Universidade Federal do Piauí
Campus Senador Helvídio Nunes de Barros
Curso Bacharelado em Sistemas de Informação

Andrei Maxwel da Silva Lima

Um protótipo de Sistema de Informação Clínico para agilizar o atendimento médico em postos de saúde da rede pública.

Picos
2014

Andrei Maxwel da Silva Lima

Um protótipo de Sistema de Informação Clínico para agilizar o atendimento médico em postos de saúde da rede pública.

Trabalho de Conclusão de Curso apresentado ao Curso Bacharelado em Sistemas de Informação, Campus Senador Helvídio Nunes de Barros da Universidade Federal do Piauí, como parte dos requisitos para obtenção do Grau de Bacharelado, sob orientação do Professor Especialista Ivenilton Alexandre.

Picos

2014

Eu, **Andrei Maxwel da Silva Lima,** abaixo identificado(a) como autor(a), autorizo a biblioteca da Universidade Federal do Piauí a divulgar, gratuitamente, sem ressarcimento de direitos autorais, o texto integral da publicação abaixo discriminada, de minha autoria, em seu site, em formato PDF, para fins de leitura e/ou impressão, a partir da data de hoje.

Picos-PI, 12 de agosto de 2014.

Andrei Manuel da S. Lima
Assinatura

FICHA CATALOGRÁFICA
Serviço de Processamento Técnico da Universidade Federal do Piauí
Biblioteca José Albano de Macêdo

L732p Lima, Andrei Maxwel da Silva.
Um protótipo de sistema de informação clínico para agilizar o atendimento médico em postos de saúde da rede pública / Andrei Maxwel da Silva Lima. - 2014.
CD-ROM : il. ; 4 ¾ pol. (115 p.)

Monografia(Bacharelado em Sistemas de Informação) – Universidade Federal do Piauí. Picos-PI, 2014.
Orientador(A): Prof. Esp. Ivenilton Alexandre de Sousa Moura

1. Saúde Pública. 2. Aplicativo Móvel. 3. Sistema Web. 4. Web Service I. Título.

CDD 004.21

Andrei Maxwel da Silva Lima

Um protótipo de Sistema de Informação Clínico para agilizar o atendimento médico em postos de saúde da rede pública.

Trabalho de Conclusão de Curso apresentado ao Curso Bacharelado em Sistemas de Informação, Campus Senador Helvídio Nunes de Barros da Universidade Federal do Piauí, como parte dos requisitos para obtenção do Grau de Bacharelado, sob orientação do Professor Especialista Ivenilton Alexandre.

Data de Aprovação:

05/08/2014

Ivenilton Alexandre de Souza Moura

UFPI/CSHNB

Flávio Henrique Duarte de Araújo

UFPI/CSHNB

Francisco das Chagas Imperes Filho

UFPI/CSHNB

Picos

2014

A Deus. Aos meus pais, Raimundo Barbosa e Valdelice Batista, irmão, André Luiz, e avó, Antônia Pereira.

Agradeço primeiramente a Deus, meu maior mestre, que me deu saúde, força e muita fé para superar as dificuldades não só nestes anos como universitário, mas em toda minha vida.

Não poderia deixar de agradecer minha incrível família, Raimundo Barbosa e Valdelice Batista, meus pais, Antônia Pereira, minha avó, e André Luiz, meu irmão. Obrigado pelo exemplo de honestidade, valor e amor incondicional. Apesar das dificuldades, me fortaleceram e me apoiaram nas horas difíceis, de desânimo e cansaço. Obrigado!

Não posso esquecer também dos meus mais fiéis amigos: Bianca Oliveira, Ana Carine, Mirielly Alves, Bruno Pereira, Bruna Pâmera, Matheus Duarte, Jackson Coutinho, Thassyo Azevedo, Patrick Matos e Adelino Luz. É com vocês que compartilho angústias, alegrias, felicidades e tantas outras coisas que uma amizade faz. Companheiros de vida e irmãos na amizade que fizeram parte da minha formação, que entenderam todos os meus momentos de cansaço e estresse. Vocês vão continuar presentes em minha vida, com certeza.

Por fim, agradeço meus professores: Ivenilton Alexandre, que me orientou neste trabalho, obrigado pela dedicação e incentivo, sem os quais não teria conseguido; Rayner Gomes, obrigado por todas as oportunidades que me destes, e desculpa se lhe desapontei em alguns momentos; Juliana Oliveira, que considero um dos símbolos do curso; Patrícia Medyna, pela dedicação em tornar tanto o curso quanto os alunos melhores a cada dia. Enfim, agradeço a todos os demais professores que mesmo não sendo citados aqui, contribuíram muito para que eu pudesse completar esta caminhada.

Este é o fim de uma etapa e o começo de uma grande jornada.

Tenha talento, trabalhe como um condenado, sue sangue, e você conseguirá tudo sem esforço.

Humberto Gessinger

Continue a nadar, continue a nadar...

Dory, Procurando Nemo

... ao infinito e além!

Buzz Lightyear

Resumo

A partir da premissa de que a tecnologia pode ser aplicada nas mais diversas áreas com o objetivo de potencializar e melhorar os procedimentos, pretende-se com este trabalho construir um sistema de informação formado por dois módulos: um aplicativo para dispositivos móveis baseados na plataforma Android e um sistema *Web*, através do framework Grails. Este sistema de informação deve ser capaz de agilizar o atendimento médico nos postos de saúde e armazenar o histórico dos pacientes de forma organizada e eficiente, visto que um dos principais problemas enfrentados por esta área são relacionados a demora no atendimento e não manutenibilidade das informações obtidas a partir de cada consulta, o que acarreta um procedimento repetitivo de coleta de dados de pacientes a cada atendimento. Para interconectar estas duas tecnologias, Grails e Android, foram utilizados Web Services do tipo RESTful. Além destas tecnologias foram realizados estudos a respeito de Interface Humano Computador, objetivando técnicas de usabilidade para melhor eficácia do usuário ao utilizar a aplicação.

Palavras-chave: Saúde Pública, Aplicativo Móvel, Sistema Web, Web Service.

Abstract

Given the fact that technology can be applied to several areas with the goal of enhancing and improving the procedures, it is intended with this project to build an information system consisting of two modules, an app for Android mobile devices and a Web system, developed in Grails framework, able to streamline medical care at clinics and file the history of the patients in an organized and efficient manner, since one of the main problems faced by this area is related to delay in treatment and non-maintainability of the information obtained from each medical appointment, which entails a repetitive procedure of collecting patient data to each service. To communicate the two technologies, Grails and Android, RESTful Web services were used. Beyond those technologies, the human-computer interaction was studied seeking usability techniques to enhance the user experience in the application.

Keywords: Medical Care, Public Health, Mobile App, Web System, Web Service.

Lista de Figuras

Figura 1 - Arquitetura Grails. 29

Figura 2 - Abstração Padrão MVC. 30

Figura 3 - Exemplo de uma closure. 33

Figura 5 - Exemplo de SiteMesh. 34

Figura 6 - Diretórios criados pelo Grails. 35

Figura 7 - Exemplo de Classe de Domínio. 37

Figura 8 - Exemplo de Controlador. 38

Figura 9 - Arquitetura Android. 41

Figura 10 - Exemplo de declaração de Activity. 43

Figura 11 - Declaração das activies e permissões no arquivo Manifest. 44

Figura 12 - Exemplo de chamada de “Intent” junto ao “Bundle”. 45

Figura 13 - Elementos da mensagem SOAP. 47

Figura 14 - Exemplo de mensagem SOAP. 48

Figura 15 - Arquitetura REST. 49

Figura 16 - Modelo de documento XML. 51

Figura 17 - Trecho em XML. 52

Figura 18 - Trecho em JSON. 52

Figura 19 - Exemplo de linguagem de comando. 54

Figura 20 - Exemplo de menu. 54

Figura 21 - Exemplo de formulário. 55

Figura 22 - Exemplo de janela. 55

Figura 23 - WIMP. 55

Figura 24 - IntelliJ IDEA. 59
Figura 25 - Phpmyadmin. 60
Figura 4 - Funcionamento do SiteMesh. 62
Figura 26 - Diagrama de Classes: Módulo 2. 69
Figura 27 - Pacotes: Módulo 1. 71
Figura 28 - Módulo 1: Arquivos de Layout. 73
Figura 29 - Utilização do comando “setContentView”. 74
Figura 30 - Declaração de permissão de Internet no Manifest. 74
Figura 31 - Classe “WebClient“. 75
Figura 32 - Declaração de permissão para escrever no cartão de memória do dispositivo. 76
Figura 33 - Exemplo de utilização da ”*intent*“ para inicializar a câmera do dispositivo. 76
Figura 34 - Sobrescrevendo o método ”onActivityResult“. 76
Figura 35 - Módulo 2: Classes de Domínio. 77
Figura 36 - Tabelas criadas pelo Grails, visualizadas através do Phpmyadmin. . . . 78
Figura 37 - Controlador ”Área“ e suas actions. 79
Figura 38 - Arquivos de visão gerados pelo ”*generate-all*“. 79
Figura 39 - Implementação do método que envia todos os sintomas para o ”Módulo 1“. 80
Figura 40 - Exemplo de utilização do ”submitToRemote“. 81
Figura 41 - Módulo 1: Tela inicial da aplicação. 82
Figura 42 - Módulo 1: Tela de seleção de médico. 82
Figura 43 - Módulo 1: Mensagem de erro de senha. 83
Figura 44 - Módulo 1: Tela de busca do paciente. 83
Figura 45 - Módulo 1: Tela apresentando um paciente cadastrado no sistema. 84
Figura 46 - Módulo 1: Tela de perfil do paciente. 84
Figura 47 - Módulo 1: Tela de consulta do paciente. 85

Figura 48 - Módulo 1: Tela com um sintoma adicionado ao paciente. 85
Figura 49 - Módulo 1: Tela com dois sintomas adicionados ao paciente. 86
Figura 50 - Módulo 1: Tela apresentando o autocomplete para adição de sintoma. . 86
Figura 51 - Módulo 1: Tela apresentando um sintoma adicional já adicionado à lista. 87
Figura 52 - Módulo 1: Mensagem confirmando a gravação da consulta no banco de dados. 87
Figura 53 - Módulo 1: Tela de geração de requisições de exames. 88
Figura 54 - Módulo 1: Tela de perfil apresentando a consulta realizado. 88
Figura 55 - Módulo 1: Tela de visualização individual da consulta. 89
Figura 56 - Módulo 1: Tela de visualização individual do exame. 89
Figura 57 - Módulo 1: Tela de apresentação da imagem do exame. 90
Figura 58 - Módulo 1: Mensagem apresentando o status da aplicação 90
Figura 59 - Módulo 2: Página de visualização de todos os pacientes. 91
Figura 60 - Módulo 2: Página de visualização de um paciente. 92
Figura 61 - Módulo 2: Página de cadastrado de novo paciente. 92
Figura 62 - Módulo 2: Página de cadastrado de novo sintoma. 93
Figura 63 - Módulo 2: Página de visualização de todos os sintomas. 93
Figura 64 - Módulo 2: Exemplo de utilização do autocomplete. 94
Figura 65 - Gráfico: tempo gasto por consulta. 96
Figura 66 - Gráfico: tempo gasto por retorno. 97
Figura 67 - Gráfico: médias ”Conforme“ por critério. 99
Figura 68 - Gráfico: médias ”Não Conforme“ por critério. 100
Figura 69 - Gráfico: panorama geral. 101
Figura 70 - Gráfico: respostas para primeira questão. 102
Figura 71 - Gráfico: respostas para segunda questão. 102
Figura 72 - Gráfico: respostas para terceira questão. 103
Figura 73 - Gráfico: respostas para quarta questão. 103

Lista de Tabelas

Tabela 1 - Módulo 1: Tabela de Requisitos Funcionais. 66
Tabela 2 - Módulo 2: Tabela de Requisitos Funcionais. 67
Tabela 3 - Módulo 1: Tabela de Requisitos Não Funcionais. 68
Tabela 4 - Módulo 2: Tabela de Requisitos Não Funcionais. 68
Tabela 5 - Médias ”Conforme“ do teste por critério. 99
Tabela 6 - Médias ”Não Conforme“ do teste por critério. 100

Lista de abreviaturas e siglas

API	Application Programming Interface
BSD	Berkeley Software Distribution
CNI	Confederação Nacional da Indústria
CSS	Cascading Style Sheets
DATASUS	Departamento de Informática do Sistema Único de Saúde
DVM	Dalvik Virtual Machine
FTP	File Transfer Protocol
GORM	Grails Object Relational Mapping
GSP	Groovy Server Page
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IHC	Interface Humano Computador
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
MVC	Model, View, Controller
NDK	Native Development Kit
OMS	Organização Mundial de Saúde
RAD	Rapid Application Development
REST	Representational State Transfer
SDK	Software Development Kit
SGML	Standard Generalized Markup Language
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SUS	Sistema Único de Saúde
TIC	Tecnologias da Informação e Comunicação
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XP	eXtreme Programming

Sumário

1 Introdução **14**

1.1 Objetivos 15

1.1.1 Objetivos Gerais 15

1.1.2 Objetivos Específicos 15

1.2 Estrutura da Monografia 15

2 Saúde Pública e Atendimento **17**

2.1 DATASUS (Departamento de Informática do Sistema Único de Saúde) 22

2.1.1 TABWIN 23

2.1.2 HOSPUB 23

2.2 Considerações Finais 24

3 Métodos e Tecnologias **25**

3.1 Metodologia de Desenvolvimento 25

3.1.1 Metodologia de Desenvolvimento Tradicional 25

3.1.2 Metodologia de Desenvolvimento Ágil 26

3.2 O *Framework* Grails (Groovy on Rails) 28

3.2.1 Características 29

3.2.2 Sistema de Diretório Grails 35

3.2.3 Classes de Domínio, Controladores e Visualizações 36

3.2.4 Vantagens e Desvantagens 39

3.3 Android 39

3.3.1 Plataforma Android 40

3.4 Web Services 46
3.4.1 Tipos de Web Services 46
3.4.2 Representações de Dados em Web Services 50
3.5 IHC (Interface Humano Computador) 53
3.5.1 Interface e Interação 53
3.5.2 Estilos de Interação 54
3.5.3 Usabilidade 56
3.5.4 IHC em Dispositivos Móveis 57
3.6 Materiais 58
3.6.1 IDE (Integrated Development Environment) 58
3.6.2 PhpMyAdmin 59
3.6.3 Sistema Operacional Linux 60

4 Desenvolvimento **63**
4.1 O Sistema 63
4.1.1 Análise do Sistema Proposto 63
4.1.2 Etapas do Desenvolvimento 70
4.1.3 Funcionamento 81

5 Resultados **95**
5.1 Testes 95
5.1.1 Testes de Funcionalidades 95
5.1.2 Testes de Usabilidade 97
5.1.3 Testes de Validação de Proposta 101

6 Conclusões e Trabalhos Futuros **104**
6.1 Trabalhos Futuros 105

Referências **106**

1 Introdução

Os avanços tecnológicos trouxeram uma série de vantagens tanto para as pessoas quanto para as organizações. É através dos computadores que são realizadas grande parte das tarefas do dia-a-dia de um indivíduo. Mas não são apenas indivíduos que se beneficiam dos avanços tecnológicos. Hoje, com um mercado totalmente mutável, em que a competitividade acirrada cresce de forma exponencial, as organizações cada vez mais aderem à tecnologia buscando um diferencial competitivo no mercado.

A importância da informação para as organizações é universalmente aceita, constituindo, senão o mais importante, pelo menos um dos recursos cuja gestão e aproveitamento estão diretamente relacionados com o sucesso desejado (MORESI, 2000). Esta importância se aplica não só a organizações privadas, mas também às organizações públicas. Segundo Facchini e Vargas (1992), as organizações públicas também tem se adaptado a essa orientação, principalmente devido à crescente demanda de rapidez e precisão de informações a que são submetidas. Infelizmente, no Brasil, rapidez e precisão de informações não são o ponto forte dos órgãos públicos. E isto não é diferentes nos hospitais públicos, principal meio de tratamento utilizado por pessoas que não possuem condições de acesso à tratamentos particulares.

Segundo pesquisa realizada em janeiro de 2012 pela Confederação Nacional da Indústria[1] (CNI), 61% da população brasileira consideram o serviço público de saúde do País "péssimo" ou "ruim" e 85% dos entrevistados não perceberam avanços no sistema de saúde do País nos últimos três anos. Ainda segundo esta pesquisa, 55% da população brasileira consideram a demora no atendimento como o principal problema do sistema público de saúde em sua cidade. Esta é a principal motivação para o desenvolvimento deste trabalho. Prover um sistema de informação que possa agilizar o atendimento médico e organizar as informações desses atendimentos de forma eficiente.

De acordo com Lemos (2011), os avanços tecnológicos mudaram o ambiente das empresas, incluindo as empresas hospitalares, fazendo a necessidade de um gerenciamento também inovador, sistematizado, oferecendo instrumentos para tomada de decisões. No Brasil, a área hospitalar ainda precisa trilhar um longo caminho em busca da modernização de sua gestão, pois tais organizações ainda se utilizam de métodos contábeis tradicionais e ultrapassados, desconhecendo efetivamente seus custos por não utilizarem qualquer tipo de sistema que oriente e proporcione parâmetros para decisões administrativas, investimentos e controle de atividades (LEMOS, 2011 apud ABBAS, 2001, p. 8). Seja na área pública, reconhecidamente carente, ou

[1] Disponível em: www.cni.org.br/portal/

na privada, em que as operadoras de planos de saúde reclamam das perdas contabilizadas, não há dúvidas de que é preciso rever processos e investir em tecnologias capazes de aumentar o controle e melhorar a qualidade da assistência (PINOCHET, 2011a).

1.1 Objetivos

1.1.1 Objetivos Gerais

O principal objetivo deste trabalho é desenvolver um sistema de informação que sirva como uma possível solução para agilizar o atendimento médico e ainda melhorar a forma de armazenar e organizar os dados dos pacientes.

1.1.2 Objetivos Específicos

Como resultado deste trabalho pretende-se:

- Desenvolver uma aplicação móvel para *tablets* Android que permita catalisar a velocidade do atendimento médico público;
- Desenvolver um sistema *web*, que trabalhe em conjunto com a aplicação móvel, comunicando-se através de serviços, e permita manter os dados dos pacientes e suas respectivas consultas armazenadas e organizadas eficientemente para posterior utilização.

1.2 Estrutura da Monografia

Além deste capítulo de introdução, o trabalho está organizado da seguinte forma:

- Capítulo 2 - Saúde Pública e Atendimento: neste capítulo é apresentada uma visão geral sobre saúde e saúde pública no Brasil, fornecendo parâmetros gerais sobre a situação da saúde pública e seus problemas. Além disto, o capítulo faz ainda uma relação entre saúde e tecnologia, identificando seus principais benefícios;
- Capítulo 3 - Métodos e Tecnologias: neste capítulo são descritas as principais metodologias de desenvolvimento de software, bem como um referencial a respeito de todas as tecnologias envolvidas no processo de desenvolvimento deste trabalho;
- Capítulo 4 - Desenvolvimento: através deste capítulo poderá ser observada a metodologia de desenvolvimento utilizada neste projeto. Ele conta ainda com um detalhamento a respeito do projeto desenvolvido, desde a análise de requisitos até as etapas que se seguem

até a conclusão do projeto. Por fim, o capítulo conta ainda com um passo-a-passo de todo o fluxo de funcionamento da aplicação;

- Capítulo 5 - Resultados: neste capítulo estão presentes os resultados deste trabalho através de simulações e testes da utilização da aplicação;
- Capítulo 6 - Conclusões e Trabalhos Futuros: por fim, o capítulo de conclusão apresenta as conclusões a respeito do trabalho proposto e possíveis trabalhos futuros.

2 Saúde Pública e Atendimento

Saúde é definida pela Organização Mundial de Saúde (OMS) não apenas como a ausência de doença, mas como a situação de perfeito bem-estar físico, mental e social (SEGRE; FERRAZ, 1997). O vínculo afetivo, embutido de confiança recíproca na dupla que empreende uma ação de saúde - médico e paciente -, a par dos aspectos cognitivos, tecnológicos e científicos é decisivo para que se possa esperar a melhora do estado do paciente (SEGRE; FERRAZ, 1997).

"A saúde é um direito de todos e um dever do Estado". Com essas palavras a Constituição Federal de 1988 abre seu art. 196 para expressar o compromisso do Estado de garantir a todos os cidadãos o pleno direito à saúde. Essa garantia, conforme a literalidade do artigo mencionado será efetivada "mediante políticas sociais e econômicas que visem à redução do risco de doença e de outros agravos e ao acesso universal e igualitário às ações e serviços para sua promoção, proteção e recuperação" (GANDINI, 2007).

Na constituição de 1988, a questão de equidade foi tomada como igualdade no acesso aos serviços de saúde, uma vez que garantiu a universalidade de cultura e do atendimento com o propósito de fornecer igual oportunidade de acesso aos serviços de saúde, uma vez que garantiu a universalidade da cobertura e do atendimento com o propósito de fornecer igual oportunidade de acesso aos serviços de saúde para indivíduos com maior poder aquisitivo busquem os serviços privados de saúde como forma de garantir o acesso quando necessário. De acordo com o princípio de equidade vertical os serviços de saúde deveriam ser distribuídos segundo a necessidade de cuidados com a saúde, independentemente das características socioeconômicas individuais (NERI; SOARES, 2009). Esse direito de equidade deveria se tornar mais evidente ainda com a criação do Sistema Único de Saúde (SUS).

O SUS é um dos maiores sistemas públicos de saúde do mundo, sendo o único a garantir assistência integral e completamente gratuita para a totalidade da população, inclusive aos pacientes portadores do HIV, sintomáticos ou não, aos pacientes renais crônicos e aos pacientes com câncer (REHEM, 2002). A política de saúde criada pelo SUS é frequentemente questionada, já que nem todos conseguem acesso a este serviço, tanto por falta de vagas o ou até mesmo devido a longa fila de espera que se forma, reflexo da burocracia e lentidão no atendimento, como também as deficiências estruturais e materiais das unidades de saúde brasileiras. Isto vai totalmente contra os objetivos do Sistema Único de Saúde, que quando criado, pretendia cobrir toda a população que não tinha acesso ao sistema ao de atendimento privado.

Para completar as dificuldades, no Brasil, os médicos enfrentam a difícil realidade de

deterioração do sistema público de saúde, alta dependência de organizações privadas do setor – como as de medicina de grupo e seguros -, pacientes com níveis crescentes de exigência e incremento da concorrência entre profissionais (URDAN, 2001). Esta exigência dos pacientes se da principalmente no âmbito da obtenção de informações precisas a respeito de seu tratamento e uma boa qualidade no atendimento.

Ainda dentre os problemas enfrentados na saúde pública está a assimetria entre os avanços da pesquisa em bancada e a velocidade da apropriação do conhecimento produzido em benefício da população. Geralmente a velocidade é bem mais baixa e os custos muito maiores. O processo de apropriação é também chamada de Avaliação Tecnológica, incluindo o desenvolvimento de produtos e processos, e com grande intensidade, a realização da testagem clínica de novos medicamentos, equipamentos, normas operacionais, aplicativos dentre outros. O núcleo mais importante da atividade de pesquisa clínica nos hospitais de ensino vincula-se à avaliação tecnológica e é sobre ela que deveria recair a maioria dos esforços de uma política de recuperação de atividade de pesquisa nessas instituições (GUIMARAES, 2004).

Apesar das dificuldades, vivemos hoje um processo consolidação do SUS. Embora com inúmeros avanços ocorridos nos últimos anos, existem ainda grandes desafios a serem enfrentados por todos os gestores envolvidos para tornar todos os princípios e diretrizes definidos para o sistema uma plena realidade para toda a população brasileira (CONASS, 2003). Além do mais, apesar de toda renovação que as ações de saúde têm sofrido, ainda é nítida as exorbitantes diferenças em critério de qualidade, atenção, métodos, estruturas, tecnologia e benefícios entre o atendimento público e o privado. Portanto há uma necessidade de mudar a prática dos serviços de saúde, superando os fatores que ocasionam seu baixo desempenho como a burocracia organizacional e buscando adotar estratégias que gerem maior eficácia e resolutividade como a inserção da tecnologia. Os critérios que conduzem a uma nova gestão hospitalar, considerados pilares essenciais da nova missão hospitalar, são a orientação ao usuário, avanço contínuo da procura por excelência e autoridade responsável no contexto de coordenação e integração em redes, acompanhados por elementos estratégicos, tais como participação social, transparência e responsabilidade no desenvolvimento das políticas públicas (SCARPI, 2004 apud LEMOS; ROCHA, 2011) . A eficácia das organizações de saúde também vai depender das relações que estabelecem pessoas, tecnologia, recursos e administração, para realizar a tarefa organizacional de prestação de serviços de saúde. Uma vez que a administração realiza o trabalho de relacionar todos estes (JUNQUEIRA, 1990).

O principal objetivo da tecnologia é aumentar a eficiência da atividade humana nas mais variadas esferas, e para isso a tecnologia produz os mais variados objetos para atender às necessidades da demanda, ou aperfeiçoa objetos tornando-os mais duráveis ao passo que melhora a produção ao reduzir o tempo ou o custo de certo objeto (KOERICH et al., 2006).

Então por que não a utilizar em prol da saúde também?

A utilização de novas tecnologias na assistência a saúde vem promovendo melhoria na qualidade e aumento da expectativa de vida no mundo (NÉRI et al., 2011). De fato, a tecnologia ultrapassou o processamento-padrão de dados para funções administrativas comuns em todas as organizações, tais como recursos humanos, folhas de pagamento, sistemas de contabilidade, entre outros, e agora desempenha um papel fundamental tanto no cuidado ao paciente, na interpretação do eletrocardiograma, como em escalas de trabalho, prescrição, relatório de resultados e sistemas de prevenção (PINOCHET, 2011b).

O conceito de tecnologia em saúde abrange qualquer intervenção que pode ser utilizada para promover saúde. Esse conceito não inclui somente as tecnologias que interagem diretamente com os pacientes, tais como medicamentos e equipamentos (tecnologias biomédicas) e procedimentos médicos como anamnese, técnicas cirúrgicas e normas técnicas de uso de equipamentos (que em conjunto com as tecnologias biomédicas), mas também os sistemas organizacionais e de suporte dentro dos quais os cuidados com saúde são oferecidos (AMORIM, 2010).

No âmbito da saúde, a tecnologia pode ser compreendida de forma ampliada: a tecnologia representada por máquinas e aparelhos (tecnologia dura), a tecnologia que engloba o saber profissional que pode ser estrutura e protocolizado (tecnologia leve-dura) e a tecnologia leve que se refere à cumplicidade, à responsabilização e ao vínculo manifestados na relação entre usuário e trabalhador de saúde (CORREA; CASATE, 2005). A tipologia que aplica à tecnologia reitera e renova o valor da Teoria do Processo de Trabalho na compreensão das práticas de saúde (AYRES, 2000). A atenção básica deve incluir diversas tecnologias de maneira adequada conforme as necessidades de saúde, que promova melhor qualidade de vida ao paciente sem prejuízo do atendimento.

A inserção da tecnologia na saúde cada vez mais crescente acelera e traz segurança nos processos organizacionais de arquivos, precisão de informações e busca complementar as limitações humana. Os sistemas de informação podem participar e influenciar focando inclusive os atendimentos, diminuindo o nível de risco de erro e acelerar os procedimentos. Medidas de auxílio tecnológico como suporte e apoio ao sistema de saúde garantem praticidade, organização e efetividade. O município de Picos, no estado do Piauí, por exemplo, demonstra enorme carência nessa área, tornando-se evidente a possibilidade de grandes benefícios.

De acordo com Benito e Licheski (2009):

> Os sistemas de informação que disponibilizam as informações de forma organizada e de fácil acessibilidade tornam-se recursos tecnológicos capazes de potencializar a busca, o acesso e principalmente a efetividade das ações dos profissionais de saúde, tornando-se uma ferramenta de apoio às atividades, auxiliando na tomada de decisão e aquisição de conhecimento, e como resultado, tem-se profissionais mais capacitados, criativos, capazes de mudar a realidade e melhorar o atendimento dos serviços de saúde através de ações de inovação de processos de saúde potencializando assim suas competências pessoais e coletivas no trabalho em saúde.

De acordo com Carvalho e Eduardo (1998), é possível classificar os Sistemas de Informação em saúde, conforme sua natureza, em:

- Sistemas de Informações Estatístico-epidemiológicos;
- Sistemas de Informações Clínicos;
- Sistemas de Informações Administrativos.

Os Sistemas de Informação Estatístico-epidemiológicos são aqueles que consistem em dados sobre expectativa de vida da população e seus critérios dependiosos ou de seus critérios da quantidade dos usuários dos serviços, dos aspectos geográficos, civis e de renda diante do bem estar da população. Ja os Sistemas de Informações Clínicos, segundo Carvalho e Eduardo (1998), referem-se aos dados clínicos sobre o paciente, desde sua identificação, problemas de saúde relatados, diagnóstico médico, até exames clínicos, laboratoriais, radiológicos, gráficos, procedimentos cirúrgicos realizados ou medicamentos prescritos, dentre outros. Enquanto os Sistemas de Informações Administrativos não são especificamente da área da saúde e referem-se à administração dos materiais, equipamento, etc.

Segundo TIC (2014), os potenciais benefícios da aplicação das TIC (Tecnologias da Informação e Comunicação) podem ser agrupados em algumas categoriais complementares entre si:

- Aumento da qualidade dos tratamentos e eficiência dos serviços de saúde;
- Redução dos custos de operação de serviços clínicos;
- Redução de custos administrativos;
- Abertura de possibilidades para novas formas de tratamento.

O impacto das novas tecnologias não é de imediato, demora-se um tempo para os indivíduos incorporarem os avanços e aprendam como utilizá-las. Não basta adquirir máquinas e equipamento é preciso saber usar para reproduzir novas condições de aprendizagem e estilo de vida. O crescente avanço científico e tecnológico na área da saúde cria a necessidade dos profissionais dessa área buscarem intensiva atualização (KOERICH et al., 2006). Um treinamento qualificado, a respeito das novas tecnologias, podem proporcionar avanços muito significativos. A tecnologia sofre a ação humana, logo convive em perfeita simbiose com o ser humano

influenciando nas relações sociais tornando a vida cotidiana mais simples, auxiliando na realização de tarefas. Para que este processo aconteça é preciso domínio e aprendizado tecnológico conforme afirma Kenski (2011).

Entretanto, as ações de saúde devem ser também direcionadas pelos princípios da humanização do cuidado, esta como um conjunto de conhecimentos, processos e métodos usados como ramo de atividade na área de saúde, tem a oferta de tecnologias e dispositivos para configuração e fortalecimento entre os diversos setores da saúde e da comunidade. Esta humanização trata da necessidade de enfatizar os valores humanos e lidar com a saúde com zelo com paciente demonstrando respeito a sua individualidade, cultura, necessidade.

Como recursos tecnológicos, acesso, acolhimento e vínculo representam uma relação estabelecida entre trabalhadores e usuários para que as ações de saúde sejam mais acolhedoras, ágeis e resolutivas (COELHO; JORGE, 2009). É importante ressaltar que a maioria dos textos das décadas de 50, 60 e 70 confronta humanização e tecnologia, enfatizando a humanização como possibilidade de resgate de valores caritativos, fazendo-se presente o "interesse humano" pelo próximo (CORREA; CASATE, 2005).

Esse discurso é limitante ao idealizar a humanização como "aquilo que é bom" e repudiar a tecnologia como impeditivo à possibilidade de humanização. "Ser humano", entretanto, não é algo idealizado. Como humanos podemos constituir ações "humanizantes" que consideram o outro em seus direitos, em sua singularidade e integralidade; enfim, em sua dignidade e, ao mesmo tempo, somos capazes também de constituir ações "desumanizantes" que "coisificam" o outro ou nós mesmos (CORREA; CASATE, 2005).

Além do mais, os sistemas de informação em saúde são, cada vez mais, um instrumento de importância crítica, para o desenvolvimento de estratégias informacionais, na área da saúde (ESPANHA, 2010). Castells (apud ESPANHA, 2010) salienta a importância fulcral das Tecnologias da Informação e Comunicação ou TIC, nas sociedades contemporâneas, enfatizando as alterações de organização social e nas estruturas de base das sociedades, para além das óbvias modificações tecnológicas.

A partir destas informações, pode-se entender que a prática da medicina enfrenta os mais diversos desafios na sua prática, dentre eles: a qualidade frente ao tempo oferecido a cada atendimento. Apesar dos avanços técnico-científicos disponibilizados para o atendimento dos indivíduos doentes e o maior contingente de profissionais, a ocorrência de eventos iatrogênicos relacionados a assistência de saúde e mais particularmente a atuação médica, data de longo tempo sendo positivo novas pesquisas e aplicações nesse meio (PADILHA, 2001).

Dir-se-á que no mundo atual, com a medicina em grande parte socializada (pré-paga) estatal ou não, com o profissional de saúde habitualmente mal ressarcido (não dispondo de tempo e de espaço estrutural efetivo para dedicar-se seriamente a cada um de seus pacientes), a

criação e preservação dessa ligação efetiva entre o profissional de saúde e o cliente é tão irreal quanto a expectativa de "respeito" e bem estar da OMS. Admite-se que assim seja, pelo menos em parte (SEGRE; FERRAZ, 1997).

Concluindo, TIC (2014) afirma que nos últimos anos, as TICs vêm desempenhando um papel cada vez maior de facilitadoras fundamentais na reforma dos sistemas de saúde, para melhorar o acesso a serviços de saúde, qualidade do atendimento e a produtividade do sistema de saúde. Completando que mudanças, contudo, envolvem desafios. O desafio está em promover e/ou adequar a inserção da tecnologia na prática em saúde, de forma a contemplar as demandas sociais da contemporaneidade e refletir sobre as questões éticas que permeiam a utilização das tecnologias frente a intersubjetividade viva no momento assistencial que extrapola o tecnológico (KOERICH et al., 2006).

Existe um órgão, no Brasil, que é responsável por coletar informações a respeito da saúde da população. Além disto, promove ainda o desenvolvimento de soluções em tecnologia que apoiam as deciões estratégicas a longo prazo. Este órgão é o DATASUS.

2.1 DATASUS (Departamento de Informática do Sistema Único de Saúde)

O DATASUS é um órgão pertencente à Secretaria Executiva do Ministério da Saúde cujo funcionamento relaciona-se diretamente à ação produtora, receptora, ordenadora e disseminadora de informações (FERRAZ, 2009). Os dados sob a administração deste orgão são usados pelo Ministério para criar uma imagem nacional e consolidada dos serviços de saúde, tais como o número de médicos e enfermeiros que trabalham em cada um dos 27 estados do país, taxas de nascimento e mortalidade, incidência de doenças como câncer e mal de Parkinson, e informações financeiras de apoio, em âmbito estadual, federal e local, à prestação de serviços de saúde por meio de autoridades brasileiras municipais (DATASUS, 2013).

Ainda segundo DATASUS (2013), as vantagens do DATASUS são:

- Informações precisas e confiáveis geradas para os processos decisórios do Ministério da Saúde do Brasil;
- Tomada de decisão mais ágil e oportuna;
- Redução de erros associados à má qualidade dos dados;
- Automação da qualidade dos dados e do processo de limpeza dos dados;
- Padronização e monitoramento do processo de qualidade de dados, com o ciclo de qualidade de dados executado no estágio de entrada dos dados nos sistemas de origem;

- Reutilização do processo de qualidade de dados para vários sistemas do DATASUS;
- Aumento da eficiência na pesquisa de qualidade e análise de dados;
- Otimização do uso dos recursos com a redução do retrabalho resultante de imprecisões nos dados.

Merecem destaque dois programas desenvolvidos para o setor público de saúde pela DATASUS, por serem públicos, gratuitos e por estarem disponíveis na Internet (CARVALHO; EDUARDO, 1998): o TABWIN (Tab para Windows) e o HOSPUB.

2.1.1 TABWIN

O programa TAB para Windows – TabWin - foi desenvolvido pelo Datasus – Departamento de Informática do SUS, com a finalidade de permitir às equipes técnicas do Ministério da Saúde, das Secretarias Estaduais de Saúde e das Secretarias Municipais de Saúde a realização de tabulações rápidas sobre os arquivos DBF que se constituem nos componentes básicos dos sistemas de informações do SUS - Sistema Único de Saúde (FEMPA, 2014). Segundo Carvalho e Eduardo (1998), o TABWIN permite tabular dados disponíveis na Internet, bem como utilizar arquivos de outras bases de dados (DBF, CNV e TXT). Permite, também, realizar operações aritméticas e estatísticas nos dados de tabelas geradas ou importadas. Carvalho e Eduardo (1998) completam que este programa é excelente para trabalhar dados de morbidade, mortalidade, produção, financeiros e tantos outros, os quais podem rapidamente configurar mapas e gráficos de boa qualidade por regiões e municípios de todo o Brasil.

2.1.2 HOSPUB

O Hospub é o Sistema Integrado de Informatização de Ambiente Hospitalar que fornece soluções de Tecnologia da Informação para gerenciamento, gestão e controle social do SUS em unidades hospitalares (HOSPUB, 2011). Ele disponibiliza sistemas para atender às necessidades de informações no gerenciamento hospitalar e é composto pelos seguintes módulos (CARVALHO; EDUARDO, 1998):

- SIGUE – Sistema de Gerenciamento de Unidades de Emergência;
- SIGHO – Sistema de Gerenciamento Hospitalar
- SIGAE – Sistema de Gerenciamento de Unidades Ambulatoriais Especializadas;
- SIADT – Sistema de Apoio à Diagnose e Terapia (laboratórios);

- SICEC – Sistema de Gerenciamento de Centro Cirúrgico;
- SINAT – Sistema de Gerenciamento Perinatal.

2.2 Considerações Finais

É notável que o sistema de saúde brasileiro necessita de uma reformulação e reestruturação em sua organização. A adoção de tecnologias bem como sistemas de informações na saúde pública podem ser de grande importância na luta contra o seu baixo rendimento e altos indíces de reclamações. Isto já vem ocorrendo com a criação do DATASUS, que se torna um importante pilar na busca por melhorias nesta área. Porém, apenas ele não é suficiente. Este trabalho então visa utilizar de tecnologias modernas e de fácil acesso para desenvolver um protótipo de Sistema de Informação Clínico, com o objetivo de se tornar uma possível solução para alguns problemas encontrados tanto no atendimento a pacientes quanto na armazenagem de dados deste paciente, de forma inovadora. Através da utilização não apenas de computadores comuns, mas também de tecnologia móvel, para garantir flexibilidade e agilidade em determinados procedimentos.

3 Métodos e Tecnologias

Neste capítulo serão abordadas as metodologias de desenvolvimento e tecnologias que foram utilizadas no decorrer do projeto como o *framework* Grails, Android, *Web Services* e técnicas de Interface Humano Computador. Inclui ainda uma breve descrição das ferramentas que foram empregadas para auxiliar o potencializar o desenvolvimento do projeto.

3.1 Metodologia de Desenvolvimento

Com o aumento da concorrência entre as empresas e com a instauração de um mercado mais exigente, no que se refere à qualidade do *software*, notou-se a necessidade de se utilizarem metodologias de desenvolvimento mais flexíveis e ágeis, para diminuir o tempo de desenvolvimento e, ao mesmo tempo, não afetar a qualidade do produto (SOUSA; SOUZA, 2012).

Metodologias de desenvolvimento de *Software* (ou Processo de *Software* é o conjunto de atividades, ordenadas ou não, com a finalidade de produzir um *software* de qualidade (LUDVIG; REINERT, 2007). Uma metodologia de desenvolvimento constitui-se de uma abordagem organizada para se atingir um objetivo, possível por meio do cumprimento de um conjunto de procedimentos preestabelecidos (PRODATER, 2011). Estas metodologias de desenvolvimento são divididas basicamente em duas categorias: as metodologias de desenvolvimento tradicionais ou "pesadas" e as metodologias de desenvolvimento ágil ou "leves". Enquanto as tradicionais prezam por uma quantidade excessiva de documentação as ágeis prezam por ter o *software* funcionando com o mínimo de documentação necessária (LUDVIG; REINERT, 2007).

3.1.1 Metodologia de Desenvolvimento Tradicional

Como já citado, Metodologia de Desenvolvimento é o conjunto de práticas recomendadas para o desenvolvimento de *softwares* (UTIDA, 2012). Utida (2012) completa ainda que as metodologias tradicionais, também conhecidas como "pesadas", têm como característica marcante sua divisão em etapas ou fases. Segundo Soares (apud LUDVIG; REINERT, 2007), essas metodologias surgiram em um contexto de desenvolvimento de *software* muito diferente do atual, baseado apenas em um *mainframe* e terminais burros. Ludvig e Reinert (2007) completam ainda que na época, o custo de fazer alterações era muito alto, uma vez que o acesso aos computadores era limitado e não existiam modernas ferramentas de apoio ao desenvolvimento do *software*, como depuradores e analisadores de código. Devido a isto, o *software* era

anteriormente bem planejado antes de ser implementado.

Os principais modelos de desenvolvimento tradicional são: cascata e espiral. O modelo em cascata, segundo CECOMP (2012), é o modelo mais antigo e o mais amplamente usado. Ele completa ainda que o modelo em cascata é modelado em função do ciclo da engenharia convencional e que o resultado de cada uma de suas fases constitui a entrada de uma outra fase. Já o modelo espiral, segundo Pressman (apud UTIDA, 2012), é um modelo evolucionário de processo de *software* que combina a natureza iterativa da prototipagem com os aspectos controlados e sistemáticos do modelo cascata. Este tipo de modelo tem a vantagem de responder facilmente a erros em iterações anteriores, ou a modificações no projeto durante o processo de desenvolvimento (OLIVEIRA, 2013a). Um fator muito importante no modelo espiral é que cada ciclo é completado com uma revisão, na presença de pessoas chave para o produto em desenvolvimento (ALVES; VANALLE, 2014).

3.1.2 Metodologia de Desenvolvimento Ágil

O termo "Metodologias Ágeis" tornou-se popular em 2001 quando um grupo de dezessete especialistas em processos de desenvolvimento de *software* decidiu se reunir nos EUA, para discutir maneiras de melhorar o desempenho de seus projetos (LIBARDI; BARBOSA, 2010). Essa abordagem é um conjunto de novas ideias, velhas ideias e velhas ideias modificadas, e enfatiza, segundo Cohen et al. (apud SILVA, 2009):

- Uma maior colaboração entre programadores e especialistas do domínio;
- Comunicação "cara-a-cara" (como sendo mais eficiente do que comunicação documentada);
- Entregas constantes de novas funcionalidades com valor de negócio;
- Equipes auto-organizadas;
- Maneiras de adaptar o código e a equipe a novos requisitos que poderiam causar grande confusão.

A partir disto, em fevereiro de 2011, foi assinado por 17 desenvolvedores o Manifesto Ágil, que, segundo Bassi (2006), pautava-se nas seguintes premissas:

- Indivíduos e interações são mais importantes que processos e ferramentas;
- *Software* funcionando é mais importante do que documentação completa e detalhada;
- Colaboração com o cliente é mais importante do que negociação de contratos;

- Adaptação a mudanças é mais importante do que seguir o plano inicial.

O Manifesto Ágil é uma declaração sobre os princípios que servem como base para o desenvolvimento ágil de *software* (SILVA, 2009). Ainda segundo Silva (2009), nomeando a si mesmos como "*Agile Alliance*", esse grupo de inovadores em desenvolvimento de *software* concordaram em chamar esses novos valores de "Manifesto para o Desenvolvimento Ágil de *Software*" (em inglês: "*Manifesto for Agile Software Development*").

Extreme Programming

Segundo Beck (apud OLIVEIRA, 2013a), *Extreme Programming* (ou simplesmente XP) é uma metodologia ágil para equipes pequenas e médias desenvolvendo *software* com requisitos vagos e em constante mudança. Ja Bassi (2006), afirma que XP é uma metodologia ágil de desenvolvimento de *software* que reúne boas práticas de programação e de gerenciamento de projetos com o objetivo de produzir *software* de alta qualidade.

As regras, práticas e valores da XP proporcionam um agradável ambiente de desenvolvimento de *software* para os seus seguidores, que são conduzidos por quatro valores: comunicação, simplicidade, *feedback* e coragem (LIBARDI; BARBOSA, 2010). O princípio da comunicação visa manter o melhor relacionamento possível entre o cliente e desenvolvedores (UTIDA, 2012). Completando ainda que isto aplica-se tanto para o relacionamento cliente/desenvolvedor como equipe/gerente. O segundo valor da XP, simplicidade, segundo Oliveira (2013a), traz uma proposta de fazer qualquer módulo de sistema da forma mais simples que atenda aos requisitos, sem antecipar necessidades futuras e caso este módulo necessite de ajustes, ele deve ser alterado com este objetivo. O princípio de *feedback*, significa que é importante ter um sistema rodando a qualquer hora (NETO, 2009). Neto (2009) completa ainda que isso dá para o desenvolvedor a informação fidedigna sobre seu funcionamento. Vale lembrar que este *feedback* é direcionado ao estado de desenvolvimento. Por último, o princípio de coragem esta relacionado ao fato de que equipes prósperas de desenvolvimento de *software* precisam constantemente trabalhar na extremidade do caos, eles precisam agir rapidamente sem perder o controle (NETO, 2009), além de principalmente ter coragem para adotar os três outros valores.

Scrum

Scrum trata-se de uma metodologia ágil de desenvolvimento de *software*, que tem como principais características ser um processo empírico, iterativo e incremental (LUDVIG; REINERT, 2007). Scrum trabalha com a complexidade de desenvolvimento de *softwares* através do controle de inspeção, adaptação e visibilidade de requisitos de um processo empírico fazendo uso de uma série de regras e práticas, onde controle não significa controle para criar o que foi

previsto e sim controlar o processo para orientar o trabalho em direção a um produto com o maior valor agregado possível (LIBARDI; BARBOSA, 2010).

Segundo Schwaber (2011), três pilares apoiam a implementação de controle do *scrum*:

- Transparência: aspectos significativos do processo devem estar visíveis aos responsáveis pelos resultados;
- Inspeção: os usuários do *scrum* devem, frequentemente, inspecionar os artefatos e o progresso em direção ao objetivo para detectar indesejáveis variações;
- Adaptação: o produto final deve ser adaptado caso seja detectado que ele está fora dos padrões aceitáveis.

Segundo Ferreira et al. (apud BISSI, 2007), as principais características do *SCRUM* são:

- é um processo ágil para gerenciar e controlar o desenvolvimento de projetos;
- é um processo que controla o caos resultante de necessidades e interesses conflitantes;
- é uma forma de aumentar comunicação e maximizar a cooperação;
- é uma forma de detectar e remover qualquer impedimento que atrapalhe o desenvolvimento de um produto;
- é escalável desde projetos pequenos até grandes projetos em toda empresa.

Existe um vocabulário específico para trabalhar com *scrum*, como por exemplo as palavras: "*backlog*", que são todas as funcionalidades do projeto a serem desenvolvidas, "*sprint*", que é um determinado período de tempo em que uma ou mais funcionalidades são desenvolvidas, "*sprint backlog*", que é um conjunto de funcionalidades que devem ser desenvolvidas em um *spring*, de modo que possibilite a apresentação ao cliente e etc.

Segundo Soares (2014), o ciclo de vida da *scrum* é baseado em três fases principais, divididas em sub-fases: pré-planejamento, desenvolvimento e pós-planejamento. Soares (2014) completa ainda que a idéia principal da *Scrum* é que o desenvolvimento de *softwares* envolve muitas variáveis técnicas e do ambiente, como requisitos, recursos e tecnologia, que podem mudar durante o processo tornando o processo de desenvolvimento imprevisível e complexo, requerendo flexibilidade para acompanhar as mudanças.

3.2 O *Framework* Grails (Groovy on Rails)

Para melhor entender o que é um *framework*, Müller (2008) diz que *framework* é uma abstração que une códigos comuns entre vários projetos de *software* provendo uma funciona-

lidade genérica. Em outras palavras, *framework* é uma abstração de projeto e uma implementação de uma aplicação em um determinado domínio de problema. A presença ou não de um *framework* influência diretamente no desenvolvimento de uma aplicação e a principal vantagem de se utilizar esta tecnologia é o alto nível de reutilização de códigos.

Criado em 2006, o Grails é um *framework* de desenvolvimento Java *Web* de última geração que baseia-se nas melhores ferramentas, técnicas de desenvolvimento *web* e tecnologias de *frameworks* Java já existentes, combinando-os com o poder e inovação do desenvolvimento da linguagem dinâmica (SMITH, 2009). Quando o Grails foi lançado, já existiam diversos *frameworks* ativos no mercado, porém Kumar (2013) diz que o Grails não reinventou a roda, mas em vez disso tem aproveitado dos *frameworks* já testados e confiáveis tais como *Spring*, *Hibernate*, *SiteMesh* e outras bibliotecas de código aberto já populares no mundo empresarial Java.

O Groovy on Rails é de código aberto, ou seja, todo seu código fonte é disponibilizado a todos os usuários, utiliza a linguagem de programação Groovy, que é uma linguagem dinâmica para a plataforma Java e é baseado em convenções, que implica na retirada da carga de serviços extras que deixam o desenvolvedor sobrecarregado, como por exemplo detalhes de configuração de bancos de dados, os quais na maioria das vezes são cansativos e repetitivos. Através da (Figura 1) é possível observar os componentes que juntos, compõem o *framework* Grails.

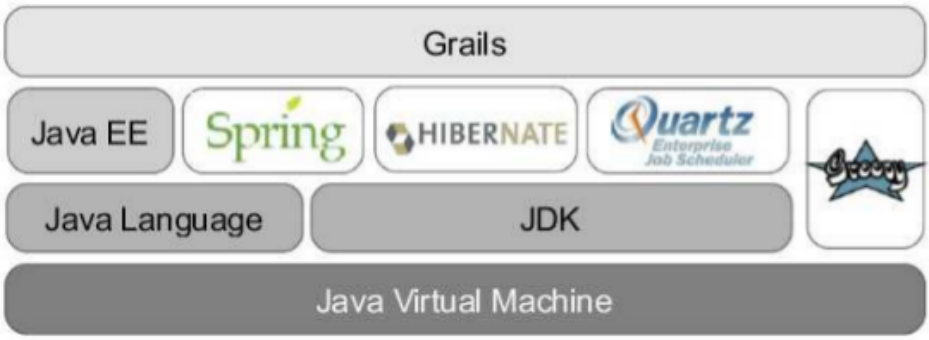

Figura 1 *– Arquitetura Grails.*

Disponível em: <http://corinnekrych.github.io/confess/images/grails-evolution.bmp>

3.2.1 Características

Desenvolvimento Ágil

Atualmente existe uma grande demanda para desenvolvimento de *softwares* complexos, sistemas e aplicações heterogêneas, porém não há desenvolvedores suficientes para suprir tal demanda. Neste contexto, o Grails é uma ferramenta muito poderosa, pois nela estão compilados alguns dos *frameworks* mais usados para desenvolvimento de aplicações *web*, e como ela

trabalha sobre a linguagem Groovy que é uma forma facilitada da linguagem Java, contém os mesmos recursos, mas com uma implementação simplificada, fazendo com que o processo de desenvolvimento ganhe produtividade e agilidade (NASCIMENTO, 2014). Um ponto importante a respeito de desenvolvimento ágil no Grails é a utilização do *scaffolding*.

O Grails possui uma propriedade chamada "*scaffold*" que permite que uma aplicação inteira seja gerada através das classes de domínio, incluindo as visões necessárias e as *actions* de controle para operações CRUD (quatro operações básicas de um objeto: *create, read, update e delete*) (FRANCO, 2011). A forma mais fácil de obter o *scaffold* é ativando o *scaffold* dinâmico. Para isso basta incluir a propriedade "*scaffold = true*" no controlador e pronto! A aplicação é gerada contendo todas as *actions* e *views* necessárias para um CRUD. Talvez neste momento, este conceito não esteja tão claro, porém, nos tópicos seguintes, com a definição de *domain class*, *controllers* e *views*, será possível entender o quão admirável é o *scaffold* dinâmico.

Padrão MVC (Model, View, Controller)

MVC é um conceito introduzido pelo inventor do *Smalltalk* (Trygve Reenskaug) para encapsular os dados (*model*), juntamente com o seu processamento, isolando-o do processo de manipulação (*controller*) e apresentação(*view*) (DEACON, 2009 apud 'UYUN; MA'ARIF, 2010). Segundo seu criador, o padrão MVC foi concebido em 1978 como a solução de projeto para um problema particular (REENSKAUG, 2003). Reenskaug (1979c) diz que a finalidade essencial do MVC é preencher a lacuna entre o modelo mental do usuário e o modelo digital que existe no computador, e ainda completa que a solução ideal MVC suporta a ilusão de que o usuário vê e manipula as informações de domínio diretamente. Este padrão é dividido em três camadas: *model, view e controller*. Através da Figura 2, pode-se observar uma ilustração a respeito do padrão MVC, em que o usuário tem a ilusão de que está manipulando diretamente as informações, quando na verdade, os controladores é que executam esta tarefa.

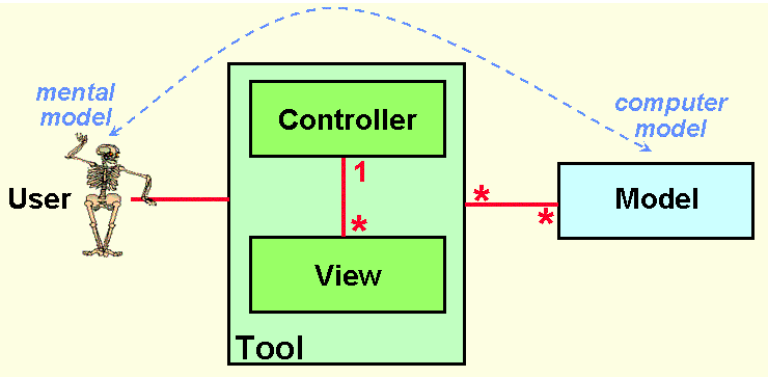

Figura 2 *– Abstração Padrão MVC.*

Disponível em: <http://heim.ifi.uio.no/ trygver/themes/mvc/MVC-2006.gif>

Para melhor compreensão de MVC Caetano (2012) diz que o o objetivo do MVC é facilitar o desenvolvimento, manutenção e reaproveitamento de código. Com o trabalho do MVC em camadas, é possível que o desenvolvedor altere, por exemplo o *layout*(camada *view*) de uma aplicação sem que a manipulação de dados(*model*) ou lógica de negócio(*controller*) seja afetada.

A definição técnica de arquitetura MVC é dividida em três camadas (BURBECK, 2009 apud 'UYUN; MA'ARIF, 2010):

- *Model*: contém as classes que descrevem os objetos da aplicação (JONATHAN, 2011). Segundo (REENSKAUG, 1979b), modelos representam conhecimento, completando que ele pode ser um único objeto ou uma estrutura de objetos.
- *Controller*: é através dos controladores que os objetos e seus atributos podem ser acessados. Cada objeto controlador é uma espécie de "gerente" da aplicação, encarregado de intermediar as solicitações dos usuários feitas através da interface com o usuário (vista) (JONATHAN, 2011). É a camada de controle que determina o fluxo da apresentação servindo como uma camada intermediária entre a camada de apresentação e a lógica (REENSKAUG, 1979a).
- *View*: a camada de visualização apenas exibe a informação, não se importando como ou de onde ela tenha sido gerada. É a *view* que serve de interface de comunicação entre o usuário e a aplicação. Através dela que o usuário pode submeter dados e receber seus resultados. A *view* associa-se a um modelo e mostra seu conteúdo na tela ('UYUN; MA'ARIF, 2010).

As vantagens do MVC são o fácil gerenciamento, manutenção e reaproveitamento de código; independências entre as camadas que o constituem, e a possibilidade de construção de inúmeras *interfaces* utilizando uma única *interface* padrão. As desvantagens são a respeito de utilização desse padrão em aplicações de pequeno porte, pois as suas configurações padrão podem transformar uma pequena aplicação em uma aplicação complexa. Outra desvantagem é que para a construção de um projeto seguindo este padrão se torna necessário uma análise mais detalhada a respeito do sistema a ser desenvolvido. Por fim, trabalhar com o MVC não é tão simples quanto parece. É necessário mão de obra qualificada com conhecimento especializado sobre o MVC.

Linguagem Groovy

James Strachan, em agosto de 2003, mostrou pela primeira vez em seu *blog* o projeto que se tornaria a Linguagem Groovy. A partir disto várias outras versões foram lançadas, até que

chegasse à versão 1.0, em janeiro de 2007. Esta versão 1.0 ainda se encontrava em versão *beta*, até que em dezembro de 2007 foi lançada então a versão final do Groovy (Groovy 1.1). James Strachan, deixou bem claro em seu *blog* que a sua idéia inicial era fazer uma pequena linguagem dinâmica, que fosse compilada diretamente em classes Java e que tivesse toda a produtividade e elegância encontrada em *Ruby* e *Python*, porém, que permitisse reusar, estender, implementar e testar código Java já existente.

Konig et al. (2013) afirma que Groovy é uma linguagem dinâmica para a plataforma Java com muitas características inspiradas por linguagens como *Python*, *Ruby* e *Smaltalk*, tornando os recursos destas linguagens disponíveis para os desenvolvedores utilizando uma sintaxe *Java-like*, ou seja, uma sintaxe semelhante ao Java original. McWhirter e Champeau (2014) completam que o Groovy roda sobre a *Java Virtual Machine*, e torna modernos recursos de programação disponíveis para os desenvolvedores Java com curva de aprendizagem quase zero, aumentando ainda mais a produtividade. Além do mais, o Groovy é uma linguagem de código aberto.

Para Doederlein (apud REZENDE, 2011), a semântica da linguagem Groovy foi planejada para ser familiar aos programadores Java, pela sua similaridade com os fontes escritos em *Java*, reduzindo assim a curva de aprendizagem, porém há algumas diferenças entre as linguagens. As principais diferenças entre Groovy e Java são:

- Ponto e vírgula é opcional: caso um único comando esteja em uma linha, em Groovy, não é necessária a utilização de ponto e vírgula, porém, caso existam dois comandos na mesma linha, a vírgula torna-se necessária;

- Verdade em Groovy: o Groovy utiliza o mesmo padrão de conceito de verdade da linguagem Java, entretanto, além disso, o Groovy considera verdade qualquer variável diferente de nulo;

- Return: em Groovy, o último comando de um determinado método é considerado como o seu retorno. Enquanto em Java é necessário a utilização da palavra-chave "*return*";

- Objeto: em Groovy, tudo é considerado objeto, enquanto em Java existem tipos primitivos e não primitivos;

- Igualdade: na linguagem Java, o sinal de igualdade "==" é utilizado apenas em tipos primitivos, enquanto o "equals" é utilizado para objetos. Em Groovy tudo pode ser comparado utilizando o sinal de igualdade "==";

- Tipagem: em Groovy não é necessário declarar o tipo de uma variável, pois esta linguagem utiliza o "*duck typing*", que será abordada mais a frente;

- Beans: o Groovy gera dinamicamente os métodos chamados assessores (por exemplo os *getters* e *setters*), ganhando então em produtividade.

É interessante ressaltar, a respeito dinâmicidade do Groovy, a utilização do *Duck Typing* para definição de tipos de variáveis. Pode-se abstrair o *Duck Typing* da seguinte forma: se age como um pato, é um pato. Ou seja, se uma determinada variável age como um determinado tipo (*String*, número), então este tipo será atribuido dinamicamente a esta variável.

Closures

Oak (2006) define *closures* como blocos reutilizáveis de código que possuem uma ou mais instruções entre chaves, podem ser passados como uma variável e só são executados quando são chamados. Ja Weissmann (2009a), utiliza a seguinte analogia para descrever as *closures*: "Podemos pensar em nossas variáveis como envelopes que armazenam informações. Ao abrir este envelope, encontramos uma *string*, um valor numérico, um objeto, enfim: um dado. Uma *closures* por sua vez consiste em um tipo diferente de variável. Seguindo a metáfora, nosso envelope conteria não uma informação, mas sim um conjunto de instruções, resumindo: código executável". Na Figura 3, podemos observar um exemplo de uma *closure*.

```
def qualquer closure = {
    println "Imprimir qualquer coisa aqui"
}
```

Figura 3 *– Exemplo de uma closure.*

Então qual a diferença entre uma *closure* e um método comum? Sena (2010), afirma que a principal diferença entre uma *closure* e um método é que a *closure* não necessita de uma classe ou um nome de método.

A respeito da utilização em Grails, é através das *closures* que são definidas as ações dos controladores. Deste modo, as *closures* são importantíssimas na utilização deste *framework*. Além disto, a maioria dos arquivos de configuração do Grails são definidas dentro do arquivos de configuração através de *closures*. Nos próximos tópicos as ações dos controladores serão abordadas com maior especificidade.

SiteMesh

O Grails utiliza uma biblioteca de *templates* bastante popular, o *SiteMesh* (DAVIS; RUDOLPH, 2010). Ele é um *framework* de renderização de *layouts* robusto e estável (KOLLIKER; SCHMUTZ, 2008) e funciona como um decorador de páginas (SMITH, 2009). Através da (Figura 4), pode-se observar o funcionamento do *SiteMesh*.

Disponível em: <http://docs.huihoo.com/sitemesh/sitemesh-aop-for-web-pages.pdf>

O *SiteMesh* funciona da seguinte maneira: primeiramente ele intercepa a requisição, posteriormente a aplicação gera apenas o conteúdo da página HTML. O *SiteMesh* então analisa o conteúdo e aplica a decoração da página. Por fim a página já decorada é retornada para o navegador.

Como pode-se observar, através do *SiteMesh* é possível definir um determinado *layout*, e reutilizá-lo em várias outras páginas. Manter um certo padrão nas diferentes páginas de uma aplicação é importante para construção de um *layout* de navegação e interface consistente.

Uma das características mais poderosas e sofisticadas do *SiteMesh* é mesclar elementos de uma determinada páginas com o seu decorador. Um decorador é um modelo de página que possui espaços em reservados para o cabeçalho e o corpo da página (CHAMBERS, 2005). Para melhor entender os decoradores, vamos a um exemplo através da (Figura 5).

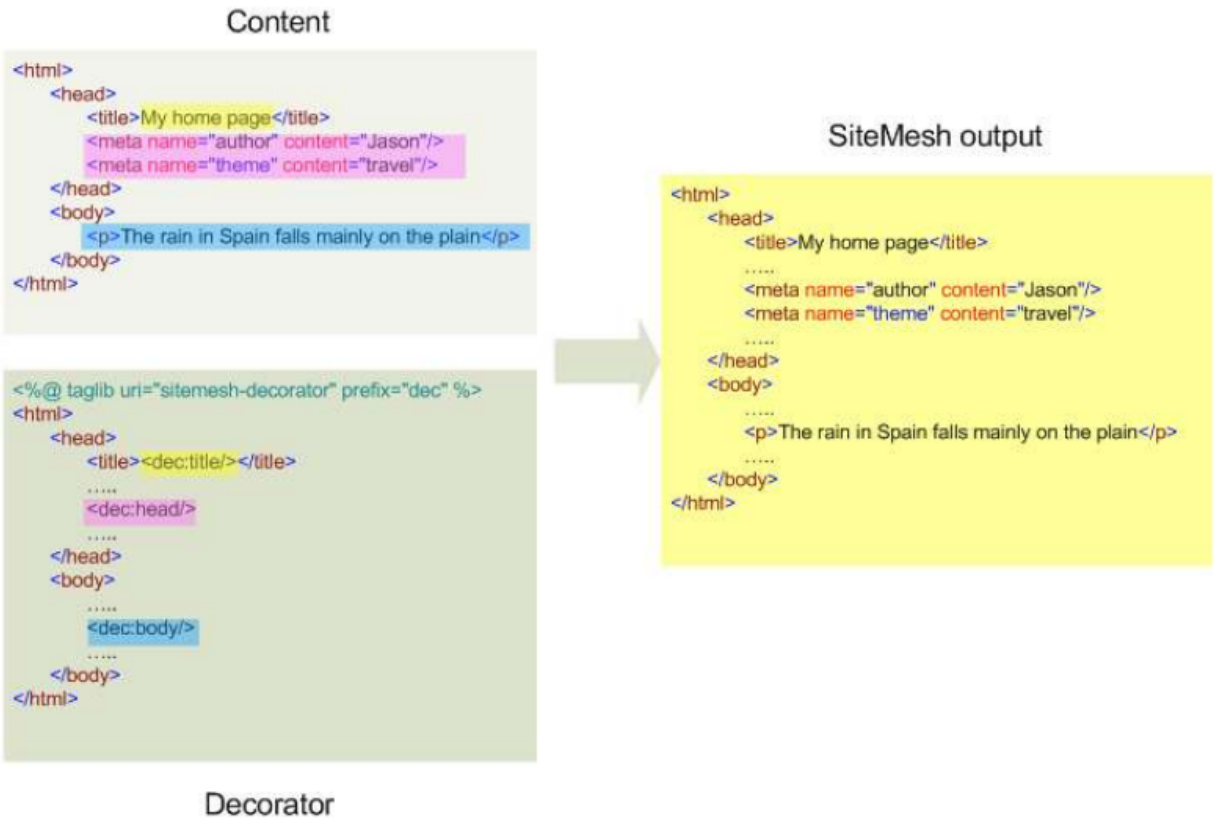

Figura 5 *– Exemplo de SiteMesh.*

Pela Figura 5 é possível observar que o decorador(*decorator*) foi mesclado ao conteúdo (*content*) gerando uma saída única. É importante observar que as *tags* “<dec:head/>”, “<dec:tittle/:>” e “<dec:body/>” da página *decorator* foram substituidas respectivamente pelo

cabeçalho, título e conteúdo da página *contente*, gerando então uma página final, que é uma mescla entre estas duas páginas, a qual será visualizada no navegador.

3.2.2 Sistema de Diretório Grails

O Grails usa "convenção sobre configuração", ou seja, o nome e a localização de arquivos são usados ao invés de configuração explícita (FRANCO, 2011). Logo, ao criar uma aplicação, o Grails estrutura os pacotes(pastas) de forma que organiza a aplicação seguindo o conceito de MVC. Controladores, Classes do domínio e Visualizações, são as principais pastas que são utilizadas na aplicação (NASCIMENTO, 2014). Na Figura 6 pode-se observar os principais diretórios gerados ao criar uma aplicação Grails.

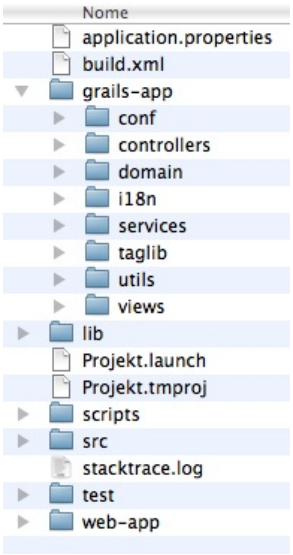

Figura 6 *– Diretórios criados pelo Grails.*

Pela Figura 6, observa-se que o Grails nomeia suas pastas de forma intuitiva, facilitando então a compreensão do seu sistema de diretórios. Para compreender o Grails, é importante conhecer a sua estrutura de diretórios:

- grails-app: é nesta pasta que está contido o código da aplicação em si. Grande maioria (senão maioria) dos arquivos estará localizado em um dos subdiretórios desta pasta;
- grails-app/conf: é nesta pasta que estão localizados os arquivos de configuração da aplicação, como configuração do banco de dados, mapeamento de url's e etc;
- grails-app/controllers: pasta onde estão armazenados os controladores do sistema;
- grails-app/domain: contém as classes de domínio da aplicação, ou seja, aquelas que serão persistidas no banco de dados;

- grails-app/i18n: pasta onde estão localizados recursos de internacionalização da aplicação. Recursos de internacionalização estão ligados a mensagens padrões emitidas pelo sistema. Através dos recursos de internacionalização é possível padronizar diferentes mensagens em diferentes idiomas;

- grails-app/services: pasta onde estão localizados os serviços da aplicação. Não é aconselhável que toda a lógica de negócio esteja presente nas classes de domínio. Sendo assim, é possível criar serviços que são gerenciados pelo Spring e podem ser utilizados para armazenar a lógica de negócio da aplicação;

- grails-app/taglib: pasta onde estão armazenadas as *tags* customizadas criadas usando o Grails;

- grails-app/views: contém todos os arquivos GSP(*Groovy Server Pages*) da aplicação, ou seja, as páginas que serão visualizadas pelos usuários da aplicação;

- test: armazena os testes unitários e de integração do Grails;

- lib: no diretório lib são incluídos as bibliotecas externas que serão utilizadas pela aplicação, como por exemplo conectores de banco de dados, bibliotecas para manipulação de pdf e etc...

- web-app: é neste diretório que é armazenado o conteúdo estático da aplicação. Podem ser armazenados neste diretório imagens, arquivos HTML e CSS.

3.2.3 Classes de Domínio, Controladores e Visualizações

Como já explanado, o Grails segue o padrão MVC (*Model, View, Controller*). Sendo assim o Grails segue o estilo composto por três camadas.

Classes de Domínio

Davis e Rudolph (2010) define que as classes de domínio são a alma de uma aplicação Grails. Oak (2006) acrescenta que classes de domínio são objetos que são mapeados para o banco de dados. De forma mais direta, Pivotal (2014) afirma que as classes de domínio cumprem o "M" no padrão *Model View Controller* (MVC) e representam uma entidade que é mapeada para uma tabela do banco de dados. Pivotal (2014) completa ainda que o nome da classe de domínio, por padrão, é mapeado para o nome da tabela, em letras minúsculas, e que

cada atributo declarado é mapeado para colunas em suas respectivas tabelas. A Figura 7 a seguir demonstra a declaração de uma classe de domínio.

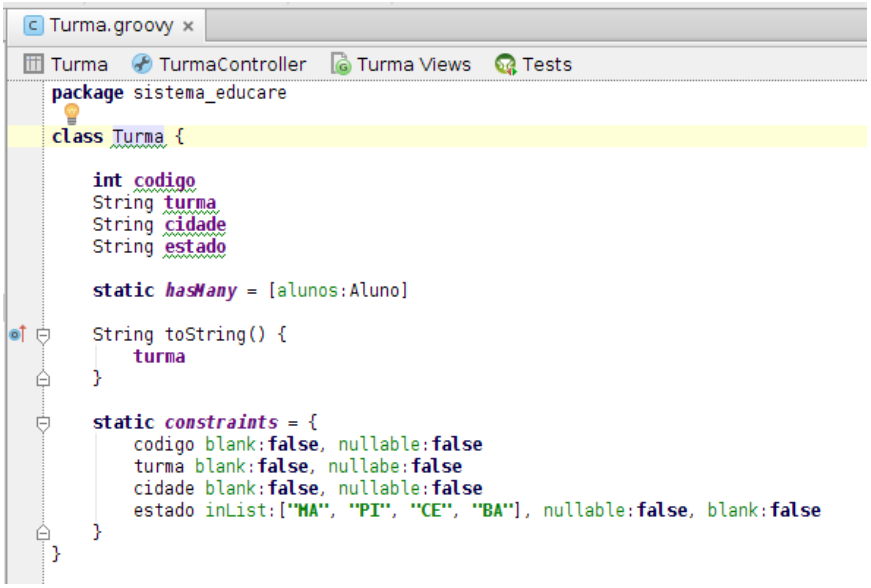

Figura 7 *– Exemplo de Classe de Domínio.*

Em se falando de classes de domínios é importante ressaltar um bloco de código que é chamado de "*constraints*".

Smith (2009) cita que a *closure* "*constraints*" afeta os *scaffolds* gerados, por exemplo, a ordem de entrada dos atributos da classe de domínio no bloco de *constraints* afeta a ordem dos campos nas páginas geradas. As *constraints* permitem definir declarativamente as regras de validação que deve-se aplicar às classes de domínio (WEISSMANN, 2011). Ou seja, as constraints são basicamente regras de validação que são verificadas antes que os dados sejam realmente persistidos no banco de dados. Vejamos alguns exemplos de *constraints*

- *blank*: em caso de "*false*", não permite que o campo seja deixado em branco ao submeter um formulário;
- *nullable*: caso esteja declarado como "*false*", não permite que o campo seja persistido com o valor "*null*";
- *inList*: gera automaticamente, na *view*, um campo no formato "*combobox*" contendo uma lista dos elementos declarados nessa *constraint*;
- e-mail: caso esteja declarado como "*true*" só permite que o usuário submeta o formulário caso o campo esteja preenchido com um e-mail válido.

Além das *constraints* que o Grails fornece, é possível ainda criar *constraints* customizadas.

O *Framework* Grails também apresenta o GORM (*Grails Object Relational Mapping*), que mapeia objetos para banco de dados relacionais e representa as relações entre esses objetos (DA, 2010). da (2010) acrescenta ainda que O GORM é característico de linguagens dinâmicas como o próprio Groovy e representa no banco de dados o CRUD dinamicamente sem se preocupar com a implementação deste banco de dados.

Pelo fato de não ser necessário a implementação direta do banco de dados, Klein (2009) afirma que esta ferramenta do Grails elimina a necessidade de grande parte do clichê, código repetitivo, que teríamos que escrever para trabalhar com outros sistemas. O grande ponto chave do GORM, segundo Fischer (2009) é que ele fornece um conjunto muito rico de configuração, com muitas predefinições convenientes e ao mesmo tempo, permite que essas configurações sejam ajustadas para melhor adequar-se ao sistema que será desenvolvido.

É importante observar que o Grails entende as classes Groovy dentro do diretório "grails-app/domain" como uma classe GORM, ou seja, que deve ser representada no banco de dados. Sendo assim, o próprio Grails adiciona uma série de configurações e funcionalidades a esta classe. É a partir disto, que o Grails consegue agilizar ainda mais o desenvolvimento.

Controladores

Os controladores consistem de um conjunto de métodos de ação(*closures*) (KOLLIKER; SCHMUTZ, 2008). Os controladores recebem as entradas dos navegadores dos usuários, interagem com a lógica de negócio e modelo de dados e por fim redirecionam o usuário para a página correta a ser exibida (SMITH, 2009). São os controladores os responsáveis pela manipulação das requisições na aplicação *web* (OAK, 2006). Cada classe de domínio tem seu controlador (KOLLIKER; SCHMUTZ, 2008). Sem controladores, a aplicação seria totalmente estática. A Figura 8 ilustra a declaração de um controlador.

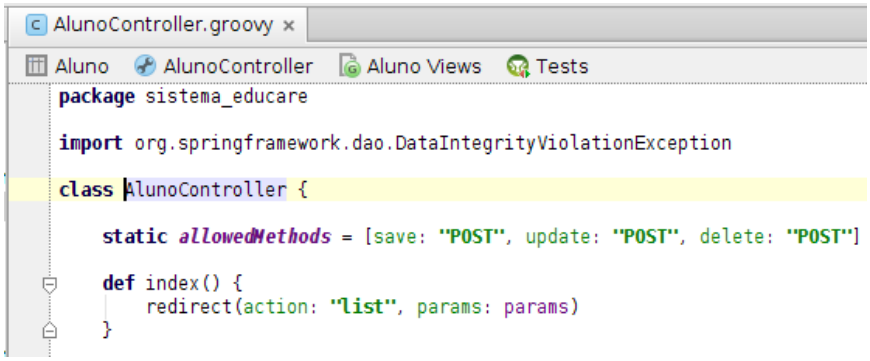

Figura 8 *– Exemplo de Controlador.*

Para manter boas práticas de desenvolvimento, segundo Kumar (2013) é importante manter os controladores enxutos, ou seja, garantir que a lógica de negócios, consultas e atualizações não sejam executados dentro dos controladores. Para este fim, é mais viável a utilização das classes de serviço. Vale lembrar que dentro dos controladores é que são definidas, em forma de *closures*, as ações do sistema, que recebem as requisições e simplesmente fazem alguma coisa com ela como por exemplo uma renderização na página, ou um redirecionamento caso necessário.

Visualizações

As visualizações (*views*) são as páginas que serão visualizadas pelos usuários finais do sistema. São de fundamental importância, pois é através delas que estes usuários poderão acessar as funcionalidades da aplicação.

As visualizações, no Grails, são arquivos com a extensão GSP (*Groovy Server Pages*) onde pode ser inserido o HTML ou as *tags* exclusivas do Grails. As gsp's também aceitam *script's* do tipo CSS.

3.2.4 Vantagens e Desvantagens

Toda tecnologia possui vantagens e desvantagens, e com o Grails não é diferente. Segundo Franco (2011), as principais vantagens do Grails são:

- Uma camada para mapeamento objeto-relacional fácil de usar construída no *Hibernate*;
- Uma tecnologia de visão expressiva, as GSP's;
- Uma camada de controle construída no *Spring* MVC;
- Suporte à internacionalização;
- Um *container Tomcat* integrado que é configurado para recarregar quando necessário;

Ja segundo Weissmann (2009b), algumas desvantagens do Grails são: conhecer a plataforma Java é obrigatório; Groovy não é supérfluo, ou seja, ter conhecimento sobre a linguagem é fundamental; em uma aplicação real, raras vezes o código gerado pelo Grails será utilizado entre 10, 15% devido as customizações necessárias de acordo com o projeto desenvolvido. Completando ainda que é necessário conhecer o mínimo sobre MVC.

3.3 Android

Uma Pesquisa divulgada pela empresa de consultoria Gartner (GARTNER, 2013), revela que o sistema operacional Android responde hoje por 85,1% dos aparelhos vendidos no

Brasil. Comparado com os 72,6% de 2012, significa um crescimento de 14,5%. Uma outra pesquisa da mesma empresa revela que o Android é o Sistema Operacional mais utilizado em *Tablets*, com 61,9% do mercado.

Segundo Rabello (2014):

> O desenvolvimento de aplicações para dispositivos móveis tem evoluído exponencialmente com o tempo e já se tornou um padrão de desenvolvimento quase onipresente no sentido de que muitas empresas estão escalonando ou recrutando novos desenvolvedores para formarem seus grupos de desenvolvimento para sistemas móveis a fim de criar novas ou adaptar soluções de serviços existentes para suprir a demanda do mercado (empresas bancárias disponibilizando serviços de operações financeiras pelo celular, controle de estoque de materiais e outros).

3.3.1 Plataforma Android

O Android é uma plataforma para *smartphones*, baseada no sistema operacional Linux. Possui diversos componentes, com uma variada disponibilidade de bibliotecas e interface gráfica, além de disponibilizar ferramentas para a criação de aplicativos (LECHETA, 2009). Desenvolvido pela *Open Handset Alliance*, um consórcio de mais com 40 empresas do setor de tecnologia e comunicação e liderada pelo Google, o Android tem como objetivos principais (LECHETA, 2009):

- A oportunidade de personalização das aplicações e componentes presentes em seu sistema, por ser de código aberto e gratuito;
- A possibilidade de desenvolvimento rápido e moderno de aplicações corporativas, uma vez que sua plataforma é moderna e flexível.

No Android, aplicações são desenvolvidas em linguagem de programação Java, utilizando o *Android Software Development Kit* (SDK), e executam na máquina *virtual Dalvik*. O Android, entretanto, possui suporte para o desenvolvimento de aplicações nativas, escritas em C e C++, através do Kit de Desenvolvimento Nativo do Android (NDK) (Brahler 2010). O *bytecode* da máquina virtual Dalvik (DVM), é diferente da *Java Virtual Machine* (JVM), por esse motivo ela não é considerada uma máquina virtual Java.

Dalvik é uma máquina virtual baseada em registradores que foi otimizada para garantir que um dispositivo possa executar várias instâncias de forma eficiente (MEIER, 2009b). Ela foi Desenvolvida pela Google, onde foi escrita por Dan Bornstein e outros engenheiros da Google. A Dalvik veio junto com o lançamento da SDK do Android no final de 2007. Criada com base na especificação da *Interface* Portável entre Sistemas Operacionais (POSIX) para sistema operacional UNIX onde o intuito de "Reinventar a roda" era de melhorar no gerenciamento de processo e isolamento de *thread* (SALABERRI, 2011).

Na plataforma Android os arquivos .java também são convertido para .class e através de uma ferramenta chamada de "dx"(faz parte do SDK do Android) são convertidos para a

extensão .dex que é interpretada pela DVM. Considerando que um arquivo.class contém apenas uma classe o arquivo .dex contém várias classes, de modo a reduzir o tamanho do arquivo e operações de entrada e saída (I / O), e acelerar a velocidade de busca. O arquivo .dex foi otimizado para o uso de memória e o design principal é impulsionado para o compartilhamento de dados. Esses tipos de arquivo uniram todas as classes formando um só arquivo e apenas uma única cópia é necessário, com essa consolidação a DVM é capaz de aumentar o seu desempenho (SALABERRI, 2011).

Aplicativos para Android são distribuídos através da loja de aplicativos do Google Play, que conta com cerca de 700.000 aplicativos e jogos, tanto pagos como gratuitos (PLAY, 2013). O sistema operacional também está presente em mais de 900 milhões de *smartphones* e *tablets* em todo o mundo, tornando o Android uma das plataformas mais interessantes para o desenvolvimento de aplicativos (TANJI, 2013).

Arquitetura da Plataforma Android

O Android é mais do que um sistema operacional. Ele é na verdade um *software stack* composto por cinco camadas. A base do Android é uma versão modificada do *kernel Linux* 2.6, que provê vários serviços essenciais, como segurança, rede e gerenciamento de memória e processos, além de uma camada de abstração de *hardware* para as outras camadas de *software* (BORDIN, 2012). A seguir, pode-se observar a Figura 9 com os principais componentes da arquitetura Android.

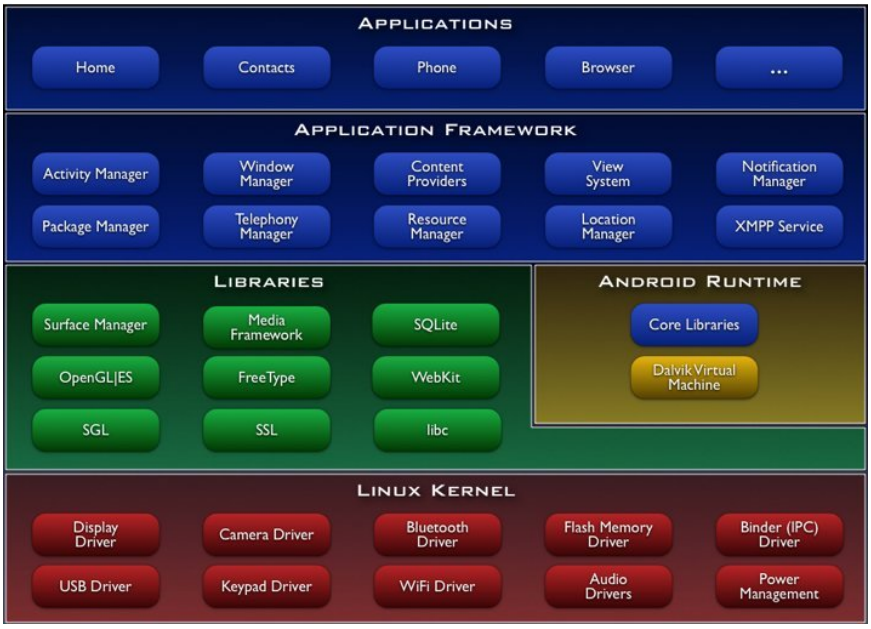

Figura 9 *– Arquitetura Android.*

Disponível em: <http://developer.android.com/images/system-architecture.jpg>

A arquitetura do Android é baseada no *kernel* do GNU/Linux, versão 2.6. O *kernel* do sistema funciona como uma camada de abstração entre o *hardware* e o restante da pilha de *softwares* da plataforma (AQUINO, 2007). Esta camada contém todos os drivers "low-level" para os vários componentes do dispositivo android.

Acima do *kernel* ficam as bibliotecas C/C++ utilizadas por diversos dos componentes do sistema, como: uma implementação da biblioteca padrão do C (libc), mas com licença de código aberto *Berkeley Software Distribution* (BSD) e otimizada para dispositivos embarcados; bibliotecas para suporte a formatos de áudio, vídeo e imagens; um gerenciador que intermedia o acesso ao *display* e compõe as camadas de imagem 2D e 3D; o *engine* para navegadores *WebKit*; bibliotecas para gráficos 2D *Graphics Library Skia* (SGL) e 3D *OpenGL for Embedded Systems* (OpenGL ES); um renderizador de fontes *bitmap* e vetoriais; e o banco de dados relacional *SQLite* (DEVELOPERS, 2012).

Na mesma camada que as bibliotecas, o *Android Runtime* fornece um conjunto de bibliotecas que permitem aos desenvolvedores escrever aplicativos Android usando a linguagem de programação Java (LEE, 2011). Toda aplicação Android executa seu próprio processo, com sua própria instância da máquina virtual Dalvik. Dalvik foi escrito de forma que um dispositivo possa executar múltiplas máquinas virtuais concorrentemente de maneira eficiente (AQUINO, 2007).

Expõe os vários recursos do sistema operacional Android para desenvolvedores de aplicativos para que eles possam utilizá-los em suas aplicações (LEE, 2011). Os desenvolvedores têm acesso completo à mesma API que é usada pelas aplicações core da plataforma Android (AQUINO, 2007). Sendo assim, ainda de acordo com (AQUINO, 2007) a arquitetura da aplicação foi projetada para simplificar o reuso dos componentes. Além disso o *framework* de aplicações, também fornece uma abstração genérica para acesso ao *hardware* e gerencia interface do usuário de aplicativos e recursos.

A camada mais alta é onde se encontram as aplicações. Que possuem as funcionalidades básicas do dispositivo. Todos os os aplicativos, tanto nativos quando de terceiros, são construídos sobre a camada de aplicativo usando as mesmas bibliotecas de API do Android (MEIER, 2009a) . A camada de aplicações fornece um conjunto de aplicações básicas que inclui um cliente de e-mail, um programa SMS, um calendário, mapas, navegador, contato entre outras (AQUINO, 2007).

Arquitetura das Aplicações Android

Segundo Meier (2009a) os seguintes serviços de aplicação são os pilares arquitetônicos de todos os aplicativos Android, fornecendo a estrutura que você vai usar para seu próprio *software*:

- Gerenciador de *activities*: controla o ciclo de vida de suas atividades. As atividades(*activities*) serão melhor abordadas mais a frente;
- Visualizações: são utilizadas para construir as interfaces de usuário para as atividades. Também serão abordadas posteriormente;
- Gerenciador de notificações: fornece um mecanismo consistente e não-instrusivo para notificar os usuários;
- Provedores de conteúdo: permite que suas aplicações compartilhe dados com outras aplicações;
- Gerente de recursos: suporta recursos como "*strings*" e gráficos que podem ser exibidos aos usuários.

Blocos de Construção Android

Existem seis componentes que fornecem blocos para construção de aplicações Android. Aqui serão explanados os quatro principais: *Activity*(atividade), *Intent Receiver* (Receptor de Intenção), *Service* (Serviço), Content Provider(Provedor de Conteúdo).

A *activity* é um dos componentes mais importantes ao se desenvolver uma aplicação Android. É basicamente a camada de apresentação da sua aplicação. Cada *activity* é implementada como uma única classe que extende da classe base *Activity* (AQUINO, 2007). Atividades usam visualizações (*views*) para formar interfaces gráficas que exibem informações e respondem a ações do usuário (MEIER, 2009a). Uma aplicação pode possuir uma quantidade ilimitada de *activities* (respeitando o limite de processamento do Android). Na figura 10, podemos observar a criação de uma *activity*.

```
public class MainActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

***Figura 10** – Exemplo de declaração de Activity.*

O método "*onCreate*", que pode ser observado na imagem anterior, é um método, que também pode ser chamado de evento, padrão que é criado no momento de criação da *activity*. Este método é automaticamente chamado quando a *activity* é criada pela primeira vez. Além

deste, existem diversos outros eventos de acordo com o ciclo de vida da *activity*, com por exemplo: "*onStart*" que é automaticamente chamado quando a *activity* se torna visível ao usuário, o "*onResume*", que é chamado quando a *activity* começa a interação com o usuário, "*onStop*" que é chamado quando a *activity* não está mais visível ao usuário, "*onRestart*" que é chamada quando a *activity* é finalizada e está reiniciando novamente, dentre outros.

Toda atividade deve ser declarada no arquivo "AndroidManifest.xml", que é o principal arquivo de configuração do Android. É neste arquivo que estão contidos as maiorias das configurações a respeito de declarações e permissões de uso de componentes específicos do Android como por exemplo internet e manipulação dos arquivos do cartão de memória. Na Figura 11 observar a declaração da "MainActivity" no arquivo Manifest.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="br.com.ufpi.gscm"
          android:versionCode="1"
          android:versionName="1.0">
    <uses-sdk android:minSdkVersion="10" android:targetSdkVersion="19"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <application android:label="GSCM" android:icon="@drawable/ic_launcher">
        <activity android:name="MainActivity"
                  android:label="GSCM">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity android:name=".BodyActivity"
                  android:screenOrientation="landscape"
                  />
```

Figura 11 *– Declaração das activies e permissões no arquivo Manifest.*

Em relação à "*Intent Receiver*", quando a aplicação é provida de mais de uma *activity* se torna necessário que a aplicação permita que o usuário navegue entre essas *activities*. É através do *Intent Receiver* que isso se torna possível no Android. É usado quando o código em uma aplicação deve executar em reação a um evento externo, por exemplo quando o telefone toca ou quando os dados da rede estão disponíveis (AQUINO, 2007). Em linhas gerais, uma Intent representa uma descrição abstrata de uma função de uma *activity* que exige outra *activity* para realizar algo, como por exemplo, tirar uma foto. Quando o usuário deseja tirar uma foto, o que acontece é que o "Menu" clicado ativa a "Intent" para que esta chame a *activity* responsável por manipular a câmera do dispositivo. Ao chamar uma "*Intent*", as vezes se torna necessário que uma activity envie dados relevantes para outras *activities*. Isto é possível graças ao "*Bundle*", que em português significa pacote. Na Figura 12, podemos observar a chamada de uma "Intent"

junto a utilização do “Bundle”.

```
Bundle extras = new Bundle();
extras.putString("chave", "qualquer conteudo aqui...");
Intent intent = new Intent(ConsultaActivity.this, PerfilActivity.class);
intent.putExtras(extras);
```

Figura 12 *– Exemplo de chamada de “Intent” junto ao “Bundle”.*

Um serviço é um código que está executando em *background* sem uma *interface* gráfica com o usuário (AQUINO, 2007). Segundo Steele e To (2011), um serviço pode ser iniciado ou interrompido por qualquer componente, inclusive por ele mesmo, completando que quando o serviço está em execução, qualquer componente pode ligar-se a ele.

Os serviços também permitem que os aplicativos compartilhem funções através de conexões de longo prazo. Esta prática é reminiscência de serviços de Internet como o FTP e o HTTP (MEDNIEKS et al., 2011). Mednieks et al. (2011) completa ainda que os serviços, assim como as *activities*, oferecem métodos que controlam seu ciclo de vida como por exemplo o “*stop-ping*”.

Quanto ao “*Content Provider*”, as aplicações podem armazenar seus dados em arquivos, como o banco de dados SQLite. Um provedor de conteúdo(*Content Provider*), todavia, é útil se os dados da aplicação podem ser compartilhados com outras aplicações (AQUINO, 2007).

Mednieks et al. (2011) afirma que o *Content Provider* é mais ou menos semelhante a um *Web Service* RESTful, que pode ser encontrado através de uma URI (*Uniform Resource Identifier*), possuindo também operações de envio e obtenção de conteúdo semelhantes a esta ferramenta. *Content Provider’s* são o coração do modelo de conteúdo do Android: fornecendo um *Content Provider*, o aplicativo pode compartilhar dados com outras aplicações e gerenciar o modelo de dados de um aplicativo.

Existem três tipos principais de tipos de aplicações android: aplicações de primeiro plano, serviços de fundo e atividades intermitentes.

- Aplicações de primeiro plano: são aplicações que necessitam estar em primeiro plano para pleno funcionamento. Quando não estão em primeiro plano estas aplicações são encerradas. Exemplos de aplicações de primeiros planos são jogos, calculadora, etc;

- Serviços de fundo: este tipo de aplicação passa a maior parte do seu tempo oculto. Geralmente aguardam algum fator de inicialização externo. Exempplo deste tipo de aplicação são as auto-respostas a mensagens ou chamadas;

- Atividades intermitentes: atividades intermitentes unem conceitos dos dois tipos de aplicações citados anteriormente. Eles aguardam uma interatividade, passam a maior parte

do tempo em segundo plano, porém também podem interagir em primeiro plano e exibir avisos ao usuário caso necessário. Um exemplo clássico deste tipo de aplicação é o *media player*.

3.4 Web Services

Os *Web Services* tem se tornado uma importante ferramenta de integração de diferentes sistemas não importando a linguagem que foram desenvolvidos. Eles surgiram da necessidade de comunicar aplicações que já existiam, afim de tornar o acesso aos processos já existentes nessas aplicações mais fácil, rápido e distribuído. Ou seja, através desta tecnologia é possível que um processo de uma aplicação possa ser utilizado de forma distribuída por outra aplicação através de protocolos padrões da Internet.

Esta tecnologia é bastante flexível e uma de suas características principais diz respeito à possibilidade de utilização de diferentes formas de transmissão de dados pela rede. Logo, a arquitetura de *Web services* pode trabalhar com protocolos, tais como HTTP, SMTP, FTP, RMI/IIOP ou protocolos de mensagem proprietários (W3C, 2004).

3.4.1 Tipos de Web Services

Existem diferentes tipos de abordagens para *Web Services*. Cada abordagem se adequa melhor a um diferente caso. Os dois principais são o SOAP e o REST.

SOAP (Simple Object Access Protocol)

O SOAP é um protocolo leve destinado à troca de informações estruturadas em um ambiente descentralizado e distribuído. Ele usa tecnologia XML, que será mais especificada mais a frente, para definir um quadro de mensagens extensível proporcionando uma construção de mensagem que pode ser trocada por uma variedade de protocolos subjacentes (GUDGIN et al., 2007).Sendo assim, este protocolo de comunicação dita um formato de envio de mensagens entre aplicações, ou seja, qualquer plataforma de comunicação pode ser utilizada, seja ela proprietária ou não (W3C, 2000).

Uma mensagem SOAP (Figura 13) consiste basicamente dos seguintes elementos: *en-*

velope, header e body.

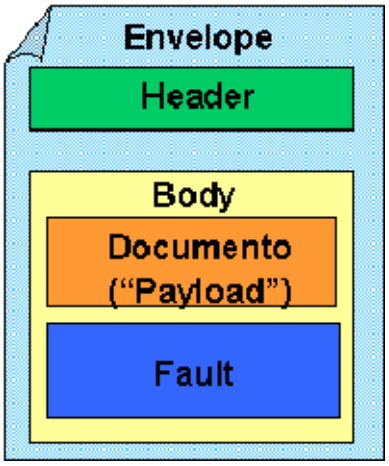

Figura 13 *– Elementos da mensagem SOAP.*

Disponível em: <http://www.oficinadanet.com.br//imagens/conteudos/160/soap-envelope.gif>

O envelope o principal elemento da mensagem. Como o próprio nome já diz, este elemento serve como um envelope para os demais elementos, ou seja, dentro dele estarão contidos o cabeçalho(*header*) e o corpo(*body*) da mensagem. Geralmente é na declaração do envelope que estão declarados o namespace e estilo de codificação da mensagem.

O "*header*" é opcional e carrega informações adicionais a respeito da mensagem SOAP. Caso seja utilizado, o "*header*" deve ser o primeiro elemento do envelope.

O "*body*" é um elemento obrigatório nas mensagens SOAP e é ele que carrega a informação(*payload*) a ser transportada. No "*body*" podem estar presentes definições de chamadas a métodos e parâmetros de entrada que serão utilizados por estes métodos. Caso a mensagem SOAP seja de resposta, ao invés de parâmetros de entrada, o "*body*" carregará parâmetros de saída provindos da chamada do método. Ainda dentro do "*body*" existe um elemento opcional, o "*Fault*", que é utilizado para carregar os status ou erros das mensagens retornados pelos nós que processaram a mensagem.

Na Figura 14, pode-se observar uma mensagem SOAP no formato XML contendo o

envelope, header e body.

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
   <n:priority>1</n:priority>
   <n:expires>2001-06-22T14:00:00-05:00</n:expires>
  </n:alertcontrol>
 </env:Header>
 <env:Body>
  <m:alert xmlns:m="http://example.org/alert">
   <m:msg>Pick up Mary at school at 2pm</m:msg>
  </m:alert>
 </env:Body>
</env:Envelope>
```

Figura 14 *– Exemplo de mensagem SOAP.*

As mensagens SOAP podem facilmente atravessar *firewalls*, que é um sistema de proteção de redes internas contra acessos não autorizados originados de uma rede não confiável (Internet), ao mesmo tempo que permite o acesso controlado da rede interna à Internet (D'ÁVILA, 2001). Isto devido sua infraestrutura ter sido criada para o HTTP, que é um dos protocolos mais ubíquos da internet (CHANG; VITTIE, 2008). Além do mais, o fato dos dados serem encapsulados utilizando o XML possibilita que estes dados possam ser entendidos na maioria dos sistemas operacionais e linguagens de programação. Outra importante vantagem é o já embutido tratamento de excessões que possibilita o tratamento de erros. E por fim, é simples de implementar, testar e usar.

Infelizmente o SOAP ainda conta com mecanismos de segurança imaturos principalmente no que diz respeito a criptografia do conteúdo da mensagem. Segundo Lima (2012), outros ponto negativos do SOAP são o fato de que ele não utiliza todo o poder do HTTP, limitando-se ao método POST para realizar múltiplas operações, todo o serviço está sob a mesma URI, logo não há encadeamento entre os recursos, e ainda o fato de o SOAP não poder ser armazenado em cache.

REST (Representational State Transfer)

Foi idealizado por Roy Fielding no ano de 2000, na sua dissertação de doutorado, na qual buscou as melhores práticas nos estilos de arquiteturas existentes para compor um novo estilo que as reunissem em apenas um estilo (LIMA, 2012).

Representational state transfer (REST), traduzido ao pé da letra, transferência de estado representacional é um estilo de arquitetura de *software*, ou seja, um "estilo arquitetônico", que basicamente explora a tecnologia existente e protocolos da Web (OLIVEIRA, 2013b). Para

quem já ouviu falar de *Web Services REST*, geralmente existe uma confusão entre *REST* e *RESTful*, portanto, é importante ressaltar esta diferença. Basicamente o *RESTful* é utilizado para se referir a um *web service* que utiliza a arquitetura *REST*, ou seja, normalmente *RESTful* são implementações baseadas no estilo arquitetônico *REST*.

Na figura 15, pode-se observar uma ilustração a respeito da arquitetura do *REST*, que é totalmente baseado em requisições e respostas HTTP.

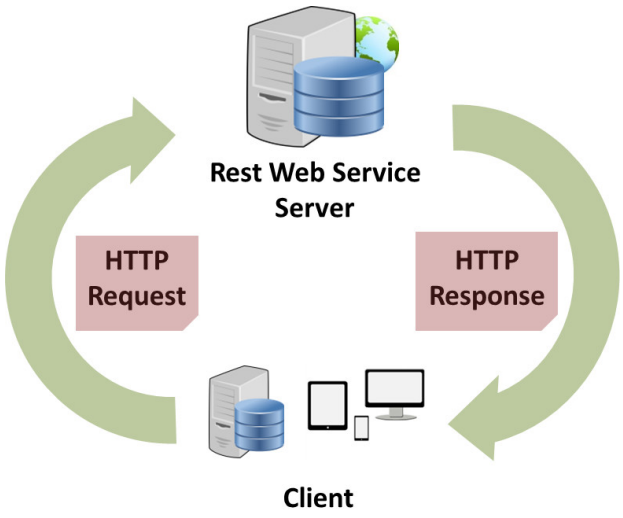

Figura 15 *– Arquitetura REST.*

Disponível em: <http://www.chemaxon.com/wp-content/themes/chemaxon/images/>

Para que um *Web Service* seja considerado do tipo *REST*, é necessário que ele obedeça aos seguintes princípios:

- Cliente-Servidor: de forma simplificada, o servidor disponibiliza uma gama de recursos ou serviços que serão utilizados pelos clientes. Estes serviços disponibilizados pelo servidor são acessados pelo cliente através da rede de computadores. É importante lembrar que o cliente não se importa em como os dados estão armazenados no servidor, assim como o servidor não se importa com a interface utilizada pelo cliente. Sendo assim, o fluxo de trabalho de ambas as partes se tornam mais simples e o seu desenvolvimento pode se dar de forma independente tanto no lado cliente, quanto no lado servidor. Uma das principais formas de acessar os serviços disponibilizados pelo servidor se da através dos navegadores *web*;

- Stateless (Sem estado): este princípio diz que o servidor não deve armazenar nenhuma informação a respeito da requisição enviada pelo cliente, ou seja, tudo o que for necessário para que o servidor processe uma informação deve estar contida nela mesmo. Segundo

{citeonlinesao-paulo.pm.orgartigo2010RESTful, este conceito em que a requisição deve ser auto-suficiente traz alguns benefícios para este tipo de *Web Service*, como: visibilidade, implicando menor trabalho no servidor, confiabilidade, pois facilita a tarefa de recuperação de falhas e escalabilidade, pois permite que o servidor se livre dos recursos alocados rapidamente;

- Cache: o *cache* foi uma forma de tentar contornar o problema de baixo desempenho dos *Web Services* do tipo *REST*. Na primeira vez que uma página é solicitada pelo cliente a mesma é armazenada num *Proxy* HTTP (FIELDING, 2000 apud LIMA, 2012). A vantagem de se utilizar *cache* é a eliminação parcial ou total de algumas interações entre cliente e servidor, o que melhora a eficiência (menos trafego de rede), escalabilidade (menos processamento) e performance, já que o servidor fica menos carregado (LIMA, 2012);

- Camadas: o intuito de utilizar camadas está em tornar o processo mais escalável. No sistema em camadas, o mesmo é divido em camadas, onde cada camada conhece apenas a interface da camada superior. As camadas intermediárias podem ser utilizadas para melhorar a escalabilidade do sistema, permitindo o balanceamento de carga de serviços, através de múltiplas redes(FIELDING, 2000 apud LIMA, 2012). A utilização de camadas no entanto pode gerar uma queda no desempenho principalmente devido ao *overhead* gerado na mensagem e latência nos dados processados.

As principais vantagens do *REST* dizem respeito a facilidade de implementação, eficiência, no que diz respeito a utilização de formatos de mensagens variados (SOAP por exemplo requer obrigatoriamente utilização do XML) e rapidez. Porém, os conceitos sobre segurança são melhores especificados e difundidos nos protocolos baseados em SOAP do que no protocolo HTTP, utilizado pelo estilo *REST* (LIMA, 2012).

3.4.2 Representações de Dados em Web Services

Um determinado serviço ou recurso pode ser disponibilizado pelo servidor através de uma URI. A partir desta URI é o cliente pode acessar o serviço desejado em diferentes tipos de representações, ficando a cargo a decisão pela representação desejada. Dentre essas representações duas se destacam: o formato XML e o formato JSON, que serão abordados nesta seção.

XML (Extensible Markup Language)

O XML foi disponibilizado em 1997 pela W3 Consortium. Ele é uma linguagem de marcação ou SGML (*Standard Generalized Markup Language*), ou seja, não possui semântica definida, apenas indica como o conteúdo deve ser interpretado. O que chama bastante atenção no XML é o fato da possibilidade do autor do documento criar "marcas" próprias, tornando o conteúdo do documento mais legível para propósitos específicos, como por exemplo, *Web Services*.

Os documentos XML são compostos principalmente por: elementos, que são definidos entre os símbolos "maior que" e "menor que"; atributos, que são características dos elementos representados e comentários, que não são processados e servem como informações relevantes sobre elementos ou atributos. A Figura 16 apresenta um modelo de documento XML.

```
<?xml version="1.0"?>
<funcionarios>
   <funcionario ID="2107">
      <nome>Alessandra</nome>
      <salario>1000</salario>
   </funcionario>

   <funcionario ID="0200">
      <nome>Cid Onir</nome>
      <salario>700</salario>
   </funcionario>

   <funcionario ID="1020">
      <nome>Bianca</nome>
      <salario>980</salario>
   </funcionario>
</funcionarios>
```

Figura 16 *– Modelo de documento XML.*

No exemplo demonstrado pode-se observar o elemento "<funcionarios>" que é considerado o elemento-pai. É importante que documentos XML bem formatados possuam o element-pai. "<?xml version="1.0"?>" é uma instrução de processamento que indica a versão do XML.

Devido a sua simplicidade e interoperabilidade o XML tem alcançado alta aceitação como representação de dados na troca de informações em Web Services. Atualmente, XML é a base para todos os *Web Services* modernos que utilizam tecnologias baseadas em XML para descrever seus padrões de comunicação. WSDL, SOAP e UDDI são exemplos de *web services* que usam mensagens baseadas em XML, o qual qualquer máquina pode interpretar.

Porém, XML não é perfeito, logo possui alguns pontos fracos com por exemplo sintaxe redundante, dificuldade de utilização da sintaxe XML para humanos e o fato de seus requisitos básicos não suportarem um grande conjunto de dados. Desta forma, tornou-se necessária a busca por outras alternativas ao XML.

JSON (JavaScript Object Notation)

O JSON foi desenvolvido buscando a simplicidade e portabilidade. É importante ressaltar que apesar de possuir "*JavaScript*" em sua nomenclatura ele não necessariamente só pode ser usado com *javascript* e além de possuir um formato, leve é também muito simples de ler. Ele pode representar quatro tipos primários (*strings*, números, booleanos e nulos) e dois tipos estruturados (objectos e vectores). Um objeto é uma coleção não ordenada de zero ou mais pares nome/valor, onde o nome é uma *string* e o valor é uma *string*, número, booleano, nulo, objecto ou vector. Um vetor é uma sequência ordenada de zero ou mais valores (CROCKFORD, 2006 apud FONSECA; SIMõES,). Atualmente empresas como Google, Facebook, Yahoo! e Twitter utilizam o formato JSON.

Através das figuras 17 e 18 é possível visualizar a diferença entre um trecho em XML e este mesmo trecho JSON.

```
<?xml version="1.0" encoding="UTF-8"?>
<id>1</id>
<nome>Andrei Maxwel</nome>
<endereco>Rua Número 1</endereco>
```

Figura 17 *– Trecho em XML.*

```
{"id":1,"nome":"Andrei Maxwel", "endereco":"Rua Número 1"}
```

Figura 18 *– Trecho em JSON.*

É notável a diferença. Visualmente os textos em JSON são mais fáceis de compreender e ler.

As principais vantagens do JSON em relação ao XML são: leitura simplificada, suporte a objetos, arquivos JSON são menores que arquivos XML, eliminação da redundância de marcadores (que torna a leitura do XML bastante confusa ao ser humano) (GAMA,).

Atualmente o JSON tem sido uma ótima escolha para as representações utilizadas em trocas de informações nos Web Services. Isto se deve ao fato de que o XML, em alguns casos, se torna mais complexo que o necessário. Para melhor compreensão pode-se utilizar a analogia, por exemplo, de um canhão para matar uma formiga quando utiliza-se o XML para *Web Services* não complexos. Em linhas melhores, isto se deve ao fato de que por padrão o XML requer que

você utilize recursos mesmo que seu *Web Service* não necessite destes recursos. Desta forma, o JSON busca reduzir essa complexidade "obrigatória" no XML, sem desmontar a idéia de que é necessário manter um padrão nas representações de dados para trocas de informações bem sucedidadas.

3.5 IHC (Interface Humano Computador)

Segundo Carvalho (1994), o avanço tecnológico transformou o computador em uma ferramenta cada vez mais indispensável às atividades humanas (CARVALHO, 2003). Desta forma, o interesse em Interação Humano Computador tem crescido na mesma proporção que amplia o número de pessoas que utilizam computadores para realizar as mais diversas tarefas (CARVALHO, 2014a). O termo interface humano computador emergiu na segunda metade dos anos 80, como forma de descrever novo campo de investigação preocupado não somente com o *design da interface* de sistemas computacionais, mas, também, com o foco de interesse e de demandas do público (GUEDES, 2009).

Existem diversos conceitos a respeito de IHC. Rocha (2003) define IHC como a área preocupada com *design*, avaliação e implementação de sistemas computacionais interativos para uso humano, e, ainda, com o estudo dos principais fenômenos subjacentes a eles. Ja Carvalho (2003), diz que a IHC tem característica multidisciplinar e seu objetivo é tornar máquinas sofisticadas mais acessíveis, no que se refere à interação, aos seus usuários potenciais. Baecker e Buxton citados por Thakkar (apud CARVALHO, 2003) definem IHC como "o conjunto de processos, diálogos, e ações por meio dos quais o usuário humano interage com um computador".

Ainda a respeito dos objetivos da IHC, Rocha e Baranauskas (apud CARVALHO, 2003) afirma que IHC tem o objetivo de produzir sistemas usáveis, seguros e funcionais. Completa ainda que esse objetivos podem ser resumidos como desenvolver ou melhorar a segurança, utilidade, efetividade e usabilidade de sistemas que incluem computadores. De qualquer modo, percebe-se que Interface Humano Computador preocupa-se com a interação entre o ser humano e o computador, melhorando o máximo a experiência do utilizador, usabilidade e utilidade.

3.5.1 Interface e Interação

Para melhor entender o que é IHC, é necessário também conhecer os conceitos a respeito de interface e interação, pois estes dois aspectos caminham junto quando o assunto é IHC. Inclusive, para Lucena e Liesenberg (1994) uma boa interface torna a interação mais fácil de aprender e usar. Lucena e Liesenberg (1994) define interface como parte do *software* de um sistema interativo responsável por traduzir ações do usuário em ativações das funcionalidades do sistema (aplicação), permitir que os resultados possam ser observados e coordenar esta interação. Tavares () aponta outros conceitos de diferentes autores como "Parte do sistema com a

qual o usuário entra em contato físico, perceptivo e cognitivo.” [Moran 83], “Ambiente virtual para interatividade.” [Laurel 93] e “Imagem do sistema.” [Norman 86]. Resumindo, a *interface* deve apresentar e receber informação de modo consistente (SANTOS; COSTA, 2014).

3.5.2 Estilos de Interação

É grande a quantidade de formas que o usuário pode interagir com uma interface ou sistema. A estas formas ou modos de interação dá-se o nome de estilos de interação. Estilos de interação é um termo genérico que inclui todas as formas usadas pelos usuários para comunicarem ou interagirem com sistemas computacionais (SOUSA, 2014). Neste tópico serão abordados de forma breve alguns dos principais estilos de interação.

- Linguagem de Comando: estilo de comunicação rápido e poderoso, preferido por usuários com habilidade de digitação. É originário de comandos de sistemas operacionais criadas para o acesso direto às funções do sistema (SILVA; SILVEIRA, 2014). A Figura 19 mostra um exemplo de utilização da linguagem de linha de comando.

Figura 19 *– Exemplo de linguagem de comando.*

- Menus: é composto por uma lista de opções, favorecem os usuários pouco treinados ou ocasionais e ainda são de fácil treinamento (SOUSA, 2014). A Figura 20 demonstra a utilização deste estilo de interação.

Figura 20 *– Exemplo de menu.*

- Formulários: este estilo se refere a campos que devem ser preenchidos pelos usuários. São semelhantes ao modo original em papel. A seguir, a figura 21 apresenta um exemplo a respeito de formulários possuindo dois campos para preenchimento pelo usuário.

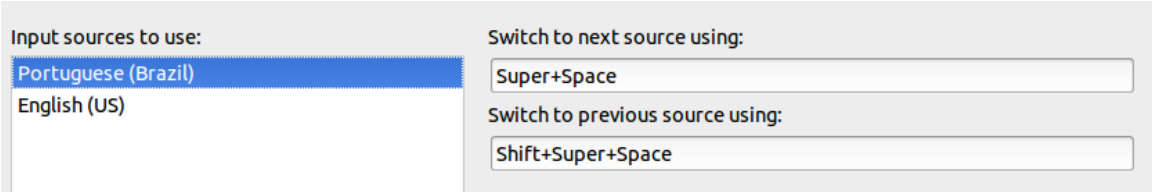

Figura 21 *– Exemplo de formulário.*

- Janelas: objeto que provê uma área de apresentação e interação com outros objetos (SILVA; SILVEIRA, 2014). A Figura 22 apresenta exemplo de sua utilização.

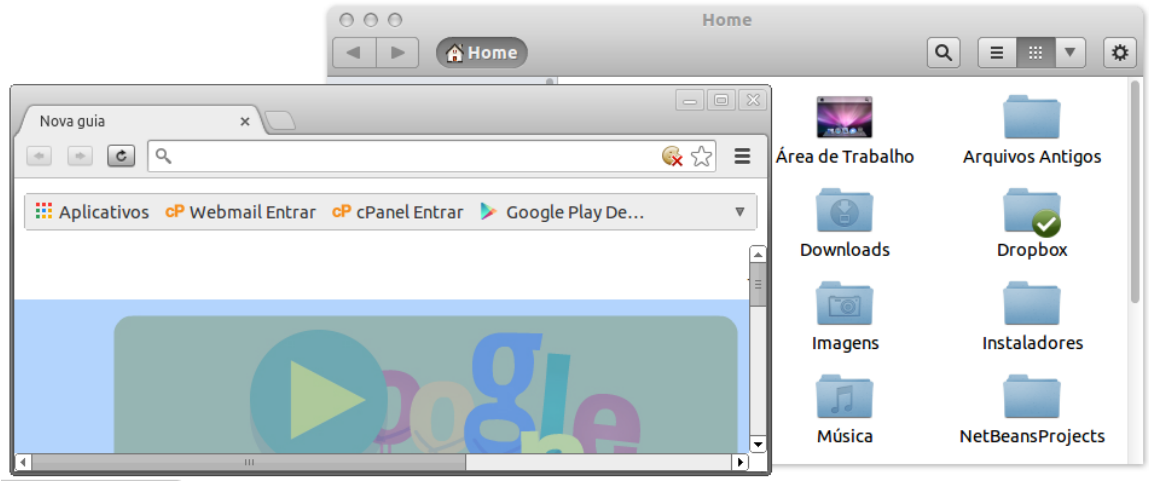

Figura 22 *– Exemplo de janela.*

- WIMP (Windows, Icons, Menus and Pointers): caracteriza *interfaces* com base em janelas, ícones, menus e cursos de *mouse*. A seguir (Figura 23), é possível observar um exemplo de uso deste estilo.

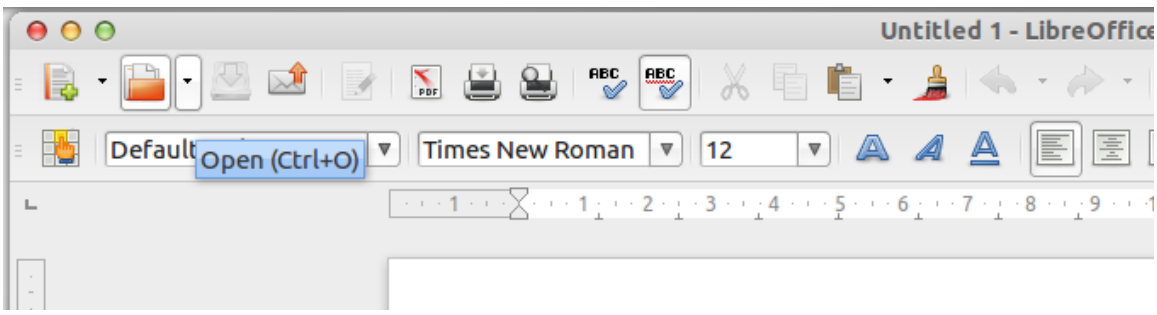

Figura 23 *– WIMP.*

3.5.3 Usabilidade

Existem diversos conceitos a respeito de usabilidade. A usabilidade pode ser definida como o estudo ou a aplicação de técnicas que proporcionem a facilidade de uso de um dado objeto (DGE, 2010). Segundo Souza (apud SOUZA; SPINOLA, 2006), por exemplo, a usabilidade de um sistema é um conceito que se refere à qualidade da interação de sistemas com os usuários e depende de vários aspectos como: Facilidade de aprendizado do sistema. Ja Nielsen (apud LOUREIRO, 2006), um grande especialista da área de usabilidade, define usabilidade como "Um conjunto de propriedades de uma interface que reúne os seguintes atributos: 1) Fácil aprendizado; 2) Eficiência; 3) Capacidade de memorização; 4) Baixo índice de erros; 5) Satisfação e prazer ao uso". O primeiro atributo, fácil aprendizado, diz que o sistema deve ser fácil de aprendizado e que o usuário consiga rapidamente entender seu funcionamento, conseguindo utilizá-lo no menor tempo possível. O segundo atributo, eficiência, deve permitir que o usuário utilize o sistema com alta produtividade. O terceiro atributo, capacidade de memorização, garante que o sistema é de fácil memorização, ou seja, as funcionalidades do sistema devem ser facilmente lembradas e dificilmente esquecidas. O quarto atributo, baixo índice de erros, diz que o sistema não pode permitir erros muito graves, e, caso ocorram, o sistema deve ser capaz de redirecionar o usuário para o ponto antes da ocorrência do erro. O quinto e último erro, satisfação e prazer ao uso, completa que os usuários devem usar o sistema porque gostam, ou seja, o sistema deve ser agradável ao usuário.

Heurísticas de Usabilidade de Nielsen

Jakob Nielsen, um dos maiores especialistas em usabilidade nos Estados Unidos, é autor de um livro clássico sobre o assunto, *Usability engineering*, de 1994, no qual ele propõe um conjunto de dez heurísticas de usabilidade (CYBIS et al., 2010). São elas: visibilidade do estado do sistema, mapeamento entre o sistema e o mundo real, liberdade e controle ao usuário, consistência e padrões, prevenção de erros, reconhecer em vez de lembrar, flexibilidade e eficiência de uso, *design* estético e minimalista, suporte para o usuário reconhecer, diagnosticar e recuperar erros e por fim, ajuda e documentação.

Com referência em Nielsen (1995), são explanados a seguir a que se refere cada heurística de Nielsen:

- Visibilidade do estado do sistema: o sistema deve sempre manter os usuários informados sobre o que está acontecendo, através do *feedback* apropriado, em tempo razoável;
- Mapeamento entre o sistema e o mundo real: o sistema deve "falar" a linguagem dos usuários, ou seja, utilizar palavras, frases e conceitos familiares a ele;

- Liberdade e controle ao usuário: os usuários muitas vezes escolhem funções do sistema por engano, necessitando então de uma saída de emergência para sair do estado indesejado sem ter que passar por um longo diálogo;

- Consistência e padrões: os usuários não devem ter que se perguntar se diferentes palavras, situações ou ações significam a mesma coisa, ou seja, o sistema deve utilizar convenções não ambíguas(BORGES, 2014);

- Prevenção de erros: melhor ainda que boas mensagens de erros é um projeto cuidadoso que, em primeiro lugar, evita o acontecimento do erro. Os erros são as principais fontes de frustração, ineficiência e ineficácia durante a utilização do sistema (BORGES, 2014);

- Reconhecer em vez de lembrar: minimizar a carga de memória do usuário, tornando os objetos, ações e opções visíveis e coerentes;

- Flexibilidade de eficiência de uso: aceleradores, como por exemplo, atalhos de teclado, podem muitas vezes acelerar a interação com o usuário experiente. O sistema deve ser flexível o suficiente para que o usuário possa realizar adequações frequentes.

- Design estético e minimalista: os diálogos não devem conter informações irrelevantes ou raramente necessárias;

- Suporte para o usuário reconhecer, diagnosticas e recuperar erros: as mensagens de erros devem ser expressas em linguagem simples (sem códigos). Devem também indicar o problema com precisão e indicar uma solução;

- Ajuda e documentação: mesmo que seja melhor que o sistema possa ser utilizado sem documentação, isto pode ser necessário para prover ajuda ao usuário;

3.5.4 IHC em Dispositivos Móveis

A experiência do usuário móvel é definida por Cybis et al. (apud SANTOS; COSTA, 2014) como uma composição de cinco fatores, dentre eles está a usabilidade. Ainda para Cybis et al. (apud SANTOS; COSTA, 2014), além da utilidade, disponibilidade e custo, um dos principais elementos para assegurar a satisfação do usuário que adquire um dispositivo móvel, é a usabilidade das interfaces desse dispositivo, que deve ser considerada um dos principais requisitos de projeto para que as aplicações e serviços atendam às necessidades do usuário móvel, permitindo que ele seja eficaz, produtivo no uso do tempo e dos recursos e esteja satisfeito em relação aos atributos do sistema.

Ainda segundo Santos e Costa (2014), o paradigma do *design* de aplicativos para

smartphones não se difere do paradigma de aplicações desktop, mas possui alguns requisitos inerentes que o separam do desenvolvimento de aplicativos para dispositivos móveis comuns. Isto se deve principalmente ao pequeno teclado disponível para digitação nos *smartphones* e também ao limitado tamanho da tela, que torna obrigatório o desenvolvimento de um *layout* o mais amigável e simples possível.

Para a computação móvel, segundo Lee et al., (2005), a usabilidade de um dispositivo móvel depende de vários fatores, incluindo o usuário, o ambiente e as características do dispositivo (SOUZA; SPINOLA, 2006). Por exemplo:

- Características do usuário: a interação do usuário com um dispositivo móvel depende até certo ponto, de suas características pessoais como "flexibilidade e destreza";
- Características do ambiente: o ambiente do usuário afeta a escolha do dispositivo. Em "condições normais de funcionamento" um dispositivo móvel pode trabalhar sob as condições normais de trabalho do usuário, devendo trabalhar também em "condições extremas" (como calor, frio, umidade, seca luz natural e artificial);
- Características de dispositivos: os dispositivos móveis tem características próprias diferentes, que podem afetar a usabilidade total;

3.6 Materiais

Nesta seção serão abordadas as tecnologias que compuseram o ambiente de desenvolvimento deste trabalho.

3.6.1 IDE (Integrated Development Environment)

O IDE é um programa de computador, geralmente utilizado para aumentar a produtividade dos desenvolvedores de *software*, bem como a qualidade desses produtos (SANTOS, 2014b). Santos (2014b) cita ainda que as IDE's podem auxiliar, através de ferramentas e características, na redução de erros e na aplicação de técnicas como o RAD(Rapid Application Development). Para Kamal (2008) uma IDE consiste de simuladores, editores, compiladores e analisadores de lógica que fornecem um ambiente de desenvolvimento integrado.

Existem diversas IDE's disponíveis no mercado. Especificamente para este projeto, a escolhida foi a *IntelliJ IDEA*. Segundo o site da *IntelliJ IDEA*, esta IDE fornece um conjunto abrangente de recursos, incluindo ferramentas e integrações com as tecnologias modernas mais importantes e *frameworks* para empresas e desenvolvimento *web* utilizando Java, Scala, Groovy e outras linguagens. Porém, de acordo com uma avaliação própria, o que mais chamou atenção

nesta IDE para sua utilização foi seu ótimo suporte ao Grails, *autocomplete* super eficiente e uma ótima forma de organização dos arquivos gerados nas criações dos projetos. O *IntelliJ IDEA* foi utilizado na versão "*Ultimate*", que é sua versão mais completa, permitindo então retirar o máximo de produtividade desta ferramenta.

Figura 24 *– IntelliJ IDEA.*

3.6.2 PhpMyAdmin

O banco de dados escolhido para desenvolver este projeto foi o MySQL. Ele possui quase tudo que seus concorrentes mais renomados têm, com a vantagem de ser : gratuito (SANTOS, 2014a). Ainda segundo Santos (2014a), as principais vantagens do MySQL são: ótimo desempenho; segurança devido a diversas características como integridade referencial, *backup*, *restore*, controle de usuários e acessos; estabilidade e alta aplicabilidade tanto em sistemas para *Internet* quanto em sistemas *desktop*. A partir disto, visando acelerar o processo de desenvolvimento e proporcionar melhor experiência na manipulação dos dados gerenciados pelo MySQL, foi utilizado um *software* chamado *phpmyadmin*.

Segundo o site do desenvolvedor[1], o *phpmyadmin* é um *software* livre, escrito em php, destinado a lidar com a administração do MySQL através da *Web*(navegador). Completa ainda que o *phpmyadmin* suporta uma ampla gama de operações em MySQL. Por fim, o autor cita que operações frequentemente usadas (gestão de bases de dados, tabelas, colunas, relações, indices, permissões, etc) podem ser realizadas através da interface do usuário, possibilitando ainda que

[1] Disponível em: www.phpmyadmin.net/

qualquer declaração SQL possa ser executada diretamente através do *phpmyadmin*.

Figura 25 *– Phpmyadmin.*

O *phpmyadmin* foi de grande valia na construção deste projeto principalmente no que diz respeito a manipulação da base de dados de forma que ela se adeque ao que foi proposto para o projeto. Como a criação do banco de dados é de responsabilidade do *framework* utilizado, neste caso, o Grails, muitas vezes este banco de dados não corresponde ao que realmente se deseja. Por este motivo, é importante que a modelagem do banco de dados seja feita como parte da documentação do projeto e que se verifique, quando utilizado *framework*, se aquele banco de dados construído pelo *framework* corresponde ao banco de dados previsto na modelagem. Caso não, é necessário que esse banco de dados seja manipulado através, por exemplo, de uma ferramenta como o *phpmyadmin*, ou diretamente através da linguagem utilizada pelo banco de dados, que no caso do MySQL é o SQL.

3.6.3 Sistema Operacional Linux

Sistema Operacional é o componente de *software* que faz a *interface* básica entre os programas do usuário e o computador, gerenciando itens como os recursos e periféricos (e. g. memória, discos, arquivos, usuários, impressoras), segurança, privilégios, comunicação e outros (CAMPOS, 2003). Segundo Zon et al. (2007), Linux é um sistema operacional livre, um dos mais conhecidos exemplos de desenvolvimento com código aberto e de *software* livre e seu código fonte está disponível sob licença GPL para qualquer pessoa utilizar, estudar, modificar e distribuir livremente. Complementando, o *Linux* é inspirado pelo sistema operacional *Unix*, o qual apareceu em 1969, e continua em uso e desenvolvimento contínuo desde então (PROJECT, 2004).

Existem várias vantagens em utilizar o sistema operacional *Linux*, as principais delas, segundo Zon et al. (2007) são: preço (gratuito); estabilidade; interface amigável; principais aplicativos são disponíveis para *Linux*; vasto material de apoio na internet; não vulnerável a *vírus*;

possibilidade de uso para usuários avançados. Entretanto, existem também algumas desvantagens em utilizar o *Linux*, estas também citadas por Zon et al. (2007): instalação e configuração não tão simples; falta de padronização; instalação e remoção de aplicativos podem ser feitas de várias maneiras, podendo confundir o usuário e falta de aplicativos específicos são as principais.

Existem diversas distribuições *Linux*, dentre elas destacam-se o *Ubuntu, Slackware, Red Hat, Debian* e etc. O escolhido para desenvolvimento deste projeto foi o *Ubuntu* pelo vasto material de apoio disponível através da internet e pelo fato da familiaridade já existente com essa distribuição.

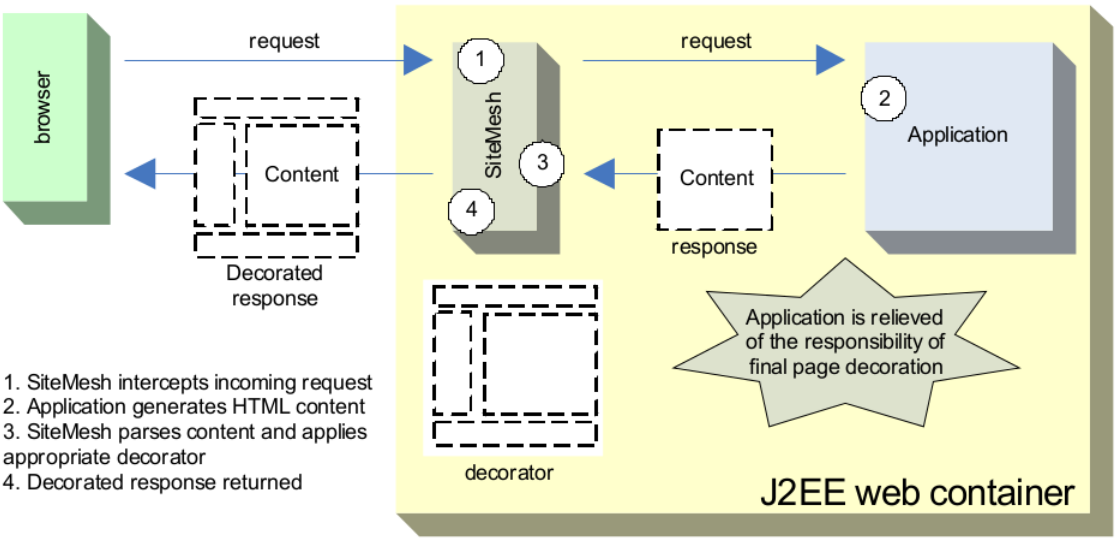

Figura 4 *– Funcionamento do SiteMesh.*

4 Desenvolvimento

Neste capítulo poderá ser observada a metodologia de desenvolvimento utilizada neste projeto. Ele conta ainda com um detalhamento a respeito do projeto desenvolvido, desde a análise de sistema, que neste caso, foi composta por análise de requisitos, diagrama de classes e dicionário de dados até as etapas que se seguem à conclusão do projeto.

De acordo com o que foi proposto para este trabalho, foram selecionadas algumas tecnologias como por exemplo o *framework* Grails, Sistema Operacional Android e utilização de *Web Services*. Estas tecnologias já foram detalhadas no capítulo anterior deste documento. Além disto, foram estudadas ainda técnicas de IHC para melhor usabilidade da aplicação.

É neste capítulo que poderá ser verificado também como essas tecnologias interagem para que todas as funcionalidades previstas fossem corretamente implementadas e ainda algumas vantagens da sua utilização neste caso específico.

4.1 O Sistema

O projeto consiste em dois módulos distintos que trabalharão em conjunto com o objetivo de agilizar o atendimento médico. Além disso, o projeto contempla o fato de que todos os dados dos pacientes serão armazenados em um banco de dados, como um histórico, os quais ficarão disponíveis aos respectivos responsáveis.

O primeiro módulo consiste de um aplicativo móvel, que será disponibilizado para *tablets* com sistema operacional android e com tamanho de tela de 7 polegadas. Este módulo será operado pelo médico. Já o segundo módulo é um sistema *Web*, desenvolvido em Grails, que será operado pela administração/recepção do hospital. Os dois módulos se comunicarão através de um *Web Service* do tipo *RESTful* e trabalharão em conjunto, trocando informações, a fim de agilizar o atendimento médico, e manter um histórico consistente a respeito dos pacientes para posterior utilização, por exemplo, na geração de relatórios.

4.1.1 Análise do Sistema Proposto

A primeira etapa que deve ser realizada para a construção de um *software* é a análise de sistemas. A análise de sistemas representa o estudo detalhado de uma área de trabalho (processo), que antecede uma ação que, quase sempre, implica no desenvolvimento de um conjunto de programas integrados (sistema) destinado à execução controle e acompanhamento do processo (JUNIOR, 1997). É na análise de sistemas que é determinado, por exemplo, o que o

sistema deve fazer, quais funcionalidades deve contemplar , onde e como ele vai operar. A um nível de abstração grande, um processo de análise de sistemas pode ser descrito como a obtenção/definição, análise e negociação, documentação e validação de requisitos (ROCHA, 2008).

Ainda segundo Rocha (2008), a obtenção/definição é a etapa em que os requisitos do sistema são descobertos por meio de consultas aos utilizadores, a documentos de mercado e etc. A etapa de análise e negociação é a etapa em que os requisitos são analisados em detalhe para decidir quais desses requisitos serão aceitos. A terceira etapa, documentação, consiste em modelar e documentar os requisitos em um nível de detalhe apropriado. Por fim, ocorre a etapa de validação que consiste em verificar cuidadosamente a coerência e totalidade dos requisitos.

Para efeito de exemplificação, dentro da análise de sistemas, foram elaborados apenas três documentos para auxiliar na implementação: o documento de requisitos, o diagrama de classes e o dicionário de dados.

Documento de Requisitos

O documento de requisitos é a especificação oficial dos requisitos do sistema para clientes, usuários finais e desenvolvedores de *software* (PEREIRA; LUCENA, 2005). Um documento de requisitos de *software* precisa ser claro, consistente e completo, porque esse documento servirá de referência aos desenvolvedores, gerente de projeto, engenheiros de *software* (responsáveis pelos testes e manutenção do sistema), além de servir de base para definir o escopo das funcionalidades a serem registradas num contrato (FILHO, 2005). É no documento de requisitos que estarão declaradas os principais requisitos do sistema a ser desenvolvido. Os requisitos dos sistemas devem estabelecer o que o sistema deve fazer ao invés de como isto será feito e as circunstâncias sob as quais deve operar (CARVALHO, 2014b). Existem dois tipos principais de requisitos: os funcionais e os não funcionais.

Os requisitos funcionais descrevem as funções, tarefas e sub tarefas que se espera que o sistema realize. Incluem aquelas coisas que os utilizadores e analistas de sistemas esperam que o sistema faça (ROCHA, 2008). Basicamente os requisitos funcionais são, de fato, as funcionalidades que o sistema pode realizar, como por exemplo, cálculo das despesas do mês, soma de dois números. Geralmente eles recebem entradas do usuário, processam essas entradas e geram uma saída, que pode ser uma mudança de interface ou até mesmo um sinal sonoro. Uma das formas de organizar os requisitos funcionais é através de tabelas. Logo a seguir (Tabelas 1 e 2), podemos observar, de forma simplificada, alguns dos principais requisitos funcionais e licitados para o sistema proposto pelo projeto. É importante ressaltar que como o projeto consiste em dois módulos, um aplicativo para dispositivo móvel e um sistema *Web*, foram elaborados requisitos funcionais, separadamente, para cada módulo. A partir daqui, chamaremos de “Módulo

1", o módulo desenvolvido para Android, e "Módulo 2", o módulo desenvolvido em Grails.

CÓDIGO	NOME	DESCRIÇÃO	PRIORIDADE	ENTRADA	SAÍDA
RF001	Iniciar aplicação.	Ao clicar no botão "Iniciar", a interface do médico é inicializada.	Essencial	Um clique no 'botão "Iniciar".	A tela inicial é encerrada e a tela de seleção de médico é inicializada.
RF002	Obtenção de médicos do servidor.	A lista de todos os médicos cadastrados no banco de dados são recuperados pelo "Módulo 2" e repassados para o "Módulo 1" através do *Web Service*.	Importante	Requisição para obtenção da lista dos médicos cadastrados no sistema.	Lista com todos os médicos obtidos do banco de dados são apresentados na tela do dispositivo móvel.
RF003	Autenticação.	O médico seleciona seu CRM e entra com sua senha pessoal para que possa ter acesso às funcionalidade da aplicação.	Essencial	CRM e Senha do médico a ser autenticado na aplicação.	O médico é autorizado a acessar as demais funcionalidades do sistema.
RF004	Buscar paciente.	O médico busca o paciente no servidor através do seu CPF.	Essencial	CPF ou número do cartão do SUS do paciente.	O "Módulo 2" envia para o aplicativo móvel os dados do paciente em questão.
RF005	Nova consulta.	O médico pode efetuar uma nova consulta que será armazenada no banco de dados do servidor.	Essencial	Um clique no botão "Nova Consulta".	A tela de "Nova consulta" é inicializada.

RF006	Selecionar sintoma.	O médico seleciona, no aplicativo móvel, os respectivos sintomas alegados pelo paciente.	Essencial	Um clique na parte do corpo referente ao sintoma alegado pelo paciente.	Todos os sintomas de determinada parte do corpo são listadas para seleção na tela do dispositivo móvel.
RF007	Solicitar exame.	O médico seleciona os respectivos exames que deseja solicitar ao paciente.	Essencial	Lista dos exames que serão solicitados ao paciente.	Os exames são enviados para o "Módulo 2" e salvos no banco de dados.

Tabela 1 *– Módulo 1: Tabela de Requisitos Funcionais.*

CÓDIGO	**NOME**	**DESCRIÇÃO**	**PRIORIDADE**	**ENTRADA**	**SAÍDA**
RF001	Cadastrar Médico.	Cadastra um médico no sistema.	Essencial	Dados do médico a ser cadastrado.	Um médico é cadastrado no sistema.
RF002	Cadastrar Sintoma.	Cadastra um sintoma no sistema.	Essencial.	Dados do sintoma a ser cadastrado.	Um sintoma é cadastrado no sistema.
RF003	Cadastrar Patologia.	Cadastra uma patologia no sistema.	Essencial.	Dados da patologia a ser cadastrada no sistema.	Uma patologia é cadastrada no sistema.
RF004	Cadastrar Exame.	Cadastra um exame no sistema.	Essencial	Dados do exame a ser cadastrado no sistema.	Um exame é cadastrado no sistema.

RF005	Associar Sintomas aos Exames.	Associa um sintoma a respectivos exames.	Essencial	Dados do sintoma e do exame que serão associados.	Um sintoma é associado a determinado exame.
RF006	Associar Sintomas a Patologias.	Associa um sintoma a respectivas possíveis patologias.	Essencial	Dados do sintoma e da patologia que serão associados.	Um sintoma é associado a uma determinada patologia.
RF007	Cadastrar Consulta.	Cadastra uma consulta no sistema salvando-a no banco de dados.	Essencial	Dados da consulta a serem cadastrados no sistema.	Uma consulta é cadastrada no sistema.
RF008	Recuperar médicos cadastrados.	Todos os médicos cadastrados no sistema são recuperados do banco de dados e enviados no formado JSON para o dispositivo móvel.	Essencial	Solicitação do aplicativo móvel para recuperar os médicos cadastrados no sistema.	Os médicos cadastrados no sistema são recuperados e enviados ao dispositivo móvel.

***Tabela 2** – Módulo 2: Tabela de Requisitos Funcionais.*

Os requisitos não funcionais também são muito importantes no desenvolvimento de um sistema. Segundo Rocha (2008), eles colocam restrições no sistema a desenvolver e no processo a usar, e especificam as restrições externas às quais o produto deve obedecer. Rocha (2008) completa ainda que requisitos não funcionais incluem, entre outros, requisitos de confiabilidade, segurança, adaptabilidade, portabilidade e desempenho. A seguir (Tabelas 3 e 4) podem ser observados alguns dos requisitos não funcionais e licitados para o projeto.

CÓDIGO	**REQUISITO NÃO FUNCIONAL**	**DESCRIÇÃO**
RNF001	Operacionais.	O sistema deve rodar em dispositivos móveis que utilizam o sistema operacional Android, e possuem tela de 7 polegadas.

RNF002	Segurança.	Apenas os médicos cadastrados e autenticados poderão utilizar o aplicativo móvel.
RNF003	Desempenho.	Para melhor experiência de usabilidade, a transição entre as telas não deve ser superior a 2 segundos.

Tabela 3 *– Módulo 1: Tabela de Requisitos Não Funcionais.*

CÓDIGO	**REQUISITO NÃO FUNCIONAL**	**DESCRIÇÃO**
RNF001	Operacionais.	Como trata-se de um sistema *web*, ele deverá ser multi plataforma, e se adaptar aos principais *browsers* utilizados atualmente.
RNF002	Segurança.	Apenas pessoal devidamente cadastrado e autenticado no sistema deverá ter acesso as funcionalidades do sistema.

Tabela 4 *– Módulo 2: Tabela de Requisitos Não Funcionais.*

Diagrama de Classes

O diagrama de classes denota a estrutura estática de um sistema, isto é, as classes, os relacionamentos entre suas instâncias (objetos), restrições e hierarquias (FALBO, 2002). Falbo (2002) completa ainda que o diagrama de classes é considerado estático pois a estrutura descrita é sempre válida em qualquer ponto no ciclo de vida do sistema. No diagrama de classes podem ser definidos: o nome da classe, os atributos da classe, ou seja, as propriedades para as respectivas classes e ainda os métodos que serão implementados por aquela classe. A seguir (Figura 26), pode-se observar o diagrama de classes elaborado para o “Módulo 2” do projeto

proposto.

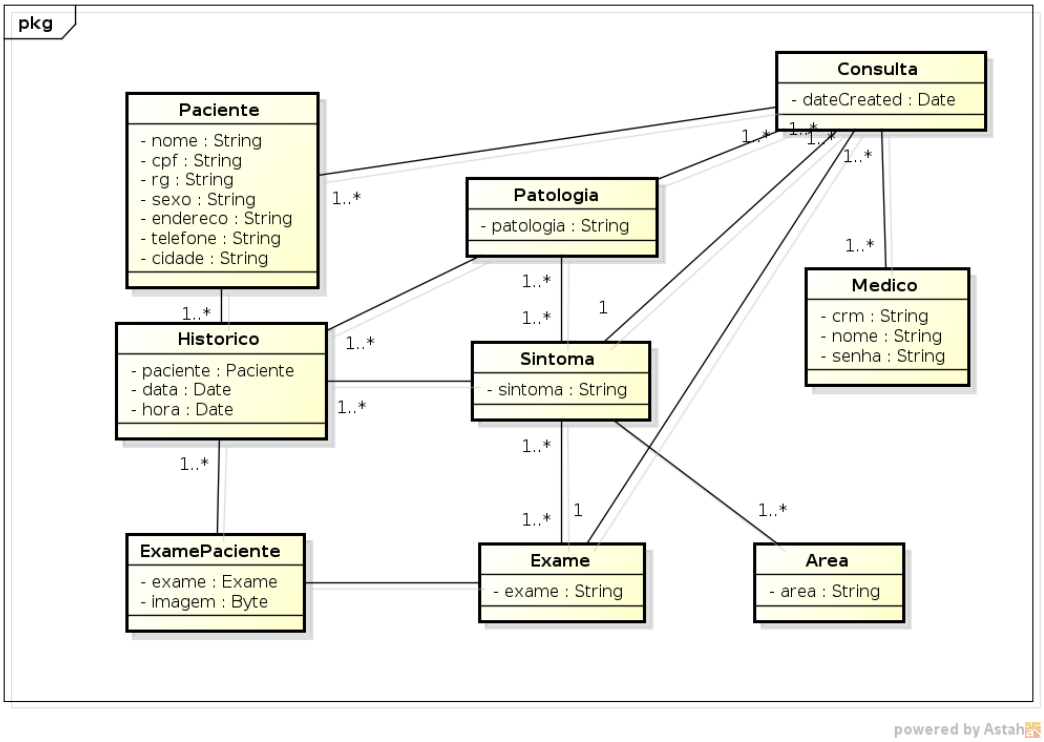

Figura 26 *– Diagrama de Classes: Módulo 2.*

Dicionário de Dados

O Dicionário de Dados é uma listagem organizada de todos os elementos de dados pertinentes ao sistema, com definições precisas para que os usuários e desenvolvedores possam conhecer o significado de todos os itens de dados manipulados pelo sistema (FALBO, 2002). Existem várias formas de se representar o dicionário de dados. A seguir, pode-se observar a apresentação de uma pequena parte do dicionário de dados elaborado para o "Módulo 2" deste projeto.

Paciente: tabela que representa os detalhes dos pacientes que serão cadastrados no sistema.

1. nome – guarda o nome do paciente;
2. cpf – guarda o cpf do paciente;
3. rg – guarda o rg do paciente;
4. sexo – guarda o gênero do paciente
5. endereco – guarda o endereço do paciente;
6. telefone – guarda o telefone do paciente;
7. telefone – guarda o telefone do paciente;

Medico: tabela que representa os detalhes dos médicos que serão cadastrados no sistema.

1. crm – guarda o número do crm do médico;
2. nome – guarda o nome do médico;
3. senha – guarda a senha cadastrada pelo médico.
4. sexo – guarda o gênero do paciente

Histórico: tabela que representa os detalhes de todas as consultas que serão realizadas.

1. paciente – guarda o respectivo paciente no qual foi realizada a consulta;
2. data – guarda a data em que a consulta foi realizada;
3. hora – guarda a hora em que a consulta foi realizada.

4.1.2 Etapas do Desenvolvimento

Como o sistema é composto de dois módulos interdependentes e existia apenas um desenvolvedor responsável, estes dois módulos foram desenvolvidos em paralelo, sendo que a alternância de desenvolvimento se dava sempre que um módulo dependia exclusivamente do outro para completar seu funcionamento. Para melhor entender como isso acontece, vamos a um exemplo. Tomamos uma das primeiras funcionalidade que foram desenvolvidas para o "Módulo 1", o de autenticação do médico no aplicativo. Para que esse requisito funcional fosse contemplado, seria necessário que já estivesse implementada, no "Módulo 2", a funcionalidade de receber os parâmetros enviados pelo "Módulo 1", fazer a comparação no banco de dados e retornar uma confirmação para o "Módulo 1" de que aquele médico está devidamente cadastrado, seu CRM e senha conferem no banco de dados, e que ele pode acessar as demais funcionalidades da aplicação. Logo, o desenvolvimento era alternado para o "Módulo 2" para implementação desta funcionalidade pendente para que se possa completar o requisito. Devido a complexidade dos módulos a serem desenvolvidos e a necessidade de testes em paralelo ao desenvolvimento, esta foi a principal tática utilizada no decorrer do projeto.

Para melhor compreensão das etapas de desenvolvimento, a implementação das principais funcionalidades serão novamente dividas por módulo. Iniciando pelo desenvolvimento do "Módulo 1".

Módulo 1

Como o "Módulo 1" consistia no desenvolvimento de um aplicativo para Android, a ordem de construção deste artefato se deu da seguinte maneira: primeiro era detectada a funcionalidade a ser implementada, posteriormente era construída a tela para contemplar aquela

funcionalidade e por fim, a funcionalidade era implementada na "*Activity*". Para melhor organização das classes, os pacotes foram organizados de acordo com a figura 27.

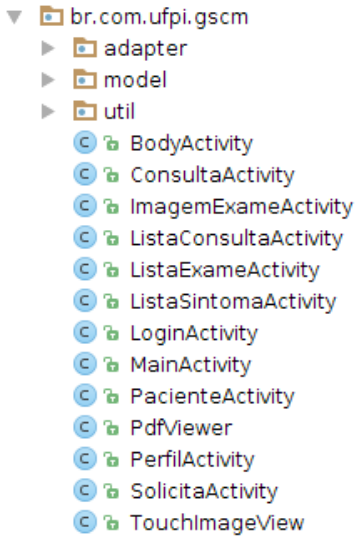

Figura 27 *– Pacotes: Módulo 1.*

Podemos observar através da imagem que existe um pacote "pai", que engloba três outros pacotes, mais as *activities*, que são basicamente as classes que darão funcionalidades ao sistema e determinam o fluxo de operações. O pacote "*adapter*" possui somente classes responsáveis por adaptar os diferentes tipos de listas (médicos, sintomas, exames) que são recebidos do servidor pelo aplicativo em listas visíveis na tela para o usuário. O pacote "*model*" serve basicamente como o "M" do modelo MVC, ou seja, este pacote contém as classes que servem como os objetos da aplicação (JONATHAN, 2011). O terceiro pacote, "*útil*", contém classes que desempenham os mais variados papéis, como por exemplo, a classe responsável pela conexão com o *web servisse* e os "*converters*", que são classes criadas exclusivamente com o objetivo de converter as listas provindas do servidor em formato JSON para objetos. As demais classes presentes no pacote "pai" são as *activities* da aplicação. De modo bem informal, podemos dizer que cada *activity* corresponde a uma tela da aplicação. A seguir, será explicado, de forma breve, a que funcionalidade cada *activity* corresponde.

- *MainActivity*: é a tela inicial da aplicação. Sua única funcionalidade é iniciar a aplicação;
- *LoginActivity*: é a tela onde o médico efetuará seu *login* na aplicação. É ela a responsável por não permitir que pessoas não devidamente cadastradas como médico no sistema possam acessar informações a respeitos de pacientes e consultas;
- *PacienteActivity*: é a tela onde o médico irá entrar com os dados do paciente (CPF) para que seu histórico seja devidamente recuperado do servidor;

- *PerfilActivity*: como o próprio nome já diz, nesta tela será mostrado um "perfil" para o paciente, através da qual poderão ser visualizados as ultimas consultas, últimos sintomas, últimos exames solicitados e poderão ainda ser acessadas outras funcionalidades como realizar nova consulta;
- *BodyActivity*: é através desta *activity* que o médico selecionará os sintomas relatados pelo paciente;
- *ConsultaActivity*: nesta *activity* poderão ser visualizados os detalhes referentes a cada consulta separadamente;
- *ListaConsultaActivity*: nesta *activity* poderão ser visualizadas todas as consultas referente ao respectivo paciente;
- *ListaExameActivity*: nesta *activity* poderão ser visualizados todos os exames solicitados pelo médico em uma determinada consulta;
- *ImagemExameActivity*: nesta tela, o médico poderá tirar uma fotografia do resultado do exame do paciente, e anexar essa imagem ao seu histórico de consulta;
- *ListaSintomaActivity*: a partir desta *activity* poderão ser visualizados todos os sintomas referentes a determinada parte do corpo escolhida pelo médico;
- *SolicitaActivity*: é basicamente a tela final de consulta. É através dela que o médico poderá solicitar os exames ao paciente.

Para concluir a criação de uma "tela" para uma aplicação Android, além da *activity*, é necessário ainda que este tela seja construída utilizando o padrão de criação de telas do Android. As telas das aplicações Android, por padrão, são "desenhadas" através de arquivos XML, contendo código XML e ficam localizados dentro do diretório "*layout*". O formato XML já foi abordado em capítulos anteriores deste documento. Na figura 28, podemos observar os princi-

pais arquivos XML responsáveis pela criação das telas.

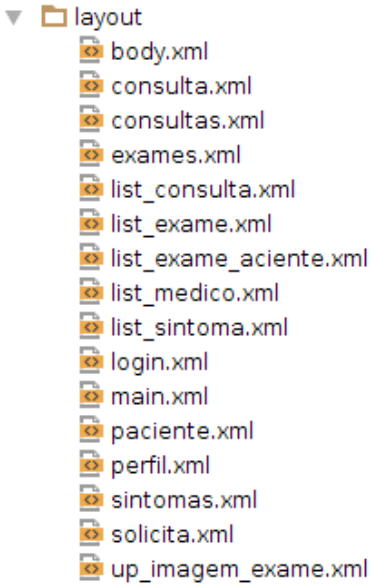

Figura 28 *– Módulo 1: Arquivos de Layout.*

Cada arquivo XML dentro do diretório "*layout*" é responsável por renderizar uma tela ou parte de uma tela de uma aplicação Android. A seguir, será explicado, a que *activity* alguns desses *layouts* correspondem.

- body.xml: esta tela está associada à "*BodyActivity*";
- consulta.xml: esta tela está associada à "*ConsultaActivity*";
- consultas.xml: esta tela está associada à "ListaConsultasActivity";
- exames.xml: esta tela está associada à "*ListaExameActivity*";
- list consulta.xml: esta arquivo renderiza apenas um componente em forma de lista que pode ser utilizado em diferentes telas da aplicação;
- login.xml: esta tela está associada à "*LoginActivity*";
- paciente.xml: está tela está associada à "*PacienteActivity*";
- perfil.xml: esta tela está associada à "*PerfilActivity*".

Sabe-se que uma tela de uma aplicação Android é composta basicamente de uma *activity*, que dá funcionalidade a tela, e um arquivo XML, que renderiza o *layout* tela. A partir disto, é importante também saber que uma *activity* é associada a um arquivo XML através do

comando "*setContentView*". Na figura 29, podemos observar a utilização deste método para associar uma *activity* a um arquivo de *layout*.

```
setContentView(R.layout.main);
```

Figura 29 *– Utilização do comando "setContentView".*

Todas as etapas de desenvolvimento do "Módulo 1" foram bastante importantes e essenciais afim de que o projeto realmente atendesse o objetivo para o qual foi proposto. Porém, existem algumas funcionalidades que provêem um certo nível de particularidade a esta aplicação. A primeira particularidade diz respeito à implementação da comunicação com o servidor. Esta funcionalidade é um dos requisitos mais importantes da aplicação. Primeiramente é importante ter noção que para acessar um serviço através da internet, a aplicação deve ter permissão de acesso a internet. Esta permissão é solicitada através do arquivo *manifest*, que já foi abordado neste documento. A Figura 30 a seguir mostra como declarar esta permissão no *manifest*.

```
<uses-permission android:name="android.permission.INTERNET"/>
```

Figura 30 *– Declaração de permissão de Internet no Manifest.*

Após obter permissão de utilizar a internet é hora de utilizar os protocolos padrões da internet para obter acesso ao servidor o qual está rodando o *web servisse*. A figura 31 mostra a

classe “WebCliente”, a qual implementa esta funcionalidade.

```
/**
 * Created by andreimaxwel on 17/02/14.
 */
public class WebClient {

    private static final int TAM_MAX_BUFFER = 10240;
    String urlServer;

    public WebClient(String urlServer) {
        this.urlServer = urlServer;
    }

    public String post(String json) {
        try {

            DefaultHttpClient httpClient = new DefaultHttpClient();

            HttpPost post = new HttpPost(urlServer);
            post.setEntity(new StringEntity(json));


            post.setHeader("Accept", "application/json");
            post.setHeader("Content-type", "application/json");

            HttpResponse response = httpClient.execute(post);

            String responseJSON = EntityUtils.toString( response.getEntity() );
            if (response.getStatusLine().getStatusCode() == 404) {
                responseJSON = "";
            }
            httpClient.getConnectionManager().shutdown();

            return responseJSON;
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
}
```

Figura 31 *– Classe “WebClient“.*

Na Figura 31, pode-se observar inicialmente a criação de um cliente HTTP através da classe “*DefaultHttpClient*”. Posteriormente é criado uma instância da classe “*HttpPost*”, que é um método de solicitação para que o servidor aceite a entidade anexada à requisição através do método “*setEntity*”. Neste caso, a entidade é composta de uma “*String*” no formato JSON. Logo após são “setados” os cabeçalhos para informar ao servidor que a entidade está em formato JSON. Por fim, a requisição é executada e a resposta do servidor é recebida através de um “*HttpResponse*”. A conexão é então encerrada e o resultado da requisição é retornado para o método que a solicitou.

Outra particularidade deste projeto é a funcionalidade de tirar uma foto do resultado do exame através da câmera do dispositivo e a enviar para que o servidor a armazene. Além disto, a aplicação pode ainda solicitar essa imagem novamente ao servidor. Para que esta funcionalidade seja implementada, além da permissão de internet, se torna necessário ainda a permissão para que o aplicativo possa escrever no cartão de memória do celular. A Figura 32 mostra como esta

permissão pode ser obtida no arquivo *manifest*.

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

Figura 32 *– Declaração de permissão para escrever no cartão de memória do dispositivo.*

Após obtida a devida permissão, é hora de instruir a aplicação a utilizar a câmera do dispositivo para obter a foto. Para isto, basta utilizar a intent demonstrada na Figura 33.

```
Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
startActivityForResult(intent, TAKE_PICTURE);
```

Figura 33 *– Exemplo de utilização da "intent" para inicializar a câmera do dispositivo.*

A *intene* "MediaStore.ACTION_IMAGE_CAPTURE" é responsável por inicializar o aplicativo padrão do dispositivo android responsável pela câmera do aparelho. O comando seguinte, "*startActivityForResult*" é utilizado para obter o status que a aplicação padrão da câmera retornou para a aplicação que a invocou. Para isto, é necessário sobrescrever o método "*onActivityResult*", que é mostrado na Figura 34.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == TAKE_PICTURE) {
        if (resultCode == RESULT_OK) {
            bitmap = (Bitmap) data.getExtras().get("data");
            img_exame.setImageBitmap(bitmap);
        }
    }
}
```

Figura 34 *– Sobrescrevendo o método "onActivityResult".*

É este método que recebe de volta o status da câmera. Isto é importante para saber se o usuário realmente tirou alguma foto com o aparelho, ou simplesmente cancelou a aplicação. Caso o usuário tenha tirado uma foto, o "*resultCode*" da aplicação padrão retornará "RESULT_OK". Desta forma, a imagem é apta a ser enviada ao servidor. A partir deste ponto, basta enviar a imagem para o servidor através da classe "*WebCliente*", que já foi explicada anteriormente.

Módulo 2

O “Módulo 2” consiste em um sistema *Web*, que foi desenvolvido em Grails. Como este *framework* segue o modelo MVC, que já foi explanado anteriormente, logo segue este padrão de desenvolvimento. Normalmente são criados os “*models*”, no caso do Grails, as classes de domínio, posteriormente são gerados as “*views*” e “*controllers*”, que serão detalhados mais a frente.

Como há pouco citado, primeiramente foram criadas as classes de domínio. A seguir (Figura 35), pode-se observar uma imagem contendo todas as classes de domínio que foram criadas.

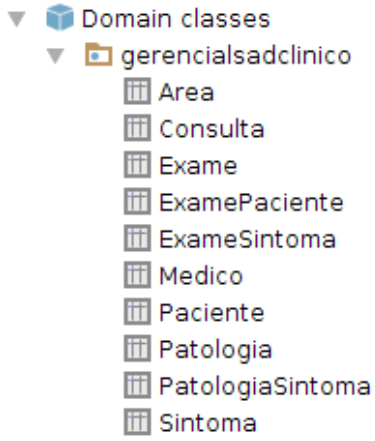

Figura 35 *– Módulo 2: Classes de Domínio.*

Para conhecer melhor cada classe de domínio criada, a seguir são detalhadas cada uma delas.

- Area: classe de domínio responsável por manipular os dados das áreas do corpo humano, como por exemplo, perna, braço, cabeça, etc;
- Consulta: classe de domínio responsável por manipular os dados das consultas;
- Exame: classe de domínio responsável por manipular os dados dos exames;
- ExamePaciente: esta classe de domínio foi criada com o objetivo de relacionar os exames aos pacientes;
- ExameSintoma: esta classe de domínio foi criada com o objetivo de relacionar os exames aos sintomas;
- Medico: classe de domínio responsável por manipular os dados médicos;

- Paciente: classe de domínio criada para manipular os dados dos pacientes;
- Patologia: classe de domínio criada para manipular os dados das patologias;
- PatologiaSintoma: classe de domínio responsável por relacionar as patologias aos sintomas;
- Sintoma: classe de domínio criada para manipular os dados dos sintomas.

Como o *framework* Grails utiliza o GORM, que já foi abordado anteriormente, estas classes de domínio não refletem exatamente desta forma no banco de dados. Através do "*phpmyadmin*", na Figura 25, podemos observar como o *framework* organizou estas classes, no banco de dados, em forma de tabelas.

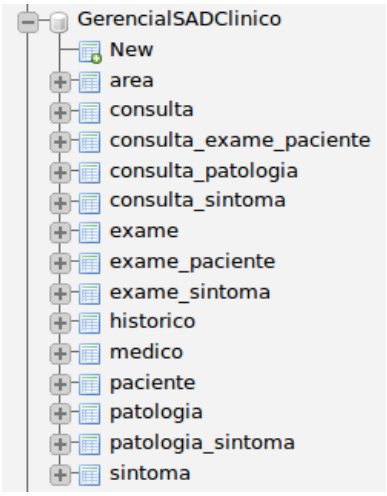

Figura 36 *– Tabelas criadas pelo Grails, visualizadas através do Phpmyadmin.*

Existem dez classes de domínio e quatorze tabelas criadas. Isto se deve ao fato de que as relações entre as classes de domínio são definidas justamente através do GORM, e isto reflete diretamente na criação das tabelas do banco de dados. Por exemplo, a relação entre as classes de domínio "Consulta" e "ExamePaciente" geraram uma nova tabela chamada de "Consulta_Exame_Paciente". Esta relação se deu através de um atributo "*hasMany*" definido na classe de domínio "Consulta". Este atributo "*hasMany*" é utilizado em casos de relação do tipo "um para muitos". Por isto, é importante que se tenha noção do GORM antes de criar as classes de domínio. Além disto, o GORM não é perfeito, ou seja, em alguns casos ele poderá falhar. A partir disto, vale a experiência do desenvolvedor em ajustar as tabelas da melhor forma que atenda o propósito ou funcionalidades do sistema.

Após a criação das classes de domínio é hora de gerar as "*views*" e "*controllers*". No Grails isto é muito simples. Basta utilizar o comando "*grails generate-all* Classe_De_Domi-

nio", onde "Classe_De_Dominio" é a classe de domínio a qual se deseja gerar os controladores e visões. A seguir (Figura 37), podemos observar, a exemplo, o controlador "Area" e suas respectivas "*actions*".

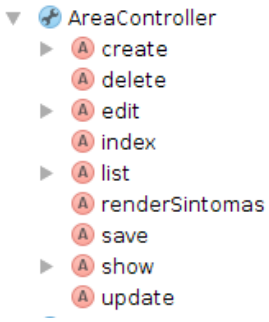

Figura 37 *– Controlador "Área" e suas actions.*

Por padrão, o Grails cria as "*actions*" responsáveis pelo CRUD(*Create, Read, Update e Delete*). As outras "*actions*" são definidas de acordo com as necessidades de cada aplicação. Por padrão, o Grails também cria as visualizações para estes métodos. A seguir (Figura 38) podemos observar os arquivos de visão gerados pelo "*generate-all*".

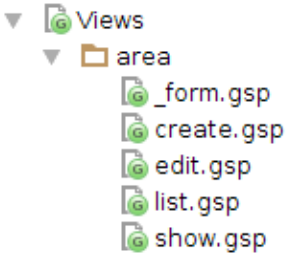

Figura 38 *– Arquivos de visão gerados pelo "generate-all".*

Todas as outras classes de domínio seguiram esta ordem de implementação e foram incrementadas de acordo com as necessidades exigidas pelo sistema. É importante ressaltar que, como *framework* de desenvolvimento ágil, o Grails não abrange todas as funcionalidades de qualquer sistema específico. Essas funcionalidades extras são de responsabilidade do desenvolvedor. O Grails apenas cria o que, por padrão, todo projeto de sistema *web* bem estruturado utiliza.

Também existiram inúmeras particularidades no desenvolvimento do "Módulo 2", afinal, cada *software* é diferente um do outro e possui suas especificidades. Um dos principais requisitos funcionais do sistema desenvolvido diz respeito a tornar grande parte dos dados armazenados no banco de dados disponível através de uma URL em forma de serviço. É a partir

destes serviços disponíveis através da internet que a aplicação móvel irá operar. Felizmente o Grails é um *framework* muito completo e fornece uma grande quantidade de recursos para lidar com *web services*. Podemos tomar como exemplo o serviço disponibilizado pelo sistema, que devolve, em formato JSON, todos os sintomas referentes a uma determinada área. Esta área é enviada pela aplicação móvel para o serviço, e a partir desta área o sistema devolve os sintomas relacionados a ela. Na figura 39, podemos observar a implementação desta funcionalidade.

```
def renderSintomas() {

    JSONObject jsonObject = request.JSON;
    int id = Long.parseLong(jsonObject.get("id").toString())
    def areaInstance = Area.get(id)

    if (areaInstance) {
        def sintomaInstanceList = Sintoma.findAllByArea(areaInstance)
        if (sintomaInstanceList) {
            render sintomaInstanceList as JSON
        } else {
            render "Sintomas não encontrados."
        }
    } else {
        render "Área do sintoma não encontrada."
    }

}
```

Figura 39 *– Implementação do método que envia todos os sintomas para o "Módulo 1".*

Como já abordado em capitulo anterior, os métodos em Grails são definidos através de *closures*, que também podem ser chamadas de ações do sistema. Como pode-se visualizar através da imagem a *action* "renderSintomas" recebe um "*JSONObject*" enviado pelo aplicativo móvel. Este JSON nada mais é que um objeto que pode ser manipulado mais facilmente através da classe "*JSONObject*". Este "*JSONObject*" traz como conteúdo o "id" de uma respectiva área do corpo humano, para que se possa recuperar do banco de dados todos os sintomas referentes a esta área. Caso os sintomas tenham sido recuperados com sucesso do banco de dados, eles são enviados de volta à aplicação móvel através do comando "render as JSON", que se encarrega do trabalho de encapsular os dados no formato JSON.

Outra particularidade bastante interessante utilizada no desenvolvimento do sistema *Web*, foi a implementação da funcionalidade de relacionar um exame a um sintoma, ou relacionar uma patologia a um sintoma. Esta funcionalidade é bastante interessante porque foi desenvolvida pensando em facilitar o máximo este processo de associação, que em alguns momentos pode se tornar bastante confuso. Para exemplo, tomaremos a funcionalidade de relacionar um exame a um sintoma. Ela funciona da seguinte forma: primeiramente o usuário acessa a página de relacionamento de exame a sintoma. A partir daí, existe um campo de texto que é dotado de autocomplete, que é bastante interessante para facilitar e agilizar processos. Ao digitar as primeiras letras no campo de autocomplete os respectivos sintomas que possuem aquelas letras

são gradativamente apresentados ao usuário, permitindo que, mesmo que o usuário não lembre perfeitamente o nome completo do sintoma, ele possa ser induzido ao sintoma correto através do recurso de auto completar. Selecionado o devido sintoma, basta clicar no botão adicionar e pronto, exame e sintoma são relacionados sem ter que recarregar a página.

Este recurso é bastante interessante no Grails e é implementado graças ao "*submitToRemote*", que faz uma solicitação ao servidor/controlador sem que seja necessário atualizar a página. A Figura 40 mostra a utilização deste recurso.

```
<g:submitToRemote url="[action:'adicionarSintomas']" update="lista" value="Adicionar"/>
```

Figura 40 *– Exemplo de utilização do "submitToRemote".*

É possível notar através da imagem que são necessário alguns parâmetros para que se possa utilizar este recurso do Grails. O primeiro deles é a "url", que identifica qual "*action*" o controlador vai utilizar para essa requisição. O outro parâmetro é o "*update*", que indica qual parte da página será alterada quando a requisição for completada. É devido a isto que não é necessária o recarregamento da página para que a relação seja concluída.

Para completar as particularidades encontradas no decorrer deste projeto e relacionadas ao desenvolvimento tanto do "Módulo 1" quando do "Módulo 2", podemos citar o fato de que a imagem do resultado do exame, na verdade não é armazenada em forma de imagem no servidor. Seus bytes são convertidos para o formato de "*String*" para melhor equiparar as diferenças entre o Android e o Grails.

4.1.3 Funcionamento

Nesta seção serão explicados detalhadamente o funcionamento dos dois módulos que compõem o projeto. Novamente, para facilitar o compreendimento, eles serão separados em "Módulo 1" e "Módulo 2", como explicado anteriormente.

Módulo 1

O "Módulo 1" é um aplicativo para Android que rodará em *tablets* de 7 polegadas. Para testes durante o desenvolvimento desta aplicação foi utilizado o Android na versão 4.4, chamada de "*KitKat*". Estes testes foram realizados através de emulador presente no kit de desenvolvimento Android, o SDK, que nada mais é que um conjunto de bibliotecas e ferramentas utilizadas para desenvolver, testar e depurar aplicativos Android.

Partindo ao que interessa, na Figura 41, pode-se visualizar a tela inicial da aplicação, que é bem simples, contendo apenas um botão que inicia, de fato, a primeira tela de funcionalidade.

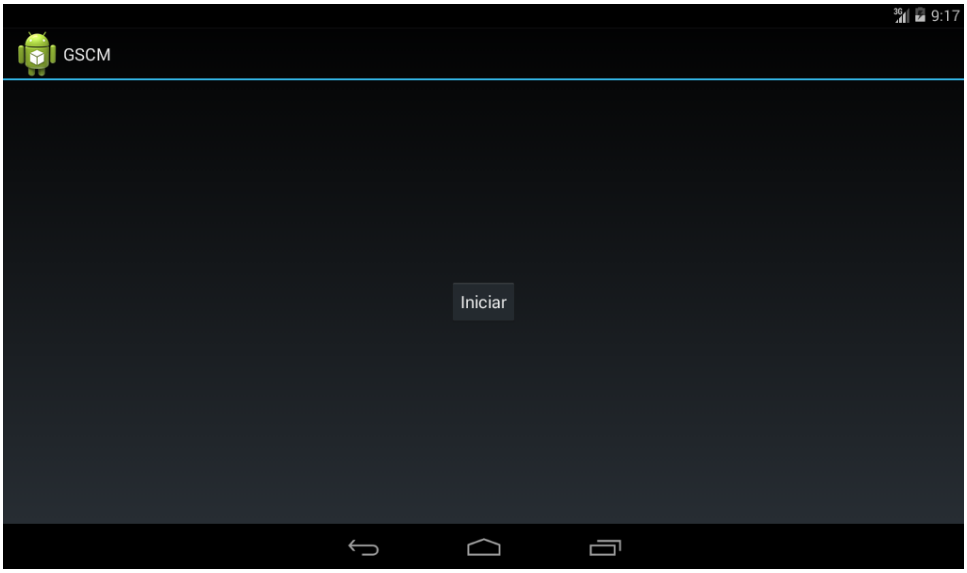

Figura 41 *– Módulo 1: Tela inicial da aplicação.*

Clicando no botão “Iniciar”, é possível acessar a tela de seleção do médico responsável pelo atendimento 42.

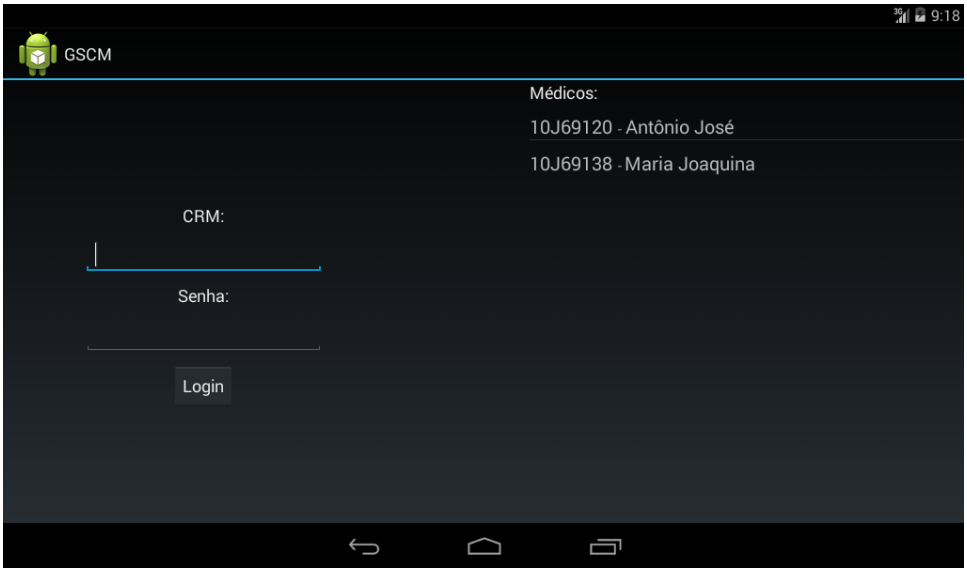

Figura 42 *– Módulo 1: Tela de seleção de médico.*

Para obter as próximas funcionalidades da aplicação é necessário que o médico esteja devidamente cadastrado no “Módulo 2”. No momento do cadastro é definida uma senha para que um médico não atenda um paciente utilizando outro cadastro, que não o dele. Pensando em

usabilidade afim de agilizar o procedimento e o médico não tenha que digitar seu CRM, esta *activity* se comunica com o "Módulo 2" e recupera a lista de todos os médicos cadastrados no sistema. Desta forma, basta que o médico toque em seu cadastro, na lista de médicos presente no lado direito da interface, e seu CRM é automaticamente preenchido no campo CRM da aplicação. Caso o médico preencha incorretamente o formulário de *login*, ele é apresentado a uma mensagem de erro, como apresentado na Figura 43.

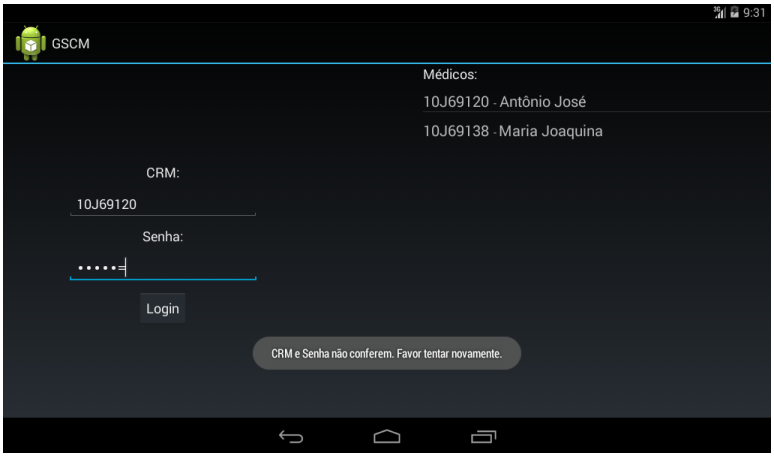

Figura 43 *– Módulo 1: Mensagem de erro de senha.*

Caso preeencha corretamente e seus dados de autenticação estejam corretos, o médico é apresentado à nova tela da aplicação, que possui a funcionalidade de buscar o paciente no banco de dados, facilitando ainda mais o procedimento de atendimento, apresentada na Figura 44.

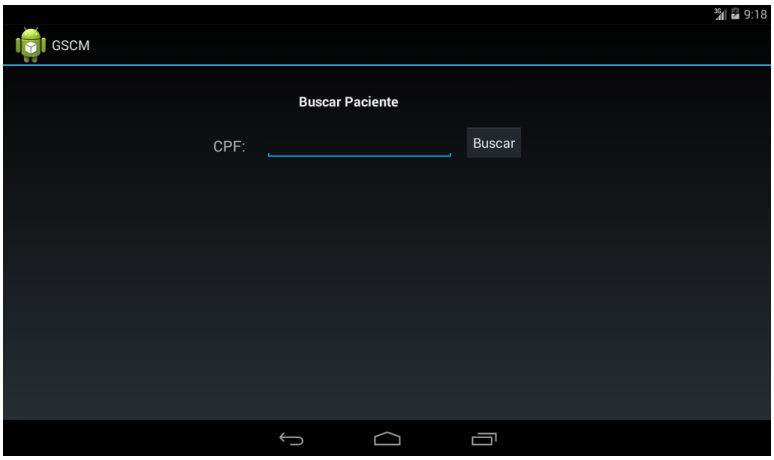

Figura 44 *– Módulo 1: Tela de busca do paciente.*

Na tela de busca, através do CPF do paciente o médico poderá acessar as informações do paciente que já foram previamente cadastradas no banco de dados através do "Módulo 2". É

notável a união dos dois módulos para prover uma melhor experiência de usabilidade e agilizar o atendimento. Ao digitar o CPF do paciente, o médico então pode verificar os dados do paciente rapidamente e confirmar, caso realmente seja o paciente que será atendido. A Figura 45 mostra um paciente já cadastrado no sistema.

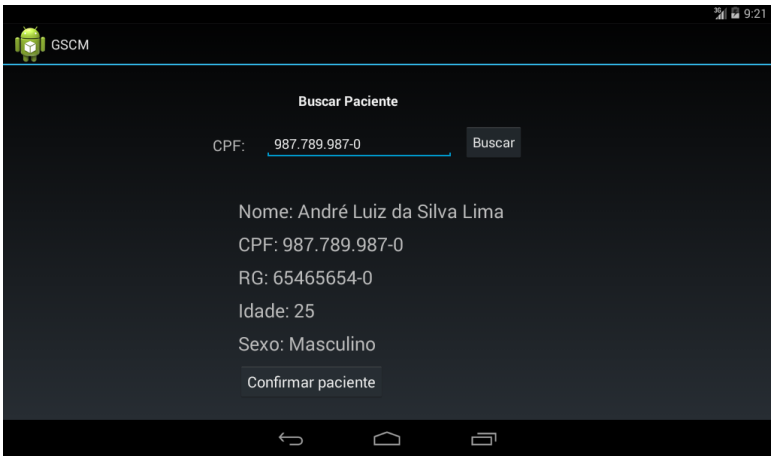

Figura 45 *– Módulo 1: Tela apresentando um paciente cadastrado no sistema.*

Após verificar os dados, basta tocar o botão "Confirmar paciente" para que a aplicação inicie uma *interface* bastante importante, a de perfil do paciente. Através desta tela de perfil é possível acessar uma grande quantidade de informações do paciente como histórico de consultas, exames, patologias, últimos sintomas, realizar nova consulta dentre outras. A seguir (Figura 46), é apresentada a tela de perfil do paciente.

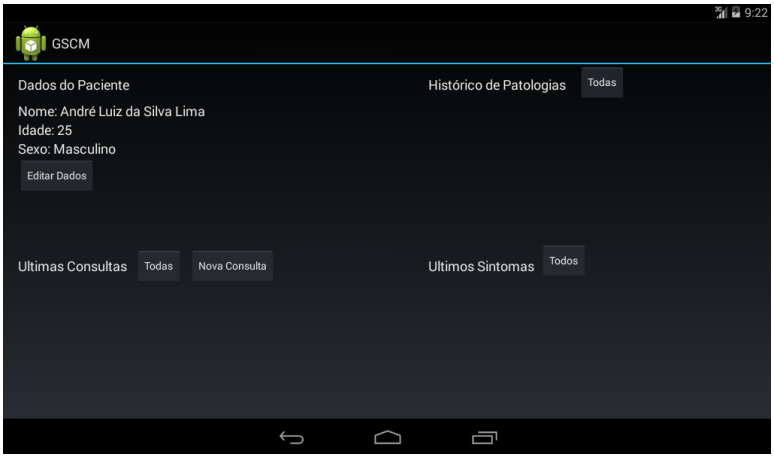

Figura 46 *– Módulo 1: Tela de perfil do paciente.*

Para dar continuidade ao fluxo de funcionamento do sistema, será então simulada uma consulta para que todas as funcionalidades da aplicação sejam demonstradas nesta seção. Co-

meçando pela funcionalidade de efetuar uma consulta, que pode ser acessada através do botão "Nova Consulta". Esta funcionalidade permite ao médico selecionar diversos sintomas relatados pelo paciente. Vale lembrar que a disponibilidade da variedade de sintomas é relacionada a quantidade de sintomas cadastradas no banco de dados através do "Módulo 2". A seguir (Figura 47) é apresentada a tela de consulta.

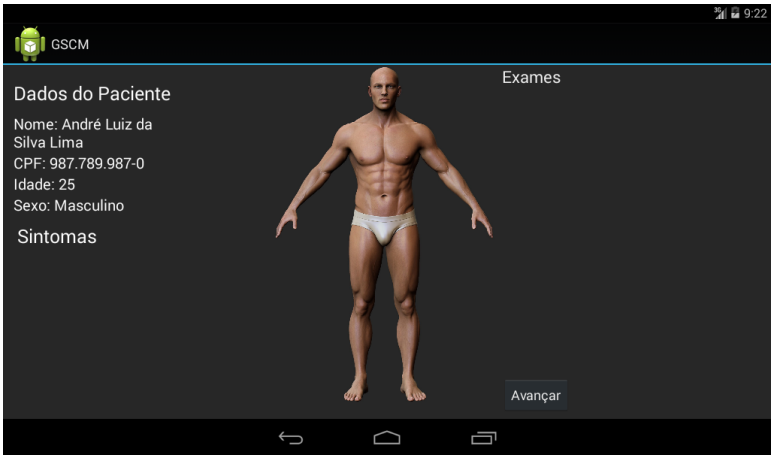

Figura 47 *– Módulo 1: Tela de consulta do paciente.*

Outro ponto importante da tela de consulta é que os sintomas são mostrados ao médico paralelamente à adição de novos sintomas. Vale lembrar que os sintomas são divididos por "área" do corpo humano, funcionando então da seguinte forma: ao tocar na cabeça da ilustração do corpo humano apresentada na *interface*, serão listados todos os sintomas relacionados a esta área do corpo, e assim sucessivamente com as outras áreas. A seguir (Figura 48 e Figura 49), pode-se observar a seleção de dois sintomas para o paciente.

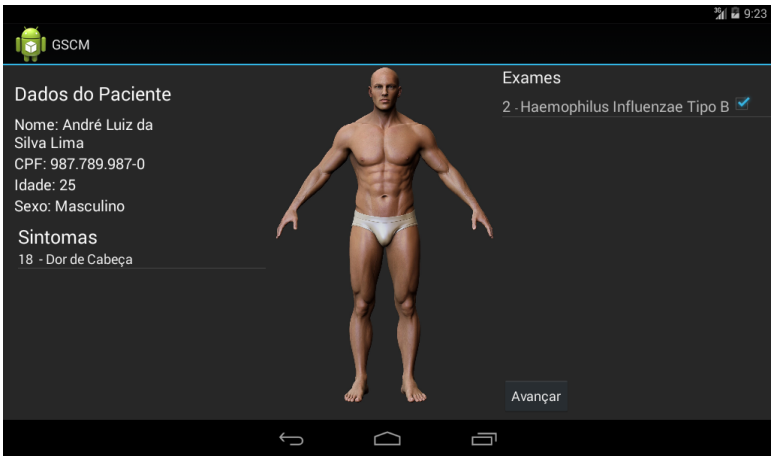

Figura 48 *– Módulo 1: Tela com um sintoma adicionado ao paciente.*

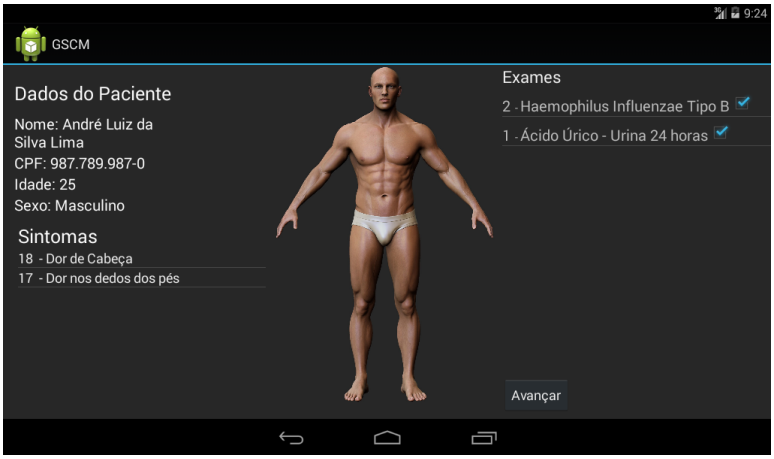

Figura 49 *– Módulo 1: Tela com dois sintomas adicionados ao paciente.*

Ao finalizar a adição de sintomas, através do botão “Avançar”, é possível acessar a próxima tela da aplicação que fornece mais opções de exames ao médico, além dos já definidos pela própria aplicação. Um recurso bastante interessante pertencente a esta nova tela, é o recurso de autocomplete aplicado ao campo de adição de novos exames. Este recurso foi implementando pensando mais uma vez em melhorar a usabilidade e agilizar o atendimento. Este recurso é apresentado na Figura 50. Onde o médico digita as três primeiras letras do exame a ser adicionado e então a aplicação apresenta ao médico os exames, cadastrados no banco de dados do “Módulo 2”, referentes a estas letras.

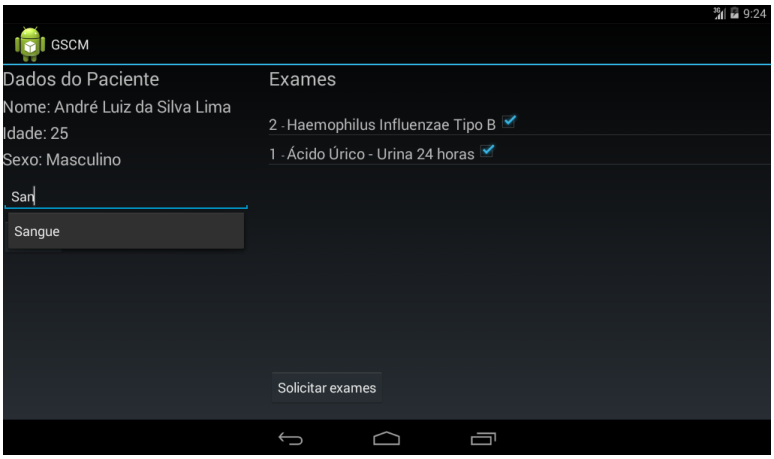

Figura 50 *– Módulo 1: Tela apresentando o autocomplete para adição de sintoma.*

Após selecionado o exame a ser adicionado, basta pressionar o botão “Adicionar”, e então o exame é adicionado a lista dos exames solicitados, como podemos observar através da

Figura 51.

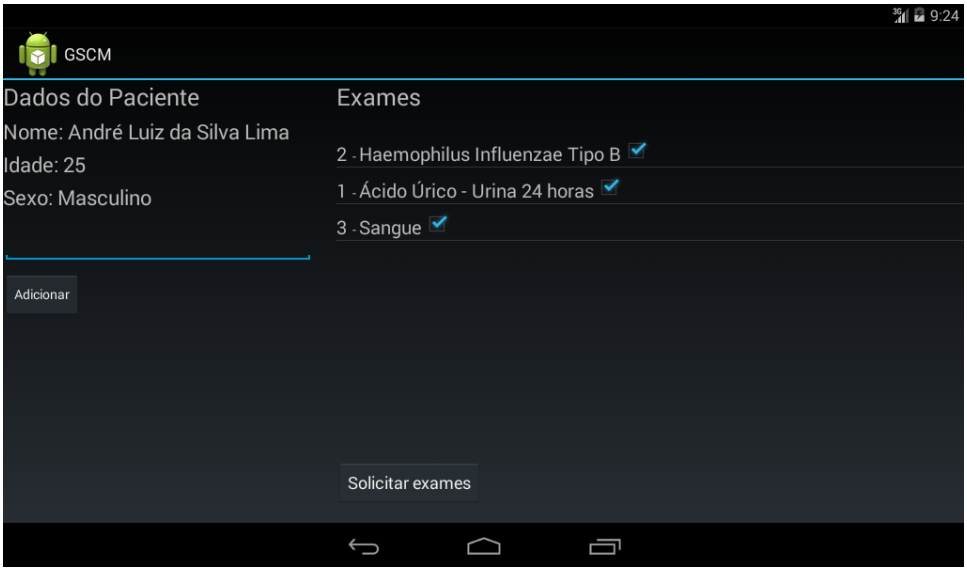

Figura 51 *– Módulo 1: Tela apresentando um sintoma adicional já adicionado à lista.*

Outro recurso bastante interessante é em relação a possibilidade de, mesmo que esteja apresentado na lista de exames a ser solicitados, desmarcar a opção do exame, ocasionando a sua não solicitação ao paciente. Ao clicar no botão “Solicitar exames”, é mostrada uma mensagem ao usuário, a consulta é enviada ao “Módulo 2” e devidamente salva no banco de dados, como apresentado na Figura 52.

Figura 52 *– Módulo 1: Mensagem confirmando a gravação da consulta no banco de dados.*

Clicando na opção “OK”, o usuário é então redirecionado a uma nova tela, onde ele

pode optar entre gerar um arquivo no formato ”pdf“ com os exames solicitados ou voltar para a tela de perfil do usuário, como apresentado na Figura 53.

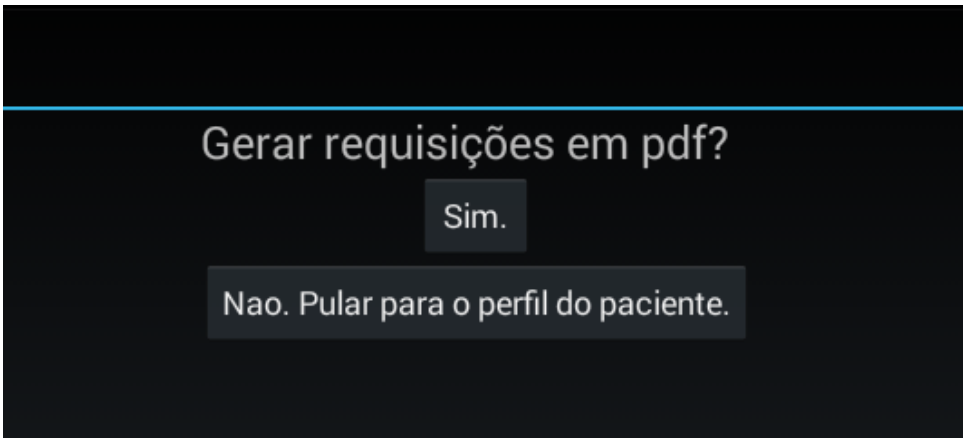

Figura 53 *– Módulo 1: Tela de geração de requisições de exames.*

Ao voltar para a tela de perfil do usuário (Figura 54, já é possível observar a consulta que acabara de ser realizada, bem como os últimos sintomas apresentados por aquele paciente.

Figura 54 *– Módulo 1: Tela de perfil apresentando a consulta realizado.*

É possível então visualizar os detalhes da consulta, como apresentado na Figura 55.

***Figura 55** – Módulo 1: Tela de visualização individual da consulta.*

Esta aplicação também foi pensada no retorno do paciente para apresentar os resultados dos exames solicitados. Através dela é possível salvar uma imagem dos resultados de cada exame, que pode ser tirada através da câmera do próprio *tablete*. Estas imagens também são gravadas no banco de dados e são passíveis de consultas posteriormente para auxiliar ainda mais o médico em possíveis tomadas de decisão.

Para acessar esta funcionalidade, basta acessar os detalhes da consulta, apresentados na imagem anterior, e clicar no itens da lista de exames. A partir disto o usuário tem acesso a seguinte tela (Figura 56).

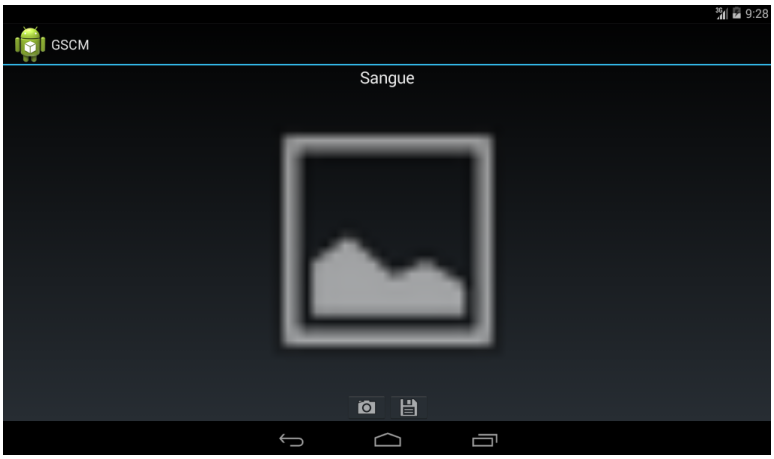

***Figura 56** – Módulo 1: Tela de visualização individual do exame.*

Ao clicar no botão com ícone de câmera, o usuário é então redirecionado ao aplicativo padrão da câmera do dispositivo, através da qual ele poderá tirar uma foto do resultado

do exame. Tirada a foto, o usuário é então encaminhado novamente para a tela apresentada anteriormente, porém, contendo a imagem do exame que será salva no banco de dados, como apresentado a seguir (Figura 57.

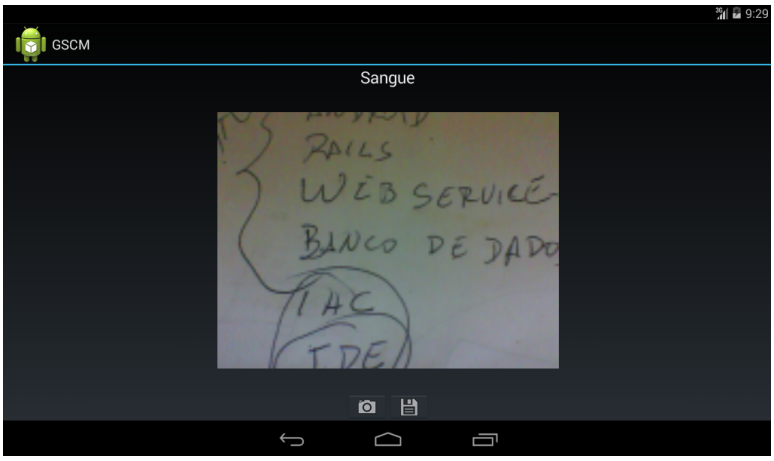

Figura 57 *– Módulo 1: Tela de apresentação da imagem do exame.*

A partir disto, basta clicar no botão salvar e pronto. A imagem é então enviada para o "Módulo 2" e salva no banco de dados. Assim, ela pode ser consultada, através da aplicação, em momentos posteriores. É importante ressaltar também que uma imagem é geralmente um tipo de dado grande e seu envio para o servidor pode demorar alguns segundos. Torna-se então necessário, de acordo com as regras de usabilidade, sempre manter o usuário informado a respeito do "*status*" do sistema. Logo, ao pressionar o botão de salvar a imagem, é apresentada um diálogo ao usuário informando que a requisição estão sendo processada, apresentado na Figura 58.

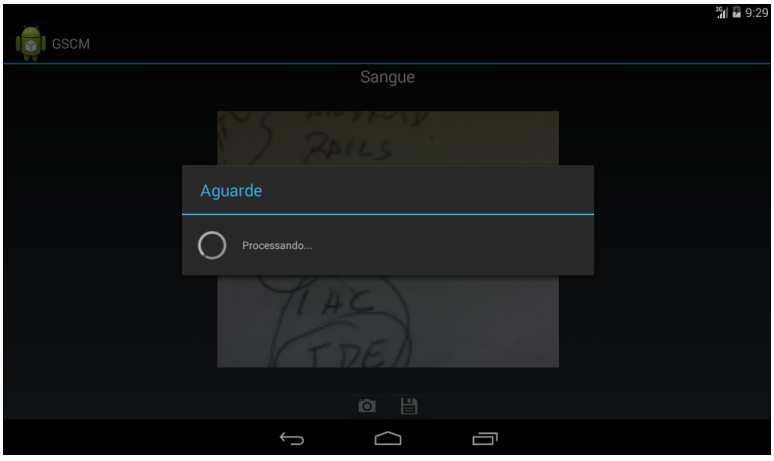

Figura 58 *– Módulo 1: Mensagem apresentando o status da aplicação .*

Finalizada a requisição, o usuário é então encaminhado novamente para a tela de detalhes da consulta, onde poderá voltar ao perfil do paciente ou adicionar imagens aos demais exames solicitados.

Módulo 2

O "Módulo 2" é um sistema *Web* desenvolvido utilizando o *framework* Grails. Ele que é responsável por enviar e receber todos os dados necessários para o funcionamento da aplicação. Através dele também é possível fazer todos os cadastros necessários como por exemplo de patologias, sintomas, exames, médicos e pacientes. Além disto, é através deste sistema que são disponibilizados os serviços, através da *web*, que dão vida a aplicação Android. É importante lembrar que todos os dados trocados entre a aplicação móvel e o sistema web são enviados e recebidos em formato JSON, através de *web servisse* do tipo *RESTful*. Tecnologias já abordadas anteriormente.

De modo geral, a aplicação *Web* funciona como um gerenciador de tudo que é necessário para o funcionamento da aplicação *web*. Desde a manipulação do banco de dados até a entrega destes dados à aplicação.

O sistema ainda não conta om um *layout* de página moderno e bem definido. Porém, ainda é possível observar, através da seguintes imagens, que é sim possível realizar todas as operações básicas e necessárias ao seu funcionamento. Como o *layout* do sistema ainda não está organizado de modo a seguir um fluxo concreto de funcionamento, serão apresentadas apenas as suas principais telas e detalhadas seu funcionamento.

A seguir (Figura 59) mostra a página de lista, contendo todos os pacientes cadastrados através do sistema. Todos estes pacientes podem ser acessados através da aplicação.

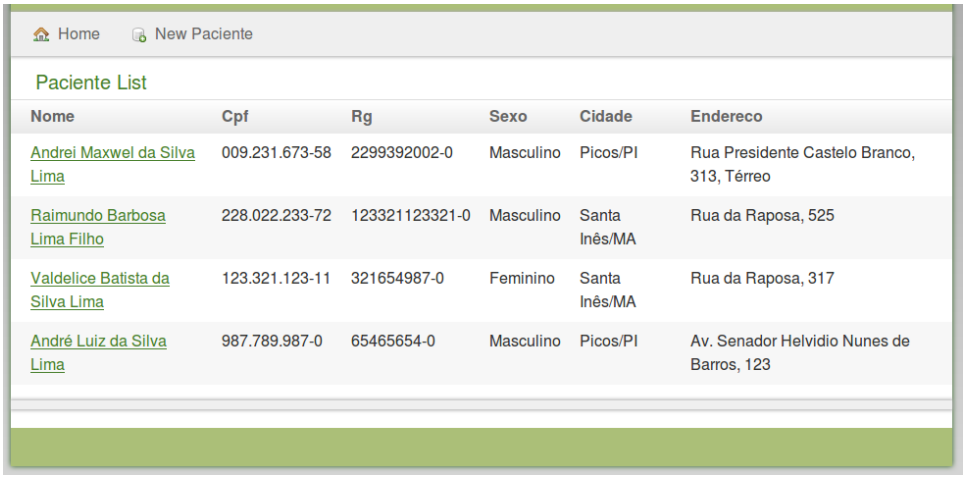

***Figura 59** – Módulo 2: Página de visualização de todos os pacientes.*

É possível efetuar todas as operações chamadas de CRUD, em relação a pacientes. Como por exemplo a de visualizar um determinado paciente (Figura 60). Contendo as opções de deletar ou editar este paciente.

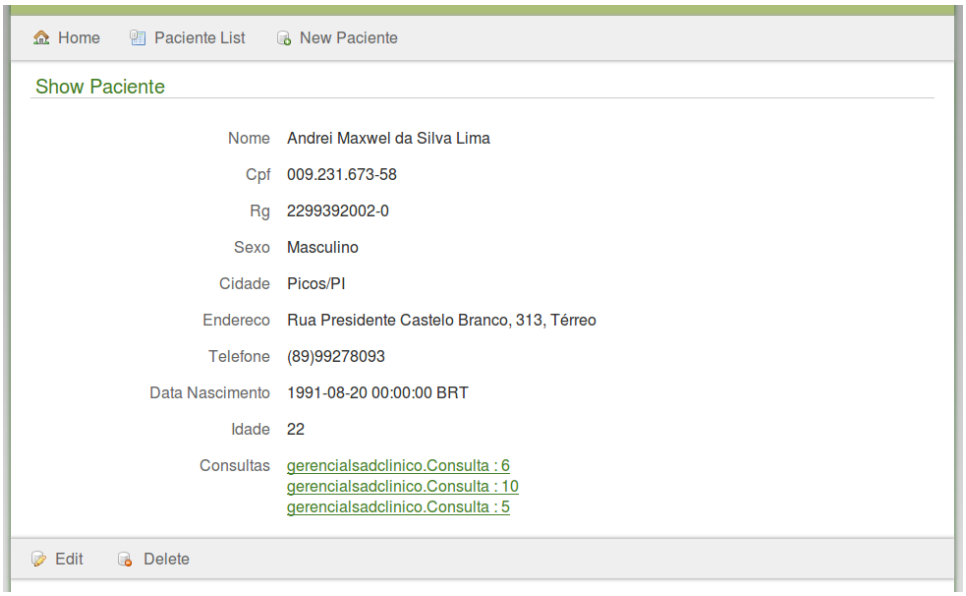

Figura 60 *– Módulo 2: Página de visualização de um paciente.*

E ainda a de criar um novo paciente, como apresentado na Figura 61, a seguir.

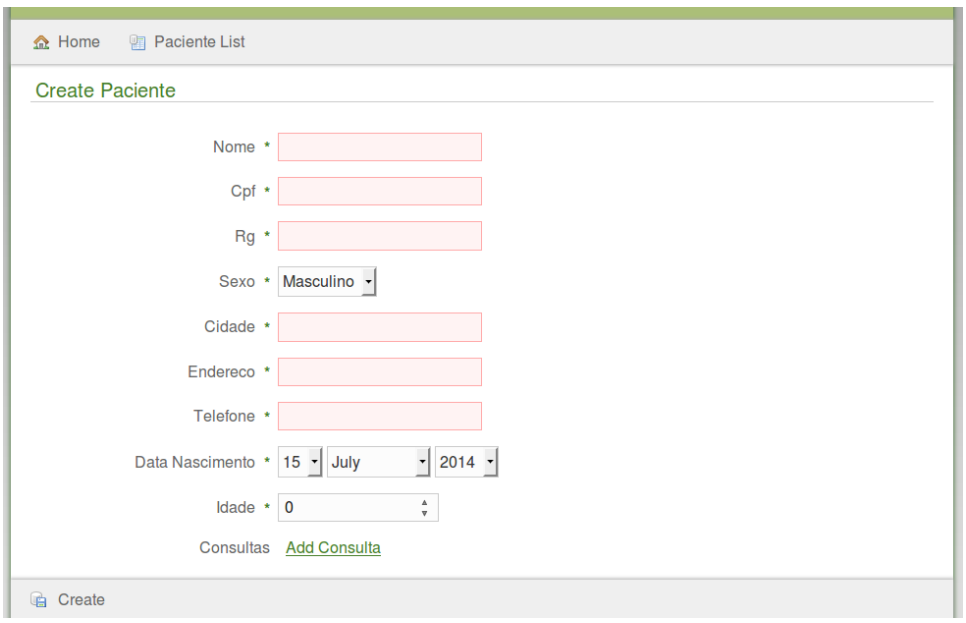

Figura 61 *– Módulo 2: Página de cadastrado de novo paciente.*

Todas estas funcionalidades também estão disponíveis para as outras entidades do sis-

tema, como sintomas, patologias, exames e etc. Na Figura 62 pode-se observar a página de criação de um novo sintoma.

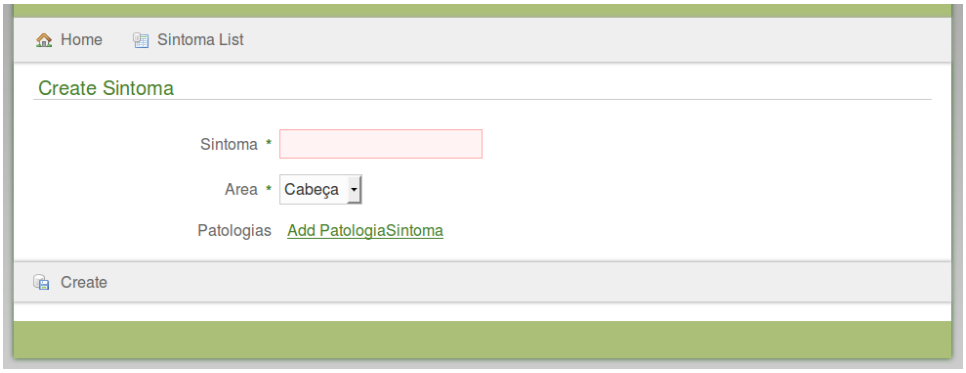

Figura 62 *– Módulo 2: Página de cadastrado de novo sintoma.*

A imagem apresentada na Figura 63, representa a página onde é possível ter acesso a todos os sintomas que já foram cadastrados no sistema.

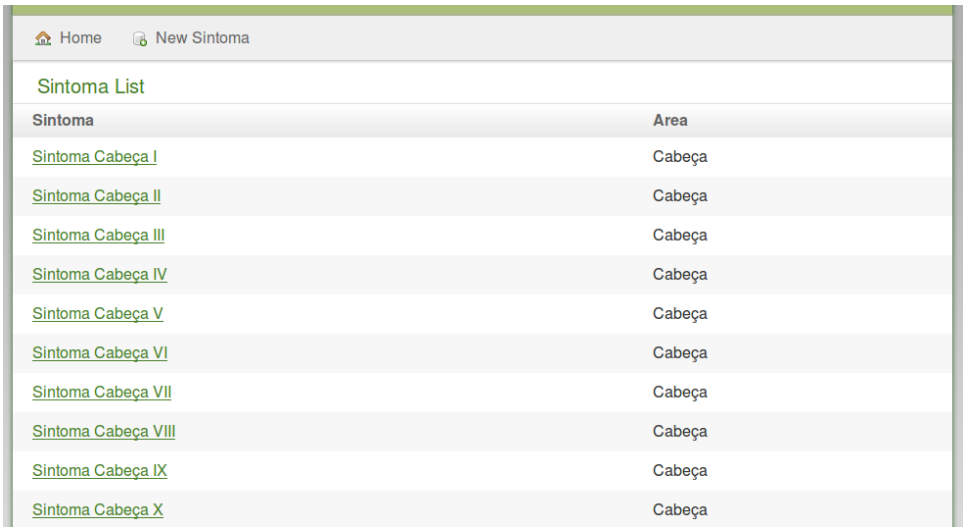

Figura 63 *– Módulo 2: Página de visualização de todos os sintomas.*

Para que o sistema funcione de forma correta, é necessário associar os sintomas aos exames. Isto pode ser realizado de forma bastante simples, através do recurso de autocompletar, que devolve automaticamente do banco de dados os sintomas que possuem as respectivas letras

digitadas neste campo. Isto pode ser observado na Figura 64.

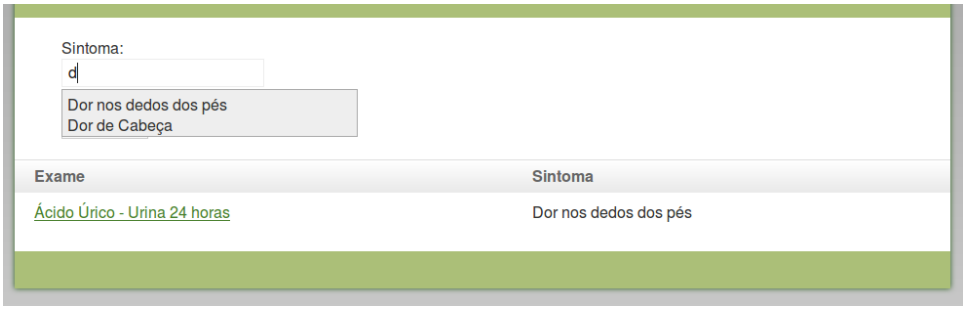

Figura 64 *– Módulo 2: Exemplo de utilização do autocomplete.*

O funcionamento do sistema *web* é bastante simples e segue os padrões de outros sistemas já disponíveis.

5 Resultados

Foram realizados diversos testes para demonstrar a viabilidade de utilização do *software* que foi desenvolvido neste trabalho. Dentre eles, por exemplo, foram aplicados testes de funcionalidades, de usabilidade e testes para verificar se o funcionamento do *software* pode ou não se tornar uma possível solução para o objetivo que foi proposto.

5.1 Testes

Todos os testes foram realizados por uma amostra de 8 pessoas com conhecimentos e níveis distintos. Os testes consistiram de:

- Testes de funcionalidades: todos os usuários operaram, desde o início, todo o procedimento de realização de uma consulta e simularam ainda o retorno do paciente para que os resultados dos exames fossem armazenados no seu histórico. Vale lembrar que estes procedimentos foram cronometrados afim de determinar quanto tempo cada usuário gastou nestas operações;

- Testes de usabilidade: após a utilização do *software* desenvolvido, os usuários responderam a um questionário *online*, que avalia a viabilidade da interface humano computador, o qual será melhor explicado mais a frente neste capítulo;

- Testes dos objetivos do trabalho: por fim, foi elaborado um questionário a respeito dos objetivos específicos deste projeto, para conhecer a opinião dos usuários que o testaram. Através deste questionário os usuários poderiam afirmar ou não se este trabalho pode ou não se tornar uma possível solução para o problema proposto.

Vale ressaltar ainda que este é apenas um protótipo de *software*, e que os resultados aqui apresentados podem ser melhorados. Além do mais, é necessário ainda que sejam realizados testes em ambientes reais, ou seja, nos postos de saúde, para que se possa obter resultados mais pontuais a respeito de seus reais benefícios.

5.1.1 Testes de Funcionalidades

Os testes de funcionalidades consistiram em simular os procedimentos de consulta e retorno do paciente. Todo o processo para a realização desses dois procedimentos já foram

abordados neste trabalho.

A primeira informação importante obtida através deste teste foi que, das 8 pessoas selecionadas, todas conseguiram realizar os dois procedimentos com sucesso e com o mínimo de dificuldade. Isto é muito relevante, já que, apesar de ser um protótipo, estas duas principais funcionalidades já estão bem definidas e possuem um fluxo de funcionamento correto.

Outro ponto relevante durante a realização destes testes foi o tempo para realização dos dois procedimentos. Vale lembrar que os dois procedimentos foram realizados separadamente, desde o início, ou seja, a consulta era efetivada, a aplicação era encerrada e iniciada novamente, para então realizar a simulação do retorno do paciente. Para melhor visualização destes dados, será utilizado o gráfico apresentados na Figura 65. Este gráfico representa o tempo gasto durante a realização do procedimento de consulta para cada usuário.

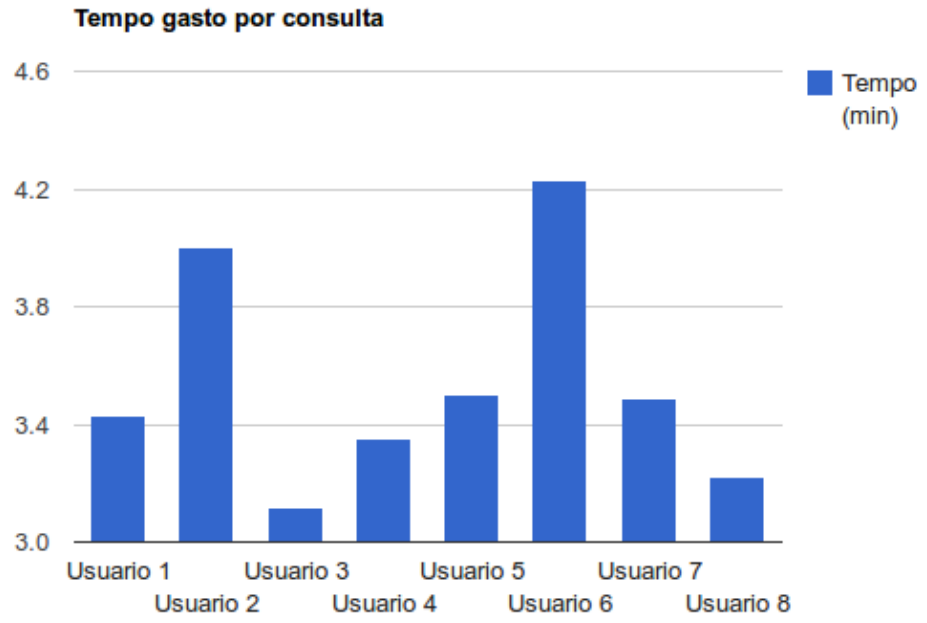

Figura 65 *– Gráfico: tempo gasto por consulta.*

Podemos observar pelo gráfico (Figura 65) que o usuário que menos tempo levou para realizar a consulta foi o "Usuário 3", que realizou este procedimento em 3.12 minutos. Em contra partida o "Usuário 6" levou 4.23 minutos para realizar o mesmo procedimento. A partir da média de tempo destas utilizações, chega-se ao número de 3.54 minutos para cada consulta, que é um ótimo resultado, se comparado por exemplo, com uma consulta realizada em um posto de saúde que pode levar de 6 a 15 minutos.

O segundo procedimento realizado, como já citado anteriormente, foi o de retorno do paciente. Este teste proporcionou resultados bastante significativos, visto que o nível de burocracia para retorno de um paciente nas unidades de saúde pode complicar o processo de retorno do paciente. Primeiramente, a partir da Figura 66, podemos observar o gráfico com os resulta-

dos do tempos gasto neste procedimento para cada usuário.

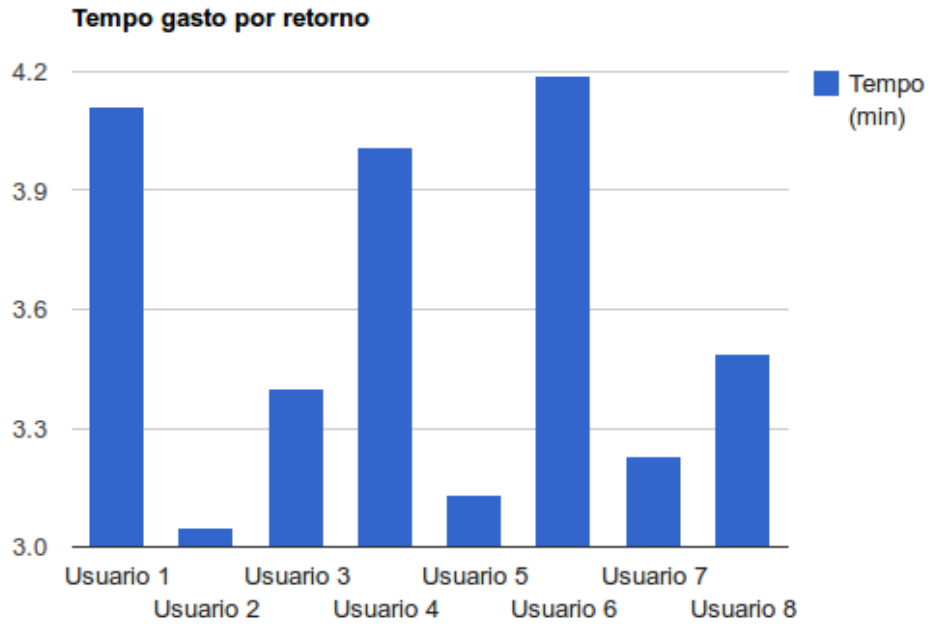

Figura 66 *– Gráfico: tempo gasto por retorno.*

A partir do gráfico apresentado na Figura 66, que mostra os resultados de tempo para a realização do procedimento de retorno do paciente, é possível observar que o "Usuário 2" foi o que realizou este procedimento em menor tempo (3.05 minutos). Enquanto o "Usuário 6" novamente apresentou o maior tempo durante o procedimento, 4.19 minutos. Novamente a partir da média aritmética, podemos analisar que o tempo médio de realização do procedimento de retorno do paciente é de 3.57 minutos. Este teste de retorno é importante para mostrar que esta aplicação pode ajudar e muito a desburocratizar este procedimento. Em postos de saúde, por exemplo, só o tempo necessário para buscar a ficha de cadastro do paciente já é muito superior a todo o procedimento realizado através da aplicação.

É importante lembrar que estes testes são apenas simulações de uso. Os resultados podem variar para mais ou para menos, dependendo das circunstâncias em que a aplicação estiver sendo utilizada.

5.1.2 Testes de Usabilidade

Os testes de usabilidade são importantes para verificar se a aplicação é fácil de utilizar, se obedece as regras para uma boa interface, se o usuário consegue encontrar facilmente as principais funcionalidades, consegue utilizar a aplicação corretamente, etc. Como ferramenta de avaliação, foi utilizado o ErgoList[1], que é um questionário online, através do qual podem ser

[1] Disponível em: http://www.labiutil.inf.ufsc.br/ergolist/index.html

realizados testes de IHC.

O questionário ErgoList é um serviço gratuito disponibilizado via *internet* composto de uma base de conhecimento em ergonomia, que inspeciona, através de um *checklist*, interfaces homem-computador (ABREU, 2010). Segundo Ergolist (2011), as listas de verificação que formam o ErgoList destinam-se a apoiar exercícios de inspeção da interface de maneira a levar o estudante a descobrir as falhas ergonômicas mais flagrantes em uma interface com o usuário. Porém, completa ainda que sua abrangência não foi testada, o que confere ao ErgoList uma natureza essencialmente didática.

O ErgoList nada mais é que um questionário em forma de "CheckList". Segundo Abreu (2010), este questionário leva em consideração os dezoito critérios ergonômicos de Bastien e Scapin (1993). Divididos nestes *checklists* estão contidas 194 questões. Para o teste de usabilidade deste trabalho, foram utilizados 7 desses critérios, totalizando 93 questões. Estas questões estão divididas da seguinte forma:

- Presteza: verifica se o sistema informa e conduz o usuário durante a interação. Composto por 17 questões;
- Legibilidade: Verifica a legibilidade das informações apresentadas nas telas do sistema. Composto por 27 questões;
- Significados: Avalia se os códigos e denominações são claros e significativos para os usuários do sistema. Composto por 12 questões;
- Agrupamento por localização: Verifica a distribuição espacial dos itens traduz as relações entre as informações. Composto por 11 questões;
- Densidade Informacional: Avalia a densidade informacional das telas apresentadas pelo sistema. Composto por 9 questões;
- *Feedback*: Avalia a qualidade do *feedback* imediato às ações do usuário; Composto por 12 questões.
- Ações Mínimas: Verifica a extensão dos diálogos estabelecidos para a realização dos objetivos do usuário. Composto por 5 questões.

Para cada *checklist*, o ErgoList possibilita as respostas: Sim, Não, Não Aplicável e Adiar Resposta.

- Sim: a aplicação está de acordo com a questão;
- Não: a aplicação não está de acordo com a questão;

- Não aplicável: o proposto na questão não se aplica a esta aplicação;
- Adiar Resposta: o usuário deseja responder a esta questão em outro momento.

Os resultados do ErgoList foram satisfatórios, visto que o projeto ainda se trata de um protótipo. Recapitulando que, imediatamente após os testes de funcionalidades, os 8 usuários foram submetidos ao questionário ErgoList para avaliar a IHC na aplicação. A Tabela 5, a seguir, apresenta as médias obtidas de acordo com cada critério avaliado, segundo a avaliação "Sim", ou seja, a aplicação está em conforme com o critério ergonômico.

	Legibilidade	**Significados**	**Presteza**	**Agrupamento**	**Densidade**	***Feedback***	**Ações Mínimas**
Média	64,51%	51,04%	66,91%	63,63%	62,5%	66,66%	52,5%

Tabela 5 *– Médias "Conforme" do teste por critério.*

Através da Tabela 5, podemos observar que o critério que obteve maior média "em conforme" foi "Presteza", que, segundo Ergolist (2011), ajuda o usuário de uma forma bastante direta, indicando as informações sobre estados das operações e rotulando corretamente os campos. Ja a menor média foi obtida no critério "Significados", isto, principalmente devido aos titulos das telas da aplicações não se distinguirem entre as telas, afetando então a usabilidade. Para melhor visualizar esta diferença, podemos observar estes mesmos resultados em forma de gráfico na Figura 67.

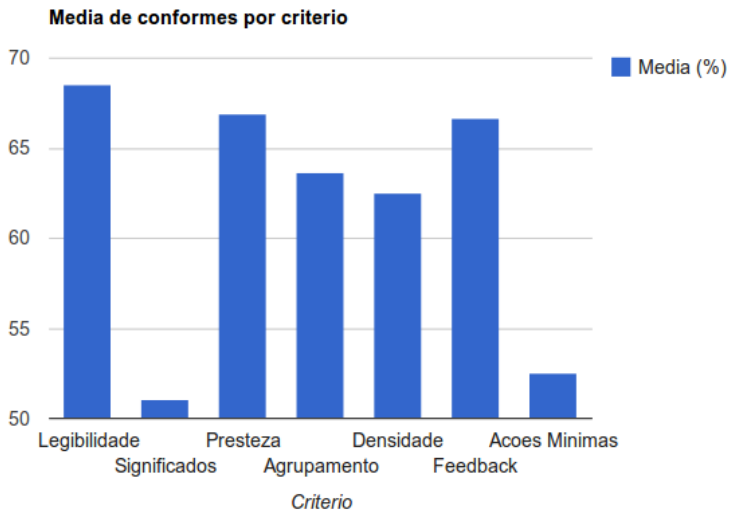

Figura 67 *– Gráfico: médias "Conforme" por critério.*

Podemos observar a média de “Não conforme”, ou seja, quando o questionamento não está de acordo com a aplicação, na tabela 6.

	Legibilidade	**Significados**	**Presteza**	**Agrupamento**	**Densidade**	***Feedback***	**Ações Mínimas**
Média	13,42%	25%	18,38%	13,63%	19,44%	17,7%	17,5%

Tabela 6 *– Médias ”Não Conforme“ do teste por critério.*

Como pode ser visto na Tabela 6, o critério que recebeu o menor indices de “Não conforme”, foi “Legibilidade”. Lembrando que este critério, caso aplicado, auxilia muito na perfomance de quem está utilizando a aplicação. Ja a maior média de “Não conforme” é visto no critério “Significados”, implicando que a aplicação não beneficia o recordamento e reconhecimento de algumas funcionalidades e não utiliza códigos e ícones significativos para o que o usuário possa reconhecer determinadas funções de acordo com suas capacidades cognitivas. Também para proporcionar melhor análisa, estes resultados também são apresentados em forma de gráfico na Figura 68.

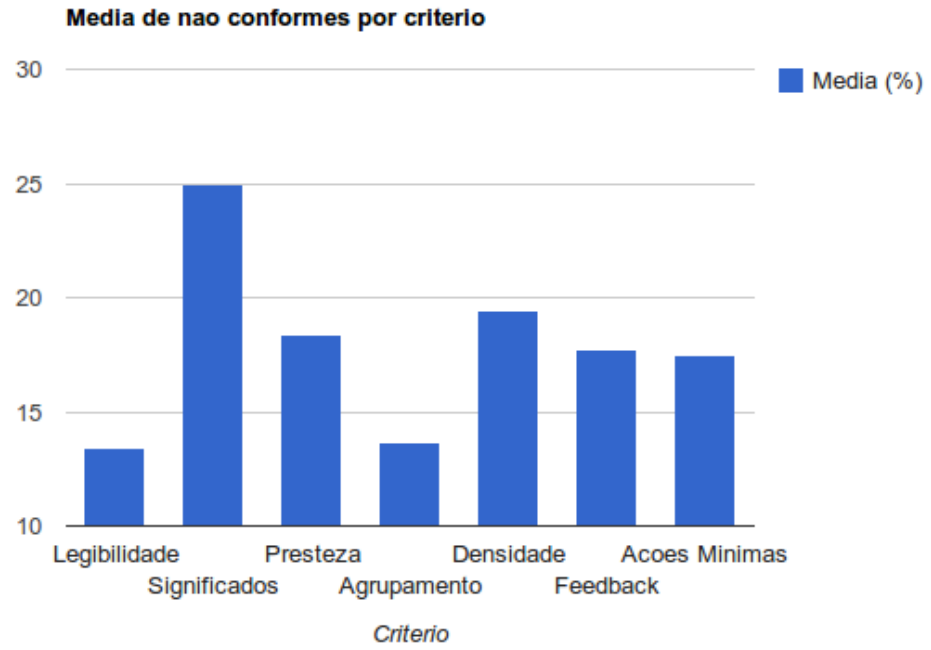

Figura 68 *– Gráfico: médias ”Não Conforme“ por critério.*

Por fim, podemos observar no gráfico apresentado na Figura 69, um panorama geral

sobre o teste do ErgoList.

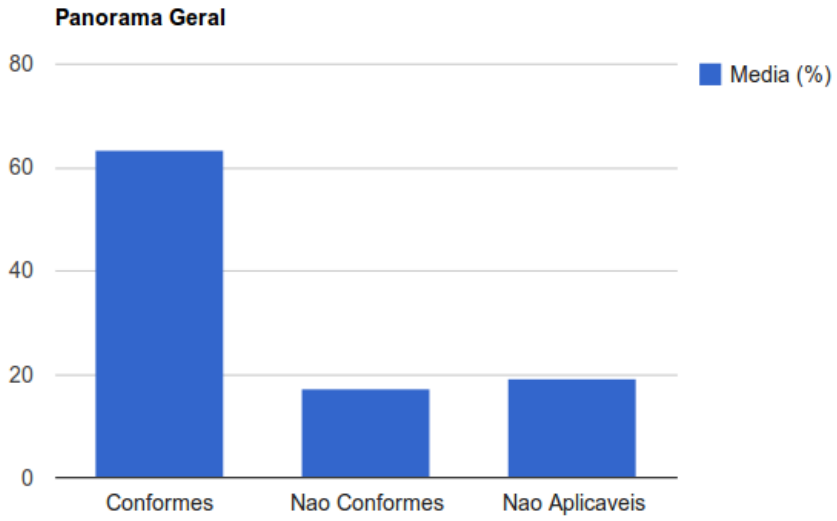

Figura 69 *– Gráfico: panorama geral.*

Através do gráfico apresentado na Figura 69, é possível observar a enorme diferença entre os itens que estão "Conforme", de acordo com o questionário ErgoList, e os itens "Não conformes". 63,7% dos itens estão conforme os critérios aplicados, enquanto apenas 17,2% estão em desacordo com os critérios.

5.1.3 Testes de Validação de Proposta

Este teste foi realizado com os 8 participantes com o objetivo de conhecer se o *software* desenvolvido pode ou não, de acordo com cada um dos usuários, se tornar uma possível solução para agilizar o atendimento nos postos de saúde e ainda manter de forma consistente os dados dos pacientes. Ele constitui de um pequeno questionário contendo apenas 4 questões, em que cada usuário só poderia responder "Sim", "Não", "Em partes" ou designar uma nota que varia de 0 a 10. "Em partes" significa que o *software* deixa a desejar em relação a algo na determinada questão.

O primeiro questionamento foi: "o *software* desenvolvido cumpre os objetivos para o qual foi desenvolvido?". Como podemos observar através do gráfico apresentado na Figura 70, dentre os 8 entrevistados, 6 responderam que "Sim", 1 respondeu que "Não" e " respondeu "Em

partes". Vale lembrar que não foi requisitado justificativa para nenhum dos questionamentos.

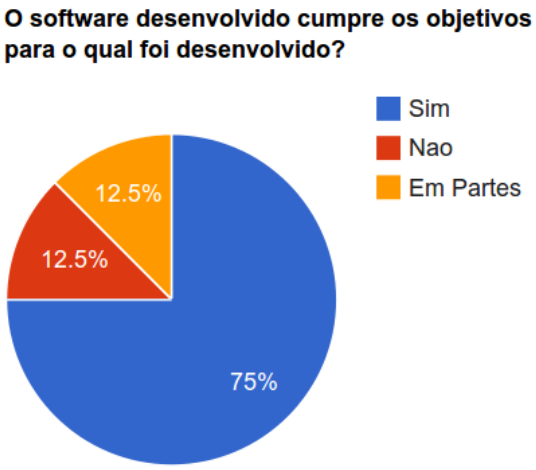

Figura 70 *– Gráfico: respostas para primeira questão.*

A segunda questionava o seguinte: "durante a implantação e treinamento para utilização do sistema, pode ocorrer de este, de alguma forma, atrapalhar os procedimentos padrões do local?". De acordo com o gráfico apresentado na Figura 71, 1 usuário afirmou que sim, a etapa de implantação e treinamento pode atrapalhar os procedimentos padrões do local, 5 usuários afirmaram que não, ou seja, a etapa de implantação não atrapalhará de forma alguma o andamento normal do posto de saúde, já 2 pessoas afirmaram que, em partes, a implantação do projeto pode atrapalhar os procedimentos padrões do posto de saúde.

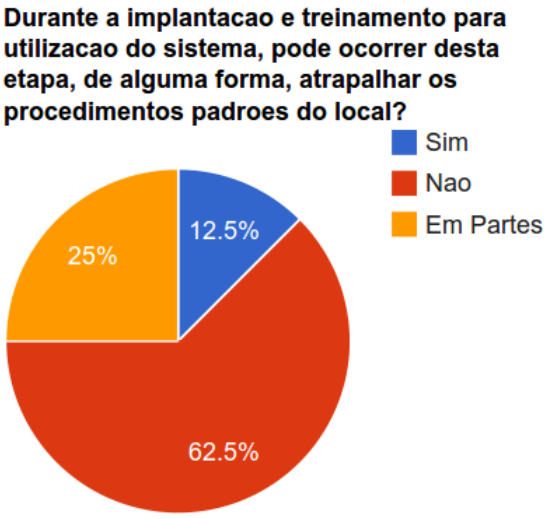

Figura 71 *– Gráfico: respostas para segunda questão.*

A terceira questão: "no geral, qual nota você daria ao *software* desenvolvido de acordo com os objetivos propostos? (0 a 10)". De acordo com essa questão, 5 usuários deram nota 9,

1 usuário deu nota 8,5, 1 usuário designou nota 8 e o último aplicou nota 7 ao questionamento, como apresentado no gráfico da Figura 72.

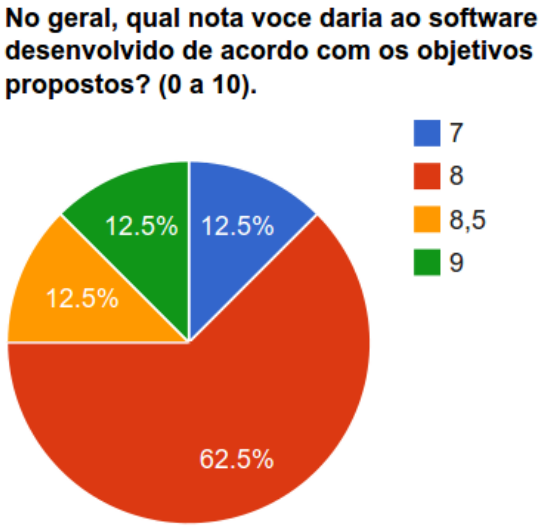

Figura 72 *– Gráfico: respostas para terceira questão.*

A quarta e última questão: “na sua opnião, em quantos % o software necessita melhorar para tornar totalmente viável sua utilização em ambiente real?”. De acordo com a pesquisa, 3 usuários afirmaram que o software necessita de 15% de melhora, 1 usuário afirmou que o software necessita de 20% de melhora, 3 afirmaram que o software necessita melhorar 30% e 1 afirmou que necessita melhorar 50%, como mostra a Figura 73.

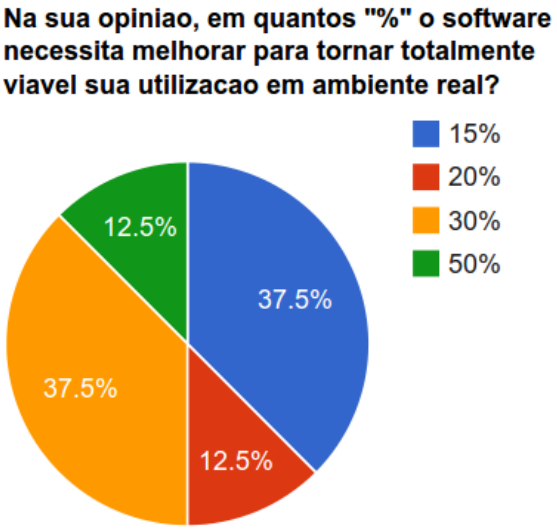

Figura 73 *– Gráfico: respostas para quarta questão.*

6 Conclusões e Trabalhos Futuros

O sistema de saúde público brasileiro necessita de uma rápida reformulação e renovação. Existe muita burocracia em determinados procedimentos dos postos de saúde que poderiam ser contornadas através do uso de tecnologias de gestão como por exemplo sistemas de informações clínicos ou outros. A partir desta problemática surgiu a idéia de criar um protótipo de sistema de informação clínico que servisse como uma possível solução para agilizar os atendimentos médicos nos postos de saúde e ainda armazenar todo o histórico dos pacientes de forma consistente, possibilitando então o reuso destas informações com outras finalidades benéficas.

Primeiramente foi necessário definir quais tecnologias se adequariam mais ao projeto que seria desenvolvido. O primeiro pilar para definir as tecnologias foi o tempo, pois como o tempo para desenvolvimento era curto, seria necessário então utilizar tecnologia de desenvolvimento ágil. A partir disto veio então a idéia de utilizar o *framework* Grails. Que é um *framework* de desenvolvimento ágil. Posteriormente surgiu a necessidade de que o médico não deveria ficar preso à um computador ou *notebook*. A partir disto veio a idéia de utilizar tecnologia móvel, no caso, o Android. É lógico que, para que estas duas tecnologias se comuniquem, tornou-se necessário que houvesse outra tecnologia que realizasse essa comunicação. E então surgiu a idéia de utilizar *Web Services* para esta funcionalidade. E foi diante deste contexto que o projeto foi desenvolvido.

Após desenvolvido deu-se início as fases dos testes de funcionalidades, usabilidade e de objetivos. Os testes de funcionalidades serviram para buscar falhas no sistema quanto ao fluxo de funcionamento em suas duas principais funcionalidades, que são realizar consulta e cadastrar as imagens dos exames nos retornos. Após testadas e validadas estas funcionalidades por terceiros, deu-se início então aos testes de usabilidade. Os testes de usabilidade possuiam o objetivo de verificar se a interface da aplicação obedecia aos principais critérios de usabilidade. Logo após os testes de usabilidade, foi aplicado então o teste de objetivo, que buscou conhecer a opinião de quem utilizou o software a respeito dos objetivos que foram propostos neste trabalho, que era desenvolver um protótipo de sistema de informação que agilizasse o atendimento médico e armazenasse os dados de forma consistente. Vale lembrar que todos os testes realizados obtiveram resultados positivos. É notável que o sistema ainda necessita de melhorias, mas, pelo fato de ser apenas um protótipo ainda existe muito trabalho a ser realizado para que ele possa estar totalmente validado para utilização em ambiente real.

Vale aqui ressaltar as principais dificuldades encontradas no decorrer do projeto. A principal delas foi aprender a utilizar novas tecnologias, como o Android e *Web Services*. Outra

dificuldade encontrada foi a de desenvolver a funcionalidade de capturar a imagem dos exames e enviá-los ao servidor devido ao fato de não encontrar materiais específicos que exemplificassem a utilização de *web services* para comunicar o Android com o *framework* Grails. Além destas, outra pedra encontrada no caminho foi a geração do PDF (*Portable Document Format*) contendo a requisição dos exames solicitados. Esta funcionalidade já está implementada, ou seja, a aplicação já esta gerando um arquivo no formato PDF com todos os exames solicitados, porém, não foi possível realizar testes com os usuários finais.

6.1 Trabalhos Futuros

Existe ainda muito trabalho a ser feito, como:

- Melhorar ainda mais a usabilidade da aplicação móvel;
- Melhorar a aparência do *layout* da aplicação móvel;
- Gerar o PDF de requisição de exames e enviá-lo diretamente para impressão a partir da aplicação móvel;
- Desenvolver um *layout* apropriado para o sistema *web*;
- Desenvolver geração de relatórios funcionais para o sistema *web*;
- Desenvolver rotinas para armazenar e imprimir receitas de prescrição de medicamentos;
- Desenvolver módulo para controle da Farmácia Básica;
- Adicionar imagem do corpo humano feminino ao realizar a consulta.

Referências

ABBAS, K. *Gestão de Custos em Organizações Hospitalares*. In: Florianópolis, 2001.

ABREU, Ana Célia Bastos de. *Avaliação de Usabilidade em Software Educativos*. In: Ceara, 2010.

ALVES, Rafael Ferreira; VANALLE, Rosângela Maria. *Ciclo de Vida de Desenvolvimento de Sistemas - Visão conceitual dos modelos clássico, espiral e prototipação*. In: São Paulo, 2014. Disponível em: <www.abepro.org.br>. Acesso em: 9 jun. 2014.

AMORIM, F. F. *Avaliação de Tecnologias em Saúde: Contexto Histórico e Perspectivas*. *Ciências Saúde V.21 (N. 4)*, 2010.

AQUINO, Juliana França Santos. *Plataformas de Desenvolvimento para Dispositivos Móveis*. In: Rio de Janeiro, Dezembro 2007.

AYRES, J. R. C. M. *Cuidado: tecnologia ou sabedoria prática*. *Interface-Comunicação, Saúde, Educação V.4 (N. 5)*, São Paulo, 2000.

BASSI, Dairton. *Desenvolvendo Software Livre com Programação eXtrema*. In: São Paulo, Abril 2006.

BECK, Kent. *Extreme Programming Explained: Embrace Change*. [S.l.]: Addison-Wesley, 2000.

BENITO, Gladys Amélia Véles; LICHESKI, Ana Paula. *Sistemas de Informação apoiando a gestão do trabalho em saúde*. *Rev Bras Enferm*, Santa Catarina, 2009.

BISSI, Wilson. *Scrum - Metodologia de Desenvolvimento Ágil*. In: Paraná, 2007.

BORDIN, M. V. *Introdução a Arquitetura Android*. 2012. Disponível em: <sites.setrem.com.br/>. Acesso em: jul. 2013.

BORGES, Roberto Cabral de Mello. *Avaliação Heurística, segundo Nielsen, Jakob e Molich, Rolf*. 2014. Disponível em: <www.inf.ufrgs.br/ cabral/13.Heuristicas.Nielsen2007.ppt>. Acesso em: 21 jun. 2014.

BURBECK, Steven. *Application Programmings in Smaltalk's 80 TM: How to use MVC*. 2009. Disponível em: <http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html>. Acesso em: 14 mar. 2010.

CAETANO, Daniel. *Programação Servidor em Sistemas Web*: Padrões mvc e dao. 2012.

CAMPOS, Augusto C. *Introdução ao Linux*. 2003. Disponível em: <www.br-linux.com>. Acesso em: 02 jun. 2014.

CARVALHO, André de Oliveira; EDUARDO, Maria Bernadete de Paula. *Saúde E Cidadania: Sistemas de Informação em Saúde para Municípios*. São Paulo, 1998.

CARVALHO, José Oscar Fontanini de. *O papel da interação humano-computador na inclusão digital*. *Transinformação, 15 (Edição Especial)*, In: Campinas, 2003.

CARVALHO, José Oscar Fontanini de. *Interação Humano-Computador (IHC) e Usabilidade*. 2014. Disponível em: <www2.dbd.puc-rio.br/>. Acesso em: 15 jun. 2014.

CARVALHO, Robson. *Análise e Levantamento de Requisitos*. 2014. Disponível em: <www.eteavare.com.br/arquivos/>. Acesso em: 12 jun. 2014.

CECOMP (Ed.). Modelos de processo de software. 2012.

CHAMBERS, Jason. *SiteMesh – AOP for web pages*. Janeiro 2005.

CHANG, Patrick; VITTIE, Lori Mac. *The Fundamentals of HTTP*. F5 Networks, In: Seattle, 2008.

COELHO, O. C.; JORGE, M. S. B. *Tecnologia das relações como dispositivo do atendimento humanizado na atenção básica à saúde na perspectiva do acesso, do acolhimento e do vínculo*. *Ciência e Saúde Coletiva*, 2009.

COHEN, D.; LINDVALL, M.; COSTA, P. e. *An introduction to agile methods. In Advances in Computers (pp. 1-66)*. Elsevier Sciente, In: New York, 2004.

CONASS. *Conselho Nacional de Secretários de Saúde: Para entender a gestão do SUS*. Brasília, 2003.

CORREA, A. K.; CASATE, J. C. *Humanização do atendimento em saúde: conhecimento veiculado na literatura brasileira de enfermagem*. *Rev Latino-am Enfermagem V. 13 (N. 1)*, Ribeirão Preto, 2005.

CROCKFORD, D. *Json specification*. Universidade do Minho, 2006. Disponível em: <http://www.ietf.org/rfc/rfc4627.txt>. Acesso em: 2006.

CYBIS, Walter; BETIOL, Adriana Holtz; FAUST, Richard. *Ergonomia e Usabilidade: Conhecimentos, Métodos e Aplicações*. 2. ed. São Paulo: Novatec, 2010.

DA, Filipe Fontinele de Almei. *Avaliação do Framework GRAILS como Plataforma no Desenvolvimento Ágil de Softwares*. In: Picos, Dezembro 2010.

DATASUS. *Com a Informatica, DATASUS é ágil e preciso na tomada de decisão sobre o sistema de saúde do Brasil*. Informática Corporation, São Paulo, Novembro 2013.

DAVIS, Scott; RUDOLPH, Jason. *Getting Started with Grails*. 2. ed. Estados Unidos: C4Media Inc, 2010.

DEACON, John. *Model-View-Controller Architecture*. 2009. Disponível em: <http://www.jdl.co.uk/briefings/index.html/#mvc>. Acesso em: 13 mar. 2010.

DEVELOPERS, Android. *Introdução a Arquitetura Android*. 2012. Disponível em: <http://developer.android.com /guide/ index.html>. Acesso em: jun. 2013.

DGE, Departamento de Governo Eletrônico. *Padrões Web em Governo Eletrônico – Cartilha de Usabilidade*. 1. ed. Brasília: e-PWG, 2010.

DOEDERLEIN, Osvaldo P. *Aprendendo Groovy*. *Java Magazine*, In: São Paulo, Janeiro 2006.

D'ÁVILA, Márcio. *Segurança de Rede*. 2001.

ERGOLIST. *Seja bem-vindo(a) ao ErgoList*. 2011. Disponível em: <http://www.labiutil.inf.ufsc.br/ergolist/index.html>. Acesso em: 13 jul. 2014.

ESPANHA, Rita. *Adenda à Análise Especializada: Tecnologias de Informação e Comunicação*. Plano Nacional de Saúde 2011-2016, Novembro 2010.

FACCHINI, Ana Rita; VARGAS, Lilia Maria. *Sistema de informação em uma organização do setor público*. Revista de Administração, In: São Paulo, 1992.

FALBO, Ricardo de Almeida. *Análise de Sistemas*. 2002. Disponível em: <http://www.paiossin.com/>. Acesso em: 15 jun. 2014.

FEMPA. *Curso Sistema TABWIN*. Paraná, Maio 2014.

FERRAZ, Lygia Helena Valle da Costa. *O SUS, o DATASUS e a informação em saúde: uma proposta de gestão participativa*. Rio d Janeiro, Dezembro 2009.

FERREIRA, D. et al. *SCRUM - Um Modelo Ágil para Gestão de Projetos de Software.* 2005. Disponível em: <<http://paginas.fe.up.pt/ aaguiar/es/artigosAcesso em: 15 set. 2006.

FIELDING, R. *Architectural Styles and the Design of Network-based Software Architectures.Tese (Doutorado)*. University of California, In: Irvine, 2000.

FILHO, Antonio Mendes da Silva. *Documento de Requisitos*: Essencial ao desenvolvimento de software. *Engenharia de Software Magazine*, 2005.

FISCHER, Robert. *Grails Persistence with GORM and GSQL*. 1. ed. Nova York: First Press, 2009.

FONSECA, Rúben; SIMõES, Alberto. *Alternativas ao XML: YAML e JSON*. Universidade do Minho.

FRANCO, Rebeca Such Tobias. *Estudo comparativo entre os frameworks Java para desenvolvimento de aplicações web: JSF 2.0, Grails e Spring Web MVC*. In: Paraná, 2011.

GAMA, Alexandre. *Introdução: JSON*. Disponível em: <http://www.devmedia.com.br/introducao-json/23166>. Acesso em: 14 jul. 2014.

GANDINI, J. A. D. *A judicialização do direito à saúde: a obtenção de atendimento médico, medicamentos e insumos terapêuticos por via judicial: critérios e experiências*. Ribeirão Preto - São Paulo, 2007.

GARTNER. *Gartner - Technology Research.* 2013. Disponível em: <http://www.gartner.com/technology/home.jsp>. Acesso em: 15 jun. 2014.

GUDGIN, Martin et al. *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. Abril 2007. Disponível em: <http://www.w3.org/TR/soap12-part1/#intro>. Acesso em: 26 jun. 2014.

GUEDES, Gildásio. *Interface Humano Computador: prática pedagógica para ambientes virtuais*. 1. ed. Teresina: Editora Gráfica da UFPI, 2009.

GUIMARAES, R. *Bases para uma política nacional de ciência, tecnologia e inovação em saúde*. *Ciência e Saúde Coletiva*, 2004.

HOSPUB. *Hospub*. 2011. Disponível em: <http://dev-hospub.datasus.gov.br/w3c/hp.php>. Acesso em: 10 jul. 2014.

JONATHAN, Miguel. *Introdução ao padrão de projeto MVC – Model-View-Controller*. In: Rio de Janeiro, Junho 2011.

JUNIOR, O. de O. B. *Técnicas de Análise de Sistema*. In: Santa Catarina, 1997.

JUNQUEIRA, L. A. P. *Gerência dos serviços de saúde*. *Caderno de Saúde Pública*, Rio de Janeiro, 1990.

KAMAL, Raj. *Embedded Systems - Architecture, Programming and Design*. 1. ed. India: McGraw-Hill, 2008.

KLEIN, Dave. *Grails: A Quick Start Guide*. 2. ed. Estados Unidos: Pragmatic Bookshelf, 2009.

KOERICH, Magda Santos et al. *Tecnologias de Cuidado em Saúde e Enfermagem e suas Perspectivas Filosóficas*. *Texto Contexto Enferm*, Florianópolis, 2006.

KOLLIKER, Mischa; SCHMUTZ, Guido. *Grails - Rapid Web Application Development for the Java Platform*. Trivadis, In: Zürich, Junho 2008.

KONIG, Dierk et al. *Groovy In Action: Covers Groovy 2*. 2. ed.: Manning Publications, 2013.

KUMAR, Rajesh. *Grails Web Framework*. [S.l.]: Sapient Corporation, 2013.

LECHETA, R. R. *Google Android - Aprenda a criar aplicações para dispositivos móveis com o Android SDK*. [S.l.]: Editora Novatec, 2009.

LEE, Wei-Meng. *Plataformas de Desenvolvimento para Dispositivos Móveis*. 1. ed. Estados Unidos: Wiley Publishing, 2011.

LEMOS, Marcius Hollanda Pereira da Rocha Vanda Márcia Ferri. *A gestão das organizações hospitalares e suas complexidades*. VII CONGRESSO NACIONAL DE EXCELÊNCIA EM GESTÃO, In: São Paulo, Agosto 2011.

LEMOS, Vanda Márcia Ferri; ROCHA, Marcius Hollanda Pereira da. *A Gestão das Organizações Hospitalares e suas Complexidades*. VII CONGRESSO NACIONAL DE EXCELÊNCIA EM GESTÃO, 2011.

LIBARDI, Paula L.O.; BARBOSA, Vladimir. *Métodos Ágeis*. In: São Paulo, 2010.

LIMA, Jean Carlos Rosário. *Web Services: SOAP x REST*. In: São Paulo, 2012.

LOUREIRO, Eduardo. *Aplicando a usabilidade em projetos web*. In: Minas Gerais, 2006.

LUCENA, Fabio Nogueira de; LIESENBERG, Hans K.E. *Interfaces Homem-Computador: Uma Primeira Introdução*. In: Campinas, Setembro 1994.

LUDVIG, Diogo; REINERT, Jonatas Davson. *Estudo do uso de Metodologias Ágeis no Desenvolvimento de uma Aplicação de Governo Eletrônico*. *Encontro de Computação e Informática do Tocantins*, In: Florianópolis, 2007.

MCWHIRTER, Bob; CHAMPEAU, Cédric. *Home*. Junho 2014. Disponível em: <http://docs.codehaus.org/display/GROOVY/Home>. Acesso em: 10 jun. 2014.

MEDNIEKS, Zigurd et al. *Programming Android*. 1. ed. Estados Unidos: O'Reilly Media, 2011.

MEIER, Reto. *Professional Android A pplication Development*. 1. ed. Estados Unidos: Wiley Publishing, 2009.

MEIER, Reto. *Professional Android Application Development*. [S.l.]: Wiley Publishing, 2009.

MORESI, Eduardo Amadeu Dutra. *Delineando o valor do sistema de informação de uma organização*. In: Brasília, 2000. Disponível em: <http://www.scielo.br/pdf/ci/v29n1/v29n1a2.pdf>. Acesso em: 10 jul. 2014.

MüLLER, Nícolas. *Metodologias Ágeis Extreme Programming e Scrum para o Desenvolvimento de Software*. Novembro 2008. Disponível em: <http://www.oficinadanet.com.br>. Acesso em: 28 mai. 2014.

NASCIMENTO, Emerson Brito do. *Desenvolvimento em Grails com abordagem mapeamento objeto relacional*. 2014. Disponível em: <http://www.din.uem.br/ ra58269/artigos/artigoAcesso em: 29 mai. 2014.

NERI, M.; SOARES, W. *Desigualdade social e saúde no Brasil*. *Ciência e Saúde Coletiva V. 4 (N. 1)*, Rio de Janeiro, 2009.

NETO, Gert Uchôa Müller. *Métodos Tradicionais versus Ágeis: um estudo comparativo através do trainingcad*. In: Pernambuco, Dezembro 2009.

NIELSEN, Jakob. *Usability Engineering*. Boston: Academic Press, 1993.

NIELSEN, Jakob. *10 Usability Heuristics for User Interface Design*. In: Minas Gerais, Janeiro 1995. Disponível em: <http://www.nngroup.com/articles/ten-usability-heuristics/>. Acesso em: 20 jun. 2014.

OAK, Harshad. *Groovy And Grails*. In: Asia, Agosto 2006.

OLIVEIRA, Ebenezer Silva de. *Uso de Metodologias Ágeis no Desenvolvimento de Software*. In: Belo Horizonte, 2013.

OLIVEIRA, Marcelo. *REST e RESTful: a diferença*. Março 2013. Disponível em: <http://marceloweb.info/rest-e-restful-a-diferenca/>. Acesso em: 15 jun. 2014.

PADILHA, K. G. *Considerações sobre as ocorrências iatrogênicas na assistência à saúde: dificuldades inerentes ao estudo do tema*. *Revista Escola de Enfermagem USP 2001*, São Paulo, 2001.

PEREIRA, Silvia Cássia; LUCENA, Márcia Jacyntha Nunes Rodrigues. *Documento de Requisitos*: Projeto; vensso. 2005.

PINOCHET, Luis Hernan Contreras. *Tendências de Tecnologia de Informação na Gestão da Saúde*. *O Mundo da Saúde*, In: São Paulo, 2011.

PINOCHET, Luis Hernan Contreras. *Tendências de Tecnologia de Informação na Gestão da Saúde*. *O Mundo da Saúde*, São Paulo, 2011.

PIVOTAL. *Domain Class Usage*. 2014. Disponível em: <http://grails.org/doc/2.4.x/ref/Domain9 jun. 2014.

PLAY, google. *Mais de 700.000 Apps e Jogos*. 2013. Disponível em: <https://play.google.com/about/apps/>. Acesso em: set. 2013.

PRESSMAN, R. S. *Engenharia de Software. 6ª Ed*. [S.l.]: McGraw-Hill, 2006.

PRODATER (Ed.). Metodologia de desenvolvimento de software. 2011.

PROJECT, Ubuntu Documentation. *Trabalhando com seu desktop*. 1. ed.: Canonical, 2004.

RABELLO, Ramon Ribeiro. *Android: um novo paradigma de desenvolvimento móvel*. 2014. Disponível em: <http://www.cesar.org.br>. Acesso em: 28 jun. 2014.

REENSKAUG, Trygve. *Modelo MVC - Model View Controller*. Dezembro 1979.

REENSKAUG, Trygve. *MODELS - VIEWS - CONTROLLERS*. Dezembro 1979.

REENSKAUG, Trygve. *MVC XEROX PARC 1978-79*. Março 1979. Disponível em: <http://heim.ifi.uio.no/ trygver/themes/mvc/mvc-index.html>. Acesso em: 1 jul. 2014.

REENSKAUG, Trygve. *The Model-View-Controller (MVC) Its Past and Present*. Agosto 2003.

REZENDE, Marcel Hugo Vandir Fernando. *Análise comparativa entre Groovy e Java, aplicado no desenvolvimento web*. In: Santa Catarina, 2011.

ROCHA, H.V; BARANAUSKAS, M.C.C. *Design e avaliação de interfaces humano-computador*. NIED/UNICAMP, In:Campinas, 2003.

ROCHA Álvaro. *O Essencial da Análise de Sistemas*. In: Portugal, 2008.

SALABERRI, D. B. *Desenvolvimento de Aplicações Embarcadas com Android*. 2011. Disponível em: <http://inf.ufpel.edu.br/>. Acesso em: ago. 2013.

SANTOS, Adriano. *MySQL: Quem é você?* 2014. Disponível em: <http://www.devmedia.com.br/mysql-quem-e-voce/1752>. Acesso em: 15 jul. 2014.

SANTOS, Alexandro Klein dos. *Os IDE's (Ambientes de Desenvolvimento Integrado) como ferramentas de trabalho em informática*. 2014. Disponível em: <www-usr.inf.ufsm.br/ alexks/elc1020/artigo-elc1020-alexks.pdf>. Acesso em: 21 jun. 2014.

SANTOS, Daniel da Cruz; COSTA, Rogério Homem da. *Interface Humano-Computador (ou Homem-Máquina) e o desafio da mobilidade*. *Sinergia*, In: São Paulo, Janeiro 2014.

SCARPI, M. J. *Gestão de Clínicas Médicas*. São Paulo: Futura, 2004.

SCHWABER, Jeff Sutherland Ken. *Um guia definitivo para o Scrum: As regras do jogo*. Outubro 2011.

SEGRE, M.; FERRAZ, F. C. *O Conceito de Saúde*. *Revista de Saúde Pública V.31 (N. 5)*, 1997.

SENA, Hugo. *Groovy e Grails*. In: Rio Grande do Norte, 2010.

SILVA, Elton; SILVEIRA, Milene. *Estilos de Interação*. 2014. Disponível em: <http://www.decom.ufop.br/>. Acesso em: 22 jun. 2014.

SILVA, Tiago de Farias. *Compondo Métodos Ágeis de Desenvolvimento de Software*. In: Recife, Julho 2009.

SMITH, Peter Ledbrook Glen. *Grails in Action*. Greenwich: Manning, 2009.

SOARES, Michel dos Santos. *Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software*. In: São Paulo, 2004. Disponível em: <http://www.dcc.ufla.br/infocomp/artigos/v3.2/art02.pdf>. Acesso em: 11 jan. 2007.

SOARES, Michel dos Santos. *Metodologias Ágeis Extreme Programming e Scrum para o Desenvolvimento de Software*. In: Minas Gerais, 2014. Disponível em: <http://revistas.facecla.com.br/index.php/reinfo/article/download/146/38>. Acesso em: 13 jun. 2014.

SOUSA, Cleydiane Lima de; SOUZA, Jackson Gomes de. *Metodologia de Desenvolvimento de Software para a Fábricade Software do CEULP/ULBRA*. *Encontro de Computação e Informática do Tocantins*, In: Palmas, 2012. Disponível em: <http://ulbra-to.br/encoinfo/artigos/2012/>. Acesso em: 15 jun. 2014.

SOUSA, Fabricio de. *Principais Estilo de Interação*. 2014. Disponível em: <http://www.fabriciosousa.com/>. Acesso em: 22 jun. 2014.

SOUZA, C. S. *Projeto de interfaces de usuário: perspectivas cognitiva e semiótica*. XIX Congresso da Sociedade Brasileira de Computação, In:Rio de Janeiro, Julho 2003.

SOUZA, Luciano Soares de; SPINOLA, Mauro de Mesquita. *Requisitos de usabilidade em projetos de interface centrado no usuário de software de dispositivos móveis*. XXVI ENEGEP, In: Fortaleza, Outubro 2006.

STEELE, James; TO, Nelson. *The Android Developer's Cookbook*. 1. ed. Estados Unidos: Pearson Education, 2011.

TANJI, T. *Android está presente em 900 milhões de aparelhos*. 2013. Disponível em: <http://exame.abril.com.br/>. Acesso em: set. 2013.

TAVARES, Tatiana Aires. *Interação Humano Computador*.

THAKKAR, U. *Ethics in the design of human-computer interfaces for the disabled*. *SIGCAPH Newsletter*, 1990.

TIC, Saúde 2013. *Pesquisa sobre o uso das tecnologias e informação e comunicação nos estabelecimentosde saúde brasileiros*. São Paulo: CETIC, 2014.

URDAN, André Torres. *A qualidade de serviços médicos na perspectiva do cliente*. *RAE Revista de Administração de Empresas*, São Paulo, 2001.

UTIDA, Kleber Hiroki. *Metodologias tradicionais e metodologias ágeis: Análise Comparativa entre Rationla Unified Process e Extreme Programming*. In: São Paulo, 2012.

'UYUN, Shofwatul; MA'ARIF, Muhammad Rifqi. *Implementation of model view controller (MVC) architeture on building web-based information system*. Seminar Nasional Aplikasi Teknologi Informasi 2010, In: Yogyakarta, Junho 2010.

W3C. *Simple Object Access Protocol (SOAP) 1.1*. 2000. Disponível em: <http://www.w3.org/TR/2000/NOTE-SOAP-20000508>. Acesso em: 14 jun. 2009.

W3C. *Web Services Architecture: W3C Working Group 2004*. 2004. Disponível em: <http://www.w3.org/TR/ws-arch>. Acesso em: 13 mai. 2009.

WEISSMANN, Henrique Lobo. *Closures*. Janeiro 2009. Disponível em: <http://www.itexto.net/devkico/?p=241>. Acesso em: 13 jul. 2014.

WEISSMANN, Henrique Lobo. *Grails não é bala de prata*. Janeiro 2009. Disponível em: <http://www.itexto.net/devkico/?p=496>. Acesso em: 13 jul. 2014.

WEISSMANN, Henrique Lobo. *Grails: entendendo as validações (constraints)*. Janeiro 2011. Disponível em: <http://www.itexto.net/devkico/?p=909>. Acesso em: 9 jun. 2014.

ZON, Alexandre et al. *Apostila Linux Básico*. In: Vitória, 2007.

ROCHA, H. V. da et al. Design e avaliação de interfaces humano-computador. Campinas: Unicamp, 2003.