

COMANDA DIGITAL DE BARES E RESTAURANTES

Guilherme Adriano Schafer, Dalton Solano dos Reis– Orientador

Curso de Bacharel em Ciência da Computação
Departamento de Sistemas e Computação
Universidade Regional de Blumenau (FURB) – Blumenau, SC – Brasil
gschafer@furb.br, dalton@furb.br

Resumo: Este artigo apresenta o desenvolvimento de uma aplicação web para restaurantes e bares que visa auxiliar os clientes na experiência dentro do estabelecimento. O aplicativo foi desenvolvido em C# com Visual Studio 2019 e React Native com Visual Studio Code e bibliotecas React Native. O objetivo principal é facilitar o cliente na visualização do cardápio e na solicitação do pedido e pagamento online, utilizando somente o dispositivo móvel do próprio cliente. Através do aplicativo a cozinha e garçons do estabelecimento, podem acompanhar as solicitações dos pedidos para executarem suas respectivas ações, por exemplo, realizar os pratos ou entregar os produtos solicitados.

Palavras-chave: React Native. Comanda Digital. Restaurantes. Bares. Pagamento online.

1 INTRODUÇÃO

O avanço da tecnologia nas mais diversas áreas está cada vez mais rápido. Entretanto ainda existe uma área pouco valorizada, a experiência de um cliente dentro de um estabelecimento, como bares e restaurantes, onde ainda a experiência é de forma “antiga”. Na maioria dos estabelecimentos o cliente precisa do atendimento de um garçom para solicitar sua comida e bebida e isso pode acarretar um degaste, caso o estabelecimento esteja cheio, gerando uma demora no atendimento, ou até na preparação do pedido.

“Acredito que muitos já se depararam com uma cena constrangedora ao serem atendidos por alguém mal-humorado ou com a ‘cara marrada’” (PUDENCE, 2018). Muitos restaurantes estão perdendo clientes devido ao mal atendimento dos funcionários e uma solução para os donos de restaurantes seria investir cada vez mais nas tecnologias atuais, principalmente nos aplicativos móveis.

De acordo com Magalhães (2018, p. 1),

Antigamente, a tecnologia era considerada um luxo, exclusivo para poucos, mas, com a busca por mais praticidade e agilidade, as modernidades tecnológicas estão disponíveis para todos e são essenciais na rotina de cada um. Por isso, colocar um restaurante na era digital é fundamental para satisfazer clientes exigentes.

“Cada vez mais as pessoas usam aplicativos para tarefas do cotidiano – e reservar mesas em restaurantes é uma delas” (FERNANDA, 2018, p. 1). Existem muitos aplicativos que facilitam a rotina das pessoas, desde a realização de compras, transportes, comunicação e até no segmento de alimentação para entrega na residência do usuário. A maioria instiga o usuário a ficar dentro da sua residência e poucos incentivam o usuário a ir até o estabelecimento.

Diante do que foi exposto, este trabalho desenvolveu um aplicativo móvel para realizar o controle de comandas de um bar ou restaurante. O cliente pode visualizar o cardápio do estabelecimento, permitindo fazer os pedidos de bebidas ou comidas através do aplicativo sem a utilização de um garçom e realizar o pagamento utilizando cartões de crédito.

Com o aplicativo o atendimento ao cliente será no momento de chegada do cliente ao estabelecimento, onde através do QR Code, o cliente pode abrir a aplicação e realizar o pedido. Haverá uma diminuição da fila no caixa, pois o cliente poderá pagar o pedido pelo aplicativo. Além de o cliente acompanhar o status do seu pedido, não se aborrecendo com uma demora sem respostas de como está o seu pedido no momento.

2 FUNDAMENTAÇÃO TEÓRICA

Nessa seção são apresentados os principais assuntos pertinentes à aplicação, transformação digital em *foodservice*, compras online, React Native e os trabalhos correlatos.

2.1 TRANSFORMAÇÃO DIGITAL EM FOODSERVICE

No mundo onde cada vez mais estamos conectados, os estabelecimentos de alimentação precisam se adaptar à nova era, e a transformação digital pode auxiliar o estabelecimento a se manter no mercado ou até mesmo a entrar no mercado de negócios. Scuadra (2018) destaca que, segundo a organização SiriusDecisions, 67% da jornada do comprador é realizado de modo digital. Para isso significa que mesmo antes de chegar ao estabelecimento, o cliente já

pesquisou sobre o restaurante, leu avaliações, procurou os pratos servidos e muitas das vezes já possui uma opinião sobre os produtos e serviços disponibilizados.

A transformação digital não está somente ligada ao marketing digital nas redes sociais. Segundo Scudra (2018), existem alguns assuntos a serem abordados para conseguir efetivamente realizar uma transformação digital, como, se relacionar com os clientes, automatizar a gestão, alimentos de qualidade e uma implantação de um cardápio digital.

A presença digital é um dos pontos mais importantes da transformação digital, por exemplo, transformar a marca de um pequeno restaurante em uma marca digital pode atrair mais clientes das gerações mais novas, ou seja, as gerações que utilizam a tecnologia para tudo e buscar novos estabelecimentos para experimentar com certeza está dentro das características dessa geração.

2.2 COMPRAS ONLINE

Segundo Schnaider (2020), as compras em lojas virtuais em *e-commerce* brasileiro registraram um crescimento de 47% no primeiro trimestre, maior alta em 20 anos. Cada vez mais a tecnologia está sendo inserida dentro de várias áreas de negócios, onde a facilidade e rapidez que a tecnologia proporciona chama a atenção ainda mais dos clientes.

Mohsin (2020) destaca uma pesquisa realizada pelo Paypal em 2019, onde descreve que 80% dos brasileiros usa seus smartphones para comprar on-line. Além de construir uma aplicação a loja/estabelecimento também precisa, otimizar o processo de *checkout* do produto, elaborar um aplicativo exclusivo, construir um canal de atendimento, investir em campanhas publicitárias e construir um *newsletter* compatível com o negócio.

Ainda segundo Mohsin (2020), sete a cada dez compras realizadas pela internet são pagas com cartão de crédito. Dessa maneira o pagamento on-line auxilia em mais uma opção para os clientes, aumentando o público a ser alcançado pelo estabelecimento, não sendo necessário o cliente sair de casa com seus cartões e somente com o próprio dispositivo móvel agrada as pessoas cada dia mais.

Na área da alimentação, o maior exemplo de uso da tecnologia para compras online seria o iFood, o qual possui um aplicativo para solicitação de produtos de diversos restaurantes para entrega em casa.

“Gastos com delivery crescem mais de 94% na pandemia”, (PATRÍCIA BÜLL, 2020). Isso demonstra que mesmo em meio a crise mundial os estabelecimentos que disponibilizam uma forma de compra online para o cliente, conseguiram diminuir o prejuízo devido as paralisações.

2.3 REACT NATIVE

O React Native é um *framework* para desenvolvimento de aplicações multiplataformas, criado em 2015, pela equipe do Facebook. Funciona como biblioteca e facilitador de recursos JavaScript que possibilita o desenvolvimento de aplicações para todas as plataformas.

Os principais recursos do React Native são: geração de código nativo, desenvolvimento por componentes, e utilização de plugins. Geração do código nativo seria a possibilidade de o desenvolvedor escrever um único código, no qual será utilizado em todas as plataformas. O *framework* constrói as aplicações com foco e estrutura no layout das interfaces, trabalhando com componentes e subcomponentes. A utilização de plugins prontos desenvolvidos por outros desenvolvedores também é possível. Na aplicação descrita nesse artigo foi utilizado um componente de validação de campos nas interfaces.

Algumas vantagens do React Native são: torna a experiência do usuário mais fluida, por ter o foco e estrutura no layout, e é extremamente ágil e mais rápido, otimizando assim os erros de interface. O *framework* possibilita realizar integrações com funções nativas dos dispositivos, seja Android ou iOS, por exemplo, utilizar a câmera e outros aplicativos instalados no dispositivo. O desempenho é muito similar com aplicações construídas com código nativos, gerando assim uma melhor performance para o usuário.

Várias empresas já utilizam o *framework*, além do próprio Facebook. Empresas como, Airbnb, Uber, Tesla e Wix possuem seus aplicativos mobiles desenvolvidos em React Native.

2.4 TRABALHOS CORRELATOS

Nessa seção são apresentados três trabalhos correlatos que possuem características semelhantes a proposta deste trabalho. O Quadro 1 descreve o trabalho de Araujo (2017), que desenvolveu uma aplicação multiplataforma para localização e recomendação de restaurantes. No Quadro 2 o protótipo de Koglin Junior (2018) detalha o desenvolvimento de um aplicativo para pagamento de estacionamento. No Quadro 3 o trabalho de Borba Junior (2015) detalha o aplicativo para controle e agendamento de medicamentos.

Quadro 1 – FIND FOOD: Aplicativo para localização e recomendação de restaurantes

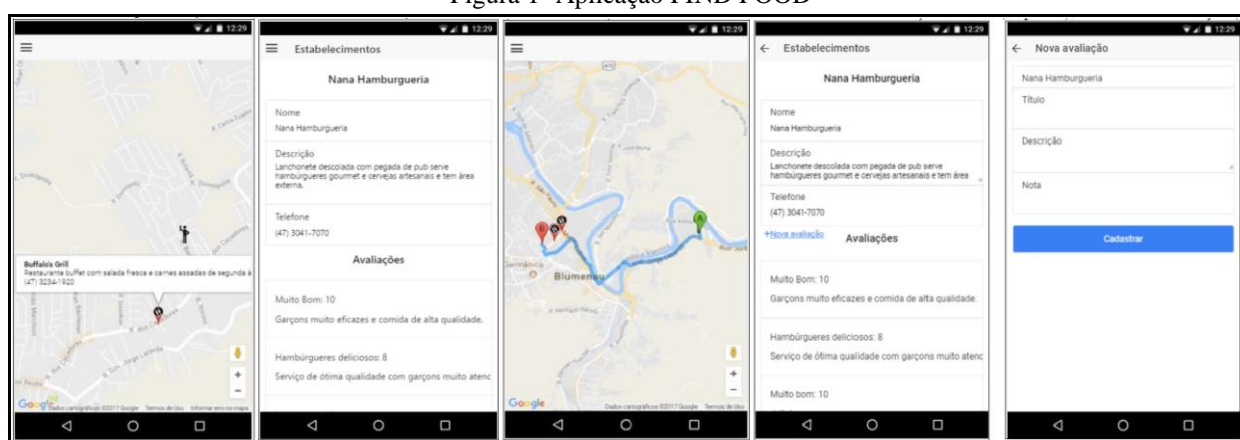
| | |
|------------|---------------|
| Referência | Araujo (2017) |
|------------|---------------|

| | |
|--------------------------------|---|
| Objetivos | Criar uma aplicação móvel para pagamento localização e recomendação de restaurantes próximos. |
| Principais funcionalidades | Conforme Figura 1, permite ao usuário localizar os restaurantes mais próximos da sua localização atual, realizar pesquisas dos restaurantes mais bem avaliados por outros usuários, realizar as avaliações dos vários restaurantes e ainda traçar a rota via Global Position System (GPS) para chegar até o restaurante desejado. |
| Ferramentas de desenvolvimento | Framework Ionic, Google Maps (por meio de Application Program Interface – API), Global Position System (GPS). |
| Resultados e conclusões | O trabalho desenvolvido atingiu seus objetivos, que eram apresentar opções de restaurantes próximos à localização atual do usuário e permitir que os usuários avaliem os restaurantes a fim de auxiliar outros usuários na escolha de bons restaurantes. |

Fonte: elaborado pelo autor.

O aplicativo (Figura 1) permite encontrar os restaurantes mais bem avaliados, nas aproximações dos usuários permitindo traçar uma rota para se locomover até o restaurante desejado. Pode-se ainda fazer avaliações positivas ou negativas em relação ao restaurante, disponibilizando essas informações para outros usuários.

Figura 1- Aplicação FIND FOOD



Fonte: Araujo (2017).

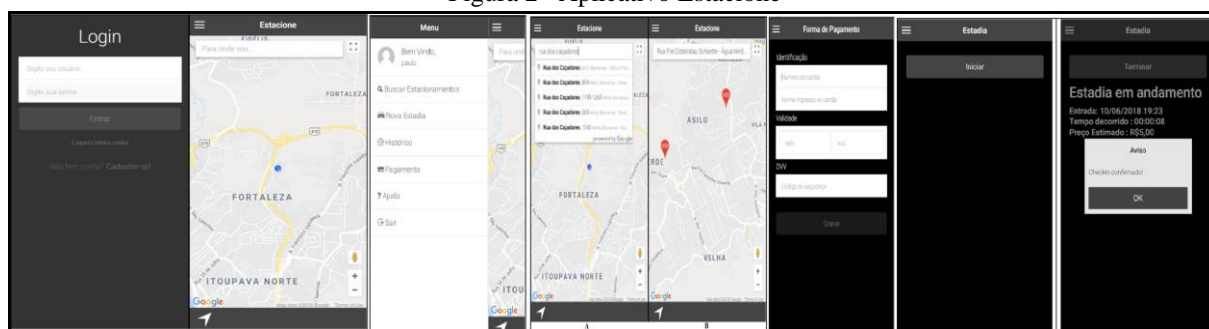
Quadro 2 – ESTACIONE: Prototipo de aplicativo para pagamento móvel de estacionamento

| | |
|--------------------------------|--|
| Referência | Koglin Junior (2018) |
| Objetivos | Criar um protótipo de aplicação móvel para pagamento de estacionamento. |
| Principais funcionalidades | Buscar estacionamentos próximos da localização e realizar o pagamento através de cartões de crédito. |
| Ferramentas de desenvolvimento | Framework Ionic, Java, banco de dados Postgress, Node JS, hospedado no Heroku. |
| Resultados e conclusões | O aplicativo alcançou o objetivo proposto, apresentando os estacionamentos credenciados em um mapa. |

Fonte: elaborado pelo autor.

Conforme visualizado na Figura 2, o usuário realiza o *login* na aplicação e visualiza a localização atual. Através do menu lateral possibilita buscar um estacionamento, incluindo o nome de uma rua para visualizar os estacionamentos no mapa para realizar a estadia e após realizar a estadia o pagamento pode ser feito através do cartão.

Figura 2– Aplicativo Estacione



Fonte: Koglin Junior (2018).

Segundo Koglin Junior (2018), o aplicativo alcançou o objetivo proposto, disponibilizando um mapa com as informações dos estacionamento credenciados, visualizando o tempo de estadia e o pagamento tudo através do seu próprio dispositivo móvel.

Quadro 3 – Aplicativo Mobile para controle e agendamento de medicamentos

| | |
|--------------------------------|--|
| Referência | Borba Junior (2015) |
| Objetivos | Criar um aplicativo para uma pessoa ou responsável receber um aviso da necessidade de tomar medicamentos. |
| Principais funcionalidades | Cadastrar prescrições diárias a pacientes e receber notificações no horário exato de cada um dos remédios. |
| Ferramentas de desenvolvimento | HTML, CSS, Javascript, JQuery e Bootstrap e framework Apache Cordova. |
| Resultados e conclusões | O aplicativo atendeu o principal objetivo, que era auxiliar um usuário a tomar os remédios na hora exata. |

Fonte: elaborado pelo autor.

O aplicativo auxiliou o usuário a controlar o seu consumo de remédios para que não haja nenhum esquecimento ou erro na repetição de remédios. O usuário pode cadastrar todos os remédios ou vários pacientes caso o responsável controle o remédio de mais de 1 uma pessoa, possibilitando analisar as prescrições diárias de cada paciente, recebendo notificações no horário exato de cada um dos remédios.

Conforme a Figura 3 primeiramente o usuário deve selecionar uma das opções: Remédios, Pacientes e Prescrições. Em cada uma das opções o usuário tem a possibilidade de cadastrar, editar, visualizar e excluir.

Figura 3 – Aplicativo mobile para controle e agendamento de medicamentos



Fonte: Borba Junior (2015).

3 DESCRIÇÃO

Esta seção apresenta os detalhes de especificação, implementação e operacionalidade da aplicação. A aplicação possui quatro tipos de usuários diferentes, e dispõem de comportamentos e telas diferentes para cada tipo. Para tanto, são apresentadas em dois blocos. O primeiro demonstra como foi realizado a especificação e o segundo a implementação e a operacionalidade.

3.1 ESPECIFICAÇÃO

O objetivo da aplicação é melhorar a experiência do cliente nos bares e restaurantes, proporcionando uma rapidez nas tarefas mais comuns de um bar e restaurante, por exemplo, agilizar a solicitação das comidas e bebidas, o cliente controlar a própria conta, facilitar e agilizar o pagamento dos pedidos solicitados e acompanhar o status do seu pedido. (APÊNDICE E)

Por tanto, os Requisitos Funcionais (RF) e Requisitos Não Funcionais (RNF) são apresentados no Quadro 4 e Quadro 5 respectivamente, e foram utilizados como base para o desenvolvimento dessa aplicação. Os requisitos foram especificados antes do início das implementações e serviram para guiar o seu desenvolvimento.

Quadro 4- Requisitos Funcionais

| | |
|-------|---|
| RF001 | Permitir o usuário se registrar |
| RF002 | Cadastrar comidas e bebidas |
| RF003 | Visualizar o cardápio de comidas e bebidas |
| RF004 | Realizar pedidos de comidas e bebidas |
| RF005 | Acompanhar o ciclo de vida dos pedidos realizados |
| RF006 | Realizar o pagamento dos pedidos via aplicação |
| RF007 | Atualizar o status do pedido pela cozinha |
| RF008 | Atualizar o status do pedido pelo estabelecimento e garçons |

Fonte: elaborado pelo autor.

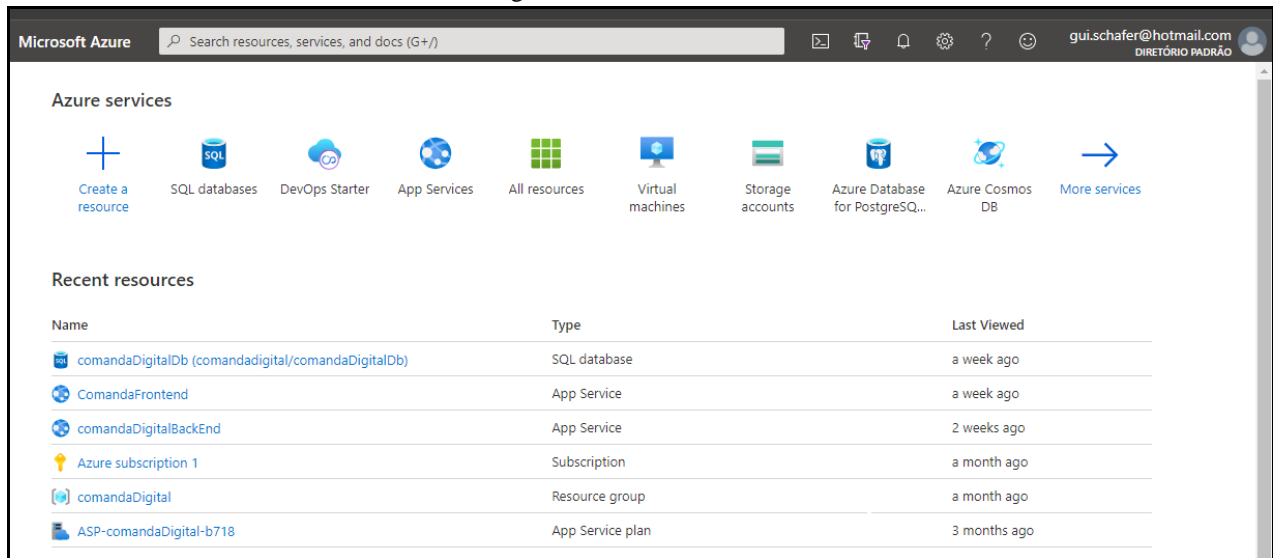
Quadro 5- Requisitos Não Funcionais

| | |
|--------|---|
| RNF001 | Acessar aplicação via QR CODE |
| RNF002 | Possibilitar acesso por qualquer dispositivo móvel |
| RNF003 | Implementar camada do cliente na linguagem React Native |
| RNF004 | Implementar camada do servidor na linguagem .NET CORE C# |
| RNF005 | Utilizar uma base de dados SQL Server |
| RNF006 | Permitir o usuário realizar o <i>login</i> com o cadastro |

Fonte: elaborado pelo autor.

Todas as camadas da aplicação (cliente, servidor e base de dados) estão hospedadas na nuvem da Microsoft Azure (Conforme demonstra a Figura 4). O Azure proporciona facilidades no momento de subir a aplicação para a nuvem, por possuir uma variedade de servidores, inclusão de certificados HTTPS, registros na internet, integração fácil com produtos da Microsoft, por exemplo, a camada de servidor desenvolvida em .NET CORE C#.

Figura 4 – Dashboard Azure

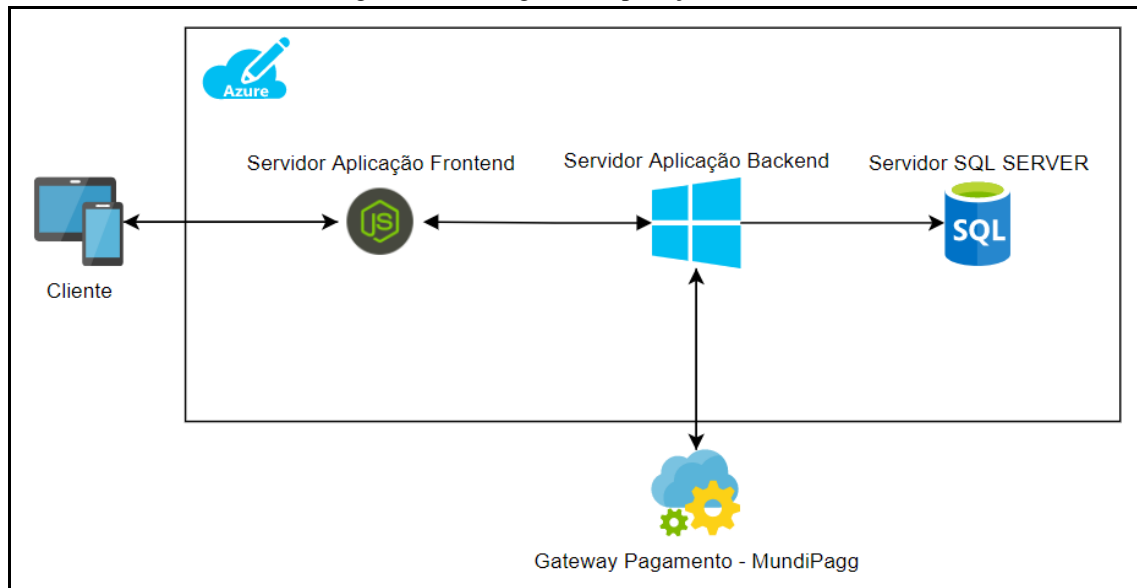


Fonte: elaborado pelo autor.

O código fonte da aplicação ficou hospedado no Github, para uma melhor integração com o Microsoft Azure, por serem os dois produtos Microsoft. O Azure já possui configurações pré-definidas no momento de hospedar a aplicação, assim toda atualização realizada no Github automaticamente realiza uma nova versão para cada servidor, realizados algumas etapas configuradas, como o *build* da aplicação para garantir que não possui nenhum erro.

A camada do cliente fica hospedado em um servidor NodeJS por ter sido desenvolvido em React Native. A camada do servidor fica em um servidor Windows, por se tratar de uma aplicação .NET CORE. O Azure disponibiliza uma configuração padrão de hospedagem de aplicações .NET. O cliente acessa o *frontend* através de uma URL pré-definida no momento da criação do servidor, para que seja realizado a comunicação com a aplicação *backend*. E através da lógica incluída no *backend* os dados são armazenados no banco de dados SQL Server. (Figura 5)

Figura 5 – Visão geral da aplicação na nuvem



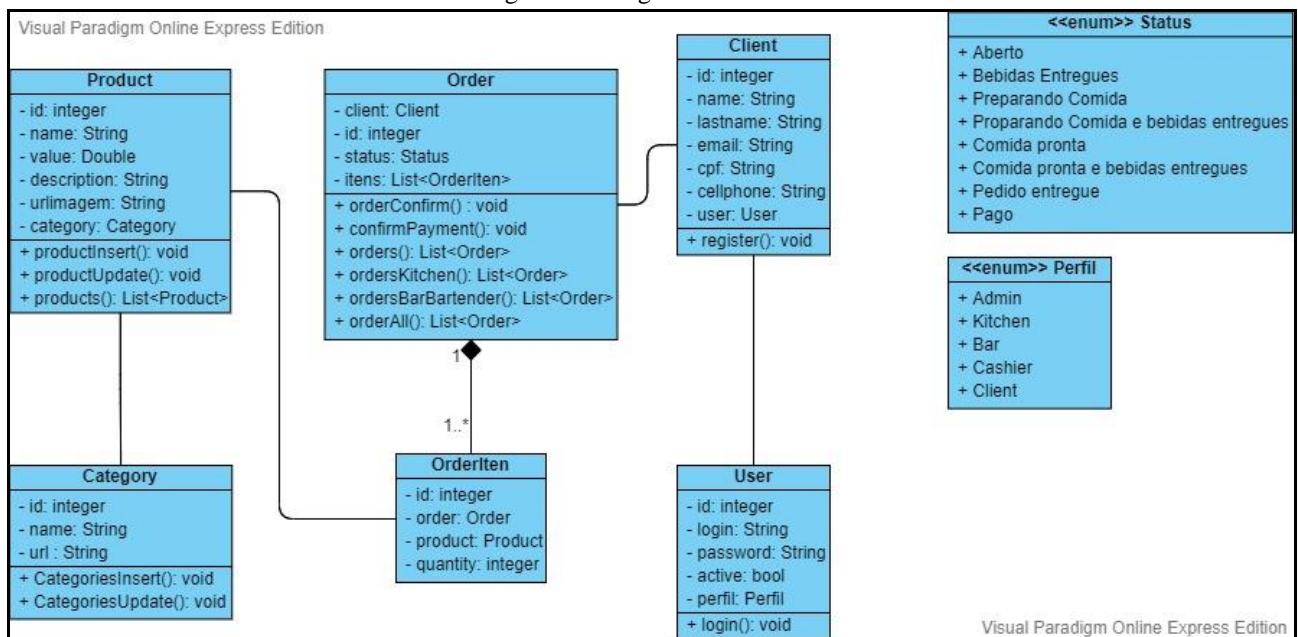
Fonte: elaborado pelo autor.

Para realizar os pagamentos dos pedidos através do aplicativo foi utilizado o *gateway* de pagamento da MundiPagg. A plataforma da MundiPagg disponibiliza várias *Application Programming Interface* (APIs) para realizar as cobranças em cartões de créditos, agendamento cobranças recorrentes e armazenar dados de cartões e do cliente com segurança. Além de um *dashboard* completo, para verificar e extrair dados de todo o tráfego de dados transmitido para a plataforma.

3.2 IMPLEMENTAÇÃO

A aplicação foi dividida em classes, cada classe representa algum objeto físico ou fictício (Figura 6). As principais classes são: *produto*, *pedidos* e *cliente*. A classe de *produto* representa cada uma das comidas ou bebidas disponibilizadas no cardápio. A classe *pedidos* representa cada solicitação dos produtos disponibilizados no cardápio, seria o pedido realizado pelo um garçom normalmente. As informações do cliente para identificação de cada um e para vincular os pedidos a um cliente, está representado na classe *cliente*.

Figura 6 – Diagrama de classes



Fonte: elaborado pelo autor.

Os enumeradores são usados para controlar o ciclo de vida de um pedido, com isso o cliente consegue acompanhar o status atualizado, vindo da cozinha ou bar. O perfil de cada usuário é controlado pelo enumerador *Perfil*, onde distingue o tipo do cada usuário e os requisitos que serão apresentados aos usuários na camada *frontend*.

Para o cliente acessar a aplicação a camada *frontend* disponibiliza duas interfaces em React Native (Figura 7), acessadas via browser, onde o usuário conseguirá realizar o registro com os dados pessoais e a interface de *login*. Os usuários com perfis que serão utilizados pelo estabelecimento já foram criados no momento da implantação da aplicação na nuvem.

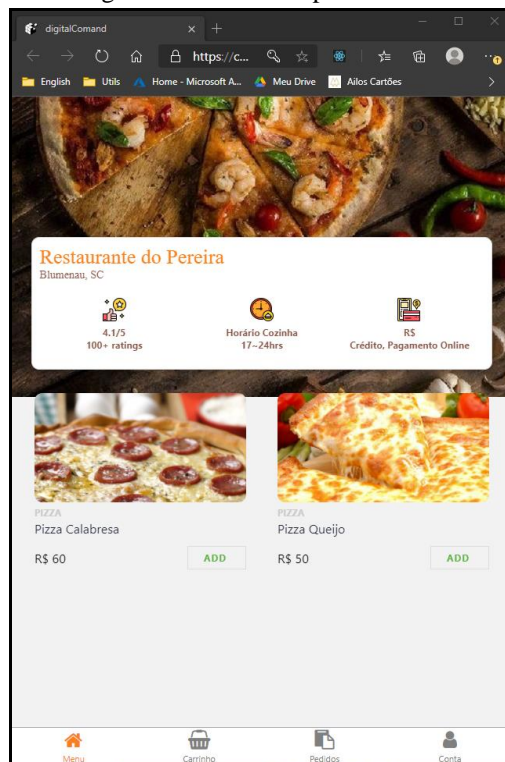
Figura 7 – Telas de Login e Registro

The image displays two side-by-side screenshots of a web application interface, likely for a restaurant named 'Restaurante do Pereira'. The left screenshot shows the login screen with fields for 'Username' and 'Password', and buttons for 'SIGN IN', 'SIGN UP', and a link for 'Forgot Password?'. The right screenshot shows the registration screen with fields for 'Nome', 'Sobrenome', 'Email', 'CPF', 'Celular', 'Password', and 'Confirm Password', and a 'REGISTER' button. Both screens are displayed within a browser window with the address bar showing 'https://coma...'.

Fonte: elaborado pelo autor.

O perfil de cliente possui algumas opções de telas em um menu inferior da aplicação: opções como o Menu/Cardápio, carrinho para acompanhar os itens adicionados e acompanhar o valor do seu pedido, os pedidos já realizados e seus respectivos status, e uma opção da conta para realizar o *logoff* da aplicação. Além das informações básicas do estabelecimento que serão visualizados em tela. (conforme demonstrado na Figura 8 – Acesso do perfil cliente).

Figura 8 – Acesso do perfil cliente



Fonte: elaborado pelo autor.

Para controlar os itens adicionados no carrinho através do botão ADD de cada produto do cardápio, a aplicação *frontend* armazena essas informações no próprio browser do dispositivo através de uma biblioteca *AsyncStorage* (conforme demonstra o Quadro 6 – Método *handleCart*) no qual utiliza o *LocalStorage* do browser. O método *handleCart* chamado através do botão ADD, armazena as informações do produto e a quantidade selecionada, dentro do browser, sem a necessidade de requisição para o servidor *backend*.

Quadro 6 – Método *handleCart*

```
handleCart = (item, qty) => {
  const itemDetails = Object.assign(item, { "Quantity": qty });
  const itemKey = `orderDetails${itemDetails.id}`;

  AsyncStorage.setItem(itemKey, JSON.stringify(itemDetails)).then(
    () => {
      this.getCartItems();
    }
  );
}
```

Fonte: elaborado pelo autor.

O método *getCartItems* (Quadro 7 – Método *getCartItems*) chamado dentro do *handleCart* obtém as informações do produto incluído dentro do *LocalStorage* para somar os valores totais de preço, quantidade de produtos selecionados e remover os produtos que foram zerados. Além de realizar esses cálculos através do método *getCartItemDetails*, é realizada a atualização das variáveis de controle do *frontend* para visualizar a barra de itens adicionados no carrinho através do método *updateCartState*.

Quadro 7 – Método `getCartItems`

```
getCartItems = () => {
  AsyncStorage.getAllKeys().then(orderList => {
    if (orderList.length !== 0) {
      AsyncStorage.multiGet(orderList).then(response => {
        let cartDetails = this.getCartItemDetails(response);
        this.updateCartState(cartDetails.totalQuantity, cartDetails.totalPrice, cartDetails.allItems);
      });
    } else {
      this.setState({
        allItems: []
      });
    }
  });
}
```

Fonte: elaborado pelo autor.

Como pode ser visto no APÊNDICE A, após a chamada do método `updateCartState` a barra de visualização parcial do carrinho aparece próximo do menu inferior. Dessa maneira o cliente realiza seu controle de itens adicionados antes mesmo de visualizar o carrinho por completo.

Para tanto, ao clicar no menu Carrinho a visualização do carrinho por completo será apresentada, com isso os produtos adicionados, o valor e quantidade de cada um e o valor total do pedido serão demonstrados em tela. Além de um botão Confirmar Pedido, onde ao ser clicado será direcionado pela primeira vez em todo o ciclo de vida do carrinho para a camada do servidor, para ser persistido as informações do pedido e ser apresentado o pedido para preparação da cozinha e do bar.

A comunicação com a camada do servidor é realizada através de endpoint *Representational State Transfer* (REST), como pode ser visto no Quadro 8 – Método confirmar pedido, ao confirmar o pedido será consumido o *endpoint* de confirmar o pedido, no qual possui o objetivo de persistir todas as informações do pedido na base de dados. Para identificar qual é o cliente/usuário que está realizando a solicitação do pedido é obtido o usuário através do token passado ao servidor.

Quadro 8 – Método confirmar pedido

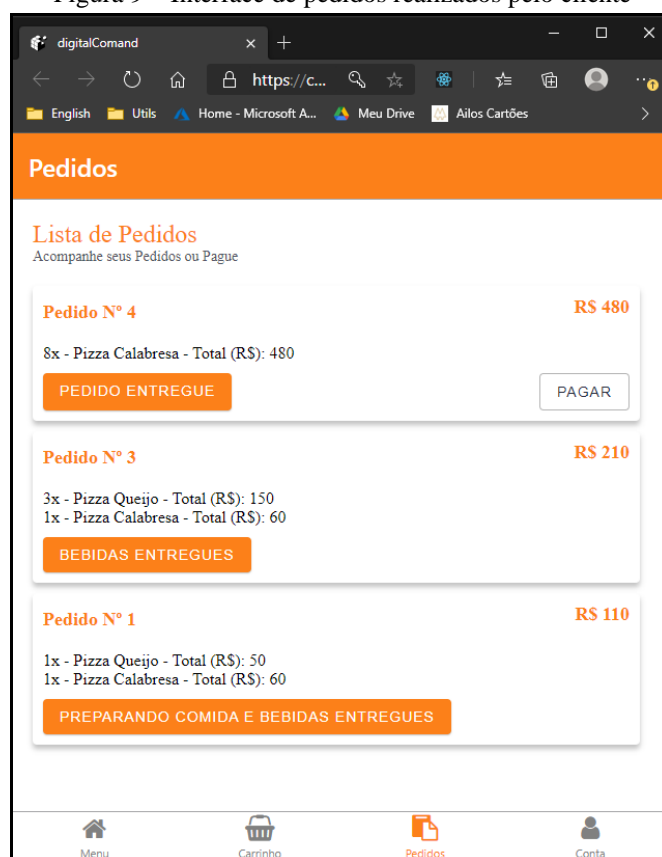
```
[HttpPost]
[Route("orderConfirm")]
[Authorize]
0 references
public IActionResult OrderConfirm([FromBody] List<OrderDto> orderItems)
{
    try
    {
        var userLogin = User.Claims.LastOrDefault(c => c.Type == ClaimTypes.Name).Value;

        orderProvider.ConfirmOrder(orderItems, userLogin);
        return Ok();
    }
    catch (Exception e)
    {
        return BadRequest(e.Message);
    }
}
```

Fonte: elaborado pelo autor.

No menu Pedido, no canto inferior, o cliente pode visualizar todos os pedidos realizados e acompanhar o status de cada um (Figura 9). A opção de pagamento via cartão de crédito será visualizada quando o pedido já foi totalmente entregue.

Figura 9 – Interface de pedidos realizados pelo cliente



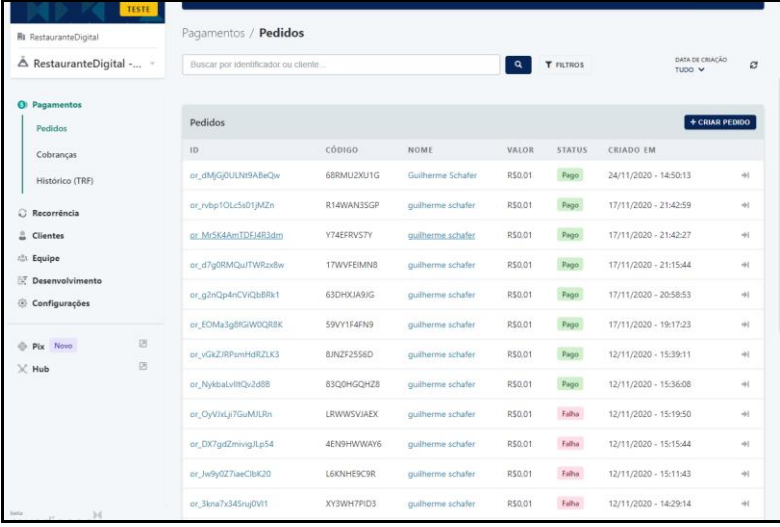
Fonte: elaborado pelo cliente.

Para realizar o pagamento o cliente será direcionado a uma interface para adicionar os dados do cartão. Após adicionar os dados e clicar no botão **Pagar**, será consumido um endpoint `confirmPayment` (APÊNDICE B), no qual executa a comunicação com a MundiPagg, para assim validar com a operadora do cartão e receber uma confirmação do pagamento.

Toda a comunicação com a MundiPagg é através de API REST, utilizando uma autenticação de chaves geradas pela própria plataforma MundiPagg. A plataforma disponibiliza interfaces para visualizar todas as informações armazenadas, por exemplo, todo o histórico de cobranças realizados com o respectivo status de cada cobrança, clientes cadastrados automaticamente ao gerar a cobrança, e possibilidade de tentar novamente gerar a cobrança em caso de alguma falha.

A plataforma MundiPagg contém um ambiente de testes para a simulações de todos as funções disponibilizadas. No caso das cobranças em cartão de crédito utilizado nessa implementação podem usar números de cartões fictícios, no qual simulam todos os status de retorno possíveis em uma cobrança. Como pode ser visto na Figura 10 – Plataforma MundiPagg, algumas cobranças foram realizadas com sucesso e outras com falha em um ambiente de teste (conforme apresentado na parte superior esquerda da Figura 10).

Figura 10 – Plataforma MundiPagg



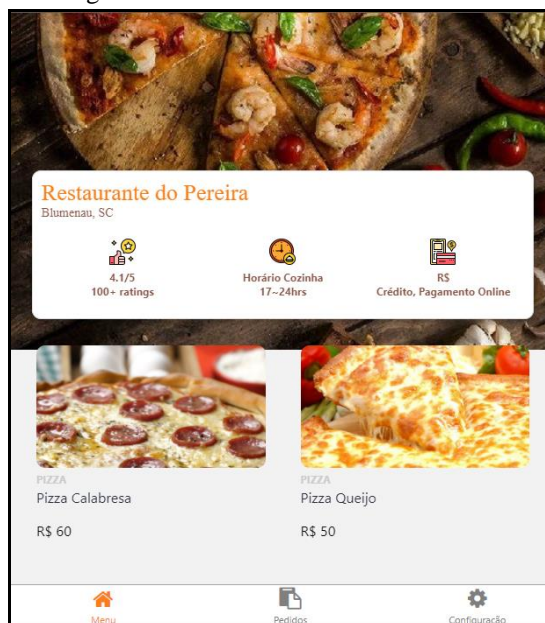
| ID | CÓDIGO | NOME | VALOR | STATUS | CRIADO EM |
|---------------------|------------|--------------------|---------|--------|-----------------------|
| or_dMgG0ULN9ABeQw | 68RMU2XU1G | Guilherme Schafer | R\$0.01 | Pago | 24/11/2020 - 14:50:13 |
| or_nv6p1OLc5d01JMzn | R14WAN3SGP | guilherme schaffer | R\$0.01 | Pago | 17/11/2020 - 21:42:59 |
| or_M5KA6m1DE483dm | Y74FRV57Y | guilherme.schafer | R\$0.01 | Pago | 17/11/2020 - 21:42:27 |
| or_d7g9RMQu1WRz8w | 17WVFEIMNB | guilherme schaffer | R\$0.01 | Pago | 17/11/2020 - 21:15:44 |
| or_g7nQp4nCvQ28Rk1 | 63DKXIA9IG | guilherme schaffer | R\$0.01 | Pago | 17/11/2020 - 20:58:53 |
| or_KOM43g8GWQ8Rk | 5WY1FAFN9 | guilherme schaffer | R\$0.01 | Pago | 17/11/2020 - 19:17:23 |
| or_vGkZlRfPmHdRZLK3 | 8JN2F2556D | guilherme schaffer | R\$0.01 | Pago | 12/11/2020 - 15:39:11 |
| or_NyKbaUaRQz2d85 | 83QHGQH28 | guilherme schaffer | R\$0.01 | Pago | 12/11/2020 - 15:36:08 |
| or_OyVhLj7GmULRn | LRWWSVJAEX | guilherme schaffer | R\$0.01 | Falha | 12/11/2020 - 15:19:50 |
| or_DX7gZmivigLp54 | 4ENBHWWAY6 | guilherme schaffer | R\$0.01 | Falha | 12/11/2020 - 15:15:44 |
| or_h9yGZ7IaeC8K20 | L6KNHE9C3R | guilherme schaffer | R\$0.01 | Falha | 12/11/2020 - 15:11:43 |
| or_3kna7c345nqDv1 | XY3WH7PID3 | guilherme schaffer | R\$0.01 | Falha | 12/11/2020 - 14:29:14 |

Fonte: elaborado pelo autor.

Para realizar a cobrança o método `confirmPayment`, utilizou o `endpoint` de Pedidos. Quando gerado um pedido na plataforma MundiPagg, automaticamente é gerado uma cobrança no cartão do cliente, passando alguns dados do cliente para identificação, além dos dados do cartão para a cobrança (conforme pode ser visto no APÊNDICE B).

O cardápio online disponibilizado para o cliente visualizar os produtos e realizar os pedidos, conforme descritos acima, serão populados através de dois cadastros existentes no perfil de administrador (Figura 11), os cadastros de categorias do produto e os produtos. Os cadastros serão acessados através da opção Configurações na barra de menu no canto inferior (conforme pode ser visto no APÊNDICE C e D).

Figura 11 – Acesso Perfil Administrador



Fonte: elaborado pelo autor.

O ciclo de vida de um pedido realizado pelo cliente, pode haver alguns status, aonde irão indicar o momento que se encontra o pedido do cliente. Esses status serão atualizados principalmente por dois perfis, cozinha e bar/garçom. A cozinha irá atualizar o status do pedido em dois momentos, quando estiver preparando a comida do pedido solicitado e quando estiver com os pratos prontos. O bar/garçom irá atualizar em outros momentos, quando entregar as bebidas solicitadas e quando entregar as comidas preparadas pela cozinha.

Após todos os produtos solicitados pelo cliente foram preparados e entregues ao cliente, o status do pedido será Pedido Entregue, nesse momento será liberado ao cliente a opção de realizar o pagamento e com isso o ciclo de vida do pedido se encerra para Pedido Pago.

A rotina que executa todas as atualizações do pedido, menos o status de Pago, é o método `updateStatusOrder`. No qual identifica de qual perfil está sendo atualizado o status, através dos parâmetros recebidos, e irá atualizar o status do pedido conforme o status atual.

Quando o método `updateStatusOrder` (Quadro 9 – Método `UpdateStatusOrder`) recebe por parâmetros `DrinkSent` ou `OrderSent` a aplicação identifica que a atualização está sendo realizada pelo perfil de bar/garçom e analisando o status atual do pedido atualiza o status para o que faz mais sentido no ciclo de vida do pedido. Ao receber `InProgress` ou `FoodFinished`, é identificado que o prato está sendo preparado ou já se encontra pronto, portanto, a atualização está sendo realizado pelo perfil da cozinha.

Quadro 9 – Método `UpdateStatusOrder`

```
public void UpdateStatusOrder(OrderDto orderDto)
{
    var order = unitOfWork.OrderRepository.GetOne(o => o.Id == orderDto.Id);

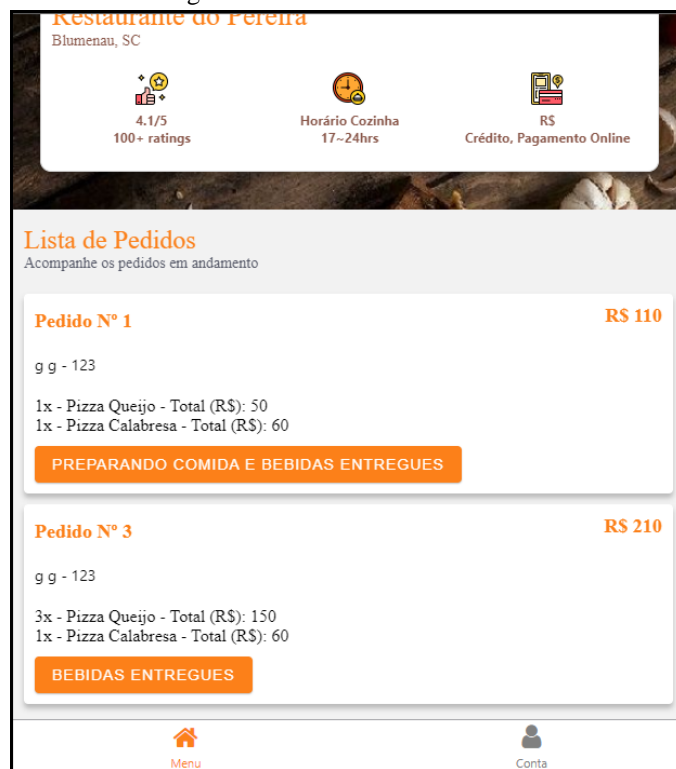
    if(order != null)
    {
        if(orderDto.Status == (int)Status.DrinksSent)
        {
            if(order.Status == Status.Open)
            {
                order.Status = Status.DrinksSent;
            }
            else if(order.Status == Status.InProgress)
            {
                order.Status = Status.InProgressDrinksSent;
            }
            else if (order.Status == Status.FoodFinishedAndDrinksSent)
            {
                order.Status = Status.OrderSent;
            }
            else if(order.Status == Status.FoodFinished)
            {
                order.Status = Status.FoodFinishedAndDrinksSent;
            }
        }
        else if (orderDto.Status == (int)Status.InProgress)
        {
            if (order.Status == Status.Open)
            {
                order.Status = Status.InProgress;
            }
            else if (order.Status == Status.DrinksSent)
            {
                order.Status = Status.InProgressDrinksSent;
            }
        }
        else if(orderDto.Status == (int)Status.FoodFinished)
        {
            if (order.Status == Status.InProgressDrinksSent)
            {
                order.Status = Status.FoodFinishedAndDrinksSent;
            }
            else if (order.Status == Status.InProgress)
            {
                order.Status = Status.FoodFinished;
            }
        }
        else if(orderDto.Status == (int)Status.OrderSent)
        {
            order.Status = Status.OrderSent;
        }
    }

    unitOfWork.Commit();
}
```

Fonte: elaborado pelo autor.

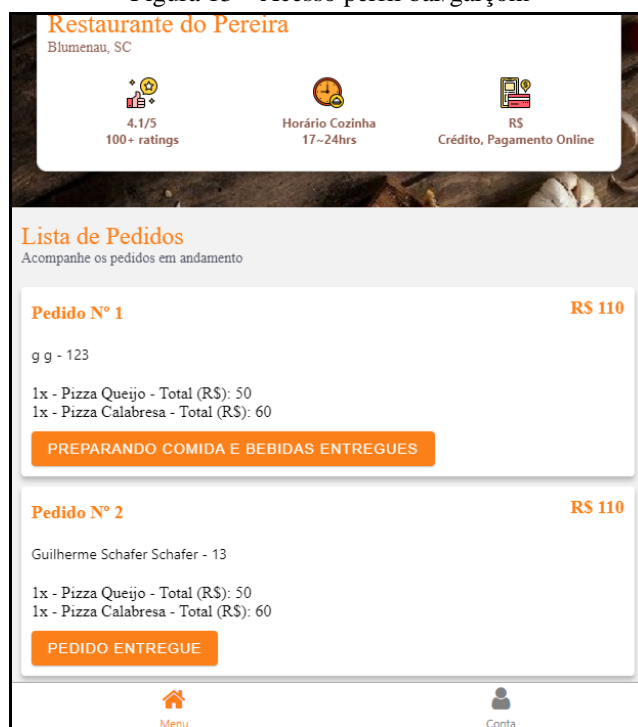
As interfaces liberadas para os perfis cozinha(Figura 12) e bar/garçom(Figura 13) são praticamente as mesmas, com alguns ajustes realizados no momento de chamar a aplicação *backend* para atualizar o status. Ao realizar o *login*, são apresentados os pedidos realizados por clientes no qual necessitam de alguma ação do respectivo perfil. Por exemplo, se todos os pratos do pedido já foram preparados e o pedido foi atualizado pela cozinha, não será mais apresentado na interface. O mesmo acontece para o perfil de bar/garçom, ao cliente realizar o pagamento do pedido.

Figura 12 – Acesso Perfil Cozinha



Fonte: elaborado pelo autor.

Figura 13 – Acesso perfil bar/garçom



Fonte: elaborado pelo autor.

A atualização do status do pedido por um perfil, vai impactar na interface do outro. Ou seja, caso a cozinha atualize o pedido indicando que a comida se encontra pronta, o status do pedido será atualizado automaticamente na interface do bar/garçom, para indicar ao garçom a necessidade de entregar o prato diretamente ao cliente.

Todas as consultas realizadas pelo *frontend* para apresentar os dados das interfaces foram feitas através de *endpoints* API REST, construídos na aplicação do servidor *backend*. Praticamente todos os *endpoints* do servidor

necessitam de um Bearer Token, gerado no momento do *login*. Aonde dentro do token criptografado, existem algumas informações como o perfil, usuário logado, data de expiração do *token*. O *token* é gerado pelo método *GenerateToken* apresentado no Quadro 10 – Método *GenerateToken*.

Quadro 10 – Método *GenerateToken*

```
public static string GenerateToken(User user)
{
    var tokenHandler = new JwtSecurityTokenHandler();
    var key = Encoding.ASCII.GetBytes(Settings.Secret);
    var tokenDescriptor = new SecurityTokenDescriptor
    {
        Subject = new ClaimsIdentity(new Claim[]
        {
            new Claim(ClaimTypes.Name, user.Login.ToString()),
            new Claim(ClaimTypes.Role, user.Perfil.ToString())
        }),
        Expires = DateTime.UtcNow.AddHours(2),
        SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)
    };
    var token = tokenHandler.CreateToken(tokenDescriptor);
    return tokenHandler.WriteToken(token);
}
```

Fonte: elaborado pelo autor.

Na camada de *frontend* o método *bootstrapAsync* (Quadro 11 – Método *bootstrapAsync*), valida o perfil adicionado ao *token* para realizar o direcionamento correto para as rotas da aplicação a data de expiração do *token* já tenha sido alcançado realiza a remoção das informações do *LocalStorage* do browser e direciona para tela de *login* novamente.

Quadro 11 – Método *bootstrapAsync*

```
_bootstrapAsync = async () => {
    const userToken = await AsyncStorage.getItem('userToken');

    if (userToken == undefined || userToken == null) {
        this.props.navigation.navigate('Auth');
    }
    else {
        let decodeToken = jwt_decode(userToken);
        console.log(decodeToken);
        if (decodeToken.exp > (new Date().getTime() + 1) / 1000) {
            if (decodeToken.role == 0) {
                this.props.navigation.navigate('HomeAdmin');
            }
            else if (decodeToken.role == 1) {
                this.props.navigation.navigate('HomeKitchen');
            }
            else if (decodeToken.role == 2) {
                this.props.navigation.navigate('HomeBarBartender');
            }
            else if (decodeToken.role == 4) {
                this.props.navigation.navigate('Home');
            }
        }
        else {
            await AsyncStorage.clear();
            this.props.navigation.navigate('Auth');
        }
    }
    // This will switch to the App screen or Auth screen and this loading
    // screen will be unmounted and thrown away.
};
```

Fonte: elaborado pelo autor.

4 RESULTADOS

Esta seção apresenta os testes realizados com a aplicação e os resultados obtidos através destes testes. A seção foi dividida em duas subseções. A primeira demonstra os testes das funcionalidades realizadas durante a fase de implementação. A segunda subseção descreve os testes realizados por um usuário simulando a utilização do aplicativo em um estabelecimento.

Em relação aos trabalhos correlatos da aplicação apresentada nesse artigo, o Quadro 12 tem-se a comparação das características dessas aplicações com a apresentada nesse artigo.

Quadro 12 – Comparação correlatos

| | Find Food (2017) | Estacione (2018) | Controle e Agend. Medicamentos (2015) | Comanda Digital de Bares e Restaurantes (2020) |
|---|------------------|------------------|---------------------------------------|--|
| Plataforma | Android e iOS | Android | Android | Android, iOS e Desktop |
| Precisa instalar via loja | Sim | Sim | Sim | Não |
| Auxiliam na experiencia em um restaurante | Não | Não | Não | Sim |
| Apresenta localização | Sim | Sim | Não | Não |
| Realiza pagamento através do app | Não | Sim | Não | Sim |
| Realiza processamento no dispositivo | Não | Não | Não | Não |
| Trabalha offline | Não | Sim | Sim | Sim (alguns requisitos) |

Fonte: elaborado pelo autor.

Como pode ser visto na comparação no Quadro 12, a aplicação desenvolvida e descrita nesse artigo, alcançou os principais objetivos. No qual os trabalhos correlatos não conseguiram alcançar, seriam esses: funcionamento em todas as plataformas, sem necessidade de instalação via loja, pagamento online via aplicação (somente o trabalho correlato Estacione obteve essa funcionalidade).

4.1 TESTE INICIAIS

Durante a implementação da ferramenta foram feitos testes em todas as funcionalidades, em todos os tipos de dispositivos (Android, iOS, Desktop), para tentar encontrar e sanar possíveis falhas. No decorrer dos testes foi visto que houve erros principalmente na parte de interface de usuários. Pois dependendo o tamanho do dispositivo do usuário, algumas informações acabam ultrapassando o limite da tela, por isso foi preciso ajustar alguns textos e modificar a visualização de alguns componentes. Alguns componentes acabavam escondendo alguns campos, conforme eram apresentadas mensagens de validações de campos, por não estar sendo tratado no componente uma barra de rolagem.

Não houve grandes complicações na utilização da aplicação em celulares Android mais antigo. No caso foi utilizado para teste um dispositivo na versão 5.0.2, e não encontrou nenhum problema em abrir a aplicação via chrome v86.

Em um dos testes de atualização dos pedidos nos perfis de cozinha e bar, a interface atualizava os pedidos a todo o tempo, o servidor Azure estava acusando falta de recursos devido a quantidade de requisições realizadas ao servidor. Para resolver esse problema foi incluído um timer no *frontend*, para a interface realizar a requisição ao servidor a cada dois segundos, dessa maneira a cada dois segundo é atualizado o status dos pedidos nos dois perfis.

Para realizar os testes na plataforma da MundiPag, foi utilizado os seguintes cartões: 4000000000000010 no qual retorna que a cobrança foi realizada com sucesso, 4000000000000028 retornando que houve uma falha na cobrança do pagamento. Os dois cartões utilizados corresponderam corretamente nos testes com seus respectivos objetivos. As requisições de cobranças podem ser visualizadas no dashboard da plataforma MundiPag, conforme demonstrado na Figura 10 – Plataforma MundiPag.

No *backend* foi percebido que a autenticação em cada *endpoint* não estava sendo considerado, ou seja, todos os *endpoint* estava sendo possível consumir sem a necessidade de *token*. Para resolver essa questão foi necessário deixar explícito em cada *endpoint* a necessidade de estar autenticado (conforme demonstrado no Quadro 13 – *Endpoint* com autorização explícito)

Quadro 13 – *Endpoint* com autorização explícito

```
[HttpPost]
[Route("orderConfirm")]
[Authorize] ←
public IActionResult OrderConfirm([FromBody] List<OrderDto> orderItems)
{
    try
    {
        var userLogin = User.Claims.LastOrDefault(c => c.Type == ClaimTypes.Name).Value;

        orderProvider.ConfirmOrder(orderItems, userLogin);
        return Ok();
    }
    catch (Exception e)
    {
        return BadRequest(e.Message);
    }
}

[HttpPost]
[Route("confirmPayment")]
[Authorize] ←
public IActionResult ConfirmPayment([FromBody] PaymentDto payment)
{
    try
    {
        var x = orderProvider.ConfirmPayment(payment).Result;
        return Ok();
    }
    catch (Exception e)
    {
        return BadRequest(e.Message);
    }
}
```

Fonte: elaborado pelo autor.

4.2 TESTES COM USUARIOS

Os testes foram realizados pelo professor orientador desse artigo. Para isso foi disponibilizado o link da aplicação hospedada na nuvem, mais um usuário de cada perfil (administrador, cliente, cozinha e bar), com isso foi testado as principais funcionalidades da aplicação.

O usuário acessou todos os perfis ao mesmo tempo de dispositivos diferentes: um usuário no perfil cliente foi acessado através do celular, outro do perfil administrador através do Safari em um desktop, outro do perfil cozinha no Chrome através de um desktop e o perfil bar foi acessado através do Safari em um iPad.

Foi realizado cadastros de novos produtos, simularam-se algumas compras de produtos através do cliente e realizou o ciclo de vida do pedido através dos perfis de cozinha e bar, além do pagamento via cartão fictício após a entrega de todo o pedido.

Nos testes não foi encontrado nenhuma situação de erro onde poderia impedir o funcionamento da aplicação. As interfaces foram intuitivas para a realização das solicitações de pedidos e pagamentos.

5 CONCLUSÕES

Apesar do pequeno número de avaliações, considera-se que a aplicação cumpriu o principal objetivo, facilitando ao cliente do restaurante realizar os pedidos no estabelecimento, apresentando o pedido para as áreas responsáveis e permitindo o pagamento realizado através do aplicativo, dessa maneira diminuindo a interação do cliente com funcionários do restaurante.

A aplicação por ser web desenvolvida em React Native, conseguiu cumprir um segundo objetivo que seria executar em todas as plataformas e isso foi alcançado com sucesso, conforme descrito nos testes, a aplicação funciona até em dispositivos mais antigos, dessa maneira podendo alcançar um maior número de pessoas, nos mais variados tipos de restaurantes.

Os clientes do estabelecimento conseguiram realizar os pedidos e acompanhar o status dos pedidos sem a necessidade de um garçom, e realizaram os pagamentos através dos cartões via aplicativo. Sem a necessidade de instalação nos próprios dispositivos móveis, somente através da leitura de um QR Code, disponibilizado pelo estabelecimento.

Por fim, ao realizar os testes finais do aplicativo, foram identificados alguns pontos a melhorar no aplicativo:

- a) implementar um filtro por categoria no cardápio para que o cliente tenha a possibilidade de selecionar a

- categoria de produtos que mais goste;
- b) implementar o perfil para o caixa do restaurante, para controlar os pedidos a serem pagos e até realizar uma baixa nos pedidos dos clientes que desejarem pagar diretamente no caixa, contemplando a lei PAF-ECF;
- c) implementar outras possibilidades de pagamento, além do cartão de crédito, como cartão de débito, picpay e outras formas, visto que existem plataformas de integração que possibilitam essa melhoria;
- d) melhorar a visualização dos pedidos na parte do administrador, para que possa ser realizadas consultas de pedidos antigos e até extração de relatórios;
- e) implementar a autenticação via redes sociais;
- f) possibilitar incluir uma observação em cada item do produto do pedido, para que possa ser solicitado remoção de algum ingrediente do prato;

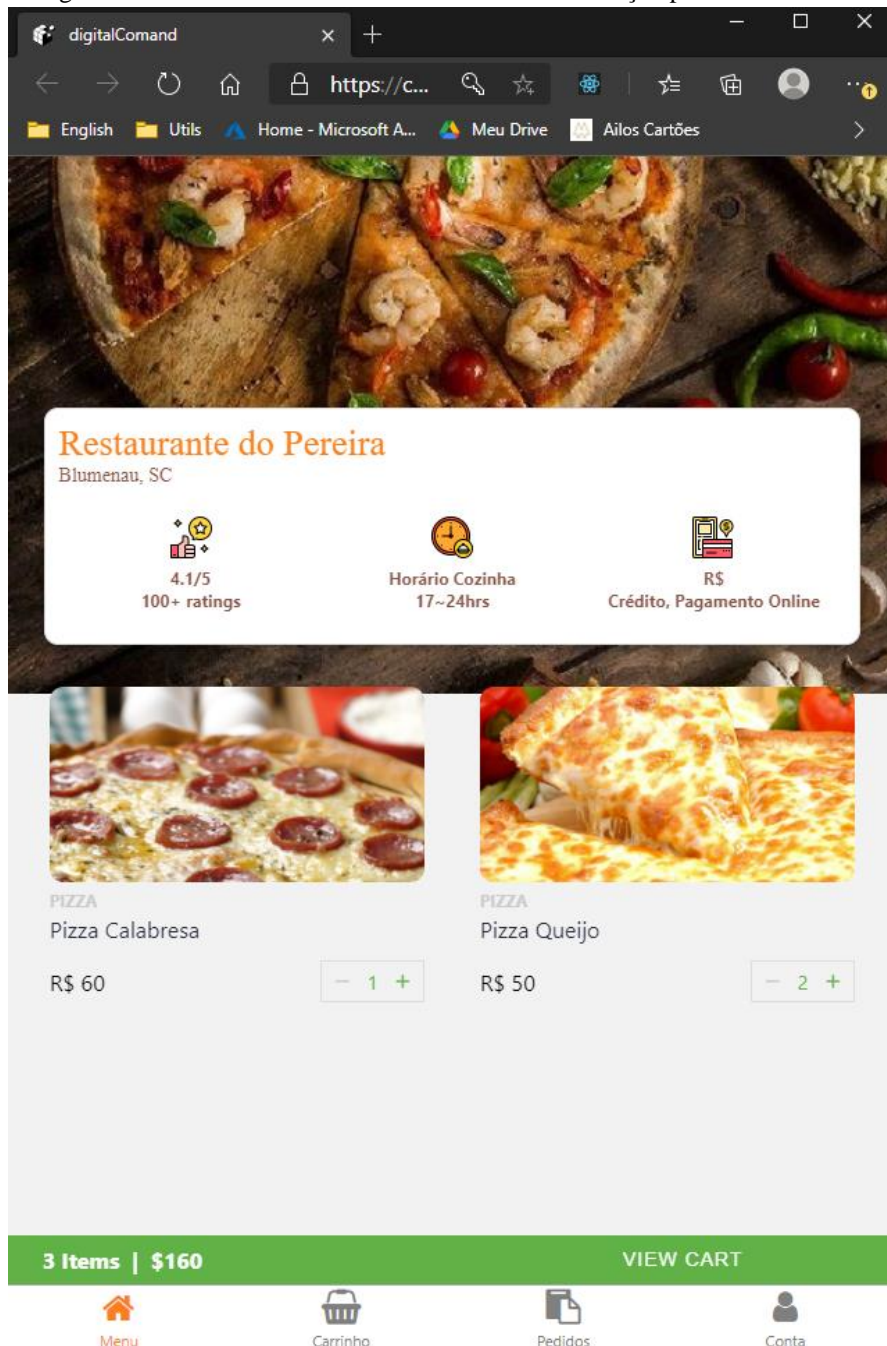
REFERÊNCIAS

- ARAÚJO, Thiago Lippel de. **Find Food**: Aplicativo para Localização e recomendação de restaurantes; 2017. 51f Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação – Centro de Ciências Exatas e Naturais; Universidade Regional de Blumenau, Blumenau.
- BORBA JUNIOR, José Celso de Borba. **Aplicativo mobile para controle e agendamento de consumo de medicamentos**; 2015. 42f Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação – Centro de Ciências Exatas e Naturais; Universidade Regional de Blumenau, Blumenau.
- BÜLL, Patrícia. **Gastos com delivery crescem mais de 94% na pandemia**, [2020]. Disponível em: <https://www.consumidormoderno.com.br/2020/07/08/gastos-com-delivery-crescem-mais-de-94-durante-a-pandemia/>. Acesso em: 04 dez. 2020.
- FERNANDA, Raísa. **Conheça as principais tendências tecnológicas em gestão de restaurantes**, [2018]. Disponível em: <https://www.ticket.com.br/blog/inovacao-e-tecnologia/conheca-as-principais-tendencias-tecnologicas-em-gestao-de-restaurantes> . Acesso em: 27 jul. 2018.
- GALVÃO, Daniel. **Crescimento do mercado mobile altera comportamento sobre uso de apps**, [2019]. Disponível em: <https://digitalks.com.br/artigos/crescimento-do-mercado-mobile-altera-comportamento-sobre-uso-de-apps/>. Acesso em: 28 nov. 2020.
- GLIZT, Fernando Clovis. **Desenvolvimento de Programa Aplicativo Fiscal (PAF), Emissor de Cupom Fiscal (ECF) e Transferência Eletrônica de Fundos (TEF)**. 2012. Trabalho de Conclusão de Curso. Universidade Tecnológica Federal do Paraná.
- KOGLIN JUNIOR, Paulo Arnaldo. **Estacione**: protótipo de aplicativo para pagamento móvel de estacionamento; 2018. 85f Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação – Centro de Ciências Exatas e Naturais; Universidade Regional de Blumenau, Blumenau.
- MAGALHÃES, Casa. **Restaurante na era digital: como satisfazer clientes exigentes?**, [2018]. Disponível em: <https://www.casamagalhaes.com.br/blog/tecnologia/restaurante-na-era-digital> . Acesso em: 10 out. 2018.
- MOHSIN, Maryam. **9 estatísticas sobre compras online para o ano de 2020**, [2020]. Disponível em: <https://www.oberlo.com.br/blog/estatisticas-compras-online#:~:text=A%20cada%2010%20compras%20realizadas%20no%20e%2Dcommerce%2C%207%20s%C3%A3o,R%24%2090%20bilh%C3%B5es%20em%202020>. Acesso em: 28 nov. 2020.
- SCHNAIDER, Amanda. **E-commerce cresce 47%, maior alta em 20 anos**, [2020]. Disponível em: <https://www.meioemensagem.com.br/home/marketing/2020/08/27/e-commerce-cresce-47-maior-alta-em-20-anos.html>. Acesso em: 28 nov. 2020.
- SCUADRA, **Transformação digital em foodservice**: saiba como aplicar no seu restaurante, [2018]. Disponível em: <https://www.scuadra.com.br/blog/transformacao-digital-em-foodservice-saiba-como-aplicar-no-seu-restaurant/>. Acesso: 28 nov. 2020.

APÊNDICE A – INTERFACE DO MENU COM ITENS ADICIONADOS NO CARRINHO

Após atualizado as variáveis de controle do carrinho, é apresentado uma barra próxima do menu inferior para o cliente visualizar quantos itens já foram adicionados ao seu carrinho e qual o valor total do pedido no momento (Figura 14). O pedido pode ser acessado por completo ao clicar no menu Carrinho.

Figura 14 – Interface do menu com a barra de visualização parcial do carrinho



Fonte: elaborado pelo autor.

APÊNDICE B – MÉTODO CONFIRMPAYMENT

Método de confirmação de pagamento, realiza o consumo do *endpoint* disponibilizado pela plataforma MundiPagg. Após a chamada do *endpoint*, recebe o retorno da cobrança gerada na operadora do cartão, e atualiza o status do pedido caso receba uma confirmação do pagamento. (Quadro 14)

Quadro 14 - ConfirmPayment

```
public async Task<bool> ConfirmPayment(PaymentDto paymentDto)
{
    var order = unitOfWork.OrderRepository.GetOne(o => o.Id == paymentDto.Id);
    var totalOrder = 0.00;

    foreach (var item in order.ListOfItens)
    {
        totalOrder += (item.Quantity * item.Product.Value);
    }

    if (order != null)
    {
        var mundiPaggItems = new MundiPaggItemDto();
        mundiPaggItems.items = new List<MundiPaggItemsDto>();
        mundiPaggItems.items.Add(new MundiPaggItemsDto
        {
            Amount = 1, //(int)totalOrder,
            Description = "Pedido nº" + order.Id,
            Quantity = 1
        });

        var customer = new MundiPaggCustomerDto
        {
            Name = order.Client.Fullname,
            Email = order.Client.Email,
        };

        var payments = new List<MundiPaggPaymentsDto>();
        payments.Add(new MundiPaggPaymentsDto
        {
            payment_method = "credit_card",
            credit_card = new MundiPaggCrediCardDto
            {
                card = new MundiPaggCardDto
                {
                    holder_name = paymentDto.CardName,
                    number = paymentDto.CardNumber,
                    exp_month = Int32.Parse(paymentDto.ValidUntil.Substring(0, 2)),
                    exp_year = Int32.Parse(paymentDto.ValidUntil.Substring(3, 2)),
                    cvv = paymentDto.Cvv
                }
            }
        });

        var request = new
        {
            items = mundiPaggItems.items,
            customer = customer,
            payments = payments
        };

        HttpClient client = new HttpClient();
        client.BaseAddress = new Uri("https://api.mundipagg.com/core/");
        client.DefaultRequestHeaders.Accept.Clear();
        client.DefaultRequestHeaders.Accept.Add(
            new MediaTypeWithQualityHeaderValue("application/json"));
        client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Basic", "[REDACTED]");
        var requestJson = JsonConvert.SerializeObject(request);
        HttpResponseMessage response = await client.PostAsync(
            "v1/orders/", new StringContent(requestJson, Encoding.UTF8, "application/json"));

        var responseObject = JsonConvert.DeserializeObject<MundiPaggResponse>(response.Content.ReadAsStringAsync().Result);

        if(responseObject.status != "paid")
        {
            var error = responseObject.charges.First().last_transaction.acquirer_message;
            throw new Exception(error);
        }
        else
        {
            order.Status = Status.Payed;
            unitOfWork.Commit();
        }
    }

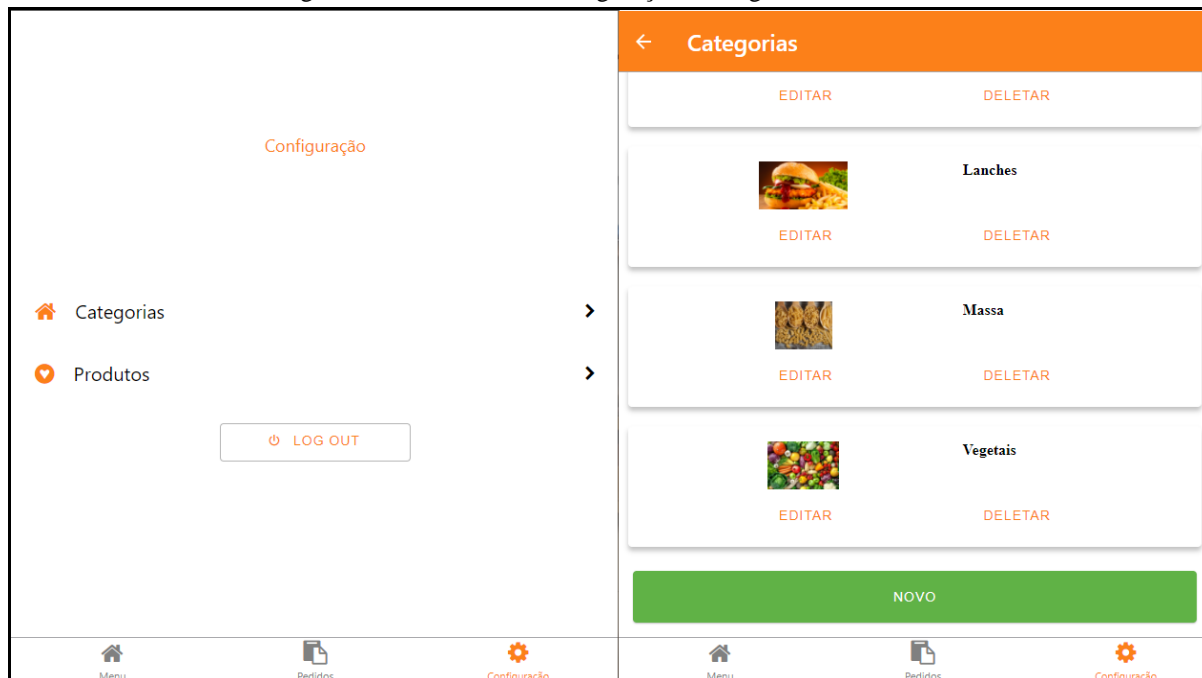
    return true;
}
```

Fonte: elaborado pelo autor.

APÊNDICE C – ORDEM DE INTERFACES PARA CADASTRO DAS CATEGORIAS

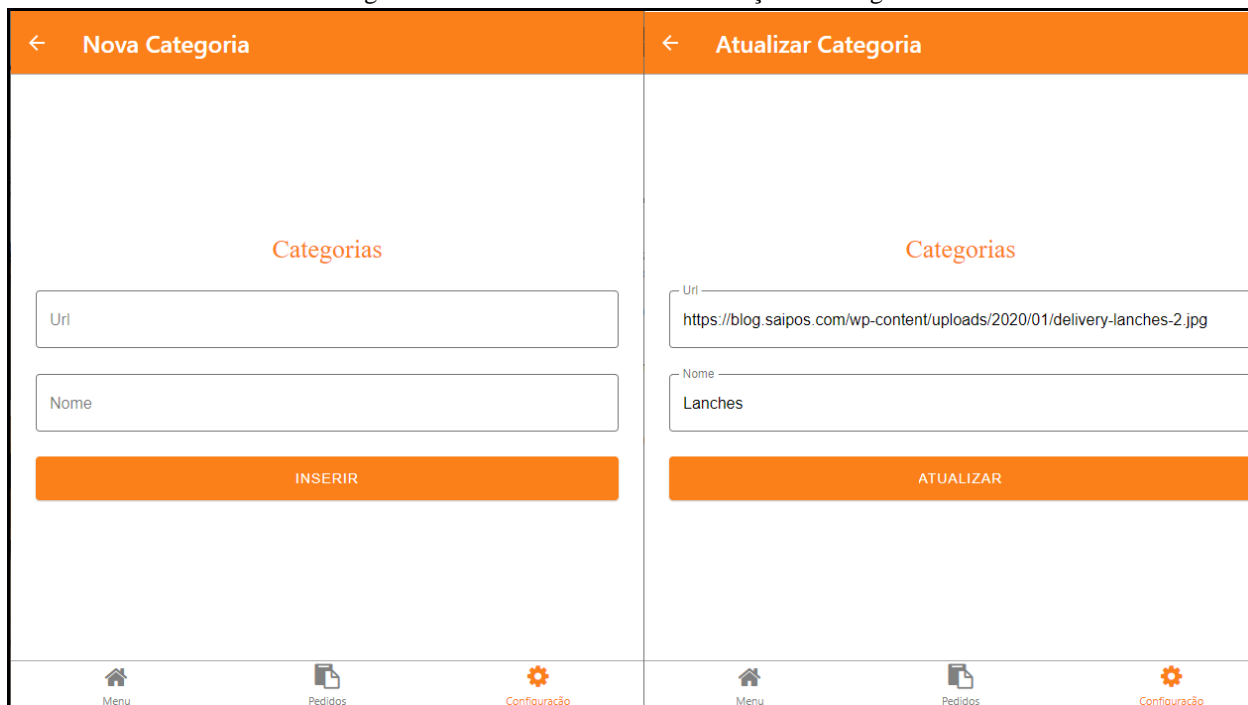
A ordem das interfaces para realizar o cadastro das categorias dos produtos, através do menu configurações do perfil de administrador. (Figura 15 e Figura 16)

Figura 15 – Interface de configuração e categorias cadastrados



Fonte: elaborado pelo autor.

Figura 16 – Interface de cadastro e edição da categoria

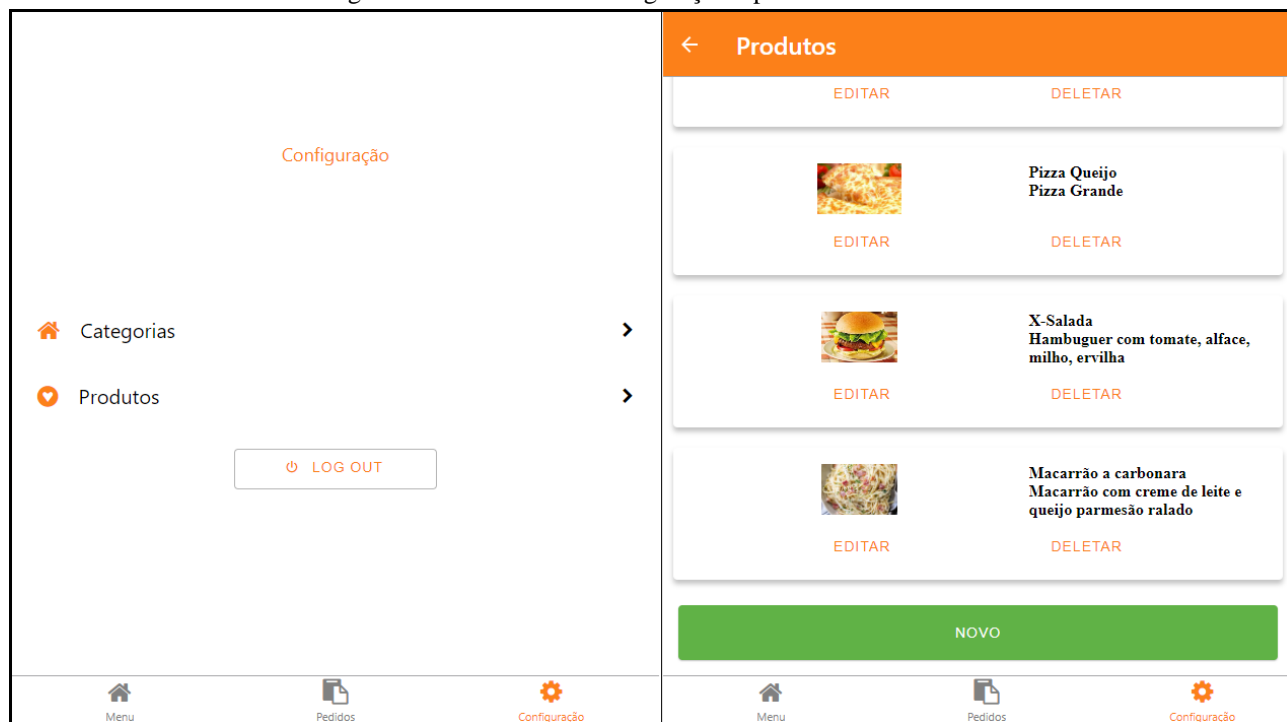


Fonte: elaborado pelo autor.

APÊNDICE D – ORDEM DE INTERFACES PARA CADASTRO DOS PRODUTOS

A ordem das interfaces para realizar o cadastro dos produtos, através do menu configurações do perfil de administrador. A Figura 17 demonstra a opções de cadastro e os produtos já cadastrados. A Figura 18, demonstra as telas para cadastrar e editar produtos.

Figura 17 – Interface de configuração e produtos cadastrados



Fonte: elaborado pelo autor.

Figura 18 – Interface de cadastrar e editar produtos

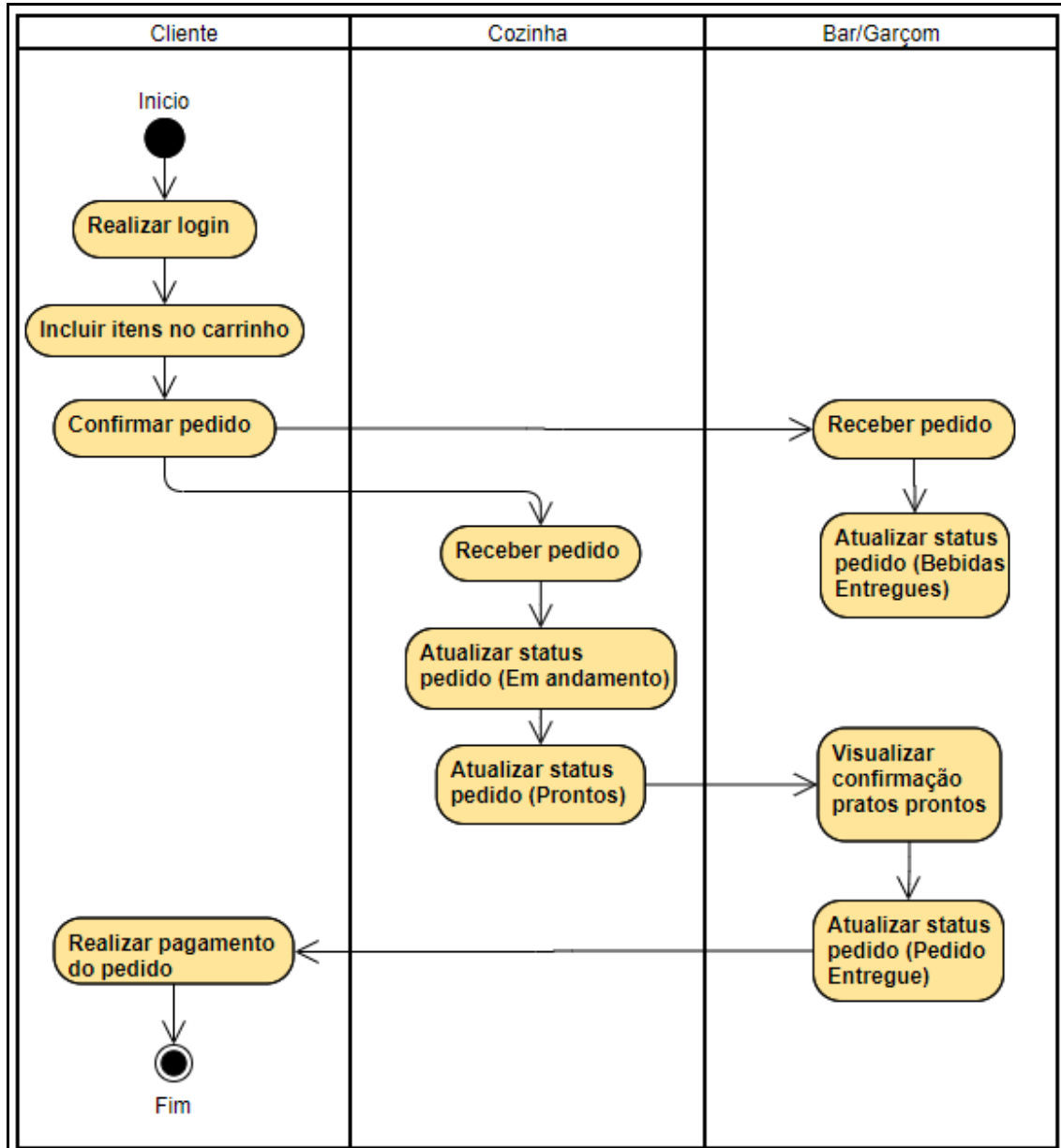
| ← Novo Produto | ← Atualizar Produto |
|---|--|
| <p>Produtos</p> <p>Nome</p> <p>Descrição</p> <p>Url Imagem</p> <p>Valor</p> <p>Pizza</p> <p>INSERIR</p> | <p>Produtos</p> <p>Nome</p> <p>X-Salada</p> <p>Descrição</p> <p>Hamburger com tomate, alface, milho, ervilha</p> <p>Url Imagem</p> <p>https://encrypted-tbn0.gstatic.com/images?q=tbn:AND9GcT8oUX1IHVxKLGm1I</p> <p>Valor</p> <p>20</p> <p>Lanches</p> <p>ATUALIZAR</p> |

Fonte: elaborado pelo autor.

APÊNDICE E – DIAGRAMA DE ATIVIDADE DO PEDIDO

O ciclo de vida do pedido dentro da aplicação inicia a partir da inclusão dos itens no carrinho, apresentado e atualizado pela área da cozinha e os garçons e sendo finalizado pelo pagamento do pedido pelo cliente. (Figura 19)

Figura 19 - Diagrama de atividade do ciclo de vida do pedido



Fonte: elaborado pelo autor.