FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
CENTRO UNIVERSITÁRIO "EURÍPIDES DE MARÍLIA" – UNIVEM
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

JOÃO PAULO DOS SANTOS MORIJO

DESENVOLVIMENTO DE UMA ARQUITETURA DE AUXÍLIO E MONITORAMENTO PARA PACIENTES PORTADORES DE DIABETES

MARÍLIA
2014

JOÃO PAULO DOS SANTOS MORIJO

DESENVOLVIMENTO DE UMA ARQUITETURA DE AUXÍLIO E MONITORAMENTO PARA PACIENTES PORTADORES DE DIABETES

Trabalho de Curso apresentado ao Curso de Graduação em Ciência da Computação, do Centro Universitário Eurípides de Marília, mantido pela Fundação de Ensino Eurípides Soares da Rocha, como requisito para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Ms. Ricardo José Sabatine.

MARÍLIA
2014

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO – AVALIAÇÃO FINAL

João Paulo dos Santos Morijo

DESENVOLVIMENTO DE UMA ARQUITETURA DE AUXÍLIO E MONITORAMENTO PARA PACIENTES PORTADORES DE DIABETES

Banca examinadora da monografia apresentada ao Curso de Bacharelado em Ciência da Computação do UNIVEM/F.E.E.S.R., para obtenção do Título de Bacharel em Ciência da Computação.

Nota: 9.5 (nove e meio)

Orientador: Ricardo José Sabatine

1°. Examinador: Mauricio Duarte

2°. Examinador: Leonardo Castro Botega

Marília, 05 de dezembro de 2014.

MORIJO, João Paulo dos Santos

DESENVOLVIMENTO DE UMA ARQUITETURA DE AUXÍLIO E MONITORAMENTO PARA PACIENTES PORTADORES DE DIABETES / João Paulo dos Santos Morijo; orientador: Prof. Ms. Ricardo José Sabatine. Marília, SP: [s.n.], 2014.

Trabalho de Conclusão de Curso (TCC) - Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha.

Dedico este trabalho a todos que me incentivaram e ajudaram na realização deste sonho, sendo em destaques meus pais, Edson e Cilene.

AGRADECIMENTOS

Agradeço primeiramente a Deus, por ter me dado a força necessária em todos os momentos de minha vida e a sabedoria para a conclusão deste trabalho. Agradeço a minha família, meus Pais Edson e Cilene, que sempre estiveram ao meu lado, me apoiando em minhas decisões e sempre torceram por mim; aos meus irmãos Daniel, Karen e Leticia; a minha namorada Lívia, pela inspiração, apoio e incentivo.

Agradeço ao meu professor e orientador Prof. Ms. Ricardo José Sabatine, o qual me encaminhou e direcionou desde o início, me mostrando os erros e parabenizando pelos acertos.

Aos amigos e companheiros que estiveram comigo durante toda essa caminhada, destes posso destacar Felipe Gadelha Ruoso e Eduardo Arakaki; aos professores da universidade pelo compartilhamento do conhecimento, sendo fundamental para o crescimento acadêmico e pessoal, a toda a equipe Persys, aos professores Adriano e Giulianna, pelo vasto conhecimento adquirido; ao Programa Escola da Família.

Por fim, agradeço ao Centro Universitário Eurípides de Marília - UNIVEM por tornar meu sonho realidade.

"Que os vossos esforços desafiem as impossibilidades, lembrai-vos que as grandes coisas do homem foram conquistadas do que parecia impossível"

Charles Chaplin

MORIJO, João Paulo do Santos. **DESENVOLVIMENTO DE UMA ARQUITETURA DE AUXÍLIO E MONITORAMENTO PARA PACIENTES PORTADORES DE DIABETES**. 2014. 74f. Trabalho de Curso (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino "Eurípides Soares da Rocha", Marília, 2014.

RESUMO

O diabetes é uma síndrome metabólica de origem relacionada ao fornecimento de insulina pelo pâncreas. O problema ocorre quando há insuficiência total ou parcial no fornecimento de insulina ou quando a insulina não tem a capacidade de exercer adequadamente sua função. Em qualquer das situações, ocorre um aumento do nível de glicose (açúcar) no sangue. Esta é uma doenças que mais cresce atualmente, tornando assim necessário uma arquitetura de software que possa atender a demanda dos pacientes, sendo esta escalável, confiante e estável para seus usuários. Propõe-se neste trabalho o desenvolvimento de uma arquitetura que reúna aplicações e informações que monitore e auxilie os controles glicêmicos facilitando a procura por medicações e integração direta com a equipe médica, com foco principal na diabetes tipo 1, ou seja, a diabetes que torna o paciente insulinodependente.

Palavras-chave: Microservice, Arquitetura de Software, Diabetes, Geolocalização, Auxílio e Controle Glicêmicos.

MORIJO, João Paulo do Santos. **DESENVOLVIMENTO DE UMA ARQUITETURA DE AUXÍLIO E MONITORAMENTO PARA PACIENTES PORTADORES DE DIABETES**. 2014. 74f. Trabalho de Curso (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino "Eurípides Soares da Rocha", Marília, 2014.

ABSTRACT

Diabetes is a metabolic syndrome source connected to the supply of insulin by the pancreas. The problem occurs when there is total or partial failure in the supply of insulin or when insulin is not able to properly perform its function. Whatever the case, there is an increased level of glucose (sugar) in the blood. This is a disease that fastest growing, thus making necessary a software architecture that can meet the demands of patients, which is scalable, confident and stable for its users. It is proposed in this paper the development an architecture that brings together applications and information to monitor and assist glycemic controls facilitating the search for medications and direct integration with the medical team, with primary focus on type 1 diabetes, or diabetes that makes the insulin patient.

Keywords: Microservice, Software Architecture, Diabetes, Geolocation, Aid and Control Blood glucose.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

FID	International Diabetes Federation
OMS	Organização mundial de saúde
A1c	Hemoglobina Glicosilada
DER	Diagrama Entidade Relacionamento
GPS	Global Position System
SOA	Service Oriented Architecture
HTTP	Hyper Text Transfer Protocol
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
MVC	Model View Controller
ORM	Object Relacional Mapping
REST	Representation State Transfer
SDK	Software Development Kit
SQL	Structured Query Language
UML	Unifield Modeling Language
GWT	Google Web Toolkit
RIA	Rich Internet Application
SGBD	Data Base Management System
DSL	Domain Specific Language
WMS	Warehouse Management System
PNG	Portable Network Graphics
JPEG	Joint Photographic Expert Groups
TIFF	Tagged Image File Format
OGC	Open Geospatial Consortium
DOM	Document Object Model
CSS	Cascading Style Sheets
HTML	HyperText Markup Language
XML	eXtensible Markup Language

SUMÁRIO

INTRODUÇÃO

O diabetes é uma síndrome metabólica de origem múltipla, decorrente da falta de insulina e/ou da incapacidade da insulina exercer adequadamente sua função, causando um aumento da glicose (açúcar) no sangue. Embora exista uma ampla variedade de aplicações móveis disponíveis para pessoas com diabetes, há lacunas entre as recomendações baseadas em evidências e as funcionalidades encontradas nas aplicações (TARIDZO, et al., 2011), ou seja a pessoa a qual desenvolveu a diabetes agora tem necessidades diferentes de quem convive com a diabetes a tempos.

Uma das principais faltas de motivação que o portador de diabetes tem em controlar os seus níveis glicêmicos é a falta de conhecimento referente aos prováveis problemas de saúde que ele pode sofrer caso não mantenha sua glicemia controlada. O diabetes é uma doença, até então, sem cura (JOSLIN, 2013), mas ela sendo bem controlada, o paciente tende a levar uma vida normal.

Levantamentos realizados pela Federação Internacional de Diabetes (FID) mostram o grande aumento de pessoas portadoras da doença pelo mundo, em 2011 foi constatado 366.2 milhões de pessoas portadoras de um dos tipos da doença, já em 2013 esse número teve um grande aumento passando para 381.8 milhões de pessoas (ATLAS FID, 2013), como ilustrados nas Tabela 1 e Tabela 2.

Tabela 1 - Levantamento por região FID 5º edição, 2011

REGIÃO	2011 (MILHÕES)	2030 (MILHÕES)	AUMENTO (%)
África	14.7	28.0	90%
Oriente Médio e Norte da África	32.8	59.7	83%
Ásia	71.4	120.9	69%
América do Sul e Central	25.1	39.9	59%
Pacífico Ocidental	131.9	187.9	42%
América do Norte e Caribe	37.7	51.2	36%
Europa	52.6	64.0	22%
Mundo	**366.2**	**551.8**	**51%**

Fonte: Portal FID, 2011

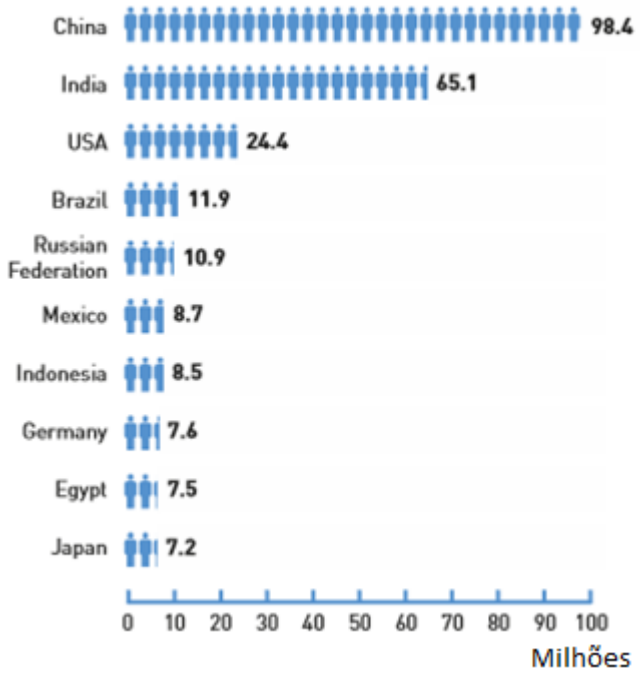
Tabela 2 - Levantamento por região FID 6º edição, 2013

REGIÃO	2013 (MILHÕES)	2035 (MILHÕES)	AUMENTO (%)
África	19.8	41.4	109%
Oriente Médio e Norte da África	34.6	67.9	96%
Ásia	72.1	123	71%
América do Sul e Central	24.1	38.5	60%
Pacífico Ocidental	138.2	201.8	46%
América do Norte e Caribe	36.7	50.4	37%
Europa	56.3	68.9	22%
Mundo	**381.8**	**591.9**	**55%**

Fonte: Portal FID, 2013

Um levantamento mais detalhado ilustra os 10 países com maior índice da doença. O Brasil se encontra em 4º colocado, perdendo apenas para China, em 1º lugar com 98,4 milhões, 2º Índia com 45,1 milhões e os Estados Unidos, com 24,4 milhões, como ilustrado na Figura 1.

Figura 1 - Países com maior número de pessoas com diabetes (20-79 anos)



Fonte: Portal FID, 2013

Este crescimento está relacionado a uma mistura de envelhecimento, sedentarismo, obesidade e alimentação nada saudável da população, sendo esses itens citados alguns dos principais fatores que contribuem para esse aumento (FID, 2013).

Com aumento crescente dos casos de diabetes, se torna necessário uma arquitetura que atenda às necessidades dos pacientes. Sendo neste trabalho apresentado uma arquitetura que se baseia no conceito de SOA ou *Microservices*, onde o desenvolvimento do sistema é constituído como um conjunto de pequenos serviços/aplicações, sendo estes trabalho implementados por ferramentas e tecnologias que visam a simplificação e aumento na produtividade, de forma que esta arquitetura possa atender as necessidades sendo em destaque escalável, estável e confiante para seus usuários.

Como podemos ressaltar esta arquitetura é composta por uma diversidade de ferramentas e tecnologias, sendo cada uma separada por suas respectivas funcionalidades.

- Aplicação Móvel – Funcionalidades que auxiliam o paciente
- Aplicação Web – Monitoramento Equipes medicas ao paciente
- API REST – Sincronização e Integração entre Aplicações
- GeoServer – Plotagem de pontos aos mapas
- Jenkins – Serviço de Entrega continua
- BitBucket – Repositorio das aplicações
- Zabbix – Monitoramento das aplicações

Como apresentado esses são os componentes que representam a arquitetura Diacon, sendo cada item detalhado com sua implementação e utilização nos capítulos seguintes.

Objetivos

Com o grande aumento de casos de diabetes, se torna necessário uma arquitetura de software que possa atender a demanda dos pacientes, sendo esta escalável, confiante e estável para seus usuários assim auxiliando suas necessidades, sendo estas não somente para controle de horários ou anotações dos níveis glicêmicos, mas sim que auxilie na prática correta da automonitorização da glicemia, como os níveis glicêmicos; a dosagem junto dos horários de medicação; o horário e a quantidade de carboidratos existente nos alimentos ingeridos; o controle por atividade física e também a forma mais rápida e eficaz de se encontrar os devidos medicamentos, tornando todas essas informações acessíveis à equipe médica do paciente.

Os objetivos específicos basicamente são:

- Desenvolver uma arquitetura escalável, confiante e estável.

- Desenvolvimento de aplicações de fácil manuseio e entendimento.
- Pesquisar e entender as funcionalidades necessárias para aplicações de auxílio a pacientes portadores de diabetes.
- Permitir o acesso às informações por parte da equipe médica.

Organização do Trabalho

Estruturalmente, este trabalho é composto por 4 capítulos.

No capítulo 1 é realizada uma contextualização sobre o tema diabetes.

No capítulo 2 é realizada uma apresentação sobre o que é a diabetes seus tratamentos e os trabalhos relacionados ao tema abordado.

No capítulo 3 são apresentados as tecnologias utilizadas para o desenvolvimento.

No capítulo 4 é contextualizado os resultados esperados desse projeto.

1 DIABETES

O diabetes é uma doença do metabolismo da glicose causada pela falta ou má absorção de insulina, hormônio produzido pelo pâncreas, cuja função é quebrar as moléculas de glicose para transformá-las em energia a fim de que seja aproveitada por todas as células. A ausência total ou parcial desse hormônio interfere não só na queima do açúcar, como na sua transformação em outras substâncias (proteínas, músculos e gordura) (JOSLIN, 2009).

1.1 Diabetes Tipo I

No diabetes tipo I, antes conhecido como “juvenil de início” ou “insulino-dependente”, o pâncreas deixa de produzir a insulina necessária pelo corpo, um hormônio que permite o organismo usar a glicose (açúcar) encontrado em alimentos como fonte de energia. Ao invés do corpo converter a glicose em energia, ele faz um estoque da glicose na corrente sanguínea e causa uma variedade de sintomas, incluindo fadiga (JOSLIN, 2009).

O diabetes tipo I é diferente de diabetes tipo II, porque é tratável apenas com insulina injetável, através de seringas, canetas injetoras ou por meio de uma bomba de insulina. Muitos dos sintomas de diabetes tipo I são iguais ao do tipo II, para isso o médico irá realizar exames

que determinarão o grupo em que o paciente se encontra (JOSLIN, 2013).

1.2 Diabetes Tipo II

No diabetes tipo II (anteriormente chamado de "adulto-início" ou "não insulino-dependentes"), o organismo não produz insulina suficiente e/ou não é capaz de utilizar adequadamente a insulina produzida (isso também é conhecido como "resistência à insulina"). Esta forma de diabetes normalmente ocorre em pessoas acima dos 40 anos de idade, pessoas com excesso de peso e/ou pessoas que tenham algum histórico da doença na família, apesar de hoje em dia ser cada vez mais encontrada em pessoas mais jovens (JOSLIN, 2013).

O diabetes tipo II pode ser causada por uma variedade de fatores: excesso de peso, ser fisicamente inativo (sedentário), ou a incapacidade do organismo de utilizar adequadamente a insulina que produz (OMS,2013).

O tratamento para diabetes tipo 2 pode ser feito com a ingestão de remédios via oral, isso acontece caso os níveis de açúcar no sangue não possam ser adequadamente estabilizados com um estilo de vida melhor (alimentos mais saudáveis, perder peso se necessário e atividades físicas), sendo então necessário a utilização de medicamentos para tratamento da diabetes (DANBURY HOSPITAL, 2007).

1.3 Diabetes Gestacional

O diabetes gestacional ocorre durante a gravidez e, na maior parte dos casos, é provocado pelo aumento excessivo de peso da mãe (DRAUZIO.V, 2014).

O tratamento da diabetes gestacional é importante para a gestante e para o feto, e pode ser feito com uma dieta com baixo teor de açúcar, gordura e carboidratos, prática regular de exercícios físicos e a ingestão de insulina, sob orientação médica (FRAZÃO.A, 2014).

O exercício e a alimentação adequada são importantes pois eles diminuem o risco da permanência da diabetes após o nascimento do bebê (FRAZÃO.A, 2014).

1.4 Tratamento

O objetivo do tratamento do diabetes, independente do tipo da doença, tem como foco controlar os níveis glicêmicos do sangue e prevenir complicações.

As dietas alimentares devem ser equilibradas e também deve haver a prática de exercícios físicos para um bom controle da diabetes, para isso torna-se necessária a orientação de uma nutricionista e um endocrinologista.

1.5 Trabalhos Relacionados

Neste capítulo, foram analisadas aplicações e artigos voltados para esta área de estudo, afim de entender melhor a influência da tecnologia nesses tipos de tratamentos, assim como as reais necessidades de um paciente.

1.5.1 Aplicações moveis

Existe uma grande variedade de aplicações para controle de diabetes, mas esses aplicativos não abrangem todas as necessidades que um portador de diabetes precisa.

Foram testados quatro aplicativos (APP) para *Smartphones* Android *(Play Store* – Acesso em Março 2014) sendo que, apenas um deles contém maiores informações a respeito da doença e só é encontrado em espanhol o aplicativo “diabetes”, os demais aplicativos testados, tinham como foco principal apenas as anotações dos níveis glicêmicos e a elaboração de gráficos. Dentre os aplicativos encontrados, alguns são mais completos e possuem um acompanhamento mais preciso, porém, sua licença não é gratuita, já os que se enquadram nessa categoria, são internacionais e/ou muito limitados, sendo que nenhum deles possuem integração com demais aplicações, conforme proposto neste projeto.

Uma lista dos aplicativos testados é apresentada e logo em seguida, a Figura 2 ilustra a interface dos mesmos, respectivamente:

- Diabetes – Diário Glucose
- Prognosis – Diabetes
- Diabetes
- Diário Diabetes

Figura 2 - Aplicações android Testadas

Fonte: Portal Play Store, 2014

1.5.2 Artigos Relacionados

Foram levantados artigos relacionados a esta área de pesquisa, para um melhor entendimento sobre a influência que a tecnologia exerce sobre os tratamentos envolvendo a diabetes.

O artigo analisado, *WellDoc™ Mobile Diabetes Management Randomized Controlled Trial*, o objetivo se situava por avaliar os impactos na A1c, utilizando uma aplicação móvel para monitoração em tempo real dos pacientes pelas equipes de saúde (CHARLENE, et al., 2008).

O aplicativo foi testado durante três meses em trinta pacientes adultos com diabetes tipo 2, enviando os dados em tempo real para as equipes médicas informando os níveis glicêmicos e medicações utilizadas pelos pacientes, com a finalidade de obter melhores resultados no tratamento (CHARLENE, et al., 2008).

A equipe médica responsável pelo acompanhamento relatou que o sistema facilitou as tomadas de decisões, desde que os dados estivessem organizados, reduzindo o tempo da análise do diário de bordo glicêmico do paciente, obtendo melhores analises e diagnósticos, levando os pacientes e a equipe médica que utilizaram o aplicativo *WellDoc*, estatisticamente, à melhorias significativas na A1c (CHARLENE, et al., 2008).

Dentre os artigos analisados, está o Mobile diabetes intervention study: Testing a

personalized treatment/behavioral communication intervention for blood glucose control (2009), com o objetivo de monitorar um grupo de pacientes durante um ano e analisar os resultados obtidos através do monitoramento.

Nesta pesquisa foi revelado que os métodos tradicionais para o tratamento da diabetes não obtiveram resultados adequados, entretanto, os pacientes que utilizaram uma ferramenta para melhorar a comunicação entre paciente e equipe médica foram capazes de obter melhores resultados (CHARLENE, et al., 2009).

O artigo *Mobile phone text messaging in the management of diabetes*, utilizou uma abordagem diferente, troca de mensagens de texto para o gerenciamento das pessoas diabéticas em um período de oito meses, contando com 23 pacientes para utilização do serviço.

O fluxo do aplicativo consistia no envio de uma mensagem do paciente para o servidor, que por sua vez, enviava a confirmação de recebimento, porém, caso o paciente ficasse sem enviar a mensagem um dia, a mesma não poderia mais ser enviada e isso gerava um pouco de preocupação (FERRER-ROCA, et al., 2004).

Entretanto, este tipo de serviço foi bem aceito entre adolescentes e idosos, fases onde o controle da diabetes se torna mais difícil de ser controlada (FERRER-ROCA, et al., 2004).

Dentre os artigos analisados, o Features of Mobile Diabetes Applications: Review of the Literature and Analysis of Current Applications Compared Against Evidence-Based Guidelines (2011) mostrou um grande deficit em relação aos aplicativos existentes nas lojas virtuais, demonstrando que embora exista uma imensa quantidade de aplicativos disponíveis, existem lacunas entre as reais recomendações e as funcionalidades obtidas pelos mesmos.

Muitos aplicativos focam somente em algumas partes especificas do diabetes, como por exemplo, anotar a glicemia e fazer cálculos de insulina necessária. Os autores dos aplicativos existentes não sabem ou apenas esquecem que não é somente isso o que um portador de diabetes precisa saber, e sim um conjunto de informações para atender suas reais necessidades, tornando-se necessário vários aplicativos diferentes.

Os critérios avaliados neste artigo foram:

- Insulina e gravação de medicação;
- Exportação de dados e comunicação;
- Gravação de dieta;
- Gestão de peso;
- Sincronização com servidor Web.

Embora as orientações clínicas amplamente referem a importância da educação, estes estão ausente das principais funcionalidades em ambos os casos (TARIDZO, et al., 2011).

1.6 Considerações Finais

Conclui-se, ao final deste capitulo que apesar de existir uma ampla variedade de aplicações, ainda existe a necessidade de uma ferramenta que abranja todas as funcionalidades que um portador de diabetes precisa.

A proposta implementada neste projeto é uma arquitetura que abranja as necessidades dos pacientes e consiga transmitir essas informações coletadas diretamente aos médicos ou equipes responsáveis, podendo assim, otimizar tratamentos, agendar consultas, consultar locais de saúde e facilitar o controle das equipes médicas e do próprio paciente.

2 TECNOLOGIAS DE DESENVOLVIMENTO

Neste capitulo são descritas as tecnologias utilizadas durante a implementação deste projeto.

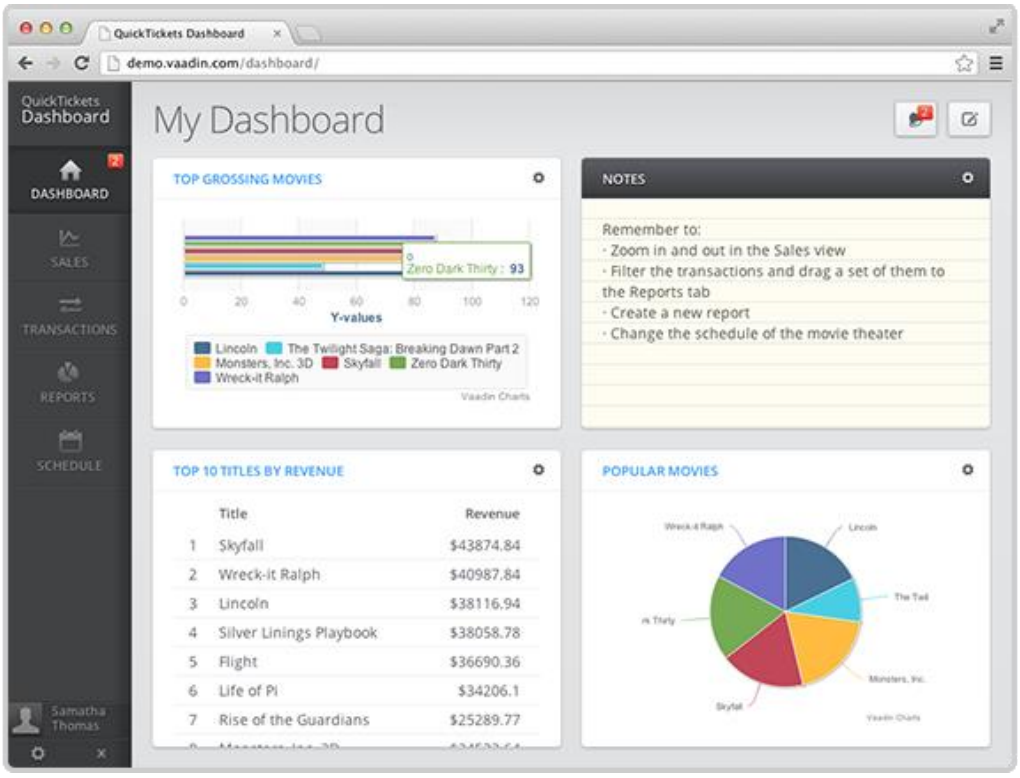
2.1 Aplicação Web

Aplicações RIA (*Rich Internet Application*) são aplicações que possuem algumas características e funcionalidades das tradicionais aplicações *desktop*. Do ponto de vista do desenvolvimento, essas aplicações seguem um padrão de modelo com controles ricos incluindo poderosas alternativas de gerenciar os dados e a capacidade de disponibilizar conteúdo multimídia, possibilitando um conjunto de informações ricas e com uma interface atrativa. Pela perspectiva do design, essas aplicações possibilitam uma alta capacidade de customizações com o uso avançado de recursos como o CSS. (SMEETS, 2009)

O VAADIN *Framework web open source* é utilizado para o desenvolvimento de interfaces (RIA), fazendo o melhor uso do AJAX (*Asynchronous JavaScript* e XML), técnicas que tornam possível a criação de *Rich Internet Applications* (RIA), que são tão ágeis e interativos como aplicações *desktop*. (VAADIN, 2014)

O lado cliente do *Vaadin* é baseado no *Google Web Toolkit* (GWT). Seu objetivo é tornar possível o desenvolvimento de interfaces de usuário web que rodam em navegadores facilmente com Java ao invés de *JavaScript*. Os Módulos do lado do cliente são desenvolvidos com Java e compilados em *JavaScript* com o *Vaadin Compiler*, que é uma extensão do quadro do cliente GWT *Compiler.* O quadro do lado do cliente também se esconde muito da manipulação DOM/HTML e permite que eventos do navegador sejam manipulados Java (VAADIN, 2014), a Figura 3 ilustra a interface do lado cliente do *Vaadin.*

Figura 3 - Modelo (RIA) framework VAADIN



Fonte: Portal VAADIN, 2014

2.2 Servidor HTTP

Os servidores web são responsáveis pela interação e comunicação entre máquinas distintas. Para isto, é necessário o envolvimento de pelo menos dois participantes para a realização de troca de mensagens, um cliente, que solicita informações, e um servidor, que atende a esses pedidos e os responde. (SILVEIRA, et al., 2014)

Ambos os lados exigem programas específicos para esta troca de mensagens, do lado do cliente é necessário um browser ou algum programa especifico para efetuar as requisições. Já do lado do servidor, as coisas não são tão simples, existem várias opções de software disponível, mas todos têm uma tarefa semelhante, negociar à transferência de dados entre clientes e servidores via HTTP (Protocolo de Transferência de Hipertexto), protocolo de comunicações Web. (SILVEIRA, et al., 2014)

O *Jetty* é um servidor HTTP e *Servlet Container* escrito em Java, com bom desempenho e boa estabilidade, sendo essas informações apresentadas em fóruns e debates na internet, é utilizado em uma ampla variedade de projetos e produtos, tanto para o desenvolvimento como para produção, podendo facilmente ser incorporado em diversos dispositivos, ferramentas, *frameworks*, e *clusters*. (Jetty, 2014).

Entre as suas características é possível destacar (Jetty, 2014).

- Projeto *open source*
- Flexível e extensível
- Assíncrono

Uma das desvantagens do *jetty* é não possuir um console de gerenciamento, sendo toda a sua configuração realizada em arquivos XML, porém, sem afetar seu desempenho.

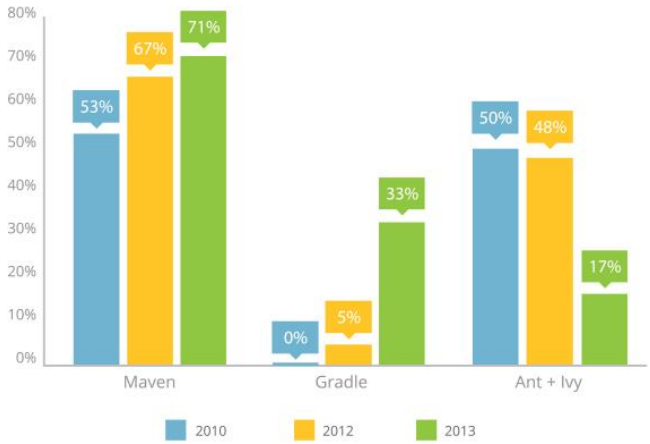
2.3 Gerenciamento de Dependências

O *Gradle,* uma ferramenta que combina o poder e a flexibilidade do *ivy* com a gestão e convenções de dependência do *Maven* em uma forma mais eficaz para construção de projeto (Gradle, 2014). Alimentado por um DSL *Groovy*, essa ferramenta fornece uma maneira declarativa para descrever todos os tipos de compilações através de padrões sensíveis (Gradle, 2014).

Uma ferramenta *open source* utilizada para o acompanhamento de todo o ciclo de vida do projeto, desde automatização, construção, teste e *deploy* de aplicações Java, Grovy e Scala. Projeto desenvolvido pela *Gradeware*, com integração a servidores de integração continua e gerenciadores de repositórios. (Gradle, 2014). De acordo com pesquisas e fóruns na internet, essa é uma ferramenta que vem crescendo muito ao longo dos tempos, sendo esta já incorporada como padrão no Android Studio.

A Figura 4 ilustra levantamento realizado pelo portal Zero Turnaround, em relação a popularidade das ferramentas de construções.

Figura 4 – Ferramentas Populares para construção, final de 2010 a meados 2013



Fonte: Portal Zero Turnaround, 2014

2.4 Persistência de dados

Ferramentas conhecidas como ORM, são capazes de fazer uma ponte de comunicação entre o paradigma entidade relacional e o orientado a objetos de forma a minimizar o abismo entre os conceitos dessas duas abordagens, conhecido como a impedância objeto-relacional (SILVEIRA, et al., 2014).

Um ORM (*Object-Relational Mapping*), é um *Framework* ou um conjunto de classes que permite que você faça este trabalho sem precisar escrever códigos de conexão com o banco, *queries* de SQL a todo momento, preservando as características de orientação a objetos da linguagem. (SILVEIRA, et al., 2014)

Dentre os *framework* para mapeamento objeto relacional (ORM) se apresenta o *Hibernate,* que tem como finalidade a facilitação do mapeamento dos atributos entre uma base tradicional de dados relacionais e o modelo objeto de uma aplicação, mediante a utilização de arquivos (XML) ou anotações como podemos ver em alguns exemplos apresentados. (HIBERNATE, 2014).

- ✓ @Entity: indica que objetos dessa classe se tornem "persistível" no banco de dados.
- ✓ @Table: indica o nome que será salvo ao banco.
- ✓ @Column: indica colunas pertencentes àquela tabela.
- ✓ @Id: indica que o atributo id é a chave primária (você precisa ter uma chave primária em toda entidade).
- ✓ @GeneratedValue: indica ao banco que este recurso seja auto-incrementado ou sequencial (*increment or sequence*).
- ✓ @Temporal: indica ao banco o armazenamento de horas, sendo ela separadas em:
- ✓ @Temporal(value=TemporalType.DATE) armazenamento apenas datas.
- ✓ @Temporal(value=TemporalType.TIME) armazenamento apenas horas.
- ✓ @Temporal(value=TemporalType.TIMESTAMP) armazenamento de data e horas.

O objetivo deste *framework* é diminuir a complexidade de desenvolvimento relacionados à persistência, conduzindo o desenvolvedor a concentrar esforços na implementação e deixar essa tarefa para o *framework*.

2.5 Aplicação Móvel

O Android é um sistema operacional baseado no núcleo do Linux, teve seu desenvolvimento iniciado em 2003 pela empresa Android Inc. Em 2005, a empresa foi adquirida pelo Google, que hoje lidera o desenvolvimento do Android (MONTEIRO, 2014).

Esse sistema é apontado em diversas pesquisas como o sistema operacional para *smartphones* que mais cresce no mundo. Sua arquitetura simples, flexível e ao mesmo tempo poderosa permite com que seja a base para muitos produtos, que se beneficiam de sua plataforma (LECHETA, 2012).

O *Kernel* utilizado pelo Android, como dito, é uma das séries do Linux 2.6 modificada para atender as necessidades especiais, como gerenciamento de energia, memoria e o ambiente de execução, na Figura 5 segue uma ilustração do *kernel* do Android.

Figura 5 – Arquitetura Android

Fonte: www.mobiltec.com.br/blog/index.php/desenvolvimento-mobile-nas-plataformas-android-e-ios, 2014

Camada de Aplicação: Onde se localizam todos os aplicativos nativos ao sistema operacional, como por exemplo, SMS, navegadores, agendas entre outros aplicativos

(FREIRE, 2012).

Camada de Biblioteca: É a camada que possui as bibliotecas C/C++ utilizadas pelo sistema, e também bibliotecas de multimídia, visualização de camadas 2D e 3D, funções para navegadores web, funções de aceleradores de *hardware* e funções de acesso a banco de dados SQLite (FREIRE, 2012).

Camada de Runtime: Nessa camada são cedidas as condições para que as aplicações baseadas na plataforma sejam executadas, onde se instancia a máquina virtual *Dalvik*, substituída na versão 4.4 do Android pela máquina virtual ART, sua diferença ao *Dalvik* é que ela ocorre antes da execução do aplicativo, assim aumentando sua velocidade de execução sobre a máquina *Dalvik* (ANDROIDPIT, 2014).

Camada de Kernel Linux: O núcleo do sistema operacional Android é derivado do *kernel* 2.6 do Linux, nesta camada se localiza o sistema operacional da plataforma, responsável por serviços de mais baixo nível da plataforma, como gerenciamento de memória, processos, threads, protocolos de rede, modelo de drives e a segurança dos arquivos (ANDROIDPIT, 2014).

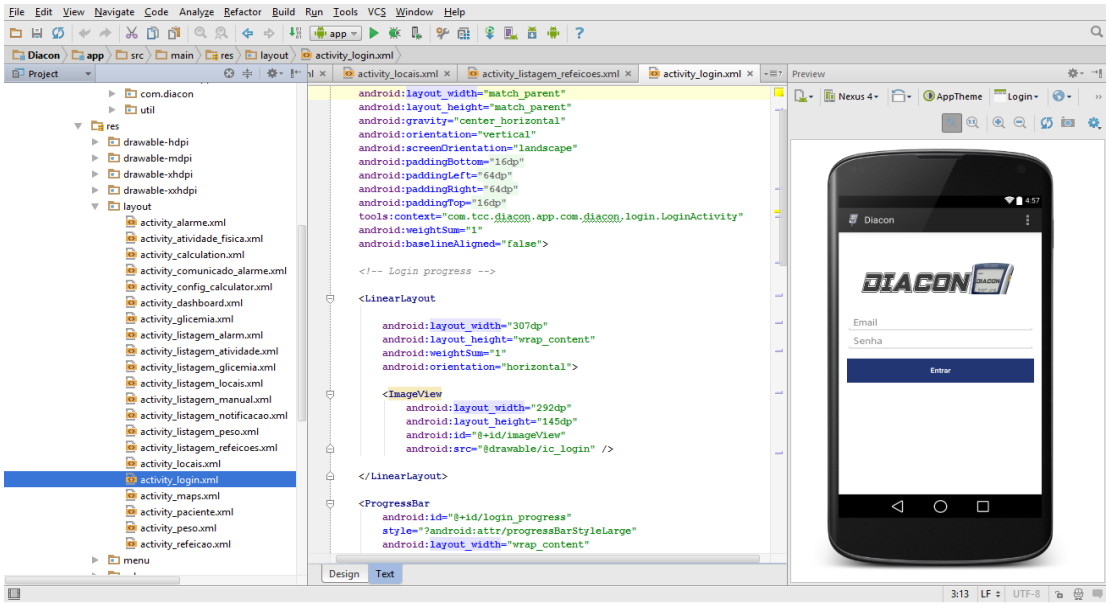
2.5.1 Armazenamento

O SQLite, ferramenta que permite armazenamento dos dados da aplicação em tabelas e manipulação desses dados através de comandos SQL. A diferença é que tudo isso pode ser feito sem que seja preciso acessar um SGBD. Sendo ele mais precisamente uma biblioteca desenvolvida em C, que pode ser integrada a programas escritos em diferentes linguagens com o intuito de possibilitar a manipulação de dados através de instruções SQL (SQLITE,2014).

2.5.2 IDE de desenvolvimento

O Android Studio, IDE de desenvolvimento que fornece as bibliotecas de API e as ferramentas necessárias para construir, desenvolver, testar e depurar aplicativos para Android, oferecendo um ambiente de implementação com maior variedade de recursos, e uma implementação mais rápida e flexível (ANDROID DEVELOPER, 2014). Na Figura 6 segue a ilustração do ambiente de desenvolvimento Android Studio.

Figura 6 - Ambiente de desenvolvimento Android Studio



Fonte: Elaborado pelo autor, 2014

2.6 API REST

O REST "*Representational State Transfer*", é um modelo de arquitetura que define um conjunto de regras para serviços web, possibilitando uma comunicação entre servidor e cliente (SAUDATE, 2014). Um dos meios para a implementação REST é o conceito de RESTful que se baseia no protocolo da aplicação HTTP, no padrão de nomenclatura URI e na linguagem XML/JSON (SAUDATE, 2014).

A utilização deste tipo de API facilita a comunicação entre diferentes plataformas sem a necessidade de mudança de uma aplicação para outra, pois o serviço se concentra em um servidor próprio.

O *Framework Jersey* visa simplificar o desenvolvimento de serviços REST e seus clientes em Java (JERSEY, 2014), sendo ele um projeto *open source*. A seguir, são apresentados alguns dos métodos e anotações mais utilizados pelo padrão REST.

- **GET** = Solicita uma representação do recurso especificado. Os pedidos usando este método só devem visualização dos dados;
- **POST** = Solicita que o servidor aceite a entidade fechada no pedido como um novo subordinado do recurso web identificado pelo URI.
- **PUT** = Atualiza um recurso em uma URI especifica, caso não exista, o mesmo poderá criá-la.
- **DELETE** = Exclui o recurso especificado.

2.7 Servidor de Mapas

O objetivo destes servidores é o fornecimento de uma interface HTTP simples para solicitação de imagens de mapa de uma base de dados geoespaciais distribuídos. Em um pedido WMS, obtemos como resposta uma ou mais imagens de mapas nos formatos PNG, JPEG e TIFF, que podem ser exibidas em aplicativos web e visualizados em *browsers* de navegação (GEOSERVER, 2014).

Entre suas extensões pode-se destacar o GeoJSON um formato que codifica para uma variedade de estruturas de dados geográficos. Um objeto GeoJSON pode representar uma geometria, uma ou várias características este formato suporta as seguintes tipos de geometria: *Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon,* and *GeometryCollection* (GeoJSON, 2014).

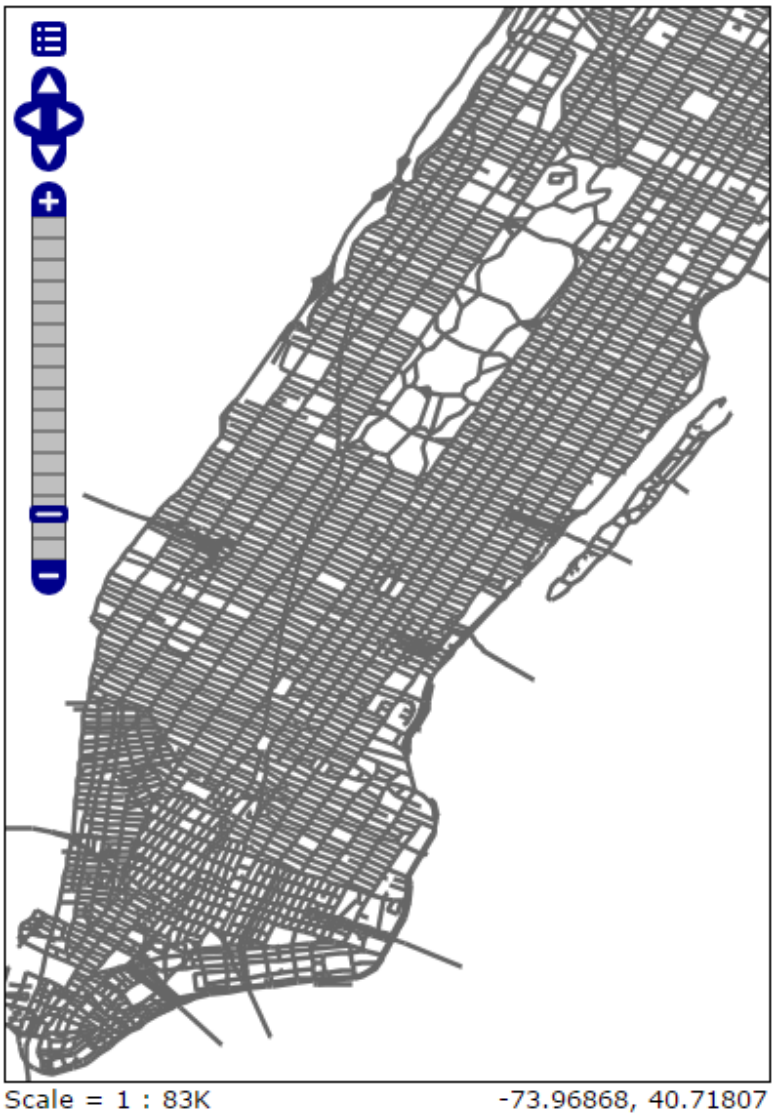
O GeoServer é um servidor de software baseado em Java que permite aos usuários visualizar e editar dados geoespaciais, construído sobre o *Geotools*, um servidor *open source* Java GIS *toolkit*, que exibe dados em qualquer um dos aplicativos de mapeamento populares como o Google Maps, Google Earth, Yahoo Maps e Microsoft Virtual Earth. Além disso, pode se conectar com arquiteturas de GIS tradicionais (GEOSERVER, 2014).

Usando padrões abertos definidos pelo *Open Geospatial Consortium* (OGC), *GeoServer* permite uma grande flexibilidade na criação de mapas e compartilhamento de dados. (GEOSERVER, 2014)

Implementando o padrão *Web Map Service* (WMS), esse servidor pode criar mapas em uma variedade de formatos de saída. O *OpenLayers*, uma biblioteca de mapeamento livre, está integrado no *GeoServer*, tornando a geração de mapa rápido e fácil. (GEOSERVER, 2014)

Este servidor funciona com o conceito de camadas, com suas coordenadas, latitude e longitude. É possível a geração de *layers*, sendo assim possível a sua sobreposição aos mapas, a Figura 7 ilustra o layer pré-configurado disponível no *Geoserver*.

Figura 7 – Layer Pré-configurado Geoserver



Fonte: Elaborado pelo Autor, 2014

2.8 Controle e Gerenciamento de Versão

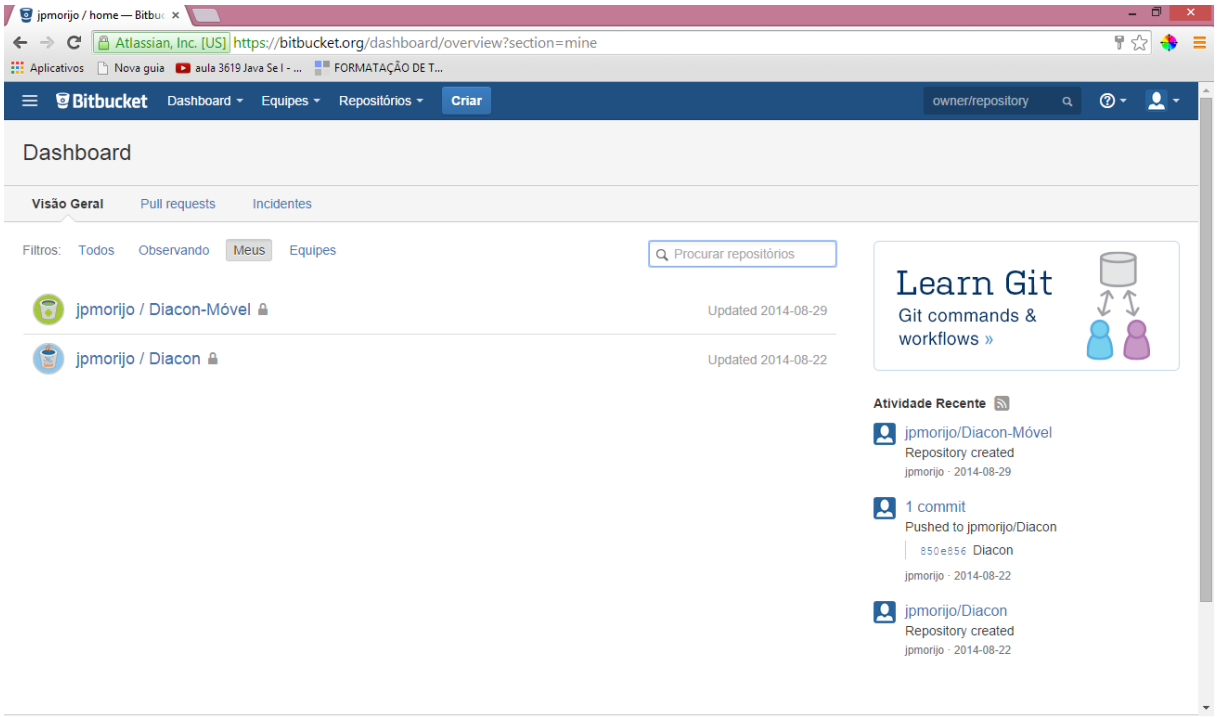
O controle de versão é um sistema distribuído e colaborativo, que registra as mudanças realizadas em um conjunto de arquivos ou em um arquivo especifico ao longo do tempo de forma a permitir a recuperação de versões específicas (GitHub, 2014).

Uma das funcionalidade mais importante é a utilização de ramos (*Branchs*) para o desenvolvimento de cada funcionalidade e ou versão a ser desenvolvida para um sistema (GitHub, 2014). Para o gerenciamento de um ramo são utilizados alguns comandos básicos *git* sendo alguns desses apresentados:

- *clone* – Clona repositórios existem na própria máquina ou em repositórios online.
- *branch* – Comando para visualização, criação ou remoção de ramos
- *add* – Adiciona um novo arquivo ao armazenamento do ramo
- *rm* – Remove um arquivo modificado do ramo
- *push* – Envia as modificações ao servidor de versão
- *pull* – Puxa novas atualizações adicionadas ao servidor de versão
- *checkout* – Desfazendo modificações de arquivos versionados
- *commit* – Encerra as modificações até aquele ponto salvando os itens modificados ou adicionados
- *merge* – Faz a integração de ramos distintos

Bitbucket, serviço de hospedagem de projetos controlados através do *Mercurial* (*Bitbucket*, 2014), um sistema de controle de versões distribuído, muito parecido com o *github*, mas com a vantagem de ser um hospedeiro gratuito, ou seja, com repositórios privados sem nenhum custo, conforme ilustrado na Figura 8.

Figura 8 - Sistema de Controle de Versão Bitbucket



Fonte: Portal Bitbucket repositório Diacon, 2014

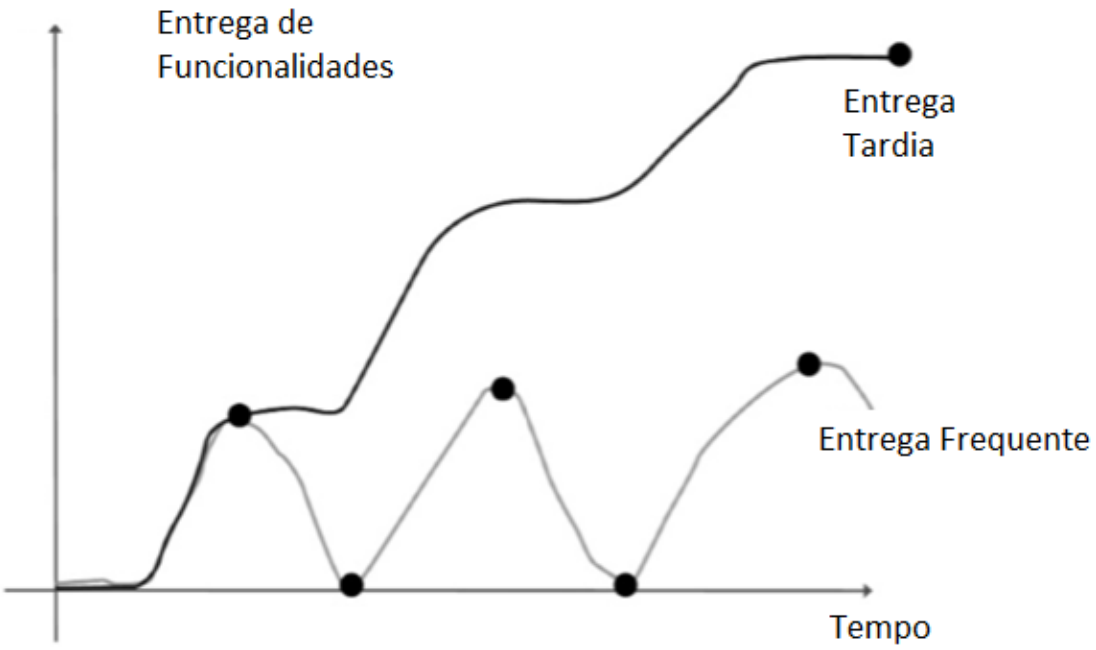
2.9 Entrega Continua

O processo de entregas continuas tem como foco diminuir os intervalos de entregas aprimorando a qualidade do software desenvolvido reduzindo os custos de refatorações.

> A criação de um software é baseada na exploração das possibilidades de implementação que ele fornece para o cliente, e o único momento em que ele pode ter certeza sobre um caminho escolhido é durante seu uso. Tudo isso requer feedback rápido e contínuo de nossas mudanças na base de código. (SILVEIRA, et al., 2014)

Quanto mais tarde o feedback for recebido a respeito de bugs ou funcionalidades, maior será o custo e o tempo para a refatoração do projeto, conforme ilustrado na Figura 9.

Figura 9 – Custo correção funcionalidades ao cliente



Fonte: Introdução à Arquitetura e Design de Software: Uma visão sobre a plataforma Java, 2014

O *Jenkins*, ferramenta de entrega continua *open source* que fornece um sistema de integração de fácil manuseio, facilitando para os desenvolvedores na integração das alterações no projeto, e para os usuários na obtenção de uma nova compilação. (Jenkins, 2014).

Essa ferramenta possibilita a automatização de testes nos aplicativos, com opção de envio de notificações aos desenvolvedores em caso de detecção de algum erro ou problema no processo de *deploy*.

2.10 Monitoramento

O monitoramento da infraestrutura e aplicações vem se tornando vital para gestão da tecnologia da informação. Este monitoramento permite obter as informações necessárias sobre a infraestrutura e aplicações com isso facilitando as tomadas de decisões para suas melhorias.

Atualmente existem várias ferramentas que monitoram e auxiliam essas tomadas de decisões, entre elas podemos destacar o *Zabbix* uma ferramenta de monitoramento de redes, servidores e serviços, ele possibilita a utilização de *templates*, para monitoramento de diversos parâmetros, sendo por estes disponibilizando relatórios bem detalhados e gráficos para melhor visualização (ZABBIX, 2014).

O mesmo possui mecanismos de notificações, envios de e-mails para alertas em relações aos eventos ocorridos ao servidores ou aplicações.

2.11 Considerações finais

Como apresentado nesta capítulo, a arquitetura Diacon foi construída com base em conceitos e tecnologias que visam a otimização e performance de seus componentes.

Foram apresentados os ciclos de implementação das tecnologias, iniciou-se com o aplicativo web (acesso as equipes médicas), aplicação móvel (utilizada pelos pacientes) e para concluir, sincronização entre os aplicativos.

O ciclo apresentado reúne as tecnologias e conceitos que vem crescendo ao longo do tempo em pesquisas e debates encontrados na internet, tornando-os cada vez mais automatizados, ou seja, independente da ação humana.

3 ESTRUTURA DA APLICAÇÃO DIACON

O objetivo desta arquitetura é mostrar a divisão de responsabilidade de cada componente, para obter um sistema flexível e mais estável aos seus usuários.

3.1 Funcionalidades

Para o mapeamento da aplicação foi utilizado o conceito de UML, uma linguagem para visualização, especificação, construção e documentação de artefatos de um software em desenvolvimento, é utilizada para enumerar as etapas mais importantes do software, facilitar as especificações de requisitos, e a padronização. (BOOCH, 2006).

Aqui está uma introdução simples e rápida de como é o funcionamento das aplicações, sendo todo o detalhamento do modelo de caso de uso incluso no arquivo anexo a este projeto.

3.1.1 Aplicação web

Médicos

- Funcionalidade responsável pela inserção de novos usuários (médicos ou agentes de saúde) que façam o acompanhamento dos pacientes.

Paciente

- Funcionalidade responsável em mostrar todos os dados coletados dos pacientes através do aplicativo móvel.

Notificações

- Funcionalidade responsável em notificar o paciente sobre o seu tratamento, sendo elas, consultas a serem realizadas ou qualquer novidade que venha a ser importante para seu tratamento.

Locais de Saúde

- Funcionalidade responsável em cadastrar pontos de saúde ou locais que sejam importantes aos pacientes do aplicativo móvel.

3.1.2 Aplicação móvel

Nível glicêmico

- Funcionalidade responsável em cadastrar índices glicêmicos apresentados pelo paciente.

Atividades Físicas

- Funcionalidade responsável em cadastrar novas atividades físicas realizadas pelo paciente, sendo cadastrados o dia, o tipo de atividade e a duração.

Peso

- Funcionalidade responsável por acompanhar o peso do paciente

Refeições

- Funcionalidade responsável em acompanhar as refeições diárias do paciente, sendo nela inseridos o tipo de refeição realizada, junto com a quantidade de carboidratos ingeridos.

Alarmes

- Funcionalidade responsável pela notificação dos horários das refeições e horário das medicações, sendo os mesmos configurados pelo paciente.

Pontos de Saúde

- Funcionalidade responsável por cadastrar novos pontos de saúdes, lugares importantes para o paciente.

3.2 Arquitetura da aplicação

Para a implementação desta aplicação, foi utilizado uma arquitetura com os princípios de SOA (*Service Oriented Architecture*) ou *Microservice*, um estilo de arquitetura de software cujo os princípios pregam que as funcionalidades implementadas pelas aplicações devem ser disponibilizadas na forma de serviços (SAUDATE, 2012).

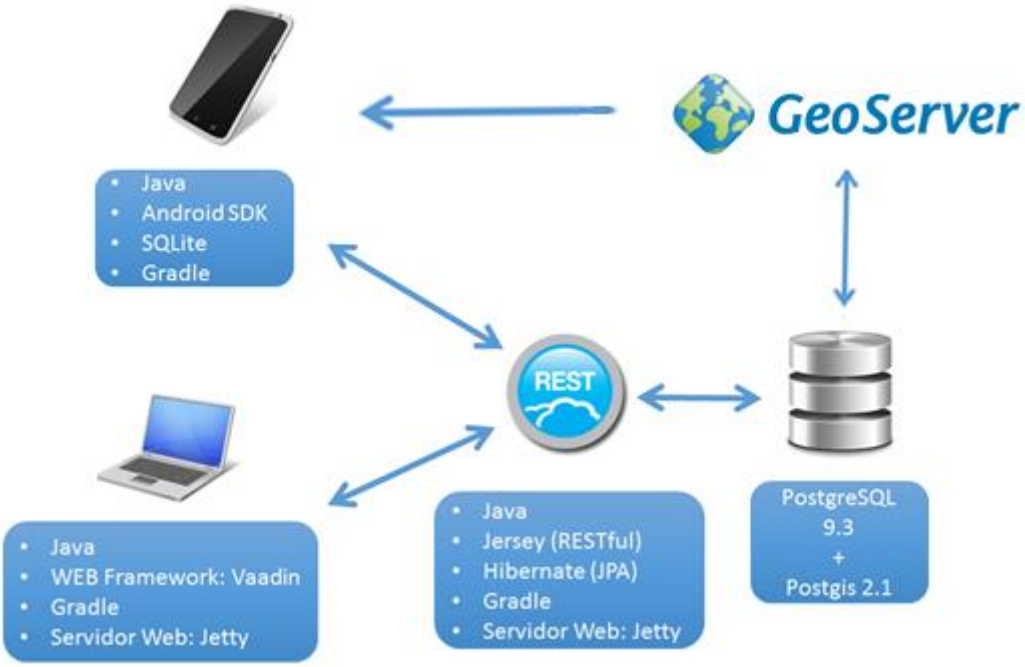
Com este tipo de arquitetura, cada componente fica independente dos demais, tornando assim, uma aplicação mais confiável aos usuários, pois independente do funcionamento de cada um dos componentes, os demais continuam sendo executados (SAUDATE, 2012).

Entre as vantagens deste tipo de arquitetura pode-se destacar que cada componente não precisa ficar preso a nenhuma linguagem, tecnologia ou ferramentas utilizadas pelos demais componentes, pois, como destacado, cada componente é independente dos demais,

tornando assim, uma aplicação bem flexível e de fácil manutenção e evolução de seus componentes (SAUDATE, 2012).

Como pode ser observado na Figura 10, cada componente é composto com suas respectivas tecnologias.

Figura 10 - Arquitetura Diacon e seus componentes



Fonte: Elaborado pelo autor, 2014

3.2.1 Elementos da arquitetura

O ambiente é formado por um conjunto de cinco componentes, cada um responsável pelo gerenciamento e execução de uma aplicação.

3.3 Aplicação Web

A aplicação web tem como foco o acompanhamento e o envio de notificações importantes para o tratamento aos usuários do aplicativo móvel, tendo como base todos os dados registrados na aplicação móvel.

3.3.1 Arquitetura web

A arquitetura da aplicação web se divide em dois projetos *Backend,* onde se localizam todas as regras de negócio e conexões ao banco e o *Frontend,* onde se localizam as *Views* e *Controllers* da aplicação.

Pacote Backend regras de negócios e conexões ao banco de dados.

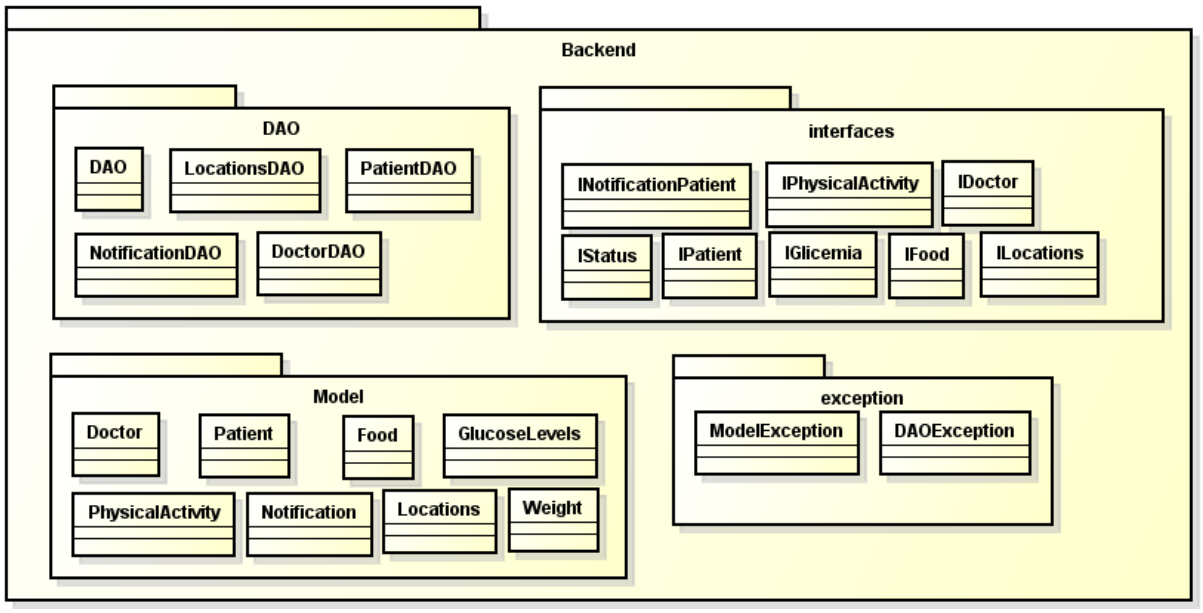
Pacote DAO: Pacote responsável pelas operações básicas ao sistema (CRUD).

Pacote Interface: Pacote responsável pela definição dos métodos, porém, sem sua implementação, expondo somente o que o objeto deve fazer.

Pacote Model: Pacote responsável por toda a regra de negócio da aplicação e mapeamento do framework *Hibernate.* Neste pacote estão contidos os modelos da aplicação.

Pacote Exception: Pacote responsável por tratar as exceções apresentadas à partir do Model e DAO da aplicação, na Figura 11 segue uma ilustração da organização dos pacotes do projeto *Backend*.

Figura 11 - Diagrama de Pacotes Aplicação Web (Backend)



Fonte: Elaborado pelo Autor, 2014

Pacote Frontend *Views* e *Controllers* da aplicação.

Pacote Dashboard: Pacote responsável pela sessão do usuário, bem como toda a definição da tela inicial do aplicativo.

Pacote Doctor: Pacote responsável por representar a *View* e *Controller* e o *container*

de dados da funcionalidade *Doctor*, o usuário principal da aplicação web.

Pacote Patient: Pacote responsável por representar a *View* e *Controller* e o *container* de dados da funcionalidade *Patient*.

Pacote Exception: Pacote responsável por tratar as exceções apresentadas da aplicação.

Pacote Login: Pacote responsável pela funcionalidade de *login* ao sistema.

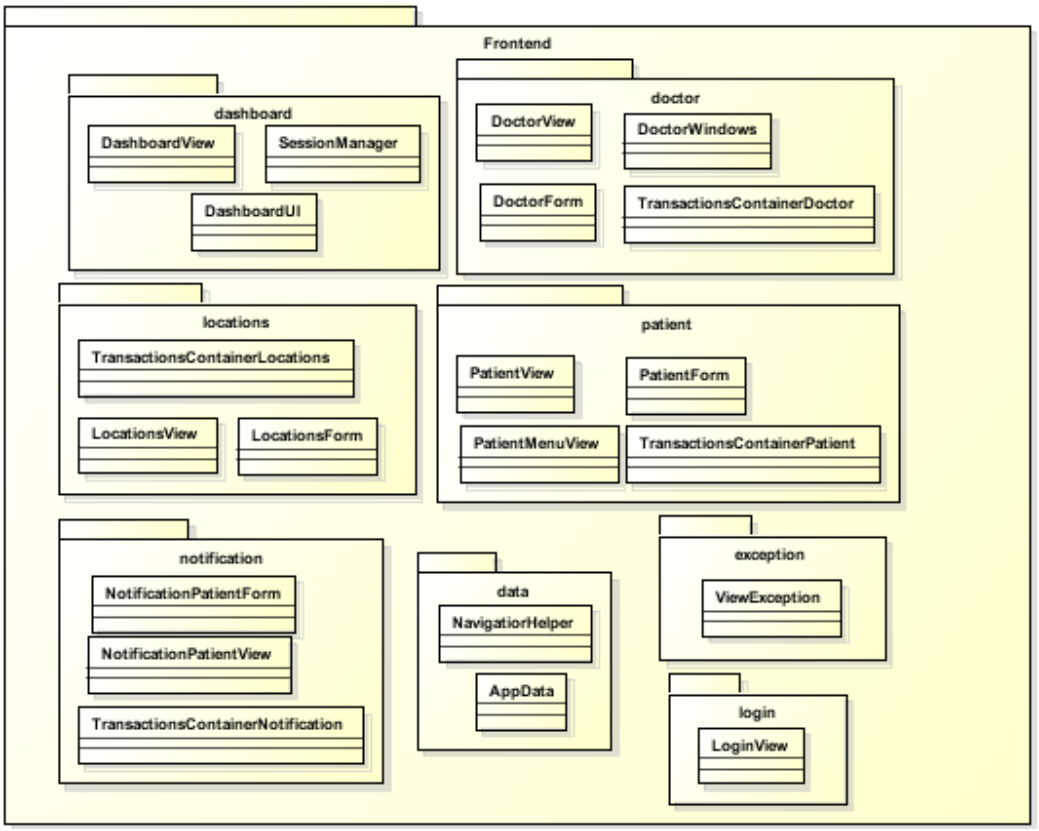
Pacote Data: Pacote responsável pelo preenchimento dos *combobox* do sistema, bem como a ordem dos itens ao menu principal da aplicação.

Pacote Locations: Pacote responsável por representar a *View* e *Controller* e o *container* de dados da funcionalidade *Locations*.

Pacote Notifications: Pacote responsável por representar a *View* e *Controller* e o *container* de dados da funcionalidade *Notifications*.

Pacote History: Pacote responsável por representar a *View* e *container* de dados do histórico do paciente.

Figura 12- Diagrama de Pacotes Aplicação Web (Frontend)



Fonte: Elaborado pelo Autor, 2014

3.4 Aplicação Móvel

A aplicação móvel funciona como um diário ao usuário, sendo nele registrado todas as informações essenciais para seu tratamento e assim sincronizadas ao seu médico afim de obter melhores diagnósticos.

3.4.1 Arquitetura móvel

Para esta criação, a estrutura segue com quatro pacotes separados por responsabilidades.

Pacote Activity: Pacote com objetivo orientar as atividades a serem executadas, ele é o responsável por determinar qual *View* será apresentada ao usuário.

Pacote Model: Pacote responsável por todo o mapeamento das regras de negócio da aplicação.

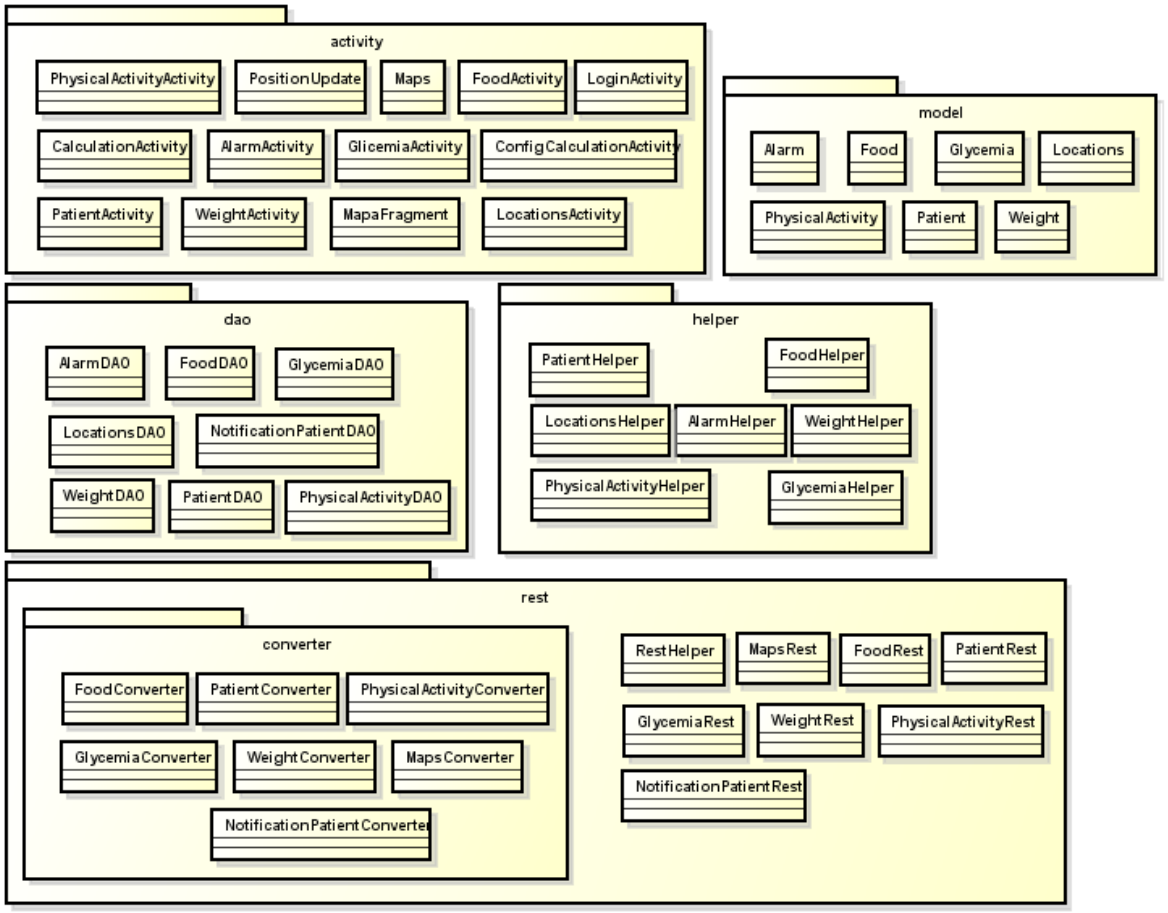
Pacote DAO: Pacote responsável pelas operações básicas ao sistema (CRUD).

Pacote Helper: Pacote responsável em mapear as informações contidas nas *View* para as classes modelos.

Pacote Rest: Pacote responsável por toda a conversão dos dados em JSON e dar suporte aos recursos básicos do HTTP (GET, POST, PUT, DELETE).

Conforme ilustrado na Figura 13 segue a divisão dos pacotes e suas responsabilidades.

Figura 13 – Diagrama de Pacotes Aplicação móvel



Fonte: Elaborado pelo Autor, 2014

3.5 Integração e sincronização

A API REST tem como funcionalidade a integração e sincronização entre os componentes, seus dados são representados no formato JSON, uma forma leve de troca de dados, conforme ilustrado na Figura 14.

Figura 14 - Exemplo formato JSON

```
{"status":"1",
 "nomePaciente":"joao paulo",
 "tipoDiabetes":"tipo 1",
 "medicacao":"humalog",
 "dataNascimento":"11/11/1111",
 "email":"jpmorijo@gmail.com",
 "telefone":"14999998888"}
```

Fonte: Elaborado pelo Autor, 2014

As Figura 15 e 16 ilustram um exemplo de inserção com o método POST e conversão para o formato JSON, sendo necessário estruturar o código a ser enviado. No aplicativo móvel foi utilizada a biblioteca HTTP, na qual passamos o método a ser utilizado, o formato do arquivo a ser enviado, o endereço e o principal, as informações a serem transmitidas.

Figura 15 - Metodo POST (Paciente)

```
public String post(String dadosJSON) {
    try {
        HttpClient client = new DefaultHttpClient();
        HttpPost post = new HttpPost(RestHelper.URLPACIENTE);

        post.setEntity(new StringEntity(dadosJSON));
        post.setHeader("Content-type", "application/json");

        client.execute(post);

        return dadosJSON;
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

Fonte: Elaborado pelo Autor, 2014

Figura 16 - Conversão JSON Class PacienteConverter.java

```
public class PatientConverter {

    public String toJSON(List<Patient> patients){
        try {
            JSONStringer js = new JSONStringer();
            for(Patient patient : patients){
                js.object();
                js.key("status").value("1");
                js.key("nomePaciente").value(patient.getNomePaciente());
                js.key("tipoDiabetes").value(patient.getTipoDiabetes());
                js.key("medicacao").value(patient.getMedicacao());
                js.key("dataNascimento").value(patient.getDatanascimento());
                js.key("email").value(patient.getEmail());
                js.key("telefone").value(patient.getTelefone());
                js.endObject();
            }
            return js.toString();
        } catch (JSONException e) {
            e.printStackTrace();
            return "";
        }
    }
}
```

Fonte: Elaborado pelo Autor, 2014

Na aplicação web foi utilizado o *framework Jersey*, responsável por todo esse tratamento e conversão dos dados, facilitando e agilizando essa integração, conforme ilustrado na Figura 17.

Figura 17 – Mapeamento REST classe PatientResource.java (GET, POST)

```
/**
 * Busca paciente Por ID
 * @param id
 * @throws ModelException
 */
@SuppressWarnings("static-access")
@Path("{id}")
@GET
@Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })
public Response getPatient(@PathParam("id") Long id) throws ModelException {
    Patient patient = null;
    try{
        return Response.ok(patient.pesquisa(id)).build();
    }finally{
        patient = null;
    }
}

/**
 * Inserir Novo paciente
 * @param patient
 * @return
 */
@POST
@Consumes({ MediaType.APPLICATION_JSON})
@Produces({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })
public Response adicionarPatient(Patient patient){
    try{
        patient.salvar();
    }catch(Exception e){
    }
    return  Response.ok(URI.create(String.valueOf(200))).entity(patient).build();
}
```

Fonte: Elabora pelo Autor, 2014

Na Tabela 3 estão apresentados os recursos HTTP utilizados na implementação da API REST do sistema.

Tabela 3 – Metodos utilizados para sincronização

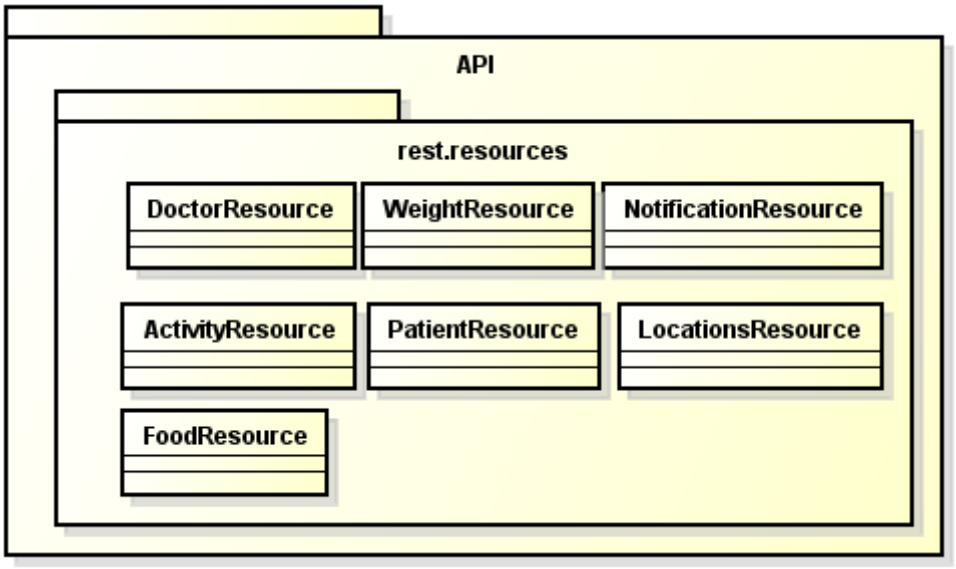
URL	Recurso HTTP	Descrição
/paciente/	POST	Insere um novo paciente
/paciente/{idpaciente}	GET	Lista os pacientes cadastrados
/paciente/{idpaciente}	DELETE	Deleta o paciente selecionado
/refeicao/{idpaciente}	POST	Insere nova refeição paciente
/atividade/{idpaciente}	POST	Insere nova atividade realizada
/peso/{idpaciente}	POST	Insere nova pesagem paciente
/notificacao/{idpaciente}	GET	Lista as notificações paciente
/glicemia/{idpaciente}	POST	Insere um novo nível glicêmico

Fonte: Elabora pelo Autor, 2014

3.5.1 Arquitetura API REST

Pacote API: Pacote responsável por toda a integração entre as aplicações. Em cada classe representada estão as operações básicas (CRUD) para cada item a ser sincronizado entre as aplicações, conforme ilustrado na Figura 18 segue o pacote API.

Figura 18 – API REST Diacon



Fonte: Elaborado pelo Autor, 2014

3.6 Banco de dados

Toda a aplicação web foi construída com auxílio do *framework Hibernate*, sendo ele o responsável por todo o mapeamento ao banco de dados. Com anotações especificas do *framework* foi possível constituir todo o nosso banco para a aplicação, como ilustrado na Figura 19.

Figura 19 – Modelo Doctor.java anotações hibernate

```
@XmlRootElement
@XmlAccessorType( XmlAccessType.FIELD )
@Entity
@Table(name="medico")
@NamedQueries({
        @NamedQuery(name = "medico.autentica", query = "select new Doctor (id, nome, cpf, status, email) "
                + "from Doctor where nome = :nome and senha = :senha"),
        @NamedQuery(name = "medico.lista", query = "select new Doctor (id, nome, cpf, status, email) "
                + "from Doctor"),
        @NamedQuery(name = "medico.altera",query = "update Doctor set nome = :nome, cpf =:cpf, status = :status, email = :email"
                + " where id = :id"),
        @NamedQuery(name = "medico.alterapwd", query = "update Doctor set senha = :senha"
                + " where id = :id"),
        @NamedQuery(name = "medico.pesquisa", query = "select new Doctor (id, nome, cpf, status, email) "
                + "from Doctor where id = :id")
})
public class Doctor implements IDoctor {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "nome",length=100)
    private String nome;

    @Column(name = "status")
    private Integer status;

    @Column(name = "senha",length=10)
    private String senha;

    @Column(name = "email",length=100)
    private String email;
```

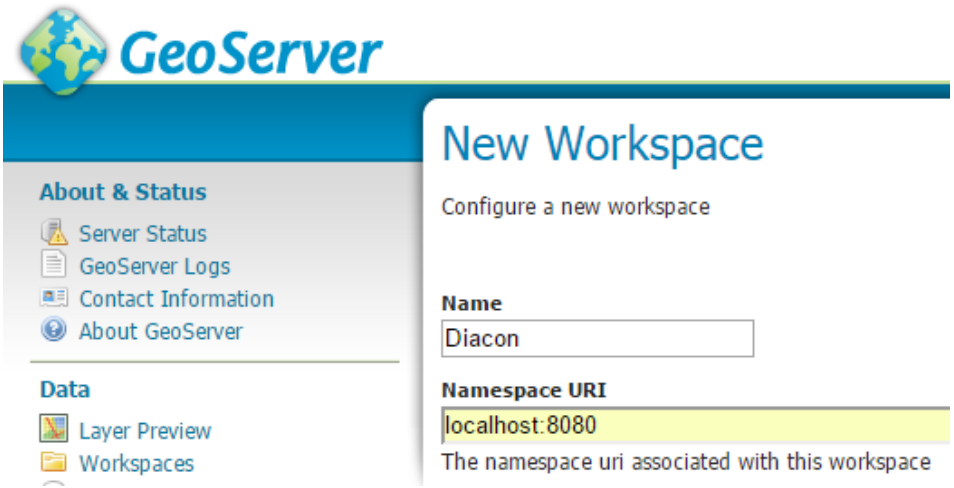
Fonte: Elabora pelo Autor, 2014

3.7 Servidor de mapa.

Geoserver foi utilizado como o serviço de plotagem de pontos do mapa, além do próprio *google maps,* responsável pela plotagem dos locais cadastrados diretamente ao aplicativo.

Para sua implementação, foi necessário a criação de um *workspace* do projeto, onde estará toda a configuração, conforme ilustração da Figura 20.

Figura 20 - Criação Workspace Diacon



Fonte: Elaborado pelo Autor, 2014

Após a criação do workspace, é necessário a criação da *store* para o *shapefile*, ou seja, uma entidade para representar o *shapefile*.

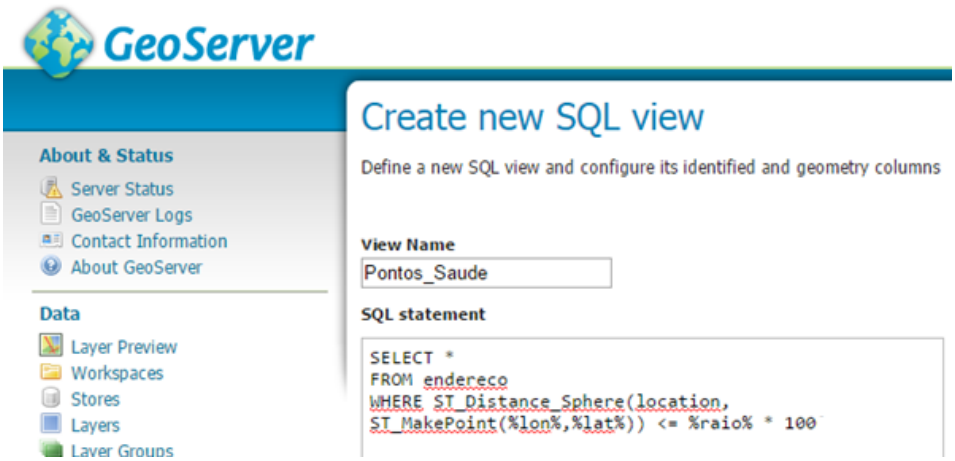
Para sua criação, é necessário selecionar a fonte de dados selecionando, que no caso do projeto Diacon é o PostGIS. Após, é necessário a transmissão dos dados para essa conexão, conforme apresentado na Figura 21.

Figura 21 - Configuração para fonte de dados (PostGIS)

Fonte: Elaborado pelo Autor, 2014

Finalizado esta etapa, é iniciada a construção da camada para plotagem dos pontos, selecionando o item *Layers* no menu de opções e adicionando que será a criação de um novo comando SQL, conforme ilustrado na Figura 22.

Figura 22 - Comando SQL endereços Cadastrados



Fonte: Elaborado pelo Autor, 2014

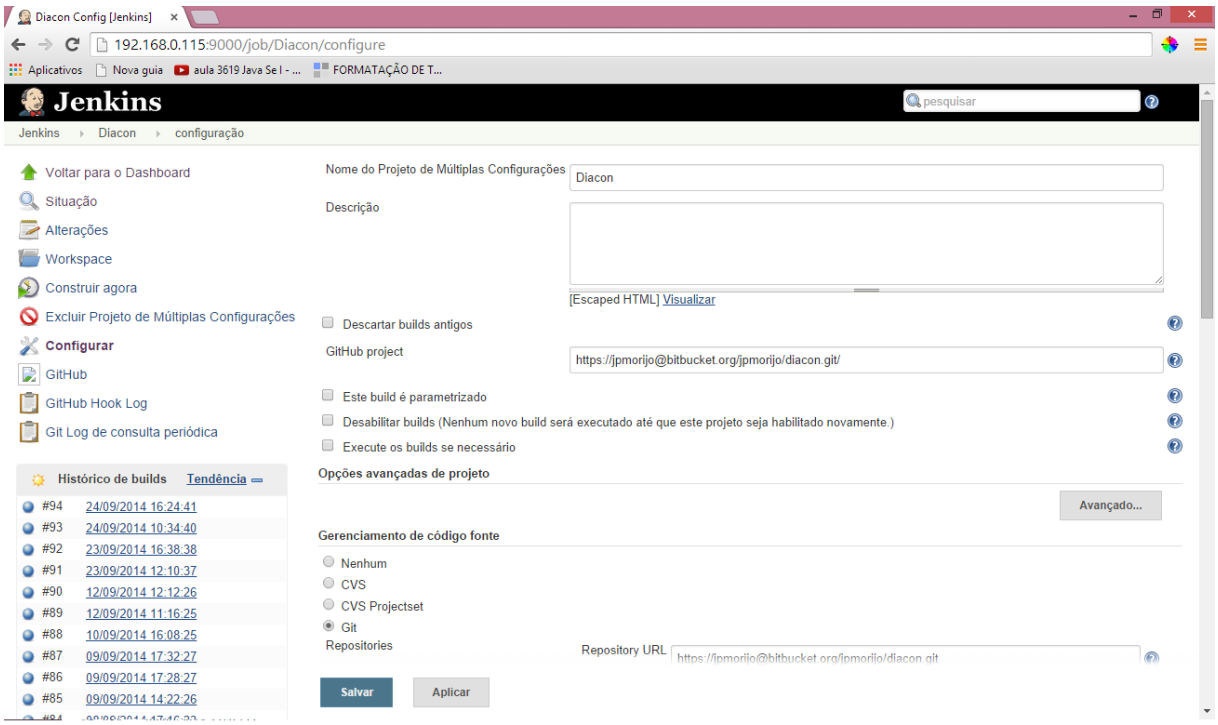
Na Figura 22 está sendo apresentada uma seleção dos pontos cadastrado dentro de um raio de 100 km de distância a partir da posição atual, o item *location* é uma máscara utilizada pelo PostGIS para o armazenamento da latitude e longitude.

Finalizando essa criação, é gerado um link pelo qual os parâmetros serão enviados através do aplicativo móvel, acessando os pontos plotados pelo *geoserver*.

3.8 Infraestrutura

Para o serviço de integração contínua, foi utilizada a ferramenta *Jenkins*, responsável por realizar as atualizações contínuas da aplicação web e API e notificar erros no processo de *deploy*. Essa ferramenta é configurada para periodicamente fazer consultas ao repositório do *Bitbucket,* para a realização deste processo será ilustrado nas Figuras 23 e Figura 24 algumas das configurações essenciais para o gerenciamento deste ciclo.

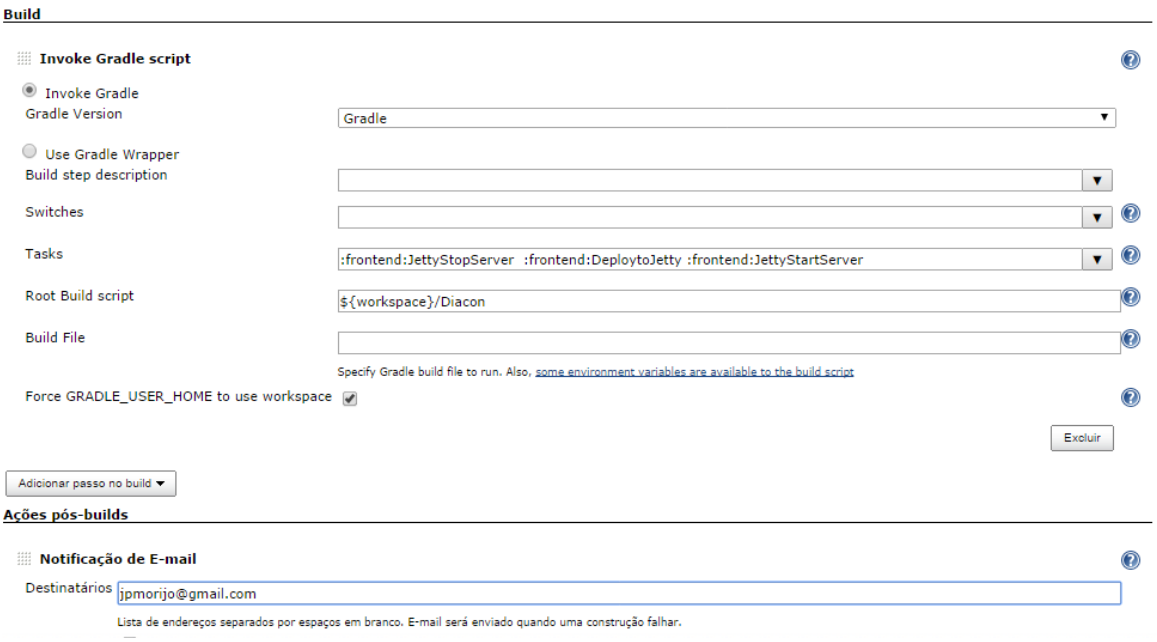
Figura 23 - Jenkins Deploy Continuo



Fonte: Elabora pelo Autor, 2014

Suas tarefas “Tasks” são invocadas pelo *gradle,* como é possível observar na Figura 24.

Figura 24 - Tarefas Executadas Gradle



Fonte: Elabora pelo autor, 2014

As tarefas configuradas no *build.gradle* da aplicação web invocadas para execução pelo Jenkins, executam as seguintes instruções:

- :frontend:JettyStopServer = Finaliza o serviço *Jetty* em execução, assim limpando o *buffer* da aplicação
- :frontend:DeploytoJetty = Copia a extensão .war para dentro do diretorio *jetty* para sua execução.
- :frontend:JettyStartServer = Inicia a aplicação web.

Além da configuração realizada na ferramenta, é necessário conceder permissão ao usuário *Jenkins* ao sistema operacional para ele poder executar as tarefas citadas anteriormente.

3.9 Gerenciamento de dependência e Deploy

Para o serviço de *deploy* e gerenciamento das dependências, foi utilizado a ferramenta *gradle*, onde suas atividades são configuradas em arquivos presentes em cada projeto:

- *build.gradle*: Arquivo implementado para aplicar os *plugins* utilizados no projeto. Os repositórios onde se localizam as dependências, dependências a serem baixadas, e as tarefas a serem executadas pelo projeto, essas tarefas são as executadas pelo *jenkins* para sua entrega contínua.

- *config.gradle*: Arquivo implementado com a pasta raiz do projeto, url de acesso ao repositório *bitbucket* e os projetos existentes na implementação.
- *setting.gradle*: Arquivo implementado para incluir todos os projetos da aplicação no caso foi o "*Backend*" e "*Frontend*".

3.10 Considerações finais

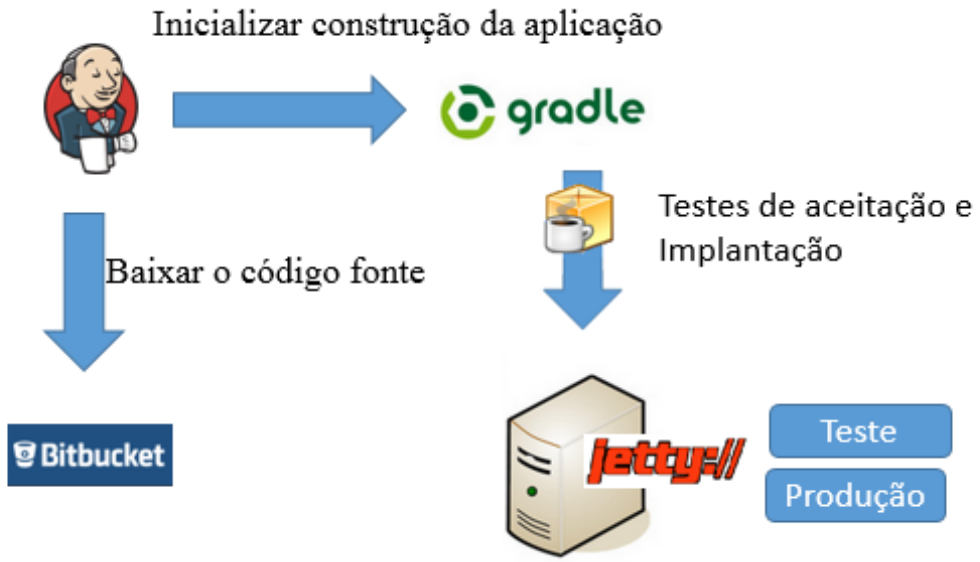
O objetivo deste capítulo foi a apresentação das ferramentas e técnicas utilizadas para a realização e implementação deste projeto, tecnologias aprendidas com a finalidade de ajudar e melhorar os resultados obtidos pelas aplicações.

Como descrito neste capitulo, foram apresentadas tecnologias distintas, sendo cada uma de responsabilidade diferentes, evitando a sobrecarga das partes, resultando em um sistema mais flexível, de manutenção rápida e atualização independente de seus componentes.

O foco para a utilização destas tecnologias apresentadas foram seus resultados apresentados em fóruns de debates, pesquisas e comparações com demais tecnologias de mesma finalidade, tendo em todas as buscas realizadas as tecnologias escolhidas pontos fortes para este tipo de desenvolvimento.

O uso da ferramenta *Jenkins* permitiu a automatização do ambiente de *deploy*, integrando o repositório *bitbucket, Gradle*, e o *Jetty*. O ambiente de automatização de *deploy* e integração contínua é ilustrado na Figura 25.

Figura 25 - Ambiente de Integração Continua



Fonte: Elaborado pelo Autor, 2014

4 RESULTADOS OBTIDOS

Os resultados deste projeto foram, o desenvolvimento de uma arquitetura composta por um conjunto de ferramentas e tecnologias que visam auxiliar e monitorar uma das doença que cresce atualmente o diabetes, para isso se torna necessário uma arquitetura escalável que possa ser incrementada e atualizada conforme a demanda de novos pacientes, confiável aos seus usuários e estável para atender suas necessidades.

Como apresentado no capitulo 3 item 3.2, esta arquitetura se baseia nos princípios de SOA ou *Microservices*, ou seja seu desenvolvimento foi constituído por um conjunto de pequenos serviços/aplicações que visam o auxílio e monitoramento para pacientes e equipes médicas a fim da obtenção de melhores diagnósticos e tratamentos.

Com este estilo de arquitetura apresentado, se torna possível atualizações em componentes distintos, ou seja a alteração de um componente não influencia diretamente no funcionamento dos demais, sendo assim possível sempre incrementar e atualizar o sistema para que possa atender as necessidades que venham a aparecer.

Sendo está arquitetura desenvolvida foram levantados algumas características importantes durante e após sua implementação.

Pontos Fortes

- Independência dos componentes – Um componente não necessita do funcionamento dos demais para sua execução.
- Descentralização Tecnológica – Toda a arquitetura Diacon foi desenvolvida na linguagem Java, podendo um componente ser substituído por outro com sua linguagem de implementação diferente da utilizada atualmente, sem que haja a necessidade de alterações nos demais componentes da arquitetura.
- Reuso de Código – Com suas reponsabilidades bem definidas e sua arquitetura bem projetada, se torna possível a reutilização de códigos já implementados ao projeto, não havendo assim necessidade de implementação para o mesmo novamente.
- Diversidade Tecnológica – Uma grande variedade de ferramentas que visam a simplificação na implementação e com isso um grande aumento na produtividade.
- Elaboração e Implementação – Por ser uma arquitetura descentralizada, se torna preciso um melhor planejamento do projeto.

Pontos Fracos

- Complexidade de ambiente. As principais IDEs não são para esse tipo de arquitetura.
- Necessidade de mais comunicação entre processos. Os serviços precisam trocas mensagens HTTP para realizar sincronização.
- Monitoramento– Por ter vários componentes independentes o monitoramento e gerência de redes torna uma fator fundamental. Ferramentas como New Relic, XRebel, Zabbix são vitais para o processo de gestão e desenvolvimento.

Em relação ao ambiente para a implementação desta arquitetura foi utilizado o sistema operacional Linux, sendo para implementação utilizado a versão Ubuntu 14.10, um sistema com características para usuários, com interfaces detalhadas deixando um visual agradável, e para o ambiente de produção foi escolhido o sistema operacional CentOs 6.5, com sua configuração para servidores web

A escolha desses sistemas operacionais se basearam em pesquisas e o maior conhecimento sobre os mesmos, tendo o sistema operacional CentOs apresentado um melhor resultados em relação ao Ubuntu, em relação as aplicações, conseguindo se manter em constante funcionamento com um excelente gerenciamento dos processos em execução e no gerenciamento do consumo da memória.

Nos itens 4.1 e 4.2 deste capitulo estão apresentados os resultados em ilustrações das aplicações móvel e web desenvolvida nesta arquitetura.

4.1 Aplicativo Móvel

Ao selecionar o aplicativo, o usuário é direcionado para a tela de *login*, caso o mesmo não tenha cadastro, poderá realiza-lo nesta tela inicial, no menu de opções, selecionando a opção “Cadastrar Novo Usuário”, conforme apresentado na Figura 26.

Figura 26 - Tela Inicial Diacon (Login)

Fonte: Elaborado pelo Autor, 2014

Após realizar a autenticação, o usuário é encaminhado para a tela inicial, onde assim poderá selecionar qual opção ele deseja.

Figura 27 - Tela Principal

Fonte: Elaborado pelo Autor, 2014

No menu de opções do sistema, é possível visualizar o perfil do paciente cadastrado e sincronizar os dados com o aplicativo web. Na tela principal podemos selecionar as funcionalidades existentes na aplicação, listadas a seguir:

- Cadastrar índices Glicêmicos
- Calculadora Glicêmica;
- Controle de refeições;
- Controle de Atividades Físicas;
- Alarme – Despertador para refeições e medicações;
- Pontos de Saúde;
- Notificações realizadas para o paciente;
- Manual de Carboidrato – Manual já incluso no sistema;
- Controle de Peso.

Como ilustrado na Figura 28, segue algumas funcionalidades da aplicação móvel.

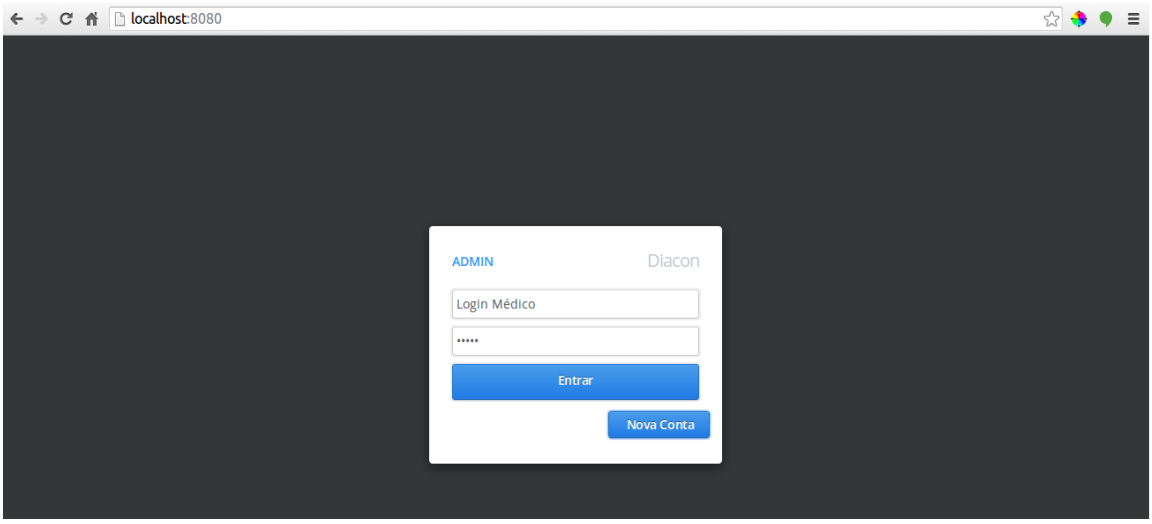
Figura 28 –Funcionalidades Diacon

Fonte: Elabora pelo Autor, 2014

4.2 Aplicativo Web

Ao selecionar o aplicativo, o usuário é direcionado para a tela de *login*, caso o mesmo não tenha cadastro, poderá realiza-lo nesta tela inicial, na opção “Nova Conta”, como ilustrado na Figura 29.

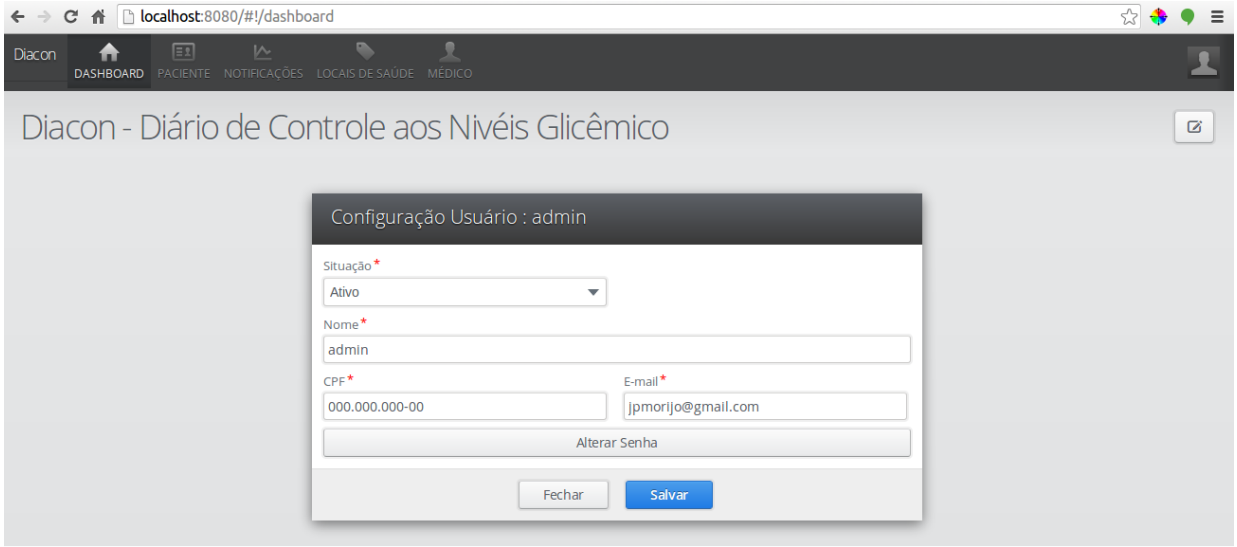
Figura 29 – Tela de Login Aplicativo Web



Fonte: Elaborado pelo Autor, 2014

Ao efetuar o *login,* o usuário é direcionado para seu *dashboard*, sua tela inicial, sendo no canto direito da tela existe um ícone de usuário, utilizado para fazer alterações no seu perfil do usuário “logado”, logo abaixo, há um outro ícone, onde é possível alterar o nome do seu *dashboard,* que por padrão recebe o nome da aplicação, como ilustrado na Figura 30.

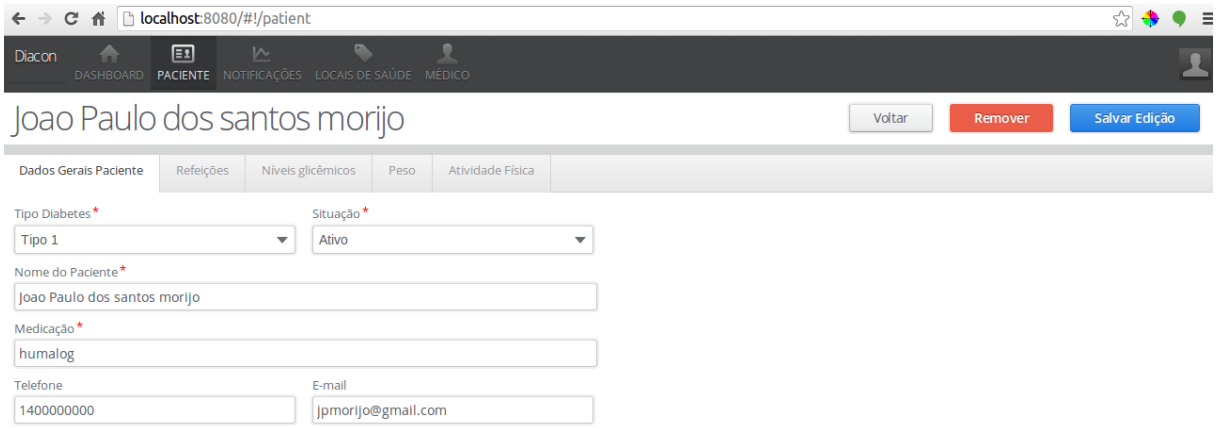
Figura 30 – Tela inicial e configuração usuário



Fonte: Elaborado pelo Autor, 2014

Ao se *logar* ao sistema se torna disponível o acesso a todas as informações sincronizadas entre as aplicações, sendo estas disponível na aba paciente conforme ilustrado na Figura 31.

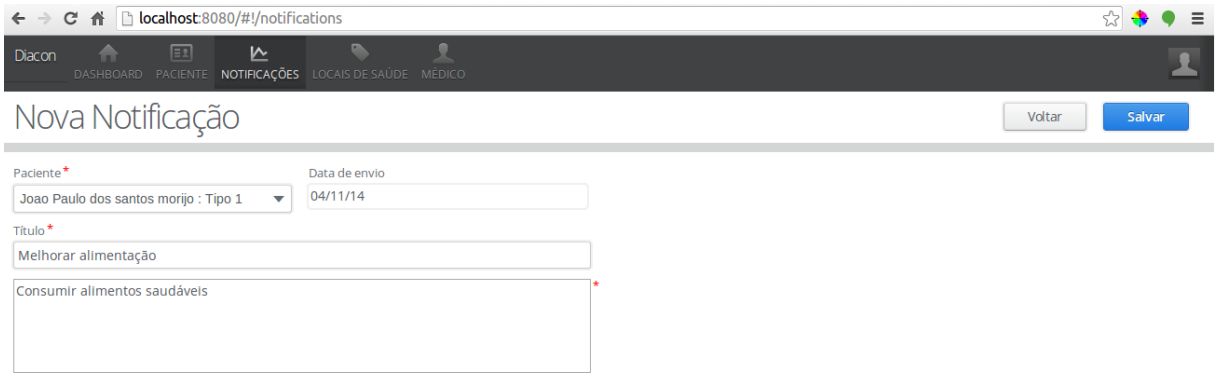
Figura 31 – Informações recebidas paciente



Fonte: Elaborado pelo Autor, 2014

Na tela notificações, estão contidas as informações enviadas ao paciente com o intuito de melhorar o seu tratamento, conforme ilustrado na Figura 32.

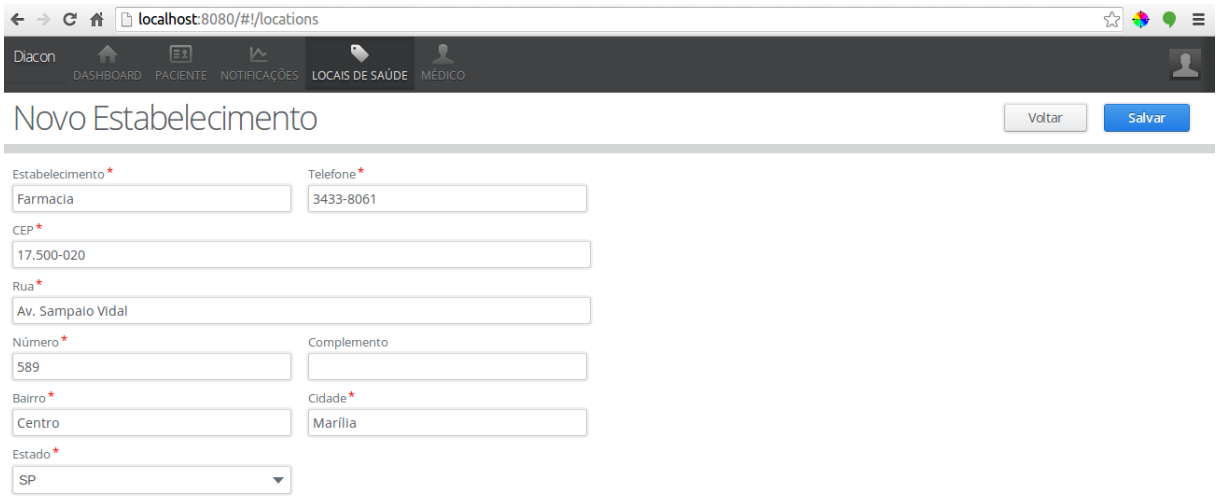
Figura 32 – Notificação ao usuário móvel



Fonte: Elaborado pelo Autor, 2014

Na tela Locais de Saúde, estão presentes os estabelecimentos registrados no sistema, contemplando a opção de inserir novos registros para auxiliar na localização de locais importantes para os usuários do aplicativo móvel, conforme ilustrado na Figura 33.

Figura 33 – Cadastro de novos estabelecimentos



Fonte: Elaborado pelo Autor, 2014

Essas apresentadas são algumas das funcionalidades encontradas na aplicação web, sendo esta de utilização para equipes medicas.

4.3 Considerações finais

O objetivo deste capítulo foi a apresentação dos resultados obtidos com o desenvolvimento e implementação de toda a arquitetura Diacon.

Como apresentado foram ilustrado as duas aplicações em que paciente e médico poderão ter um melhor contato afim de obtenção de melhorias em seus tratamentos e com isso a realização de diagnósticos mais preciso com base aos dados disponibilizados pelos pacientes.

Além disso, neste projeto foi apresentado uma proposta de arquitetura que tem como principal objetivo, reduzir o acoplamento, melhorar a qualidade do código e do processo e facilitar a manutenção do aplicativo. Sendo os principais benefícios:

- Permitir pensar com mais cuidado na arquitetura e nas tecnologias. Onde a implantação de funcionalidades que abrangem vários serviços deverá, de modo cauteloso, uma coordenação entre as partes envolvidas.
- Facilidade na detecção e correção de erros, por se tratar de uma arquitetura desacoplada, se torna mais fácil a detecção e correção para possíveis erros
- Flexibilidade e facilidade em alterações e modificações em componentes implementados nesta arquitetura.
- Com suas reponsabilidades bem definidas e sua arquitetura bem projetada, se torna possível a reutilização de códigos já implementados ao projeto.

CONCLUSÃO

O objetivo deste trabalho foi o desenvolvimento de uma arquitetura que possa atender as necessidades dos pacientes e equipes médicas, sendo esta escalável pelo constante crescimento da doença.

O desenvolvimento foi composto por uma aplicação móvel para dispositivos *smartphones* com o sistema operacional Android, uma aplicação web e um serviço de integração em RESTful.

Em relação ao serviço de integração (REST), foi possível perceber que um grande número de produtos estão se aderindo a esse tipo de tecnologia devido a sua rapidez na implementação, velocidade na comunicação e simplicidade.

Em relação ao dispositivo móvel, foram identificados alguns problemas com o tráfego de dados em relação ao tipo de conexão que estava sendo utilizada, mais precisamente, ocorrendo na falta de conexão de internet ou uma conexão instável, esta é uma das melhorias para os trabalhos futuros.

Em relação a aplicação web, foi possível obter um maior conhecimento sobre as tendências e tecnologias que estão cada dia mais sendo adotadas para facilitar e agilizar a criação de novas aplicações RIA, além de entender um pouco mais em relação a este conceito.

Com o desenvolvimento desta arquitetura e dos estudos feitos, foi possível perceber o crescimento das tecnologias, principalmente em *smartphones,* voltados para a saúde, existindo diversos tipos de aplicações para cada tipo de necessidade, sendo a princípio, muito raras ou nenhuma delas, aplicações gratuitas com integração aos outros tipos de aplicações, conforme apresentada por este projeto.

Finalizado o desenvolvimento desta arquitetura, destaca-se o conhecimento adquirido em relação a arquitetura de servidores web, padrões de projetos, sincronização e integração entre dispositivos distintos, além de outras tecnologias e ferramentas utilizadas para a desenvolvimento deste trabalho, conhecimento este considerado essencial para formação profissional e acadêmica.

Como diferencial e importância pessoal, foram compreendidas as complicações decorrentes a falta de tratamento da diabetes e ausência do acompanhamento médico, que foram itens de motivação deste trabalho. Acredita-se que o resultado desta pesquisa e desenvolvimento possa contribuir na redução destes complicadores na vida das pessoas que necessitam de acompanhamento diário.

TRABALHOS FUTUROS

Como trabalhos futuros é possível destacar:

- Design da aplicação móvel – Modificar a apresentação da aplicação, alterar o estilo, deixando-a com uma visão mais agradável ao usuário.
- Design responsivo – Utilização de design responsivo para utilização da aplicação web em qualquer tipo de dispositivo. Este tipo de design já se encontra disponível nas novas atualizações realizadas pelo *framework vaadin*.
- Implementação de nível de acesso – A aplicação web está modelada para o tipo de usuário cuja função é o monitoramento dos pacientes, o próximo passo será a implementação do modulo de acesso do paciente ao sistema web.
- Implementação de relatórios e gráficos – Geração de relatórios e gráficos em ambas as aplicações.
- Otimização das funcionalidades – Procurar cada vez mais otimizar o aplicativo para que pessoas portadoras da doença possam utilizá-lo com a finalidade de melhorar seus tratamentos.
- Sincronização dos dados – Utilização de *websockets* para melhoria do tráfego de mensagens entre as aplicações.
- Traçar rota – O sistema móvel exibe a localização do paciente e locais de saúde através de um mapa. Futuramente é possível fazer com que a aplicação exiba a rota de trânsito necessária para chegada no estabelecimento escolhido. Para que isso seja possível, é necessário integrar métodos da API do Google Maps

REFERÊNCIAS BIBLIOGRÁFICAS

ANDROID DEVELOPER, componentes da aplicação android, Disponível em: <http://developer.android.com/guide/components/index.html/> Acesso em : 17 abril 2014

ANDROID PIT, Tecnologia ART, Disponível em: <http://www.androidpit.com.br/tecnologia-art-dalvik> Acesso em : 20 outubro 2014

BIBLIOTECA VIRTUAL EM SAÚDE, Ministerio da Saúde , Disponível em : <http://pesquisa.bvsalud.org/bvsms/resource/pt/oai-bvs-ms-ms-29264> Acesso em : 10 março 2014

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivan. UML Guia do Usuário, 2º Edição 2006, Editora CAMPUS LTDA.

CHARLENE C. Q, et al., Mobile diabetes intervention study, 2009 .Disponível em: < http://www.sciencedirect.com/science/article/pii/S1551714409000342#fig1 >, Acesso em 29 Novembro 2013

CHARLENE C. Q, et al., WellDoc™ Mobile Diabetes Management Randomized Controlled Trial, 2008 Disponível em : <http://online.liebertpub.com/doi/abs/10.1089/dia.2008.0283>Acesso em 29 Novembro 2013

COLBERG,S.Atividade física e diabetes,Editora Manole LTDA , 2003

DANBURY HOSPITAL, Disponível em : <http://www.danburyhospital.org/~/media/Files/Patient%20Education/patiented-portuguese/pdf_DiabetesBrazPort/DiabetesOralMeds_BrazPort.ashx /> Acesso em : 04 fevereiro 2014

DRAUZIO, V, Diabetes tipos e sintomas Disponível em: <http://drauziovarella.com.br/diabetes/diabetes/>Acesso em: 9 fevereiro 2014

FERRER.O R, et al., Mobile phone text messaging in the management of diabetes, Disponível em: <http://www.researchgate.net/publication/8223176_Mobile_phone_text_messaging_in_the_management_of_diabetes> Acesso em 29 Novembro 2013

FERRER-ROCA. Olga.et al., Mobile phone text messaging in the management of diabetes, 2004, Disponível em : <http://jtt.sagepub.com/content/10/5/282.short> Acesso em 29 Novembro 2013

FRAZÃO.A. PORTAL TUASAUDE,Tratamento Diabetes Gestacional. Disponível em: <http://www.tuasaude.com/tratamento-para-diabetes/ >Acesso em 18 Março 2014

FREIRE, F. G. J. Análise de desempenho plataformas para desenvolvimento com Sistema Operacional Android. Disponível em: <http://www.cin.ufpe.br/~tg/2012-2/jgff.pdf/> Acesso em : 6 abril 2014

GITHUB, Controle de Verão, Disponível em: <http://git-scm.com/book/pt-br/v1/Primeiros-passos-Sobre-Controle-de-Vers%C3%A3o> Acesso em : 19 Setembro 2014

GEOJSON, Formato de Codificação de estrutura de dados Geográficos em: <http://geojson.org/ > Acesso em : 19 setembro 2014

GEOSERVER, Servidor de Mapas, Disponível em: <http://br.geoserver.org/> Acesso em : 19 setembro 2014

GRAIG, Larman; UTILIZANDO UML E PADRÕES, Uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo 3º Edição 2007, Editora ARTMED EDITORA S.A

IDF (FID) Federeção Internacional de Diabetes, Diabetes Atlas 2011, 5º Edição 2011 Disponível em : <http://www.idf.org/sites/default/files/SP_6E_Atlas_Full.pdf> Acesso em : 19 Novembro 2013

IDF (FID) Federeção Internacional de Diabetes, Diabetes Atlas 2013, 6º Edição 2013 Disponível em : <http://www.idf.org/sites/default/files/SP_5EAtlas_Full.pdf> Acesso em : 19 Novembro 2013

JERSEY service web RESTfull in Java , Disponível em : <https://jersey.java.net/> Acesso em : 28 maio 2014

JOSLIN DIABETES CENTER, Centro especializado em Diabetes. Disponível em: <http://www.joslin.org/LDI/Como_se_trata_la_diabetes.html>2013 Acesso em :15 maio 2014

KAHN, C. RONALD,et al., Joslin - Diabetes Melito, Editora : ARTMED, 2009

LECHETA, R. R, Google Android para Tablet, Editora Novatec, 2012

MONTEIRO, João Bosco. Google Android - crie aplicações para celulares e tablets , Editora Casa do Codigo

ROWINSKI. D, How google may be planning to make android apps faster wirh ART, Disponível em : <http://readwrite.com/2013/11/07/how-google-may-be-planning-to-make-android-apps-faster-with-art#awesm=~oGbbPK0Vwx05Yl /> Acesso em : 29 abril 2014

SAUDATE, Alexandre. SOA Aplicado, 2012 Editora Casa do Codigo

SAUDATE, Alexandre. REST: Construa API's inteligentes de maneira simples, 2012 Editora Casa do Codigo

SILVEIRA, Paul.et al., Introdução à Arquitetura e Design de Software, Editora Casa do Codigo

SMEETS, Bram. Programando Google Web Toolkit - do Iniciante ao Profissional, 1º Edição 2009, Editora ALTA BOOKS

SOCIEDADE BRASILEIRA DE DIABETES, SBD. Disponível em : <http://www.diabetes.org.br/o-que-e-diabetes> Acesso em: 22 Janeiro 2014

SQLITE, Software de implementação SQL. Disponível em :
< http://www.sqlite.org/about.html> Acesso em :12 maio 2014

TARIDZO.S, et al., Features of Mobile Diabetes Applications, Disponível em :
< http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3222161/ >Acesso em 29 Novembro 2013

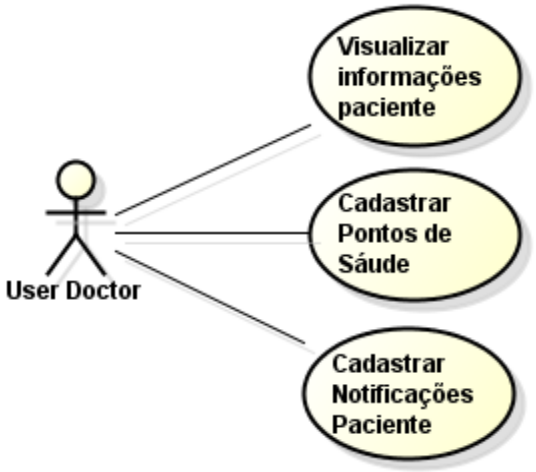
VAADIN, *Framework* de aplicações web (RIA). Disponível em : <https://vaadin.com/learn /> Acesso em :15 maio 2014

ZABBIX, Monitoramento para Infraestrutura, Serviços e Redes, Disponível em: <https://www.zabbix.com/documentation/1.8/pt/manual/sobre/introducao_ao_zabbix> Acesso em : 17 Setembro 2014

ZERO TURNA ROUD, Comparativo ferramentas de build Maven, Ivy Gradle, Disponível em: <http://zeroturnaround.com/rebellabs/java-build-tools-part-2-a-decision-makers-comparison-of-maven-gradle-and-ant-ivy/> Acesso em : 15 Julho 2014

ANEXO 1 – DIAGRAMA CASO DE USO APLICAÇÕES WEB E MÓVEL

Figura 34 - Modelo caso de uso aplicação web



Fonte: Elabora pelo autor, 2014

Modelo de Especificação de Caso de Uso

Caso de Uso: Visualizar ou remover paciente

1. Descrição:

Este caso uso tem por objetivo permitir consultar e excluir o registro de pacientes no sistema.

2. Ator:

Administrador (Médico);

3. Pré-Condição:

- O ator deverá estar devidamente cadastrado no sistema.

4. Pós-Condição:

- Acompanhamento ou exclusão de paciente do sistema.

6. Fluxo de Eventos

6.1. Fluxo Principal

P1- O caso de uso é iniciado quando o Ator Administrador acessa o sistema e seleciona a opção “Paciente” no menu principal.

P2- O sistema apresenta lista com todos os pacientes cadastrados no sistema.

P3- O ator seleciona o paciente o qual gostaria de informações ou caso não ache o paciente o ator poderá buscar pelos filtros encontrados na sua visualização.

P4- Com o paciente selecionado o ator tem a sua disposição uma aba contendo informações gerais do paciente e outra aba com o histórico coletado do dispositivo móvel.

6.2. Fluxo Alternativo

A.1. Excluir Paciente.

A.1.1. O ator seleciona a opção “Excluir”.

A.1.2. O ator confirma a exclusão selecionando a opção “OK”.

Caso de Uso: Cadastro de Pontos de saúde

1. Descrição:

Este caso uso tem por objetivo permitir o cadastro de pontos de saúdes os quais serão sincronizados aos dispositivos móveis.

2. Ator:

Administrador (Médico);

3. Pré-Condição:

- O ator deverá estar devidamente cadastrado no sistema.

4. Pós-Condição:

- Inserção ou exclusão de pontos de saúdes.

6. Fluxo de Eventos

6.1. Fluxo Principal

P1- O caso de uso é iniciado quando o Ator Administrador acessa o sistema e seleciona a opção “Locais de Saúde” no menu principal.

P2- O sistema apresenta lista com todos os pontos de saúdes cadastrados ao sistema.

P3- O ator seleciona o item adicionar.

P4- Com o preenchimento de todos os campos o ator seleciona salvar.

6.2. Fluxo Alternativo

A.1. Excluir Ponto.

A.1.1. O ator seleciona a opção “Excluir”.

A.1.2. O ator confirma a exclusão selecionando a opção “OK”.

Caso de Uso: Envio de Notificações ao paciente

1. Descrição:

Este caso uso tem por objetivo permitir o envio de notificações aos usuários da aplicação móvel.

2. Ator:

Administrador (Médico);

3. Pré-Condição:

- O ator deverá estar devidamente cadastrado no sistema.

4. Pós-Condição:

- Acompanhamento ou notificações aos pacientes do sistema.

6. Fluxo de Eventos

6.1. Fluxo Principal

P1- O caso de uso é iniciado quando o Ator Administrador acessa o sistema e seleciona a opção “Notificações” no menu principal.

P2- O sistema apresenta uma campo para preenchimento das informações e seleção do paciente.

P3-Com os dados preenchidos o ator seleciona a opção salvar.

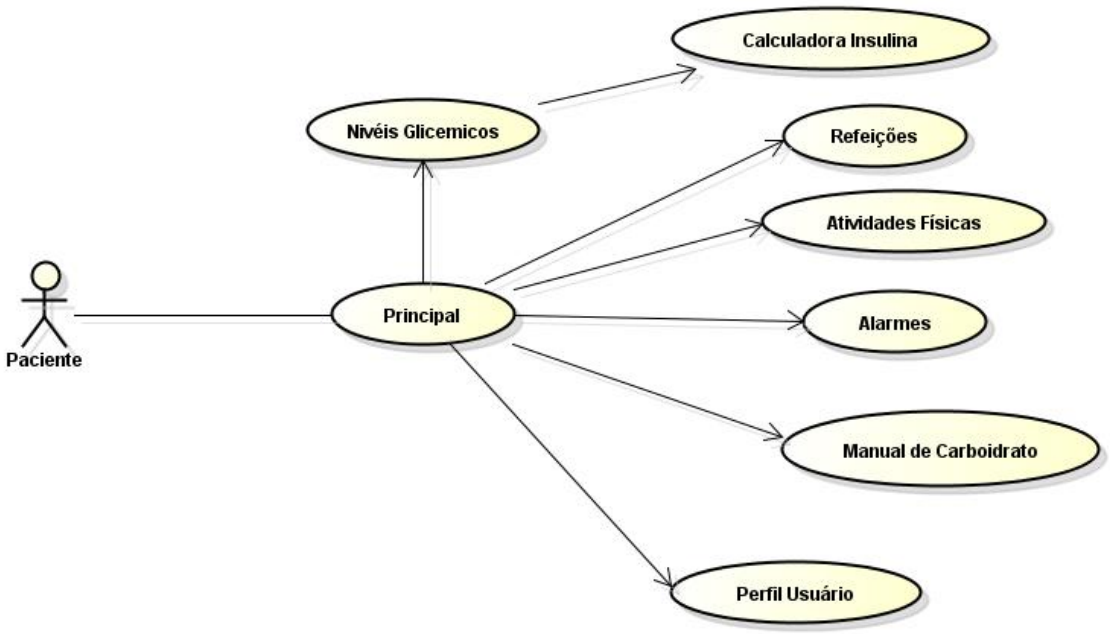
6.2. Fluxo Alternativo

A.1. Excluir Paciente.

A.1.1. O ator seleciona a opção “Excluir”.

A.1.2. O ator confirma a exclusão selecionando a opção “OK”.

Figura 35 - Modelo caso de uso Aplicação móvel



Fonte: Elabora pelo autor, 2014

Caso de Uso: Cadastrar refeição

1. Descrição:

Este caso uso tem por objetivo permitir consultar, adicionar, alterar e excluir o registro de refeições do paciente no sistema.

2. Ator:

Administrador (Paciente);

3. Pré-Condição:

- O ator deverá estar devidamente cadastrado no sistema.

4. Pós-Condição:

- Inserção, alterações ou exclusão das refeições cadastradas no sistema.

6. Fluxo de Eventos

6.1. Fluxo Principal

P1- O caso de uso é iniciado quando o Ator Administrador acessa o sistema e seleciona a opção “Refeições” no menu principal.

P2- O sistema apresenta uma lista com as refeições realizadas.

P3- O ator seleciona o seleciona a opção cadastrar nova refeição.

P4- Com os dados preenchidos, seleciona a opção salvar.

6.2. Fluxo Alternativo

A.1. Excluir Refeição.

A.1.1. O ator seleciona a opção “Excluir”.

A.1.2. O ator confirma a exclusão selecionando a opção “OK”.

Caso de Uso: Cadastrar Índice glicêmicos

1. Descrição:

Este caso uso tem por objetivo permitir consultar, adicionar, alterar e excluir o registro de níveis glicêmicos do paciente no sistema.

2. Ator:

Administrador (Paciente);

3. Pré-Condição:

- O ator deverá estar devidamente cadastrado no sistema.

4. Pós-Condição:

- Inserção, alterações ou exclusão dos níveis glicêmicos cadastrados no sistema.

6. Fluxo de Eventos

6.1. Fluxo Principal

P1- O caso de uso é iniciado quando o Ator Administrador acessa o sistema e seleciona a opção “Níveis Glicêmicos” no menu principal.

P2- O sistema apresenta uma lista com as últimas medições realizadas.

P3- O ator seleciona o seleciona a opção cadastrar índice glicêmico.

P4- Com os dados preenchidos, seleciona a opção salvar.

6.2. Fluxo Alternativo

A.1. Excluir índice glicêmico.

A.1.1. O ator seleciona a opção “Excluir”.

A.1.2. O ator confirma a exclusão selecionando a opção “OK”.

Caso de Uso: Atividades Físicas

1. Descrição:

Este caso uso tem por objetivo permitir consultar, adicionar, alterar e excluir o registro de atividades físicas do paciente no sistema.

2. Ator:

Administrador (Paciente);

3. Pré-Condição:

- O ator deverá estar devidamente cadastrado no sistema.

4. Pós-Condição:

- Inserção, alterações ou exclusão das atividades físicas cadastradas no sistema.

6. Fluxo de Eventos

6.1. Fluxo Principal

P1- O caso de uso é iniciado quando o Ator Administrador acessa o sistema e seleciona a opção "Atividades Física" no menu principal.

P2- O sistema apresenta uma lista com as atividades realizadas.

P3- O ator seleciona o seleciona a opção cadastrar nova atividade.

P4- Com os dados preenchidos, seleciona a opção salvar.

6.2. Fluxo Alternativo

A.1. Excluir Atividade.

A.1.1. O ator seleciona a opção "Excluir".

A.1.2. O ator confirma a exclusão selecionando a opção "OK".

Caso de Uso: Manual de Carboidrato

1. Descrição:

Este caso uso tem por objetivo permitir consultar, o registro de alimentos cadastrados no sistema, sendo os mesmos com seus índices de carboidratos.

2. Ator:

Administrador (Paciente);

3. Pré-Condição:

- O ator deverá estar devidamente cadastrado no sistema.

4. Pós-Condição:

- Consultar Alimentos cadastrados no sistema.

6. Fluxo de Eventos

6.1. Fluxo Principal

P1- O caso de uso é iniciado quando o Ator Administrador acessa o sistema e seleciona a opção “Manual Carboidratos” no menu principal.

P2- O sistema apresenta uma lista com os alimentos cadastrados no sistema.

Caso de Uso: Perfil Usuário

1. Descrição:

Este caso uso tem por objetivo permitir alterar o registro do paciente no sistema.

2. Ator:

Administrador (Paciente);

3. Pré-Condição:

- O ator deverá estar devidamente cadastrado no sistema.

4. Pós-Condição:

- Alterações do perfil do paciente cadastrado no sistema.

6. Fluxo de Eventos

6.1. Fluxo Principal

P1- O caso de uso é iniciado quando o Ator Administrador acessa o sistema e seleciona a opção “Perfil” no menu principal.

P2- Com os dados alterados ou somente para visualização, o ator seleciona a opção salvar para confirmar as alterações.