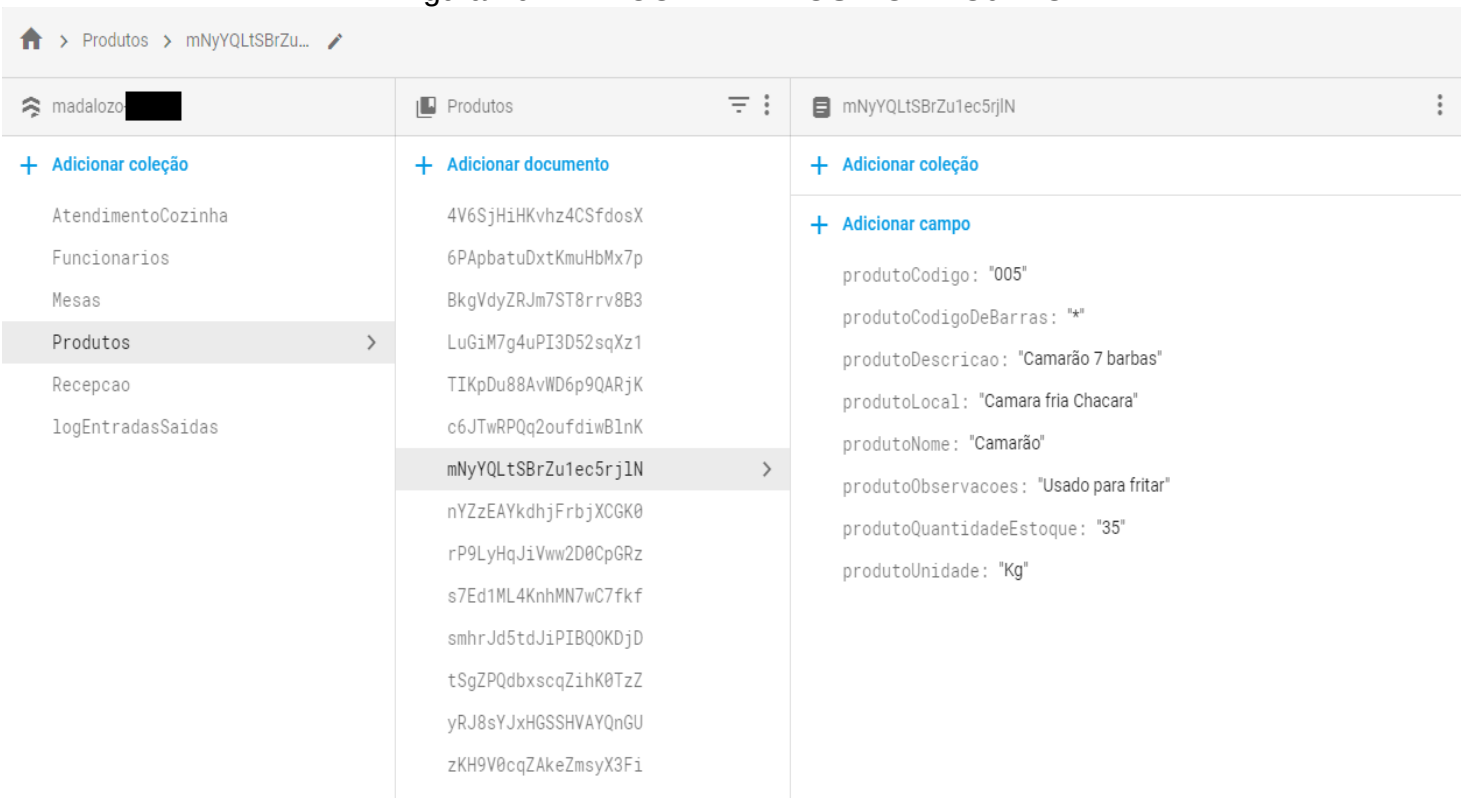
Figura 9 - EXEMPLO DE FAMÍLIAS DE COLUNAS



FONTE: (GROFFE 2016)

A escolha do autor pelo banco de dados NoSQL devido a sua atualização em tempo real, pela oferta de ferramentas gratuitas como FireBase pela Google. Na Figura 10 está ilustrada a estrutura do banco de dados do trabalho.

Figura 10 - BANCO DE DADOS DO PROJETO



FONTE: O autor, 2018.

2.9. FIREBASE

O Firebase surgiu como uma alternativa aos usuais bancos de dados SQL e suas limitações, inicialmente foi desenvolvido pela Firebase Inc., 2011, e após

3 anos foi incorporado pela Google, que suporta a aplicação atualmente. Esta ferramenta possibilita usar um banco de dados NoSQL para aplicações mobile e web, assim como uma gama de outros serviços, alguns exemplos são:

• Cloud Firestore, é um banco de dados NoSql do tipo documento altamente escalonável e com servidores na nuvem. Uma de suas principais vantagens é a sincronização offline que permite que o usuário acesse e modifique seus dados mesmo sem rede, atualizando no banco de dados quando a conexão voltar (Google s.d.);

• Realtime Database, um dos motivos que fez o Firebase ficar tão famoso é o seu banco de dados ser sincronizável com praticamente todas as grandes bibliotecas disponíveis para IOS, Objective-C, Swift, Android, JS, Java e Node.js. Como o servidor e a aplicação são compatíveis e usam a biblioteca nativa, isso elimina a necessidade de módulos separados para realizar as conexões entre ambos, o que aumenta consideravelmente o tempo de resposta dos servidores (Google s.d.);

• Firebase Storage, este serviço é responsável pelo fluxo de dados, downloads e uploads, realizados no servidor (Google s.d.);

• Firebase Analytics, que fornece dados sobre o uso dos servidores e de seus serviços para o cliente (Google s.d.);

• Firebase Cloud Messaging, que permite enviar mensagens de alerta para dispositivos mobile (Android e IOS) e para aplicações web (Google s.d.);

• Firebase Auth, é um serviço de autenticação fornecido pela Google que é processado no lado do cliente (Front-end). Também oferece suporte a autenticação por redes sociais como Facebook, GitHub, Twitter, entre outros (Google s.d.).

2.10. ANGULAR

O AngularJS é um framework que foi criado para JavaScript, porém após algumas versões também passou a ser funcional para códigos em TypeScript, com foco no desenvolvimento do front-end para aplicações web. O Angular foi disponibilizado para o público em outubro de 2010 e é suportado majoritariamente pela Google (Gavigan 2018).

A escolha do Angular foi feita por causa de várias vantagens apresentadas, entre elas:

• A possibilidade do uso do Typescript para o desenvolvimento tanto do front-end como do back-end com o Node.js;
• É suportado pelas principais plataformas de desenvolvimento como o Node.js, .NET e PHP, possibilitando a execução em diferentes ambientes;
• O Code Splitting possibilita separar o código em componentes e processa somente o que é necessário para rodar a página solicitada, aumentando o desempenho;
• Possui um grande variedade de templates para a criação de interfaces gráficas, dando ao usuário maiores opções para uma UI agradável e prática;
• Seu Ambiente de Desenvolvimento Integrado possui um sistema responsivo à erros e uma ótima complementação inteligente de código (Angular s.d.).

2.11. Visual Studio Code

O Visual Studio Code, também chamado de VS Code, é um editor de código aberto desenvolvido pela Microsoft em 2015 que roda em Windows, MacOS e Linux. Seu sistema nativo suporta TypeScript, JavaScript e Node.js, porém também possui extensões adicionais para C#, C++, Java, Python e PHP (Blog 2015).

O VS Code segue os mesmos princípios do Visual Studio quanto ao processamento, a construção e ao debug de programas, porém alguns recursos mais avançados estão desabilitados para fornecer uma versão mais leve e multiplataforma.

3. MATERIAIS E MÉTODOS

3.1. REQUISITOS DO PROJETO

O levantamento dos requisitos tem como objetivo compreender as expectativas dos usuários do software a ser desenvolvido. Os requisitos precisam ser claros para que o desenvolvedor possa elaborar a melhor solução para o projeto, sem deixar espaço para ambiguidades (Point s.d.).

Esta etapa é crítica no que diz respeito ao retorno de investimento no projeto. Como um sistema de informações geralmente é utilizado para automatizar processos de um negócio, esses processos da organização devem ser bem compreendidos para que o restante das atividades do processo de desenvolvimento flua de acordo com as reais necessidades do cliente.

3.1.1. RECEPÇÃO

• Gerar senhas para os clientes com possibilidade de impressão;
• Chamar as senhas através de painel eletrônico ou TV;
• Mudar o status da mesa (disponível, em uso, aguardando limpeza, pagando, reservada).
• Agendar ou cancelar reservas de mesa no sistema.

3.1.2. ATENDIMENTO ÁS MESAS

• Comandar no sistema os pedidos e envia-los para a cozinha e bar;
• Fechar a conta;
• Mudar o status da mesa (disponível, em uso, aguardando limpeza, pagando, reservada).

3.1.3. COZINHA E BAR

• Receber os pedidos, impressos, criados pelos garçons;

3.1.4. CAIXA

• Realizar todas as funções atribuídas aos outros módulos exceto administração;
• Registrar no sistema a forma de pagamento da conta e emitir nota fiscal;
• Mudar o status da mesa (disponível, em uso, aguardando limpeza, pagando, reservada).
• Realizar alterações, quando necessárias, em itens cadastrados na mesa;

3.1.5. ADMINISTRAÇÃO

• Realizar todas as funções atribuídas aos outros módulos;
• Gerar relatórios de um determinado período de tempo sobre as compras, vendas ou do faturamento do restaurante;
• Visualizar o balanço do caixa;
• Visualizar as vendas e a comissão por operador;
• Fazer a escala de funcionários;
• Cadastrar funcionários, produtos, fornecedores e clientes.

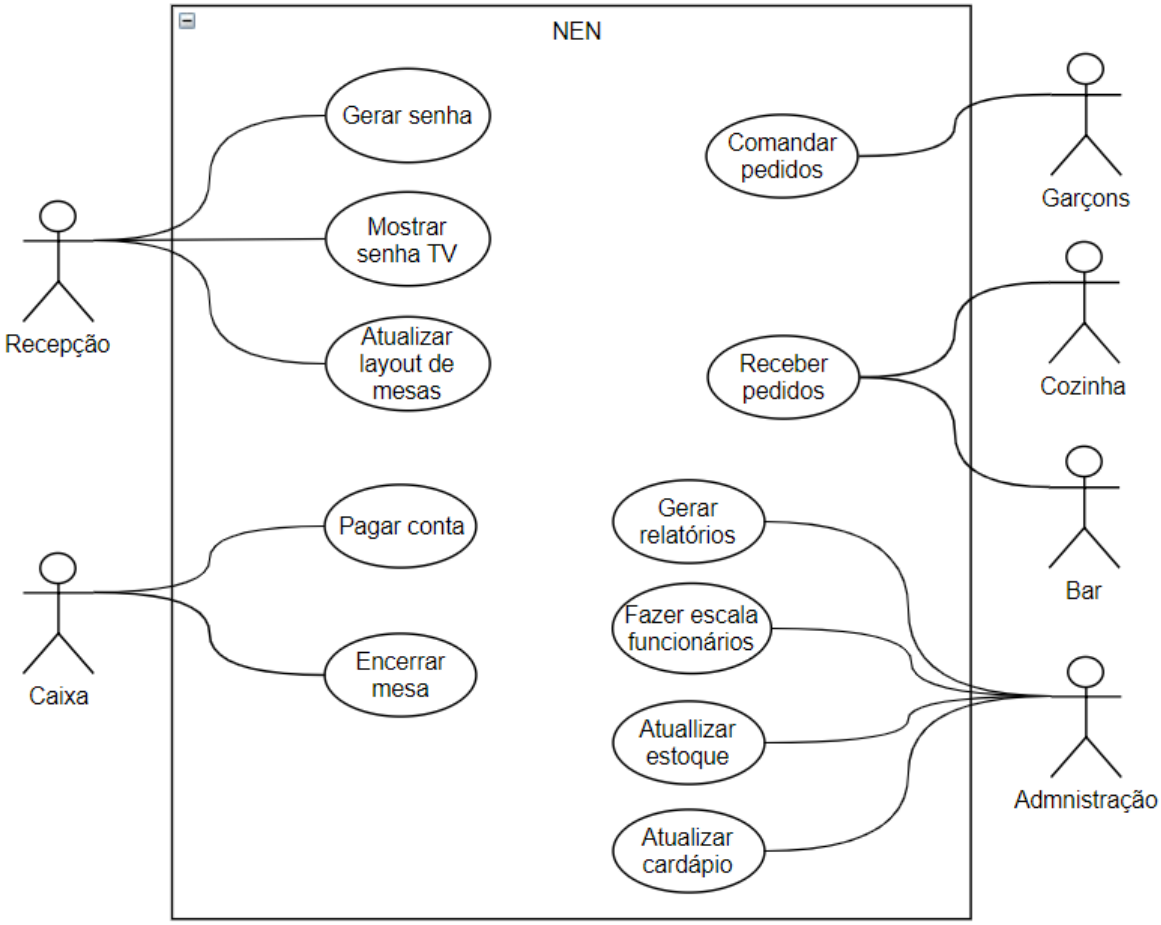
3.2. MODELAGEM DO PROJETO

A partir dos requisitos levantados, foram elaborados os diagramas essenciais para a implementação do projeto, e serão apresentados na sequência do trabalho. O primeiro diagrama, ilustrado na Figura 11 é o diagrama de caso de uso. Nele estão contidos os principais requisitos em forma de funcionalidades que a aplicação deverá ser capaz de executar.

O diagrama de classes foi separado de maneira a simplificar sua já complexa visualização devido à complexidade do projeto. A Figura 12 possui as relações entre as classes propostas enquanto a Figura 13 contempla os atributos e métodos detalhados de cada classe.

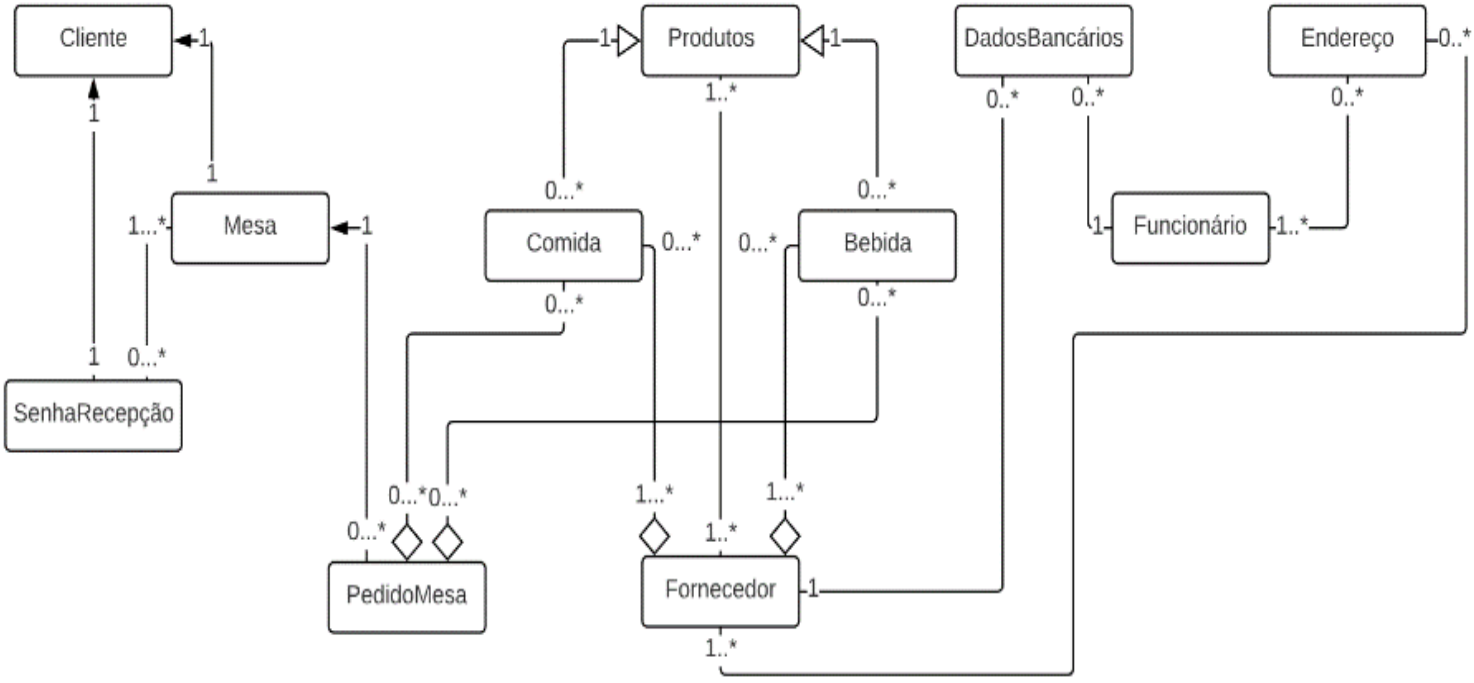
Após a construção do diagrama de classes, foram criados os diagramas sequenciais, que como citado neste trabalho no capítulo 2, explica a ordem em que acontecem as interações entre atores e a aplicação. O primeiro diagrama sequencial está disposto na Figura 14, e diz respeito as interações dos garçons com o NEN. Os outros dois diagramas dizem respeito a Recepção e a Administração e estão respectivamente representados nas Figura 15 e Figura 16.

Figura 11 - DIAGRAMA DE CASO DE USO



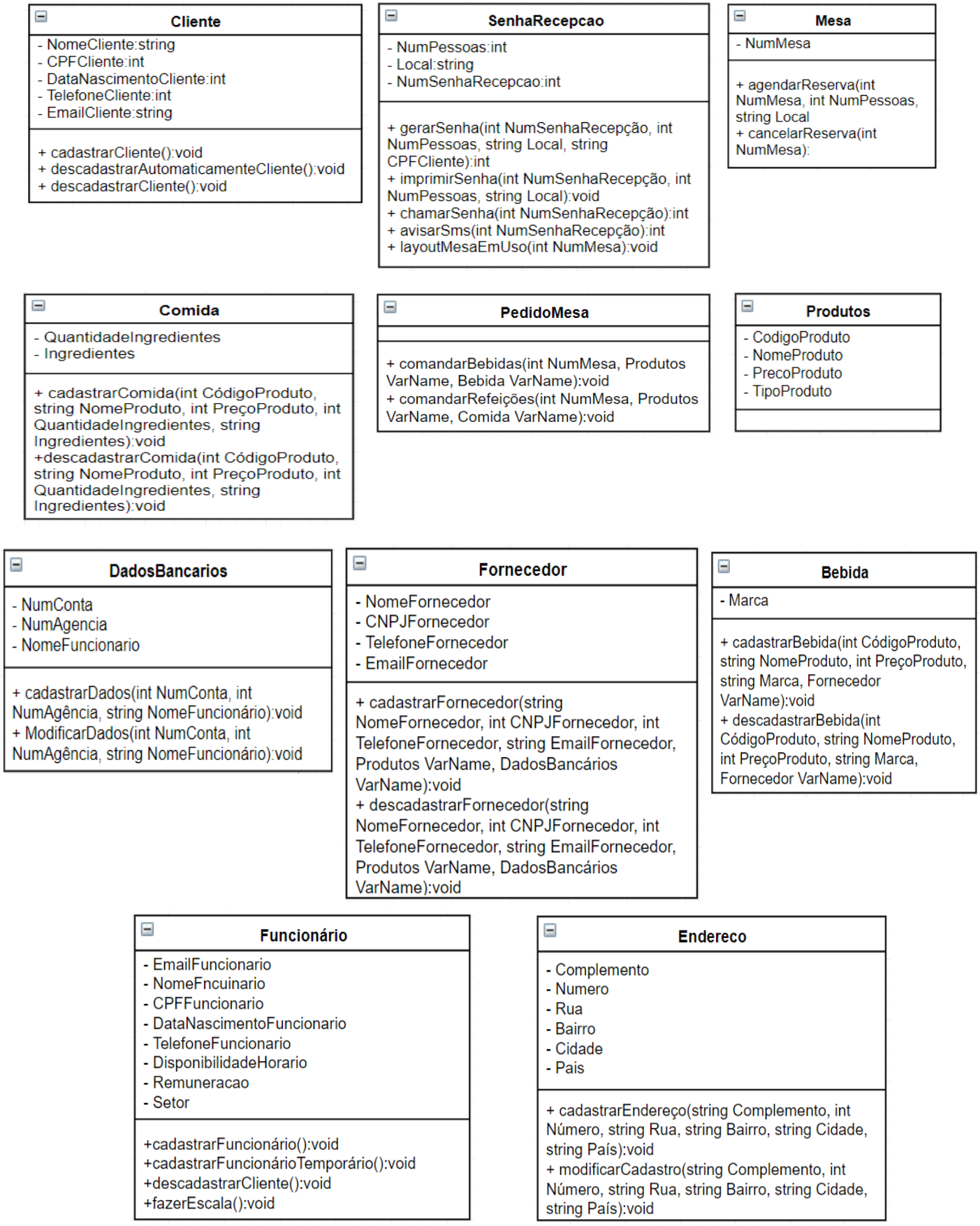
FONTE: O autor, 2018.

Figura 12 - DIAGRAMA DE CLASSES



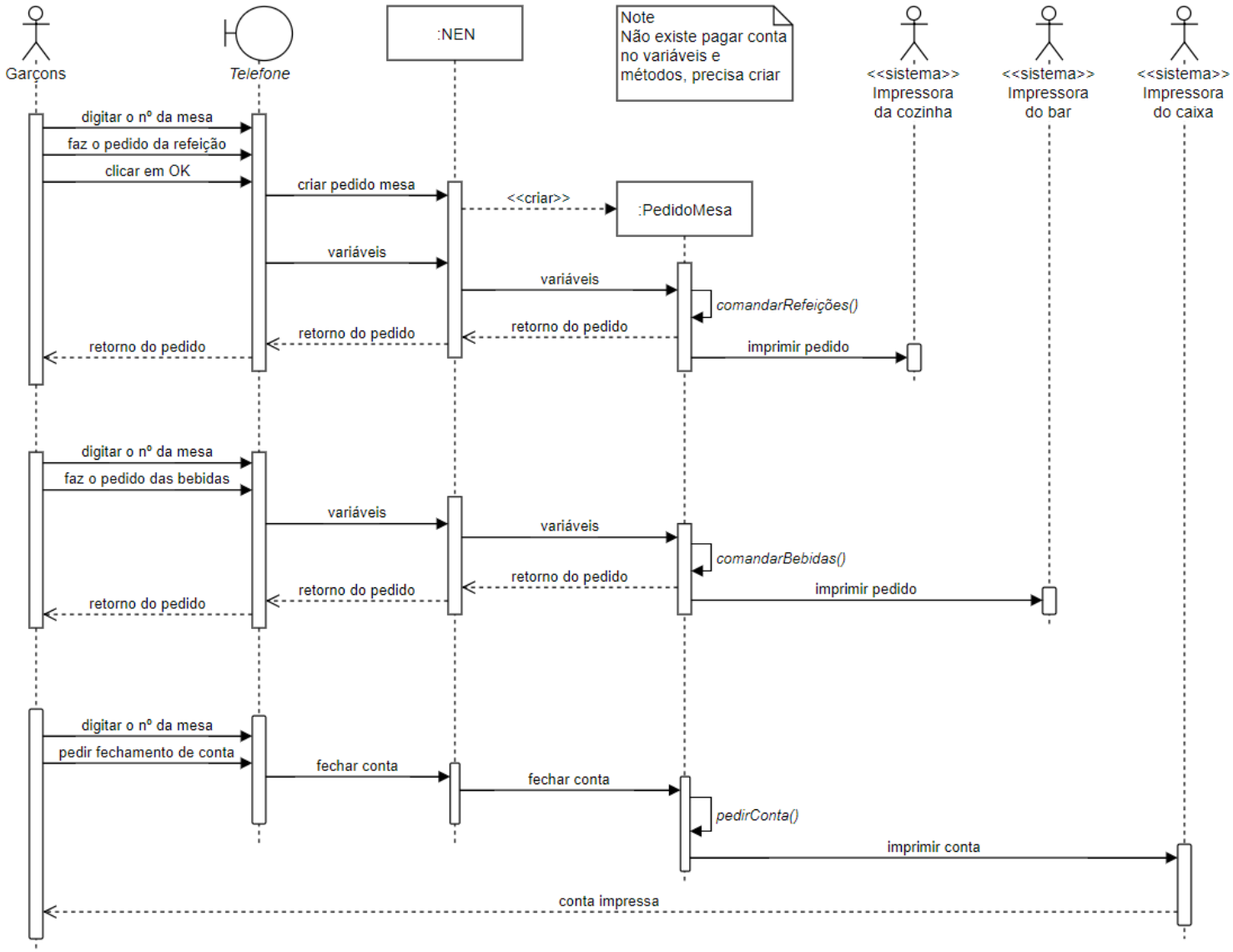
FONTE: O autor, 2018.

Figura 13 - DETALHE DAS CLASSES



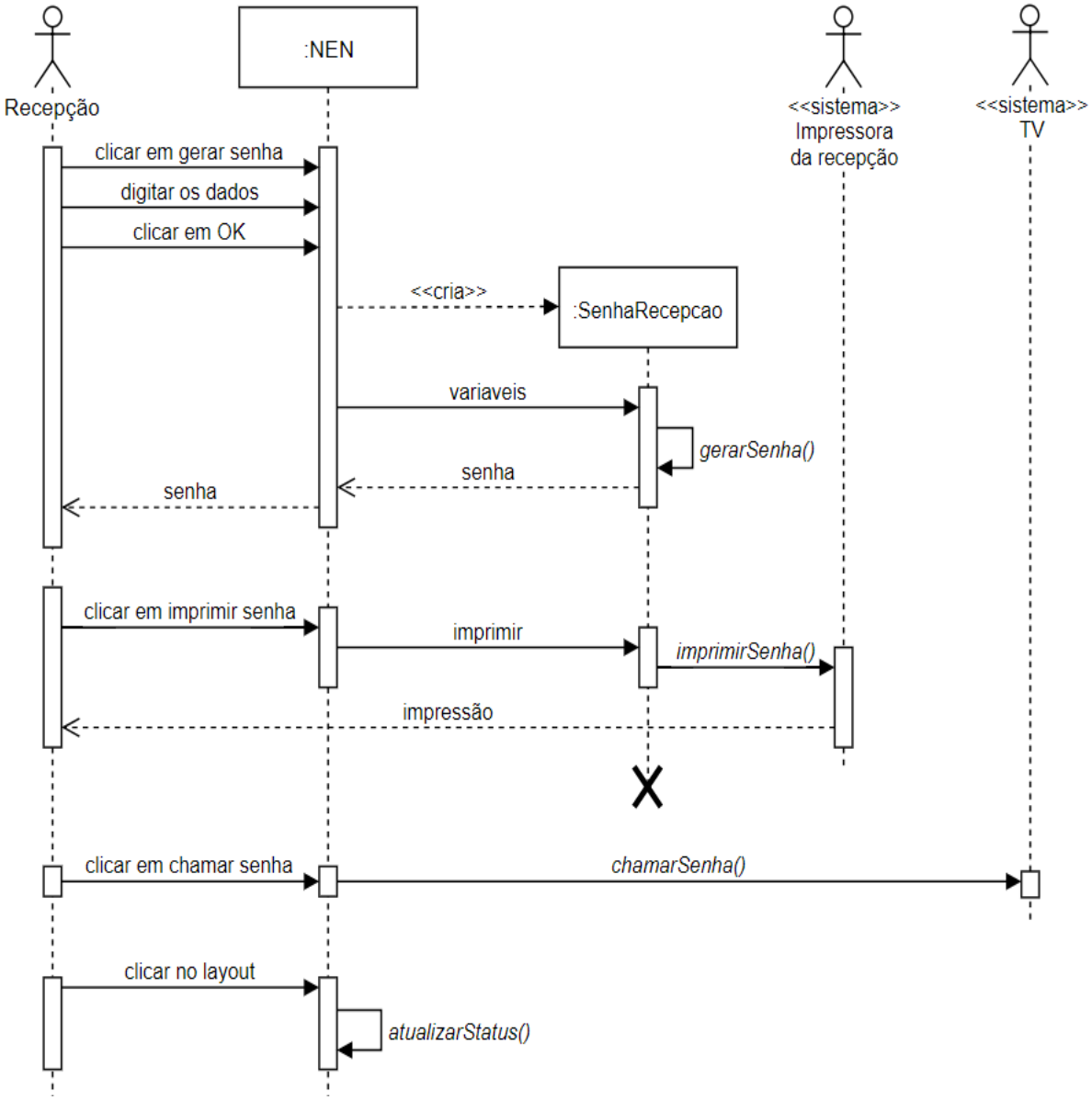
FONTE: O autor, 2018.

Figura 14 - DIAGRAMA SEQUENCIAL GARÇONS



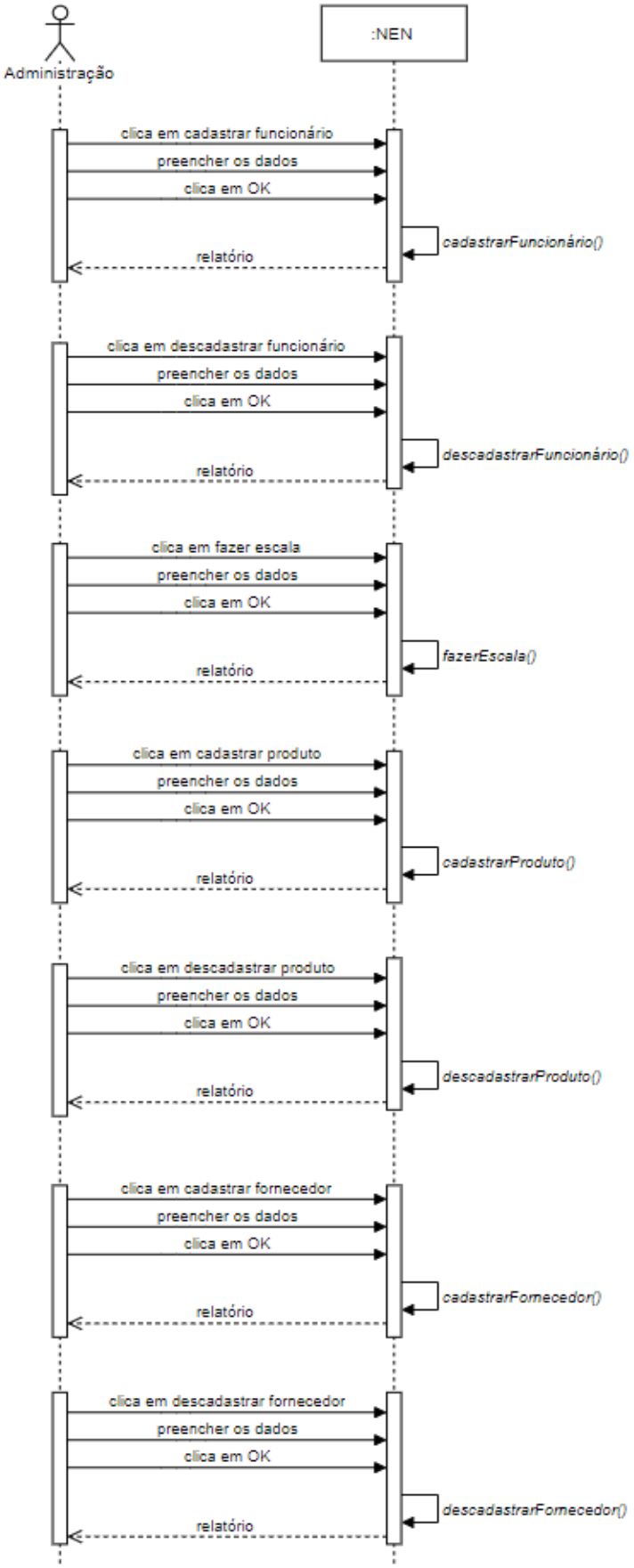
FONTE: O autor, 2018.

Figura 15 - DIAGRAMA SEQUENCIAL RECEPÇÃO



FONTE: O autor, 2018.

Figura 16 - DIAGRAMA SEQUENCIAL ADMINSTRAÇÃO



FONTE: O autor, 2018.

4. DIFICULDADES NO PROJETO

4.1. Plano de riscos e controle

Para melhor entender quais são as possíveis dificuldades durante desenvolvimento, foi elaborado na primeira etapa do projeto um plano de risco levando em consideração a probabilidade de cada condição proposta acontecer e do seu impacto no projeto. Na escala abaixo H representa a classificação menos preocupante, enquanto S representa o maior risco possível.

Tabela 8 - PROBABILIDADE X IMPACTO

	Muito baixo	Baixo	Moderado	Alto	Muito alto
Muito baixa	H	G	F	E	D
Baixa	G	F	E	D	C
Moderada	F	E	D	C	B
Alta	E	D	C	B	A
Muito alta	D	C	B	A	S

FONTE: O autor, 2018

Tabela 9 - CLASSIFICAÇÃO DAS CONDIÇÕES

Nº	Condição	Consequência	Ação	Probabilidade	Impacto	Classificação
1	Deficiência técnica da equipe.	Atraso no desenvolvimento do projeto.	Pesquisa e estudos relacionados ao tema	Moderada	Alto	C
2	Número reduzido de membros da equipe.	Atraso no desenvolvimento do projeto.	Planejamento e organização das atividades	Moderado	Alto	C
3	Pouco tempo de desenvolvimento.	Atraso da versão final do projeto.	Aumento da produtividade dos membros	Alta	Muito alto	A

(Continua)

(Conclusão)

4	Ajustes durante o desenvolvimento.	Mais atrasos no desenvolvimento	Desenvolver de forma a evitar retrabalho	Muito alta	Alto	A
5	Ambiente web precário para execução do software.	Dificuldade na implementação do projeto	Testar o ambiente previamente à instalação do software	Baixa	Muito alto	C
6	Perder arquivos do projeto.	Recomeçar as atividades do zero	Backups periódicos	Muita baixa	Muito alto	D

FONTE: O autor, 2018

Após a conclusão do projeto, das condições previstas, apenas os itens 5 e 6 não tiveram impacto de fato no desenvolvimento do projeto. Os itens que promoveram maiores perdas foram o 1 e 3. Ressaltando a ingenuidade dos autores pela sua falta de experiência na área. Na sequência estão listados alguns dos principais problemas identificados:

• Escolha do Framework: Por não conhecermos os frameworks mais utilizados no mercado, nem a complexidade de sua utilização, precisou ser realizada uma extensa pesquisa para a escolha do ambiente de trabalho;

• Após a escolha do ambiente, foi preciso aprender as linguagens utilizadas na plataforma. Os integrantes não possuíam proficiência em HTML, Typescript e CSS.

• Configuração e utilização do banco de dados: ao escolher trabalhar com um banco de dados NoSql, devido a suas vantagens para o projeto, os integrantes da equipe precisaram investir muito tempo até conseguir realizar comunicações eficientes o suficiente para que fossem implementadas em uma versão para o cliente.

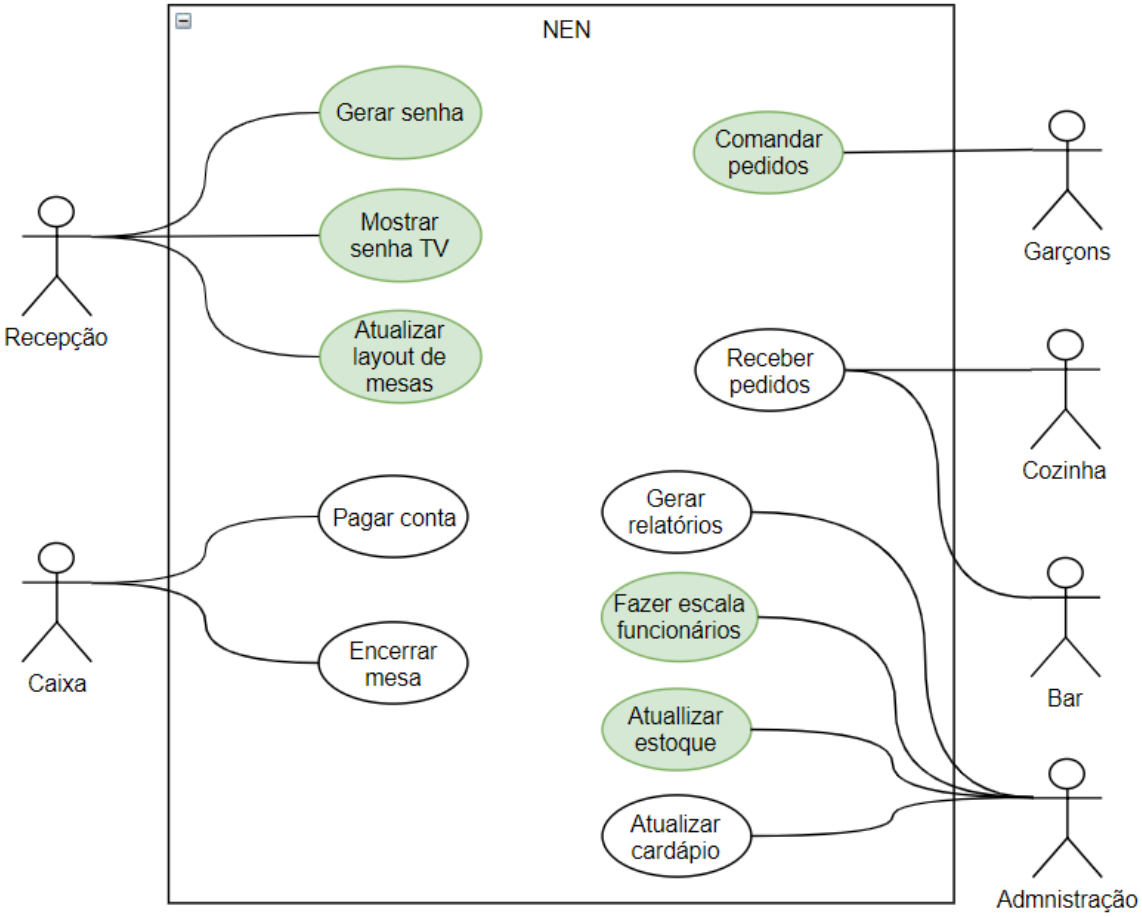
• Mudança dos requisitos do cliente: Durante o desenvolvimento houveram várias mudanças no escopo que foram requisitadas pelo cliente, gerando muito retrabalho, possível fruto dos requisitos do projeto não estarem totalmente adequados no início.

• Limite de transações do banco de dados: ao optar por um plano do Firebase gratuito, tínhamos um limite de transações que poderiam ser feitas diariamente o que limitou o desenvolvimento em alguns momentos.

5. RESULTADOS

Os resultados obtidos podem ser de maneira simples visualizadas através da Figura 17, onde as funcionalidades implementadas com sucesso estão destacadas em verde.

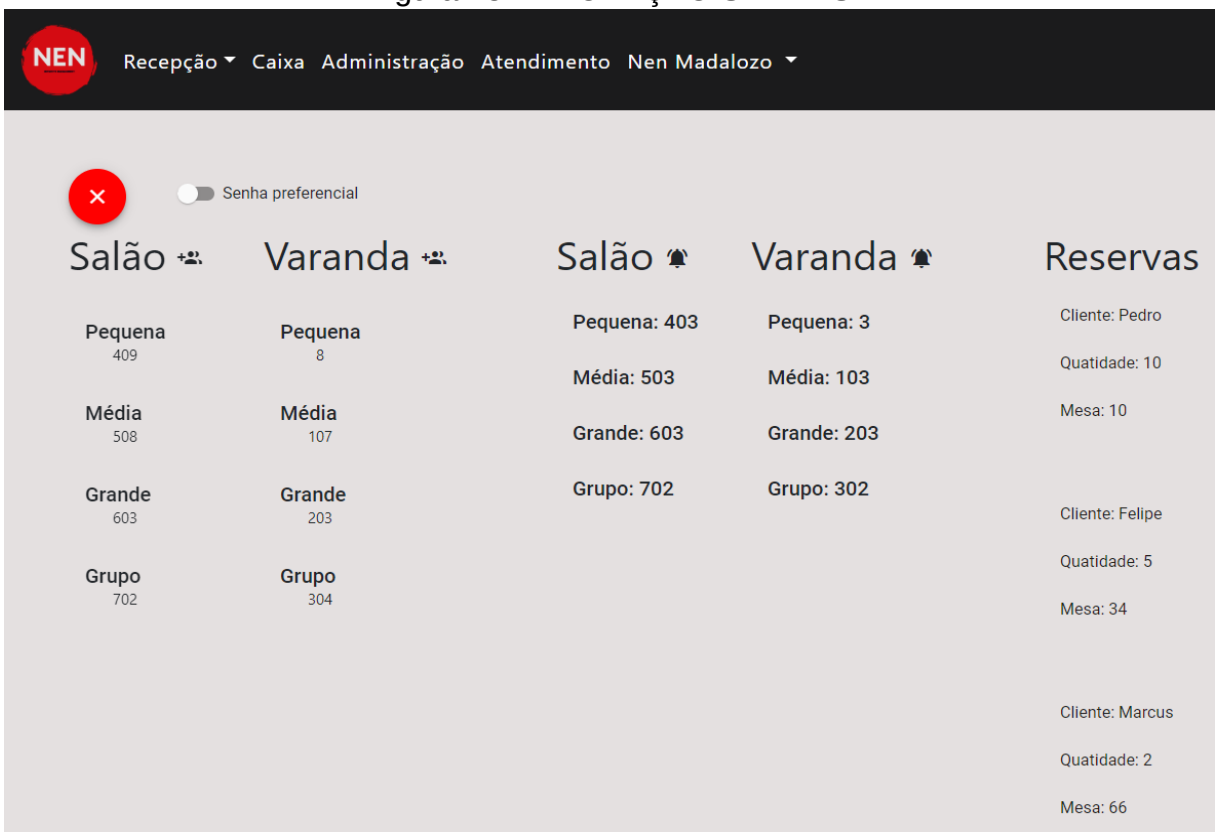
Figura 17 - FUNCIONALIDADES IMPLEMENTADAS



FONTE: O autor, 2018

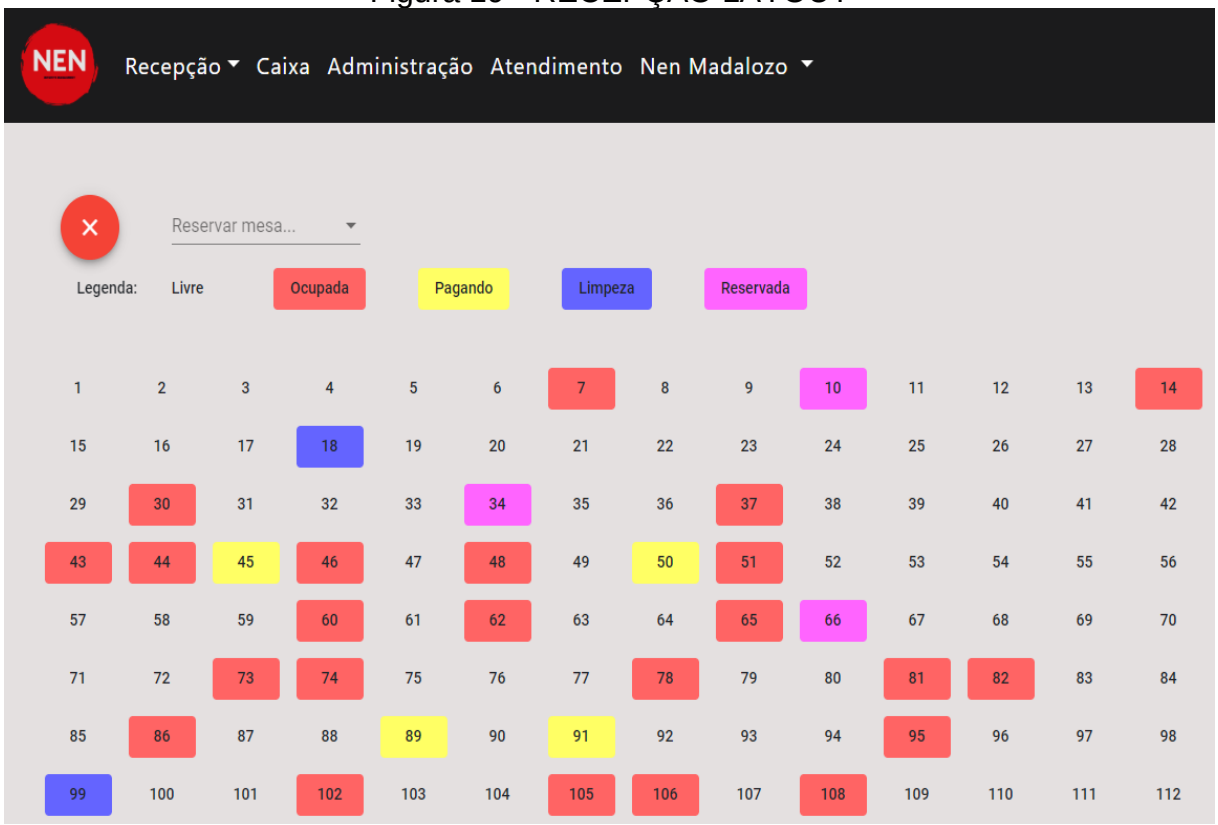
Na sequência serão apresentadas as telas que foram implementadas, começando pelo módulo da recepção. A Figura 18 mostra onde ocorrem a criação das senhas, a chamada das senhas e são mostradas as reservas feitas. A Figura 19 ilustra onde acontecem as mudanças de status das mesas, e a criação de reservas. A Figura 20 mostra a última senha chamada, para a visualização dos clientes.

Figura 18 - RECEPÇÃO SENHAS



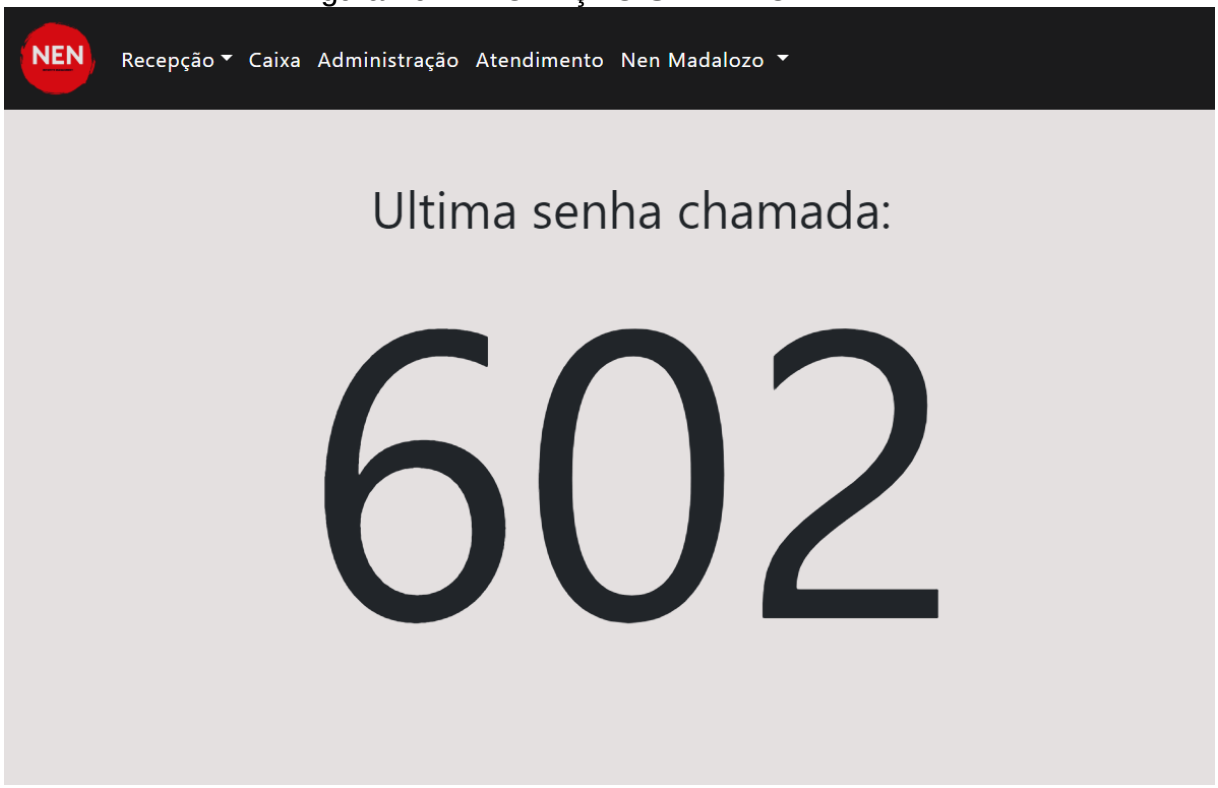
FONTE: O autor, 2018

Figura 19 - RECEPÇÃO LAYOUT



FONTE: O autor, 2018

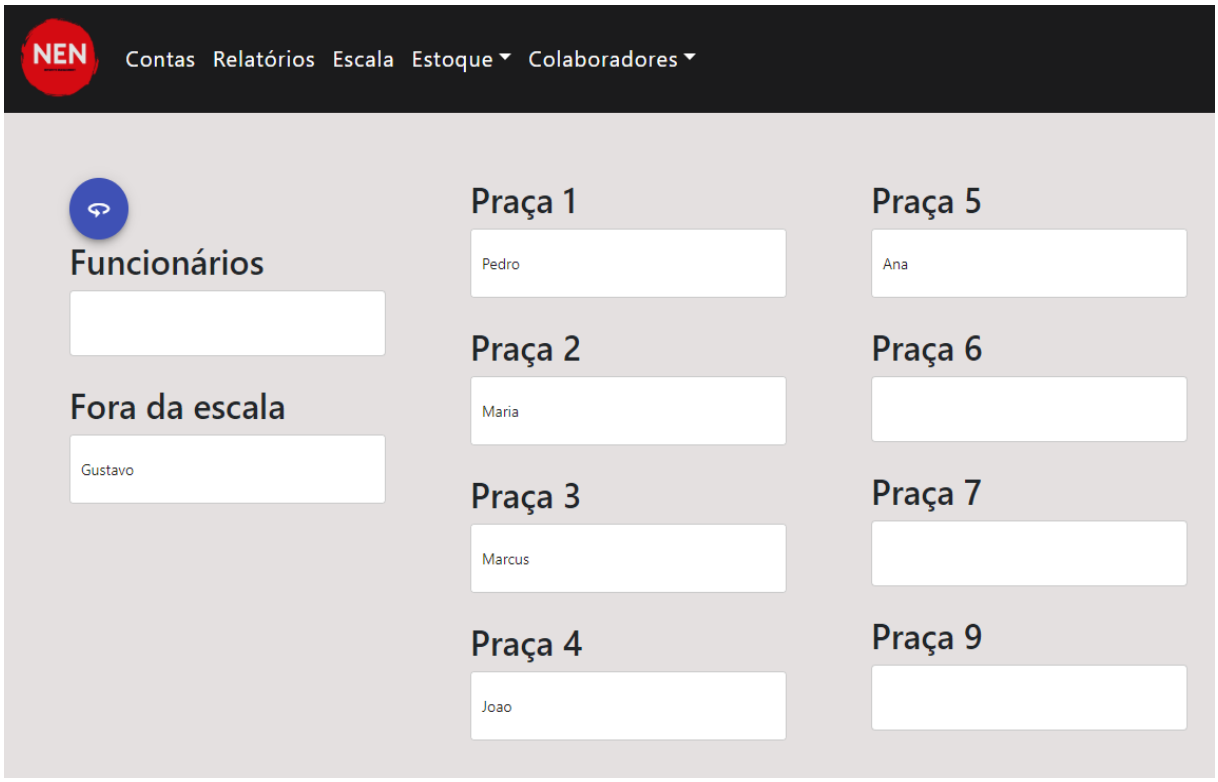
Figura 20 - RECEPÇÃO SENHA CHAMADA



FONTE: O autor, 2018

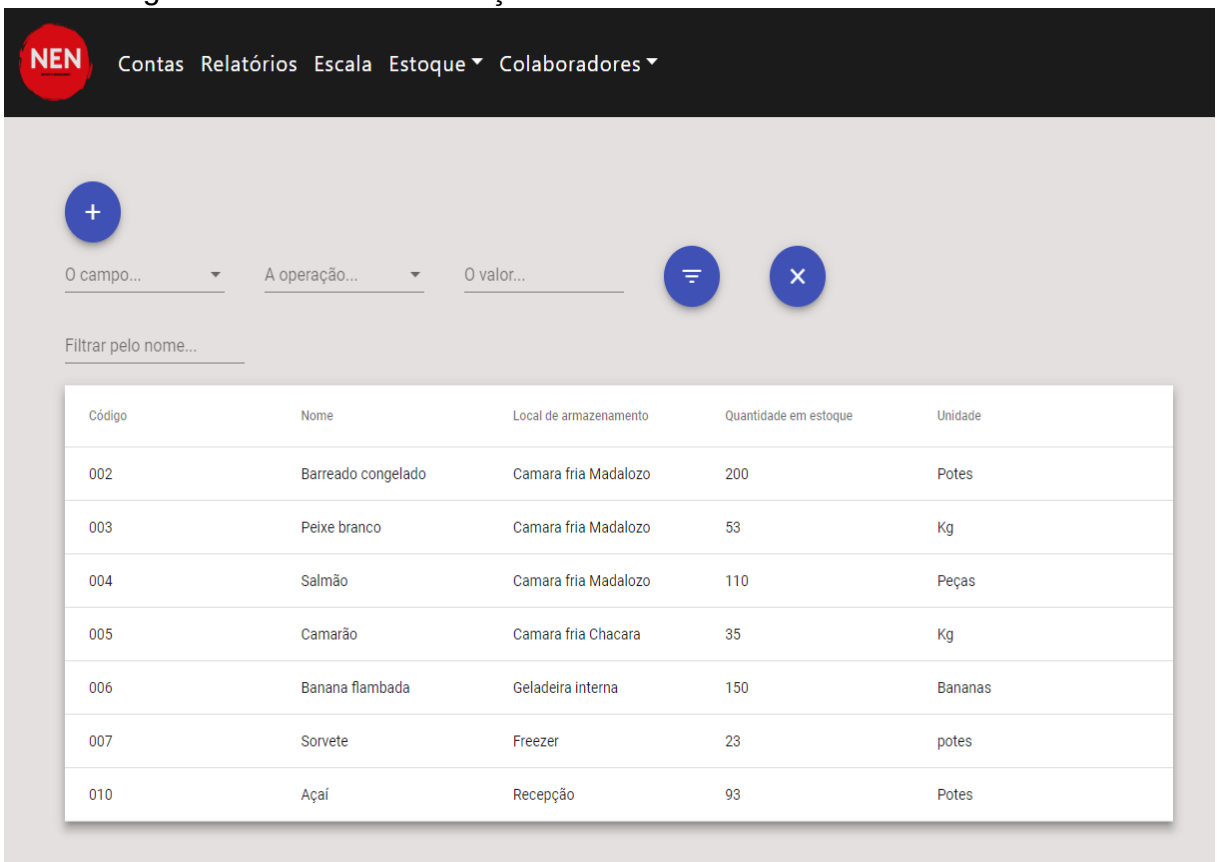
Continuando a apresentação, a Figura 21 esboça a tela onde é possível organizar a escala dos funcionários através de um arrasta e solta (*drag and drop)*. Já a Figura 22 mostra a tela de manejo do estoque, onde o usuário é capaz de visualizar todo o estoque, filtrar, adicionar e remover itens. A Figura 23 apresenta a tela de detalhes do produto, acessada através de um clique em um dos itens na Figura 22. Através de um clique no botão azul com o ícone de lápis em seus interior, na Figura 23, é possível acessar a tela de modificação ou edição de produto, ilustrada pela Figura 24. Ainda relacionado aos produtos, na Figura 25 é a área onde o usuário cadastra novos produtos ao banco de dados do estoque. Por último, mas não menos importante, a tela de cadastro de produto, ilustrada pela Figura 26.

Figura 21 - ADMINISTRAÇÃO ESCALA



FONTE: O autor, 2018

Figura 22 - ADIMINISTRAÇÃO ADICIONAR/REMOVER PRODUTOS

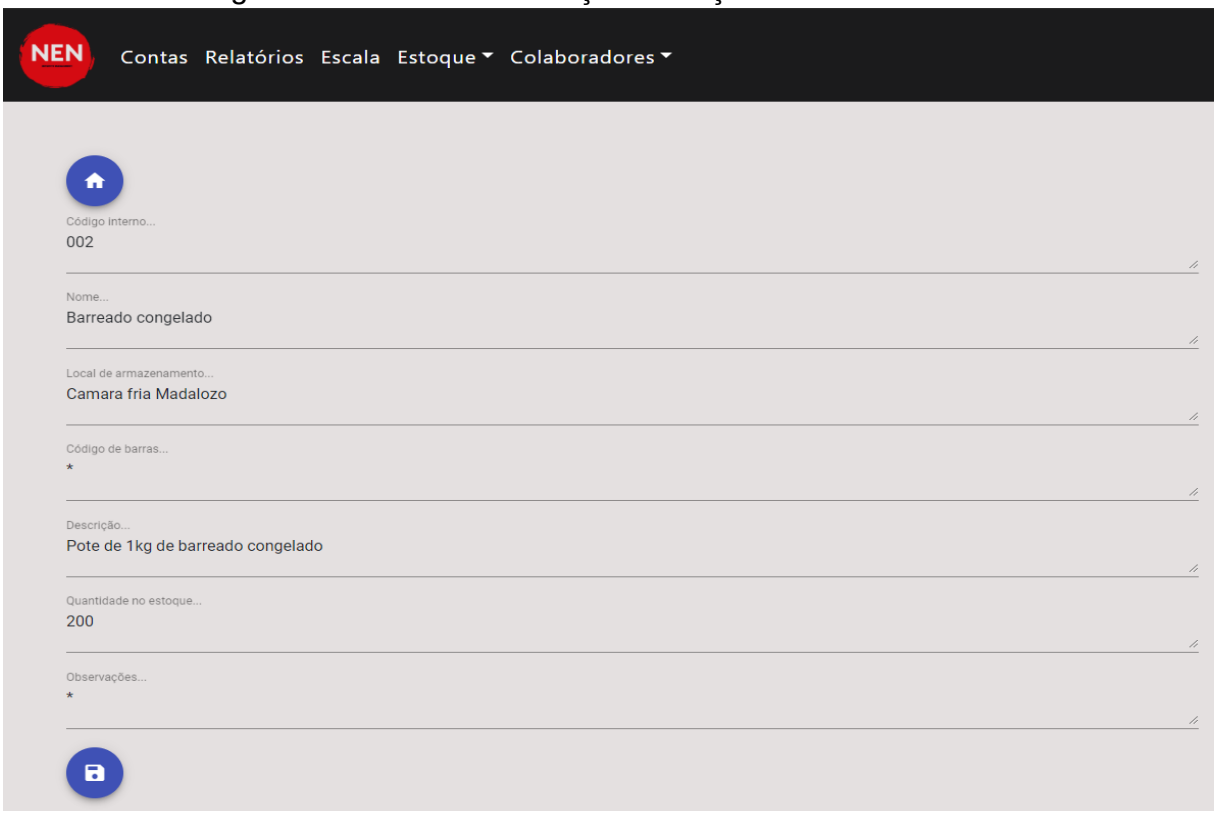


FONTE: O autor, 2018

Figura 23 - ADIMINISTRAÇÃO DETALHES DO PRODUTO

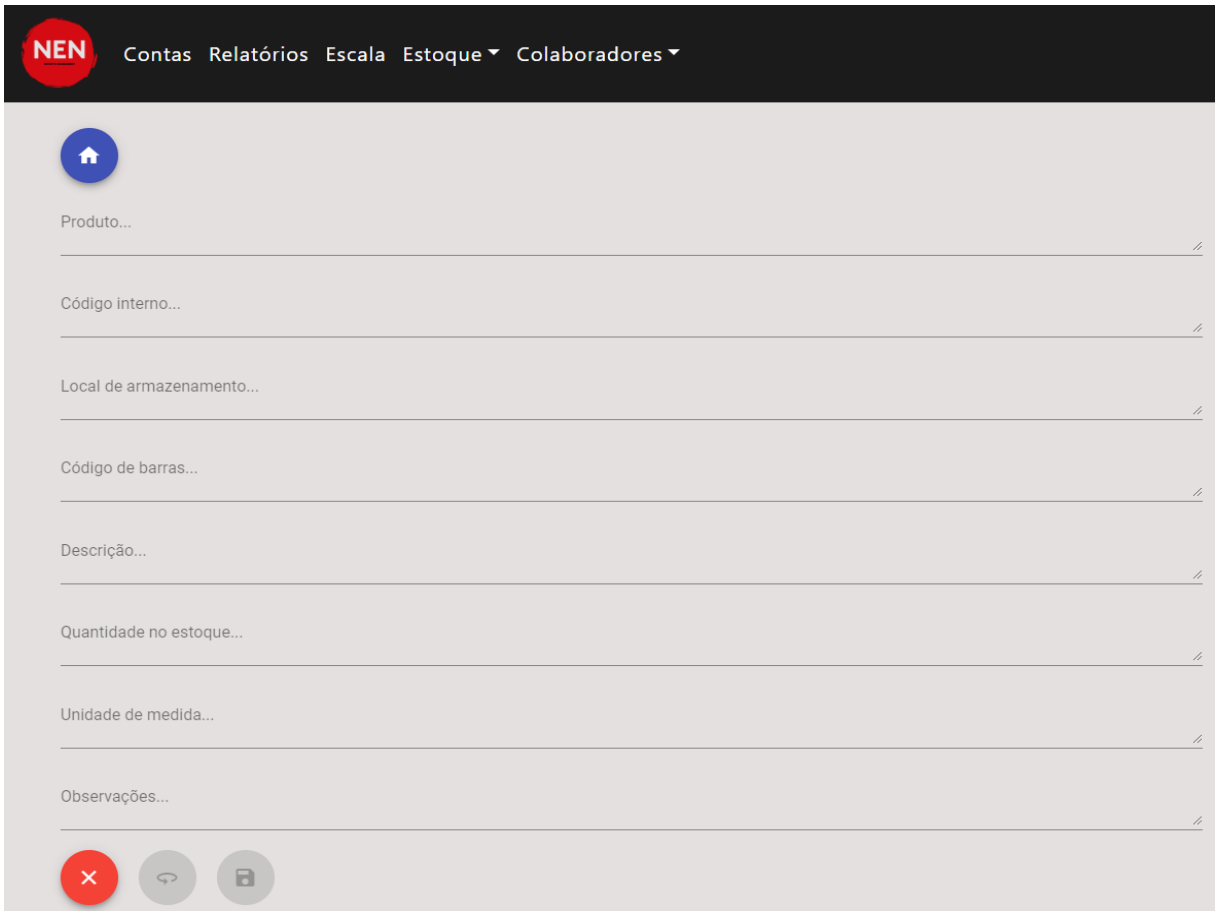
FONTE: O autor, 2018

Figura 24 - ADIMINISTRAÇÃO EDIÇÃO DE PRODUTO



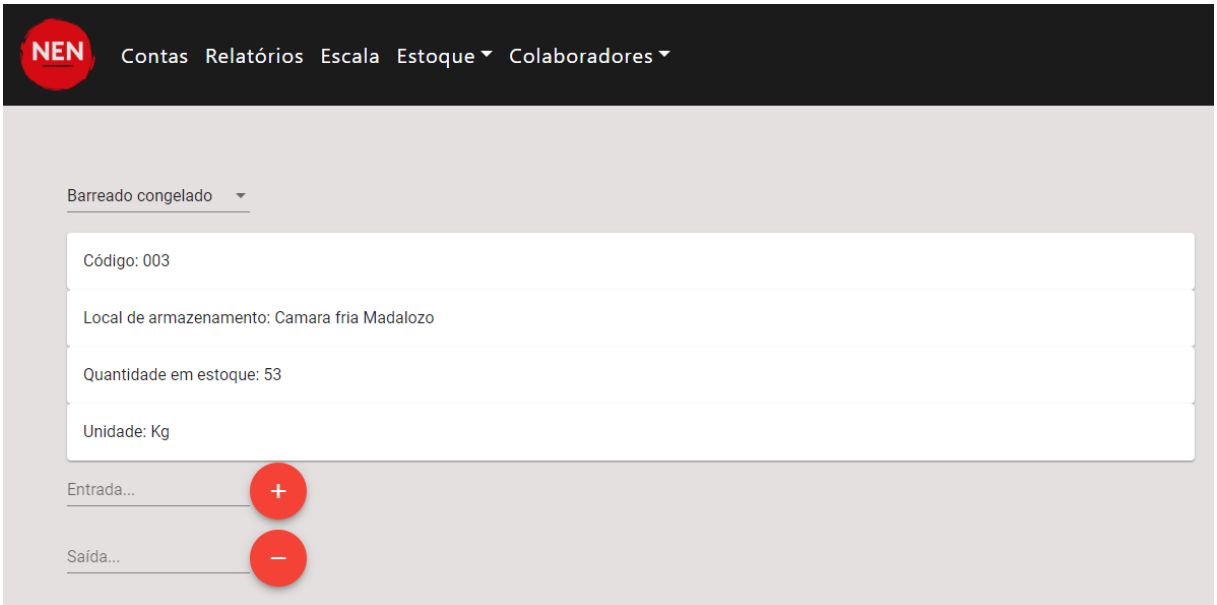
FONTE: O autor, 2018

Figura 25 - ADIMINISTRAÇÃO CADASTRO DE PRODUTO



FONTE: O autor, 2018

Figura 26 - ADIMINISTRAÇÃO ATUALIZAR ESTOQUE



FONTE: O autor, 2018

Uma funcionalidade que foi implementada e que não foi prevista nos requisitos do projeto é a autorização e a autenticação de usuários. O próprio Firebase possui módulos nativos no Angular para a construção desta proteção de uso do software. A Figura 27 mostra a estrutura dos usuários no banco de dados, onde os campos começados com “is” são referentes aos possíveis níveis de acesso de cada usuário, ou seja, caso o campo “isAdmin” receba o valor “*true*”, quando este usuário estiver conectado, ele terá acesso apenas a tela de administração, e assim por diante. Caso o usuário não possua acesso, ele será redirecionado para a tela inicial.

Figura 27 - ESTRUTURA DE ACESSO DA APLICAÇÃO


```
{
"usuarios" : {
"e9Cpt4Ua81PsJGNDm4z8dzESZdH2" : {
"email" : "nen.madalozo@gmail.com",
"isAdmin" : true,
"isAtendimento" : true,
"isCaixa" : true,
"isRecepcao" : true,
"name" : "Nen Madalozo"
},
"yvlHU4R105XUI3BJmB4R6r9LFGc2" : {
"email" : "felipesjmendes@gmail.com",
"isRecepcao" : true,
"isCaixa" : true ",
"name" : "Felipe Mendes"
}
}
}
```


FONTE: O autor, 2018

6. CONCLUSÃO

No início do projeto, os autores foram questionados quanto a motivação para a escolha do tema. A motivação é o começo da criação de um produto, um sistema que será comercializado e aplicado no restaurante Madalozo, servindo como ponta pé para a criação de uma empresa de desenvolvimento, almejada pelos autores.

Na visão da equipe, a formação para este campo do mercado de trabalho não era o suficiente para suprir as necessidades de conhecimento em banco de dados e desenvolvimento web, tornando o TCC uma oportunidade de forçar os autores a se aprimorarem no assunto e resolverem o problema.

Durante o desenvolvimento do projeto foram encontradas diversas dificuldades por causa da falta de experiência e de conhecimentos técnicos dos integrantes da equipe. Foi investido muito tempo em estudos e pesquisa para entender quais são as melhores tecnologias para cada necessidade e como aplicá-las para o projeto, o que acabou reduzindo o tempo disponível para o desenvolvimento do software.

Os resultados obtidos para os componentes finalizados foram extremamente satisfatórios. O layout agradou tanto os desenvolvedores quanto os clientes. Todas as funcionalidades coletadas no levantamento de requisitos, para estes componentes completados, foram atendidas e a interação com o banco de dados foi abordada da maneira muito eficiente, a ponto que a versão grátis (com número de transações limitadas) possa ser usada pelo cliente na atual versão do projeto.

Este trabalho teve um papel muito importante para a formação dos autores, visto que sem este projeto, suas experiências como desenvolvedores seria mínima. Agora é possível entender como fazer o levantamento de requisitos de um projeto, o design do produto, a estimativa do tempo necessário para desenvolver, entre outros.

7. REFERÊNCIAS

Angular. *Angular.* s.d. https://angular.io/features (acesso em 03 de setembro de 2018).

Bell, Donald. *IBM.* 19 de Dezembro de 2016. https://www.ibm.com/developerworks/br/rational/library/content/RationalEdge/sep04/bell/index.html (acesso em 04 de 09 de 2018).

Ben-Gan, Itzik. *T-SQL Fundamentals.* Sebastopol, California: SolidQ, 2012.

Blog, VisutalStudo. 29 de abril de 2015. https://blogs.msdn.microsoft.com/visualstudio/2015/04/29/build-2015-news-visual-studio-code-visual-studio-2015-rc-team-foundation-server-2015-rc-visual-studio-2013-update-5/ (acesso em 01 de setembro de 2018).

Booch, James Rumbaugh Ivar Jacobson Grady. *The Unified Modeling Language Reference Manual.* Boston: Pearson Education, Inc, 2004.

COLLA, ANDRÉ LUIS. “DESENVOLVIMENTO DE SISTEMA DE CONTROLE DE ESTOQUE E.” Trabalho de conclusão de curso, PATO BRANCO, 2013.

Denning, Steve. *Forbes.* Forbes. 8 de Setembro de 2016. https://www.forbes.com/sites/stevedenning/2016/09/08/explaining-agile/#2675e216301b (acesso em 16 de 08 de 2018).

Fenton, Steve. *Pro TypeScript.* Apress, 2014.

Fowler, Adam. *NoSQL for DUMMIES.* Hoboken: John Wiley & Sons, Inc., 2015.

Gavigan, Dave. *Medium.* 3 de abril de 2018. https://medium.com/the-startup-lab-blog/the-history-of-angular-3e36f7e828c7 (acesso em 3 de setembro de 2018).

Google. *Firebase.* s.d. https://firebase.google.com/products/ (acesso em 6 de setembro de 2018).

GROFFE, RENATO. *iMasters.* 30 de Setembro de 2016. https://imasters.com.br/banco-de-dados/bancos-de-dados-nosql-uma-visao-geral (acesso em 28 de agosto de 2018).

Haverbeke, Marijn. *Eloquent Javascript.* 2011.

Ken Schwaber, Jeff Sutherland. *Guia do Scrum.* 2013.

Luis Rei, Sara Carvalho, Marco Alves, João Brito. “A Look at Dynamic Languages.” Porto, Portugal, 30 de Novembro de 2007.

Micreiros. *Micreiros.* s.d. http://micreiros.com/tipos-de-bancos-de-dados-nosql (acesso em 26 de agosto de 2018).

Node.js. “https://nodejs.org/en/.” s.d. (acesso em 22 de 08 de 2018).

Organization, UML. *Multiplicity.* s.d. https://www.uml-diagrams.org/multiplicity.html (acesso em 17 de 09 de 2018).

—. *Sequence Diagrams.* s.d. https://www.uml-diagrams.org/sequence-diagrams.html#execution (acesso em 17 de 09 de 2018).

Patel, Priyesh. “What exactly is Node.js?” *freeCodeCamp.* 18 de 04 de 2018. https://medium.freecodecamp.org/what-exactly-is-node-js-ae36e97449f5 (acesso em 20 de 11 de 2018).

Point, Team @ Tutorials. *Tutorials Point.* s.d. https://www.tutorialspoint.com/software_engineering/software_requirements.htm (acesso em 26 de 08 de 2018).

Rauch, Guillermo. *SMASHING Node.js.* Reino Unido: SMASHING MAGAZINE, 2012.

Silva, Daisy Eliana dos Santos, Ingredy Thaís de Souza, e Talita Camargo. “METODOLOGIAS ÁGEIS PARA O DESENVOLVIMENTO DE SOFTWARE:.” (UnG) 2 (2013).

Souza, Diogo Rodrigues de. “IMPLANTAÇÃO DA METODOLOGIA ÁGIL SCRUM EM.” Araranguá, 2014.

“SQL Server Documentation.” *Microsoft Docs.* 2016. https://opdhsblobprod02.blob.core.windows.net/contents/264cf0df66eb4cbf86eac6677c504c4d/48de9841ff8e173a3518a1abf658ccc4?sv=2015-04-05&sr=b&sig=eDJNaamaKKUp0XJC7e7k49I7RZEjdWz%2BuQgd7zNVTKw%3D&st=2018-11-25T14%3A00%3A45Z&se=2018-11-26T14%3A10%3A45Z&sp=r (acesso em 28 de 08 de 2018).

Sutherland, Dr. Jeff. “AGILE DEVELOPMENT: LESSONS LEARNED FROM THE FIRST SCRUM.” *Cutter Agile Project Management Advisory Service*, 2004.

VIEIRA, RICARDO. “Uso da UML na Especificação do Sistema de Informação da Área de Infra-Estrutura.” Monografia (Especialização), Porto Alegre, 2003.

Weisfeld, Matt. *The Object-Oriented Thought Process 3ª ed.* USA: Pearson Education, 2009.