UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CÂMPUS CORNÉLIO PROCÓPIO
DIRETORIA DE GRADUAÇÃO E EDUCAÇÃO PROFISSIONAL
TECNOLOGIA EM INFORMÁTICA
MODALIDADE SISTEMAS DE INFORMAÇÃO

MARCELO FERREIRA DA SILVA

SISTEMA DE GESTÃO E ACOMPANHAMENTO DE METAS

TRABALHO DE CONCLUSÃO DE CURSO

CORNÉLIO PROCÓPIO
2015

MARCELO FERREIRA DA SILVA

SISTEMA DE GESTÃO E ACOMPANHAMENTO DE METAS

Trabalho de Conclusão de Curso apresentada à disciplina de Trabalho de Diplomação do Curso Superior de Tecnologia em Informática - Modalidade Sistemas de Informação da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Msc. Francisco Pereira Junior

CORNÉLIO PROCÓPIO

2014

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Cornélio Procópio
Diretoria de Graduação e Educação Profissional
Tecnologia em Informática
Modalidade Sistemas de Informação

FOLHA DE APROVAÇÃO

Sistema de Gestão e Contratação de Metas

por

Marcelo Ferreira da Silva

Este trabalho foi apresentado em 17 de março de 2015 às 16h, como requisito parcial a obtenção do grau de Tecnólogo em Análise e Desenvolvimento de Sistemas pela Universidade Tecnológica Federal do Paraná – Campus Cornélio Procópio. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após a deliberação, a Banca Examinadora considerou o trabalho como aprovado.

_____

Msc. Francisco Pereira Junior
Orientador

_____

Willian Massami Watanabi
Membro da Banca Examinadora

_____

Antonio Carlos Fernandes da Silva
Membro da Banca Examinadora

"A Folha de Aprovação assinada encontra-se na Coordenação do Curso"

A aquele que vocês podem não conhecer,
mas que pelo resultado de seu esforço me fez ser aquilo que vocês conhecem,
Meu Pai.

AGRADECIMENTOS

Agradeço a Deus por colocar em diversos momentos de minha vida, pessoas que souberam sempre me oferecer aquilo que necessitava em cada momento de dificuldade.

Pessoas maravilhosas, que nos momentos em que as nuvens do desanimo me abatia, vinham como ventania para afastar as nuvens e permitir que eu voltasse novamente a ver o sol.

Entre estes estão meus pais, que através do seu exemplo e do seu trabalho sempre me mostraram o caminho da dignidade.

Estão também meus amigos que me ensinaram que a amizade é muito mais importante que qualquer premio ou reconhecimento que eu já recebi, já que os laços que criei nessa instituição se perpetuaram para além dos limites que esta vida nos impõem.

A Adriana, que me incentivou a largar a matemática e a fazer essa faculdade.

A minha família que sempre me apoiou.

A minha esposa Silvia, que muito cobrou para que chegasse até esse momento.

Ao meu orientador, que mais do que isso foi um amigo. Não apenas nesses últimos 7 ou 8 meses, mas durante todo o período de faculdade. Obrigado Thesko.

E especialmente ao meu pai, que por muito pouco não pode me ver chegar aqui.

RESUMO

SILVA, Marcelo Ferreira da. **Sistema de gestão e acompanhamento de metas**. 2015. 140 f. Trabalho de Conclusão de Curso (Graduação) – Tecnologia em Informática - Modalidade Sistemas de Informação. Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2015.

Este trabalho apresenta as características da computação na nuvem, esclarecendo as diferenças entre os serviços disponibilizados no mercado, de acordo com o nível de abstração que o desenvolvedor ou cliente necessita. Exemplifica por meio da implementação de uma aplicação de apoio a gestão, como é possível utilizar os recursos da computação na nuvem para desenvolvimento de projetos em tecnologia Java escalonáveis, apresentando as tecnologias utilizadas e a arquitetura de um fornecedor de serviços de computação na nuvem.

Palavras-chave: PAAS. Nuvem. Heroku. Java.

ABSTRACT

SILVA, Marcelo Ferreira da. **System Management and Monitoring Goals**. 2015. 140 f. Trabalho de Conclusão de Curso (Graduação) – Tecnologia em Informática - Modalidade Sistemas de Informação. Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2015.

This paper presents the characteristics of cloud computing, clarifying the differences between the services available on the market , according to the level of abstraction that the developer or customer needs . Exemplified by the implementation of a management application, show how you can use the resources of cloud computing for development projects in Java scalable technology, presenting the technologies used and the architecture of a supplier of computing services in the cloud.

Keywords: PAAS. Cloud. Heroku. Java.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

AJAX	*Asynchronous Javascript and XML*
API	*Application Programming Interface*
ASP	*Active Server Pages*
CASE	*Computer-Aided Software Engineering*
CGI	*Common Gateway Interface*
CRM	*Customer Relationship Management*
CSS	*Cascading Style Sheets*
EAR	*Enterprise ARchive*
EJB	*Enterprise Java Beans*
GNU	*GNU's Not Unix*
IaaS	*Infrastructure as a service*
JAR	*Java ARchive*
JDO	*Java Data Objects*
JEE	*Java Enterprise Edition*
JPA	*Java Persistence API*
JSP	*Java Server Pages*
JSR	*Java Specification Requests*
MIT	*Massachusetts Institute of Technology*
MVC	*Model-View-Controller*
PaaS	*Plataform as a service*
PHP	*PHP: Hypertext Preprocessor*
POC	*Proof of Concept*
REST	*Representational State Transfer*
SaaS	*Software as a service*
SGBD	Sistema Gerenciador de Banco de Dados
UML	*Unified Modeling Language*
UTFPR	Universidade Tecnológica Federal do Paraná
WAR	*Web application ARchive*
XML	*eXtensible Markup Language*

Sumário

1 INTRODUÇÃO

A evolução tecnológica e a acessibilidade às tecnologias proporcionam uma nova perspectiva sobre a agilidade na gestão de informações. O mercado de tecnologia da informação tornou-se amplo e extremamente necessário, pois todos os dias são produzidas milhares de informações relevantes, que mudam rapidamente e precisam estar consistentes e atualizadas para auxiliarem nas tomadas de decisões (WHERTHEIN, 2000).

A proposta da tecnologia da informação é justamente tratar este volume de dados, permitindo organizar, manipular e recuperar as informações desejadas, de maneira rápida e fácil. Esta necessidade precisa ser suprida em diferentes escalas, em grandes instituições ou pequenos setores, ter as informações consistentes e facilmente acessíveis é imprescindível para a eficácia de sua utilização (SOMMERVILLE, 2007).

Hoje o mercado corporativo em geral, depende plenamente da tecnologia da informação que se tornou um serviço de primeira necessidade, assim como a energia ou a água. Pequenas empresas a partir do momento em que começam a crescer sentem-se obrigadas a se informatizar para não perder em competitividade para os concorrentes.

No início de sua informatização, uma única estação de trabalho com um software de gestão supri as necessidades, mas a medida que cresce a demanda, busca-se uma solução cliente-servidor para que vários usuários também usem o sistema. Com o crescimento esperado, em breve uma necessidade de ampliação traz custos de infraestrutura e pessoal especializado na área técnica, criando departamentos de informática, muitas vezes com custo significativo perante o área de negócio da própria empresa.

Nesse ambiente, um conceito tecnológico que vem se estabelecendo demonstrando que esse é o futuro das aplicações: é o chamado *Cloud Computing*, ou Computação nas Nuvens. Este conceito tem adquirido grande maturidade no mercado e, conforme novas soluções vão surgindo, vai se tornando mais acessível para o desenvolvedor a experimentação e até o uso no dia-a-dia.

Por meio dele é possível que uma empresa escolha onde deseja investir e a partir de qual nível ela quer se preocupar, permitindo assim focar muito mais em seu negócio principal do que na infraestrutura, que deveria ser um

elemento de apoio.

No governo e nas empresas privadas, o uso da tecnologia é um processo prioritário, que faz parte de seus orçamento e onde se investe em profissionais e tecnologias novas como a Computação na Nuvem. Por outro lado, no terceiro setor da economia, as instituições filantrópicas, associações, igrejas e fundações, ainda estão engatinhando em sua jornada para organização e aos poucos estão expandindo suas atividades. Nesse cenário o uso da tecnologia se torna indispensável para melhorar o atendimento prestado.

Para apresentarmos os conceitos aprendidos durante a realização do Curso de Tecnologia em Informática da Universidade Tecnológica Federal do Paraná (UTFPR) – Campus Cornélio Procópio, escolhemos um estudo de caso, nesse meio, para criação de um sistema de apoio a gestão de metas.

2 REVISÃO BIBLIOGRÁFICA

2.1 METODOLOGIA DE DESENVOLVIMENTO

Uma metodologia de desenvolvimento é formado por um conjunto de atividades que auxiliam na produção de software. Segundo, Sommerville (2003), existem atividades fundamentais comuns a todas as metodologias, sendo elas, a Especificação de Software, o Projeto e Implementação de Software, a Validação de Software e a Evolução de Software.

Dentre as metodologias existentes, algumas delas foram criadas a partir da necessidade de um desenvolvimento mais flexível a mudanças e menos custoso em relação aos métodos tradicionais, elas foram chamadas de metodologias ágeis. É possível citar entre elas o XP, Scrum e o Kanban.

A escolha do Scrum se deu por ser uma metodologia ágil para gerência de projetos. Ela é baseada em ciclos de uma a quatro semanas, que são chamados de Sprint, como pode ser visto na figura 1, no qual se trabalha para alcançar objetivos bem definidos para a Sprint (SABBAGH, 2013).

Os objetivos gerais de um projeto que utiliza Scrum são representados no *Product Backlog*, uma lista de coisas para fazer que é constantemente atualizada e repriorizada. Ao se iniciar uma Sprint é criado a Sprint *Backlog*, contendo os itens que serão desenvolvidos no ciclo.

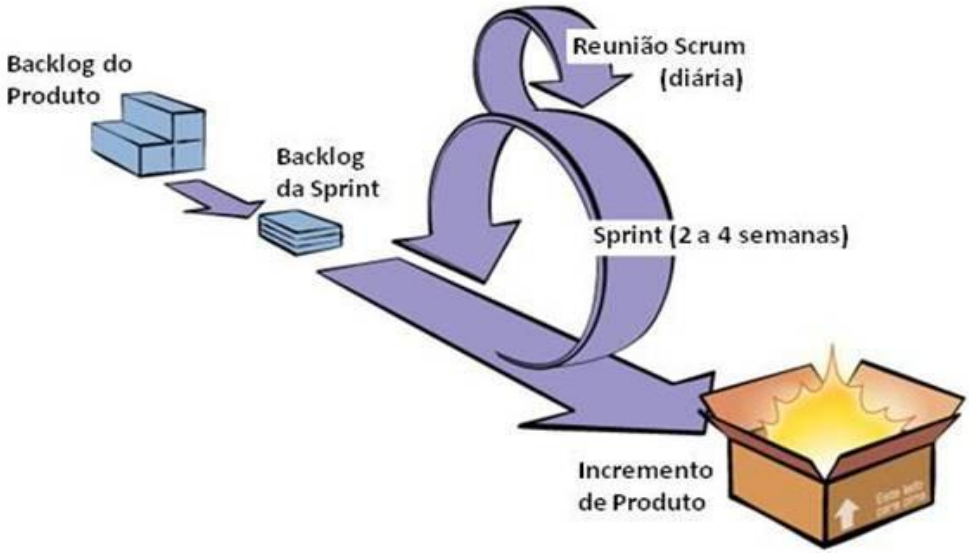


Figura 1 Ciclo do Scrum

Algumas de suas principais características visando minimizar os riscos são:

- Divisão do projeto em pedaços menores e curtos, variando de uma semana a um mês;
- Comunicação entre a equipe e o cliente em tempo real;
- Participação ativa do cliente;
- Alta capacidade de resposta a mudanças;
- Entregas continuas e incrementais, facilitando a validação se o que está sendo desenvolvido realmente atende a necessidade do cliente;

O processo Scrum fornece a habilidade de declarar o produto “pronto” sempre que necessário (porque a concorrência acabou de entregar, porque a empresa precisa de dinheiro, porque o usuário/cliente precisa das funções, porque foi para essa data que foi prometido) (PRESSMAN, 2006).

Os princípios Scrum são usados para guiar as atividades de desenvolvimento dentro de um processo que incorpora as seguintes atividades: requisitos, análise, projeto, evolução e entrega. Em cada atividade, as tarefas ocorrem dentro de um padrão de processo chamado de Sprint. O trabalho conduzido dentro de um Sprint, é adaptado ao problema em mãos e é definido e modificado com frequência, em tempo real pela equipe Scrum.

Para esse projeto trabalhamos com alguns conceitos do Scrum, a saber:

Pendência: uma lista priorizada de requisitos ou características do projeto que fornecem valor de negócio para o cliente. Itens podem ser adicionados à pendência a qualquer momento (é assim que as modificações são introduzidas). O gerente de produto avalia a pendência e atualiza as prioridades quando necessário.

Eventos Scrum: Eventos no Scrum são reuniões rotineiras, desta maneira minimizando a chance de ocorrer reuniões não programadas. Toda reunião no Scrum tem o conceito de *Time-boxed*, isso significa que ela tem uma duração fixa, não podendo levar mais, ou menos tempo, caso ocorra de levar mais ou menos tempo que o preciso, a reunião deve ser finalizada e na reunião de Retrospectiva da

Sprint deve ser analisado os motivos que o mesmo veio a ocorrer. Seguem abaixo os Eventos que o Scrum atende.

2.1.1 Eventos Scrum

2.1.1.1 Sprints

Consiste de unidades de trabalho que são necessárias pra satisfazer a um requisito definido na pendência que precisa ser cumprido em um intervalo de tempo predefinido (de 2 a 4 semanas). Durante o Sprint, os itens em pendência a que a unidade de trabalho do Sprint se destinam são congelados (isto é, não são introduzidas as modificações durante o Sprint). Assim, o Sprint permite que os membros da equipe trabalhem em um ambiente de curto prazo, mas estável.

2.1.1.2 Reunião Diária

A Reunião Diária do Scrum é um evento *time-boxed* de 15 minutos, para que o time de desenvolvimento possa sincronizar as atividades e criar um plano para as próximas 24 horas. Esta reunião é feita para inspecionar o trabalho desde a última Reunião Diária, e prever o trabalho que deverá ser feito antes da próxima Reunião Diária.

2.1.1.3 Revisão da Sprint

A Revisão da Sprint é executada no final da Sprint para inspecionar o incremento e adaptar o *Product Backlog* se necessário. Durante a reunião de Revisão da Sprint o time Scrum e as partes interessadas colaboram sobre o que foi feito na Sprint. Com base nisso e em qualquer mudança no *Product Backlog* durante a Sprint, os participantes colaboram nas próximas coisas que podem ser feitas para otimizar valor. Esta é uma reunião informal, não uma reunião de status, e a apresentação do incremento destina-se a motivar e obter comentários e promover a colaboração.

2.1.1.4 Retrospectiva da Sprint

A Retrospectiva da Sprint é uma oportunidade para o time Scrum inspecionar a si próprio e criar um plano para melhorias a serem aplicadas na próxima Sprint.

2.1.2 Scrum Solo

Para o desenvolvimento do Sistema de Gestão e Acompanhamento de Metas usamos uma adaptação do Scrum, ver figura 2, para uso em equipes de apenas um desenvolvedor, onde as práticas Scrum o auxiliam a planejar e executar todo processo de desenvolvimento de forma ágil e sem sustos durante o projeto.

Nesta projeto utilizamos a *Product Backlog*, o *Sprint Backlog*, os Sprints e a Retrospectiva de Sprint. Em substituição as reuniões diárias, montamos um diário com as informações de andamento. As Sprints teriam prazos de 15 dias e a cada período aproximado de 30 dias o produto seria validado junto ao grupo de validação, no nosso caso os clientes do sistema. Quinzenalmente o aluno participaria de uma *Oriented Meeting – OM* (reunião de orientação – aluno, professor orientador).

O processo é alicerçado pela planta de desenvolvimento (artefatos gerados durante a construção do software, por exemplo: diagramas, casos de teste, etc.) e pela atividade de gerenciamento de projeto. Os artefatos que a planta de desenvolvimento deve conter e a forma a ser utilizada para a gestão são definidos pelos participantes da OM.

Figura 2 Adaptação para o Scrum Solo

2.2 CONCEITOS

Apesar de haver um capítulo apenas para tratar sobre as tecnologias utilizadas e outro sobre arquitetura do sistema, é preciso definir alguns conceitos muito importantes para compreender como essa aplicação foi construída, e como a mesma pode servir de base para construção de qualquer sistema.

2.2.1 Computação na Nuvem

Segundo Silva(2012), definir Computação na Nuvem, ou do inglês *Cloud Computing*, é algo delicado, uma vez que é possível achar em qualquer sistema de busca, várias definições diferentes de diversos autores importantes, que, além de totalmente válidas, são muitas vezes complementares. Dependendo da perspectiva, a Computação na Nuvem pode ter diferentes aspectos: para um Administrador de Infraestrutura com foco em Sistema Operacional, pode ser uma máquina virtual provida e gerenciada por uma ferramenta de virtualização. Já para um Desenvolvedor de Software, pode ser também uma plataforma flexível, que visa dar mais recursos de processamento em momentos de sobrecarga, assim como economizar os recursos quando estes não mais forem necessários. Para um

Profissional de Negócios, a Computação na Nuvem pode dar uma tranquilidade “não-técnica” resumida em “atender bem e sempre”.

Existe também o conceito de “pagar pelo que se consome” (pay-as-you-go), ou seja, se você tem alto número de usuários, vai precisar de mais recursos (processamento, memória, storage, etc.), logo vai de alguma maneira pagar mais por isso; ao passo que se sua aplicação não consumir tantos recursos, você não vai ocupar tanto o ambiente, logo pagará de alguma forma menos por isto.

Quando falamos desta perspectiva em Computação na Nuvem, é impossível não associarmos ao fornecimento, por exemplo, de energia elétrica. Em geral, os consumidores não têm a mínima ideia de por onde os cabos passam, quantas linhas a eletricidade atravessou até chegar a seus lares ou mesmo de onde sua energia elétrica vem. Em resumo, querem simplesmente ligar a luz, que isto funcione sempre que for desejado, e que se possa pagar pelo consumido. Os fornecedores de Computação na Nuvem emulam o mesmo comportamento e, por mais que ocorram falhas, estas devem permanecer escondidas e contornadas, para que o consumidor sempre tenha seus serviços contratados disponíveis de forma transparente.

Segundo Celestino(2014), as empresas agora dispõe de mais um tipo de serviço contratado sob demanda, que é o serviço de TI. Assim elas passam a pagar por ele, como já pagam pelos serviços de eletricidade, gás e água que utilizam. Uma das primeiras características dessa tecnologia foi em 1999 com a chegada da Saleforce.com que inaugurou o conceito de fornecimento de aplicativos empresariais através de uma simples Website. Posteriormente em 2002 a AmazonWeb Services passou a fornecer um conjunto de serviços baseados em nuvem, incluindo armazenamento e processamento; já em 2006 a Amazon deu um importante passo para a história da computação na nuvem com o lançamento do *Elastic Compute Cloud* (EC2), um serviço Web que permite que as pequenas empresas e usuários comuns alugassem computadores nos quais podem executar suas próprias aplicações. Hoje a Amazon é conhecida como a pioneira em fornecimento de serviços em nuvem.

2.2.2 Modelos de Computação na Nuvem

Atualmente a Computação na Nuvem é fornecida em três modelos de serviço, sendo eles: IaaS (*Infrastructure as a service*), PaaS (*Plataform as a service*) e SaaS (*Software as a service*), além de alguns modelos de implementações contendo diversas características específicas para plataformas e linguagens.

Para entender se a nuvem é uma Infraestrutura como Serviço, Plataforma como Serviço ou Software como Serviço, a Figura 3 ilustra como gerenciar as diferentes partes deste serviço.

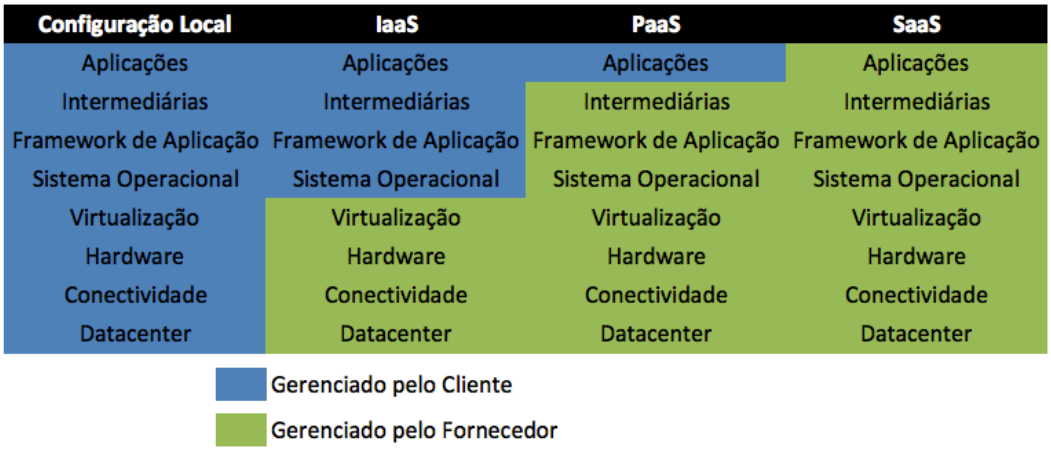


Figura 3 Modelos de Serviços de Computação na Nuvem (SNOWMAN, 2010)

As empresas que tendem a considerar o uso do sistema na nuvem em seus ambientes devem calcular a economia de custos que este sistema pode oferecer e quais riscos adicionais estão sujeitos.

2.2.2.1 IaaS (*Infrastructure as a Service*)

Em março de 2006, o economista Nicholas Carr, no seu blog RoughType (CARR, 2006), criou o termo Hardware as a Service (HAAS), para descrever serviços como o EC2 oferecido pela Amazon. No final do mesmo ano esses serviços começaram a ser tratados por algumas empresa como IAAS

(Infraestrutura como Serviço).

Este é o modelo onde você tem controle total do recurso de máquina virtual que lhe é disponibilizada. Da mesma forma que você tem todo o poder e direito do usuário Administrador (*root*), você também tem a responsabilidade de instalar tudo que é necessário para atender a sua aplicação – desde o banco de dados, servidor de aplicações, entre outros – e também cuidar de todas as questões e aspectos de segurança. Como exemplo de fornecedores destes serviços de IaaS, pode-se citar: Amazon EC2[1], RackSpace[2] e alguns fornecedores brasileiros;

Os fornecedores de IAAS gerenciam a transição e a hospedagem de aplicações compiladas em suas infraestruturas. Os clientes que utilizam IAAS mantêm o gerenciamento de todas as suas aplicações, sem se preocupar com a manutenção da infraestrutura.

2.2.2.2 PaaS (Platform as a Service)

No PaaS (Plataforma como Serviço), o fornecedor dispõe de parte dos serviços e softwares que você precisa já instalados e em execução. Se no modelo IaaS você deveria se preocupar em instalar tudo partindo do zero, no PaaS você tem algumas combinações já pré-construídas, fazendo com que você se preocupe muito mais com o negócio e implementação da sua aplicação propriamente do que com a instalação de um servidor de aplicações, banco de dados e assim por diante. Em resumo, PaaS já lhe oferece o ambiente preparado para você instalar ("*deploy*") as aplicações sem grandes preocupações. Entre os fornecedores desta categoria de soluções, pode-se citar o Microsoft Azure[3], AWS Elastic Beanstalk[4], Jelastic[5], CloudBees[6], OpenShift da Red Hat [7]e o Heroku da SalesForce[8], que será o PaaS que iremos nos aprofundar neste trabalho.

Basicamente o que difere cada um são as tecnologias que eles permitem utilizar, como linguagens ou *containers*, quantidades de serviços

[1] http://aws.amazon.com/pt/ec2/
[2] http://www.rackspace.com/pt/
[3] http://azure.microsoft.com/pt-br/
[4] http://aws.amazon.com/pt/elasticbeanstalk/
[5] http://jelastic.com/
[6] https://www.cloudbees.com/
[7] https://www.openshift.com/
[8] https://www.heroku.com/

adicionais e, é claro, a precificação, ou seja, o custo por memória, CPU e disco de cada instância da aplicação. Outra variável é a facilidade de criação e instalação de um aplicativo. Neste ponto o Heroku está um passo à frente dos outros fornecedores, pela simplicidade que é criar, manter e escalar uma aplicação.

2.2.2.3 SaaS (Software as a Service)

Na modalidade de SaaS (Software como Serviço) você não se preocupa nem em instalar seu software, uma vez que o fornecedor já lhe entrega tudo pronto. Toda a questão de bancos de dados, segurança e a própria aplicação é oferecida em um único contrato ou acordo. Neste modelo, ao mesmo tempo em que dispõe de maior comodidade, você recebe também quase nenhum controle além de algumas parametrizações ou customizações, uma vez que você tem que confiar que seu fornecedor irá cuidar bem das informações de seus usuários, seus dados e garantir a disponibilidade da aplicação. Neste modelo há fornecedores como SalesForce, que oferece um dos CRMs mais utilizados no mercado, e Google, com suas soluções de colaboração como Google Docs e Google Apps.

Segundo Snowman (2010), SaaS é uma aplicação completa que está rodando na nuvem e disponível para os negócios do cliente sem adicionais trabalhos técnicos.

2.3 ARQUITETURA DO SISTEMA

Para desenvolvimento do sistema foi escolhida a plataforma Heroku. O Heroku se enquadra na categoria de serviços da computação em nuvem conhecida como Plataforma como Serviço (Platform as a Service, ou PaaS), no qual o fornecedor entrega para o cliente um ambiente pronto para receber a aplicação. Diferente do IaaS (Infraestrutura como Serviço), no qual cliente contrata máquinas (reais ou virtuais) e é responsável pela instalação de bibliotecas, montagem das estruturas do sistema de arquivos, entre outros recursos, o PaaS é uma solução de alto nível que abstrai este tipo de preocupação, informações vistas na seção 2.5.2.

Como o ambiente é entregue pelo fornecedor, ao cliente basta se concentrar em desenvolver e instalar a aplicação. Normalmente nos serviços PaaS a instalação ou atualização é feita através de *commits* em repositórios remotos vinculados à aplicação.

O Heroku trabalha com projetos modelos, que podem ser clonados em um repositório Git, e a partir dessa personalização já é possível ter acesso a sua aplicação em endereço específico.

Ao fazer *commit* das suas alterações, através de um processo de integração contínua, o Heroku, já compila a sua aplicação, se for o caso, e já promove a implantação no subdomínio escolhido.

No caso da linguagem Java, o Heroku fornece quatro modelos de projetos, sendo um projeto com o *Web Container* Jetty, e uso de Jax-rs para fornecer serviços Web REST, um projeto Spring MVC usando o *Web Container* Tomcat, um projeto usando o *framework* Play e um projeto JSP/Servlet para uso com Jetty.

Os projetos usam Maven para automatizar o processo de gerenciamento das bibliotecas, compilação do projeto e implantação no servidor, permitindo ao desenvolvedor se preocupar apenas com a codificação do projeto.

A Figura 4, apresenta a arquitetura do Heroku, mostrando que o desenvolvedor pode codificar em sua linguagem, fazer o deploy via Git/Maven e gerenciar a aplicação através das interfaces fornecidas pelo Heroku. Já o usuário terá acesso a aplicação através de uma interface Web criada pelo desenvolvedor ou através do uso de API. O acesso do usuário é de responsabilidade da aplicação do desenvolvedor, e o Heroku fica responsável pelo balanceamento das chamadas e

roteamento da aplicação.

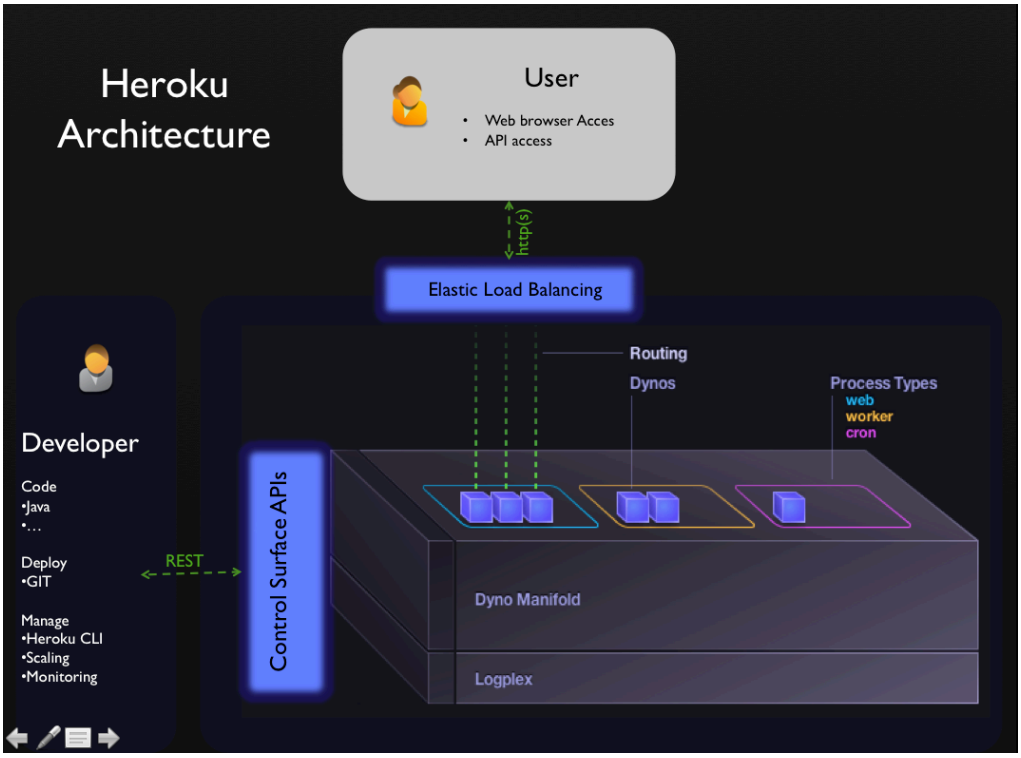


Figura 4 Arquitetura do Heroku

(Fonte: https://developer.salesforce.com/page/Webinar:_Social_Enterprise_Java_Apps_on_Heroku_(2012-May)

O Heroku fornece toda infraestrutura ao desenvolvedor, de forma transparente para o usuário.

2.4 TECNOLOGIAS UTILIZADAS

2.4.1 UML

Para modelar o sistema foi utilizada a UML, linguagem visual para softwares construídos sob o paradigma orientado a objetos, que se tornou o padrão adotado pela indústria de engenharia de software (GUEDES, 2009). Para este fim utilizar-se-á a ferramenta CASE Astah em sua versão *Community* que oferece os recursos suficientes para suprir a amplitude do projeto.

2.4.2 Java Enterprise Edition - Java EE

A plataforma Java EE é o padrão utilizado para o desenvolvimento de aplicações em ambientes corporativos com Java. Esta plataforma provê aos

desenvolvedores um poderoso conjunto de APIs, reduzindo o tempo de desenvolvimento, reduzindo a complexidade e aumentando a performance da aplicação.

Os componentes base e a plataforma independente desta tecnologia, fazem com que se torne fácil escrever aplicações Java EE, uma vez que a lógica de negócio está organizada em componentes reutilizáveis. Além disso, o servidor Java fornece serviços sob a forma de *container* para cada tipo de componente.

Na arquitetura Java EE a finalidade básica de um *container* é fornecer o ambiente de execução para os componentes. A especificação Java EE define um contrato entre componentes e *container* e especifica o modelo de desenvolvimento destes componentes. Este contrato especifica como estes componentes devem ser desenvolvidos e implementados e como os componentes podem acessar serviços fornecidos pelo *container*.

A especificação Java EE destaca cinco componentes ou *containers* suportados em um produto Java EE. São eles:

- **Java EE Server:** é a parte de execução de um produto Java EE. Um Java EE Server provê serviços EJB (*Enterprise Java Beans*) e *Web containers*;
- **Enterprise Java Beans (EJB):** gerencia a execução de *enterprise beans* para aplicações Java EE. *Enterprise bean* e outros *containers* rodam dentro de um Java EE *Server*;
- **Web Container:** gerencia a execução de páginas, servlets e alguns componentes EJB para aplicações Java EE. Componentes Web rodam dentro do Java EE Server;
- **Aplicação Cliente:** gerencia a execução de componentes de aplicações cliente, que podem ser desenvolvidos em *Swing*, AWT ou outra tecnologia desktop, porém com acesso a todos os recursos do Java EE;
- **Applet:** gerencia a execução de Applets que consistem de um plug-in Java e um browser rodando juntos no lado cliente;

A figura 5 ilustra o servidor Java EE, seus containers e

componentes.

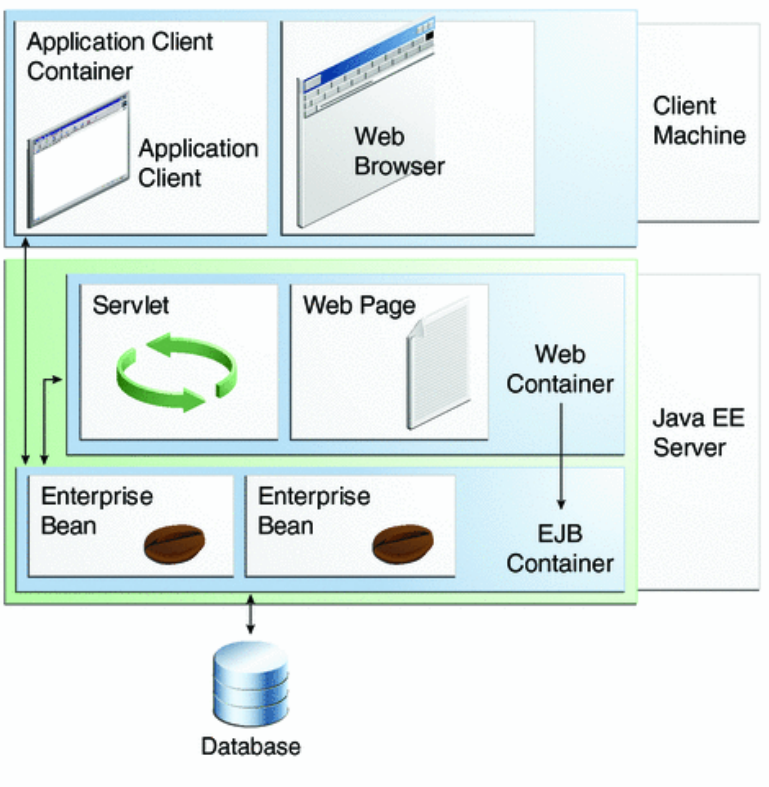


Figura 5 Java EE Server, Componentes e Containers

Fonte: http://download.oracle.com/javaee/6/tutorial/doc/figures/overview-serverAndContainers.gif.

Neste trabalho é abordado o *Web Container*, pois é nele que o sistema proposto está inserido.

Um *Web Container* tem a responsabilidade de instanciar, inicializar e invocar componentes como páginas JSP e Servlets, é ele o responsável pelo gerenciamento do ciclo de vida destes, os quais são abordados nas próximas sessões.

O *Web Container* faz parte de servidores Web ou servidores de aplicações, como exemplo pode-se citar o Tomcat ou JBoss, que fornecem serviços de rede através de requisições e respostas que são enviadas aos mesmos.

A seguir detalhado sobre duas importantes tecnologias contidas na plataforma Java EE, são elas Servlets e JSP.

2.4.2.1 Servlets/JSPs

Servlets são classes Java, desenvolvidas de acordo com uma estrutura bem definida que quando instaladas e configuradas em um servidor que implemente um *Web Container*, podem tratar requisições recebidas de clientes Web, como por exemplo os *Browsers*.

Ao receber uma requisição, um Servlet pode capturar os parâmetros desta requisição, efetuar qualquer processamento inerente a uma classe Java, e devolver uma página HTML.

As páginas JSP, ou Java Server Pages, foram criadas para contornar algumas das limitações no desenvolvimento com Servlets: se em um servlet a formatação da página HTML resultante do processamento de uma requisição se mistura com a lógica da aplicação em si, dificultando a alteração dessa formatação, em uma página JSP essa formatação se encontra separada da programação, podendo ser modificada sem afetar o restante da aplicação.

Assim, um JSP consiste de uma página HTML com alguns elementos especiais, que conferem o caráter dinâmico da página. Esses elementos podem tanto realizar um processamento por si, como podem recuperar o resultado do processamento realizado em um servlet, por exemplo, e apresentar esse conteúdo dinâmico junto à página JSP.

A utilização de servlets e de páginas JSP oferecem diversas vantagens em relação ao uso de outras tecnologias (como PHP, ASP e CGI). As principais vantagens são herdadas da própria linguagem Java, como:

- **Portabilidade:** a aplicação desenvolvida pode ser implantada em diversas plataformas, como por exemplo, Windows, Unix, GNU/Linux e Macintosh, sem que seja necessário modificar ou mesmo reconstruir a aplicação;
- **Facilidade de programação:** a programação é orientada a objetos, simplificando o desenvolvimento de sistemas complexos. Além disso, a linguagem oferece algumas facilidades, como por exemplo, o gerenciamento automático de memória (estruturas alocadas são automaticamente liberadas, sem que o desenvolvedor precise se preocupar em gerenciar esse processo);
- **Flexibilidade:** o Java já se encontra bastante difundido, contando com uma enorme comunidade de desenvolvedores, ampla documentação e

diversas bibliotecas e códigos prontos, dos quais o desenvolvedor pode usufruir.

Além dessas vantagens, a arquitetura de servlets e páginas JSP possibilitam alguns benefícios adicionais, como:

- **Escalabilidade:** na maior parte dos servidores de aplicações modernos, é possível distribuir a carga de processamento de aplicações desenvolvidas em diversos servidores, sendo que servidores podem ser adicionados ou removidos de maneira a acompanhar o aumento ou decréscimo dessa carga de processamento;
- **Eficiência:** os servlets carregados por um servidor persistem em sua memória até que ele seja finalizado. Assim, ao contrário de outras tecnologias, não são iniciados novos processos para atender cada requisição recebida; por outro lado, uma mesma estrutura alocada em memória pode ser utilizada no atendimento das diversas requisições que chegam a esse mesmo servlet;
- **Recompilação automática:** páginas JSP modificadas podem ser automaticamente recompiladas, de maneira que passem a incorporar imediatamente as alterações sem que seja necessário interromper o funcionamento da aplicação como um todo.

2.4.3 Spring MVC

O Spring MVC é o *framework* Web nativo do Spring que se baseia no padrão *Model-View-Controller* (MVC).

Conforme explica (SPRING, 2014) a documentação do *framework*, o módulo Web do Spring contém inúmeros recursos, tais como:

- Separação clara de papéis;
- Adaptabilidade e flexibilidade;
- Código de negócio reutilizável sem a necessidade de duplicação;
- Customização de mapeamento de manipuladores e de resolução de visualização;
- Transferência flexível do *Model*;

- Uma biblioteca de *tags* JSP, introduzida no Spring 2.0, que torna a escrita de formulários em páginas JSP muito mais fácil.

Como a maioria dos *frameworks* MVC em Java, o Spring MVC é projetado em torno de um *DispacherServlet*. O *DispacherServlet* tem o papel de um *front controller*, que consiste em um modelo de aplicativo comum onde todas as requisições são passadas para um único servlet e este é responsável por delegar a requisição a servlets ou componentes de um outro aplicativo.

De acordo com a documentação de referência do Spring o *DispacherServlet* recebe todas as requisições e as despacha para manipuladores que geralmente são classes baseadas nas anotações *@Controller* e *@RequestMapping* que oferecem uma grande gama de flexíveis métodos que são responsáveis pelo tratamento da requisição.

Para que seja possível entender o funcionamento deste *framework* vamos descrever o caminho que uma requisição percorre desde o momento em que ela deixa o navegador até o momento em que ela retorna com uma resposta ao usuário conforme representado na figura 6.

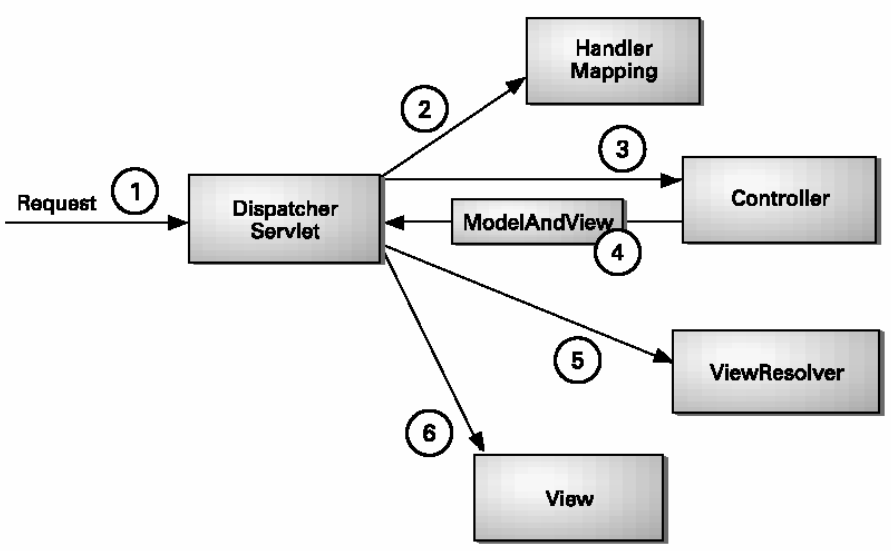


Figura 6 O ciclo de vida de uma requisição no Spring MVC

Fonte: http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/mvc.html

Todas as requisições devem passar pelo *DispacherServlet*, este, por

sua vez determinará através da URL da requisição o *controller* apropriado para tal solicitação.

Após escolhido o *controller*, o mesmo irá tratar a requisição e fornecer os dados que serão mostrados ao usuário. Estes dados são chamados de modelo (*model*) e serão enviados para uma *view*, que é a responsável por exibir de forma amigável as informações ao usuário. O *controller* utilizará um objeto *ModelAndView* para enviar ao *DispacherServlet* os dados do *model* e o nome da *view*. Por fim, os dados do *model* serão entregues a *view* correspondente que geralmente será uma JSP (Java Servers Pages). Assim termina o ciclo de vida de uma requisição no Spring.

2.4.4 Java Persistence API

Na camada de persistência da aplicação foi utilizado o *Java Persistence API* - JPA, que é a especificação Java para o gerenciamento da camada de persistência e mapeamento objeto/relacional, facilitando desta forma este tipo de mapeamento em aplicações Java. Neste projeto esta API terá seu funcionamento integrado ao Spring.

O JPA lida com a forma como os dados relacionais são mapeados para objetos Java e como esses objetos são armazenados em um banco de dados relacional, para que estes possam ser posteriormente acessados. Lida também com a continuação da existência do estado de uma entidade mesmo após o término de um aplicativo. Além de simplificar o modelo de persistência de uma entidade o JPA padroniza o mapeamento objeto-relacional. O JPA é baseado nos principais *frameworks* de persistência e API's como Hibernate, TopLink e o Java Data Objects - JDO.

Esta especificação consiste de quatro partes:

- A API Java de Persistência;
- A linguagem query;
- A API Critéria Java de Persistência;
- Os metadados para mapeamentos objeto/relacional.

O JPA está contido no pacote javax.persistence e utiliza como

linguagem para suas sentenças a *Java Persistence query language* - JPQL. Neste trabalho foi utilizada sua versão 2.0 que é descrita pela JSR 317. Entre as propostas desta versão estão: a expansão do mapeamento objeto/relacional; alguns adicionais para o JPQL; contrato adicional para entidades e extensão dos contextos de persistência; suporte para validação com funcionamento do JSR 303.

A Figura 7 mostra como é feito o acesso aos dados utilizando o JPA. Pode-se notar que é necessário escolher um provedor que irá implementar a especificação JPA, como Hibernate ou TopLink por exemplo. O provedor irá lidar com o *Java Database Connection*-JDBC que através de um *driver* irá acessar um determinado Sistema Gerenciador de Base de Dados - SGBD. Como exemplo de SGBD's, pode-se citar o PostgreSQL, MySQL, DB2, etc.

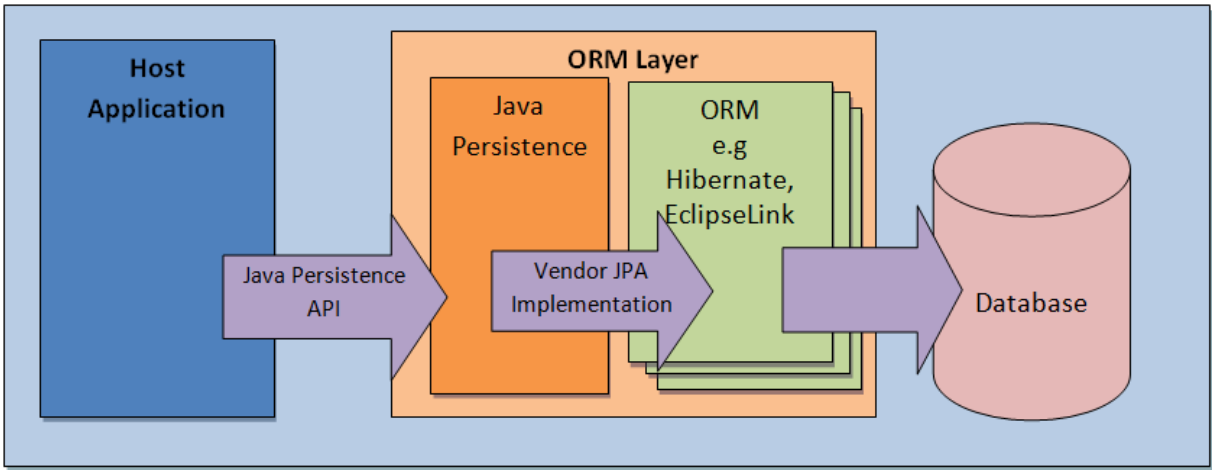


Figura 7 Aplicação utilizando JPA

Fonte: http://www.calcey.com/wp-content/uploads/2014/01/jpa-story-21.png

2.4.5 HIBERNATE

O Hibernate é um *framework* para o mapeamento objeto-relacional escrito na linguagem Java, mas também é disponível em .Net como o nome NHibernate. Este *framework* facilita o mapeamento dos atributos entre uma base tradicional de dados relacionais e o modelo objeto de uma aplicação, mediante o uso de arquivos (XML) ou anotações Java.

2.4.6 jQuery

jQuery é uma biblioteca de código aberto cuja sintaxe foi desenvolvida para tornar mais simples a navegação do documento HTML, a seleção de elementos DOM, criar animações, manipular eventos e desenvolver aplicações AJAX. A biblioteca também oferece a possibilidade de criação de plug-ins sobre ela. Fazendo uso de tais facilidades, os desenvolvedores podem criar camadas de abstração para interações de mais baixo nível, simplificando o desenvolvimento de aplicações Web dinâmicas de grande complexidade.

Principais funcionalidades do jQuery:

- Resolução da incompatibilidade entre os navegadores;
- Redução de código;
- Reutilização do código através de plug-ins;
- Utilização de uma vasta quantidade de plug-ins criados por outros desenvolvedores;
- Trabalha com AJAX e DOM;
- Implementação segura de recursos do CSS1, CSS2 e CSS3.

2.4.7 Tiles

Apache Tiles é um *framework* de composição de modelo de layouts. O Tiles foi originalmente criado para simplificar o desenvolvimento de interfaces de aplicativos Web em Java, mas não está mais restrito ao ambiente Web Java EE.

Tiles permite aos desenvolvedores definir fragmentos de páginas que podem ser montados como uma única página completa em tempo de execução. Estes fragmentos, ou tiles, pode ser utilizado como simples "*includes*" para reduzir a duplicação de elementos comuns entre as páginas ou incorporado dentro de outros tiles para desenvolver uma série de modelos reutilizáveis. Esses modelos agilizam o desenvolvimento, permitindo ter uma aparência consistente na aplicação como um todo.

2.4.8 Maven

O Apache Maven é uma poderosa ferramenta utilizada para gerenciar projetos Java. O Maven fornece todo o controle de compilação da aplicação, controle de bibliotecas, *deployment* e relatórios estatísticos. A configuração do Maven se baseia em um arquivo chamado pom.xml (*Project Object Model*), onde são declaradas todas as dependências do projeto. Depois de feita a configuração, o Maven se encarrega de analisar as dependências declaradas, fazer o download de todas as elas a partir de um repositório, e utilizá-las para compilar, empacotar e distribuir o artefato que pode ser um JAR, WAR ou EAR.

Algumas características

- Facilitar e unificar o processo de *build*;
- Fornecer mais qualidade de informação sobre o projeto;
- Permitir transparente migração para novas versões;
- Ajudar com boas práticas no desenvolvimento.

2.4.9 Git

Git é um sistema de controle de versão distribuído e um sistema de gerenciamento de código fonte, com ênfase em velocidade. O Git foi inicialmente projetado e desenvolvido por Linus Torvalds para o desenvolvimento do *kernel* Linux, mas foi adotado por muitos outros projetos.

Normalmente a maioria dos controles de versão guardam as mudanças do código como alterações de um determinado arquivo. Ou seja, a cada mudança no arquivo, o sistema guarda essa mudança apenas e não o arquivo inteiro.

O Git pensa um pouco diferente: ele trata os dados como *snapshots*. Cada vez que commitamos (commitar é enviar alterações para o controle de versão) ou salva o estado do projeto no Git, ele basicamente guarda um *snapshot* de como todos os arquivos estão naquele momento e guarda a referência desse estado. Para os arquivos que não foram modificados, ele não guarda uma nova versão, ele apenas faz um link para a versão anterior idêntica que já foi guardada em outro

momento.

2.4.10 JasperReports e IReport

O JasperReports é um *framework* para a geração de relatórios. É uma ferramenta totalmente *open source* e gratuita, e a mais utilizada com esse propósito atualmente. Entre as funcionalidades do JasperReports pode-se destacar:

- É capaz de exportar relatórios para diversos formatos diferentes, tais como PDF, HTML, XML, XLS, etc;
- Aceita diversas formas de entrada de dados, tais como um arquivo XML ou CSV, conexão com o banco de dados, uma sessão do Hibernate, uma coleção de objetos em memória, etc;
- Permite o uso de diagramas, gráficos, e até códigos de barras.

Um aspecto importante do JasperReports é que o layout do relatório é definido em um arquivo XML, geralmente com a extensão .jrxml. Este XML possui todas as informações de formatação do relatório, e além disso, possui os campos que serão preenchidos posteriormente, de acordo com a fonte de dados utilizada (*data source*). Como dito anteriormente, a fonte de dados pode variar, e ser uma tabela em uma base de dados, ou ser um arquivo CSV, porém a formatação do relatório será a mesma em ambos os casos.

Os passos para gerar um relatório são bem simples. O primeiro passo é compilar o relatório em XML. Depois da compilação, o resultado é um objeto do tipo JasperReport. O próximo passo é preencher o relatório com os dados, e o resultado dessa etapa fica armazenado em um objeto do tipo JasperPrint. Esse objeto já representa o relatório finalizado, e a partir dele pode-se enviar para impressão diretamente, ou exportar para um outro formato, tal como PDF por exemplo. Veja na figura 8 um diagrama ilustrando o processo completo:

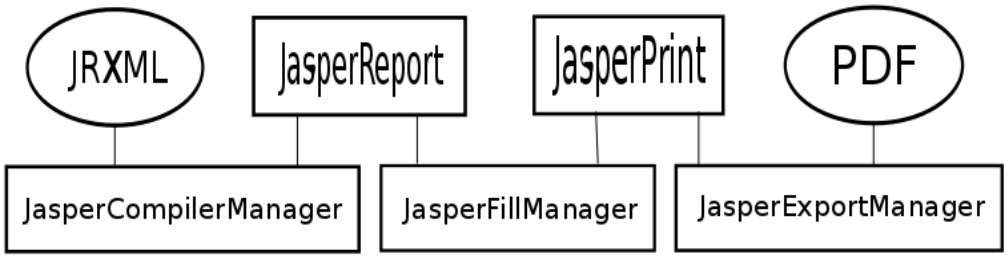


Figura 8 Processo de exportação para formato PDF

Fonte: http://www.k19.com.br/artigos/wp-content/uploads/2010/11/diagrama.png

O iReport é uma ferramenta desenvolvida pela mesma empresa do JasperReports, a JasperForge, e por isso é muito comum ver os dois sendo usados em conjunto. Uma das dificuldades ao trabalhar com os relatórios, está na definição do layout. É complicado escrever o layout totalmente em XML, sem ter que se aprofundar em todas as *tags* e atributos possíveis, e além disso posicionar todos os elementos corretamente. Na prática, é muito raro alguém editar o JRXML manualmente, e sim apenas para fazer alguns pequenos ajustes quando necessários. O processo normal é utilizar alguma ferramenta para gerar o JRXML automaticamente, e o iReport é utilizado com esse propósito.

O iReport é um aplicativo gráfico, que permite que você “desenhe” um relatório, utilizando uma palheta, e arrastando e soltando componentes, de forma bem parecida com a criação de interfaces e janelas para programas. Ao salvar, automaticamente será gerado um JRXML que você poderá utilizar na aplicação que estiver desenvolvendo. A vantagem é que não é necessário que você conheça a fundo o XML a ser editado, economizando tempo de desenvolvimento. Ele também traz um conjunto pronto de modelos que você já pode utilizar diretamente, ou então, escrever seus próprios modelos e reaproveitá-los sempre que precisar criar um novo tipo de relatório.

3. ESTUDO DE CASO

3.1 DESCRIÇÃO DO PROBLEMA

Todo ano, centenas de instituições de assistência social de todo o Brasil, pertencentes a um movimento religioso nacional, reúnem-se em Brasília, no período da páscoa, durante três dias, para traçarem as metas de atividades que serão implantadas em suas cidades no próximo ano exercício.

Esse grupo, atua na disseminação e na propagação da Promoção Social Espírita, auxiliando a todos as instituições que participam desse evento a se organizarem como instituições referenciais no atendimento de crianças, jovens e adultos, na prevenção e tratamento de vícios e apoio a família.

Essas atividades são divididas em comissões de trabalho, agrupadas de acordo com a finalidade e/ou público alvo. Assim existem comissões específicas para infância, juventude, caridade, esclarecimento, dentre outras.

Cada atividade, possui uma metodologia para aplicação, material de apoio e objetivo específico, proposto e adaptado de acordo com o resultado obtido em anos anteriores pelas instituições que já implantaram a mesma atividade através da troca de experiências.

O processo no qual cada instituição escolhe as suas metas de implantação para o ano é chamado rodízio. O rodízio tem esse nome devido ao formato que é utilizado a mais de dez anos, onde cada comissão oferece facilitadores, que são pessoas com conhecimento avançado e específico dessa comissão, que auxiliam as instituições interessadas na implantação das atividades em suas dúvidas sobre como implantar, ou como se planejar para uma implantação parcial ou futura. Hoje esses facilitadores se dividem em seis grupos por comissão, capazes de atender até doze instituições/cidades por período. No total são catorze comissões, divididas em 28 salas, cada sala com 3 grupos, num total de 84 grupos. Com capacidade de atendimento efetivo de 168 instituições por período.

No rodízio cada período é composto por 12 minutos. Cada cidade presente com suas instituições é direcionada a um grupo de facilitadores. Ocupando

assim todos os 84 grupos presentes. Ao dar início ao rodízio, as instituições começam a preencher suas fichas com as atividades que querem implantar em suas cidades. A quantidade de atividades disponíveis por comissão, variam de 15 à 90 atividades aproximadamente. A cada término de período, as instituições são direcionadas para a troca de facilitadores para poder contratar as atividades de outra comissão e assim se repete até que se tenha passado por facilitadores das catorze comissões.

Hoje, todo processo é feito em folhas carbonadas (vide Anexo IV), onde ao término de cada período a instituição deixa uma cópia com a o facilitador e leva o original consigo. A instituição decide qual atividade irá implantar, quando irá implantar e se será completa ou parcialmente.

Como o passar dos anos, o número de instituições tem aumentado consideravelmente, chegando hoje perto de 300 instituições. Devido a dificuldades de local, que não permite que seja expandido o número de facilitadores, e de tempo, que não permite que se estenda o tempo de atendimento, o processo acaba sobrecarregando os facilitadores que em determinados períodos chegam a atender mais de 30 instituições, causando quase sempre um prejuízo na coleta das informações.

Outro ponto de dificuldade está na base histórica. Devido a quantidade de fichas guardadas, letras incompreensíveis, falhas nas cópias carbonadas, não é possível resgatar os dados, sendo estas fichas usadas apenas para orientação da própria instituição no sentido do que haveria de ser feito ou não. Mas o responsável por cada comissão a nível nacional, acaba não tendo controle sobre as atividades que foram contratadas, nem sobre quais estão implantadas ou não.

Convidado pelos organizadores do evento a avaliar a possibilidade de utilização das tecnologias disponíveis no mercado para melhorar o processo, planejamos e desenvolvemos o Sistema de Acompanhamento de Metas, que será descrito nos próximos capítulos.

3.2 JUSTIFICATIVA E OBJETIVOS

Diante da necessidade de ter controle direto sobre o andamento das atividades realizadas pelas instituições vinculadas ao esse movimento espírita, esse sistema se mostra como solução que contempla tanto a evolução na coleta das informações como no armazenamento, e posteriormente o seu uso, permitindo aos responsáveis ter acesso as informações estatísticas que auxiliem na tomada de decisões.

3.2.1 OBJETIVOS GERAIS

O Sistema de Acompanhamento de Metas, está divido em 2 módulos.

O módulo de contratação, que é utilizado na preparação e na execução das atividades do dia de rodízio, possui as funcionalidades de cadastro básico, e de contratação das atividades. Esse módulo é de uso dos organizadores do evento e dos facilitadores, que o utilizarão para determinar junto com os presidentes destas instituições, quais as atividades encontram-se implantadas em cada instituição, assim como sua situação, avaliar os motivos de não terem cumprido as metas do ano anterior e auxiliar com maior exatidão na contratação das metas para o próximo ano.

O segundo módulo, responsável pelo de acompanhamento, permite que os dirigentes locais possam acompanhar suas metas para cada mês por meio de relatórios, e também possam atualizar as informações sobre o cumprimento de metas, replanejamento ou cancelamento das mesmas. Os dirigentes nacionais podem acompanhar por relatórios os andamentos de cada comissão podendo ter uma visão por mês, ano, região e outros filtros disponíveis. Ambos podem ser alertados para metas que estão a vencer e que ainda não tiveram um posicionamento cadastrado.

3.2.2 OBJETIVOS ESPECÍFICOS

Para compreender adequadamente o escopo da solução são necessárias atividades de levantamento de requisitos, portanto foi realizado um levantamento preliminar, que caracteriza o núcleo do projeto (PRESSMAN, 2006).

O levantamento foi realizado por meio de entrevistas com os organizadores do rodízio. Foram levantadas as necessidades mais urgentes, e detalhadas as características específicas da aplicação. Estas informações foram relevantes para determinar o funcionamento do sistema, assim como mensurar a afinidade entre usuários e sugestão dos mesmos baseando-se em suas especificações de características.

Baseando-se na metodologia selecionada para o desenvolvimento não é possível mensurar todos os requisitos detalhadamente na etapa inicial, uma visão generalizada das principais funcionalidades foi documentada para que durante o processo as diversas iterações contemplassem o refinamento e revisão dos requisitos.

3.3. DESENVOLVIMENTO

Nos dias 18, 19 e 20 de abril de 2014, ocorreu em Brasília uma reunião com do grupo, onde se realizaram as primeiras discussões sobre a possibilidade da informatização do sistema de contratação de metas, chamado de “Rodízio”. Nesta data, aproveitando a presença dos envolvidos, foram realizadas entrevista com o presidente do Conselho, a coordenadora da secretaria, alguns facilitadores e alguns presidentes de entidades, os quais conforme citado no capítulo 2 são os envolvidos diretamente com o sistema. Com base nas informações levantadas, foi elaborado um escopo, onde foram definidas a lista de funcionalidades, que também compõe a proposta apresentada para este trabalho, alguns protótipos iniciais (que podem ser vistos no capitulo sobre as funcionalidades do sistema), e que foram apresentadas em reunião de aprovação em Brasília no dia 06 e 07 de julho de 2014, iniciando assim oficialmente o desenvolvimento do Sistema de Contratação de Metas.

Nesse período também foram definidos os Sprint necessários para a conclusão do sistema, a sua arquitetura e também um cronograma inicial para execução. Todos esses itens podem ser vistos na proposta.

Tendo o primeiro sprint sido planejado para início de maio de 2014.

3.3.1 Sprint 1 – (05/05 a 31/05)

Esse primeiro Sprint, foi o mais importante de todos, nele foram tomadas todas as decisões que nortearam a conclusão deste projeto.

Após o levantamento de requisitos, foram realizados os primeiros testes com a arquitetura. Por se tratar de uma instituição sem fins lucrativos, busca-se sempre o menor custo, para que não impacte nas atividades da instituição, sendo assim buscamos uma solução de arquitetura que tivesse um custo inicial baixo, no caso de nossa escolha gratuito, e que pudesse ser facilmente escalonada com o possível crescimento do sistema, a um investimento baixo.

Foram testados diversos serviços de PaaS (Plataforma como Serviço), todos com suas características especificas, mas naquele momento a sua facilidade de criação, versionamento, implantação e execução com apenas um clique (hoje a página que gerava a aplicação em um dos modelos bases com apenas um clique não está mais disponível, mas ainda é fácil criar um projeto), foi o principal fator que levou a escolhe da plataforma Heroku, como a plataforma base para construção do sistema. No capítulo onde fala-se sobre a arquitetura, é detalhado com maiores informações o funcionamento do Heroku e suas principais características. É disponibilizado no Anexo II um tutorial básico para criar sua primeira aplicação no Heroku, e um pequeno comparativo entre dois dos PaaS testados, está disponível no Anexo III.

Tendo sido escolhida a plataforma, passou-se para a escolha da linguagem. A linguagem Java é extremamente poderosa, assim como sua concorrente direta no mercado corporativo, o .Net. Apesar das diversas vantagens conhecidas, o Java ainda sofre críticas quando se colocam em discussão, a infraestrutura necessária para se colocar uma aplicação em funcionamento comercialmente com uma velocidade satisfatória. Como o serviço de PaaS abstrai toda a parte de infraestrutura de nosso desenvolvimento, conforme explicado no capítulo 2.2.2.2, não encontramos desvantagens na utilização da linguagem Java.

O próximo passo foi a escolha dos *frameworks* de desenvolvimento a serem utilizados. Desde o início pensamos na utilização de um *framework* MVC, com o objetivo de diminuir o impacto da API de Servlets em nosso trabalho e permitir que passemos a nos preocupar exclusivamente com a lógica de negócios, que é o

código que possui valor para a aplicação. Um dos *frameworks* MVC para Java mais famosos no mercado é o Spring MVC. O Spring é um *framework* que inicialmente não foi criado para o desenvolvimento Web. Na sua essência o Spring é um *container* leve que visa fornecer serviços para sua aplicação como por exemplo o gerenciamento de objetos ou transações. Mas com o tempo a comunidade Spring começou criar um *framework* MVC próprio. O Spring MVC é um *framework* moderno que usa os recursos atuais da linguagem além de usar todo poder do *container* Spring. Outro ponto interessante na escolha do Spring MVC foi devido o baixo acoplamento entre a camada de visão e a camada de controle, pois assim, permite facilmente integrar, por exemplo, um aplicativo para celular, que acesse diretamente a mesma camada de controle utilizada pela camada de visão, com poucas ou até nenhuma intervenção em código.

Aproveitando o uso do Spring MVC, já incorporamos o uso da Injeção de Dependência e a Inversão de Controle do próprio Spring. Para persistência escolhemos o uso de JPA que é a especificação oficial da Sun, agora Oracle para persistência de dados com Java. Através da JPA, o programador evita o uso excessivo de códigos SQL em seus sistemas.

Todo o trabalho do programador, tratando-se de persistência, é executado por meio de interfaces implementadas e baseadas na especificação JPA.

O próprio Heroku fornece em seus *templates* um projeto básico de cadastro de "*Person*" com inclusão, exclusão e listagem, utilizando Spring MVC e JPA. Este *template* foi aproveitado com estrutura base desse sistema.

Algumas das tecnologias a própria arquitetura já nos entregou definidas, como o uso de Git para versionamento de arquivos; o uso do Maven para gerenciamento de bibliotecas e *deploy* da aplicação; o *Web Container* Tomcat, para servir as páginas e aplicativos JSP/Servlet e o banco de dados PostgreSQL já disponível por padrão na conta básica.

Todas essas e demais tecnologias, são detalhadas em capítulo próprio na seção 2.4.

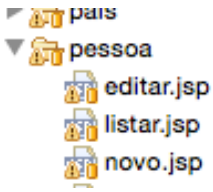
Com todas as tecnologias definidas, realizamos a configuração do pom.xml gerado em nossa aplicação Heroku, para que o maven realizasse o gerenciamento e downloads de todas as bibliotecas necessárias. A partir de então pegamos duas funcionalidades para servirem de POC (*Proof of Concept*) ou prova

de conceito, no qual criamos as funcionalidades de importação Caravaneiros (Pessoas) e Entidades.

Para modelagem da base de dados foram utilizadas as informações coletadas nas entrevistas e nas fichas modelo entregues pelo cliente. A base foi modelada através da criação das classes e relacionamentos utilizando as anotações de JPA e após a primeira execução local, pegou o script DDL gerado pelo JPA e utilizando a ferramenta MySQLWorkBench, foi feito os devidos ajustes no modelo e implantado novamente na base de dados.

Nessa etapa foi definido então a arquitetura base de todos os CRUDs que seriam construídos no sistema, criando uma padrão que permitiu facilmente criar os casos de usos rapidamente:

- Neste padrão cada funcionalidade tem uma pasta com o nome chave do modelo a ser trabalhado exemplo: Cadastro de Pessoas, pasta “pessoa”. Nessa pasta são colocados então três arquivos base : novo.jsp, editar.jsp, listar.jsp.



- Três novos mapeamentos são incluídos no arquivo tiles.xml (ver sobre a Tiles no capítulo sobre as tecnologias utilizadas).

```
  <definition name="pessoa.listar" extends="base.definition">
    <put-attribute name="title" value="Metas :: Caravaneiros :: Listar" />
    <put-attribute name="header" value="Cadastro de Caravaneiros" />
    <put-attribute name="bodyPage" value="/WEB-INF/jsp/pessoa/listar.jsp" />
</definition>

  <definition name="pessoa.novo" extends="base.definition">
    <put-attribute name="title" value="Metas :: Caravaneiros :: Novo" />
    <put-attribute name="header" value="Novo Caravaneiro" />
    <put-attribute name="bodyPage" value="/WEB-INF/jsp/pessoa/novo.jsp" />
</definition>

  <definition name="pessoa.editar" extends="base.definition">
    <put-attribute name="title" value="Metas :: Caravaneiro :: Editar" />
    <put-attribute name="header" value="Alterar Caravaneiro" />
    <put-attribute name="bodyPage" value="/WEB-INF/jsp/pessoa/editar.jsp" />
</definition>
```

- Uma interface de Serviço é criada com os métodos de CRUD.

```
import java.util.List;

public interface PessoaService {
    public void addPessoa(Pessoa pessoa);
    public void updatePessoa(Pessoa pessoa);
    public Pessoa getPessoa(Integer id);
    public List<Pessoa> listPessoa();
    public void removePessoa(Integer id);
}
```

- Uma implementação base desse Serviço era criada, sendo anotada com @Service, e recuperando o EntityManager da JPA pela anotação @PersistenceContext. Todos os métodos foram anotados com @Transactional, e já transferiam a responsabilidade de abertura e fechamento da transação para o *container*.

```
@Service
public class PessoaServiceImpl implements PessoaService {

    @PersistenceContext
    EntityManager em;

    @Transactional
    public void addPessoa(Pessoa pessoa) {
        em.persist(pessoa);
    }

    @Transactional
    public void updatePessoa(Pessoa pessoa) {
        em.merge(pessoa);
    }

    @Transactional
    public List<Pessoa> listPessoa() {
        CriteriaQuery<Pessoa> c = em.getCriteriaBuilder().createQuery(Pessoa.class);
        c.from(Pessoa.class);
        List<Pessoa> retorno = em.createQuery(c).getResultList();
        return retorno;
    }

    @Transactional
    public void removePessoa(Integer id) {
        Pessoa person = em.find(Pessoa.class, id);
        if (null != person) {
            em.remove(person);
        }
    }

    @Transactional
    public Pessoa getPessoa(Integer id){
        Pessoa person = em.find(Pessoa.class, id);
        return person;
    }
}
```

- Da mesma forma, definiu-se uma classe de controle base:

```java
@Controller
@RequestMapping("pessoa")
public class PessoaController {

    @Autowired
    private PessoaService pessoaService;

    @RequestMapping("/")
    public String listPeople(Map<String, Object> map) {
        map.put("pessoaList", pessoaService.listPessoa());
        return "pessoa.listar";
    }


    @RequestMapping(value = "/add", method = RequestMethod.GET)
    public String addPerson(Map<String, Object> map) {
        Pessoa nova = new Pessoa();
        map.put("pessoa", nova);
        return "pessoa.novo";
    }

    @RequestMapping(value = "/add", method = RequestMethod.POST)
    public String addPerson(@ModelAttribute("pessoa") Pessoa pessoa,
            BindingResult result) {
        pessoaService.addPessoa(pessoa);
        return "redirect:/services/pessoa/";
    }

    @RequestMapping(value="/edit/{pessoaId}", method=RequestMethod.POST)
    public String editPessoa(Map<String, Object> map, @PathVariable("pessoaId") Integer pessoaId) {
        map.put("pessoa", pessoaService.getPessoa(pessoaId));
        return "pessoa.editar";
    }


    @RequestMapping(value="/edit/save/{pessoaId}", method=RequestMethod.POST)
    public String editPessoa(@ModelAttribute("pessoa") Pessoa pessoa, @PathVariable("pessoaId") Intege
        pessoaService.updatePessoa(pessoa);
        return "redirect:/services/pessoa/";
    }

    @RequestMapping("/delete/{personId}")
    public String deletePerson(@PathVariable("personId") Integer personId) {
        pessoaService.removePessoa(personId);
        return "redirect:/services/pessoa/";
    }
}
```

Esse padrão foi seguido para todas as funcionalidades do sistema, tornando rápido a geração de novos CRUDs. Optou-se por não utilizar reflexão nessa etapa o que teria facilitado ainda mais o processo de construção dos Services, para facilitar a posterior manutenção do código, já que por experiência, aplicativos com reflexão mal documentados se tornam extremamente difíceis de ajustar sem impactos.

Nas funcionalidades de Pessoa e Entidade, as funcionalidades padrões não foram implementadas, já que não estavam no escopo desta versão do sistema. Apenas uma funcionalidade de upload de arquivos excel e importação na

base de dados foi desenvolvida e testada para ambas as funcionalidades conforme previsto.

Havia uma reunião com o cliente agendada para 03 e 04 de junho onde seriam apresentados os protótipos e as lista das funcionalidades, mas ele desmarcou devido a problemas de agenda. Uma nova reunião foi marcada para 07 e 08 de junho de 2014.

3.3.2 Sprint 2 – (09/06 a 04/07)

A primeira Sprint acabou tendo um atraso de uma semana para sua conclusão, e a espera pela reunião com o cliente no dia 07 e 08 de junho, acabaram atrasando o início da segunda Sprint. A reunião novamente foi remarcada, dessa vez para 05 e 06 de julho, mas por meio de conversas com a coordenadora da secretaria (cliente), optou-se por dar continuidade às atividades antes da reunião.

Nessa Sprint, foram implementadas as funcionalidades de Cadastro de Institutos, de Comissões e de Atividade, com base na estrutura padrão definida na primeira Sprint.

No dia 06 e 07 de julho, houve nova reunão com o cliente em Brasília. Foram apresentadas as funcionalidades do sistema, e diversos questionamentos foram feitos por todos os presentes, muitos viraram solicitações de ajustes.

Dentre os pontos levantados, dois tiveram grande impacto: o primeiro foi a implementação das funcionalidades de Cadastro de Pessoas e Entidades.

No início do projeto, as funcionalidades de Cadastro de Pessoas e Entidades, não teriam interfaces para cadastro, alteração ou exclusão, e sim apenas uma interface de importação cada, que nesse ponto já estavam implementas. Estas seriam utilizadas para carregar os dados de um sistema já existente. O cliente em contato com responsável pelo sistema já existente, chegou a conclusão que os dados dele possuem integridade e confiabilidade, e então decidiu que não poderiam ser utilizadas em nosso sistema de metas, optando assim pela implementação das interfaces completas de Cadastro de Pessoas e Entidades.

O segundo ponto foi a inclusão da funcionalidade de importação das

atividades de cada comissão/institutos, já que os coordenadores das mesmas, já possuem essas informações em planilhas.

Houve também uma mudança em relação a implantação do aplicativo, por parte do cliente, postergando para março de 2015 ao invés de novembro de 2014. Para novembro de 2014, pediram um teste funcional da interface de contratação de metas com usuários reais. Nova reunião para apresentação dos artefatos foi marcada então para a primeira semana de setembro.

3.3.3 Sprint 3 – (07/07 a 26/07)

Nessa Sprint foi incorporadas todas as alterações solicitadas pelo cliente em sua última reunião. Assim foram implementadas e geradas as interfaces do cadastro de Pessoas e de Entidades, assim como a funcionalidade de importação de Atividades. Para a construção do cadastro de Pessoas e Entidades, foi preciso a construção das funcionalidade auxiliares de Cadastro de País, Cadastro de Estado, Cadastro de Cidade, todas geradas utilizando a estrutura padrão definida no início do projeto. Também desenvolvidas os Cadastro de Plano de Metas e de Rodízio. Alterações foram realizadas na base de dados para abranger os dados adicionais das atividades, de pessoas e entidades.

No dia 25 de julho, a proposta foi apresentada na UTFPR Campus Cornélio Procópio, sendo aprovada e permitindo dar continuidade no projeto para validação do Curso de Análise e Desenvolvimento de Sistemas.

3.3.4 Sprint 4 – (25/08 a 24/10)

Na Sprint 4 foi prevista e realizada a construção da principal funcionalidade do sistema que é a contratação das metas. Por sua importância optou-se por manter o cliente o mais próximo possível do desenvolvimento para que a funcionalidade atendesse plenamente a necessidade dele. Foi esta, a funcionalidade com maior desgaste e retrabalho, tendo sido implementada e corrigida diversas vezes.

Um dos problemas localizados, logo no início da implementação desta tela, foi não ter previsto uma funcionalidade para cadastro de facilitadores.

Assim esta nova funcionalidade foi implementada seguindo a estrutura padrão. Sem impactar no processo.

Nessa fase, para que o cliente pudesse ter acesso ao sistema, foi implementada uma das funcionalidades de *Login* que estava prevista para a Sprint 5. Foi desenvolvida conforme o previsto usando o *framework* Spring Security para gerenciamento dos usuários, senhas e autorizações. Também começamos a testar todos os recursos diretamente no Heroku para garantir o funcionamento remoto igual ao funcionamento local. Nesse momento começamos a encontrar os primeiros problemas com a plataforma Heroku, relacionados a uso excessivo de memória que não permitiu que fosse possível importar algumas planilhas de atividades. Apesar desses problemas a base remota foi populada com os dados fictícios.

Dessa forma, o cliente tinha acesso a funcionalidade de contratação de metas, publicada diretamente no Heroku, quase em tempo real, e era possível durante as conferências por telefone que pequenos ajustes fossem sendo realizados enquanto ele testava a funcionalidade.

Essa funcionalidade mudou na sua estrutura, sendo desmembrada de uma para quatro interfaces. Em sua navegabilidade e usabilidade, passou por diversas modificações até chegar ao formato que foi aprovado pelo cliente.

Na reunião da primeira semana de setembro, as telas já desenvolvidas do Sprint 2 e 3 foram passadas para aprovação do cliente o que aconteceu sem grandes problemas, apenas adaptando rótulos e tamanhos de campos.

Após a reunião de setembro o cliente passou a enviar as planilhas de atividades reais para serem cadastradas no sistema.

No início de outubro, também recebemos do cliente as primeiras planilhas com informações de entidades para serem carregadas no sistema.

Devido ao tempo gasto com a funcionalidade, optou por retirar dessa Sprint a parte de acompanhamento e os Relatórios, deixando para a Sprint 6.

3.3.5 Sprint 5 – (01/12 a 24/01)

A Sprint 5 começou com grandes mudanças. Como estava previsto,

no dia 15 e 16 de novembro em Brasília, as funcionalidades prontas foram colocadas em teste para execução da contratação de metas por usuários finais do sistema. Isso gerou vários apontamentos de mudanças que devem ser implementadas no sistema.

A primeira mudança significativa no escopo, foi a possibilidade de que o acesso ao sistema seja feito de duas maneiras distintas. Na primeira, o administrador cadastra usuários com um login e senha, e estes acessam o sistema normalmente. Na segunda, qualquer pessoa que tenha o CPF e email vinculado a uma entidade, tendo um cargo de presidente ou coordenador, devem ter acesso ao sistema, e mediante o complemento do seu cadastro, automaticamente se tornar um usuário válido para o sistema, sem a interferência do administrador.

Para atender a solicitação, nessa Sprint implementamos o cadastro de usuários, que antes estava configurado de forma estática nos arquivos de configuração do Spring Security. Também criamos um AutheticationProvider para buscar nesse cadastro e outro AutheticationProvider para buscar na base de Pessoas e Entidades a partir do CPF/email informados.

A funcionalidade de *Login* foi alterada, e as permissões de acesso foram feitas. Foi implementada uma classe AuthenticationSuccessHandler, para redirecionar o usuário de acordo com seu nível de acesso para uma página inicial diferente.

As funcionalidades de cadastro de Links foram removidas do escopo da atividade. Permanecendo o acesso as páginas através das permissões já definidas, utilizando para implementação do controle de acesso nas páginas jsps, as tags do Spring Security.

Outro ponto revisto na reunião foi as entregas, assim mudamos as atividades da Sprint seguinte e adicionamos uma nova.

Houve uma reunião com o cliente no dia 15 de fevereiro. Nessa reunião foram reavaliadas as alterações da Sprint 5.

3.3.6 Sprint 6 – 18/02 a 04/03

A Sprint 6, contemplou a criação da tela de acompanhamento que faz parte da funcionalidade de Contratação de Metas, assim como as funcionalidades de Relatórios. Um relatório das entidades que participaram do rodízio, já encontra-se disponível. Um gráfico com informações sobre o cumprimento das metas agrupado por região, também já está disponível. Um gráfico para acompanhamento do presidente, sobre suas metas também já está disponível.

Os primeiros testes com relatórios foram feitos na Sprint 4, e por conta do uso da conta gratuita, acabou exaurindo os recursos oferecidos. Estuda-se a possibilidade de troca do fornecedor de PaaS.

3.3.7 Sprint 7 - Previsão 07/03 a 05/04

Na Sprint 7, serão feitos os ajustes finais, correções, testes integrados, implantação e que terá primeiro teste massivo na reunião nacional das entidades em Brasília no dia 05 de abril de 2015.

3.4 APLICAÇÃO - SISTEMA DE CONTRATAÇÃO DE METAS

Nos itens a seguir são apresentadas as funcionalidades do sistema. O cliente preferiu nesse momento deixar o sistema sem nenhuma identidade visual, que o ligasse ao grupo. Em um momento futuro o grupo irá modificar toda parte visual para atender essa necessidade.

3.4.1 SISTEMA DE LOGIN

O sistema de *Login* (Figura 9) foi desenvolvido utilizando Spring Security, facilitando assim a autenticação e a autorização dos usuários. Foram implementados dois AuthenticationProvider, uma para autenticação através da base usuários. E um para usuários novos previamente cadastrados no sistema, na base de pessoas.

Para o *login* na base de pessoas foi considerado o cadastro prévio do CPF e email para validação deste usuário, sendo o email o nome de usuário e o CPF a senha.

Dessa forma foi possível permitir que os dirigentes pudessem entrar no sistema, completar seus dados cadastrais, os dados da entidade que administram e cadastrar e delegar novos usuários para acessar as informações da entidade. Após esse cadastro prévio o usuário acessa apenas pelo seu usuário cadastrado no sistema. Esta funcionalidade não estava prevista no projeto inicial.

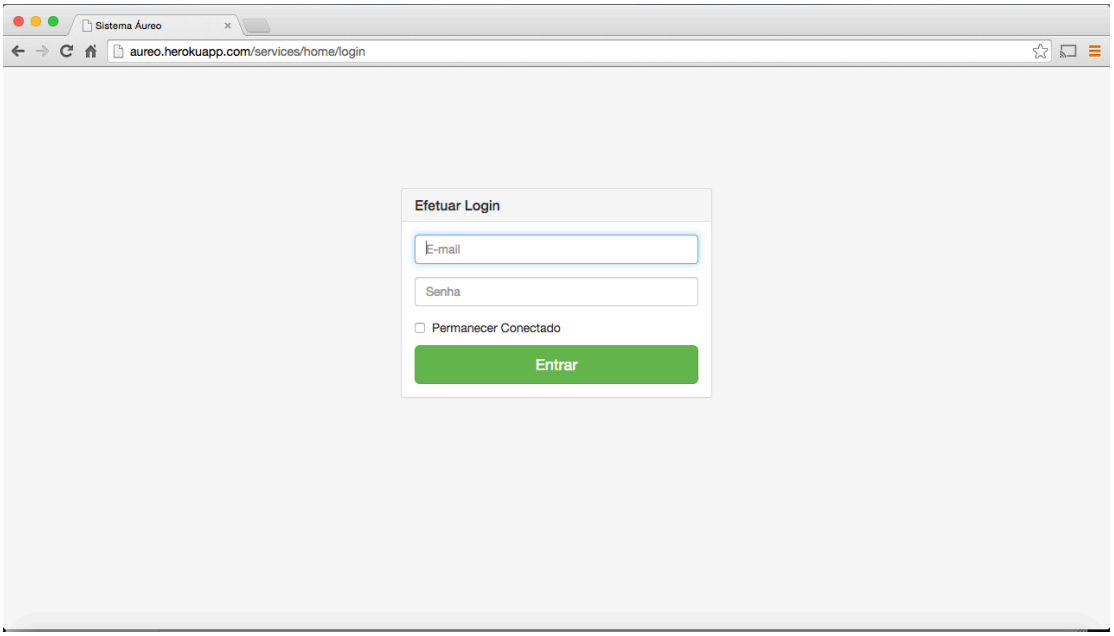


Figura 9 Tela de Login
A tela inicial não possui identidade visual da organização.

3.4.1.1 DESCRIÇÃO DOS USUÁRIOS DO SISTEMA

A dinâmica que envolverá o sistema abrange oito perfis de usuários, conforme Tabela 1.

PERFIL	*DESCRIÇÃO*
Facilitador	*Responsável por auxiliar o Dirigente Local/Presidente na contratação de metas. Tem acesso a tela de Contratação de Metas, apenas no dia do rodízio.*
Secretaria	*Responsável pelos cadastro básicos.*
Administrador	*Além dos acesso da Secretaria, tem também acesso aos módulos de permissões.*
Dirigente (Local)	*Tem visão das metas e relatórios da comissão de sua responsabilidade ao nível da sua instituição.*
Presidente (Local)	*Tem visão das metas e relatórios de todas as comissões de sua instituição. Tem acesso a alteração das metas de sua instituição.*
Coordenador Grupo Fraterno (Regional)	*Tem visão das metas e relatórios das as Instituições que compõem a sua região de atuação.*
Dirigente (Nacional)	*Tem visão das metas e relatórios da comissão de sua responsabilidade em âmbito nacional.*
Conselho	*Tem visão das metas e relatórios de todas as comissões em âmbito nacional.*

Tabela 1 Perfis de Usuários

Os últimos cinco níveis (Dirigente Local, Presidente, Coordenador

Grupo Fraterno, Dirigente Nacional e Conselho), terão acesso as mesmas funcionalidades, sendo limitado a abrangência dos dados disponíveis pra cada um.

3.4.2. Cadastro de Links/Funcionalidades

O Cadastro Links/Funcionalidades seria uma funcionalidade parte do sistema de segurança. Toda página (URL) não cadastrada, terá acesso para todo usuário logado, enquanto as páginas cadastradas teriam acesso apenas aos níveis (roles) definidas no momento do cadastro da URL. Também poderiam ser definidos níveis Leitura, Inclusão, Alteração e Exclusão e Relatório, que controlariam a exibição de cada botão referente a uma funcionalidade.

Esta funcionalidade foi retirada do escopo desta entrega, por ser considerada no momento não essencial pelo próprio cliente.

O controle de acesso existe mas é feito via validação em código.

3.4.3. Cadastro de Permissões

O Cadastro de Permissões, seria uma funcionalidade para cadastros dos níveis de acesso (*roles*), mas não foi implementada. Também devido ao tempo, optou-se por carregar os níveis já definidos no levantamento inicial, e não implementar as interfaces de inclusão, alteração e exclusão. A listagem, não possui interface, mas é utilizada pelo cadastro de usuários.

3.4.4. Cadastro de Rodízio

O cadastro de rodízio (Figuras de 10 a 12) é uma funcionalidade que marca o início de um novo processo de contratação de metas. Esse processo hoje é feito anualmente, então o rodízio é cadastrado informando a data em que será realizado, o ano a que se refere, e o período de ajustes. Foi implementado com as estrutura padrão

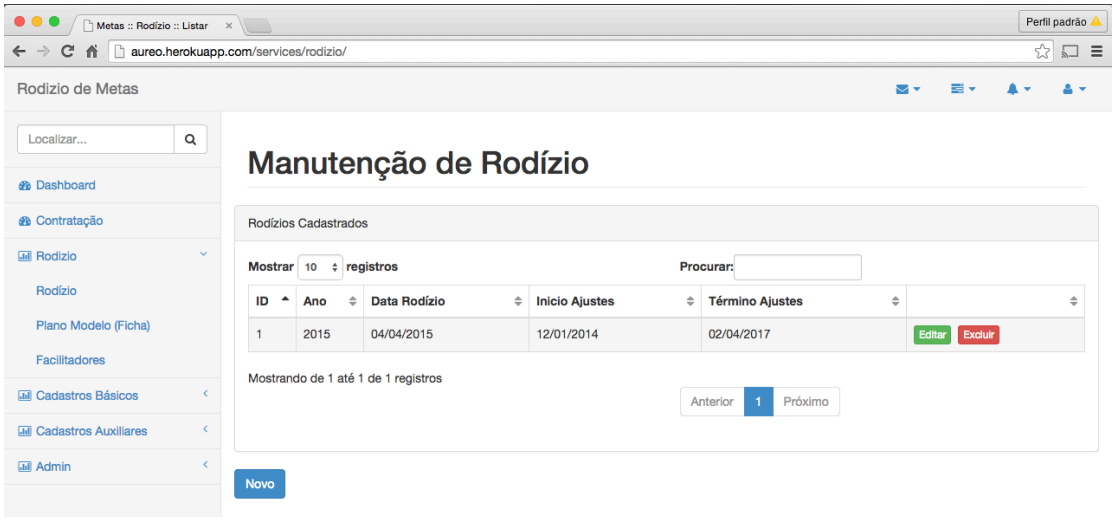


Figura 10 Tela de Manutenção de Rodízio

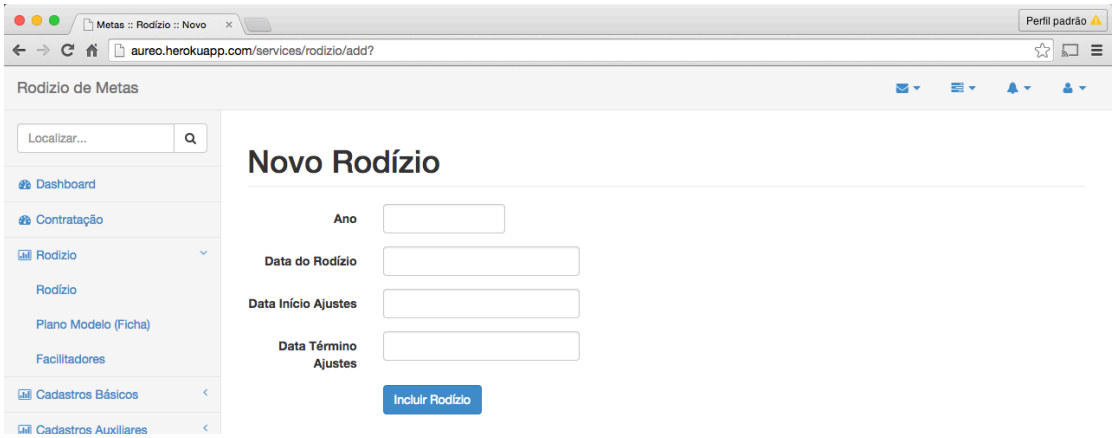


Figura 11 Tela Inclusão de rodízio

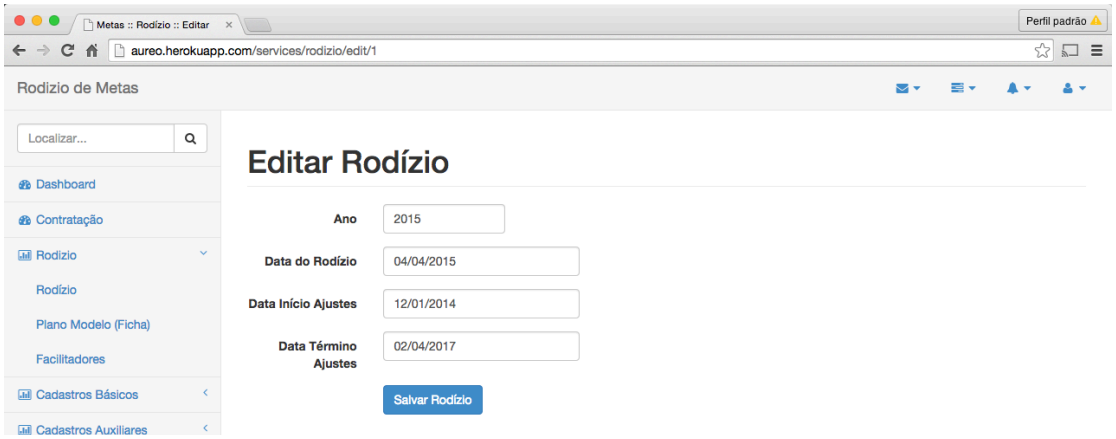


Figura 12 Tela de Edição de Rodízio

3.4.5. Cadastro de Institutos/Comissões

O cadastro de Institutos e comissões (Figuras de 13 a 18) serve

como cadastro básico para inclusão de “departamento”. Uma ou mais comissões pertencem a um instituto. Sendo assim necessário o cadastro do instituto antes da comissão.

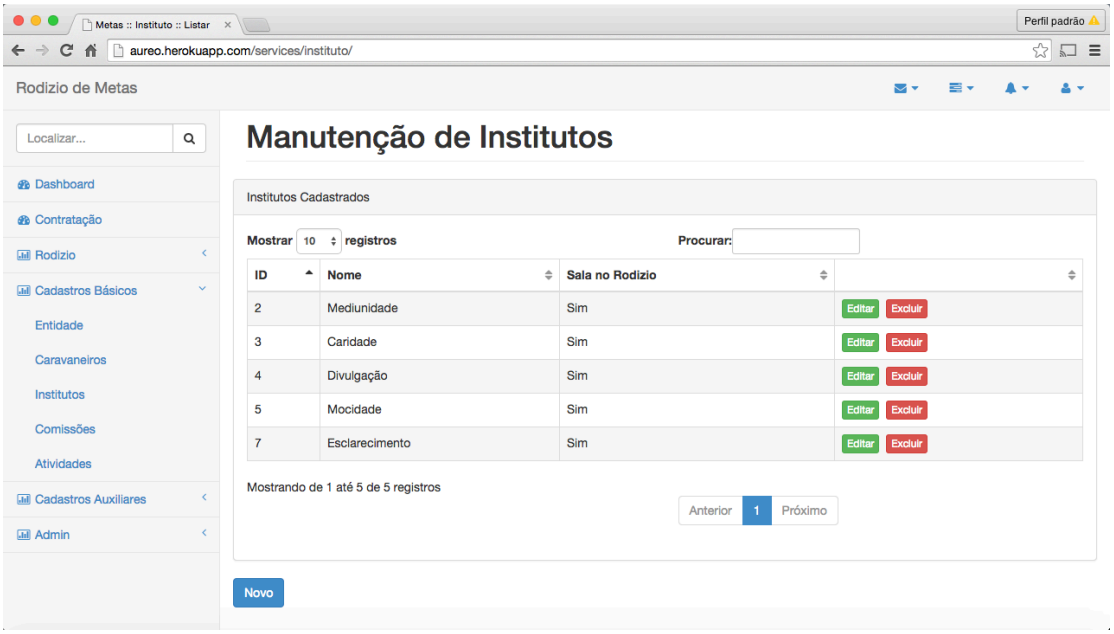


Figura 13 Tela de Listagem de Institutos

Figura 14 Tela Inclusão de Institutos

Figura 15 Tela Alteração de Institutos

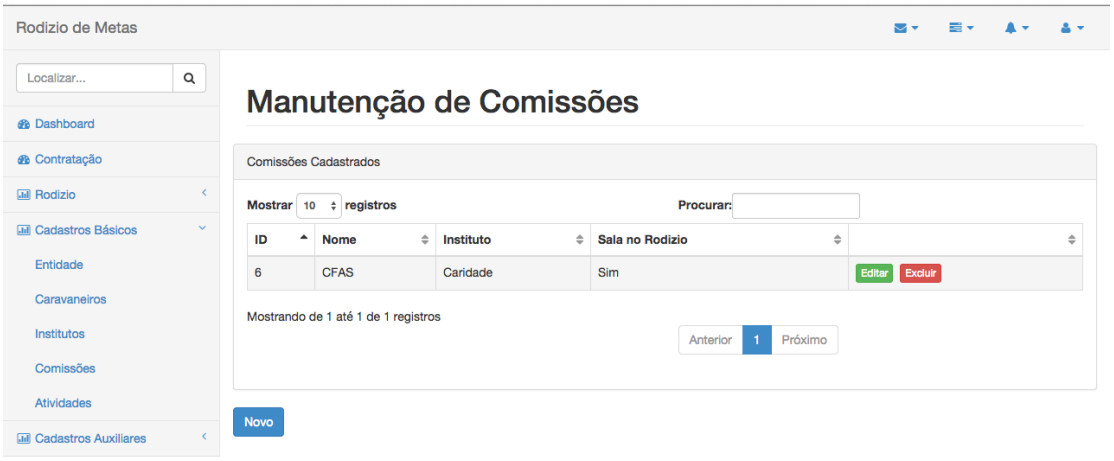


Figura 16 Tela de Listagem de Comissões

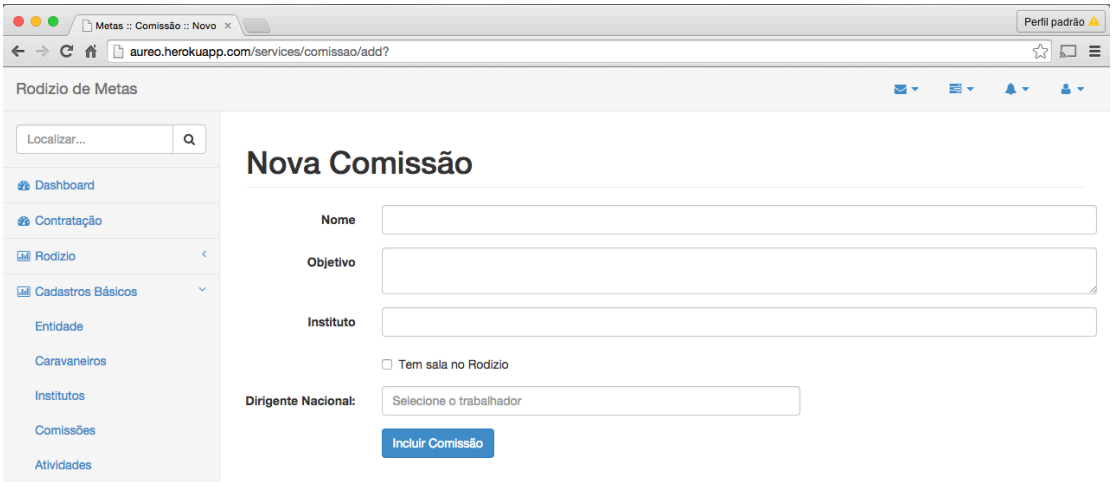


Figura 17 Tela de Inclusão de Comissões

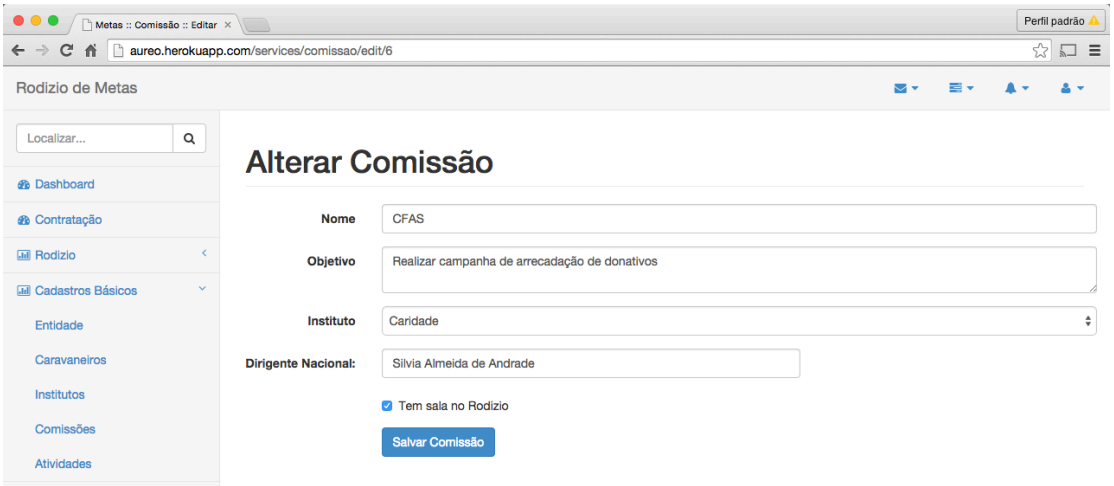


Figura 18 Tela de Alteração de Comissões

3.4.6. Cadastro de Atividades

As atividades compõem os itens que podem ou não ser classificados como metas. São classificados por institutos ou comissões. Para iniciar um cadastro de atividade é necessário antes selecionar a qual instituto/comissão ele pertencerá.

Assim, o cadastro de atividades (Figuras de 19 a 24) possui três (03) particularidades. A primeira é a possibilidade de importação de planilha excel contendo as atividades com suas informações. A segunda, é possível alterar todas as atividades de uma só vez, através do botão “Editar Todos”. A terceira, por ser uma tela em que a ordem das atividades influi, há um botão “Organizar” que permite através de *drag’n drop* movimentar ou agrupar as atividades.

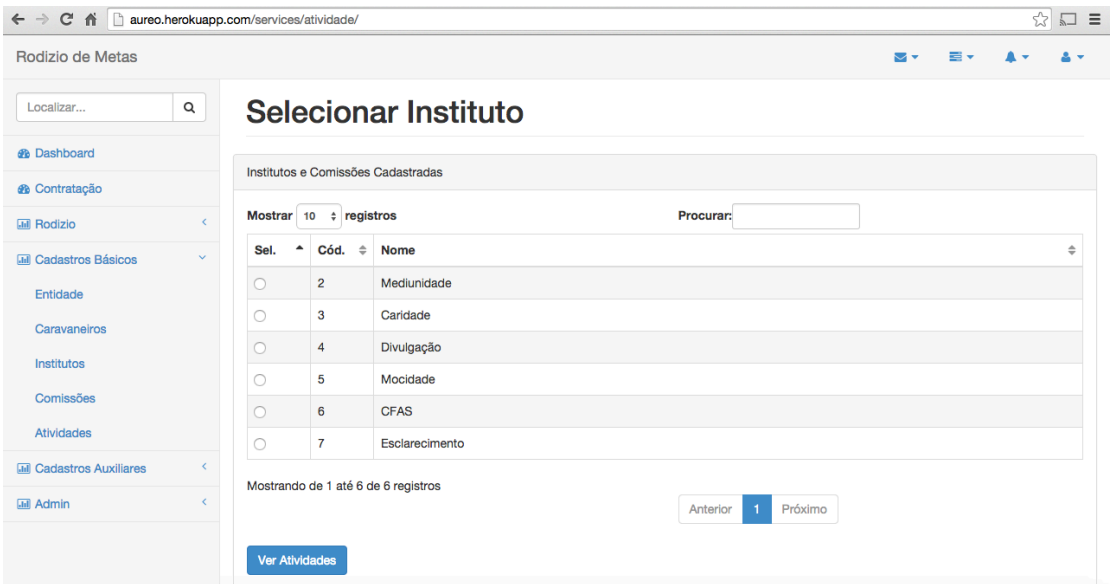


Figura 19 Tela de Seleção de Instituto/Comissão

Figura 20 Tela de Listagem de Atividade

Figura 21 Tela de Alteração de Atividade

Figura 22 Tela de Inclusão de Atividade

Figura 23 Tela de Edição Coletiva

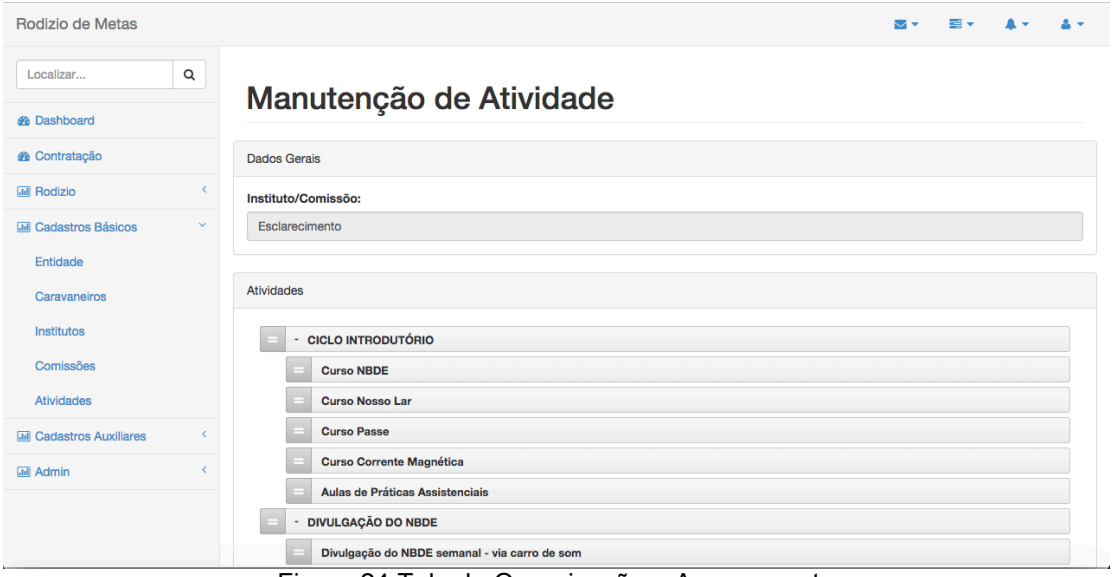


Figura 24 Tela de Organização e Agrupamento

3.4.7 Cadastro de Plano Modelo

Através do cadastro de Plano Modelo, os coordenadores podem escolher quais atividades estarão disponíveis para contratação em um determinado ano. É feito um plano modelo por rodízio por instituto/comissão.

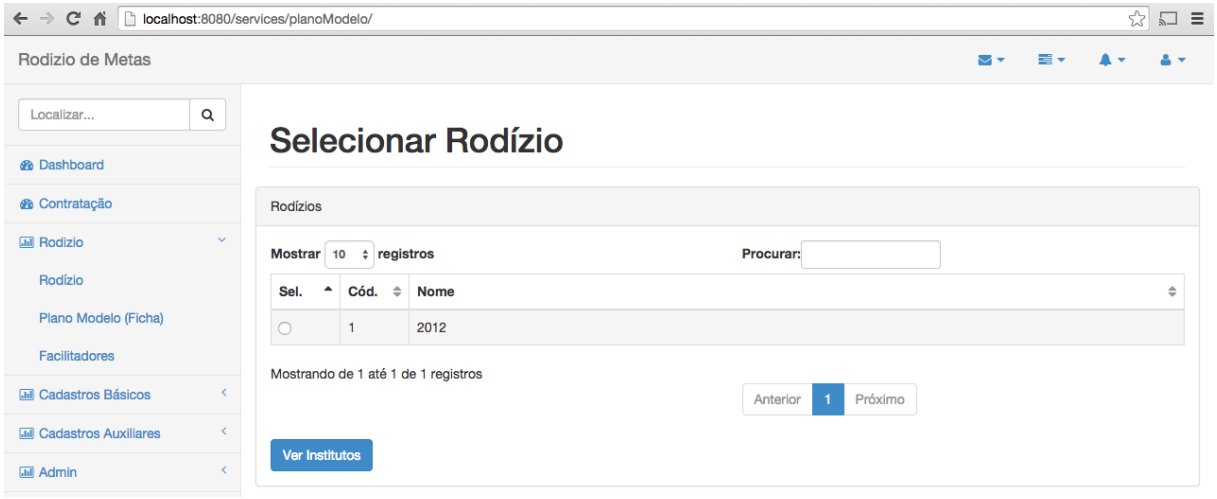


Figura 25 Seleção do Rodízio

Nesse momento o usuário seleciona o rodízio/ano e clica em “Ver Institutos”, conforme Figura 25.

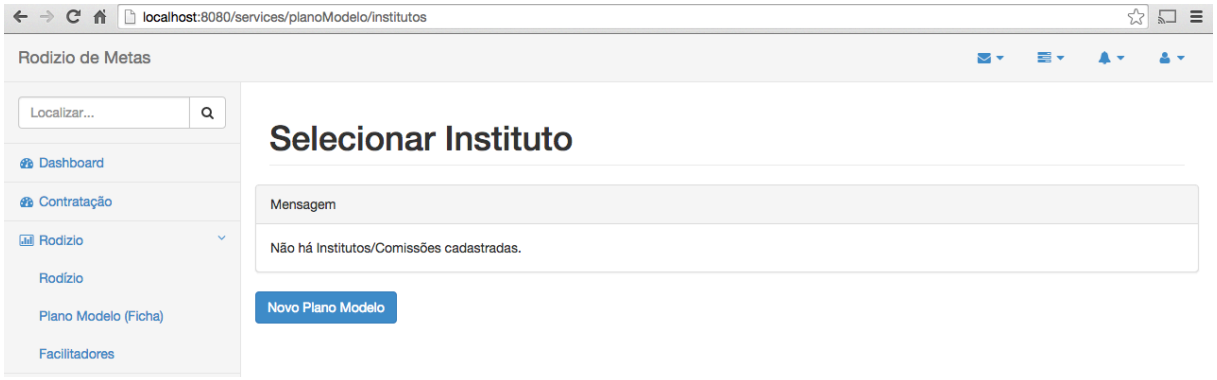


Figura 26 Seleção de Institutos

Caso haja algum instituto com plano modelos já cadastrados eles serão exibidos. Também é exibido o botão “Novo Plano Modelo”, conforme a Figura 26.

Figura 27 Novo Plano Modelo

Na tela seguinte (Figura 27) é possível selecionar o Instituto/Comissão e buscar através do botão “Puxar Atividades”, todas as atividades cadastradas no Instituto/Comissão.

Figura 28 Listagem de Atividades

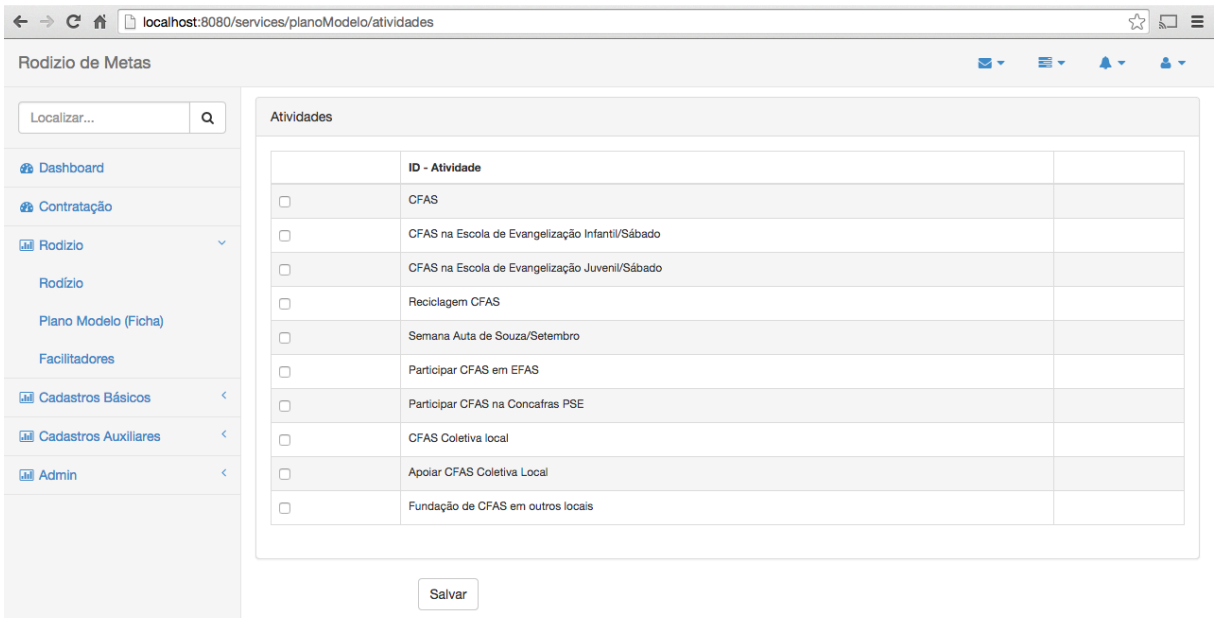


Figura 29 Listagem de Atividades

As Figuras 28 e 29 mostram a tela que lista as atividades o instituto/comissão selecionado. O usuário então pode selecionar as atividades que deseja incluir em seu plano modelo e salvar.

3.4.7.1 Cadastro de Facilitador

Durante o desenvolvimento, chegamos a conclusão da necessidade de um cadastro de facilitadores. O Facilitador é uma pessoa com conhecimento da regra de negócio de um instituto/comissão. Assim todo ano de acordo com a disponibilidade tem-se novos facilitadores. As figuras de 30 a 33 apresentam o cadastro de facilitador, que também é feito por rodizio/ano e por instituto.

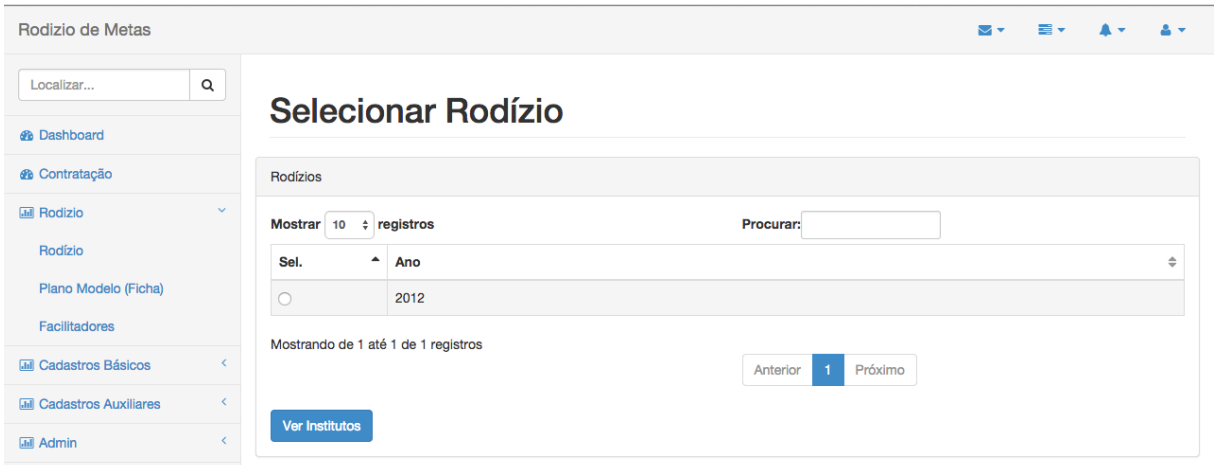


Figura 30 Seleção de Rodízio

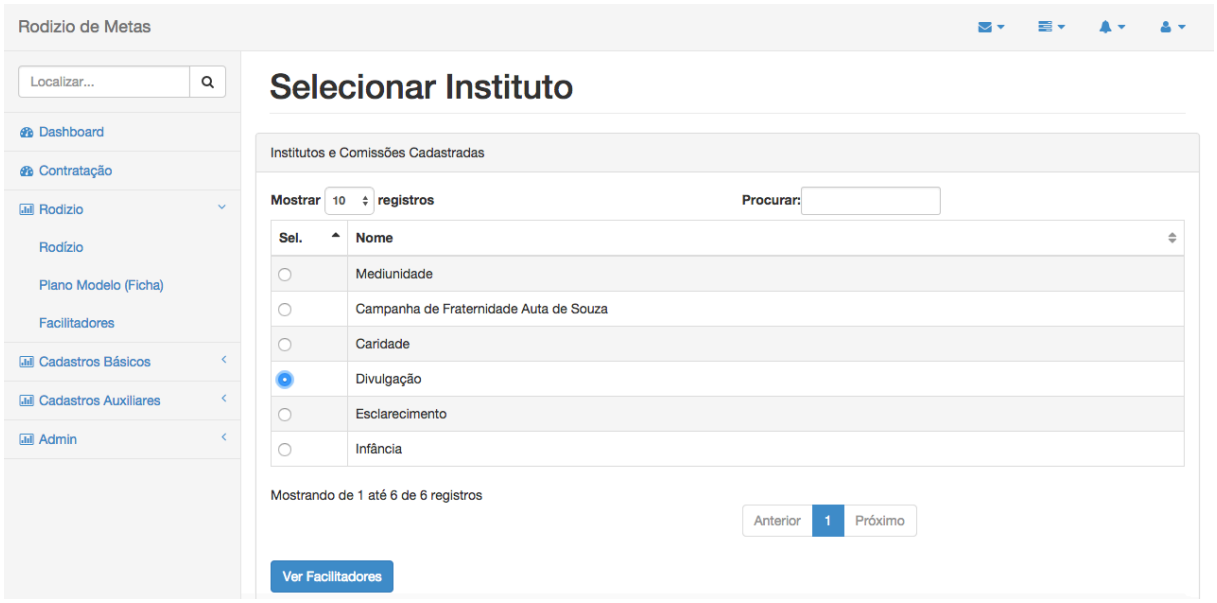


Figura 31 Seleção de Instituto

Figura 32 Listagem de Facilitadores por Instituto

Figura 33Cadastro de Novo Facilitador

3.4.8 Contratação de Metas

3.4.8.1 Contratação

A funcionalidade de Contratação de Metas é o cadastro chave do sistema. Através deles são informados os dados que servirão para análise e acompanhamento das atividades ao longo do ano. Ele se utiliza de todos os cadastros básicos feitos no sistema sem exceções. Para seu uso é preciso que toda a base esteja populada previamente com as informações.

A Figura 34, apresenta o protótipo inicial da tela de contratação de metas.

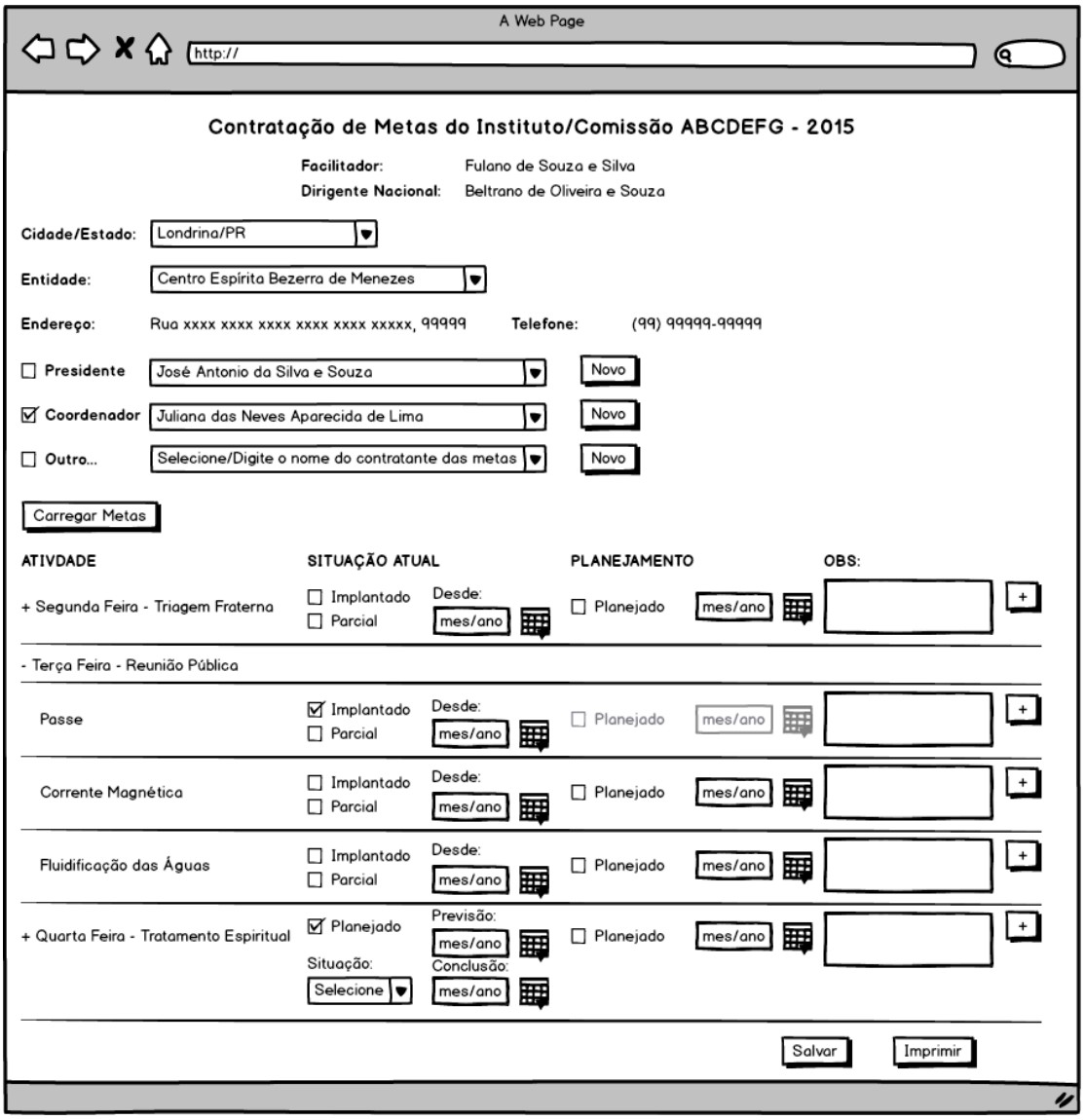


Figura 34 Tela de Contratação de Metas – Protótipo

Durante o início do projeto essa tela foi projetada para ser preenchida em uma única etapa, mas durante os testes com o cliente, constatou-se a necessidade de dividir essa tela em etapas, devido a quantidade de informações que alguns formulários poderiam ter.

O maior conjunto de regras do sistema se encontram nessa interface. Para desenvolver foi preciso utilizarmos tanto validações *client-side*, quanto *server-side*. Javascript foi utilizado na página, além dos recursos já disponíveis no bootstrap, outros componentes foram inseridos como typehead e datatable do jquery, além da construção de funções para ordenação dinâmica de atividades, para permitir ao usuário seleciona de forma rápida em que ordem os itens devem aparecer em tela. Outras funções javascript foram criadas para criação

dinâmica de caixas de comentários, e validação de preenchimentos dos campos.

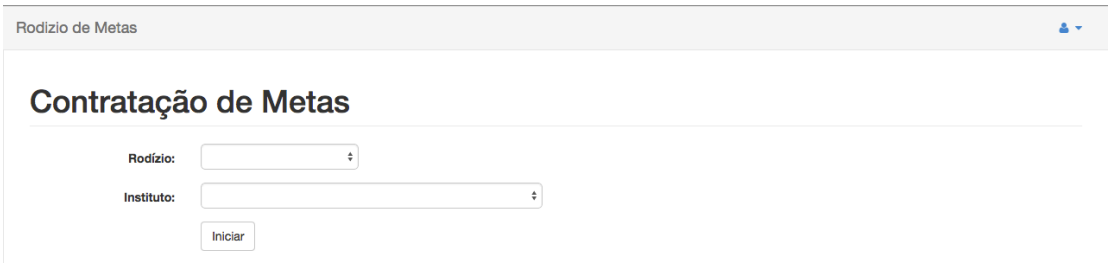


Figura 35 Tela de Contratação de Metas – Passo 1

Na primeira tela (Figura 35) o usuário seleciona o Rodízio que será trabalhado, e o instituto/comissão no qual serão contratadas as metas. Nesse processo ao clicar no botão iniciar, as metas serão carregadas e as informações do usuário logado e do dirigente nacional do instituto/comissão serão carregados conforme a Figura 36. Nesta o usuário irá escolher a cidade a qual pertence através de um campo com autopreenchimento, e também selecionará a entidade pelo nome, sendo está filtrada pela cidade escolhida no campo anterior.

Figura 36 Tela de Contratação de Metas – Passo 2

Ao escolher a entidade, o sistema irá carregar o endereço e telefone da mesma, caso tenha vinculado um presidente ou coordenador irá trazê-lo selecionado nos respectivos combos (Figura 37).

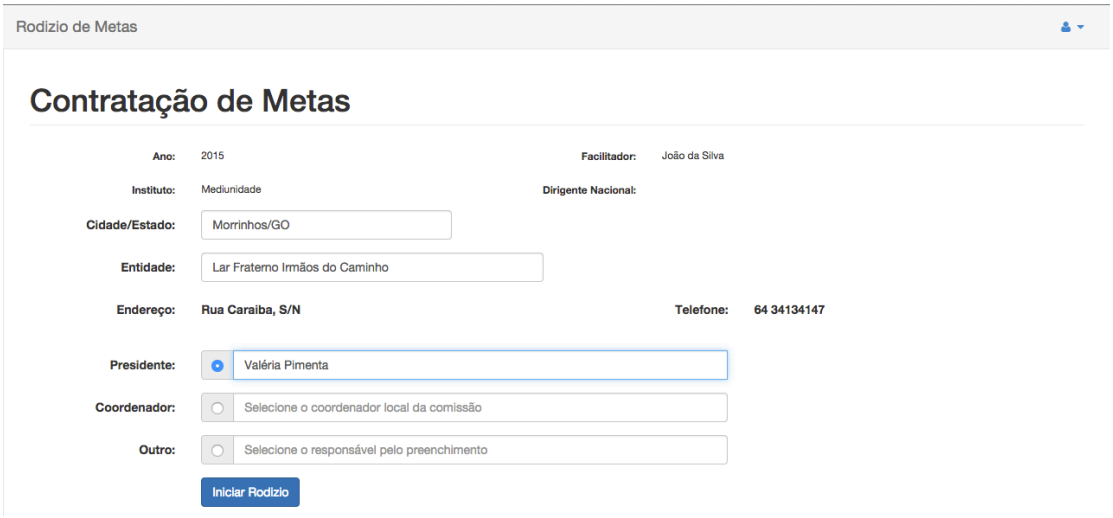


Figura 37 Tela de Contratação de Metas – Passo 2

Essas informações podem ser alteradas nesse momento, sem impacto na base da entidade, essas alterações permanecerão disponíveis apenas para efeitos de contratação de metas. Caso o sistema esteja errado o usuário será instruído a buscar a secretaria para corrigir seus dados. No dia do rodízio, devido o alta demanda esperada no sistema, nenhuma outra alteração será feita diretamente pelos facilitadores.

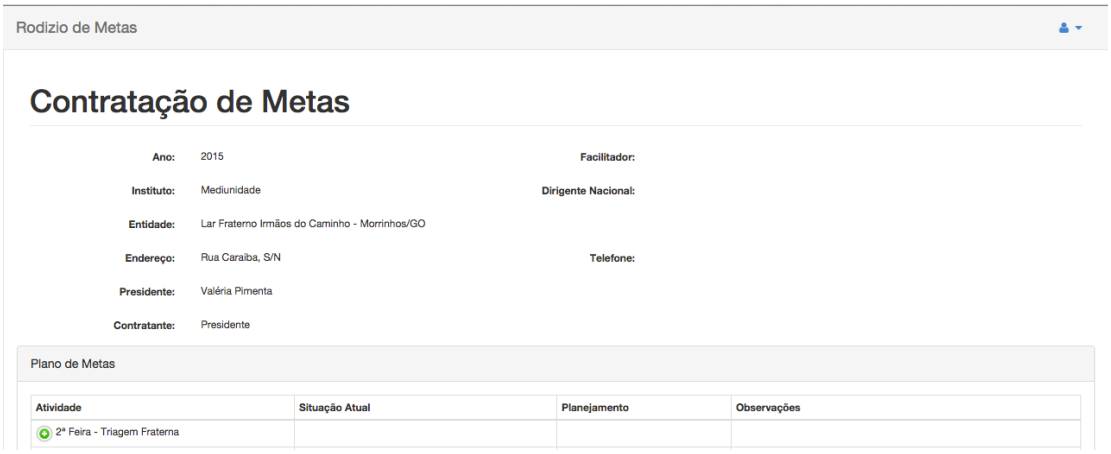


Figura 38 Tela de Contratação de Metas – Passo 3

Na tela seguinte o usuário irá visualizar todas as atividades do instituto/comissão selecionado. Elas serão apresentadas em grupos conforme o cadastro realizado, e com ícones de “+” para que possam ser expandidos.

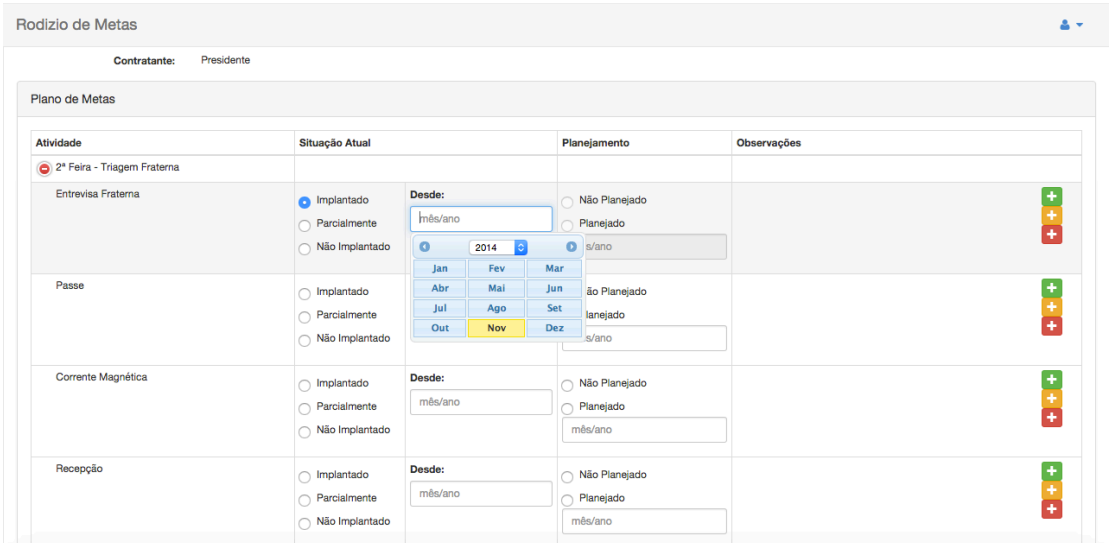


Figura 39 Tela de Contratação de Metas – Passo 3

Ao expandir um tela o usuário poderá selecionar a situação atual. As opções disponíveis podem variar de acordo com as seleções realizadas no ano anterior. No caso exibido, as atividades não possuem histórico, por isso a pergunta se está Implantando, Parcialmente Implantado, Não implantado e desde quando.

Já a opção de planejamento permite escolher entre planejar para aquele período ou não e dizer quando dentro do período que implantação será realizada.

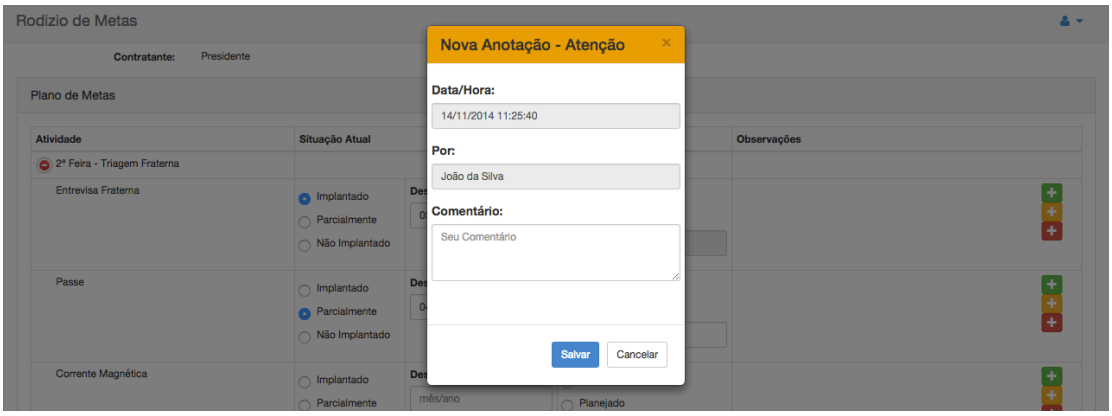


Figura 40 Tela de Contratação de Metas – Comentários

É possível adicionar comentários a cada atividade (Figura 40), contratada ou não, dando um indicador de severidade ao comentário, de acordo com as cores: Verde, uma observação; Amarelo, um alerta, ou ponto de atenção; Vermelho, situação crítica informada pelo contratante.

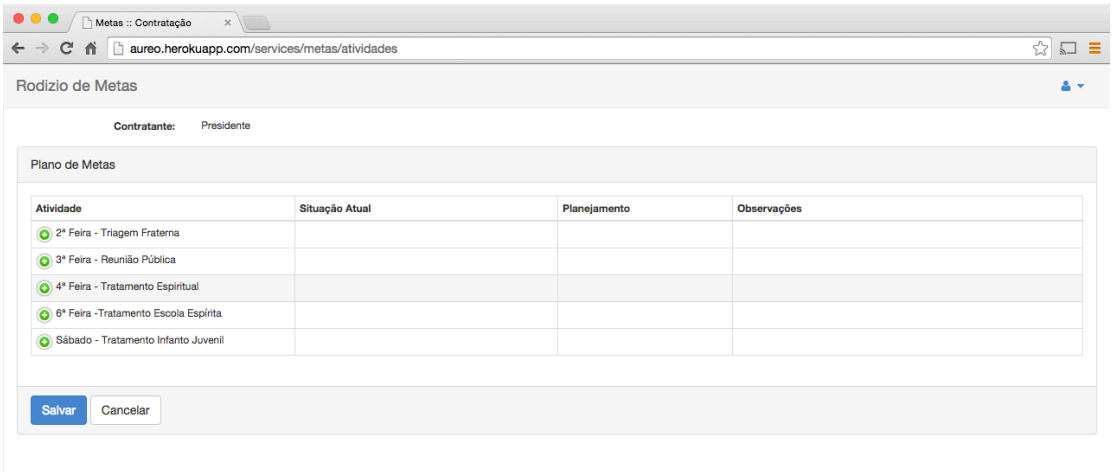


Figura 41 Tela de Contratação de Metas – Salvar

Ao término do cadastro basta salvar a contratação de metas.

3.4.8.2 Acompanhamento de Metas

Dentro da contração de Metas esta previsto uma tela para acompanhamento das metas. Esta tela está em desenvolvimento no Sprint 6 junto com outros relatórios. Na Figura 42 e 43 é possível ver os protótipos desta funcionalidade.

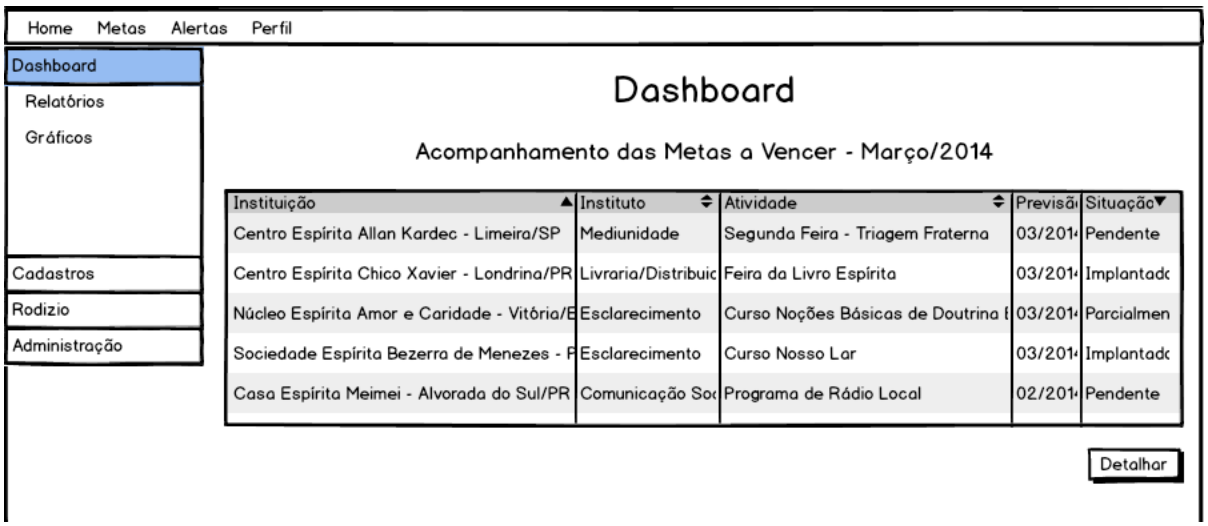


Figura 42 Acompanhamento de Metas

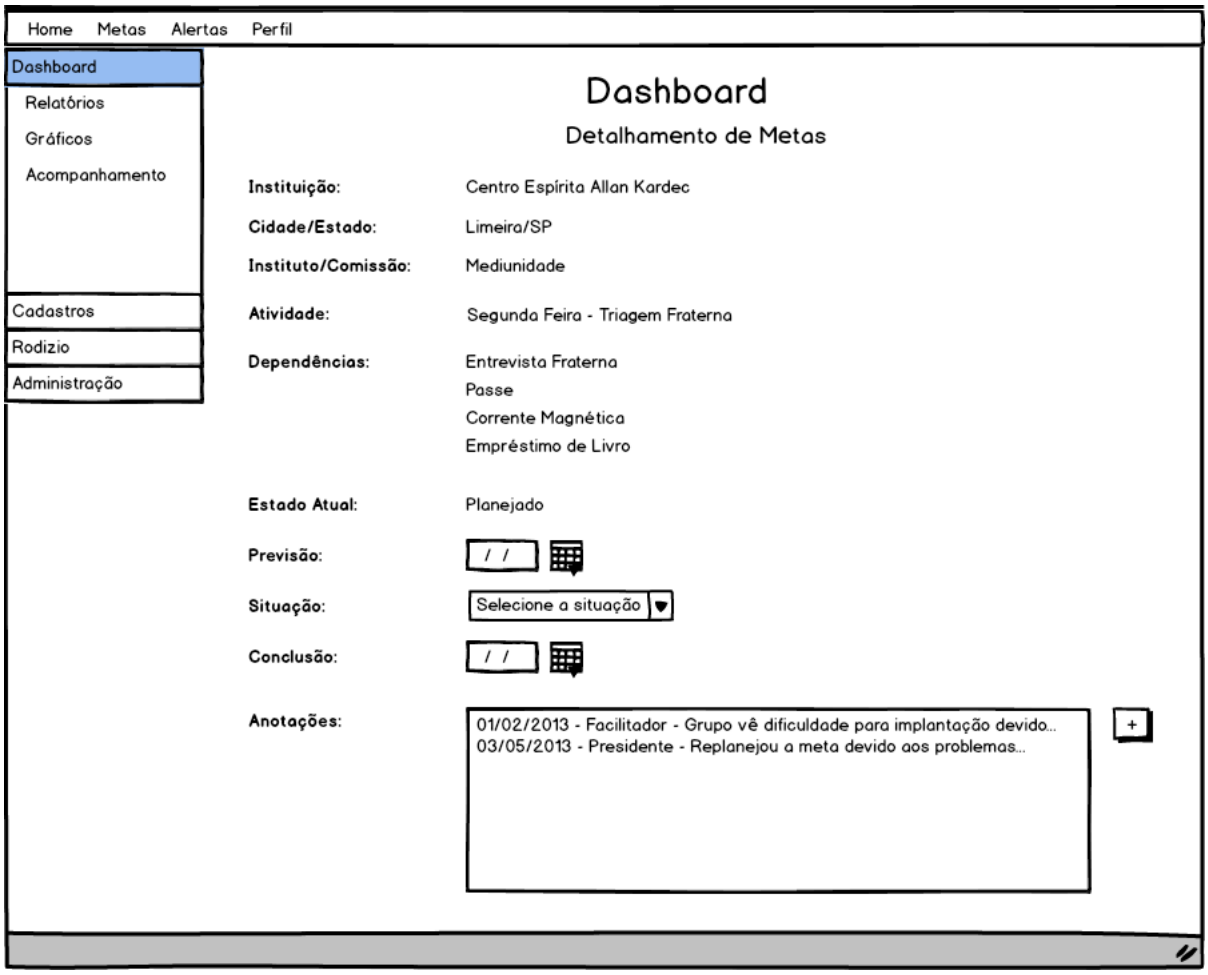


Figura 43 Detalhamento de Meta

3.4.9 Relatórios

Os relatórios e gráficos permitirão acompanhar de forma visual o andamento das metas contratadas. No painel de controle do sistema é possível ver o desempenho geral das entidades de acordo com o seu nível de visão. Esta funcionalidade está representada na Figura 45 que mostra o painel de controle. A Figura 44 mostra o protótipo desta tela.

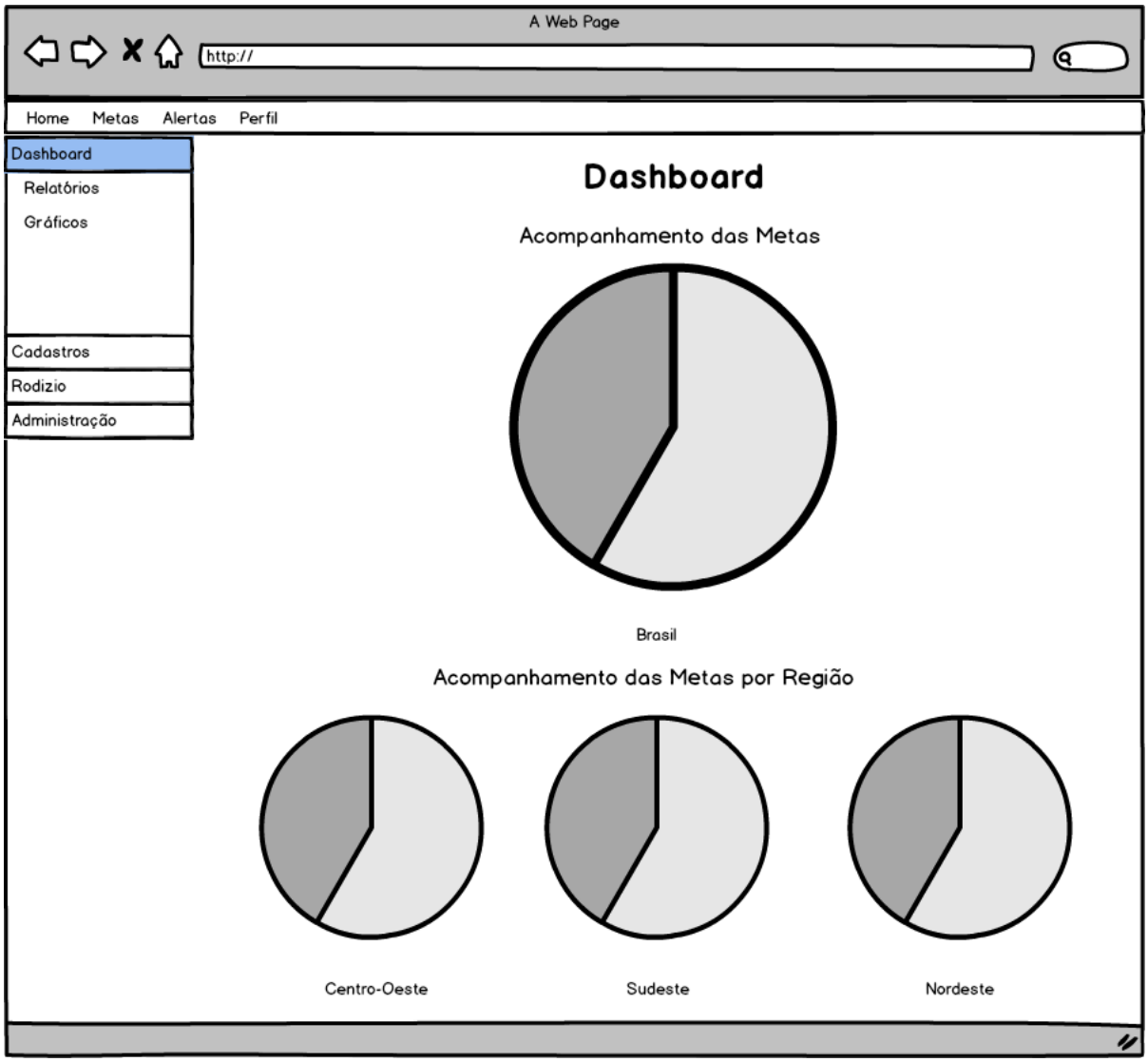


Figura 44 Painel de controle – Gráficos – Protótipo

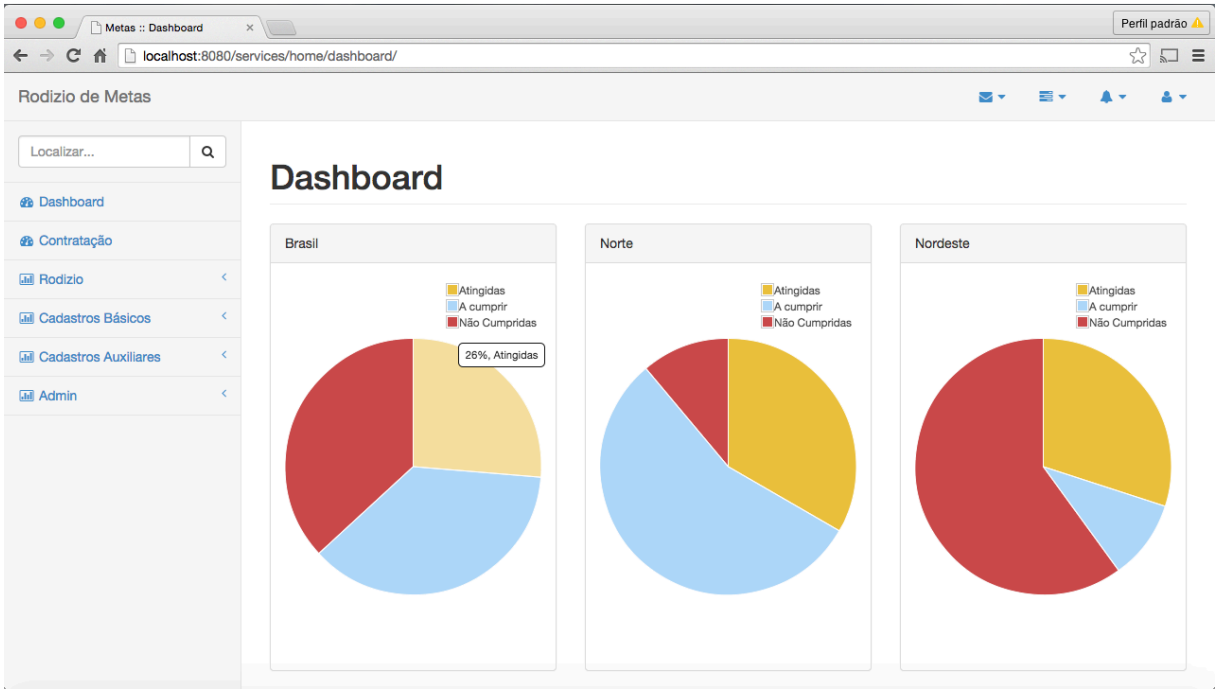


Figura 45 Painel de Controle Gráficos

3.4.10 Cadastro de Caravaneiros

As figuras 46 a 52 mostram o Cadastro de Caravaneiros (Pessoas), que é um cadastro básico, que contém muitas informações, por isso foi divido em abas. No início estas interfaces não seriam desenvolvidas sendo apenas criado um modo de importação.

Como apenas o nome é obrigatório, é possível salvar as informações a qualquer instante escolhendo o botão Salvar.

Esse cadastro será preenchido parcialmente pela secretaria, e todas as pessoas cadastradas, com CPF e email, poderão ter acesso ao sistema, desde que completem suas informações, acessando posteriormente o sistema.

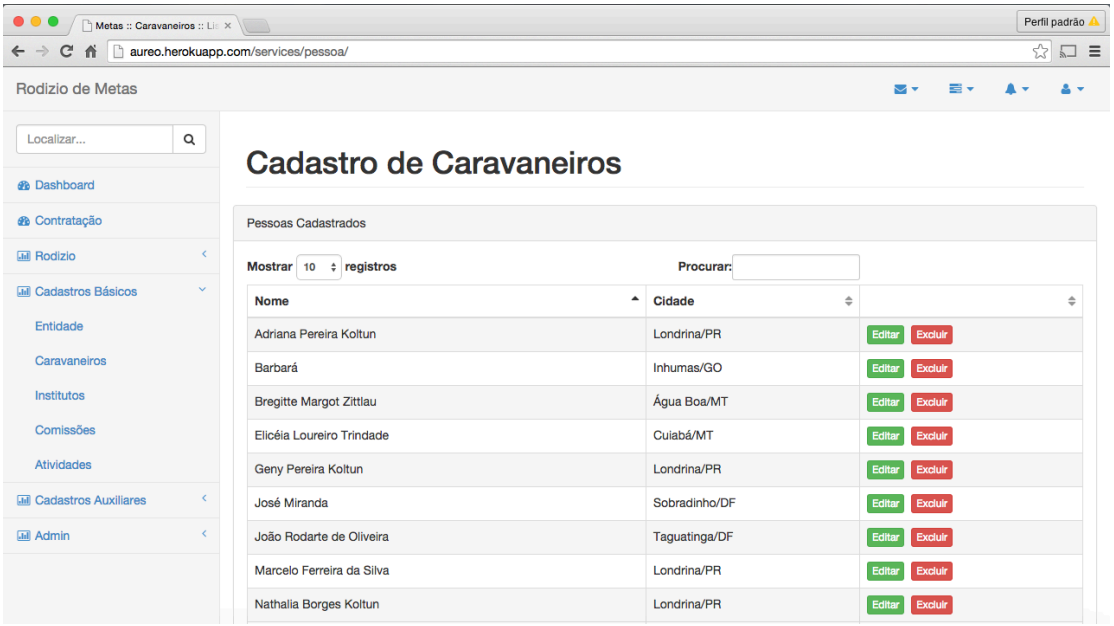


Figura 46 Tela de Listagem de Caravaneiros

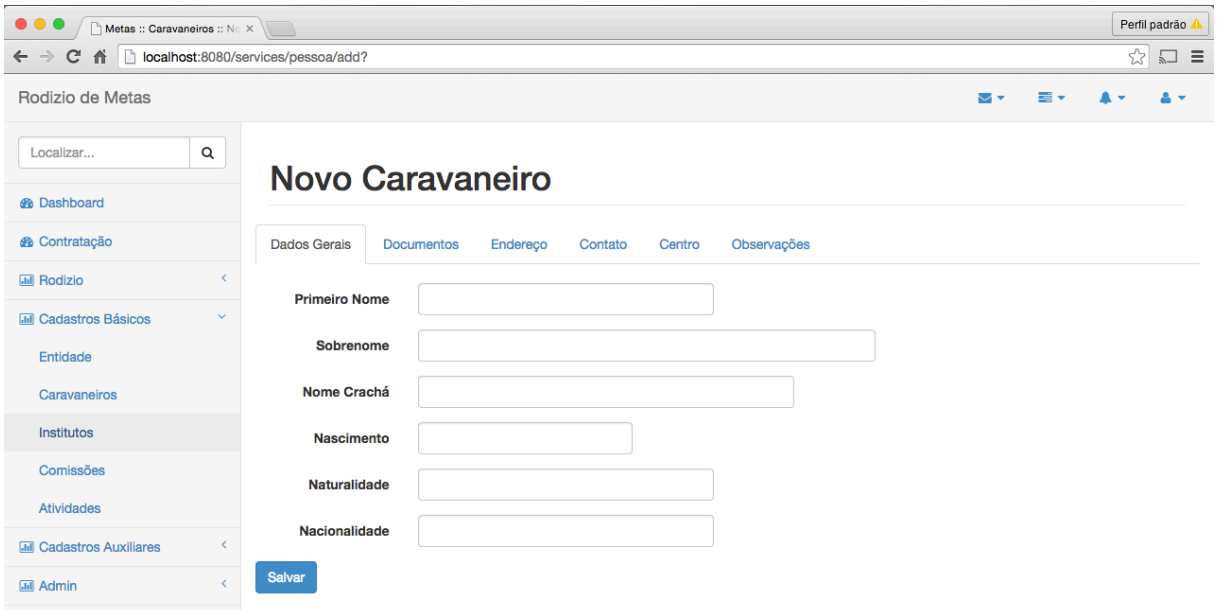


Figura 47 Tela de Cadastro de Caravaneiro - Dados Gerais

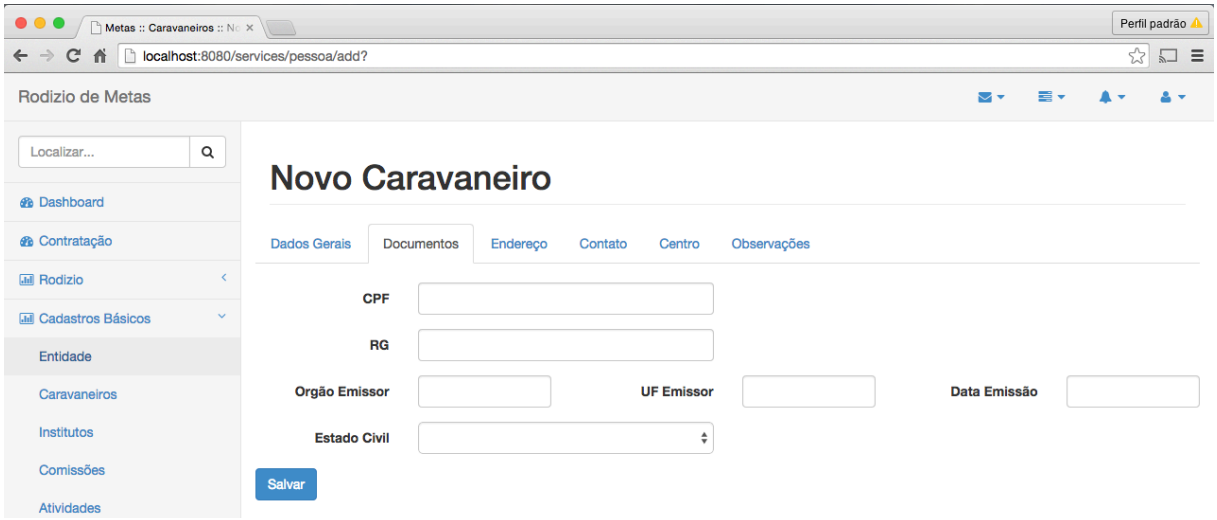


Figura 48 Tela de Cadastro de Caravaneiro - Documentos

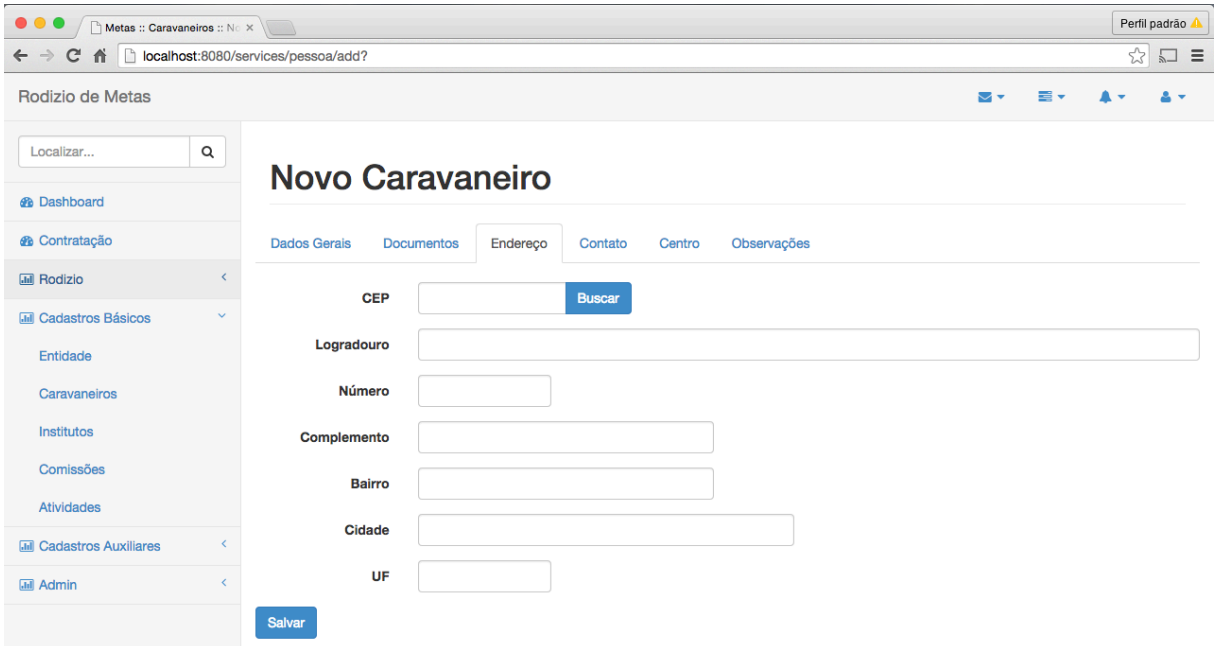


Figura 49 Tela de Cadastro de Caravaneiro - Endereço

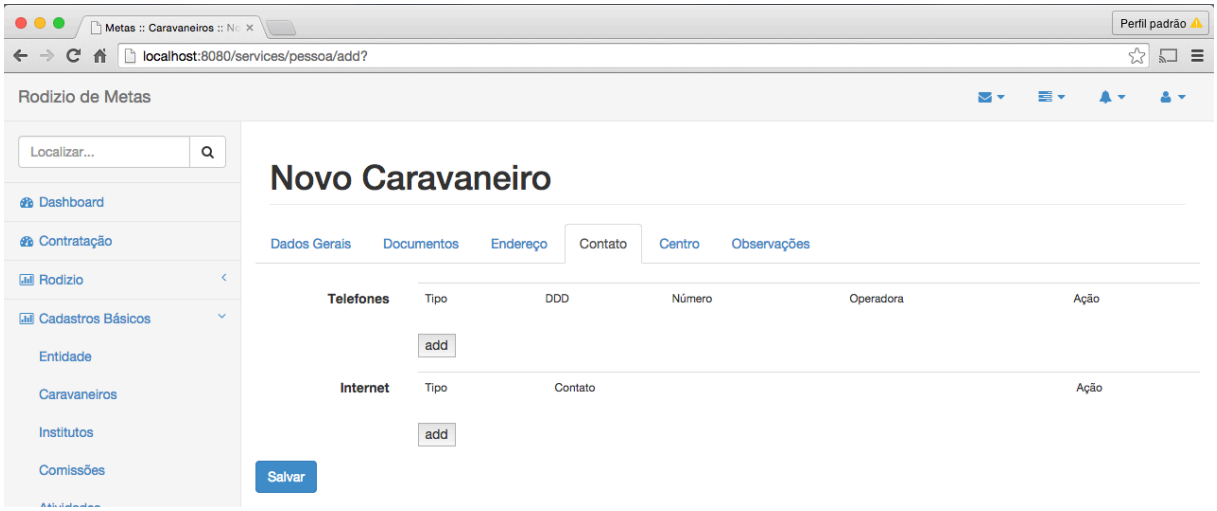


Figura 50 Tela de Cadastro de Caravaneiro - Contato

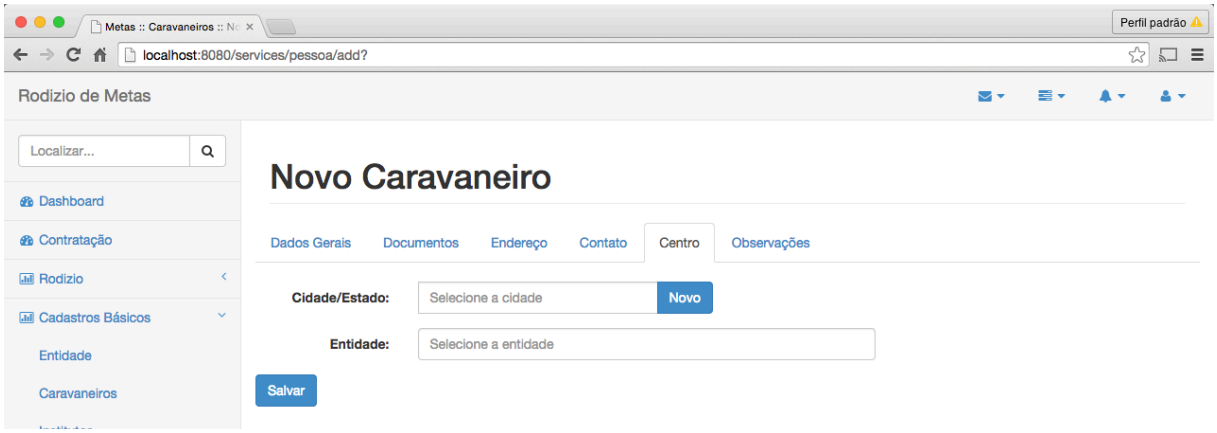


Figura 51 Tela de Cadastro de Caravaneiro - Centro

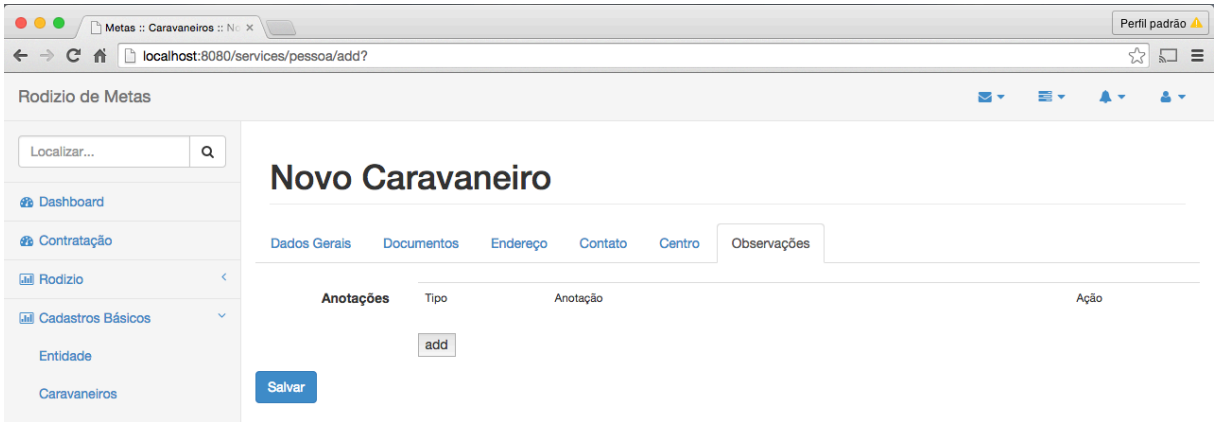


Figura 52 Tela de Cadastro de Caravaneiro - Observações

3.4.11 Cadastro de Entidades

Assim como o Cadastro de Caravaneiros, o Cadastro de Entidades (Figuras de 53 a 58) é um cadastro básico, que contém muitas informações, por isso foi divido em abas. No início estas interfaces, também não seriam desenvolvidas sendo apenas criado um modo de importação.

Para o Cadastro de Entidades, a razão social, a cidade e o CNPJ, são informações obrigatórias. Serão cadastradas pela secretaria e poderão ser complementadas pelo seu presidente, desde que o mesmo esteja vinculado através da aba Diretoria.

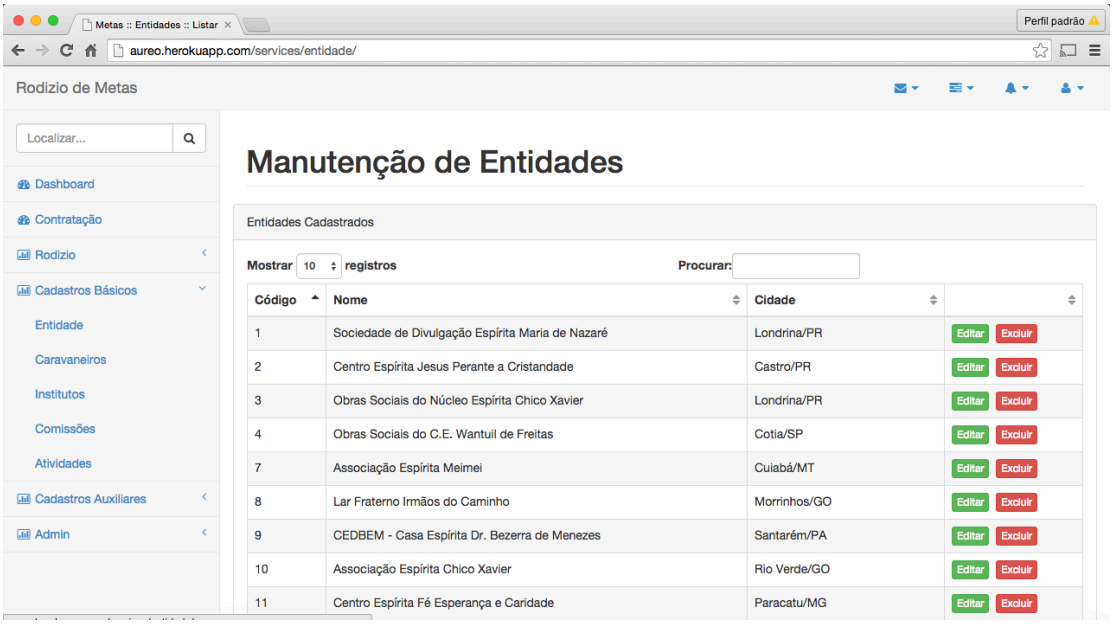


Figura 53 Tela de Listagem de Entidades

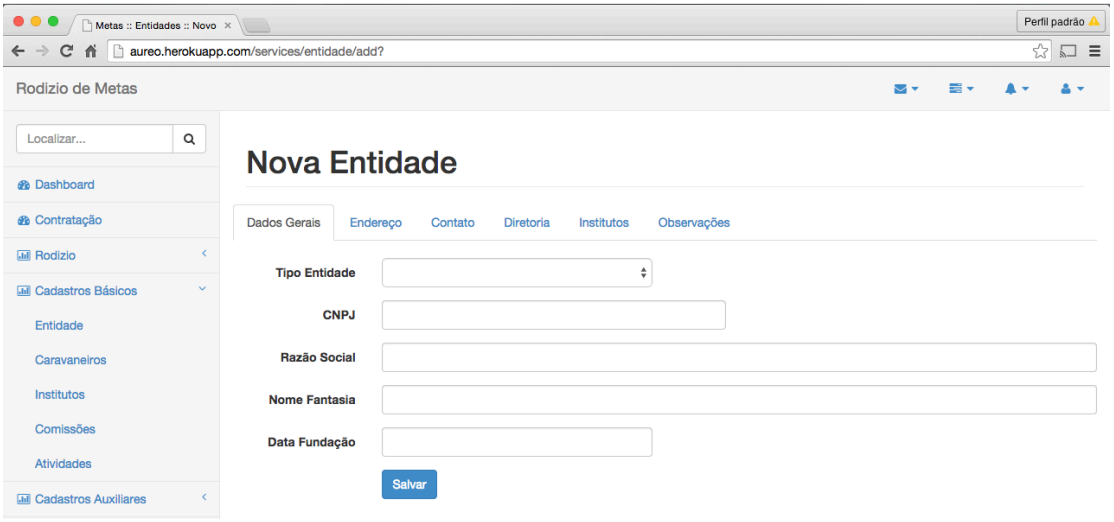


Figura 54 Tela de Inclusão de Entidades – Aba Dados Gerais

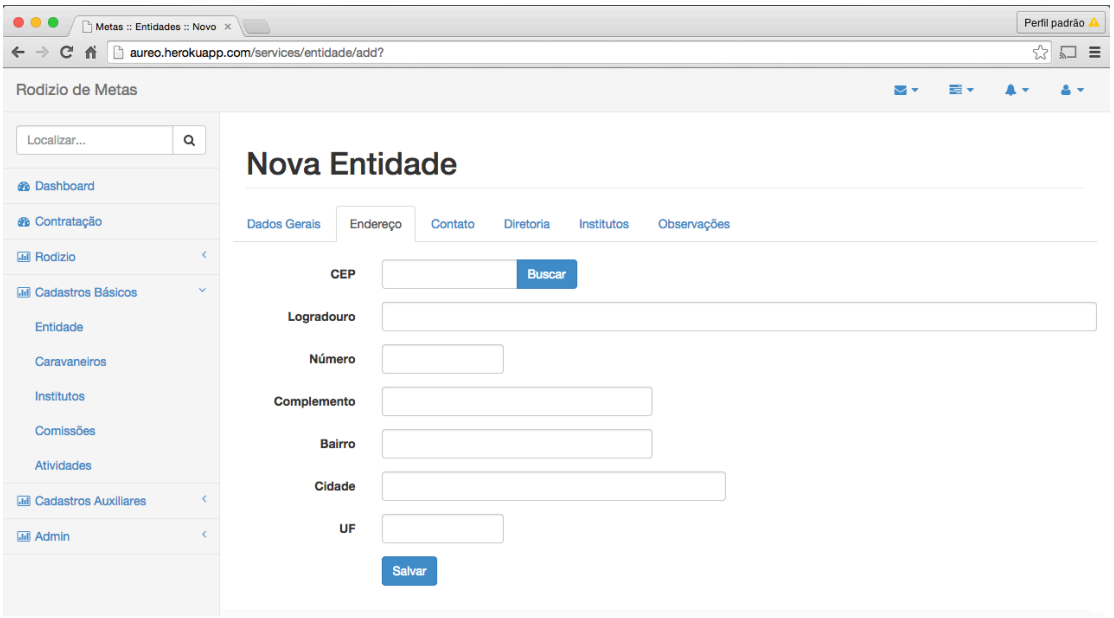


Figura 55 Tela de Inclusão de Entidades – Aba Endereço

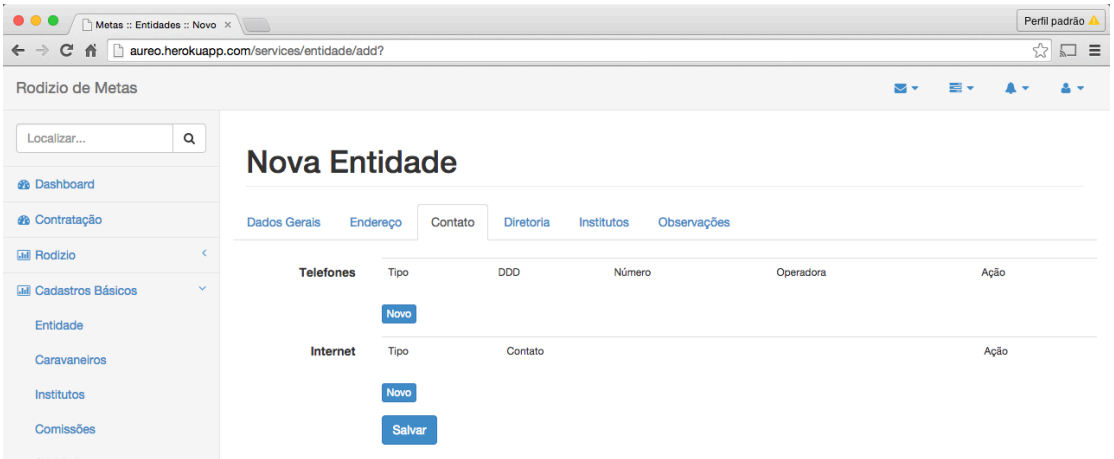


Figura 56 Tela de Inclusão de Entidades – Aba Contato

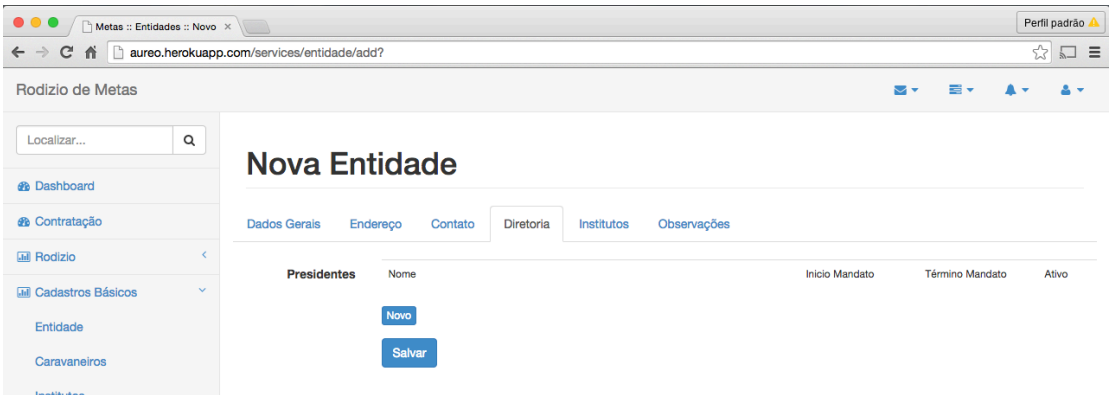


Figura 57 Tela de Inclusão de Entidades – Aba Diretoria

Figura 58 Tela de Inclusão de Entidades – Aba Institutos

3.5 DIAGRAMA DE ENTIDADE RELACIONAMENTO

O modelo entidade relacionamento é usado para descrever os dados ou aspectos de informação de um domínio de negócio ou seus requerimentos de processo, de uma maneira abstrata que em última análise se presta a ser implementada em um banco de dados, como um banco de dados relacional. Os principais componentes dos Modelos Entidades Relacionamento são as entidades, suas relações e armazenamento em bancos de dados.

No sistema de contratação de metas, a principal funcionalidade, tem para sua execução o apoio de diversos cadastros auxiliares, que fornecem a base de sua execução.

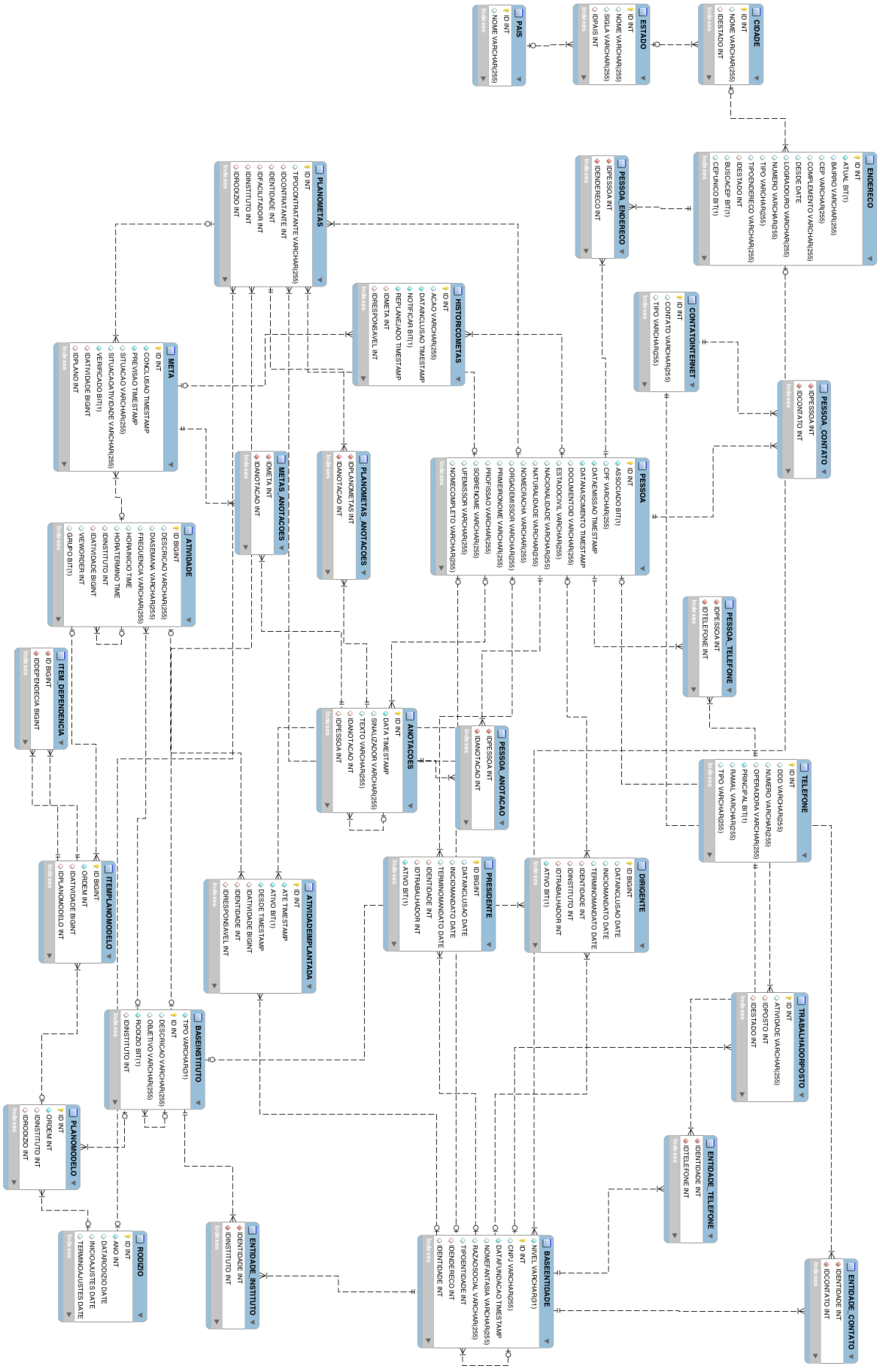


Figura 59 Diagrama de entidade relacionamento

3.6 DIAGRAMA GERAL DE CASOS DE USO

O diagrama de caso de uso descreve a funcionalidade proposta para um novo sistema que será projetado. Ele representa uma unidade discreta da interação entre um usuário e o sistema.

Como a maioria das funcionalidades do sistema são Cadastros, composto por criação, recuperação, alteração e exclusão, colocamos na Figura 60 , um Diagrama que represente a interação geral entre usuários e funcionalidades, e não um para cada funcionalidade.

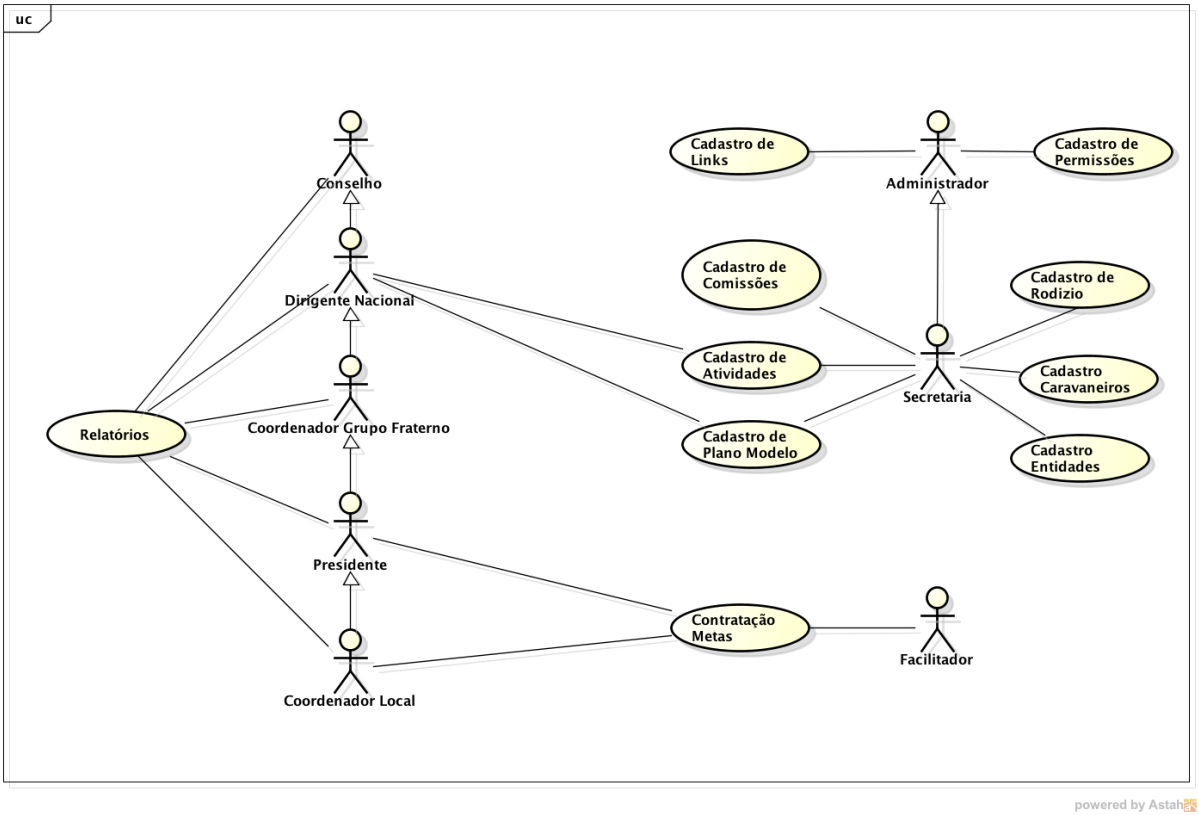


Figura 60 Diagrama de Caso de Uso do Sistema

3.7 STATUS GERAL DO APLICATIVO

Na proposta desse projeto, foram apresentados os requisitos fundamentais aos quais o sistema deverá suprir, adequados a uma das seguintes categorias:

- **Essencial:** requisitos funcionais os quais devem obrigatoriamente integrar o escopo do produto final;
- **Desejável:** requisitos funcionais que, se possível, deverão integrar o escopo do produto final;
- **Não Implementar:** requisitos que não serão implementados no momento, mas que serão necessários em momento futuro;

O Tabela 2 nos fornece uma visão destas funcionalidades:

ID	Funcionalidade	Prioridade Inicial	Status Atual
1	**Segurança**		
1.1	Login	Essencial	Implementado
1.2	Cadastro de Senha	Essencial	Implementado
1.3	Recuperação de Senha	Desejável	Implementado
2	**Cadastro de Links/Funcionalidades (Segurança)**		Funcionalidade Removida
2.1	Lista de Links	Essencial	--
2.2	Inclusão de Links	Essencial	--
2.3	Alteração de Links	Desejável	--
2.4	Exclusão de Links	Desejável	--
3	**Cadastro de Permissões (Segurança)**		
3.1	Lista de Permissões	Desejável	Implementado
3.2	Inclusão de Permissões	Desejável	Não Implementado
3.3	Alteração de Permissões	Desejável	Não Implementado
3.4	Exclusão de Permissões	Desejável	Não Implementado
4	**Cadastro de Rodizio**		
4.1	Lista de Rodízios	Essencial	Implementado
4.2	Inclusão de Rodizio	Essencial	Implementado
4.3	Alteração de Rodizio	Essencial	Implementado
5	**Cadastro de Institutos/Comissões**		
5.1	Lista Institutos/Comissões	Essencial	Implementado
5.2	Inclusão de Institutos/Comissões	Essencial	Implementado
5.3	Alteração de Institutos/Comissões	Essencial	Implementado
5.4	Exclusão Lógica de Institutos/Comissões	Essencial	Implementado
6	**Cadastro de Atividades**		
6.1	Lista de Atividades	Essencial	Implementado
6.2	Inclusão de Atividade	Essencial	Implementado
6.3	Alteração de Atividade	Desejável	Implementado
6.4	Exclusão Lógica de Funcionalidade	Desejável	Implementado
6.5	Importação Excel	--	Implementado
7	**Cadastro de Plano Modelo**		
7.1	Seleção de Atividades p/ Plano Modelo	Desejável	Implementado
8	**Contratação de Metas**		
8.1	Inclusão de Plano de Metas	Essencial	Implementado
8.2	Listagem dos Planos de Metas	Essencial	Implementado
8.3	Alteração de Metas	Essencial	Implementado
8.4	Notificação via Email	Essencial	Implementado
8.5	Listagem de Facilitador	--	Implementado
8.6	Inclusão de Facilitador	--	Implementado
8.7	Exclusão de Facilitador	--	Implementado
9	**Relatórios**		
9.1	Relatório de Acompanhamento Mensal	Essencial	Implementado

9.2	Gráfico Atingimento de Metas	Essencial	Implementado
9.3	Relatório de Atingimento de Metas	Essencial	Não implementado
9.4	Relatório de Estatísticas do Rodízio	Essencial	Implementado
9.5	Envio de Relatório de Acompanhamento Mensal	Desejável	Não implementado
10	**Cadastro de Caravaneiros**		
10.1	Listagem de Caravaneiros	Essencial	Implementado
10.2	Inclusão de Caravaneiros	Não Implementar	Implementado
10.3	Alteração de Caravaneiros	Não Implementar	Implementado
10.4	Importar dados de Caravaneiros	Essencial	Implementado
11	**Cadastro de Entidades**		
10.1	Listagem de Entidades	Essencial	Implementado
10.2	Inclusão de Entidades	Não Implementar	Implementado
10.3	Alteração de Entidades	Não Implementar	Implementado
10.4	Importar dados de Entidades	Essencial	Implementado

Tabela 2 Requisitos do sistema

Os itens 9.3, 9.5 não foram finalizados, por falta de informações do cliente. Serão implementados nos Sprints 6 e 7. Os itens 3.2 a 3.4 serão implementados no futuro.

4. CONCLUSÃO

Com o uso de PaaS, o grande empecilho de um projeto de software como esse acabou sendo o trato com o Cliente. Este apesar de conhecer bem as suas próprias regras de negócio, ainda não tinha certeza de onde gostaria de chegar. Isso fez com que muitas informações discutidas no início do projeto fosse “esquecidas” ao longo do desenvolvimento, criando situações desconfortáveis, mas superadas. O fato de lidar com uma instituição sem fins lucrativos, onde seus gestores, não tem isso como principal ocupação, também ajudou na dificuldade em conseguir organizar uma reunião para aprovação, ou uma discussão sobre funcionalidades.

Posso afirmar também, que esse projeto me ajudou a adentrar mais em conceitos de computação em nuvem, me abrindo horizontes de oportunidades incríveis, já que a partir de um custo de infraestrutura zero, é possível iniciar um projeto que futuramente poderá ser uma nova *Startup* de tecnologia. Com isso sem uma limitação antes técnica, agora resta apenas organização de tempo e ideias criativas para criarmos e facilmente testarmos qualquer novo empreendimento.

REFERÊNCIAS BIBLIOGRÁFICAS

CARR, Nicholas. **Here comes HaaS**. Disponível em <http://www.roughtype.com/?p=279>. Acesso em 01 dez. 2014.

GUEDES, Gilleanes T. A. **UML 2: uma abordagem prática**. São Paulo: Novatec, 2009.

CELESTINO, Eder, TRINDADE, Fernando R; MOMI, Robson; MIZOGUCHI, Thiago Hideki. **Análise e Pesquisa de Soluções Cloud Computing**. Disponível em: <http://engenharia.anhembi.br/tcc-11/si-10.pdf>. Acesso em 01 dez. 2014.

ORACLE Disponível em: <http://docs.oracle.com/javaee/7/firstcup/doc/java-ee002.htm>. Acesso em 01 jul. 2014.

PRESSMAN, Roger S. **Engenharia de software**. 6. ed. São Paulo: McGraw-Hill, 2006.

SABBAGH, Rafael. SCRUM: **Gestão Ágil para Projetos de Sucesso**. São Paulo: Casa do Código, 2013.

SPRING Disponível em: <http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html>. Acesso em 01 jul. 2014.

SILVA, Edgar; Franz Fabiano **"OpenShift e Java EE 6"**, Java Magazine nr. 100, 2012.

SNOWMAN, G. **Neve na Nuvem: Diferença nos tipos de Computação nas Nuvens**. The SolidQ™ Journal, v. 40, 2010.

SOMMERVILLE, Ian. **Engenharia de Software**. 8. ed. São Paulo, Addison-Wesley, 2007.

WHERTHEIN, Jorge. **A Sociedade da Informação e seus Desafios. Revista**

Ciência da Informação, Brasília, v. 29, n. 2, maio/ago. 2000. Disponível em: < http://revista.ibict.br/ciinf/index.php/ciinf/article/view/254/1705>. Acesso em 01 jul. 2014.

ANEXO I – PROPOSTA.

MINISTÉRIO DA EDUCAÇÃO
UTFPR – UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CAMPUS CORNÉLIO PROCÓPIO
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

MARCELO FERREIRA DA SILVA

PROPOSTA DE TRABALHO DE CONCLUSÃO DE CURSO

SISTEMA DE ACOMPANHAMENTO DE METAS

CORNÉLIO PROCÓPIO
2014

MINISTÉRIO DA EDUCAÇÃO
UTFPR – UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CAMPUS CORNÉLIO PROCÓPIO
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

MARCELO FERREIRA DA SILVA

PROPOSTA DE TRABALHO DE CONCLUSÃO DE CURSO

SISTEMA DE ACOMPANHAMENTO DE METAS

Proposta de Trabalho de Conclusão de Curso apresentada à disciplina de Trabalho de Diplomação do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Msc. Francisco Pereira Junior

CORNÉLIO PROCÓPIO
2014

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

AJAX	*Asynchronous Javascript and XML*
API	*Application Programming Interface*
ASP	*Active Server Pages*
CASE	*Computer-Aided Software Engineering*
CGI	*Common Gateway Interface*
CSS	*Cascading Style Sheets*
EAR	*Enterprise ARchive*
EJB	*Enterprise Java Beans*
GNU	*GNU's Not Unix*
JAR	*Java ARchive*
JDO	*Java Data Objects*
JEE	*Java Enterprise Edition*
JPA	*Java Persistence API*
JSP	*Java Server Pages*
JSR	*Java Specification Requests*
MIT	*Massachusetts Institute of Technology*
MVC	*Model-View-Controller*
PHP	*PHP: Hypertext Preprocessor*
SGBD	Sistema Gerenciador de Banco de Dados
UML	*Unified Modeling Language*
UTFPR	Universidade Tecnológica Federal do Paraná
WAR	*Web Archive*
XML	*eXtensible Markup Language*

Sumário

1 INTRODUÇÃO

A evolução tecnológica e a acessibilidade às tecnologias proporcionam uma nova perspectiva sobre a agilidade na gestão de informações. O mercado de tecnologia da informação tornou-se amplo e extremamente necessário, pois todos os dias são produzidas milhares de informações relevantes, que mudam rapidamente e precisam estar consistentes e atualizadas para auxiliarem nas tomadas de decisões (WHERTHEIN, 2000).

A proposta da tecnologia da informação é justamente tratar este grande volume de dados, permitindo organizar, manipular e recuperar as informações desejadas, de maneira rápida e fácil. Esta necessidade precisa ser suprida em diferentes escalas, em grandes instituições ou pequenos setores, ter as informações consistentes e facilmente acessíveis é imprescindível para a eficácia de sua utilização (SOMMERVILLE, 2007).

Hoje o mercado corporativo em geral, depende plenamente da tecnologia da informação que se tornou um serviço de primeira necessidade assim com a energia ou a água. Pequenas empresas a partir do momento em que começam a crescer, sentem-se obrigadas a informatizar para não perder em competitividade para os concorrentes.

No terceiro setor da economia, instituições filantrópicas, associações, igrejas e fundações, estão expandindo suas atividades e necessitam se organizar para melhor atender aqueles a quem atende. Para apresentarmos os conceitos aprendidos durante a realização do Curso de Tecnologia em Informática da Universidade Tecnológica Federal do Paraná (UTFPR) – Campus Cornélio Procópio, escolhemos um estudo de caso, nesse meio, para criação de um sistema de apoio a gestão.

1.1 Descrição do Problema

Todo ano, centenas de instituições de assistência social de todo o Brasil, pertencentes ao um movimento religioso nacional, reúnem-se em Brasília, no período da páscoa, durante três dias para traçarem as metas de atividades que serão implantadas em suas cidades no próximo ano exercício.

Essas atividades são divididas em comissões de trabalho,

agrupadas de acordo com a finalidade e/ou público alvo. Assim temos comissões específicas para infância, juventude, caridade, esclarecimento, dentre outras.

Cada atividade, possui uma metodologia para aplicação, material de apoio e objetivo específico, proposto e adaptado de acordo com o resultado obtido em anos anteriores pelas instituições que já implantaram a mesma atividade através da troca de experiências.

O processo no qual cada instituição escolhe as suas metas de implantação para o ano é chamado rodízio. O rodízio tem esse nome devido ao formato que é utilizado a mais de dez anos, onde cada comissão oferece facilitadores, que são pessoas com conhecimento avançado e específico dessa comissão, que auxiliam as instituições interessadas na implantação das atividades em suas dúvidas sobre como implantar, ou como se planejar para uma implantação parcial ou futura. Hoje esses facilitadores se dividem em seis grupos por comissão, capazes de atender até doze instituições/cidades por período. No total são catorze comissões, divididas em 28 salas, cada sala com 3 grupos, num total de 84 grupos. Com capacidade de atendimento efetivo de 168 instituições por período.

No rodízio cada período é composto por 12 minutos. Cada cidade presente com suas instituições é direcionada a um grupo de facilitadores. Ocupando assim todos os 84 grupos presentes. Ao dar início ao rodízio, as instituições começam a preencher suas fichas com as atividades que querem implantar em suas cidades. A quantidade de atividades disponíveis por comissão, variam de 15 à 90 atividades aproximadamente. A cada término de período, as instituições são direcionadas para a troca de facilitadores para poder contratar as atividades de outra comissão e assim se repete até que se tenha passado por facilitadores das catorze comissões.

Hoje todo processo é feito em folhas carbonadas (vide Anexo I), onde ao término de cada período a instituição deixa uma cópia com a o facilitador e leva o original consigo. A instituição decide qual atividade irá implantar, quando irá implantar e se será completa ou parcialmente.

Como o passar dos anos, o número de instituições tem aumentado consideravelmente, chegando hoje perto de 300 instituições. Devido a dificuldades de local, que não permite que seja expandido o número de facilitadores, e de tempo, que não permite que se estenda o tempo de atendimento, o processo acaba

sobrecarregando os facilitadores que em determinados períodos chegam a atender mais de 30 instituições, causando quase sempre um prejuízo na coleta das informações.

Outro ponto de dificuldade está na base histórica. Devido a quantidade de fichas guardadas, letras incompreensíveis, falhas nas cópias carbonadas, não é possível resgatar os dados, sendo estas fichas usadas apenas para orientação da própria instituição no sentido do que haveria de ser feito ou não. Mas o responsável por cada comissão a nível nacional, acaba não tendo controle sobre as atividades que foram contratadas, nem sobre quais estão implantadas ou não.

Convidado pelos organizadores do evento a avaliar a possibilidade de utilização das tecnologias disponíveis no mercado para melhorar o processo, traçamos aqui nessa proposta um Sistema de Acompanhamento de Metas, que será descrito melhor nos próximos capítulos.

2 JUSTIFICATIVA

Diante da necessidade de ter controle direto sobre o andamento das instituições vinculadas ao movimento Concafras-PSE, esse sistema se mostra como solução que contempla tanto a evolução na coleta das informações como no armazenamento, e uso posterior delas, permitindo ao responsáveis ter acesso as informações estatísticas que auxiliem na tomada de decisões.

3 VISÃO DA SOLUÇÃO – OBJETIVOS GERAIS

O Sistema de Acompanhamento de Metas, será um sistema divido em 2 módulos.

O módulo de contratação, será utilizado na preparação e na execução do dia do rodízio, onde terá as funcionalidades de cadastro básico, e de contratação das atividades. Esse módulo será de uso dos organizadores do evento e dos facilitadores que o utilizarão para poder ver as atividades já implantadas por cada instituição, seus motivos de não cumprimento das metas e auxiliar com maior exatidão na contratação das metas para o próximo ano.

O segundo módulo de acompanhamento permitirá que os dirigentes locais possam acompanhar suas metas para cada mês através de relatórios, e também possam atualizar as informações sobre o cumprimento de metas, replanejamento ou cancelamento das mesmas. Os dirigentes nacionais poderão acompanhar através de relatórios os andamentos de cada comissão podendo ter uma visão por mês, ano, região e outros filtros disponíveis. Ambos poderão ser alertados para metas que estão a vencer e que ainda não tiveram um posicionamento cadastrado.

4 ESCOPO DA SOLUÇÃO – OBJETIVOS ESPECÍFICOS

Para compreender adequadamente o escopo da solução são necessárias atividades de levantamento de requisitos, portanto foi realizado um levantamento preliminar, que caracteriza o núcleo do projeto (PRESSMAN, 2006). O levantamento foi realizado através de entrevistas com os organizadores do rodízio. Foram levantadas as necessidades mais urgentes, e detalhadas as características específicas da aplicação. Estas informações serão relevantes para o funcionamento do sistema, o qual irá mensurar a afinidade entre usuários e sugestão dos mesmos baseando-se em suas especificações de características.

Baseando-se na metodologia selecionada para o desenvolvimento não é possível mensurar todos os requisitos detalhadamente na etapa inicial, uma visão generalizada das principais funcionalidades foi documentada para que durante o processo as diversas iterações contemplem o refinamento e revisão dos requisitos. Nesta seção serão apresentados os requisitos fundamentais aos quais o sistema deverá suprir, adequados a uma das seguintes categorias:

- **Essencial:** requisitos funcionais os quais devem obrigatoriamente integrar o escopo do produto final;
- **Desejável:** requisitos funcionais que, se possível, deverão integrar o escopo do produto final;
- **Não Implementar:** requisitos que não serão implementados no momento, mas que serão necessários em momento futuro;

As necessidades observadas estão no Quadro 1 onde constam suas respectivas descrição e categoria:

ID	Funcionalidade	Descrição	Prioridade
1	**Segurança**		
1.1	Login	Interface de Login, apresentando a tela inicial do sistema.	Essencial
1.2	Cadastro de Senha	Interface para cadastro da senha inicial, com base no e-mail utilizado no momento da inscrição no evento.	Essencial
1.3	Recuperação de Senha	Interface para geração de senha temporária e envio por e-mail. Interface para validação da senha temporária e troca por nova senha.	Desejável

2	**Cadastro de Links/Funcionalidades (Segurança)**		
2.1	Lista de Links	Listagem dos links/funcionalidades disponíveis no sistema	Essencial
2.2	Inclusão de Links	Interface para inclusão de novos links/funcionalidades.	Essencial
2.3	Alteração de Links	Interface para alteração de links/funcionalidades existentes	Desejável
2.4	Exclusão de Links	Funcionalidade para exclusão de link/funcionalidades.	Desejável
3	**Cadastro de Permissões (Segurança)**		
3.1	Lista de Permissões	Listagem de Permissões cadastradas no sistema.	Desejável
3.2	Inclusão de Permissões	Interface para inclusão de Permissões de acesso aos links/funcionalidades de acordo com o nível (fixos) e permissões de Consulta (somente leitura) ou Edição, Especial (Para funcionalidades especificas)	Desejável
3.3	Alteração de Permissões	Interface para alteração de Permissões cadastradas	Desejável
3.4	Exclusão de Permissões	Funcionalidade para exclusão de permissões cadastradas.	Desejável
4	**Cadastro de Rodizio**		
4.1	Lista de Rodízios	Listagem de Rodízios já realizados (Base Histórica)	Essencial
4.2	Inclusão de Rodizio	Interface para Inclusão de Rodízio para o próximo exercício.	Essencial
4.3	Alteração de Rodizio	Interface para Alteração de Rodizio	Essencial
5	**Cadastro de Institutos/Comissões**		
5.1	Lista Institutos/Comissões	Listagem de Institutos/Comissões	Essencial
5.2	Inclusão de Institutos/Comissões	Interface para inclusão de Institutos/Comissões	Essencial
5.3	Alteração de Institutos/Comissões	Interface para alteração de Institutos/Comissões	Essencial
5.4	Exclusão Lógica de Institutos/Comissões	Funcionalidade para exclusão de Institutos/Comissões	Essencial
6	**Cadastro de Atividades**		
6.1	Lista de Atividades	Listagem de Atividades	Essencial
6.2	Inclusão de Atividade	Interface para inclusão de Atividade	Essencial
6.3	Alteração de Atividade	Interface para alteração de Atividade	Desejável
6.4	Exclusão Lógica de Funcionalidade	Funcionalidade para exclusão de Atividades	Desejável
7	**Cadastro de Plano Modelo**		
7.1	Seleção de Atividades p/ Plano Modelo	Cadastro/Alteração de Plano Modelo (Ficha de Metas) por Instituto/Comissão para o Rodízio Atual	Desejável
8	**Contratação de Metas**		
8.1	Inclusão de Plano de Metas	Inclusão do Plano de Metas da Instituição por Instituto/Comissão com base no plano modelo.	Essencial
8.2	Listagem dos Planos de Metas	Listagem dos Planos de Metas por níveis de visão: Nacional, Comissão, Grupo Fraterno, Instituição	Essencial
8.3	Alteração de Metas	Interface para alteração do status das metas e inclusão de anotações	Essencial
8.4	Notificação via Email	Envio automático de emissão em caso de alterações no status da meta.	Essencial
9	**Relatórios**		
9.1	Relatório de Acompanhamento Mensal	Relatório de metas vencidas não atingidas e a vencer no mês corrente. Visões por níveis: Nacional, Comissão, Grupo Fraterno, Instituição.	Essencial
9.2	Gráfico Atingimento de Metas	Percentuais de metas cumpridas e não cumpridas com visões por níveis: Nacional (Geral, Região, Estado) / Comissão (Geral, Região, Estado)/ Grupo Fraterno / Instituição).	Essencial
9.3	Relatório de Atingimento de Metas	Percentuais de metas cumpridas, cumpridas parcialmente e não cumpridas com visões por níveis: Nacional (Geral, Região, Estado) / Comissão (Geral, Região, Estado)/ Grupo Fraterno / Instituição).	Essencial
9.4	Relatório de Estatísticas do Rodízio	Informações Gerais sobre o Rodízio: Entidades Presentes, Quantidades de metas contratadas, Quantidade de metas cumpridas no ano anterior.	Essencial
9.5	Envio de Relatório de Acompanhamento Mensal	Funcionalidade para envio de e-mail com o Relatório de Acompanhamento Mensal	Desejável

10	**Cadastro de Caravaneiros**		
10.1	Listagem de Caravaneiros	Funcionalidade será implementada a partir de base da secretaria.	Essencial
10.2	Inclusão de Caravaneiros	Funcionalidade já existente no sistema da secretaria, apenas importaremos os dados	Não Implementar
10.3	Alteração de Caravaneiros	Funcionalidade já existente no sistema da secretaria, apenas importaremos os dados	Não Implementar
10.4	Importar dados de Caravaneiros	Importar dados dos Caravaneiros do sistema da secretaria	Essencial
11	**Cadastro de Entidades**		
10.1	Listagem de Entidades	Funcionalidade será implementada a partir de base da secretaria.	Essencial
10.2	Inclusão de Entidades	Funcionalidade já existente no sistema da secretaria, apenas importaremos os dados	Não Implementar
10.3	Alteração de Entidades	Funcionalidade já existente no sistema da secretaria, apenas importaremos os dados	Não Implementar
10.4	Importar dados de Entidades	Importar dados dos Caravaneiros do sistema da secretaria	Essencial

Tabela 1 Requisitos do sistema

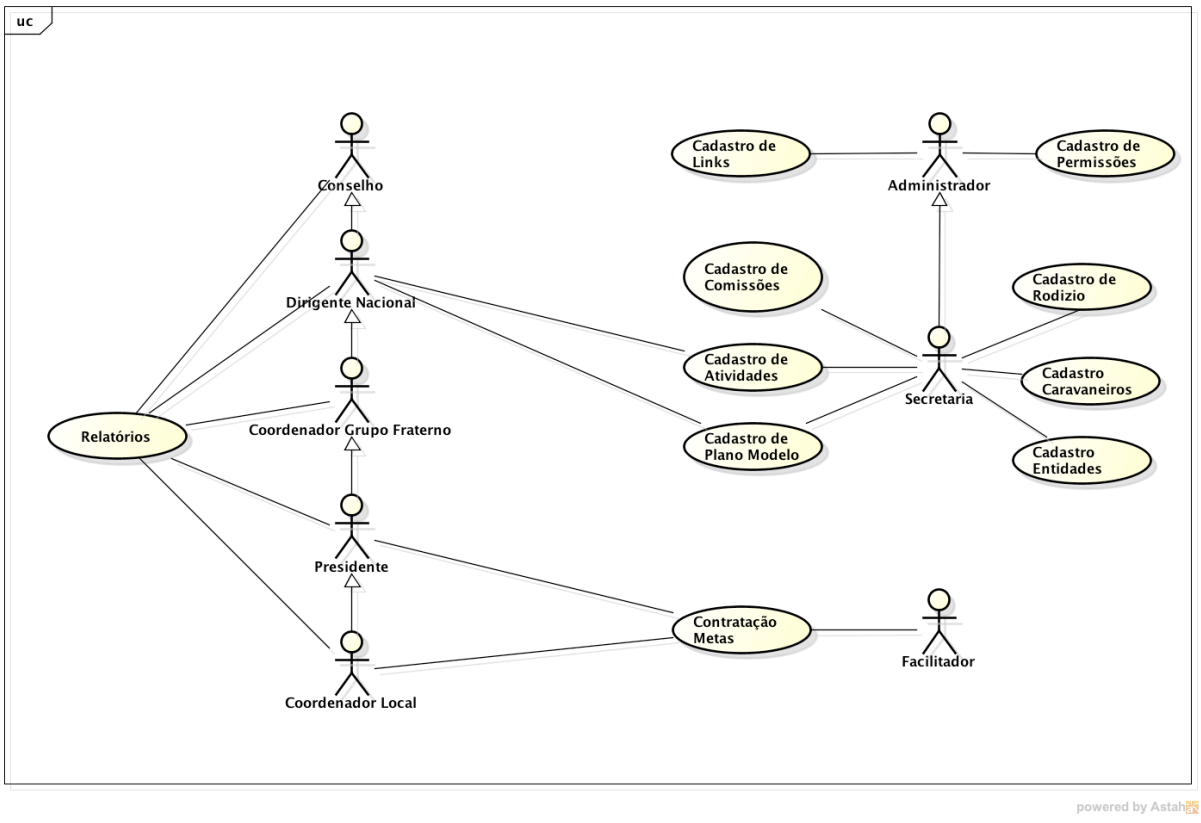


Figura 1: Diagrama de Caso de Uso do Sistema

5 DESCRIÇÃO DOS USUÁRIOS DO SISTEMA

A dinâmica que envolverá o sistema abrange oito perfis de usuários, conforme tabela abaixo:

PERFIL	***DESCRIÇÃO***
Facilitador	*Responsável por auxiliar o Dirigente Local/Presidente na contratação de metas. Tem acesso a tela de Contratação de Metas, apenas no dia do Rodízio.*
Secretaria	*Responsável pelos Cadastro Básicos*
Administrador	*Além dos acesso da Secretaria, tem também acesso aos módulos de permissões.*
Dirigente Local	*Tem visão das metas e relatórios da comissão de sua responsabilidade ao nível da sua instituição.*
Presidente (Local)	*Tem visão das metas e relatórios de todas as comissões de sua instituição. Tem acesso a alteração das metas de sua instituição.*
Coordenador Grupo Fraterno (Regional)	*Tem visão das metas e relatórios das as Instituições que compõem a sua região de atuação.*
Dirigente Nacional	*Tem visão das metas e relatórios da comissão de sua responsabilidade em âmbito nacional.*
Conselho	*Tem visão das metas e relatórios de todas as comissões em âmbito nacional.*

Tabela 2 Perfis de Usuários

Os últimos cinco níveis (Dirigente Local, Presidente, Coordenador Grupo Fraterno, Dirigente Nacional e Conselho), terão acesso as mesmas funcionalidades, sendo limitado a abrangência dos dados disponíveis pra cada um.

6 LIMITES E RESTRIÇÕES DA SOLUÇÃO

A solução terá seu desenvolvimento voltado para uso Web. Para esta instância do projeto optou-se em não contemplar o desenvolvimento específico para portáteis, não descartando a possibilidade do desenvolvimento futuro de tal melhoria.

7 TECNOLOGIAS UTILIZADAS

7.1 UML

Para modelar o sistema será utilizada a UML, linguagem visual para softwares construídos sob o paradigma orientado a objetos, que se tornou o padrão adotado pela indústria de engenharia de software (GUEDES, 2009). Para este fim utilizar-se-á a ferramenta CASE Astah em sua versão *Community* que oferece os recursos suficientes para suprir a amplitude do projeto.

7.2 Java Enterprise Edition - Java EE

A plataforma Java EE é o padrão utilizado para o desenvolvimento de aplicações em ambientes corporativos com Java. Esta plataforma provê aos desenvolvedores um poderoso conjunto de APIs, reduzindo o tempo de desenvolvimento, reduzindo a complexidade e aumentando a performance da aplicação.

Os componentes base e a plataforma independente desta tecnologia, fazem com que se torne fácil escrever aplicações Java EE, uma vez que a lógica de negócio está organizada em componentes reutilizáveis. Além disso, o servidor Java fornece serviços sob a forma de *container* para cada tipo de componente.

Na arquitetura Java EE a finalidade básica de um *container* é fornecer o ambiente de execução para os componentes. A especificação Java EE define um contrato entre componentes e *container* e especifica o modelo de desenvolvimento destes componentes. Este contrato especifica como estes componentes devem ser desenvolvidos e implementados e como os componentes podem acessar serviços fornecidos pelo *container*.

A especificação Java EE destaca cinco componentes ou *containers* suportados em um produto Java EE. São eles:

- **Java EE Server:** é a parte de execução de um produto Java EE. Um Java EE Server provê serviços EJB (*Enterprise Java Beans*) e *Web*

containers;

- **Enterprise Java Beans (EJB):** gerencia a execução de *enterprise beans* para aplicações Java EE. *Enterprise bean* e outros *containers* rodam dentro de um Java EE *Server*;
- **Web Container:** gerencia a execução de páginas, servlets e alguns componentes EJB para aplicações Java EE. Componentes Web rodam dentro do Java EE Server;
- **Aplicação Cliente:** gerencia a execução de componentes de aplicações cliente, que podem ser desenvolvidos em Swing, AWT ou outra tecnologia desktop, porém com acesso a todos os recursos do Java EE;
- **Applet:** gerencia a execução de Applets que consistem de um plug-in Java e um browser rodando juntos no lado cliente;

A figura 2 ilustra o servidor Java EE, seus containers e componentes.

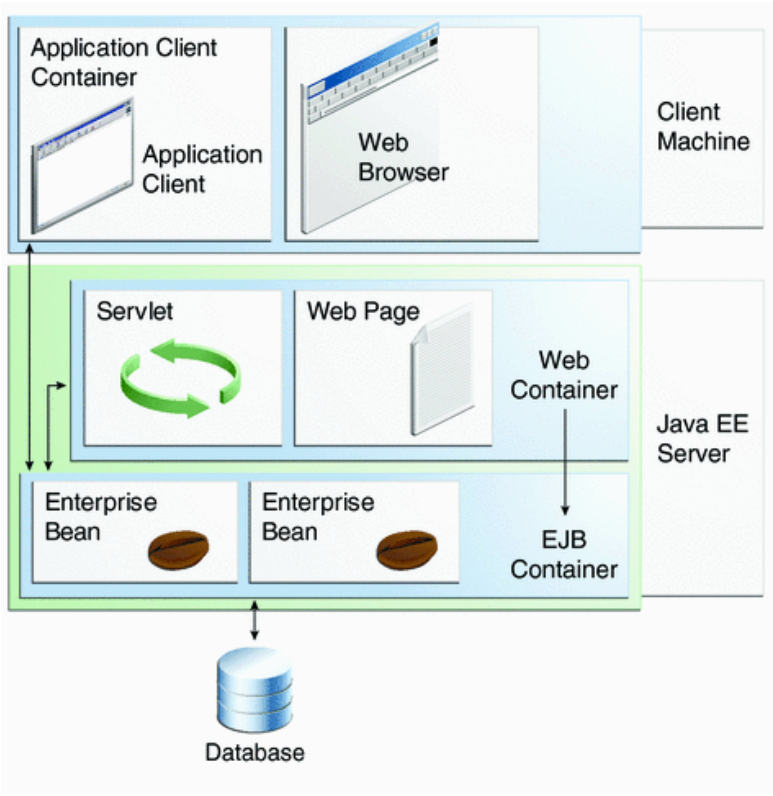


Figura 2 Java EE Server, Componentes e Containers
Fonte: http://download.oracle.com/javaee/6/tutorial/doc/figures/overview-serverAndContainers.gif.

Neste trabalho será abordado o *Web Container*, pois é nele que o sistema proposto está inserido.

Um *Web Container* tem a responsabilidade de instanciar, inicializar e invocar componentes como páginas JSP e Servlets, é ele o responsável pelo gerenciamento do ciclo de vida destes, os quais são abordados nas próximas sessões.

O *Web Container* faz parte de servidores Web ou servidores de aplicações, como exemplo pode-se citar o Tomcat ou JBoss, que fornecem serviços de rede através de requisições e respostas que são enviadas aos mesmos.

A seguir será falado sobre duas importantes tecnologias contidas na plataforma Java EE, são elas Servlets e JSP.

7.2.1 Servlets/JSPs

Servlets são classes Java, desenvolvidas de acordo com uma estrutura bem definida que quando instaladas e configuradas em um servidor que implemente um *Web Container*, podem tratar requisições recebidas de clientes Web, como por exemplo os Browsers.

Ao receber uma requisição, um Servlet pode capturar os parâmetros desta requisição, efetuar qualquer processamento inerente a uma classe Java, e devolver uma página HTML.

As páginas JSP, ou Java Server Pages, foram criadas para contornar algumas das limitações no desenvolvimento com Servlets: se em um servlet a formatação da página HTML resultante do processamento de uma requisição se mistura com a lógica da aplicação em si, dificultando a alteração dessa formatação, em uma página JSP essa formatação se encontra separada da programação, podendo ser modificada sem afetar o restante da aplicação.

Assim, um JSP consiste de uma página HTML com alguns elementos especiais, que conferem o caráter dinâmico da página. Esses elementos podem tanto realizar um processamento por si, como podem recuperar o resultado

do processamento realizado em um servlet, por exemplo, e apresentar esse conteúdo dinâmico junto à página JSP.

A utilização de servlets e de páginas JSP oferecem diversas vantagens em relação ao uso de outras tecnologias (como PHP, ASP e CGI). As principais vantagens são herdadas da própria linguagem Java, como:

- **Portabilidade:** a aplicação desenvolvida pode ser implantada em diversas plataformas, como por exemplo, Windows, Unix, GNU/Linux e Macintosh, sem que seja necessário modificar ou mesmo reconstruir a aplicação;
- **Facilidade de programação:** a programação é orientada a objetos, simplificando o desenvolvimento de sistemas complexos. Além disso, a linguagem oferece algumas facilidades, como por exemplo, o gerenciamento automático de memória (estruturas alocadas são automaticamente liberadas, sem que o desenvolvedor precise se preocupar em gerenciar esse processo);
- **Flexibilidade:** o Java já se encontra bastante difundido, contando com uma enorme comunidade de desenvolvedores, ampla documentação e diversas bibliotecas e códigos prontos, dos quais o desenvolvedor pode usufruir.

Além dessas vantagens, a arquitetura de servlets e páginas JSP possibilitam alguns benefícios adicionais, como:

- **Escalabilidade:** na maior parte dos servidores de aplicações modernos, é possível distribuir a carga de processamento de aplicações desenvolvidas em diversos servidores, sendo que servidores podem ser adicionados ou removidos de maneira a acompanhar o aumento ou decréscimo dessa carga de processamento;
- **Eficiência:** os servlets carregados por um servidor persistem em sua memória até que ele seja finalizado. Assim, ao contrário de outras tecnologias, não são iniciados novos processos para atender cada requisição recebida; por outro lado, uma mesma estrutura alocada em

memória pode ser utilizada no atendimento das diversas requisições que chegam a esse mesmo servlet;

- **Recompilação automática:** páginas JSP modificadas podem ser automaticamente recompiladas, de maneira que passem a incorporar imediatamente as alterações sem que seja necessário interromper o funcionamento da aplicação como um todo.

7.3 Spring MVC

O Spring MVC é o *framework* Web nativo do Spring que se baseia no padrão *Model-View-Controller* (MVC).

Conforme explica (Spring) a documentação do framework, o módulo Web do Spring contém inúmeros recursos, tais como:

- Separação clara de papéis;
- Adaptabilidade e flexibilidade;
- Código de negócio reutilizável sem a necessidade de duplicação;
- Customização de mapeamento de manipuladores e de resolução de visualização;
- Transferência flexível do *Model*;
- Uma biblioteca de *tags* JSP, introduzida no Spring 2.0, que torna a escrita de formulários em páginas JSP muito mais fácil.

Como a maioria dos *frameworks* MVC em Java, o Spring MVC é projetado em torno de um *DispacherServlet*. O *DispacherServlet* tem o papel de um *front controller*, que consiste em um modelo de aplicativo comum onde todas as requisições são passadas para um único servlet e este é responsável por delegar a requisição a servlets ou componentes de um outro aplicativo.

De acordo com a documentação de referência do Spring o *DispacherServlet* recebe todas as requisições e as despacha para manipuladores que geralmente são classes baseadas nas anotações *@Controller* e *@RequestMapping* que oferecem uma grande gama de flexíveis métodos que são responsáveis pelo tratamento da requisição.

Para que seja possível entender o funcionamento deste *framework* vamos falar do caminho que uma requisição percorre desde o momento em que ela deixa o navegador até o momento em que ela retorna com uma resposta ao usuário conforme representado na figura 3.

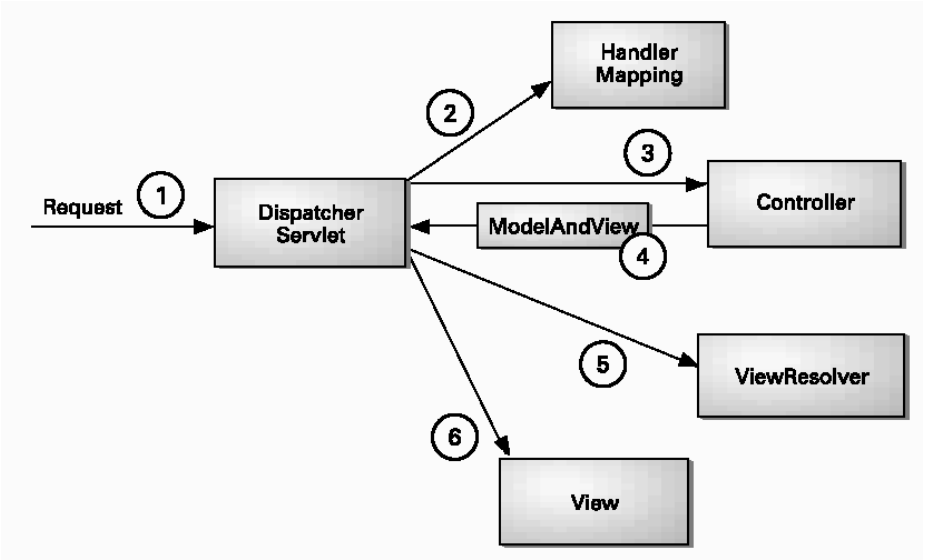


Figura 3 O ciclo de vida de uma requisição no Spring MVC

Fonte: http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/mvc.html

Todas as requisições devem passar pelo *DispacherServlet*, este, por sua vez determinará através da URL da requisição o *controller* apropriado para tal solicitação.

Após escolhido o *controller*, o mesmo irá tratar a requisição e fornecer os dados que serão mostrados ao usuário. Estes dados são chamados de modelo (*model*) e serão enviados para uma *view*, que é a responsável por exibir de forma amigável as informações ao usuário. O *controller* utilizará um objeto *ModelAndView* para enviar ao *DispacherServlet* os dados do *model* e o nome da *view*. Por fim, os dados do *model* serão entregues a *view* correspondente que geralmente será uma JSP (Java Servers Pages). Assim termina o ciclo de vida de uma requisição no Spring.

7.4 Java Persistence API

Na camada de persistência da aplicação foi utilizado o *Java Persistence API* - JPA, que é a especificação Java para o gerenciamento da camada de persistência e mapeamento objeto/relacional, facilitando desta forma este tipo de mapeamento em aplicações Java. Neste projeto esta API terá seu funcionamento integrado ao Spring.

O JPA lida com a forma como os dados relacionais são mapeados para objetos Java e como esses objetos são armazenados em um banco de dados relacional, para que estes possam ser posteriormente acessados. Lida também com a continuação da existência do estado de uma entidade mesmo após o término de um aplicativo. Além de simplificar o modelo de persistência de uma entidade o JPA padroniza o mapeamento objeto-relacional. O JPA é baseado nas ideias dos principais frameworks de persistência e API's como Hibernate, TopLink e o Java Data Objects - JDO.

Esta especificação consiste de quatro partes:

- A API Java de Persistência;
- A linguagem query;
- A API Critéria Java de Persistência;
- Os metadados para mapeamentos objeto/relacional.

O JPA está contido no pacote javax.persistence e utiliza como linguagem para suas sentenças a *Java Persistence query language* - JPQL. Neste trabalho foi utilizada sua versão 2.0 que é descrita pela JSR 317. Entre as propostas desta versão estão: a expansão do mapeamento objeto/relacional; alguns adicionais para o JPQL; contrato adicional para entidades e extensão dos contextos de persistência; suporte para validação com funcionamento do JSR 303.

A figura 4 mostra como é feito o acesso aos dados utilizando o JPA. Pode-se notar que é necessário escolher um provedor que irá implementar a especificação JPA, como Hibernate ou TopLink por exemplo. O provedor irá lidar com o *Java Database Connection*-JDBC que através de um driver irá acessar um determinado Sistema Gerenciador de Base de Dados - SGBD. Como exemplo de SGBD's, pode-se citar o PostgreSQL, MySQL, DB2, etc.

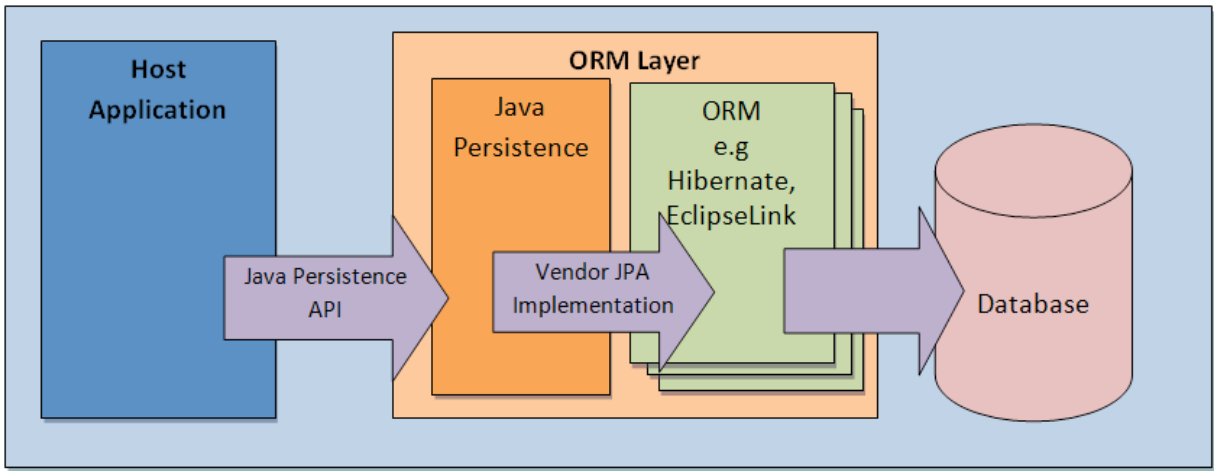


Figura 4 Aplicação utilizando JPA

Fonte: http://www.calcey.com/wp-content/uploads/2014/01/jpa-story-21.png

7.5 HIBERNATE

O Hibernate é um *framework* para o mapeamento objeto-relacional escrito na linguagem Java, mas também é disponível em .Net como o nome NHibernate. Este *framework* facilita o mapeamento dos atributos entre uma base tradicional de dados relacionais e o modelo objeto de uma aplicação, mediante o uso de arquivos (XML) ou anotações Java.

7.6 jQuery

jQuery é uma biblioteca de código aberto cuja sintaxe foi desenvolvida para tornar mais simples a navegação do documento HTML, a seleção de elementos DOM, criar animações, manipular eventos e desenvolver aplicações AJAX. A biblioteca também oferece a possibilidade de criação de plug-ins sobre ela. Fazendo uso de tais facilidades, os desenvolvedores podem criar camadas de abstração para interações de mais baixo nível, simplificando o desenvolvimento de aplicações Web dinâmicas de grande complexidade.

Principais funcionalidades do jQuery:

- Resolução da incompatibilidade entre os navegadores;
- Redução de código;
- Reutilização do código através de plug-ins;
- Utilização de uma vasta quantidade de plug-ins criados por outros desenvolvedores;
- Trabalha com AJAX e DOM;
- Implementação segura de recursos do CSS1, CSS2 e CSS3.

7.7 Tiles

Apache Tiles é um *framework* de composição de modelo de layouts. O Tiles foi originalmente criado para simplificar o desenvolvimento de interfaces de aplicativos Web em Java, mas não está mais restrito ao ambiente Web Java EE.

Tiles permite aos desenvolvedores definir fragmentos de páginas que podem ser montados como uma única página completa em tempo de execução. Estes fragmentos, ou tiles, pode ser utilizado como simples “includes” para reduzir a duplicação de elementos comuns entre as páginas ou incorporado dentro de outros tiles para desenvolver uma série de modelos reutilizáveis. Esses modelos agilizam o desenvolvimento, permitindo ter uma aparência consistente na aplicação como um todo.

7.8 Maven

O Apache Maven é uma poderosa ferramenta utilizada para gerenciar projetos Java. Com Maven temos todo o controle de compilação da aplicação, controle de bibliotecas, *deployment* e relatórios estatísticos. A configuração do Maven se baseia em um arquivo chamado pom.xml (*Project Object Model*), onde são declaradas todas as dependências do projeto. Depois de feita a configuração, o Maven se encarrega de analisar as dependências declaradas, fazer o download de todas as elas a partir de um repositório, e utilizá-las para compilar, empacotar e distribuir o artefato que pode ser um JAR, WAR ou EAR.

Algumas características

- Facilitar e unificar o processo de build;
- Fornecer mais qualidade de informação sobre o projeto;
- Permitir transparente migração para novas versões;
- Ajudar com boas práticas no desenvolvimento.

7.9 Git

Git é um sistema de controle de versão distribuído e um sistema de gerenciamento de código fonte, com ênfase em velocidade. O Git foi inicialmente projetado e desenvolvido por Linus Torvalds para o desenvolvimento do *kernel* Linux, mas foi adotado por muitos outros projetos.

Normalmente a maioria dos controles de versão guardam as mudanças do código como alterações de um determinado arquivo. Ou seja, a cada mudança no arquivo, o sistema guarda essa mudança apenas e não o arquivo inteiro.

O Git pensa um pouco diferente: ele trata os dados como snapshots. Cada vez que commitamos (commitar é enviar alterações para o controle de versão) ou salva o estado do projeto no Git, ele basicamente guarda um snapshot de como todos os arquivos estão naquele momento e guarda a referência desse estado. Para os arquivos que não foram modificados, ele não guarda uma nova versão, ele apenas faz um link para a versão anterior idêntica que já foi guardada em outro momento.

7.10 JasperReports e IReport

O JasperReports é um *framework* para a geração de relatórios. É uma ferramenta totalmente *open source* e gratuita, e a mais utilizada com esse propósito atualmente. Entre as funcionalidades do JasperReports podemos destacar:

- É capaz de exportar relatórios para diversos formatos diferentes, tais como PDF, HTML, XML, XLS, etc;
- Aceita diversas formas de entrada de dados, tais como um arquivo XML ou CSV, conexão com o banco de dados, uma sessão do Hibernate, uma coleção de objetos em memória, etc;
- Permite o uso de diagramas, gráficos, e até códigos de barras.

Um aspecto importante do JasperReports é que o layout do relatório é definido em um arquivo XML, geralmente com a extensão .jrxml. Este XML possui todas as informações de formatação do relatório, e além disso, possui os campos que serão preenchidos posteriormente, de acordo com a fonte de dados utilizada (data source). Como dito anteriormente, a fonte de dados pode variar, e ser uma tabela em uma base de dados, ou ser um arquivo CSV, porém a formatação do relatório será a mesma em ambos os casos.

Os passos para gerar um relatório são bem simples. O primeiro passo é compilar o relatório em XML. Depois da compilação, o resultado é um objeto do tipo JasperReport. O próximo passo é preencher o relatório com os dados, e o resultado dessa etapa fica armazenado em um objeto do tipo JasperPrint. Esse objeto já representa o relatório finalizado, a partir dele podemos enviar para impressão diretamente, ou podemos exportar para um outro formato, tal como PDF por exemplo. Veja um diagrama ilustrando o processo completo:

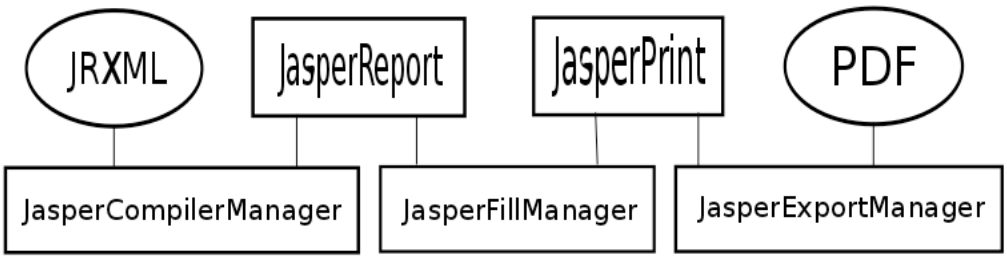


Figura 5 Processo de exportação para formato PDF

Fonte: http://www.k19.com.br/artigos/wp-content/uploads/2010/11/diagrama.png

O iReport é uma ferramenta desenvolvida pela mesma empresa do

JasperReports, a JasperForge, e por isso é muito comum ver os dois sendo usados em conjunto. Uma das dificuldades ao trabalhar com os relatórios, está na definição do layout. É complicado escrever o layout totalmente em XML, sem ter que se aprofundar em todas as *tags* e atributos possíveis, e além disso posicionar todos os elementos corretamente. Na prática, é muito raro alguém editar o JRXML manualmente, e sim apenas para fazer alguns pequenos ajustes quando necessários. O processo normal é utilizar alguma ferramenta para gerar o JRXML automaticamente, e o iReport é utilizado com esse propósito.

O iReport é um aplicativo gráfico, que permite que você “desenhe” um relatório, utilizando uma palheta, e arrastando e soltando componentes, de forma bem parecida com a criação de interfaces e janelas para programas. Ao salvar, automaticamente será gerado um JRXML que você poderá utilizar na aplicação que estiver desenvolvendo. A vantagem é que não é necessário que você conheça a fundo o XML a ser editado, economizando tempo de desenvolvimento. Ele também traz um conjunto pronto de modelos que você já pode utilizar diretamente, ou então, escrever seus próprios modelos e reaproveitá-los sempre que precisar criar um novo tipo de relatório.

8 ARQUITETURA DO SISTEMA

Para desenvolvimento do sistema foi escolhida a plataforma Heroku. O Heroku se enquadra na categoria de serviços da computação em nuvem conhecida como Plataforma como Serviço (Platform as a Service, ou PaaS), no qual o fornecedor entrega para o cliente um ambiente pronto para receber a aplicação. Diferente do IaaS (Infraestrutura como Serviço), no qual cliente contrata máquinas (reais ou virtuais) e é responsável pela instalação de bibliotecas, montagem das estruturas do sistema de arquivos, entre outros recursos, o PaaS é uma solução de alto nível que abstrai este tipo de preocupação.

Como o ambiente é entregue pelo fornecedor, ao cliente basta se concentrar em desenvolver e instalar a aplicação. Normalmente nos serviços PaaS a instalação ou atualização é feita através de commits em repositórios remotos vinculados à aplicação.

No mercado, atualmente, existem muitas opções de fornecedores de serviços de PaaS além do próprio Heroku. Podemos citar como exemplo o RedHat OpenShift, AWS Elastic Beanstalk, Jelastic, CloudBees, dentre outros. Basicamente o que difere cada um são as tecnologias que eles permitem utilizar, como linguagens ou containers, quantidades de serviços adicionais e, é claro, a precificação, ou seja, o custo por memória, CPU e disco de cada instância da aplicação. Outra variável é a facilidade de criação e instalação de um aplicativo. Neste ponto o Heroku está um passo à frente dos outros fornecedores, pela simplicidade que é criar, manter e escalar uma aplicação.

O Heroku trabalha com projetos modelos, que podem ser clonados em um repositório Git, e a partir dessa personalização já é possível ter acesso a sua aplicação em endereço específico.

Ao fazer commit das suas alterações, através de um processo de integração contínua, o Heroku, já compila a sua aplicação, se for o caso, e já promove a implantação no subdomínio escolhido.

No caso da linguagem Java, o Heroku fornece quatro modelos de projetos, sendo um projeto com o Web Container Jetty, e uso de Jax-rs para fornecer serviços Web REST, um projeto Spring MVC usando o Web Container Tomcat, um projeto usando o framework Play e um projeto JSP/Servlet para uso

com Jetty.

Os projetos usam Maven para automatizar o processo de gerenciamento das bibliotecas, compilação do projeto e implantação no servidor, permitindo ao desenvolvedor se preocupar apenas com a codificação do projeto.

A figura 6, mostra a arquitetura do heroku, mostrando que o desenvolvedor pode codificar em sua linguagem, fazer o deploy via Git/Maven e gerenciar a aplicação através das interfaces fornecidas pelo heroku. Já o usuário terá acesso a aplicação através de uma interface Web criada pelo desenvolvedor ou através do uso de API. O acesso do usuário é de responsabilidade da aplicação do desenvolvedor, e o heroku fica responsável pelo balanceamento das chamadas e roteamento da aplicação.

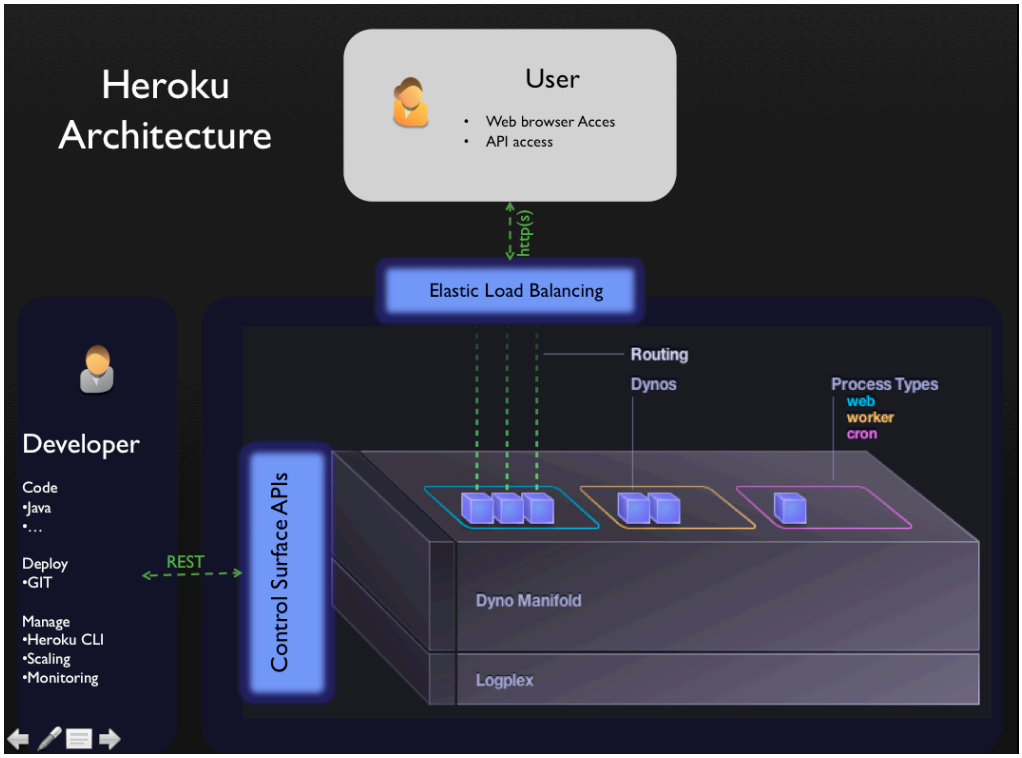


Figura 6 Arquitetura do Heroku

O Heroku fornece toda infraestrutura ao desenvolvedor, de forma transparente para o usuário.

https://developer.salesforce.com/page/Webinar:_Social_Enterprise_Java_Apps_on_Heroku_(2012-May)

9 METODOLOGIA DE DESENVOLVIMENTO

A escolha do Scrum se deu por ser uma metodologia ágil para gerência de projetos. Ela é baseada em ciclos de uma a quatro semanas , esse ciclo é chamado Sprint, no qual se trabalha para alcançar objetivos bem definidos para a Sprint (SABBAGH, 2013).

Os objetivos gerais de um projeto que utiliza Scrum são representados no *Product Backlog*, uma lista de coisas para fazer que é constantemente atualizada e repriorizada. Ao se iniciar uma Sprint é criado a Sprint Backlog, contendo os itens que serão desenvolvidos no ciclo.

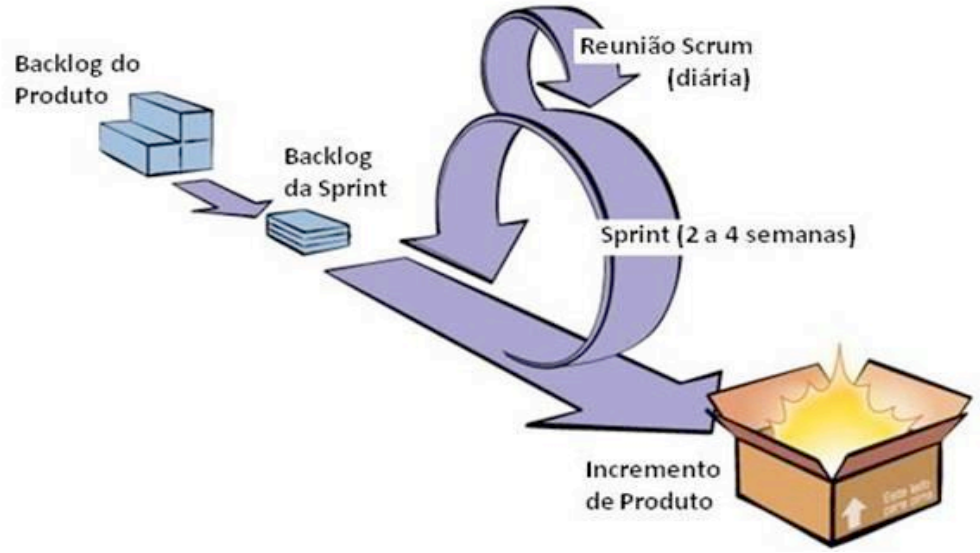


Figura 7 Ciclo do Scrum

Algumas de suas principais características visando minimizar os riscos são:

- Divisão do projeto em pedaços menores e curtos, variando de uma semana a um mês;
- Comunicação entre a equipe e o cliente em tempo real;
- Participação ativa do cliente;
- Alta capacidade de resposta a mudanças;
- Entregas continuas e incrementais, facilitando a validação do que está

sendo desenvolvido realmente atende a necessidade;

Os princípios Scrum fornecem a “habilidade de declarar o produto “pronto” sempre que necessário (porque a concorrência acabou de entregar, porque a empresa precisa de dinheiro, porque o usuário/cliente precisa das funções, porque foi para essa data que foi prometido...)”

Os princípios Scrum são usados para guiar as atividades de desenvolvimento dentro de um processo que incorpora as seguintes atividades de arcabouço: requisitos, análise, projeto, evolução e entrega. Em cada atividade de arcabouço, as tarefas ocorrem dentro de um padrão de processo chamado de Sprint. O trabalho é conduzido dentro um Sprint (a quantidade de Sprint necessária para cada atividade de arcabouço varia dependendo da complexidade e do tamanho do produto) é adaptado ao problema em mãos e é definido e frequentemente, modificado em tempo real pela equipe Scrum.

Para esse projeto trabalharemos com alguns conceitos do Scrum, a saber:

Pendência: uma lista priorizada de requisitos ou características do projeto que fornecem valor de negócio para o cliente. Itens podem ser adicionados à pendência a qualquer momento (é assim que as modificações são introduzidas). O gerente de produto avalia a pendência e atualiza as prioridades quando necessário.

Eventos Scrum: Eventos no Scrum são reuniões rotineiras, desta maneira minimizando a chance de ocorrer reuniões não programadas. Toda reunião no Scrum tem o conceito de Time-boxed, isso significa que ela tem uma duração fixa, não podendo levar mais, ou menos tempo, caso ocorra de levar mais ou menos tempo que o preciso, a reunião deve ser finalizada e na reunião de Retrospectiva da Sprint deve ser analisado os motivos que o mesmo veio a ocorrer. Segue abaixo os Eventos que o Scrum atende.

9.1 Eventros Scrum

9.1.1 Sprints

Consiste de unidades de trabalho que são necessárias pra satisfazer

a um requisito definido na pendência que precisa ser cumprido em um intervalo de tempo predefinido (de 2 a 4 semanas). Durante o Sprint, os itens em pendência a que a unidade de trabalho do Sprint se destinam são congelados (isto é, não são introduzidas as modificações durante o Sprint). Assim, o Sprint permite que os membros da equipe trabalhem em um ambiente de curto prazo, mas estável.

9.1.2 Reunião Diária

A Reunião Diária do Scrum é um evento time-boxed de 15 minutos, para que o Time de Desenvolvimento possa sincronizar as atividades e criar um plano para as próximas 24 horas. Esta reunião é feita para inspecionar o trabalho desde a última Reunião Diária, e prever o trabalho que deverá ser feito antes da próxima Reunião Diária.

9.1.3 Revisão da Sprint

A Revisão da Sprint é executada no final da Sprint para inspecionar o incremento e adaptar o *Product Backlog* se necessário. Durante a reunião de Revisão da Sprint o Time Scrum e as partes interessadas colaboram sobre o que foi feito na Sprint. Com base nisso e em qualquer mudança no *Product Backlog* durante a Sprint, os participantes colaboram nas próximas coisas que podem ser feitas para otimizar valor. Esta é uma reunião informal, não uma reunião de status, e a apresentação do incremento destina-se a motivar e obter comentários e promover a colaboração.

9.1.4 Retrospectiva da Sprint

A Retrospectiva da Sprint é uma oportunidade para o Time Scrum inspecionar a si próprio e criar um plano para melhorias a serem aplicadas na próxima Sprint.

9.2 Scrum Solo

Para o desenvolvimento desse sistema usaremos uma adaptação do Scrum para uso em equipes de apenas um desenvolvedor, onde as práticas Scrum o auxiliam a planejar e executar todo processo de desenvolvimento de forma ágil e sem sustos durante o projeto.

Para nossa proposta utilizaremos a *Product Backlog*, o *Sprint Backlog*, os Sprints e a Retrospectiva de Sprint. Em substituição as reuniões diárias, montaremos um diário com as informações de andamento. As Sprints terão prazos de 15 dias e a cada 30 dias o produto será validado junto ao grupo de validação, no nosso caso os clientes do sistema. Quinzenalmente o aluno participa de uma Oriented Meeting – OM (reunião de orientação – aluno, professor orientador). Para avaliação dos clientes será definido um modelo de formulário para registrar o seu feedback. Esse formulário será definido na OM.

O processo é alicerçado pela planta de desenvolvimento (artefatos gerados durante a construção do software, por exemplo: diagramas, casos de teste, etc.) e pela atividade de gerenciamento de projeto. Os artefatos que a planta de desenvolvimento deve conter e a forma a ser utilizada para a gestão são definidos pelos participantes da OM.

Figura 8 Adaptação para o Scrum Solo

10 CRONOGRAMA INICIAL

O cronograma de desenvolvimento contempla seis Sprints distribuídas no prazo de seis meses, sendo estes de maio a outubro de 2014.

A organização cronológica foi concebida de maneira a suportar a metodologia de desenvolvimento iterativa e incremental.

A última iteração é destinada à revisão do sistema construído, permitindo realizar os ajustes finais necessários. Inclui também os testes e implantação do sistema.

As iterações têm como objetivo o desenvolvimento dos seguintes itens:

- Iteração 1: Cadastro de Entidades e Pessoas
- Iteração 2: Cadastro de Institutos, Comissões e Atividades
- Iteração 3: Cadastro de Plano Modelo, Rodizio
- Iteração 4: Contratação de Metas e Relatórios
- Iteração 5: Segurança
- Iteração 6: Ajustes e correções, testes integrados e implantação.

Tarefa/mês	**Maio**				**Junho**				**Julho**				**Agosto**				**Setembro**				**Outubro**			
	1ª	2ª	3ª	4ª	1ª	2ª	3ª	4ª	1ª	2ª	3ª	4ª	1ª	2ª	3ª	4ª	1ª	2ª	3ª	4ª	1ª	2ª	3ª	4ª
Interação 1																								
Interação 2																								
Interação 3																								
Interação 4																								
Interação 5																								
Interação 6																								
Texto																								

Tabela 3 Cronograma

11 REFERÊNCIAS BIBLIOGRÁFICAS

GUEDES, Gilleanes T. A. **UML 2:** uma abordagem prática. São Paulo: Novatec, 2009.

ORACLE Disponível em: <http://docs.oracle.com/javaee/7/firstcup/doc/java-ee002.htm>. Acesso em 01 jul. 2014.

SABBAGH, Rafael. SCRUM: Gestão Ágil para Projetos de Sucesso. São Paulo: Casa do Código, 2013.

SPRING Disponível em: <http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html>. Acesso em 01 jul. 2014.

SOMMERVILLE, Ian. Engenharia de Software. 8. ed. Addison-Wesley, 2007.

WHERTHEIN, Jorge. A Sociedade da Informação e seus Desafios. Revista Ciência da Informação, Brasília, v. 29, n. 2, maio/ago. 2000. Disponível em: < http://revista.ibict.br/ciinf/index.php/ciinf/article/view/254/1705>. Acesso em 01 jul. 2014.

ANEXO I – Ficha exemplo de contratação de metas.

Instituição:		
Coordenador:	Cidade:	Comissão: MOCIDADE
	E-mail:	Tel.:

ATIVIDADES	2012		2013		2014		2015		2016		DIFICULDADES PARA IMPLANTAÇÃO
	P	I	P	I	P	I	P	I	P	I	
1-Instituto do Jovem											
a) Conselho do Instituto											
b) Reunião Mensal											
2-Escola para Formação de Instrutores de Jovens											
a) Reunião mensal de Instrutores											
b) Acompanhamento dos alunos na mediunidade e na prática											
c) Cursos:											
1. Noções Básicas de Evang. Juvenil											
2. Trabalhando com o Jovem											
3 Adolescência um desafio p/pais e Educadores											
4. Reforma Íntima											
5. O jovem e a Caridade											
3-Mocidade											
a) Equipe de Instrutores											
b) Coordenadores de Áreas											
• Secretaria											
• Doutrina											
• Prática											
• Alegria Cristã											
• Oficinas											
• Apoio Fraterno											
c)Parte Pedagógica											
• Planejamento Semanal											
• Cursos para Nível III											
• Cursos para Nível IV											
d) Prática											
• Laborterapia											
• C.E Humberto de Campos											

AVALIADOR: Na coluna P assinale com um "X" se a atividade fora planejada para o respectivo ano. Caso nenhum planejamento foi feito, deixe o quadrinho em branco. Na coluna "I" assinale com um "X" se a atividade foi implantada no respectivo ano. Caso a implantação não tenha ocorrido, deixe o quadrinho em branco. A primeira cópia da ficha deve ser entregue para a Secretaria e a segunda cópia para o dirigente do Centro Espírita.

ANEXO II – TUTORIAL DE CRIAÇÃO DE APLICATIVO NO HEROKU.

E

O HEROKU

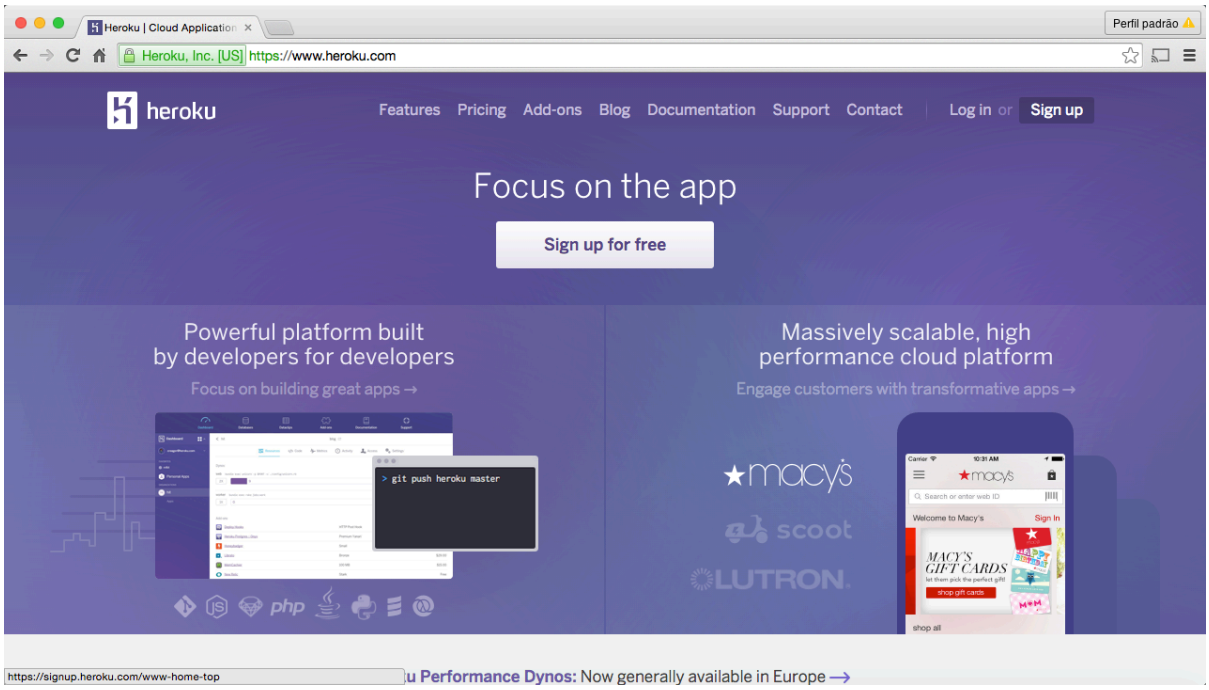
O Heroku, é uma plataforma de computação na nuvem, que oferece um PaaS (Plataforma como Serviço). Nela é possível iniciar gratuitamente um projeto utilizando uma das linguagens suportadas pela aplicação.

Nesse pequeno tutorial, vamos mostrar os passos básicos para ter sua aplicação rodando diretamente no heroku, utilizando a linha de comando. Nosso foco será em Java, mas no Heroku, é possível trabalhar em outras linguagens como Ruby, Node.js, Python, Django e outras opções.

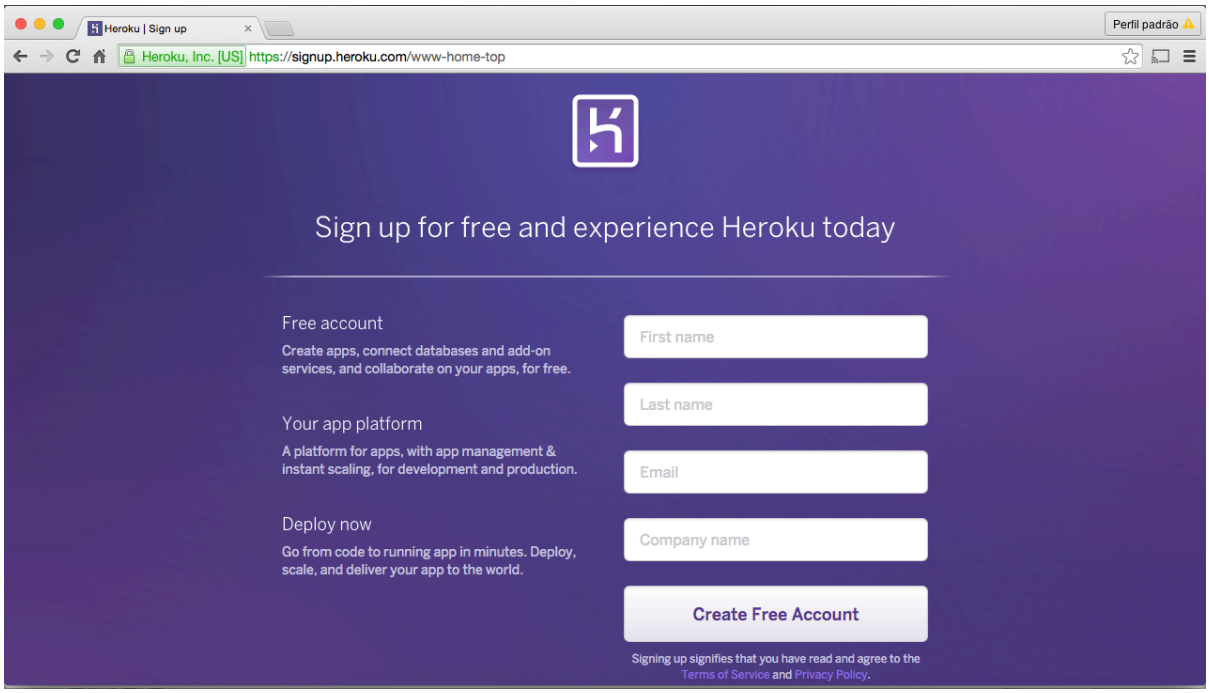
É possível também instalar os plug-ins necessários e fazer a integração facilmente com o Eclipse, mas deixaremos esse assunto para uma próxima oportunidade.

CADASTRO DA CONTA NO HEROKU

Para ter acesso ao serviços fornecidos pelo Heroku, é preciso realizar o cadastro em seu site http://www.heroku.com/, e no centro da página acione o botão ***"Sign up for free".***



O formulário de cadastro é bem simples, onde são necessários apenas seu nome, seu email e uma empresa caso faça parte de alguma. Preencha-os com seus dados e acione no botão **"Create Free Account".**



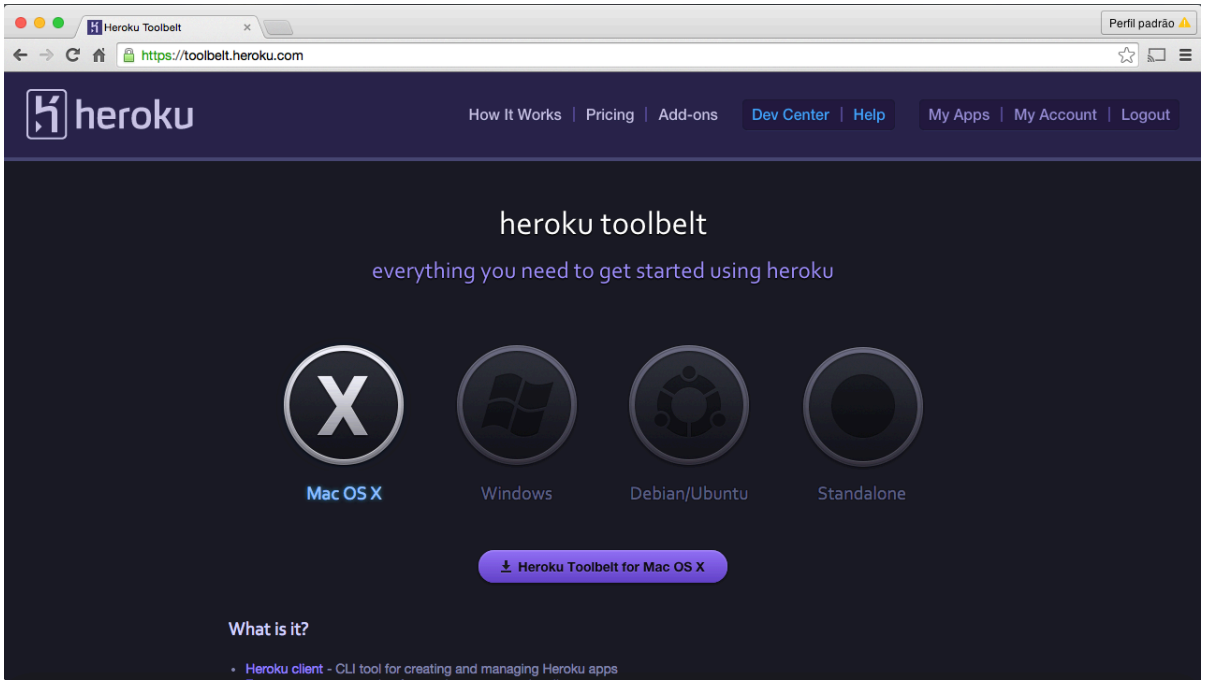
Após esse processo, você receberá um e-mail confirmando a criação de sua conta. Nele também haverá um link, para que você possa comprovar a propriedade do email utilizado durante o cadastro. Basta clicar no link e você será redirecionado para uma página do heroku que ativará automaticamente sua conta.

Baixar o Heroku ToolBelt

Tendo a sua conta ativada, vamos iniciar o processo de desenvolvimento de nossa aplicação no Heroku, precisaremos ter o SDK do Java, o Apache Maven e o Heroku Toolbelt instalados em nosso computador. Nessa sessão vamos ver sobre a instalação do Heroku Toolbelt.

O Heroku Toolbelt é um conjunto de aplicativos em linha de comando para gerenciamento de sua aplicação.

É possível baixá-lo a partir do site: http://toolbelt.heroku.com.



Configurando

Nesta etapa, você irá instalar o Heroku Toolbelt. Isto proporciona-lhe o acesso ao utilitário de linha de comando Heroku, bem como Git e Foreman, ferramentas que você vai usar em etapas posteriores.

Uma vez instalado, você pode usar o comando “heroku” do seu shell de comando.

Faça login usando o endereço de e-mail e senha que você usou quando criou sua conta Heroku:

```
$ heroku login
Enter your Heroku credentials.
Email: java@example.com
Password:
Could not find an existing public key.
Would you like to generate one? [Yn]
Generating new SSH public key.
Uploading ssh public key /Users/java/.ssh/id_rsa.pub
Press enter at the prompt to upload your existing ssh key or create a new one, used for pushing code later on.
```

Para verificar se a chave foi adicionada, digite heroku keys. Se a chave não estiver lá, você pode adicioná-la manualmente, digitando heroku keys:add.

Preparando o aplicativo

Nesta etapa, você irá preparar um aplicativo simples que pode ser implantado.

Execute os seguintes comandos para clonar o aplicativo de amostra:

```
$ git clone https://github.com/heroku/java-getting-started.git
$ cd java-getting-started
```

Agora você tem um repositório Git funcionando que contém um aplicativo simples, bem como um arquivo pom.xml, que é usado pelo gestor de dependências Maven.

Implantar o aplicativo

Nesta etapa, você irá implantar a aplicação no Heroku.

Use o comando “heroku create”, que prepara o Heroku para receber o seu código-fonte:

```
$ heroku create
Creating warm-eyrie-9006... done, stack is cedar
http://warm-eyrie-9006.herokuapp.com/ | git@heroku.com:warm-eyrie-9006.git
Git remote heroku added
```

Isso também cria um repositório remoto (chamado heroku), e configura-o em seu repositório Git local. O Heroku gera um nome aleatório (neste caso warm-eyrie-9006) para seu aplicativo - você pode passar um parâmetro para especificar o seu próprio, ou renomeá-lo mais tarde com o comando heroku:rename.

Agora basta implantar o seu código:

```
```

```
$ git push heroku master
Initializing repository, done.
Counting objects: 68, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (19/19), done.
Writing objects: 100% (68/68), 7.07 KiB | 0 bytes/s, done.
Total 68 (delta 22), reused 65 (delta 22)

-----> Java app detected
-----> Installing OpenJDK 1.7... done
-----> Installing Maven 3.0.3... done
-----> executing /app/tmp/cache/.maven/bin/mvn -B -Duser.home=/tmp/build_4244c199-7f9b-4f62-
bf60-bbd0aff3f978 -Dmaven.repo.local=/app/tmp/cache/.m2/repository -DskipTests=true clean
install
       [INFO] Scanning for projects...
       [INFO]
       [INFO] ------------------------------------------------------------------------
       [INFO] Building helloworld 1.0-SNAPSHOT
       [INFO] ------------------------------------------------------------------------
       Downloading:        http://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-
dependency-plugin/2.4/maven-dependency-plugin-2.4.pom

...

-----> Discovering process types
       Procfile declares types -> web

-----> Compressing... done, 62.7MB
-----> Launching... done, v7
       http://warm-eyrie-9006.herokuapp.com/ deployed to Heroku

To git@heroku.com:warm-eyrie-9006.git
 * [new branch]      master -> máster
```

A aplicação agora está implantada. Certifique-se de que pelo menos uma instância da aplicação está sendo executada, com o comando abaixo:

```
$  heroku ps:scale web=1
```

Agora, visite a aplicação na URL com o nome gerado. Como um atalho, você pode abrir o seu site da seguinte forma:

```
$ heroku open
```

Ver registros

O Heroku trata os registros na forma de fluxos de eventos ordenados cronologicamente, agregando os fluxos de toda a sua aplicação e componentes do Heroku, proporcionando um canal único para todos os eventos gerados.

Veja as informações sobre o seu aplicativo em execução usando um dos comandos de log, heroku logs:

```
$ heroku logs --tail
2014-08-08T13:55:07.254053+00:00 heroku[web.1]: State changed from starting to up
2014-08-08T13:55:05.214125+00:00 heroku[web.1]: Starting process with command `java -Xmx384m
-Xss512k -XX:+UseCompressedOops -cp target/classes:target/dependency/* Main`
```

```
2014-08-08T13:56:08.662643+00:00 heroku[router]: at=info method=GET path="/" host=warm-eyrie-
9006.herokuapp.com    request_id=f99b4889-1b16-4862-876d-4f752ad3d843    fwd="94.174.204.242"
dyno=web.1 connect=1ms service=3ms status=200 bytes=579
```

Visite o seu aplicativo no navegador novamente, e você verá outra mensagem de log gerada. Pressione Control + C para interromper o fluxo de logs.

Definir um Procfile

Você pode usar um Procfile, que é um arquivo de texto no diretório raiz de sua aplicação, para declarar explicitamente qual comando deve ser executado para iniciar a sua aplicação.

O Procfile presente na aplicação de exemplo que você implantou deve parecer com isso:

```
web: java $JAVA_OPTS -cp target/classes:target/dependency/* Main
```

Isto declara um único processo do tipo Web, e o comando necessário para executá-lo. O nome Web é importante aqui. Ele declara que este tipo de processo será anexado à pilha de roteamento HTTP do Heroku, e receberá o tráfego da Web, quando implantado.

Procfiles podem conter tipos de processo adicionais. Por exemplo, você pode declarar um processo em segundo plano que processa itens fora de uma fila.

Escalonar a aplicação

Neste momento, sua aplicação está sendo executado em um único dyno Web. Pense em um dyno como um contêiner leve que executa o comando especificado no seu arquivo Procfile.

É possível verificar quantas dynos estão rodando com o comando ps:

```
$ heroku ps
=== web (1X): `java $JAVA_OPTS -cp target/classes:target/dependency/* Main`
web.1: up 2014/08/08 14:55:07 (~ 2m ago)
```

Tendo um único web dyno em execução fará com que este se desligue automaticamente depois de uma hora de inatividade. Isso gera um atraso de alguns segundos na primeira requisição realizada após esse período. As solicitações subseqüentes a este pedido funcionarão normalmente.

Para evitar isso, você pode escalar mais de um dyno web. Por exemplo:

```
```

```
$ heroku ps:scale web=2
```

Para a prevenção do abuso, quando você escala o aplicativo, pode ser requisitado uma verificação da sua conta. Se a sua conta ainda não foi verificada, você será direcionado para visitar o site de verificação.

Para cada aplicação, Heroku oferece 750 horas/dyno gratuitas. Executando o seu aplicativo em 2 dynos ultrapassaria esse subsídio mensal gratuito, então escalemos de volta:

```
$ heroku ps:scale web=1
```

Declarar dependências de aplicativo

Heroku reconhece um aplicativo Java como pela existência de um arquivo pom.xml no diretório raiz. Por seus próprios aplicativos, você pode criar um executando mvn clean install.

O aplicativo de demonstração você já implantado tem um pom.xml. Aqui está um trecho:

```
<dependencies>
    <dependency>
      <groupId>org.eclipse.jetty</groupId>
      <artifactId>jetty-servlet</artifactId>
      <version>7.6.0.v20120127</version>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>servlet-api</artifactId>
      <version>2.5</version>
    </dependency>
</dependencies>
```

O arquivo pom.xml especifica dependências que devem ser instalados com o aplicativo. Quando um aplicativo é implantado, Heroku lê esse arquivo e instala as dependências usando o comando mvn clean install.

Outro arquivo, system.properties, determina a versão do Java para usar. O conteúdo deste arquivo, que é opcional, é bastante simples:

```
java.runtime.version=1.7
```

Execute mvn clean install no seu diretório local para instalar as dependências, preparando o sistema para executar o aplicativo localmente:

```
$ mvn clean install
INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building helloworld 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
```

```
...
Downloading:    http://repo.maven.apache.org/maven2/org/apache/maven/maven-archiver/2.5/maven-archiver-2.5.pom
Downloaded:     http://repo.maven.apache.org/maven2/org/apache/maven/maven-archiver/2.5/maven-archiver-2.5.pom (5 KB at 14.9 KB/sec)
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 5.584s
[INFO] Finished at: Fri Aug 08 15:05:39 BST 2014
[INFO] Final Memory: 19M/222M
[INFO] ------------------------------------------------------------------------
```

Uma vez que as dependências estejam instaladas, você estará pronto para executar sua aplicação localmente.

Execute a aplicação localmente

Agora inicie a sua aplicação localmente usando Foreman, que foi instalada como parte do Toolbelt:

```
$ foreman start web
15:10:15 web.1  | started with pid 2071
15:10:15 web.1  | 2014-08-08 15:10:15.329:INFO:oejs.Server:jetty-7.6.0.v20120127
15:10:15     web.1          |     2014-08-08     15:10:15.378:INFO:oejsh.ContextHandler:started o.e.j.s.ServletContextHandler{/,null}
15:10:15     web.1          |     2014-08-08     15:10:15.411:INFO:oejs.AbstractConnector:Started SelectChannelConnector@0.0.0.0:5000
```

Assim como Heroku, Foreman examina o Procfile para determinar o que executar.

Seu aplicativo será agora funcionando em http://localhost:5000. Teste que ele está trabalhando com a onda ou um navegador web, em seguida, Ctrl-C para sair.

Foreman não apenas executa sua aplicação, ele também define as "config vars", algo que você vai encontrar em um tutorial mais tarde.

ANEXO III – COMPARATIVO ENTRE OS PAAS TESTADOS

Comparação Heroku e Openshift

Existe hoje no mercado diversos fornecedores de PaaS (Plataforma como Serviço), cada com um leque de recursos diferentes. Após testar vários serviços diferentes optei por dois serviços, o Heroku, serviço fornecido pela SalesForce.com, e o Openshift, que é um serviço da RedHat. Vou utilizá-los para exemplificar os recursos disponíveis na maioria dos PaaS e compará-los através um conjunto de perguntas nos auxiliarão na escolha do serviço que melhor atenda ao nossos objetivos.

Escolha da Linguagem:

A primeira de nossas perguntas é: “Quais linguagens este ou aquele fornecedor de PaaS suportam?”

Hoje temos fornecedores de PaaS que atendem as principais linguagens de uso na Web como PHP, Java, Python ou Ruby, como também tecnologias mais inovadoras como AngularJS, NodeJS entre outros.

Por isso é preciso ter em mente em qual linguagem meu aplicativo será escrito para escolher as opções que atendam a minha necessidade.

No caso do Heroku, ele permite que sua aplicação seja escrita em Ruby, Python, Node.js, PHP, Java (Tomcat, Jetty), Scala, Play e Clojure.

Já o Openshift, permite Ruby, Python, Node.js, PHP (Zend), Java (JBoss/Tomcat), Perl e Vert.x.

Para o meu projeto os dois fornecedores atendiam a minha necessidade: Java rodando no Tomcat.

Além da linguagem é preciso ter certeza se o fornecedor lhe atende com a compatibilidade de frameworks que precisará.

Quantidade de Aplicativos

Nossa segunda pergunta deve ser: “Quantos aplicativos ou sistemas poderemos criar em uma conta, gratuitamente?

Alguns fornecedores de PaaS não limitam o número de aplicativos por conta (e-mail), permitindo assim que cada aplicativo seja totalmente independente. Assim você é capaz de gerenciar os recursos de cada aplicativo de forma isolada, pagando pelo que cada aplicativo usa em excesso.

Em outro modelo de fornecimento, os fornecedores de PaaS limitam o número de aplicativos por conta, dando a você uma quantidade limitada de máquinas virtuais. Um alguns deles você pode usar as máquinas virtuais como blocos de montar, em que você pode usar cada bloco como uma aplicação separada, ou alocar dois ou mais blocos para escalonar uma única aplicação.

O Heroku utiliza o primeiro conceito. Cada conta criada tem direito a criar quantos aplicativos quiser, e seu domínio será sempre <nome_app>.herokuapp.com. Sua aplicação é tratada de forma totalmente independente. Tanto que você pode transferir a propriedade de uma aplicação para outra pessoa, desvinculando da sua conta. Cada aplicativo terá disponível uma máquina (chamada por eles de dyno) com 512MB de memória RAM e 1GB Swap e 750 horas de cpu. Você pode comprar uma nova dyno especializada para escalonar sua aplicação, ou multiplicar os recursos de sua dyno.

Outra particularidade sobre a sua aplicação é que no heroku não há sistema de arquivo, estão não é possível persistir nada em disco, sendo necessário utilizar apis ou o próprio serviço da Amazon de armazenamento de arquivos.

Ja no Openshift, no seu cadastro inicial, lhe é fornecido uma conta gratuita com a possibilidade de criar até 3 aplicações não escalonadas utilizando suas 3 “gears”. Cada gears da conta gratuita vem com 512 MB de memória RAM e 1GB de espaço em disco. Assim é possível escalonar sua aplicação para usar as 3 gears, ou criar cada aplicação em separado. Dessa

forma caso queira uma quarta aplicação é preciso pagar por uma gear adicional. É possível também mudar o tipo da gear, aumentando assim a memória e o espaço em disco utilizado. Uma outra alternativa é contratar espaço em disco adicional.

No openshift cada projeto recebe um nome criado por você acrescido de um sufixo criado para sua conta ficando <projeto>-<sufixo>.rhcloud.com. Assim se sua conta é empresa01, e seu aplicativo é aplicativo01, o endereço de seu aplicativo será aplicativo01-empresa01.rhcloud.com.

Banco de Dados

Quais bancos são suportados pelo fornecedor do PaaS? Nesse ponto diversos bancos estão disponíveis, sendo que a principal diferença esta na forma como o banco é fornecido, já que alguns PaaS oferecem apenas um tipo de Servidor de Banco de Dados por conta, sendo necessário adquirir outro a parte. É importante saber se o banco é compatível com sua aplicação.

O Heroku fornece apenas banco de dados em sua conta padrão que é o PostgreSQL. Outros bancos como MongoDB, MySQL estão disponíveis para aquisição via add-ons.

O banco fornecido na conta gratuita do Heroku, é chamado de Hobby Dev. Nesse banco existem algumas limitações como por exemplo o número de linhas que não pode ultrapassar 10.000. O número de conexões simultâneas também não pode superar as 20 conexões. Não está disponível o rollback do banco. Por um valor pequeno mensal, é possível mudar a limitação de linhas para 10.000.000, mas mantendo todas as demais restrições.

O Openshift, nos fornece o MySQL e o PortgreSQL como opções de banco de dados, sem custo adicional. Tudo já está incluso no espaço e no processamento de sua Gear. Através de sua página ele não deixa muito claro quais as reais limitações que os bancos fornecem.

Conexão SSL

Outro fator importante a ser levado em consideração é a necessidade de conexões segura usando SSL. Se você precisar usar transações em seu aplicativo e descobrir que seu provedor não dispõe desse recurso?

O Heroku fornece esse recurso através de um add-on, com um custo mensal.

Já o Openshift fornece o certificado a nível de subdomínio de forma automática, através de um certificado SSL compartilhado por subdomínio de rhcloud.com. Também está disponível a opção de usar o certificado por domínio através de um add-on.

Conclusão

O Heroku é uma excelente solução para o Desenvolvedor. Sua interface é bem tranquila e suas ferramentas muito práticas. Traz uma grande quantidade de add-ons que podem ser adicionados ao seu projeto.

O Openshift ainda tem pouco add-ons, mas fornece por default mais funcionalidades que o Heroku, como MySQL, PostgreSQL, Jenkins CI e outros. A possibilidade de crescimento horizontal também facilita muito o escalonamento. Um ponto importante é o espaço em disco para sua aplicação e o recurso de acesso via SSH.

Para a minha aplicação escolhi o Heroku. Mas já penso em mudar para o Openshift.

ANEXO IV – FICHAS UTILIZADAS NO RODIZIO

Instituição:		
Coordenador:	Cidade:	Comissão: MOCIDADE
	E-mail:	Tel.:

ATIVIDADES	2012		2013		2014		2015		2016		DIFICULDADES PARA IMPLANTAÇÃO
	P	I	P	I	P	I	P	I	P	I	
1-Instituto do Jovem											
a) Conselho do Instituto											
b) Reunião Mensal											
2-Escola para Formação de Instrutores de Jovens											
a) Reunião mensal de Instrutores											
b) Acompanhamento dos alunos na mediunidade e na prática											
c) Cursos:											
1. Noções Básicas de Evang. Juvenil											
2. Trabalhando com o Jovem											
3 Adolescência um desafio p/pais e Educadores											
4. Reforma Íntima											
5. O jovem e a Caridade											
3-Mocidade											
a) Equipe de Instrutores											
b) Coordenadores de Áreas											
• Secretaria											
• Doutrina											
• Prática											
• Alegria Cristã											
• Oficinas											
• Apoio Fraterno											
c)Parte Pedagógica											
• Planejamento Semanal											
• Cursos para Nível III											
• Cursos para Nível IV											
d) Prática											
• Laborterapia											
• C.E Humberto de Campos											

AVALIADOR: Na coluna P assinale com um "X" se a atividade fora planejada para o respectivo ano. Caso nenhum planejamento foi feito, deixe o quadrinho em branco. Na coluna "I" assinale com um "X" se a atividade foi implantada no respectivo ano. Caso a implantação não tenha ocorrido, deixe o quadrinho em branco. A primeira cópia da ficha deve ser entregue para a Secretaria e a segunda cópia para o dirigente do Centro Espírita.