UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DEPARTAMENTO ACADÊMICO DE INFORMÁTICA

CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

RENAN DE PAULA ROSA

ROBINSON VICENTE RONDON MACHADO

SISTEMA PARA CLASSIFICAÇÃO DE ESTUDANTES À SEREM CONTEMPLADOS NO PROGRAMA BOLSA-PERMANÊNCIA DA UTFPR UTILIZANDO *FRAMEWORK GRAILS* E *IREPORT*: DESENVOLVIMENTO E PROPOSTA DE IMPLANTAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA

2013

RENAN DE PAULA ROSA

ROBINSON VICENTE RONDON MACHADO

SISTEMA PARA CLASSIFICAÇÃO DE ESTUDANTES À SEREM CONTEMPLADOS NO PROGRAMA BOLSA-PERMANÊNCIA DA UTFPR UTILIZANDO *FRAMEWORK GRAILS* E *IREPORT*: DESENVOLVIMENTO E PROPOSTA DE IMPLANTAÇÃO

Trabalho de Conclusão de Curso apresentada à coordenação de informática como requisito parcial para obtenção do grau de Tecnólogo no curso superior de tecnologia em análise e desenvolvimento de sistemas da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Marcos Vinicius Fidelis.

PONTA GROSSA

2013

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Ponta Grossa

Diretoria de Graduação e Educação Profissional

TERMO DE APROVAÇÃO

SISTEMA PARA CLASSIFICAÇÃO DE ESTUDANTES À SEREM CONTEMPLADOS NO PROGRAMA DE BOLSA-PERMANÊNCIA DA UTFPR UTILIZANDO FRAMEWORK *GRAILS* E IREPORT: DESENVOLVIMENTO E PROPOSTA DE IMPLANTAÇÃO.
por

RENAN DE PAULA ROSA
e
ROBINSON VICENTE RONDON MACHADO

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 27 de março de 2013, como requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas. Os candidatos foram arguidos pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Prof. Dr. Marcos Vinícius Fidelis
Prof. Orientador

_____
Prof. Dr. Daniel Costa de Paiva
Membro titular

_____
Prof. Msc. Wellton Costa de Oliveira
Membro titular

_____
Profª. Drª. Helyane B. Borges
Responsável pelos Trabalhos
de Conclusão de Curso

_____
Profª. Drª. Simone de Almeida
Coordenadora do Curso
UTFPR - Campus Ponta Grossa

- O Termo de Aprovação assinado encontra-se no Departamento Acadêmico de Informática -

EPÍGRAFE

Your work is going to fill a large part of your life, and the only way to be truly satisfied is to do what you believe is great work. And the only way to do great work is to love what you do. If you haven't found it yet, keep looking, and don't settle. As with all matters of the heart, you'll know when you find it. (JOBS, Steve).

Seu trabalho vai preencher uma parte grande da sua vida, e a única maneira de ficar realmente satisfeito é fazer o que você acredita ser um bom trabalho. E a única maneira de fazer um excelente trabalho é amar o que você faz. Se você ainda não encontrou, continue procurando, e não se acomode. Como em todos os assuntos do coração, você saberá quando encontrar. (JOBS, Steve).

AGRADECIMENTOS

Agradecimentos do Renan.

Quero agradecer primeiramente a Deus por tudo que tem me dado, pelas experiências de vida, pelo conhecimento recebido, pelos desafios que ele me impôs e ajudou a solucionar.

Agradeço a minha mãe Marli, meu pai Ednilson, meus irmãos Dieyson e Monique, e ao resto da minha vasta família, da qual tenho recebido apoio e amor durante todos esses anos.

Agradeço também a UTFPR por me proporcionar novos amigos, como meus colegas e professores, que contribuíram com sua amizade, conhecimento e inspiração.

Agradecimentos do Robinson.

Agradeço aos meus pais e meus irmãos por sempre me apoiarem, passando confiança, motivação e tendo muita compreensão para com minha pessoa. À minha namorada por todo incentivo e paciência tido nos tempos bons e ruins.

Agradeço aos bons amigos que também sempre me apoiaram e nunca me deixaram sozinho, sempre que precisei, estavam ao meu lado. Aos companheiros de faculdade, enfrentando juntos a caminhada árdua de chegar até o final, mesmo com alguns tropeços, sempre de cabeça erguida.

Minha terna gratidão por todos aqueles que não desistiram de mim e sempre colaboraram por mais esta vitória.

Agradeço a Deus, acima de tudo, por mais esta conquista.

Agradecimentos em conjunto.

Ao Prof. Marcos Vinicius Fidelis que nos orientou no desenvolvimento deste trabalho.

Aos nossos colegas de curso e demais professores que compartilharam conhecimento e contribuíram para nossa formação.

RESUMO

ROSA, Renan de Paula; MACHADO, Robinson Vicente Rondon. **Sistema para classificação de estudantes à serem contemplados no programa de bolsa-permanência da UTFPR utilizando framework *Grails* e iReport:** desenvolvimento e proposta de implantação. 2013. 89 f. Trabalho de Conclusão de Curso – curso superior de tecnologia em análise e desenvolvimento de sistemas, Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2013.

Existem hoje no mercado diversas ferramentas que nos auxiliam no processo de desenvolvimento de sistemas de informação, essas ferramentas visam prover soluções pré-estabelecidas, como funcionalidades previamente implementadas. Os *Frameworks* funcionam como uma caixa de ferramentas, das quais podemos usufruir satisfatoriamente se soubermos como elas trabalham. Há diversos *frameworks* no mercado atual, e é muito importante sabermos qual escolher para nos ajudar no desenvolvimento de projetos. Para testar a eficiência do *framework Grails* na resolução de um problema real, o programa Bolsa-Permanência da UTFPR foi escolhido. Este processo é todo manual, leva cerca de um mês para ser finalizado e não possui um número de pessoas que trabalham na seleção compatível com a quantidade de informações à serem processadas. Este trabalho propõe a criação de um sistema *Web* desenvolvido em *Grails*, para informatizar o processo do programa Bolsa-Permanência da UTFPR, visando demonstrar a eficiência do *framework Grails*, bem como melhorar o processo do Bolsa-Permanência. Em conjunto ao uso do *Grails*, utilizou-se para geração de relatórios o *framework ireport*.

Palavras-Chave: *Grails*, *Framework*, *iReport*, *Groovy*, UTFPR.

ABSTRACT

ROSA, Renan de Paula; MACHADO, Robinson Vicente Rondon. System for classification of students to be included in the scholarship program-permanence UTFPR using *Grails* framework and iReport: proposal development and deployment. In 2013. 89 f. Completion of course work - college of technology analysis and development of systems, Federal Technological University of Paraná. Ponta Grossa, 2013.

There are several tools on the market today that assist us in the development of information systems, these tools provide solutions aimed at pre-established, as previously implemented features. The *Framework*s serve as a toolbox from which we can enjoy if we know how well they work. There are several *framework*s on the market today, and it is very important to know which one to choose to help us in developing projects. To test the efficiency of the *Grails framework*, we tried to use it to solve a real problem, the problem was chosen Grant-Stay of UTFPR, this program aims to select students for obtaining a scholarship. This whole process is manual, it takes considerable time to complete and does *not* have a number of people working in selecting compatible with the amount of information to be processed. This paper proposes the creation of a *web*-based system developed in *Grails*, to computerize the process of Grant-Stay of UTFPR, to demonstrate the efficiency of the *Grails framework* and improve the process of Grant-Stay. In conjunction with the use of *Grails* was used for reporting the *framework ireport*.

Keywords: *Grails, Framework, iReport, Groovy*, UTFPR.

LISTA DE FIGURAS

LISTA DE SIGLAS

API	Application Programming Interface
CRUD	Create, Read, Update and Delete
DRY	Don’t Repeat Yourself
GORM	*Grails* Object Relational Mapping
HQL	Hibernate Query Language
IDE	Integrated Development Environment
JDK	*Java* Development Kit
JPA	*Java* Persistence API
JSF	*Java* Server Faces
JVM	*Java* Virtual Machine
MVC	Model, View, Controller
NUAPE	Núcleo de Acompanhamento Psicopedagógico e Assistência Estudantil
ORM	Object Relational Mapping
PDF	Portable Document Format
PHP	Personal *Home* Page
SQL	Strutured Query Language
UTFPR	Universidade Tecnológica Federal do Paraná
XML	Extensible Markup Language

Sumário

1 INTRODUÇÃO

Agilidade, facilidade e precisão são qualidades que garantem a qualidade de determinados processos nas mais diversas organizações. Para Laudon e Laudon (1998), os Sistemas de Informação também estão fornecendo aos indivíduos novas ferramentas para melhorar suas vidas e de suas comunidades, tornando os serviços mais ágeis e eficazes. Com o auxílio da tecnologia obtemos essas qualidades de maneira mais fácil, os computadores nos ajudam nos processos, tornando-os mais rápidos, fáceis e precisos. A tecnologia da informação está nos mais diversos setores da economia, auxiliando processos desde os mais simples até os mais complexos. Automatizando processos, reduzimos custos, tempo, de execução, capital humano, etc. Mas para tanto, é necessário escolher rigorosamente as ferramentas que irão nos ajudar a desenvolver o nosso trabalho, devemos levar em consideração diversos fatores na hora da escolha, como: facilidade de uso, desempenho, dificuldade de aprendizagem, e até mesmo, devemos levar em consideração a facilidade de podermos substituir eventualmente, a ferramenta escolhida por uma mais nova.

Conforme Solomon (1986), uma dada tecnologia não é automaticamente boa ou má para a pequena empresa. Seu resultado dependerá da maneira como esta tecnologia será aplicada. Na verdade, o aumento da precisão organizacional, auxiliada por sistemas de informação, trará maior eficiência na administração de seus processos, recursos e atividades e maior eficácia na obtenção de resultados previamente estabelecidos. Com a crescente disponibilidade de recursos que a Internet oferece aos seus usuários, as aplicações na *web* tornam-se mais atrativas. O desenvolvimento de aplicações *web* utilizando a plataforma *Java* EE tem crescido nos últimos anos e proporcionado novos *framework*s e bibliotecas.

A plataforma *Java* possui uma arquitetura para aplicações *web* bastante rica, porém muito complexa, por tentar atender a diferentes situações e propósitos. Essa complexidade, somadas a diversas opções de *framework*s, trazem alguns problemas à plataforma *Java*, tornando-a pouco produtiva comparada a outras tecnologias *web*. Pode-se perceber na prática alguns problemas, como perda no tempo para configurar uma aplicação, incompatibilidade de alguns *framework*s entre si devido as suas versões, exaustivas repetições em criações de CRUD's (*Create, Read, Update e Delete*), configuração de logs, internacionalização, acesso aos dados, relatórios, etc.

Logo, os desenvolvedores *Java* nunca conseguiram um nível de produtividade exigido pelos princípios e práticas ágeis. Esta falta de agilidade fez com que linguagens dinâmicas como Ruby e Python crescessem e tivessem cada vez mais adeptos, dentre os quais se destacavam os que seguem as metodologias ágeis (JUDD; NUSAIRAT; SHINGLER, 2008). Com o intuito de suprir estas deficiências surgiu o *Groovy*, uma linguagem de programação estável, rica de recursos que manteve a compatibilidade com a API existente no *Java*, que busca o que faltava para a plataforma: flexibilidade e produtividade.

1.1 OBJETIVOS

Nesta seção seguem o objetivo geral e os objetivos específicos do trabalho.

1.1.1 Objetivo Geral

Desenvolver uma aplicação *Web* para o programa de Bolsa Permanência da UTFPR, com o propósito de auxiliar e agilizar o processo de cadastro de alunos no programa de Bolsa Permanência, utilizando ferramentas ágeis de desenvolvimento de *software*.

1.1.2 Objetivos Específicos

- Para tornar possível a realização deste trabalho, pretende-se atingir os seguinte objetivos específicos:
- Estudar sobre utilização de *framework*s, *Grails*, linguagem de programação *Groovy* e a ferramenta *ireport*.
- Analisar o problema proposto.
- Desenvolver o sistema utilizando o *framework Grails*.
- Avaliar o sistema proposto com o usuário.

1.2 ORGANIZAÇÃO DO TRABALHO

Este trabalho está divididos em 6 capítulos. O capítulo 2 apresenta o referencial teórico. O capítulo 3 apresenta o sistema proposto. O capítulo 4 apresenta os experimentos feitos. O capítulo 5 a conclusão e trabalhos futuros. O capítulo 6 mostra as referências utilizadas para desenvolver este trabalho.

2 Referencial Teórico

2.1 MVC

Antes de entrar em detalhes sobre o padrão MVC, um dos padrões usados em nosso sistema, observar-se-á um pouco sobre padrões de projeto.

Um padrão descreve uma solução para um problema que ocorre com frequência durante o desenvolvimento de *software*, podendo ser considerado como um par "problema/solução" (Buschmann, 96). Projetistas familiarizados com certos padrões podem aplicá-los imediatamente a problemas de projeto, sem ter que redescobri-los (Gamma, 95). Um padrão é um conjunto de informações instrutivas que possui um nome e que capta a estrutura essencial e o raciocínio de uma família de soluções comprovadamente bem sucedidas para um problema repetido que ocorre sob um determinado contexto e um conjunto de repercussões (Appleton, 97).

Padrões de projetos descrevem soluções para problemas recorrentes no desenvolvimento de *software* orientado a objetos. Um padrão de projeto estabelece um nome e define o problema, a solução, quando aplicar esta solução e quais suas consequências quando usá-los.

Agora que foi apresentado o que é e para que serve os padrões de projetos, pode-se entender o padrão de projeto conhecido como MVC ou *Model, View* e *Controller* (modelo, visão, controlador).

MVC é composto por três tipos de objetos. O modelo é o objeto de aplicação, a visão é a apresentação na tela e o controlador define a maneira como a interface do usuário reage às entradas do mesmo. Antes do MVC, os projetos de interface para o usuário tendiam em agrupar esses objetos. A MVC separa esses objetos para aumentar a flexibilidade e a reutilização. (GAMMA et al. 2000, p. 20).

Já para GONÇALVES, 2007, MVC é um conceito (paradigma) de desenvolvimento e design que tenta separar uma aplicação em três partes distintas. Uma parte, a Model, está relacionada ao trabalho atual que a aplicação administra outra parte a *View* está relacionada a exibir os dados ou informações dessa uma aplicação e a terceira parte, Controller, em coordenar os dois anteriores exibindo a interface correta ou executando algum trabalho que a aplicação precisa completar.

Conhecendo os conceitos do padrão MVC, irá ser mostrado cada uma das partes de sua aquitetura isoladamente.

- Model: Segundo Trygve (1979) O modelo é uma representação ativa de uma abstração em forma de dados em um sistema de computação. Pode-se dizer que é o coração da aplicação. Representa as informações do domínio da aplicação e contém as funcionalidades do mesmo. É neste que definimos as regras de negócio de nossa aplicação, a persistência, comunicação com outros sistemas.
- *View* (Camada de Visão): esta camada é responsável por renderizar os dados trazidos pela camada do modelo de dados. Também representa a parte do sistema em que o usuário interage com a aplicação em questão.

  Controller (camada de controle): responsável por orquestrar os eventos de manipulação da interface gráfica pelo usuário e traduzi-los como chamadas à lógica de negócio, armazenada na camada de modelo.

A figura 1 apresenta o funcionamento do padrão MVC. Um bom exemplo seria um sistema de cadastro de cliente: o usuário inicia o sistema e logo na tela principal aparece um formulário para cadastrar um novo cliente. Esta tela representa a *View*, ou camada de visão. Logo após a inicialização do sistema, o usuário preenche os dados para cadastrar um novo cliente e clica em "Salvar", por exemplo. Ao início desta ação, seus dados são enviados para o controlador, que os receberá e transformará em chamadas à camada de negócio (*model*). Estas chamadas são executadas pela camada de modelo que irá retornar agora que já vimos os resultados para o controlador. Desta vez a camada de modelo transformará novamente, se houver necessidade, e retornará os dados para a camada de visão, para que haja a interação com o usuário. Um bom exemplo deste retorno seria uma pesquisa de um cliente que o usuário precisará de um retorno do sistema. Este ciclo está sendo representado na figura 1.

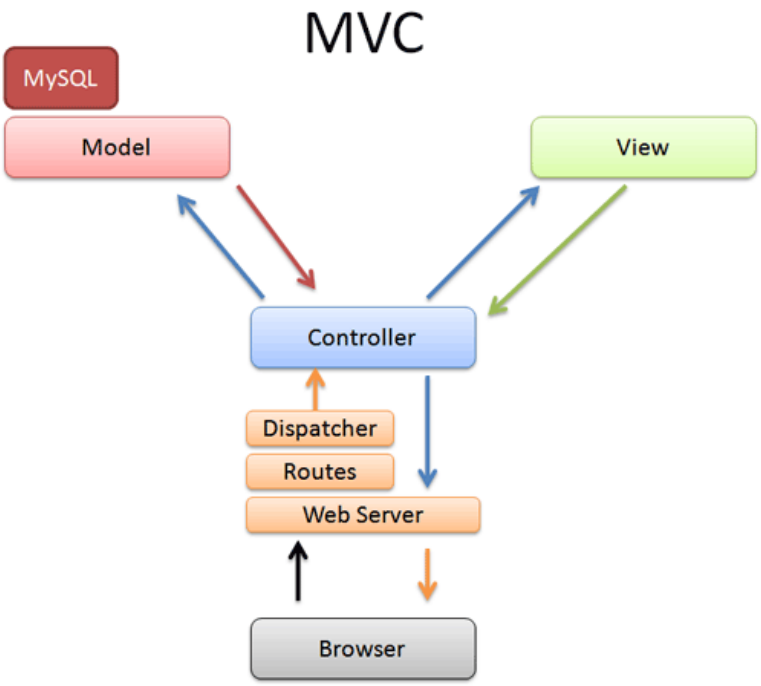

Figura 1: Arquitetura MVC

Fonte: wiki.cercomp.ufg.br

2.1.1 Quais as Vantagens e Desvantagens de Usar o Padrão MVC?

O uso do padrão MVC exige um estudo aprofundado para que seja possível a sua utilização de forma correta e trazendo todas as vantagens que nele apresenta.

Há projetos que são extremamente simples e que não passam de formulários comunicando-se com o banco de dados. Para tais projetos, o padrão MVC acaba trazendo desvantagens consigo, logo esta aplicação não precisa de uma regra de negócio mais estruturada nem mesmo muitas visões para o usuário, e o uso deste padrão gera bastante complexidade, que talvez seja desnecessária.

Porém, até onde esta pedra no caminho é uma desvantagem?

Pois bem, muitas vezes em projetos dados como pequenos, ainda devem ser levado em consideração que os requisitos dos clientes são bastante flexíveis e que a aplicação num dado momento é pequeno sofre acréscimos de várias outras funcionalidades e acaba tornando-se uma aplicação gigantesca.

Segundo Basham, Sierra e Bates (2005) a aplicação do padrão MVC possibilita o desacoplamento das partes do programa deixando-o modularizado. O sistema é decomposto

em componentes com funções específicas e independentes uns dos outros, aumentando assim, a coesão dos mesmos.

As características do padrão MVC torna o projeto mais reutilizável devido ao fraco acoplamento, pois não há dependência entre os componentes. Acrescentando ainda que, ao se usar uma aplicação que segue o padrão MVC, é aceitável dizer que foi abordado as melhores práticas de pessoas experientes nesta aplicação.

Ainda para Basham, Sierra e Bates (2005) o MVC aumenta a complexidade da aplicação pelo fato de, ao aumentar a coesão dos componentes individualmente, muitos novos componentes são necessários na aplicação. Em contrapartida, ele minimiza os impactos das mudanças feitas na aplicação. A divisão em componentes pode aumentar o overhead de comunicação entre os mesmos.

Usando o padrão MVC o desenvolvimento da aplicação, inicialmente, pode-se aumentar e ficar bastante complexo. Em contrapartida a esta pequena dificuldade, o rendimento no desenvolvimento da aplicação aumentará devido à possibilidade de se terem equipes divididas desenvolvendo em paralelo cada parte da aplicação em questão e, também, tem-se a possibilidade de melhor entendimento entre as equipes que estão trabalhando no projeto, pois ao adotar um padrão, o vocabulário entre as equipes é o mesmo. Assim sendo, pode-se dizer que, ao entrar um novo membro para a equipe de desenvolvimento, este terá um melhor desempenho, pois o projeto em questão está usufruindo de um padrão de projeto.

2.2 GROVY

James Strachan teve a *ide*ia de criar uma linguagem de programação baseada e mais robusta que *Java*, e que tivesse a elegância de Ruby ou Python. Mais tarde James conheceu Bob McWhirter, e juntos, fundaram o projeto *Groovy*, em 2003.

Linguagem com maior índice de crescimento rápido do Tiobe Index, da estaca zero para o 38º lugar dentre 200 linguagens de programação, com apenas 36 meses de lançamento. Atualmente, ocupa o 44º colocação e é a linguagem número 1 em uso dentre as linguagens da *Java* Virtual *Machine* (JVM), superando cerca de 240 linguagens compatíveis a JVM.

Groovy oferece uma sintaxe flexível, parecida com *Java*, que a maioria dos desenvolvedores *Java* podem aprender em questão de horas. *Groovy* fornece características observadas em outras linguagens dinâmicas como Ruby, Python ou Smalltalk. Baseia-se nos pontos fortes do *Java*, torna as características modernas de programação disponíveis para os

desenvolvedores *Java* com curva de aprendizado quase zero, suporta linguagens específicas de domínio e sintaxe compacta, que torna o seu código fácil de ler e manter. Aumenta a produtividade do desenvolvedor, reduzindo códigos no desenvolvimento *web*, GUI, aplicações de banco de dados ou console, simplifica os testes, apoiando o teste de unidade e out-of-the-box. Se integra perfeitamente com todas as classes *Java* existentes e bibliotecas, compila diretamente para *bytecode Java* para que você possa usá-lo em qualquer lugar você pode usar *Java*.

Groovy foi criada para ser muito parecida com *Java*, através disso, os desenvolvedores que já conhecem *Java* podem se adequar a linguagem de uma maneira descomplicada, sem precisar aprender uma nova linguagem do zero. Uma aplicação *Java* válida pode ser utilizada como aplicação *Groovy*, com a mesma sintaxe. Ou seja, se você criar uma objeto .jar e alterá-lo para *.groovy*, o objeto será reconhecido pelo *Groovy*.

No *Groovy*, algumas bibliotecas são importadas automaticamente para o projeto, sem a necessidade da declaração de importação das mesmas, prática que torna o código mais limpo, e sem as possíveis referências não utilizadas, que podem ficar no código dos programadores mais desatentos. Algumas classes já importadas são representadas na figura 2:

```
java.io.*
java.lang*
java.math.BigDecimal
java.math.BigInteger
java.net.*
java.util.*
groovy.lang.*
groovy.util.*
```

Figura 2: Classes importadas no *Groovy*
Fonte: Autoria Própria

Moderadores de Acesso: Se não declaramos o tipo do moderador de acesso no *Groovy*, ele interpretará que esse moderador é do tipo public, ou seja, o método declarado com este moderador é público e pode ser chamado a partir de métodos contidos em qualquer outra classe. Esta é a condição de menor restrição possível.

A palavra reservada return é de uso opcional, devido que o *Groovy* interpreta a última linha de instrução do método como sendo a instrução que deverá ser retornada ao método que o invocou.

O famoso erro por esquecimento de ponto e vírgula está com seus dias contados, devido ao *Groovy* não precisar da inserção dessa pontuação para representar o final de uma instrução, entretanto, para os programadores mais tradicionais, o uso de ponto e vírgula continua sendo permitido. Além do ponto e vírgula, os parênteses também não são mais obrigatórios.

O *Groovy* interpreta valores do tipo *null* (nulos) como sendo falsos na lógica booleana, dessa maneira, pode-se utilizar essas variáveis na verificação de condições verdadeiras ou falsas.

Os Servlets são vastamente usados em aplicações *Java Web*, e possuem diversas funções dentro de aplicações *Web*, como a manipulação do fluxo de dados da aplicação representa a camada Controller do padrão MVC (Model, *View*, Controller). Os Servlets podem ser tornar uma classe difícil de compreender, devido à alta responsabilidade atribuída à eles em uma aplicação, para isso, *Groovy* surgiu com o conceito de *Controllers*, ou seja, tem as mesmas responsabilidades dos Servlets, entretanto, sua sintaxe é muito mais clara.

Quando o Controller é gerado, recebe métodos que irão executar instruções pré-determinadas, que podem ser editadas pelo desenvolvedor, são elas:

Nome	Parâmetros	Descrição
List		Responsável pos listar os registros presentes na base de dados relacionados à classe de domínio em questão.
Create		Redireciona o usuário ao formulário de inserção de novos registros na base de dados
Edit	id (obrigatório)	Busca na base de dados o registro cuja chave primária possua o valor passado pelo parâmetro id. O encontrando, será feito o redirecionamento do usuário ao formulário de edição dos

		dados. Caso contrário, o usuário será redirecionado à *action list*
Show	id (obrigatório)	Assim como a *action edit*, *show* buscará na base de dados do registro cuja chave primária corresponde ao parâmetro id, expondo a página de detalhes do registro caso seja encontrado. Caso contrário, o usuário será redirecionado à action list.
Delete	id (obrigatório)	Como o nome sugere, exclui o registro da base de dados e redireciona o usuário a *action list*
Save	presente nos formulários relativos à entidade	Executada após submissão do formulário presente na *View* relacionado à action *create*. Possui a função de incluir um novo registro na base de dados
update	presente nos formulários relativoàentidade. id (obrigatório)	Assim como a *action save*, é executada após os dados presentes no formulário terem sido submetidos.Possui a função de atualizar os registros na base de dados.

Quadro 1: Actions Default do *Grails*

Fonte: Adaptada *Java* Magazine, Edição 77

Esses métodos ajudam e muito a vida do desenvolvedor, que poderá customizá-los da maneira que quiser.

O *Controller* é o responsável por toda a Lógica da Aplicação, é ele que faz o fluxo de dados fluir, é ele que atribui responsabilidades às classes da aplicação e também é ele que gerencia acessos e redirecionamentos.

Quando criam-se variáveis ou objetos em uma aplicação *Java*, deve-se obrigatoriamente criar os métodos *Sets*, se assim desejar alterá-los através de uma outra classe, ou então, criar os métodos *Gets* para pegar quaisquer informações inerentes ao objeto ou variável criada, consequentemente, o código se torna maior e deselegante.

No *Groovy*, quando criam-se as variáveis ou objetos, os métodos *Sets* e *Gets* são gerados automaticamente, e o melhor, ficam transparentes na classe para o qual foram

criados, o que torna o código muito mais limpo do que se comparar ao jeito tradicional de *Java*.

Em *Groovy*, as *Strings* são tratadas em duas classes distintas: String e GStrings, que partilham de algumas características comuns, mas se diferem em outras.

No contexto geral, Strings e GStrings são totalmente mutáveis, podem ser delimitadas por aspas duplas ou simples, podem ocupar uma ou várias linhas, pode-se inserir diretamente nelas operações e variáveis, como mostra a figura 3:

```
'Hello World'

"Hello World"
```

Figura 3: *Strings e GStrings* no *Groovy*
Fonte: Autoria Própria

Para um conjunto de caracteres que ocupam múltiplas linhas, deve declará-las com 3 aspas duplas, antes e depois do conjunto de caracteres:

```
"""Hello
World"""
```

Figura 4: *GStrings* no *Groovy*
Fonte: Autoria Própria

GStrings: Permitem a inserção de operações e variáveis diretamente dentro delas:

```
Em Groovy:

print "1 + 1 = \${1 + 1}"

Em Java:

print("1 + 1 = " + ( 1 + 1 ));
```

Figura 5: *GStrings* em *Groovy*
Fonte Autria Própria

É visível que em *Java* essa operação se torna um tanto quanto mais complicada do que em *Groovy*, o que pode eventualmente, confundir o desenvolvedor.

O *Groovy* permite a utilização de operadores aritméticos em Strings:

```
Concatenação:

a = "World"
print "Hello" + a

Multiplicação

a = "Hello"
print a * 2
```

Figura 6: Operadores Aritméticos no *Groovy*
Fonte: Autoria Própria

É importante ressaltar que necessita-se invocar o método toString() para converter uma GString para *String*, agora de *String* para GString não há a necessidade, devido a essa classe interpretar perfeitamente a classe String.

- *Closures:* Conjunto de instruções que se assemelham às classes internas em *Java*.
- *Constraints*: a nova regra de negócios: Representa a camada de regra de negócios da aplicação, é através dela que definem-se e validam-se os valores aceitos pela aplicação. As *Constraints* poupam um tempo enorme do desenvolvedor, devido a facilidade na qual são implementadas, contém um vasto leque de *plugins*, desenvolvidos para validar e restringir diversos tipos de dados.

2.3 REUTILIZAÇÃO DE CÓDIGO

O reuso é necessário para todo e qualquer tipo de problema a ser solucionados. Na medida que soluções são encontradas, estas são utilizadas em outros problemas similares.

A capacidade de abstração do *home*m garante adaptações necessárias ao novo contexto encontrado.

O problema na reutilização de código para os desenvolvedores está na resistência em que eles tem para ler o código a ser reutilizado. Para os desenvolvedores é mais fácil começar do zero do que "aprender" o código já escrito. Mas será isso verdade ou essa resistência acaba

atrapalhando as equipes de desenvolvimento? Provavelmente eles estão errados. Percebem-se, logo abaixo, alguns benefícios de reutilização de código para poder, assim sendo, comprovar as vantagens de se fazer isso.

- Melhores índices de produtividade - quando um programador reusa um código, ele ganha um grande tempo na produtividade do projeto, pois não precisa reescrever o código e sim reutilizá-lo, apenas adaptando-o ao novo projeto.
- Produtos de melhor qualidade, mais confiáveis, consistentes e padronizados - um código a ser reutilizado, já está em funcionamento em algum outro projeto. Sendo assim, ele já está testado, avaliado e com correções de *bugs*, que podem ter surgidos anteriormente.
- Redução dos custos e tempo envolvidos no desenvolvimento de *software* - tendo um melhor índice de produtividade, o projeto estará pronto em menos tempo e, então, terá uma despesa menor em comparação de um projeto começado do zero.
- Conformidade aos Padrões - um código já escrito estará, teoricamente, em conformidade com os padrões de projetos, facilitando o entendimento para manutenção do projeto.
- Desenvolvimento acelerado - o projeto terá uma economia de tempo de desenvolvimento e validação.

Pois bem, pode-se observar que a reutilização de código não é tão ruim assim. Mas temos que ter um grande cuidado para reutilizar algum código. Existem alguns requisitos para que isso seja feito de forma correto, tais como:

- Deve ser possível encontrar componentes reutilizáveis adequados (catalogação e documentação externa devem existir).
- Deve-se ter certeza de que o código se comportará de acordo com o especificado e que será confiável (certificação).
- Deve ser possível compreender o componente para adaptá-lo à nova situação.

A partir destes requisitos, encontram-se dificuldades em reutilizar código, que são:

- Identificação, recuperação e modificação dos códigos reutilizáveis.
- Aumento nos custos de manutenção.
- Falta de ferramenta de apoio.

- Resistencia dos desenvolvedores.

Existem também técnicas para reuso de código, tais como:

Padrões de *software*: Descrevem soluções para problemas que ocorrem com frequencia no desenvolvimento de *software*.

- Vantagens de padrões de *software*: reuso de soluções encontradas por especialista experientes, tendo assim um aumento de produtividade e qualidade; melhoria na comunicação entre os projetistas; uniformidade na estrutura de *software*;
- Bibliotecas de classe: Classes de uso genérico podem ser disponibilizadas para reuso e importadas em múltiplas aplicações. Em geral são encorpadas ao código final da aplicação, ou seja, são compiladas juntamente com o restante do código.
- *Framework*s: Aplicação semi-completa reutilizável que, quando especializada, produz aplicações personalizadas, (Johnson & Foote, 1988). Coleção de classes abstratas e concretas e a interface entre elas, representando o projeto de um subsistema (Pree, 1995).

A partir deste ponto, observa-se então o que realmente interessa, *Framework*.

2.4 O QUE É *FRAMEWORK*?

Framework é um conjunto de classes que colaboram para realizar uma responsabilidade para um domínio de um subsistema da aplicação. Fayad e Schmidt.

Um *Framework* pode ser definido como uma aplicação completa que possui uma estrutura estática e outra dinâmica para resolver um conjunto restrito de problemas (GOVONI 1999). Seguindo esta máxima, pode-se concluir que vários "entretantos" que o desenvolvedor encontra em seu caminho, pode-se deixar de lado e focar-se na regra de negócio do sistema, no problema proposto e tendo assim uma alta produtividade.

2.4.1 Conceitos básicos sobre *Framework*

Hot-spot: Representam as partes de um *framework* de aplicação que são específicas de sistemas individuais. Eles são projetados para serem genéricos. Podem ser adaptados às necessidades da aplicação.

Frozen-Spot: Definem a arquitetura geral de um sistema de *software*, seus componentes básicos e os relacionamentos entre eles. Permanecem fixos em todas as instanciações de *framework* de aplicação.

2.4.2 Tipos

Frameworks Verticais são confeccionados através de uma experiência obtida em um determinado contexto específico. Esses são mais comumente chamado de *framework*s especialistas. Tentam resolver um determinado problema de um determinado domínio e são usados em vários *software* de mesmo domínio, como por exemplos cont*role* de estoque, recursos humanos, cont*role* financeiros entre outros. Após alguns projetos em um domínio específico são observados semelhanças entre estes projetos. A partir deste ponto, são construídos o dito *framework* vertical.

Frameworks Horizontais não dependem de um domínio específico e podem ser utilizados em diferentes domínios, como por exemplo *Hibernate* que é um *framework* de persistência.

2.4.3 Pontos Fortes

Framework é um conjunto de ferramentas desenvolvidas para auxiliar no desenvolvimento de projetos, tornando o trabalho mais produtivo. Os *framework*s contribuem em diversos aspectos no desenvolvimento do projeto, através de algumas práticas como o re-uso de código.

Pode-se destacar a enorme gama de recursos disponíveis, devido ao elevado número de *frameworks* disponíveis no mercado, e também, a infinidade de funções que cada *framework* pode ter.

Os *frameworks* são baseados em padrões de projetos já consolidados no meio de desenvolvimento, a maioria no padrão MVC, o que provê uma melhor manutenção posterior no projeto.

Os *frameworks* possibilitam o desenvolvimento mais rápido de projetos, através de ferramentas desenvolvidas para resolver problemas específicos, a *ide*ia tão difundida de "não reinvente a roda" se encaixa perfeitamente na prática do uso dos *frameworks*, visto que eles fornecem grande variedade de funções já prontas, das quais incorporam-se em nosso projeto, o que nos possibilita a redução expressiva do tempo de desenvolvimento, além de diminuir as dores de cabeça dos desenvolvedores quando tratam de desenvolver novas soluções.

A tendência do mercado de trabalho é de exigir cada vez mais o conhecimento em desenvolvimento de projetos através do apoio de *frameworks*, devido ao seu auxílio ser de extrema ajuda, e através deles, podemos adotar um novo indicador, que diferencia o desenvolvedor com habilidades de uso de *frameworks* do desenvolvedor comum.

2.4.4 Pontos Fracos

Como foi visto na sessão 5.0.3, os *frameworks* ajudam, e muito, no desenvolvimento de projetos, mas existe algumas desvantagem.

Os *frameworks* exigem conhecimento elevado de sua tecnologia, da linguagem em que foi desenvolvido, e também, de sua estrutura, que nos "amarram", ou seja, devem-se seguir os padrões impostos pelo *framework* para que se possa trabalhar com ele, isso pode ser decisivo na escolha de determinado *framework.*

Outro ponto fraco que devemos levar em consideração, é o fato de não saber ao certo como determinado processo trabalha dentro do *framework*}, já que a ferramenta pode deixar isso transparente ao desenvolvedor, além disso, pode-se destacar o uso de código que por vezes pode se tornar desnecessário, o que torna o código da aplicação sujo e ruim de ser interpretado.

Para se trabalhar com *frameworks* é preciso que o desenvolvedor já tenha certa experiência com desenvolvimento, padrões de projeto, e profundo conhecimento da

linguagem de programação que irá utilizar, se ele contém todos esses requisitos, a opção para se trabalhar com *frameworks* é uma ótima alternativa, já para os menos experientes, ter os *frameworks* como apoio em seu projeto pode tornar as coisas mais difíceis, tornar o desenvolvimento demorado, o que deixa o trabalho improdutivo, o que está totalmente fora do propósito dos *frameworks*

2.4.5 Características básicas

- Um *framework* deve ser reusável, afinal esse é o grande propósito de um *framework*. Antes de ser reusável, precisa ser usável, bem documentado e que seja fácil de usar.
- Deve ser seguro, o desenvolvedor não pode destruir o *framework* usado por ele.
- Deve ser extensível, precisa possuir funcionalidades abstratas que, se necessário, possam ser implementadas pelo desenvolvedor.
- Deve ser completo, para resolver o problema que se propôs resolver.

2.4.6 PHP Versus *Java*

Java consolidou-se como uma das linguagens de programação mais populares entre os desenvolvedores, atualmente ocupa o segundo lugar no ranking TIOBE Programming Community Index, sendo a principal linguagem de programação de quase 17% dos desenvolvedores que participaram da pesquisa. Possui duas *ide*'s que são referência em desenvolvimento, *Netbeans* e Eclipse, além de pregar a portabilidade através de sua máquina virtual.

Existe uma grande quantidade de livros, artigos, seminários e empregos voltados à tecnologia *Java*, o que garante uma vasta gama de fontes sobre a tecnologia, essas fontes em sua maioria são free, ou seja, são desenvolvidas e disponibilizadas pela comunidade *Java* ao redor do mundo. Devemos lembrar que a Oracle, a gigante empresa norte americana de tecnologia, comprou a Sun Micro Systems, empresa criadora da tecnologia *Java*, dessa forma, passou a ser a detentora dos direitos de *Java*. A Oracle demonstrou o interesse em manter e investir na tecnologia *Java*, através do lançamento da versão 7 do kit de desenvolvimento e das novas versões da sua *ide Netbeans*.

A sintaxe da linguagem *Java* é um dos pontos fortes da linguagem, devido ao seu nível de aprendizado ser relativamente fácil, o que implica numa melhor compreensão tanto da parte de quem escreve o código, tanto da parte de quem deve manutenir o código posteriormente.

Vejamos alguns números de *Java*, de acordo com a Oracle:

- Mais de 9 milhões de desenvolvedores Presente em mais de 800 milhões de PCs
- 2,1 bilhões de telefones celulares e outros dispositivos portáteis (fonte: Ovum).
- 3,5 bilhões de cartões inteligentes.
- Além de set-top boxes, impressoras, *web*cams, jogos, sistemas de navegação para automóveis, terminais lotéricos, dispositivos médicos, guichês de pagamento de estacionamento, etc.

Ao todo, são mais de 4,5 bilhões de dispositivos que utilizam a tecnologia *Java* ao redor do mundo.

Mas infelizmente, nem tudo são flores, *Java* tem tido seu prestígio arranhado graças à algumas falhas da tecnologia, recentemente, um vírus desenvolvido para Mac OS (Sistema Operacional da *App*le), tinha como sua porta de acesso ao sistema uma brecha criada pela máquina virtual *Java* (JVM), o que levou a Oracle a lançar uma correção da JVM.

Existia uma grande expectativa por parte da comunidade *Java*, devido ao lançamento da versão 7 da JDK (*Java* Development Kit), lançamento do qual previa inúmeras melhorias e novidades da ferramenta, o que não aconteceu. A Oracle trouxe poucas inovações, e corrigiu algumas falhas da antiga JDK, o que provocou certo alvoroço na comunidade, além disso, não se sabe ao certo quais são as intenções da Oracle, se ela irá manter a tecnologia free.

O desenvolvimento de projetos baseado na tecnologia *Java*, tende a ser mais penoso e demorado, se compararmos com outras tecnologias, *Java* exige conhecimento sólido na tecnologia orientada à objetos.

Um dos *frameworks* conhecidos em *java* é o *Java Server Faces*. O JSF (*Java Server Faces*) como é popularmente conhecido, é um *framework* MVC de desenvolvimento *Web*, baseado na linguagem *Java*. Criado com o intuito de agilizar o processo de desenvolvimento de sistemas *Web*, o JSF é um dos *frameworks* mais aceitos pela comunidade *Java*.

Ele possuí recursos bastante interessantes e úteis, como suporte à internacionalização, *Managed Beans*, injeção de dependências, etc. Pode ser usado com outros *frameworks*, como *Junit*, *Facelets* e JSP.

O JSF gera as *Views*, contendo a estilização e rótulos de acordo com o objeto DAO, além de possuir diversas suítes de componentes como *PrimeFaces*, *RichFaces* e *IceFaces*. Essas suítes são bibliotecas que contém componentes visuais extremamente ricos visualmente, com usabilidade aprimorada, além de que a maioria dos componentes disponíveis possuem recursos compatíveis com requisições assíncronas.

Recomendado para desenvolvedores que tenham ampla experiência em *Java* e MVC, devido à sua curva de aprendizado ser alta. O JSF possuí uma comunidade de desenvolvedores bastante vasta, o que contribui para o número de documentações de como trabalhar com o *framework.*

PHP

Linguagem criada especificamente para atender o desenvolvimento *web*, ocupa a sexta colocação no ranking TIOBE Programming Community Index, com um pouco mais de 5,7% do gosto dos desenvolvedores que participaram da pesquisa.

O PHP é uma linguagem que perdeu bastante força ao longos dos últimos anos, devido ao crescimento de outras tecnologias como *Java* e Asp.net, que trouxeram melhorias significativas para o desenvolvimento de projetos, principalmente devido às suas doutrinas serem voltadas diretamente ao conceito orientado à objetos. Isso levou o time responsável pelo PHP a implementar mudanças, das quais, fizeram da linguagem mais amigável aos desenvolvedores.

O PHP possui uma sintaxe bastante complexa, misturando código HTML com o seu, o que torna o código mais difícil de ser interpretado, para os iniciantes, o PHP pode se tornar uma linguagem mais difícil de ser aprendida, graças a sua sintaxe não ser muito amigável.

Existem diversas extensões para o PHP, extensões essas das quais melhoram o desenvolvimento com a linguagem, essas extensões servem para complementar a linguagem em si, através do fornecimento de códigos prontos, o que diminui o trabalho do desenvolvedor.

Vale destacar a constante mudança na qual o PHP vem sofrendo ao passar do tempo, essas mudanças visam manter a linguagem de programação competitiva no mercado, visto que suas concorrentes ganharam uma vantagem enorme nos últimos anos. Mostrar-se-á dois *frameworks* PHP:

Codeigniter

Definição de acordo com o site oficial:

"CodeIgniter is a proven, agile open PHP *web app*lication *framework* with a small footprint. It is powering the next generation of *web app*s."
Algo como:
"CodeIgniter é um *framework* PHP *open-source* de aplicações *Web* de agilidade comprovada com uma pequeno *footprint*. Ele está fortalecendo a próxima geração de aplicações *Web*."

Um dos *frameworks* para PHP mais utilizados no mundo, o *Codeigniter* ganhou adeptos por possuir um rápido carregamento das páginas, ótima documentação e por ser relativamente fácil de aprender. É totalmente *open-source*, o que possibilita uma maior contribuição da comunidade em geral para a ferramenta. O *Codeigniter* quando comparado com outro *framework* PHP, como por exemplo o Yii, não possui tantos recursos, e também necessita do *download* de muitos *add-ons* para se ter a sua total funcionalidade.

2.5 *GRAILS*

Breve Histórico.

- 2003: O projeto *Groovy* foi oficialmente fundado.
- 2004: Fundação do projeto *Groovy*One
- 2005: Criado o G2one, inicialmente chamava-se *Groovy* on Rails, porém teve que ser alterado por pedido direto do fundador do projeto *Ruby on Rails*.
- 2006: já estava na posição de número 38 entre 200 linguagens de programação.
- Nos dias atuais encontra-se na versão 2.2.1.

2.5.1 O Que é?

Grails é um *framework* para desenvolvimento de aplicações *Web*, baseado no padrão MVC, utiliza a liguagem *Groovy* e roda sobre JVM (*java* virtual *machine* - máquina virtual *java*) e tem como ponto ápice a alta produtividade. Utiliza-se dos melhores e mais usados *frameworks* da plataforma *java*, como o *Hibernate* e *Spring*. Utiliza-se também do conceito de programação por convenção, paradigma este que busca diminuir o número de decisões

tomadas pelo desenvolvedor, visa ganhar simplicidade, assim sendo, ganhando em produtividade.

2.5.2 Vantagens e Desvantagens

As vantagens do *Grails* nada mais é que suas próprias características citadas anteriormente: *Full Stack*, Expansibilidade, início rápido para desenvolvimento e, considerada a maior de todas as vantagens, convenção por configuração.

Outra grande vantagem é que *Grails* tem um ambiente de desenvolvimento completo. Após sua instalação, o *framework* já vem com todos, ou quase todos, os componentes necessários para que o desenvolvedor possa iniciar seu trabalho duro e árduo (não com *Grails*). Existem várias *ide*'s para desenvolvimento com suporte para *Grails*, entre eles o *Netbeans*, Eclipse e Intellij *ide*A, mas com apenas um editor de texto é possível utilizar todos os recursos que o *Grails* nos oferece.

Mas, como tudo na vida, existem as desvantagens de usar *Grails*. São elas:

- A resistência de desenvolvedores a aderir a esta nova tecnologia, não tão nova assim, diga-se de passagem. Desenvolvedores *java* tendem a continuar não aderindo ao *Grails* em seus projetos até perceberem de vez que, com *Grails*, tudo ficará mais fácil e aumentarão significamente a produtividade de seus projetos.
- A outra desvantagem deslancha em paralelo com a primeira citada. Uma vez que não existem muitas pessoas usando *Grails*, também existem poucas fontes de pesquisas, como por exemplo comunidades de *Java*.

2.5.3 Principais Características

As principais características deste *framework* são:

- DRY (*dont repeat yourself*) - este conceito visa definir nomes e códigos em apenas um lugar e reaproveitar essas informações em todos os outros lugares necessários, sem precisar modificar nada. Um bom exemplo é o CRUD e os controladores. O *Grails* faz o trabalho árduo para nós e não precisamos nos preocupar, diminuindo significamente

o desenvolvimento destas tarefas e, assim sendo, aumentando a produtividade de forma incrível.

- *Full Stack* - ao contrário do que ocorre com o tradicional *java*, no qual para criar uma aplicação *Web* usamos vários componentes como *Hibernate*, *Commons*, *Spring* e muitos outros e, ainda, precisamos configurar todos eles, na qual passam-se horas e horas para poder configurar todos esses componentes e, em certos casos, passam-se dias nos preocupando em fazer tudo de acordo para que fique tudo muito bem integrado, o *Grails* traz vários componentes pré instalados, tudo configurado e pronto para ser usado. Ficar horas e horas configurando componente se tornou uma coisa impraticável com a presença do *Grails*.
- Convenções sobre configurações - paradigma este proposto por outro *framework*, o *Ruby on Rails*. Este paradigma propõe reduzir a necessidade de arquivos de configurações e exageros de codificações. Este paradigma, como já dito anteriormente, busca diminuir o número de decisões tomadas pelo desenvolvedor visando ganhar simplicidade, ganhando muito mais em produtividade. Em termos simplificado, nos faz o seguinte: "Siga essas convenções e deixe o resto comigo".
- Expansibilidade - mesmo sendo um *framework* poderosíssimo, *Grails* não é completo, sempre precisa de algo. Observa-se o seguinte caso: um programador precisa desenvolver uma aplicação *Web* seguindo vários requisitos pré-levantados com o seu cliente, tais como CRUD e autenticação do usuário. *Grails*, por ser uma ferramenta poderosa, faz-se o CRUD de forma muito simples, porém ainda temos que pensar na parte de autenticação do usuário. Como desenvolver? *Grails*, mesmo sendo este magnífico *framework*, não atende à estas necessidade. Daí surge os *plugins* para o *Grails*, você adiciona *plugins* de acordo com sua necessidade. Neste nosso exemplo, basta instalar o *Plugin Acegi*, que ele já vem pronto para autenticação de usuário. Além deste, existe, hoje, 792 *plugins* para *Grails*, segundo o site oficial *Grails*.org.
- Início rápido do desenvolvimento - ao iniciar um projeto com *Grails*, já vem pronto para ser desenvolvido uma aplicação *Web* com padrão MVC, *hibernate* e outros tantos *plug-ins* e *frameworks* que, normalmente, demoramos a configurar todos eles.

O *Grails* tem muitos recursos junto à ele, tais como:

- *Scaffolding*: o *Grails* tem uma estrutura para gerar aplicações CRUD (*Create, Read, Update e Delete*) com muito pouco código, deixando para o desenvolvedor

concentrar-se na regra de negócio da aplicação. A funcionalidade *Scaffolding* gera automaticamente as tabelas no banco de dados. Ele, por padrão, gera, para cada classe de domínio criada em sua aplicação, uma tabela no banco de dados com todos os atributos definidos em sua classe.

- Classes de Domínios: onde encontra-se toda a regra de negócio do sistema e que serão persistidas no banco de dados.
- Mapeamento Objeto-Relacional: o *Grails* há uma estrutura de mapeamento objeto-relacional chamado de *GORM* (*Grails Object Relational Mapping*). *Gorm* pode mapear objetos para banco de dados relacionais.
- *Plugins*: o *Grails* por si só não garante todas as soluções para todo o tipo de aplicação porém, tendo em vista isto, ele provê uma gama de *plugin* que pode ser adicionado em sua aplicação como por exemplo Ajax, segurança e auditoria, pesquisas entre outros. Este recurso acaba tornando mais fácil adicionar funcionalidades que, de modo geral, poderia ser complicado de desenvolver.

2.5.4 *Plugins*

Os *plugins* utilizados para o sistema proposto foram:

- *Mail* (1.0.1) - Este *plugin* visa validar os *e-mail*s inseridos pelo usuário, utilizando o padrão texto@texto.com, ou seja, se o usuário informar um endereço de *e-mail* que não esteja nesse padrão, o *plugin* automaticamente irá alertar o usuário sobre esse problema.
- *Cloud-Foundry* (1.2.3) - Este *plugin* auxilia o desenvolvedor no momento em que ele deseja colocar a sua aplicação *online*, basta criar uma conta no *site* http://www.cloudfoundry.com/ e configurar a aplicação para que ela seja colocada na nuvem. Edita-se o arquivo "BuildConfig.*groovy*" colocando as duas linhas destacadas em vermelho na figura abaixo, contendo o nome do usuário e sua senha. O nome do usuário deve ser o mesmo informado no cadastro do *cloud-foundry* e a senha é aquela da qual o *site* lhe envia, vale ressaltar que esta senha é temporária, e é de suma importância que o usuário troque-a assim que possível. A figura 7 apresenta o arquivo "BuildConfig.*groovy*" editado, conforme explicado.

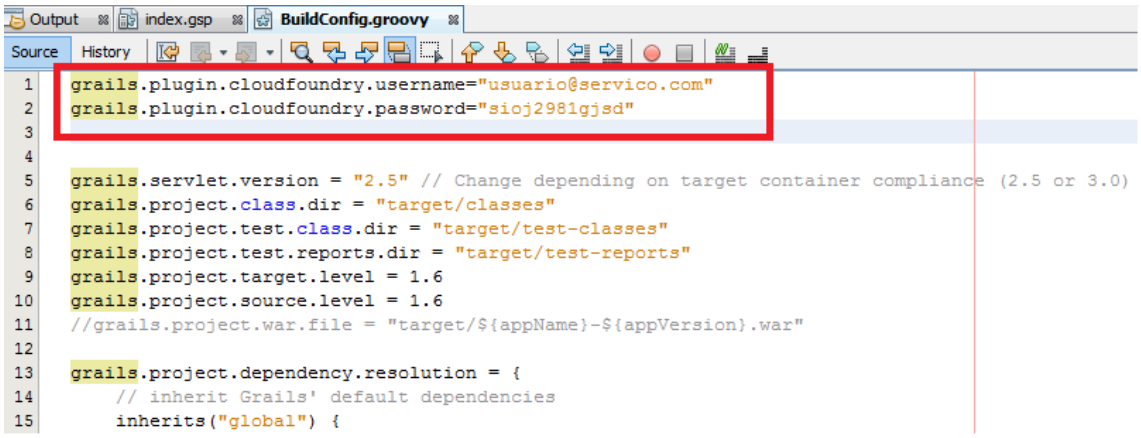

Figura 7: *Plugin* Cloud-Foundry

Fonte: Autoria Própria

- CPF - Validador de Cadastro de Pessoas Físicas (CPF) que utiliza o padrão 999.999.999-99. Esse *plugin* calcula o CPF informado e informa se este é válido. Um ponto fraco deste *plugin* é que ele exige obrigatoriamente o uso da pontuação.

- *Acegi* - O *plugin Acegi* é um mecanismo de segurança, que facilita no desenvolvimento do cont*role* de acesso da aplicação.

Existem três classes que o *Acegi* trabalha:

- *Role*: Tipos de usuários do sistema, exemplo: Administrador, usuário.
- *Person*: O usuário do sistema.
- *Request Map*: Trata-se do relacionamento entre os tipos de usuários e as páginas que estes usuários poderão acessar.
- Para criar essas três classes, pode-se utilizar o comando

"*Grails create-auth-domains User Role RequestMap*". Pode-se escolher outros nomes para as classes.

A figura 8 mostra como é criado a classe *User*:

```
/**
 * User domain class.
 */
class User {
   static transients = ['pass']
   static hasMany = [authorities: Role]
   static belongsTo = Role
  /** Username */
  String username
  /** User Real Name*/
  String userRealName
  /** MD5 Password */
  String passwd
  /** enabled */
  boolean enabled

  String email
  boolean emailShow

  /** description */
  String description = ''

  /** plain password to create a MD5 password */
  String pass = '[secret]'

  static constraints = {
    username(blank: false, unique: true)
    userRealName(blank: false)
    passwd(blank: false)
    enabled()
  }
}
```

Figura 8: Classe *User*

Fonte: Autoria Própria

A figura 9 apresenta como é criada a classe *role*:

```
/**
 * Authority domain class.
 */
class Role {
  static hasMany = [people: User]

  /** description */
  String description
  /** ROLE String */
  String authority

  static constraints = {
    authority(blank: false, unique: true)
    description()
  }
}
```

Figura 9: Classe *Role*

Fonte: Autoria Própria

A figura 10 apresenta como é criado a classe *RequestMap*:

```
/**
 * Request Map domain class.
 */
class Requestmap {
  String url
  String configAttribute

  static constraints = {
    url(blank: false, unique: true)
    configAttribute(blank: false)
  }
}
```

Figura 10: Classe Request Map

Fonte: Autoria Própria

Depois disso, pode-se gerar as telas e os controladores para as classes através do comando "*Grails generate-manager*".

Depois de gerar as telas e os controladores, devemos configurar nosso mecanismo de segurança. Começa-se, então, pelos tipos de usuários do sistema. Na figura 11, cadastramos um novo tipo de usuário, e o descrevemos. É importante salientar que devemos criar um nome para o tipo de usuário no padrão "*ROLE_* " sucedido do nome que queremos, por exemplo, "*ROLE_* ADMIN".

Figura 11: Tela *Role*

Fonte: Autoria Própria

Na figura 12, pode-se observar uma lista com os tipos de usuários que estão cadastrados no sistema.

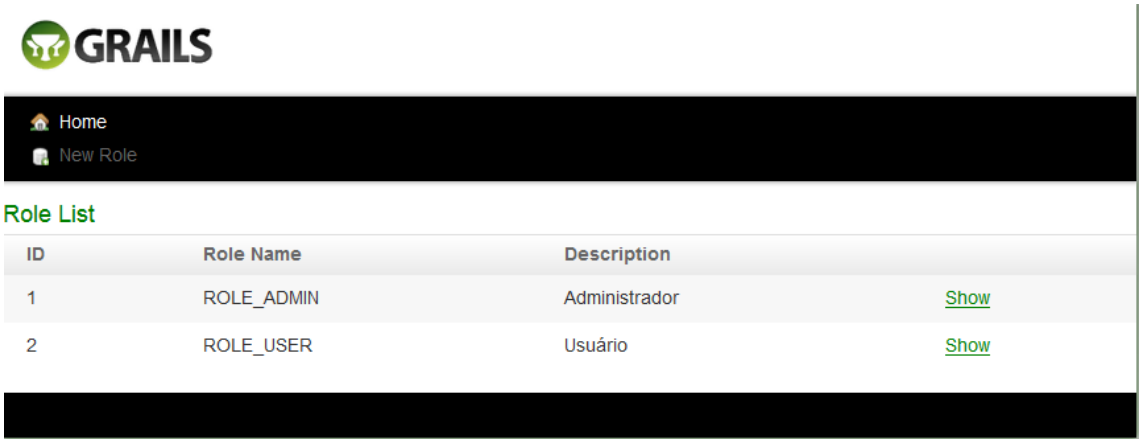

Figura 12: Lista de Tipos

Fonte: Autoria Própria

Depois disso, podem-se criar os usuários do sistema.

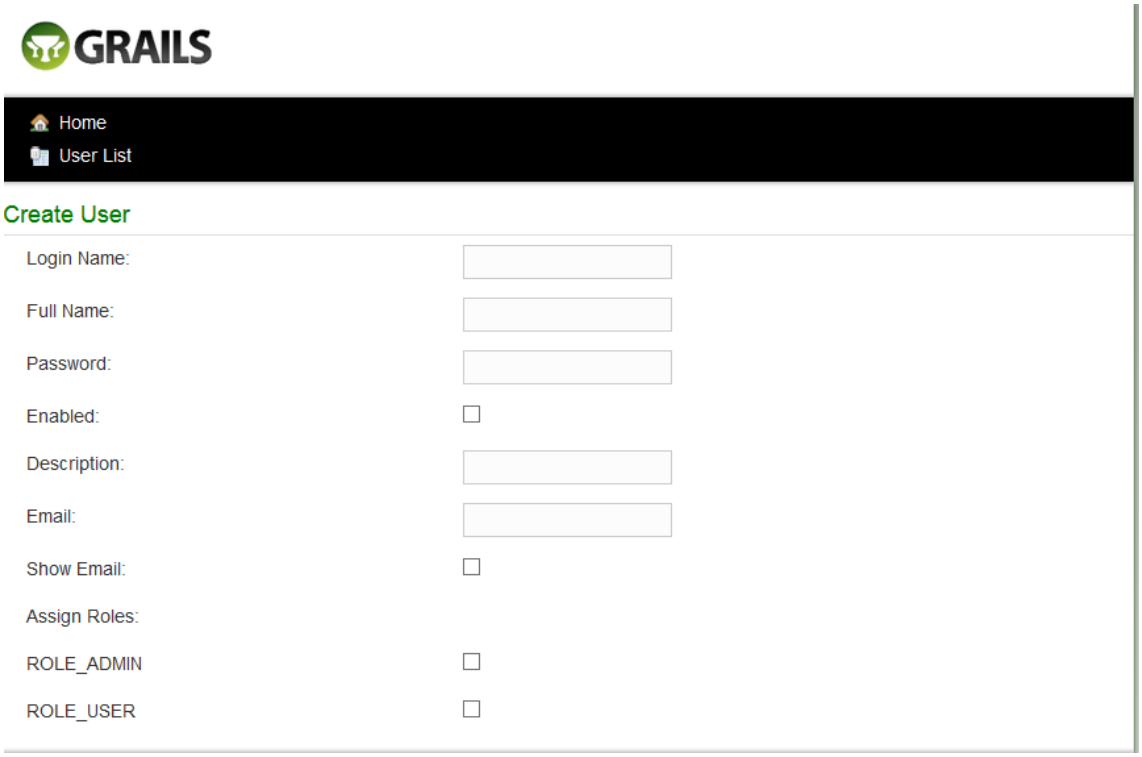

Figura 13: Cria Usuário

Fonte: Autoria Própria

E então, relacionam-se os tipos de usuários com as páginas que eles podem acessar, na figura 14 segue uma lista contendo a respectiva página e qual tipo de usuário pode acessá-la.

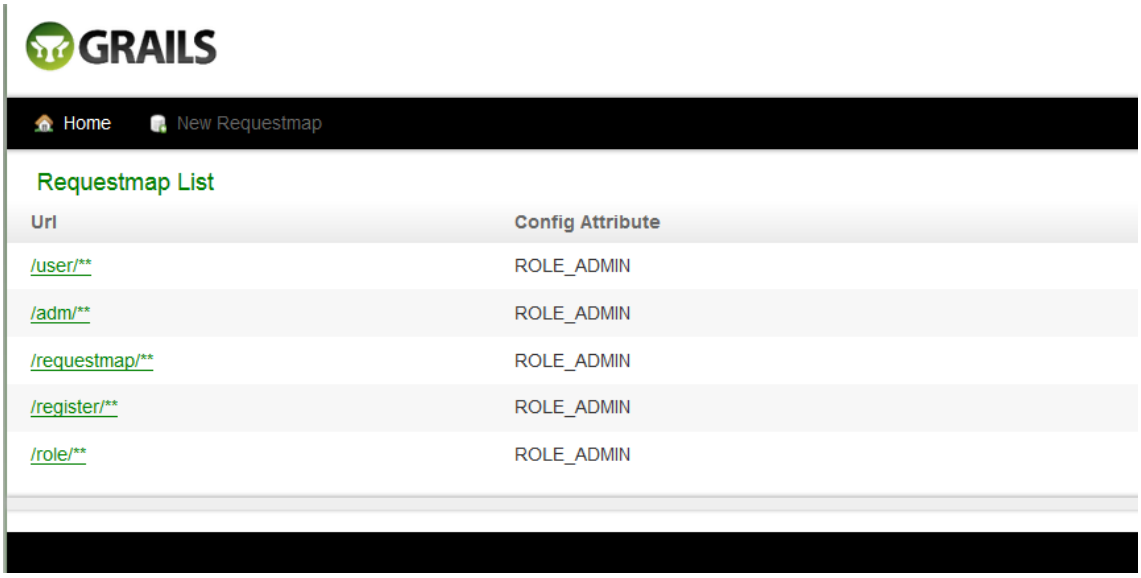

Figura 14: Definir Página

Fonte: Autoria Própria

2.5.5 Estrutura do *Grails*

Grails é composto de uma estrutura bastante simples baseada no padrão MVC. Na figura 15 pode-se observar a estrutura de diretórios do *Grails*, que será explicada item por item de sua estrutura.

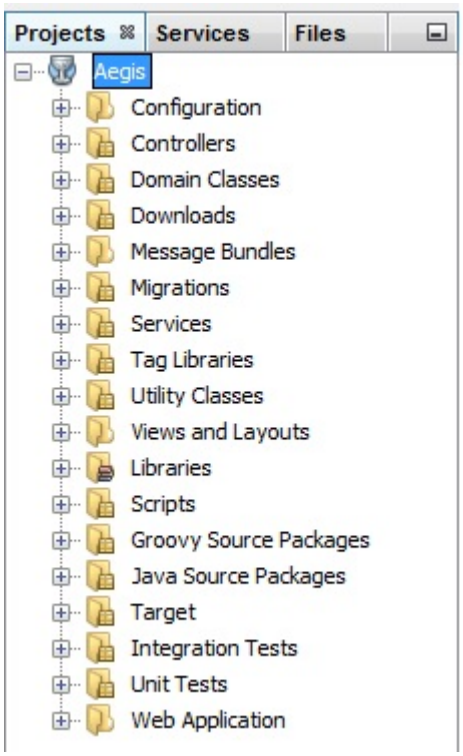

Figura 15: Estrutura de Diretórios do *Grails*
Fonte: Autoria Própria

- *Controllers* - este diretório fica responsável por armazenar todos os cotroladores da aplicação.
- *Domain* - este diretório fica responsável por armazenar todas as classes de domínios do sistema, classes estas que fica a regra de negócio do sistema e que serão persistidas no banco de dados.
- *Conf* - arquivos de configuração da aplicação como o *dataSource* (banco de dados) e configuração do *hibernate* e *Spring*.
- i18n - recursos para internacionalização de uma aplicação.
- *Services* - responsável por armazenar todos os serviços de sua aplicação. Um serviço é responsável por armazenar aspectos referentes à regra de negócio da aplicação do sistema, para que o desenvolvedor não inclua regra de negócio nas classes de controladores. As classes de serviços são gerenciados pelo *Spring*, e por sua vez é responsável de encapsular determinados aspectos da lógica de negócio da aplicação a ser desenvolvida.
- *Taglib* - este será responsável por armazenar todos os arquivos responsáveis pelas *tag* uilizadas pelo *Grails*.
- *Utils* - este diretório ficará responsável por armazenar classes utilitárias para a aplicação a ser desenvolvida.
- *Views* - neste diretório ficará todos os arquivos responsáveis por mostrar as páginas utilizadas em nossa aplicação, no caso do *Grails* os arquivos são .gsp. Cada classe de domínio contém um diretório equivalente dentro do diretório *View*. Como por exemplo, em nossa aplicação teremos a classe de domínio chamada campus, então dentro do diretório *View* terá outro diretório chamado campus e dentro deste terá os arquivos .gsp que se refere à sua classe.
- *Lib* - este diretório ficará todas as bibliotecas usadas em sua aplicação. Em nosso caso usamos a biblioteca ne conexão com o banco de dados MySql.
- *Scripts* - diretório responsável por armazenar todos os *scripts* necessários e desenvolvido pelo desenvolvedor da aplicação. Os *scripts* usados pelo *Grails* são desenvolvidos em GANT (*groovy* Ant).

- Src - este diretório é responsável por armazenar os códigos fontes a ser reaproveitados por sua aplicação. Em geral são códigos de aplicações de sistemas legados que não se encaixa na estrutura padrão do *Grails*. Dentro deste há mais dois diretórios, *Groovy* e *Java*, onde coloca-se código referente a cada um deles.
- Test - diretório que armazena todos os testes unitários e testes de integração.
- *Web-app* - todo o conteúdo estático da aplicação se encontra neste diretório, como por exemplo imagens, arquivos css entre outros.

2.5.6 Arquitetura do *Grails*

Agora que já foi apresentado algumas das características do *Grails* e sua estrutura, vamos entender a arquitetura deste *framework* poderosíssimo. A figura 16 mostra sua arquitetura.

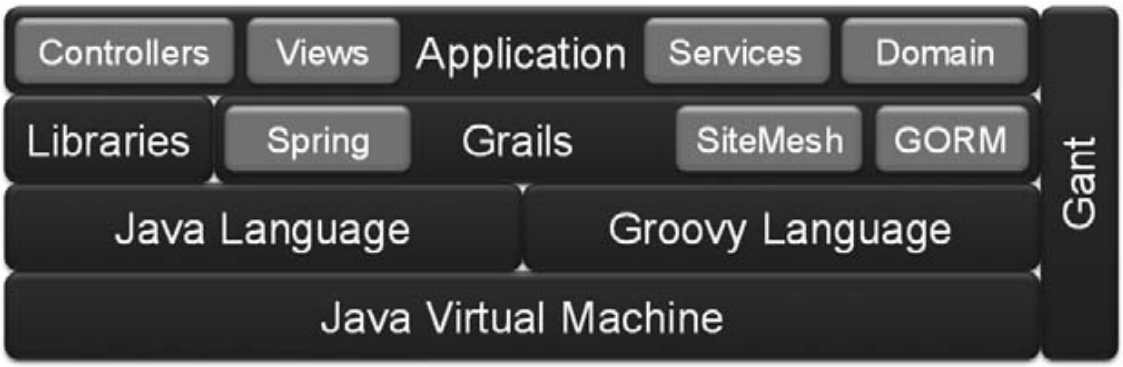

Figura 16: Arquitetura do *Grails*

Fonte: Livro Beginning *Groovy* and *Grails*, pg. 68.

Pode-se observar que o *Grails* utiliza da JVM (*Java Virtual Machine*). Pode-se observar também que utiliza-se do padrão MVC. Observa-se também que utiliza-se tanto da linguagem *Groovy* quanto linguagem *Java*. Vários *framework*s pré instalados e configurados e bibliotecas pré carregadas pelo *Grails*. A linha de comando do *Grails* é construída em cima de Gant, um sistema que utiliza a linguagem de *Script Groovy* para Apache Ant, em vez do formato XML do Ant.

2.5.7 Por quê utilizar *Grails*?

Grails foi planejado para ter alta produtividade, para isso, utiliza tecnologias que já estão consolidadas, como *Hibernate* e *Spring*, o *Grails* também utiliza do paradigma "programação por convenção", dessa forma ele simplifica a configuração da aplicação.

A persistência dos dados é outro fator importante na escolha deste *framework* para se trabalhar, visto que ele preserva o desenvolvedor de criar a camada de persistência, ou seja, basta o desenvolvedor escrever uma classe de domínio e o *Grails* irá gerar a tabela no banco de dados, bem como os mecanismos de acesso à esses dados.

A geração dos controladores e das telas também é outro fator importante, visto que ganha-se um tempo enorme com isso, o *Grails* além de gerar o banco de dados e mecanismos de persistência, ele também gera os controladores e as telas. Naturalmente dessa forma, o *Grails* implementa o padrão MVC, vastamente utilizado pela comunidade de desenvolvedores.

Por utilizar a linguagem *Groovy*, a curva de aprendizado do *framework* é quase zero, visto que para os desenvolvedores *Java*, aprender *Groovy* é uma tarefa simples, isto porque a linguagem é muito parecida com *Java*.

O *Grails* conta com casos de sucesso como os das empresas: Atlassian, Sky, Linked In, eHarmony, ESPN, Virtuwell e Netflix.

Jon Mullen, Scrum Master do site sky.com disse:

"*Groovy is so much quicker and simpler to write code with, so we can get applications up and running faster. With Groovy and Grails we can create a new feature in a week, when before it could easily take a month or more*".

Algo como:

"*Groovy* é muito mais rápido e mais simples de escrever código, então nós podemos obter e terminar as aplicações mais rápido. Com o *Groovy* e *Grails*, podemos criar uma nova funcionalidade em uma semana, antes isto poderia facilmente levar um mês ou mais".

Existe uma grande quantidade de documentações como tutoriais, vídeos, artigos, demonstrando ao desenvolvedor como trabalhar com a ferramenta, bem como a comunidade do *Grails* está em constante expansão, fato que ajuda o *framework* à ganhar cada vez mais colaboradores e adeptos.

Vale a pena conhecer o *framework Grails*, principalmente para os desenvolvedores que já trabalham com *Java* e com alguns *framework*s como *Java Server Faces* ou *Java Server Pages*.

Por estas e outras razões acredita-se que o *Grails* seja uma boa opção na escolha de um *framework* para desenvolvimento *Web*, devido à sua agilidade de desenvolvimento, robustez e simplicidade.

2.6 *JASPER REPORTS*

Para o desenvolvimento de relatórios para a aplicação foi usado o *framework Jasperreport*. O *Jasperreport* é um *framework* para geração de relatórios dinâmicos em diversas extensões, entre elas o PDF e planilhas do excel. É uma ferramenta *openSource*. O desenvolvimento de relatório usando apenas este *framework* é uma tarefa bastante árdua e que precisa bastante domínio para configurações de arquivos XML para que, depois, o *framework* possa gerar o relatório. Tudo bem, temos um *framework* para gerar relatórios porém iremos encontrar algumas dificuldades para poder gerá-los, certo? A resposta é sim, se nós assim desejarmos, porém não almejamos tal feito. Então qual a solução? *Ireport*, esta é a solução.

2.6.1 O Que é *Ireport*?

Uma das grandes dificuldades de trabalhar com relatórios está na composição de seu *layout*. É bastante complicado desenvolver o *layout* do relatório totalmente em XML. Por este motivo, a maioria dos desenvolvedores trabalham com ferramentas que auxiliam o desenvolvimento do *design* de relatórios. Uma das ferramentas bastante usada, até por ser do mesmo desenvolvedor do framework *Jasperreport*, é o *ireport*. É muito comum os dois sendo usados concomitantemente.

O *ireport* é uma ferramenta para o desenvolvimento do *design* de relatórios, conexão com o banco de dados e configuração dos arquivos XML. De forma simples pode-se gerar relatório sem muito domínio de XML, apenas arrastando componentes gráficos na tela, para que se possa desenhar o *template* de seu relatório.

O ambiente para desenvolvimento do *ireport* é muito simples de se usar, mas que não deixa a desejar. É uma ferramenta poderosíssima com vários recursos. A figura 17 apresenta o ambiente de desenvolvimento do *ireport*.

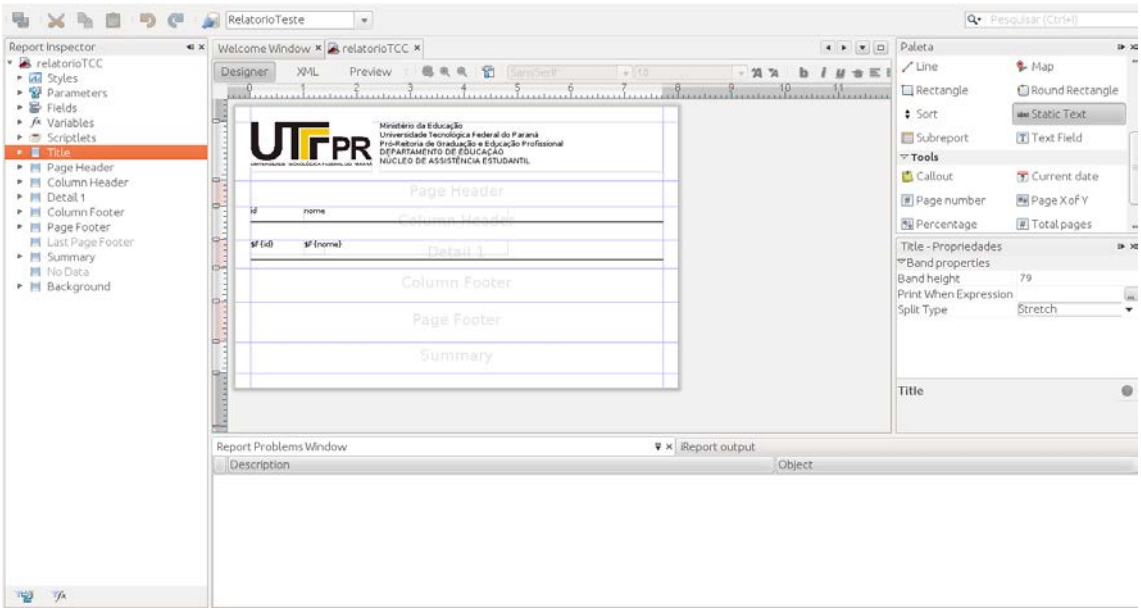

Figura 17: Ambiente *ireport*

Fonte: Autoria Prórpria

Observa-se que na lateral esquerda da imagem encontra-se as opções para se colocar no relatório a ser criado, tais como os parâmetros a ser mostrados, título do relatório entre outros. No meio fica a parte do *designer* do relatório. Na lateral direita da figura encontra-se as opções para o *designer* do relatório.

2.7 *FRAMEWORK* DE PERSISTÊNCIA

Seguem abaixo alguns *framework*s de persistência estudados neste trabalho.

2.7.1 *Hibernate*

Trata-se de um *framework* escrito e disponibilizado nas linguagens *Java* e .Net, que mapeia objetos para banco de dados relacionais, ou seja, o *Hibernate* tem a missão de

transformar objetos *Java* ou .Net em dados relacionais para posterior transação com um banco de dados relacional.

O *Hibernate* mapeia os objetos através de arquivos XML (*Extensible Markup Language*) ou *not*ações *Java*, gera comandos SQL (*Structured Query Language*) o que diminui consideravelmente o trabalho do desenvolvedor. O *framework* também conta com sua própria linguagem, chamada de HQL (*Hibernate Query Language*) baseada em SQL, porém orientada à objetos.

Hibernate torna as transações com o banco de dados um pouco mais lentas, mas em compensação, diminui o trabalho do desenvolvedor. Pode-se observar na figura 18 como é composta a arquitetura do *Hibernate*.

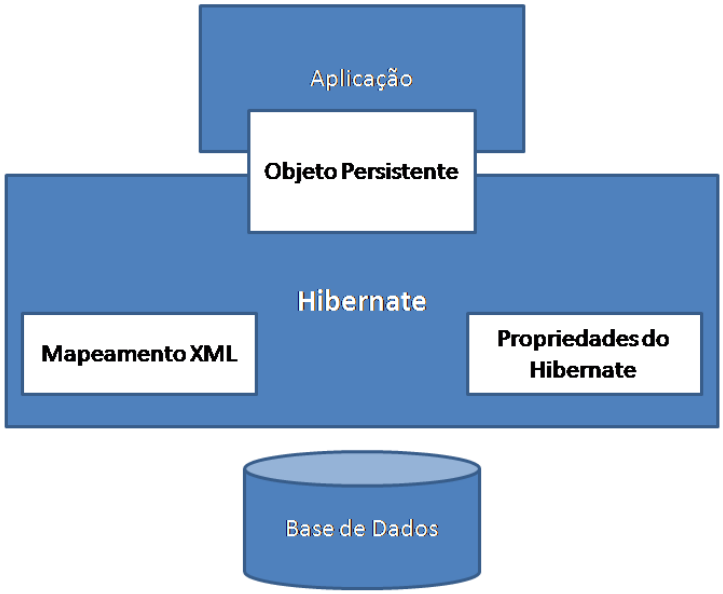

Figura 18: Arquitetura do *Hibernate*

Fonte: Adaptada - http://*java*programmercomunidade.blogspot.com.br/2011/07.html

O *Hibernate* é um pouco complicado de início, devido as suas configurações, mas se demonstra uma ótima ferramenta no decorrer do desenvolvimento.

O *framework* foi desenvolvido em 2001 por Gavin King, que buscava uma maneira de otimizar as transações com o banco de dados. Mais tarde, a empresa Jboss, empregou o time de desenvolvedores do *Hibernate*, e os ajudou a melhorar a ferramenta.

2.7.2 ORM - *Object Relational Mapping*

Os desenvolvedores de *software* muitas vezes encontram dificuldades em trabalhar eficientemente com bancos de dados relacionais, devido à que os *softwares* de hoje não são baseados nos modelos relacionais que os bancos de dados populares utilizam, mas sim, baseados na tecnologia orientada à objetos.

Existem no mercado diversos *framework*s que nos auxiliam nesse trabalho árduo de criar um *link* entre a tecnologia orientada à objetos e a teoria relacional de banco de dados, os chamados ORMs. Pode-se citar os mais populares: *Java Persistence* API (JPA), *Hibernate* e *TopLink*.

O mapeamento entre objetos e entidades é algo de extrema dificuldade, fato que rendeu a essa prática o termo *object-relational impedance mismatch*, algo como, "Conflito de impedância no mapeamento objeto-relacional".

2.7.3 *GORM - Grails Object Relational Mapping*

Construído baseado no *framework Hibernate*, entretanto possui diferenças consideráveis, principalmente quando avaliamos a sua facilidade de uso, o *Gorm* é muito mais simples de ser usado do que o *Hibernate*, além de não mapear as classes através de XML.

O *Gorm* gera automaticamente todo o banco de dados da aplicação, incluindo relacionamentos entre tabelas, tipos de campos, etc. Além de gerar as operações básicas de: Inserção, edição, exclusão e pesquisa dos dados. Tudo isso é gerado à partir das classes de domínio, ou seja, quando uma nova classe desse tipo é criada, o *Gorm* mapeia todas os atributos da classe, fazendo com que os mesmos tornem-se os campos da tabela, respeitando os seus tipos.

O *Gorm* adiciona métodos de persistência nas classes de domínio, não possuí objetos equivalentes aos *Hibernate Session* e dispensa a necessidade de injeções de dependência.

Na figura 19 pode-se observar como o mapeamento de classes funciona:

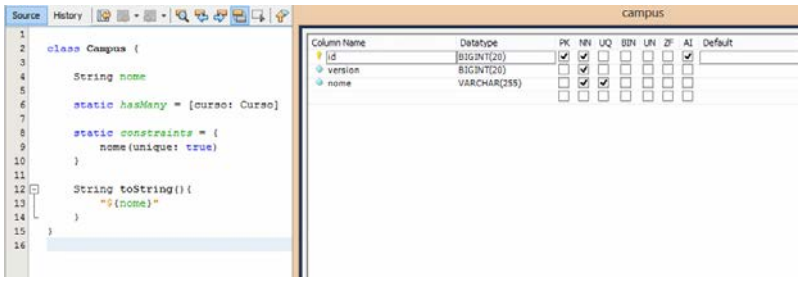

Figura 19: Mapeamento da Classe

Fonte: Autoria Própria

O *Gorm* baseia-se nas mesmas convenções adotadas pelo *Hibernate* como: O nome da tabela provém do nome da classe, com todas as letras minúsculas. Cada atributo da classe é uma coluna da tabela, com todas as letras minúsculas. Existe os atributos implícitos "id" e "*version*", que representam a chave primária da tabela e o campo para versionamento dos dados, respectivamente.

Na figura 20, pode-se verificar a existência de uma chave estrangeira chamada "campus-id".

Figura 20: Mapeamento da Classe

Fonte: Autoria Própria

A chave estrangeira é criada quando declara-se na classe "Curso" o objeto campus, que é a instanciado a partir de uma classe de domínio chamada "Campus". As chaves estrangeiras por padrão são criadas com as propriedades "*not null*" e "*auto increment*".

O *Gorm* suporta os relacionamentos: um para muitos, muitos para um, uma para um. Uma grande desvantagem é a que o *Gorm* não suporta múltiplos *Data Sources*.

3 O SISTEMA PROPOSTO

Nesta seção é descrito como o sistema atual funciona, bem como o sistema Aegis foi desenvolvido.

3.1 OBJETIVO DO SISTEMA

O sistema para classificação de Estudantes à serem contemplados pela UTFPR tem a finalidade de agilizar e facilitar o processo feito pelo NUAPE. O sistema visa facilitar, não apenas os membros organizadores, mas também os alunos que utiliza deste sistema atual de classificação. O sistema deverá fazer o cadastro de alunos que pretende obter o auxílio da bolsa-permanência, deverá fazer a classificação destes alunos, se está apto a concorrer à uma vaga, e após essa seleção, gerar um relatório dos alunos aptos por ordem de classificação.

3.2 O SISTEMA ATUAL

O sistema atual de classificação de alunos para obtenção do auxílio da bolsa-permanência é bastante precário. A primeira etapa do processo de seleção consta no cadastro dos alunos interessados pelo auxílio. O NUAPE disponibiliza as fichas de inscrições, no primeiro semestre de 2012 eram 4 páginas de ficha para a inscrição. Os alunos interessados pegavam os formulários de inscrições, preenchiam e então entregavam para a comissão responsável, neste caso o NUAPE. A segunda etapa é a conferência das fichas de inscrições e dos documentos entregues para o NUAPE para que possa ser feita a primeira seleção dos alunos, se ele está apto ou não para obtenção da bolsa-permanência ou seja, se todos os pré-requisitos estabelecidos para a obtenção do auxílio foram obedecidos. Por último é feito a seleção dos alunos por ordem de classificação.

Após obedecidos todas as etapas é colocado em edital os alunos classificados para obtenção do auxílio.

3.2.1 Problemas do Sistema Atual

Visto todas as etapas seguidas pelo programa de bolsa-permanência da UTFPR, percebemos que há um grande défice no processo de seleção dos alunos, tais como:

- Primeiro, pode-se observar que, por causa da própria etapa, é preciso imprimir várias fichas de inscrições para que os alunos possam pegar para se inscrever no programa. Observa-se que muitas folhas são "jogadas fora" para poder obedecer esta etapa.
- Segundo, percebe-se que o NUAPE disponibilizava de seu pessoal para poder fazer toda a conferência necessária, perdendo muito tempo ao se fazer isso e, na terceira etapa, ocorre o mesmo, várias pessoas fazendo a mesma coisa, perdendo muito tempo, afinal é um processo bastante minucioso, e acabava sendo demorado.
- Outro problema encontrado são reclamações dos próprios participantes, neste caso os alunos, do processo de seleção. Além de ser muitas folhas a serem preenchidas, acaba levando muitos dias desde a entrega da ficha de inscrição até a comissão organizadora obter os resultados finais, afinal por se tratar de um auxílio que o campus da UTFPR oferece aos alunos, precisa ser um processo bastante minucioso, logo torna-se demorado.

3.3 ENTREVISTAS

Para se ter um sistema completo, foram feitas algumas entrevistas com a comissão responsável pelo programa de bolsa-permanência.

Entre as técnicas mais comuns e, também, uma das mais importantes para levantamento de requisitos é a entrevista. Esta técnica está presente em outras técnicas como questionários, *brainst*orm entre outros. Mesmo não usando a técnica direta de entrevista, normalmente usa-se indiretamente ela, como por exemplo, o questionário, para se interrogar o usuário com o questionário proposto, normalmente terá que estar frente à frente com seu usuário ou cliente para que possa obter os resultados esperados.

As primeiras coisas a serem feitas em uma entrevista é determinar, entre outros aspectos que também são importantes, seus propósitos e objetivos, as informações necessárias.

Segundo Xexéo (2007), existem vários tipos de entrevistas. Durante o processo de análise, elas normalmente seguem o seguinte processo:, são elas:

- Introdução inicial
- Familiarização com o ambiente
- Busca de fatos
- Verificação de informações conseguidas de outras formas
- Confirmação de informações conseguidas com os candidatos
- Acompanhamento, amplificação ou clarificação de entrevistas anteriores.
- Outra grande variável da entrevista é o seu objetivo, entre outros:
- Levantar informações sobre a organização
- Levantar informações sobre uma função de negócio
- Levantar informações sobre um processo ou atividade
- Descobrir problemas
- Verificar fatos previamente levantados
- Fornecer informação
- Obter dicas para entrevistas futuras

Conhecendo e entendendo esses conceitos, a primeira entrevista foram colocados à mesa como o processo atual é feito, o que precisaria ser melhorado e o que o cliente esperava do sistema proposto.

3.3.1 Levantamento de Requisitos

Segundo a norma IEEE Std 1220-1994, um requisito é uma sentença *ide*ntificando uma capacidade, uma característica física ou um fator de qualidade que limita um produto ou um processo.

De acordo com Xexéo (2007), qualquer que seja o sistema, existe várias formas de entendê-lo. Similarmente, existem vários tipos de requisitos, que são aplicáveis ou não, dependendo da visão necessária naquele instante.

Os requisitos podem ser classificados em três partes distintas, são elas: requisitos de usuário, requisitos de sistema ou requisitos de *software*.

Um requisito do usuário é algum comportamento ou característica que o usuário deseja do *software* ou o sistema como um todo; o que o usuário quer. São escritos pelo próprio usuário ou levantados por um analista de sistemas que consulta o usuário.

Um requisito do sistema é algum comportamento ou característica exigido do sistema como um todo, incluindo hardware e *software*. O comportamento desejado do sistema. São normalmente levantados por engenheiros ou analistas de sistemas, refinando os requisitos dos usuários e os transformando em termos de engenharia.

Um requisito do *software* é algum comportamento ou característica que é exigido do *software*. São normalmente levantados por analistas de sistemas. O objetivo da análise é capturar todos os requisitos para o *software* sendo desenvolvido e apenas aqueles requisitos verdadeiros.

Tendo esta máxima, a segunda entrevista foi feita com o objetivo de levantar todos os requisitos necessários para que, assim, pudesse começar a desenvolver o sistema.

Os requisitos levantados nesta entrevista foram:

- Todos os dados que estão na ficha de inscrição atual precisa estar no sistema proposto.
- Todos os campos desta ficha são obrigatórios.
- O sistema terá que ter uma parte para cadastro de aluno e outra para visualização dos alunos cadastrados.
- O sistema precisará gerar a relação dos alunos inscritos em ordem de classificação, segundo critérios pré-determinados pelo NUAPE.
- O sistema deverá guardar o histórico de todas as inscrições feitas anteriormente para que, se necessário, seja feita consultas aos mesmos.
- O sistema deverá gerar relatórios separados por cursos de alunos cadastrados.
- O sistema deverá gerar relatório do número total de alunos que pediram o auxílio.
- O sistema deverá imprimir a relação dos alunos classificados para obtenção do auxílio, bem como os alunos que estarão na lista de espera.

3.4 MODELAGEM

Nesta seção, observa-se a modelagem do banco de dados do sistema proposto. Na figura 21, encontram-se todas as entidades presente no sistema proposto, seus relacionamentos, atributos, quais são as chaves primárias e as chaves estrangeiras.

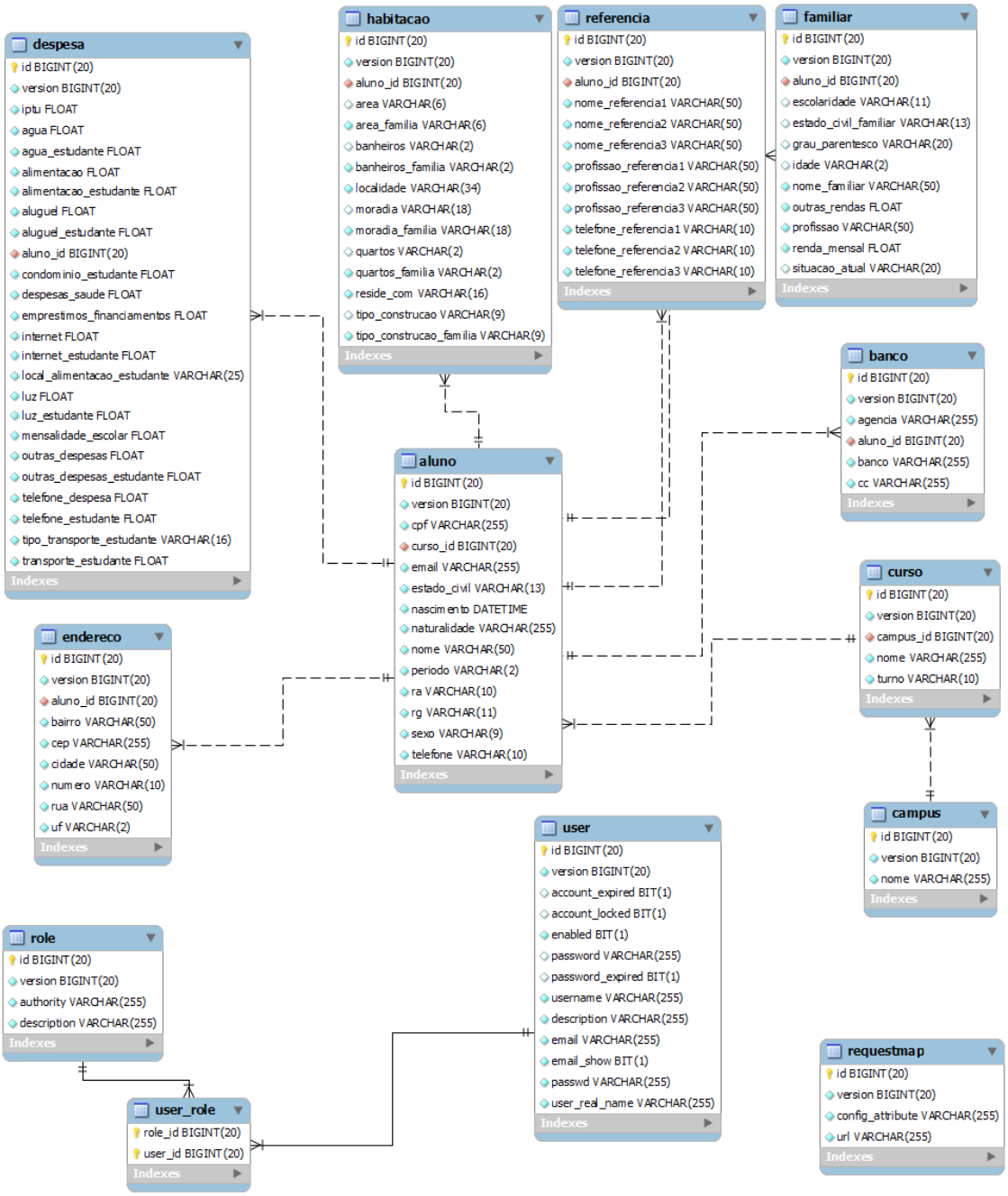

Figura 21: Modelo Relacional

Fonte: Autoria Própria

- Entidade aluno - Nesta entidade encontram-se os dados pessoais do aluno. A tabela 1 apresenta os relacionamentos da entidade aluno:

Tabela 1: Tabela x Multiplicidade da Entidade Aluno

Tabela	Multiplicidade
Despesa	1:1
Habitação	1:1
Referência	1:1
Familiar	1:N
Banco	1:1
Endereço	1:1
Curso	1:1

Fonte: Autoria Própria

- Entidade Despesa - Nesta entidade estão relacionados todos os dados relativos às despesas do aluno e de sua família.
- Entidade Habitação - Dados referentes à habitação do aluno (se independente ou morar em outra residência que não seja a de seus pais) e da sua família.
- Entidade Referência - Dados de três pessoas que possam dar informações sobre o aluno.
- Entidade Familiar - Nesta entidade encontram-se os dados dos familiares que residem na mesma casa que o aluno.
- Entidade Banco - Dados da conta bancária do aluno.
- Entidade Endereço - Dados sobre o local onde o aluno mora.

As Entidades despesa, habitação, referência, familiar, banco e endereço estão relacionadas apenas com a entidade aluno.

- Entidade Curso - O curso que o aluno faz.

Na tabela 2, observa-se os relacionamentos que a entidade Curso tem:

Tabela 2: Tabela x Multiplicidade da Entidade Curso

Tabela	Multiplicidade
Aluno	1:N

Campus	1:1

Fonte: Autoria Própria

- Tabela Campus - Descrevem quais são os campus cadastrados no sistema.

Relacionamentos:

Tabela 3: Tabela x Multiplicidade da Entidade Campus

Tabela	Multiplicidade
Curso	1:N

Fonte: Autoria Própria

- Entidade *Request Map* - Contém o relacionamento entre as páginas que necessitam de permissão de acesso e qual é o tipo de usuário que pode acessá-la.
- Entidade *Role*} - Contém os tipos de usuários do sistemas.

Na tabela 4, pode-se observar os relacionamentos da entidade *Role* tem:

Tabela 4: Tabela x Multiplicidade da Entidade *Role*

Tabela	Multiplicidade
User_role	N:N

Fonte: Autoria Própria

3.5 PROTOTIPAÇÃO

Nesta seção será apresentado as telas do sistema Aegis.

3.5.1 Tela inicial do sistema

A tela inicial do sistema é onde pode-se encontrar as primeiras informações do sistema, para o que ele se destina, as informações sobre o NUAPE e os documentos pertinentes ao programa Bolsa-Permanência para *download*.

É nesta primeira tela que o aluno poderá entrar para começar a se cadastrar para o programa Bolsa-Permanência ou para os responsáveis pelo Bolsa-Permanência fazer o *login* no sistema para a administração que necessitar fazer.

Na figura 22 pode-se visualizar a tela inicial do sistema.

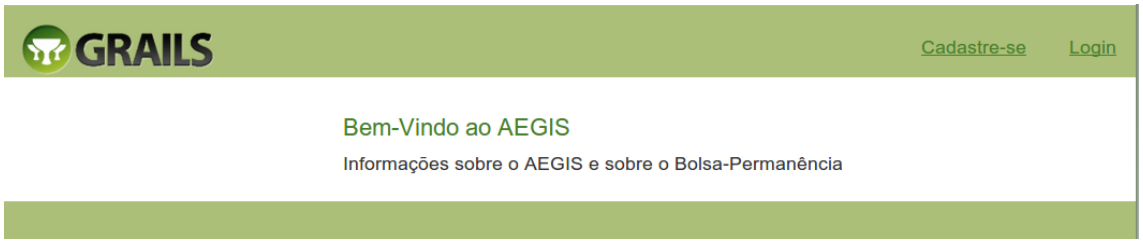

Figura 22: Tela Inicial do Sistema

Fonte: Autoria Própria

3.5.2 Tela de cadastro - Dados Pessoais do Aluno

Esta tela representa os campos necessários, sobre os dados pessoais do aluno, a ser preenchido pelo aluno para que ele possa cadastrar-se para o programa Bolsa-Permanência.

A figura 23 apresenta a tela de cadastro dos dados pessoais do aluno.

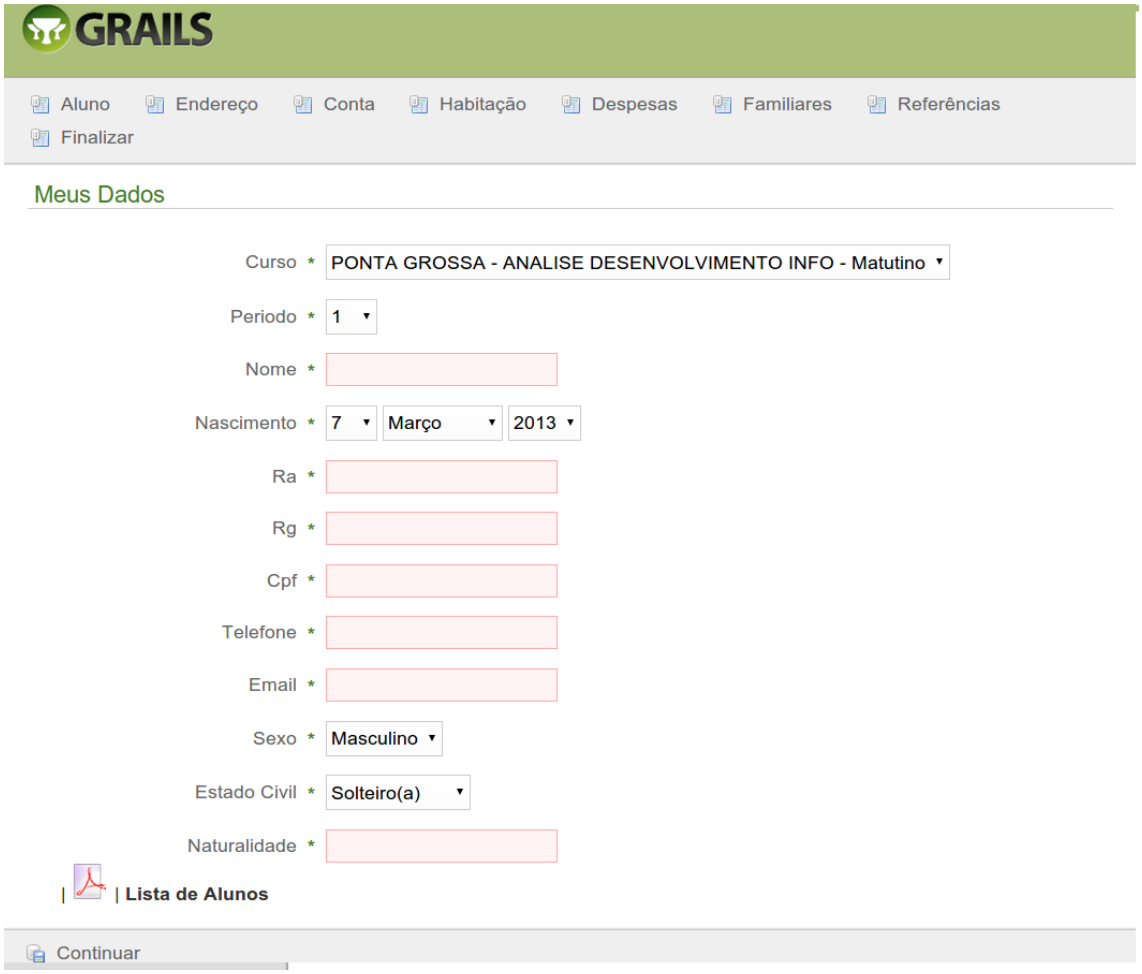

Figura 23: Tela de Cadastro – Dados Pessoais do Aluno

Fonte: Autoria Própria

Pode-se observar nesta tela os campos necessários a ser preenchidos.

No campo "Curso", o aluno deverá escolher o curso que ele faz, o campus em que ele estuda e o turno regular de seu curso.

No campo "Período", o aluno deverá colocar o período regular que está cursando.

No campo "Nome", o aluno deverá preencher seu nome completo.

No campo "Nascimento", o aluno deverá colocar a data de seu nascimento.

No campo "RA", o aluno deverá colocar o número de seu RA (Registro Acadêmico).

No campo "Cpf", o aluno deverá colocar seu número de CPF, no formato "***.***.***-**", onde os "*" representam o número de seu CPF.

No campo "Telefone", o aluno deverá colocar o número de seu telefone, precedido do DDD de seu telefone. Neste campo deverá colocar somente números, sem traços, pontos ou qualquer outros caracter diferente de números.

No campo "*Email*", o aluno deverá colocar seu *e-mail* para contato com o NUAPE. Este campo deverá estar no formato "exemplo@exemplo.com".

No campo "Sexo", o aluno deverá escolher entre "Masculino" e "Feminino".

No campo "Estado Civil", o aluno deverá escolher entre "Solteiro", "Casado", "Divorciado" e "Viúvo".

No campo Naturalidade, o aluno deverá colocar qual a sua Naturalidade.

Após todos os campos devidamente preenchidos, o aluno deverá clicar em "Continuar", que fica localizado no canto inferior esquerdo da página, para que ele possa prosseguir com o cadastro.

Uma coisa que devemos nos atentar, é que nesta tela, todos os campos são de preenchimento obrigatório.

3.5.3 Tela de cadastro - Endereço

Nesta tela o aluno deverá colocar o endereço de sua moradia atual.

A figura 24 apresenta a tela de cadastro de endereço.

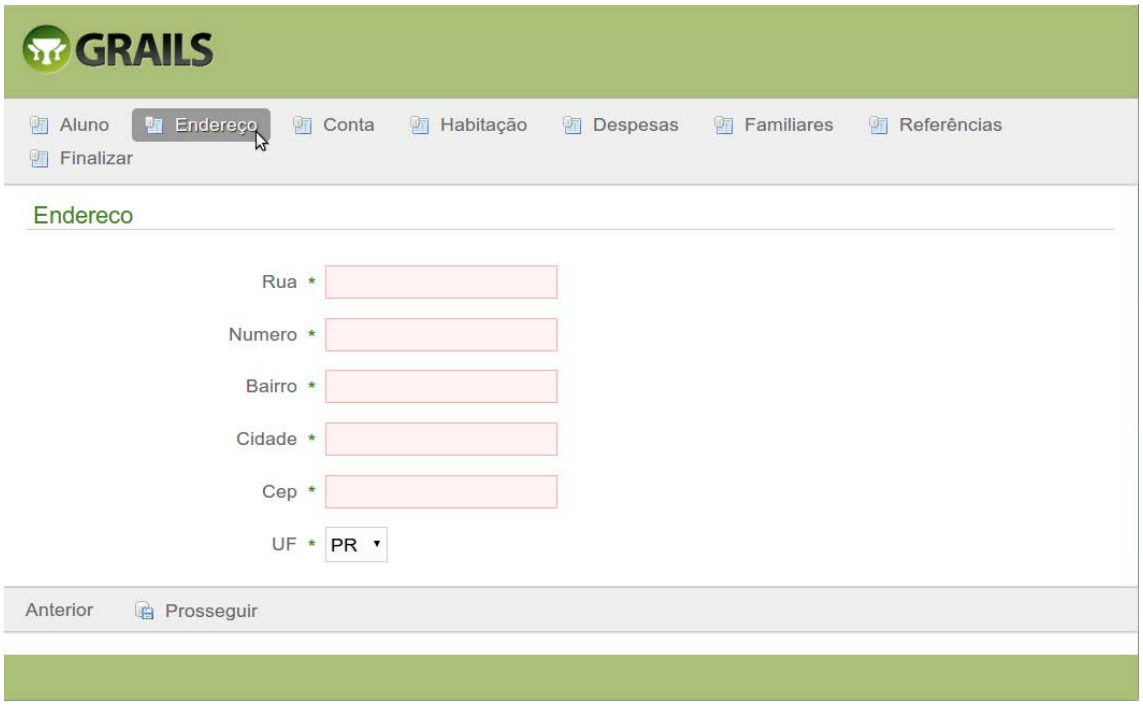

Figura 24: Tela de Cadastro - Endereço

Fonte: Autoria Própria

Os campos necessários desta tela são:

Campo "Rua", o aluno deverá colocar o nome da rua onde mora.

Campo "Número", o aluno deverá colocar o número de onde mora.

Campo "Bairro", o aluno deverá colocar o bairro onde mora.

Campo "Cidade", o aluno deverá colocar a cidade onde mora.

Campo "Cep", o aluno deverá colocar o número do Cep onde mora.

Campo "UF", o aluno deverá escolher o estado onde mora.

Após todos os campos estiverem devidamente preenchidos, o aluno deverá clicar em "Prossegui", para que ele possa continuar com o seu cadastro no programa. Caso o aluno clique em "Anterior", o sistema irá voltar para a tela de cadastro dos dados pessoais do aluno.

Nota-se que nesta tela, todos os campos são de preenchimento obrigatório.

3.5.4 Tela de Cadastro - Conta Bancaria

Nesta tela o aluno deverá preencher os dados bancários de sua conta, para que ele possa receber o dinheiro do programa Bolsa-Permanência, caso ele tenha sido classificado para tal feito.

Na figura 25 é mostrado os campos desta tela.

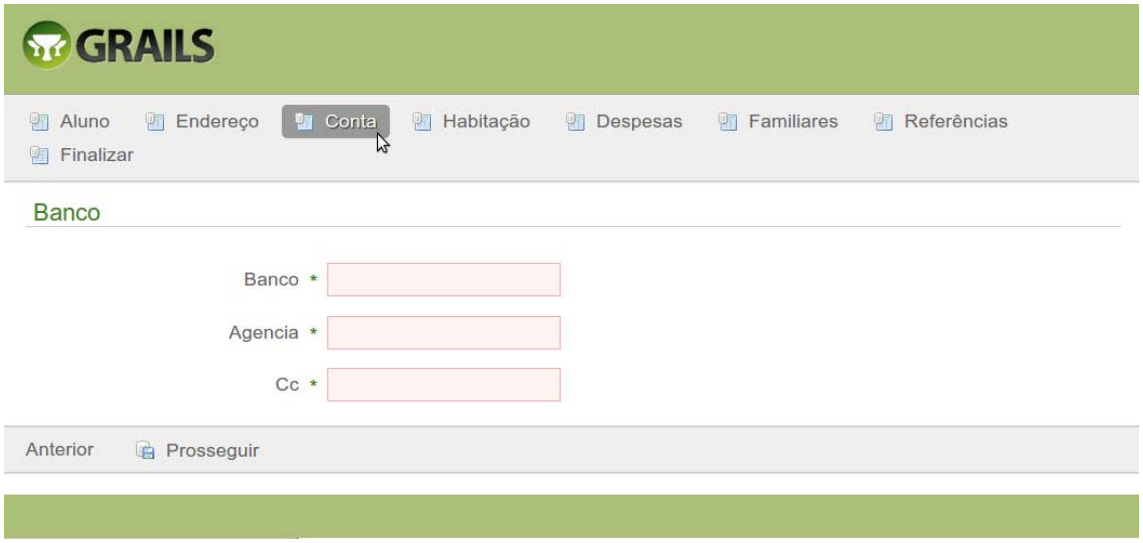

Figura 25: Tela de Cadastro – Conta Bancária

Fonte: Autoria Própria

No campo "Banco", o aluno deverá colocar o nome do banco em que ele queira receber o auxílio, caso seja classificado para isso.

No campo "Agência", o aluno deverá colocar o número da agência bancária.

No campo "CC", o aluno deverá colocar o número de sua conta corrente.

Após todos os campos estiverem devidamente preenchidos, o aluno deverá clicar em "Prosseguir". Caso clique em "Anterior", voltará à tela anterior, que neste caso seria a tela de cadastro de endereço do aluno.

3.5.5 Habitação

Nesta tela o aluno deverá preencher os campos relacionados à sua moradia de sua família e, caso residir sozinho, relacionados à sua própria moradia.

Na figura 26, pode-se observar campo-a-campo desta tela.

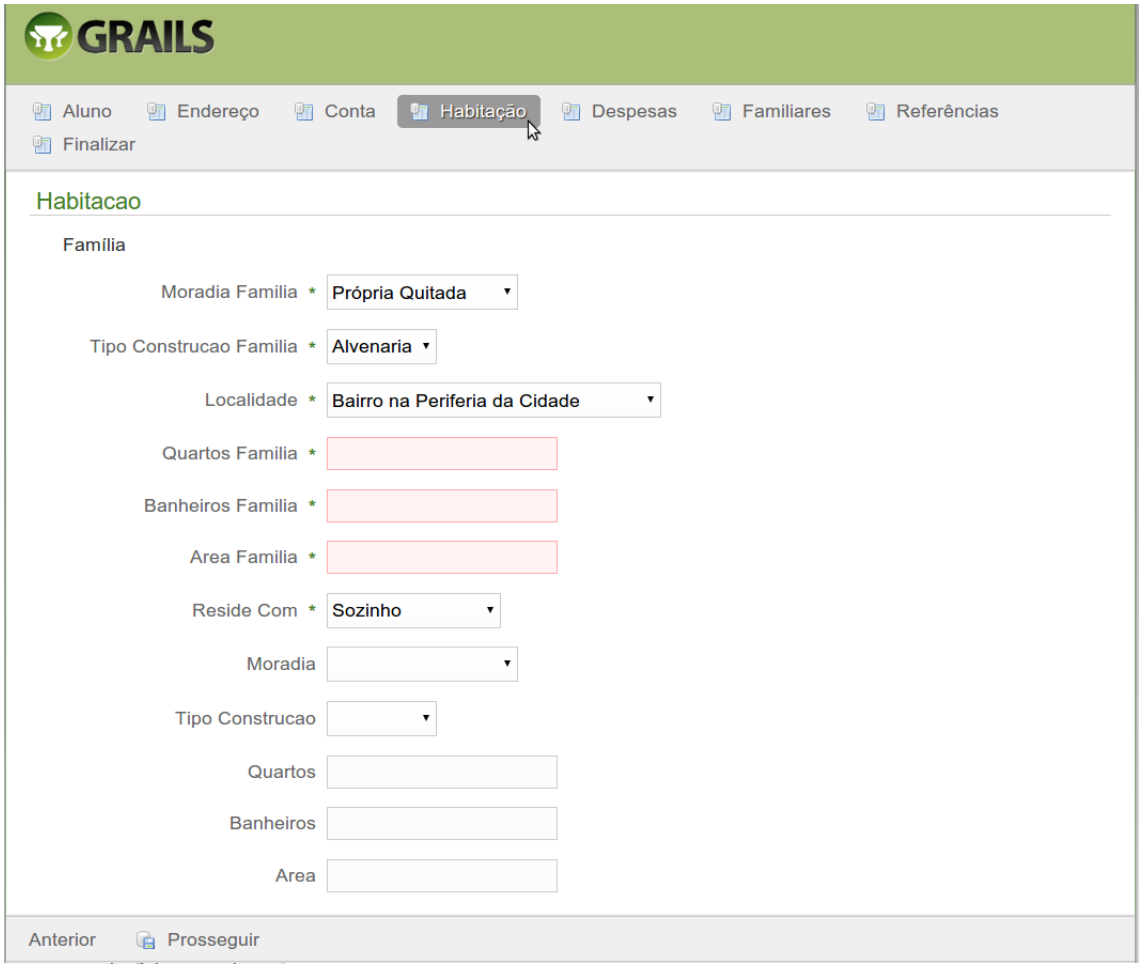

Figura 26: Tela de Cadastro – Habitação

Fonte: Autoria Própria

No campo "Moradia Família", o aluno deverá escolher o tipo em que se encontra a moradia de sua família, por exemplo, se a casa é própria e está quitada, se a casa é financiada, se a cada está alugada entre outros tipos que se encontra para a escolha.

No campo "Tipo Construção Família", o aluno deverá escolher o tipo da construção da casa de sua família, como por exemplo, se a casa é de alvenaria, se a casa é de madeira ou se a casa é mista.

No campo "Localidade", o aluno deverá escolher onde se encontra a casa de sua família, como por exemplo, em um bairro da periferia da cidade, área rural, condomínio fechado entre outras localidades existentes para a escolha.

No campo "Quartos Família", o aluno deverá colocar quantos quartos existe na casa de sua família.

No campo "Banheiros Família", o aluno deverá colocar a quantidade de banheiros existentes na casa de sua família.

No campo "Área Família", o aluno deverá colocar o tamanho da casa de sua família. Este tamanho deverá ser a área total da casa, contando o tamanho do terreno também.

No campo "Reside Com", o aluno deverá escolher com quem ele reside atualmente, como por exemplo, sozinho, se ele mora com a família, entre outros escolhas possíveis que existe no sistema.

Caso ele morar com a família, o aluno deverá desconsiderar os restante dos campos, caso o aluno residir sozinho ou outra escolha que não seja com a família, ele deverá preencher o restante dos campos. Os campos restantes, o aluno deverá preencher levando em conta onde ele reside atualmente.

3.5.6 Despesas

Nesta tela o aluno deverá preencher os campos relacionados às despesas de sua família e suas próprias despesas.

A figura 27 apresenta a tela de cadastro de despesas.

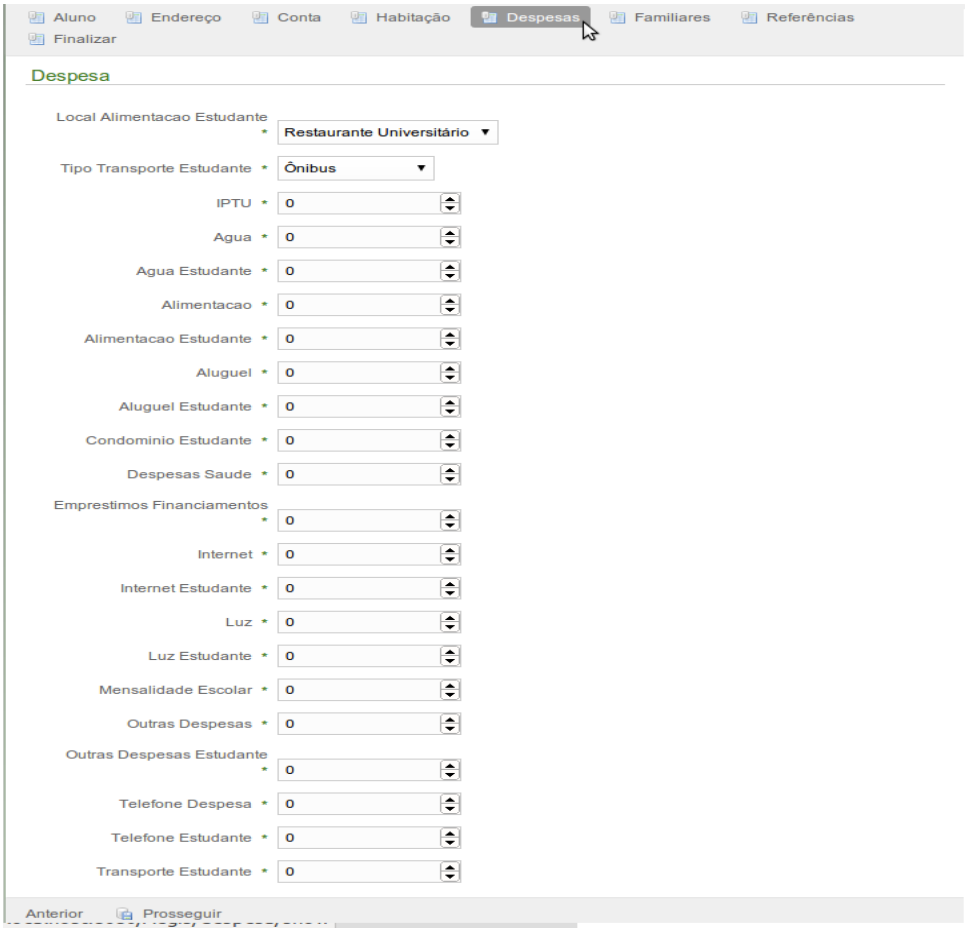

Figura 27: Tela de Cadastro - Despesas

Fonte: Autoria Própria

Nesta tela todos os campos são de preenchimento obrigatório. Mas deve-se observar que os campos onde precisa colocar os valores, iniciam com o valor "0". O aluno deverá mudar o valor de acordo com o valor real relacionado àquele campo.

3.5.7 Familiares

Nesta tela o aluno deverá preencher os campos relacionados a todos os familiares, que reside com ele ou não.

A figura 28 apresenta a tela de cadastro de familiares.

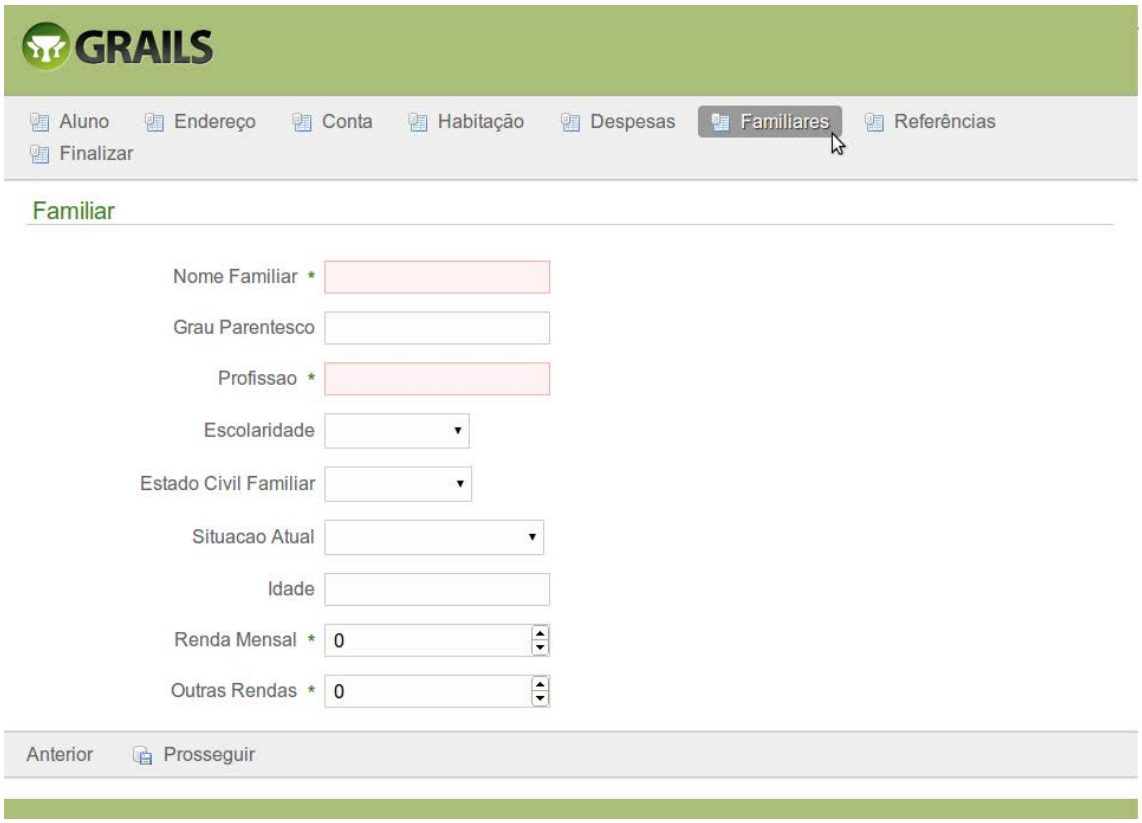

Figura 28: Tela de Cadastro – Familiares

Fonte: Autoria Própria

Nesta tela deverá preencher obrigatoriamente os campos "Nome Familiar", "Profissão", "Renda Mensal" e "Outras Rendas". Os outros campos não são de preenchimento obrigatório.

No campo "Nome Familiar", o aluno deverá colocar o nome do elemento em questão.

No campo "Profissão", o aluno deverá colocar a profissão da pessoa que ele está cadastrando no momento.

No campo "Renda Mensal", o aluno deverá colocar a renda que este familiar tem mensalmente.

No campo "Outras Rendas", o aluno deverá colocar outros tipos de renda que este familiar pode estar recebendo.

Lembrando que os campos que demonstram valores, como é o caso de "Renda Mensal", iniciam com o valor zero, tendo assim, o aluno mudar com o valor real do campo em questão.

Nota-se nesta tela que tem o preenchimento de apenas 1 (um) familiar. Assim que o aluno terminar o cadastro deste familiar, aparecerá uma opção para que ele possa cadastrar outros familiares, se assim ele desejar.

3.5.8 Referências

Nesta tela o aluno deverá cadastrar três pessoas como referências, podendo ser um amigo, vizinho ou qualquer outra pessoa que possa servir como referência.

A figura 29 apresenta a tela de cadastro de referências do aluno.

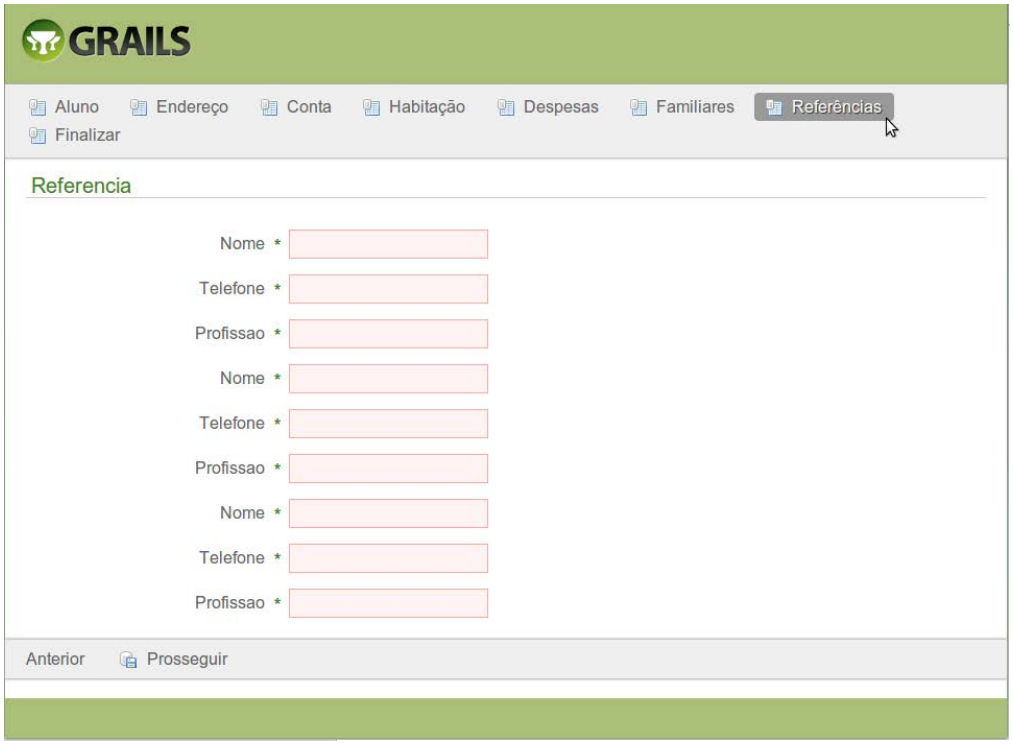

Figura 29: Tela de Cadastro – Referências

Fonte: Autoria Própria

Nesta tela todos os campos são de preenchimento obrigatório.

No campo "Nome", o aluno deverá preencher o nome da pessoa que servirá de referência.

No campo "Telefone", o aluno deverá colocar o número de telefone desta pessoa que servirá de referência.

No campo "Profissão", o aluno deverá colocar qual é a profissão da pessoa que está colocando como referência.

Por fim, após todos os campos estiverem devidamente preenchidos, o aluno deverá clicar em "Prosseguir", e por diante estará cadastro para a seleção do programa Bolsa-Permanência.

3.5.9 Tela de *Login*

Esta tela é responsável para que usuários pré-cadastrados possam acessar o sistema com privilégios de administrador.

A figura 30 pode-se observar como é composta a tela de *login*.

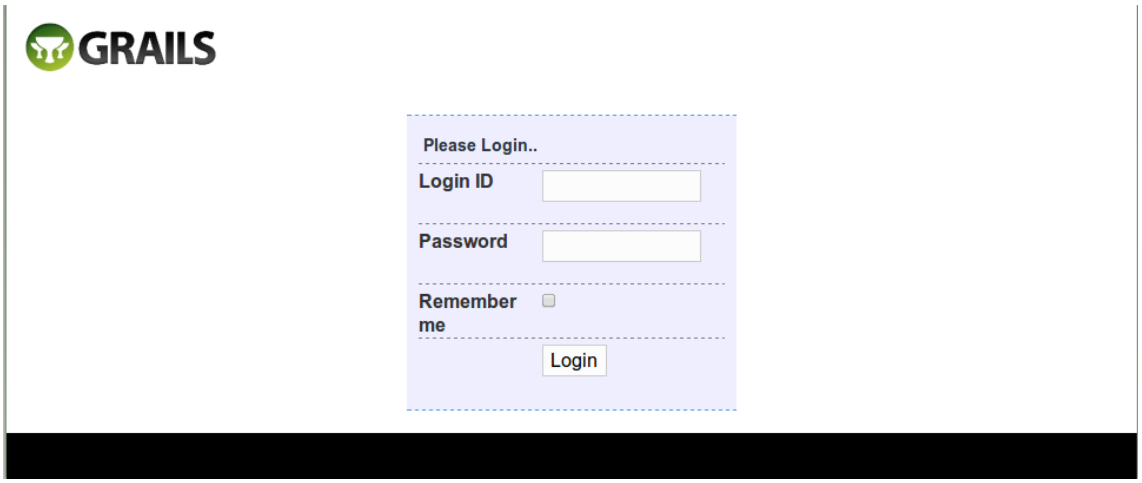

Figura 30: Tela de *Login*

Fonte: Autoria Própria

No campo "*Login ID*", o usuário deverá colocar seu nome de usuário.

No campo "Password", o usuário deverá colocar sua senha.

O campo "*Remember-me*", é marcado opcional pelo usuário, caso ele queira que o computador salve aquele usuário e senha para ele.

Logo após o preenchimento dos campos, o usuário deverá clicar no botão "*Login*".

Caso esteja tudo correto, o usuário é redirecionado para o sistema com seus devidos privilégios, caso algum dos campos não estejam corretos, mostrará uma mensagem de erro e o usuário deverá completar os campos novamente.

3.5.10 Tela Listagem de Aluno - Modo Usuário Cadastrado.

Nesta tela, o usuário que acessou sistema, poderá ver a listagem de todos os alunos cadastrados no programa Bolsa-Permanência.

Na figura 31, pode-se observar como é composta a tela da listagem dos alunos cadastrados.

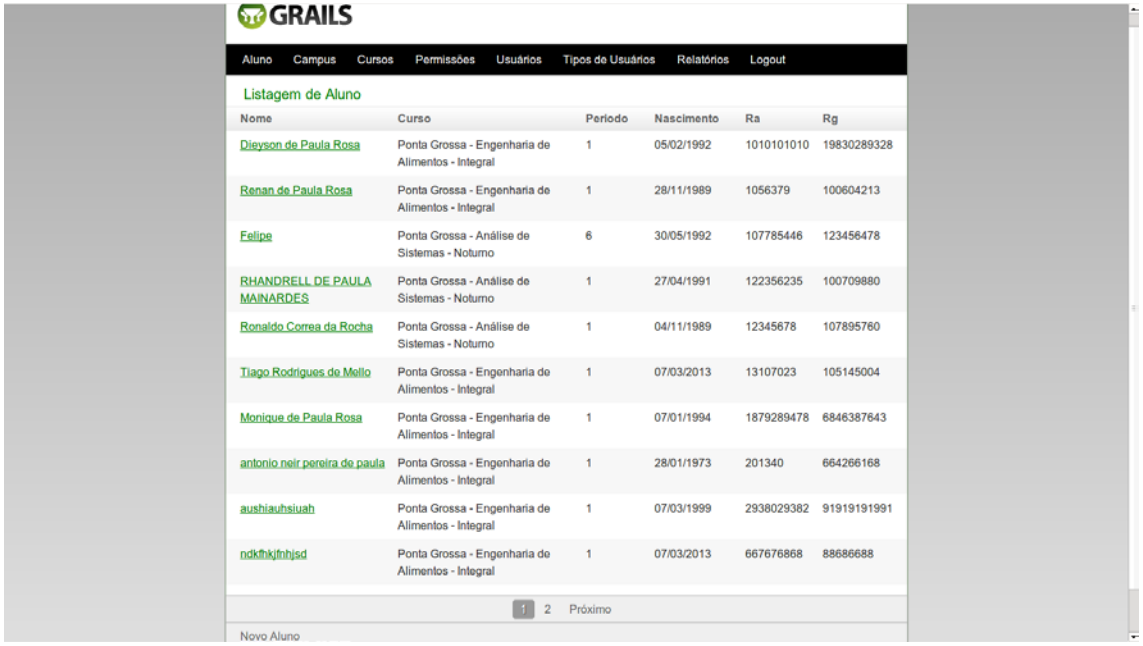

Figura 31: Lista de Alunos Cadastrados

Fonte: Autoria Própria

Nesta tela, vemos todos os alunos cadastrados no sistema por ordem de cadastro. Quando o usuário administrador clicar em algum dos alunos, poderá todas as informações relacionadas àquele aluno e poderá editar alguma informação, quando necessário.

No canto inferior esquerdo, vemos um botão cujo nome é "Novo Aluno". Está opção está presente no modo administrador também, podendo ele cadastrar novos alunos quando necessário.

3.5.11 Tela Listagem de Campi - Modo Usuário Cadastrado

Nesta tela, o usuário poderá ver a listagem de todos os campus cadastrados.

A figura 32 apresenta a tela de listagem de campus.

Figura 32: Lista de Campi

Fonte: Autoria Própria

Nesta tela podem-se ver todos os campi cadastrados no sistema.

Clicando em um dos campi, o usuário poderá editar o nome do campus.

No botão "Novo Campus", o usuário poderá cadastrar um novo campus, quando necessário.

3.5.12 Tela Listagem de Usuários - Modo Usuário Cadastrado

Nesta tela podemos ver todos os usuários cadastrados no sistema.

Na figura 33, pode-se ver a tela de listagem de usuários.

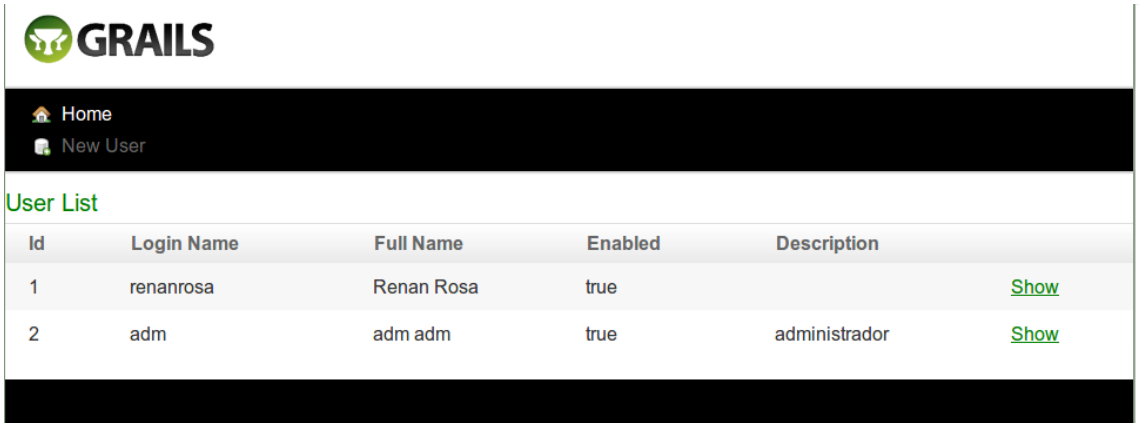

Figura 33: Lista de Usuários

Fonte: Autoria Própria

No botão "Novo Usuário", o administrador poderá clicar e adicionar um novo usuário, quando necessário, com suas devidas permissões.

3.5.13 Tela Listagem de Tipos de usuários - Modo Usuário Cadastrado

Nesta tela podem-se ver todos os tipos de usuários existentes no sistema.

Na figura 34, observa-se a tela de tipo de usuário.

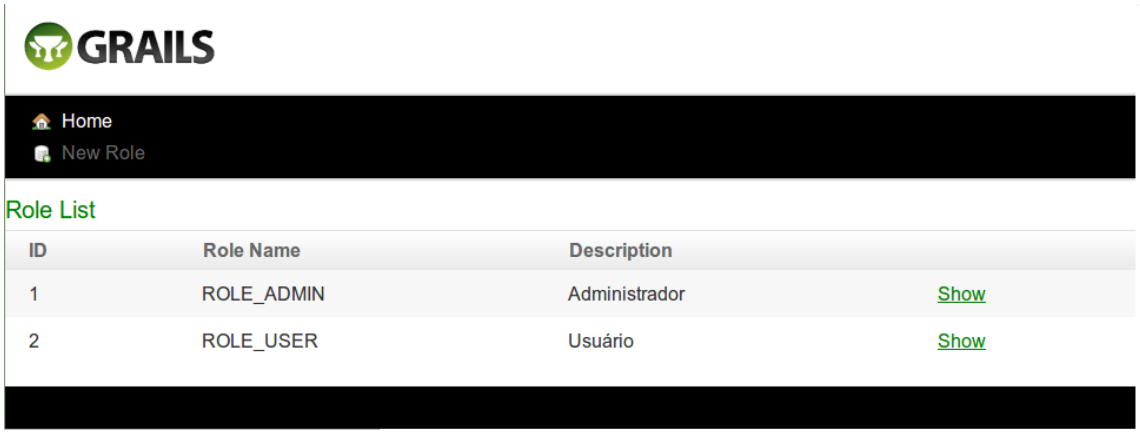

Figura 34: Lista de Tipo de Usuários

Fonte: Autoria Própria

Quando o usuário clicar no botão "Nova Regra", ele poderá criar um novo tipo de usuário.

3.5.14 Tela Relatórios

A figura 35 apresenta a tela dos relatórios necessários para o sistema proposto. Podem-se observar dois relatórios. Um relatório lista todos os alunos cadastrados no programa Bolsa-Permanência e o outro relatório faz a classificação dos estudantes cadastrados.

Figura 35: Tela de Relatórios

Fonte: Autoria Própria

4 EXPERIMENTOS

Para testar o sistema Aegis, disponibilizou-se no endereço aegis.cloudfoundry.com, para que os desenvolvedores pudessem testá-lo no ambiente *Web* real, ou seja, testar a aplicação no ambiente para o qual ela foi desenvolvida, em situações reais.

A implantação do sistema demonstrou ser de maneira extremamente fácil e rápida, devido ao aspecto do *framework Grails* "Convenção sobre configuração" e também ao *plugin cloudfoundry*. Através da configuração do arquivo *BuildConfig.groovy* e do comando "*cf-push*", a aplicação foi implantada rapidamente no servidor *cloudfoundry*, além disso, o banco de dados MySQL também foi implantado com sucesso através do mesmo comando.

Constatou-se que a aplicação se encontrou estável, com rápidas respostas às requisições efetuadas, e com renderização de tela igual dentre os *Web browsers* testados, como: *Internet Explorer* 10, *Internet Explorer* 9, *Mozilla Firefox* 19.0 e *Chromium* 24.0.

Alguns dados foram cadastrados no sistema pelos desenvolvedores, bem como usuários foram convidados pelos desenvolvedores para se cadastrarem no sistema, e informassem se encontrassem algum tipo de erro. Falhas foram ocasionadas propositalmente pelos desenvolvedores, para que estes pudessem verificar se a lógica e as validações da aplicação estavam corretas.

Dos usuários que foram convidados a efetuar o cadastro, todos apresentaram satisfação com o sistema, não encontraram nenhum tipo de erro e conseguiram realizar o cadastro sem qualquer problema.

Por fim, o sistema Aegis foi apresentado à responsável pelo programa Bolsa-Permanência da UTFPR, para que ela pudesse avaliá-lo. O sistema Aegis teve uma resposta positiva, sendo aprovado pela responsável, que declarou que o sistema Aegis "supre as necessidades do NUAPE da UTFPR, no que se refere à informatização do processo de inscrição de alunos no programa Bolsa-Permanência".

4.1 PROBLEMAS ENCONTRADOS

Nesta seção serão apresentados os problemas encontrados no sistema.

4.1.1 *Java* Heap Space Out of Memory

Foi encontrado problemas em diversas etapas da nossa aplicação com relação à falta de memória reservada para o sistema, esse problema de falta de memória se aplicou principalmente no desempenho do sistema, fato que o deixava lento, além disso, na geração de relatórios e no número de campos por tela.

Como os formulários necessitavam de muitos campos para serem preenchidos, muitas vezes a memória reservada para a aplicação não era suficiente, bem como na geração de relatórios, que não eram gerados devido à falta de memória disponível.

Solução encontrada.

Depois de pesquisas sobre esse erro, descobrimos que para resolvê-lo precisaríamos disponibilizar mais memória para aplicação, através da instrução:

*JAVA*_OPTS "-XX:MaxPermSize=512m -Xms512M -Xmx512M"

Essa instrução deve ser inserida como parâmetro em "cf-env-add" , como ilustrado na figura 36:

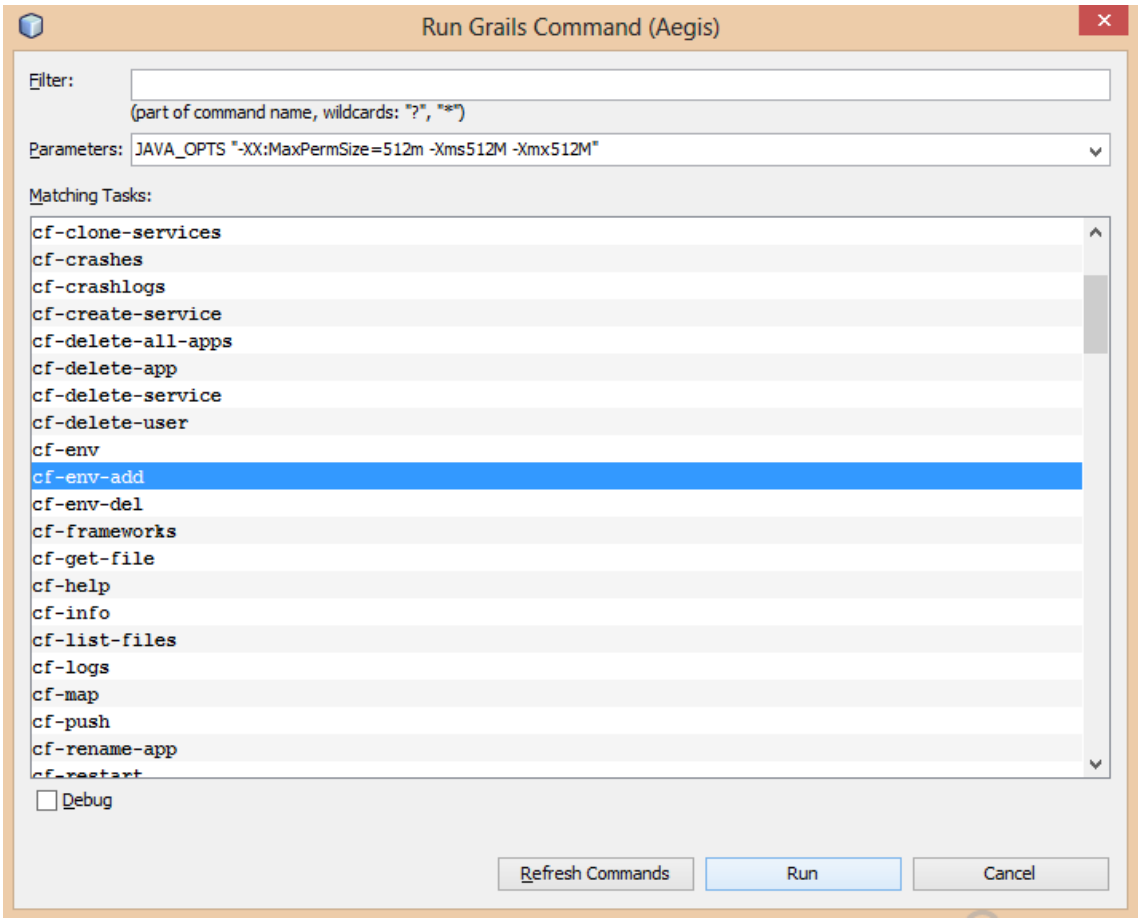

Figura 36: Problemas Encontrados
Fonte: Autoria Própria

Depois de fazer isso, não foi encontrado mais problemas com relação à falta de memória.

4.1.2 *Plugins* Indisponíveis

Por diversas vezes os servidores que hospedam os *plugins* do *Grails* passavam por problemas, dessa forma, os *plugins* ficavam impossibilitados de serem baixados por tempo indeterminado até que os responsáveis por eles conseguissem resolver os problemas.

5 CONCLUSÃO

O sistema descrito neste trabalho pretendia desenvolver um sistema *Web*, utilizando o *framework Grails* e *iReport*, para testar a eficiência dos mesmos na resolução de um problema real, para isso, o programa Bolsa-Permanência da UTFPR foi escolhido como problema à ser resolvido.

O sistema foi desenvolvido para atender às necessidades do Núcleo de Acompanhamento Psicopedagógico e Assistência Estudantil (NUAPE), no processo de cadastros de informações feitas pelos alunos candidatos à receberem a bolsa.

Grails, *iReport* e outros *frameworks* foram estudados, bem como os padrões e as tecnologias que cada *framework* utiliza, afim de serem comparados nos quesitos relevantes ao contexto de desenvolvimento de sistemas.

Contatou-se que o *framework Grails* é de extrema facilidade, rapidez e robustez, devido a imensa gama de recursos que possui e foco na simplicidade. Comportou-se eficientemente na resolução do problema proposto, realizando seu propósito. O *framework iReport* é uma ótima ferramenta para a confecção de relatórios, pois um dos seus pontos fortes é a simplicidade na qual os relatórios são construídos, o que garante uma maior facilidade por parte do desenvolvedor ao criar relatórios.

5.1 SUGESTÕES DE TRABALHOS FUTUROS

Nesta seção podem-se observar alguns pontos que pode ser trabalhados futuramente, são eles:

- Integração do sistema Aegis (nome dado ao sistema proposto) com o Banco de Dados da UTFPR - O sistema Aegis poderia ter acesso à dados dos alunos através de uma integração com o banco de dados da UTFPR, dessa forma, o sistema poderia validar informações do usuário, como o registro acadêmico, e também poderia buscar dados sobre o aluno através do próprio registro acadêmico. Além disso, o sistema poderia verificar as notas do aluno, sua frequência nas aulas, sua situação acadêmica, para que esses dados também possam ser usados como critérios de classificação.

- Criação de contas de usuários para os alunos - Permitir aos alunos criar contas no sistema, dessa forma, eles poderiam entrar no sistema com seu *login* e senha, informar e editar seus dados quando achassem necessário, e posteriormente, disponibilizar os dados para serem processados na etapa de classificação dos candidatos. Esta medida auxilia também no reaproveitamento dos dados dos alunos, ou seja, se o aluno desejar participar de uma nova edição do programa Bolsa-Permanência basta que ele entre no sistema, atualize seus dados se necessário, e os disponibilize para processamento, sem a necessidade de recadastrar todos os seus dados novamente.
- Períodos de inscrições do programa Bolsa-Permanência separados por Campus - Cada campus da UTFPR pode ter períodos diferentes de inscrições do programa Bolsa-Permanência , sendo assim, o aluno informaria primeiramente qual campus ele pertence, o sistema verificaria a data e informaria ao usuário se seu campus está dentro ou não do período de inscrições.
- Informações por *email* - O sistema poderia enviar informações no *email* cadastrado do usuário, como: Confirmação de inscrição, informações da inscrição, mensagens do NUAPE, alunos contemplados, mecanismo de recuperação de senha, etc.
- Área para contato - Uma página poderia ser criada para que o usuário pudesse informar algum erro encontrado no sistema, bem como para fazer perguntas ou sugestões.

6 Referências

APPLETON, Brad. **Patterns and Software**: Essential Concepts and Terminology. disponível na URL: http://www.sci.brooklyn.cuny.edu/~sklar/teaching/s08/cis20.2/papers/appleton-patterns-intro.pdf.

BASHAM, Brian; SIERRA, Kathy; BATES, Bert. **Use a Cabeça. Servlets JSP**. Rio de Janeiro: Alta Books, 2005

BROWN, Jeff; ROCHER, Graeme, **The Definitive Guide to *Grails*.** (2nd ed.), Apress, 2009, 648 p.

BUSCHMANN, F. et at. **A System of Patterns.** Wiley. 1996.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design Patterns** - Elements of Reusable Object-Oriented Software. Reading-MA, AddisonWesley, 1995.

GAMMA, Erich et al. **Padrões de Projeto**: Soluções reutilizáveis de *software* Orientado a Objetos. Porto Alegre: Bookman, 2000.

GONÇALVES, Edson. **Desenvolvendo aplicações *web* com JSP, *Servlet, Java Server Faces, Hibernate, EJB3 persistence, Ájax*.** Rio de Janeiro: Ciência Moderna, 2007.

GOVONI, Darren. ***Java App*lication *Framework*s.** Wiley New York u.a., 1999.

JOHNSON, R., FOOTE, B. ***Designing reusable classes***. Journal of object-oriented programming. SIGS, v.1, n.5, 1988, p.22-35.

JUDD, Christopher M.; NUSAIRAT, Joseph F.; SHINGLER, James. **Beginning *Groovy* and *Grails***. New York: Apress, 2008.

KLEIN, Dave. ***Grails***: A Quick-Start Guide. O'Reilly Vlg. GmbH \& Company, 2009. 222 p.

LAUDON, Kenneth C; Laudon, Jane Price. **Informantion Systems and Internet**: A problem-Solving Approach. Dryden Press, 1998.

PREE, Wolfgang. ***Design Patterns for Object-Oriented Software Development***. 1st edition. Addison-Wesley, 1994.

REENSKAUG, T. **The Model-*View*-Controller (MVC)**: its past and present. [S.l.], 2003.

ROCHER, Graemer. **The Definitive Guide to *Grails***. Apress, 2006. 384 p.

RUDOLPH, Jason, **Getting Started with *Grails***. (1st ed.), Lulu.com, 2007, 132 p.

SMITH, Glen; LEDBROOK, Peter, ***Grails* in Action**. (1st ed.), Manning Publications, 2009, 520 p.

SOLOMON, S. **A grande importância da pequena empresa**: a pequena empresa nos Estados Unidos no Brasil e no mundo. Rio de Janeiro: Editorial Nórdica, 1986.

TOFFOLI, Giulio. **The Definitive Guide to *ireport***. Apress, 2007. 352 p.

XEXÉO, Geraldo. **Modelagem de Sistemas de Informação**: Da Análise de Requisitos ao Modelo de Interface. 2007.

Arquitetura do MVC, Disponível em <wiki.cercomp.ufg.br>, acesso em 10-jan-2013.

Casos de Sucesso com *Grails*, Disponível em <http://*Grails*.org/Success+Stories>, acesso em 03-02-2013.

HIBERNATE, Disponível em <http://*java*programmercomunidade.blogspot.com.br/2011/07.html> , acesso em 19-jan-2013.

Ranking de linguagens de programação segundo site da Tiobe, Disponível em, <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>, acesso em 05-jan-2013

APÊNDICE A – TUTORIAL GRAILS

Instalação no Sistema Operacional Windows

Requisitos:

- *Java Development Kit (JDK)*, versão 1.4 ou superior.
- *Grails*
- *Netbeans ide*, versão 6.5 ou superior.
- *MySql Workbenck*

A *JDK* pode ser obtida através do link: http://www.oracle.com/technetwork/*java*/*java*se/downloads/index.html. O *Grails* pode ser obtido através do link: http://*grails*.org/download. Preferencialmente, opte por baixar as últimas versões da JDK e do *Grails*. Após baixar o *Grails*, descompacte o arquivo em um local de sua preferência. Depois de baixar e instalar a JDK é necessário setar as variáveis:

Variáveis do Usuário:

- *JAVA_HOME*: Local onde a JDK foi instalada. Ex: C:\Program Files*Java*\jdk1.7.0_03
- *GRAILS_HOME*: Local onde o *grails* foi descompactado. Ex: C:*Grails*
- *PATH*: Apontar para a pasta *bin* onde o *Grails* foi instalado. Ex: C:*Grails*\bin

Caso deseje que todos os usuários da máquina onde será instalado o *Grails* possam acessá-lo, configure as variáveis de ambiente, ao invés das variáveis do usuário, os valores são exatamente os mesmos, como podemos ver na figura 1:

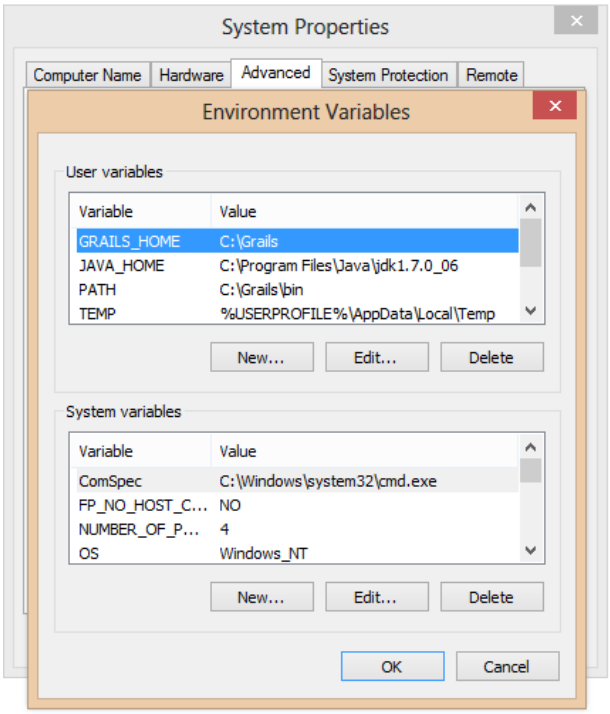

Figura 1: Variáveis do usuário.

Fonte: Autoria Própria

Feito isso, o *Grails* está instalado e configurado em seu computador.

Utilização do *Grails* através do *Netbeans ide*.

Para este tutorial, utilizamos o *Netbeans ide*, por ser uma *ide* de vasta aceitação na comunidade, ser popular, e por suportar diretamente o *Groovy*, que éa linguagem padrão utilizada pelo *Grails*.

Baixe e instale o *Netbeans ide*, que pode ser obtido através do link: http://*netbeans*.org/downloads/

Você pode optar por baixar a tecnologia *Groovy* juntamente com o *Netbeans ide*, ou baixá-la separadamente, através do gerenciador de *plugins* do *Netbeans*, entretanto, se optar pela segunda opção, será um pouco mais complicado do que a primeira.

Caso opte por baixar o *plugin* separadamente, abra seu *Netbeans* e selecione a opção "Ferramentas", que está localizada na barra superior da *ide*, e logo em seguida selecione a opção "*Plugins*". Feito isso, selecione a opção "*Plugins* disponíveis" e procure por "*grails*", como mostra a figura 2.

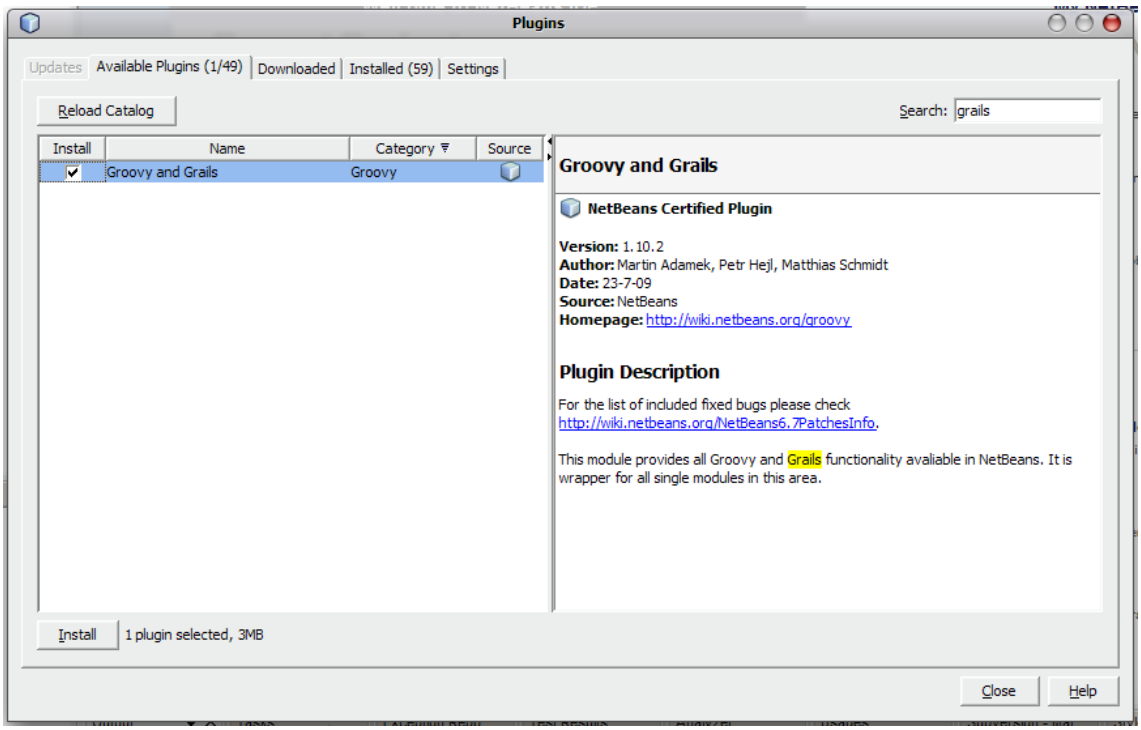

Figura 2: Baixar *Groovy* e *Grails* pelo *Netbeans*.

Fonte: Autoria Própria.

Depois de feito o download, o *Netbeans ide* irá instalá-lo e requisitará a sua reinicialização, por fim, reinicie-o e então o *Netbeans* estará pronto para o desenvolvimento de aplicações *Grails*.

Meu primeiro projeto em *Grails*

Neste tutorial, criaremos uma sistema gerenciador de biblioteca, no qual consistirá na inserção, edição e exclusão dos livros no acervo. Então, vamos colocar a mão na massa.

Configurando o acesso ao Banco de Dados

Para este tutorial utilizaremos o *MySQL Workbench*, primeiramente, crie um novo banco de dados, chame-o de "Biblioteca", dessa forma, quando o *Grails* gerar o banco de dados, ele irá gerar por padrão com o nome da aplicação.

No *Netbeans ide*, vá até o arquivo "*DataSource.groovy*" que está localizado no diretório "*Configuration*". Altere as propriedades do arquivo conforme a figura 3

.

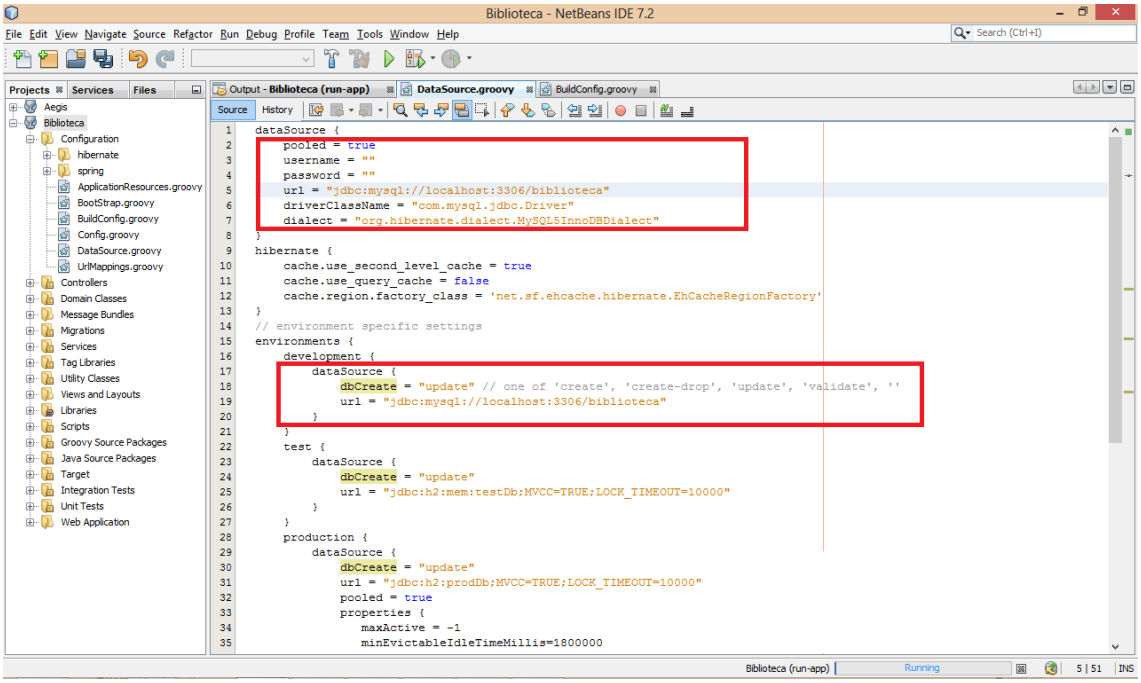

Figura 3: Propriedades para ser alteradas

Fonte: Autoria Própria.

Feito isso, abra o arquivo “*BuildConfig.groovy*” que se encontra na mesma pasta do arquivo anterior, descomente a linha ”runtime *'mysql:mysql-connector-java:5.1.20'*”.

Iniciando nosso projeto em *Grails*

Crie um novo projeto no *Netbeans ide*, na categoria, escolha “*Groovy*”, e em projetos “aplicação *Grails*”. Em seguida, dê um nome para o seu projeto, neste caso, colocaremos simplesmente o nome “Biblioteca”. Feito isso, o framework *Grails* será carregado, e assim, criará o ambiente necessário para que possamos iniciar o desenvolvimento de nossa aplicação, como mostra a figura 4.

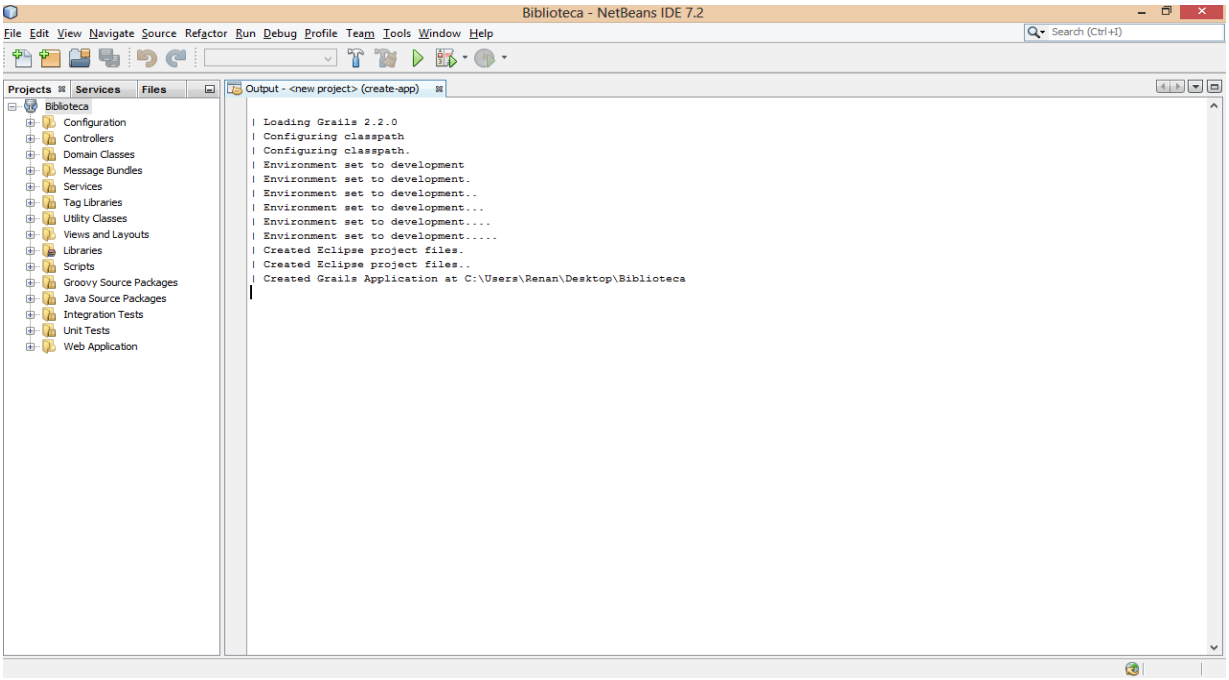

Figura 4: Ambiente Carregando do *Grails*

Fonte: Autoria Própria

Feito isso, clique com o botão direito do mouse em “Biblioteca”, depois em “*New*”, e então em “*Grails Domain class*”, como na figura 5:

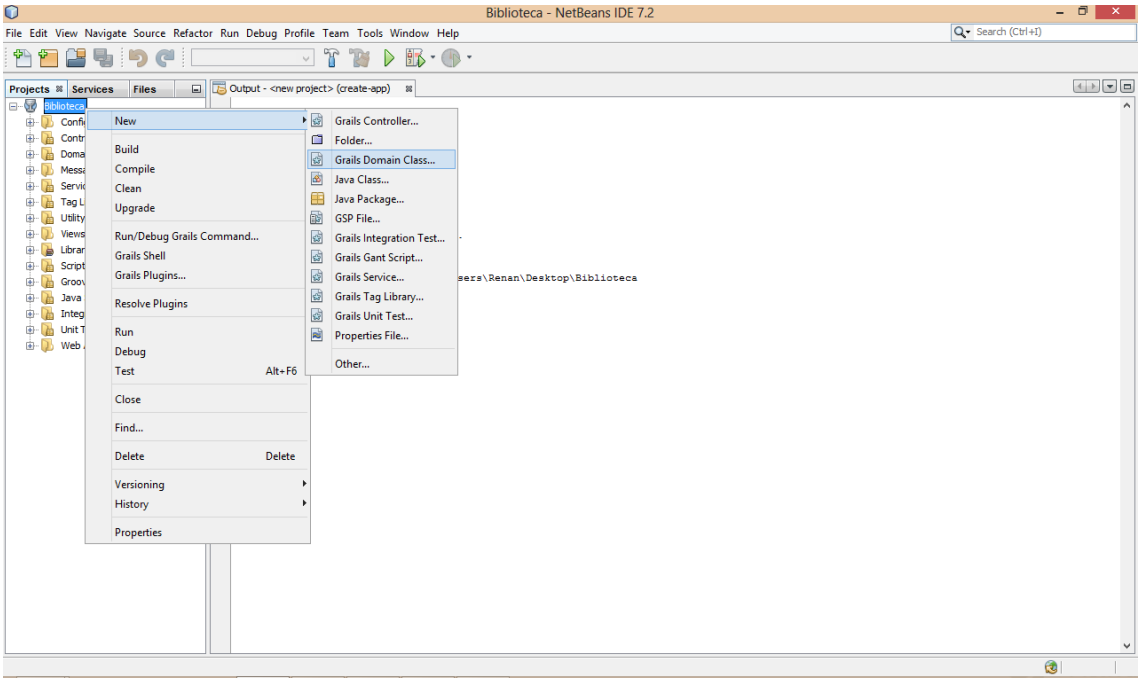

Figura 5: Criando *Domain class*

Fonte: Autoria Própria

Uma nova tela irá aparecer e então no campo "*Artifact Name*" insira "Livro", como mostra a figura 6.

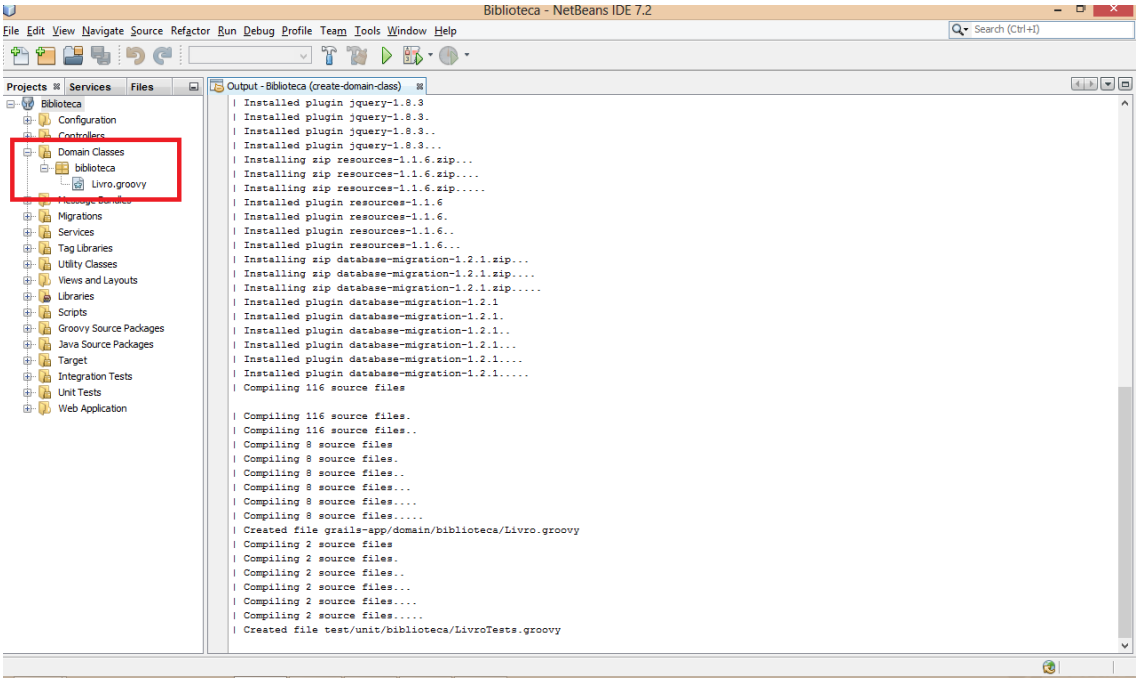

Figura 6: Criar Primeira Classe

Fonte: Autoria Própria

Feito isso, o *Grails* irá gerar a classe de domínio “Livro” no diretório “*Domain classes*”, por padrão, é nesse diretório que todas as classes de domínio ficarão localizadas.

Agora, vamos dar uma olhada na classe em que o *Grails* acabou de gerar, como mostra a figura 7:

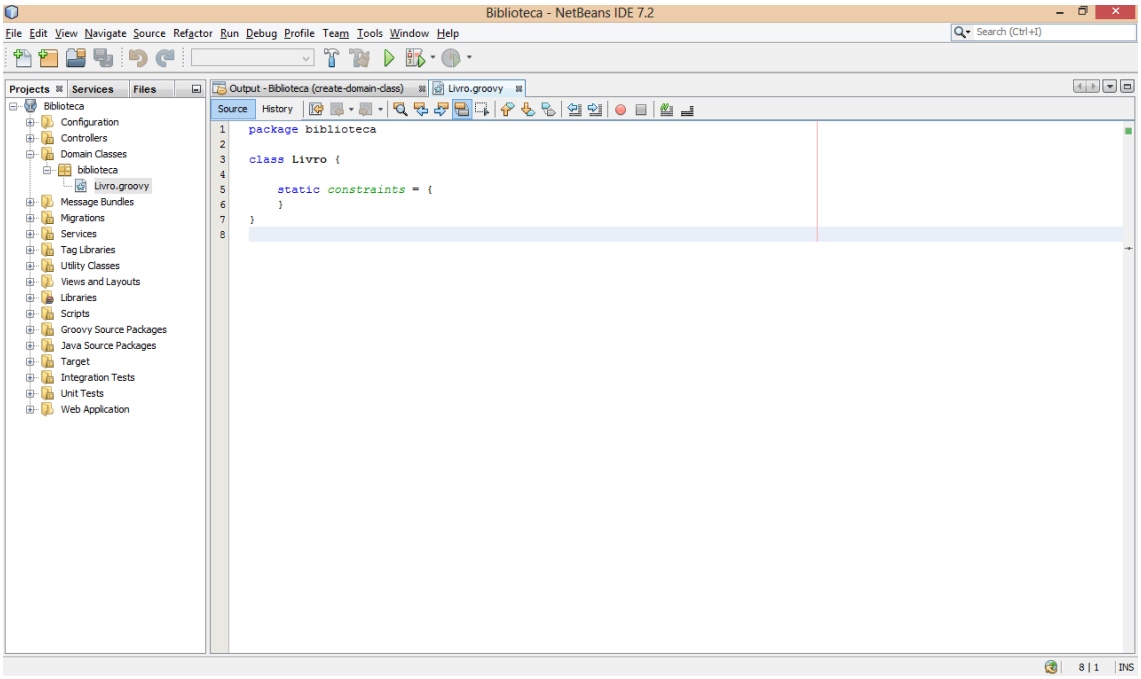

Figura 7: Classe gerada pelo *Grails*

Fonte: Autoria Própria

Note que a classe contém as informações: “*Package* biblioteca”, ou seja, o diretório no qual a classe está localizada, dentro de “*Domain classes*”, “classe Livro”, o nome da nossa classe, e “*static contraints*”, é aqui que vamos definir as validações que devemos aplicar às nossas classes de domínio.

Vamos inserir na nossa classe alguns atributos como: Título e Autor.

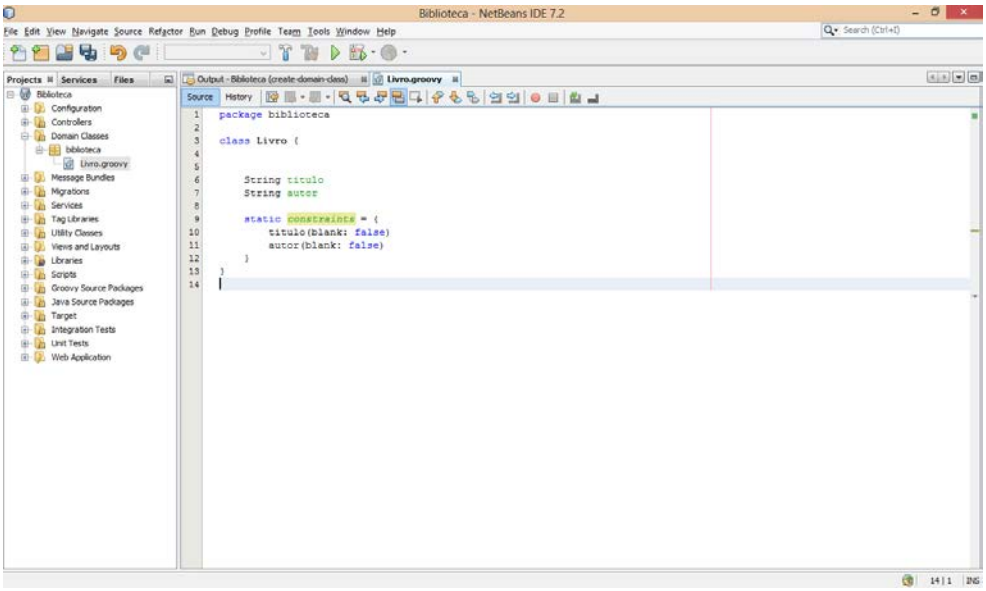

Figura 8: Inserindo Atributos

Fonte: Autoria Própria

Nas constraints, colocamos “blank : false” para que seja obrigatória a inserção de dados nos campos, dessa forma campos em branco não serão aceitos.

Feito isso, vamos criar um controlador para a nossa classe, seguindo o modelo da figura 9:

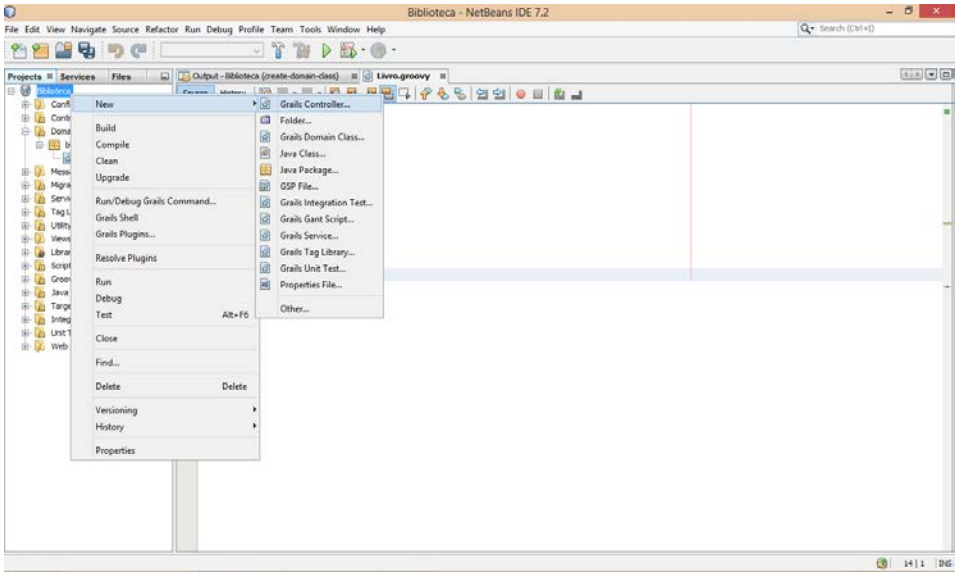

Figura 9: Criando Controlador

Fonte: Autoria Própria

Os controladores são responsáveis pelas requisições feitas à nossa aplicação.

Dê o nome do controlador de “Livro”, por padrão, o controlador será criado com o nome dado pelo desenvolvedor seguido de “*Controller*”, ou seja, o nosso controlador “Livro” tornou-se “*LivroController*”, que foi criado no diretório “biblioteca”, que está dentro do diretório “*Controllers*”, como você deve ter percebido, é nesse diretório que todos os controladores estarão.

Ao abrirmos o controlador que acabamos de criar, veremos “*package* biblioteca” e “*class* Livro *Controller*” que significam exatamente o mesmo que na classe de domínio “Livro” que vimos anteriormente, entretanto, notamos a presença do método “index”, ele é o primeiro método chamado pelo controlador quando este é acessado.

Depois de criarmos o controlador, vamos criar as telas e a camada de persistência, tudo isso em poucos passos. Primeiramente clique com o botão direito sobre a classe de domínio que deseja gerar as telas e a camada de persistência, no nosso caso, a classe “Livro”, depois clique em “*Generate All*”.

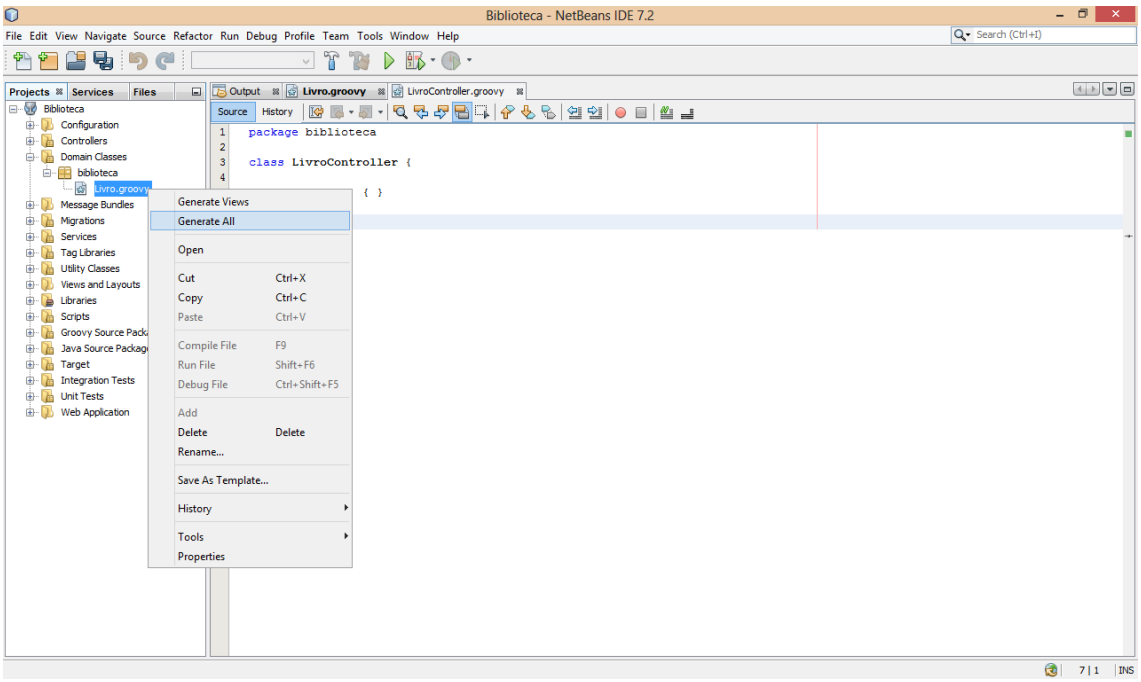

Figura 10: Gerando telas e camada de Persistência

Fonte: Autoria Própria

Feito isso, o *Grails* irá gerar as telas e camada de persistência, caso algum arquivo precise ser sobrescrito ele irá avisá-lo e perguntará se pode sobrescrever o arquivo, caso isso aconteça, permita que o *Grails* o sobrescreva, depois disso podemos executar o nosso projeto.

Ao navegarmos até a tela de criação de um novo livro e se tentarmos criar um novo livro com os campos “Título” e “Autor ” vazios, a aplicação irá emitir um aviso, nos dizendo que devemos inserir valores nos campos. Essa validação foi feita na classe de domínio “Livro”, através da cláusula “*blank: false*”.

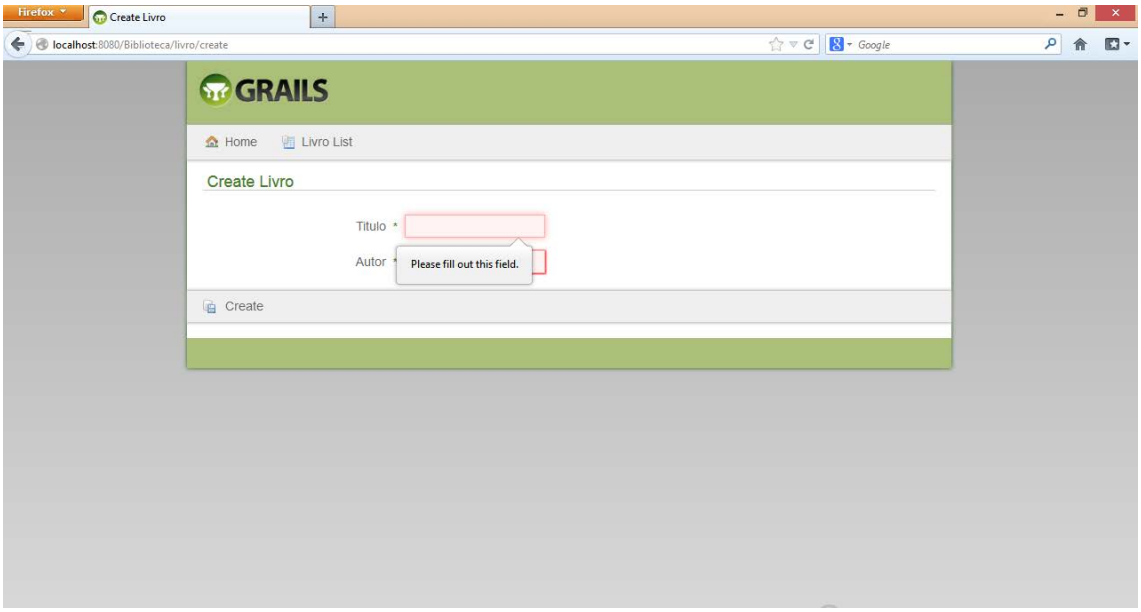

Figura 11: Validações

Fonte: Autoria Própria

Navegando pela aplicação, podemos ver que o *Grails* criou as telas de inserção, edição e listagem dos dados, bem como a opção de excluirmos determinado objeto de nosso banco de dados.

Através da tela de listagem podemos ver todos os objetos salvos no nosso banco de dados, bem como acessá-los para visualizá-los, editá-los ou excluí-los.

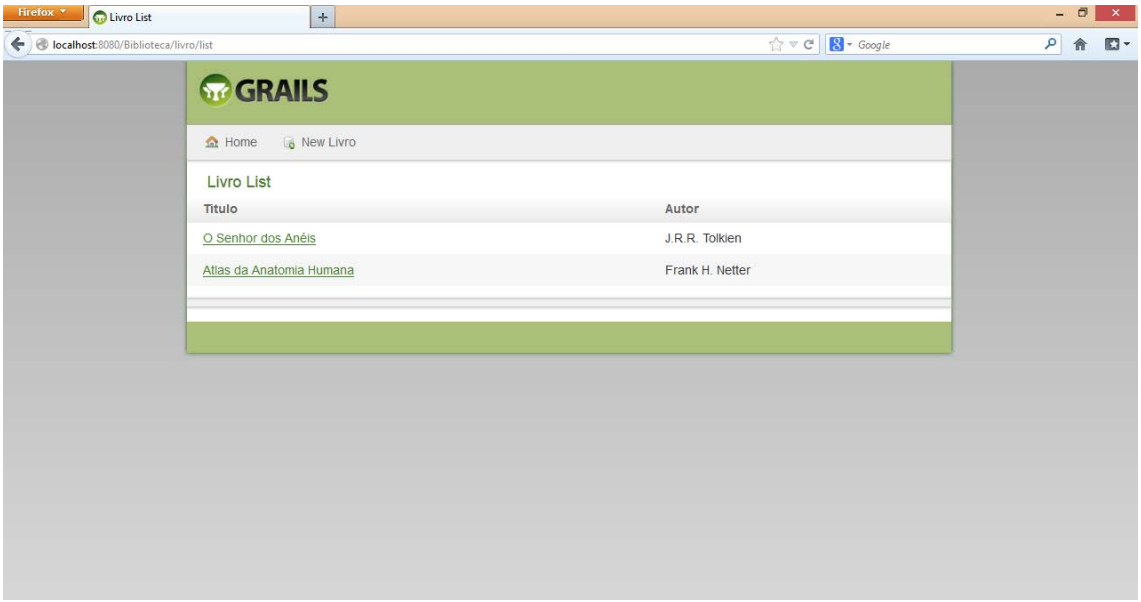

Figura 12: Tela Listagem

Fonte: Autoria Própria

Assim, concluímos o nosso primeiro projeto em *Grails*.

APÊNDICE B – ENTREVISTA

Entrevista com o NUAPE(Núcleo de Acompanhamento Psicopedagógico e Assistência Estudantil) sobre o Bolsa-Permanência

27 de Abril de 2012

1 O que é o Bolsa-Permanência?

Neste semestre (primeiro semestre de 2012), no campus Ponta Grossa, a bolsa é disponibilizada em dinheiro, no valor de R$200,00/mês, e também na forma de refeições servidas no Restaurante Universitário (os estudantes contemplados no programa podem fazer suas refeições, sem custo para os mesmos, sendo almoço e jantar de segunda à sexta-feira, e almoço nos sábados).

2 Qual a Finalidade do Bolsa-Permanência?

O Programa de Bolsa-Permanência ao Estudante da UTFPR terá a finalidade de apoiar o discente para sua permanência na Instituição, buscando reduzir os índices de evasão decorrentes de dificuldades de ordem socioeconômica. O Programa é destinado ao estudante regular dos cursos presenciais da UTFPR.

3 Como funciona o Bolsa-Permanência?

A Reitoria estabelecerá o número de Bolsa-Permanência para cada Campus e em cada semestre letivo, considerando os recursos orçamentários disponíveis e o número de estudantes matriculados nos cursos regulares da UTFPR. No início de cada semestre será publicado o Edital do Programa que estabelecerá as condições para a participação e critérios de seleção dos estudantes interessados.

4 Quais são as informações necessárias que os candidatos devem informar ?

Todas as informações contidas no formulário do Bolsa Permanência são necessárias.

5 Quais são os critérios levados em consideração para classificação do aluno?

A classificação será de acordo com a renda familiar per capita, obtida pela divisão da renda familiar total pelo número de integrantes do grupo familiar.

5.1 E quais são os critérios para que o pedido de bolsa do aluno seja indeferido ?

Os critérios usados para que o pedido seja indeferido são:
I Por renda maior que o estipulado (renda per capita não pode exceder a 1,5 – um e meio - salários mínimos nacional);
II – Por índice de reprovação maior que 30 por cento no último semestre cursado, exceto para o inscrito pela primeira vez no Programa;
III – Por falta de documentos;
IV – Por sanção disciplinar no semestre anterior.
V – Estar cursando ~~apenas~~ estágio, trab conclusão de curso ou atividades complementares

5.2 Existem critérios de desempate ? Quais?

Sim, existem critérios de desempates. Os critérios de desempates são:
I – Maior número de integrantes do grupo familiar; e
II – maior idade.

6 Quais requisitos são necessários para o sistema de Bolsa-Permanência?

O sistema precisara fazer o cadastro do aluno no programa Bolsa-Permanência e fazer a classificação dos alunos por ordem, segundo os critérios pré-estabelecidos. Precisará também guardar o histórico de todos os inscritos nos semestres subsequentes, para eventual consulta.

Marcia Angelica Bartmeyer

APÊNDICE C – QUESTIONÁRIO FINAL

Questionário

1. Quantas pessoas trabalham e quais são suas funções no processo de classificação dos estudantes ?

No processo de inscrição e recebimento dos documentos temos a participação dos membros do Nuape (atualmente 5 pessoas). Já no processo da avaliação socioeconômica, são 2 pessoas que atuam.

2. Qual é o tempo estimado para a classificação dos estudantes ?

O tempo (prazo) é determinado pelo Edital. Tradicionalmente o prazo é de 10 a 15 dias.

3. Quais são as maiores dificuldades do processo de classificação?

Tempo/prazo escasso.

Alimentação manual de dados em planilhas e arquivos diversos; falta de programa que possibilite extrair relatórios variados; falta integração com sistema Acadêmico para busca de dados e acompanhamento acadêmico

4. Você acredita ser necessário a informatização do processo de classificação ?

Sim. Facilitaria tanto para os alunos quanto para a equipe do Nuape. Os dados ficariam registrado eletronicamente, bastando a atualização semestral pelos estudantes. Minimiza ainda as chances de eventual equívocos na alimentação de planilhas Excel e também de perda de dados por arquivo corrompido ou falta de backup.

5. Deixe suas sugestões.

As sugestões foram repassadas em conversa com os alunos que estão desenvolvendo o trabalho.

Obs: Integração com banco de dados (sist. acadêmico) da UTFPR (próxima etapa)

Nome: Márcia Angélica Bartmeyer

Função: Assistente Social

Assinatura:

Data: 12/03/2013

APÊNDICE D – DECLARAÇÃO DE APROVAÇÃO DO SISTEMA AEGIS

Declaração

Eu, Márcia Angélica Bertmeyer, portador(a) do CPF ___, declaro que o projeto "Aegis" supre as necessidades do Núcleo de Acompanhamento Psicopedagógico e Assistência Estudantil (NUAPE) da UTFPR, no que se refere à informatização do processo de inscrição de alunos no programa Bolsa-Permanência.

Ponta Grossa, 13 de março de 2013.

Assinatura