HELP-ME – APLICATIVO PARA LISTAGEM E CONTROLE DE ATENDIMENTO À PACIENTES EM UBS.

HELP-ME - APPLICATION FOR LISTING AND CONTROL OF CARE TO PATIENTS IN HEALTH UNITS.

Bruno Eduardo Medeiros, Gilmar Ignacio Pereira Junior

Orientador: Prof. Diego Fiori de Carvalho

[1]Curso de Sistemas de Informação – Centro Universitário UNIFAFIBE

Bebedouro –SP – Brazil

bruno.emedeiros99@gmail.com, gilmar.ignaciopereirajunior@gmail.com, si@unifafibe.com.br

Resumo.

Helpme é uma aplicação mobile voltada para auxiliar profissionais de saúde no atendimento de pacientes. Tem o objetivo de ser uma ferramenta servindo como prontuário digital do paciente para uso da equipe de saúde, acessando os dados de maneira rápida e simples, desde a passagem pela triagem, encaminhamento para demais setores e alterações de informações. Com fim de resolver os problemas de organização, agilidade e comunicação entre a equipe interna, sendo uma ferramenta pessoal de fácil acesso a informações dos pacientes em tempo real, auxiliando no controle e melhorando a qualidade do atendimento.

Alguns dos principais problemas que afetam a qualidade do atendimento é a ineficiência do sistema organizacional, atritos entre a equipe e alto fluxo de pacientes como o Cardiologista Dr. José Aldair Morsch em matéria para o site de telemedicina da Morsch, comenta sobre a superlotação em hospitais: "Falhas na gestão, falta de triagem dos pacientes e de infraestrutura estão entre as principais causas desse cenário." E cita algumas consequências: "Aumento no tempo de espera pelo primeiro atendimento, sobrecarga de profissionais de saúde, equipes de saúde exaustas e desmotivadas".

Para a manipulação dos dados dos pacientes será feito o uso de um banco de dados em tempo real garantindo a interação instantânea das informações. Para a parte de controle administrativo e de usuário, é realizado um modelo de dupla verificação de identidade para garantir a segurança das informações manipuladas.

Palavras-chave: Pacientes, Prontuário Digital, Aplicativo, Mobile, Médico.

Abstract.

Helpme is a mobile application aimed at assisting healthcare professionals in patient care. It aims to be a tool serving as the patient's digital medical record for use by the health team, accessing data quickly and simply, from passing through screening, forwarding to other sectors and changing information. In order to solve the problems of organization, agility and communication among the internal team, being a personal tool with easy access to patient information in real time, assisting in the control and improving the quality of care.

Some of the main problems that affect the quality of care is the inefficiency of the organizational system, friction between the team and high flow of patients such as Cardiologist Dr. José Aldair Morsch in matters for the Morsch telemedicine website, comments on overcrowding in hospitals : "Management failures, lack of patient screening and infrastructure are among the main causes of this scenario." And he cites some consequences: "Increased waiting time for the first appointment, overload of health professionals, exhausted and unmotivated health teams".

For the manipulation of patient data, a real-time database will be used, ensuring instant interaction of information. For the administrative and user control part, a double identity verification model is performed to guarantee the security of the manipulated information.

Keywords: Patients, Digital Health Record, Application, Mobile, Doctor.

1. INTRODUÇÃO

A tecnologia está cada vez mais presente em todos os aspectos na área da saúde, sempre visando aumentar a qualidade e eficiência dos serviços, diminuindo os erros e otimizando processos. Porem a implementação de tais tecnologias vem com desafios, que em sua maior parte dizem respeito a adaptação da equipe, em matéria para Cadernos de Saúde Pública (CSP) é citado sobre o tema “uma quantidade grande e crescente de novas tecnologias tem sido incorporada nos sistemas de saúde, implicando novos desafios para todas as partes interessadas. Tais desafios incluem, por exemplo, questões éticas relativas ao uso seguro dos dados relativos à saúde (big data), novos padrões para a aprovação de inovações que combinam diferentes tecnologias, impacto orçamentário da incorporação de tecnologias de alto custo e questões de equidade relacionadas à cobertura e ao acesso de grupos vulneráveis aos serviços de saúde.” (Silva, Hudson Pacífico da; Elias, Flavia Tavares Silva .2018).

Para unidades de saúde que atendem um fluxo quase incessante de pacientes, se acentuam problemas de comunicação entre a equipe, o que resulta em atrasos na execução de tarefas, além da queda da qualidade de serviço, como citado na Revista Brasileira de Enfermagem em matéria de vários anos atrás problemas enfrentados ainda hoje, mais de 40 anos depois, alguns problemas relatados foram “ Uma enfermeira, chefe do serviço de Enfermagem de um hospital, informou que, em virtude de inúmeros desentendimentos havidos com médicos, adotara a sistemática de só anotar a medicação prescrita, os dados do pulso e da pressão arterial. Relatou que esses desentendimentos eram decorrentes de anotações do pessoal de enfermagem sobre reações observadas, queixas de dores ou outros sintomas subjetivos dos pacientes. Em seguida a essas anotações, o pessoal fazia constar também a comunicação feita ao próprio médico do paciente ou ao plantonista. Entretanto, por sobrecarga de outros setores do hospital (Pronto Socorro), principalmente à noite, tais chamados, frequentemente, não eram atendidos. Em outro dia, o médico ao passar visita e verificando tal anotação, simplesmente inutilizava a folha de "Prescrição Médica e Relatório de Enfermagem", não se incomodando até em transcrever a prescrição anterior, mas exigindo que o pessoal de enfermagem também refizesse as anotações, retirando aquelas que pudessem determinar glosa da conta hospitalar por parte da instituição conveniente.”(Oguisso, Taka ; Schmidt, Maria José . 1976).

Helpme foi idealizado pensando nessas dificuldades enfrentadas todos dias em centros de saúde por todo Brasil, pensado para ser um aplicativo simples com consultas rápidas para manipulação de dados relevantes, ele objetiva resolver problemas de comunicação entre a

equipe, resolver falhas de organização como poucos pontos de acesso ao sistema interno, focando na praticidade e usabilidade.

2. REFERENCIAL TEÓRICO

2.1. Banco de Dados

Segundo Date a definição de sistema de banco de dados é:

> ...Um sistema computadorizado de manutenção de registros; em outras palavras, é um sistema computadorizado cuja finalidade geral é armazenar informações e permitir que os usuários busquem e atualizem essas informações quando as solicitar. (DATE, C. J. 2004, pg 40).

Tendo definido o conceito base, é necessário pontuar os modelos mais comumente abordados, não excluindo a existem de outros modelos independentes ou derivantes em qualquer grau, os relacionais e não relacionais.

2.1.2. Relacionais

Refere-se ao modelo relacional de dados que tem por base a *SQL* ("Linguagem de Consulta Estruturada"), Carla Geovana do N. Macário e Stefano Monteiro Baldo explicam o conceito aplicado a esse modelo:

> No modelo relacional a principal construção para representação dos dados é a relação, uma tabela com linhas não ordenadas e colunas. Uma relação consiste de um esquema é de uma instância. O esquema especifica o nome da relação e o nome e o domínio de cada coluna, também denominada atributo ou campo da relação. (MACÁRIO, Carla G. do N., BALDO, Stefano Monteiro. 2005, pg 3).

Este sendo o mais comumente utilizado pelo mercado que apontam alguns dos motivos tais como: similaridade entre *SGBDs* que o usam como padrão, capacidade de dados ao se encaixar desde sistemas computacionais simples até aqueles com centenas de tabelas e relacionamentos e ser fortemente estruturado.

2.1.3. PostgreSQL.

PostgreSQL é um sistema de gerenciamento de banco de dados relacional baseado no *POSTGRES* em sua versão 4.2 desenvolvido pelo Departamento de Ciência da Computação da Universidade da Califórnia.

> Devido à sua licença liberal, o *PostgreSQL* pode ser utilizado, modificado e distribuído por qualquer pessoa para qualquer finalidade, seja particular, comercial ou acadêmica, livre de encargos. (*PostgreSQL Global Development Group*, 2006, Prefacio).

Por dar suporte a grande parte do padrão *SQL* e pela licença livre, é muito

utilizado focado como substituto de outros *SGBDs* do mercado, apresentando muitas vezes maior performance e liberdade no gerenciamento e configurações.

2.2 NoSQL

NoSQL se refere à "não apenas *SQL*" cuja história e arquitetura são

exploradas por Sadalag e Fowler sobre o tema:

> ...Para começar, há o fato óbvio de que bancos de dados NoSQL não utilizam SQL.
>
> ...Os bancos de dados NoSQL atuam sem um esquema, permitindo que sejam adicionados, livremente, campos aos registros do banco de dados, sem ter de definir primeiro quaisquer mudanças na estrutura. Isso é especialmente útil ao lidar com dados não uniformes e campos personalizados. (SADALAG, Pramod J., FOWLER, Martin. 2013, pg 34,35).

Sendo evidente a grande diferença entre bancos de dados relacionais e

NoSQL, onde se tem por princípio à utilização "descontruída" de modelos com foco na adaptabilidade e performance. Tendo como base a modelo chave/valor que seguindo a mesma lógica que "*collection*" nas linguagens de programação é o conjunto de pares chave/valor, isto é, ter uma chave e logo a seguir um valor para ela. Após o surgimento desse modelo outros surgiram como variantes e derivações, dentre eles os por documento e coluna.

2.2.1 Por documento

Como já citado os bancos *NoSQL* baseados em documentos segue o

conceito que todos os dados para uma única entidade podem ser armazenados como um documento, e os documentos podem ser armazenados juntos, Cardoso em sua tese de mestrado escreve:

> ...Estes documentos normalmente são decompostos em um identificador e em um valor. Este modelo de dados codifica os documentos no formato em cima descrito com *XML, JSON, BSON* e os formatos binários mais conhecidos como *PDF, DOC, XLS*, etc. (CARDOSO, Ricardo Manuel Fonseca. 2012, pg 19).

2.2.2 Firebase

Firebase é uma plataforma de desenvolvimento mobile (usado também para

web) adquirida pela *Google* em 2014. Desenvolvida para ser um "*back-end*" completo e simples, disponibilizando diversos serviços para auxiliar no desenvolvimento e gerenciamento de aplicativos. Laurence Moroney o descreve como:

> É um aprimoramento, oferecendo serviços comuns de que você pode precisar - como *back-end* de banco de dados, autenticação segura, mensagens e muito mais. Isso poupa a necessidade de construí-los, permitindo que você se concentre no que diferencia seu aplicativo. (MORONEY, Laurence. 2017, pg 1).

2.3. Desenvolvimento mobile

O Desenvolvimento *Mobile* é a área que tem por foco a criação de aplicações

para *smartphones* e dispositivos moveis de todos os tipos, sendo os mais comuns aqueles que usam os sistemas operacionais *IOS*(*Apple*) e *Android* (*Google*)

As grandes empresas de tecnologia passaram a desenvolver essas aplicações,

aumentando assim a facilidade de acesso por estarem, literalmente, nas mãos dos usuários a qualquer momento.

2.3.1 Desenvolvimento híbrido

Aplicações hibridas são aquelas desenvolvidas em uma plataforma, linguagem,

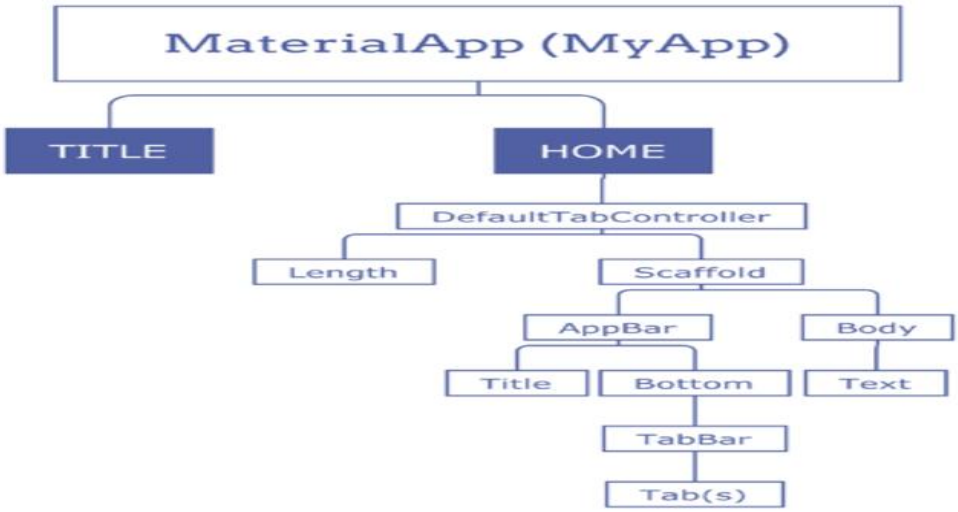
framework ou *SDK* onde através de um único código é possível geral aplicações para múltiplos sistemas operacionais mobile, tais como *IOS*, *Android* entre outros. Essas aplicações são focadas no tempo gasto em desenvolvimento, a não necessidade de uso de recursos exclusivos dos sistemas operacionais, eliminação de divergências de funcionalidades, *layout* entre outros pontos de versões de sistemas diferentes e redução do tempo e custo de manutenção.

2.3.2 Flutter

Flutter é um *SDK* do *Google* para construir aplicações hibridas compiladas para

mobile, web e desktop. Utiliza à linguagem *Dart* (*Google*), diferentes de outras linguagens hibridas que atuam dentro de uma "*bridge*" entre a UI e o dispositivo, *Flutter* fica na camada do UI e não chama os componentes nativos do sistema operacional, o que aumenta a performance e fluidez a nível de um aplicativo desenvolvido nativo.

O *Flutter* possui os "*widgets*", um *widget* é uma árvore que pode conter um ou mais filhos(*widgets*), e esses filhos são construídos conforme a construção desta árvore.



Fonte: *Tableless* (acesso em 29 de abril de 2020).

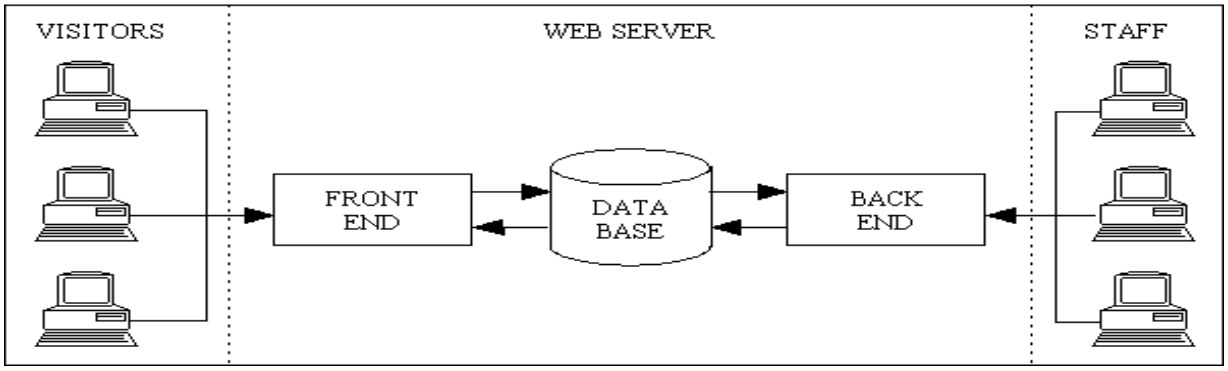
Figura 01: Modelo de arvore de *Widgets*.

2.4 Desenvolvimento de Back-end

2.4.1 Server-side

Back-End é o conceito do que "vem por trás" de uma aplicação, diferentemente

do *front-end(client-side*) onde o usuário acessa diretamente um serviço, site ou aplicação, o *back-end* trabalha na maior parte dos casos fazendo a ponte entre os dados e/ou requisições da aplicação ao banco de dados e vice-versa, sempre aplicando as regras de negócio, validações, criptografias e segurança de um ambiente onde o usuário final não tenha acesso.



Fonte: Oficina da *Net* (acesso em 29 de abril de 2020)

Figura 03: Arquitetura *Back-end* x Arquitetura *Front-end.*

2.4.2 API

"*Application Programming Interface*" ou Interface de Programação de

Aplicativos é uma interface com conjuntos de rotinas e padrões com a finalidade de realizar a comunicação de diferentes programas através de protocolos *web*, o mais comum sendo o *HTTP*, além de proporcionar a integração entre sistemas que possuem distintas(os) linguagem, sistemas de banco de dados dentro vários outros tipos de maneira ágil e segura.

2.4.3 API Rest

REST significa" *Representational State Transfer*" (Transferência de Estado

Representacional). Resumidamente, o *REST* consiste em padrões que, quando seguidas, permitem a criação de um projeto com interfaces bem definidas, permitindo o uso do protocolo

HTTP de uma forma muito mais próxima da nossa realidade, dando sentido para as requisições. Através dessa padronização surgiram os métodos *GET, POST, PUT, PATH e DELETE* usados como referência para as requisições.

Figura 04: Arquitetura *API Rest.*

Fonte: Medium (acesso em 29 de abril de 2019)

2.5. Linguagens de programação *Back-end*(Server-side)

2.5.1. PHP

O *PHP* (*Hypertext Preprocessor*) é uma linguagem Orientada a Objetos,

open source de uso geral, muito utilizada, e especialmente adequada para o desenvolvimento web e que pode ser embutida dentro do *HTML*.

Carlos E. discorre em uma sua postagem no blog da *Hostinger*:

> trata-se de uma linguagem de script criada para comunicações do lado do servidor. Consequentemente, ela é capaz de lidar com várias funções de *backend* como coletar formulários de dados, gerenciar arquivos do servidor, modificar bases de dados e muito mais.

2.5.2. Node.js

Node.js de maneira simplificada pode-se definir como um ambiente de

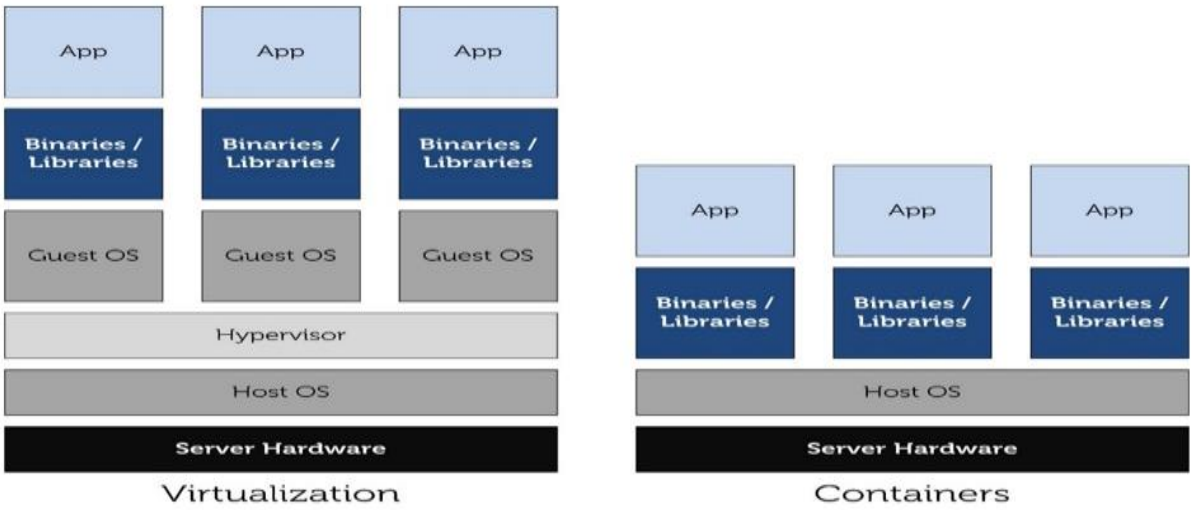
execução *Javascript* para *server-side*. O *Javascript* foi criada em 1995, e se tornou a linguagem padrão dos navegadores para desenvolvimento client-side*(front-end*), porem com o tempo e a necessidade de adaptar os professionais que usavam essa linguagem, as aplicações server-side passaram a ser implementadas, e em 2009 foi criado o primeiro ambiente de execução com este propósito: O *Node.js*. Oque significa que através dele é possível criar aplicações não dependendo de um navegador para à execução.

2.5.3. Containers

John Martin em seu livro descreve o funcionamento e arquitetura dos *Containers*:

> Diferente da virtualização de *hipervisor*, onde uma ou mais máquinas independentes são executadas virtualmente no *hardware* físico por meio de uma camada de intermediação, os contêineres são executados no espaço do usuário em cima do *kernel* do sistema operacional. Como resultado, a virtualização de *contêiner* costuma ser chamada de virtualização em nível de sistema operacional. A tecnologia de contêiner permite que várias instâncias isoladas do espaço do usuário sejam executadas em um único *host*. Como resultado de seu status como convidados do sistema operacional, os contêineres às vezes são vistos como menos explicáveis: geralmente eles só podem executar o mesmo ou um sistema operacional semelhante ao host subjacente. (MARTIN. 2016, pg 6).

Em contraste com uma virtualização através do *hipervisor*, os *containers* são uma opção mais leve com foco na performance e eficiência do serviço executado.



Fonte: Mundo Docker (acessado em 30 de abril de 2020)

Figura 05: Diferença entre virtualização e *containers*.

2.5.4 Docker

Docker é uma plataforma *Open Source* feita em *Go* uma linguagem

desenvolvida pelo *Google*, que facilita a criação e administração de ambientes isolados através de *containers*, serviços, *networks* dentre outras funcionalidades.

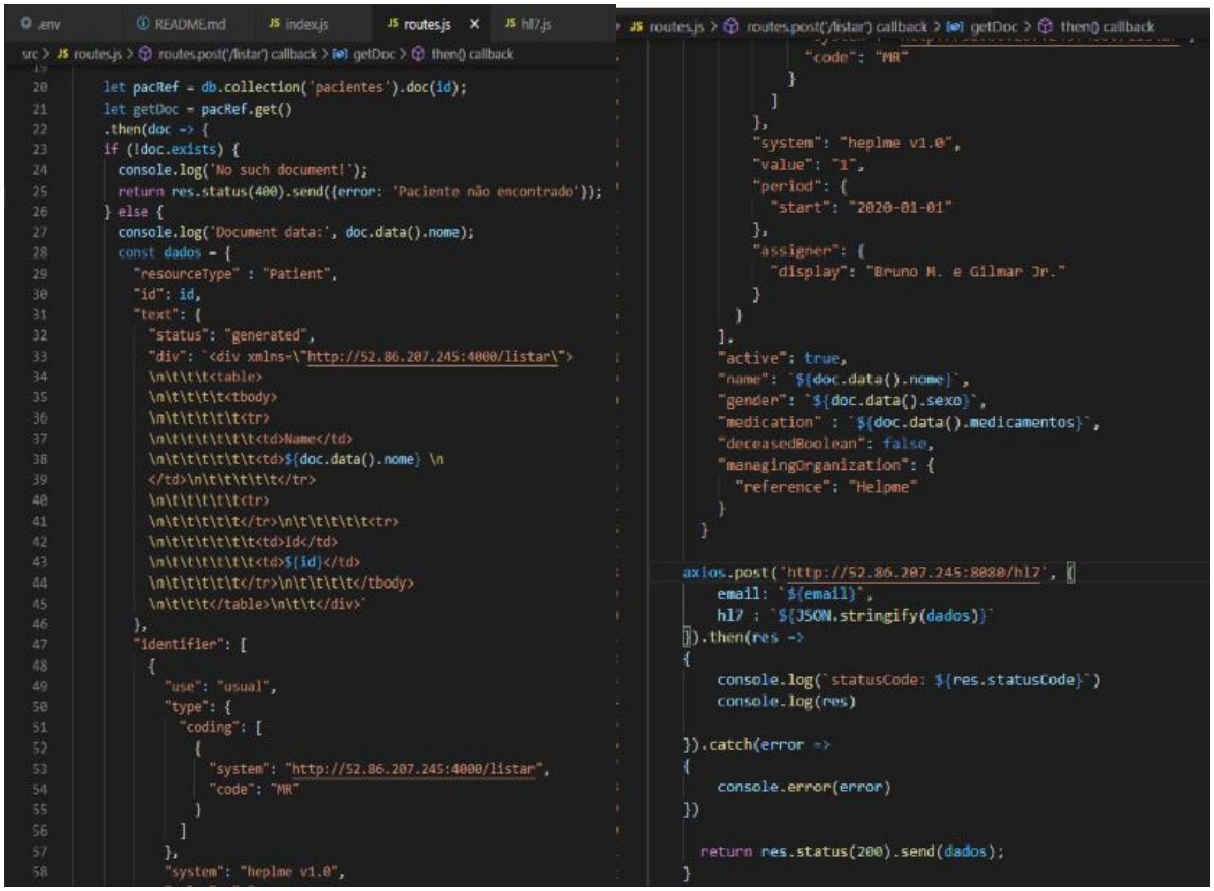
O já citado Martin pontua sobre *Docker*:

> Então, o que há de especial no *Docker*? O *Docker* adiciona um mecanismo de implantação de aplicativos no topo de um ambiente de execução de *contêiner* virtualizado. Ele foi projetado para fornecer um ambiente leve e rápido para executar seu código, bem como um fluxo de trabalho eficiente para obter esse código do seu *laptop* para o ambiente de teste e, em seguida, para produção. (MARTIN. 2016, pg 7,8).

2.5.5. HL7

Health Level 7 ou *HL7* é um conjunto de normas para a representação e

transferência de dados clínicos e administrativos entre sistemas de saúde, que englobam hospitais, clínicas e quaisquer unidade de saúde. Definindo esse padrão se estabelece a transferência uniforme e simplificada garantindo um fluxo de trabalho otimizado.



2.6. Aplicativos semelhantes

2.6.1. iClinic

O *iClinic* é um aplicativo para gestão de clínicas e consultórios, tem como

principal objetivo funcionar como prontuário eletrônico e realizar agendamento de pacientes. Mas, ele é personalizável e novas funções podem ser acopladas de acordo com a necessidade que cada paciente exige.

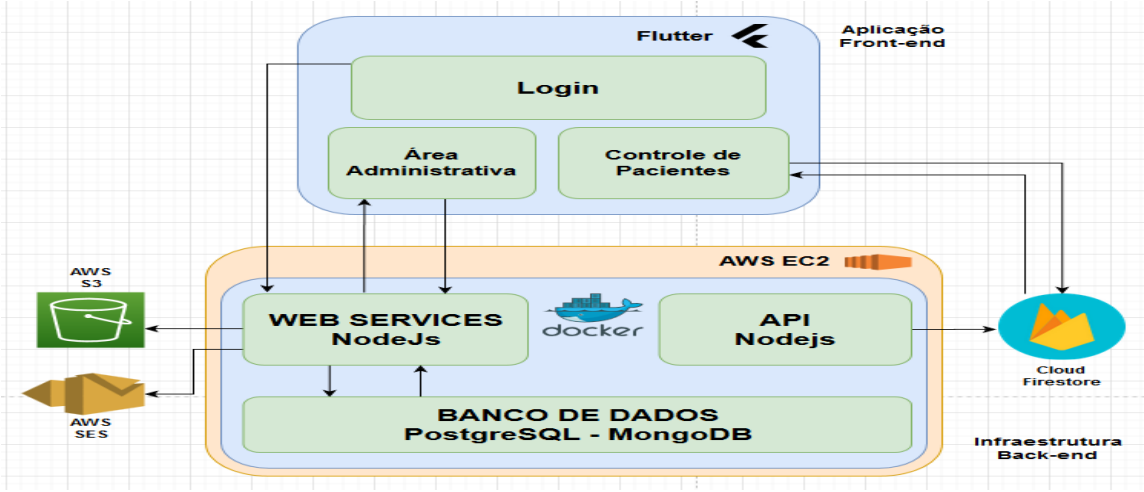
2.6.2. Consultório

Consultório é o aplicativo desenvolvido para gerência de pacientes em clínicas

de todos os tipos oferecendo suporte as mais diversas tarefas, tais como atualização e cadastro prontuários e consultas.

3. METODOLOGIA

Para o desenvolvimento da aplicação é montado um modelo de arquitetura com infraestrutura hospedada em nuvem onde uma rede de containers engloba os serviços de bancos de dados, *web services* e *API* de integração. As informações dos pacientes ficaram gravadas em um banco separado para tratamento em tempo real. À aplicação *mobile* utiliza a linguagem de programação orientada a objetos *Dart* com o *framework Flutter* aplicando design *pattern Flutter Modular* e gestão de estado com *MobX*.

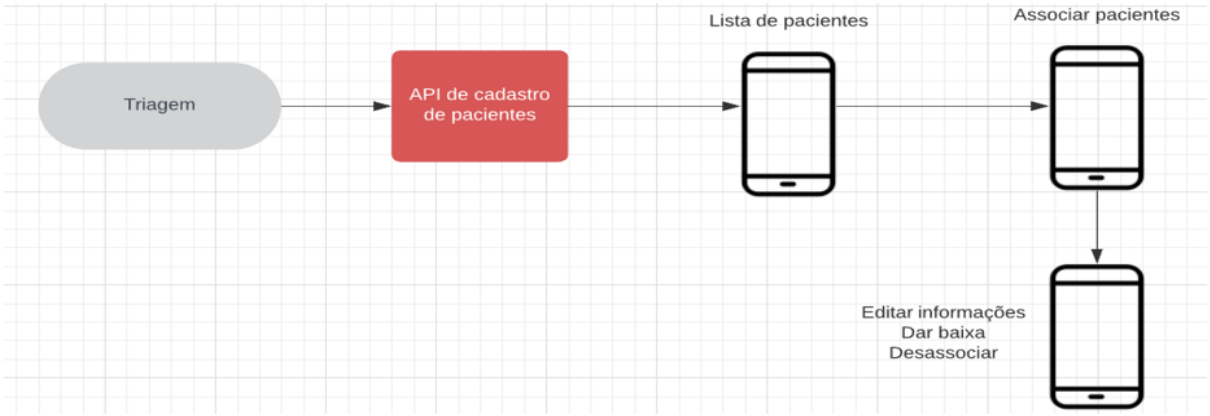


Fonte: Autoral (2020).

Figura 06: Modelo de arquitetura da aplicação

3.1 HELPME

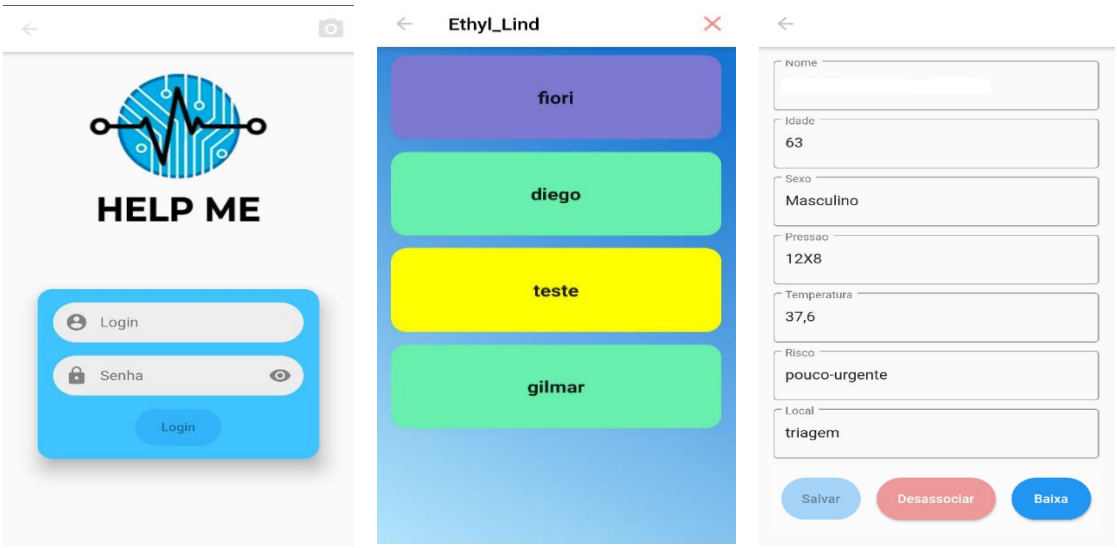
Fluxo normal de uso de aplicativo



Fonte: Autoral (2020).

Figura 07: Fluxograma da aplicação

Para os usuários padrão do sistema, ao logar será mostrado a lista geral de pacientes entrantes após a passagem pela triagem, eles ficaram exibidos por nome e cor relativa à classificação de risco. Sendo disponível a função de associar os pacientes, depois editar e salvar suas informações assim como dar baixa do mesmo ou desassociá-lo para que retorne a lista principal.

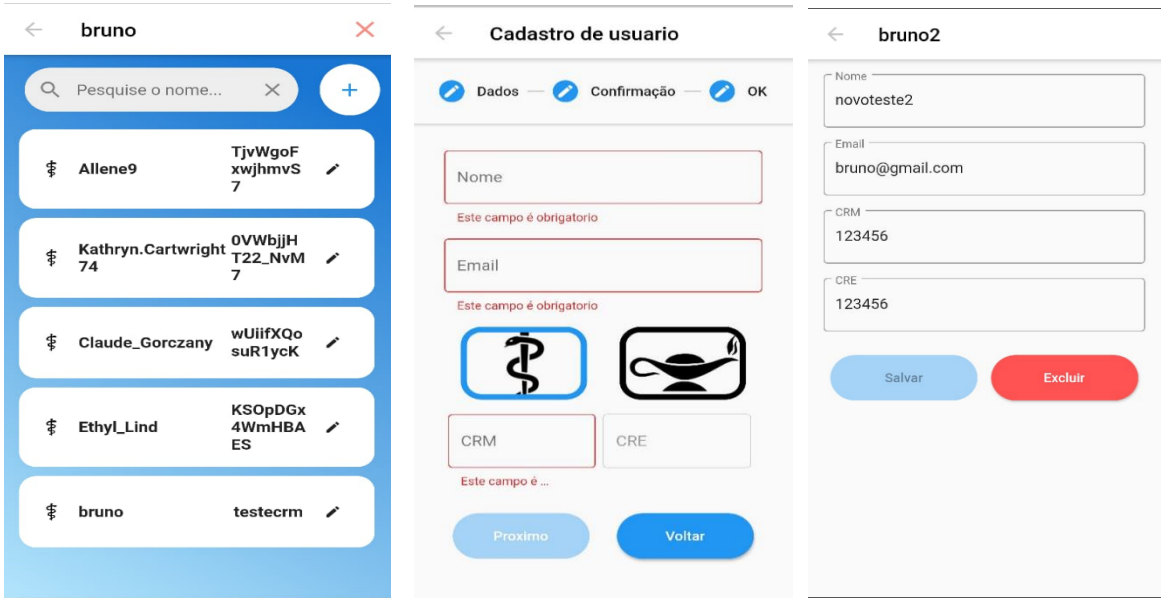


Fonte: Autoral (2020).

Figura 08: Telas de *login*, lista geral e edição de pacientes respectivamente

Para a parte administrativa é exibida uma lista de usuários, a opção de editá-los ou inativá-los e o cadastro de novos usuários que segue um modelo de dupla verificação para garantir a segurança do acesso ao aplicativo. Após a confirmação do *e-mail* do novo usuário é enviado

um código *QR* que deverá ser lido dentro do aplicativo para o primeiro acesso do usuário em questão.



Fonte: Autoral (2020).

Figura 09: Telas de administrador, novo usuário e edição de usuário respectivamente

Para a utilização do *HL7* na aplicação foi realizado um botão de exportação, onde os dados do paciente selecionado são encaminhados via *e-mail* para a clínica requerida pelo paciente, onde todo o processo é realizado sobre ampla segurança encontrada nos servidos de encaminhamento de dados.

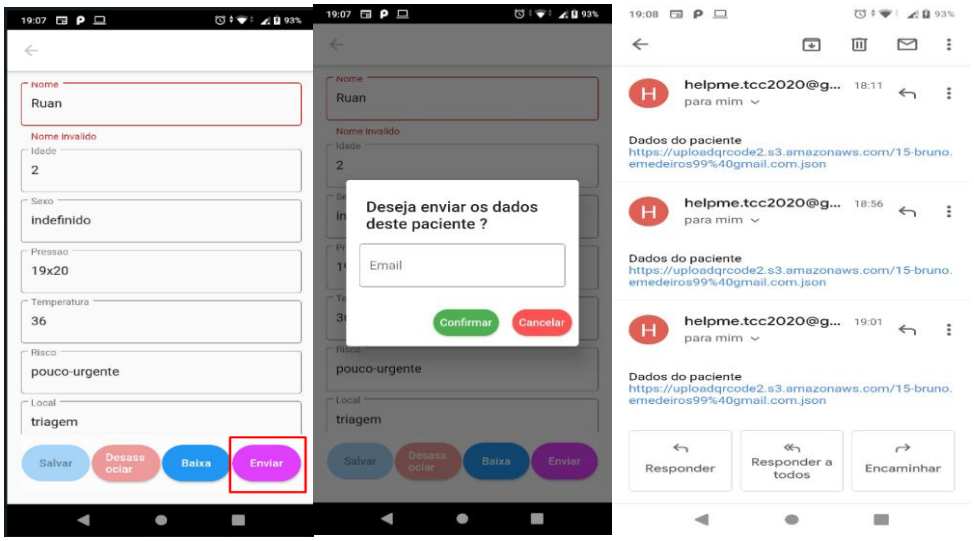


Figura 10: Telas de Paciente, Tela de encaminhamento via e-mail e e-mail recebido com as informações do paciente

Fonte: Autoral (2020).

4. DESENVOLVIMENTO

4.1. Arquitetura de desenvolvimento

O código do *Back-End* segue o padrão *MVC*, possuindo na camada "*Model*" todos os *schemas* de objetos do banco de dados *PostgreSQL* e *MongoDB* para armazenar as informações, na camada "*Controller*" um controlador para cada entidade do aplicativo e na camada "*View*" todas as rotas disponíveis para as requisições entre o *cliente-side* e o *server-side*.

Já o *front-end*, entende-se a parte da aplicação *Flutter*, segue o mesmo modelo porem orientado pelo design *Pattern Modular*, interagindo com o *back-end* através de requisições *HTTP* e diretamente com o banco *Firebase* para tratamento em tempo real das informações dos pacientes.

4.1.2 Back-end

O *Back-End* do aplicativo foi desenvolvido utilizando o *runtime* da linguagem *Java Script* chamado *Node.js*, criando o servidor para o *web service* administrativo do aplicativo.

src > JS server.js > ...

```
1  require('dotenv').config();
2
3  const express = require('express');
4  const routes = require('./routes');
5  require('./database');
6  const app = express();
7
8  app.use(express.json());
9  app.use(routes);
10 app.listen(3333);
```

Figura 11: Configuração do *Router*

Fonte: Autoral (2020).

Conseguimos observar na Figura 11 a criação do servidor com a biblioteca *Express* do próprio *Node.js*, configurando o componente *Router* para criar as rotas da *API Restful*, junto com as configurações de banco de dados, nesse caso para o *PostgreSQL* e outras configurações em um arquivo separado. Na última linha podemos ver que ele será publicado para a porta que está contido nas variáveis de ambiente do *Enviroment* para manter a segurança dos dados ou na 3333 para quando não encontrar essa porta indicada.

```
src > JS routes.js > [e] NivelController
const express = require('express');
const UserController = require('./controllers/UserController');
const LoginController =  require('./controllers/LoginController')
const NivelController = require('./controllers/NivelController');
const routes = express.Router();

routes.post('/users', UserController.store);
routes.post('/logar/:usr_id/ativar', LoginController.ativar);
routes.get('/index', UserController.index);
routes.post('/index', UserController.unico);
routes.get('/listar/:nome', UserController.listar);
routes.post('/excluir/:usr_id', UserController.excluirprovisorio);
routes.post('/alterar/:usr_id', UserController.alterar);
routes.post('/inativar/:usr_id', UserController.inativar);
routes.post('/users/:usr_id/login', LoginController.store);
routes.post('/logar', LoginController.logar);

module.exports = routes;
```

Figura 12: Configuração de Rotas apontando ao *Controller*

Fonte: Autoral (2020).

Na figura 12 está disposto o arquivo de configuração de rotas para organizar de maneira mais especifica por nomes e apontando para os arquivos de *Controller* aonde os métodos em si estão.

```
async logar(req, res)
{
    try{
        const { login} =  req.body;
        var { senha } = req.body;
        senha = await criptografar(senha);

        const novo = await Login.findOne({
            where: {
                [Op.and]: [
                    { login: login },
                    { senha: senha },
                ]
            }
        });
```

Figura 13: Método de logar no aplicativo

Fonte: Autoral (2020).

```
        if(!novo)
        {
            return res.status(400).json({ error: 'Login ou senha estão incorretos'});
        }

        const user = await User.findByPk(novo.usr_id);

        if(!user){

            return res.status(400).json({ error: 'Usuario nao encontrado'});
        }

        return res.status(200).json(user);
    }
    catch(err){
        return res.status(400).json({ error: 'Usuario nao encontrado'});
    }

}
```

Figura 14: Método de logar no aplicativo

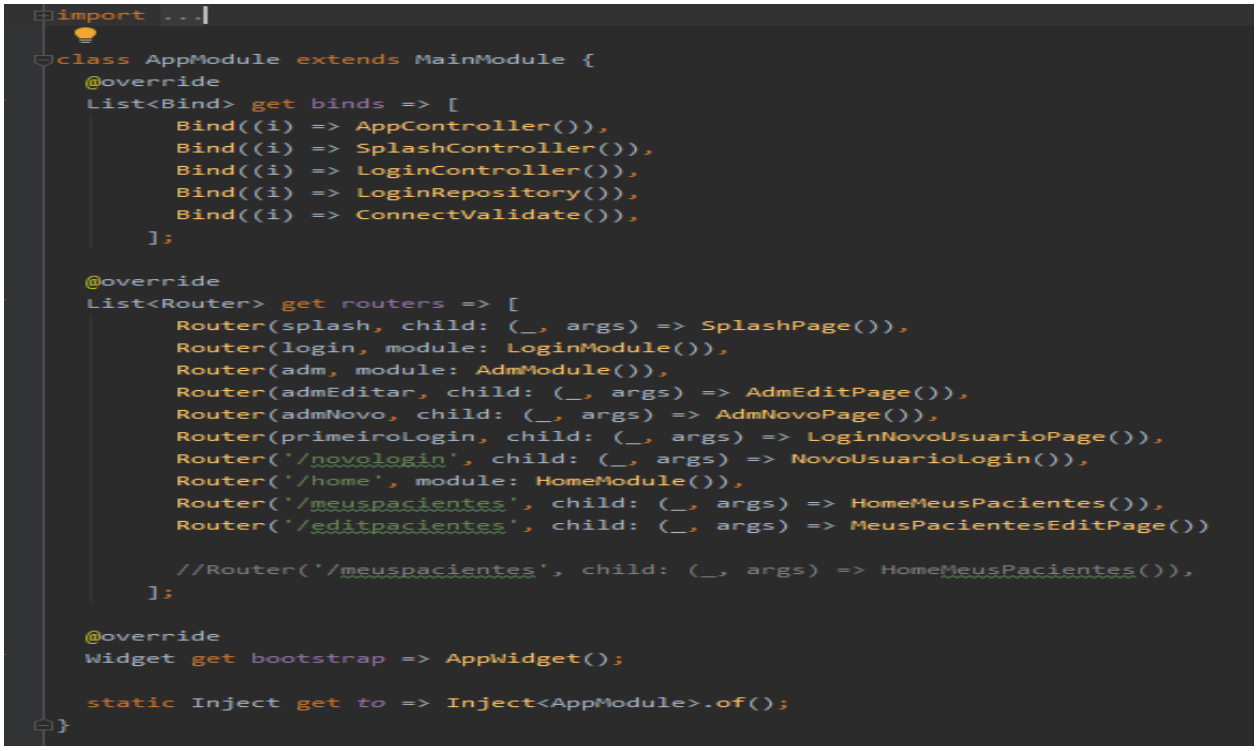
Fonte: Autoral (2020).

As figuras 13 e 14 mostram o método usado para logar no aplicativo, fazendo uso do *ORM Sequelize* para realizar as consultas no banco *PostgreSQL*, adicionando uma camada de segurança e utilizando a validação em etapas para em acaso de algum erro o mesmo ser retornado diretamente.

Figura 15: Métodos da tela *Home*

Fonte: Autoral (2020).

A figura 15 mostra a parte da tela de *home* do usuário padrão onde efetivamente são construídos os "*cards*" listando apenas os pacientes ativos no sistema.

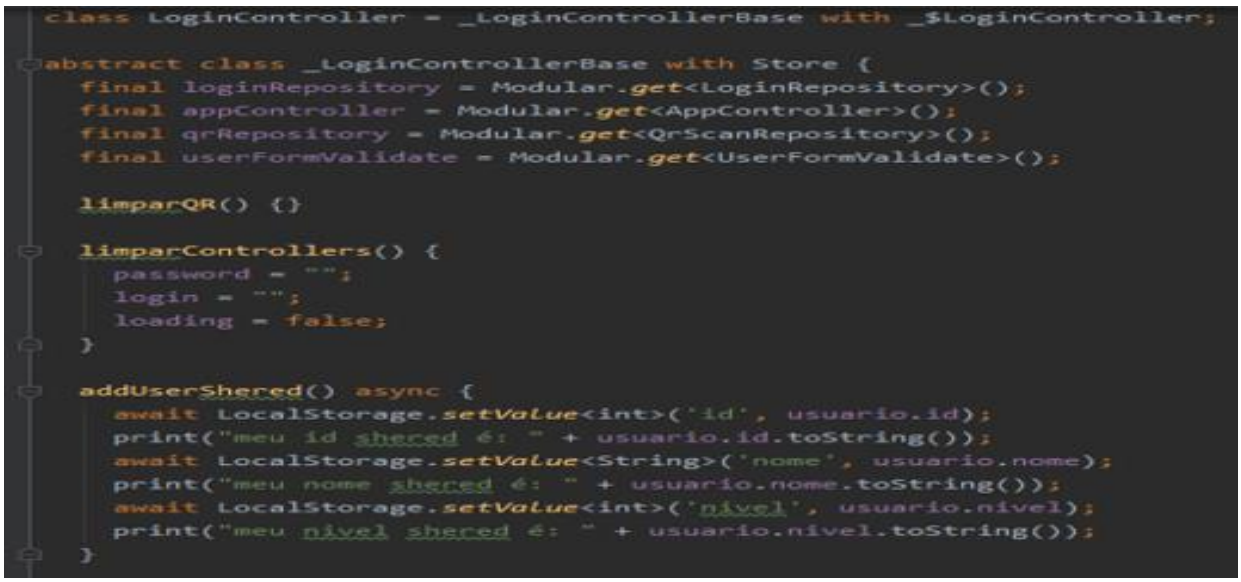


```
import ...

class AppModule extends MainModule {
  @override
  List<Bind> get binds => [
        Bind((i) => AppController()),
        Bind((i) => SplashController()),
        Bind((i) => LoginController()),
        Bind((i) => LoginRepository()),
        Bind((i) => ConnectValidate()),
      ];

  @override
  List<Router> get routers => [
        Router(splash, child: (_, args) => SplashPage()),
        Router(login, module: LoginModule()),
        Router(adm, module: AdmModule()),
        Router(admEditar, child: (_, args) => AdmEditPage()),
        Router(admNovo, child: (_, args) => AdmNovoPage()),
        Router(primeiroLogin, child: (_, args) => LoginNovoUsuarioPage()),
        Router('/novologin', child: (_, args) => NovoUsuarioLogin()),
        Router('/home', module: HomeModule()),
        Router('/meuspacientes', child: (_, args) => HomeMeusPacientes()),
        Router('/editpacientes', child: (_, args) => MeusPacientesEditPage())

        //Router('/meuspacientes', child: (_, args) => HomeMeusPacientes()),
      ];

  @override
  Widget get bootstrap => AppWidget();

  static Inject get to => Inject<AppModule>.of();
}
```

Figura 16: *Design Pattern*

Fonte: Autoral (2020).

A figura 16 mostra dentro da aplicação *Flutter* é usado um *design pattern* chamado *Modular*, aplicando os conceitos de modularidade, injeção de dependências e rotas internas entre arquivos como mostrado na figura 16 acima, onde o mesmo se refere ao modulo principal do aplicativo.



```
class LoginController = _LoginControllerBase with _$LoginController;

abstract class _LoginControllerBase with Store {
  final loginRepository = Modular.get<LoginRepository>();
  final appController = Modular.get<AppController>();
  final qrRepository = Modular.get<QrScanRepository>();
  final userFormValidate = Modular.get<UserFormValidate>();

  limparQR() {}

  limparControllers() {
    password = "";
    login = "";
    loading = false;
  }

  addUserShered() async {
    await LocalStorage.setValue<int>('id', usuario.id);
    print("meu id shered é: " + usuario.id.toString());
    await LocalStorage.setValue<String>('nome', usuario.nome);
    print("meu nome shered é: " + usuario.nome.toString());
    await LocalStorage.setValue<int>('nivel', usuario.nivel);
    print("meu nivel shered é: " + usuario.nivel.toString());
  }
```

Figura 17: *Mobx* para a gestão de componentes

Fonte: Autoral (2020).

A figura 17 mostra além do *design pattern* é utilizado uma biblioteca chamada *Mobx* para gestão do estado dos componentes da aplicação, podendo ser aplicado a qualquer *widget* da aplicação, realizando funções reativas automaticamente para mudar características deles ou até executar funções. A figura em questão mostra uma parte do *Controller* utilizado no modulo de *Login*.

5. RESULTADOS

Os resultados obtidos após testes em campo em uma clínica radiológica na cidade de Bebedouro – SP, mostram a eficácia da aplicação como plataforma comum de acesso a dados que necessitariam de um ponto de acesso ao sistema interno e buscas aplicando diversos filtros. Demonstrando-se como uma facilitadora nos processos, reduzindo o tempo necessário para executá-los.

Testes foram realizados com 35 pacientes demonstrando interações instantâneas, limitando o tempo de uso da aplicação ao tempo do paciente dentro da clínica radiológica. Mostrando-se capaz de realizar a interação concreta sobre os objetivos com os usuários, testes futuros da aplicação serão submetidos a cenários cada vez mais complexos a fim de maximizar sua utilidade, usabilidade e funcionalidades

Após os testes foi feito uma verificação com a equipe, colocando em questão duas perguntas, onde na primeira foi questionado se realizaria novamente a utilização do aplicativo e na segunda questão se o aplicativo é acessível, formando o seguinte resultado:

Grupo Amostral: 11 pessoas.

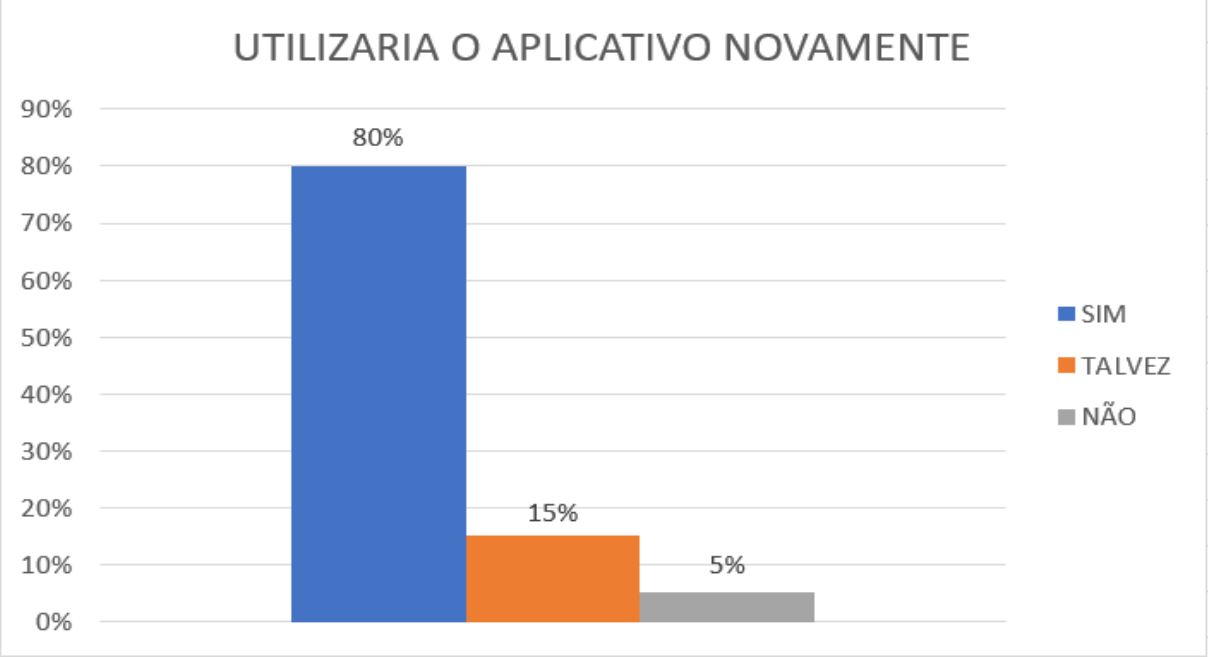


Figura 18: Gráfico de utilização do Aplicativo

Fonte: Autoral (2020).

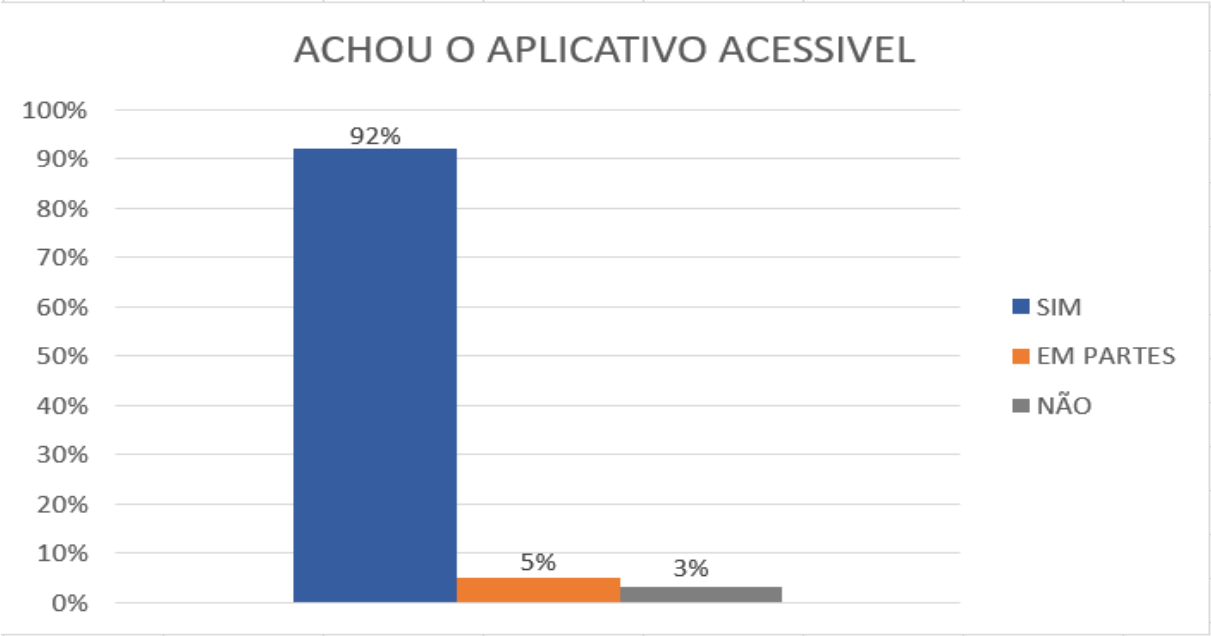


Figura 19: Gráfico de acessibilidade do aplicativo

Fonte: Autoral (2020).

As figuras 18 e 19 representam os gráficos gerados a partir dos resultados obtidos nos testes necessários.

6. CONSIDERAÇÕES FINAIS

Com base nas informações coletadas e em todo o processo de validação, foi constatado que houve um grande auxilio da aplicação no cotidiano da equipe, acima de tudo no tempo decorrido para consultar as informações do paciente, além disso, houve uma melhor interação da equipe para o controle de informações e localização do paciente. Como trabalhos futuros podemos sugerir a criação de mais dados para o cadastro do paciente, além de existir uma integração direta no processo de transmissão de dados que existe pelo *HL7* hoje presente na aplicação.

7. REFERÊNCIAS BIBLIOGRÁFICAS

MORSCH, José Aldair. **Comunicação Médico Paciente: Importância, Benefícios e Desafios**. Acessado em: 17 de out. de 2020..

LENON. **Node.js – O que é, como funciona e quais as vantagens** Disponível em:

< https://www.opus-software.com.br/node-js/> acessado em 29 de abril 2020.

TURNBULL, James. The Docker Book: Containerization is the new virtualization. James Turnbull, 2014. Acessado em: 17 de out. de 2020

FLUTTER - **Flutter: porque você deveria apostar nesta tecnologia** Disponível em:

< https://tableless.com.br/flutter-porque-investir-nessa-tecnologia/> Acessado em: 17 de out. de 2020MORONEY, Laurence; MORONEY; ANGLIN. Definitive Guide to Firebase. Apress, 2017.

DA SILVA, Marcelo Moro; SANTOS, Marilde Terezinha Prado. **Os paradigmas de desenvolvimento de aplicativos para aparelhos celulares**. Revista TIS, v. 3, n. 2, 2014. Acessado em: 17 de out. de 2020

PostgreSQL - **Documentação do PostgreSQL 8.2.0** Disponível em:

< http://pgdocptbr.sourceforge.net/pg82/index.html >. Acessado em: 17 de out. de 2020.

SADALAGE, Pramod J.; FOWLER, Martin. **NoSQL Essencial: Um guia conciso para o Mundo emergente da persistência poliglota.** Novatec Editora, 2019. Cap.1 Acessado em: 17 de out. de 2020.,

DATE, Christopher J. **Introdução a sistemas de banco de dados–Tradução da 8º Edição Americana**. Rio de Janeiro: Campus, 2004. p. 40. Acessado em: 17 de out. de 2020.

MACÁRIO, Carla Geovana do N.; BALDO, Stefano Monteiro. **O modelo relacional. Instituto de Computação–Universidade Estadual de Campinas. Campinas, São Paulo, 2005. p. 3.** Acessado em: 17 de out. de 2020.,

SILVA, Hudson Pacifico da ; ELIAS, Flavia Tavares Silva . **Incorporação de tecnologias nos sistemas de saúde do Canadá e do Brasil: perspectivas para avanços nos processos de avaliação.** Cad. Saúde Pública, Rio de Janeiro, v. 35, n. 14, e00071518, Abr. 2018. Disponível em: http://cadernos.ensp.fiocruz.br/csp/artigo/811/incorporacao-de-tecnologias-nos-sistemas-de-saude-do-canada-e-do-brasil-perspectivas-para-avancos-nos-processos-de-avaliacao. acessos em 06 Out:. 2020. http://dx.doi.org/10.1590/0102-311X00071518. Acessado em: 17 de out. de 2020.