UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ELETRÔNICA E COMPUTAÇÃO
ENGENHARIA DE COMPUTAÇÃO

MODELAGEM E DESENVOLVIMENTO DE UM SISTEMA DE MONITORAMENTO REMOTO DE SINAIS VITAIS HUMANOS

TRABALHO DE CONCLUSÃO DE CURSO

José Augusto Comiotto Rottini

Santa Maria, RS, Brasil

2015

Modelagem e Desenvolvimento de um Sistema de Monitoramento Remoto de Sinais Vitais Humanos

José Augusto Comiotto Rottini

Trabalho de conclusão apresentado ao curso de Engenharia de Computação da Universidade Federal de Santa Maria (UFSM,RS), como requisito parcial para a obtenção do grau de **Engenheiro de Computação**.

Orientador: Prof. Dr. Mateus Beck Rutzig

Santa Maria, RS, Brasil

2015

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Engenharia de Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Conclusão de Curso

MODELAGEM E DESENVOLVIMENTO DE UM SISTEMA DE MONITORAMENTO REMOTO DE SINAIS VITAIS HUMANOS

elaborado por
José Augusto Comiotto Rottini

COMISSÃO EXAMINADORA:

_____
Prof. Dr. Mateus Beck Rutzig (UFSM)
(Presidente/orientador)

_____
Prof. Dr. Carlos Henrique Barriquello (UFSM)

_____
Prof. Dr. Cesar Augusto Prior (UFSM)

Santa Maria, 20 de novembro de 2015.

RESUMO

Trabalho de Conclusão de Curso
Curso de Engenharia de Computação
Universidade Federal de Santa Maria

MODELAGEM E DESENVOLVIMENTO DE UM SISTEMA DE MONITORAMENTO REMOTO DE SINAIS VITAIS HUMANOS

AUTOR: JOSÉ AUGUSTO COMIOTTO ROTTINI
ORIENTADOR: MATEUS BECK RUTZIG

Data e Local da Defesa: Santa Maria, 20 de novembro de 2015.

Nos centros de saúde atuais, as informações clínicas dos pacientes só estão acessíveis internamente e os equipamentos de monitoramento são restritos à sala em que estão instalados, exigindo que uma consulta médica seja realizada pessoalmente. A evolução da computação móvel propiciou a utilização de tais dispositivos nas mais variadas áreas, permitindo o proveito das vantagens da mobilidade que essa tecnologia oferece. Na medicina, destaca-se o telemonitoramento de pacientes, oportunizando o acesso a dados médicos independentemente da distância entre os profissionais da saúde e os pacientes. Este trabalho objetivou desenvolver um sistema hospitalar que faz uso de dispositivos móveis com plataforma Android e servidor web para explorar a mobilidade nesse ambiente. Em uma primeira etapa, foram avaliados os requerimentos para um sistema dessa natureza e se modelou o projeto de acordo com tais necessidades. A partir disso, realizou-se sua implementação e, por fim, verificou-se se foram atingidos os requerimentos funcionais e não funcionais levantados na fase de planejamento, considerando as limitações existentes nos dispositivos móveis. As análises de performance da aplicação Android revelaram ótimos resultados, mostrando que as limitações quanto ao consumo de bateria, memória e tráfego de dados da internet foram atendidas, o que garantiu a compatibilidade da aplicação e a possibilidade de sua utilização durante longos períodos. Obteve-se, ao final, um sistema hospitalar robusto e com capacidade para satisfazer as necessidades de seus usuários, visando a acessibilidade às informações médicas e promovendo a eficiência dentro das instituições hospitalares.

Palavras-chave: Sistema Hospitalar. Telemonitoramento. Dispositivos Móveis. Android. Web Server.

ABSTRACT

Undergraduate Final Work
Computer Engineer
Federal University of Santa Maria

MODELING AND DEVELOPMENT OF A REMOTE MONITORING SYSTEM OF HUMAN VITAL SIGNS

AUTHOR: JOSÉ AUGUSTO COMIOTTO ROTTINI
COUNSELLOR: MATEUS BECK RUTZIG
Date and Location of Defence: Santa Maria, 20 November 2015.

In the current healthcare centers, clinical information about patients is accessible only internally and monitoring equipments are restricted to the room where they are installed, requiring a medical appointment to be taken in person. The evolution of mobile computing allowed the use of such devices in the most varied fields, bringing all the mobility benefits that these technologies have to offer. In medicine, special interest is given to patient telemonitoring, through which medical data can be directly accessed regardless of the distance between health professionals and patients. The present study aimed the developing of a hospital system that makes use of mobile devices equipped with Android platform and web server to explore mobility in this environment. The first stage comprised an evaluation of the requirements for such a system, and the designing of a project according to those needs. Based on that, the implementation took place and, finally, it was verified if the functional and nonfunctional requirements raised in the planning stage were met, given the limitations found in mobile devices. The performance analysis of the Android application revealed excellent results, showing that the limitations on battery power, memory and internet data traffic were respected, ensuring the compatibility of the application and the possibility of use for long periods of time. The final result was a robust hospital system capable of meeting the needs of its users, in order to provide accessibility to medical information and promote efficiency within the health institutions.

Key-words: Hospital System. Telemonitoring. Mobile Devices. Android. Web Server.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

AOT – *Ahead-Of-Time*

API – *Application Programming Interface*/ Interface de Programação de Aplicação

APK – *Android Package*

ART – *Android Runtime*

CID – Classificação Internacional de Doenças

DEX – *Dalvik Executable*/ Executável Dalvik

DPOC – Doença Pulmonar Obstrutiva Crônica

DVM – *Dalvik Virtual Machine*/ Máquina Virtual Dalvik

EULA – *End User License Agreement*/ Contrato de Licença de Usuário Final

GCM – *Google Cloud Messaging*

GSM – *Global System for Mobile Communications*/ Sistema Global para Comunicações Móveis

HTTP – *Hypertext Transfer Protocol*/ Protocolo de Transferência de Hipertexto

HTTPS – *Hypertext Transfer Protocol Secure*/ Protocolo de Transferência de Hipertexto Seguro

IDE – *Integrated Development Environment*/ Ambiente de Desenvolvimento Integrado

JIT – *Just-In-Time*

JSON – *JavaScript Object Notation*/ Notação de Objetos JavaScript

JSP – *Java Server Pages*

JVM – *Java Virtual Machine*/ Máquina Virtual Java

OHA – *Open Handset Alliance*

SDK – *Software Development Kit*/ Kit de Desenvolvimento de Software

SMS – *Short Message Service*/ Serviço de mensagens curtas

SPP – *Serial Port Profile*

TTS – *Text-To-Speech*/ Texto para Voz

UML – *Unified Modeling Language*/ Linguagem de Modelagem Unificada

URL – *Uniform Resource Locator*/ Localizador Padrão de Recursos

UTI – Unidade de Terapia Intensiva

LISTA DE APÊNDICES

SUMÁRIO

1. INTRODUÇÃO

O corpo humano, em sua atividade normal, desenvolve uma série de processos fisiológicos. Esses processos consistem em atividades químicas e elétricas que, de forma direta ou indireta, podem ser mensuradas fazendo-se uso de sensores (POTTER; PERRY, 2009). Em um ambiente hospitalar é necessário o constante monitoramento dos sinais fisiológicos dos pacientes, como temperatura, pressão arterial, taxa de batimento cardíaco, entre outros. Tais parâmetros permitem avaliar as funções corporais básicas e, desta forma, diagnosticar diversas patologias, assim como inferir o seu tratamento adequado. Além disso, permite acompanhar a resposta do paciente a esses tratamentos e a evolução do seu quadro clínico.

A saúde, uma das principais preocupações da humanidade, sempre esteve relacionada com o desenvolvimento tecnológico. As últimas conquistas em diferentes campos da tecnologia permitiram um grande avanço no desenvolvimento de sistemas de aquisição de dados médicos, os quais foram possíveis graças à evolução da microeletrônica e à acessibilidade a tais equipamentos, impulsionando as pesquisas no segmento. Com o surgimento da nanotecnologia, que permitiu a construção de estruturas em escala atômica, houve a miniaturização dos dispositivos - cada vez menores e mais potentes - originando os computadores portáteis. De forma simultânea, outra tecnologia que excluía a utilização de cabos para a transmissão de dados foi sendo aprimorada e implantada. A partir da integração dessas duas frentes, surgiu a computação móvel (MARTINS, 2012).

Com base na utilização de dispositivos portáteis e comunicação sem fio, a computação móvel proporciona que os usuários tenham acesso a serviços, independente da sua localização, introduzindo a mobilidade como foco principal nesses sistemas. Com a diminuição dos custos dos dispositivos, a computação móvel se tornou viável não somente para o segmento empresarial, mas para as pessoas de uma forma geral. Dirigida à área da medicina, a computação móvel, entre outras aplicações, apresenta o telemonitoramento de pacientes. Por meio desse, os parâmetros de saúde de um paciente podem ser controlados de forma remota, ou seja, pode-se ter acesso a essas informações, em tempo real, de outra localidade. Esse conceito possibilitou a transmissão dos sinais biológicos a grandes distâncias, trazendo inúmeras vantagens aos pacientes e profissionais da saúde (SUNDARAM, 2013).

Nos últimos anos, tem-se visto a aplicação crescente das tecnologias da comunicação

na área da saúde. Inclusive, smartphones de última geração possuem, nativamente, sensores fisiológicos acoplados. Há também diversos aplicativos e módulos adicionais que podem ser conectados ao smartphone para realizar exames cada vez mais sofisticados como, por exemplo, lentes adaptadas para realizar exames de visão. Na mesma linha, os smart watches vem ganhando espaço no mercado, unindo o conceito de dispositivos vestíveis com o uso de sensores.

Nesse sentido, este projeto visa o desenvolvimento de um sistema de monitoramento de pacientes e gerenciamento de dados médicos em um ambiente hospitalar. Por meio do uso da plataforma Android e servidor web, busca-se explorar o telemonitoramento de pacientes de forma a garantir os cuidados necessários ao paciente mesmo com a mobilidade dos funcionários envolvidos (por exemplo: médicos, enfermeiros, etc.). A implementação de um banco de dados que armazena não só os parâmetros monitorados, mas também diversas variáveis pertinentes ao quadro clínico dos enfermos, proporciona a centralização dos dados e a acessibilidade dos mesmos para futuras pesquisas que buscam a evolução dos protocolos de tratamento das doenças.

1.1. Motivação

Os equipamentos de medição de sinais fisiológicos encontrados atualmente em centros de saúde são frequentemente complexos e dispendiosos. Aliado a isso, a utilização de fios para seu funcionamento faz com que fiquem restritos a sala em que estão instalados, não se empenhando nas questões de mobilidade aqui introduzidas. Ademais, os dados monitorados são exibidos apenas no próprio aparelho, necessitando que um profissional de saúde se desloque até a sala do paciente para verifica-los. Em caso de emergência, ocorre a emissão de um aviso sonoro que serve o propósito de alertar a equipe médica. Entretanto, a condição atual do paciente só é confirmada quando algum profissional chega no ambiente onde o paciente se encontra.

Em grande parte dos centros de saúde atuais, nem todas as informações a respeito do quadro clínico do paciente são informatizadas, sendo mantidas em registros em papel. Em instalações mais modernas, onde esses dados são informatizados, os mesmos só estão acessíveis internamente ao hospital. Com isso, para o médico verificar essas informações ele

deve estar no local e acessar essas informações por meio de um computador, muitas vezes na central do hospital. Esse fato acaba por diminuir a acessibilidade a tais dados, sendo necessário tarefas extras para sua obtenção e manipulação. Ademais, para solicitar que a equipe médica realize um procedimento é necessária a presença destes funcionários, acarretando em um tempo maior de reatividade que pode causar pioras no quadro clínico do paciente.

Visto isso, destaca-se a importância do desenvolvimento de um sistema que propicie a mobilidade da equipe médica, garantindo o fácil acesso aos dados e promovendo um meio de comunicação entre as equipes. O envio de alertas médicos diretamente ao smartphone consiste em uma forma eficaz de atendimento e de exibição de informações extras, como detalhar a condição que o paciente está enfrentando. Assim, os profissionais responsáveis podem realizar o atendimento remoto prescrevendo medicação, ou até mesmo in loco mais preparados, aumentando a efetividade da operação.

1.2. Descrição da tarefa

Baseado na seção de motivação é possível formular os objetivos e definir o escopo do projeto. Combinando técnicas de comunicação via rádio e os últimos desenvolvimentos no mercado de dispositivos móveis, consegue-se projetar um sistema que englobe todas as informações necessárias para a análise médica e propicie as vantagens da mobilidade nesse ambiente. O diagrama da Figura 1 retrata a visão geral do sistema. Inicialmente, será descrito o seu funcionamento como um todo e, após, será definida as tarefas que competem a este projeto.

O paciente tem os seus sinais vitais mensurados por intermédio de sensores. Os dados obtidos são repassados a um microcontrolador, localizado próximo ao paciente, que fará o processamento dessas informações. Após, esse microcontrolador fará o envio dos dados, via rádio a um outro microcontrolador, situado na central do hospital, o qual tem a função de se comunicar com o servidor para a inserção das informações no sistema.

O projeto conta com um banco de dados que armazenará todas as informações pertinentes a um ambiente hospitalar. Os dados armazenados consistem nas variáveis relevantes ao quadro clínico dos pacientes e detalhes da sua internação no hospital, além de informações dos usuários do sistema.

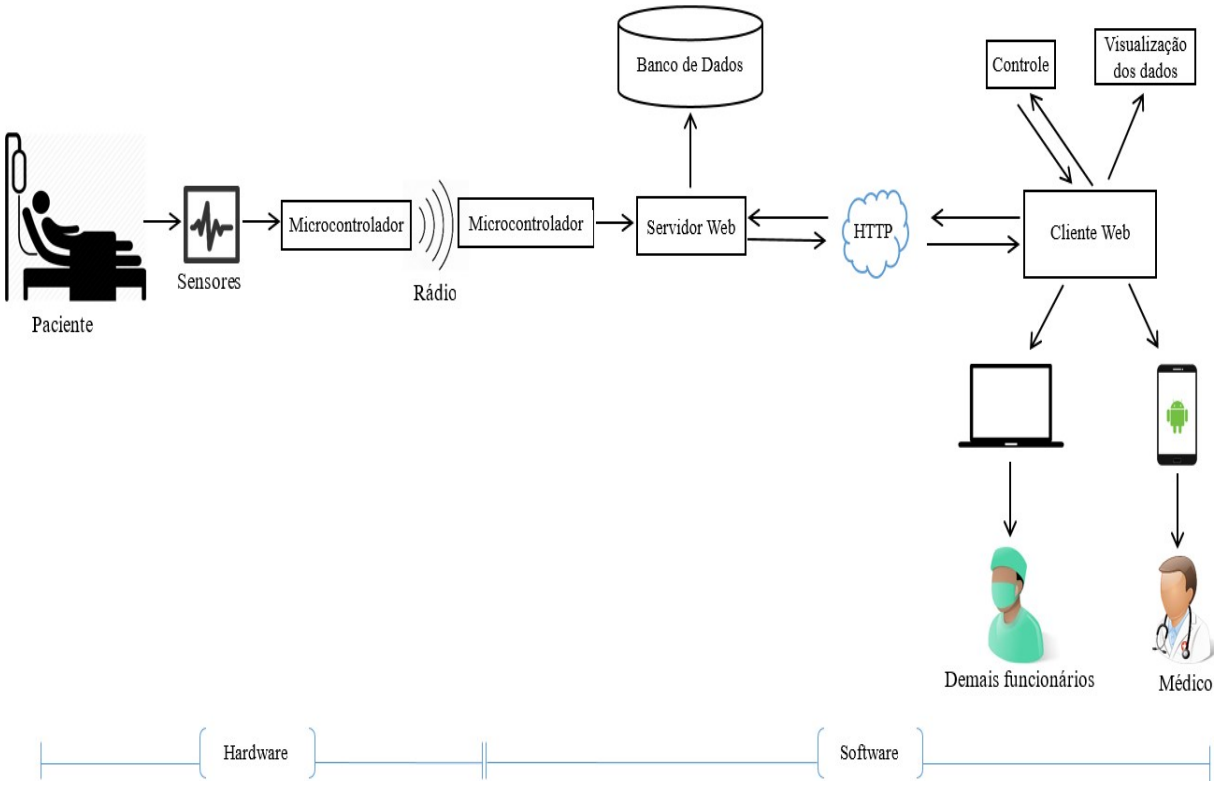


Figura 1 - Visão geral do sistema.

O servidor será responsável por fazer a verificação dos sinais vitais vindos dos microcontroladores. Ou seja, a tarefa do servidor consiste em averiguar se as leituras estão dentro dos valores normais para o paciente, e inseri-los no banco de dados. Também, é incumbido por processar as requisições realizadas pelas aplicações clientes, onde a comunicação ocorrerá utilizando uma conexão com a internet e protocolo HTTP (*Hypertext Transfer Protocol)*.

Duas aplicações clientes são destinadas à interação dos usuários com o sistema: uma aplicação para a plataforma Android e um website. A aplicação Android é direcionada aos médicos e deve fornecer a visualização das principais informações clínicas do paciente e a possibilidade de monitoramento, em tempo real, dos seus sinais vitais. A ideia é que, com esses dados, seja possível a realização de uma consulta médica sem a necessidade de estar no local. Assim, a aplicação deve implementar ainda funções para diagnosticar e registrar uma evolução do quadro clínico do paciente. Caso o médico julgue necessário a realização de um procedimento, seja um exame ou a administração de um medicamento, haverá uma função para realizar uma solicitação à equipe de enfermagem que se encontra no hospital no momento. Ainda, a aplicação deverá receber notificações quando uma solicitação for

respondida pela equipe de enfermagem ou quando um sinal vital de um paciente estiver fora dos valores normais.

O website é voltado para os demais funcionários que interagirão com o sistema, isto é, enfermeiros, recepcionista e administradores do hospital. As responsabilidades desses funcionários necessitam que os mesmos estejam no hospital para a realização de suas tarefas. Por isso, não é necessário o uso de um dispositivo móvel para sua interação com o sistema. Todavia, o website deve fornecer uma boa experiência em telas menores caso seja feita a opção por esse tipo de acesso. Basicamente, as recepcionistas gerenciarão as informações dos pacientes; os enfermeiros terão acesso às solicitações feitas pela equipe médica, podendo inserir uma resposta para a mesma após sua conclusão, havendo ainda a possibilidade de, assim como o médico, registrar a evolução de um paciente; os administradores do hospital, por sua vez, farão o gerenciamento dos usuários do sistema e dos dados registrados nele, em geral.

Este trabalho tem como objetivo o desenvolvimento da parte de software do sistema descrito. Ou seja, o projeto se constitui na implementação de um banco de dados, servidor web, aplicação Android e website, assim como a comunicação entre aplicações clientes e servidor. Dessa maneira, não engloba os detalhes de hardware, partindo do ponto que os dados dos sensores já foram obtidos e estão prontos para serem repassados ao servidor.

Durante a elaboração do projeto, deu-se atenção especial a etapa de modelagem. Os requisitos para um sistema dessa natureza foram levantados e o seu comportamento foi retratado através de modelos específicos. Essa aproximação propicia a efetuação de um sistema que atinge os requerimentos necessários e a obtenção de uma documentação completa sobre o trabalho desenvolvido.

1.3. Organização do texto

Este trabalho está estruturado em 8 capítulos, cuja as ideias centrais estão resumidas a seguir.

Capítulo 1: apresenta a introdução ao projeto que está sendo desenvolvido. Nessa seção, enquadra-se o tema dentro do contexto atual da tecnologia, elucidando a motivação para a sua realização e delimitando o escopo do trabalho.

Capítulo 2: fornece uma visão geral dos estudos realizados nas áreas de

monitoramento remoto de pacientes e sistemas voltados para ambientes hospitalares.

Capítulo 3: esclarece os conceitos gerais relacionados ao trabalho que está sendo desenvolvido, abordando questões em nível de saúde e também de tecnologias utilizadas no projeto.

Capítulo 4: introduz os principais aspectos da plataforma Android, discorrendo a respeito da sua arquitetura e o processo de desenvolvimento de aplicações para o sistema.

Capítulo 5: apresenta a etapa de modelagem do sistema proposto. Os modelos construídos durante essa fase são introduzidos por meio de pesquisas bibliográficas, descrevendo em detalhes a metodologia utilizada e a relevância de cada um dentro do planejamento do projeto.

Capítulo 6: fornece detalhes de como ocorreu a implementação do sistema, elucidando as decisões de projeto e minuciando os conceitos por trás desse processo.

Capítulo 7: oferece uma descrição da etapa de teste do sistema e expõe uma análise de desempenho da aplicação Android desenvolvida.

Capítulo 8: apresenta as conclusões finais e discussão de trabalhos futuros.

2. TRABALHOS RELACIONADOS

O progresso da tecnologia propiciou a modernização nos sistemas médicos e diversos trabalhos relacionados foram desenvolvidos. A aquisição de parâmetros vitais de forma confiável é uma busca constante por parte da indústria médica, sendo o alvo de muitos desses estudos. No contexto específico de monitoramento de parâmetros fisiológicos podem ser encontradas diferentes abordagens, desde o uso em esportes, por meio da avaliação do desempenho físico através desses parâmetros, até a utilização de sinais neurológicos para o controle de dispositivos eletrônicos (RAMINHOS, 2009). Este capítulo tem como objetivo fornecer uma visão geral dos estudos realizados na área, com foco nos sistemas de monitoramento remoto de pacientes e sistemas voltados para ambientes hospitalares. Uma seção à parte traz a revisão bibliográfica sobre o processo de modelagem de softwares voltados para fins similares.

2.1. Monitoramento remoto de pacientes

O monitoramento dos parâmetros vitais de forma remota visa diminuir a lacuna entre pacientes e profissionais da saúde por meio da possibilidade da transferência de dados médicos mesmo em grandes distâncias. Nesse contexto, Prakash e Venkatesh (2013) propõem um sistema de monitoramento de sinais cardíacos voltado para pacientes domiciliares utilizando uma rede de sensores sem fio. O objetivo do projeto é reduzir ao máximo os custos do mesmo através da eliminação da necessidade de um computador para realizar o envio dos dados coletados. A figura 2 exibe uma visão geral do sistema desenvolvido.

O sistema consiste em duas partes distintas, sendo uma composta pelos elementos na casa do paciente e a outra localizada no hospital. No lado do paciente, os sinais cardíacos são capturados por meio de sensores e processados através de um filtro passa-baixa e um amplificador, resultando em um sinal de pouco ruído e pronto para ser digitalizado. A conversão de analógico para digital é realizada utilizando um microprocessador PIC com uma resolução de 10 bits. Usando transceptores CC2500 de 2.4GHz, os dados são enviados a uma base (que também utiliza PIC), localizada na própria casa. Com o auxílio de um conversor

Serial para Ethernet, envia-se os dados para o servidor do hospital.

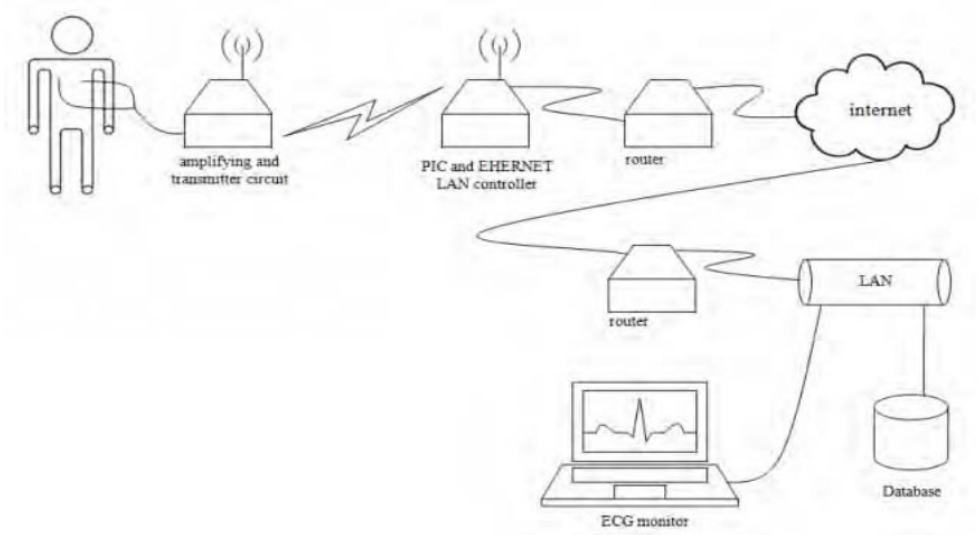


Figura 2 - Sistema projetado por Prakash e Venkatesh.
Fonte: PRAKASH; VENKATESH, 2013.

Na parte do hospital, um software desenvolvido em C++ realiza a coleta e armazenamento dos dados em um banco de dados SQL Server. O software é baseado nos protocolos TCP/IP e realiza a leitura dos dados em intervalos definidos. O banco de dados implementado é constituído por apenas duas tabelas: uma para as informações pessoais do paciente e outra para o armazenamento dos sinais adquiridos. Outro software, implementado utilizando JSP (*Java Server Pages*), é responsável pela visualização dos dados armazenados no sistema. Através desse, é possível adicionar, modificar e deletar informações de um paciente, assim como visualizar graficamente os seus sinais cardíacos.

Por não necessitar de um computador ou dispositivo móvel para realizar o envio dos dados para o hospital, conseguiu-se atingir o objetivo de reduzir os custos do sistema. Em contrapartida, fica clara a limitação do mesmo em questão de funcionalidades. A opção de acesso e visualização dos dados do sistema fica restrita à área do hospital. Além disso, apenas as informações básicas são armazenadas no banco de dados o que reduz o potencial do sistema projetado.

As facilidades obtidas com o uso de dispositivos móveis inovaram a forma de aquisição, armazenamento e transferência de informações iátricas (SIQUEIRA, 2007). Nesse tema, Koshmak (2011) apresenta um sistema de monitoramento e alarme para pacientes com doenças crônicas obstrutivas. Para obter as informações de saúde de um paciente, é utilizado o dispositivo Nonin WristOx2 3150, que possui sensores capazes de obter o pulso e a saturação

de oxigênio no sangue do usuário e funciona como um dispositivo escravo. A conexão com um dispositivo mestre ocorre via Bluetooth, o qual possui um alcance máximo de 10 metros. Após a iniciação do aparelho, o mesmo se encontra detectável por 2 minutos, período no qual enviará um *broadcast* com a sua identificação. Para parear um dispositivo com o Nonin, o mestre deve enviar uma solicitação a ele durante esse período. Uma vez pareado, a conexão é estabelecida e o Nonin envia os dados adquiridos de forma contínua ao mestre utilizando protocolo SPP (*Serial Port Profile*). No caso desse projeto, o dispositivo mestre é um smartphone que utiliza a plataforma Android.

Os dados provenientes dos sensores são então repassados ao dispositivo móvel, onde uma aplicação desenvolvida para o projeto tem a função de adquirir esses dados e armazená-los no aparelho. Além disso, a aplicação obtém a informação proveniente do seu acelerômetro interno, a qual permite avaliar a atividade do paciente durante o período de monitoramento. O funcionamento da aplicação é bastante simples e a mesma contém duas *Activities* e uma classe de serviço responsável por estabelecer a comunicação Bluetooth.

Um aspecto positivo desse trabalho é a análise dos dados coletados utilizando técnicas de processamento e algoritmos para a detecção de anomalias, visando a confiabilidade na detecção e envio de alertas. A primeira técnica consiste em calcular o valor médio para um parâmetro e compará-lo com o valor atual mensurado. Um paciente saudável normalmente mantém seus sinais vitais em uma faixa regular de valores. É supérfluo analisar uma situação onde não há nada de errado com o paciente e todos os seus parâmetros se encontram dentro das faixas normais. Essa medida é extremamente importante para o sistema, uma vez que se trata de um procedimento simples e evita um processamento mais pesado sobre os dados sem necessidade, economizando o consumo de memória e, consequentemente, de energia.

Caso haja uma diferença considerável entre os valores, uma segunda técnica de verificação é utilizada. Os sinais armazenados são divididos em frames de mesmo comprimento e, para cada frame, a média dos valores é calculada. Então, forma-se um vetor com os valores das médias de cada frame. Depois, com o intuito de detectar anomalias, um algoritmo de força bruta busca por discordâncias nos dados desse vetor.

A pulsação do paciente tem relação direta com a atividade do mesmo. Ao se movimentar, a tendência é que a pulsação aumente. Por meio dos dados obtidos do acelerômetro do smartphone, é possível relacionar a informação da atividade do enfermo com os valores dos sinais mensurados. Com isso, obtém-se uma análise que retrata as reais condições do paciente, permitindo que a equipe médica consiga chegar a um diagnóstico mais preciso.

O projeto foi testado em pacientes reais e se mostrou eficiente na detecção de anomalias nos sinais monitorados. Contudo, deve-se frisar que os resultados apresentaram uma pequena divergência entre os pacientes – o que para um foi considerado anomalia, para outro não. Esse fato está relacionado ao número de variáveis que afetam o valor desses sinais e também a condição de saúde do paciente. Devido a isso, torna-se difícil a implementação de um único algoritmo que consiga cobrir todos os cenários para chegar a um resultado definitivo.

Focando-se ao monitoramento de pacientes em ambientes hospitalares, alvo desse estudo, Indumathy e Patil (2014) desenvolveram um sistema de alerta para monitoramento remoto de pacientes. Um microcontrolador Philips P89V51RD2 é conectado a sensores de temperatura, frequência cardíaca e de piscar de olhos e é responsável por fazer a conversão analógica-digital do sinal proveniente desses sensores. A ideia de utilizar o sensor de piscar de olhos é ajudar pacientes em processo pós-operatório, com dificuldades para se movimentar e falar, a chamar a atenção da equipe médica através do movimento dos olhos. O microcontrolador é programado em linguagem C e nele se encontra gravado o número dos celulares da equipe médica. Por meio de um módulo GSM (*Global System for Mobile Communications)* conectado ao microcontrolador, os dados são enviados para os smartphones via SMS (*Short Message Service*). A mensagem enviada contém uma palavra chave referenciando o sinal medido seguido do seu valor.

Uma aplicação desenvolvida para Android recebe as mensagens via SMS. A aplicação possui valores de *threshold* para cada uma das medidas. Ou seja, existem valores limites definidos para cada sinal. Assim, pega-se a palavra chave presente no conteúdo da mensagem e se compara o valor recebido na mensagem com o valor de *threshold* gravado na aplicação. A figura 3 retrata a sequência de eventos do sistema do ponto de vista do dispositivo móvel.

Se o valor recebido via SMS se encontra dentro da faixa normal para ele definido, o mesmo é gravado em um banco de dados SQLite do Android. Caso contrário, antes da gravação no banco de dados um alerta de voz é gerado utilizando uma ferramenta TTS (*Text-To-Speech*), avisando o usuário do problema encontrado. Por fim, utilizando uma requisição HTTP Post, os dados do banco de dados interno ao Android são enviados a um servidor PHP, o qual armazenará as informações em um banco de dados MySQL. Além disso, uma página criada utilizando HTML permite que os médicos façam login no sistema para a visualização dos dados armazenados.

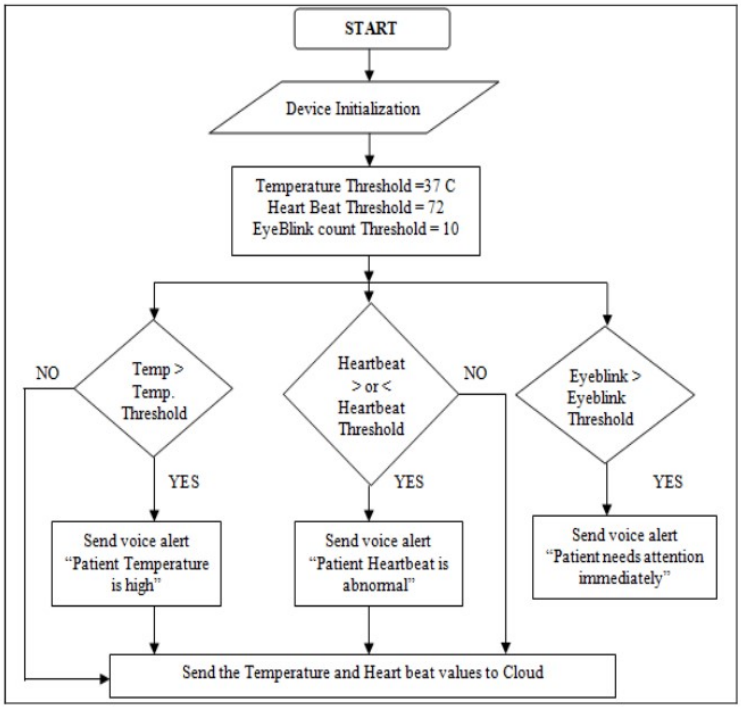


Figura 3 - Fluxo de eventos da aplicação desenvolvida por Indumathy e Patil.
Fonte: INDUMATHY; PATIL, 2014.

O papel do dispositivo móvel no sistema foi, de certa forma, subestimado. Sua função é focada apenas na emissão de alertas, não oferecendo nenhum tipo de interação extra com o usuário. Para visualizar os dados do sistema, o médico necessita utilizar um navegador, já que o aplicativo desenvolvido não suporta essa função. Além disso, as informações armazenadas no sistema são extremamente básicas, contendo apenas os dados pessoais do paciente e o valor dos sinais mensurados. Assim, não há qualquer informação a respeito dos sintomas, diagnóstico, histórico e evolução do paciente.

Pacientes com problemas cardíacos precisam realizar a leitura do nível de glicose várias vezes ao dia. Pelo método tradicional, que envolve a retirada de sangue, torna-se um processo cansativo e que demanda muito tempo. Com base nisso, Murakami, Gutierrez, *et al.*, (2006) desenvolveram um sistema de monitoramento de glicose em tempo real, chamado vMonGluco. A figura 4 retrata a arquitetura do sistema elaborado.

Para a leitura da glicose foi utilizado um monitor de glicose comercial, o MiniMed, composto por um sensor subcutâneo conectado a um monitor do tamanho de um *pager*. O MiniMed não foi projetado para gerar informações em tempo real, visto que a calibração do equipamento é realizada apenas uma vez, ao final do período de uso. Para utilizá-lo em um sistema de tempo real é necessário fazer o download das leituras a cada 5 minutos e calibrá-lo

a cada nova leitura. Para atender a esses requisitos, foi desenvolvido um software para Pocket PC.

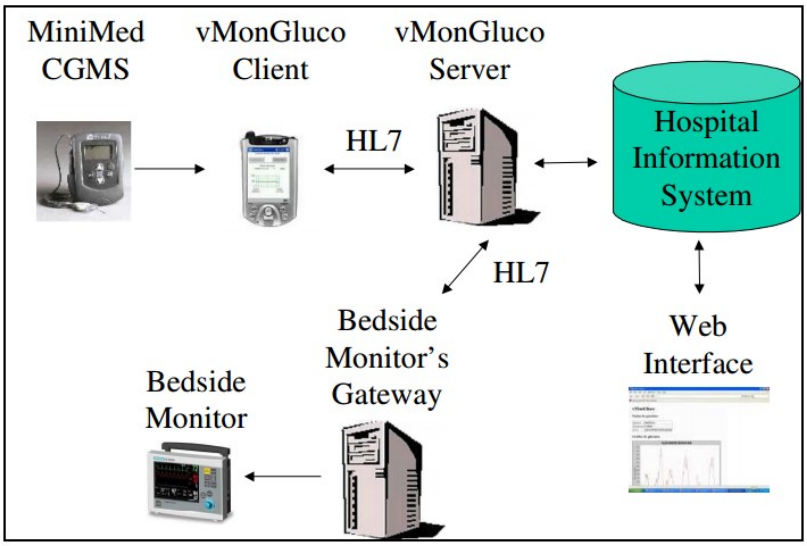


Figura 4 - Arquitetura do vMonGluco.

Fonte: MURAKAMI, GUTIERREZ, *et al.*, 2006.

O Pocket PC é conectado a estação de comunicação do MiniMed através de um cabo serial. O software implementado é responsável por receber periodicamente os dados provenientes do monitor - que consistem em sinais elétricos – e traduzi-los em concentração de glicose no sangue. Além disso, tem a função de calibrar e enviar os dados a um servidor. Esse último processo ocorre utilizando o protocolo HL7, padrão para integração de sistemas médicos.

O servidor transmite as leituras para um monitor Siemens, que fica ao lado da cama do paciente, e também as armazena no sistema. Desta forma, a informação pode ser facilmente acessada através de uma conexão com a Web.

O projeto possibilitou que pacientes sejam monitorados de forma automatizada e com uma frequência maior de amostragem, representando um grande avanço para o segmento. O sistema foi utilizado em UTI (Unidade de Terapia Intensiva) e apresentou resultados satisfatórios. Salienta-se, entretanto, que para valores de glicose menores que 100mg/dl e maiores que 250mg/dl, obteve-se os maiores desvios nos resultados entre a medida contínua obtida pelo sistema projetado e a realizada pelo método tradicional. Segundo o autor, o problema pode ser explicado pela imprecisão do método tradicional, porém um estudo mais amplo a respeito do assunto deve ser realizado para uma análise conclusiva.

2.2. Modelagem de sistemas

Por meio da utilização de vários modelos é possível descrever um sistema sob diferentes aspectos (SOUZA; SOUZA, 2014), com isso a complexidade do sistema é particionada, facilitando a sua compreensão. Nessa linha, Dzubow, Domenic e Slough (2011) apresentam o projeto de um sistema voltado para pequenas clínicas, as quais possuem requerimentos mais simples, envolvendo tarefas básicas da parte administrativa e armazenamento das informações relacionadas à condição do paciente. As necessidades para uma clínica desse porte foram devidamente avaliadas e elaborou-se um sistema que atenda às exigências da mesma. O processo de modelagem foi o foco do estudo, sendo muito bem detalhado e discutido ao longo do mesmo.

Primeiro, elaborou-se um modelo de casos de uso, onde as funcionalidades do sistema foram definidas de acordo com as necessidades do mesmo. Então, por meio de um diagrama de classes as entidades e suas relações foram retratadas, assim como suas responsabilidades dentro do sistema. A partir de então o foco passou para o aspecto dinâmico. Ou seja, modelou-se a interação dos usuários com o sistema, fazendo uso de diagramas de sequência para demonstrar esse comportamento. Por fim, construiu-se um modelo que representa o banco de dados, contendo todas as informações que devem ser armazenadas e evidenciando como os dados serão acessados. Ao final do trabalho, obteve-se um modelo que descreve os reais requerimentos e funcionalidades do sistema, aproximando-se do modelo real.

A popularização dos smartphones e a facilidade de disponibilização dos softwares para o público em geral impulsionou o desenvolvimento de aplicações para dispositivos móveis, sendo a área da saúde uma das mais exploradas. Kautzmann (2012) propõe o desenvolvimento de uma aplicação Android para acesso ao prontuário médico. O objetivo da ferramenta é disponibilizar algumas informações básicas sobre o estado de saúde do paciente antes mesmo que este chegue ao hospital. Desta forma, agiliza-se o atendimento e evita-se a lotação nos centros de saúde.

Ao iniciar o aplicativo, a tela de login é exibida. Após realizar o login, tem-se uma tela de menu com as opções de cadastrar um paciente, pesquisar e visualizar agenda. Caso a opção de pesquisar um paciente seja a escolhida, uma lista de pacientes cadastrados abrirá. Clicando sobre um dos pacientes, suas informações pessoais são exibidas na tela (Figura 5). Para acessar o menu de ações do paciente (Figura 6), deve-se pressionar o botão voltar a partir dessa tela.

Figura 5 - Tela de dados do paciente.
Fonte: KAUTZMANN, 2012.

Figura 6 - Menu do paciente.
Fonte: KAUTZMANN, 2012.

Para modelar a aplicação, realizou-se um levantamento de requisitos e o detalhamento das informações essenciais que precisam ser armazenadas foi obtido por intermédio de um diagrama de classes. Todavia, o estudo carece de uma análise prévia sobre o fluxo de atividades do aplicativo, minuciando como a interação com o usuário irá ocorrer. Devido a isso, apesar de simples, o aplicativo não tem um uso intuitivo. Os menus, implementados na forma de botões em uma *Activity* própria, só podem ser acessados a partir de pontos específicos do aplicativo. Ademais, algumas telas de cadastro são excessivamente carregadas de componentes visuais, conforme pode ser verificado na figura 5, dificultando a visualização e o manuseio em dispositivos de tela pequena.

Conforme é descrito no capítulo 1, este projeto tem como propósito desenvolver um sistema hospitalar que faz uso de dispositivos móveis para o monitoramento remoto de pacientes e acesso às informações médicas. Nesse contexto, a exploração de trabalhos pertinentes ao tema se faz necessária, visando investigar as vantagens e desvantagens de cada abordagem. Com isso, pode-se unir as principais características desses sistemas e introduzi-las no projeto proposto. Os pontos negativos, por sua vez, devem ser atentamente evitados. No

decorrer deste trabalho, serão detalhadas as funcionalidades inseridas no sistema, as quais o comportamento foi estabelecido levando em consideração às informações levantadas nesta seção, principalmente no que diz respeito ao monitoramento de pacientes e envio de alertas médicos. Ainda, através do estudo da modelagem de sistemas hospitalares, foi possível tomar ciência de alguns requerimentos para um projeto dessa natureza.

3. CONCEITOS RELACIONADOS

Este capítulo busca introduzir o leitor aos conceitos gerais relacionados ao desenvolvimento deste trabalho. Para tanto, é realizada uma fundamentação teórica acerca de tais assuntos por meio de pesquisas bibliográficas.

Na seção sobre sinais vitais serão abordados os aspectos gerais em relação ao monitoramento de pacientes. As faixas de valores normais para cada um dos sinais serão discutidas, assim como as terminologias empregadas e o que esses sinais podem indicar em relação a saúde do paciente. No segmento de banco de dados, serão levantados os principais fundamentos que envolvem um sistema dessa natureza. Em especial, o modelo relacional – utilizado nesse projeto - é debatido, apresentando informações a respeito de seu funcionamento e vantagens sobre os modelos concorrentes.

Por fim, é realizada uma introdução ao protocolo HTTP. Nessa parte, serão analisadas as principais características do protocolo e a sua importância na troca de mensagens através da internet.

3.1. Sinais Vitais

A obtenção de sinais clínicos de vida – considerados desde a antiguidade como um dos mais importantes dados do exame físico – é um dos procedimentos mais comuns na enfermagem. Esses sinais são utilizados como indicadores do estado de saúde do paciente, apontando a eficiência das funções: circulatória, respiratória, neural e endócrina do corpo (POTTER; PERRY, 2009). Dessa forma, a monitoração desses parâmetros pode orientar a identificação de um diagnóstico inicial e acompanhar a evolução do quadro clínico de um paciente.

Diversos sinais podem ser utilizados para verificar o funcionamento do corpo, sendo os mais utilizados: frequência cardíaca, pressão arterial, temperatura e frequência respiratória. A saturação de oxigênio no sangue e a dor, uma experiência subjetiva, também são indicadores frequentemente mensurados junto com os demais. As faixas de valores normais para esses indicadores geralmente são bem definidas e qualquer anormalidade pode indicar

problemas de saúde. Ressalta-se, porém, que no caso de o paciente já ter sido diagnosticado, esses dados devem ser interpretados no contexto de cada doença – em geral, as doenças perturbam esses valores, portanto uma anormalidade já é esperada. Além disso, alguns medicamentos também podem interferir nos resultados.

Muitas outras variáveis provocam alterações nos sinais vitais e devem ser levadas em conta no momento de sua aquisição a fim de obter valores que representem a real condição do paciente. Se a alteração não tem relação com esses fatores, uma intervenção médica se faz necessária. Quando se conhece as variáveis que influenciam nos parâmetros e se identifica a relação entre uma alteração nos sinais e outros achados de uma avaliação física, é possível determinar precisamente o problema de saúde do paciente. Desse modo, os sinais vitais e outras medidas fisiológicas são a base para a resolução de um problema clínico (VANDER; SHERMAN; LUCIANO, 2001). A seguir são descritos os principais sinais monitorados e o que eles apontam em relação a saúde do paciente.

3.1.1. Temperatura

Para o corpo funcionar de maneira apropriada, a temperatura corporal deve se manter praticamente constante. Essa função é exercida pelo hipotálamo, que é localizado no cérebro e funciona como um termostato. Para garantir uma temperatura adequada, é necessário balancear o calor produzido e o calor perdido pelo corpo. Basicamente, o corpo humano produz calor através de contrações musculares, processos fisiológicos e pelo metabolismo celular. A perda de calor, por sua vez, ocorre por condução, radiação e convecção – mecanismos de transferência de calor – e também pela excreção (BONEWIT-WEST; HUNT; APPLEGATE, 2013).

Em adultos saudáveis, a temperatura interna varia entre 36,6ºC e 37,5ºC. Caso a temperatura seja inferior a 36,1ºC, caracteriza-se uma hipotermia. Ou seja, o corpo está perdendo mais calor do que produzindo. Para aquecer o corpo, o hipotálamo envia um sinal que provoca tremores e também a vasoconstrição, o estreitamento dos vasos sanguíneos. Por outro lado, em temperaturas superiores a 37,5ºC tem-se uma pirexia, conhecida como febre. A febre é um importante mecanismo de defesa, visto que elevações moderadas da temperatura intensificam o sistema imunológico do organismo. Nessa situação, a produção de leucócitos é estimulada e a redução na concentração de ferro no plasma suprime o crescimento de

bactérias. Para reduzir a temperatura, o hipotálamo envia um sinal que desencadeia a sudorese. Além disso, ocorre a vasodilatação. Ao atingir 41ºC, qualifica-se uma hiperpirexia, que geralmente é fatal quando chega aos 43ºC (VAUGHANS, 2012).

Alguns dos maiores indícios que se pode inferir de uma mensuração anormal de temperatura do paciente é a potencial presença de uma infecção. Ainda, fornece um retorno com o qual é possível avaliar a resposta metabólica ao exercício.

3.1.2. Frequência cardíaca

A frequência cardíaca é a medida do número de vezes que o coração bate por minuto. Em adultos saudáveis, a frequência média é de 70 batidas por minuto, sendo que uma faixa entre 60 e 100 batidas é considerada normal, com algumas exceções. Em atletas que fazem bastante condicionamento cardiovascular, por exemplo, é normal um valor próximo a 40 batidas por minuto (STEWART, 2003). Uma frequência cardíaca superior a 100 batidas por minuto indica uma taquicardia. Quando a frequência de pulsação é lenta, tem-se uma bradicardia.

Ao avaliar a frequência cardíaca, deve-se considerar uma variedade de fatores que podem influenciar no resultado. Mudanças posturais modificam a frequência cardíaca: sentar e ficar em pé normalmente eleva seu valor, enquanto deitar o reduz. Condições pulmonares também têm impacto, uma vez que doenças como asma e DPOC (Doença Pulmonar Obstrutiva Crônica) levam a uma oxigenação pobre (POTTER; PERRY, 2009).

A mensuração desse parâmetro permite à equipe médica avaliar o funcionamento circulatório do corpo. Através dele, é analisado se o ritmo e a frequência estão adequados, podendo identificar arritmias (intervalos irregulares entre as batidas) e doenças cardiovasculares. Também é verificado o volume do pulso, que se refere a força da batida do coração. Se o volume de sangue diminui, o pulso fica fraco. Geralmente esse fato ocorre associado com uma alta frequência cardíaca e pode indicar hemorragia.

3.1.3. Frequência respiratória

A respiração possui duas funções principais no organismo: respiração celular e trocas de oxigênio e dióxido de carbono entre o organismo e o ambiente externo (VANDER, 2001). Essas funções envolvem os mecanismos de ventilação (movimentação dos gases para dentro e fora do corpo), difusão (movimentação do oxigênio e do dióxido de carbono entre alvéolos e hemácias) e perfusão (distribuição das hemácias para os capilares sanguíneos e a partir deles). Através da determinação da frequência, profundidade e ritmo da respiração, examina-se o processo de ventilação (POTTER; PERRY, 2009).

A troca de gases ocorre quando o ar chega aos alvéolos pulmonares, parte funcional do pulmão. O oxigênio do ar inalado entra no sangue e o dióxido de carbono é exalado para a atmosfera, transformando o sangue venoso em sangue arterial. A frequência respiratória é determinada através da expansão torácica, contando o número de inspirações por minuto. O padrão em adultos fica entre 12 e 20 vezes por minuto, com frequência regular.

A respiração está ligada à função de vários órgãos. O coração e o cérebro exigem uma certa quantidade de oxigênio para funcionar e uma apneia por mais de 7 minutos geralmente ocasiona um dano cerebral irreversível (STEWART, 2003).

3.1.4. Saturação de oxigênio

Assim como a frequência respiratória permite avaliar o mecanismo de ventilação, a saturação de oxigênio do sangue permite aferir a difusão e perfusão. Após a inspiração, a maior parte do oxigênio se fixa às moléculas de hemoglobina presentes nas hemácias, as quais são transportadas para os capilares dos tecidos do corpo. Quando o sangue passa por esses capilares, o oxigênio se separa da hemoglobina e se difunde para as células.

Cada molécula de hemoglobina pode carregar até quatro moléculas de oxigênio. Se todos os sítios de ligação na molécula de hemoglobina estão carregando oxigênio, a saturação é de 100%. Um indivíduo saudável possui uma saturação entre 95% e 100% (WILSON, 2013).

Durante um procedimento cirúrgico, é necessária a contínua monitoração desse parâmetro. Devido a anestesia, é necessário que a saturação do paciente esteja sempre dentro

da faixa normal. Se a saturação for igual a 94%, o paciente está hipóxico e deve ser rapidamente tratado para evitar problemas mais graves.

3.1.5. Pressão Arterial

De acordo com Bonewit-West, Hunt e Applegate (2013, p.331): “A pressão arterial é a medida da pressão ou força exercida pelo sangue nas paredes das artérias na qual está contido”. A contração dos ventrículos força o sangue sob alta pressão para dentro da aorta. Essa fase representa a pressão sistólica – pico máximo de pressão. Quando os ventrículos relaxam, o sangue que permanece nas artérias exerce uma pressão mínima, conhecida como pressão diastólica. Quando a pressão sistólica se aproxima da diastólica, a pressão é dita convergente. Caso contrário, chama-se de divergente. Uma pressão convergente pode indicar febre, anemia e hipertireoidismo, enquanto a pressão divergente está associada ao hipotireoidismo e insuficiência cardíaca, por exemplo.

A pressão arterial depende da força de contração do coração, da quantidade de sangue circulante e da resistência dos vasos, sendo um ótimo indicador do funcionamento do sistema circulatório (saúde cardiovascular). A unidade padrão para medir a pressão arterial é dada em milímetros de mercúrio (mmHg). De acordo com o Instituto Nacional Norte-Americano do Coração, Pulmão e Sangue (NHLBI), uma pessoa saudável possui uma pressão arterial de 120/80mmHg. Um valor maior que 160/90 mmHg é considerada hipertensão e uma pressão menor que 90/60 mmHg, hipotensão. Uma anormalidade na pressão arterial, se não tratada, pode afetar a visão e provocar sérios danos aos órgãos vitais (POTTER; PERRY, 2009).

3.2. Banco de dados

O armazenamento de registros consiste em uma atividade que os seres humanos vêm desenvolvendo há muito tempo, ainda antes do surgimento da escrita. Atualmente, com a informatização dos sistemas, os bancos de dados são essenciais para qualquer negócio. Sua utilização envolve uma alta gama de aplicações, desde sistemas simples para controlar o estoque de material a sistemas avançados, como sistemas bancários e de segurança pública.

Esta seção visa introduzir, de forma sucinta, os principais fundamentos de um sistema de banco de dados, focando-se nos conceitos relevantes a este projeto.

Um banco de dados é uma coleção de dados relacionados e armazenados em algum dispositivo. O sistema é projetado atendendo a uma proposta específica, segundo a necessidade de uma aplicação. A principal vantagem na utilização desses sistemas é manter a informação centralizada, de forma organizada e facilmente acessível. Além disso, os dados podem ser acessados simultaneamente por vários usuários (MEIRA, 2013). De acordo com Geremia (2010), um banco de dados tem quatro objetivos principais:

1. Acesso rápido aos dados: Em um sistema informatizado, o acesso aos dados ocorre de modo muito mais eficiente do que em métodos convencionais. Um software para consulta ao banco de dados consegue obter a informação em um intervalo muito pequeno de tempo. Ademais, a probabilidade de erro diminui consideravelmente.
2. Redução na redundância e inconsistência de dados: Em sistemas que não utilizam um banco de dados, cada aplicação precisa armazenar seus próprios arquivos. Isso costuma provocar uma redundância considerável nos dados armazenados. O uso de banco de dados tende a eliminar também a inconsistência. Ao atualizar um campo, tudo que estiver relacionado a ele automaticamente é atualizado e disponível para todos os usuários.
3. Compartilhamento de dados: os dados podem ser acessados simultaneamente por vários usuários, para posteriormente serem utilizados para diferentes finalidades.
4. Aplicação de restrições de segurança: Um sistema de banco de dados pode restringir quais dados um usuário tem acesso, de acordo com a sua necessidade. Assim, torna-se a manipulação dos dados mais segura.

Ainda, podem ser citadas outras características como: disponibilização de uma forma de acesso gráfica, mecanismos que possibilitem a compreensão do relacionamento existente entre as tabelas e de sua eventual manutenção, controle de integridade e backup.

3.2.1. Software Gerenciador de Banco de Dados

Um banco de dados é criado e mantido por um conjunto de softwares projetados especialmente para essa tarefa. Esse conjunto de softwares é denominado Software Gerenciador de Banco de Dados (SGBD).

Um banco de dados requer um grande espaço de armazenamento, podendo, em ambientes corporativos, chegar à casa de terabytes. Esses dados são armazenados em discos secundários e movidos para a memória principal de acordo com a necessidade. Comparada com a velocidade de processamento, a movimentação de dados entre a memória secundária e a memória principal é muito lenta. Assim, uma das funções do SGBD é minimizar o movimento de dados entre os discos. Outrossim, grande parte das vantagens da utilização de um banco de dados citadas anteriormente, como acesso concorrente aos dados, integridade, segurança e consistência são implementados pelo SGBD. O suporte a tais funções, em conjunto com uma interface de alto nível aos dados, reduz drasticamente o tempo de desenvolvimento de uma aplicação, possibilitando que o desenvolvedor se foque em questões mais específicas acerca do projeto (RAMAKRISHNAN; GEHRKE, 2003).

Em suma, o SGBD garante um acesso robusto e eficiente ao banco de dados, enquanto proporciona uma interface simples. Como alguns dos gerenciadores mais populares, cita-se o SQL Server, PostgreSQL, Microsoft Access e Oracle, sendo que, neste projeto, utilizou-se o MySQL como banco de dados externo e o SQLite, o qual é interno ao Android.

3.2.2. O modelo relacional

O modelo relacional consiste em uma teoria matemática criada por Edgar Frank Codd em 1970 para descrever como as bases de dados devem funcionar. Em seu trabalho, Codd mostrou que uma visão relacional dos dados permite a sua descrição em uma maneira natural, sem a necessidade de estruturas adicionais para a sua representação (MACÁRIO; BALDO, 2005).

O modelo se baseia em conceitos da matemática – teoria dos conjuntos e lógica de predicado. A estrutura fundamental do modelo relacional é a relação (tabela). Aplicando-se a terminologia do modelo relacional, diz-se que as linhas se denominam tuplas e as colunas, atributos.

Em complementação, o modelo apresenta as bases para tratamento de redundância e consistência. Regras de integridade podem ser implementadas de forma declarativa ou procedural. A integridade declarativa é implementada através de parâmetros opcionais da linguagem de definição de dados (DDL), sendo os tipos mais comuns: integridade de domínio, integridade da entidade e integridade referencial. A integridade de domínio visa

restringir o universo de valores válidos para uma coluna, evitando inserções errôneas de dados. A integridade de entidade, por sua vez, consiste na utilização de chaves primárias para certificar que uma tabela contenha um identificador único e não nulo. Esse mecanismo permite distinguir uma linha das demais dentro de uma tabela. Para a implementação de um relacionamento dentro de um banco de dados relacional, faz-se uso de chaves estrangeiras, utilizando o conceito de integridade referencial.

Chaves estrangeiras consistem em colunas cujos valores aparecem, necessariamente, na chave primária de outra tabela (ou na própria tabela, em um auto relacionamento). Assim, mantem-se o sincronismo entre as tabelas e se evita a ocorrência de registros órfãos no banco de dados – registros "filhos" sem a correspondente linha na tabela "pai". Essas ações propiciam ainda a propagação de atualizações e exclusões no banco. Ou seja, a exclusão de um registro pode provocar a exclusão automática de seus respectivos filhos (exclusão em cascata), ou a alteração no valor de uma chave primária é refletida automaticamente para os registros que a referenciam (atualização em cascata).

Para suprir as necessidades não atendidas pelas técnicas anteriores, há ainda a possibilidade de utilizar a integridade procedural. Esse mecanismo se apresenta sobre a forma de um programa, na linguagem nativa do SGBD. A integridade procedural pode ser criada através de *triggers, stored procedures* e *stored functions*. Um *trigger* (gatilho) é utilizado para disparar ações automaticamente ao detectar um dos comandos de acesso a uma tabela. *Stored procedures* e *functions* são métodos armazenados no próprio servidor do banco de dados que permitem acessar dados e realizar operações sobre eles. A diferença básica entre os dois modos é que, em *functions*, há sempre um retorno (GEREMIA, 2010).

A linguagem padrão dos bancos de dados relacionais é a Structured Query Language (SQL). A linguagem, que possui características baseadas na álgebra relacional, fornece funções de criação, consulta, recuperação e manutenção da estrutura de dados. Trata-se de uma linguagem declarativa, o que a torna simples e de fácil compreensão. A padronização da linguagem assegura a portabilidade, tornando o banco de dados facilmente adaptável a outros gerenciadores (COLARES, 2007).

A maior vantagem do modelo relacional em relação a seus concorrentes é a representação dos dados de forma simples e a facilidade com que consultas complexas podem ser expressas. Desta forma, revelou-se o modelo mais flexível e adequado ao solucionar os vários problemas que se colocam no nível da concepção e implementação de uma base de dados (MACÁRIO; BALDO, 2005).

3.3. HTTP

O HTTP (*Hypertext Transfer Protocol*) é um protocolo de comunicação entre sistemas de informação em nível de aplicação que permite a transferência de dados entre redes de computadores. Desde 1990, HTTP é a base da comunicação através da internet, fornecendo um padrão para a troca de mensagens entre sistemas (YANNAKOPOULOS, 2003).

O HTTP não entra em detalhes do protocolo de transmissão empregado, presumindo apenas uma transmissão confiável. Assim, qualquer protocolo que ofereça tais garantias pode ser utilizado, sendo que o mais comum é o TCP/IP. Essa característica torna o HTTP fácil de implementar em linguagens de alto nível, sem que seja necessário um conhecimento profundo do protocolo de transmissão (FIELDING, GETTYS, *et al.*, 1999).

O protocolo HTTP é baseado no paradigma de requisição/resposta no modelo cliente-servidor, adotando um URI (*Uniform Resource Identifier*) para identificar um recurso e estabelecer uma conexão. Basicamente, o cliente estabelece uma conexão com o servidor e envia uma requisição para o mesmo. O servidor, então, retorna uma resposta que deverá conter os dados referentes a requisição por ele recebida. O HTTP define quais requisições podem ser enviadas, quais respostas podem ser recebidas e como as mensagens são formatadas (ADAMS, 2001).

Uma mensagem HTTP consiste em uma linha inicial, cabeçalho e corpo da mensagem. A linha inicial, caso se trate de uma requisição, possui informações a respeito do método, URI e versão do HTTP. Caso contrário, contém a versão do HTTP e o código de status. O cabeçalho possui informações gerais a respeito da mensagem, como: tamanho, data, tipo de conteúdo, codificação utilizada, entre outros. O corpo da mensagem é opcional. Se disponível, deverá conter os dados associados com a requisição ou resposta que está sendo enviada.

Os métodos GET e POST são os dois principais métodos de requisição HTTP. No método GET, os dados da requisição são enviados na própria URL (*Uniform Resource Locator*), em pares de nome/valor, e ficam visíveis ao usuário. Por isso, geralmente é utilizado para solicitar dados do servidor, onde são enviados alguns termos de consulta. O método POST, em contraste, envia os dados no corpo da mensagem. Com isso, as informações não estarão explicitamente visíveis ao usuário, tendo sido projetado para o envio de dados que serão armazenados. Porém, pode também ser utilizado para realizar uma solicitação, se for necessário encapsular a informação a ser enviada. Esses dados serão processados pelo servidor e uma resposta será retornada (FIELDING, GETTYS, *et al.*, 1999).

4. A PLATAFORMA ANDROID

Android é uma plataforma, de código aberto, projetada para dispositivos móveis. O sistema foi desenvolvido pela OHA (*Open Handset Alliance*), um grupo formado por empresas relacionadas com a tecnologia móvel. A aliança foi formada com o objetivo de estabelecer padrões para a indústria de tecnologia móvel, sendo a criação do Android o primeiro passo para alcançar esse objetivo (ACKER; WEBER; CECHIN, 2010).

O Android é um sistema operacional desenvolvido sobre um kernel Linux, com algumas mudanças arquiteturais. O Linux consiste em um sistema com ótima segurança, tendo sido testado e aprimorado durante anos. As aplicações do Android rodam como processos separados do Linux, com permissões definidas pelo sistema. Além disso, o Linux é relativamente fácil de compilar em várias arquiteturas de hardware, sem fazer suposições a respeito das características físicas do aparelho (GARGENTA, 2011). De acordo com os dados da empresa OpenSignal em agosto de 2014, 18.796 aparelhos distintos utilizam a plataforma Android, demonstrando o potencial de portabilidade que o mesmo possui (OPEN SIGNAL, 2014). Trata-se do sistema operacional mais utilizado em dispositivos móveis atualmente. No primeiro quadrimestre de 2015, smartphones que utilizam a plataforma representaram 78% do total de aparelhos vendidos no mundo (IDC, 2015).

Os aplicativos para Android são distribuídos através da Google Play, loja de aplicativos da Google. A figura 7 representa a quantidade de aplicativos disponíveis nas principais lojas do gênero.

De acordo com os dados, em julho de 2015, data da realização da pesquisa, a Google Play contava com 1,6 milhão de aplicativos disponíveis, tanto pagos como gratuitos. Trata-se da maior loja do gênero, superando as lojas de outros sistemas móveis em variedade de aplicativos (STATISTA, 2015). Nas subseções desse capitulo serão abordados os detalhes da arquitetura da plataforma Android, assim como o processo de desenvolvimento de aplicações para o sistema.

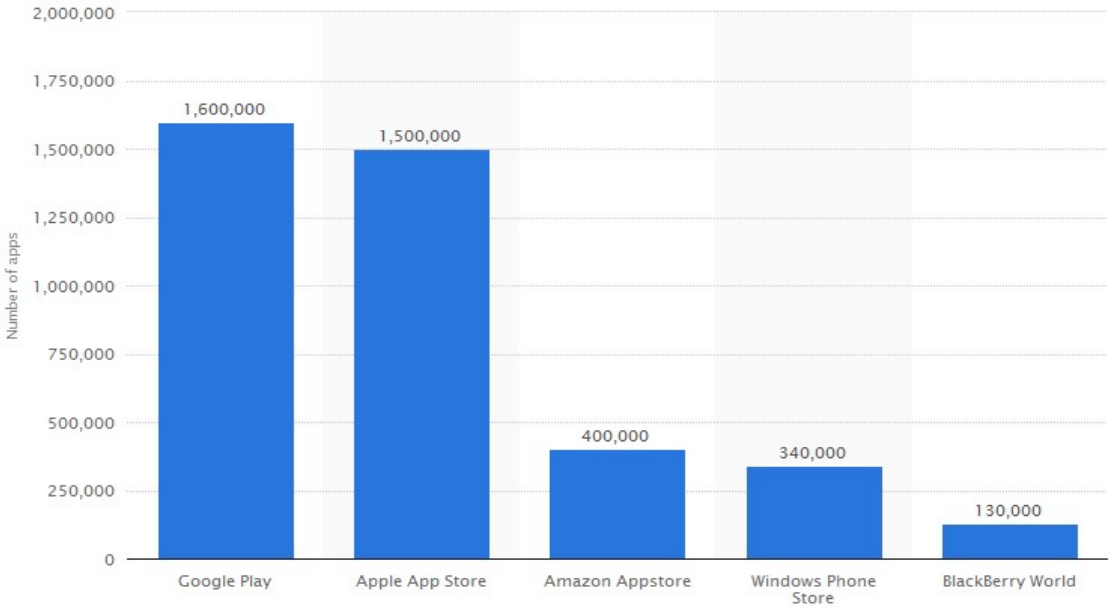


Figura 7 - Comparação entre as principais lojas de aplicativos.

Fonte: STATISTA, 2015.

4.1. Arquitetura do sistema

O sistema operacional Android é formado por diversas camadas, cada uma com uma função específica dentro da plataforma. O objetivo dessa seção é dar ao leitor uma visão geral do sistema. A figura 8 mostra as camadas que formam a estrutura do mesmo.

4.1.1. Linux Kernel

Conforme introduzido anteriormente, o sistema foi projetado sobre um kernel Linux. Basicamente, essa camada contém todos os drivers essenciais, sendo responsável pela interação com o hardware. Atribui-se a ela a capacidade atuar como um nível de abstração entre o hardware e as demais camadas. Além disso, é responsável pelas tarefas básicas do sistema, tais como gerenciamento de memória, gerenciamento de energia e controle de acesso a recursos (GARGENTA, 2011).

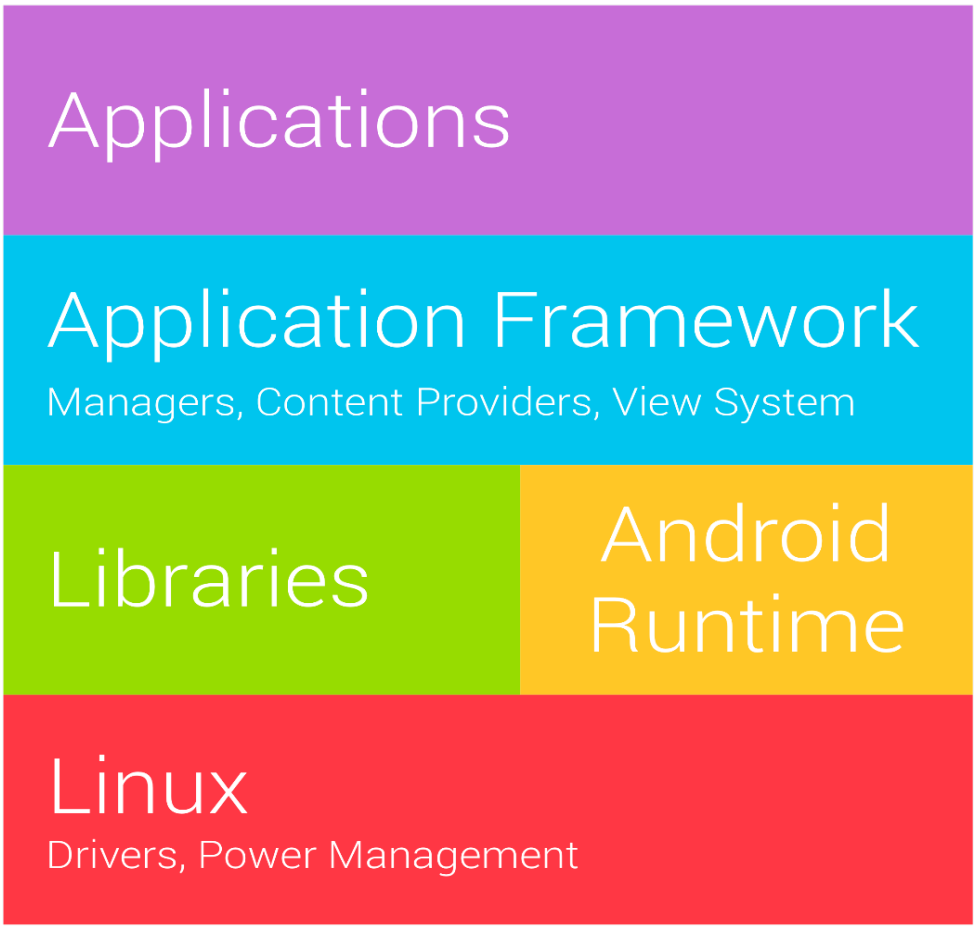


Figura 8 - Arquitetura do Android.

Fonte: LATISLAW, 2013.

4.1.2. Bibliotecas

A segunda camada consiste nas bibliotecas C/C++, que fornecem alguns serviços necessários a camada de aplicação. Dentre esses serviços, encontram-se o suporte a formatos de áudio, vídeo e imagens; bibliotecas para gráficos 2D e 3D e o banco de dados relacional SQLite (BORDIN, 2012).

4.1.3. Android Runtime

No Android, as aplicações são executadas como processos separados com a sua própria instancia da máquina virtual. O kernel Linux concede à aplicação um ID e outorga a

ela as permissões dos arquivos de sua competência, os quais só ficarão visíveis para aquela aplicação durante sua execução. Até o Android KitKat (versão 4.4) era utilizado a máquina virtual Dalvik e, a partir do Lollipop (versão 5.0), passou-se a utilizar a ART. A seguir são abordadas as principais informações a respeito das duas tecnologias.

a. Dalvik

DVM (*Dalvik Virtual Machine*) consiste em uma máquina virtual projetada especificamente para dispositivos móveis. Desenvolvida com foco no consumo de energia, a DVM garante que aplicações possam ser executadas em processadores mais lentos e utilizando menos RAM do que em desktops (OLIVEIRA, 2011).

Diferente de uma JVM (*Java Virtual Machine*), que é baseada em pilha, a DVM é baseada em registradores. A figura 9 representa a comparação entre JVM e DVM. Conforme pode ser visualizado, o código fonte em Java é compilado na JVM e transformado em formato dex (*Dalvik Executable*) por uma ferramenta inclusa. Esse arquivo, então, é executado pela Dalvik. Uma das vantagens de utilizar Java é a independência que ele agrega ao sistema, visto que o *bytecode* pode ser executado em qualquer plataforma (desde que a mesma possua a JVM). O fato da Dalvik ter seu próprio *bytecode* também traz benefícios. Como o *bytecode* é menor e utiliza menos instruções do que o *bytecode* Java, reduz-se o overhead de *fetch*, levando a uma interpretação mais eficiente (ARON; HANÁCEK, 2015).

A máquina virtual lê linha-por-linha do código e a executa. Esse processo, que envolve a busca e execução da instrução, pode ser bem mais demorado do que a execução de programas nativos, os quais são compilados e traduzidos para instruções nativas da arquitetura antes do seu início. Para resolver esse problema, utiliza-se um compilador JIT (*Just-In-Time*) na máquina virtual. Nessa abordagem, o compilador JIT inicia após o programa ser executado. A parte do código (*bytecode*) exigida para essa atividade é traduzida para código de máquina, em tempo de execução. A medida que o aplicativo é executado, o código adicional é compilado e armazenado em cache, de modo que o sistema possa reutilizá-lo. Como o JIT compila apenas a parte do código necessária para a execução, menos memória física é utilizada no dispositivo (CHENG; BUZBEE, 2010).

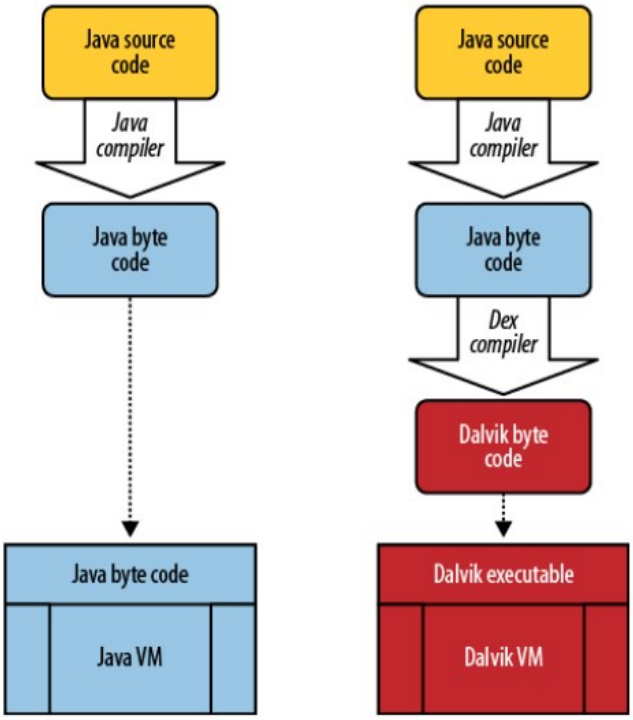


Figura 9 - Comparação entre Java e Dalvik.

Fonte: GARGENTA, 2011.

b. ART

Embora a utilização da Dalvik e a compilação JIT tenha trazido bons ganhos de desempenho e economia de memória física nos aparelhos, alguns pontos ainda precisavam ser aprimorados. A compilação Just-In-Time, por ser realizada em tempo de execução, introduz um atraso na execução das aplicações e afeta o desempenho geral do dispositivo. Com a evolução tecnológica e diminuição dos preços de componentes eletrônicos, principalmente das memórias, possibilitou-se a implantação da ART (*Android Runtime*) para resolver tais problemas. ART e Dalvik são máquinas virtuais compatíveis que executam Dex *bytecode*, de forma que aplicativos desenvolvidos para Dalvik possam funcionar com ART (ANDROID, 2015).

Para aumentar o desempenho, a ART introduziu a compilação AOT (*Ahead-Of-Time*). Essa técnica realiza a tradução do *bytecode* DEX em código de máquina durante a instalação da aplicação no dispositivo. Assim, remove-se o atraso, dado que não é necessária a compilação JIT em tempo de execução. Ademais, como ART executa diretamente em instruções nativas, a necessidade de processamento de CPU durante a execução da aplicação é

menor do que comparado à Dalvik. Consequentemente, o consumo de bateria é menor (ARON; HANÉCK, 2015).

Como contrapontos, cita-se que o processo de instalação de aplicações é mais demorado. Porém, como é executado uma vez só, não afeta o desempenho geral do aparelho. Além disso, diferente da Dalvik que traduz o trecho de código necessário em tempo de execução, o código inteiro é traduzido de uma só vez, durante a instalação, e é armazenado no dispositivo. Como consequência, a aplicação ocupará um maior espaço em disco e a utilização de memória RAM durante a execução da aplicação será maior, visto que todo o código será carregado para a memória.

4.1.4. Framework de Aplicações

A Framework de Aplicações consiste em uma camada, escrita em Java, que fornece todas as funcionalidades necessárias para o desenvolvimento de aplicativos. A camada expõe a capacidade do sistema operacional na forma de serviços que interagem tanto com o sistema operacional quanto com demais aplicações (PANDIYAN; PARANJAPE, 2013).

4.1.5. Aplicações

Por último, tem-se a camada de aplicações, que consiste nas aplicações criadas pelos desenvolvedores. Essas podem vir pré-instaladas com o Android ou baixadas pelo usuário através das lojas de aplicativos.

4.2. Anatomia de uma Aplicação

Uma aplicação é formada por quatro componentes essenciais: *Activity*, *Service*, *Content Provider* e *Broadcast Receiver*. Cada componente exerce um papel específico dentro

da aplicação, definindo o comportamento da mesma. A seguir é detalhada a função de cada um dos blocos principais que constituem uma aplicação.

4.2.1. Activity

As *Activities* são classes que fornecem uma interface com o usuário. Ou seja, representam as telas do aplicativo e refletem uma ação específica que um usuário pode realizar dentro dele. Estas ações podem incluir a inicialização de outras *Activities* através de *Intents*. Assim, tem-se um fluxo de funcionamento da aplicação.

A criação de uma *Activity* é um processo dispendioso, que envolve a criação de um novo processo Linux, alocação de memória para os elementos de interface, carregamento dos objetos presentes no *layout* e configuração da tela. Para evitar realizar todo esse processo novamente, quando o usuário sai de uma *Activity* a mesma é armazenada em segundo plano. De acordo com esse funcionamento, toda *Activity* possui um ciclo de vida que vai da sua criação até a sua destruição. A figura 10 mostra o ciclo de vida de uma *Activity* e os estágios pelo qual ela passa.

Durante a criação de uma *activity*, a mesma passa por um conjunto de métodos de configuração, aos quais cabe ao desenvolvedor complementar. Quando uma *activity* está em execução, significa que ela possui o foco do usuário no momento. Por isso, possui prioridades em termos de alocação de memória e demais recursos. Quando uma *activity* não está em foco (isto é, o usuário não está interagindo com ela), mas ainda está visível na tela, dizemos que a mesma está pausada. Se a *activity* não está visível na tela, mas ainda está na memória, encontra-se em estado parado (GARGENTA, 2011).

Uma tarefa pode conter várias *Activities*, que são organizadas em uma pilha na ordem em que foram criadas. Desta forma, o usuário pode retornar às *Activities* em que estavam anteriormente. Quando uma *activity* é destruída, ela é retirada memória. Assim, se muitas tarefas estiverem em segundo plano, o sistema pode começar a destruí-las para liberar memória. (BORDIN, 2012).

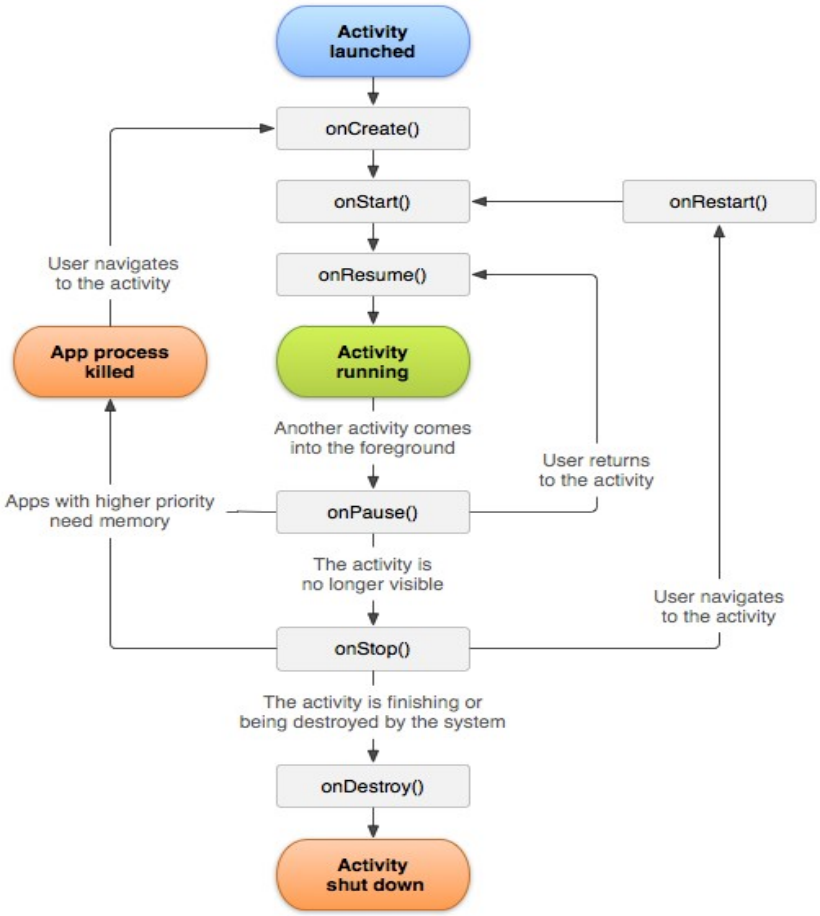


Figura 10 - Ciclo de vida de uma *activity*.

Fonte: ANDROID DEVELOPERS, 2015.

4.2.2. Service

Service é um componente que possibilita à aplicação realizar tarefas longas em segundo plano. Esse componente não fornece uma interface com o usuário. Em vista disso, trata-se de um elemento utilizado para ações que se quer realizar de forma independente do que está visível na tela no momento da sua execução. Por exemplo, um serviço pode reproduzir músicas enquanto o usuário utiliza outros aplicativos. (PANDIYAN; PARANJAPE, 2013).

4.2.3. Content Providers

Um *content provider* possibilita o compartilhamento de dados entre aplicações. Por exemplo, os dados de uma aplicação podem ser armazenados em um arquivo do sistema, no banco de dados SQLite ou na internet. Através de um *content provider*, outras aplicações podem solicitar esses dados e até modifica-los, se houver permissão (ANDROID DEVELOPERS, 2015).

4.2.4. Broadcast Receivers

Broadcast receiver é um sistema de publicação e recebimento de mensagens implementado pelo Android, onde as mensagens sinalizam eventos do sistema. Diversos eventos, tais como recebimento de SMS e chamadas, bateria fraca e inicialização do sistema são transmitidos na forma de *broadcast*. Se algum aplicativo ou arquivo tem manifestado um interesse por um desses eventos, um gatilho irá ativá-los no momento de sua ocorrência. Apesar de um *broadcast receiver* não possuir uma representação visual ou estar ativo na memória, quando são engatilhados eles executam um código, iniciando uma atividade, serviço ou algum outro procedimento no sistema (GARGENTA, 2011).

4.3. Desenvolvendo aplicações para Android

Um dos principais objetivos do Android é criar uma plataforma de software aberta disponível tanto para fabricantes quanto desenvolvedores. Desta maneira, possibilita-se que esses possam transformar suas ideias em realidade introduzindo-as em um produto de sucesso no mundo-real, aperfeiçoando a experiência móvel para os usuários (ANDROID, 2015). O sistema é licenciado de modo que qualquer um possa estendê-lo e usá-lo para uma variedade de propósitos (GARGENTA, 2011).

Os aplicativos Android são desenvolvidos principalmente em linguagem Java, embora exista a possibilidade de utilizar C++. As ferramentas SDK (*Software Development Kit*) – conjunto de ferramentas necessárias para o desenvolvimento para a plataforma - compilam o código, juntamente com os dados e demais arquivos. Como resultado, é gerado um APK (*Android Package*), arquivo que possui todos os conteúdos de um aplicativo Android e é utilizado pelo sistema na instalação do mesmo. O site da plataforma disponibiliza para download o Android Studio, uma IDE (*Integrated Development Environment*) com todas as ferramentas necessárias para o desenvolvimento de software para o sistema. A IDE inclui ainda um emulador de dispositivos móveis, permitindo que o desenvolvedor teste as aplicações sem a necessidade de utilizar um dispositivo físico (ANDROID DEVELOPERS, 2015).

Os desenvolvedores podem disponibilizar suas aplicações para os usuários, seja distribuindo diretamente, disponibilizando o download pelo site da empresa ou publicando em uma loja de aplicativos, como a Google Play. A figura 11 representa a sequência de processos para o desenvolvimento de uma aplicação.

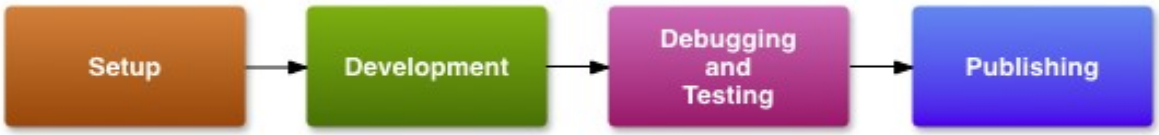


Figura 11 - Sequência de etapas para o desenvolvimento de uma aplicação.
Fonte: ANDROID DEVELOPERS, 2015.

Inicialmente, configura-se o ambiente de desenvolvimento que será utilizado. Após, o desenvolvimento da aplicação é, de fato, realizado, num processo que envolve modelagem e implementação da mesma. Uma vez implementada todas as funcionalidades desejadas, parte-se para a etapa de testes, a qual visa garantir a estabilidade e desempenho da aplicação. Realizado esse passo, a aplicação está apta para a disponibilização ao público em geral.

Para publicar uma aplicação, é necessário obter uma chave privada, que corresponde a assinatura da aplicação. Algumas bibliotecas utilizadas no desenvolvimento também exigem a obtenção de uma chave de liberação. Além disso, é interessante a criação de um EULA (*End User License Agreement*) para proteger a propriedade intelectual e a organização ou pessoa responsável pelo desenvolvimento. Caso o desenvolvedor opte por publicar o aplicativo na Google Play, é necessário criar uma conta de desenvolvedor, a qual exige 25 dólares como

taxa de registro. A loja possui ferramentas para análise de vendas e distribuição do software, permitindo a identificação do público alvo e demais recursos (ANDROID DEVELOPERS, 2015).

Com o intuito de garantir uma excelente experiência para os usuários, independente do dispositivo que o mesmo utiliza, devem ser tomados alguns cuidados por parte dos desenvolvedores. A seguir são detalhadas as principais preocupações que devem ser devidamente analisadas para assegurar a compatibilidade da aplicação.

4.3.1. Compatibilidade de versões

Em primeiro lugar, deve-se estar ciente de utilizar a mínima API (*Android Programming Interface*) de desenvolvimento necessária para executar o aplicativo. Com isso, garante-se que um maior número de versões suportará a aplicação e, desta forma, um maior número de usuários será alcançado. A figura 12 apresenta as principais versões do Android, a API que a mesma utiliza, e a quantidade de usuários que executam a versão.

Version	Codename	API	Distribution
2.2	Froyo	8	0.3%
2.3.3 - 2.3.7	Gingerbread	10	4.6%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	4.1%
4.1.x	Jelly Bean	16	13.0%
4.2.x		17	15.9%
4.3		18	4.7%
4.4	KitKat	19	39.3%
5.0	Lollipop	21	15.5%
5.1		22	2.6%

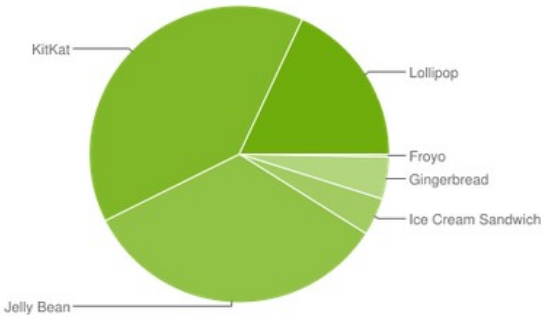


Figura 12 - Versões do Android e sua distribuição.

Fonte: ANDROID DEVELOPERS, 2015.

As informações foram disponibilizadas pela própria Google, e consistem nos dados coletados pela Google Play em um período de sete dias, terminando em 3 de agosto de 2015.

De acordo com a figura, o Android KitKat (versão 4.4) lançado ao final de 2013, que utiliza a API 19, é o que possui a maior parcela de usuários. Em segundo lugar vem o Jelly Bean e, após, o Android Lollipop, versão mais recente do sistema, que se tornou disponível ao final de 2014 (ANDROID DEVELOPERS, 2015).

Ao lançamento de uma nova versão, novas APIs são adicionadas à plataforma. Cada versão sucessiva fornece compatibilidade para aplicativos que foram desenvolvidos usando as APIs anteriores. Assim, o aplicativo sempre será compatível com as versões futuras do Android.

5. MODELAGEM DO SISTEMA

O crescimento da complexidade das aplicações e a existência de limitações no ser humano em lidar com a mesma é uma das principais razões para adotar a modelagem como o primeiro passo no desenvolvimento de uma aplicação. Com o objetivo de reduzir a complexidade, divide-se o sistema em modelos. De acordo com Bezerra (2006), um modelo pode ser visto como uma representação idealizada de um sistema a ser construído. Como cada modelo exprime uma perspectiva do sistema, estudando-se separadamente cada modelo facilita-se a compreensão do projeto como um todo. Segundo Booch, Rumbaugh e Jacobson (2005) há quatro motivos principais para se criar modelos:

1. Modelos ajudam a visualizar o sistema do jeito que ele é ou como queremos que ele seja.
2. Os modelos permitem especificar a estrutura ou comportamento do sistema.
3. Os modelos proporcionam um guia para a construção do sistema.
4. Documentam as decisões tomadas na fase de planejamento.

O processo de modelagem é iniciado anteriormente à implementação do sistema para que sirva como base para esse. O planejamento usualmente adota uma postura que parte dos princípios básicos que se deseja colocar no sistema e, a partir do entendimento desses, insere-se informações mais detalhadas. Na elaboração desses modelos, detalhes irrelevantes podem ser ignorados por hora, dando possibilidade para se focar nos pontos chave do sistema. A criação de uma documentação referente ao projeto permite também a difusão de informações relativas a ele entre os indivíduos envolvidos na sua construção. Essa etapa possibilita a detecção de erros e otimização no tempo de desenvolvimento, uma vez que se tem um entendimento melhor sobre o que está sendo desenvolvido (BEZERRA, 2006).

Muitos sistemas têm seu prazo retardado ou até mesmo morrem durante seu percurso devido a um planejamento ineficaz. Ao não adotar uma política de desenvolvimento que preze pela qualidade do software, visando a urgência na finalização, resulta em um tempo de produção mais rápido. Em contrapartida, o sistema acaba não atingindo as necessidades dos clientes e gera mais gastos no futuro para a manutenção e correção de problemas causados pela não estruturação do software.

Com o intuito de padronizar o processo de modelagem de um sistema, foi criada a UML (*Unified Modeling Language*). A UML é uma linguagem de modelagem visual de

propósito geral usada para especificar, visualizar, construir e documentar um sistema. A linguagem utiliza um conjunto de técnicas de notação gráfica para criar os modelos de sistemas intensivos, utilizando conceitos de comum acordo entre a comunidade da computação. Desta forma, consiste em uma linguagem única, comum e amplamente utilizável (RUMBAUGH; JACOBSON; BOOCH, 1998).

A UML foi desenvolvida para ser simples, compreensível, fácil de se comunicar com outras aplicações e com capacidade de constantes atualizações. A UML é usada no desenvolvimento de diversos tipos de sistemas. Naturalmente, o uso mais comum é em sistemas de software, mas também pode ser usada para representar sistemas mecânicos, que não envolvem nenhum software. A linguagem é aplicada em diferentes fases do desenvolvimento, desde o levantamento de requisitos até a fase de testes. Durante todo o processo, os modelos são atualizados até que, por fim, obtém-se uma documentação o mais consistente possível com o proposto (STADZISZ, 2002).

Uma das características interessantes da UML é a existência de mecanismos de extensão. Por meio desses, permite-se estender a linguagem, resultando na criação de novos tipos de diagramas que melhor representem o sistema. Portanto, apesar de buscar padronizar a representação, ainda se trata de uma linguagem flexível, garantindo que o objetivo principal seja a compreensão do projeto (GUDWIN, 2010).

A modelagem de um sistema não tem uma sequência bem definida de modelos para seguir. Ao mesmo tempo que a efetuação de um modelo servirá de base para o desenvolvimento do próximo, ele também vai dar ao desenvolvedor as informações necessárias para que seja possível complementar um outro modelo, realizado anteriormente. Assim, o processo consiste de recorrentes revisões e complementações dos modelos construídos. Essas atualizações podem vir a acontecer também durante a etapa de implementação, enriquecendo cada vez mais a documentação sobre o sistema.

Neste capítulo será apresentada a etapa de modelagem do sistema proposto, exibindo detalhes do processo de elaboração desde as fases iniciais, onde foi realizado o levantamento de requisitos, até a fase de implementação. Durante essa etapa, construiu-se diversos modelos, os quais são introduzidos por meio pesquisas bibliográficas. Conforme citado, os modelos foram elaborados de forma complementar, não seguindo uma ordem. Portanto, a sequência de modelos apresentada nesse capítulo não representa necessariamente a ordem em que foram desenvolvidos, mas está organizada visando a melhor compreensão da descrição do que foi realizado. A metodologia utilizada para a criação dos modelos é descrita em detalhes, assim

como sua relevância dentro do planejamento do sistema e as decisões de projeto tomadas durante o processo.

5.1. Requisitos do sistema

O levantamento de requisitos é uma das partes mais importantes do desenvolvimento de um software. Essa etapa consiste em compreender o que o cliente (ou usuário) deseja e as regras do negócio. De nada adianta um sistema bem elaborado a nível de programação, mas que não seja capaz de fornecer as funções que o cliente necessita e não se enquadre no ambiente em que será implementado (MELLO; MEYER, 2010).

Requisitos são uma coleção de sentenças que devem descrever de modo claro e conciso os aspectos significativos do sistema proposto. Ou seja, são condições que devem ser alcançadas pelo sistema para satisfazer uma especificação.

> Sem uma base de requisitos relativamente estável, um projeto em desenvolvimento só pode fracassar. É como sair para uma viagem marítima sem ideia do destino e sem mapa de navegação. Requisitos fornecem tanto o mapa de navegação quanto os meios de caminhar na direção correta (HULL; JACKSON; DICK, 2005, p. 2).

O principal objetivo do levantamento de requisitos é que os usuários e os desenvolvedores tenham a mesma visão do problema a ser resolvido. O desenvolvimento de uma especificação completa e consistente visa servir como base para um acordo entra as partes envolvidas, descrevendo o que o produto irá fazer, mas não como será feito. Por isso, trata-se de uma etapa onde se deve analisar os dois lados do problema, exigindo constante interação entre as partes. Ademais, expõe não apenas conhecimentos técnicos, mas também gerenciais, organizacionais, econômicos e sociais.

A dificuldade de comunicação ou de compreensão do problema é um contratempo comum nessa etapa. Portanto, deve-se levar o tempo necessário e fazer um esforço para que se chegue a um senso comum sobre o sistema que se está projetando (BEZZERRA, 2006). Se a interação entre usuário e desenvolvedor não for possível, como no caso do desenvolvimento de um software que será disponibilizado ao público em geral, cabe ao desenvolvedor buscar informações acerca das necessidades dos usuários alvo. A figura 13 representa as diferentes

visões do sistema por parte dos usuários e desenvolvedores e como isso pode afetar o resultado final do projeto.

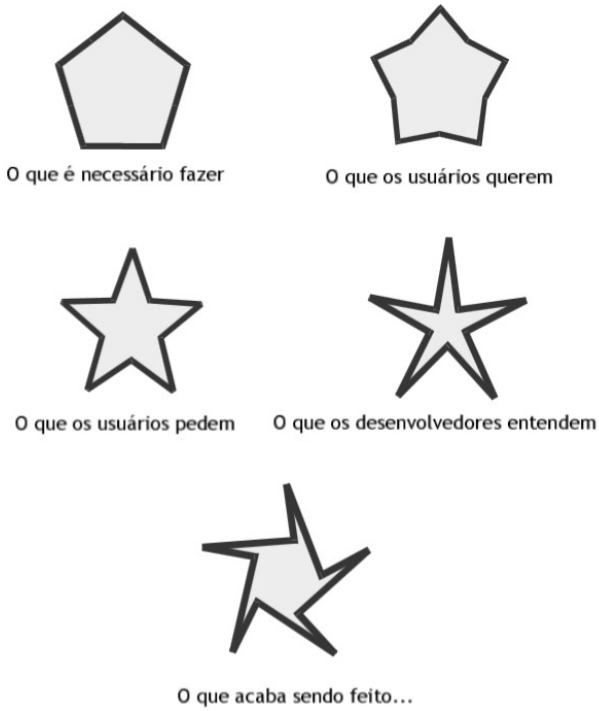


Figura 13 - Diferentes pontos de vista sobre os requisitos.

Fonte: PAULA FILHO, 2000.

Os requisitos de um sistema podem ser categorizados em funcionais e não funcionais. De acordo com Sommerville (2006), requisitos funcionais são serviços que o sistema deve fornecer, como irá reagir a partir de determinadas entradas e como deve se comportar em situações particulares. Requisitos não funcionais, como o nome sugere, são requisitos que não estão ligados a funções específicas. Eles podem estar relacionados às propriedades do sistema, como confiabilidade e desempenho, e também às restrições impostas a ele, indo além das necessidades dos usuários. Em outras palavras, são restrições colocadas sobre como o sistema deve realizar seus requisitos funcionais.

Por meio de análises bibliográficas de sistemas hospitalares foi possível realizar o levantamento de requisitos para o sistema proposto. Buscou-se contemplar as principais atividades que envolvem a análise do quadro clínico dos pacientes, assim como os registros e decisões que devem ser realizados pela equipe médica. O quadro 1 expressa os requisitos funcionais definidos para o projeto, tanto em funcionalidades exigidas pelos usuários para que seja possível a realização de uma solução médica quanto os requisitos para o sistema proposto, dadas as formas de interação entre o usuário e o sistema.

(continua)

ID	Nome	Descrição
RF01	Log in/Log out.	O usuário deve realizar log in no sistema para utilizar as funções do mesmo. O mecanismo também permite que cada classe de funcionários faça uso das funções específicas atribuídas a ele. Ao final da sessão, deve ser possível fazer log out do sistema.
RF02	Visualizar informações do paciente.	O sistema precisa fornecer às principais informações a respeito do paciente, tal como: nome completo, data de nascimento, idade, sintomas, diagnóstico e quarto do hospital em que o mesmo se encontra.
RF03	Monitorar paciente.	O aplicativo deve suportar a função de exibir os sinais vitais atuais mensurados do paciente. A frequência de atualização dos sinais deve ser constante e suficiente para avaliar a condição momentânea do mesmo.
RF04	Diagnosticar paciente.	O aplicativo deve fornecer a função de associar um paciente a um dos diagnósticos cadastrados no banco de dados do sistema.
RF05	Visualizar histórico médico do paciente.	O sistema deve disponibilizar a visualização do histórico médico do paciente.
RF06	Registrar uma solicitação.	O aplicativo precisa fornecer uma opção para o registro de uma solicitação. A solicitação pode conter um procedimento médico ou estar associada a um medicamento cadastrado no banco de dados do sistema, incluindo ainda informações a respeito da dosagem a ser utilizada.
RF07	Registrar uma evolução do quadro clínico.	O sistema deve implementar a função de registrar uma evolução do quadro clínico do paciente. Nesse registro, haverá uma descrição de como ocorreu essa evolução, assim como a data.
RF08	Geração de alertas médicos.	O aplicativo deve gerar notificações no smartphone. Quando um dos pacientes do médico autenticado no aplicativo apresentar um de seus sinais vitais fora da faixa de valores normais para ele registrado, um alerta deverá ser enviado informando o médico da situação. Também são geradas notificações para avisar o médico que uma solicitação realizada por ele foi atendida pela equipe de enfermagem.
RF09	Visualizar e responder uma solicitação.	O sistema deve fornecer um meio para a visualização das solicitações registradas, assim como a possibilidade de resposta para as mesmas.
RF10	Manutenção dos pacientes do hospital.	O sistema precisa conceder um meio para o cadastro, edição e remoção dos pacientes do hospital.
RF11	Manutenção dos usuários do sistema.	O sistema precisa conceder um meio para o cadastro, edição e remoção dos usuários do sistema.

RF12	Manutenção dos medicamentos cadastrados.	O sistema precisa fornecer um meio para o cadastro e remoção dos medicamentos salvos no sistema.
RF13	Manutenção das doenças registradas no sistema.	O sistema precisa conceder um meio para o cadastro e remoção das doenças salvas no sistema (tabela CID).

Quadro 1 - Requisitos funcionais do sistema.

Para identificar os requisitos não funcionais, analisou-se as restrições do projeto. Nesse ponto, levou-se em consideração a plataforma na qual o usuário irá interagir com o sistema. Dispositivos móveis possuem uma interação com o usuário de forma peculiar, baseada em telas sensíveis ao toque. Assim, é necessário que funcionalidades sejam otimizadas para esse propósito. Também, para garantir que as questões de mobilidade, uma das preocupações centrais desse estudo, sejam usufruídas em sua total capacidade, deve-se ter atenção no que diz respeito ao consumo de bateria. O quadro 2 apresenta a relação de requisitos não funcionais extraídos para o sistema.

(continua)

ID	**Nome**	**Descrição**
RNF01	Compatibilidade	Deve ser possível executar a aplicação no maior número de versões da plataforma Android possível. Além disso, deve-se garantir que a aplicação funcione corretamente e ofereça uma experiência semelhante em todos os dispositivos, independente das características físicas do mesmo.
RNF02	Usabilidade	O sistema deve ser simples de usar, de modo que sejam exigidas poucas interações para acessar a função desejada. Ademais, deve-se assegurar que o usuário encontre facilmente as informações que procura, consistindo assim em um uso prático e intuitivo.
RNF03	Segurança de log in	O processo de autenticação deve possuir uma segurança mínima, assegurando que as funções não serão acessadas para fins indevidos.
RNF04	Desempenho quanto ao tempo de resposta	Deve-se garantir que as operações no aplicativo e a troca de informações com o servidor ocorram dentro dos limites de tempo onde não há prejuízo na fluidez da aplicação.
RNF05	Desempenho quanto ao consumo de bateria	Deve-se certificar de não ocorram atividades desnecessárias durante a execução do aplicativo e buscar reduzir a utilização da internet quando possível.

RNF06	Tratamento de erros	O sistema deve tratar erros inerentes a sua execução, como problemas na comunicação com o servidor, armazenamento de dados e conexão com a internet. Sempre que necessário, o usuário deve ser avisado a respeito do problema.

Quadro 2 - Requisitos não funcionais do sistema.

Alguns aspectos dependem de outros serviços para operarem de forma adequada. Para que tanto a parte do servidor quanto a aplicação na qual o usuário interagirá com o sistema possam operar e se comunicar corretamente a fim de entregar todas as funções explicitadas, é mister que se atente a algumas limitações. A seguir, essas limitações são esclarecidas, descrevendo alguns pontos decisivos tomados durante a fase de planejamento do sistema.

ID	**Nome**	**Descrição**
L01	Conexão com a internet	Para a comunicação entre as aplicações e o servidor se faz necessário uma conexão com a internet. No caso do dispositivo móvel, essa limitação não é suprida apenas com a possibilidade de conexão por parte do aparelho, visto que ainda depende da disponibilidade de uma rede sem fio no local ou um plano de internet móvel.
L02	Play Services	Para que o Google Cloud Messaging funcione, é necessário ter o Google Play Services instalado no dispositivo Android.

Quadro 3 - Limitações do sistema.

Conforme mostrado no quadro 3, a comunicação entre o servidor e as aplicações necessita de uma conexão com a internet. Uma possibilidade estudada para amenizar essa limitação foi a criação de um modo off-line na aplicação desenvolvida para dispositivos móveis. Para isso, poderia ser utilizado um banco de dados SQLite interno ao Android, o qual seria mantido sincronizado com o banco de dados externo MySQL. Assim, o usuário teria a possibilidade de utilizar o aplicativo mesmo sem internet, baseado nas informações salvas no banco de dados interno. Para implementar um mecanismo de sincronismo entre os bancos, uma solução seria acrescentar um campo em cada tabela do banco de dados externo indicando o estado do registro (sincronizado ou não). Então, periodicamente, verifica-se se houve uma alteração no banco de dados externo. Caso houver, essas informações são requisitadas junto ao servidor e salvas no banco de dados interno. Contudo, as funções de monitorar um paciente em tempo real e receber alertas médicos não funcionariam sem uma conexão com a internet. Além disso, o banco de dados interno ficaria desatualizado em relação ao banco de dados externo. Diversos aplicativos disponibilizados na Google Play funcionam de maneira semelhante, entretanto, para uma aplicação médica, essa abordagem não é interessante. Um

médico não pode basear suas decisões a respeito das condições de saúde do paciente em informações desatualizadas. Ademais, as funcionalidades que envolvem o registro de alguma informação só seriam, de fato, armazenadas no banco de dados principal na próxima vez que a conexão com a internet fosse estabelecida. Devido a esses motivos, essa alternativa foi descartada, optando por manter apenas o uso do aplicativo em modo online. O banco de dados interno ao Android, todavia, foi utilizado com outra função: armazenar dados no dispositivo para não precisar realizar uma comunicação com o servidor em tempo de execução. Mais detalhes sobre esse recurso serão discutidos no próximo capítulo.

O GCM (*Google Cloud Messaging*) é um serviço da Google que possibilita a troca de mensagens entre um servidor e um aplicativo. Para que esse serviço funcione corretamente no Android, é necessário ter o Google Play Services instalado no dispositivo. Esse software vem instalado de fábrica, ou sua instalação é solicitada nas primeiras utilizações do aparelho. Assim, apesar de consistir em uma exigência para o uso do aplicativo, não é um problema. O serviço só não estará disponível se o usuário realizar a desinstalação manualmente, sendo que dificilmente há razão para isso. No capítulo que trata da implementação do sistema, será detalhado o funcionamento do GCM, sua função no sistema e o motivo da escolha desse serviço.

Nesta seção, deixou-se claro as funcionalidades, restrições e limitações do sistema a ser desenvolvido. Uma vez que se tem uma ideia clara do que deve ser realizado, pode-se passar para as próximas etapas do processo de modelagem, as quais abordarão diferentes aspectos do projeto.

5.2. Modelo de Casos de uso

O modelo de casos de uso representa as funcionalidades externamente observáveis do sistema e dos elementos externos que interagem com ele. Segundo Bezerra (2006, p.46): “Um caso de uso representa quem faz o que com o sistema, sem considerar o comportamento interno do sistema”. Ou seja, descreve um cenário que mostra as funcionalidades do sistema do ponto de vista do usuário. Portanto, pode-se dizer que essa atividade força os desenvolvedores a moldarem o sistema de acordo com o usuário. Trata-se de um modelo bastante importante, uma vez que é a base para os demais processos de modelagem do sistema.

O modelo estabelece um contrato entre as partes interessadas sobre o comportamento do sistema. Esse processo descreve o sistema sobre várias condições conforme ele responde às solicitações das partes interessadas. Inicialmente, devem ser identificados os atores - elementos externos que interagem com o sistema. Então, deve-se representar os objetivos dos atores. Ou seja, a interação de cada um desses atores com o sistema é estabelecida, definindo as funcionalidades do mesmo. Basicamente, um ator primário inicia uma interação com o sistema com o objetivo de atingir um determinado objetivo. O sistema irá responder, levando o usuário a um cenário seguinte, com diferentes opções de interação. Assim, diferentes sequências podem gerar diferentes cenários. O objetivo do modelo de casos de uso é justamente esse, reunir uma coleção dos diferentes cenários possíveis e documenta-los. Também podem participar das ações atores secundários – não participam diretamente da ação, mas possibilitam que uma ação seja tomada por atores primários. Para descrever esses cenários, o modelo de casos de uso faz uso de diagramas e narrativas (COCKBURN, 2001).

Muitas vezes, documenta-se os casos de uso através de uma sequência de passos para realizar uma das funcionalidades do sistema. As pessoas têm mais facilidade de entender as sequências de passos descritas que constituem uma atividade maior. Baseado nisso, sua representação simples permite a compreensão para usuários leigos (sem conhecimento técnico no desenvolvimento e/ou implementação de sistemas).

Para fazer o modelo de casos de uso, é necessário saber os requerimentos, em certo nível, do sistema. Então, uma vez realizado esse modelo, ele ajuda a compreender os requisitos, visto que as informações são representadas de outra forma, dando uma nova perspectiva ao desenvolvedor e propiciando uma análise diferenciada.

Durante o desenvolvimento desse modelo, o foco deve estar em capturar as ações dos usuários e as respostas do sistema associadas, sem entrar em questões internas de como são geradas. Tanto o fluxo normal das ações como os fluxos alternativos devem ser analisados. Em outras palavras, deve se descrever tanto os acontecimentos usuais quanto os atípicos, como, por exemplo, quando o usuário tenta realizar uma ação que pode vir a dar errado. Deste modo, a elaboração deste modelo possui grande importância na questão de tratamento de erros. (ROSENBERG; STEPHENS, 2007).

Apesar do modelo de casos de uso tratar da interação do usuário com o sistema, a documentação deve ser o mais independente possível da interface. A interface de um sistema geralmente sofre modificações constantes e, se a documentação a retratar de forma específica, uma alteração na interface resultará numa modificação da narrativa do caso de uso. Para

exemplificar essa consideração, o ideal é utilizar o termo “Envia uma requisição” ao invés de “duplo clique sobre o botão de envio” (BEZERRA, 2006).

Baseando-se nos requisitos levantados na seção anterior, construiu-se o modelo de casos de uso do sistema. A figura 14 exibe o diagrama de casos de uso elaborado.

O diagrama permite perceber de forma clara os objetivos dos usuários. Os requisitos levantados estão agora associados a um ator que os executa. Os atores primários são os funcionários do hospital, que irão interagir diretamente com o sistema. O paciente aparece como um ator secundário, uma vez que não terá interação direta com o sistema, mas irá possibilitar a realização das ações por parte dos atores primários. Nota-se ainda que o diagrama apresenta limitações retangulares. Essas limitações são uma representação das fronteiras do sistema, para evidenciar o que é interno e externo a ele. No caso do médico, a interação ocorre com a aplicação desenvolvida para Android, que irá se comunicar com o servidor. No caso das enfermeiras, recepcionistas e administradores, a interação é por meio de um website, utilizando um navegador.

Basicamente, o médico terá acesso a todas as informações relevantes ao quadro clínico do paciente e poderá, através do aplicativo, registrar informações no sistema. As enfermeiras poderão visualizar e responder as solicitações após sua conclusão. A recepcionista (e demais funcionários do gênero), farão a manutenção das informações dos pacientes no sistema. Por sua vez, os usuários com privilégios de administrador serão responsáveis peça gerência geral do sistema. Todas as funções requerem que os usuários estejam autenticados.

Percebe-se, ainda, que para a enfermeira são necessárias duas relações de inclusão: o caso de uso base corresponde a responder uma solicitação e inclui os casos de visualizar uma solicitação e atender um paciente. Nessa situação, os casos de uso incluídos devem preceder o caso de uso base. Ou seja, antes de responder uma solicitação, a enfermeira deve visualizá-la no sistema e realizar o atendimento ao paciente. A UML não possui um relacionamento que represente, de forma gráfica, um caso de uso como pré-requisito de outro. Para evidenciar esse comportamento, é necessário descreve-lo na documentação do modelo.

O diagrama de casos de uso possibilita uma forma visual de análise, permitindo uma investigação de alto nível. Contudo, a documentação do modelo é essencial, visto que contém informações mais detalhadas sobre a realização das funções. No Apêndice A pode ser encontrada a documentação do modelo, com detalhes sobre o fluxo normal, fluxos alternativos e exceções. Para descrever como um caso de uso ocorre, foi utilizada a abordagem de sequência de passos mencionada anteriormente.

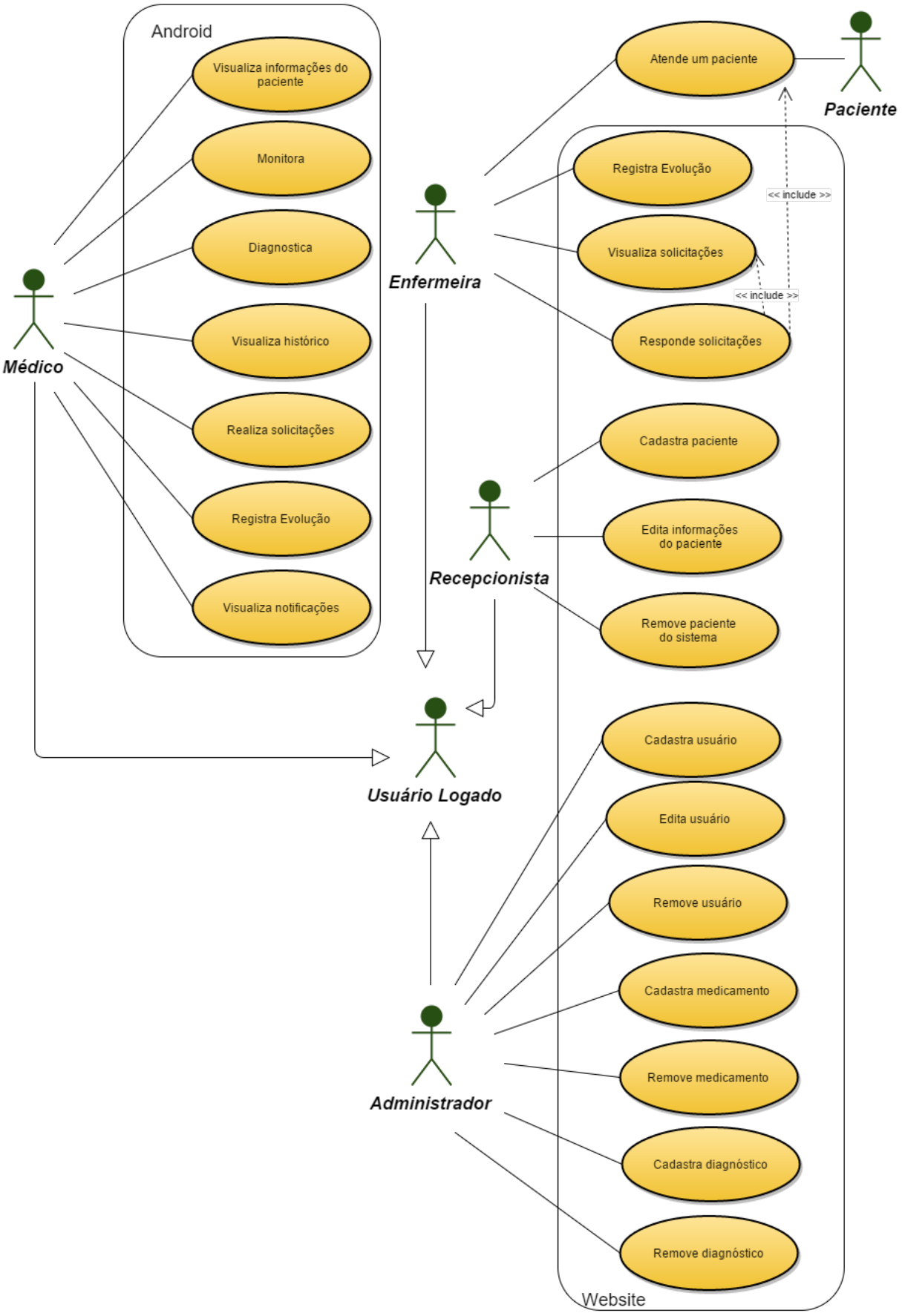


Figura 14 - Diagrama de casos de uso.

Ao oferecer uma visão abstrata das funcionalidades, a elaboração desse modelo auxilia na manutenção dos requisitos. Por ser realizada uma descrição mais detalhada de como se dá cada funcionalidade, este modelo serve como base para posteriores testes funcionais e molde para futuros manuais (STADZISZ, 2002b).

5.3. Modelo de classes (de domínio)

O diagrama de classes visa capturar a estrutura do sistema e é considerado por muitos autores como o mais importante e o mais utilizado diagrama da UML. No modelo, são abordadas as classes que compõem o sistema, a relação estática que existe entre elas e como se complementam e transmitem informações entre si. É dito estático pois não apresenta informações sobre como os objetos do sistema interagem no decorrer do tempo. Além disso, esse modelo demonstra as propriedades e operações das classes (FOWLER, 2003).

Uma classe pode ser entendida como descrições genéricas de entidades do mundo real. Ou seja, é um descritor para um conjunto de objetos com estrutura, comportamento e relações similares. A notação para uma classe consiste em um retângulo contendo o nome, atributos e operações (STADZISZ, 2002b). Além da associação entre as classes, também se denota a multiplicidade – determinação do número de objetos envolvidos em uma associação.

O primeiro passo no desenvolvimento desse modelo é a identificação das classes que compõem o sistema. A existência de uma classe só se justifica se ela participar do comportamento externamente visível do sistema. Deste modo, é possível derivar as classes a partir da descrição dos casos de uso, denominando o sistema dirigido a casos de uso. Para tanto, destacam-se os substantivos utilizados na descrição, os quais constituirão as classes candidatas para o diagrama. Caso não seja necessário manter informações sobre uma classe candidata, ela pode ser descartada. Já para identificar as responsabilidades das classes, analisa-se os verbos que representam as ações (BEZERRA, 2006). A partir disso, pode-se fazer um refinamento das informações obtidas. Primeiramente, cria-se um modelo utilizando grandes classes e, após, decompõem-se estas classes em classes menores, mais detalhadas. Adota-se assim uma abordagem *top-down* (RUMBAUGH; JACOBSON; BOOCH, 2005). A figura 15 demonstra a interdependência entre os modelos.

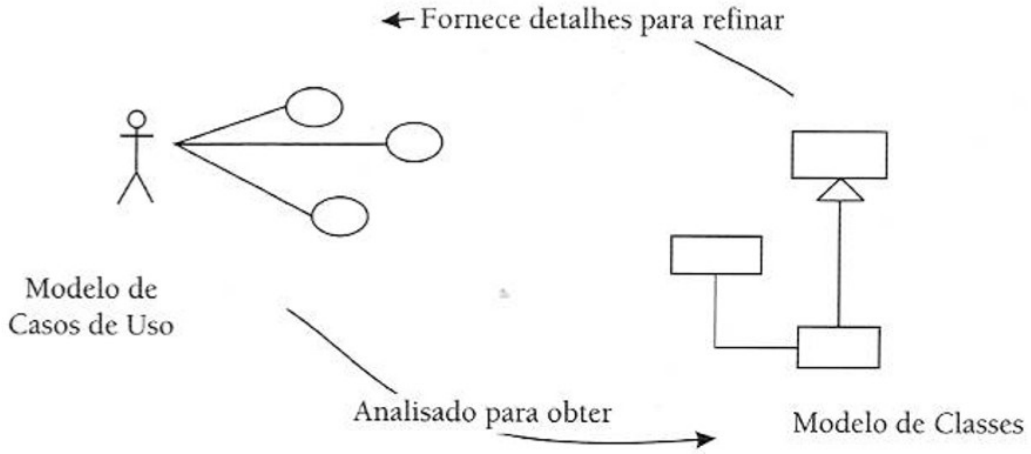


Figura 15 - Interdependência entre o modelo de casos de uso e o modelo de classes.

Fonte: BEZERRA, 2006.

O modelo de classes se foca exclusivamente na estrutura do sistema e ignora o seu comportamento. Para uma melhor compreensão acerca do software, é interessante o desenvolvimento desse modelo em conjunto com alguma técnica comportamental. Assim, pode-se alterar frequentemente de perspectiva e obter uma melhor noção do projeto (FOWLER, 2003).

Para a construção do diagrama de classes desse projeto, adotou-se uma perspectiva conceitual. Nessa abordagem, são exibidas as classes conceituais de uma situação real, ao invés de classes de software. Isto é, o diagrama foi projetado para representar os conceitos do domínio que está sendo estudado, com pouca preocupação com o software que poderá implementá-lo, sendo independente da linguagem utilizada. A principal ideia dessa abordagem é propiciar um entendimento dos principais elementos do domínio (SILVA, 2013). A figura 16 representa o diagrama de classes construído.

Utilizando a técnica dirigida a casos de uso, identificou-se as principais classes do sistema e suas relações. Manteve-se apenas as classes que, de fato, necessitam ter suas informações armazenadas e serão utilizadas durante o funcionamento do sistema. Limitou-se em adicionar os atributos importantes para a compreensão dos conceitos, a fim de não tornar o modelo muito complexo desnecessariamente. Então, os atributos presentes consistem nas informações principais que definem as características da classe. Na fase de implementação, pode-se estender os atributos, completando-os com informações mais detalhadas. Os métodos descritos adotam o mesmo raciocínio e são formados, basicamente, de *gets* (consultas) e *sets* (cadastros), além de funções mais específicas de cada classe que se sabe de antemão que serão necessárias, como por exemplo, verificação de login e análise dos sinais vitais. Tais métodos

devem ir ao encontro das funções que serão implementadas pelo servidor, uma vez que caberá a esse realizar o acesso ao banco para obter e registrar os dados. Por ser um modelo conceitual e não se preocupar com implementação, não há informação de visibilidade dos atributos e métodos.

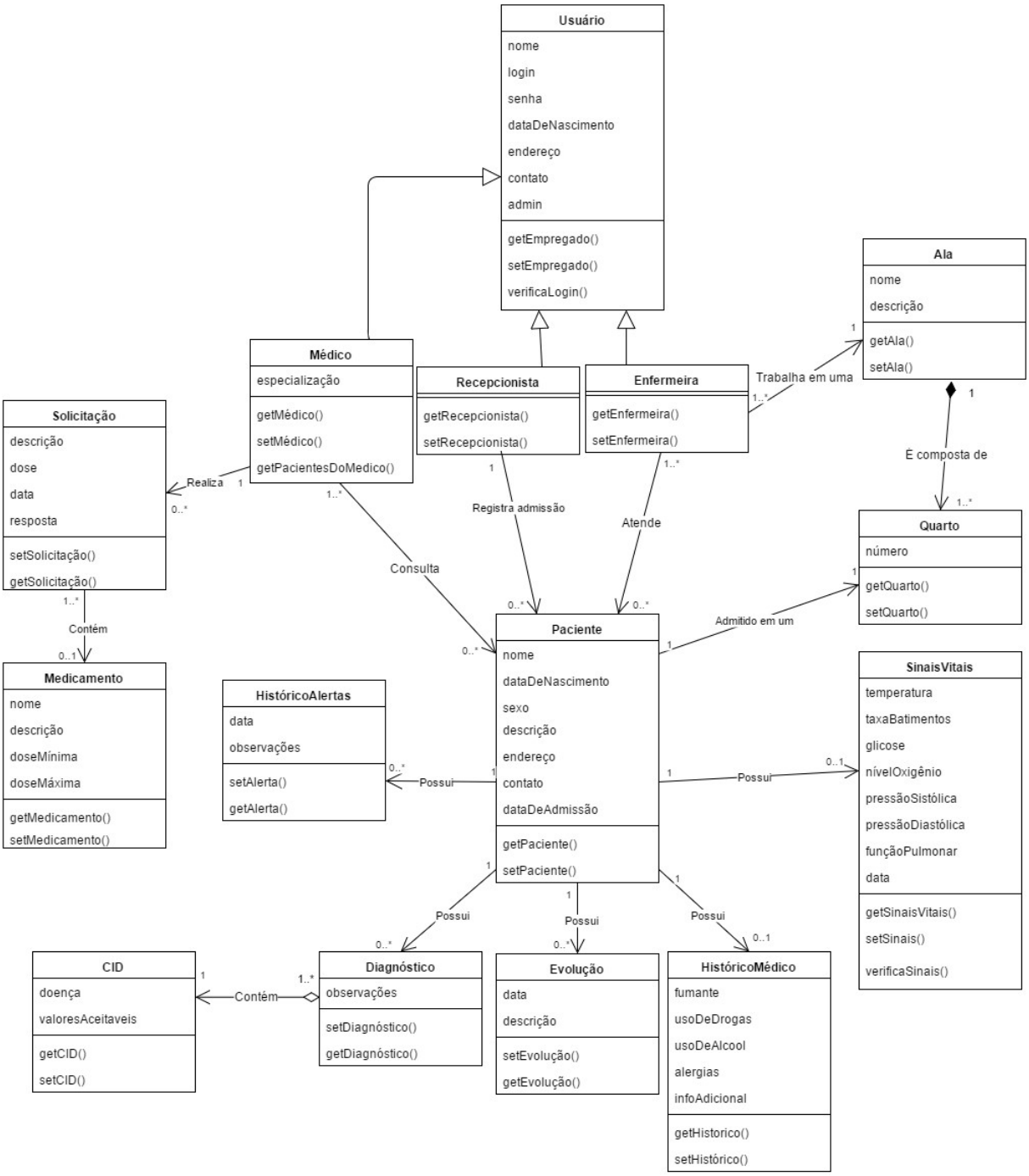


Figura 16 - Diagrama de classes.

Para a representação dos usuários, utilizou-se o conceito de herança. Por meio desse mecanismo, as características comuns às classes do médico, recepcionista e enfermeira foram colocadas em uma classe base (superclasse), a de usuários do sistema. As subclasses, portanto, herdam seus atributos e métodos, simplificando o diagrama. A seguir é feita uma breve descrição das classes identificadas, suas funções e relações dentro do sistema proposto e seus principais atributos:

Usuário: Consiste na classe base dos profissionais do hospital que interagem com o sistema. Possui informações básicas a respeito do funcionário e informações de log in. As classes Médico, Enfermeira e Recepcionista herdam essa classe. Um usuário pode ter privilégios de administrador, tendo acesso às funções restritas do sistema.

Médico: Possui um atributo para salvar a especialização do médico. No sistema, consulta seus pacientes por meio da visualização de informações a respeito do quadro clínico dos mesmos.

Recepcionista: Realiza a manutenção dos dados dos pacientes, tendo a possibilidade de cadastrar, editar ou remover pacientes do sistema.

Enfermeira: De acordo com as solicitações do médico, realiza o atendimento a um paciente que se encontra na mesma ala em que a enfermeira trabalha.

Ala: Corresponde a uma ala do hospital, a qual é formada por quartos. As enfermeiras têm seu trabalho limitado a uma das alas.

Quarto: Constituem as alas de um hospital. Um quarto tem a capacidade de alocar um paciente.

Paciente: Representa um paciente internado no hospital. A classe possui as informações pessoais e clínicas do paciente, como a descrição dos sintomas.

HistóricoMédico: Associado a um paciente, contém informações a respeito do uso de substâncias nocivas à saúde, alergias e informações adicionais, que podem conter procedimentos cirúrgicos e complicações anteriores, por exemplo. O médico, na consulta, tem acesso a esses dados.

HistóricoAlertas: Representa um histórico contendo todos os alertas médicos do paciente durante a sua internação.

SinaisVitais: Associado a um paciente, consiste no último sinal monitorado do mesmo. O médico pode visualizar essas informações.

CID (Classificação Internacional de Doenças): Classe para representação de uma doença. Contém o nome da doença e a faixa de valores aceitáveis dos sinais vitais de um paciente que a possui.

Diagnóstico: corresponde ao registro de um diagnóstico por parte do médico. Portanto, possui uma doença da tabela CID associado. Um paciente pode possuir vários diagnósticos associados.

Evolução: representa o registro de uma evolução clínica do paciente. Um paciente pode possuir vários registros associados a ele. Médicos e enfermeiras podem realizar o registro de uma evolução do paciente.

Solicitação: representa uma solicitação feita por um dos médicos. Pode ou não conter um medicamento associado. No caso de envolver um procedimento, o mesmo deve ser descrito no campo "descrição".

Medicamentos: representa os medicamentos disponíveis no hospital. Contém nome, informações de dosagem e descrição do conteúdo/posologia.

A construção do modelo de classes, através do aspecto estrutural estático de uma cooperação, permite compreender como o sistema está estruturado externamente para que as funcionalidades sejam produzidas (ROSENBERG; STEPHENS, 2007). Além disso, o panorama que o modelo fornece é de suma importância para a modelagem e implementação do banco de dados, uma vez que se realizou um levantamento inicial das informações que devem ser armazenadas pelo sistema e a base das relações entre elas.

5.4. Diagrama de atividades

Um diagrama de atividades é utilizado para modelar os aspectos comportamentais (dinâmicos) do sistema. Para tanto, modela-se o fluxo de ações que são parte de uma atividade maior. Ou seja, descreve-se os passos sequenciais (e, às vezes, concorrentes) para a realização de uma atividade. (RUMBAUGH; JACOBSON; BOOCH, 1998).

Uma ação representa um passo elementar de uma atividade, isto é, um passo que não pode ser decomposto. Somente quando uma ação é terminada que a ação subsequente é habilitada. Atividades, por sua vez, são representadas por sequências de ações e também de subatividades. Além desses componentes, um diagrama de atividades também pode conter nós de controle, assim chamados por controlar o fluxo do processo (GUDWIN, 2010). Esses nós de controle são responsáveis pelas ramificações e sincronização do fluxo, tornando o diagrama, apesar de parecido, mais poderoso que fluxogramas tradicionais (RUMBAUGH; JACOBSON; BOOCH, 2005).

O modelo de casos de uso, em comparação, também apresenta uma descrição das funções. Entretanto, o foco é em reunir todos os cenários possíveis e documentar a resposta do sistema para eles. Em contraste, o diagrama de atividades se concentra na sequência de execução e nas condições que engatilham as mesmas, descrevendo-as em um nível maior de detalhes e as exibindo de forma gráfica. Assim, o modelo de casos de uso não é uma condição necessária para a realização do diagrama de atividades, mas um complemento ao mesmo – os modelos podem ser totalmente independentes, dependendo do projeto e do nível em que se está trabalhando (BELL, 2003).

Muitas vezes, é útil modelar o fluxo entre objetos que participam de uma atividade. Para isso, pode-se utilizar partições, que consistem em colunas verticais separando os objetos. Desta forma, pode-se indicar que diferentes ações são executadas por diferentes agentes dentro de um processo. No caso de desenvolver o diagrama de atividades para modelar os casos de uso, uma partição é utilizada para representar o usuário e outra para representar o sistema (BELL, 2003) (GUDWIN, 2010).

Para simplificar a representação do modelo, em vez de construir um diagrama para cada uma das atividades possíveis dentro do sistema, optou-se por elaborar um diagrama por tipo de usuário, os quais possuem acesso a diferentes funções. Essa abordagem parte da identificação de que as atividades possuem seu início de forma semelhante, diferindo apenas nas opções de ações que o usuário toma, dependendo da função a ser exercida. A figura 17 exibe o diagrama de atividades para o aplicativo Android demonstrando a interação do médico com o sistema.

Conforme pode ser observado, foram criadas três partições, sendo uma com as ações do usuário (médico), uma com as ações específicas da aplicação Android e a última representando o comportamento do servidor. Assim, fica claro o papel de cada parte na realização das funcionalidades implementadas.

Cita-se também que o diagrama contém todas as funcionalidades do aplicativo. Tais funcionalidades são tratadas como subatividades de uma atividade maior: o uso da aplicação. Ao selecionar um dos pacientes, o médico se depara com um menu de opções com todas as ações possíveis para um dado paciente. Em qualquer uma dessas subatividades o menu estará visível, permitindo uma utilização fácil e intuitiva, que exige o mínimo de toques na tela para encontrar o que deseja. Desta forma, uma subatividade também pode ser encerrada a qualquer momento. No diagrama, esse comportamento pode ser notado pela presença do nó de fim de fluxo (círculo com um “x”), que representa o final do fluxo atual, sem necessariamente ter encerrado o uso da aplicação.

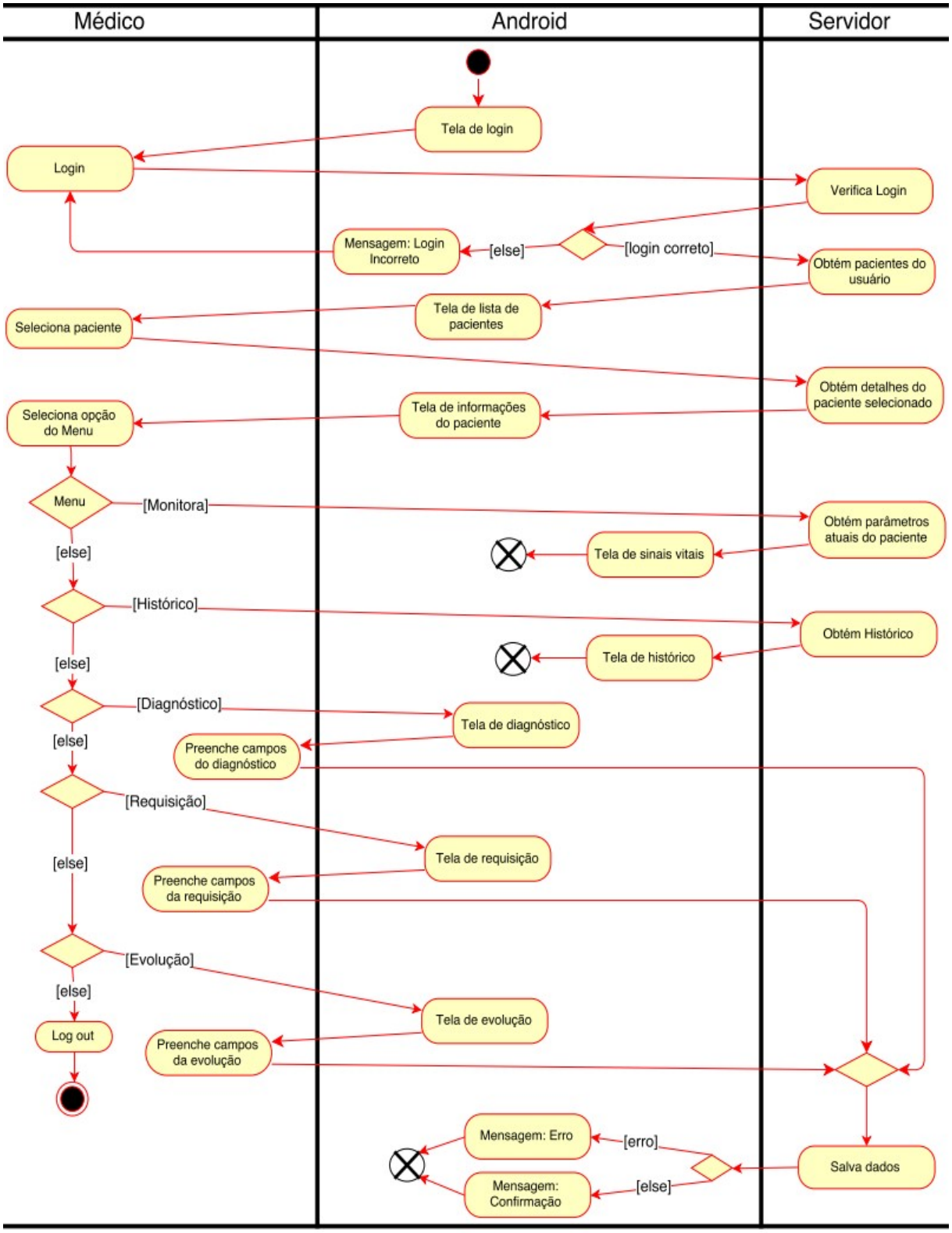


Figura 17 - Diagrama de atividades para o médico utilizando o aplicativo.

Embora não esteja visível no diagrama, a cada requisição feita pelo usuário, o Android é responsável por fazer o envio dos dados para o servidor e obter sua resposta. Como todas as

ações do usuário resultam em uma comunicação com o servidor, representar isso no diagrama requer que todas as ações se conectem a um estado de envio, tornando o diagrama complexo e, de certa forma, redundante. Apesar de não haver uma ação desenhada explicitamente na partição do Android, esse comportamento é claro, uma vez uma ação do usuário está conectada a uma ação do servidor passando, necessariamente, pela partição do Android.

Além disso, é possível perceber por meio do diagrama uma hierarquia de telas do aplicativo. Com exceção das ações de mensagem, as outras ações presentes na partição do Android correspondem as telas que deverão ser implementadas através de *activities* ou utilizando *fragments*. Com isso, tem-se uma boa perspectiva da utilização do aplicativo. Mais informações a respeito da implementação da aplicação Android serão abordadas no capítulo 6.

Utilizando os mesmos princípios, foram elaborados os diagramas para os usuários que interagem com o sistema por meio do website desenvolvido. As figura 18, figura 19 e figura 20 demonstram as atividades para a enfermeira, recepcionista e administrador, respectivamente.

De forma análoga ao Android, é possível verificar uma hierarquia de páginas da web que devem ser desenvolvidas. A ideia é que as páginas tragam as informações do sistema em tabelas, as quais possuem botões individuais para cada um de seus itens. Desta forma, busca-se atingir um uso fácil, necessitando poucos cliques para que uma atividade seja completada.

Por intermédio deste modelo, delineiam-se as funções a nível do sistema. Através do detalhamento da sequência de atividades que devem ser tomadas para se chegar a um objetivo, o modelo auxilia na construção de um sistema executável. Outrossim, identifica-se a resposta do sistema para as ações realizadas pelo usuário, especificando o seu comportamento. Desta forma, esta etapa possui grande impacto na usabilidade do sistema, sendo vital para a boa aceitação do projeto. (GUDWIN, 2010).

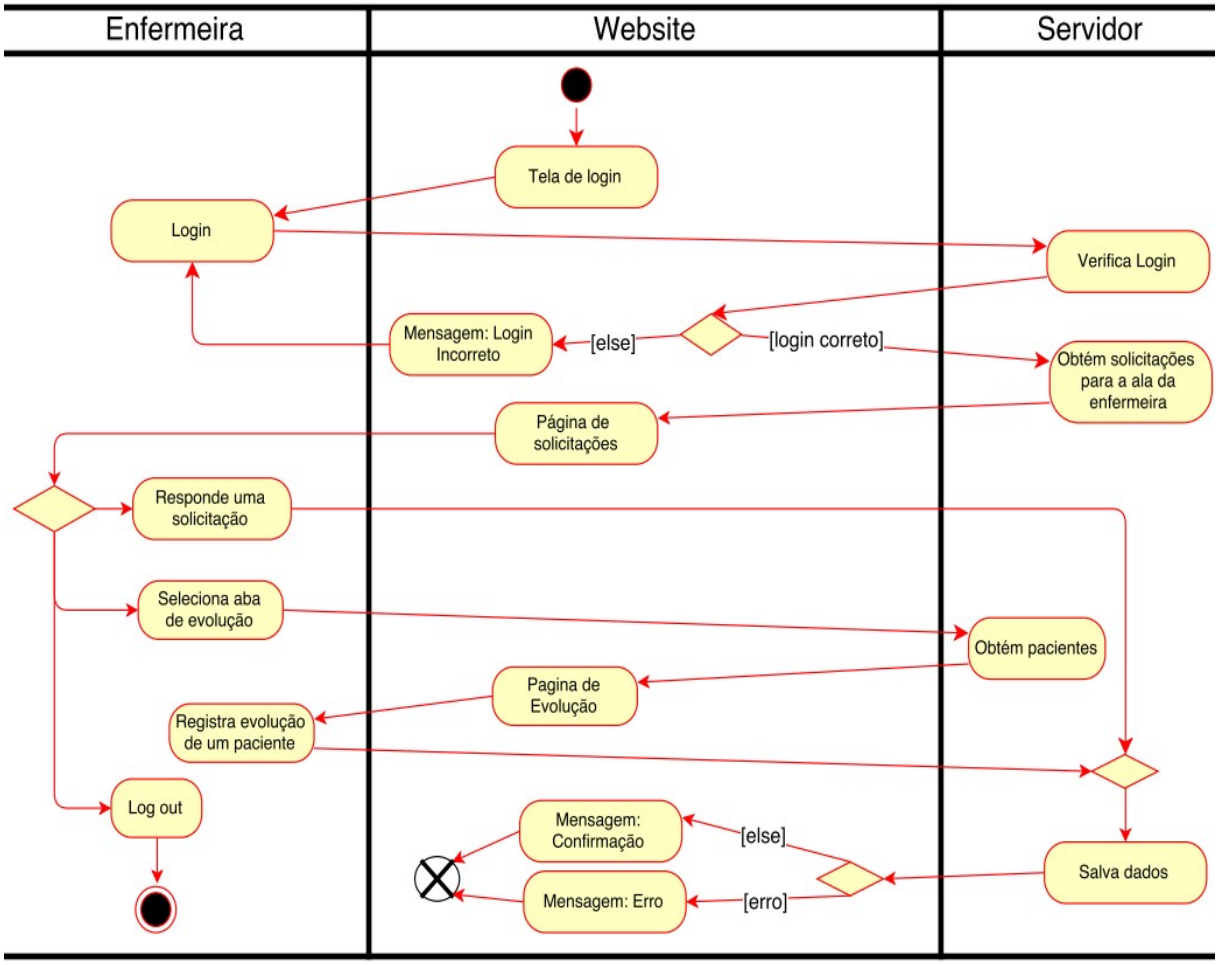


Figura 18 - Diagrama de atividades para a enfermeira.

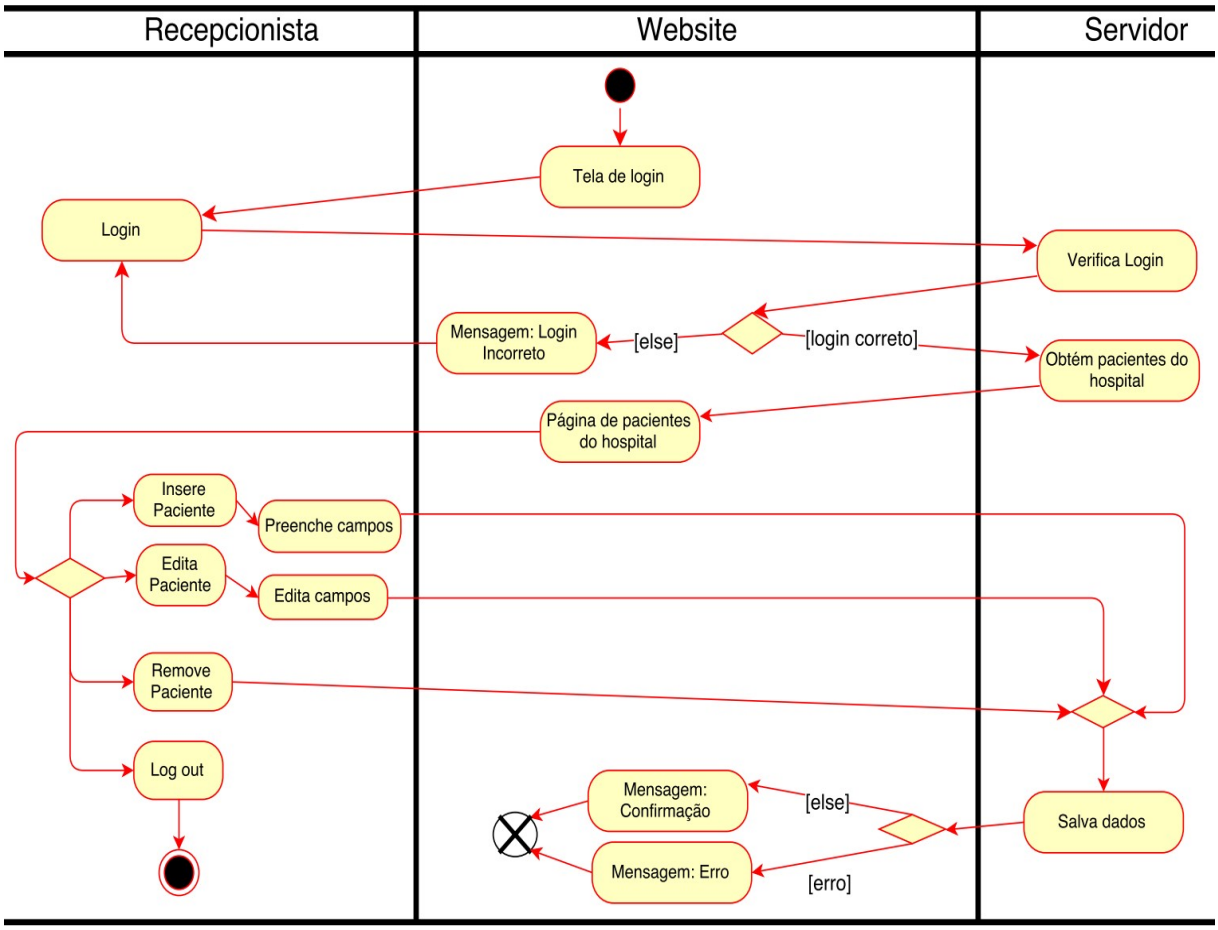


Figura 19 - Diagrama de atividades para a recepcionista.

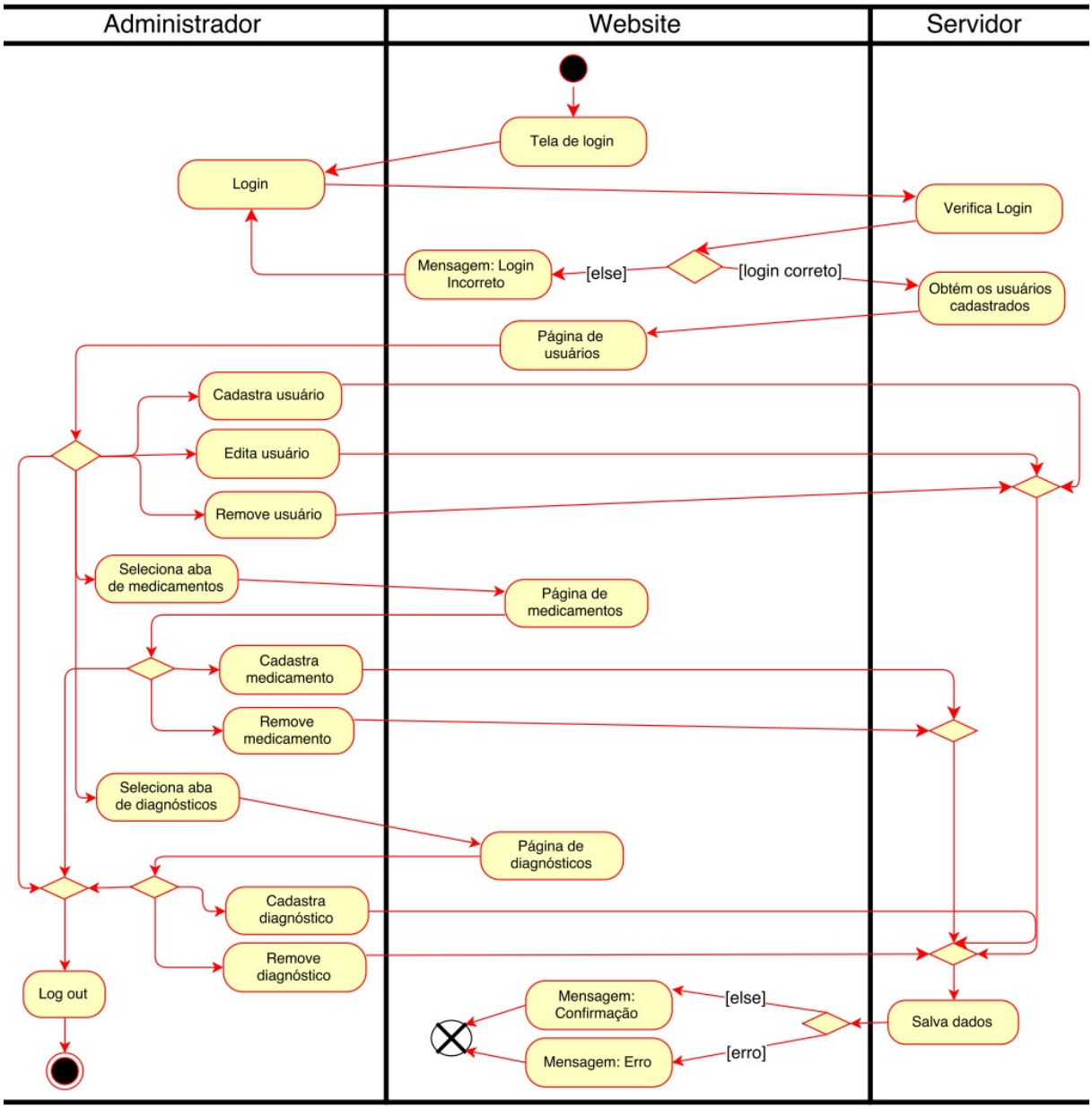
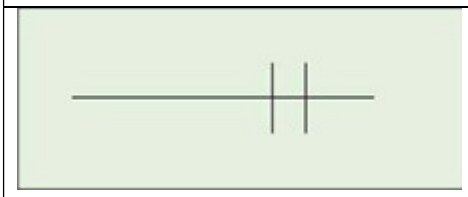
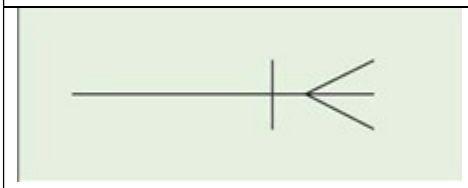
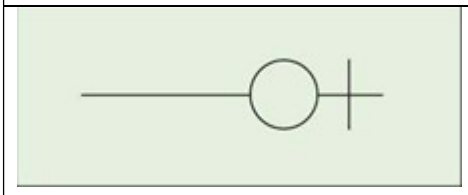
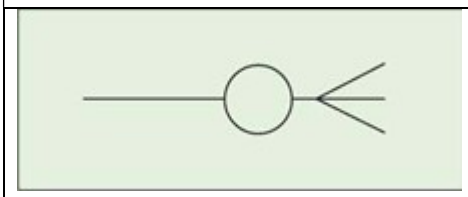


Figura 20 - Diagrama de atividades para o administrador.

5.5. Modelo Entidade Relacionamento

Estando ciente dos requisitos e funcionalidades que devem ser implementadas, é possível modelar um banco de dados que dê suporte ao armazenamento dos dados necessários. Para a modelagem do banco de dados foi utilizado o diagrama de entidade relacionamento. Apesar de não pertencer a UML, trata-se do modelo mais utilizado para projetar banco de dados. Neste trabalho, foi utilizada a notação de *Crow's foot* para o

desenvolvimento do diagrama. Trata-se de uma notação bastante adotada no projeto de banco de dados e empregada pelas principais ferramentas de desenvolvimento, como o MySQL Workbench, utilizado nesse estudo, podendo haver pequenas variações. Basicamente, as entidades são representadas por retângulos e as relações por linhas que as conectam, com símbolos especiais em cada extremidade indicando a cardinalidade da relação. Diferente do modelo de classes, aqui as entidades representam as tabelas do banco (coleção de dados) ao invés de uma instância do mesmo (ROSENBERG; STEPHENS, 2007). O quadro 4 mostra as cardinalidades possíveis entre as entidades, de acordo com a notação utilizada.

Símbolo	Significado
	1 e somente 1
	1 ou mais
	0 ou 1
	0 ou mais

Quadro 4 - Notação Crow's foot utilizada.

O diagrama de classes desenvolvido anteriormente, embora não seja focado no projeto do banco de dados, pode ser considerado como uma visão conceitual desse, uma vez que na abordagem utilizada preocupou-se em expor apenas as entidades que, de fato, necessitarão serem salvas no sistema. Assim sendo, pode-se basear nele para, agora, elaborar um modelo físico fazendo uso do diagrama de entidade relacionamento. Esse modelo irá expor de forma

mais clara como os dados serão armazenados e acessados pelo sistema. A preocupação é, portanto, com a estrutura dos dados e não com as operações sobre eles, não apresentando assim métodos associados com uma entidade (GARCIA-MOLINA; ULLMAN; WIDOM, 2009).

A modelagem do banco de dados em um nível físico se preocupa com os detalhes de implementação. Assim, a elaboração do modelo se estendeu até as fases finais da implementação, sendo constantemente revisado e atualizado. Para cada atributo de uma entidade, define-se o tipo do dado que será armazenado e as suas propriedades. Além disso, informa-se sobre *triggers* e *constraints* do projeto (SPARX SYSTEMS, 2011). A figura 21 demonstra o diagrama de entidade-relacionamento construído para o banco de dados desse projeto.

As chaves primárias foram adicionadas às entidades a fim de identificar as instâncias dentro das mesmas. Nota-se que grande parte das relações estabelecidas no diagrama de classes se mantiveram. Agora, entretanto, as relações são implementadas através das chaves primárias e associações por chaves estrangeiras. Contudo, essa forma de criar relações dentro do banco de dados não permite que se tenha, diretamente, relações vários-para-vários. Por isso, conforme pode ser observado na figura 21, foram adicionadas duas tabelas associativas: Medico_Paciente e Paciente_CID, com a função de vincular um paciente a um médico e um paciente a uma doença, respectivamente.

Com o objetivo de otimizar o armazenamento dos dados, alguns cuidados foram tomados. Alguns campos possuem a opção de adotarem um valor nulo. De modo geral, procura-se evitar que existam campos nulos dentro de uma linha da tabela, uma vez que espaço em disco estará sendo ocupado sem guardar informação. Em alguns casos, esse comportamento é inevitável. Em outros, entretanto, pode-se separar tais campos em uma nova tabela, identificada por chave estrangeira com a tabela original. Assim, caso aquelas informações não existam no sistema, não haverá uma correspondência para ela na tabela criada. Um exemplo desse comportamento é a tabela de histórico médico, onde esses atributos poderiam estar inseridos na tabela de paciente, mas optou-se pela criação de uma nova tabela. Outro ponto simples, mas que pode vir a economizar espaço é a utilização de uma chave estrangeira como própria chave primária, como no caso das tabelas associativas, por exemplo. Nesses casos, a identificação de uma instância é dada pelas próprias chaves estrangeiras, uma vez que só é permitida uma associação para cada tipo. Em sistemas de menor porte, essas providências não terão grande impacto no espaço utilizado. Porém, é sempre interessante realizar a otimização dos dados, evitando uma complexidade desnecessária.

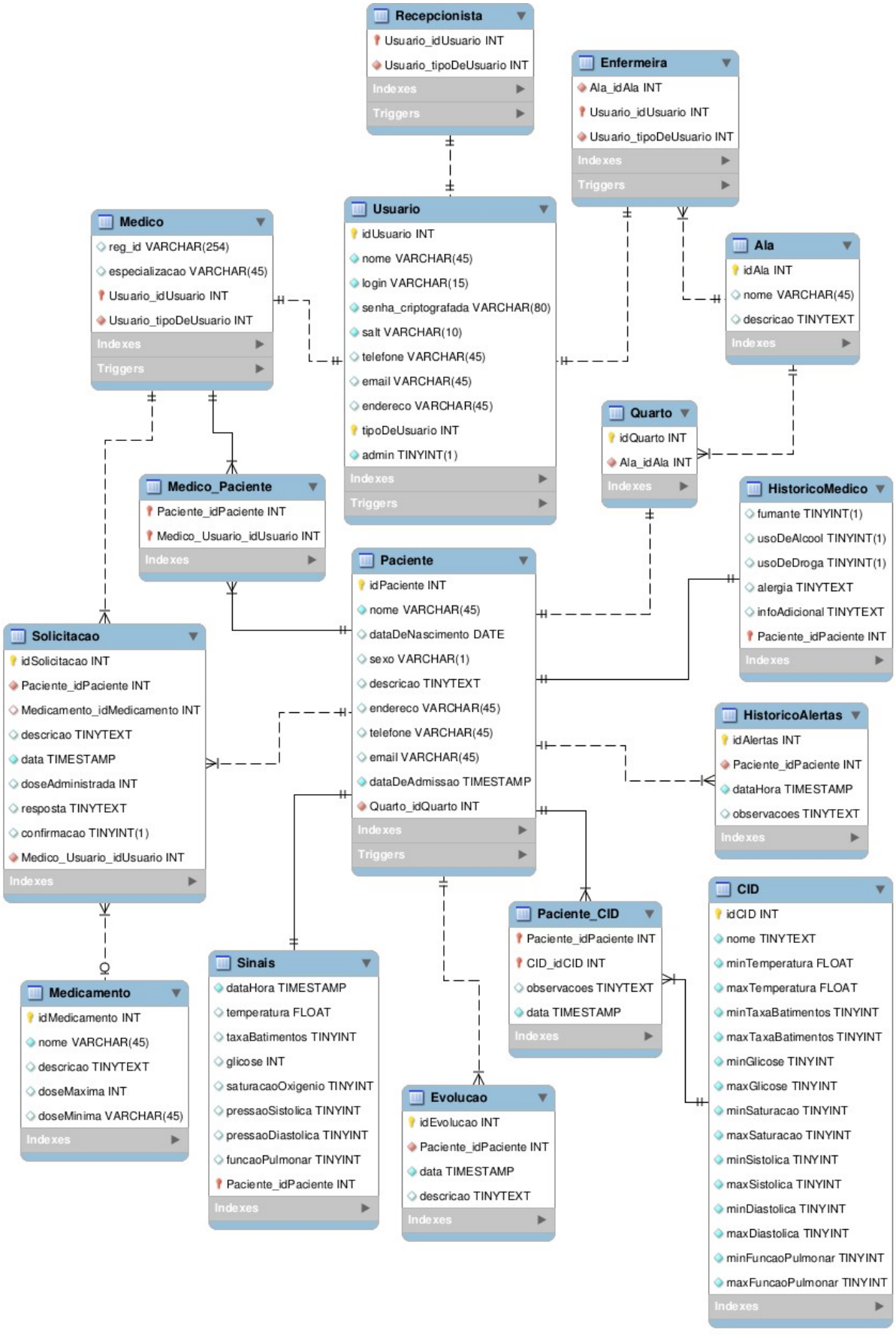


Figura 21 - Diagrama entidade-relacionamento do banco de dados.

Em relação ao diagrama de classes, algumas modificações foram feitas para suprir as exigências impostas na fase de implementação. O MySQL não suporta, de forma direta, a implementação de herança entre as tabelas. Portanto, para modelar a tabela de usuários e suas respectivas tabelas-filho conforme o diagrama de classes, deve-se utilizar as chaves primárias e estrangeiras de forma a assegurar que um usuário se enquadre somente em uma das classes de funcionários existentes. Para isso, acrescentou-se a coluna “tipoDeUsuario” como chave primária da tabela de usuários. A decisão de separar a tabela de usuários das tabelas dos funcionários partiu do princípio de que as informações devem estar o mais independente possível. Isto é, todas as informações relativas ao cadastro de um usuário no sistema se encontram na tabela de usuários e os dados específicos para cada classe se encontram como atributos das classes-filho. Para inserir ou modificar alguma informação no sistema, então, é necessário acessar somente a tabela correspondente.

Como consequência das alterações no diagrama de classes, a tabela do médico teve o acréscimo do campo reg_id, que corresponde ao ID do dispositivo móvel em que o médico está autenticado e servirá para o envio de notificações utilizando o GCM. Outra modificação que deve ser citada é o uso do campo senha_criptografada e salt na tabela dos usuários. Em conjunto, esses campos implementam o mecanismo de encriptação de senha e, posteriormente, a verificação da mesma quando o login é efetuado. Mais informações acerca do uso desses campos durante a implementação serão discutidas no capítulo 6, que trata da implementação do sistema.

Durante a elaboração do modelo, preocupou-se em evitar a redundância dos dados e estrutura-los de forma a facilitar o armazenamento e o acesso aos mesmos. Uma modelagem de dados bem realizada é de notável importância no desenvolvimento de um sistema. As decisões tomadas nessa fase impactam na forma com que os dados serão acessados pelo servidor. Geralmente, os dados armazenados em um banco de dados são compartilhados entre as aplicações. Qualquer mudança, ainda que pequena, na forma de como esses dados estão estruturados pode requerer modificações nas aplicações que acessam o banco de dados, tornando o processo complicado. Além disso, também possibilita atingir os requerimentos de desempenho do sistema (MERSON, 2009).

6. IMPLEMENTAÇÃO

Neste capítulo serão tratadas as questões relativas a implementação do sistema. Visa-se, por meio deste, debater os pontos fundamentais desse quesito, elucidando as decisões de projeto aderidas nesta etapa e minuciando os conceitos por trás da sua implementação. Ainda, é abordado o desenvolvimento das funcionalidades do sistema, garantindo o suporte aos requerimentos estabelecidos na fase de planejamento.

O capítulo se divide em seções que tratam cada uma das partes que compõem o sistema: servidor, banco de dados, Android e website. Ademais, são apresentadas seções separadas para discorrer sobre os assuntos cuja implementação depende de vários segmentos, como no caso da comunicação entre a aplicação Android e o servidor e do mecanismo de recebimento de notificações.

6.1. Servidor web

Um servidor web consiste em um servidor responsável por receber requisições HTTP de clientes web e retornar respostas HTTP, geralmente na forma de uma página da web. Neste projeto, utilizou-se um servidor Apache versão 2.4.7. A linguagem de programação empregada na implementação foi o PHP, tendo como ambiente de desenvolvimento o Netbeans.

O servidor desenvolvido recebe as requisições HTTP por meio do método POST. A requisição, seja para obter ou registrar dados, exige um acesso ao banco de dados do sistema. Após o processamento da requisição, o servidor retornará uma resposta ao cliente. No caso de requisições realizadas pelo cliente Android, as respostas serão retornadas no formato JSON (*JavaScript Object Notation*), enquanto as realizadas pelo navegador (isto é, por meio do website desenvolvido) serão retornadas na própria página, onde os dados são apresentados, de forma tabelada, ao usuário. Ao longo do capítulo, será detalhado o processo de comunicação entre as aplicações clientes e o servidor. A figura 22 exibe a estrutura do servidor projetado, descrevendo a função de cada arquivo.

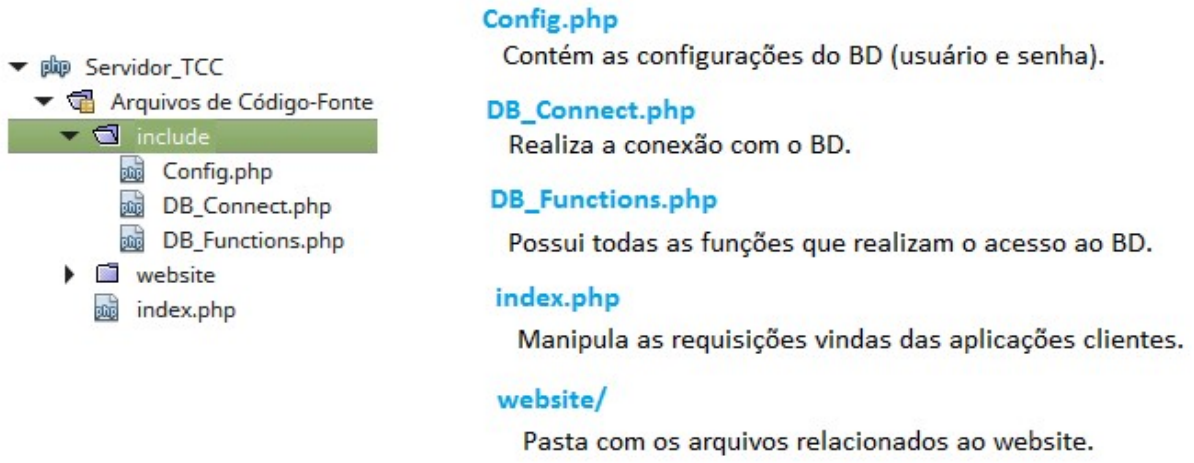


Figura 22 - Estrutura do projeto do servidor.

Além de obter e registrar informações no banco de dados, o servidor também é responsável por algumas funcionalidades mais específicas. A seguir é explicitado o processo de implementação de segurança de log in e verificação dos sinais vitais que estão sendo monitorados.

6.1.1. Segurança

Com o objetivo de adicionar um nível de segurança ao sistema, evita-se o armazenamento puro das senhas dos usuários no banco de dados. Caso a senha original fosse salva, uma pessoa mal-intencionada que conseguisse o acesso ao banco de dados poderia obter a senha de todos os usuários do sistema. Para evitar esse problema, os servidores armazenam a senha encriptada (também chamada de *hash*) no banco de dados.

O *hash* é computado aplicando um método de criptografia – que transforma a senha original do usuário em uma *string* de tamanho fixo, aparentemente aleatória. O método utilizado para gerar o *hash* é, geralmente, de mão única. Isso significa que não há uma forma de reverter a encriptação. Entretanto, se a pessoa que está tentando o ataque sabe qual a função utilizada para gerar o *hash*, é factível, através de algoritmos de força bruta, descobrir a senha original. Ou seja, é possível tentar diferentes senhas até que uma delas gere o mesmo

hash armazenado no banco de dados. Para prevenir essa falha, utiliza-se um *salt* na geração do *hash*.

O *salt* consiste em uma *string* complexa de dados que é concatenada a senha antes de encriptá-la. Desta forma, a geração do *hash* depende tanto do método de encriptação como do *salt* utilizado na sua geração. Contudo, o *salt* deve ser armazenado de alguma forma do sistema – seja em um arquivo ou no próprio banco de dados. Se o sistema utilizar um *salt* fixo (isto é, o mesmo *salt* para encriptar todas as senhas) e o atacante conseguir ter acesso a ele, então, da mesma forma que antes, através de tentativas será possível descobrir a senha.

A fim de dificultar esse processo, podem ser utilizados *salts* dinâmicos, de modo que para cada usuário cadastrado um *salt* diferente seja utilizado. Assim, a pessoa que está tentando o ataque teria que testar todas as combinações para cada um dos usuários, utilizando o seu próprio *salt*, dificultando muito o processo. Apesar de a descoberta ser difícil nesse ponto, ainda é factível, pois os métodos de encriptação foram criados para serem extremamente rápidos, visto que são utilizados na verificação de integridade de arquivos. Com isso, a descoberta da senha é uma questão de tempo – quanto mais rápido o algoritmo, mais fácil um ataque de força bruta encontra a senha original. A solução é simples: atrasar o algoritmo. Para isso, implementou-se um laço que realiza a encriptação diversas vezes. Embora simples, trata-se de uma solução eficiente. Imagine que um algoritmo de encriptação possa ser executado 1 bilhão de vezes por segundo. Utilizando um loop que execute o método 1000 vezes, a velocidade do processo cairá para 1 milhão de execuções por segundo. Em termos práticos, um ataque que antes demorava 60 horas para descobrir a senha, agora irá demorar aproximadamente 7 anos, tornando o processo impraticável.

A encriptação da senha baseou-se nos conceitos aqui introduzidos. A seguir, é exibida a sequência de passos utilizada para a implementação desse mecanismo.

1. Gerar um *salt* aleatório.
2. Concatenar a senha com o *salt* gerado.
3. Utilizando a função SHA-256 do PHP, encriptar o resultado.

Após a encriptação, tanto o *hash* quanto o *salt* foram salvos no banco de dados. O processo de verificação da senha inserida pelo usuário no momento do log in utiliza os mesmos princípios da encriptação. Os passos para a verificação são listados a seguir.

1. Obter a senha digitada pelo usuário.
2. Obter o *salt* utilizado para a geração do *hash* do usuário em questão (salvo no banco de dados).

3. Realizar a encriptação da senha digitada pelo usuário utilizando o *salt* obtido no passo anterior.
4. Verificar se o *hash* gerado nesse processo é igual ao *hash* salvo no banco de dados.

Caso o *hash* gerado no passo 4 for igual ao valor que está salvo no banco de dados do sistema, as informações inseridas pelo usuário no momento do log in estão corretas. A partir disso, a aplicação pode continuar o seu fluxo normal.

6.1.2. Verificação dos sinais vitais

Um dos requisitos funcionais do sistema é o recebimento de notificações no Android quando problemas forem encontrados com os sinais vitais que estão sendo monitorados de um paciente. A tabela CID do banco de dados consiste no registro das doenças, contendo as faixas de valores aceitáveis para cada um dos sinais vitais monitorados.

Para realizar a verificação dos sinais, os valores monitorados são comparados com os valores cadastrados, para cada um dos diagnósticos do paciente. Ou seja, obtêm-se os limites para cada doença que o paciente está associado e então os sinais vitais mensurados são comparados com esses dados. Caso um dos sinais se apresente fora dos limites estabelecidos, uma notificação GCM será gerada (ver seção 6.5).

O objetivo dessa verificação é avisar o médico de que o paciente apresentou alguma anormalidade em um dos seus sinais vitais. Essa anormalidade não precisa, necessariamente, representar uma emergência médica – qualquer variação de valor em consideração às faixas normais geram um alerta. Assim, é possível investigar a causa dessa anormalidade antes que a mesma acarrete em algo mais grave.

6.2. Banco de dados

O banco de dados desenvolvido nesse projeto utiliza o gerenciador MySQL e foi modelado por meio do software MySQL Workbench versão 6.1. Através dessa ferramenta, foi possível gerar o *script* SQL que implementa o banco. A implementação foi realizada empregando como ambiente de desenvolvimento o Netbeans.

Seguindo o diagrama da figura 21 e as decisões de projeto explicitadas no capítulo de modelagem, construiu-se o banco de dados. O principal ponto dessa etapa consiste na implementação do mecanismo de herança mutualmente exclusiva entre a tabela de usuários e as diferentes categorias do mesmo, uma vez que o MySQL não possui suporte, nativamente, a esse recurso. A seguir, é descrito como ocorreu a implementação de tal mecanismo.

A coluna “tipoDeUsuario” foi inserida como chave primária da tabela de usuários para realizar a implementação de herança entre as tabelas. Essa coluna pode assumir os valores 1, 2 e 3, significando médico, enfermeira e recepcionista, respectivamente. Nas tabelas-filho, então, essa coluna será uma chave estrangeira e se assegura que o seu valor equivale ao valor da classe em que está contida, conforme a representação adotada. Para verificar o valor da coluna “tipoDeUsuario” antes da inserção em uma das tabelas-filho, são utilizados triggers. Desta forma, um usuário cadastrado só pode estar associado a uma única classe-filho, consistindo em um modelo mutualmente exclusivo. Por exemplo, a tabela de médicos só aceita um usuário que possui o valor 1 na coluna “tipoDeUsuario”. Assim, tem-se a implementação de um mecanismo de herança, garantindo a consistência dos dados. Os demais pontos não exigiram maiores esforços, sendo possível implementá-los utilizando conceitos básicos de banco de dados.

6.3. Aplicação Android

A aplicação construída nesse projeto foi desenvolvida utilizando o Android Studio versão 1.3.2 como ambiente desenvolvimento. A aplicação exige, como API mínima, a versão 15 do Android, a qual, de acordo com os dados apresentados no capítulo 4, é compatível com 95,1% dos dispositivos que utilizam o sistema atualmente.

Construiu-se um diagrama de classes referente à aplicação. O modelo tem como objetivo retratar a aplicação a um nível de implementação, onde as classes representam, de fato, as classes Java implementadas na sua elaboração. Apesar de o diagrama ser considerado como parte da modelagem, por apresentar os detalhes de implementação ele será utilizado para descrever como a mesma foi realizada.

A figura 23 exibe o diagrama de classes em questão, o qual contém algumas notas que descrevem brevemente as funções das classes. Os tópicos a seguir têm como objetivo explicar os principais pontos relacionados com o desenvolvimento da aplicação.

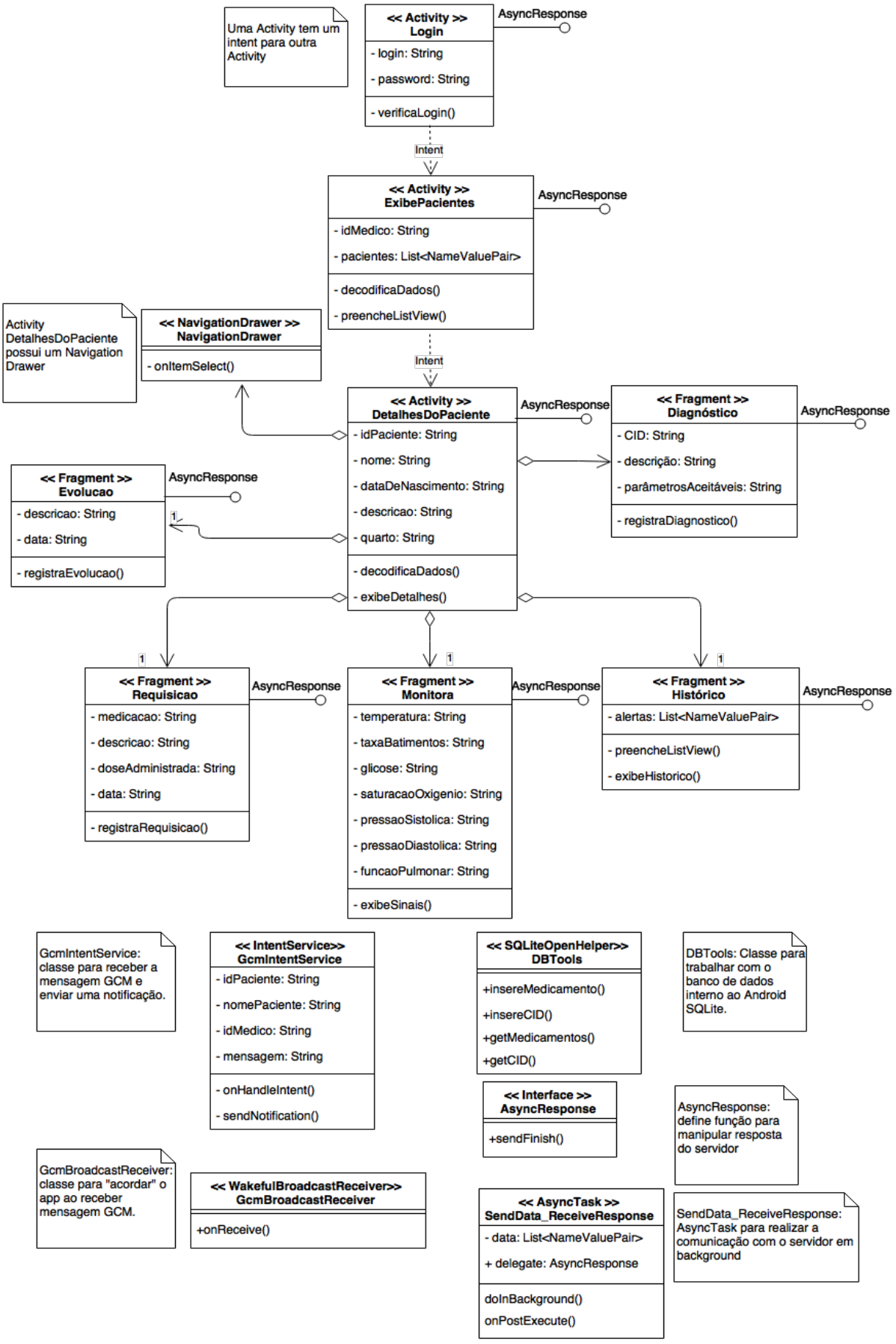


Figura 23 - Diagrama de classes da aplicação Android.

6.3.1. Conexão HTTP

Para realizar a conexão HTTP no Android, utilizou-se a classe *HttpUrlConnection*, que corresponde ao cliente mais atual e eficiente para a tarefa, sendo o seu uso recomendado pelo próprio Android. Esse cliente possui suporte a HTTPS (*Hypertext Transfer Protocol Secure*), *upload* e *download*, *timeout* configurável e IPV6 (ANDROID DEVELOPERS, 2015).

Antes de tentar realizar uma comunicação com o servidor é verificado se uma conexão com a internet está disponível no momento. Esse passo é de grande importância no tratamento de erros, visto que, caso uma conexão não esteja disponível, o usuário é avisado do problema para que o mesmo possa ser corrigido. Para realizar essa etapa, adota-se a classe *ConnectivityManager*, que possui métodos para verificação do estado da conexão.

As operações realizadas através da internet podem conter atrasos imprevistos, posto que a obtenção de uma resposta depende da velocidade da conexão, tempo de resposta do servidor e quantidade de dados transferidos. Portanto, com o propósito de evitar que isso ocasione problemas de desempenho, as operações relacionadas a internet não devem ser realizadas na *thread* principal. Para isso, emprega-se a classe *AsyncTask* do Android. Através dela, as tarefas de transferência de dados pela internet são disparadas a partir da *thread* principal e executam em background, sem comprometer o desempenho da aplicação.

No diagrama de classes da figura 23, a classe Java que fará a conexão é a *SendData_ReceiveResponse*, que recebe como parâmetro os dados a serem enviados para o servidor. Assim, as classes que necessitam se comunicar com o servidor irão criar uma instância dessa classe e executá-la. Os principais passos para realizar a conexão HTTP para a aplicação se comunicar com o servidor são listados a seguir.

1. Criar uma URL com o endereço que deve ser acessado.
2. Obter uma nova *HttpURLConnection* chamando o método *URL.openConnection*().
3. Definir as configurações de conexão. No caso, utilizou-se o método POST e *timeouts* de 2 segundos.
4. Obter o *output stream* da conexão e escrever nele os dados a serem enviados ao servidor, que consistem em pares de nome/valor.
5. Obter o *input stream* da conexão e realizar a leitura da resposta.
6. Desconectar.

O servidor retornará a resposta em JSON. Cada classe que se comunica com o servidor espera receber uma resposta específica, equivalente a requisição efetuada. Portanto, a decodificação da resposta será diferente para cada classe. Para isso, criou-se a interface *AsyncResponse* que define o método *sendFinish*, o qual irá executar ao final da comunicação com o servidor. Cada uma das classes que se comunicam com o servidor, então, irá sobrescrever esse método, a fim de tratar a resposta recebida da forma apropriada e exibi-la para o usuário. No decorrer do capítulo será discutido como ocorre a comunicação entre a aplicação Android e o servidor.

6.3.2. Log in / Log out

Com o intuito de assegurar que somente usuários cadastrados no sistema tenham acesso a aplicação, implementou-se um mecanismo de log in e log out. Ao iniciar a execução da aplicação pela primeira vez, a tela de log in é apresentada ao usuário, conforme demonstrado na figura 24. O usuário irá inserir os seus dados e, ao confirmar a operação, essas informações serão enviadas ao servidor para a verificação. Caso os dados fornecidos estiverem corretos, alguns passos relevantes para o funcionamento da aplicação são realizados e, após, a aplicação continuará o seu fluxo normal.

Primeiramente, a identificação do médico que realizou log in é salva nas preferências da aplicação, utilizando a classe *SharedPreferences*. Essa classe permite armazenar e recuperar dados de uma aplicação na forma de pares de chave/valor, sendo que tais dados são mantidos além da sessão da aplicação (isto é, os dados são mantidos mesmo se a aplicação for destruída) (ANDROID DEVELOPERS, 2015). Quando o usuário realiza log out da aplicação, as informações salvas no *SharedPreferences* são apagadas. Assim, ao executar a aplicação, primeiro verifica-se se há algum dado salvo nas preferências. Caso existam dados salvos, significa que o usuário realizou log in na aplicação e ainda não encerrou sua sessão. Então, ao abrir a aplicação novamente não será requisitada a autenticação e o uso da aplicação será baseado nas informações salvas nas preferências. A autenticação só será requisitada novamente após a realização de um log out.

Ainda durante o processo de log in, o dispositivo é registrado no GCM, obtendo um identificador para o mesmo. Esse identificador é enviado ao servidor para ser armazenado no

banco de dados externo, de modo que posteriormente será possível o envio de notificações para o dispositivo em questão.

Por último, é realizada uma requisição ao servidor para obter todos os medicamentos e doenças (tabela CID) do banco de dados externo. Ao obter essas informações, as mesmas são salvas no banco de dados SQLite interno ao Android. No decorrer desse capítulo, a implementação dos processos aqui enunciados será discutida com mais detalhes.

Realizados esses passos, a aplicação poderá continuar o seu fluxo normal. A segunda *activity* a ser iniciada, de acordo com a hierarquia de *activities* da aplicação, é a que exibirá a lista de pacientes do médico que está autenticado (Figura 25).

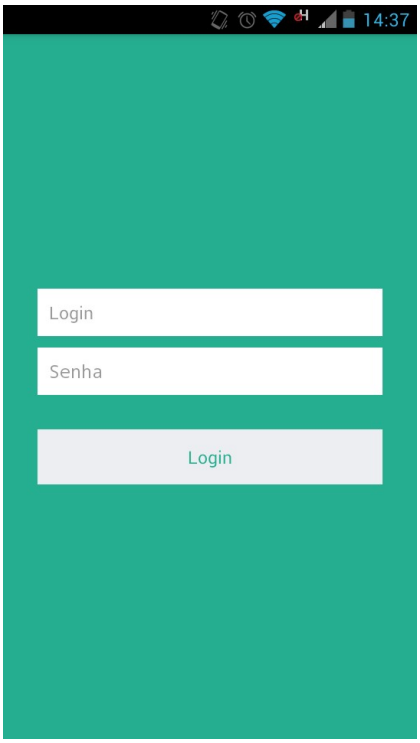


Figura 24 - Tela de log in da aplicação.

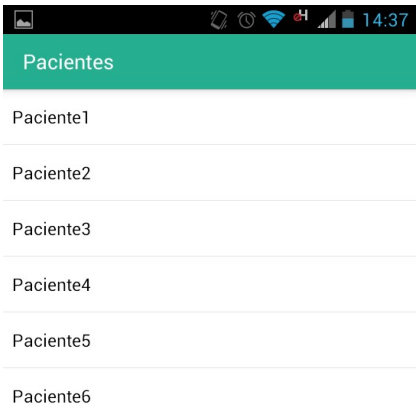


Figura 25 - Tela de lista de pacientes.

6.3.3. Fragments e Navigation Drawer

Uma *activity* é o ponto de partida para uma tela. Contudo, ao adicionar elementos gráficos e suas lógicas, a complexidade desse sistema cresce. Para modularizar o projeto, o Android 3.0 introduziu o conceito de *fragments*. Um *fragment* é uma classe que implementa

uma porção de uma *activity*. Ou seja, representa uma parte da interface gráfica em uma *activity* e seu comportamento.

Com o objetivo de modularizar a aplicação, os *fragments* são implementados como objetos independentes da *activity*. Cada *fragment* define seu próprio layout e comportamento com suas *callbacks* referentes ao seu ciclo de vida, sendo possível reutilizar um *fragment* em múltiplas *activities*. Apesar dessa independência, uma dada instância de um *fragment* é diretamente afetado pela *activity* que o contém, uma vez que uma *callback* do ciclo de vida de uma *activity* resultará em uma *callback* semelhante nos *fragments*. Com isso, é possível estabelecer uma comunicação entre a *activity* e o *fragment*, permitindo que um obtenha a referência para outro e associando seus eventos de *callbacks*. Esse mecanismo propicia o comportamento que consiste na principal vantagem da utilização de *fragments* no aplicativo desenvolvido: modificar a aparência da *activity* durante o tempo de execução. Assim, é possível adicionar, remover ou substituir um *fragment* em resposta a uma interação do usuário. É nesse ponto que a utilização de um *navigation drawer* entra em cena (ANDROID DEVELOPERS, 2015).

O *navigation drawer* é um painel que exibe as principais opções de navegação do aplicativo em uma lista. Na maioria do tempo, esse painel fica oculto. Quando o usuário arrasta o dedo a partir da borda esquerda da tela ou pressiona o botão de menu na barra de ações da *activity*, o *navigation drawer* é revelado. A figura 26 mostra o *navigation drawer* dentro do contexto da aplicação desenvolvida (ANDROID DEVELOPERS, 2015).

Conforme pode ser visto, o *navigation drawer* é implementado na *activity* do paciente (iniciada após a seleção de um paciente da lista) e exibe todas as funcionalidades do aplicativo. De acordo com a opção do menu que o usuário seleciona, o *fragment* dessa *activity* é substituído, alterando o seu conteúdo. Assim, a mesma *activity* é utilizada para todas as funções.

A utilização de um *navigation drawer* em combinação com *fragments* consiste em uma forma eficiente e intuitiva de projetar a navegação pelo aplicativo. Como a *activity* que implementa o *navigation drawer* é compartilhada pelos *fragments*, o *navigation drawer* estará sempre disponível, possibilitando que o usuário vá facilmente para a função que deseja. Caso fosse utilizada uma nova *activity* para cada uma das funções, cada uma dessas *activities* teria que implementar o seu próprio *navigation drawer*, tornando o código redundante e ineficiente.

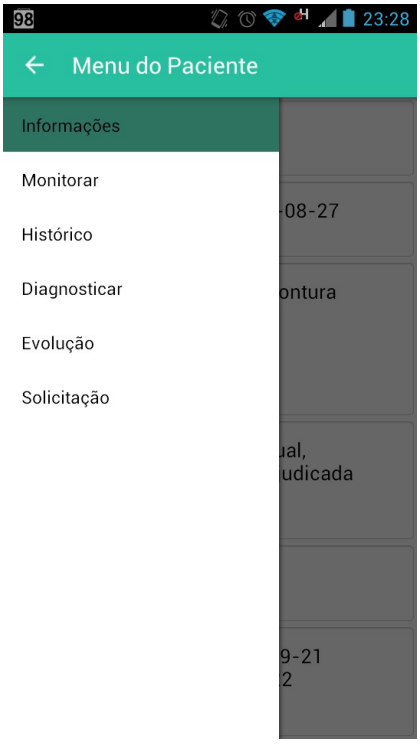


Figura 26 - *Navigation Drawer* com as funcionalidades da aplicação.

6.3.4. Banco de dados interno

Conforme pode ser visto no diagrama da figura 23, a aplicação desenvolvida utiliza o banco de dados SQLite interno ao Android. A maioria dos dados transferidos através da conexão com a internet consiste em poucas informações, relativas a apenas um dos pacientes. Por exemplo, quando o médico seleciona um de seus pacientes, a aplicação se comunicará com o servidor para obter apenas as informações básicas desse paciente, como: sintomas, data de nascimento, diagnóstico, data de admissão e quarto. Continuando o uso da aplicação, se o usuário selecionar a opção de exibir o histórico médico, somente os dados referentes ao histórico médico daquele paciente serão transferidos. Ou seja, para cada ação do usuário, a aplicação fará uma conexão HTTP para obter apenas os dados que essa ação requer – o que corresponde a apenas uma linha de uma tabela do banco de dados. Assim, a quantidade de dados transferida a cada conexão é pequena, assegurando que as informações serão rapidamente obtidas e exibidas para o usuário. Entretanto, há duas funcionalidades na aplicação que exigem que uma maior quantidade de dados seja transferida em uma conexão, são elas: diagnosticar um paciente e realizar uma solicitação médica.

A função de diagnosticar um paciente exibe uma lista com todas as doenças cadastradas no sistema (tabela CID do banco de dados) e o médico irá selecionar uma dessas opções para associar ao paciente. Da mesma maneira, a função de realizar uma solicitação exibe uma lista dos medicamentos cadastrados para que o médico possa selecionar (tabela de medicamentos do banco de dados). Essas duas tabelas, em um sistema real, podem possuir centenas/milhares de entradas e realizar a transferência desses dados em tempo de execução pode ocasionar um atraso no carregamento da lista que exibirá tais informações. Ademais, esses dados teriam que ser transferidos a cada vez que o médico selecionasse uma dessas opções, para todos os pacientes. Assim, uma considerável quantidade de dados teria que ser transferida toda vez que uma dessas operações fosse realizada pelos médicos.

Com o objetivo de corrigir esse problema, utilizou-se o banco de dados interno ao Android. No momento que o usuário realiza o log in na aplicação, os dados da tabela de doenças e de medicamentos do banco de dados externo MySQL são obtidos e armazenados no banco de dados interno. Com isso, durante a utilização da aplicação essas informações já estão disponíveis no aparelho, sem a necessidade de realizar uma conexão com a internet para obtê-los. Reduz-se, dessa forma, o tráfego de dados pela internet e, consequentemente, o consumo de bateria por parte da aplicação.

Para a implementação do banco de dados interno, fez-se uso da classe *SQLiteOpenHelper*. O *SQLiteOpenHelper* é uma classe projetada para ser estendida visando a implementação de tarefas relativas a criação, utilização e gerenciamento do banco de dados. Para manipular o banco de dados, utiliza-se a classe base *SQLiteDatabase*, a qual possui métodos para abrir, fazer buscas e fechar o banco de dados (ANDROID DEVELOPERS, 2015).

6.3.5. Funcionamento da aplicação

Nesta seção, será exposto como ocorre a utilização da aplicação do ponto de vista do usuário. Ao longo desta divisão, serão detalhadas as telas da aplicação e como o usuário deve proceder para realizar uma função.

Na implementação das funcionalidades, buscou-se simplificar ao máximo o modo com que as mesmas são exercidas. Assim, a aplicação exibe somente as informações relevantes a uma dada função. Para as funções que exigem a entrada de dados por parte do usuário, são

requeridos apenas os campos essenciais para a realização da atividade desejada. Dessa forma, a execução de uma função pode ser realizada de forma rápida, intuitiva e exigindo o mínimo de toques na tela.

6.3.5.1. Visualizar informações do paciente

Assim que o usuário seleciona um paciente na tela exibida na figura 25, a aplicação apresenta as informações do mesmo. Conforme pode ser visualizado na figura 27, os dados exibidos se resumem nas informações básicas: nome completo, data de nascimento, descrição do problema atual, diagnósticos associados, quarto em que está internado e data de admissão no hospital.

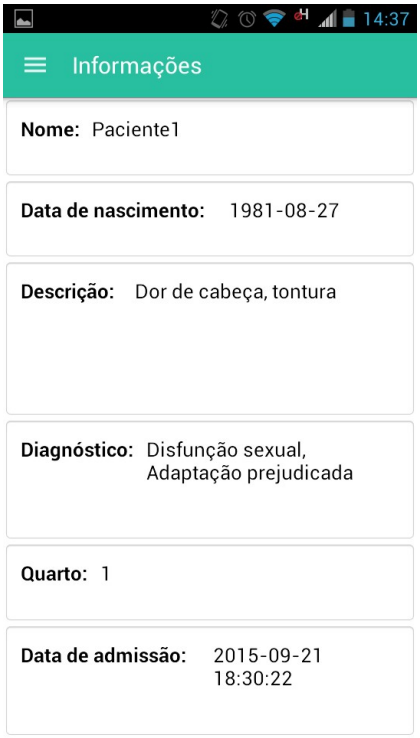


Figura 27 - Tela de informações do paciente.

6.3.5.2. Monitorar os sinais vitais de um paciente

A aplicação deve fornecer a função de monitorar os sinais vitais de um paciente. O objetivo desse serviço é possibilitar que o médico acompanhe a situação atual do paciente de forma remota. A implementação dessa funcionalidade é bastante simples: periodicamente, a aplicação realiza uma conexão com o servidor para obter os sinais mais atualizados de um paciente. No caso, o período de atualização utilizado é de 2 segundos, possibilitando um acompanhamento contínuo da situação de um paciente. O período de mensuração dos sinais por parte dos sensores geralmente é menor que esse valor, garantindo que a cada atualização se obtenha um dado novo. A figura 28 mostra a tela de monitoração dos sinais vitais.

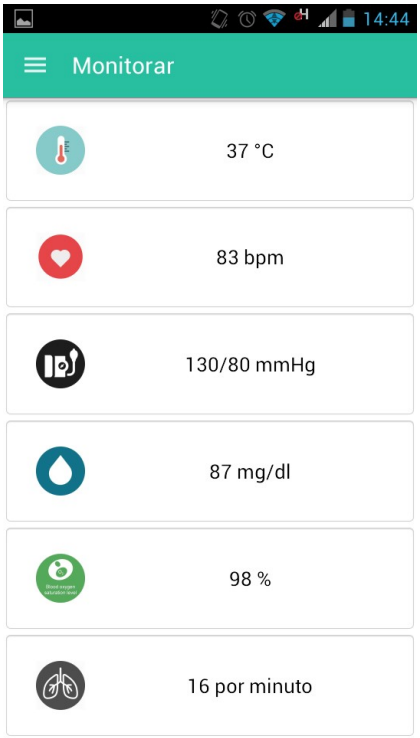


Figura 28 - Tela de monitoração dos sinais vitais.

Sabe-se que conexões periódicas com o servidor pode aumentar o consumo de bateria por parte da aplicação. Nesse caso, entretanto, esse processo é inevitável, dado que o objetivo é obter as informações com uma taxa constante de atualização. A conexão com o servidor para a atualização dos sinais é realizada apenas enquanto o usuário estiver dentro da função de

monitorar um paciente. Com isso, certifica-se que a operação só será realizada quando realmente for necessário, assegurando que a função não consista em um problema quanto ao consumo geral de bateria por parte da aplicação.

6.3.5.3. Visualizar o histórico médico

Ao selecionar a opção “Histórico” no menu lateral, a aplicação revela o histórico médico do paciente. As informações exibidas se referem ao uso de substâncias que podem causar danos à saúde, alergias e um campo de informações adicionais, que pode conter complicações de saúde enfrentadas pelo paciente anteriormente. A figura 29 mostra a tela de histórico médico de um paciente.

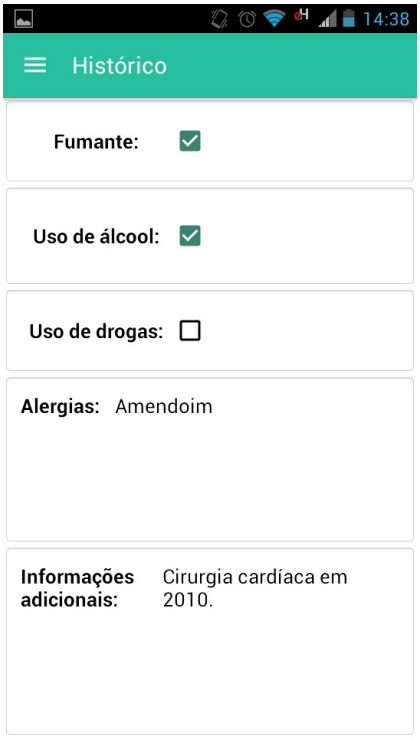


Figura 29 - Tela de histórico médico.

6.3.5.4. Diagnosticar

A aplicação permite que o médico registre um diagnóstico para um paciente. Ao selecionar essa opção no menu, a tela representada na figura 30 é exibida. O primeiro campo dessa tela é um *Spinner*, que consiste em um menu *dropdown*. Esse *spinner* é preenchido com os diagnósticos armazenados no banco de dados interno, em ordem alfabética, permitindo que o médico selecione uma opção. No segundo campo, o médico pode fornecer uma descrição textual informando detalhes do diagnóstico que está sendo registrado.

Figura 30 - Tela de diagnóstico.

6.3.5.5. Registrar evolução do quadro clínico

Para registrar uma evolução do quadro clínico do paciente o médico deve inserir uma descrição de como essa evolução ocorreu, a fim de manter um registro da evolução do paciente durante sua internação. A figura 31 mostra a tela construída para o uso dessa função.

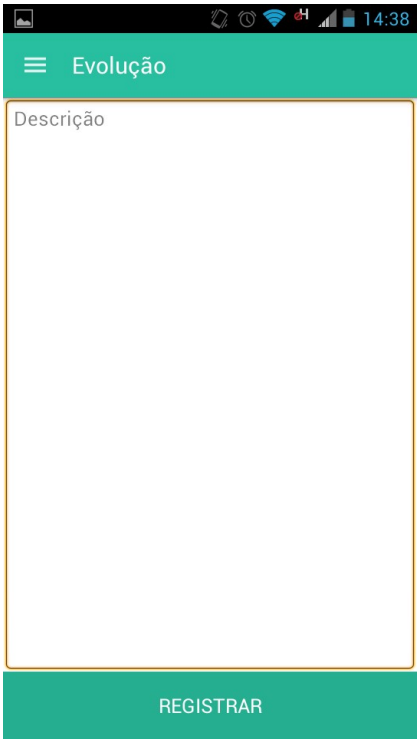


Figura 31 - Tela de registro de evolução.

6.3.5.6. Realizar uma solicitação

A aplicação possui uma função para que o médico realize uma solicitação à equipe de enfermagem que está no hospital. O primeiro campo é composto por um *Spinner,* que funciona de forma análoga ao da tela de diagnóstico explicada anteriormente. Entretanto, agora ele será preenchido com os medicamentos armazenados no banco de dados interno, sendo que a seleção de um medicamento é opcional. Se o médico selecionar um dos medicamentos listados, surge um segundo campo para inserção da dosagem a ser utilizada. Por último, tem-se um campo de texto que possibilita a inserção de uma descrição detalhando a solicitação que está sendo realizada. Com isso, essa funcionalidade permite que o médico requisite um procedimento a equipe de enfermagem, o qual pode ou não conter um medicamento envolvido – pode ser a requisição de um exame ou o início de administração de um medicamento, por exemplo. A figura 32 exibe a tela para realizar uma solicitação.

Figura 32 - Tela de solicitação.

6.4. Comunicação Android-servidor

Para implementar a comunicação entre a aplicação Android e o servidor, utilizou-se o protocolo HTTP, por meio do método POST. Essa escolha se deu devido ao acréscimo de segurança promovido por esse método, visto que as informações não estarão visíveis na URL (enviadas no corpo da mensagem) e não podem ser mantidas no histórico do navegador, por exemplo. Outra razão para essa escolha consiste no fato de que, ao contrário do método GET, que limita os dados em 2048 caracteres (tamanho máximo de uma URL), o método POST não possui restrições quanto ao tamanho dos dados a serem enviados (ADAMS, 2001).

A fim de estabelecer uma comunicação confiável, de modo que os dados possam ser enviados e recebidos de acordo com o esperado, e com capacidade de trazer informações extras em caso de erros, padronizou-se o formato das mensagens. A Figura 33 apresenta as etapas principais da comunicação entre a aplicação Android e o servidor.

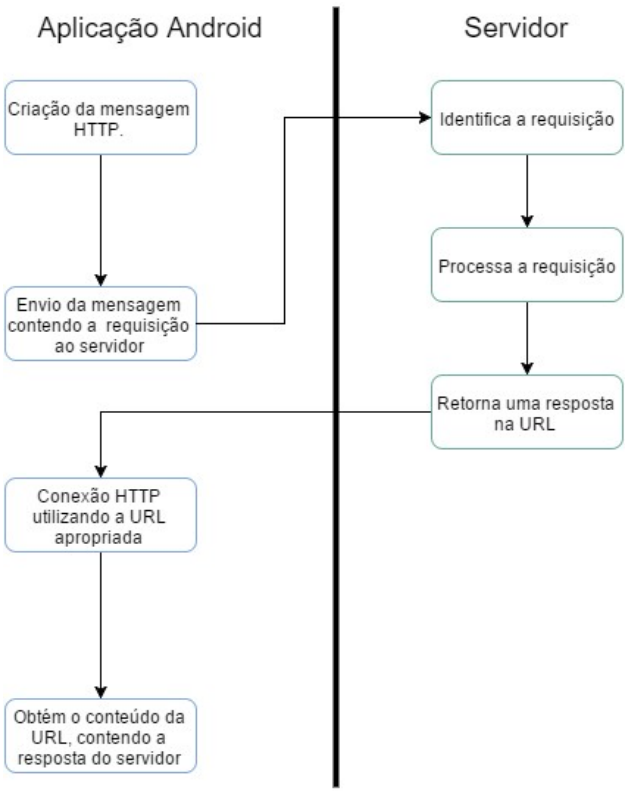


Figura 33 - Etapas da comunicação Android-servidor.

A aplicação Android, ao realizar uma requisição, envia um parâmetro nomeado “tag”, o qual possui como valor a identificação da requisição que está sendo tratada. Além desse dado, também são enviadas as demais informações necessárias para que a requisição seja processada pelo servidor. Para ilustrar esse recurso, abaixo estão os dados que são enviados para uma requisição de login, assim como seu valor:

Tag = “login”.

Login = login digitado pelo usuário.

Senha = senha digitada pelo usuário.

Todas as solicitações realizadas são enviadas para um único arquivo do servidor (index.php). Ao receber uma requisição, então, a primeira tarefa do servidor é obter o valor do campo nomeado como “tag”. Assim, o servidor identificará a requisição que está sendo recebida. Continuando com o exemplo de uma requisição de log in, o servidor verificará que o conteúdo do campo “tag” equivale a “login”. Assim, sabe-se que a tarefa que o mesmo deve exercer é obter as credenciais que estão sendo recebidas (login e senha) e realizar a verificação com as informações salvas no banco de dados para o usuário.

Após processar a requisição recebida, a resposta será retornada em uma URL, cujo o endereço equivale ao arquivo do servidor que receberá as requisições (isto é:

http://.../.../index.php). A resposta é escrita no corpo da página, em PHP e utilizando codificação JSON, de modo que a aplicação possa lê-la e obter os dados no formato de uma *string*.

O JSON consiste em uma forma de estruturar os dados para realizar a transmissão entre um servidor e aplicações web. Apesar de ser um subconjunto da notação de objeto JavaScript, trata-se de um formato completamente independente de linguagem, visto que utiliza convenções familiares a uma variedade muito grande das mesmas. A notação é baseada em uma coleção de pares nome/valor e em lista de valores, estruturas universais que todas as linguagens de programação modernas suportam, de uma forma ou de outra (TUTORIALS POINT). As figuras a seguir demonstram a notação JSON.

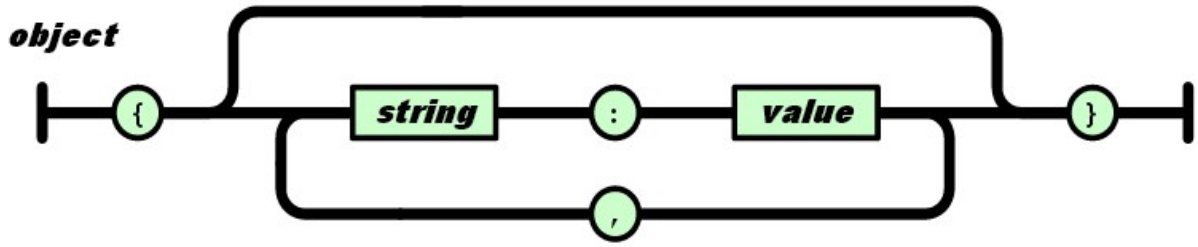


Figura 34 - Objeto JSON.

Fonte: ECMA INTERNATIONAL, 2013.

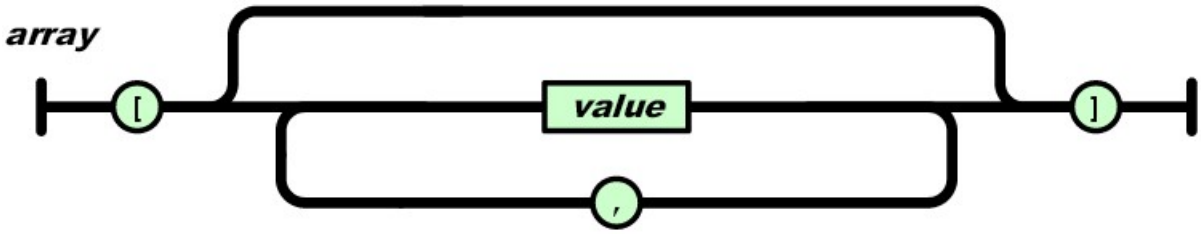


Figura 35 - *Array* JSON.

Fonte: ECMA INTERNATIONAL, 2013.

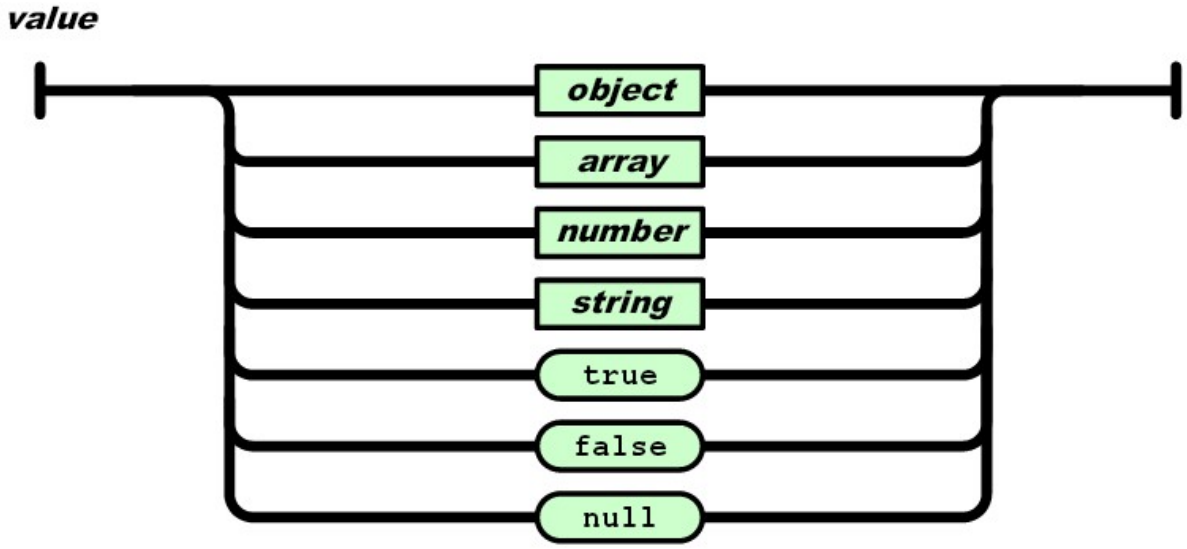


Figura 36 - Valor JSON.

Fonte: ECMA INTERNATIONAL, 2013.

De acordo com a notação JSON, um objeto é um conjunto desordenado de pares nome/valor. Um objeto é apresentado entre chaves, seus elementos são separados por vírgula e os pares de nome/valor são separados por dois pontos (nome:valor), conforme a figura 34. Um *array* consiste em uma coleção ordenada de valores, sendo apresentado entre colchetes e tendo seus valores separados por uma vírgula (Figura 35). Um valor JSON pode assumir um objeto, *array*, número, *string*, *boolean* ou *null* (Figura 36) (ECMA INTERNATIONAL, 2013).

Além de ser independente da linguagem, outra vantagem do JSON é a facilidade para trabalhar com o mesmo. O formato, baseado em texto, é leve e a codificação e decodificação de dados é simples de ser implementada. O PHP e o Java, linguagens utilizadas na implementação do lado servidor e cliente, respectivamente, possuem funções que permitem a manipulação de dados em JSON.

Da mesma forma que o envio de dados para o servidor, a resposta do servidor para a aplicação Android também foi padronizada. A resposta, em JSON, contém o rótulo que identifica a requisição, um campo indicando a ocorrência de erro que pode assumir verdadeiro ou falso como valor e, em caso de o erro ser falso, os dados processados pelo servidor (isto é, a resposta, de fato). A seguir é demonstrada a resposta do servidor para uma requisição de log in.

```
{"tag":"login","erro":false,"user":{"idUsuario":"2"}}
```

Para a requisição de log in mostrada acima, a resposta do servidor consiste na identificação do usuário que realizou a autenticação. Se as informações que o usuário inseriu não estiverem corretas, será indicado um erro. Na ocorrência de um erro, o último campo será o detalhamento do erro encontrado. Abaixo é exibida a resposta do servidor para o caso das informações de log in estarem incorretas:

```
{"tag":"login","erro":true,"erro_msg":"Login ou senha incorretos!"}
```

Outras causas de erros que podem ser indicadas são: requisição fora do padrão utilizado, informações de requisição incompletas, realização de uma requisição desconhecida pelo servidor e erro na operação com o banco de dados. A seguir é apresentada a resposta para um erro na comunicação com o servidor, onde o campo “tag” enviado na requisição não pôde ser identificado.

```
{"tag":"login2","erro":true,"erro_msg":"'tag' desconhecida"}
```

As demais requisições funcionam de maneira análoga. Essa padronização tanto da requisição quanto da resposta permite o tratamento de erros relacionados a comunicação. No momento do recebimento da resposta do servidor pelo Android, a primeira coisa a ser verificada é a ocorrência de um erro. Se o erro for verdadeiro, a descrição do mesmo é exibida ao usuário por meio de um *toast*. Caso contrário, dá-se prosseguimento à decodificação da resposta e sua utilização por parte da aplicação.

6.5. Notificações

Conforme citado anteriormente, uma das funcionalidades do sistema é o recebimento de notificações quando um dos sinais vitais do paciente apresentar uma anomalia e também quando a enfermeira registra uma resposta para uma solicitação realizada pelo médico. Para implementar o sistema de notificação foi utilizado o Google Cloud Messaging.

O GCM é um serviço gratuito da Google que permite o envio de mensagens entre servidores e aplicações clientes, com suporte a diversas plataformas. A troca de mensagens inclui tanto envios do servidor para o cliente quanto dos clientes para o servidor (GOOGLE DEVELOPERS, 2015). Neste projeto, enviam-se mensagens do servidor para a aplicação

quando uma notificação é exigida (notificação *push*). A figura 37 mostra a arquitetura do GCM.

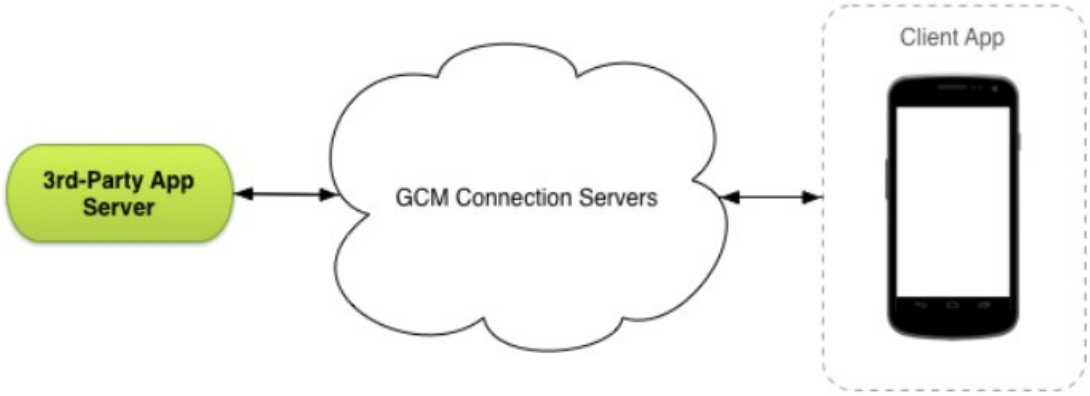


Figura 37 - Arquitetura do GCM.
Fonte: GOOGLE DEVELOPERS, 2015.

Quando o usuário faz log in na aplicação, o dispositivo que o mesmo está utilizando é registrado no GCM, obtendo um identificador para o mesmo. Esse identificador é salvo no banco de dados externo, tornando possível enviar mensagens diretamente para dispositivos individuais. Dessa maneira, quando o servidor reconhece que uma notificação é necessária, buscam-se os identificadores dos dispositivos dos médicos relacionados com essa notificação no banco de dados e o envio das mensagens é realizado.

Conforme pode ser observado na figura 37, a troca de mensagens entre o servidor e o cliente passa pelos servidores de conexão do GCM. Sob uma visão geral, os principais passos para o envio de uma mensagem do servidor para o cliente, considerando que o cliente já está registrado no GCM, são os seguintes:

1. O servidor envia uma mensagem para os servidores de conexão do GCM.
2. Os servidores de conexão do GCM enfileiram a mensagem e, caso o dispositivo esteja off-line, a armazena.
3. Quando o dispositivo estiver online, os servidores GCM enviam a mensagem para o dispositivo.
4. No dispositivo, a aplicação cliente recebe a mensagem.

O conteúdo das mensagens consiste no nome do paciente envolvido na situação, seu identificador no banco de dados e uma breve descrição do problema ocorrido. No caso de um

alerta por anomalia nos sinais, a descrição contém o sinal vital que apresentou um problema, assim como o valor mensurado para ele. Por exemplo, se uma temperatura de 39ºC for verificada para um paciente, será enviada uma mensagem com a seguinte descrição: “Temperatura: 39ºC”. No caso de uma notificação ocorrer devido a resposta de uma enfermeira para uma solicitação, o conteúdo da descrição será a própria resposta da enfermeira. Na notificação, também é exibido o nome do paciente envolvido. O envio do identificador do paciente permite que, se o médico clicar sobre a notificação recebida, o aplicativo abra as informações desse paciente. A figura 38 exibe o recebimento de uma notificação no dispositivo.

Conforme pode ser notado, o emprego do GCM envolve a implementação tanto no servidor do sistema quanto na aplicação Android. A seguir é descrito o processo de implementação para as duas partes.

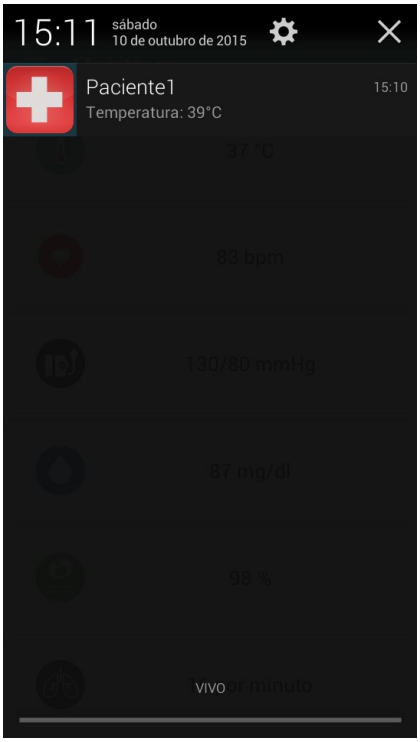


Figura 38 - Recebimento de uma notificação no dispositivo.

a. Implementação no Servidor

No servidor, após identificado que uma notificação se faz necessária, uma mensagem deve ser enviada para os servidores GCM com os dados que devem ser entregues aos dispositivos desejados. A mensagem destinada aos servidores GCM é formada por duas partes: um cabeçalho e o corpo da mensagem. No cabeçalho são inseridas as informações de autorização (chave da API obtida para utilizar o GCM) e o tipo de dado que está sendo enviado, no caso, JSON. O corpo da mensagem contém um *array* com a identificação de todos os dispositivos aos quais a mensagem deve ser enviada e outro *array* contendo o conteúdo da mensagem em si (descrição). A seguir são exibidos os passos necessários para a implementação no servidor.

1. O cabeçalho e o corpo da mensagem são criados contendo as informações necessárias.
2. O corpo da mensagem é codificado para JSON.
3. Utilizando a URL de *endpoint* do GCM (**https://android.googleapis.com/gcm/send**), a mensagem é enviada para os servidores GCM através do método POST.

b. Implementação no Android

Para receber as mensagens enviadas pelo servidor, a classe *WakefulBroadcastReceiver* do Android é utilizada. Basicamente, trata-se de um *BroadcastReceiver* que recebe um evento que "acorda" o dispositivo e passa a tarefa para um serviço. Essa classe é necessária para que as mensagens sejam recebidas mesmo quando o dispositivo não está ativo. No caso, a tarefa é repassada para um *IntentService*.

O *IntentService* tem a função de manipular requisições assíncronas, de acordo com a demanda. No contexto da aplicação desenvolvida, o *IntentService* será responsável por obter a mensagem GCM recebida e gerar uma notificação no dispositivo com essas informações. A notificação gerada possui um *intent* associado a ela que, ao clicar sobre a mesma, as informações do paciente são abertas na aplicação.

A principal vantagem que esse serviço fornece é a redução do consumo de bateria. A primeira abordagem utilizada para a geração de notificações consistia em realizar um *polling* com o servidor para, periodicamente, verificar se havia um dado novo no banco de dados que ocasionasse uma notificação. Essa abordagem é extremamente ineficiente em relação ao

consumo de bateria, uma vez que muitas vezes será realizada uma comunicação com o servidor sem que um dado novo seja encontrado no banco de dados. A utilização do GCM permite que o servidor informe a aplicação de que um dado novo (o qual gera uma notificação) foi encontrado, sem a necessidade de a aplicação iniciar uma ação. Assim, a comunicação entre servidor e aplicação só é efetuada, de fato, quando necessária.

Outra vantagem da utilização do GCM é a sua escalabilidade. O serviço funciona em integração com a conta da Google (GOOGLE DEVELOPERS, 2015). Dessa forma, o dispositivo mantém uma única conexão aberta com os servidores do GCM para aguardar por notificações de todas as aplicações no dispositivo. Conforme as mensagens chegam, elas são destinadas para as aplicações as quais são intencionadas. Essa aproximação é muito mais eficiente do que manter uma conexão com a internet aberta para cada aplicação.

6.6. Website

Utilizando o ambiente de desenvolvimento Netbeans e o *framework* Bootstrap, desenvolveu-se o website para este projeto. O Bootstrap é, atualmente, o *framework* mais popular para desenvolvimento *front-end* e contém modelos HTML, CSS e extensões JavaScript. Por meio de sua utilização, permite-se a implementação de um website responsivo, se adaptando a diferentes tamanhos de tela, garantindo uma boa experiência para o usuário independentemente do dispositivo que está sendo utilizado para acessá-lo. Além disso, o Bootstrap é suportado por todos os navegadores populares.

Ao iniciar a interação com o website, é exigido que o usuário realize a autenticação no sistema. A figura 39 mostra a página de log in desenvolvida.

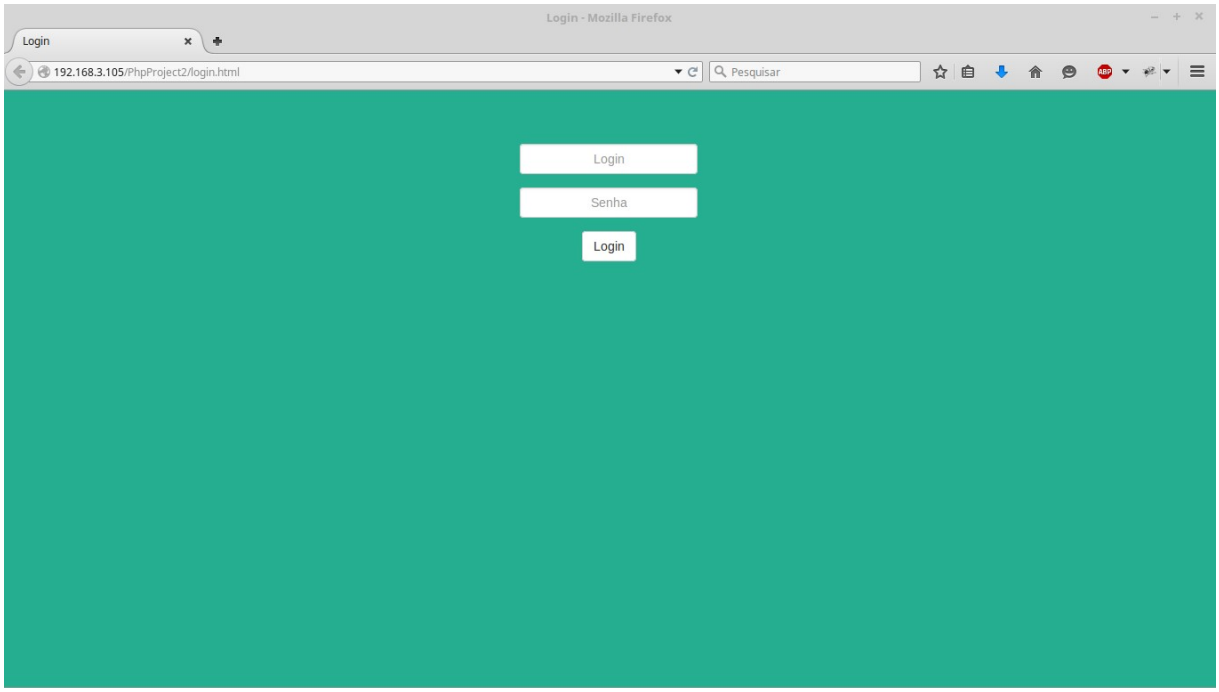


Figura 39 - Página de log in.

Após inserir os dados e confirmar a operação, os mesmos são enviados ao servidor, que irá recebê-los através do arquivo index.php para a sua verificação. Caso os dados estejam corretos, o servidor definirá um cookie contendo as informações do usuário que acabou de realizar o log in. Esse cookie será armazenado pelo navegador que o usuário está utilizando para acessar o website e servirá para verificar o estado da autenticação (isto é, se o usuário está autenticado ou não no sistema). Assim, caso um usuário não autenticado tente navegar por uma página, a mesma não será carregada e um aviso requisitando a autenticação é exibido.

O cookie contém ainda a informação do tipo de usuário que está utilizando o sistema. As páginas desenvolvidas são de acesso restrito a cada tipo de funcionário. Ou seja, uma recepcionista autenticada no sistema não pode acessar as funções de uma enfermeira, por exemplo. Para isso, antes do carregamento de uma página é verificado se o usuário que está autenticado pertence à classe de usuário autorizada a acessá-la.

Após a efetuação do log in, cada usuário é direcionado a página inicial da sua sessão. As figuras Figura 40, Figura 41 e Figura 42 mostram as páginas iniciais da sessão da recepcionista, enfermeira e administrador, respectivamente.

Como pode ser visualizado nas três figuras, os dados obtidos do banco de dados são apresentados ao usuário na forma de tabelas. Cada linha da tabela possui um campo de ações,

onde se encontram botões que permitem realizar uma função para aquele determinado registro. Dessa forma, minimizam-se as ações necessárias para que uma função seja executada. Além disso, as tabelas são editáveis. Assim, basta clicar sobre uma célula para editá-la e, em seguida, confirmar a operação para que os dados sejam salvos no banco de dados.

A fim de tornar o uso mais intuitivo, as sessões possuem abas que visam separar as funções por categorias. Como pode ser notado na figura 42, a sessão do administrador possui três abas que separam as funcionalidades que podem ser tomadas para os usuários, medicamentos e diagnósticos cadastrados no sistema. A mesma consideração é adotada na sessão da enfermeira, categorizando as funções para as solicitações e para registro de uma evolução clínica de um paciente.

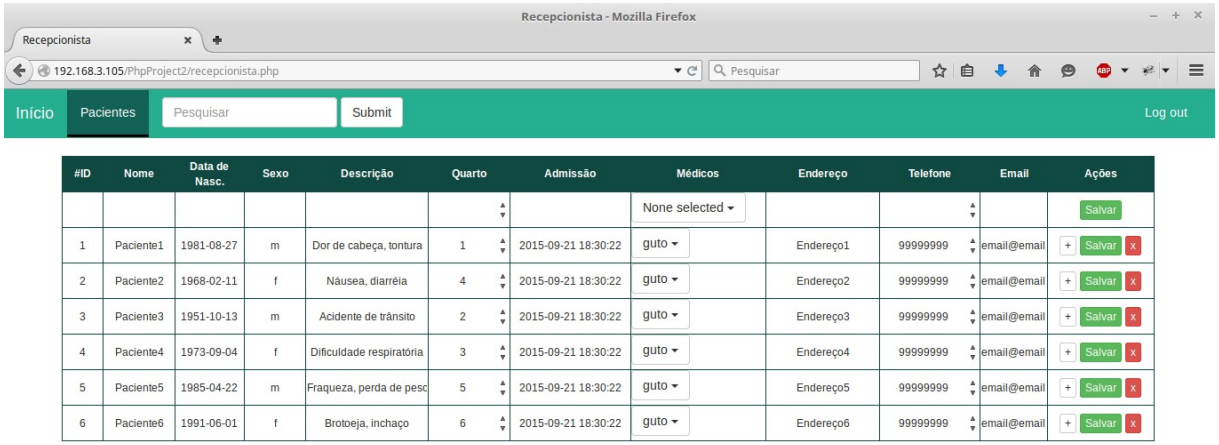


Figura 40 - Página da recepcionista.

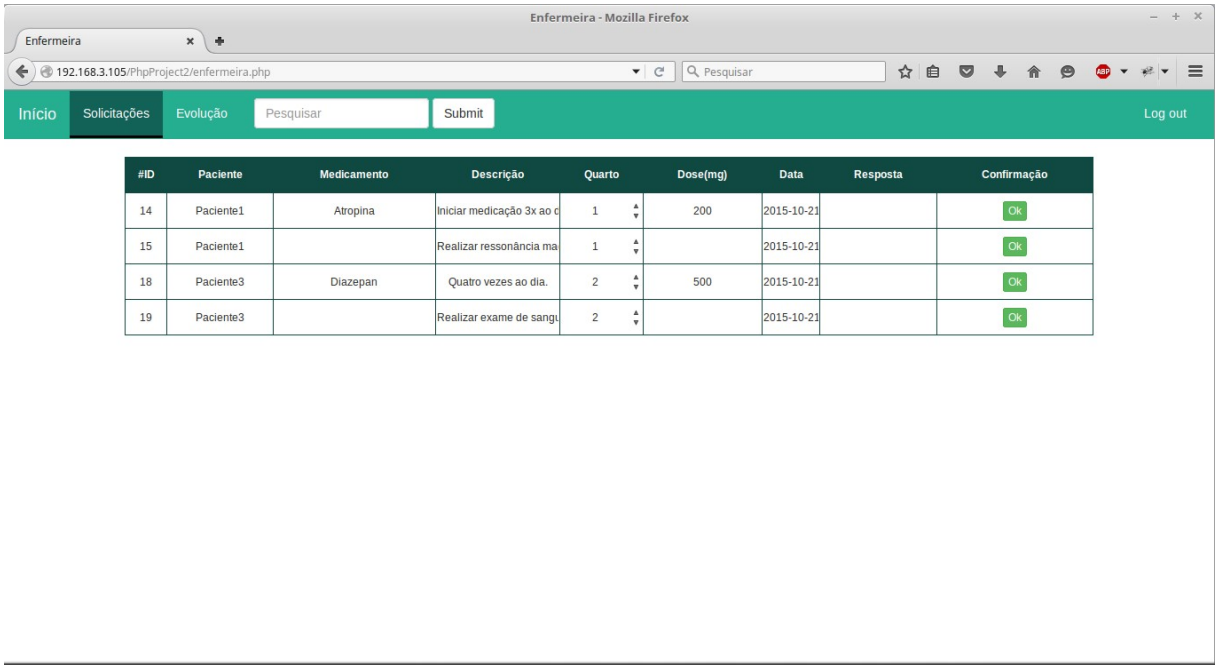


Figura 41 - Página inicial da enfermeira.

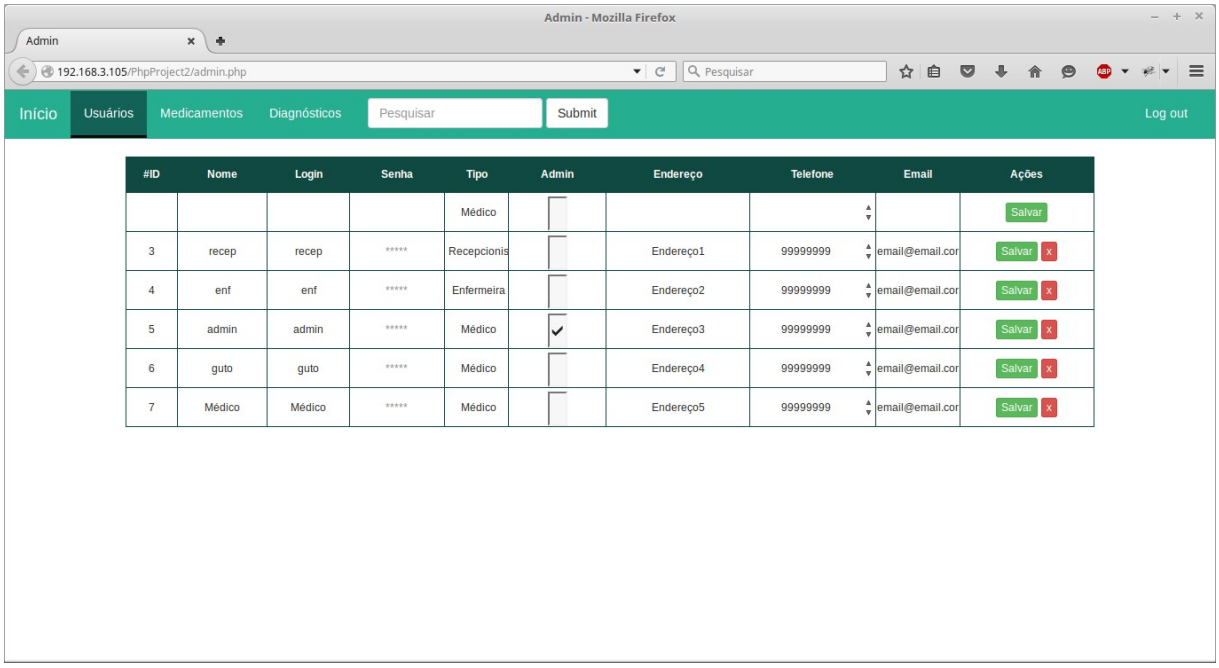


Figura 42 - Página inicial do administrador.

7. RESULTADOS

Neste capítulo será tratada a etapa de testes e análises do sistema projetado. Inicialmente, será descrito como o método de teste do sistema foi realizado e como esse processo auxiliou na garantia de funcionamento do mesmo. Ainda, dedica-se uma seção para análise de desempenho da aplicação Android, destacando-se as limitações presentes nos dispositivos móveis.

Esta fase do projeto tem como propósito garantir a estabilidade e o desempenho do sistema, o qual, devido sua aplicação médica, necessita uma alta confiabilidade. Assim sendo, ao realizar uma função, o usuário deve ter a certeza de que a mesma tenha sido executada corretamente ou, caso isso não seja possível, o mesmo deve ser alertado do problema para que as devidas providências sejam tomadas.

Na questão de desempenho tem-se uma preocupação especial com os dispositivos móveis, os quais possuem limitações conhecidas. Por isso, essa seção visa averiguar tais comportamentos com o intuito de garantir que os requerimentos de desempenho sejam atendidos.

7.1. Testes

Para a verificação do funcionamento do sistema, inicialmente se fez uso do website projetado para popular o banco de dados. Nele inseriram-se dados hipotéticos para realizar o cadastro de pacientes e usuários do sistema. Então, através do website se testou todas as funcionalidades implementas para cada tipo de usuário. Após, deu-se início ao processo de teste da aplicação, onde as funções foram exercidas várias vezes, de forma a cobrir todas as interações possíveis.

Como este trabalho não engloba a parte de hardware do sistema, os sinais vitais foram inseridos manualmente. Para isso, criou-se uma página da web contendo um formulário onde podem ser inseridos os valores dos sinais vitais para um paciente. Dessa forma, simulou-se essa etapa e tais valores foram repassados ao servidor, o qual não faz suposições a respeito da

procedência desses valores – desde que seja feita uma requisição da forma adequada, enviando os sinais vitais de um paciente, o servidor tratará essa requisição da mesma maneira.

Nesta fase, empenhou-se em englobar todas as possibilidades do sistema de forma a verificar a sua operação. Durante o processo, alguns erros foram identificados e uma investigação mais específica sobre sua ocorrência foi realizada, possibilitando sua posterior correção. Com isso, garantiu-se o correto funcionamento do sistema.

7.2. Análises de desempenho da aplicação Android

Os dispositivos móveis estão em uma evolução exponencial, apresentando rápidos avanços em relação ao hardware, tal como poder de processamento, memória RAM e armazenamento. Devido a isso, os usuários não conseguem se manter com um dispositivo de última geração por muito tempo, os quais se tornam rapidamente defasados. Essa circunstância promove a existência de dispositivos em uso com uma grande variedade de configurações. Consequentemente, as aplicações devem se enquadrar a essa realidade, buscando alcançar o maior número de usuários possível.

Em contraste, a capacidade da bateria desses dispositivos não está manifestando a mesma evolução exponencial que as demais tecnologias. Tal fato faz da bateria a maior limitação da computação móvel atualmente. Nessa perspectiva, é importante fazer um uso consciente do tráfego de dados na internet, uma vez que a sua utilização está diretamente ligada ao consumo de energia (RAVI; SCOTT; *et al.*, 2008). Também serão realizadas as análises do consumo de bateria e tráfego de internet por parte da aplicação desenvolvida, visando garantir que o usuário possa fazer seu uso pelo maior tempo possível. Para a realização dos ensaios, foi utilizado o smartphone Moto G, da Motorola, executando a versão 5.1 do Android, onde o ART é a *runtime* padrão e cuja capacidade da bateria é de 2070 mAh. Com a finalidade de reproduzir o seu emprego real, buscou-se um uso intenso da aplicação durante o período de análises.

7.2.1. Memória RAM

A principal questão se tratando de memória RAM é verificar que a aplicação faz um uso razoável da mesma. Portanto, o primeiro passo desta análise compreendeu em investigar a ocorrência de vazamentos de memória. O Android Studio possui ferramentas que permitem visualizar o uso de memória durante a depuração da aplicação. Através de testes, então, apurou-se que o uso de memória se manteve estável, permitindo inferir que não há problemas de vazamento.

O Android 4.4 KitKat introduziu um serviço chamado estatísticas de processos. Por meio desse serviço, é possível analisar o uso de memória RAM das aplicações sendo executadas ao longo do tempo. O recurso traz ainda detalhes acerca da execução das aplicações, tais como tempo total que a aplicação foi executada e consumo médio e máximo de memória. Por conseguinte, torna-se possível compreender o comportamento da aplicação e identificar possíveis falhas de desempenho (ANDROID DEVELOPERS, 2015). A figura 43 representa a visão geral do consumo de memória retornada por essa ferramenta.

Figura 43 - Visão geral da memória utilizada pelos processos em primeiro plano.

Figura 44 - Detalhes do uso de memória por parte da aplicação desenvolvida.

Conforme indicado na tela, os dados coletados se referem as aplicações executadas em primeiro plano nas últimas 3 horas e 46 minutos. Durante o período, a memória do dispositivo esteve em boa forma, em concordância com a barra verde que é exibida. Se a memória livre do dispositivo estivesse baixa, haveriam seções em amarelo e vermelho indicando tal comportamento. Também, pode-se notar uma lista dos processos sendo executados. A barra azul aponta a carga de memória relativa de cada processo, que consiste no tempo de execução multiplicado pelo uso médio de memória. A figura 44 mostra a tela que contém os detalhes para a aplicação desenvolvida, denominada TCC_App_v1.

De acordo com a figura 44, a aplicação foi executada durante 25% do período em que a análise foi realizada, o equivalente a 56 minutos e 30 segundos. Analisou-se a memória Uss (Unique set size) que corresponde ao total de memória privada de um processo, isto é, a memória que é completamente única a esse processo. Quando o processo for destruído, o Uss é a memória total que vai ser retornada ao sistema. Desta forma, é o valor que melhor representa o real custo de execução de um processo no dispositivo, permitindo avaliar seu desempenho (ANDROID DEVELOPERS, 2015).

Conforme os dados exibidos, o uso médio de RAM foi de 34 MB e o máximo, 41 MB. Para fins de comparação, verificou-se o uso de RAM no smartphone LG G2, executando o Android 4.4 e Dalvik. Como resultado, obteve-se um uso médio de 15MB. Conforme explicado no capítulo 4, essa constatação se justifica pelo fato da Dalvik compilar apenas a parte do código necessária para execução no momento. O ART, por sua vez, compila todo o código durante a instalação da aplicação, resultando em um maior uso de memória durante a execução. Investigou-se, ainda, o uso de memória quando a aplicação está em segundo plano, ou seja, quando o usuário não está interagindo com diretamente com ela, conforme a figura 45.

Segundo as informações exibidas, a aplicação utiliza 17MB quando em segundo plano. Ressalta-se, entretanto, que esse valor corresponde ao estado instantâneo da memória, sem o contexto no tempo. Ou seja, esse valor pode apresentar uma pequena variação de acordo com o momento da verificação. Apesar de o Android gerenciar a memória do dispositivo, destruindo processos em segundo plano quando houver necessidade de recuperar memória, em geral, visa-se minimizar esse parâmetro, evitando ocupar memória com conteúdo desnecessário.

Dado o tamanho da aplicação desenvolvida e a quantidade de recursos utilizados, os resultados anunciados nessa análise se enquadram dentro do esperado. Fazendo um paralelo, no mesmo dispositivo, o Facebook utilizou 102 MB em primeiro plano e o Whatsapp, 144

MB. Obviamente, são aplicações voltadas para fins totalmente diferentes. Entretanto, essa informação permite inferir que o consumo de memória RAM pela aplicação desenvolvida tem pouco impacto na memória total do dispositivo quando comparada com aplicações populares, que quase todos possuem instaladas em seus smartphones.

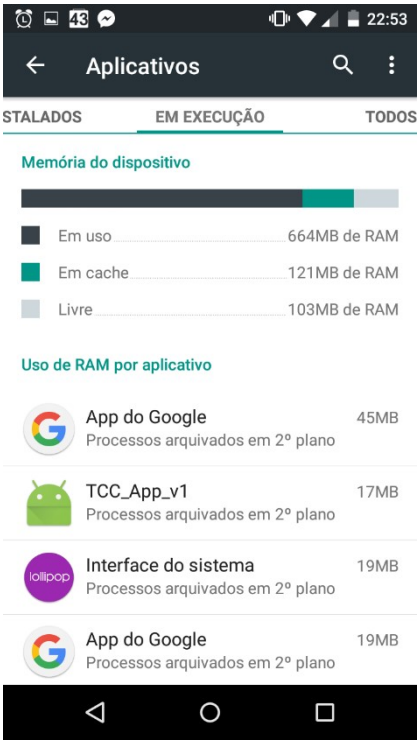


Figura 45 - Uso de memória da aplicação em segundo plano.

7.2.2. Tráfego de dados na internet

A quantidade de dados transferida através da internet está diretamente relacionada com o consumo de bateria e com o desempenho da aplicação. Com base nisso, visa-se transmitir apenas os dados necessários. Para analisar o comportamento da aplicação nesse quesito, obtiveram-se os dados transferidos para a realização das funções projetadas. A figura 46 exibe a quantidade de dados para uma requisição de obter as informações de um paciente, indicando também a velocidade dessa transferência.

De acordo com a figura, a quantidade de dados enviados e recebidos fica próxima a 1 KB para esse caso. Investigou-se esse comportamento para todas as funcionalidades suportadas e as mesmas apresentaram pouca variação, não ultrapassando 1500 bytes. A

pequena quantidade de dados transferida possibilita que a operação seja completada rapidamente. Conforme o gráfico da figura 46, a velocidade chega a 8 KB/s, levando aproximadamente 250ms para que a transferência completada, lembrando que as operações relacionadas a internet são executadas em background. Ou seja, enquanto a transferência de dados é realizada, a aplicação continua a sua execução normal, podendo haver o carregamento dos elementos de layout de forma paralela, por exemplo. Assim, o usuário não tem um atraso perceptível entre o gatilho de uma função e a exibição das informações na tela.

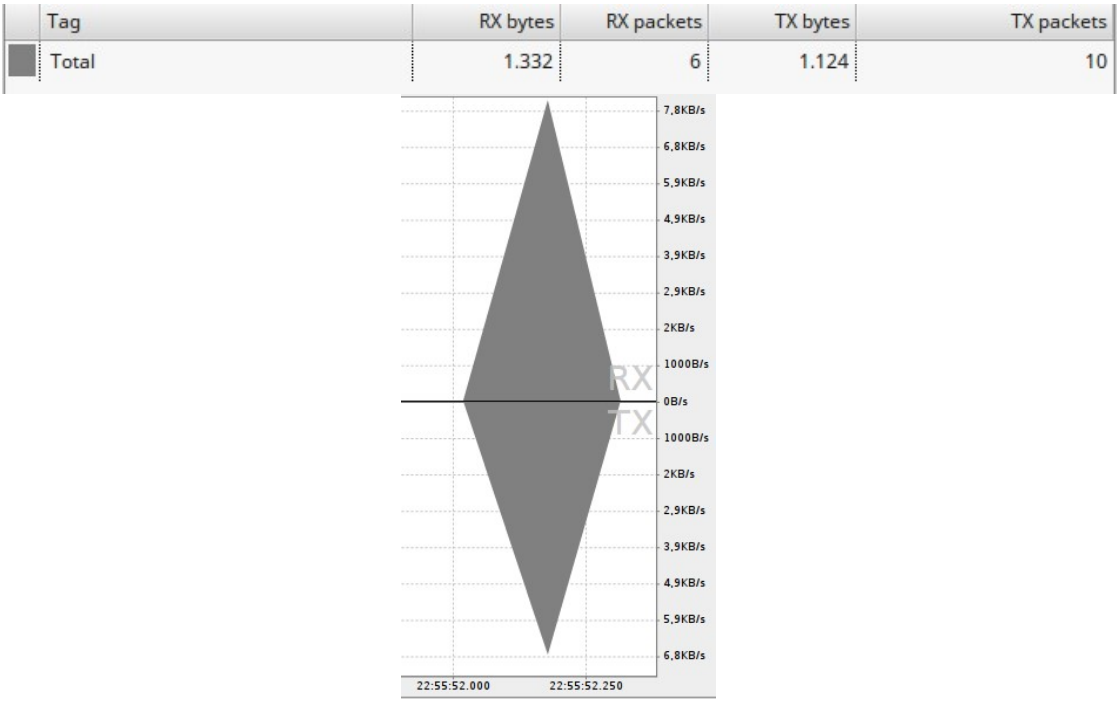


Figura 46 - Transferência de dados na internet para obter as informações de um paciente.

7.2.3. Bateria

Dentro do período em que foi analisada a utilização de memória, também se averiguou o consumo de bateria. Para tanto, utilizou-se a aplicação GSam Battery Monitor, a qual traz informações detalhadas a respeito do consumo de bateria no dispositivo.

A figura 47 apresenta a tela de visão geral do consumo de bateria, onde a inspeção coletou dados relativos a aproximadamente uma hora. Nota-se que 9% da bateria do dispositivo foi consumida nesse período, sendo que desses, 78% foi devido a tela, 16% devido

a execução das aplicações, 4% pelo sinal GSM e 2% pelo Wifi. Verifica-se assim que a tela é a grande responsável pela drenagem da bateria nesse caso – o que se estende também para a maioria das situações. Na figura 48, tem-se o consumo de bateria por parte das aplicações executadas no intervalo.

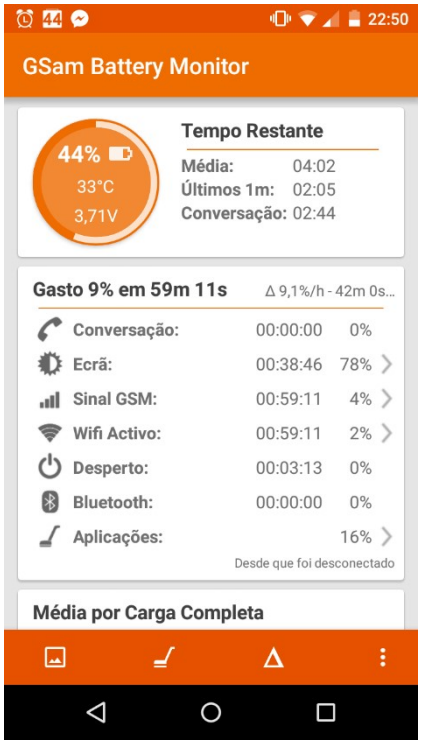


Figura 47 - Visão geral do consumo de bateria.

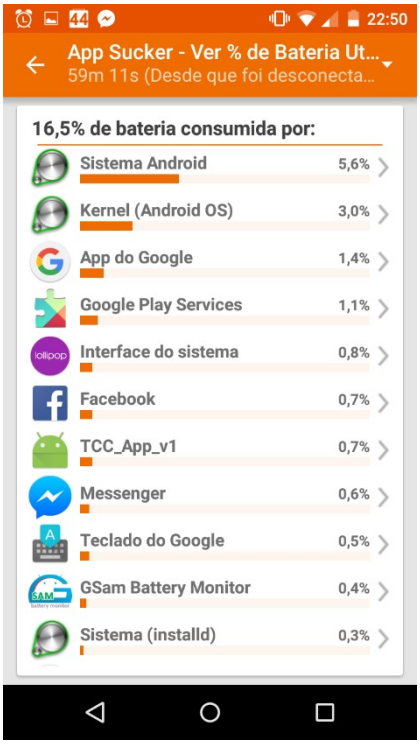


Figura 48 - Consumo de bateria por parte das aplicações executando no dispositivo.

Como pode ser observado na figura 48, a aplicação desenvolvida foi responsável por consumir 0,7% (14,49 mAh) da bateria do dispositivo, lembrando que, durante o período, foi a aplicação mais utilizada. Assim, detecta-se um consumo de bateria mais baixo do que o esperado, certificando que o usuário possa utilizá-la por um longo período sem grande impacto nesse quesito. Os detalhes referentes a essa aplicação podem ser conferidos na figura 49, tornando possível uma verificação mais profunda sobre como ocorreu o consumo de bateria.

No total, a aplicação utilizou o processador durante 33 segundos. Durante o período de testes, aproximadamente 327 KB de dados foram transferidos pela internet. Dado o uso intenso da aplicação durante o intervalo, trata-se de uma quantia moderada, a qual foi possível

devido a pequena quantidade de dados transferidos em uma requisição, de acordo com as informações explicitadas na seção anterior.

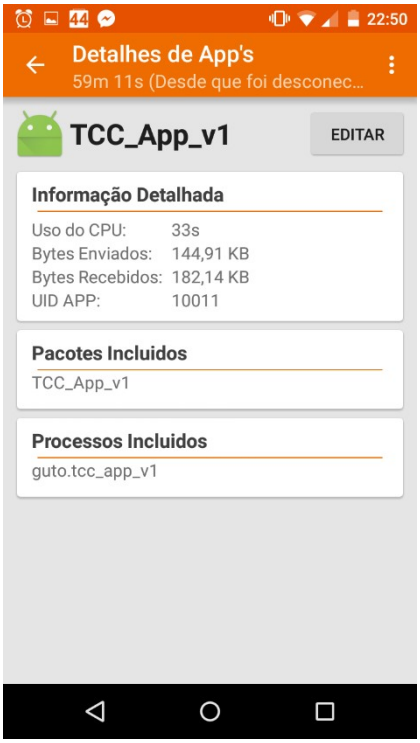


Figura 49 - Detalhes do consumo de bateria por parte da aplicação desenvolvida.

Na utilização de um plano de dados móveis, sobretudo, há uma preocupação com a quantidade de dados transferidos. Fazendo uma projeção mensal com base nos valores obtidos, com um uso médio de 8 horas ao dia, tem-se um consumo de, aproximadamente, 78,5 MB por mês. Essa quantidade de transferência se mostra eficiente nesse sentido, sendo suportada por um plano de dados básico.

8. CONCLUSÃO

Baseado nos objetivos introduzidos no início desse estudo é possível formular as contribuições do trabalho elaborado. Este projeto tem como objetivo o desenvolvimento de um sistema hospitalar que faz uso de dispositivos móveis para explorar as vantagens da mobilidade nesse ambiente. Essa característica possibilitou a acessibilidade às informações do sistema e o telemonitoramento de pacientes. Assim sendo, eliminam-se os contratempos devido a distância entre profissionais de saúde e pacientes, permitindo que o médico realize consultas sem a necessidade de estar presente no hospital.

O trabalho apresentado é fundamentado em três fases principais: modelagem, implementação e análise de resultados. O projeto proposto teve um planejamento claro a respeito dos objetivos que deveriam ser alcançados pelo sistema. Na etapa de implementação, projetou-se as funcionalidades levantadas, com atenção aos requerimentos não funcionais estabelecidos.

No capítulo de resultados, apresentou-se constatações sobre a estabilidade e desempenho do sistema, levando em consideração as limitações existentes nos dispositivos móveis atualmente. Por meio de testes variados, verificou-se o correto funcionamento de todas as funções projetadas. A análise de desempenho da aplicação desenvolvida apresentou resultados bastante satisfatórios. Durante aproximadamente uma hora a aplicação foi testada sob uso intenso, e a mesma foi responsável por consumir apenas 0,7% da bateria do dispositivo (equivalente a 14,49 mAh). Essa constatação garante que a aplicação possa ser utilizada durante longos períodos sem comprometer a bateria do aparelho.

O tráfego de dados na internet tem impacto direto no consumo de bateria e, no caso de plano de dados móveis, existe uma preocupação com a quantidade de dados transferidos. Em vista disso, empenhou-se em reduzir a transferência de dados através da internet. Realizando uma projeção mensal com base nos dados apurados nos ensaios, obteve-se um consumo de 78,5 MB por mês, propiciando o suporte por um plano de dados básico. Tal valor foi auferido devido a pequena quantidade de dados requeridos para a realização de uma função: em média, 1 KB de dados enviados e recebidos. Assim, garante-se que as operações na internet sejam completadas rapidamente, sem a ocorrência de um atraso perceptível entre o gatilho de uma função e a exibição das informações na tela.

Investigou-se também o uso de memória RAM por parte da aplicação. Através de testes, comprovou-se que não há vazamentos de memória. O uso médio de memória, quando a aplicação se encontra em primeiro plano, é de 34 MB. Já em segundo plano, isto é, quando o usuário não está interagindo diretamente com a aplicação, esse valor cai para 17 MB. Tratam-se de valores condizentes com o esperado, dado o tamanho da aplicação desenvolvida, e que, comparado a aplicações populares, tem baixo impacto no uso de memória. Assim, sua utilização pode ser realizada sem problemas em dispositivos mais antigos, com menor capacidade de memória. Com isso, as limitações quanto ao consumo de bateria, memória e tráfego de dados na internet foram atendidas, garantindo que a utilização da aplicação possa ser realizada durante longos períodos, com baixo impacto em tais quesitos.

As aplicações para interação dos usuários com o sistema foram desenvolvidas visando a simplicidade e intuitividade na realização das funções. Dessa maneira, os profissionais que farão seu uso poderão atingir seus objetivos da forma mais ágil possível. Portanto, como resultado final, obteve-se um sistema hospitalar robusto, com capacidade para atender as necessidades de seus usuários e que vai de encontro aos requisitos estabelecidos para o mesmo. Obteve-se, também, por intermédio da etapa de modelagem, uma documentação completa a respeito do projeto. Na seção seguinte, serão discutidos os próximos passos em relação ao trabalho.

8.1. Trabalhos Futuros

Este trabalho oportuniza uma variedade de linhas de desenvolvimento que permitem dar continuidade ao mesmo. Os sistemas de software, em especial, possuem fases de maturação que propiciam o seu aprimoramento com o passar do tempo. Neste projeto, foram englobadas as principais funções no que diz respeito a um ambiente hospitalar. No entanto, há espaço para aperfeiçoar as funcionalidades existentes e também acrescentar novas ao sistema. Como sugestão, cita-se a possibilidade da visualização de gráficos que retratam os sinais vitais de um paciente no tempo. Além disso, é possível englobar novos aspectos ao projeto, como por exemplo, dar suporte a parte de administração financeira do hospital.

Em uma aplicação real, é necessário tomar todas as providências cabíveis para garantir a segurança do sistema. Os dados transmitidos através do protocolo HTTP são enviados como texto simples, podendo ser lidos caso alguém consiga invadir a conexão. Nesse sentido, é

interessante a utilização do protocolo HTTPS, que consiste na implementação do protocolo HTTP sobre uma camada adicional de segurança SSL/TLS (ANTTALAINEN; JÄÄSKELÄINEN, 2014). Desse modo, emprega-se a autenticação entre as aplicações cliente e servidor e os dados são transmitidos sob uma conexão encriptada, evitando a sua obtenção por pessoas mal-intencionadas. Devido a necessidade de possuir um domínio para a configuração do HTTPS no servidor, essa tarefa acabou ficando fora do escopo do projeto. Em questão de implementação não haveria maiores dificuldades em utilizar esse mecanismo, dado que o Android possui suporte a esse protocolo da mesma forma que o HTTP, sendo necessário, no máximo, a definição de alguns parâmetros extras no momento do estabelecimento da conexão.

A parte do projeto mais abrangente em relação a trabalhos futuros é a verificação dos sinais vitais de um paciente. Nos últimos anos, diversas pesquisas têm buscado aprimorar os sistemas de monitoramento de pacientes por meio da utilização de técnicas de inteligência artificial, visando tornar o processo mais hábil. A ideia é que, com o uso de tais técnicas, seja possível interpretar os sinais monitorados para detectar anomalias nos resultados. Essa interpretação deve levar em conta as informações clínicas do paciente, obtendo resultados mais precisos e que podem envolver, inclusive, funções de tomada de decisão (SHIN; CHA; *et al.*, 2000). Com isso, pode-se detectar com antecedência condições anormais ou tendências no desenvolvimento de problemas médicos, auxiliando os profissionais de saúde e tornando o processo mais efetivo.

REFERÊNCIAS

ACKER, E. V.; WEBER, T. S.; CECHIN, S. L. Injeção de Falhas para Validar Aplicações em Ambientes Móveis. **XI Workshop de Testes e Tolerância a Falhas**, Porto Alegre, 2010.

ADAMS, D. **The 4D Web Companion**. [S.l.]: [s.n.], 2001.

ANDROID. The Android Source Code, 2015. Disponivel em: <https://source.android.com/source/index.html>. Acesso em: 21 Agosto 2015.

ANDROID. ART and Dalvik. **Android**. Disponivel em: <https://source.android.com/devices/tech/dalvik/ >. Acesso em: 20 Agosto 2015.

ANDROID DEVELOPERS. **Android Developers**, 2015. Disponivel em: <http://developer.android.com/intl/pt-br/develop/index.html>. Acesso em: 10 Outubro 2015.

ANTTALAINEN, T.; JÄÄSKELÄINEN, V. **Introduction to Communication Networks**. 1ª. ed. [S.l.]: Artech House, 2014.

ARON, L.; HANÁCEK, P. **Introduction to Android 5 Security**. Brno University of Technology. [S.l.]. 2015.

BELL, D. **The activity diagram**. IBM. [S.l.]. 2003.

BEZERRA, E. **Princípios de análise e projeto de sistemas com UML**. 3ª. ed. Rio de Janeiro: Elsevier, 2006.

BONEWIT-WEST, K.; HUNT, S. A.; APPLEGATE, E. **Today's Medical Assistant:** Clinical & Adminsitrative Procedures. Missouri: Elsevier Saunders, 2013.

BOOTSTRAP. **Bootstrap**, 2015. Disponivel em: <http://getbootstrap.com/ >. Acesso em: 22 Agosto 2015.

BORDIN, M. V. **Introdução a Arquitetura Android**. Três de Maio: [s.n.], 2012.

CHENG, B.; BUZBEE, B. **A JIT Compiler for Android's Dalvik VM**. Google. [S.l.]. 2010.

COCKBURN, A. **Writing effective use cases**. 1ª. ed. [S.l.]: Addison-Wesley, 2001.

COLARES, F. M. **Análise comparativa de banco de dados gratuitos**. Faculdade Lourenço Filho. [S.l.]. 2007.

DZUBOW, D.; DOMENIC, R.; SLOUGH, S. Analysis & Design of a Simplified Patient Care Clinic System. **Information Systems Analysis and Design**, 2011.

ECMA INTERNATIONAL. **The JSON data interchange format**. 1ª. ed. [S.l.]: [s.n.], 2013.

FALBO, R. D. A. **Engenharia de software: Notas de aula**. Universidade Federal do Espírito Santo. [S.l.]. 2005.

FIELDING, R. et al. **Hypertext Transfer Protocol -- HTTP/1.1**. [S.l.]. 1999.

FOWLER, M. **UML Distilled:** A brief guide to the standard object modeling language. 3ª. ed. Massachusetts: Addison-Wesley, 2003.

GARCIA-MOLINA, H.; ULLMAN, J.; WIDOM, J. **Database Systems:** The Complete Book. 2ª. ed. New Jersey: Prentice Hall, 2009.

GARGENTA, M. **Learning Android**. Sebastopol: O'Reilly, 2011.

GEREMIA, J. **Tutorial de introdução a Banco de Dados**. Universidade Federal Fluminense. [S.l.]. 2010.

GOOGLE DEVELOPERS. Cloud Messaging. **Google Developers**, 2015. Disponivel em: <https://developers.google.com/cloud-messaging/android/client>. Acesso em: 10 Outubro 2015.

GUDWIN, R. R. **Diagramas de Atividade e Diagramas de Estado**. [S.l.]. 2010.

GUDWIN, R. R. **Introdução à Linguagem UML**. UNICAMP. [S.l.]. 2010.

HULL, E.; JACKSON, K.; DICK, J. **Requirements Engineering**. 2ª. ed. Newtownabbey: Springer, 2005.

IDC. Smartphone OS Market Share, Q1 2015. **IDC**, 2015. Disponivel em: <http://www.idc.com/prodserv/smartphone-os-market-share.jsp>. Acesso em: 9 Agosto 2015.

INDUMATHY, N.; PATIL, K. K. Medical alert system for remote health monitoring using sensors and cloud computing. **International Journal of Research in Engineering and Technology**, 2014.

KAUTZMANN, T. **Uma aplicação móvel de acesso ao prontuário médico**. Universidade Regional do Noroeste do Estado do Rio Grande do Sul. [S.l.]. 2012.

KOSHMAK, G. **An Android based monitoring and alarm system for patients with chronic obstructive disease**. Department of Technology at Örebro University. Örebro. 2011.

LATISLAW, C. L. **Intro to Android**. [S.l.]: [s.n.], 2013.

MACÁRIO, C. G.; BALDO, S. **O modelo relacional**. Universidade Estadual de Campinas. [S.l.]. 2005.

MARTINS, F. J. C. **Monitoração de sinais fisiológicos: Projeto de um eletrocardiógrafo wireless**. Universidade da Beira Interior. Covilhã. 2012.

MEIRA, R. **Apostila de banco de dados**. IFBA - Campus Ilhéus. [S.l.]. 2013.

MELLO, L. C. D. S.; MEYER, J. F. **Levantamento de requisitos**. [S.l.]. 2010.

MERSON, P. **Data model as an architectural view**. Carnegie Mellon University. [S.l.]. 2009.

MURAKAMI, A. et al. A Continuous Glucose Monitoring System in Critical Cardiac Patients in the Intensive Care Unit. **IEEE Computers in Cardiology**, 2006.

OLIVEIRA, T. F. **Desenvolvimento de aplicação para auditoria de processos empresariais para dispositivos móveis**. Faculdade de Tecnologia de São José dos Campos. São José dos Campos. 2011.

OPEN SIGNAL. Android Fragmentation Visualized. **Open Signal**, 2014. Disponivel em: <http://opensignal.com/reports/2014/android-fragmentation/>. Acesso em: 9 Agosto 2015.

PANDIYAN, D.; PARANJAPE, S. **Android Architecture and Binder**. [S.l.]. 2013.

PAULA FILHO, W. D. P. **Engenharia de Software:** fundamentos, métodos e padrões. [S.l.]: LTC, 2000.

POTTER, P. A.; PERRY, A. G. **Fundamentos de Enfermagem**. 7ª. ed. Rio de Janeiro: Elsevier, 2009.

PRAKASH, S.; VENKATESH, V. Real time monitoring of ECG signal using PIC and web server. **International Journal of Engineering and Technology**, 2013.

RAMAKRISHNAN, R.; GEHRKE, J. **Database Management Systems**. 3ª. ed. Singapore: McGraw-Hill, 2003.

RAMINHOS, J. P. B. D. V. **Aquisição de Sinais Fisiológicos: Aplicação ao controlo de uma plataforma móvel a partir do EOG**. Universidade Técnica de Lisboa. Lisboa. 2009.

RAVI, N. et al. Context-aware Battery Management for Mobile Phones, 2008.

ROSENBERG, D.; STEPHENS, M. **Use Case Driven Object Modeling with UML**. New York: Apress, 2007.

RUMBAUGH, J.; JACOBSON, I.; BOOCH, G. **The unified modeling language user guide**. Massachusetts: Addison-Wesley, 1998.

RUMBAUGH, J.; JACOBSON, I.; BOOCH, G. **The unified modeling language reference manual**. 2ª. ed. Massachusetts: Addison-Wesley, 2005.

SHIN, J. W. et al. The web-based fuzzy patient monitor system. **EMBS International Conference**, Chicago, Julho 2000.

SILVA, P. C. B. Utilizando UML: Diagrama de Classes. **SQL Magazine**, v. 63, 2013.

SIQUEIRA, E. **Tecnologias que mudam nossa vida**. São Paulo: Saraiva, 2007.

SOMMERVILLE, I. **Software Engineering**. 8ª. ed. [S.l.]: [s.n.], 2006.

SOUZA, C. A. R.; SOUZA, V. E. S. Modelagem de software com UML. **Easy Java Magazine 4**, 2014.

SPARX SYSTEMS. **Data Modeling: From Conceptual Model to DBMS**. [S.l.]. 2011.

STADZISZ, P. C. **Linguagem de modelagem unificada**. Centro Federal de Educação Tecnológica do Paraná. [S.l.]. 2002.

STADZISZ, P. C. **Projeto de software usando a UML**. Centro Federal de Educação Tecnológica do Paraná. [S.l.]. 2002.

STATISTA. Number of apps available in leading app stores as of July 2015. **Statista**, 2015. Disponivel em: <http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/>. Acesso em: 13 Agosto 2015.

STEWART, J. V. **Vital Signs and Ressuscitation**. Georgetown: Landes Bioscience, 2003.

SUNDARAM, P. Patient monitoring system using Android technology. **International Journal of Computer Science and Mobile Computing**, Maio 2013.

TUTORIALS POINT. **JSON: JavaScript Object Notation**. [S.l.].

VANDER, A. J.; SHERMAN, J. H.; LUCIANO, D. S. **Human Physiology:** The Mechanisms of Body Function. [S.l.]: McGraw-Hill, 2001.

VAUGHANS, B. W. **Fundamentos de enfermagem – Desmistificados:** Um guia de aprendizado. São Paulo: Artmed, 2012.

WILSON, I. Oximetria de pulso – Parte 1. **Sociedade Brasileira de Anestesiologia**, 2013.

YANNAKOPOULOS, J. **HyperText Transfer Protocol: A short course**. University of Crete. [S.l.]. 2003.

APÊNDICES

Apêndice A

Documentação de casos de uso

Identificação:	CSU01		
Nome do Caso de Uso:	Log in		
Criado por:	José Augusto Comiotto Rottini		
Data de Criação:	04/08/2015	**Data de revisão:**	12/10/2015
Ator Principal:	Usuários (Médico, enfermeira, recepcionista, administrador)		
Resumo:	Este caso de uso descreve os passos necessários um funcionário do hospital realizar log in no sistema.		
Pré-condições:	1. Estar cadastrado no sistema.		
Pós-condições:	O usuário estará autenticado no sistema e habilitado a utilizar as funções do mesmo.		
Fluxo Principal:	1. O sistema exibe os campos de login e senha. 2. O usuário preenche os campos com suas credenciais. 3. O usuário confirma a operação. 4. O sistema informa o sucesso no log in.		
Fluxo de Exceção 1: Campos em branco	a. Se o usuário não fornece alguma informação, o sistema reporta o fato e o caso de uso retorna ao passo 1.		
Fluxo de Exceção 2: Erro na comunicação com o servidor	a. Se ocorrer um erro na comunicação com o servidor, o sistema reporta o fato e o caso de uso retorna ao passo 1.		
Fluxo de Exceção 3: Sem conexão com a internet	a. Se não houver uma conexão com a internet, o sistema reporta o fato e o caso de uso retorna ao passo 1.		
Fluxo de Exceção 4: Credenciais incorretas	a. Caso os dados informados estejam incorretos, o sistema reporta o fato e o caso de uso retorna ao passo 1.		

(continua)

Identificação:	CSU02		
Nome do Caso de Uso:	Log out		
Criado por:	José Augusto Comiotto Rottini		
Data de Criação:	04/08/2015	**Data de revisão:**	12/10/2015
Ator Principal:	Usuários (Médico, enfermeira, recepcionista, administrador)		
Resumo:	Este caso de uso descreve os passos necessários um funcionário do hospital realizar log out do sistema.		
Pré-condições:	1. Estar cadastrado no sistema. 2. Estar autenticado no sistema.		

Pós-condições:	O usuário estará descredenciado do sistema.
Fluxo Principal:	1. O sistema exibe a opção de log out. 2. O usuário seleciona a opção de log out.
Fluxo de Exceção 1: Erro na comunicação com o servidor	a. Se ocorrer um erro na comunicação com o servidor, o sistema reporta o fato e o caso de uso retorna ao passo 1.
Fluxo de Exceção 2: Sem conexão com a internet	a. Se não houver uma conexão com a internet, o sistema reporta o fato e o caso de uso retorna ao passo 1.

Identificação:	CSU03		
Nome do Caso de Uso:	Monitorar um paciente		
Criado por:	José Augusto Comiotto Rottini		
Data de Criação:	04/08/2015	**Data de revisão:**	12/10/2015
Ator Principal:	Médico		
Resumo:	Este caso de uso descreve os passos necessários para o médico monitorar um paciente em tempo real.		
Pré-condições:	1. Estar autenticado no sistema.		
Pós-condições:	Os sinais vitais do paciente são exibidos.		
Fluxo Principal:	1. O aplicativo exibe a lista de pacientes do médico que está autenticado. 2. O médico seleciona o paciente que deseja obter as informações. 3. O aplicativo exibe as informações do paciente selecionado, juntamente com um menu de ações relativas ao paciente. 4. O médico seleciona a opção de monitorar. 5. O aplicativo exibe os parâmetros atuais do paciente em questão.		
Fluxo de Exceção 1: Erro na comunicação com o servidor	a. Se ocorrer um erro na comunicação com o servidor, o sistema reporta o fato.		
Fluxo de Exceção 2: Sem conexão com a internet	a. Se não houver uma conexão com a internet, o sistema reporta o fato e a operação deve ser realizada novamente após a conexão com a internet ser reestabelecida.		

(continua)

Identificação:	CSU04		
Nome do Caso de Uso:	Diagnosticar um paciente		
Criado por:	José Augusto Comiotto Rottini		
Data de Criação:	04/08/2015	**Data de revisão:**	12/10/2015
Ator Principal:	Médico		
Resumo:	Este caso de uso descreve os passos necessários para o médico registrar um diagnóstico para um paciente.		
Pré-condições:	1. Estar autenticado no sistema.		
Pós-condições:	O diagnóstico do paciente é salvo no sistema.		

Fluxo Principal:	1. O aplicativo exibe a lista de pacientes do médico que está autenticado. 2. O médico seleciona o paciente que deseja obter as informações. 3. O aplicativo exibe as informações do paciente selecionado, juntamente com um menu de ações relativas ao paciente. 4. O médico seleciona a opção de diagnosticar. 5. O aplicativo exibe os campos para registro do diagnóstico. 6. O médico preenche os campos exibidos na tela. 7. O médico confirma a operação. 8. O aplicativo grava os dados no banco de dados e emite uma mensagem de confirmação.
Fluxo de Exceção 1: Campos em branco ou inválidos	a. Se o médico não fornece alguma informação ou fornece dados inválidos, o sistema reporta o fato e o caso de uso retorna ao passo 6.
Fluxo de Exceção 2: Erro na comunicação com o servidor	a. Se ocorrer um erro na comunicação com o servidor, o sistema reporta o fato e a operação deve ser realizada novamente.
Fluxo de Exceção 3: Sem conexão com a internet	a. Se não houver uma conexão com a internet, o sistema reporta o fato e a operação deve ser realizada novamente após a conexão com a internet ser reestabelecida.

(continua)

Identificação:	CSU05		
Nome do Caso de Uso:	Visualizar o histórico médico de um paciente		
Criado por:	José Augusto Comiotto Rottini		
Data de Criação:	04/08/2015	**Data de revisão:**	12/10/2015
Ator Principal:	Médico		
Resumo:	Este caso de uso descreve os passos necessários para o médico visualizar o histórico médico de um paciente.		
Pré-condições:	1. Estar autenticado no sistema.		
Pós-condições:	O histórico médico do paciente é exibido na tela.		
Fluxo Principal:	1. O aplicativo exibe a lista de pacientes do médico que está autenticado. 2. O médico seleciona o paciente que deseja obter as informações. 3. O aplicativo exibe as informações do paciente selecionado, juntamente com um menu de ações relativas ao paciente. 4. O médico seleciona a opção de visualizar histórico médico. 5. O aplicativo exibe o histórico médico do paciente em questão.		
Fluxo de Exceção 1:	a. Se ocorrer um erro na comunicação com o servidor, o		

Erro na comunicação com o servidor	sistema reporta o fato e a operação deve ser realizada novamente.
Fluxo de Exceção 2: Sem conexão com a internet	a. Se não houver uma conexão com a internet, o sistema reporta o fato e a operação deve ser realizada novamente após a conexão com a internet ser reestabelecida.

Identificação:	CSU06		
Nome do Caso de Uso:	Realiza solicitação.		
Criado por:	José Augusto Comiotto Rottini		
Data de Criação:	04/08/2015	**Data de revisão:**	12/10/2015
Ator Principal:	Médico		
Resumo:	Este caso de uso descreve os passos necessários para o médico realizar uma solicitação para a equipe médica.		
Pré-condições:	1. Estar autenticado no sistema.		
Pós-condições:	Uma solicitação para o paciente é salva no sistema.		
Fluxo Principal:	1. O aplicativo exibe a lista de pacientes do médico que está autenticado. 2. O médico seleciona o paciente que deseja obter as informações. 3. O aplicativo exibe as informações do paciente selecionado, juntamente com um menu de ações relativas ao paciente. 4. O médico seleciona a opção de realizar uma solicitação. 5. O aplicativo exibe os campos para registro da solicitação. 6. O médico preenche os campos exibidos na tela. 7. O médico confirma a operação. 8. O aplicativo grava os dados no banco de dados e emite uma mensagem de confirmação.		
Fluxo de Exceção 1: Campos em branco ou inválidos	a. Se o médico não fornece alguma informação ou fornece dados inválidos, o sistema reporta o fato e o caso de uso retorna ao passo 5.		
Fluxo de Exceção 2: Erro na comunicação com o servidor	a. Se ocorrer um erro na comunicação com o servidor, o sistema reporta o fato e a operação deve ser realizada novamente.		
Fluxo de Exceção 3: Sem conexão com a internet	a. Se não houver uma conexão com a internet, o sistema reporta o fato e a operação deve ser realizada novamente após a conexão com a internet ser reestabelecida.		

Identificação:	CSU07
Nome do Caso de Uso:	Registrar evolução do paciente
Criado por:	José Augusto Comiotto Rottini

Data de Criação:	04/08/2015	**Data de revisão:**	12/10/2015
Ator Principal:	Médico		
Resumo:	Este caso de uso descreve os passos necessários para o médico registrar a evolução de um paciente.		
Pré-condições:	1. Estar autenticado no sistema.		
Pós-condições:	Um registro de evolução é salvo no sistema.		
Fluxo Principal:	1. O aplicativo exibe a lista de pacientes do médico que está autenticado. 2. O médico seleciona o paciente que deseja obter as informações. 3. O aplicativo exibe as informações do paciente selecionado, juntamente com um menu de ações relativas ao paciente. 4. O médico seleciona a opção de registrar uma evolução. 5. O aplicativo exibe os campos para registro da evolução. 6. O médico preenche os campos exibidos na tela. 7. O médico confirma a operação. 8. O aplicativo grava os dados no banco de dados e emite uma mensagem de confirmação.		
Fluxo de Exceção 1: Campos em branco ou inválidos	a. Se o médico não fornece alguma informação ou fornece dados inválidos, o sistema reporta o fato e o caso de uso retorna ao passo 5.		
Fluxo de Exceção 2: Erro na comunicação com o servidor	a. Se ocorrer um erro na comunicação com o servidor, o sistema reporta o fato e a operação deve ser realizada novamente.		
Fluxo de Exceção 3: Sem conexão com a internet	a. Se não houver uma conexão com a internet, o sistema reporta o fato e a operação deve ser realizada novamente após a conexão com a internet ser reestabelecida.		

(continua)

Identificação:	CSU08		
Nome do Caso de Uso:	Visualizar notificações.		
Criado por:	José Augusto Comiotto Rottini		
Data de Criação:	24/08/2015	**Data de revisão:**	12/10/2015
Ator Principal:	Médico		
Ator Secundário:	Paciente		
Resumo:	Este caso de uso descreve os passos necessários para o médico visualizar uma notificação.		
Pré-condições:	1. Estar autenticado no sistema.		
Pós-condições:	Uma notificação médica é exibida.		
Fluxo Principal:	1. O aplicativo exibe uma notificação relativa a um problema médico ou resposta de uma solicitação. 2. O médico visualiza a notificação.		
Fluxo Alternativo: Selecionar notificação	a. O médico clica sobre a notificação. b. Os detalhes da notificação são exibidos.		
Fluxo de Exceção 1: Erro na comunicação	a. Se ocorrer um erro na comunicação com o servidor, o sistema reporta o fato e a operação deve ser realizada		

com o servidor	novamente.
Fluxo de Exceção 2: Sem conexão com a internet	a. Se não houver uma conexão com a internet, o sistema reporta o fato e a operação deve ser realizada novamente após a conexão com a internet ser reestabelecida.

Identificação:	CSU09		
Nome do Caso de Uso:	Visualizar solicitações.		
Criado por:	José Augusto Comiotto Rottini		
Data de Criação:	16/08/2015	**Data de revisão:**	12/10/2015
Ator Principal:	Enfermeira		
Resumo:	Este caso de uso descreve os passos necessários para a enfermeira visualizar as solicitações para um paciente.		
Pré-condições:	1. Estar autenticado no sistema. 2. Trabalhar na mesma ala em que o paciente em questão se encontra.		
Pós-condições:	As solicitações foram visualizadas.		
Fluxo Principal:	1. A enfermeira entra na opção de visualizar solicitações. 2. As solicitações pendentes são exibidas na tela.		
Fluxo de Exceção 1: Erro na comunicação com o servidor	a. Se ocorrer um erro na comunicação com o servidor, o sistema reporta o fato e a operação deve ser realizada novamente.		
Fluxo de Exceção 2: Sem conexão com a internet	a. Se não houver uma conexão com a internet, o sistema reporta o fato e a operação deve ser realizada novamente após a conexão com a internet ser reestabelecida.		

(continua)

Identificação:	CSU10		
Nome do Caso de Uso:	Atender um paciente.		
Criado por:	José Augusto Comiotto Rottini		
Data de Criação:	04/08/2015	**Data de revisão:**	12/10/2015
Ator Principal:	Enfermeira		
Ator Secundário:	Paciente		
Resumo:	Este caso de uso descreve os passos necessários para a enfermeira atender um paciente.		
Pré-condições:	1. Trabalhar na mesma ala em que o paciente em questão se encontra.		
Pós-condições:	O paciente foi atendido.		
Fluxo Principal:	1. A enfermeira realiza o atendimento a um paciente.		
Fluxo Alternativo:	a. A enfermeira visualiza uma solicitação do médico e o caso de uso continua a partir do passo 1.		
Fluxo de Exceção 1: Erro na comunicação com o servidor	a. Se ocorrer um erro na comunicação com o servidor, o sistema reporta o fato e a operação deve ser realizada novamente.		
Fluxo de Exceção 2:	a. Se não houver uma conexão com a internet, o sistema		

Sem conexão com a internet	reporta o fato e a operação deve ser realizada novamente após a conexão com a internet ser reestabelecida.

Identificação:	CSU11		
Nome do Caso de Uso:	Responde solicitação.		
Criado por:	José Augusto Comiotto Rottini		
Data de Criação:	04/08/2015	**Data de revisão:**	12/10/2015
Ator Principal:	Enfermeira		
Resumo:	Este caso de uso descreve os passos necessários para a enfermeira responder a uma solicitação do médico.		
Pré-condições:	1. Estar autenticado no sistema. 2. Ter visualizado uma solicitação. 3. Ter realizado o atendimento ao paciente.		
Pós-condições:	A resposta da solicitação é gravada no sistema e uma notificação é enviada ao médico relacionado.		
Fluxo Principal:	1. Uma requisição feita pelo médico é exibida no sistema. 2. A enfermeira insere uma resposta para a solicitação e confirma a operação. 3. O sistema grava os dados no banco de dados e emite uma mensagem de confirmação.		
Fluxo de Exceção 1: Erro na comunicação com o servidor	a. Se ocorrer um erro na comunicação com o servidor, o sistema reporta o fato e a operação deve ser realizada novamente.		
Fluxo de Exceção 2: Sem conexão com a internet	a. Se não houver uma conexão com a internet, o sistema reporta o fato e a operação deve ser realizada novamente após a conexão com a internet ser reestabelecida.		
Inclusões:	Visualiza solicitações e Atende paciente.		

(continua)

Identificação:	CSU12		
Nome do Caso de Uso:	Registrar evolução do paciente.		
Criado por:	José Augusto Comiotto Rottini		
Data de Criação:	04/08/2015	**Data de revisão:**	12/10/2015
Ator Principal:	Enfermeira		
Resumo:	Este caso de uso descreve os passos necessários para a enfermeira registrar uma evolução do paciente.		
Pré-condições:	1. Estar autenticado no sistema. 2. Trabalhar na mesma ala em que o paciente em questão se encontra.		
Pós-condições:	Um registro de evolução do paciente é salvo no sistema.		
Fluxo Principal:	1. A enfermeira entra na opção de registrar uma evolução. 2. O sistema exibe os campos para o registro de uma evolução do paciente.		

	3. A enfermeira preenche os campos exibidos pelo sistema. 4. A enfermeira confirma a operação. 5. O sistema grava os dados no banco de dados e emite uma mensagem de confirmação.
Fluxo de Exceção 1: Campos em branco ou inválidos	a. Se a enfermeira não fornece alguma informação ou fornece dados inválidos, o sistema reporta o fato e o caso de uso retorna ao passo 2.
Fluxo de Exceção 2: Erro na comunicação com o servidor	a. Se ocorrer um erro na comunicação com o servidor, o sistema reporta o fato e a operação deve ser realizada novamente.
Fluxo de Exceção 3: Sem conexão com a internet	a. Se não houver uma conexão com a internet, o sistema reporta o fato e a operação deve ser realizada novamente após a conexão com a internet ser reestabelecida.

Identificação:	CSU13		
Nome do Caso de Uso:	Cadastra paciente.		
Criado por:	José Augusto Comiotto Rottini		
Data de Criação:	04/08/2015	**Data de revisão:**	12/10/2015
Ator Principal:	Recepcionista		
Resumo:	Este caso de uso descreve os passos necessários para o cadastro de um paciente no hospital.		
Pré-condições:	1. Estar autenticado no sistema.		
Pós-condições:	Um paciente é cadastrado no sistema.		
Fluxo Principal:	1. O sistema exibe os campos para o registro de admissão do paciente. 2. A recepcionista preenche os campos exibidos pelo sistema. 3. A recepcionista confirma a operação. 4. O sistema grava os dados no banco de dados e emite uma mensagem de confirmação.		
Fluxo de Exceção 1: Campos em branco ou inválidos	a. Se a recepcionista não fornece alguma informação ou fornece dados inválidos, o sistema reporta o fato e o caso de uso retorna ao passo 1.		
Fluxo de Exceção 2: Erro na comunicação com o servidor	a. Se ocorrer um erro na comunicação com o servidor, o sistema reporta o fato e a operação deve ser realizada novamente.		
Fluxo de Exceção 3: Sem conexão com a internet	a. Se não houver uma conexão com a internet, o sistema reporta o fato e a operação deve ser realizada novamente após a conexão com a internet ser reestabelecida.		

Identificação:	CSU14		
Nome do Caso de Uso:	Edita informações do paciente.		
Criado por:	José Augusto Comiotto Rottini		
Data de Criação:	04/08/2015	**Data de revisão:**	12/10/2015

Ator Principal:	Recepcionista
Resumo:	Este caso de uso descreve os passos necessários para a recepcionista editar as informações de um paciente.
Pré-condições:	1. Estar autenticado no sistema.
Pós-condições:	As informações de um paciente foram editadas.
Fluxo Principal:	1. O sistema exibe os pacientes do hospital em uma tabela, com os campos editáveis. 2. A recepcionista edita os campos exibidos pelo sistema. 3. A recepcionista confirma a operação. 4. O sistema grava os dados no banco de dados e emite uma mensagem de confirmação.
Fluxo Alternativo: Editar histórico médico	a. A recepcionista clica no botão de mais informações do paciente. b. O histórico médico do paciente abre e o caso de uso continua a partir do passo 2.
Fluxo de Exceção 1: Campos em branco ou inválidos	a. Se a recepcionista não fornece alguma informação ou fornece dados inválidos, o sistema reporta o fato e o caso de uso retorna ao passo 1.
Fluxo de Exceção 2: Erro na comunicação com o servidor	a. Se ocorrer um erro na comunicação com o servidor, o sistema reporta o fato e a operação deve ser realizada novamente.
Fluxo de Exceção 3: Sem conexão com a internet	a. Se não houver uma conexão com a internet, o sistema reporta o fato e a operação deve ser realizada novamente após a conexão com a internet ser reestabelecida.

(continua)

Identificação:	CSU15		
Nome do Caso de Uso:	Remove paciente do sistema.		
Criado por:	José Augusto Comiotto Rottini		
Data de Criação:	04/08/2015	**Data de revisão:**	12/10/2015
Ator Principal:	Recepcionista		
Resumo:	Este caso de uso descreve os passos necessários para a recepcionista remover um paciente do sistema.		
Pré-condições:	1. Estar autenticado no sistema.		
Pós-condições:	Um paciente foi removido do sistema.		
Fluxo Principal:	1. O sistema exibe os pacientes do hospital em uma tabela, com os campos editáveis. 2. A recepcionista clica no botão de remover em um dos pacientes. 3. O sistema salva os dados no banco de dados e emite uma mensagem de confirmação.		
Fluxo de Exceção 2: Erro na comunicação com o servidor	a. Se ocorrer um erro na comunicação com o servidor, o sistema reporta o fato e a operação deve ser realizada novamente.		
Fluxo de Exceção 3: Sem conexão com a	a. Se não houver uma conexão com a internet, o sistema reporta o fato e a operação deve ser realizada novamente		

internet	após a conexão com a internet ser reestabelecida.

Identificação:	CSU16		
Nome do Caso de Uso:	Cadastra usuário.		
Criado por:	José Augusto Comiotto Rottini		
Data de Criação:	04/08/2015	**Data de revisão:**	12/10/2015
Ator Principal:	Administrador		
Resumo:	Este caso de uso descreve os passos necessários o administrador cadastrar um usuário no sistema.		
Pré-condições:	1. Estar autenticado no sistema.		
Pós-condições:	Um usuário foi cadastrado no sistema.		
Fluxo Principal:	1. O administrador seleciona a opção de visualizar os usuários. 2. O sistema exibe os campos para cadastrar um usuário. 3. O administrador preenche os campos. 4. O administrador confirma a operação 5. O sistema salva os dados no banco de dados e emite uma mensagem de confirmação.		
Fluxo de Exceção 1: Campos em branco ou inválidos	a. Se o administrador não fornece alguma informação ou fornece dados inválidos, o sistema reporta o fato e o caso de uso retorna ao passo 2.		
Fluxo de Exceção 2: Erro na comunicação com o servidor	a. Se ocorrer um erro na comunicação com o servidor, o sistema reporta o fato e a operação deve ser realizada novamente.		
Fluxo de Exceção 3: Sem conexão com a internet	a. Se não houver uma conexão com a internet, o sistema reporta o fato e a operação deve ser realizada novamente após a conexão com a internet ser reestabelecida.		

(continua)

Identificação:	CSU17		
Nome do Caso de Uso:	Edita usuário.		
Criado por:	José Augusto Comiotto Rottini		
Data de Criação:	04/08/2015	**Data de revisão:**	12/10/2015
Ator Principal:	Administrador		
Resumo:	Este caso de uso descreve os passos necessários o administrador editar um usuário.		
Pré-condições:	1. Estar autenticado no sistema.		
Pós-condições:	As informações de um usuário foram editadas.		
Fluxo Principal:	1. O administrador seleciona a opção de visualizar os usuários. 2. O sistema exibe os usuários em uma tabela, com os campos editáveis. 3. O administrador edita os campos.		

	4. O administrador confirma a operação 5. O sistema salva os dados no banco de dados e emite uma mensagem de confirmação.
Fluxo de Exceção 1: Campos em branco ou inválidos	a. Se o administrador não fornece alguma informação ou fornece dados inválidos, o sistema reporta o fato e o caso de uso retorna ao passo 2.
Fluxo de Exceção 2: Erro na comunicação com o servidor	a. Se ocorrer um erro na comunicação com o servidor, o sistema reporta o fato e a operação deve ser realizada novamente.
Fluxo de Exceção 3: Sem conexão com a internet	a. Se não houver uma conexão com a internet, o sistema reporta o fato e a operação deve ser realizada novamente após a conexão com a internet ser reestabelecida.

Identificação:	CSU18		
Nome do Caso de Uso:	Remove usuário.		
Criado por:	José Augusto Comiotto Rottini		
Data de Criação:	04/08/2015	**Data de revisão:**	12/10/2015
Ator Principal:	Administrador		
Resumo:	Este caso de uso descreve os passos necessários o administrador remover um usuário.		
Pré-condições:	1. Estar autenticado no sistema.		
Pós-condições:	Um usuário é removido do sistema.		
Fluxo Principal:	1. O administrador seleciona a opção de visualizar os usuários. 2. O sistema exibe os usuários em uma tabela, com os campos editáveis. 3. O administrador seleciona a opção de remover um dos usuários. 4. O sistema salva os dados no banco de dados e emite uma mensagem de confirmação.		
Fluxo de Exceção 1: Erro na comunicação com o servidor	a. Se ocorrer um erro na comunicação com o servidor, o sistema reporta o fato e a operação deve ser realizada novamente.		
Fluxo de Exceção 2: Sem conexão com a internet	a. Se não houver uma conexão com a internet, o sistema reporta o fato e a operação deve ser realizada novamente após a conexão com a internet ser reestabelecida.		

Identificação:	CSU19		
Nome do Caso de Uso:	Cadastra medicamento.		
Criado por:	José Augusto Comiotto Rottini		
Data de Criação:	04/08/2015	**Data de revisão:**	12/10/2015

Ator Principal:	Administrador
Resumo:	Este caso de uso descreve os passos necessários o administrador cadastrar um medicamento no sistema.
Pré-condições:	1. Estar autenticado no sistema.
Pós-condições:	Um medicamento foi cadastrado no sistema.
Fluxo Principal:	1. O administrador seleciona a opção de visualizar os medicamentos. 2. O sistema exibe os campos para cadastrar um medicamento. 3. O administrador preenche os campos. 4. O administrador confirma a operação 5. O sistema salva os dados no banco de dados e emite uma mensagem de confirmação.
Fluxo de Exceção 1: Campos em branco ou inválidos	a. Se o administrador não fornece alguma informação ou fornece dados inválidos, o sistema reporta o fato e o caso de uso retorna ao passo 2.
Fluxo de Exceção 2: Erro na comunicação com o servidor	a. Se ocorrer um erro na comunicação com o servidor, o sistema reporta o fato e a operação deve ser realizada novamente.
Fluxo de Exceção 3: Sem conexão com a internet	a. Se não houver uma conexão com a internet, o sistema reporta o fato e a operação deve ser realizada novamente após a conexão com a internet ser reestabelecida.

(continua)

Identificação:	CSU20		
Nome do Caso de Uso:	Remove medicamento.		
Criado por:	José Augusto Comiotto Rottini		
Data de Criação:	04/08/2015	**Data de revisão:**	12/10/2015
Ator Principal:	Administrador		
Resumo:	Este caso de uso descreve os passos necessários o administrador remover um medicamento do sistema.		
Pré-condições:	1. Estar autenticado no sistema.		
Pós-condições:	Um medicamento foi removido do sistema.		
Fluxo Principal:	1. O administrador seleciona a opção de visualizar os medicamentos. 2. O sistema exibe os medicamentos em uma tabela. 3. O administrador seleciona a opção de remover um dos medicamentos. 4. O sistema salva os dados no banco de dados e emite uma mensagem de confirmação.		
Fluxo de Exceção 2: Erro na comunicação com o servidor	a. Se ocorrer um erro na comunicação com o servidor, o sistema reporta o fato e a operação deve ser realizada novamente.		
Fluxo de Exceção 3: Sem conexão com a internet	a. Se não houver uma conexão com a internet, o sistema reporta o fato e a operação deve ser realizada novamente após a conexão com a internet ser reestabelecida.		

Identificação:	CSU21		
Nome do Caso de Uso:	Cadastra diagnóstico.		
Criado por:	José Augusto Comiotto Rottini		
Data de Criação:	04/08/2015	**Data de revisão:**	12/10/2015
Ator Principal:	Administrador		
Resumo:	Este caso de uso descreve os passos necessários o administrador cadastrar um diagnóstico no sistema.		
Pré-condições:	1. Estar autenticado no sistema.		
Pós-condições:	Um diagnóstico foi cadastrado no sistema.		
Fluxo Principal:	1. O administrador seleciona a opção de visualizar os diagnósticos. 2. O sistema exibe os campos para cadastrar um diagnóstico. 3. O administrador preenche os campos. 4. O administrador confirma a operação 5. O sistema salva os dados no banco de dados e emite uma mensagem de confirmação.		
Fluxo de Exceção 1: Campos em branco ou inválidos	a. Se o administrador não fornece alguma informação ou fornece dados inválidos, o sistema reporta o fato e o caso de uso retorna ao passo 2.		
Fluxo de Exceção 2: Erro na comunicação com o servidor	a. Se ocorrer um erro na comunicação com o servidor, o sistema reporta o fato e a operação deve ser realizada novamente.		
Fluxo de Exceção 3: Sem conexão com a internet	a. Se não houver uma conexão com a internet, o sistema reporta o fato e a operação deve ser realizada novamente após a conexão com a internet ser reestabelecida.		

(continua)

Identificação:	CSU22		
Nome do Caso de Uso:	Remove diagnóstico.		
Criado por:	José Augusto Comiotto Rottini		
Data de Criação:	04/08/2015	**Data de revisão:**	12/10/2015
Ator Principal:	Administrador		
Resumo:	Este caso de uso descreve os passos necessários o administrador remover um diagnóstico do sistema.		
Pré-condições:	1. Estar autenticado no sistema.		
Pós-condições:	Um medicamento foi cadastrado no sistema.		
Fluxo Principal:	1. O administrador seleciona a opção de visualizar os diagnósticos. 2. O sistema exibe os diagnósticos em uma tabela. 3. O administrador seleciona a opção de remover um dos diagnósticos. 4. O sistema salva os dados no banco de dados e emite uma		

	mensagem de confirmação.
Fluxo de Exceção 2: Erro na comunicação com o servidor	a. Se ocorrer um erro na comunicação com o servidor, o sistema reporta o fato e a operação deve ser realizada novamente.
Fluxo de Exceção 3: Sem conexão com a internet	a. Se não houver uma conexão com a internet, o sistema reporta o fato e a operação deve ser realizada novamente após a conexão com a internet ser reestabelecida.