BillSystem – Solução para Comandas

Lucas Sutelo[1]

Silvio Cesar Viegas[2]

RESUMO

Imagine o seguinte cenário: Você está em uma grande clínica estética, recebendo uma massagem relaxante. Ao finalizar a sessão da massagem, você decide cortar o cabelo com outra profissional do estabelecimento. Para haver um controle dos procedimentos realizados e um conforto para você, a massoterapeuta se desloca até o caixa e informa a pessoa responsável sobre o serviço realizado, juntamente com o valor a ser cobrado no final de sua visita. Ao finalizar o corte de cabelo, a cabelereira também se desloca ao caixa para informá-lo sobre o procedimento e adicionar à sua conta. Quando você deseja ir embora, se dirige ao caixa e paga pelos procedimentos realizados. Este processo, apesar de ser aparentemente confiável, é extremamente desgastante e improdutivo.

1. INTRODUÇÃO

Pensando neste problema, foi desenvolvido o BillSystem, que transformou este processo em algo simples, através de uma comanda totalmente online e acessível por todas as partes. Com o uso deste software, os estabelecimentos puderam ter uma maior agilidade na entrega de seus serviços, não abrindo mão da confiança e consistência de informações.

1.1 Objetivo geral

Desenvolver um software capaz de gerenciar uma comanda eletrônica, a fim de controlar os procedimentos realizados em uma clínica estética.

1.2 Objetivos específicos

- Definir as linguagens de programação e frameworks que serão utilizados;
- Usar conceitos de UML;
- Organizar processos;
- Modelar banco de dados;

[1] Acadêmico de Análise e Desenvolvimento de Sistemas da FAQI / FAQI / Gravataí / Rio Grande do Sul / Brasil / lucassutelo@gmail.com

[2] Mestre / FAQI / Gravataí / Rio Grande do Sul / Brasil / silvio.viegas@qi.edu.br

1.3 Tema

Um sistema capaz de gerenciar e manter em total organização, os serviços realizados em uma clínica estética, através de uma comanda eletrônica, que conterá os valores pendentes de pagamento.

1.4 Delimitação do tema

O sistema será apenas para controle dos gastos e exibição do valor total devido. Ele não irá abranger o faturamento do mesmo.

1.5 Justificativa

O sistema é importante pois a clínica estética oferece diversos serviços, com diversas profissionais. Com o uso deste software pelas profissionais, as clientes poderão usufruir de diversos procedimentos, com variadas profissionais, e realizar um único pagamento no final de sua visita.

1.6 Problema

O sistema garantirá um controle dos gastos da cliente e evitará cobranças indevidas, ou mesmo, que seja esquecido de cobrar determinado valor.

2. FUNDAMENTAÇÃO TEÓRICA

Nesta sessão, serão abordadas as tecnologias utilizadas para o desenvolvimento e hospedagem do software.

2.1 Linguagem de programação

Para criação de softwares, é necessário conhecer uma forma de comunicação humano computador, chamada de linguagem de programação, que é constituída por conjuntos de instruções. De acordo com Leitão (1995) as linguagens de programação devem obter simplicidade para o melhor entendimento e juntas serem capazes de criar 20 abstrações complexas.

2.2 Banco de dados

Um banco de dados é uma coleção organizada de dados (esquemas, tabelas, consultas, relatórios, exibições e outros objetos). Os dados são normalmente organizados para modelar aspectos da realidade de uma forma que suporta os processos de pedidos de informação, tais como modelar a disponibilidade de quartos

em hotéis de uma forma que permita encontrar um hotel com vagas. (www.portalgsti.com.br/banco-de-dados/sobre)

2.3 JavaScript

JavaScript é uma linguagem de programação que permite a criação de conteúdo que se atualiza dinamicamente, controlar múltimídias, imagens animadas, e tudo o mais que há de interessante. (developer.mozilla.org/pt-BR/docs/Learn/JavaScript/)

O JavaScript também é muito utilizado em frameworks específicos, para a criação de back-end para servir dados a uma API e, até mesmo, criação de aplicações mobile.

2.4 Node

Node.JS se trata de uma plataforma server-side que promete reduzir a carga de processamento na máquina do servidor através de uma arquitetura que atende a todas as requisições feitas ao servidor de forma que não ocorram bloqueios de I/O e

livre de deadlocks. Além disso o Node.js permite um maior controle do servidor por parte do desenvolvedor por este se tratar de uma plataforma em linguagem baixa. (www.devmedia.com.br)

2.5 React-native

React Native é um Framework para a criação aplicações mobile nativas, no qual a principal linguagem de desenvolvimento é o JavaScript.(www.devmedia.com.br)

3. METODOLOGIAS

Este projeto está sendo desenvolvido através de um método de pesquisa aplicada, na qual foi visitado o ambiente real, que fará uso do software de forma experimental.

Para o uso do software, serão utilizados smartphones e tablets com sistema operacional android, e conexão à internet.

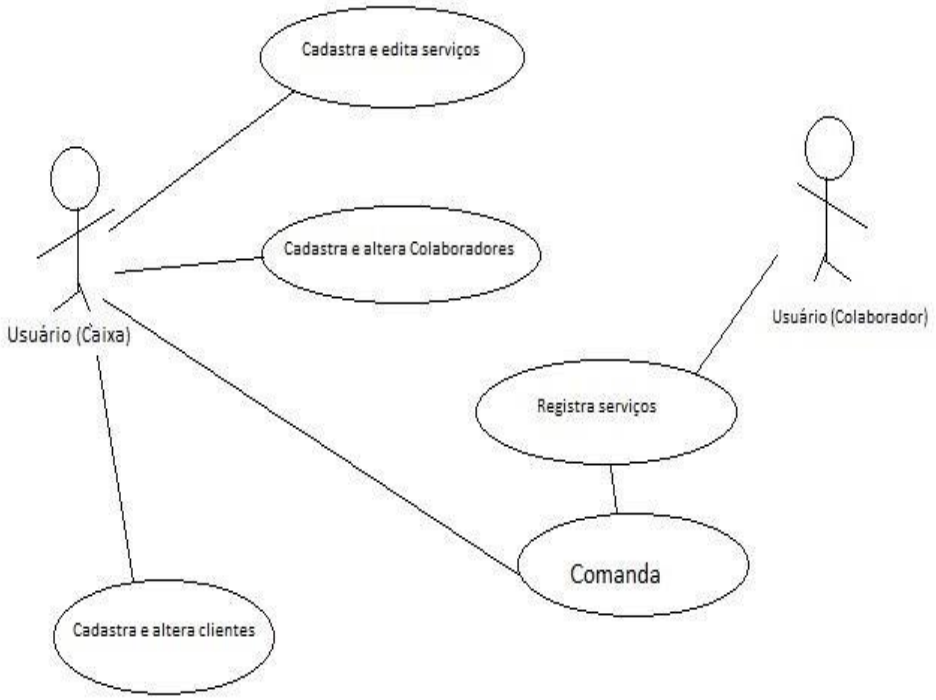
A versão front-end do software, que será desenvolvida em React-native, permanecerá instalada em todos os dispositivos. A versão back-end, que será desenvolvida em node, juntamente com o banco de dados mySql, será hospedada

em um servidor em nuvem. Isto possibilitará o uso do software de qualquer local que possua uma conexão à internet, seja por wi-fi ou rede móvel.

4. DESENVOLVIMENTO

Primeiramente, foi realizada uma pesquisa ao local e pessoas que farão o uso do software. Nesta ocasião, foi feito um levantamento de requisitos e elaborado um breve diagrama de casos de uso.

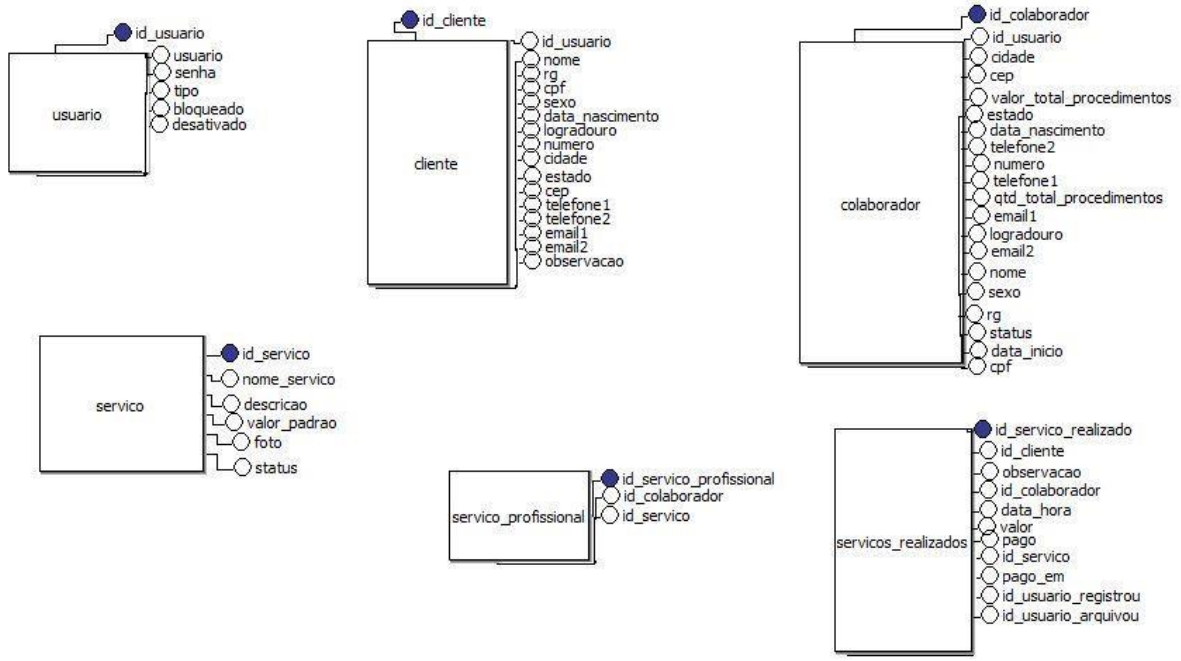
Figura 1 – Diagrama de Casos de Uso



Tendo elaborado este diagrama, pôde ser aprofundado o estudo das necessidades, tanto de rotinas, dados que necessitarão ser armazenados, processos e fluxos.

O próximo passo foi modelar o banco de dados, que será o responsável por armazenar e fornecer todos os dados de registro. Para esta etapa, foi utilizada a ferramenta opensource chamada brModelo. O banco de dados escolhido foi mySql, que trata-se de um banco relacional, destacando-se pela sua ótima performance e seu fácil gerenciamento.

Figura 2 - Modelo conceitual do banco de dados



Para que o banco de dados fique sempre disponível para acesso, e para que este seja performático, foi contratado um plano de hospedagem node, da empresa *KingHost.*

Estando contratado e disponível para uso, foi criado o banco de dados, juntamente com todas as tabelas e atributos desta.

O próximo passo foi criar o back-end. Nesta etapa, foi utilizado os gerenciadores de pacotes npm – presente no pacote node – e também o yarn, ambos sendo acessados através do prompt de comando do Windows.

O projeto foi criado utilizando a versão 11.14.0 do node.

Na sequência, foram instaladas todas as dependências necessárias para o uso da api:

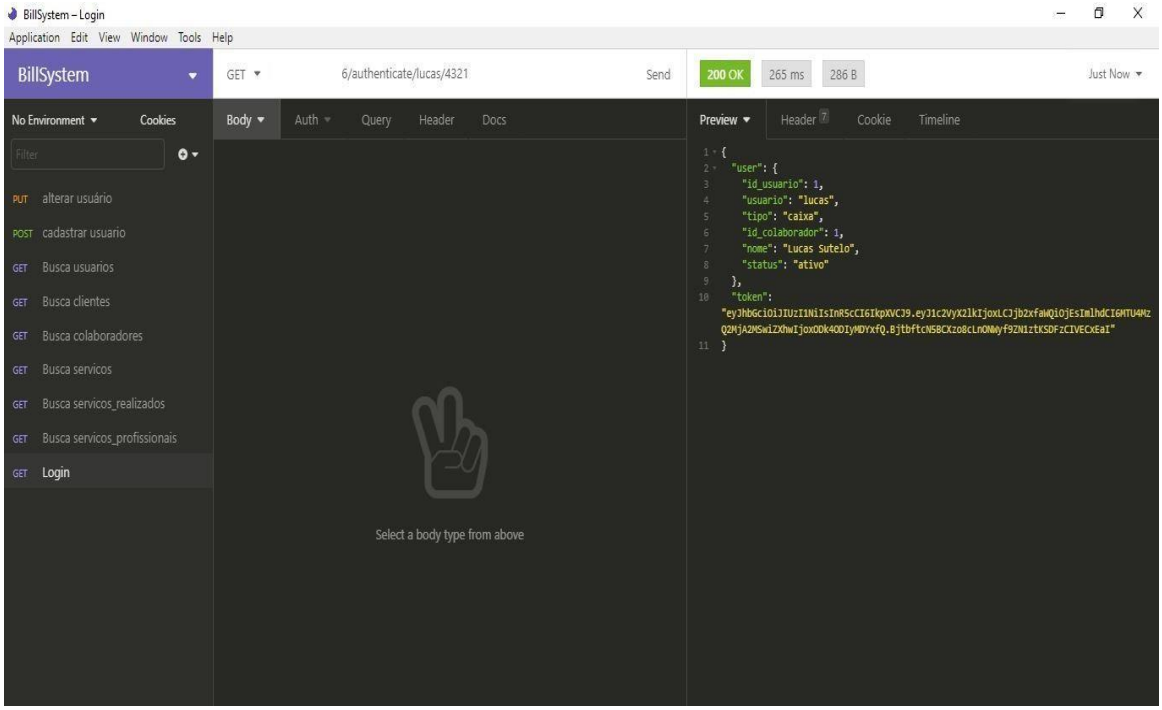
```
"dependencies": {
    "axios": "^0.19.2", //Para realizar requisições à API
    "cors": "^2.8.5", //Para permitir acesso externo
    "express": "^4.17.1", //Auxilia na construção das nossas aplicações Web
    "jsonwebtoken": "^8.5.1", //Usado para gerar tokens criptografados e passar segurança no acesso à API
    "knex": "^0.20.10", //Responsável pela comunicação do node com o banco de dados mySql
    "mysql": "^2.18.1", //Auxiliar na comunicação ao banco de dados
    "socket.io": "^2.3.0" //Permite a transferência de dados cliente-servidor em tempo real
```

Para realizar a codificação de todo o projeto, foi utilizado o gerenciador de código Visual Studio Code (Microsoft, 2015). Este editor, além de ser extremamente leve, possui mecanismos que auxiliam a geração de código, como o autocomplete (que sugere as funções, variáveis e propriedades disponíveis para uso), e demais verificações de sintaxe.

Estando pronto o back-end, com suas rotas e controllers criados, foram realizados testes com o auxílio do software *Insomnia, que faz chamadas HTTP à API de forma simples e rápida.*

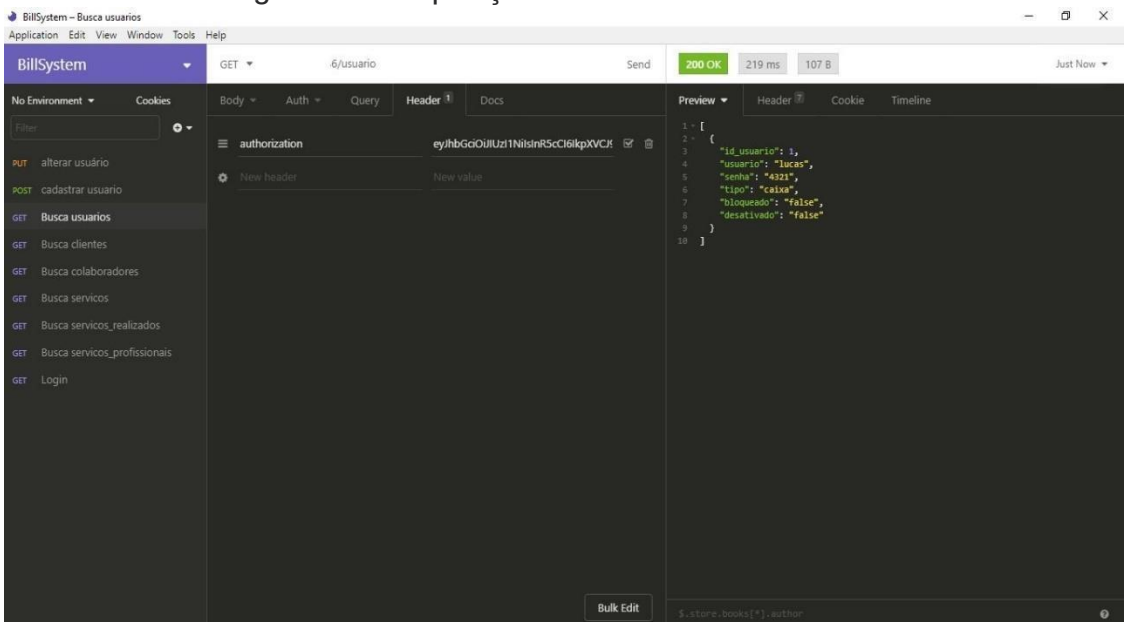
A figura a seguir, mostra a rota de login em funcionamento, onde são informadas as credenciais corretas (que já foram criadas anteriormente), e são retornados os dados do usuário e colaborador respectivo a este, juntamente com um token criptografado. Este token será armazenado futuramente, no banco de dados local do aparelho, na qual usará para que consiga realizar as demais requisições à API.

Figura 4 – Geração de token para login



Tendo posse de um token válido, já é possível realizar requisições para as demais funções, bastando informá-lo no cabeçalho da requisição.

Figura 5 – Requisição listando usuários cadastrados



Estando com a API desenvolvida e rodando no servidor, partiu-se para a etapa de desenvolvimento do front-end. Para isto, foi gerado um novo projeto em react-native, utilizando o comando “npx react-native init BillSystem”. O projeto foi criado utilizando a versão 0.61.5 do react-native.

Após criado, foram instaladas todas as dependências que serão necessárias.

```
"dependencies": {
"@react-native-community/async-storage": "^1.8.1",
"@react-native-community/masked-view": "^0.1.7",
"@react-navigation/native": "^5.0.9",
"@react-navigation/stack": "^5.1.1",
"axios": "^0.19.2",
"native-base": "^2.13.8",
"react-native-elements": "^1.2.7",
"react-native-gesture-handler": "^1.6.0",
"react-native-reanimated": "^1.7.0",
"react-native-safe-area-context": "^0.7.3",
"react-native-screens": "^2.2.0",
"react-native-vector-icons": "^6.6.0",
"react-navigation": "4.0.10",
"react-navigation-stack": "1.10.3",
"socket.io-client": "^2.3.0"
}
```

REFAQI

Para realizar a codificação, também foi utilizado o software Visual Studio Code. Neste processo, foram criadas todas as interfaces do aplicativo.

O projeto está configurado para iniciar no arquivo index.js. Este arquivo, direciona para o arquivo de rotas (routes.js), que permitirá toda a navegação no aplicativo e direcionará diretamente para a tela de login.

routes.js

```
import {createSwitchNavigator, createAppContainer} from "react-
navigation";
import { createStackNavigator } from 'react-navigation-stack';

import Login from './pages/login'; import
Inicio from './pages/inicio';
import SelecionarCliente from './pages/selecionarCliente'; import
GerarComanda from './pages/gerarComanda';

const Routes = createAppContainer(
    createSwitchNavigator({ //Todas as rotas da aplicação devem esta
r aqui dentro       Login,
      App: createStackNavigator({
        Inicio, //Esta tela mostra todos os procedimentos em aberto
SelecionarCliente, //Permite selecionar um cliente para cria r uma
comanda
        GerarComanda //Esta tela exibe a comanda do cliente, permite
 adicionar mais serviços e permite encerrá-la
      })
    })
  );
  export default Routes;
```

Figura 6 – Tela de login

Figura 7 – Verificação de senha

REFAQI

Se o usuário digitar um login e senha válido, e se o usuário não estiver como status “bloqueado” no banco de dados, será retornado um objeto com algumas informações do usuário, juntamente com o token gerado. Estes dados serão gravados no banco de dados local do dispositivo.

REFAQI

```
this.setState({messageLogin: 'Acessando...'});
   const { login } = this.state; //busca o valor do username que está no
state   const { senha } = this.state;
   if (!login.length){     this.setState({messageLogin: 'Você
não digitou o Login'});     return; //se o username estiver
vazio, pára a execução aqui
  }   if (!senha.length){     this.setState({messageLogin:
'Você não digitou a Senha'});     return; //se a senha estiver
vazia, pára a execução aqui
  }    const response = await api.get('authenticate/'+login+'/'+senha); //Faz a
requisi ção para verificar o login   const usuario_api = response.data; //Gera um
json apartir do objeto retornado
   if(!usuario_api.token){     this.setState({messageLogin: 'Erro:
'+response.data}); //Se não retornar um to ken, significa que deu problema no
login     return;
  }    await AsyncStorage.setItem("@app:id_usuario",
usuario_api.user.id_usuario.toStri ng());
  await AsyncStorage.setItem("@app:usuario", usuario_api.user.usuario);
await AsyncStorage.setItem("@app:tipo", usuario_api.user.tipo);
  await AsyncStorage.setItem("@app:id_colaborador", usuario_api.user.id_colaborado
r.toString());
  await AsyncStorage.setItem("@app:token", usuario_api.token);
  if(usuario_api.token){ //Se retornou token, significa que o login foi feito

this.props.navigation.navigate("App");
  }
```

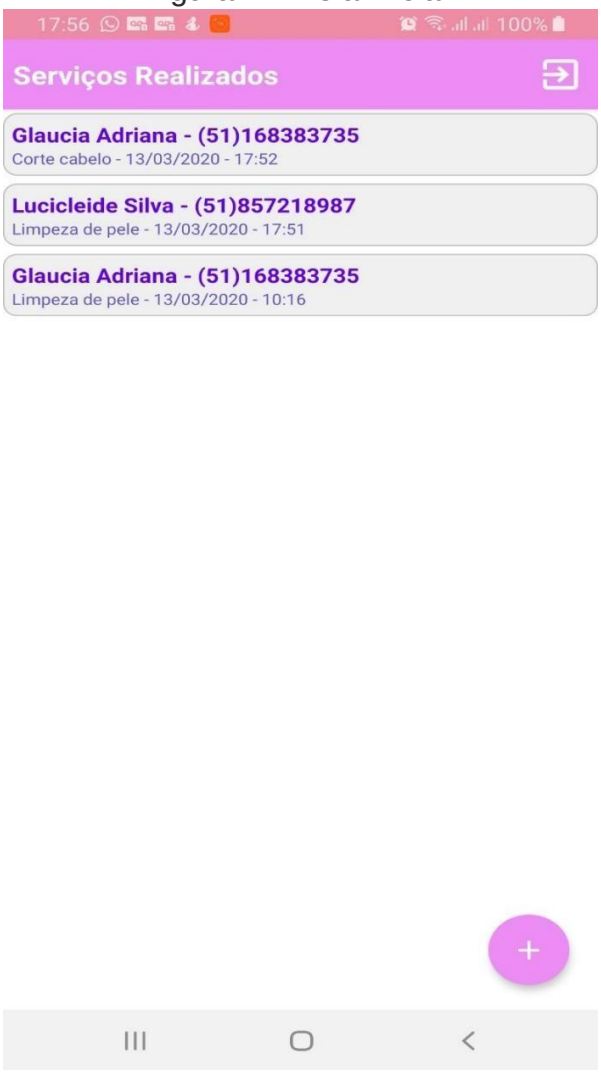
Com estes dados gravados localmente, na próxima vez que o usuário abrir o aplicativo, ele não precisará digitar suas credenciais novamente (exceto se ele tenha encerrado a sessão através do botão sair). Esta verificação é realizada através do seguinte código:

REFAQI

```
async componentDidMount() { //Esta função é chamada assim que esta tela é montada   const
id_usuario = await AsyncStorage.getItem("@app:id_usuario"); //Verifica se existe um
usuário no db local   if(id_usuario){ //Caso exista...
    this.props.navigation.navigate("App"); //Direciona para a as rotas protegidas
}
}
```

A tela inicial do App exibe todos os serviços realizados que ainda não foram pagos, ou seja, que constam em uma comanda.

Figura 7 – Tela inicial



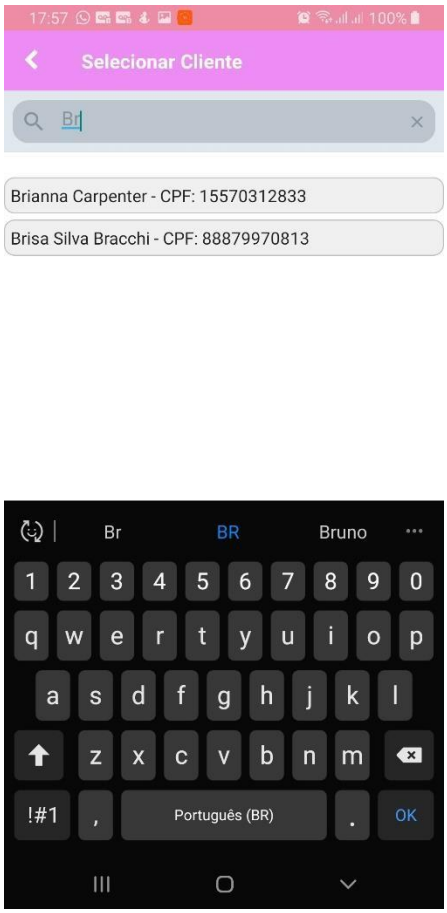
Para que o usuário possa registrar algum novo serviço na comanda de um cliente, pode clicar em algum serviço já realizado para este cliente. Caso este cliente ainda não tenha uma comanda ativa, basta tocar no botão inferior, que levará diretamente para a tela com todos os clientes.

Nesta tela será possível realizar uma busca, através de um campo de pesquisa, localizado na parte superior.

REFAQI

Figura 8 – Lista de clientes

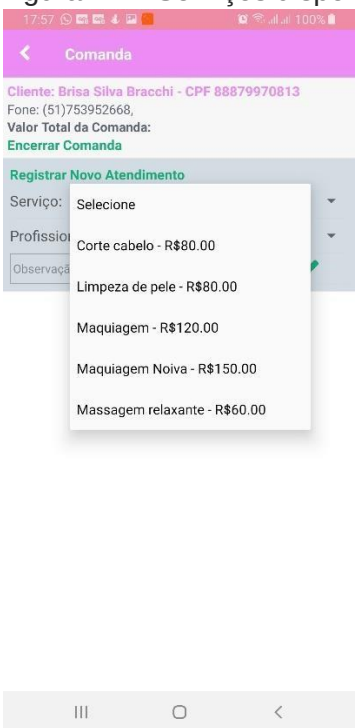
Figura 9 – Pesquisando um cliente



Após selecionar um cliente, será aberta a comanda. Nesta tela, serão exibidos todos os serviços que estão sendo cobrados.

Figura 10 – Comanda

Figura 11 – Serviços disponíveis



Na parte superior da comanda, são exibidos os dados do cliente para conferência.

Logo abaixo, existe a opção “Encerrar comanda”, que só é permitida quando solicitada por um usuário do tipo “caixa”.

Na sessão “Registrar novo atendimento”, exibe todos os serviços disponíveis para ser incluídos na comanda, juntamente com seu valor. Logo abaixo, permite selecionar o profissional que realizou o serviço. Se o serviço foi realizado pelo próprio usuário (maioria dos casos), basta prosseguir com uma observação (opcional) e clicar em ***V***.

O usuário será redirecionado para a tela de Início, que já estará exibindo o serviço gerado. No mesmo momento que foi gerado o serviço, todos os outros dispositivos que estiverem em uso do software, terão suas telas atualizadas automaticamente, ou seja todas as inclusões, e também encerramentos de comandas, serão sincronizadas em tempo real.

Caso o usuário deseje deslogar, basta selecionar o ícone em formato de porta, no canto superior da tela Inicial.

5. CONSIDERAÇÕES FINAIS

Como o software foi desenvolvido com o armazenamento dos dados de forma manual, devido ao curto prazo de desenvolvimento, futuramente ele possuirá as telas de cadastro de clientes, colaboradores, usuários e serviços. Além disto, ele poderá oferecer o envio da fatura para o e-mail do cliente, ao realizar o fechamento desta. Outro recurso importante, seria a notificação do tipo Push, que aparecerá no aparelho do usuário caixa, a cada novo serviço realizado por alguma profissional.

Contudo, os resultados obtidos do projeto BillSystem se mostraram satisfatórios para o segmento na qual ele se destina. Nesta ocasião, o ambiente na qual ele foi projetado e implantado, teve uma melhora significativa em questão de organização de suas cobranças e uma maior confiabilidade dessas informações. O sistema se mostrou prático e de rápido acesso, já que não existe mais a necessidade da profissional se dirigir até o caixa para informar o serviço realizado e valor.

REFERÊNCIAS

BELL, D. Noções básicas sobre UML: uma introdução à linguagem de modelagem unificada. Disponível em: . Acesso em: 16 set. 2018. BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. UML Guia do Usuário. 2. ed. [s.l.] Campus, 2006.

ELMASRI, R.; NAVATHE, S. Sistemas de Banco de Dados. [s.l.] Person, 2005.

GERHARDT, T. E.; SILVEIRA, D. T. Métodos de Pesquisa. [s.l.] UFRGS, 2009.

LEITÃO, A. M. Linguagem de Programação. Disponível em: . Acesso em: 7 set. 2018.

OLIVEIRA, M. V. Modelagem de Sistemas - Aula 1. Disponível em: . Acesso em: 15 set. 2018b.

SOMMERVILLE, I. Engenharia de Software. 9. ed. São Paulo: Person, 2011.