UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

EZEQUIEL CORREIA DE SOUZA
LUCAS RODRIGO CORTINOVE CADASQUEVES

APLICATIVO PARA ATENDIMENTO AO CLIENTE DE OPERADORA DE PLANOS DE SAÚDE

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA
2018

EZEQUIEL CORREIA DE SOUZA
LUCAS RODRIGO CORTINOVE CADASQUEVES

APLICATIVO PARA ATENDIMENTO AO CLIENTE DE OPERADORA DE PLANOS DE SAÚDE

Trabalho de Conclusão de Curso de Graduação, apresentado à disciplina de Trabalho de Conclusão de Curso, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Câmpus Ponta Grossa, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof. Msc. Rogério Ranthum

PONTA GROSSA
2018

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Ponta Grossa
Diretoria de Graduação e Educação Profissional
Departamento Acadêmico de Informática
Tecnologia em Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

APLICATIVO PARA ATENDIMENTO AO CLIENTE DE OPERADORA DE PLANOS DE SAÚDE

por

EZEQUIEL CORREIA DE SOUZA
LUCAS RODRIGO CORTINOVE CADASQUEVES

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 06 de junho de 2018 como requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas. Os candidatos foram arguidos pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____

Rogério Ranthum
Orientador

_____

Geraldo Ranthum
Membro titular

_____

Marcos Vinicius Fidelis
Membro titular

_____

Profª. Dra. Helyane Bronoski Borges
Responsável pelo Trabalho de Conclusão de Curso

_____

Profº. Dr. André Pinz Borges
Coordenador do curso

- O Termo de Aprovação assinado encontra-se na Coordenação do Curso -

AGRADECIMENTOS

Em primeiro lugar agradecemos a Deus por todas as oportunidades nos proporcionadas e as forças no decorrer de nossas dificuldades, agradecemos a nossas famílias, esposas e filho(a) pelas ausências todas as noites que deixamos nossos lares para ir as aulas, aos nossos amigos de empresa do setor de tecnologia da Consaúde e Hospital Bom Jesus que muitas vezes nos auxiliaram no decorrer do trabalho, em especial ao Sr. Cleber Adriano de Oliveira que fez parte deste projeto gerando algumas funcionalidades externas ao sistema, por fim nosso orientador, Rogério Ranthum que aceitou nos orientar com todo o entusiasmo desde o início e nos guiou por todo o decorrer dos trabalhos, aqui o nosso obrigado.

RESUMO

CADASQUEVES, Lucas Rodrigo Cortinove, SOUZA, Ezequiel Correia. **Aplicativo para atendimento ao cliente de operadora de planos de saúde**. 2018. Número total de folhas 48f. Trabalho de Conclusão de Curso (Tecnologia em Análise e Desenvolvimento de Sistemas) - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2018.

Com a aceitação e a facilidade do uso de tecnologia nos dias atuais a redução de custos com impressões e a prevenção do meio ambiente com menos uso de papéis, tudo isso aliado a atualizações frequentes de constantes credenciamentos e descredenciamentos da lista de médicos clínicas e hospitais conveniados é criado o aplicativo para dispositivos móveis das plataformas Android e iOS a fim de auxiliar o beneficiário da operadora de planos de saúde.
Funcionalidades adicionais como informativo da empresa, opções de contatos com setores específicos e dicas de saúde foram adicionados a aplicação.

Palavras-chave: Android. iOS. Operadora de Planos de Saúde

ABSTRACT

CADASQUEVES, Lucas Rodrigo Cortinove, SOUZA, Ezequiel Correia. **Application for customer service of health plans operator**. 2018. Total number of sheets 48p. Course Completion Work (Technology in Analysis and Development of Systems) - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2018.

With the acceptance and the easiness of the use of technology in the present day, the reduction of costs with impressions and the prevention of the environment with less use of paper, all this combined with frequent updates of constant accreditations and accreditations of the list of medical clinics and hospitals the Android and iOS platforms mobile app is created to assist the health plan carrier's payee.
Additional features such as company information, contact options with specific sectors and health tips were added to the application.

Keywords: Android. iOS. Health Plan Operator

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS

API	*Application Program Interface*
IDE	*Integrated Development Environment*
GPS	*Global Positioning System*
GSM	*Global System for Mobile Communications*
XML	*Extensible Markup Language*
JSON	*JavaScript Object Notation*
ABRAMGE	Associação Brasileira de Planos de Saúde
IMC	Índice de Massa Corporal
URL	*Uniform Resource Locator*

SUMÁRIO

1 INTRODUÇÃO

Com 198 milhões de celulares em uso no país, as empresas estão buscando o desenvolvimento de aplicativos que flexibilizam o seu atendimento, permitindo o contato mais ágil com seus clientes.

Grandes empresas também têm encontrado a solução para a redução de custos com impressões de manuais, revistas e informativos que muitas vezes não chegam até ao cliente da forma desejada. Estes serviços estão disponibilizados de forma digital para que todos tenham acesso ao material atualizado.

Aplicativos para empresas da área da saúde permitem que os clientes se desloquem até o local mais próximo e recebam o atendimento imediato evitando transtornos aos seus clientes.

Soluções de aplicativos móveis buscam encurtar a distância dos clientes que precisam se deslocar para o hospital ou clínica tanto em casos de urgência e emergência, quanto em atendimentos clínicos em local desconhecido estando em uma cidade diferente ou até mesmo quem não costuma ir com frequência ao médico desconhecendo seus telefones e endereços, agilidade e precisão são fundamentais neste momento.

Este trabalho busca desenvolver um aplicativo para o Sistema Operacional Móvel Android, codificando na linguagem de programação Java e a *Integrated Development Environment* (IDE) de programação Android Studio como também um aplicativo para o Sistema Operacional iOS, codificando na linguagem *Swift* e a IDE *Xcode*. Com foco no cenário dos planos de saúde e seus clientes pretende-se por meio deste projeto implementar o aplicativo para agilizar e facilitar o atendimento e comunicação entre operadora e os seus beneficiários.

1.2 OBJETIVOS

Neste tópico serão abordados todos os objetivos do projeto.

1.2.1 Objetivo Geral

Desenvolver um aplicativo para a plataforma móvel nos sistemas operacionais Android e iOS que facilite a busca da rede credenciada e a comunicação dos clientes com a operadora de plano de saúde

1.2.2 Objetivos Específicos

- Levantar os requisitos para desenvolver os aplicativos de forma nativa.
- Configurar o ambiente de desenvolvimento.
- Estudar a IDE de cada plataforma.
- Efetuar o desenvolvimento.
- Efetuar testes.
- Estudar a forma de publicação.
- Publicar o aplicativo.

1.3 JUSTIFICATIVA

A empresa de planos de saúde em foco utiliza-se de um manual do beneficiário impresso contendo a rede credenciada de médicos, clínicas, hospitais e laboratórios, este por sua vez é utilizado por todos os 18 mil beneficiários aproximadamente, exigindo um alto custo nas impressões, além do que esse material tem que ser atualizado regularmente devido a novos credenciamentos e descredenciamentos de profissionais, prestadores de serviços, clinicas e etc.

Outro fator que pesa na escolha em desenvolver um meio para disponibilizar estes dados de forma digital é a consciência ambiental pois muitos quilos de papel que seriam necessários para a confecção de aproximadamente 10 mil manuais, que são feitos a cada 6 meses, várias árvores ao ano deixarão de ser cortadas para suprir essa necessidade. A proposta do projeto é reduzir isso a zero.

1.4 ESTRUTURA DO TRABALHO

Este trabalho foi estruturado da seguinte forma: o capítulo 2 apresenta a fundamentação teórica sobre os temas abordados no trabalho, apresentando um breve histórico dos sistemas Android e iOS e suas particularidades funcionais e melhorias e o detalhamento das arquiteturas nestas duas plataformas. O capítulo 3 apresenta as análises, estrutura do sistema e o desenvolvimento do projeto com uma terceira opção, o Ionic, este que foi inserido ao final das implementações iniciais, com o objetivo de testes com desempenho e desenvolvimento visando pontos positivos e negativos na programação nativa x hibrida. Por fim, o capítulo 4 conclui o trabalho, realizando uma análise sobre o que foi feito e indicando possíveis trabalhos futuros.

2 REFERENCIAL TEÓRICO

Neste capítulo será apresentado os detalhes das duas plataformas que está sendo desenvolvido o projeto, bem como um breve histórico sobre a computação móvel, a definição e arquitetura do sistema Android e iOS e como todos os sistemas operacionais possuem suas atualizações para correções e ajustes visando melhorias no desempenho e confiabilidade na segurança será apresentado as diferenças entre versões.

2.1 HISTÓRICO

A evolução da computação móvel passa por várias etapas. Observando pontos marcantes dessa trajetória que começa com Hans Christian Oersted, em 1820, quando descobre experimentalmente que a corrente elétrica produz um campo magnético. O primeiro sistema de comunicação foi o telégrafo, que já na metade do século XIX permitia a transferência de palavras faladas a longa distâncias pelo código Morse. Esse sistema era baseado na comunicação com fio.

O *SO (*Sistema Operacional*)* iOS foi lançado em janeiro de 2007 Integrado ao primeiro *iPhone* e funciona como uma interface entre as aplicações desenvolvidas pelos programadores (*apps*) e o *hardware* dos dispositivos (*iPhone*, *iPad*, *iPod*). (Apple Inc.,2007). Dessa forma, a comunicação com o *hardware* do dispositivo se dá por meio de um conjunto bem definido de interfaces do sistema, o que facilita o desenvolvimento de *apps* que funcionam entre os variados tipos de *hardware* dos dispositivos da Apple (Apple Inc., 2016b)

O *SO* Android foi criado pela *start-up* homônima Android Inc. em outubro de 2003. Em agosto de 2005 foi adquirida pela empresa Google, que lançou em novembro de 2007, juntamente com a *OHA (Open Handset Alliance)*, o sistema Android, *open-source* e baseado no *kernel* do Linux. Uma semana depois, foi liberada a primeira versão do *SDK* para Android. O sistema foi concebido originalmente para câmeras fotográficas, no entanto, foi percebido por seus criadores um mercado maior no ramo da telefonia e desviou-se o foco para *smartphones*, competindo diretamente com Symbian e Windows Mobile (SOFTWARE, 2016).

2.2 DEFINIÇÃO

O desenvolvimento de aplicativos móveis diferencia-se do desenvolvimento de outros tipos de software por possuir particularidades e restrições. Os desenvolvedores devem ter em mente aspectos como as capacidades e especificações dos dispositivos móveis, a mobilidade, o design e navegabilidade de interface gráfica, segurança e privacidade do usuário(SOFTWARE, 2016).

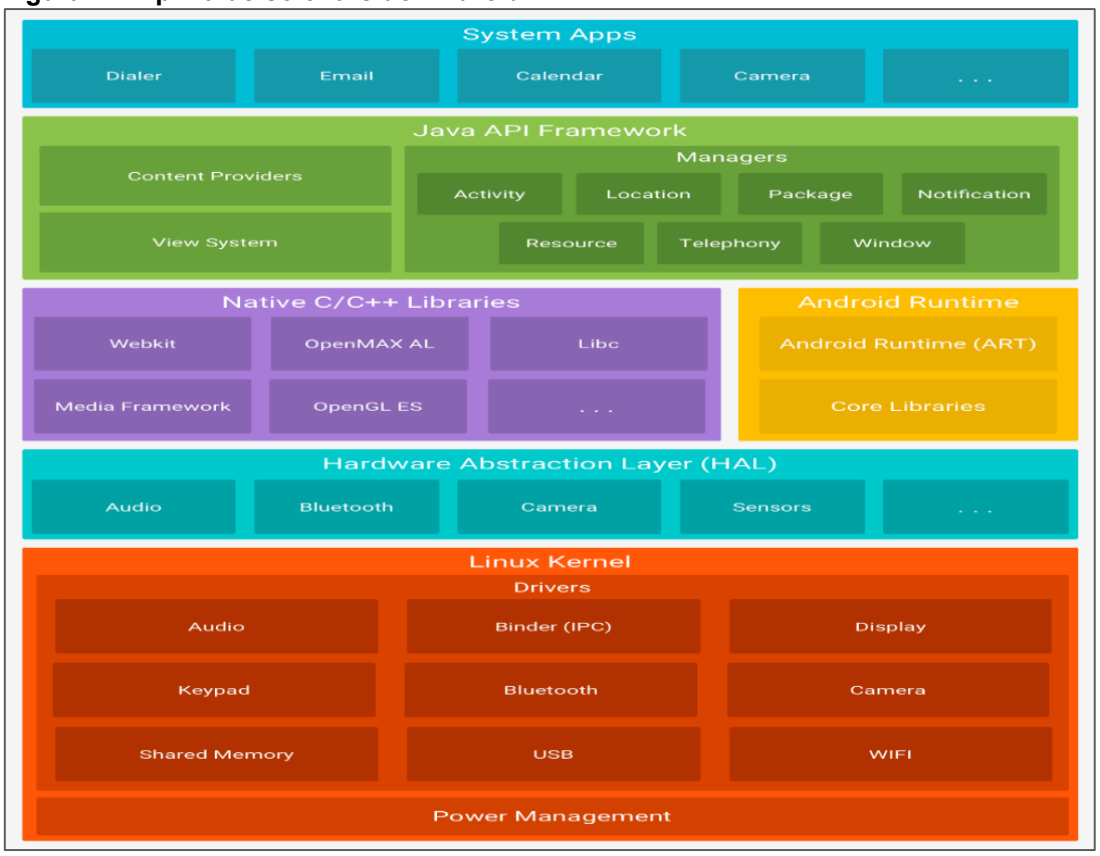
Aplicações nativas são desenvolvidas com o uso de ferramentas e linguagens de programação específicas para determinada plataforma, usando o SDK e frameworks providos por ela. Os apps ficam vinculados a esse ambiente, executando apenas nos dispositivos da plataforma alvo (SOFTWARE, 2016).

Caso haja a necessidade de implementação para múltiplas plataformas, a aplicação deve ser desenvolvida separadamente para cada uma delas (HEITKOTTER; HANSCHKE; MAJCHRZAK, 2013). Será desenvolvido para a plataforma do Android e iOS atingindo uma fatia de 99,6% do mercado de usuários ativos no início de 2017 conforme cita autor (GROSSMANN, [s.d.]) no texto: “Ao longo do ano foram vendidos 1,5 bilhão de aparelhos, com a consolidação dos sistemas operacionais Android e iOS, que juntos estão em 99,6% dos *smartphones*”. Caso haja a necessidade de implementação para múltiplas plataformas, a aplicação deve ser desenvolvida separadamente para cada uma delas desenvolver para as plataformas Android e iOS atinge uma fatia de 99,6% do mercado de usuários ativos no início de 2017. (GROSSMANN, [s.d.])

2.3 ARQUITETURA DO ANDROID

Conforme (ATTRIBUTION, 2017) a arquitetura do Android é uma pilha de software com base em Linux de código aberto criada para diversos dispositivos e fatores de forma. O diagrama a seguir mostra a maioria dos componentes da plataforma Android.

Figura 1 - A pilha de software do Android



Fonte: (OLIVEIRA, 2013)

2.3.1 Aplicativos do Sistema

O Android vem com um conjunto de aplicativos principais para e-mail, envio de SMS, calendários, navegador de internet, contatos etc. Os aplicativos inclusos na plataforma não têm status especial entre os aplicativos que o usuário opta por instalar. Portanto, um aplicativo terceirizado pode se tornar o navegador da Web, o aplicativo de envio de SMS ou até mesmo o teclado padrão do usuário (existem algumas exceções, como o aplicativo Configurações do sistema (ATTRIBUTION, 2017).

2.3.2 Estrutura da Java API

O conjunto completo de recursos do SO Android está disponível pelas APIs programadas na linguagem Java. Estas APIs formam os blocos de programação que

você precisa para criar os aplicativos Android simplificando a reutilização de componentes e serviços de sistema modulares e principais (ATTRIBUTION, 2017).

2.3.3 Bibliotecas C/C++ Nativas

Vários componentes e serviços principais do sistema Android, como ART e HAL, são implementados via código nativo que exige bibliotecas nativas programadas em C e C++. A plataforma Android fornece as Java Framework APIs para expor a funcionalidade de algumas dessas bibliotecas nativas aos aplicativos. Por exemplo, é possível acessar OpenGL ES pela Java OpenGL API da estrutura do Android para adicionar a capacidade de desenhar e manipular gráficos 2D e 3D no seu aplicativo (ATTRIBUTION, 2017).

2.3.4 Android Runtime

O ART é projetado para executar várias máquinas virtuais em dispositivos de baixa memória executando arquivos DEX, um formato de *bytecode* projetado especialmente para Android, otimizado para oferecer consumo mínimo de memória. Construa cadeias de ferramentas, como Jack, e compile fontes Java em *bytecodes* DEX, que podem ser executadas na plataforma Android (ATTRIBUTION, 2017).

2.3.5 Camada de Abstração de Hardware (HAL)

A camada de abstração de hardware (HAL) fornece interfaces padrão que expõem as capacidades de hardware do dispositivo para a estrutura da Java API de maior nível. A HAL consiste em módulos de biblioteca, que implementam uma interface para um tipo específico de componente de hardware, como o módulo de câmera ou *bluetooth*. Quando uma *Framework* API faz uma chamada para acessar o hardware do dispositivo, o sistema Android carrega o módulo da biblioteca para este componente de *hardware* (*ATTRIBUTION*, 2017).

2.3.6 Kernel do Linux

A fundação da plataforma Android é o kernel do linux. Por exemplo: o Android Runtime (ART) confia no kernel do Linux para cobrir funcionalidades como encadeamento e gerenciamento de memória de baixo nível. Usar um kernel do Linux permite que o Android aproveite os recursos de segurança principais e que os fabricantes dos dispositivos desenvolvam drivers de hardware para um kernel conhecido (ATTRIBUTION, 2017).

2.4 VERSÕES DO ANDROID

Abaixo é apresentado a evolução desde o primeiro sistema conforme descrito por SANTINO (2017) até o sistema atual.

Desde seu lançamento, em 2008, o Android já teve dez versões diferentes, sem contar as pequenas variações, entre as versões. Abaixo listamos as principais mudanças implantadas em cada uma delas (SANTINO, 2017).

Quadro 1- Tabela com as versões do sistema operacional Android

(continua)

VERSÃO	APELIDO	MUDANÇAS
1.0	Nenhum	A primeira versão comercial do sistema operacional, lançada em 23 de setembro de 2008. Já possuía aplicativos do Google e vários outros recursos básicos, mas que na época eram inovadores, como um *Media Player*, navegador e suporte a *Wi-Fi* e *Bluetooth*. Ele já tinha acesso ao Android *Market*, que mais tarde seria renomeado para *Google Play*, para *download* de aplicativos.
1.1	Nenhum	Primeira atualização do sistema, lançada em 9 de fevereiro de 2009, corrigiu falhas e *bugs* da versão 1.0 e não trouxe grandes inovações. Entre as novidades, estão o detalhamento e exibição de *reviews* de locais quando o usuário faria uma busca no *Maps* e melhorias na interface para realizar chamadas
1.5	Cupcake	Foi a primeira versão do sistema operacional a receber um apelido carinhoso de sobremesa, que virou padrão daí em diante. Lançada em 27 de abril de 2009, ela teve a inclusão dos *Widgets*, que até hoje são marca registrada do sistema e gravação e reprodução de vídeos em formato MPEG-4 e 3GP
1.6	Donut	A versão foi lançada em 15 de setembro de 2009 e trouxe suporte à resolução 800x480 e a inclusão de uma caixa de buscas já na tela inicial, para facilitar pesquisas internas e na *web*. Também teve melhorias em acessibilidade e a inclusão de um sistema de síntese de voz. Também trouxe mais facilidade de uso para o *Google Play*, possibilitando a inclusão de *screenshots* de aplicativos

Quadro 1- Tabela com as versões do sistema operacional Android

(continua)

2.0 a 2.1	Eclair	A “bomba de chocolate” foi lançada em 26 de outubro de 2009 e marcou a primeira atualização radical do sistema operacional móvel do *Google*. Trouxe uma nova interface, velocidade de hardware otimizada e suporte ao HTML5 no navegador. O sistema ainda apresentou a possibilidade de inclusão de várias contas no aparelho, para sincronização de contatos de várias fontes diferentes, além de trazer suporte ao protocolo de e-mail *Microsoft Exchange*. As atualizações posteriores apenas trouxeram correções de *bugs*
2.2 a 2.2.3	Froyo	Lançada em 10 de maio de 2010. A versão foi marcada por várias novidades que rodavam “sob o capô” do sistema e praticamente eram invisíveis ao usuário comum, com otimização de velocidade, memória e desempenho. Trouxe a possibilidade de transformar o celular em um *hotspot* de *Wi-Fi* e instalação de aplicativos em cartões de memória removíveis
2.3 a 2.3.7	Gingerbread	A mais popular do Android até pouco tempo, e também mais duradoura, presente até hoje em dispositivos mais baratos. Lançada em 6 de dezembro de 2010, trouxe interface renovada e simplificada e suporte à resolução HD e tecnologia NFC. Passou a ter suporte nativo a sensores como barômetro e giroscópio e a aceitar múltiplas câmeras em um mesmo dispositivo. Assim, as câmeras frontais passam a se popularizar
3.0 a 3.2	Honeycomb	Foi o único sistema operacional desenvolvido para tablets, lançado em 22 de fevereiro de 2011. Sua nova interface “holográfica” foi otimizada para este tipo de dispositivo. Ele trouxe melhorias de câmera e simplificação de multitarefas e suporte a processadores com múltiplos núcleos. A navegação na internet também foi melhorada, com a novidade do modo incógnito. O sistema também passou a permitir a encriptação de todos os dados do usuário
4.0 a 4.0.4	Ice Cream Sandwich	Lançado em 19 de outubro de 2011, a versão trouxe para os *smartphones* os botões virtuais disponíveis nos tablets com *Honeycomb*, abolindo a necessidade de teclas físicas nos dispositivos. Apresentou o Android Beam, que permitia o envio rápido de arquivos por aproximação de aparelhos, por meio de NFC. Também incluiu a possibilidade de acessar aplicativos diretamente da tela de bloqueio e desbloqueio por meio de reconhecimento facial. O Chrome passou a aceitar navegação em abas (até 16 abas simultâneas), e o sistema trouxe editor de fotos nativo.
4.1 a 4.2.2	Jelly Bean	A versão mais recente do Android, presente nos dispositivos mais modernos, tanto tablets quanto *smartphones*. Foi lançada em 9 de julho de 2012 e trouxe uma interface renovada e mais elegante e notificações expansíveis. A edição também trouxe o suporte ao Android Beam via *Bluetooth*.
4.4	KitKat	Conforme análise pode-se ressaltar os prós sendo: Mais elegante, sutil e rápido, comando “OK Google” está sempre disponível, o requisito mínimo do sistema é de 512 MB de RAM e a gratuidade do sistema e contras: é preciso fazer uma configuração adicional para usar o *Android Device Manager*, ainda indisponível para a grande maioria dos dispositivos Android na data em que ocorreu a análise e principais melhorias estavam quase invisíveis para o usuário comum nesta versão.

Quadro 1- Tabela com as versões do sistema operacional Android

(conclusão)

5.0 a 5.1	Lollipop	Lançado em 2014, o Lollipop que em inglês significa “pirulito”, essa versão possui um ajuste nas funcionalidades do sistema para o usuário final contendo interatividade utilizando gestos, cantos arredondados diferentes das versões anteriores e utilização de cores secundárias. No quesito desempenho é proporcionado um aumento de até 4x a velocidade de execução dos aplicativos sem o aumento do consumo da bateria, também nesta versão surge a opção de criação de perfis podendo desta forma o aparelho ser de uso compartilhado
6.0	Marshmallow	Possui melhorias na performance, possui um menu mais simples e intuitivo podendo rolar a barra ou pesquisar pela barra situada acima, pode-se navegar de forma alfabética pela barra lateral. Pode ser personalizado o menu de atalhos conforme necessidade. As permissões aos aplicativos a partir desta versão são concedidas apenas quando é preciso, garantindo desta forma uma maior segurança e em contrapartida alguns toques a mais na tela na utilização
7.0	Nougat	Nesta versão as notificações possuem uma “cara” nova, estando mais condensadas, podendo também ser respondido por exemplo uma mensagem recebida direto pela central sem precisar abrir o app. A multitarefa está disponível podendo ser utilizado duas aplicações em uma única tela. Passa a possuir o *JIT Compiler* que compila as aplicações baixadas convertendo em um código de máquina deixando as aplicações até 75% mais rápidas, e reduzindo o tamanho da aplicação em até 50%.
8.0	Oreo	Neste ano mais precisamente no dia 21 de março é lançado esta versão, possui grandes novidades, muitas delas já implementadas das plataformas das fabricantes dos dispositivos, mas que agora fazem partes da versão oficial. As notificações dos apps podem ser acessadas apenas pressionando os aplicativos sem precisar “entrar” neles. O consumo de bateria como em outras versões promete sempre está sendo melhorado, nas configurações podem ser ajustados este consumo conforme necessidade e utilização dos usuários

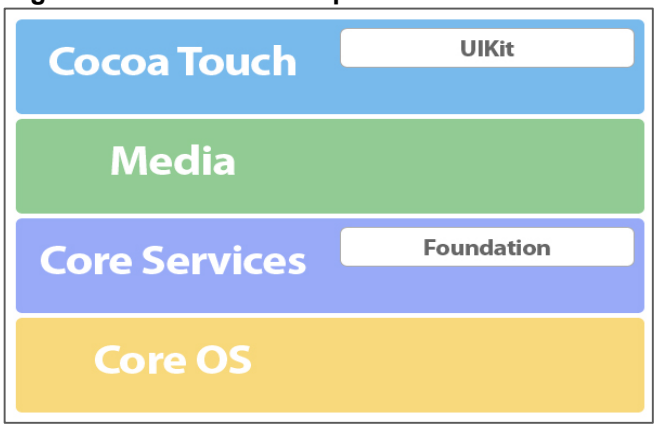
Fonte: (SANTINO, 2017)

Em todas as atualizações percebe-se nitidamente a preocupação com a usabilidade do sistema pelo usuário não alterando drasticamente a interface, mas apresentando melhorias claras ao usuário sem deixar de lado a preocupação com o consumo de energia das baterias dos aparelhos.

2.5 ARQUITETURA DO IOS

A arquitetura do iOS é formada por 4 camadas que fornecem o conjunto de *frameworks* que podem ser utilizados durante o desenvolvimento de aplicativos móveis da Apple Inc. Essas camadas são, *Cocoa Touch, Media, Core Services e Core OS.* (“Segurança do iOS”, 2016)

Figura 2 - Camadas da arquitetura iOS



Fonte: (APPLE INC., [s.d.])

2.5.1 Cocoa Touch

É a camada de mais alto nível onde fornece serviços básicos de interação com o usuário como multitoque, Acelerômetro, Câmera, Alertas, *Pickers*, Sistema de Localização, interface gráfica, comunicação com arquivos, comandos de interação com usuários. (MILANI, 2012)

A estrutura das aplicações UIKit baseia-se no padrão de design *Model-View-Controller* (MVC), em que os objetos são divididos por sua finalidade. Objetos de modelo gerenciam os dados do aplicativo e a lógica de negócios. Os objetos de visão fornecem a representação visual de seus dados. Os objetos do controlador atuam como uma ponte entre seu modelo e exibem objetos, movendo dados entre eles nos momentos apropriados.

2.5.2 Media

É a camada responsável pelo gerenciamento de animações, áudio, vídeo, e tecnologias como a OpenGL ES e a *Quartz*, ambas utilizadas para a criação de aplicações gráficas como jogos. (MILANI, 2012)

A estrutura Media define o pipeline de mídia usado pela *AVFoundation* e outras estruturas de mídia de alto nível encontradas nas plataformas da Apple. Use os tipos de dados de baixo nível e interfaces do Media para processar eficientemente amostras de mídia e gerenciar filas de dados de mídia. (APPLE, 2017)

2.5.3 Core Services

É a camada que disponibiliza alguns dos principais serviços do sistema operacional para o programador, como o de manipulação de arquivos e o de acesso ao SQLite, entre outros. (MILANI, 2012)

Essa camada está mais próxima do hardware e, portanto, possui acesso a funcionalidades de mais baixo nível como localização, telefonia, threads e SQLite. Aqui residem dois dos frameworks mais importantes do iOS que são o Foundation e o Core Foundation, ambos relacionados com o gerenciamento de dados e alguns serviços e definem todos os tipos básicos de dados que todos os apps usam, como por exemplo, coleções, *strings*, data e hora, *sockets* e *threads*. (APPLE, 2017)

2.5.4 Core OS

É a camada de mais baixo nível, que é o Core OS, considerado o núcleo do sistema operacional. Esta camada é responsável por gerenciar os sockets, certificados e energia, entre outros dos principais recursos do iOS. Basicamente, é a camada que gerencia a parte de segurança e da comunicação do sistema com o mundo externo. (MILANI, 2012)

A camada Core OS também encapsula o kernel e interfaces de baixo nível do UNIX o qual seu aplicativo não tem acesso, por razões de segurança. Contudo, através da biblioteca *libSystem*, que é baseada em C, muitas características de baixo nível podem ser acessadas diretamente, tais como os sockets BSD, threads POSIX e serviços de DNS. (APPLE, 2017)

2.6 VERSÕES DO IOS

Quadro 2 - Tabela com as versões do sistema operacional iOS

(continua)

VERSÃO	DATA LANÇAMENTO	MUDANÇAS
Iphone OS 1	03/03/2008	A primeira versão do sistema de multitoque para dispositivos móveis, foi apresentado inicialmente como uma versão móvel do OS X, porém recebeu o nome de iPhone OS, com o lançamento do kit de desenvolvimento do iPhone, ou iPhone SDK.

Quadro 2 - Tabela com as versões do sistema operacional iOS

(continua)

Iphone OS 2	-	Lançada juntamente com o iPhone 3G, a principal melhoria desta versão foi a inclusão pela primeira vez da App *Store*, permitindo a instalação de aplicativos criados por terceiros disponíveis no iPhone e iPod *touch*
Iphone OS 3	17/06/2009	A versão foi lançada juntamente com o iPhone 3GS, e trouxe como destaques a função copiar e colar, e MMS
iOS 4	21/06/2010	Foi o primeiro lançamento do sistema chamado simplesmente de "iOS". A grande novidade desta versão foi a inclusão da função multitarefa no sistema.
iOS 5	03/06/2011	Uma prévia do sistema iOS 5 foi apresentada no dia 6 de junho de 2011, durante o evento da WWDC 2011. No mesmo evento foi lançado uma prévia também do novo sistema operacional para computadores Apple, o Mac OS X Lion, e anunciado um serviço baseado na nuvem, chamado iCloud. Nesta versão foi apresentado pela primeira vez a central de notificações, possibilidade de editar fotos e integração com o novo serviço, o iCloud, além de integração com o Twitter, acesso rápido a câmera pela tela de bloqueio, também foi apresentado aos usuários dos Estados Unidos, Reino Unido e Austrália a assistente virtual Siri no idioma Inglês e para a França no idioma Francês.
iOS 6	-	Esta versão trouxe aproximadamente 100 novos recursos, dentre eles, os mais relevantes, a função Não Perturbe (faz com que não receba notificações por um período determinado pelo usuário), o novo aplicativo Mapas e acesso guiado para GPS, *Passbook*, mudança no layout da App *Store* e compatibilidade da rede 3G com o *Facetime*. A última versão foi a versão 6.1.3, onde foram corrigidas falhas que burlavam a tela de bloqueio do sistema que davam acesso a agenda do iPhone sem precisar digitar o código de segurança, além de correção nas brechas do *Jailbreak*. Também foi liberada logo a seguir, a versão 6.1.4 somente para o iPhone 5, que atualizou o perfil de áudio para viva voz. A sexta versão do iOS é compatível com iPhone 3GS, iPhone 4 e 4s, iPhone 5, iPod Touch de quarta e quinta geração, iPad 2, iPad 3ª geração e iPad Retina e iPad mini.
iOS 7	18/09/2013	foi apresentada ao público no WWDC 2013, evento anual da Apple que ocorreu dia 10 de junho de 2013. O sistema é compatível com iPhone 4 e posteriores, iPad 2, Retina, Air e iPod *touch* de 5ª geração. O iOS 7 apresentou a maior mudança de interface gráfica entre uma versão atual e a anterior desde sua primeira versão. Além das mudanças visuais, o sistema também adquiriu uma nova ferramenta de acesso rápido aos aplicativos mais utilizados, como calculadora, lanterna (luzes do flash), temporizador, câmera, *AirDrop*, além de poder ativar e desativar algumas funções do sistema, como desativar a rede Wi-Fi e ajustar o brilho. Todos ícones de aplicativos nativos ganharam novo design e layouts internos. Siri (assistente de voz da Apple), agora possui também a opção de voz masculina para a voz inglesa. A última versão lançada foi a 7.1, no dia 10 de março de 2014, apresentando mudanças estéticas no sistema, como reparos de segurança e algumas mudanças no design

Quadro 3 - Tabela com as versões do sistema opreracional iOS

(conclusão)

iOS 8	17/09/2014	Uma das principais melhorias incluídas nesta versão foi o '*HealthKit Framework'* que permite que um aplicativo leia e escreva dados de saúde e atividade para seu aplicativo de Saúde, seu aplicativo pode se tornar uma fonte de dados de saúde valiosa e pode usar os dados compartilhados para trazer soluções de saúde e fitness mais poderosas, foi liberado para o Brasil na versão 8.3 do iOS suporte ao idioma Português da assistente virtual Siri.
iOS 9	08/06/2015	Esta versão foi disponibilizada no dia 8 de junho de 2015 e foi projetado para trabalhar com o novo toque 3d do Iphone 6s e 6s Plus; esta versão trouxe atualização de aplicativos que incluem notas, mapas, notícias, pagamento e sendo uma das principais melhorias o recurso *Picture in Picture* (também conhecido como PiP) que permite aos usuários assistirem vídeos em uma janela que flutua acima de outras aplicações na tela. Sua última versão foi lançada em 16 de setembro de 2015.
iOS 10	13/06/2016	Aplicativo de mensagens foi revisado permitindo textos com uma variedade de efeitos especiais. O aplicativo Maps foi redesenhado para melhorar sua navegação. Sua última versão foi lançada em 13 de setembro de 2016.
iOS 11	05/06/2017	Disponibilizado o suporte para realidade aumentada; Apple pagamentos, nova loja de aplicativos, centro de controle atualizado, síri e mapas, não perturbe durante a condução com configuração automática.

Fonte: **(APPLE, 2017)**

3 DESENVOLVIMENTO

Este capítulo mostra como o trabalho foi realizado, quais os ambientes e as tecnologias utilizadas, como foi estruturado o sistema, os documentos que foram gerados e as etapas do desenvolvimento.

3.1 DESCRIÇÃO DO PROJETO

O trabalho desenvolvido consiste em um aplicativo que permite o acesso facilitado e atualizado da rede credenciada e descredenciada da operadora de planos de saúde Consaúde, rede credenciada junto a Abramge, rede credenciada a Rede Filantrópica, Acesso Restrito (Beneficiário, Empresa e Prestador de Serviço), Dicas de saúde, contato, cálculo de IMC e sobre (a Consaúde, programas oferecidos e desenvolvimento).

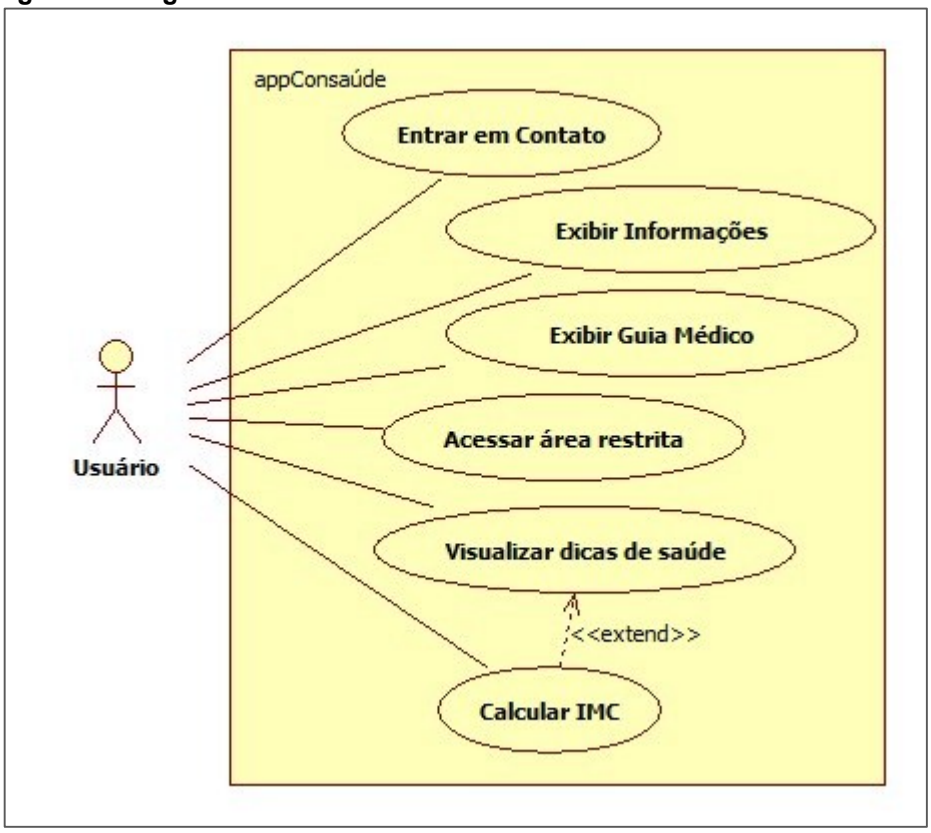
Através destas funcionalidades acredita-se que os usuários tenham o acesso facilitado a informação. Sendo assim temos como base para os requisitos da aplicação, Guia medico, acesso restrito, dicas de saúde, contato, cálculo de IMC e sobre.

3.2 CASO DE USO

A análise de requisitos com caso de uso deve descrever somente uma funcionalidade ou objetivo do sistema. É então comum, para sistemas minimamente complexos, serem necessários muitos casos de uso para uma correta e completa descrição de todas as funcionalidades requeridas pelo sistema.

A Figura 3 apresenta o projeto através da análise sendo ilustrado o caso de uso da aplicação.

Figura 3 - Diagrama de casos de uso



Fonte: Autoria própria

3.4 ESTRUTURA DO SISTEMA

O processo de desenvolvimento do projeto utilizou diversas características da metodologia de desenvolvimento em cascata nos fornece, sendo assim foi buscado desenvolver cada funcionalidade em separado realizando assim os testes devidos resolvidos os problemas identificados durante os testes e após isso segue-se para a próxima funcionalidade até chegar ao produto final.

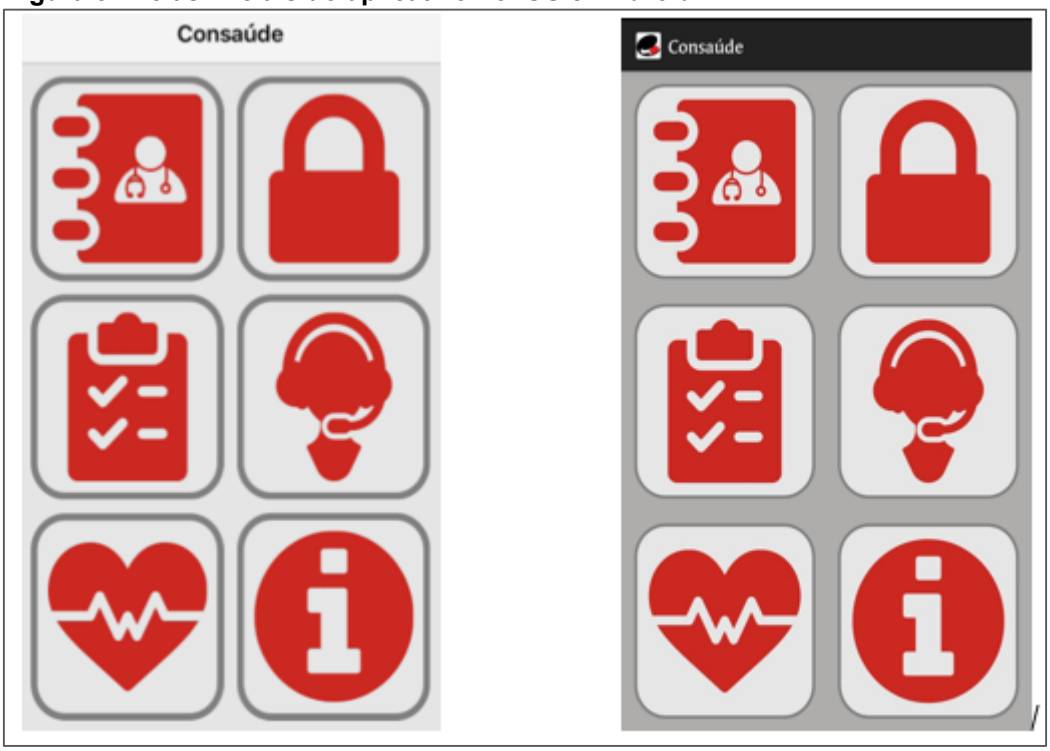
O aplicativo foi desenvolvido para se ter o acesso rápido e simplificado a todas as informações disponíveis no sistema. A Figura 4 representa a forma como cada dispositivo disponibiliza as informações para cada plataforma (Android e iOS).

Figura 4 - Disponibilização das informações

Fonte: Autoria própria

Para o dispositivo Android o desenvolvimento foi em programação na linguagem Java, no Android Studio, os dados são armazenados em um arquivo *eXtensible Markup Language* (XML) salvo localmente no dispositivo do usuário. Para o dispositivo Apple iOS foi utilizado a linguagem de programação Swift e o acesso das informações se dá através de um arquivo json que é consumido em tempo real de um *web service* desenvolvido pela própria operadora.

Figura 5 - Telas iniciais do aplicativo no iOS e Android

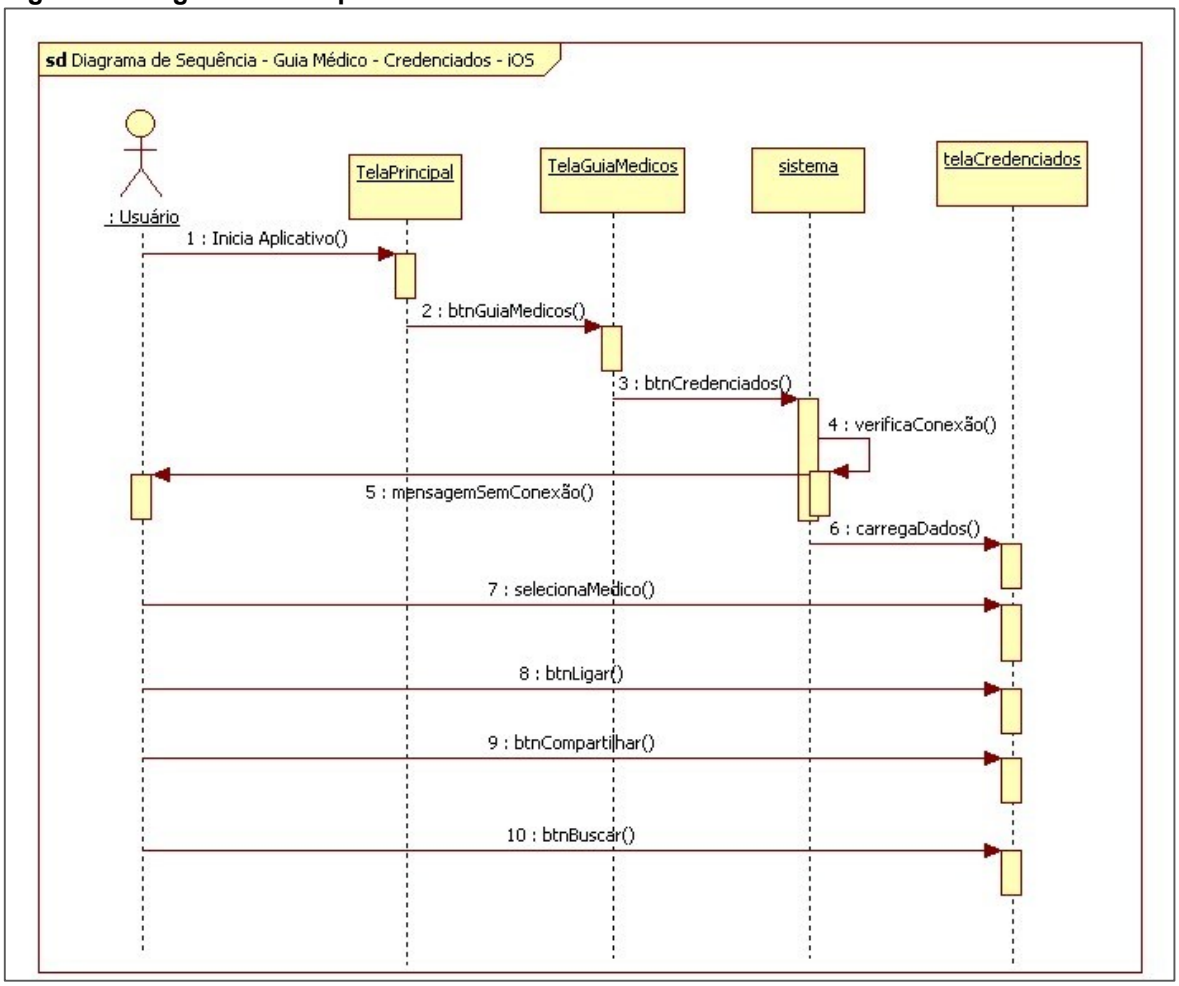


Fonte: Autoria própria

3.4.1 Desenvolvimento Tela Credenciados

A *Figura 6 - Diagrama de sequência* apresenta o diagrama de sequência para exibir a rede credenciada referenciando o caso de uso “Exibir Guia Médico”, apresentando o método de acesso e de persistência na tabela de credenciados.

Figura 6 - Diagrama de sequência



Fonte: Autoria própria

3.4.1.1 Desenvolvimento tela credenciados – ios

Para consumir as informações do *webservice* foi utilizado no desenvolvimento do aplicativo iOS as seguintes funções próprias da linguagem *SWIFT*.

A função abaixo faz a requisição a url onde está o armazenado o arquivo *json* no *webservice* da operadora.

Figura 7 - Código de requisição url

```
let task = URLSession.shared.dataTask(with: url!)
```

Fonte: Autoria própria

Em seguida é feita a serialização do arquivo para que o mesmo seja lido, interpretado.

Figura 8 - Código de serialização

```
let myJson = try JSONSerialization.jsonObject(with: content,
    options: .mutableContainers)
```

Fonte: Autoria própria

Após feita a serialização e a interpretação dos dados consumidos do web service, estes dados são armazenados na estrutura “GUIAmedicos”.

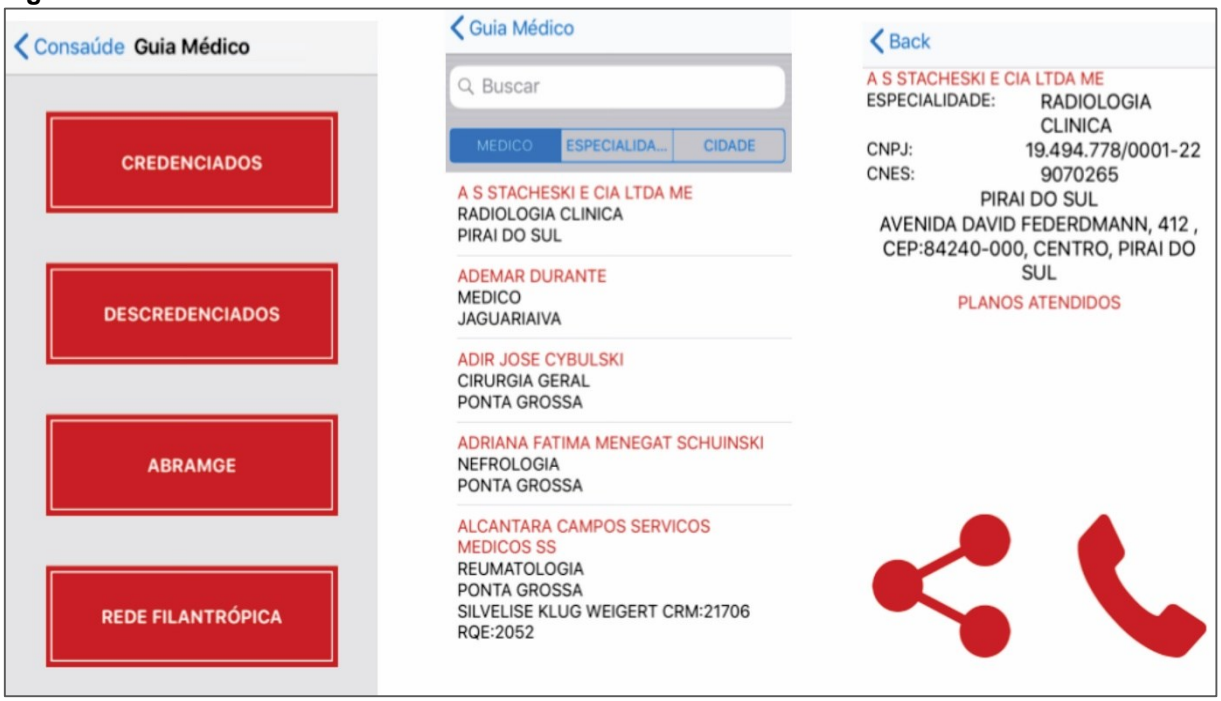
Figura 9 - Código da estrutura GUIAmedicos

```
struct GUIAmedicos {
    var ID = ""
    var NOME = ""
    var CONSELHO = ""
    var ESPECIALIDADE = ""
    var RQE = ""
    var CORPOCLINICO = ""
    var CNES = ""
    var URL_CNES = ""
    var CIDADE = ""
    var ENDERECO = ""
    var ENDERECO_PRINCIPAL = ""
    var FONE = ""
    var MAPS = ""
    var CNPJ = ""
    var SITE = ""
    var ID_CLASSIFICACAO = ""
}
```

Fonte: Autoria própria

Após armazenado os dados na estrutura basta fazer a leitura da mesma e exibir na tela da rede credenciada conforme Figura 10.

Figura 10 - Telas da rede credenciada no iOS



Fonte: Autoria própria

Para a funcionalidade de compartilhamento de informações foi utilizada a seguinte função:

- UIActivityViewController: Esta classe nos fornece vários serviços padrões do sistema, como copiar itens para a área de trabalho, postar conteúdo em sites de mídia social, enviar itens por e-mail ou SMS e dentre outros.

Figura 11 - Código de compartilhamento

```
let activityController = UIActivityViewController(activityItems: [shareContent],
    applicationActivities: nil)

shareContent = "CONSULTÓRIO: \(nome)\n\nENDEREÇO:\n\(arrayEnd)\n\nTELEFONE:\(tele)\n*ENVIADO VIA APP
    CONSAÚDE*"
```

Fonte: Autoria própria

3.4.1.2 Desenvolvimento tela credenciados – android

O manual do beneficiário será gravado no dispositivo em um arquivo xml, onde temos a busca dos dados na web e inserção do arquivo como os dados no dispositivo conforme imagem abaixo.

Figura 12 - Código de busca e gravação do arquivo no dispositivo

```
        url = new URL(ENDERECO_WEB_REDE_DESCREDENCIADA_CONSAUDE);
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        connection.setRequestMethod("GET");
        InputStream inputStream = connection.getInputStream();
        DocumentBuilderFactory builderFactory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = builderFactory.newDocumentBuilder();
        xmlDoc = builder.parse(inputStream);
        if (xmlDoc != null) {
            escreveArquivoDesc(xmlDoc);
        }
public boolean escreveArquivo(Document document) {
    try {
        FileOutputStream fos = new FileOutputStream(new File(getFilesDir(), NOME_ARQUIVO_REDE_CREDENCIADA));
        FileOutputStream fosData = new FileOutputStream(new File(getFilesDir(), NOME_DATA_ATUALIZACAO));
        if (document != null) {

            //escreve o arquivo xml baixado da web
            DOMSource domSource = new DOMSource(document);
            StreamResult sr = new StreamResult(fos);
            Transformer transformer = TransformerFactory.newInstance().newTransformer();
            transformer.transform(domSource, sr);
            fos.close();
```

Fonte: Autoria própria

Para leitura dos dados contido no arquivo cada nó pai médico é percorrido e cada nó filho de atributo é analisado e manipulado conforme necessidade, abaixo um exemplo de busca onde é retornado os municípios contidos no arquivo de forma ordenada.

Figura 13 - Código de demonstração de uma busca no arquivo

```
public ArrayList<String> getMunicipios() {
    ArrayList<String> municipios = null;
    ArrayList<String> cidade = new ArrayList<>();
    Document data = capturaDocumentoDoDispositvo();
    if (data != null) {
        NodeList listaDePessoas = data.getElementsByTagName("medico");
        int tamanhoLista = listaDePessoas.getLength();
        for (int i = 0; i < tamanhoLista; i++) {
            Node noPessoa = listaDePessoas.item(i);
            if (noPessoa.getNodeType() == Node.ELEMENT_NODE) {
                Element elementMedico = (Element) noPessoa;
                NodeList noMedico = elementMedico.getChildNodes();
                for (int j = 0; j < noMedico.getLength(); j++) {
                    Node noAtributo = noMedico.item(j);
                    if (noAtributo.getNodeType() == Node.ELEMENT_NODE) {
                        Element elementAtributo = (Element) noAtributo;
                        switch ((elementAtributo.getTagName())) {
                            case "Cidade": {
                                cidade.add(elementAtributo.getTextContent());
                            }
                        }
                    }
                }
            }
        }
        municipios = new ArrayList<String>(new LinkedHashSet<String>(cidade));
        Collections.sort(municipios);
    }
    return municipios;
```

Fonte: Autoria própria

O compartilhamento foi utilizado a *Intent ACTION_SEND* que recebe como parâmetro o os conteúdos do profissional ou local selecionado.

Figura 14 - Método responsável pelo compartilhamento de conteúdo

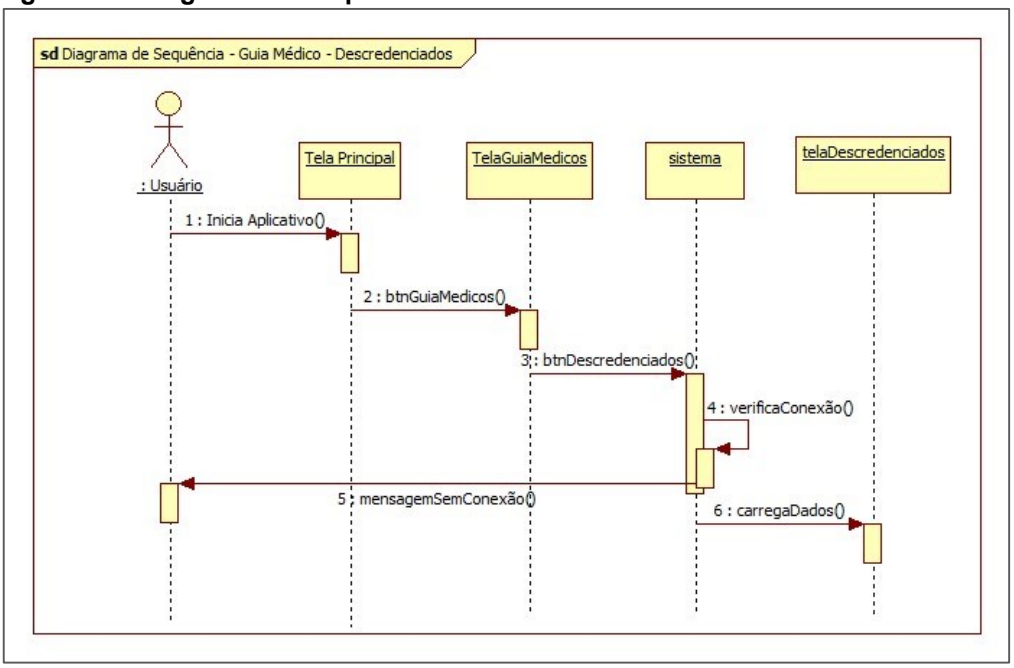
```
public void abreCompartilhamento(View v) {
    Intent share = new Intent(android.content.Intent.ACTION_SEND);
    share.setType("text/plain");
    share.addFlags(Intent.FLAG_ACTIVITY_CLEAR_WHEN_TASK_RESET);
    share.putExtra(Intent.EXTRA_TEXT, CapturaDadosParaCompartilhar());
    startActivity(Intent.createChooser(share, title: "Compartilhar com:"));
}
```

Fonte: Autoria própria

3.4.2 Desenvolvimento Tela Descredenciados

A Figura 15 apresenta o diagrama de sequência para exibir a rede credenciada referenciando o caso de uso “Exibir Guia Médico”, apresentando o método de acesso e de persistência na tabela de descredenciados.

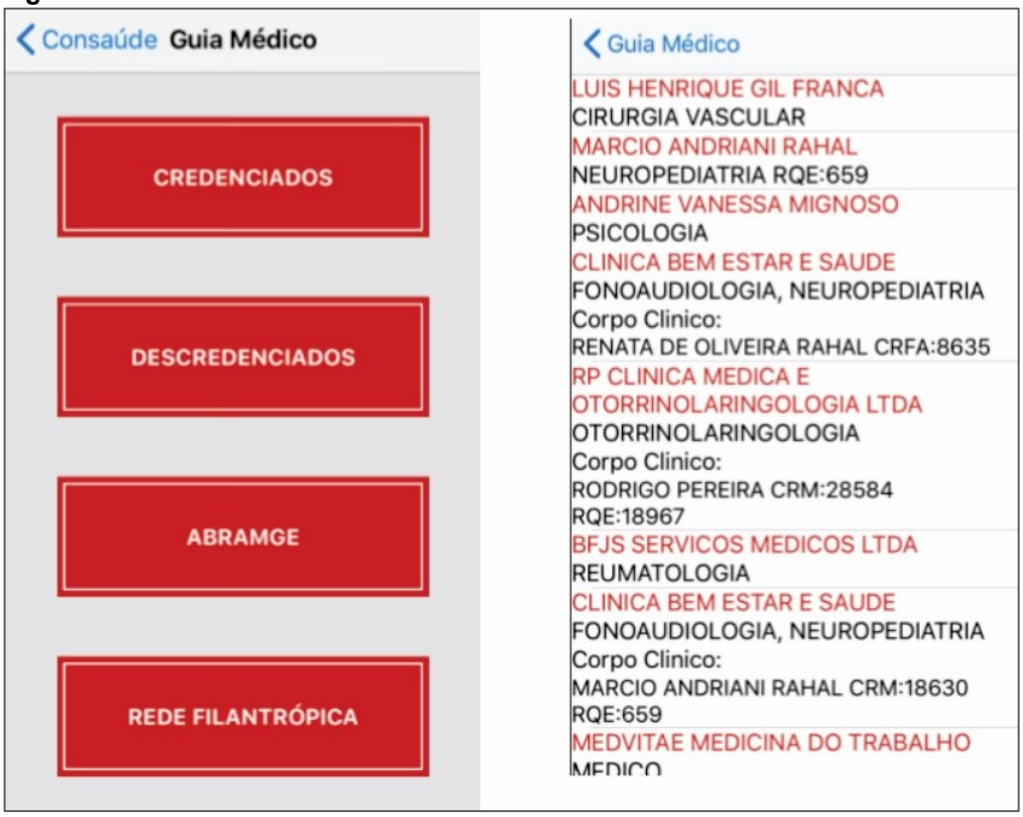
Figura 15 - Diagrama de sequência dos descredenciados



Fonte: Autoria própria

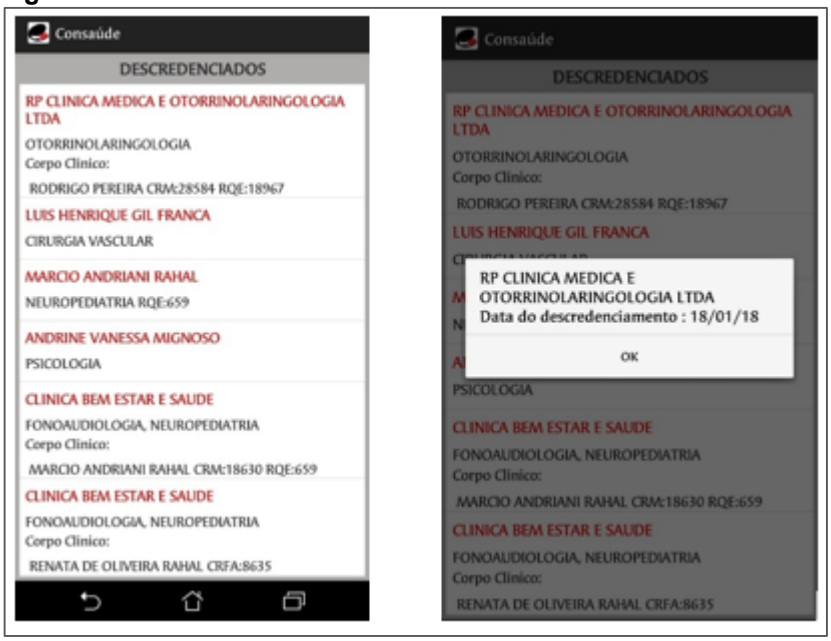
O desenvolvimento da tela de descredenciados utiliza as mesmas funções da tela de credenciados, porém não tem a tela de detalhamento médico.

Figura 16 - Tela dos descredenciados no iOS



Fonte: Autoria própria

Figura 17 - Tela dos descredenciados no Android

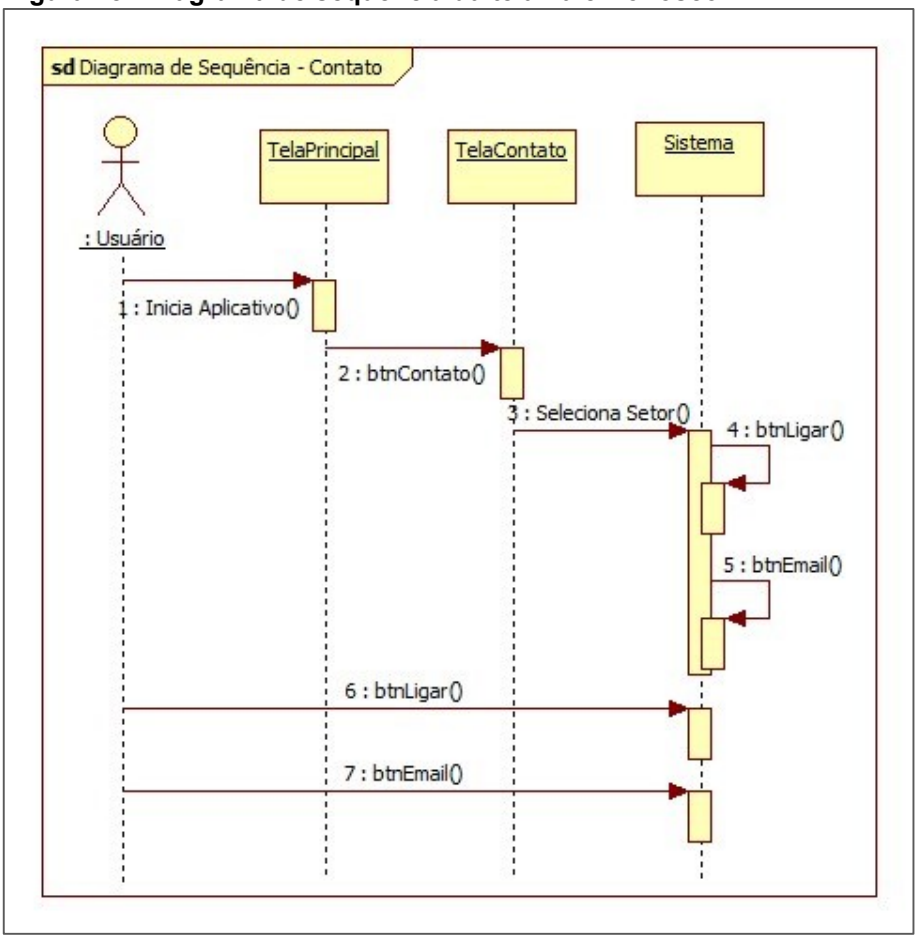


Fonte: Autoria própria

3.4.3 Desenvolvimento Tela Contato

A Figura 18 apresenta o diagrama de sequência da tela de contato referenciando o caso de uso “Entrar em contato”.

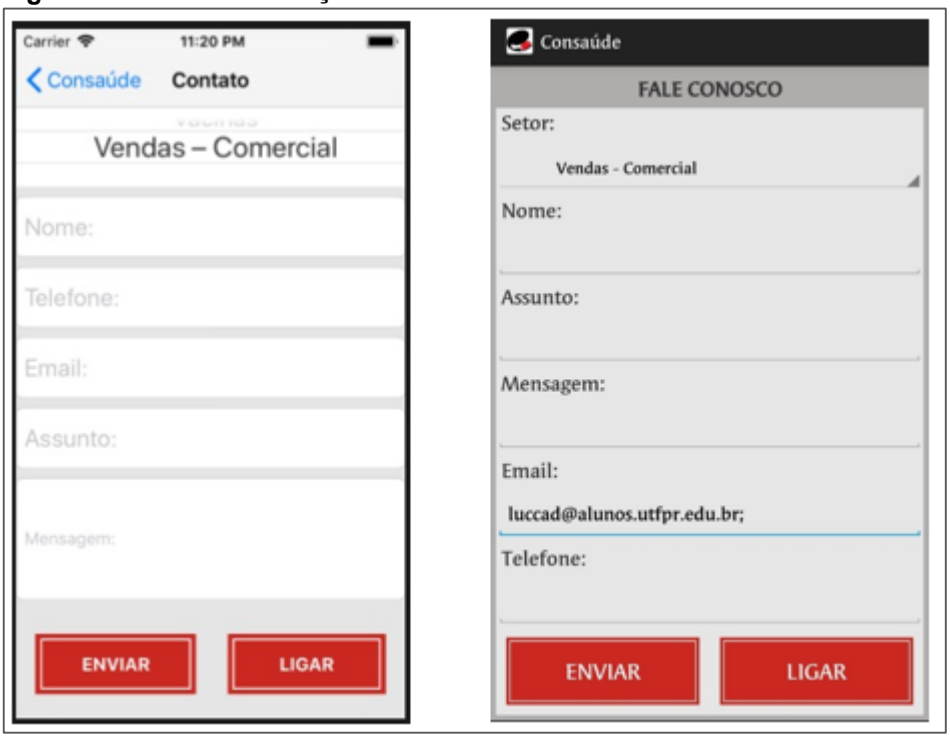
Figura 18 - Diagrama de sequência da tela Fale Conosco



Fonte: Autoria própria

A Figura 19 apresenta as telas de interação do fale conosco no iOS e Android.

Figura 19 - Tela de interação do fale conosco no iOS e Android



Fonte: Autoria própria

3.4.3.1 Desenvolvimento tela contato – ios

Para as funcionalidades de chamada do “btnLigar” foi utilizada as seguintes classes:

- NSURL: Tem a função de fazer a ponte entre a URL e funcionalidades especificas da arquitetura do iOS.

- UIAplication: Tem a função de reconhecer a URL e chamar a funcionalidade de chamada do dispositivo.

Figura 20 - Código de chamada do iOS

```
let url: NSURL = URL(string: "TEL://\(contato ?? 04232205000)")! as NSURL
UIApplication.shared.open(url as URL, options: [:], completionHandler: nil)
```

Fonte: Autoria própria

Para a funcionalidade de enviar e-mail do btnEmail foi utilizada a seguinte função.

- MFMailComposeViewController: Esta classe fornece o controlador de interface de e-mail padrão dentro do aplicativo, após o usuário inserir os valores solicitados o mesmo é direcionado para o cliente de e-mail padrão do seu dispositivo e após ser enviada ela será enfileirada na caixa de saída de e-mail do usuário.

Figura 21 - Código envio e-mail

```
let mailComposeViewController = configureMailController()
if MFMailComposeViewController.canSendMail(){
    self.present(mailComposeViewController, animated: true, completion: nil)
```

Fonte: Autoria própria

3.4.3.2 Desenvolvimento tela contato – android

No contato por telefone apenas uma *Intent* de chamada do discador é instanciada e como parâmetro o telefone é passado como parâmetro conforme código apresentado abaixo.

Figura 22 - Código de ligação do Android

```
private void Ligar(String Telefone) {
    String telefone = "tel:" + Telefone;
    Uri uri = Uri.parse(telefone);
    Intent it = new Intent(Intent.ACTION_DIAL, uri);
    if (it.resolveActivity(getPackageManager()) != null) {
        startActivity(it);
    } else {
        Toast.makeText( context: this,  text: "Não" + " encontrou um discador para " +
                "realizar a chamada.", Toast.LENGTH_SHORT).show();
    }
}
```

Fonte: Autoria própria

O contato de e-mail por sua vez faz a comunicação via protocolo SMTP com dados de uma conta Gmail criada apenas para o aplicativo onde a classe Mail.java se encarrega de fazer todo o processo de envio após instanciada e receber os parâmetros necessários nos seus métodos. Tratando de interação com o usuário esse processo apenas é executado apenas quando todos os campos da interface são preenchidos, como funcionalidade adicional no Android preenchemos automaticamente o campo de e-mail conforme contas de e-mail cadastradas no dispositivo, abaixo:

Figura 23 - Código de envio do e-mail no Android

```
public boolean send() throws Exception {
    Properties props = _setProperties();
    if (!_user.equals("") && !_pass.equals("") && _to.length > 0 && !_from.equals("") &
        Session session = Session.getInstance(props, authenticator: this);
        MimeMessage msg = new MimeMessage(session);
        msg.setFrom(new InternetAddress(_from));
        InternetAddress[] addressTo = new InternetAddress[_to.length];
        for (int i = 0; i < _to.length; i++) {
            addressTo[i] = new InternetAddress(_to[i]);
        }
        msg.setRecipients(MimeMessage.RecipientType.TO, addressTo);
        //se tem de enviar cópia oculta para alguém
        if (_cco != null && _cco.length > 0) {
            InternetAddress[] addressCco = new InternetAddress[_cco.length];
            for (int i = 0; i < _cco.length; i++) {
                addressCco[i] = new InternetAddress(_cco[i]);
            }
            msg.addRecipients(Message.RecipientType.BCC, addressCco);
        }
        msg.setSubject(_subject);
        msg.setSentDate(new Date());
        // corpo da mensagem
        BodyPart messageBodyPart = new MimeBodyPart();
        messageBodyPart.setText(_body);
        if (_isHtmlBody) {
            messageBodyPart.setHeader( s: "charset", s1: "utf-8");
            messageBodyPart.setHeader( s: "content-type", s1: "text/html");
        }
        _multipart.addBodyPart(messageBodyPart);
        msg.setContent(_multipart);
        // envia o email
        Transport.send(msg);
        return true;
```

Fonte: Autoria própria

O compartilhamento foi implementado para facilitar os usuários a enviar a outras pessoas o local/profissional filtrado, aproveitando desta situação a divulgação do aplicativo na última linha, abaixo está ilustrado a função responsável por esta funcionalidade:

Figura 24 - Código de compartilhamento de conteúdo

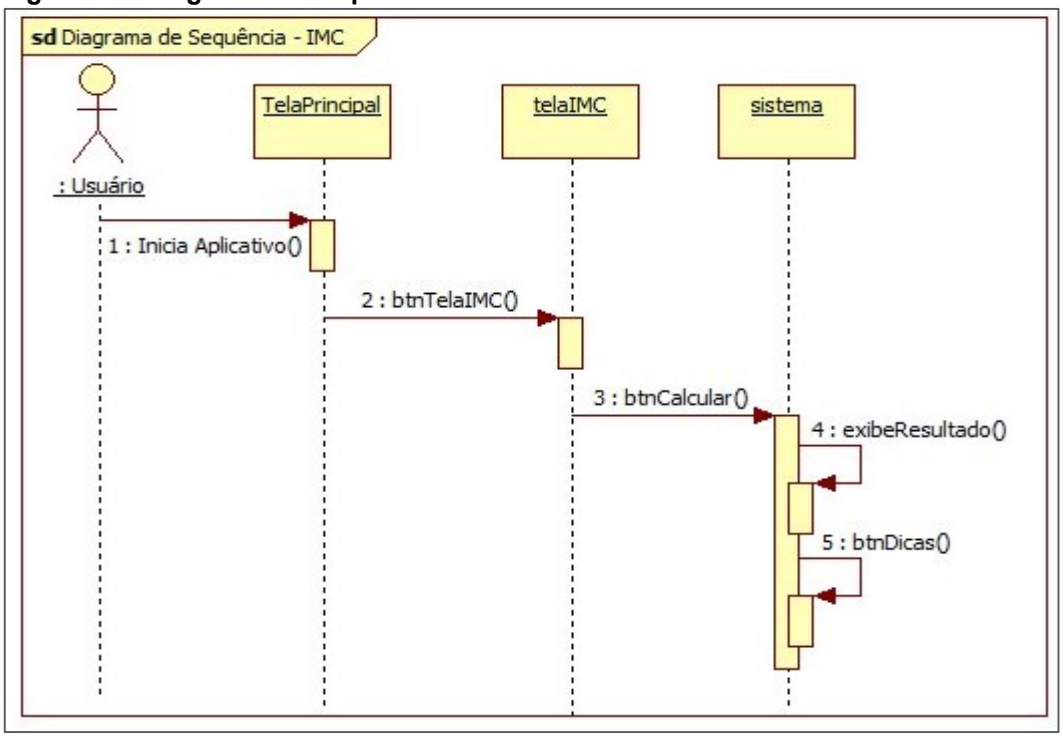
```
public void abreCompartilhamento(View v) {
    Intent share = new Intent(android.content.Intent.ACTION_SEND);
    share.setType("text/plain");
    share.addFlags(Intent.FLAG_ACTIVITY_CLEAR_WHEN_TASK_RESET);
    share.putExtra(Intent.EXTRA_TEXT, CapturaDadosParaCompartilhar());
    startActivity(Intent.createChooser(share, title: "Compartilhar com:"));
}

public String CapturaDadosParaCompartilhar() {
    String end;
    if (quantidade > 1) {
        end = "Endereços:";
    } else {
        end = "Endereço:";
    }
    String valor = medico.getId_classificacao().toString() + ": " + medico.getNome().toString()
            + "\n\n" + end + enderecoCompartilhar + "\n\nTelefone: " + medico.getFone().toString() +
            " \n**ENVIADO VIA APP CONSAÚDE**";
    return valor;
}
```

Fonte: Autoria própria

3.4.3 Desenvolvimento Tela Imc

A Figura 25 abaixo apresenta o diagrama de sequência da tela de IMC referenciando o caso de uso “Calcular IMC”.

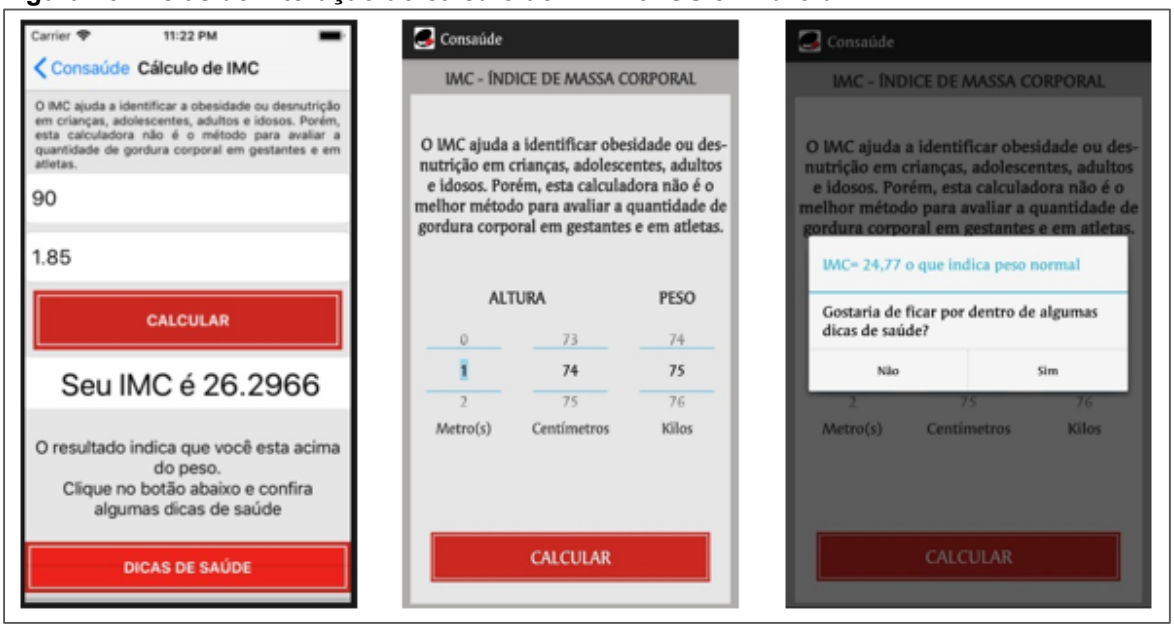
Figura 25 - Diagrama de sequência do IMC



Fonte: Autoria própria

A interação com o usuário foi implementada de uma maneira interativa com componentes existentes nas plataformas a fim de facilitar as inclusões dos valores para uma maior facilidade nos cálculos conforme ilustrado abaixo.

Figura 26 - Telas de interação do cálculo de IMC no iOS e Android



Fonte: Autoria própria

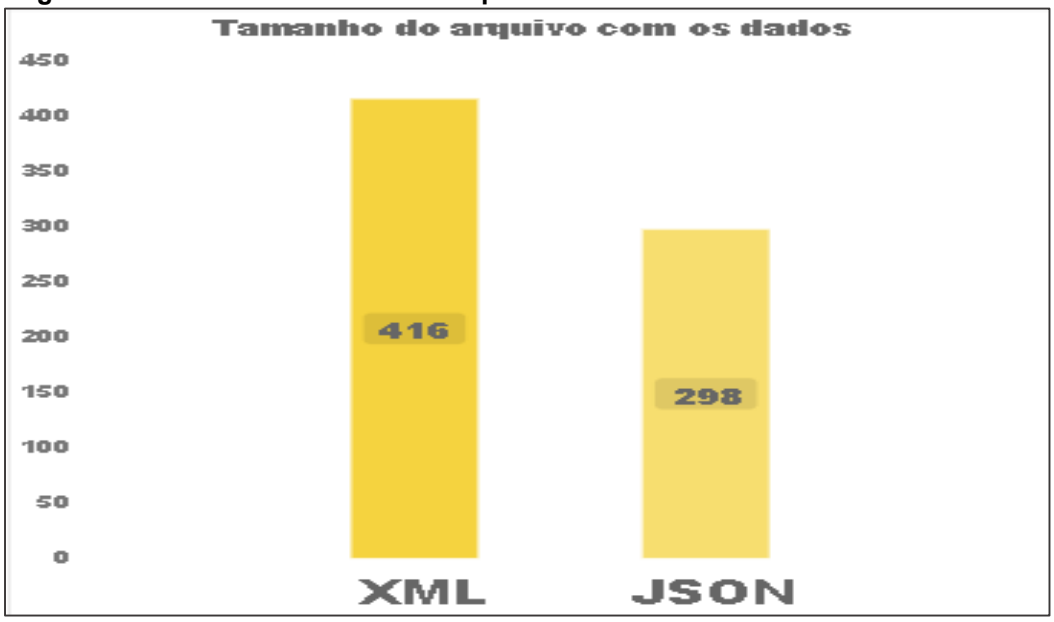
3.5.1.1 Comparativo de desenvolvimento ios x android x ionic

Após o projeto concluído nas duas plataformas nativas, foi desenvolvido uma versão utilizando a linguagem Ionic que permite o desenvolvimento multiplataforma de forma híbrida a fim de compararmos com os resultados obtidos com o desenvolvimento nativo.

O desempenho na inicialização, o tamanho da aplicação no dispositivo após instalado, o tamanho do arquivo contendo os dados pois foi utilizado xml e Json como banco de dados, o tamanho que a aplicação consome na memória do *rom* dispositivo, a quantidade de memória *ram* utilizada ao iniciar a e ao processar a exibição dos dados bem como o tempo de download e inserção dos dados.

Abaixo temos alguns resultados dos testes realizados, algumas categorias não foram possíveis obter os dados devido ao pouco conhecimento com benchmark.

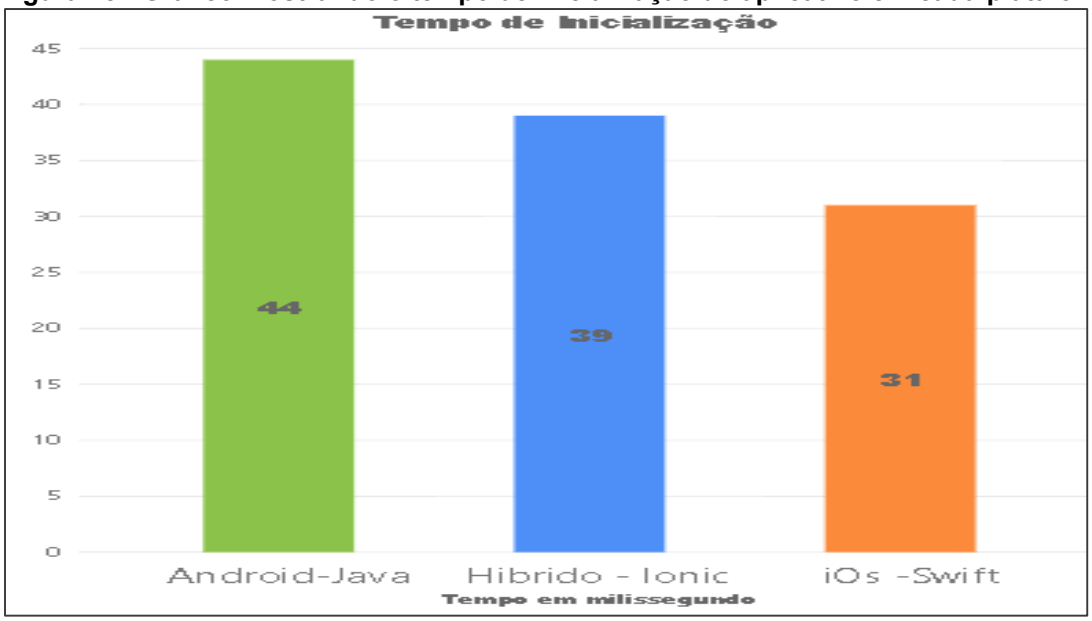
Figura 27- Gráfico do tamanho do arquivo com os dados XML / JSON



Fonte: Autoria própria

A Figura 27 apresenta a diferença do tamanho dos arquivos xml e json, essa diferença é por causa da estrutura do xml.

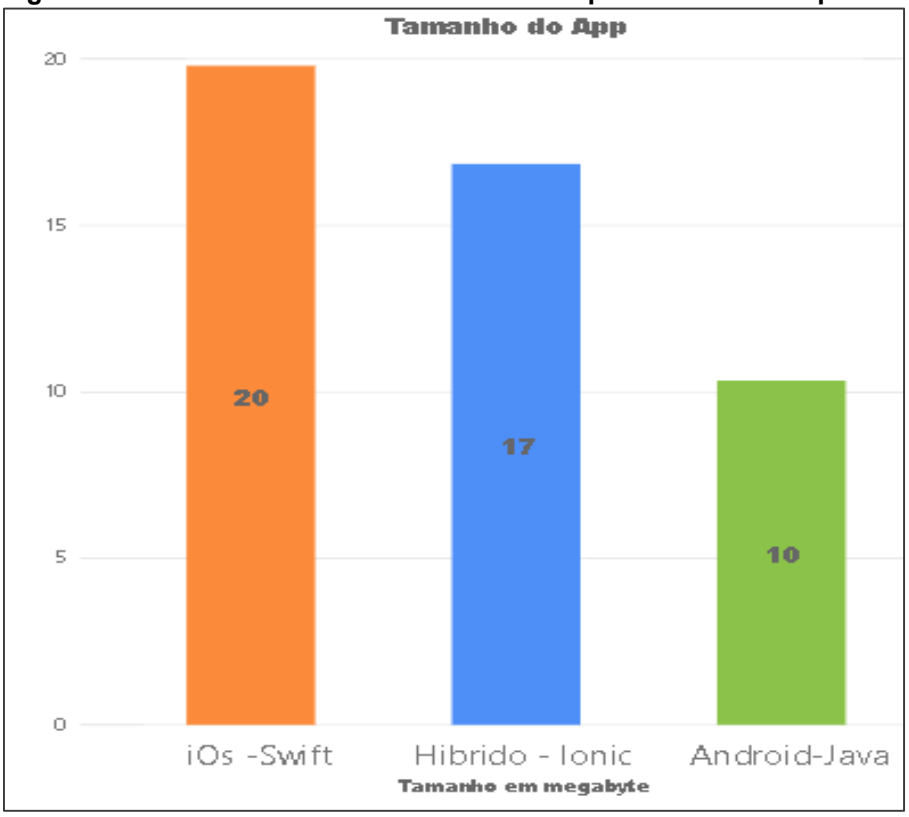
Figura 28 - Gráfico mostrando o tempo de inicialização do aplicativo em cada plataforma



Fonte: Autoria própria

A Figura 28 apresenta a diferença do tempo de inicialização do aplicativo em cada plataforma, vemos que o aplicativo hibrido fica na média de inicialização dos aplicativos desenvolvido de forma nativa.

Figura 29 - Gráfico mostrando o tamanho do aplicativo em cada plataforma

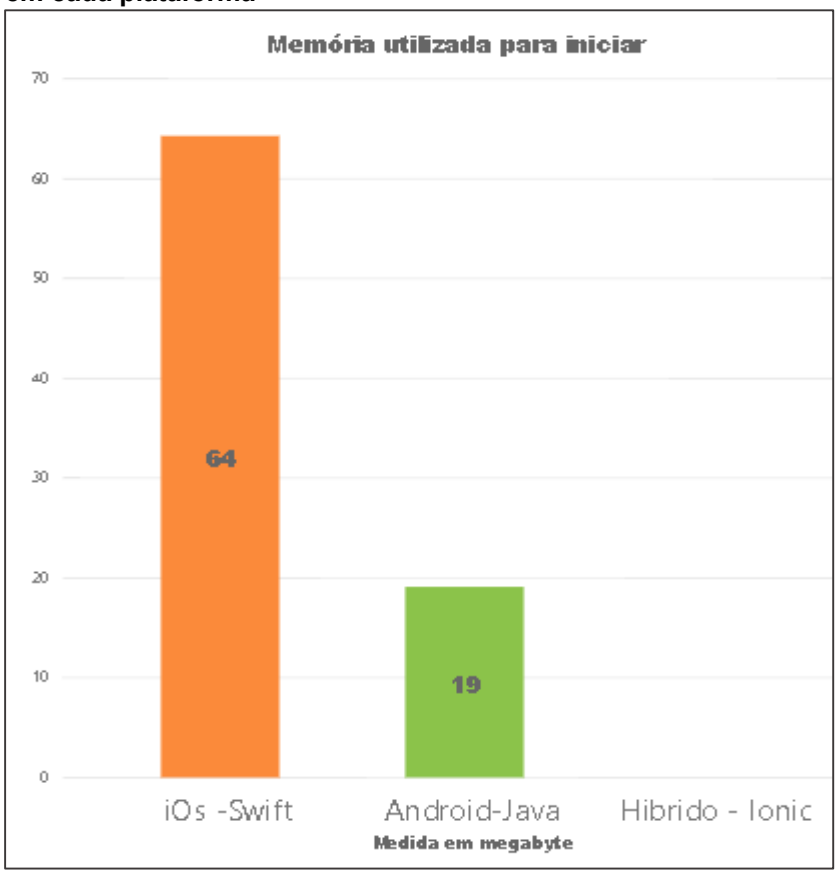


Fonte: Autoria própria

A Figura 29 apresenta a diferença do tamanho do aplicativo em cada plataforma, vemos que o aplicativo desenvolvido em Swift tem o dobro do tamanho do Java, isso se dá pois no swift uma imagem de resolução padrão tem um fator de escala de 1,0 e é referida como uma imagem @ 1x. Imagens de alta resolução têm um fator de escala de 2,0 ou 3,0 e são referidas como imagens @ 2x e @ 3x.

Então uma resolução de imagem padrão @ 1x seja 100×100px, por exemplo A versão @ 2x desta imagem seria 200x200px. A versão @ 3x seria 300x300px.

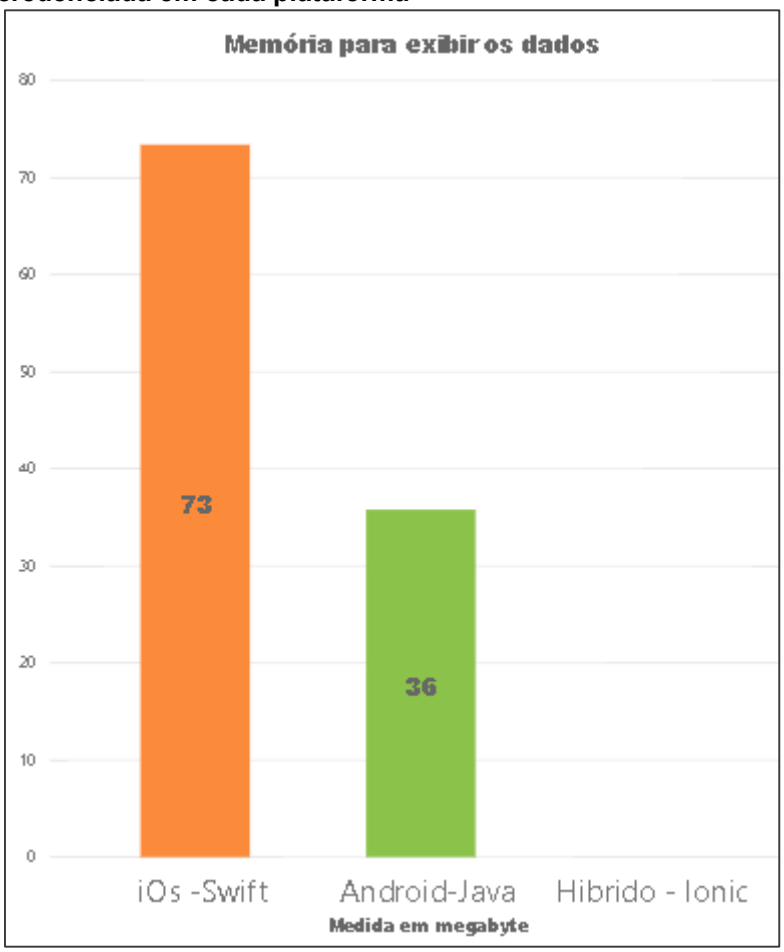
Figura 30 - Gráfico mostrando a memória utilizada para inicializar o aplicativo em cada plataforma



Fonte: Autoria própria

A figura 30 apresenta a memória utilizada para iniciar o aplicativo em cada plataforma, no aplicativo hibrido não foi possível obter essa informação de forma exata por isso ficou com valor nulo.

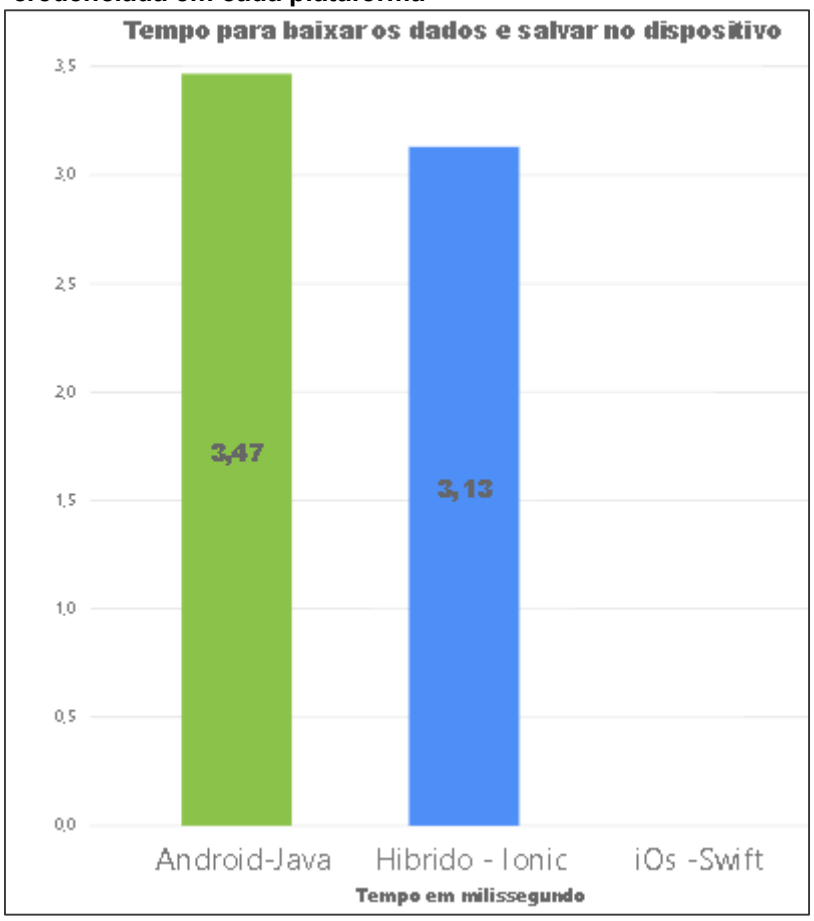
Figura 31 - Gráfico mostrando a memória utilizada para mostrar a rede credenciada em cada plataforma



Fonte: Autoria própria

A Figura 31 apresenta a memória utilizada para carregar a rede credenciada no aplicativo em cada plataforma, o aplicativo desenvolvido em Swift ocupa mais que o dobro da memória do desenvolvido em Java pois no Swift ele carrega toda a rede credenciada imediatamente no dispositivo e no Java ele carrega somente os dados de acordo com a busca do usuário, no aplicativo hibrido não foi possível obter essa informação de forma exata por isso ficou com valor nulo.

Figura 32 - Gráfico mostrando o tempo para baixar e salvar a rede credenciada em cada plataforma



Fonte: Autoria própria

A Figura 32 apresenta o tempo necessário para buscar os dados no webservice e salvar na memória do aplicativo em cada plataforma, no aplicativo Swift não foi possível obter essa pois o mesmo não salva os dados no dispositivo.

4 CONCLUSÃO

A seguir está apresentado as conclusões finais de todo e trabalho bem como funcionalidades adicionais disponíveis para implementações futuras.

4.1 CONSIDERAÇÕES FINAIS

O principal objetivo deste trabalho foi facilitar a comunicação e a atualização da rede credenciada de toda operadora disponibilizada aos beneficiários, e com a disponibilização destes dados diretamente no aplicativo de forma simples e objetiva a qualidade do atendimento ao beneficiário será melhorada.

A análise foi documentada com diagramas de casos de uso e de sequência, após isso as programações se iniciaram nas duas plataformas e todos os processos onde foram desenvolvidos em paralelo.

Alguns meses antes da aplicação finalizada foi trocado a diretoria da empresa a qual iria ser implementado, como objetivo é a redução de custos a aplicação novamente foi aceita e o processo de lançamento da aplicação em suas respectivas lojas está sendo prosseguida e a aplicação com custo zero para a empresa e para quem deseja instalar a aplicação em seu dispositivo.

A respeito do desenvolvimento podemos concluir que tanto o desenvolvimento nativo ou hibrido possuem seus pontos negativos e positivos; no desenvolvimento nativo temos a disposição todos os recursos da plataforma, porém na linguagem hibrida muitas das vezes nos deparamos com determinadas funcionalidades que tem seu desenvolvimento descontinuado e sendo assim tem que ser analisado qual equipe que está envolvida com a funcionalidade não poderá interromper o seu desenvolvimento.

Sendo assim não existe plataforma ideal e sim aquela que melhor se adapta a sua necessidade pois em termos de desempenho ambas se equiparam.

4.2 TRABALHOS FUTUROS

A implementação do *web service* para que o usuário realize a administração e atualização das telas do aplicativo; Também a implementação da revista eletrônica

com assuntos relacionados a área da saúde que hoje é impressa, sendo possível a leitura deste conteúdo via aplicativo.

Fica também em aberto a opção de implementação de agendamento online direto na tela do médico ou profissional selecionado onde o aplicativo exibirá as datas e horários disponíveis ao beneficiário este por sua vez escolher a melhor opção o aplicativo também poderá confirmar a presença do paciente na consulta a fim de evitar horários vagos na agenda do profissional.

Figura 33 - Revistas Consaúde

Fonte: Autoria própria

REFERÊNCIAS

ANDROID DEVELOPER. **Mudanças do Android 6.0 | Android Developers**. Disponível em: <https://developer.android.com/about/versions/marshmallow/android-6.0-changes.html>. Acesso em: 22 nov. 2017.

ANDROID DEVELOPERS. **Android Lollipop | Android Developers**, 2015. Disponível em: <https://developer.android.com/about/versions/lollipop.html>. Acesso em: 22 nov. 2017

APPLE. **Core ML | Apple Developer Documentation**. Disponível em: <https://developer.apple.com/documentation/coreml>.

APPLE INC. **Apple Developer Documentation**. Disponível em: <https://developer.apple.com/documentation/>. Acesso em: 22 nov. 2017.

BRITO, E. **Android 4.4 KitKat | Informática | TechTudo**. Disponível em: <http://www.techtudo.com.br/tudo-sobre/android-4-4.html>. Acesso em: 22 nov. 2017.

CAPELAS, B. **Até o fim de 2017, Brasil terá um smartphone por habitante, diz FGV - Link - Estadão**. Disponível em: <http://link.estadao.com.br/noticias/gadget,ate-o-fim-de-2017-brasil-tera-um-smartphone-por-habitante-diz-pesquisa-da-fgv,70001744407>. Acesso em: 22 nov. 2017.

GOOGLE. **Android Oreo**. Disponível em: <https://developer.android.com/about/versions/oreo/index.html>.

GOOGLE. **Android Developers**. Disponível em: <https://developer.android.com/about/versions/nougat/android-7.0.html>. Acesso em: 22 nov. 2017.

GROSSMANN, L. O. **Android e IOS estão em 99,6% dos smartphones vendidos - Convergência Digital - Negócios**. Disponível em: <http://www.convergenciadigital.com.br/cgi/cgilua.exe/sys/start.htm?UserActiveTemplate=site%2Cmobile&infoid=44572&sid=5>. Acesso em: 22 nov. 2017.

IDC. **IDC: Smartphone OS Market Share**. Disponível em: <https://www.idc.com/promo/smartphone-market-share/os>. Acesso em: 22 nov. 2017.

MILANI, A. **Programando para iPhone e iPad**. São Paulo, SP – Brasil: [s.n.].

OLIVEIRA, R. **Arquitetura da Plataforma Android**, 2013. Disponível em: <https://developer.android.com/guide/platform/index.html?hl=pt-br#linux-kernel>. Acesso em: 23 nov. 2017

SANTANA, R. C. Histórico Da Evolução . [s.d.].

Segurança do iOS. 2016.

SOFTWARE, E. DE. Estudo comparativo entre o desenvolvimento de aplicativos móveis utilizando plataformas nativas e multiplataforma. p. 103, 2016.